Sahara Specs

Release 0.0.1.dev366

Sahara Team

Sep 20, 2023

CONTENTS

Implemented specs by release:

CHAPTER

ONE

ROCKY SPECS

1.1 placeholder

placeholder

1.1.1 Problem description

placeholder

1.1.2 Proposed change

placeholder

Alternatives

placeholer

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

1.1.3 Implementation

Assignee(s)

None

Work Items

None

1.1.4 Dependencies

None

1.1.5 Testing

None

1.1.6 Documentation Impact

None

1.1.7 References

None

1.2 Plugins outside Sahara core

Plugins are a very important part of Sahara, they allow the creation of clusters for different data processing tools and the execution of jobs on those clusters. This proposal is to remove the plugins code from Sahara core and create a new project to host the plugins.

1.2.1 Problem description

With the plugins code inside Sahara core we are limited to upgrade plugins versions with the cycle milestone. It also forces the user to upgrade OpenStack version whenever he/she needs to upgrade Sahara plugins.

1.2.2 Proposed change

We are going to move the plugins to its own project, releasing new versions when we upgrade new plugins, thus allowing the users to upgrade to newer versions without the hussle of upgrading the whole cloud.

In order to keep the projects as less coupled as possible we are implementing a mediator under sahara/plugins that will be used as an API between the projects. Also this API aims to facilitate manutenability of both projects.

Alternatives

Keep the plugins code as it is. Not changing will not break anything or make things more difficult to the users.

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

User will be able to upgrade plugins versions a lot faster once this is done. Deployer will have to keep an eye for the compatibility between sahara and sahara-plugins if there is significant changes to the API.

There is also an impact for packagers and translators since we will need to do one-time work to setup and copy translations in the new repository.

Developer impact

With a new project, developers will have to get used to the fact that plugins don't live with the core anymore. There is a new API (mediator) implemented on the sahara side that will be the bridge between the two projects. Developers must respect that mediator and significant changes to that will require version bumping or branching.

Image Packing impact

Image packing using the new image generation and validation system will require to have sahara-plugins installed as well.

Sahara-dashboard / Horizon impact

None

1.2.3 Implementation

Assignee(s)

Primary assignee:
tellesnobrega

Work Items

- Split the plugins code from Sahara core
- Bring plugins unit test to the plugins repo
- Make sure imports in sahara-plugins from sahara are well structured so not to break with sahara changes later on

1.2.4 Dependencies

None

1.2.5 Testing

Move plugins tests from sahara core to sahara-plugins

1.2.6 Documentation Impact

We need to update the documentation to reflect the change and make sure users and developers are well aware of this new structure.

1.2.7 References

None

CHAPTER

TWO

QUEENS SPECS

2.1 Add HBase on Vanilla cluster

https://blueprints.launchpad.net/sahara/+spec/hbase-on-vanila

Apache HBase provides large-scale tabular storage for Hadoop using the Hadoop Distributed File System(HDFS). This document serves as a description to add the support of HBase and ZooKeeper services on Vanilla cluster.

2.1.1 Problem description

Sahara vanilla plugin allows user to quickly provision a cluster with many core services, but it doesn't support HBase and ZooKeeper.

2.1.2 Proposed change

To go against the Vanilla cluster distributed architecture, we only support fully-distributed HBase deployment. In a distributed configuration, the cluster contains multiple nodes, each of which runs one or more HBase Daemon. These include HBase Master instance, multiple ZooKeeper nodes and multiple RegionServer nodes.

A distributed HBase installation depends on a running ZooKeeper cluster. HBase default manages a ZooKeeper "cluster" for you, but you can also manage the ZooKeeper ensemble independent of HBase. The variable "HBASE_MANAGES_ZK" in "conf/hbase-env.sh", which default to true, tells HBase whether to start/stop the ZooKeeper ensemble servers as part of HBase.

We should expose this variable in "cluster_configs" to let user determine the creator of ZooKeeper service.

In production, it is recommended that run a ZooKeeper ensemble of 3, 5 or 7 machines; the more members an ensemble has, the more tolerant the ensemble is of host failures. Also, run an odd number of machines. An even number of peers is supported, but it is normally not used because an even sized ensemble requires, proportionally, more peers to form a quorum than an odd sized ensemble requires.

- If we set "HBASE_MANAGES_ZK" to false, Sahara will validate the number of ZooKeeper services in node groups to keep ZK instances in odd number.
- If we set "HBASE_MANAGES_ZK" to true, Sahara will automatically determine the instances to start ZooKeeper. The cluster contains ZK nodes more than 1 nodes, less than 5 nodes. If we want to have more ZK nodes, setting HBASE_MANAGES_ZK to false would be a good choice.

If we want to scale the cluster up or down, ZooKeeper and HBase services will be restarted. And after scaling up or down, the rest of ZooKeeper nodes should also be kept in odd number. If there is only one ZooKeeper node, the status of ZooKeeper service will be "standalone".

One thing should be specified is the default value used in configuration:

ZooKeeper Configuration in "/opt/zookeeper/conf/zoo.cfg":

```
dataDir=/var/data/zookeeper
clientPort=2181
server.1=zk-0:2888:3888
server.2=zk-1:2888:3888
```

HBase Configuration in "/opt/hbase/conf/hbase-site.xml":

```
hbase.tmp.dir=/var/data/hbase
hbase.rootdir=hdfs://master:9000/hbase
hbase.cluster.distributed=true
hbase.master.port=16000
hbase.master.info.port=16010
hbase.regionserver.port=16020
```

Security Group will open ports (2181, 2888, 3888, 16000, 16010, 16020) after this change if configuration is not changed.

Alternatives

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

- Build new Vanilla image includes ZK and HBase packages

Sahara-dashboard / Horizon impact

- An option should be added to the Node Group create and update forms.

2.1.3 Implementation

Assignee(s)

Primary assignee:
Shu Yingya

Work Items

- Build new image by sahara-image-elements
- Add ZooKeeper to Vanilla in sahara
- Add HBase to Vanilla in sahara
- Update Sahara-dashboard to choose ZK creator in sahara-dashboard

2.1.4 Dependencies

None

2.1.5 Testing

- Unit test coverage in sahara

2.1.6 Documentation Impact

- Vanilla plugin description should be updated

2.1.7 References

None

2.2 There and back again, a roadmap to API v2

https://blueprints.launchpad.net/sahara/+spec/v2-api-experimental-impl

As sahara's API has evolved there have been several features introduced in the form of routes and methods that could be crafted in a more consistent and predictable manner. Additionally, there are several new considerations and methodologies that can only be addressed by updating the major version of the API. This document serves as a roadmap to implement an experimental v2 API which will form the basis of the eventual stable version.

Note: This is an umbrella specification covering many changes, there will be followup specifications to cover some of the more intricate details.

2.2.1 Problem description

The current version of sahara's REST API, 1.1, contains several methodologies and patterns that have created inconsistencies within the API and with respect to the API Working Group's evolving guidelines[1]. Many of these are due to the iterative nature of the design work, and some have been created at a time before stable guidelines existed.

Examples of inconsistencies within the current API:

- Inaccurate names for method endpoints, for example "jobs" instead of "job-templates".
- Technology specific parameters in JSON resources, for example "oozie_job_id" instead of "engine_job_id".
- Improper HTTP method usage for some operations, for example using PUT for partial resource updates instead of PATCH.

In addition to resolving the inconsistencies in the API, a new version will provide an opportunity to implement features which will improve the experience for consumers of the sahara API.

Examples of features to implement in the new API:

- Micro-version support to aid in feature discovery by client applications.
- Creation of tasks endpoint and infrastructure to improve usage of asynchronous operations.
- HREF embedding in responses to improve resource location discovery.

These are just a few examples of issues which can be addressed in a new major version API implementation.

2.2.2 Proposed change

To address the creation of a new major version API, an experimental `/v2` endpoint should be created. This new endpoint will be clearly marked as experimental and no contract of stability will be enforced with regards to the content of its sub-endpoints. Changes to the `/v2` endpoint will be tracked through features described in this specification, and through further specifications which will be created to better describe the details of larger changes.

When all the changes to the `/v2` endpoint have been made such that it has a 1:1 feature compliance with the current API, and the Python sahara client has been updated to use these new endpoints, the experimental status of the API should be assessed with the goal of marking it as stable and ready for public consumption.

The individual changes will be broken down into individual tasks. This will allow the sahara team members to more easily research and implement the changes. These efforts will be coordinated through a page on the sahara wiki site[2].

Initial v2 commit

The initial changes to create the `/v2` endpoint should also include moving the Identity project identifier to a header named `OpenStack-Project-ID`. In all other respects, the endpoints currently in place for the `/v1.1` API will be carried forward into the new endpoint namespace. This will create a solid base point from which to make further changes as the new API evolves and moves towards completion of all features described in the experimental specifications.

Removing the project identifier from the URI will help to create more consistent, reusable, routes for client applications and embedded HREFs. This move will also help decouple the notion of URI scoped resources being tied to a single project identifier.

Roadmap of changes

The following list is an overview of all the changes that should be incorporated into the experimental API before it can be considered for migration to stable. These changes are not in order of precedence, and can be carried out in parallel. Some of these changes can be addressed with simple bugs, which should be marked with `[APIv2]` in their names. The more complex changes should be preceeded by specifications marked with the same `[APIv2]` moniker in their names. For both types of changes, the commits should contain `Partial-Implements: bp v2-api-experimental-impl` to aid in tracking the API conversion process.

Overview of changes:

- Endpoint changes
  - /images/{image_id}/tag and /images/{image_id}/untag should be changed to follow the guidelines on tags[3].
  - /jobs should be renamed to /job-templates.
  - /job-executions should be renamed to /jobs.
  - executing a job template through the /jobs/{job_id}/execute endpoint should be changed to a POST operation on the new /jobs endpoint.

  - cancelling a job execution through the /job-executions/{job_execution_id}/cancel endpoint should be removed in favor of requesting a cancelled state on a PATCH to the new /jobs/{job_id} endpoint.
  - /job-binary-internals should be removed in favor of /job-binaries as the latter accepts internal database referenced items, an endpoint under /job-binaries can be created for uploading files(if required).
  - /job-executions/{job_execution_id}/refresh-status should be removed in favor of using a GET on the new /jobs/{job_id} endpoint for running job executions.
  - all update operations should synchronize around using PATCH instead of PUT for partial resource updates.

- JSON payload changes
  - hadoop_version should be changed to plugin_version.
  - oozie_job_id should be changed to engine_job_id.
  - all returned payloads should be wrapped in their type, this is currently true for the API and should remain so for consistency.
  - HREFs should be embedded in responses that contain references to other objects.

- New features
  - Identity project identifier moved to headers. This will be part of the initial version 2 commit but is worth noting as a major feature change.
  - Micro-version support to be added, this should be documented fully in a separate specification but should be based on the work done by the ironic[4] and nova[5] projects. Although implemented during the experimental phase, these microversions will not implement the backward compatibility features until the API has been declared stable. Once the API has moved into the stable phase, the microversions will only implement backward compatibility for version 2, and only for features added after the stable release.
  - Version discovery shall be improved by adding support for a "home document" which will be returned from the version 2 root URL. This document will follow the json-home[6] draft specification. Additionally, support for microversion discovery will be added using prior implementations and the API working group guidelines as guides.
  - Creation of an actions endpoint for clusters to provide a single entrypoint for operations on those clusters. This endpoint should initially allow operations such as scaling but will be used for further improvements in the future. The actions endpoint will be the subject a separate specification as it will describe the removal of several verb-oriented endpoints that currently exists, and the creation of a new mechanism for synchronous and asynchronous operations.

This list is not meant to contain all the possible future changes, but a window of the minimum necessary changes to be made before the new API can be declared as stable.

The move to stable for this API should not occur before the Python sahara client has been updated to use the new functionality.

Alternatives

An alternative might be to make changes to the current version API, but this is inadvisable as it breaks the API version contract for end users.

Although the current version API can be changed, there is no way to safely make the proposed changes without breaking backward compatibility. As the proposed changes are quite large in nature it is not advisable to create a "1.2" version of the API.

Data model impact

Most of these changes will not require modifications to the data model. The two main exceptions are the payload name changes for `hadoop_version` and `oozie_job_id`. As the data model will continue to be used for the v1.1 API until it is deprecated, it is not advisable to rename these fields at this time. When the v2 API has been made stable, and the v1.1 API has been deprecated, these fields should be revisisted and changed in the data model.

During the experimental phase of the API, these translations will occur in the code that handles requests and responses. After the API has transitioned to production mode, migrations should be created to align the data models with the API representations and translations should be created for the older versions only as necessary. As the older version API will eventually be deprecated, these changes should be scheduled to coincide with that transition.

REST API impact

As this specification is addressing a high level change of the API, the following changes are enumerated in brief. Full details should be created for changes that will require more than just renaming an endpoint.

- creation of /v2 root endpoint
- removal of {tenant_id} from URI, to be replaced by `OpenStack-Project-ID` header on all requests.
- removal of POST to /images/{image_id}/tag
- removal of POST to /images/{image_id}/untag
- creation of GET/PUT/DELETE to /images/{image_id}/tags, this should be followed with a specification describing the new tagging methodology.
- creation of GET/PUT/DELETE to /images/{image_id}/tags/{tag_id}, this should also be in the previously mentioned specification on tagging.
- move operations on /jobs to /job-templates
- move operations /job-executions to /jobs
- removal of POST to /jobs/{job_id}/execute
- creation of POST to /jobs, this should be defined in a specification about restructuring the job execution endpoint.
- creation of jobs via the /jobs endpoint should be transitioned away from single input and output fields to use the newer job configuration interface[7].
- removal of GET to /job-executions/{job_execution_id}/cancel

- creation of PATCH to /jobs/{job_id}, this should be defined in the specification about restructuring the job execution endpoint.
- removal of GET to /job-executions/{job_execution_id}/refresh-status
- removal of all /job-binary-internals endpoints with their functionality being provides by /job-binaries, this may require creating a separate sub-endpoint for uploading.
- refactor of PUT to /node-group-templates/{node_group_template_id} into PATCH on same endpoint.
- refactor of PUT to /cluster-templates/{cluster_template_id} into PATCH on same endpoint.
- refactor of PUT to /job-binaries/{job_binary_id} into PATCH on same endpoint.
- refactor of PUT to /data-sources/{data_source_id} into PATCH on same endpoint.

Other end user impact

In the experimental phase, this change should have no noticeable affect on the end user. Once the API has been declared stabled, users will need to switch python-saharaclient versions as well as upgrade their horizon installations to make full use of renamed features.

Deployer impact

During the experimental phase, this change will have no effect on deployers.

When the API reaches the stable phase, deployers will be responsible for upgrading their installations to ensure that sahara and python-saharaclient are upgraded as well as changing the service catalog to represent the base endpoint.

Developer impact

As this change is targeted for experimental work, developers should know that the details of the v2 API will be constantly changing. There is no guarantee of stability.

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

This change should not require changes to horizon as many of the primitives that are changing already display the proper names, for example "Job Templates". When this change moves to the stable phase, horizon should be re-evaluated.

2.2.3 Implementation

Assignee(s)

Primary assignee:
Telles Nobrega

Other contributors:
mimccune (Michael McCune)

Work Items

The main work item for this specification is the initial v2 commit.

- create v2 endpoint
- create code to handle project id in headers
- create mappings to current endpoints

2.2.4 Dependencies

This change should not require new dependencies.

2.2.5 Testing

Unit tests will be created to exercise the new endpoints. Additionally, the gabbi[8] testing framework should be investigated as a functional testing platform for the REST API.

To improve security testing, tools such as Syntribos[9] and RestFuzz[10] should be investigated for use in directed testing efforts and as possible gate tests.

These investigations should result in further specifications if sufficient results are discovered to warrent their creation as they will deal with new testing modes for the sahara API server.

As the v2 API reaches stable status, and the python-saharaclient has been ported to use the new API, the current functional tests should provide the necessary framework to ensure successful end-to-end testing.

2.2.6 Documentation Impact

During the experimental phase, this work will not produce documentation. As the evaluation for stable approaches there will need to be a new version of the WADL files for the api-ref[11] site, if necessary. There is the possibility that this site will change its format, in which case these new API documents will need to be generated.

Further, the v2 API should follow keystone's model[12] of publishing the API reference documents in restructured text format to the specs repository. This would make the API much easier to document and update as new specification changes could also propose their API changes to the same repo. Also, the WADL format is very verbose and the future of this format is under question within the OpenStack documentation community[13]. The effort to make accurate documentation for sahara's API should also include the possibility of creating Swagger[14] output as the v2 API approaches stable status, this should be addressed in a more separate specification as that time approaches.

2.2.7 References

[1]: http://specs.openstack.org/openstack/api-wg/#guidelines

[2]: https://wiki.openstack.org/wiki/Sahara/api-v2

[3]: http://specs.openstack.org/openstack/api-wg/guidelines/tags.html

[4]: http://specs.openstack.org/openstack/ironic-specs/specs/kilo-implemented/api-microversions.html

[5]: http://specs.openstack.org/openstack/nova-specs/specs/kilo/implemented/api-microversions.html

[6]: https://tools.ietf.org/html/draft-nottingham-json-home-03

[7]: http://specs.openstack.org/openstack/sahara-specs/specs/liberty/unified-job-interface-map.html

[8]: https://github.com/cdent/gabbi

[9]: https://github.com/openstack/syntribos

[10]: https://github.com/redhat-cip/restfuzz

[11]: http://developer.openstack.org/api-ref.html

[12]: https://github.com/openstack/keystone-specs/tree/master/api

[13]: http://specs.openstack.org/openstack/docs-specs/specs/liberty/api-site.html

[14]: https://github.com/swagger-api/swagger-spec

Liberty summit etherpad, https://etherpad.openstack.org/p/sahara-liberty-api-v2

Mitaka summit etherpad, https://etherpad.openstack.org/p/sahara-mitaka-apiv2

2.3 Decommission of specific instance

https://blueprints.launchpad.net/sahara/+spec/decommission-specific-instance

When facing issues with a cluster it can be useful to remove an specific instance. The way that Sahara is constructed today allows the user to scale a cluster down but it will choose a random instance from the selected node group to be removed. We want to give the users the opportunity to choose which instance(s) he/she would like to remove from the cluster.

2.3.1 Problem description

Users may need to remove specific node to make cluster healthier. This is not possible today.

2.3.2 Proposed change

We will add the possibility for the user to choose the specific instance to remove from the cluster.

After selecting the node group from which the instance will be removed the user will be allowed to choose the instance or instances to be removed.

We will also allow wildcard removal, if the user wants to randomly select the instance he can just leave it blank as well as if more than one instance is being deleted the user will be able to choose each instance to be deleted or just a subset and Sahara will choose the rest.

Alternatives

Keep randomly selecting instance to be scaled down.

Data model impact

None

REST API impact

We will change the body request for scale cluster.

Currently the body should be like this:


```
{
    "add_node_groups": [
        {
            "count": 1, "name": "b-worker", "node_group_template_id": "bc270ffe-a086-4eeb-9baa-2f5a73504622"
        }
    ],
    "resize_node_groups": [
        {
            "count": 4, "name": "worker"
        }
    ]
}
```


We will change the second part to support an extra parameter:


```
{
    "add_node_groups": [
        {
            "count": 1, "name": "b-worker", "node_group_template_id": "bc270ffe-a086-4eeb-9baa-2f5a73504622"
        }
    ],
    "resize_node_groups": [
        {
            "count": 4, "name": "worker", "instances": ["instance_id1", "instance_id2"]
        }
    ]
```

}

In case the user does not specify instances to be removed the parameter will not be passed and we will act on removing with the current approach.

Other end user impact

CLI command will have a new option to select instances.

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

We will need to add a place for user to select the instances to be removed. It can be after the NG selection and we add the number of selector of instances to be removed with the option to leave it blank.

2.3.3 Implementation

Assignee(s)

Primary assignee:
Telles Nobrega

Other contributors:
None

Work Items

- Add the possibility to select an instance when scaling down
- Add CLI option to select instances to be removed
- Add UI option to select instances to be removed
- Unit tests
- Documentation

2.3.4 Dependencies

None

2.3.5 Testing

Unit tests will be needed.

2.3.6 Documentation Impact

Nothing out of the ordinary, but important to keep in mind both user and developer perspective.

2.3.7 References

None

2.4 Force Delete Clusters

https://blueprints.launchpad.net/sahara/+spec/sahara-force-delete

Let's resolve a long-standing complaint.

2.4.1 Problem description

An oft-reported issue is that a Sahara cluster hangs on deletion. This can be a major user headache, and also looks really bad. Usually the problem is not the fault of Sahara, and rather the fault of some undeletable resource.

2.4.2 Proposed change

We will make use of Heat's stack abandoning feature. Essentially, this means the stack will get deleted but leave resources in place. It's a last-ditch effort to be made when undeleteable (or slow-to-delete) resources are present in the stack.

This will be done through a new addition to Sahara API which wraps Heat stack abandon. And it's probably best to release as part of APIv2 rather than continue to break the rules that we have broken so often for APIv1.1.

Note that even though Heat stack abandon does not clean up any resources, we should not have Sahara try to clean them up manually.

The above point is justified by two things:

- This gerrit comment [0]
- If abandoning is needed, it's really hard to delete the resource anyway

It's best to create an API which wraps stack abandon, rather than just telling users to use abandon directly, because there's other cleanup to do during cluster delete. See [1] for more info.

The change will enable the following workflow: force delete gets called and cleans up cluster as best it can, then user gets to handle orphaned resources themselves. Thanks to explicit abandon through Sahara API, users are always encouraged to make sure stack is in deleting state first, so that amount of orphaned resources is minimized.

With regards to the above point, it is absolutely crucial that we do not simply enhance the regular delete call to include an abandon call. There are two key reasons to avoid that:

- In normal operation, Heat stack delete does retry
- In normal use, probably the user wants cluster to stay in deleting state until resources actually gone: force delete is just for emergencies

Alternatives

Just tell users/operators to use Heat's stack abandon manually. That's not a great choice for the reasons discussed above.

Data model impact

None

REST API impact

We'll add the following endpoint:

```
DELETE /v2/clusters/{cluster_id}/force
```

- It'll give 204 on success just like regular DELETE.
- It'll give the usual 4xx errors in the usual ways.
- It'll give 503 (on purpose) when Heat stack abandon is unavailable.
- Request body, headers, query, etc are the same as regular DELETE.

Again, best practice says make this a v2 exclusive, rather than further dirty the existing v1.1 API.

Other end user impact

Need to extend Saharaclient API bindings and OSC in order to support the new API methods.

Deployer impact

They need to enable_stack_abandon=True in heat.conf and make sure that their cloud policy does allow users to do such an operation.

We could try to do something fancy with RBAC and trusts so that Sahara service user may abandon Heat stacks on behalf of the user, when the operator wishes to restrict stack abandon. But it might not be worth messing with that...

Developer impact

None

Image Packing impact

None

Sahara-dashboard / Horizon impact

Yes, expose the functionality in the UI. We could put some warnings about force-delete's implications as well.

2.4.3 Implementation

Assignee(s)

Primary assignee:
Jeremy Freudberg

Other contributors:
None

Work Items

- Remove the last bit of direct engine code (without this change, even an abandoned stack may still be stuck, as Sahara engine is trying manual cleanup; it also entails that after this change is made, the bug of fully deleted stack but cluster still in deleting state is essentially resolved...)
- Create the new API bits and corresponding operation
- Add functionality to client and dashboard

2.4.4 Dependencies

None

2.4.5 Testing

Probably scenario tests are not strictly needed for this feature.

Beyond the obvious unit tests, there will also be updates to the API tests in the tempest plugin.

2.4.6 Documentation Impact

Nothing out of the ordinary, but important to keep in mind both operator and developer perspective.

2.4.7 References

[0] https://review.openstack.org/#/c/466778/20/sahara/service/engine.py@276

[1] https://github.com/openstack/sahara/blob/master/sahara/service/ops.py#L355

2.5 Remove Job Binary Internal

https://blueprints.launchpad.net/sahara/+spec/remove-job-binary-internal

Job binary internal is not needed now, since swift and manila are available (and possibly other storage options in the future) and these are more suitable options for storage.

2.5.1 Problem description

Job binary internal is a job binary storage option that is kept in Sahara's internal database, this option should not be available anymore since there are better storage options (swift, manila, ...).

Besides that, Sahara's internal database should be used only to Sahara's data storage. Allowing job binaries to be stored in Sahara's database isn't only unneeded, but also a possible source of problems, since it increases the size of the database. It also opens a loophole for free storage. It's definitely the wrong tool for the job.

Also, it's important to notice that this change is related to APIv2 and should not break APIv1.

2.5.2 Proposed change

This change proposes to remove job binary internal in favour of other storage options. Planning to deprecate it when APIv2 is stable, in tandem with a deprecation of APIv1. Both APIv1 and JBI will be fully removed together after APIv2 has been stable for long enough.

This change can be divided in the patches below.

- remove job binary internal from APIv2
- remove internal code that deals with job binary internal (possibly a few patches)

- remove job binary internal from database
- remove job binary internal from saharaclient
- remove job binary internal option from Horizon (sahara-dashboard)
- update documentation involving job binary internal

Alternatives

Another option for this change would be not remove job binary internal, which could bring future problems to Sahara.

Data model impact

Job binary internal will be removed from the data model. This will require a database migration.

REST API impact

Job binary internal related requests must be removed only in APIv2.

Other end user impact

Job binary internal option should not be available through Horizon.

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

Job binary internal option should not be available through Horizon.

2.5.3 Implementation

Assignee(s)

Primary assignee: mariannelinharesm

Work Items

- remove job binary internal from APIv2
- remove internal code that deals with job binary internal (possibly a few patches)
- remove job binary internal from database
- remove job binary internal from saharaclient
- remove job binary internal option from Horizon (sahara-dashboard)
- update documentation involving job binary internal
- (all but first and last steps are done in tandem with APIv1 removal)

2.5.4 Dependencies

None

2.5.5 Testing

None

2.5.6 Documentation Impact

Update documentation to related to job binary internal:

- add a warning about job binary internal's deprecation whenever APIv2 becomes stable and default.

2.5.7 References

[0] http://eavesdrop.openstack.org/meetings/sahara/2017/sahara.2017-04-06-14.00.log.txt [1] http://eavesdrop.openstack.org/meetings/sahara/2017/sahara.2017-03-30-18.00.log.txt

CHAPTER

THREE

PIKE SPECS

3.1 Deprecation of CentOS 6 images

https://blueprints.launchpad.net/sahara/+spec/deprecate-centos6-images

Starting from the Newton release, the images based on CentOS 6 are available also with CentOS 7, sometimes even with more choices for CentOS 7 version (CDH). This spec propose to deprecate and remove the support for CentOS 6 images.

3.1.1 Problem description

Keeping the support for CentOS 6 is increasinglly difficult.

The cloud image provided by the CentOS team can not be used as it is (lack of resize) so a special image should be prepared. This is documented but the default image, which should be manually regenerated, is hosted on sahara-files.mirantis.com, which will be discontinued.

Also, diskimage-builder's support for CentOS 6 is not so effective as it should be, as most of the focus is (rightfully) on CentOS 7.

Example of issues which requires a workaround:

- https://bugs.launchpad.net/diskimage-builder/+bug/1534387
- https://bugs.launchpad.net/diskimage-builder/+bug/1477179

A blocker bug right now is:

- https://bugs.launchpad.net/diskimage-builder/+bug/1698551

The (non-blocking, even if they should be blocking) gate jobs for sahara-image-builder fails due to the latter.

3.1.2 Proposed change

The support for CentOS 6 images should be deprecated starting from Pike and removed as soon as the compliance with the follows-standard-deprecation allows us to do it.

The change mainly affects sahara-image-elements. The CentOS 6 would not be built anymore by default while building all the images for a certain plugin and a warning message would be printed if one of them is selected.

The code path which checks for CentOS 6 in Sahara services should be kept as they are and not changed as long as the features is available even if deprecated; after the removal the code can be restructured, if needed, to not consider the CentOS 6 use case.

Alternatives

Keep CentOS 6 support until it is retired officially (November 30, 2020) or until diskimage-builder removes the support, but make sure that the current issues are fixed. A change is needed anyway in the sahara-image-elements jobs, as the building fails right now.

Data model impact

None

REST API impact

None

Other end user impact

Users won't be able to use CentOS 6 as base.

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

Most of the described changes are in sahara-image-elements (see above).

Sahara-dashboard / Horizon impact

Minor: remove the reference to CentOS 6 and the default cloud image from the image registration panel when the feature is removed.

3.1.3 Implementation

Assignee(s)

Primary assignee:
ltoscano

Work Items

- do not build CentOS 6 image by default for a certain plugin
- add a warning message if one of them is requested
- inform the operators (openstack-operators@) about the change to evaluate the time for the removal

3.1.4 Dependencies

None

3.1.5 Testing

If the change is implemented, the existing jobs for sahara-image-elements will only test the supported images and won't fail.

3.1.6 Documentation Impact

Add or change the list of supported base images.

3.1.7 References

None

3.2 Node Group Template And Cluster Template Portability

https://blueprints.launchpad.net/sahara/+spec/portable-node-group-and-cluster-templates

Sahara allows creation of node group templates and cluster templates. However, when theres need to create template with the same parameters on another openstack deployment one must create new template rewriting parameters by hand. This change proposes to create functions to export these templates to JSON files and import them later.

3.2.1 Problem description

Sahara provides the ability to create templates for node groups and clusters. In the case where the user has access to multiple clouds or is constantly redeploying development environment it is very time consuming to recreate all the templates. We aim to allow the user to have the option to download a template and also upload an existing template to Sahara for a quicker setup.

3.2.2 Proposed change

This change proposes to allow import and export of cluster templates and node group templates. Uses node_group_template_get(node_group_template_id) and node_group_template_create(parameters) to connect to database. Templates will be changed before export so that IDs and other sensitive information is not exported. Some additional information will be requested for import of a template. REST API changes: Node group template interface * node_group_template_export(node_group_template_id) - exports node group template to a JSON file Cluster template interface * cluster_template_export(cluster_template_id) - exports cluster template to a JSON file UI changes: Node group template interface: * field for exporting node group template * field for importing node group template - uses ngt create Cluser template interface: * field for exporting cluster template * field for importing cluster template - uses cluster create CLI changes: dataprocessing node group template export dataprocessing node group template import dataprocessing cluster template export dataprocessing cluster template import

Alternatives

A clear alternative is let things be the way they are, but it makes Sahara hard to configure and reconfigure if it was configured before.

Data model impact

None

REST API impact

- node-group-templates/{NODE_GROUP_TEMPLATE_ID}/export
- cluster-template/{CLUSTER_TEMPLATE_ID}/export

Other end user impact

Export and import of node group templates and cluster templates available in both CLI and UI.

Deployer impact

Simplified deployment, deployer can download pre existing templates if needed.

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

An option to export and another to import a template will be added. An option to import a template will have needed fields to complete the template.

3.2.3 Implementation

Assignee(s)

- Iwona
- tellesmvn

Work Items

- Add export of a node group template
- Add import of a node group template
- Add export of a cluster template
- Add import of a cluster template
- Testing
- Documentation

3.2.4 Dependencies

None

3.2.5 Testing

Unit tests will be added.

3.2.6 Documentation Impact

Documentation about new features will be added.

3.2.7 References

None

3.3 Support for S3-compatible object stores

https://blueprints.launchpad.net/sahara/+spec/sahara-support-s3

Following the efforts done to make data sources and job binaries "pluggable", it should be feasible to introduce support for S3-compatible object stores. This will be an additional alternative to the existing HDFS, Swift, MapR-FS, and Manila storage options.

A previous spec regarding this topic existed around the time of Icehouse release, but the work has been stagnant since then: https://blueprints.launchpad.net/sahara/+spec/edp-data-source-from-s3

3.3.1 Problem description

Hadoop already offers filesystem libraries with support for s3a:///path URIs, so supporting S3-compatible object stores on Sahara is a reasonable feature to add.

Within the world of OpenStack, many cloud operators choose Ceph RadosGW instead of Swift. RadosGW object store supports access through either the Swift or S3 APIs. Also, with some extra configuration, a "native" install of Swift can also support the S3 API. For some users we may expect the Hadoop S3 library to be preferable over the Hadoop Swift library as it has recently received several enhancements including support for larger objects and other performance improvements.

Additionally, some cloud users may wish to use other S3-compatible object stores, including:

- Amazon S3 (including AWS Public Datasets)
- LeoFS
- Riak Cloud Storage
- Cloudian HyperStore
- Minio
- SwiftStack
- Eucalyptus

It is clear that adding support for S3 datasources will open up a new world of Sahara use cases.

3.3.2 Proposed change

An "s3" data source type will be added, via new code in *sahara.service.edp.data_sources*. We will need utilities to validate S3 URIs, as well as to handle job configs (access key, secret key, endpoint, bucket URI).

Regarding EDP, there should not be much work to do outside of defining the new data source type, since the Hadoop S3 library allows jobs to be run against S3 seamlessly.

Similar work will be done to enable an "s3" job binary type, including the writing of "job binary retriever" code.

While the implementation of the abstraction is simple, a lot of work comes from dashboard, saharaclient, documentation, and testing.

Alternatives

Do not add support for S3 as a data source for EDP. Since the Hadoop S3 libraries are already included on the image regardless of this change, users can run data processing jobs against S3 manually. We still may wish to add the relevant JARs to the classpath as a courtesy to users.

Data model impact

None

REST API impact

None (only "s3" as a valid data source type and job binary type in the schema)

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

On most images, hadoop-aws.jar needs to be added to the classpath. Generally images with Hadoop (or related component) installed already have the JAR. The work will probably take place during the transition from SIE to image packing, so the work will probably need to be done in both places.

Sahara-dashboard / Horizon impact

Data Source and Job Binary forms should support s3 type, with fields for access key, secret key, S3 URI, and S3 endpoint. Note that this is a lot of fields, in fact more than we have for Swift. There will probably be some saharaclient impact too, because of this.

3.3.3 Implementation

Assignee(s)

Primary assignee:
Jeremy Freudberg

Other contributors:
None

Work Items

- S3 as a data source
- S3 as a job binary
- Ensure presence of AWS JAR on images
- Dashboard and saharaclient work
- Scenario tests
- Documentation

3.3.4 Dependencies

None

3.3.5 Testing

We will probably want scenario tests (although we don't have them for Manila).

3.3.6 Documentation Impact

Nothing out of the ordinary, but important to keep in mind both user and developer perspective.

3.3.7 References

None

CHAPTER

FOUR

OCATA SPECS

4.1 Data Source and Job Binary Pluggability

https://blueprints.launchpad.net/sahara/+spec/data-source-plugin

Sahara allows multiple types of data source and job binary. However, there's no clean abstraction around them, and the code to deal with them is often very difficult to read and modify. This change proposes to create clean abstractions that each data source type and job binary type can implement differently depending on its own needs.

4.1.1 Problem description

Currently, the data source and job binary code are spread over different folders and files in Sahara, making this code hard to change and to extend. Right now, a developer who wants to create a new data source needs to look all over the code and modify things in a lot of places (and it's almost impossible to know all of them without deep experience with the code). Once this change is complete, developers will be able to create code in a single directory and will be able to write their data source by implementing an abstract class. This will allow users to enable data sources that they write themselves (and hopefully contribute upstream) much more easily, and it will allow operators to disable data sources that their own stack does not support as well.

4.1.2 Proposed change

This change proposes to create the data source and job binary abstractions as plugins, in order to provide loading code dynamically, with a well defined interface. The existing types of data sources and job binaries will be refactored.

The interfaces that will be implemented are described below:

Data Source Interface

- prepare_cluster(cluster, kwargs) - Makes a cluster ready to use this data source. Different implementations for each data source: for Manila it will be mount the share, for Swift verify credentials, etc.
- construct_url(url, job_exec_id) - Resolves placeholders in data source URL.
- get_urls(cluster, url, job_exec_id) - Returns the data source url and the runtime url the data source must be referenced as. Returns: a tuple of the form (url, runtime_url).
- get_runtime_url(url, cluster) - If needed it will construct a runtime url for the data source, by the default if a runtime url is not needed it will return the native url.

- validate(data) - Checks whether or not the data passed through the API to create or update a data source is valid.
- _validate_url(url) - This method is optional and can be used by the validate method in order to check whether or not the data source url is valid.

Job Binary Interface

- prepare_cluster(cluster, kwargs) - Makes a cluster ready to use this job binary. Different implementations for each data source: for Manila it will be mount the share, for Swift verify credentials, etc.
- copy_binary_to_cluster(job_binary, cluster) - If necessary, pull binary data from the binary store and copy that data to a useful path on the cluster.
- get_local_path(cluster, job_binary) - Returns the path on the local cluster for the binary.
- validate(data) - Checks whether or not the data passed through the API to create or update a job binary is valid.
- _validate_url(url) - This method is optional and can be used by the validate method in order to check whether or not the job binary url is valid
- validate_job_location_format(entry) - Pre checks whether or not the API entry is valid.
- get_raw_data(job_binary, kwargs) - Used only by the API, it returns the raw binary. If the type doesn't support this operation it should raise NotImplementedException.

These interfaces will be organized in the following folders structure:

- services/edp/data_sources - Will contain the data source interface and the data source types implementations.
- services/edp/job_binaries - Will contain the job binary interface and the job binary types implementations.
- services/edp/utils - Will contain utility code that can be shared by data source implementations and job binary implementations. Per example: Manila data source implementation will probably share some code with the job binary implementation.

Probably some changes in the interface are possible until the changes are implemented (parameters, method names, parameter names), but the main structure and idea should stay the same.

Also a plugin manager will be needed to deal directly with the different types of data sources and job binaries and to provide methods for the operators to disable/enable data sources and job binaries dynamically. This plugin manager was not detailed because is going to be similar to the plugin manager already existent for the cluster plugins.

Alternatives

A clear alternative is let things the way they are, but Sahara would be more difficult to extend and to understand; An alternative for the abstractions defined in the Proposed Change section would be to have only one abstraction instead of two interfaces for data sources and job binaries since these interfaces have a lot in common, implementing this alternative would remove the edp/service/utilities folder letting the code more unified and compact, but job binary and data source code would be considered only one plugin, which could difficult the pluggability feature of this change (per example: the provider would not be able to disable manila for data sources, but enable it for job binaries) and because of this it was

not considered the best approach, instead we keep job binaries and data sources apart, but in contrast we need the utilities folder to avoid code replication.

Data model impact

None

REST API impact

Probably some new methods to manage supported types of data sources and job binaries will be needed (similar to the methods already offered by plugins).

- data_sources_types_list() ; job_binaries_types_list()
- data_sources_types_get(type_name) ; job_binaries_types_get(type_name)
- data_sources_types_update(type_name, data) ; job_binaries_types_update(type_name, data)

Other end user impact

None

Deployer impact

None

Developer impact

After this change is implemented developers will be able to add and enable new data sources and job binaries easily, by just implementing the abstraction.

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

4.1.3 Implementation

Assignee(s)

Primary assignee: mariannelinharesm

Other contributors: egafford

Work Items

- Creation of a Plugin for Data Sources containing the Data Source Abstraction
- Creation of a Plugin for Job Binaries containing the Job Binary Abstraction
- HDFS Plugin Intern
- HDFS Plugin Extern
- Swift Plugin
- Manila Plugin
- Allow job engines to declare which data sources/job binaries they are capable of using (this may be needed or not depending if exists a job type that does not support a particular data source or job binary type)
- Changes in the API

4.1.4 Dependencies

None

4.1.5 Testing

This change will require only changes in existing unit tests.

4.1.6 Documentation Impact

Will be necessary to add a devref doc about the abstractions created.

4.1.7 References

None

CHAPTER

FIVE

NEWTON SPECS

5.1 Designate Integration

https://blueprints.launchpad.net/sahara/+spec/designate-integration

Designate provides DNS as a service so we can use hostnames instead of IP addresses. This spec is proposal to implement this feature in Sahara.

5.1.1 Problem description

Users want to use meaningful hostnames instead of just ip addresses. Currently Sahara only changes `/etc/hosts` files when deploying the cluster and it allows to resolve vms by hostnames only through console and only between these vms. But with Designate integration the hostnames of vms could be used in the dashboard and we don't need to change `/etc/hosts`.

5.1.2 Proposed change

With this change we'll be able to act in such way: user have pre installed Designate on controller machine and set up it with network (see [0]). Also user have added DNS address to `/etc/resolv.conf` file on client machine. User creates a cluster template in which it chooses domain names for internal and external resolution. Then user launches the cluster and all its instances can be resolved by their hostnames:

1. from user client machine (e.g., links in Sahara-dashboard)
2. between instances

Designate integration will work if we point that we want to use it in sahara configs. So new config option should be added in the `default` section:

- `use_designate`: boolean variable which indicates should Sahara use Designate or not (by default is False).
- `nameservers`: list of servers' ips with pre installed Designate. This is required if 'use_designate' option is True.

Domain records format will be `instance_name.domain_name.`. Domain records will be created by Heat on create heat stack step of cluster creation process. The hostnames collision isn't expected: 1. Designate allows only unique domain names so user can't create domain with the same name in different tenants. Also domain names are created in one tenant and aren't available in another. 2. Nova allows to launch instances with the same name but Sahara adds indexes (0,1,2,...) for each instance name. The single collision case is when we launch two clusters with the same node group names and the same cluster names then Designate doesn't allow to create duplicated records so user can just change cluster name.

We should maintain backward compatibility. Backward compatibility cases:

- Old version of Openstack. Then user can switch off designate with `use_designate = false` (this value by default).
- Cluster already exists. Then user can't use designate feature.

Addresses of the domain servers should be written to `/etc/resolv.conf` files on each of vms in order to successfully resolve created domain records across these vms. It could be done with cloud-init capabilities.

Alternatives

None. We can leave all as is: now we change `/etc/hosts` files for resolving hostnames between vms.

Data model impact

Cluster, Cluster Template and Instance entities should have two new columns:

- `internal_domain_name`
- `external_domain_name`

REST API impact

None

Other end user impact

User need to pre install and setup designate by self (see [0]). Also user should change `resolv.conf` files on appropriate machines in order to resolve server with designate.

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

Cluster template creation page will contain additional 'DNS' tab with two dropdown fields: internal domain and external domain.

5.1.3 Implementation

None

Assignee(s)

Primary assignee:
msionkin (Michael Ionkin)

Work Items

- implement designate heat template
- implement writing dns server address to `resolv.conf` files
- provide backward compatibility
- add new db fields
- add tab and fields for cluster template page in Sahara dashboard
- add unit tests

5.1.4 Dependencies

- python-designateclient for Sahara-dashboard

5.1.5 Testing

Unit tests should be added.

5.1.6 Documentation Impact

This feature should be documented.

5.1.7 References

[0] http://docs.openstack.org/mitaka/networking-guide/adv-config-dns.html

5.2 CLI for Plugin-Declared Image Generation

This specification describes the next stage of the image generation epic, and proposes a CLI to generate images from the plugin-defined recipes described in the validate-image-spi blueprint.

5.2.1 Problem description

https://blueprints.launchpad.net/sahara/+spec/image-generation-cli

Sahara has historically used a single monolithic repository of DIB elements for generation of images. This poses several problems:

1) Unlike most use cases for DIB, Sahara end-users are not necessarily OpenStack experts, and are using OpenStack as a means to manage their data processing clusters, but desire to customize their nodes at the image level. As these users are concerned with data processing over cloud computing, ease of image modification is critical in the early stages of technology evaluation. The Sahara team has found that while DIB is a very powerful tool, it is not overly friendly to new adopters.

2) In order to mature to a stage at which new Sahara users can add features to the codebase cleanly, Sahara must draw firmer lines of encapsulation around its plugins. Storing all image generation recipes (whatever the implementation) in a single repository that cuts across all plugins is counter to the end goal of allowing the whole functionality of a new plugin to be exposed at a single path.

3) Sahara end users will very seldom need to modify base OS images in a particularly deep way, and will seldom need to create new images. The changes Sahara must make to images in order to prepare them for use are actually quite basic: we install packages and modify certain configuration files. Sacrificing some power and speed for some ease of use is very reasonable in our use case.

For these reasons, a means to encapsulate image generation logic within a plugin, a command line utility to exercise these recipes, and a backing toolchain that emphasizes reliability over flexibility and speed are indicated.

5.2.2 Proposed change

The https://blueprints.launchpad.net/sahara/+spec/validate-image-spi blueprint describes the first steps of this epic, enabling us to define recipes for image validation and generation of clusters from "clean" images.

CLI input

The current specification proposes that a CLI script be added to sahara at sahara.cli.sahara_pack_image.py. This script will not start a new service as others do; rather, it will use the pre-existent sahara.services.images module to run the image generation recipe on a base image through a new remote implementation. This remote implementation will use libguestfs'python API to alter the image to the plugin's specifications.

--help text for this script follows:

```
Usage: sahara-image-create
    --image IMAGE_PATH
    [--root-filesystem]
```

(continues on next page)

(continued from previous page)

```
    [--test]
    PLUGIN PLUGIN_VERSION
    [More arguments per plugin and version]
* --image: The path to an image to modify. This image will be modified
    in-place: be sure to target a copy if you wish to maintain a
    clean master image.
* --root-filesystem: The filesystem to mount as the root volume on the
    image. No value is required if only one filesystem is detected.
* --test-only: If this flag is set, no changes will be made to the
    image; instead, the script will fail if discrepancies are found
    between the image and the intended state.
[* variable per plugin and version: Other arguments as specified in
  the generation script.
* ...]
* -h, --help: See this message.
```

Both PLUGIN and PLUGIN_VERSION will be implemented as required subcommands.

If --test-only is set, the image will be packed with the `reconcile` option set to False (meaning that the image will only be tested, not changed.)

Each image generation .yaml (as originally described in the validate-images-spi spec,) may now register a set of 'arguments' as well as a set of 'validators'. This argument specification should precede the validator declaration, and will take the following form:

```
arguments:
  - java-version:
    description: The java distribution.
    target_variable: JAVA_VERSION
    default: openjdk
    required: false
    choices:
      - oracle-java
      - openjdk
```

A set of arguments for any one yaml must obey the following rules:

1) All argnames must be unique, and no argnames may collide with global script argument tokens.

2) If required is false, a default value must be provided.

3) The target_variable field is required, for clarity.

An ImageArgument class will be added to the sahara.service.images module in order that all plugins can use the same object format. It will follow the object model above.

New SPI methods

In order to facilitate the retrieval of image generation arguments, a get_image_arguments SPI method will be added to the plugin SPI. The arguments returned from this method will be used to build help text specific to a plugin and version, and will also be used to validate input to the CLI. It will return a list of ImageArguments.

A pack_image method will also be added to the plugin SPI. This method will have the signature:

```
def pack_image(self, hadoop_version, remote, reconcile=True,
               image_arguments=None)
```

This method will take an ImageRemote (see below). In most plugins, this will:

1) Validate the incoming argument map (to ensure that all required arguments have been provided, that values exist in any given enum, etc.)
2) Place the incoming arguments into an env_map per their target_variables.
3) Generate a set of ImageValidators (just as is done for image validation).
4) Call the validate method of the validator set using the remote and argument_map.

However, the implementation is intentionally vague, to allow plugins to introduce their own image packing tools if desired (as per the validate-images-spi spec.)

ArgumentCaseValidator

Now that these image definitions need to be able to take arguments, a new ArgumentCaseValidator will be added to the set of concrete SaharaImageValidators to assist image packing developers in writing clean, readable recipes. This validator's yaml definition will take the form:

```
argument_case:
    argument_name: JAVA_VERSION
    cases:
        openjdk:
            - [action]
        oracle-java:
            - [action]
```

The first value case key which matches the value of one of the variable will execute its nested actions. All subsequent cases will be skipped.

ImageRemote

A new ImageRemote class will be added to a new module, at sahara.utils.image_remote. This class will be encapsulated in its own module to allow distribution packagers the option of externalizing the dependency on libguestfs into a subpackage of sahara, rather than requiring libguestfs as a dependency of the main sahara python library package in all cases.

This class will represent an implementation of the sahara.utils.remote.Remote abstraction. Rather than executing ssh commands from the provided arguments, however, this class will execute scripts on a target image file using libguestfs' python API.

The CLI will use generate a remote targeting the image at the path specified by the 'image' argument, and use it to run the scripts which would normally be run over ssh (on clean image generation) within the image file specified.

Alternatives

We have discussed the option of bringing DIB elements into the Sahara plugins. However, this was nixed due to the issues above related to tradeoff between speed and power and usability, and because of certain testing issues (discussed in an abandoned spec in the Mitaka cycle).

It is also possible that we could maintain our current CLI in sahara-image-elements indefinitely. However, as more plugins are developed, a single monolithic repository will become unwieldy.

Data model impact

None.

REST API impact

None.

Other end user impact

The new CLI will be (at present) the only means of interacting with this feature.

Deployer impact

Packaging this script as a separate python (and Debian/RPM package) is an option worth discussing, but at this time it is intended that this tool be packed with the Sahara core.

Developer impact

This feature will reuse definitions specified in the validate-image-spi spec, and thus should have minimal developer impact from that spec.

Sahara-image-elements impact

This feature will hopefully, once it reaches full maturity and testability, supplant sahara-image-elements, and will provide the baseline for building image packing facility into the sahara API itself.

Sahara-dashboard / Horizon impact

No dashboard representation is intended at this time.

5.2.3 Implementation

Assignee(s)

Primary assignee:
egafford

Other contributors:
None

Work Items

1) Create and unit test images.py changes.
2) Test plugin-specific image packing for any plugins to implement this feature. Intended first round of plugins to implement:
3) Implement CI tests for this feature.

5.2.4 Dependencies

This feature will introduce a dependency on libguestfs, though it will only use libguestfs features present in all major distributions.

5.2.5 Testing

As this feature does not touch the API, it will not introduce new Tempest tests. However, testing the image generation process itself will require attention.

It is proposed that tests be added for image generation as each plugin implements this generation strategy, and that nightly tests be created to generate images and run these images through cluster creation and EDP testing.

These should be implemented as separate tests in order to quickly differentiate image packing failure and cluster failure.

As these recipes stabilize for any given plugin, we should begin to run these tests when any change to the sahara repository touches image generation resources for a specific plugin (which should be well-encapsulated in a single directory under each version for each plugin.) Toward the end of this epic (as we are nearing the stage of authoring the API to pack images, we may consider removing integration tests for SIE to save lab time. Still, these tests will be, compared to sahara service tests, very resource-light to run.

5.2.6 Documentation Impact

This feature should be documented in both devref (for building image generation recipes) and in userdoc (for script usage).

5.2.7 References

None.

5.3 Improve anti-affinity behavior for cluster creation

https://blueprints.launchpad.net/sahara/+spec/improving-anti-affinity

Enable sahara to distribute node creation in a more equitable manner with respect to compute hardware affinity.

5.3.1 Problem description

Current anti-affinity in sahara allows nodes in the anti-affinity group, equal to the number of hypervisors (https://bugs.launchpad.net/sahara/+bug/1426398).

If the number of nodes in the anti-affinity group are more than the number of hypervisors, sahara throws an error.

5.3.2 Proposed change

User will be able to define a ratio i.e number of nodes per hypervisor when requesting anti-affinity for a process.

The ratio would be a field while creating a cluster if user selects anti-affinity

Based on the ratio given by the user and number of nodes, more server groups will be created.

Number of server groups would be equal to the number of nodes per hypervisor.

In terms of heat templates, the server groups would be created while serializing the resources if anti-affinity is enabled for that cluster.

Instances would be allocated to those server groups while serializing the instance using "group" property of "scheduler_hints" which will be set to different server group for each instance in round robin fashion.

For allocation of server groups, following changes would be required:

- Create a parameter named SERVER_GROUP_NAMES of type list in the OS::Heat::ResourceGroup resource
- Store the server group name for each instance in the node group in this parameter. So the size of the parameter list would be equal to the number of instances in the node group
- Now the instance with index i would belong to the server group name stored at SERVER_GROUP_NAMES[i]
- This parameter will then be accessed from the scheduler hints

So in the node group template, scheduler hints will look like this,

```
"scheduler_hints": {
 "group": {
  "get_param": [SERVER_GROUP_NAMES, {"get_param": "instance_index"}]
 }
}
```

E.g

A = Number of hypervisors = 5

B = Total number of nodes in the a-a group = 10

C = Number of nodes per hypervisor = nodes:hypervisor = 2

Number of server groups = C = 2

Nodes would be distributed in each of the created server groups in round-robin fashion.

Although, placement of any node in any of the server groups does not matter because all the nodes are anti-affine.

In case the ratio given by the user in the above example is 1, user will still get an error which will be thrown by nova.

We won't allow old clusters to scale with new ratio

When a user requests to scale a cluster after the ratio has changed or requests a new ratio on an existing cluster, an error would be thrown saying "This cluster was created with X ratio, but now the ratio is Y. You will need to recreate".

Alternatives

None

Data model impact

Ratio would be a field in the cluster object

REST API impact

None

Other end user impact

The change would provision the instances without any error even in case of more nodes in the anti-affinity group than the number of hypervisors if user defines the ratio correctly.

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

Yes a field has to be added in the sahara dashboard for collecting the ratio. The field will be displayed only when anti-affinity is selected.

5.3.3 Implementation

Assignee(s)

Primary assignee:
akanksha-aha

Work Items

- Add a ratio field in sahara-dashboard
- Add the same field in sahara wherever required (Data Access Layer)
- Add a new API which creates more server groups when required
- Write Unit tests and run those tests
- Write documentation

5.3.4 Dependencies

None

5.3.5 Testing

Will need to write unit tests

5.3.6 Documentation Impact

Need to add about improved anti-affinity behavior

5.3.7 References

None

5.4 Initial Kerberos Integration

https://blueprints.launchpad.net/sahara/+spec/initial-kerberos-integration

There is no way to enable kerberos integration for clusters deployed by Sahara. By the way, default managament services like Cloudera Manager and Ambari Management console already have support of configuring Kerberos for deployed clusters. We should make initial integration of this feature in sahara for Cloudera and Ambari plugins.

5.4.1 Problem description

Users wants to use Kerberos to make clusters created by Sahara more secure.

5.4.2 Proposed change

The proposed changes will be the following:

- the initial MIT of KDC integration on cluster nodes. Also admin principal will be created for sahara usages (and sahara will store that using improved secret storage);
- ability to configure Kerberos on clusters created by using Cloudera Manager API (see the reference below [0], [1], [2]) and Ambari (see the reference below [3]) will be added;
- remote operations which are requires auth should be wrapped by ticket granting method so that sahara will be able to perform operations with hdfs and something like that;
- Oozie client should be re-implemented to reflect these changes. By default, in case when cluster is not deployed with Kerberos (like vanilla) we will continue using requests python library to bind Oozie API (without needed auth). The new Oozie client should be implemented in case of kerberos implemented. This client will use standard remote implementation and curl in order to process auth to Oozie with ticket granting. As another way, we can use request-kerberos for that goal, but it's not good solution since this lib are not python3 compatible.

New config options will be added in general section of cluster template (or cluster):

- `Enable Kerberos`: will enable the kerberos security for cluster (Ambari or CDH);

As additional enhancement support of using existing KDC server can be added. In such case, additional options are required for that:

- `Use existing KDC`: will enable using existing KDC server, additional data should be provided then;
- `Admin principal`: admin principal to have ability to create new principals;
- `Admin password`: will be hidden from API outputs and will be stored in improved secret storage;
- `KDC server hostname`: hostname of KDC server

If something additional will be needed to identify KDC server it also will be added to general section of configs.

Other possible improvements can be done after implementation of steps above. By the way, initial implementation will only include steps above.

Alternatives

None

Data model impact

None. If needed, extra field will be used for that goal.

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

All packages required for KDC infrastucture should be included on our images. If something is not presented, it will be additionally installed.

Sahara-dashboard / Horizon impact

Nothing additional is required on Sahara dashboard side, since all needed options are in general section and will be prepared as usual

5.4.3 Implementation

Assignee(s)

Primary assignee:
vgridnev (Vitaly Gridnev) and msionkin (Michael Ionkin)

Work Items

All working items are will described in Proposed change. Additionally, we can include testing initial Kerberos integration on Sahara CI, but only when all steps are completed. For sure, unit tests will be added.

5.4.4 Dependencies

Depends on OpenStack requirements. As initial idea, nothing additional should not be added to current sahara requirements, all needed packages will be included only on sahara images.

5.4.5 Testing

Sahara CI will cover that change, unit tests for sure will be added.

5.4.6 Documentation Impact

New sections can be added with description of Kerberos integration in Ambari and Cloudera.

5.4.7 References

[0] https://github.com/cloudera/cm_api/blob/f4431606a690d95208457a64d1cc2610d9cfa2bf/python/src/cm_api/endpoints/cms.py#L134 [1] https://github.com/cloudera/cm_api/blob/f4431606a690d95208457a64d1cc2610d9cfa2bf/python/src/cm_api/endpoints/clusters.py#L585
[2] http://www.cloudera.com/documentation/enterprise/5-5-x/topics/cm_sg_using_cm_sec_config.html [3] https://cwiki.apache.org/confluence/display/AMBARI/Automated+Kerberizaton#AutomatedKerberizaton-TheRESTAPI

5.5 Adding pagination and sorting ability to Sahara

https://blueprints.launchpad.net/sahara/+spec/pagination

This specification describes an option for pagination ability which will admit to work with objects in Sahara with API, CLI and Dashboard. Also this specification describes sorting ability to Sahara API.

5.5.1 Problem description

We are working on adding the ability for pagination to work with objects. But having too many objects to be shown is not very user friendly. We want to implement pagination so the dashboard can split the list of pages making the UI better. User has ability to sort objects by columns in dashboard, but if we add pagination ability, we will need to add sorting ability in API, because ordering on UI side only will cause inconvenience since the service will continue returning pages ordered by the default column.

5.5.2 Proposed change

Will be added four optional parameters in API GET requests, which return a lists of objects.

Note: Now elements on lists sorted by date of creation.

`marker` - index of the last element on the list, which won't be in response.

`limit` - maximum count of elements in response. This argument must be positive integer number. If this parameter isn't passed, in response will be all the elements which follow the element withăidăinămarker parameter. Also ifămarkerăparameter isn't passed, response will contain first objects of the list with count equalălimit. If both parameters aren't passed, API will work as usual.

`sort_by` - name of the field of object which will be used by sorting. If this parameter is passed, objects will be sorting by date of creation, otherwise by this field.

The field can possess one of the next values for every object:

For Node Group Template: `name`, `plugin`, `version`, `created_at`, `updated_at`

For Cluster Templates: `name`, `plugin`, `version`, `created_at`, `updated_at`

For Clusters: `name`, `plugin`, `version`, `status`, `instance_count`

For Job Binaries and Job Binaries Internal `name`, `create`, `update`

For Data Sources: `name`, `type`, `create`, `update`

For Job Templates: `name`, `type`, `create`, `update`

For Jobs: `id`, `job_template`, `cluster`, `status`, `duration`

By default Sahara api will return list in ascending order. Also if the user wants a descending order list, he can use - prefix for `sort_by` argument.

Examples:

Get list of jobs in ascending order sorted by name.

request `GET http://sahara/v1.1/775181/jobs?sort_by=name`

Get list of jobs in descending order sorted by name.

request `GET http://sahara/v1.1/775181/jobs?sort_by=-name`

For convenience, collections contain atom "next" and "previous" markers. The first page on the list doesn't contain a previous marker, the last page on the list doesn't contain a next link. The following examples illustrate pages in a collection of cluster templates.

Example:

Get one cluster template after template with `id=3`.

request

`GET http://sahara/v1.0/775181/cluster-templates?limit=1&marker=3`

response

```
{
    "cluster_templates": [
        {
            "name": "cluster-template",
"plugin_name": "vanilla",
            "id": "4",
            "node_groups": [
                {
                    "name": "master",
                },
                {
                    "name": "worker",
                }
            ],
        }
    ],
    "markers":
        {
            "next": "32",
            "previous": "22"
        }
}
```

Example: Let cluster template with id = 5 will be the last of collection. Response will contain only "previous" link.

request

`GET http://sahara/v1.0/775181/cluster-templates?limit=1&marker=4`

response

```
{
   "cluster_templates":[
      {
         "description":"",
         "node_groups":[
            {
```

(continues on next page)

(continued from previous page)

```
                "name":"master",
            },
            {
                "name":"worker",
            }
        ],
        "name":"cluster-template-2",
        "id":"5",
    }
  ],
  "markers":
      {
          "previous": "3"
      }
}
```

Alternatives

None

Data model impact

None

REST API impact

Add ability get `marker`, `limit`, `sort_by` parameters in next requests:

Sahara API v1.0

`GET /v1.0/{tenant_id}/images`

`GET /v1.0/{tenant_id}/node-group-templates`

`GET /v1.0/{tenant_id}/cluster-templates`

`GET /v1.0/{tenant_id}/clusters`

Sahara API v1.1

`GET /v1.1/{tenant_id}/data-sources`

`GET /v1.1/{tenant_id}/job-binary-internals`

`GET /v1.1/{tenant_id}/job-binaries`

`GET /v1.1/{tenant_id}/jobs`

`GET /v1.1/{tenant_id}/job-executions`

Sahara API v2

`GET /v2/cluster-templates`

`GET /v2/clusters`

`GET /v2/data_sources`

`GET /v2/images`

`GET /v2/job-binaries`

`GET /v2/jobs`

`GET /v2/job-templates`

`GET /v2/node-group-templates`

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

Pagination will be added to Sahara-Dashboard via Horizon abilities. Now we are using `DataTable` class to represent lists of data objects. This class supports pagination abilities.

5.5.3 Implementation

Assignee(s)

Primary assignee:
mlelyakin (mlelyakin@mirantis.com)

Work Items

- Adding ability of take arguments `marker`, `limit` in Sahara API
- Adding unit tests for new features.
- Adding ability of take argument `sort_by` in Sahara API
- Adding this abilities in Sahara CLI Client
- Adding this abilities in Dashboard
- Documented pagination and sorting features

5.5.4 Dependencies

None

5.5.5 Testing

Will be covered with unit tests.

5.5.6 Documentation Impact

Will be adding to documentation of Sahara API.

5.5.7 References

None

5.6 Admin API for Managing Plugins

https://blueprints.launchpad.net/sahara/+spec/plugin-management-api

This is a proposal to implement new admin API for management plugins for different projects. Also, smarter deprecation can be implemented with a new management API for plugins.

5.6.1 Problem description

Right now we have the following problems:

- plugin state is totally unclear for user. From the user point of view, there are only default plugins, which are enabled for all projects;
- deprecation of the particular plugin version is not perfect right now: validation error with deprecation note will be raised during creation - which is not so good for user;
- all plugins and versions are enabled for all projects, which is not perfect. Someone wants to use only Ambari plugin, and other one wants to use only CDH plugin (or a particular version)

5.6.2 Proposed change

It's proposed to implement several new API calls for plugins management. Actually, there is plan to support that in API v1.1 correctly so that we can put this to API v2 without additional stuff.

First of all we need to enable all plugins by default. There is no sense to continue separating default and non-default plugins. There is an agreement that having default/non-default plugins is redundant, just because such process don't allow to deliver probably non-stable plugins at least for testing in several projects.

Additionally as a part of this blueprint migration of `hadoop_version` to `plugin_version` should be done to keep this api consistent with new management API.

New database should be created to store all required metadata about current state of each plugin. This metadata is a combination of labels for plugins itself and for each version of the plugin. The plugin label is an indicator of plugin state (or version) that we will help user to understand the current state of plugin itself (like a stability, deprecation status) or of some features. If no metadata will be stored in DB, plugin SPI method will help us to return default metadata for plugin. This also will help us to avoid possible issues with upgrades from old releases of sahara.

This metadata should describe each plugin and its versions. The example of return value of this plugin SPI method is the following:

```
{
   "version_labels": {
       "2.3": {
           "stable": {
               "status": true
           },
           "enabled": {
               "status": true
           }
       },
       "2.2": {
           "deprecated": {
               "status": true
           },
           "enabled": {
               "status": true
           }
       }
   },
   "plugin_labels": {
       "enabled": {
           "status": false,
       }
   }
}
```

PluginManager on requests of all plugins will collect all data stored in DB for plugins and will accumulate that data with default data for plugins without entry in DB. Also, for each label entry manager will additionally put the description of the label and possibility of changing of label by admins. The collected data will be exposed to the user. See example of return value below in REST API section.

The initial set of labels is the following:

- `enabled` if plugin is enabled, then each user will be able to use plugin for cluster creation and will have ability to perform all CRUD operations on top the cluster. If plugins doesn't have this label, only deletion can be performed with this cluster.
- `hidden` if plugin is hidden, it's still available for performing actions using saharaclient, but it will be hidden from UI side and CLI. It's special tag for fake plugin.
- `stable` plugin is stable enough to be used. Sahara CI should be enabled for plugin to prove it's stability in terms of cluster creation and running EDP jobs. This label can't be removed. Will not be stored in DB and will be handled by plugin SPI method only.
- `deprecated` plugin is deprecated. Warnings about deprecation will be shown for this plugin. Not intended to be used. Sahara CI (nightly) will continue testing this plugin to prove it's still works well. Label can't be removed. Recommendations should be provided about what operations are available for this cluster.

Admin user will be able to perform `PATCH` actions with labels via API, if label is really can be removed. `oslo_policy` will be used for handling that user have admin role. Only status can be changed for each label. Mutability and description can't be changed.

Alternatives

None

Data model impact

New table is needed for this feature to store data about plugin tags.

```
+----------------+--------------+
| plugins        | column type  |
+----------------+--------------+
| tenant_id      | String       |
+----------------+--------------+
| plugin_labels  | JsonDictType |
+----------------+--------------+
| version_labels | JsonDictType |
+----------------+--------------+
| id (Unique)    | String       |
+----------------+--------------+
| name           | String       |
+----------------+--------------+
```

An simple example of stored data:

```
{
   'name': "fake",
   "plugin_labels": {
       "enabled": {
           "status": true,
       }
```

(continues on next page)

(continued from previous page)

```
    },
    "tenant_id": "uuid",
    "id": "uuid just to be unique",
    "version_labels": {
        "0.1": {
            "enabled": {
                "status": true
            }
        }
    }
}
```

REST API impact

There are bunch of changes in REST API are going to be done.

Endpoint changes:

1. for `GET /plugins` to following output will be expected after implementation. All labels will be additionally serialized with description, mutability.

```
{
    "plugins": [
        {
            "description": "HDP plugin with Ambari",
            "versions": [
                "2.3",
                "2.4",
            ],
            "name": "ambari",
            "plugin_labels": {
                "enabled": {
                    "description": "Indicates that plugin is switched on",
                    "mutable": true,
                    "status": true
                }
            },
            "version_labels": {
                "2.3": {
                    "enabled": {
                        "description": "Indicates that version is switched on",
                        "mutable": true,
                        "status": false,
                    },
                    "deprecated": {
                        "description": "Plugin is deprecated, but can be used"
                        "mutable": false,
                        "status": true
                    },
```

(continues on next page)

(continued from previous page)

```
                    "stable": {
                        "description": "Plugin stability",
                        "mutable": false,
                        "status": false
                    }
                },
                "2.4": {
                    "enabled": {
                        ..
                    },
                    "stable": {
                        ..
                    },
                },
            },
            "title": "HDP Plugin"
        },
    ]
}
```

2. new `PATCH /plugins/<name>` which is intended for updating tags for plugin or/and its versions. Update will be done successfully if all modified labels are mutable. Validation will be done for user if updating only status of each labels. To update a label you need to send request with only this label in body. Mutability and description are fields that can't be changed.

```
{
    "plugin_labels": {
        "enabled": {
            "status": false,
        }
    }
    "version_labels: {
        "2.3": {
            "enabled": {
                "status": true,
            },
        },
        "2.4": {
            "enabled": {
                "status": false,
            },
        },
    }
}
```

Other end user impact

New CLI will be extended with plugin updates. Warnings about deprecation label will be added too.

Deployer impact

Nothing additional is required from deployers; anyway we should notify about new default value for `plugins` option.

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

Things to do:

1. New tab for management plugins should be implemented. All labels will be shown in this tab. Each label will have checkboxes that will add this label to plugin. Only admin will have ability to produce changes.
2. Warning regarding deprecation label will be added to templates/cluster creation tabs. If the only plugin enabled we will not have dropdown for plugin choice, and the same thing for version. If the only plugin and version is enabled, plugin choice action will be skipped.

5.6.3 Implementation

Assignee(s)

Primary assignee:
vgridnev (Vitaly Gridnev)

Work Items

The following items should be covered:

- enable all plugins by default;
- implement database side;
- new API methods should be added;
- plugin SPI method for default metadata;
- document new api features in API docs;
- python-saharaclient implementation;

- sahara-dashboard changes

5.6.4 Dependencies

Depends on OpenStack requirements

5.6.5 Testing

Feature will covered by unit tests.

5.6.6 Documentation Impact

All plugin labels should be documented properly.

5.6.7 References

None

5.7 Allow creation of python topologies for Storm

In order to allow user to develop storm topologies that we can also can call storm jobs in a pure python form we are adding support to Pyleus in Sahara. Pyleus is a framework that allows the creation of Storm topologies in python and uses yaml to wire how the flow is going to work.

5.7.1 Problem description

https://blueprints.launchpad.net/sahara/+spec/python-storm-jobs

Storm is the plugin in Sahara responsible for real time processing. Storm is natively written in Java, being the common languange to write topologies. Storm allows topologies to be written in different languages, including python, but the default way to implement this still requires a java shell combining the python components together being not very pythonic.

5.7.2 Proposed change

We are OpenStack and we love our python, so in order to allow OpenStack users to create a submit python written topologies we propose to integrate the Pyleus framework into the Storm plugin. Pyleus allows the user to create Storm topologies components in python and it provides an abstraction to help that construction, making the implementation easier. Also it uses a yaml file that will be used to compile the topology. The final object of the compilation is a jar file containing the python written topology. The change we need to do is to integrate pyleus command line into the Storm plugin to submit the topologies using its CLI instead of Storm's. The overall UX will remain the same, since the user will upload a jar file and start a topology. We will create a new job type for Storm called pyleus so the plugin can handle the new way of submitting the job to the cluster.

The command line will like this:

- pyleus submit [-n NIMBUS_HOST] /path/to/topology.jar
- pyleus kill [-n NIMBUS_HOST] TOPOLOGY_NAME
- pyleus list [-n NIMBUS_HOST]

Alternatives

The option is to leave Storm as it is, accepting only default jobs java or python.

Data model impact

None.

REST API impact

There will be a minor REST API impact since we are introducing a new Job Type.

Other end user impact

None.

Deployer impact

None.

Developer impact

None.

Sahara-image-elements impact

We will need to install pyleus on the Storm images.

Sahara-dashboard / Horizon impact

Minor changes will be made to add new Job Type.

5.7.3 Implementation

Assignee(s)

Primary assignee:
tellesmvn

Other contributors:
None

Work Items

1) Create new Job type.
2) Change Storm plugin the deal with the new job type.
3) Implement tests for this feature.

5.7.4 Dependencies

None.

5.7.5 Testing

To this point only unit tests will be implemented.

5.7.6 Documentation Impact

This feature should be documented in the user doc.

5.7.7 References

None.

5.8 Refactor the sahara.service.api module

https://blueprints.launchpad.net/sahara/+spec/v2-api-experimental-impl

The HTTP API calls that sahara receives are processed by the `sahara.api` package, the functions of this module then call upon the `sahara.service.api` and `sahara.service.edp.api` modules to perform processing before being passed to the conductor. To help accommodate future API changes, these modules should be refactored to create a more unified package for future implementors.

5.8.1 Problem description

The current state of the API service level modules can be confusing when comparing it to the API route level modules. This confusion can lead to misunderstandings in the way the service code interacts with the routing and conductor modules. To improve the state of readability, and expansion, this spec defines a new layout for these modules.

More than confusion, as the API is being reworked for the new version 2 features there will need to be additions in the service layer to assist the endpoint and JSON notational changes. Refactoring these modules will create a more clear pathway for adding to the sahara API.

5.8.2 Proposed change

This change will create a new package named `sahara.service.api` which will contain all the service level API modules. This new package will create a unified location for service level API changes, and provide a clear path for those wishing to make said changes. The new package will also contain the base service level files for the v2 API.

The new package layout will be as follows:

```
sahara/service/api/__init__.py
                   v10.py
                   v11.py
                   v2/__init__.py
                      clusters.py
                      cluster_templates.py
                      data_sources.py
                      images.py
                      job_binaries.py
                      job_executions.py
                      jobs.py
                      job_types.py
                      node_group_templates.py
                      plugins.py
```

This new layout will provide a clean, singular, location for all service level API code. The files created for the `sahara.service.api.v2` package will be simple copies of the current functionality.

Alternatives

One alternative is to do nothing and leave the `sahara.service.api` and `sahara.service.edp.api` modules as they are and create new code for v2 either in those locations or a new file. A downside to this approach is that it will be less clear where the boundaries between the different versions will exist. This will also leave a larger questions as to where the new v2 code will live.

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

This change should improve the developer experience with regard to creating and maintaining the API code as it will be more clear which modules control each version.

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

5.8.3 Implementation

Assignee(s)

Primary assignee:
Michael McCune (elmiko)

Work Items

- create new package
- move current implementations into new package
- add v2 related service files
- fix outstanding references
- add documentation to v2 developer docs

5.8.4 Dependencies

None

5.8.5 Testing

As these changes will leave the interfaces largely intact, they will be tested through our current unit and tempest tests. The will not require new tests specifically for this change.

5.8.6 Documentation Impact

A new section in the API v2 developer docs will be added to help inform about the purpose of these files and how they will be used in the work to implement features like JSON payload changes.

5.8.7 References

None

5.9 Refactor the logic around use of floating ips in node groups and clusters

https://blueprints.launchpad.net/sahara/+spec/refactor-use-floating-ip

Currently, there is a boolean in the configuration file called *use_floating_ips* which conflates the logic around the existence of floating ips for instances and the use of floating ips for management by sahara. This logic should be refactored.

5.9.1 Problem description

The *use_floating_ips* boolean when set True has the following implications:

- All instances must have a floating ip, except for the case of using a proxy node.
- Every node group therefore must supply a floating ip pool value when using neutron, and nova must be configured to auto-assign floating ips when using nova networking.
- Sahara must use the floating ip for management.
- These requirements are in force across the application for every user, every node group, and every cluster.

When *use_floating_ips* is False and neutron is used:

- Any node group template that has a floating ip pool value set is not usable and will fail to launch. Again, there is an exception for this when using proxy nodes.
- Therefore simply modifying the value of *use_floating_ips* in the sahara configuration file will invalidate whole groups of existing, functioning templates.

As we move toward a world where virtual and baremetal clusters co-exist, we need to modify the logic around floating ips to provide more flexibility. For instance, a baremetal cluster that uses a flat physical network with no floating ips should be able to co-exist in sahara with a VM cluster that uses a virtual network with floating ips (currently this is not possible).

As nova, neutron, and ironic make further advances toward hybrid networking and instance scenarios, sahara needs the flexibility to adapt to new features. The current conflated logic is a roadblock to this flexibility.

5.9.2 Proposed change

Logically the change is very simple, but implementation will require careful analysis and testing since knowledge of floating ips and the use of nova vs neutron networking is spread throughout the code base. This becomes particularly complex in the logic around ssh connections and proxy gateways and in the quota logic.

The proposed logical change goes like this:

- The semantics of *use_floating_ips* will be changed to mean "if an instance has a floating ip assigned to it, sahara will use that floating ip for management, otherwise it will use the internal ip". This flag will impose no other requirements, and no other expectations should be associated with it.
- In the case of neutron networking, if a node group specifies a floating ip pool then the instances in that node group will have floating ips. If the node group does not specify a floating ip pool this will not be an error but the instances will not have floating ips.
- In the case of nova networking, if nova is configured to auto-assign floating ips to instances then instances will have floating ips. If nova does not assign a floating ip, the instances will not have floating ips.
- In the case of neutron networking, *use_namespaces* set to True will continue to mean "use ip netns to connect to machines that do not have floating ips" but each instance will have to be checked individually to determine if it has a floating ip assigned. It will no longer be valid to test *CONF.use_namespases and not CONF.use_floating_ips*

Alternatives

None. This logic has been in sahara since the beginning when nova networking was exclusively used and things were simpler; it's time to refactor.

Data model impact

None

REST API impact

None, although changes are needed in the validation code. The API itself does not change, but what is semantically allowable does change (ie, whether or not a node group *must* have a floating ip pool value)

Other end user impact

Users will need to be educated on the shifting implications of setting *use_floating_ips* and the choice they have when configuring node groups.

Deployer impact

This change *should be* transparent to existing instances of sahara. Templates that are functioning should continue to function, and clusters that are running should not be affected. What will change is the ability to control use of floating ips in new clusters.

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

Uncertain. In the past there were configuration settings for the sahara dashboard that touched on floating ip use. The current configuration parameters should be reviewed to see if this holds any implication for horizon.

5.9.3 Implementation

Assignee(s)

Trevor McKay has produced a patch for this which should be fairly complete, but he is unable to finish it. He was in the testing phase when work on this stopped, with fair confidence that the solution works in neutron (but more testing needs to be done in a nova networking environment)

Primary assignee: tellesnobrega

Other contributors: tmckay

Work Items

- Refactor floating-ip use
- Implement tests

5.9.4 Dependencies

None

5.9.5 Testing

Unit tests are sufficient to cover changes to template validation routines and logical flow in sahara (is sahara in a particular case trying to use a floating ip or not?)

Scenario tests should be constructed for both values of *use_floating_ips*, for both neutron and nova networking configurations, and for node groups with and without floating ip pool values.

5.9.6 Documentation Impact

The new implications of *use_floating_ips* should be covered in the documentation on configuration values and set up of nova in the nova networking case.

It should also be noted in discussion of node group templates that floating ip pool values are no longer required or disallowed based on the value of *use_floating_ips*

As mentioned above, it's unclear whether anything needs to change in sahara dashboard configuration values. If something does change, then horizon docs should be changed accordingly.

5.9.7 References

None

5.10 Run Spark jobs on vanilla Hadoop 2.x

https://blueprints.launchpad.net/sahara/+spec/spark-jobs-for-vanilla-hadoop

This specification proposes to add ability to run Spark jobs on cluster running vanilla version of Hadoop 2.x (YARN).

5.10.1 Problem description

Support for running Spark jobs in stand-alone mode exists as well as for CDH but not for vanilla version of Hadoop.

5.10.2 Proposed change

Add a new edp_engine class in the vanilla v2.x plugin that extends the SparkJobEngine. Leverage design and code from blueprint: https://blueprints.launchpad.net/sahara/+spec/spark-jobs-for-cdh-5-3-0

Configure Spark to run on YARN by setting Spark's configuration file (spark-env.sh) to point to Hadoop's configuration and deploying that configuration file upon cluster creation.

Extend sahara-image-elements to support creating a vanilla image with Spark binaries (vanilla+spark).

Alternatives

Withouth these changes, the only way to run Spark along with Hadoop MapReduce is to run on a CDH cluster.

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

Requires changes to sahara-image-elements to support building a vanilla 2.x image with Spark binaries. New image type can be vanilla+spark. Spark version can be fixed at Spark 1.3.1.

Sahara-dashboard / Horizon impact

None

5.10.3 Implementation

Assignee(s)

Primary assignee:
None

Work Items

New edp class for vanilla 2.x plugin. sahara-image-elements vanilla+spark extension. Unit test

5.10.4 Dependencies

Leveraging blueprint: https://blueprints.launchpad.net/sahara/+spec/spark-jobs-for-cdh-5-3-0

5.10.5 Testing

Unit tests to cover vanilla engine working with Spark.

5.10.6 Documentation Impact

None

5.10.7 References

None

CHAPTER

SIX

MITAKA SPECS

6.1 Add ability of suspending and resuming EDP jobs for sahara

https://blueprints.launchpad.net/sahara/+spec/ add-suspend-resume-ability-for-edp-jobs

This spec is to allow suspending and resuming edp jobs in sahara.

6.1.1 Problem description

Currently sahara does not allow suspending and resuming edp jobs. But in some use cases, for Example, one edp job containing many steps, after finishing one step, user want to suspend this job and check the output, and then resume this job. So by adding suspending and resuming ability to sahara edp engine, we can have different implementation for different engine.(oozie,spark,storm etc)

6.1.2 Proposed change

Add one api interface in sahara v11 API.

Define suspend_job() and resume_job() interface in sahara base edp engine, then implement this interface to the oozie engine. (Spark and storm engine will be drafted in later spec)

Add "SUSPENDED" and "PREPSUSPENDEDED" in the sahara. only the job's status is "RUNNING" or "PREP" can we suspend this job. and make the job's status shown as "SUSPENDED" or "PREPSUSPENDED".

Add a validation dict named suspend_resume_supported_job_type = {} to check which job type is allowed to suspend and resume when request comes in.

If the job's status is not in RUNNING or PREP, for example, job is already finished, we do validation check, and there is no suspend or resume action.

Example of suspending or resuming an edp job

PATCH /v1.1/{tenant_id}/job-executions/<job_execution_id>

response:

HTTP/1.1 202 Accepted Content-Type: application/json

Add one api interface in the python-sahara-client

For oozie implementation, we just call oozie client to invoke suspend and resume API.

For spark and storm implementation, there is no implementation now, and we will add them later.

Alternatives

None

Data model impact

None

REST API impact

Add one API interface. PATCH /v1.1/{tenant_id}/job-executions/<job_execution_id>

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

add two combobox item named "suspend job" and "resume job" option at the right side of the Job list table.

6.1.3 Implementation

Assignee(s)

Primary assignee:
luhuichun(lu huichun)

Other contributors:
None

Work Items

- add one api in v11
- add suspend_job and resume_job in base engine and oozie engine
- add two new job status "SUSPENDED" and "PREPSUSPENDED".
- add two api interface in python-sahara-client
- modify sahara api reference docs
- Add task to update the WADL at api-site

6.1.4 Dependencies

None.

6.1.5 Testing

unit test in edp engine add scenario integration test

6.1.6 Documentation Impact

Need to be documented.

6.1.7 References

oozie suspend and resume jobs implementation https://oozie.apache.org/docs/4.0.0/CoordinatorFunctionalSpec.html

6.2 Allow 'is_public' to be set on protected resources

Since *is_public* is meta-information on all objects rather than content, sahara should allow it to be changed even if *is_protected* is True. This will make it simpler for users to share protected objects.

https://blueprints.launchpad.net/sahara/+spec/allow-public-on-protected

6.2.1 Problem description

Currently checks on *is_protected* prevent any field in an object from being modified. This guarantees that the content of the object will not be changed accidentally.

However, *is_public* is an access control flag and does not really pertain to object content. In order to share a protected object, a user must currently set *is_protected* to False while making the change to *is_public*, and then perform another operation to set the *is_protected* flag back to True.

6.2.2 Proposed change

As a convenience, allow *is_public* to be modified even if *is_protected* is True. The *is_public* field will be the only exception to the normal checks.

Alternatives

Leave it unchanged

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

This may require a Horizon change (croberts please comment)

6.2.3 Implementation

Assignee(s)

Primary assignee:
tmckay

Other contributors:
croberts

Work Items

Modify *is_protected* checks in the sahara engine Modify unit tests Horizon changes

6.2.4 Dependencies

None

6.2.5 Testing

Unit tests

6.2.6 Documentation Impact

None, unless there is a current section discussing protected/public

6.2.7 References

None

6.3 add cdh 5.5 support into sahara

https://blueprints.launchpad.net/sahara/+spec/cdh-5-5-support

This specification proposes to add CDH 5.5 plugin with Cloudera Distribution of Hadoop and Cloudera Manager in Sahara.

6.3.1 Problem description

Now we have already supported plugins for CDH 5.3.0 and 5.4.0 versions in liberty. With the release of the CDH 5.5.0 by Cloudera, we can add the new version support into Sahara.

6.3.2 Proposed change

Since we already support 5.4.0, we can follow the current implemention to avoid too many changes. We must guarantee all the services supported in previous version work well in 5.5.0. Supporting for new services in CDH 5.5.0 will be discussed later.

Cloudera starts to support ubuntu 14.04 in CDH 5.5.0. So we decide to provide Ubuntu 14.04 as other plugins do. And the building for Ubuntu 14.04 image with CDH 5.5 should aslo be supported in sahara-image-elements project. CentOS 6.5 will still be supported.

Due to the refactoring for previous CDH plugin versions, we should not merge patches related with 5.5.0 until all the refactoring patches are merged.

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

Sahara-image-elements support for CDH 5.5.0 need to be done.

Sahara-dashboard / Horizon impact

None

6.3.3 Implementation

Assignee(s)

Primary assignee:
jxwang92 (Jaxon Wang)

Other contributors:
None

Work Items

The work items will be:

- Add python codes in sahara/sahara/plugins/cdh/v5_5_0.
- Add service resource files in sahara/sahara/plugins/cdh/v5_5_0/resources.
- Add test cases including unit and scenario.
- Test and evaluate the change.

6.3.4 Dependencies

None

6.3.5 Testing

Follow what exsiting test cases of previous version do.

6.3.6 Documentation Impact

Cloudera Plugin doc needs some little changes. http://docs.openstack.org/developer/sahara/userdoc/cdh_plugin.html

6.3.7 References

None

6.4 Code refactoring for CDH plugin

https://blueprints.launchpad.net/sahara/+spec/cdh-plugin-refactoring

This spec is to do some refactoring to the code to allow easier support to new versions in the future.

6.4.1 Problem description

CDH plugin contains many duplicated code. Current implementation extracts some general and base behavior of the plugin, and each version has its own implementation for something not included in base classes and modules. But there are many overlaps between versions because of the downward compatibility. For example, sahara.plugins.cdh.v5.config_helper extends sahara.plugins.cdh.db_helper, but functions such as get_plugin_configs are written again in sahara.plugins.cdh.v5_3_0.config_helper.

And currently the low test coverage of CDH plugin makes it hard to guarantee the quality of the new code after refactoring. So some new unit test cases need to be added. And some old test cases may be altered according to refactoring.

6.4.2 Proposed change

For duplicate codes in each version of plugins, move them to the base class. In validation module, function validate_cluster_creating is too long to be read easily. Seprate it into serveral small clearly functions. We can encapsulate funtions in module into a class for better extensibility. ClouderaUtils and deploy modules are not going to be changed util CDH v5 are totally removed, because these modules' codes in v5 are quite different from other versions.

Alternatives

There is another way, just let a new version extends an old one instead of all extends from base. This may bring problems when deperate an old version.

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

6.4.3 Implementation

Assignee(s)

Primary assignee:
jxwang

Work Items

This will require following changes:

- move the duplicate codes to base class. Files need to be modified: sahara/plugin/cdh/version/edp_engine.py: EdpOozieEngine, EdpSparkEngine sahara/plugin/cdh/version/plugin_utils.py: PluginUtils sahara/plugin/cdh/version/versionhandler.py: VersionHandler sahara/plugin/cdh/version/config_helper.py sahara/plugin/cdh/version/validation.py
- Separate Validation.validate_cluster_creating function
- Add unit test case for low covered modules.
- Remove useless test cases dedicated for refactoring useless test cases in: sahara/tests/unit/plugins/cdh/v5/test_versionhandler.py sahara/tests/unit/plugins/cdh/v5_3_0/test_versionhandler.py sahara/tests/unit/plugins/cdh/v5_4_0/test_versionhandle

6.4.4 Dependencies

None

6.4.5 Testing

Before starting refactoring, keep current scenario test and provide new unit tests to ensure the CDH works as well as before. For each version, unit tests are aslo added individually. Test cases are written for the codes before refactoring, so we may need a little changes for new codes.

6.4.6 Documentation Impact

None

6.4.7 References

None

6.5 Implement Sahara cluster verification checks

https://blueprints.launchpad.net/sahara/+spec/cluster-verification

Now we don't have any check to take a look at cluster processes status through Sahara interface. Our plans is to implement cluster verifications and ability to re-trigger these verifications for the particular cluster.

6.5.1 Problem description

Sahara doesn't have any check for cluster processes health monitoring. Cluster can be broken or unavailable but Sahara will still think that in ACTIVE status. This may result in end-users losses and so on.

6.5.2 Proposed change

First of all let's make several important definitions in here. A cluster health check is a perform a limited functionality check of a cluster. For example, instances accessibility, writing some data to `HDFS` and so on. Each health check will be implemented as a class object, implementing several important methods from an abstract base class(abc):

```
class ExampleHealthCheck(object):
    def __init__(self, cluster, *args, **kwargs):
        self.cluster = cluster
        # other stuff

    @abc.abstractmethod
    def available(self):
        # method will verify
```

(continues on next page)

(continued from previous page)

```
    @abc.abstractmethod
    def check(self):
        # actual health will in here

    def execute(self):
        # based in availability of the check and results
        # of check will write correct data into database
```

The expected behaviour of `check` method of a health check is return some important data in case when everything is ok and in case of errors to raise an exception with detailed info with failures reasons. Let's describe important statuses of the health checks.

- `GREEN` status: cluster in healthy and health check passed correctly. In description in such case we can have the following info: `HDFS available for writing data` or `All datanodes are active and available`.
- `YELLOW` status: `YellowHealthError` will be raised as result of the operation, it means that something is probably wrong with cluster. By the way cluster is still operable and can be used for running jobs. For example, in exception message we will see the following information: `2 out of 10 datanodes are not working`.
- `RED` status: `RedHealthError` will be raised in such case, it means that something is definitely wrong we the cluster we can't guarantee that cluster is still able to perform perations on it. For example, `Amount of active datanodes is less then replication` is possible messange in such case.
- `CHECKING` state, health check is still running or was just created.

A cluster verification is a combination of the several cluster health checks. Cluster verification will indicate the current status for the cluster: `GREEN`, `YELLOW` or `RED`. We will store the latest verification for the cluster in database. Also we will send results of verifications to Ceilometer, to view progress of cluster health.

Also there is an idea of running several jobs as part of several health checks, but it will be too harmful for the cluster health and probably will be done later.

Also we can introduce periodic task for refreshing health status of the cluster.

So, several additional options in `sahara.conf` should be added in new section `cluster_health_verification`:

- `enable_health_verification` by default will be True, will allow disabling periodic cluster verifications
- `verification_period` variable to define period between two consecutive health verifications in periodic tasks. By default I would suggest to run verification once in 10 minutes.

Proposed checks

This section is going to describe basic checks of functionalities of clusters. Several checks will affect almost all plugins, few checks will be specific only for the one plugin.

Basic checks:

There are several basic checks for all possible clusters in here.

- Check to verify all instances access. If some instances are not accessible we have `RED` state.

- Check that all volumes mounted. If some volume is not mounted, we will have `YELLOW` state.

HDFS checks:

- Check that will verify that namenode is working. Of course, `RED` state in bad case. Actually this will affects only vanilla plugin clusters and clusters deployed without HA mode in case of CDH and Ambari plugins.
- Check that will verify amount of living datanodes. We will have `GREEN` status only when all datanodes are active, `RED` in case when amount of living datanodes are less then `dfs.replicaton`, and `YELLOW` in all other cases.
- Check to verify ability of cluster to write some data to `HDFS`. We will have `RED` status if something is failed.
- Amount of free space in HDFS. We will compare this value with reserved memory in HDFS, and if amount of free space is less then provided value, check will be supposed to be failed. If check is not passed, we will have `YELLOW` state we will advice to scale cluster up with some extra datanodes (or clean some data). I think, we should not have any additional configuration options in here, just because this check will never report `RED` state.

HA checks:

- `YELLOW` state in case when at least one stand-by service is working; and `RED` otherwise. Affects YARN and HDFS both.

YARN checks:

- Resourcemanger is active. Obviously, `RED` state if something is wrong.
- Amount of active nodemanagers. `YELLOW` state if something is not available, and `RED` if amout of live nodemanagers are less then `50%`.

Kafka check:

- Check that kafka is operable: create example topic, put several messages in topic, consuming messages. `RED` state in case of something is wrong.

CDH plugin check:

This section is going to describe specific checks for CDH plugin. For this checks we will need to extend current sahara's implementation of `cm_api` tool. There is an API methods to get current health of the cluster. There are few examples of responses for yarn service.

There is the bad case example:

```
"yarn01": {
  "checks": [
    {
      "name": "YARN_JOBHISTORY_HEALTH",
      "summary": "GOOD"
    },
    {
      "name": "YARN_NODE_MANAGERS_HEALTHY",
      "summary": "CONCERNING"
    },
    {
      "name": "YARN_RESOURCEMANAGERS_HEALTH",
```

(continues on next page)

(continued from previous page)

```
      "summary": "BAD"
    }
  ],
  "summary": "BAD"
}
```

and good case example:

```
"yarn01": {
  "checks": [
    {
      "name": "YARN_JOBHISTORY_HEALTH",
      "summary": "GOOD"
    },
    {
      "name": "YARN_NODE_MANAGERS_HEALTHY",
      "summary": "GOOD"
    },
    {
      "name": "YARN_RESOURCEMANAGERS_HEALTH",
      "summary": "GOOD"
    }
  ],
  "summary": "GOOD"
}
```

Based on responses above we will calculate health of the cluster. Also possible states which Cloudera can return through API are `DISABLED` when service was stopped and `CONCERNING` if something is going to be bad soon. In this health check sahara's statuses will be calculated based on the following table:

```
+--------------+---------------------------------+
| Sahara state |   Cloudera state                |
+--------------+---------------------------------+
| GREEN        | All services GOOD               |
+--------------+---------------------------------+
| YELLOW       | At least 1 service CONCERNING   |
+--------------+---------------------------------+
| RED          | At least 1 service BAD/DISABLED|
+--------------+---------------------------------+
```

Some additional information about Cloudera health checks are in here: [0]

Ambari plugin:

Current `HDP 2.0.6` will support only basic verifications. The main focus in here is to implement additional checks for the Ambari plugin. There are several ideas of checks in Ambari plugin:

- Ambari alerts verification. Ambari plugin have several alerts if something is wrong with current state of the cluster. We can get alerts through Ambari API. If we have at least one alert in here it's proposed to use `YELLOW` status for the verification, and otherwise we will use `GREEN` status for that.
- Ambari service checks verification. Ambari plugin have a bunch of services checks in here, which can be re-triggered by user through the Ambari API. These checks are well described in [1]. If at

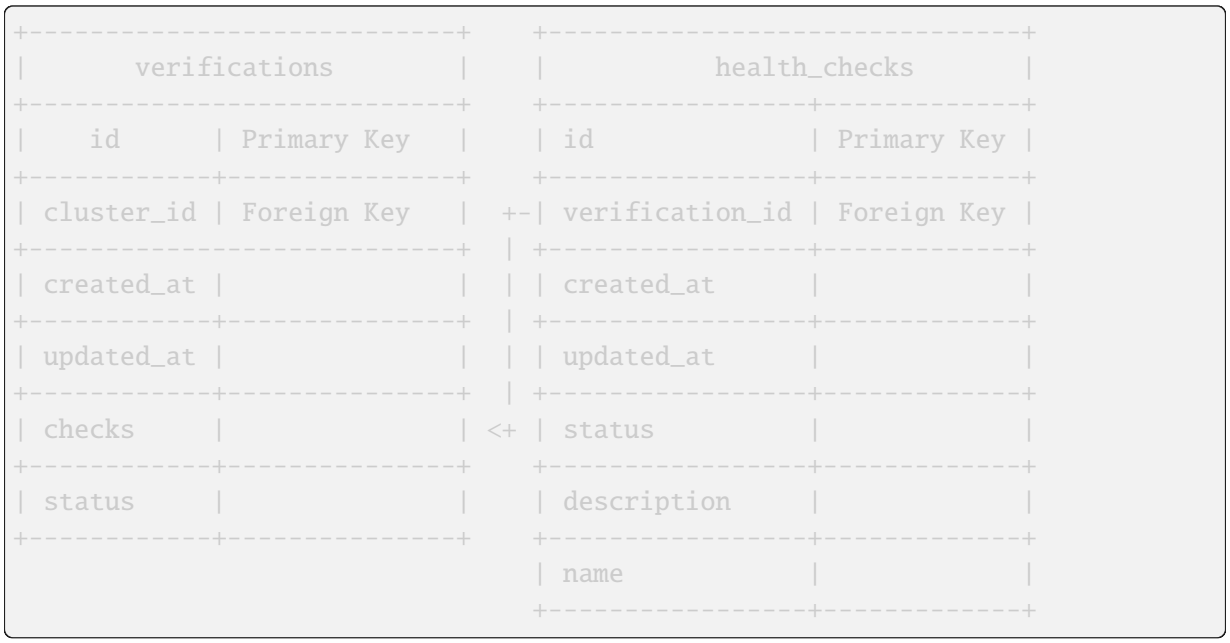
least one check failed, we will use `RED` status for that sutiation, otherwise it's nice to use `GREEN`.

Alternatives

All health checks can be disabled by the option.

Data model impact

Graphical description of data model impact:



We will have two additional tables where we will store verifications and health checks.

First table with of verifications will have following columns id, cluster_id (foreign key), created_at, updated_at.

Also will be added new table to store health check results. This table will have the following columns: id, verification_id, description, status, created_at and updated_at.

We will have cascade relationship (checks) between cluster verifications and cluster health checks to get correct access from health check to cluster verification and vice versa. Also same relationship will be between cluster and verification for same purpose.

Also to aggregation results of latest verification and disabling/enabling verifications for particular cluster will be added the new column to cluster model: `verifications_status`. We will not use `status` for that purpose just to keep these two variables separately (we already using status in many places in sahara).

For example of verifications:

1. One health check is still running:

```
"cluster_verification": {
  "id": "1",
  "cluster_id": "1111",
  "created_at": "2013-10-09 12:37:19.295701",
```

(continues on next page)

(continued from previous page)

```
  "updated_at": "2013-10-09 12:37:19.295701",
  "status": "CHECKING",
  "checks": [
    {
      "id": "123",
      "created_at": "2013-10-09 12:37:19.295701",
      "updated_at": "2013-10-09 12:37:19.295701",
      "status": "GREEN",
      "description": "some description",
      "name": "same_name"
    },
    {
      "id": "221",
      "created_at": "2013-10-09 12:37:19.295701",
      "updated_at": "2013-10-09 12:37:19.295701",
      "status": "CHECKING",
      "description": "some description",
      "name": "same_name"
    },
  ]
}
```

2. All health checks are completed but one was failed:

```
"cluster_verification": {
  "id": "2",
  "cluster_id": "1112",
  "created_at": "2013-10-09 12:37:19.295701",
  "updated_at": "2013-10-09 12:37:30.295701",
  "STATUS": "RED",
  "checks": [
    {
      ..
      "status": "RED",
      "description": "Resourcemanager is down",
      ..
    },
    {
      ..
      "status": "GREEN",
      "description": "HDFS is healthy",
    }
  ]
}
```

REST API impact

Mechanism of receiving results of cluster verifications will be quite simple. We will just use usual `GET` method for clusters.

So, the main API method will be the following: `GET <tenant_id>/clusters/<cluster_id>`. In such case, we will return detailed info of the cluster with verifications.

Example of response:

```
{
  "status": "Active",
  "id": "1111",
  "cluster_template_id": "5a9a09a3-9349-43bd-9058-16c401fad2d5",
  "name": "sample",
  "verifications_status": "RUNNING",
  ..
  "verification": {
    "id": "1",
    "cluster_id": "1111",
    "created_at": "2013-10-09 12:37:19.295701",
    "updated_at": "2013-10-09 12:37:19.295701",
    "checks": [
      {
        "id": "123",
        "created_at": "2013-10-09 12:37:19.295701",
        "updated_at": "2013-10-09 12:37:19.295701",
        "status": "GREEN",
        "description": "some description",
        "name": "same_name"
      },
      {
        "id": "221",
        "created_at": "2013-10-09 12:37:19.295701",
        "updated_at": "2013-10-09 12:37:19.295701",
        "status": "CHECKING",
        "description": "some description",
        "name": "same_name"
      },
    ]
  }
}
```

For re-triggering to cluster verification, some additional behaviour should be added to the following API method:

`PATCH <tenant_id>/clusters/<cluster_id>`

If the following data will be provided to this API method we will re-trigger verification:

```
{
  'verification': {
    'status': 'START'
```

(continues on next page)

(continued from previous page)

```
  }
}
```

Start will be reject when verifications disabled for the cluster or when verification is running on the cluster.

Also we can disable verification for particular cluster to avoid unneeded noisy verifications until health issues are fixed by the following request data:

```
{
  'verification': {
    'status': 'DISABLE'
  }
}
```

And enable in case we need to enable health checks again. If user is trying to disable verification only future verifications will be disabled, so health checks still will be running.

If something additional will be added to this data we will mark request as invalid. Also we will implement new validation methods to deny verifications on cluster which already have one verification running.

Other end user impact

Need to implement requests for run checks via python-saharaclient and get their results.

Deployer impact

None.

Developer impact

None.

Sahara-image-elements impact

None.

Sahara-dashboard / Horizon impact

Dashboard impact is need to add new tab in cluster details with results of verifications.

6.5.3 Implementation

Assignee(s)

Primary assignee:
vgridnev

Other contributors:
apavlov-n, esikachev

Work Items

1. Implement basic skeleton for verifications (with base checks)
2. Python-saharaclient support addition
3. CLI support should be implemented
4. Implement tab with verification results to Horizon
5. Need to add new WADL docs with new api-method
6. All others checks should be implemented
7. Should be added support to scenario framework to allow re-triggering.
8. Implement sending history to Ceilometer.

6.5.4 Dependencies

None

6.5.5 Testing

Feature will be covered by the unit tests, and manually. New test commit (not for merging) will be added to show that all verifications are passed (since we are at the middle of moving scenario framework).

6.5.6 Documentation Impact

Documentaion should updated with additional information of `How to` repair issues described in the health check results.

6.5.7 References

[0] http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/cm_ht.html
[1] https://cwiki.apache.org/confluence/display/AMBARI/Running+Service+Checks

6.6 Remove unsupported versions of MapR plugin

https://blueprints.launchpad.net/sahara/+spec/deprecate-old-mapr-versions

Remove MapR plugin 3.1.1 4.0.1 4.0.2 5.0.0.mrv1 and mapr-spark.

6.6.1 Problem description

Some of supported MapR versions are old and not used and others have their benefits contained in newer versions.

6.6.2 Proposed change

We will not support the following MapR versions and remove them in Mitaka release.

The following MapR version is removed due to end of support: * MapR 5.0.0 mrv1 * MapR 4.0.2 mrv1 mrv2 * MapR 4.0.1 mrv1 mrv2 * MapR 3.1.1

mapr-spark version will also be removed because Spark will go up with latest MapR as a service on YARN.

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

- Remove MapR v3.1.1 support
- Remove MapR v4.0.1 support
- Remove MapR v4.0.2 support
- Remove MapR v5.0.0 support
- Remove MapR mapr-spark

Sahara-dashboard / Horizon impact

None

6.6.3 Implementation

Assignee(s)

Primary assignee:
groghkov

Other contributors:
None

Work Items

- Remove MapR v3.1.1
- Remove all tests for v3.1.1
- Remove MapR v4.0.1
- Remove all tests for v4.0.1
- Remove MapR v4.0.2
- Remove all tests for v4.0.2
- Remove MapR v5.0.0.mrv1
- Remove all tests for v5.0.0.mrv1
- Remove MapR Spark
- Remove all tests for MapR Spark
- Remove default templates for MapR 3.1.1 4.0.1 4.0.2 5.0.0.mrv2 in sahara/plugins/default_templates/mapr
- Update documentation

6.6.4 Dependencies

None

6.6.5 Testing

None

6.6.6 Documentation Impact

Documention needs to be updated: * doc/source/userdoc/mapr_plugin.rst

6.6.7 References

None

6.7 Support of distributed periodic tasks

https://blueprints.launchpad.net/sahara/+spec/distributed-periodics

This specification proposes to add distributed periodic tasks.

6.7.1 Problem description

Currently periodic tasks are executed on each engine simultaneously, so that the amount of time between those tasks can be very small and load between engines is not balanced. Distribution of periodic tasks between engines can deal with both problems.

Also currently we have 2 periodic tasks that make clusters cleanup. Clusters terminations in these tasks are performed in cycle with direct termination calls (bypassing OPS). This approach leads to cessation of periodic tasks on particular engine during cluster termination (it harms even more if some problems during termination occurs).

Moreover distribution of periodic tasks is important for peridic health checks to prevent extra load on engines.

6.7.2 Proposed change

This change consists of two things:

- Will be added ability to terminate clusters in periodic tasks via OPS;
- Will be added support of distributed periodic tasks.

Distributed periodic tasks will be based on HashRing implementation and Tooz library that provides group membership support for a set of backends [1]. Backend will be configured with `periodic_coordination_backend` option.

There will be only one group called `sahara-periodic-tasks`. Once a group is created, any coordinator can join the group and become a member of it.

As an engine joins the group and builds HashRing, hash of its ID is being made, and that hash determines the data it's going to be responsible for. Everything between this number and one that's next in the ring and that belongs to a different engine, is now belong to this engine.

The only remaining problem is that some engines will have disproportionately large space before them, and this will result in greater load on them. This can be ameliorated by adding each server to the ring a number of times in different places. This is achieved by having a replica count (will be 40 by default).

HashRing will be rebuilt before execution of each periodic task to reflect actual state of coordination group.

`sahara.service.coordinator.py` module and two classes will be added:

- `Coordinator` class will contain basic coordination and grouping methods:

```
class Coordinator(object):
    def __init__(self, backend_url):
        ...

    def heartbeat(self):
        ...

    def join_group(self, group_id):
        ...

    def get_members(self, group_id):
        ...
```

- `HashRing` class will contain methods for ring building and subset extraction:

```
class HashRing(Coordinator):
    def __init__(self, backend_url, group_id, replicas=100):
        ...

    def _build_ring(self):
        ...

    def get_subset(self, objects):
        ...
```

Now we have 4 periodic tasks. For each of them will be listed what exactly will be distributed:

- `update_job_statuses` Each engine will get a list of all job executions but will update statuses for only a part of them according to a hash values of their IDs.
- `terminate_unneeded_transient_clusters` Each engine will get a list of all clusters, check part of them and request termination via OPS if needed.
- `terminate_incomplete_clusters` Same as for `terminate_unneeded_transient_clusters`.
- `check_for_zombie_proxy_users` Each engine will get a list of users but will check if it is zombie or not only for part of them.

Also we will have a periodic task for health checks soon. Health check task will be executed on clusters and this clusters will be split among engines in the same way as job executions for `update_job_statuses`.

If a coordination backend is not provided during configuration, periodic tasks will be launched in an old-fashioned way and HashRing will not be built.

If a coordination backend is provided, but configured wrong or not accessible, engine will not be started with corresponded error.

If a connection to the coordinator will be lost, periodic tasks will be stopped. But once connection is established again, periodic tasks will be executed in the distributed mode.

In order to keep the connection to the coordination server active, `heartbeat` method will be called regularly (every `heartbeat_timeout` seconds) in a separate thread.

Configurable number of threads (each thread will be a separate member of a group) performing periodic tasks will be launched.

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

Coordination backend should be configured.

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

6.7.3 Implementation

Assignee(s)

Primary assignee:
apavlov-n

Work Items

- Adding ability to terminate clusters in periodic tasks via OPS;
- Implementing HashRing for distribution of periodic tasks;
- Documenting changes in periodic tasks configuration.
- Adding support of distributed periodics to devstack with ZooKeeper as a backend

6.7.4 Dependencies

Tooz package [2]

6.7.5 Testing

Unit tests, enabling distributed periodics in intergration tests with one of the supported backends (for example, ZooKeeper) and manual testing for all available backends [1] supported by Tooz library.

6.7.6 Documentation Impact

Sahara REST API documentation in api-ref will be updated.

6.7.7 References

[1]: http://docs.openstack.org/developer/tooz/compatibility.html#driver-support

[2]: https://pypi.python.org/pypi/tooz

6.8 Improved secret storage utilizing castellan

https://blueprints.launchpad.net/sahara/+spec/improved-secret-storage

There are several secrets (for example passwords) that sahara uses with respect to deployed frameworks which are currently stored in its database. This blueprint proposes the usage of the castellan package key manager interface for offloading secret storage to the OpenStack Key management service.

6.8.1 Problem description

There are several situations under which sahara stores usernames and passwords to its database. The storage of these credentials represents a security risk for any installation that exposes routes to the controller's database. To reduce the risk of a breach in user credentials, sahara should move towards using an external key manager to control the storage of user passwords.

6.8.2 Proposed change

This specification proposes the integration of the castellan package into sahara. Castellan is a package that provides a single point of entry to the OpenStack Key management service. It also provides a pluggable key manager interface allowing for differing implementations of that service, including implementations that use hardware security modules (HSMs) and devices which support the key management interoperability protocol (KMIP).

Using the pluggable interface, a sahara specific key manager will be implemented that will continue to allow storage of secrets in the database. This plugin will be the default key manager to maintain backward compatibility, furthermore it will not require any database modification or migration.

For users wishing to take advantage of an external key manager, documentation will be provided on how to enable the barbican key manager plugin for castellan. Enabling the barbican plugin requires a few modifications to the sahara configuration file. In this manner, users will be able to customize the usage of the external key manager to their deployments.

Example default configuration:

```
[key_manager]
api_class = sahara.utils.key_manager.sahara_key_manager.SaharaKeyManager
```

Example barbican configuration:

```
[key_manager]
api_class = castellan.key_manager.barbican_key_manager.BarbicanKeyManager
```

To accomodate the specific needs of sahara, a new class will be created for interacting with castellan; `SaharaKeyManager`. This class will be based on the abstract base class `KeyManager` defined in the castellan package.

The `SaharaKeyManager` class will implement a thin layer around the storage of secrets without an external key manager. This class will allow sahara to continue operation as it exists for the Kilo release and thus maintain backward compatibility. This class will be the default plugin implementation to castellan.

Example usage:

```
from castellan import key_manager as km
from castellan.key_manager.objects import passphrase

keymanager = km.API()

# create secret
new_secret = passphrase.Passphrase('password_text_here')

# store secret
new_secret_id = keymanager.store_key(context, new_secret)

# retrieve secret
retrieved_secret = keymanager.get_key(context, new_secret_id)
secret_cleartext = retrieved_secret.get_encoded()

# revoke secret
keymanager.delete_key(context, new_secret_id)
```

This solution will provide the capability, through the barbican plugin, to offload the secrets in such a manner that an attacker would need to penetrate the database and learn the sahara admin credentials to gain access to the stored passwords. In essence we are adding one more block in the path of a would-be attacker.

This specification focuses on passwords that are currently stored in the sahara database. The following is a list of the passwords that will be moved to the key manager for this specification:

- Swift passwords entered from UI for data sources
- Swift passwords entered from UI for job binaries
- Proxy user passwords for data sources
- Proxy user passwords for job binaries
- Hive MySQL passwords for vanilla 1.2.1 plugin
- Hive database passwords for CDH 5 plugin
- Hive database passwords for CDH 5.3.0 plugin
- Hive database passwords for CDH 5.4.0 plugin
- Sentry database passwords for CDH 5.3.0 plugin
- Sentry database passwords for CDH 5.4.0 plugin
- Cloudera Manager passwords for CDH 5 plugin
- Cloudera Manager passwords for CDH 5.3.0 plugin
- Cloudera Manager passwords for CDH 5.4.0 plugin

Alternatives

One possible alternative to using an external key manager would be for sahara to encrypt passwords and store them in swift. This would satisfy the goal of removing passwords from the sahara database while providing a level of security from credential theft.

The downside to this methodology is that it places sahara in the position of arbitrating security transactions. Namely, the use of cryptography in the creation and retrieval of the stored password data.

Data model impact

None.

REST API impact

None.

Other end user impact

None.

Deployer impact

A new configuration option will be provided by the castellan package to set the key manager implementation. This will be the SaharaKeyManager by default. Deployers wishing to use barbican might need to set a few more options depending on their installation. These options will be discussed in the documentation.

Use of an external key manager will depend on having barbican installed in the stack where it will be used.

Developer impact

Developers adding new stored passwords to sahara should always be using the key manager interface.

Sahara-image-elements impact

None.

Sahara-dashboard / Horizon impact

None.

6.8.3 Implementation

Assignee(s)

Primary assignee:
mimccune (Michael McCune)

Other contributors:
None

Work Items

- create SaharaKeyManager class
- add tests for new class
- add tests for secret storage
- create documentation for external key manager usage
- migrate passwords to key manager

6.8.4 Dependencies

Castellan package, available through pypi. Currently this version (0.1.0) does not have a barbican implementation, but it is under review[1].

6.8.5 Testing

Unit tests will be created to exercise the SaharaKeyManager class. There will also be unit tests for the integrated implementation.

Ideally, functional integration tests will be created to ensure the proper storage and retrieval of secrets. The addition of these tests represents a larger change to the testing infrastructure as barbican will need to be added. Depending on the impact of changing the testing deployment these might best be addressed in a separate change.

6.8.6 Documentation Impact

A new section in the advanced configuration guide will be created to describe the usage of this new feature.

Additionally this feature should be described in the OpenStack Security Guide. This will require a separate change request to the documentation project.

6.8.7 References

[1]: https://review.openstack.org/#/c/171918

castellan repository https://github.com/openstack/castellan

note, the castellan documentation is still a work in progress

barbican documentation https://docs.openstack.org/barbican/latest/

barbican wiki https://github.com/cloudkeep/barbican/wiki

6.9 Move scenario tests to a separate repository

https://blueprints.launchpad.net/sahara/+spec/move-scenario-tests-to-separate-repo

Scenario tests represent an independent framework for testing Sahara. Moving scenario tests to a separate repository will make testing Sahara stable releases more comfortable. The migration should be done in a way that there will be no need to pre-configure the CI or any other external tool, just install the scenario framework and start the tests.

6.9.1 Problem description

There are 2 issues to be solved: 1. It is difficult to package tests independently from Sahara. 2. It is impossible to release those tests independently.

6.9.2 Proposed change

First it's proposed to implement a method for the correct tests installation. Next a new repository should be created and all Sahara scenario tests should be moved to that repository. After that all sahara-ci jobs should be migrated to the new mechanism of running scenario tests.

The scenario framework itself should be improved to be able to correctly skip tests for the features that are not available in earlier Sahara releases. Testing stable/kilo and stable/liberty branches may require some updates to appropriate scenarios. The CI should initially start running scenario tests for stable/kilo and stable/liberty and keep testing 2 latest stable releases.

Having a separate repository allows using these tests as a plugin for Sahara (like client tests for tempest, without releases). Perhaps, scenario tests will be installed in a virtualenv.

The scenario framework and tests themselves may be packaged and released. All requirements should then be installed to the current operating system. This however implies that scenario framework dependencies should be packaged and installed as well. Many of these requirements are not packaged neither in MOS nor in RDO.

There are 2 ways to solve this: 1. Manual update of requirements to keep them compatible with 2 previous stable releases 2. Move all changing dependencies from requirements to test-requirements

Yaml-files defining the scenarios are a part of the test framework and they should be moved to the new repository with scenario tests.

Next step is to add support of default templates for every plugin in each stable release with correct flavors, node processes. For example, list of plugins for the 2 previous stable releases:

- kilo: vanilla 2.6.0, cdh 5.3.0, mapr 4.0.2, spark 1.0.0, ambari 2.2/2.3

- liberty: vanilla 2.7.1, cdh 5.4.0, mapr 5.0.0, spark 1.3.1, ambari 2.2/2.3

Each default template for should be implemented as yaml file which will be run on CI.

Alternatives

Alternative would be to package the tests for easier installation and usage. For example, command for running tests for vanilla 2.7.1 plugin will be:

```
$ sahara_tests -p vanilla -v 2.7.1
```

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

Deployers should switch CI to use the new repository with scenario tests instead of current Sahara repository.

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

6.9.3 Implementation

Assignee(s)

Primary assignee:
esikachev

Work Items

This will require following changes:

- Implement installation mechanism for tests.
- Move tests to separate repository (with scenario_unit tests).
- Update sahara-ci jobs for new scenario tests repository.
- Add default templates for each plugin and release.
- Update scenario framework for correct testing of previous releases.
- Add new jobs on sahara-ci for testing Sahara on previous releases.
- Add ability for tests packaging.

6.9.4 Dependencies

None

6.9.5 Testing

None

6.9.6 Documentation Impact

Need to add documentation with description "How to run Sahara scenario tests".

6.9.7 References

None

6.10 Add recurrence EDP jobs for sahara

https://blueprints.launchpad.net/sahara/+spec/recurrence-edp-job

This spec is to allow running recurrence edp jobs in sahara.

6.10.1 Problem description

Currently sahara only supports one time click edp job. But sometimes, we need recurrence edp jobs. For example, from 8:00AM to 8:00PM, run a job every 5mins. By adding recurrence edp jobs to sahara edp engine, we can have different engine implementation. (oozie,spark,storm etc)

6.10.2 Proposed change

Define run_recurrence_job() interface in sahara base edp engine, then implement this interface to the oozie engine. (Spark engine will be drafted in later spec)

Add one job execution type named "recurrence". Sahara base job engine will run different type of jobs according to the job execution type.

Add one sahara perodic task named update_recurrence_job_statuses() to update the recurrence jobs's sub-jobs running status.

Add validation for invalid data input and check if the output url is already exist.

Example of recurrence edp job request

POST /v1.1/{tenant_id}/jobs/<job_id>/execute

For supporting this feature,add a new table named child_job_execution which will store sub-jobs of the father recurrence job. When create a recurrence edp job, in the same time, we create M(configured) sub-jobs,so we can show up the status of the M sub-jobs. perodic task will check the status of the M child jobs's status and update them, and also if there is one child job is finished, we create a new feature-run child job to fill the vacancy, in this case, we maintain a M-row-window in the DB, so this will avoid the endless child job creation.if we delete this recurrence job, we delete it's child jobs in the child_job_execution table,including finished jobs and future-run jobs.

For creating the child job execution table,we just add one more column named "father_job_id" based on job execution table which points to it's father recurrence job.

For Horizon changes, considering we may have many child jobs, so we only show the latest M sub jobs when user click the recurrence edp job, and add two query datetime picker with a button for user to search the history finished child jobs.

For oozie engine implementation of recurrence edp jobs. we have changes as below:

Implement the run_recurrence_job in the base edp engine. call oozie client to submit the recurrence edp job. add perodic task update recurrence job statuses to update child jobs's status in the child_job_execution table.

Example of coordinator.xml

For spark implementation, no implementation here, will add them later.

Alternatives

Run edp job manually by login into the VM and running oozie command.

Data model impact

add a new table named child job execution.Just add one more column named father_job_id than the current job execution table. Other columns are totally the same as job execution table.

REST API impact

There is no change here, we can use current API, POST /v1.1/{tenant_id}/jobs/<job_id>/execute We can pass job_execution_type, start time, end time, period_minutes into job_configs.

```
CREATE TABLE child_job_execution (
    id VARCHAR(36) NOT NULL,
    job_id VARCHAR(36),
    father_job_id VARCHAR(36),
    tenant_id VARCHAR(80),
    input_id VARCHAR(36),
    output_id VARCHAR(36),
    start_time DATETIME,
    end_time DATETIME,
    info TEXT,
    cluster_id VARCHAR(36),
    oozie_job_id VARCHAR(100),
    return_code VARCHAR(80),
    job_configs TEXT,
    extra TEXT,
    data_source_urls TEXT,
    PRIMARY KEY (id),
    FOREIGN KEY (job_id)
        REFERENCES jobs(id)
        ON DELETE CASCADE
);
```

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

In Job launch page, add select choose box for user to choose the job execution type, it has three values(basic, scheduled, recurrence),default value is basic, which is the one-click running job.if user choose recurrence, there will be two datetime picker named as start and end time and a textbox for user to input the period_minutes.

6.10.3 Implementation

Assignee(s)

Primary assignee:
luhuichun(lu huichun)

Other contributors:
None

Work Items

- define recurrence job type
- add one perodic task named update_recurrence_job_statuses
- create coordinator.xml before run job in edp engine
- upload the coordinator.xml to job's HDFS folder
- add run_recurrence_job in sahara base engine
- modify sahara api reference docs
- add task to update the WADL at api-site

6.10.4 Dependencies

None.

6.10.5 Testing

unit test in edp engine add scenario integration test

6.10.6 Documentation Impact

Need to be documented.

6.10.7 References

oozie scheduled and recursive job implementation https://oozie.apache.org/docs/4.0.0/CoordinatorFunctionalSpec.html

6.11 Reduce Number Of Dashboard panels

https://blueprints.launchpad.net/sahara/+spec/reduce-number-of-panels

The sahara UI currently consists of 10 panels. That is more than any other Openstack service. Additionally, the way that the panels are meant to interact with each other is not intuitively obvious and can lead to confusion for users. The purpose of this spec is to propose reorganizing the sahara UI into a more logical layout that will give both a cleaner look and a more intuitive set of tools to the users.

6.11.1 Problem description

The sahara dashboard has too many panels without enough context provided to convey how each panel relates to the others. They are currently grouped vertically so that each appears close to the related panels in the list, but there is no other visual cue given. Given that, learning how to use the dashboard can often be a frustrating exercise of trial and error.

6.11.2 Proposed change

In the 10 panels that are currently in the dashboard, 4 of them (Clusters, Cluster Templates, Node Group Templates, Image Registry) are focused around cluster creation. Given that, it makes sense to fold them into a single panel which will be called "Clusters". This will make it very obvious where to go for all things cluster-related. Each of the current cluster-centric panels will have their own tab within the newly created panel. There are 5 panels (Jobs, Job Templates, Job Binaries, Data Sources and Plugins) that are focused around job creation and execution.

In addition to the conversion from panels to tabs, I'm proposing that the code also gets reorganized to reflect the new layout (ie: moving code to be a subdir of the panel that it will reside in -- there will be a top level "clusters" panel that will contain subdirectories for each of the tabs in that panel (clusters, cluster templates, node group templates, image registry). That reorganization will result in the changing of the URL definitions and will also change how we reference template names (Instead of "project/data_processing.nodegroup-templates/whatever.html", the "/project" part can be dropped,

resulting in "data_processing.nodegroup-templates/whatever.html", which is a slight improvement in brevity. Additionally, we can also elect to drop the "data_processing." portion by renaming the template folders. The oddity that forced us to use that as a workaround has since been solved in the horizon "enabled" mechanism.

Here is a quick ascii diagram showing the proposed UI layout: +------------+--+ | | | | Clusters | Clusters|Cluster Templates|Node Group Templates | | Jobs | +---+ | | | | | | | | | Usual table for chosen object | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | +---+ | | | | +------------+--+

Addionally, the current "Guide" panel will be split to fit more readily into the job and cluster spaces. It currently occupies a separate tab and is linked to from buttons in the other panels. The buttons will remain the same, but the code will be moved into subdirectories under clusters for the clusters guide and jobs for the job guide.

Alternatives

We could choose to leave the current panels alone and rely on stronger, more verbose documentation and possibly provide links to that documentation from within the dashboard itself. After multiple discussions with the sahara team, it is clear that this approach is not as desirable as reorganizing the layout.

We could possibly avoid some of the code reorganization, but in my opinion, it is important to have the code accurately reflect the layout. It makes it easier to design future changes and easier to debug.

Data model impact

No data model impact

REST API impact

No REST API impact

Other end user impact

End users will see a different layout in the sahara dashboard. While the difference in moving to tabs is significant, once an experienced user gets into the tabs, the functionality there will be unchanged from how it currently works. Some of the URLs used will also change a bit to remove any ambiguity (for instance the "details" URLs for each object type will be renamed to become something like "ds-details" for the data source details URLs).

Deployer impact

No deployer impact

Developer impact

No developer impact

Sahara-image-elements impact

No sahara-image-elements impact

Sahara-dashboard / Horizon impact

There will be changes to the "enabled" files in the sahara-dashboard project, but the process for installation and running will remain the same. If you are upgrading an existing installation, there will be some "enabled" files that are no longer required (in addition to updated files). We will need to be sure to document (and script wherever possible) which files should be removed.

6.11.3 Implementation

Assignee(s)

Primary assignee:
croberts

Other contributors:
None

Work Items

Reorganize sahara UI code to convert multiple panels into 2 panels that contain multiple tabs. This will likely be spread into multiple patches in order to try to minimize the review effort of each chunk. Each chunk is still going to be somewhat large though since it will involve moving entire directories of code around.

6.11.4 Dependencies

None

6.11.5 Testing

Each of the existing unit tests will still apply. They will all need to be passing in order to accept this change. It is likely that the (few) existing integration tests will need to be rewritten or adapted to expect the proposed set of changes.

6.11.6 Documentation Impact

The sahara-dashboard documentation is written and maintained by the sahara team. There will be several documentation updates required as a result of the changes described in this spec.

6.11.7 References

None

6.12 Removal of Direct Engine

https://blueprints.launchpad.net/sahara/+spec/remove-direct-engine

Direct Infrastructure Engine was deprecated in Liberty and it's time to remove that in Mitaka cycle.

6.12.1 Problem description

We are ready to remove direct engine. Sahara have ability to delete clusters created using direct engine, heat engine works nicely.

6.12.2 Proposed change

First it's proposed to remove direct engine first. Then after the removal we can migrate all direct-engine-jobs to heat engine. After that direct engine will be completely removed.

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

Deployers should switch to use Heat Engine instead of Direct Engine finally.

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

6.12.3 Implementation

Assignee(s)

Primary assignee:
vgridnev

Work Items

This will require following changes:

- Remove direct engine from the codebase (with unit tests).
- Remove two gate jobs for direct engine.
- Document that Direct Engine was removed finally.

6.12.4 Dependencies

None

6.12.5 Testing

None

6.12.6 Documentation Impact

Need to document that Direct Engine was removed.

6.12.7 References

None

6.13 Remove plugin Vanilla V2.6.0

https://blueprints.launchpad.net/sahara/+spec/deprecate-plugin-vanilla2.6.0

6.13.1 Problem description

As far as Vanilla v2.6.0 plugin is deprecated, we should remove all the code completely.

6.13.2 Proposed change

(1) Remove all plugin v2_6_0 code under plugins

(2) Remove all tests including unit and integration tests.

(3) Update Vanilla plugin documentation.

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

6.13.3 Implementation

Assignee(s)

Primary assignee:
luhuichun

Other contributors:
None

Work Items

- Remove Vanilla v2.6.0
- Remove all tests for Vanilla v2.6.0
- Update documentation

6.13.4 Dependencies

Dependend on change Iff2faf759ac21d5bd15372bae97a858a3d036ccb

6.13.5 Testing

None

6.13.6 Documentation Impact

Documention needs to be updated: * doc/source/userdoc/vanilla_plugin.rst

6.13.7 References

None

6.14 Modify 'is_default' behavior relative to 'is_protected' for templates

https://blueprints.launchpad.net/sahara/+spec/replace-is-default

With the addition of the general *is_protected* field on all objects, the *is_default* field on cluster and node group templates is semi-redundant. Its semantics should be changed; specifically, gating of update and delete operations should be handled exclusively by *is_protected*. The *is_default* field should still be used to identify default templates for bulk update and delete operations by the sahara-templates tool.

6.14.1 Problem description

The *is_protected* and *is_default* fields both control gating of update and delete operations. This overlap is not clean and should be resolved to avoid confusion for end users and developers alike and to decrease complexity in code maintenance.

Gating of update or delete operations should no longer be associated with the *is_default* field. Instead, it should only mark the default templates created with the *sahara-templates* tool so that bulk update and delete operations continue to function as is.

6.14.2 Proposed change

Remove the checks in the sahara db layer that raise exceptions based on the *is_default* field for template update and delete operations. This will leave the gating under the control of *is_protected*. Note that there will no longer be an error message that informs the user of failure to update or delete a default template -- it will be phrased in terms of the *is_protected* field instead.

Modify all of the default template sets bundled with sahara to include the *is_protected* field set to *True*. This will ensure that templates used as-is from the sahara distribution will be created with update and delete operations hindered.

Provide a database migration that sets *is_protected* to **True** for all templates with *is_default* set to **True**. This will guarantee the hindrance on update and delete operations continues for existing default templates.

Currently, sahara will allow update and delete of default templates if the *ignore_defaults* flag is set on the conductor methods. Change the name and handling of this flag slightly so that if it is set, sahara will ignore the *is_protected* flag instead. This will allow the *sahara-templates* tool to easily update and delete default templates without having to first clear the *is_protected* flag.

Write a documentation section on default templates best practices. Given the addition of *is_protected* and *is_public* and the changed semantics for *is_default*, provide some suggestions for best practices as follows: operators will be encouraged to load the default template sets in the *admin* or *service* tenant and then make them available to users with *is_public*. If modifications will be made to the default templates, they can be

loaded, copied and modified and the resultant templates made available to users with *is_public*. This will ensure that the original default templates as bundled with sahara will always be available as a reference with *is_protected* set to *True*. Alternatively, an admin could copy the default templates directory from the sahara distribution to another location and modify them on disk before loading them.

Implications of the change: it will now be possible for tenants with default templates to set *is_protected* to *True* and then edit those templates. If a subsequent update of those templates is done from the *sahara-templates* tool, those edits will be overwritten. This should be made clear in the best practices documentation described above. Additionally, *is_default* will remain a field that is only settable from the *sahara-templates* tool so that the default sets can be managed as a group.

Alternatives

None. Goals of the original default templates implementation were:

1 allow offline update and delete of default templates 2 treat a group of node group and cluster templates as a coherent set 3 allow operation filtered by plugin and version

The first requirement precludes use of the sahara client to manage default templates. If we did not have the first requirement, we could conceivably use the sahara client to manage them but requirements two and three mean that we would need logic around the operations anyway. Lastly, removing the *is_default* field completely leaves us with a need for some other mechanism to identify default templates. It seems best to remove the overlapping functionality of *is_default* and *is_protected* but leave the rest of the current implementation unchanged.

Also, this could be deferred as part of the API V2 initiative, but since it has so little impact on the API (see below) it seems unnecessary to wait

Data model impact

None, but provide an appropriate migration to set *is_protected* to **True** for all existing templates with *is_default* set to **True**.

REST API impact

None, *is_default* is only accessible directly through the conductor layer.

Other end user impact

None

Deployer impact

Deployers should be familiar with the best practices documentation we will add concerning default templates.

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

6.14.3 Implementation

Assignee(s)

Primary assignee:
tmckay

Other contributors:
None

Work Items

Remove code in the sahara db layer that rejects update or delete for defaults Update default templates bundled with sahara to include *is_protected* Add alembic migration for setting *is_protected* on existing defaults Review documentation and add a best practices section

6.14.4 Dependencies

None

6.14.5 Testing

Unit tests will be sufficient. Existing tests will be modified to prove that default templates may be updated and edited after this change, assuming *is_protected* is False. The existing tests on *is_protected* cover normal update/delete control.

6.14.6 Documentation Impact

Documentation on the sahara-templates tool may change slightly where it describes how to update or delete a default template, and we will add a best practices section as noted

6.14.7 References

None

6.15 Add ability of scheduling EDP jobs for sahara

https://blueprints.launchpad.net/sahara/+spec/enable-scheduled-edp-jobs

This spec is to allow running scheduled edp jobs in sahara.

6.15.1 Problem description

Currently sahara only supports one time click edp job. But in many use cases, we need scheduled edp jobs. So by adding scheduled ability to sahara edp engine, we can have different implementation for different engine.(oozie,spark,storm etc)

6.15.2 Proposed change

Define run_scheduled_job() interface in sahara base edp engine, then implement this interface to the oozie engine. (Spark and storm engine will be drafted in later spec)

Define two job execution types which indicate the job execution. But we need not to change the API, instead we can add parameters into job configs. Sahara will run different type of jobs according to the job execution type. In the api request, user should pass the job_execution_type into job_configs.

Two job execution types: (1)basic. runs simple one-time edp jobs, current sahara implementation (2)scheduled. runs scheduled edp jobs

Example of a scheduled edp job request

POST /v1.1/{tenant_id}/jobs/<job_id>/execute

For oozie engine implementation of scheduled edp jobs, we have changes as blow:

Before running the job, sahara will create a coordinator.xml to describe the job, then upload it to the HDFS EDP job lib folder. With this file, sahara call oozie client to submit this job, the job will be run at the scheduled time, the job status will be shown as "PREP" in the Horizon page. Certainly, user can delete this job in preparing status as welll as in running status.

Example of coordinator.xml

For spark and storm implementation, there is no implementation now, and we will add them later.

Alternatives

(1)Run edp job manually by login into the VM and running oozie command. (2)users can create cron jobs

Data model impact

None

REST API impact

There is no change here, and we can use current API, POST /v1.1/{tenant_id}/jobs/<job_id>/execute We can pass job_execution_type, start time, into job_configs to sahara.

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

In Job launch page, add textbox for user to input start job time, default value is now, to compatible with current implementation

6.15.3 Implementation

Assignee(s)

Primary assignee:
luhuichun(lu huichun)

Other contributors:
None

Work Items

- define scheduled job type
- create coordinator.xml before run job in edp engine
- upload the coordinator.xml to job's HDFS folder
- add run_schedule_job in oozie engine
- modify sahara api reference docs
- Add task to update the WADL at api-site

6.15.4 Dependencies

None.

6.15.5 Testing

unit test in edp engine add scenario integration test

6.15.6 Documentation Impact

Need to be documented.

6.15.7 References

oozie scheduled and recursive job implementation https://oozie.apache.org/docs/4.0.0/CoordinatorFunctionalSpec.html

6.16 SPI Method to Validate Image

https://blueprints.launchpad.net/sahara/+spec/validate-image-spi

This specification details the addition of a method to the Sahara plugin SPI to validate that a chosen image is up to the specification that the plugin requires. While it is not expected that plugin writers will be able to test the image deeply enough to ensure that a given arbitrary image will succeed in cluster generation and be functional in all contexts, it is hoped that by implementing this method well, plugin authors can provide a well-defined, machine-actionable contract which will be versioned with the plugin itself.

6.16.1 Problem description

At present, Sahara's image generation and cluster provisioning features are almost entirely decoupled: sahara-image-elements generates an image, and this image is taken by the server and assumed to be valid. This introduces the possibility of version incompatibility between sahara-image-elements and sahara itself, and failure (complete or partial, immediate or silent) in the case of the addition or modification of features on either side.

Larger context

This issue is only one part of a larger problem, which will not be wholly addressed in this spec, but for which this spec is an incremental step toward a solution.

At present, the processes involved in image generation and use are:

1) Image packing (pre-cluster spawn)
2) Clean image provisioning (building a cluster from an OS-only base image)
3) Image validation (ensuring that a previously packed image really is ok to use for the plugin, at least for the rules we can easily check)

The first, image-packing, is currently only possible via a command line script. The ideal user experience would allow generation of images either outside of OpenStack, via a command-line script, or with OpenStack, via a sahara API method. At present, this is not possible.

The second, in our present architecture, requires essentially rewriting the logic required to generate an image via the command line process in the plugin code, leading to duplicate logic and multiple maintenance points wherever cluster provisioning from clean images is allowed. However, it should be noted that in the clean image generation case, this logic is in its right place from an encapsulation perspective (it is maintained and versioned with the plugin code, allowing for easy separation, rather than maintained in a monolithic cross-cutting library which serves all plugins.)

The third is not formally undertaken as a separate step at all; it will be implemented by the feature this specification describes.

Within the context of this larger problem, this feature can be seen as the first incremental step toward a unified solution for image validation, unification of clean and packed image generation logic, and facilitation of image packing via an API. Once this SPI method is stable, functional, and expresses a complete set of tests for all maintained plugins, the validation specification can then be reused as a series of idempotent state descriptions for image packing, which can then be exposed via an API for any plugins which support it.

6.16.2 Proposed change

SPI method contract

A new method will be added to the plugin SPI in Sahara:

```
validate_images(self, cluster, reconcile=True)
```

This method will be called after cluster provisioning (as this will be necessary for machine access) and before cluster configuration. This method will receive the cluster definition as an argument, as well as a boolean flag describing whether or not the plugin should attempt to fix problems if it finds them.

If this method is not implemented by a plugin, provisioning will proceed as normal; as this is purely a safety feature, full backward compatibility with previous plugin versions is acceptable.

The contract of this method is that on being called, the plugin will take any steps it sees fit to validate that any utilized images are fit for their purposes. It is expected that all tests that are run will be necessary for the cluster to succeed, but not that the whole set of tests will be absolutely sufficient for the cluster to succeed (as this would essentially be disproving a universal negative, and would require such in-depth testing as to become ludicrous.)

If the reconcile flag is set to False, this instructs the plugin that it should only test the image, but change nothing, and report error if its tests fail. If reconcile is True (this will be set by default,) then the plugin will also take any steps it is prepared to take to bring the instances of the cluster into line with its expectations. Plugins are not required to provide this functionality, just as they are not required to implement validate_image; if they wish to fail immediately in the case of an imperfect image, that is their choice. However, if a plugin does not support reconciliation, and reconcile is set to True, it must raise an error; likewise, if a plugin receives reconcile=False but it is not able to avoid reconciliation (if, for instance, its implementation uses Puppet and will by definition make changes if needed,) it must raise as well.

sahara.plugins.images

The sahara base service will provide a set of utilities to help plugin authors to validate their images. These will be found in sahara.plugins.images. Usage of these utilities is wholly optional; plugin authors may implement validation using whatever framework they see fit. It is noted that this module could be immediately written to allow a great deal of deep functionality in terms of matching image validations to services, allowing custom images to be used for specific nodegroups and service sets. However, as no plugins are currently implementing such a feature set, a more basic first iteration is reasonable, and the methods described below will allow a plugin author to perform such specific validations if it is desired.

The images module will provide several public members: the definitions of the most notable (if not all) are given below:

```
def validate_instance(instance, validators, reconcile=True, **kwargs):
    """Runs all validators against the specified instance."""

class ImageValidator(object):
"""Validates the image spawned to an instance via a set of rules."""

    __metaclass__ = abc.ABCMeta

    @abc.abstractmethod
    def validate(self, remote, reconcile=True, **kwargs):
        pass

class SaharaImageValidatorBase(ImageValidator):
    """Still-abstract base class for Sahara's native image validation,
    which provides instantiation of subclasses from a yaml file."""

    @classmethod
    def from_yaml(cls, yaml_path, validator_map, resource_roots):
        """Constructs and returns a validator from the provided yaml file.
        :param yaml_path: The path to a yaml file.
        :param validator_map: A map of validator name to class. Each class
```

(continues on next page)

(continued from previous page)

```
            is expected to descend from SaharaImageValidator. This method
            will use the static map of validator name to class provided in
            the sahara.plugins.images module, updated with this map, to
            parse he appropriate classes to be used.
        :param resource_root: The roots from which relative paths to
            resources (scripts and such) will be referenced. Any resource
            will be pulled from the first path in the list at which a file
            exists."""

class SaharaImageValidator(SaharaImageValidatorBase):
    """The root of any tree of SaharaImageValidators."""

    def validate(self, remote, reconcile=True, env_map=None, **kwargs):
        """Validates the image spawned to an instance."""
        :param env_map: A map of environment variables to be passed to
            scripts in this validation."""
```

Additionally, two classes of error will be added to sahara.plugins.exceptions:

- ImageValidationError: Exception indicating that an image has failed validation.
- ImageValidationSpecificationError: Exception indicating that an image validation spec (yaml) is in error.

SaharaImageValidator

It is entirely possible for a plugin author, in this framework, to use idempotent state enforcement toolsets, such as Ansible, Puppet, Chef, and the like, to validate and reconcile images. However, in order that Sahara need not absolutely depend on these tools, we will provide the SaharaImageValidator class.

This validator will provide a classmethod which allows it to build its validations from a .yaml file. The first iteration of this validator will be very limited, and as such will provide only a few abstract validation types. This yaml will be interpreted using whatever ordering is available; as dicts are unordered in yaml, this scheme makes extensive use of lists of single- item dicts.

An example .yaml file showing the revision-one validator set follows. Note that these are not intended to be realistic, sahara-ready definitions, merely examples taken from our experience:

```
validators:
  - os_case:
      - redhat:
          - package: nfs-utils
      - debian:
          - package: nfs-common
  - any:
    - package: java-1.8.0-openjdk-devel
    - package: java-1.7.0-openjdk-devel
  - script: java/setup-java-home
  - package:
    - hadoop
    - hadoop-libhdfs
```

(continues on next page)

(continued from previous page)

```
    - hadoop-native
    - hadoop-pipes
    - hadoop-sbin
    - hadoop-lzo
    - lzo
    - lzo-devel
    - hadoop-lzo-native
```

These resource declarations will be used to instantiate the following basic validator types:

Validator types

SaharaPackageValidator (key: package)

Verifies that the package or packages are installed. In the reconcile=True case, ensures that local package managers are queried before resorting to networked tools, along the lines of:

```
`dpkg -s $package || apt-get -y install $package`  # debian
`rpm -q $package || yum install -y $package`  # redhat
```

The input to this validator may be a single package definition or a list of package definitions. If the packages are grouped in a list, any attempt to install the packages will be made simultaneously. A package definition may be a single string or a nested structure, which may support a version attribute as follows:

```
- package: hadoop
- package:
  - hadoop-libhdfs
  - lzo:
      version: xxx.xxx
```

Because reliable version comparison will often require reference to epochs, and because the tool must succeed in an offline context, the initial, Sahara core-provided package validator will allow only exact version pinning. As this version is yaml-editable, this is not adequate to our purposes, and can be extended by plugin developers if needed and appropriate.

SaharaScriptValidator (key: script)

Runs an arbitrary script from source, as specified by a relative path from the resource root.

The input to this validator must be a single script definition. A script definition may be a single string or a nested structure, which may support attributes as follows (the example is purely explanatory of the structure):

```
- script: simple_script.sh
- script:
  java/find_jre_home:
    output: JRE_HOME  # Places the stdout of this script into the env map
                      # for future scripts
- script:
```

(continues on next page)

(continued from previous page)

```
java/setup_java_home:
  env_vars:         # Sets only the named env vars from the env map
    - JDK_HOME
    - JRE_HOME
```

Scripts are always provided the env var $SIV_DISTRO, which specifies the linux distribution per our current SIE distro conventions, and the env var $SIV_RECONCILE, which is set to 0 if only validation should occur and 1 if corrective action should be taken.

Additional variables are referenced from the env_map argument passed originally to SaharaImageValidator.from_yaml (and are presumably parsed from cluster configuration information). The output attribute of the script resource can be used to modify this map in flight, placing the output of a script into the (single) named variable. More complex interactions require extension.

This validator is intentionally lightweight. These image validations and manipulations should not be overwhelmingly complex; if deep configuration is needed, then the more freeform configuration engine should run those steps, or the plugin author should utilize a more fully-featured state enforcement engine, with all the dependencies that entails (or write a custom validator).

NOTE THAT ALL SCRIPTS REVIEWED BY THE SAHARA TEAM MUST BE WRITTEN TO BE IDEMPOTENT. If they are to take non-reproducible action, they must test to see if that action has already been taken. This is critical to the success of this feature in the long term.

SaharaAnyValidator (key: any)

Verifies that at least one of the validators it contains succeeds. If reconcile is true, runs all validators in reconcile=False mode before attempting to enforce any. If all fail in reconcile=False mode, it then attempts to enforce each in turn until one succeeds.

Note that significant damage can be done to an image in failed branches if any is used with reconcile=true. However, guarding against this sort of failure would impose a great deal of limitation on the use of this validator. As such, warnings will be documented, but responsible use is left to the author of the validation spec.

SaharaAllValidator (key: all)

Verifies that all of the validators it contains succeed. This class will be instantiated by the yaml factory method noted above, and will contain all sub-validations.

SaharaOSCaseValidator (key: os_case)

Switches on the distro of the instance being validated. Recognizes the OS family names redhat and debian, as per DIB. Runs all validators under the first case that matches.

Notes on validators

Plugin authors may write their own validator types by extending the SaharaImageValidator type, implementing the interface, and passing the key and class into the validator_map argument of SaharaImageValidator.from_yaml.

It should be noted that current "clean" image generation scripts should be moved into this layer as part of the initial effort to implement this method for any given plugin, even if they are represented as a monolithic script resource. Otherwise clean images will very likely fail validation.

Note also that the list above are certain to be needed, but as the implementer works, it may become useful to create additional validators (file, directory, and user spring to mind as possible candidates.) As such, the list above is not necessarily complete; I hesitate, however, to list all possible validator types I can conceive of for fear of driving over-engineering from the spec, and believe that review of the design of further minor validator types can wait for code review, so long as this overall structure is agreeable.

Alternatives

We have many alternatives here.

First, to the problem of merging our validation, packing, and clean image provisioning logic, we could opt to merge our current image generation code with our service layer. However, this poses real difficulties in testing, as our image generation layer, while functional, lacks the stability of our service layer, and merging it as-is could slow forward progress on the project as we wrestle with CI.

Assuming that we do not wish to merge our current image generation layer, we could begin immediately to implement a new image generation layer in the service side. However, this sort of truly revolutionary step frequently ends in apathy, conflict, or both. Providing an image validation layer, with the possibility of growing into a clean image generation API and, later, an image packing API, is an incremental step which can provide real value in the short term, and which is needed regardless.

Assuming that we are, in fact, building an image validation API, we could wholly separate it from any image preparation logic (including clean image provisioning.) There is a certain purist argument for separation of duties here, but the practical argument that resource testing and enforcement are frequently the same steps suggests that we should merge the two for efficiency.

Assuming that we are allowing reconciliation of the image with the validation layer, we could, instead of building our own lightweight validation layer, demand that plugin authors immediately adopt one of Ansible, Puppet, Chef, Salt, etc. However, three factors lead me not to embrace this option. First, normal usage of these tools expects network access by default; in our context, we do not want to use the external network unless absolutely necessary, as our instances may not be network-enabled. While it is possible to use them offline, it requires some care to do so, which might be offputting for newcomers to Sahara who are versed in the chosen tool. Second, Sahara should not be that opinionated about toolchains, either within our team or to our userbase. Facilitating the usage of devops toolchains by providing a clear, well-encapsulated API point is a good goal, but it is not Sahara's job to pick a winner in that market. Third, such a framework is a significant dependency for the sahara core, and such massive dependencies are always to be regarded with suspicion. As such, providing a very lightweight framework for validations is worthwhile, so that we do not need to depend absolutely on any such framework, even in the short term before plugins are abstracted out of the service repo.

Assuming that we do not wish to immediately adopt such a framework, we could instead decide to immediately build a full-featured idempotent resource description language, building many more validators with many more options. While I may well have missed required, basic options, and welcome feedback, I strongly suggest that we start with a minimal framework and build upon it, instead of trying to build

the moon from the outset. I have aimed in this spec for extensibility over completeness (and as such have left some explicit wiggle room in the set of validators to be implemented in the first pass.)

Data model impact

None.

REST API impact

None; this change is SPI only.

Other end user impact

For plugins using SaharaImageValidators, end-users will be able to modify the .yaml files to add packages or run validation or modification scripts against their images on spawn.

Deployer impact

None.

Developer impact

This SPI method is optional; plugins may, if they're feeling a bit cowboy about things today, continue to spawn from any provided image without testing it. As such, there is no strictly required developer impact with this spec.

Sahara-image-elements impact

None. Sahara-image-elements can keep doing its thing if this is adopted. Future dependent specs may drive changes in how we expect images to be packed (hopefully via an OpenStack API,) but this is not that spec, and can be approved wholly independently.

Sahara-dashboard / Horizon impact

None.

6.16.3 Implementation

Assignee(s)

Primary assignee:
egafford

Other contributors:
ptoscano

Work Items

- Add SPI method and call in provisioning flow; wrap to ensure that if absent, no error is raised.
- Build sahara.plugins.images as specified above, and all listed validators.
- Write .yaml files for CDH and Ambari plugins using this mechanism (other plugins may adopt over time, as the SPI method is optional.)
- Add unit tests.

6.16.4 Dependencies

No new dependencies (though this does provide an extension point for which plugins may choose to adopt new dependencies.)

6.16.5 Testing

Unit testing is assumed, as in all cases. The image validation mechanism itself does not need extensive new integration testing; the positive case will be covered by existing tests. Idempotence testing requires whitebox access to the server, and is not possible in the scenario framework; if this system ever is adopted for image generation, at that point we will have the blackbox hooks to test idempotence by rerunning against a pre-packed image (which should result in no change and a still-valid image.)

6.16.6 Documentation Impact

We will need to document the SPI method, the SaharaImageValidator classes, and the .yaml structure that describes them.

6.16.7 References

None.

CHAPTER

SEVEN

LIBERTY SPECS

7.1 Adding custom scenario to scenario tests

https://blueprints.launchpad.net/sahara/+spec/custom-checks

This specification proposes to add custom tests to scenario tests for more exhaustive testing of Sahara.

7.1.1 Problem description

Now, scenario tests testing of basic functionality and user can not add custom personal tests for check of other functionality in Sahara.

Extra tests should be added:

- checks for mount and available cinder volumes;
- checks for started services on cluster;
- checks for other processes that now not testing

7.1.2 Proposed change

Custom test need add to sahara/tests/scenario/custom_checks and need implement support of this scenarios in scenario tests.

For implementation this spec, need change field parameters for field "scenario" in scenario tests. Now is type "enum", need change to "string" for adding ability set custom tests.

Additionally, should be rewrite sahara/tests/scenario/testcase.py.mako template. Custom tests will be called from module with name in format *check_{name of check}* with method *check()* inside.

All auxiliary methods for current custom check will be written in module with this tests. Methods, for global using in several custom scenario can be implemented in sahara/tests/scenario/base.py.

Alternatives

Tests can be added manually to scenario tests in Base class.

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

7.1.3 Implementation

Assignee(s)

Primary assignee:
esikachev

Work Items

- Adding ability to run custom scenario tests;
- Move scripts from old integration tests to scenario tests as custom checks;
- Adding new custom checks.

7.1.4 Dependencies

None

7.1.5 Testing

None

7.1.6 Documentation Impact

None

7.1.7 References

None

7.2 Adding cluster/instance/job_execution ids to log messages

https://blueprints.launchpad.net/sahara/+spec/logs-improvement

This specification proposes to add more information to Sahara logs.

7.2.1 Problem description

Looking at some Sahara logs it is difficult to determine what cluster/instance/job_execution to which they refer.

Extra information should be added:

- logs associated with cluster creation/scaling/deletion should contain cluster id;
- logs associated with job execution/canceling/deletion should contain job execution id;
- logs associated with operations executed on specific instance should contain id of this instance.

7.2.2 Proposed change

Information can be added to context resource_uuid field and then can be used by ContextAdapter in openstack.common.log for a group of logs.

This change requires additional saving of context in openstack.common.local.store to access context from openstack.common.log

We need to set cluster id and job execution id only once. So, it could be done with 2 methods that will be added to sahara.context: set_current_cluster_id(cluster_id) and set_current_job_execution_id(je_id)

Additionally, instances and their ids are changing during the thread. So, instance id should be set only when operation executes on this instance. It will be provided by class SetCurrentInstanceId and will be used with wrapping function set_current_instance_id(instance_id) this way:

```
with set_current_instance_id(instance_id):
```

Code inside "with" statement will be executed with new context (which includes instance id in resource_uuid field) but outside of it context will stay the same.

If instance and cluster specified, log message will looks like:

```
2014-12-22 13:54:19.574 23128 ERROR sahara.service.volumes [-] [instance:
3bd63e83-ed73-4c7f-a72f-ce52f823b080, cluster: 546c15a4-ab12-4b22-9987-4e
38dc1724bd] message
```

If only cluster specified:

```
2014-12-22 13:54:19.574 23128 ERROR sahara.service.volumes [-] [instance:
none, cluster: 546c15a4-ab12-4b22-9987-4e38dc1724bd] message
```

If job execution specified:

```
2014-12-22 13:54:19.574 23128 ERROR sahara.service.edp.api [-] [instance:
none, job_execution: 9de0de12-ec56-46f9-80ed-96356567a196] message
```

Field "instance:" is presented in every message (even if it's not necessary) because of default value of instance_format='[instance: %(uuid)s] ' that cannot be fixed without config changing.

After implementation of this changes, Sahara log messages should be checed and fixed to avoid information duplication.

Alternatives

Information can be added manually to every log message.

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

7.2.3 Implementation

Assignee(s)

Primary assignee:
apavlov-n

Work Items

- Adding ability to access context from openstack.common.log;
- Adding information about cluster/instance/job execution ids to context;
- Fixing log messages to avoid information duplication.

7.2.4 Dependencies

None

7.2.5 Testing

None

7.2.6 Documentation Impact

None

7.2.7 References

None

7.3 Allow the creation of multiple clusters simultaneously

https://blueprints.launchpad.net/sahara/+spec/simultaneously-creating-multiple-clusters

We want to improve user experience when creating new clusters by allowing the user to create multiple clusters at the same time.

7.3.1 Problem description

When creating new clusters, the user has only the option to start a single cluster. In some cases, for example when dealing with researches, the user needs more than a single cluster with a given template. In order to reduce the work of creating a cluster then going back to create a second one and so on, we want to include the option of creating multiple clusters simultaneously by adding an option of number of clusters.

7.3.2 Proposed change

We want to introduce an option for the user to select how many clusters will be spawned. When creating multiple clusters we will add a sequential number to the given cluster name (hadoop-cluster1, hadoop-cluster2, ...).

The creation workflow would go as follows:

- 1) The users will request the creation of multiple clusters POST v1.1/<project_id>/clusters/multiple using a body as described below.
- 2) The return of this call will be a list of clusters id
- 3) Finally the user will be able to track the cluster state using the ids.

Alternatives

The user keeps creating a cluster at a time.

Data model impact

None.

REST API impact

We need to create a new API call (create_multiple_clusters()) to allow the creation of multiple clusters by passing a new parameter specifying the number of clusters that will be created.

POST v1.1/<project_id>/clusters/multiple

Other end user impact

We will also need to change the python-saharaclient to allow the creation of multiple clusters.

- Request

{
"plugin_name": "vanilla", "hadoop_version": "2.4.1", "cluster_template_id": "1beae95b-fd20-47c0-a745-5125dccbd560", "default_image_id": "be23ce84-68cb-490a-b50e-e4f3e340d5d7", "user_keypair_id": "doc-keypair", "name": "doc-cluster", "count": 2, "cluster_configs": {}
}

- Response

{clusters: ["c8c3fee5-075a-4969-875b-9a00bb9c7c6c",
"d393kjj2-973b-3811-846c-9g33qq4c9a9f"]
}

Deployer impact

None.

Developer impact

None.

Sahara-image-elements impact

None.

Sahara-dashboard / Horizon impact

We need to add a box to allow the user to insert the number of clusters that will be created (default set to 1).

7.3.3 Implementation

Assignee(s)

Primary assignee:
tellesmvn

Work Items

- Implement the backend for creating multiple clusters
- Implement the API change
- Implement Unit tests
- Implement changes to the python-saharaclient
- Implement changes to the UI
- Update WADL file in the api-site repo

7.3.4 Dependencies

None.

7.3.5 Testing

We will implement unit tests.

7.3.6 Documentation Impact

The documentation needs to be updated datailing the new option.

7.3.7 References

None.

7.4 Objects update support in Sahara API

https://blueprints.launchpad.net/sahara/+spec/api-for-objects-update

This specification proposes to add api calls that allows to update objects that currently cant be updated this way.

7.4.1 Problem description

Current Sahara API doesn't support update of some objects, which can be required by some other features (for example it's needed for shared and protected resources implementation that will be proposed in ACL spec later).

Updates are already implemented for node group templates, cluster templates, job binaries, data sources and should be done for clusters, jobs, job executions and job binary internals.

7.4.2 Proposed change

Update operation will be added for cluster, job, job execution and job binary internal objects.

For clusters and jobs only description and name update will be allowed for now. For job binary internals only name will be allowed to update. There is nothing will be allowed to update for job executions, only corresponding methods will be added.

Also will be added support of PATCH HTTP method to modify existing resources. It will be implemented the same way as current PUT method.

Alternatives

None

Data model impact

None

REST API impact

Following API calls will be added:

PATCH /v1.1/{tenant_id}/clusters/{cluster_id}

PATCH /v1.1/{tenant_id}/jobs/{job_id}

PATCH /v1.1/{tenant_id}/job-executions/{job_execution_id}

PATCH /v1.1/{tenant_id}/job-binary-internals/{job_binary_internal_id}

Other end user impact

This update methods will be added to saharaclient API.

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

This update methods will not be added to Horizon yet, but will be added later as part of ACL spec.

7.4.3 Implementation

Assignee(s)

Primary assignee:
apavlov-n

Work Items

- Adding PATCH method;
- Adding new API calls;
- Adding operations to saharaclient;
- Documentation update in api-ref.

7.4.4 Dependencies

None

7.4.5 Testing

Unit tests and API tests in tempest will be added.

7.4.6 Documentation Impact

Sahara REST API documentation in api-ref will be updated.

7.4.7 References

None

7.5 Retry of all OpenStack clients calls

https://blueprints.launchpad.net/sahara/+spec/clients-calls-retry

This specification proposes to add ability of retrying OpenStack clients calls in case of occasional errors occurrence.

7.5.1 Problem description

Sahara uses a bunch of OpenStack clients to communicate with other OpenStack services. Sometimes during this clients calls can be occurred occasional errors that lead to Sahara errors as well. If you make a lot of calls, it may not be surprising if one of them doesn't respond as it should - especially for a service under heavy load.

You make a valid call and it returns a 4xx or 5xx error. You make the same call again a moment later, and it succeeds. To prevent such kind of failures, all clients calls should be retried. But retries should be done only for certain error codes, because not all of the errors can be avoided just with call repetition.

7.5.2 Proposed change

Swift client provides the ability of calls retry by its own. So, only number of retries and retry_on_ratelimit flag should be set during client initialisation.

Neutron client provides retry ability too, but repeats call only if `ConnectionError` occurred.

Nova, Cinder, Heat, Keystone clients don't offer such functionality at all.

To retry calls `execute_with_retries(method, *args, **kwargs)` method will be implemented. If after execution of given method (that will be passed with first param), error occurred, its `http_status` will be compared with http statuses in the list of the errors, that can be retried. According to that, client call will get another chance or not.

There is a list of errors that can be retried:

- `REQUEST_TIMEOUT (408)`
- `OVERLIMIT (413)`
- `RATELIMIT (429)`

- INTERNAL_SERVER_ERROR (500)
- BAD_GATEWAY (502)
- SERVICE_UNAVAILABLE (503)
- GATEWAY_TIMEOUT (504)

Number of times to retry the request to clients before failing will be taken from `retries_number` config value (5 by default).

Time between retries will be configurable (`retry_after` option in config) and equal to 10 seconds by default. Additionally, Nova client provides `retry_after` field in `OverLimit` and `RateLimit` error classes, that can be used instead of config value in this case.

These two config options will be under `timeouts` config group.

All clients calls will be replaced with `execute_with_retries` wrapper. For example, instead of the following method call

```
nova.client().images.get_registered_image(id)
```

it will be

```
execute_with_retries(nova.client().images.get_registered_image, id)
```

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

7.5.3 Implementation

Assignee(s)

Primary assignee:
apavlov-n

Work Items

- Adding new options to Sahara config;
- `execute_with_retries` method implementation;
- Replacing OpenStack clients call with `execute_with_retries` method.

7.5.4 Dependencies

None

7.5.5 Testing

Unit tests will be added. They will check that only specified errors will be retried

7.5.6 Documentation Impact

None

7.5.7 References

None

7.6 Use trusts for cluster creation and scaling

https://blueprints.launchpad.net/sahara/+spec/cluster-creation-with-trust

Creation of cluster can be a pretty long operation. Currently we use sessions that usually expire in less than one hour. So, currently it is impossible to create a cluster that requires more than one hour for spawn.

Sahara could get trust from user and use it whenever it is needed for cluster creation or scaling.

7.6.1 Problem description

Sahara communicates with OpenStack services using session provided by user. Session is created by keystone for comparably small amount of time. Creation of large cluster could require more than session life time. That's why Sahara will not be able to operate cluster at the end of the process.

7.6.2 Proposed change

Sahara could perform all operations with cluster using trust obtained from user. Now trusts are used for termination of transient cluster. The same logic could be used for all operations during cluster creation and scaling.

Since we still support keystone v2, the option should be configurable. I suggest making it enabled by default.

The proposed workflow:

1. User requests cluster creation or scaling.
2. Sahara creates trust to be used for OpenStack operation. This trust is stored in the DB in the cluster's trust_id field.
3. Sahara finishes cluster provisioning or the periodic cluster cleanup task recognizes that cluster activation has timed out and uses the trust to delete the cluster.
4. Sahara deletes the trust.

For safety reasons created trusts should be limited by time, but their life time should be sufficient for cluster creation. Parameter in config file with 1 day default should work well.

Alternatives

The trust id could be stored in memory rather than in the database. However, this implementation would not allow the periodic cluster cleanup task (which is not run in the context of the tenant cluster provisioning request) to successfully delete stale clusters.

It is notable that storing the trust_id in the database will also be of use to us in improving the HA capabilities of cluster creation if we move to a multi-stage, DAG-based provisioning flow.

While potential concerns about storing trust ids in the DB exist, these ids require a valid auth token for either the admin or tenant user to utilize, adding some degree of security in depth in case of a control plane database breach. This mechanism may be further secured by storing all trust ids (for both transient and long-running clusters) via a secret storage module in the future. This change, however, falls outside the scope of this specification.

Data model impact

None.

REST API impact

None.

Other end user impact

None.

Deployer impact

None.

Developer impact

None.

Sahara-image-elements impact

None.

Sahara-dashboard / Horizon impact

None.

7.6.3 Implementation

Assignee(s)

Primary assignee:
alazarev (Andrew Lazarev)

Other contributors:
None

Work Items

- Add option to Sahara config
- Implement new authentication strategy
- Document new behavior

7.6.4 Dependencies

None.

7.6.5 Testing

Manually. CI will cover the feature.

7.6.6 Documentation Impact

Because trusts are now being used to clean up clusters, we will need to document that the periodic cluster cleanup task should be run on a schedule that fits within the expiration period of a trust.

7.6.7 References

None.

7.7 Deprecation of Direct Engine

https://blueprints.launchpad.net/sahara/+spec/deprecate-direct-engine

Currently Sahara has two types of infrastructure engines. First is Direct Infrastructure Engine and second is Heat Infrastructure Engine. This spec proposes deprecating the Direct Engine.

7.7.1 Problem description

Each time when Sahara start support new feature, Sahara should support it in both engines. So, it became much harder to support both versions of engines, because in case of Direct Engine it would need to have duplication of work, which is already done in Heat.

7.7.2 Proposed change

It's proposed to deprecate Direct Engine in Liberty release, but it will be available to use. After merging this spec Direct Engine should be `freezed` for new feautures which will be added in Liberty. It will be opened for fixes of `High` and `Critical` bugs. We should make Heat Engine used by default in Sahara.

This change will allow to switch most testing jobs in Sahara CI to use Heat Engine instead of Direct Engine.

In M release we should remove all operations from direct engine. After that the only operation which can be done with direct-engine-created cluster is the cluster deletion. We should rewrite cluster deletion behavior to support deletion direct-engine-created cluster via Heat Engine. Now heat engine removes cluster from database but doesn't remove all cluster elements (for example, instances).

Alternatives

Sahara can continue support of both engines.

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

Deployers should switch to use Heat Engine instead of Direct Engine.

Developer impact

New features, which impacts infrastructure part of Sahara, should be supported only in Heat Engine.

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

7.7.3 Implementation

Assignee(s)

Primary assignee:
vgridnev

Other contributors:
None

Work Items

This change will require following changes:

- Add deprecation warnings at sahara startup.
- Mark Heat Engine as default in Sahara.
- Document deprecation of Direct Engine.

7.7.4 Dependencies

None

7.7.5 Testing

This change require manual testing of deletion direct-engine-created cluster after switch to heat engine.

7.7.6 Documentation Impact

Need to document that Direct Engine became deprecated.

7.7.7 References

None

7.8 Drop Hadoop v1 support in provisioning plugins

https://blueprints.launchpad.net/sahara/+spec/drop-hadoop-1

This specification proposes to drop old Hadoop versions

7.8.1 Problem description

Support for Hadoop 1 in provisioning plugins has become an unused feature. The Hadoop development for v1 is almost frozen and Hadoop vendors are also dropping its support.

As the sahara-plugins and sahara main repository split is going to happen it will be a lot easier move less plugins to the new repo. Also the number of unit and scenario tests will reduce.

The list of versions to be dropped is:

- Vanilla 1.2.1
- HDP 1.3.2

This spec does not suppose to drop the deprecated versions of plugins. That should be a regular part of the release cycle.

7.8.2 Proposed change

Drop the plugins code for Hadoop v1 along with:

- xml/json resources
- unit and scenario tests
- sample files
- image elements

Alternatives

TBD

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

Elements for building Hadoop v1 images should be dropped as well

Sahara-dashboard / Horizon impact

None

7.8.3 Implementation

Assignee(s)

Primary assignee:
nkonovalov

Work Items

- Disable Hadoop v1 tests in sahara-ci
- Drop related code and resource from sahara main repo
- Drop image elements

7.8.4 Dependencies

None

7.8.5 Testing

Sahara-ci should not test Hadoop v1 anymore

7.8.6 Documentation Impact

Add a note that Hadoop v1 is not supported in new releases.

7.8.7 References

None

7.9 [EDP] Add Spark Shell Action job type

https://blueprints.launchpad.net/sahara/+spec/edp-add-spark-shell-action

The EDP Shell action job type allows users to run arbitrary shell scripts on their cluster, providing a great deal of flexibility to extend EDP functionality without engine changes or direct cluster interface. This specification proposes the addition of this job type for the Spark engine.

A fuller explication of the benefits of this feature can be found in the edp-add-oozie-shell-action specification, which need not be repeated here.

7.9.1 Problem description

While the Oozie engine now supports Shell actions, Spark users do not presently have access to this job type. Its addition would allow the creation of cluster maintenance tools, pre- or post-processing jobs which might be cumbersome to implement in Spark itself, the retrieval of data from filesystems not supported as Sahara data sources, or any other use case possible from a shell command.

7.9.2 Proposed change

From an interface standpoint, the Spark Shell action implementation will follow the Oozie Shell action implementation almost precisely:

- Shell jobs will require a single script binary in `mains`, which will be pushed to the cluster's master node and executed.
- Shell jobs will optionally permit any number of file binaries to be passed as `libs`, which will be placed in the script's working directory and may be used by the script as it executes.
- `configs` will be permitted to allow Sahara EDP-internal features (`substitute_data_source_for_uuid` and `subsitute_data_source_for_name` will be implemented for this job type, as they are for Oozie Shell actions.)
- `params` key-value pairs will be passed to the script as environment variables (whether these are passed into a remote ssh client or injected into the script itself is left to the discretion of the implementer.)
- `args` will be passed to the script as positional command-line arguments.
- The Shell engine for Spark will store files as the main Spark engine does, creating a directory under /tmp/spark-edp/`job_name`/`job_execution_id`, which will contain all required files and the output of the execution.

- The Spark Shell engine will reuse the `launch_command.py` script (as used by the main Spark engine at this time,) which will record childpid, stdout, and stderr from the subprocess for record-keeping purposes.

Spark Shell actions will differ in implementation from Oozie Shell actions in the following ways:

- As Spark jobs and Shell actions which happen to be running on a Spark cluster differ quite entirely, the Spark plugin will be modified to contain two separate engines (provided via an extensible strategy pattern based on job type.) Sensible abstraction of these engines is left to the discretion of the implementer.
- `configs` values which are not EDP-internal will not be passed to the script by any means (as there is no intermediary engine to act on them.)
- Spark Shell actions will be run as the image's registered user, as Spark jobs are themselves. As cluster and VM maintenance tasks are part of the intended use case of this feature, allowing sudo access to the VMs is desirable.

Alternatives

Do nothing.

Data model impact

None.

REST API impact

No additional changes after merge of the Oozie Shell action job type implementation.

Other end user impact

None.

Deployer impact

None.

Developer impact

None.

Sahara-image-elements impact

None.

Sahara-dashboard / Horizon impact

No additional changes after merge of the Shell action job type UI implementation.

7.9.3 Implementation

Assignee(s)

Primary assignee:
sgotliv

Other contributors:
egafford

Work Items

- Add a Shell engine to the Spark plugin, and refactor this plugin to provide an appropriate engine branching on job type.
- Add an integration test for Spark Shell jobs (as per previous plugin- specific Shell job tests).
- Update the EDP documentation to specify that the Spark plugin supports the Shell job type.
- Verify that the UI changes made for Oozie Shell jobs are sufficient to support the Shell job type in the Spark case (as is anticipated).

7.9.4 Dependencies

This change builds on the change [EDP] Add Oozie Shell Job Type.

7.9.5 Testing

- Unit tests to cover the Spark Shell engine and appropriate engine selection within the plugin.
- One integration test to cover running of a simple shell job through the Spark plugin.

7.9.6 Documentation Impact

The EDP sections of the documentation need to be updated.

7.9.7 References

None.

7.10 Allow placeholders in datasource URLs

https://blueprints.launchpad.net/sahara/+spec/edp-datasource-placeholders

This spec is to allow using placeholders in EDP data source URL.

7.10.1 Problem description

Common use case: user wants to run EDP job two times. Now the only way to do that with the same data sources is to erase result of the first run before running job the second time. Allowing to have random part in URL will allow to use output with random suffix.

7.10.2 Proposed change

Introduce special strings that could be used in EDP data source URL and will be replaced with appropriate value.

The proposed syntax for placeholder is %FUNC(ARGS)%.

As a first step I suggest to implement two functions only:

- %RANDSTR(len)% - will be replaced with random string of lowercase letters of length `len`.
- %JOB_EXEC_ID% - will be replaced with the job execution ID.

Placeholders will not be allowed in protocol prefix. So, there will be no validation impact.

List of functions could be extended later (e.g. to have %JOB_ID%, etc.).

URLs after placeholders replacing will be stored in `job_execution.info` field during job_execution creation. This will allow to use them later to find objects created by a particular job run.

Example of create request for data source with placeholder:

```
{
    "name": "demo-pig-output",
    "description": "A data source for Pig output, stored in Swift",
    "type": "swift",
    "url": "swift://edp-examples.sahara/pig-job/data/output.%JOB_EXEC_ID%",
    "credentials": {
        "user": "demo",
        "password": "password"
    }
}
```

Alternatives

Do not allow placeholders.

Data model impact

`job_execution.info` field (json dict) will also store constructed URLs.

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

Horizon need to be updated to display actual URLs for job execution. Input Data Source and Output Data Source sections of job execution details page will be extended to include information about URLs used.

REST will not be changed since new information is stored in the existing 'info' field.

7.10.3 Implementation

Assignee(s)

Primary assignee:
alazarev (Andrew Lazarev)

Other contributors:
None

Work Items

- Implement feature
- Document feature

7.10.4 Dependencies

None.

7.10.5 Testing

Manually.

7.10.6 Documentation Impact

Need to be documented.

7.10.7 References

None

7.11 [EDP] Allow editing datasource objects

https://blueprints.launchpad.net/sahara/+spec/edp-edit-data-sources

Currently there is no way to edit a datasource object. If a path needs to be changed, for example, the datasource must be deleted and a new one created. The most common use case is a situation where a user creates a datasource, runs a job, and receives an error from the job because the path does not exist. Although it is not strictly necessary, editable datasource objects would be a convenience when a user needs to correct a path or credentials.

7.11.1 Problem description

There are no API methods for updating a datasource object in the REST API or at the conductor level.

The only way to correct a datasource is to delete an existing one and create a new one with corrected information. Although it is possible to use the same name, the object id will be different.

If editing is allowed, a user only needs to do a single operation to make corrections. Additionally, the id is preserved so that objects which reference it will reference the corrected path.

In the general case, editing a datasource should not be a problem for a job execution which references it. Once a job execution enters the "RUNNING" state, any information in datasource objects it references has been extracted and passed to the process running the job. Consequently, editing a datasource referenced by running or completed jobs will cause no errors. On relaunch, a job execution will extract the current information from the datasource.

There is only a small window where perhaps editing should not be allowed. This is when a datasource object is referenced by a job execution in the "PENDING" state. At this point, information has not yet

been extracted from the datasource object, and a change during this window would cause the job to run with paths other than the ones that existed at submission time.

7.11.2 Proposed change

Add an update operation to the REST API for datasource objects. Do not allow updates for datasource objects that are referenced by job executions in the "PENDING" state (this can be checked during validation).

Datasource objects referenced by job executions that are not in the PENDING state may be changed. In an existing blueprint and related CR (listed in the reference section) the URLs used by a job execution will be recorded in the job execution when the job enters the RUNNING state. This means that for any running or completed job execution, the list of exact datasource URLs used in the execution will be available from the job execution itself even if the referenced datasource has been edited.

Allow any fields in a datasource object to be updated except for id. The object id should be preserved.

Add the corresponding update operation to the python-saharaclient.

Alternatives

Do nothing

Data model impact

None

REST API impact

Backward compatiblity will be maintained since this is a new endpoint.

PUT /v1.1/{tenant_id}/data-sources/{data_source_id}

Normal Response Code: 202 (ACCEPTED)

Errors: 400 (BAD REQUEST), 404 (NOT FOUND)

Update the indicated datasource object

Example

request

```
PUT http://sahara/v1.1/{tenant_id}/data-sources/{data_source_id}
```

```
{
    "description": "some description",
    "name": "my_input",
    "url": "swift://container/correct_path"
}
```

response

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json
```

```
{
    "created_at": "2015-04-08 20:27:13",
    "description": "some_description",
    "id": "7b25fc64-5913-4bc3-aaf4-f82ad03ea2bc",
    "name": "my_input",
    "tenant_id": "33724d3bf3114ae9b8ab1c170e22926f",
    "type": "swift",
    "updated_at": "2015-04-09 10:27:13",
    "url": "swift://container_correct_path"
}
```

Other end user impact

This operation should be added to the python-saharaclient API as well

$ sahara data-source-update [--name NAME] [--id ID] [--json]

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

To take advantage of this from the Horizon UI, we would need a selectable "Edit" action for each data-source on the datasources page

7.11.3 Implementation

Assignee(s)

Primary assignee:
Trevor McKay

Other contributors:
Chad Roberts

Work Items

Add REST and support methods to Sahara Add operation to python-saharaclient Add operation to data-source screens in Horizon Add to WADL in api-ref

7.11.4 Dependencies

None

7.11.5 Testing

Unit tests in Sahara and python-saharaclient

7.11.6 Documentation Impact

Potentially any user documentation that talks about relaunch, or editing of other objects like templates

7.11.7 References

https://blueprints.launchpad.net/sahara/+spec/edp-datasource-placeholders https://review.openstack.org/#/c/158909/

7.12 [EDP] Allow editing job binaries

https://blueprints.launchpad.net/sahara/+spec/edp-edit-job-binaries

Currently there is no way to edit a job binary. If a path needs to be changed, for example, the job binary must be deleted and a new one created. The most common use case is a situation where a user creates a job binary, runs a job, and receives an error from the job because the path does not exist. Although it is not strictly necessary, editable job binary objects would be a convenience when a user needs to correct a path or credentials.

7.12.1 Problem description

There are no API methods for updating a job binary object in the REST API or at the conductor level.

The only way to correct a job binary is to delete an existing one and create a new one with corrected information. Although it is possible to use the same name, the object id will be different.

If editing is allowed, a user only needs to do a single operation to make corrections. Additionally, the id is preserved so that objects which reference it will reference the corrected path.

In the general case, editing a job binary should not be a problem for a job object that references it. Once a job execution enters the "RUNNING" state, any job binary objects it references indirectly through the job object have been uploaded to the cluster for execution. Consequently, editing a job binary object will cause no errors.

There is only a small window where editing should not be allowed. This is when a job binary object is referenced by a job execution in the "PENDING" state. At this point, binaries have not yet been uploaded

to the cluster and a change during this window would cause the job to run with paths other than the ones that existed at submission time.

Note, the paths of binaries used by a job execution should be recorded in the job execution. This will remove a restriction on editing of paths in a job binary that is referenced by an existing job execution. This will be done in a separate blueprint listed in the references section (similar recording of data source paths used during an execution is supported in another blueprint).

7.12.2 Proposed change

Add an update operation to the REST API for job binary objects. Do not allow updates for job binaries that are referenced by job executions in the "PENDING" state (this can be checked during validation).

Allow the following fields in the job binary to be edited:

- name
- description
- url if the value is not an "internal-db://" path

For binaries stored in the Sahara database, the URL is generated by Sahara and should not be editable.

Add the corresponding update operation to the python-saharaclient.

Alternatives

Do nothing

Data model impact

None

REST API impact

Backward compatiblity will be maintained since this is a new endpoint.

PUT /v1.1/{tenant_id}/job-binaries/{job_binary_id}

Normal Response Code: 202 (ACCEPTED)

Errors: 400 (BAD REQUEST), 404 (NOT FOUND)

Update the indicated job-binary object

Example

request

```
PUT http://sahara/v1.1/{tenant_id}/job-binaries/{job_binary_id}
```

```
{
    "description": "some description",
    "name": "my.jar",
```

(continues on next page)

(continued from previous page)

```
    "url": "swift://container/correct_path"
}
```

response

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json
```

```
{
    "created_at": "2015-04-08 20:48:18",
    "description": "",
    "id": "640ca841-d4d9-48a1-a838-6aa86b12520f",
    "name": "my.jar",
    "tenant_id": "33724d3bf3114ae9b8ab1c170e22926f",
    "updated_at": "2015-04-09 10:48:18",
    "url": "swift://container/correct_path"
}
```

Other end user impact

This operation should be added to the python-saharaclient API as well

$ sahara job-binary-update [--name NAME] [--id ID] [--json]

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

To take advantage of this from the Horizon UI, we would need a selectable "Edit" action for each job binary on the job binaries page

7.12.3 Implementation

Assignee(s)

Primary assignee:
Trevor McKay

Other contributors:
Chad Roberts

Work Items

Add REST and support methods to Sahara Add operation to python-saharaclient Add operation to job binary screens in Horizon Add to WADL in api-ref

7.12.4 Dependencies

This is a blueprint to store the job binary paths in the job execution object. Implementing this first will allow editing of job binaries as long as they are not in the PENDING state. Otherwise, editing will have to be disallowed for job binaries referenced by an existing job execution.

https://blueprints.launchpad.net/sahara/+spec/edp-store-binary-paths-in-job-executions

7.12.5 Testing

Unit tests in Sahara and python-saharaclient

7.12.6 Documentation Impact

Potentially any user documentation that talks about editing of other objects like templates

7.12.7 References

None

7.13 CDH HDFS HA Support

https://blueprints.launchpad.net/sahara/+spec/cdh-ha-support

This blueprint aims to implement HDFS High-Availability (HA) for Cloudra plugin.

7.13.1 Problem description

Currently Cloudera plugin does not support HA for services. We plan to implement HDFS HA as the first step. HA for Yarn and other services will be the later steps.

7.13.2 Proposed change

The implementation of the HDFS HA will be done via CM API enable_nn_ha(). This API will help us enable HDFS HA by giving several info augments.

CDH 5 only supports Quorum-based Storage as the only HA implementation, so we will only implement this. To achieve this, we need to add a Standby NameNode, and several JournalNodes. The JournalNode number should be odd and at least 3. When HDFS HA is enabled, SecondaryNameNode will not be used. So we can reuse the node for SecondaryNameNode for StandbyNameNode.

HDFS HA has several hardware constraints (see the reference link). However, for all resources are virtual in Openstack, we will only require NameNode and StandbyNameNode are on different physical hosts.

Overall, we will implement HDFS as below:

- Add a role JournalNode.
- If JournalNode was selected by user (cluster admin), then HA will be enabled.
- If HA is enabled, we will validate whether JournalNode number meet requirements.
- JournalNode roles will not be really created during cluster creation. In fact they will be used as parameters of CM API enable_nn_ha.
- If HA is enabled, we will use SecondaryNameNode as the StandbyNameNode.
- If HA is enabled, we will set Anti-affinity to make sure NameNode and SecondaryNameNode will not be on the same physical host.
- If HA is enabled, Zookeeper service is required in the cluster.
- After the cluster was started, we will call enable_nn_ha to enable HDFS HA.
- If HA is enabled, in Oozie workflow xml file, we will give nameservice name instead of the NameNode name in method get_name_node_uri. So that the cluster can determine by itself which NameNode is active.

Alternatives

None.

Data model impact

None.

REST API impact

None.

Other end user impact

None.

Deployer impact

None.

Developer impact

None.

Sahara-image-elements impact

None.

Sahara-dashboard / Horizon impact

None.

7.13.3 Implementation

Assignee(s)

Primary assignee:
Ken Chen

Work Items

Changes will be only in sahara/plugins/cdh directory. We will only do this based on CDH 5.4.0 at this stage. CDH 5.0.0 and CDH 5.3.0 plugins will not be supported. Changes were described in the Proposed change section.

7.13.4 Dependencies

None.

7.13.5 Testing

We will only do primitive checks: create a Cloudera cluster with HDFS HA, and see whether it is active.

7.13.6 Documentation Impact

The documentation needs to be updated with information about enabling CDH HDFS HA.

7.13.7 References

- *NameNode HA with QJM <http://www.edureka.co/blog/namenode-high-availability-with-quorum-journal-manager-qjm/>*
- *Introduction to HDFS HA <http://www.cloudera.com/content/cloudera/en/documentation/core/latest/topics/cdh_h*
- *Enable HDFS HA Using Cloudera Manager <http://www.cloudera.com/content/cloudera/en/documentation/core/l*
- *Configuring Hardware for HDFS HA <http://www.cloudera.com/content/cloudera/en/documentation/core/latest/to*

7.14 CDH YARN ResourceManager HA Support

https://blueprints.launchpad.net/sahara/+spec/cdh-ha-support

This blueprint aims to implement YARN ResourceManager (RM) High-Availability (HA) for Cloudra plugin.

7.14.1 Problem description

Currently Cloudera plugin does not support HA for YARN ResourceManager. Therefore we plan to implement RM HA.

7.14.2 Proposed change

The implementation of the RM HA will be done via CM API enable_rm_ha(). This API will help us enable ResourceManager HA by giving several info augments.

To achieve RM HA, we need to add one Standby NodeManager in the node of the cluster templates (Cloudera Plugin only support one Standby NodeManager, while the other implementations might allow >1 Standby NodeManager). When RM HA is enabled, we will start both the Primary and Standby NodeManagers, let the Primary NodeManager be active, and leave the Standby NodManager standby.

CM API enable_rm_ha accepts a new ResourceManager Host new_rm_host_id as parameter. A Standby ResourceManager will be started on new_rm_host_id node.

ResourceManager HA requires Primary ResourceManager and Standby ResourceManager roles deployed on different physical hosts. Zookeeper service is required too.

When ResourceManager HA is enabled, Oozie or applications depended on RM should be able to check all available RMs and get the active RM by themselves. There is no way to automatically switch between RMs smoothly.

Overall, we will implement RM HA as below:

- Add a role YARN_STANDBYRM.
- If YARN_STANDBYRM was selected by user (cluster admin), then YARN RM HA will be enabled.
- If RM HA is enabled, we will check Anti-affinity to make sure YARN_STANDBYRM and YARN_RESOURCEMANAGER will not be on the same physical host.
- If RM HA is enabled, Zookeeper service is required in the cluster.
- If RM HA is enabled, we will create cluster with ResourceManager on node where YARN_RESOURCEMANAGER role is assigned.
- If RM HA is enabled, after the cluster is started, we will call enable_rm_ha to enabled RM HA by using the YARN_STANDBYRM node as parameter.

It should be noted that, if HA is enabled, in Oozie workflow xml file, we need to detect the active ResourceManager in method get_resource_manager_uri and pass it to Oozie each time when we use it. I plan to include this part of codes in later patches.

Alternatives

None.

Data model impact

None.

REST API impact

None.

Other end user impact

None.

Deployer impact

None.

Developer impact

None.

Sahara-image-elements impact

None.

Sahara-dashboard / Horizon impact

None.

7.14.3 Implementation

Assignee(s)

Primary assignee:
Ken Chen

Work Items

Changes will be only in sahara/plugins/cdh directory. We will only do this based on CDH 5.4.0 at this stage. CDH 5.0.0 and CDH 5.3.0 plugins will not be supported. Changes were described in the Proposed change section.

7.14.4 Dependencies

None.

7.14.5 Testing

We will only do primitive checks: create a Cloudera cluster with RM HA, and see whether it is active.

7.14.6 Documentation Impact

The documentation needs to be updated with information about enabling CDH YARN ResourceManager HA.

7.14.7 References

- *Configuring High Availability for ResourceManager (MRv2/YARN) <http://www.cloudera.com/content/cloudera/en/documentation/core/latest/topics/cdh_hag_rm_ha_config.html/>*

7.15 Add support HDP 2.2 plugin

https://blueprints.launchpad.net/sahara/+spec/hdp-22-support

This specification proposes to add new HDP plugin based on Ambari Blueprints [1] with Ambari Management Console.

7.15.1 Problem description

Currently we support old HDP plugin which contains old HDP distribution. Also old HDP plugin looks like unsupported by HortonWorks team every year [2]. Many customers want new version of HDP. New HDP plugin will be based on Ambari Blueprints. Ambari Blueprints are a declarative definition of a cluster. With a Blueprint, you specify a Stack, the Component layout and the Configurations to materialize a Hadoop cluster instance via REST API.

7.15.2 Proposed change

New HDP plugin will support provisioning HDP stack via Ambari Blueprints.

Plugin will support key Sahara features:

- Cinder integration
- Swift integration
- EDP
- Scaling
- Event logs

New HDP plugin will support the following OS: Ubuntu 12.04 and CentOS 6. Aslo new plugin will support mirrors with HDP packages.

New HDP plugin will support all services which supports Ambari. Also new plugin will support HA for NameNode and ResourceManager. Client will be installed on all nodes if selected our process. For example if selected Oozie then will be installed Oozie client on all nodes.

Plugin wil be support the following services:

Service	Process
Ambari	Ambari
Falcon	Falcon Server
Flume	Flume
HBase	HBase Master
	HBase RegionServer
HDFS	NameNode
	DataNode
	SecondaryNameNode
	JournalNode
Hive	Hive Metastore
	HiveServer
Kafka	Kafka Broker
Knox	Knox Gateway
Oozie	Oozie
Ranger	Ranger Admin
	Ranger Usersync
Slider	Slider
Spark	Spark History Server
Sqoop	Sqoop
Storm	DRPC Server
	Nimbus
	Storm UI Server
	Supervisor
YARN	YARN Timeline Server
	MapReduce History Server
	NodeManager
	ResourceManager
ZooKeeper	ZooKeeper

Alternatives

Add support of HDP 2.2 in old plugin, but it is very difficult to do without Ambari Blueprints.

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

Need to add elements for building images with pre-installed Ambari packages. For installing HDP Stack plugin should use mirror with HDP packages. Also should add elements for building local HDP mirror.

Sahara-dashboard / Horizon impact

None

7.15.3 Implementation

Assignee(s)

Primary assignee:
sreshetniak

Other contributors:
nkonovalov

Work Items

- Add base implementation of plugin [3] [4]
- Add elements for building image with Ambari [5]
- Add EDP support [6]
- Add additional services support [7]
- Add scaling support [8]
- Add HA support [9]
- Add elements for building HDP mirror [10]

7.15.4 Dependencies

None

7.15.5 Testing

- Add unit tests for plugin
- Add scenario tests and job on sahara-ci

7.15.6 Documentation Impact

New plugin documentation should be added to Sahara docs.

7.15.7 References

[1] https://cwiki.apache.org/confluence/display/AMBARI/Blueprints

[2] http://stackalytics.com/?module=sahara-group&release=all&company=hortonworks&metric=commits

[3] https://review.openstack.org/#/c/184292/

[4] https://review.openstack.org/#/c/185100/

[5] https://review.openstack.org/#/c/181732/

[6] https://review.openstack.org/#/c/194580/

[7] https://review.openstack.org/#/c/195726/

[8] https://review.openstack.org/#/c/193081/

[9] https://review.openstack.org/#/c/197551/

[10] https://review.openstack.org/#/c/200570/

7.16 Migrate to HEAT HOT language

https://blueprints.launchpad.net/sahara/+spec/heat-hot

This blueprint suggests to rewrite cluster template for Heat from JSON to HOT language.

7.16.1 Problem description

Heat supports two different template languages: YAML-based HOT templates and JSON-based CFN templates (no documentation about it, only a number of examples).

HOT is the de-facto main markup language for Heat. Template Guide recommends to use HOT and contains examples for HOT only.

CFN templates are supported mostly for compatibility with AWS CloudFormation.

Sahara historically uses CFN templates. Given that Sahara is an integrated OpenStack project it would be nice to switch to HOT.

7.16.2 Proposed change

There is no urgent need in switching to HOT. But it would be nice to be compliant with current tendencies in community.

This spec suggests to use HOT template language for sahara heat templates.

This will require changes mostly in .heat resources. Code that generate template parts on the fly should be changed too.

Having templates written on HOT will simplify implementation of new heat-related features like template decomposition.

Alternatives

Do not change anything.

Data model impact

None.

REST API impact

None.

Other end user impact

None.

Deployer impact

None.

Developer impact

None.

Sahara-image-elements impact

None.

Sahara-dashboard / Horizon impact

None.

7.16.3 Implementation

Assignee(s)

Primary assignee:
alazarev (Andrew Lazarev)

Other contributors:
None

Work Items

- Change all .heat files used by Sahara
- Update code that generates parts of template
- Update unit tests
- Make sure that sahara with heat engine still works in all supported configurations

7.16.4 Dependencies

None

7.16.5 Testing

Mostly manually. CI should also cover heat changes.

7.16.6 Documentation Impact

None.

7.16.7 References

- http://docs.openstack.org/developer/heat/template_guide/hot_guide.html

7.17 Decompose cluster template for Heat

https://blueprints.launchpad.net/sahara/+spec/heat-template-decomposition

Currently Sahara creates one large template with a lot of copy-paste resources. Heat features like template composition could help to move all composition work from Sahara to Heat. This will also allow sahara to have individual cluster parts as separate templates and insert them as resources (in comparison to current text manipulations).

7.17.1 Problem description

Currently Sahara serializes cluster resources as text to heat template. There are several issues with this approach:

1. Code duplication. If a node group contains 10 instances the template will contain all instance-dependent resources 10 times.
2. No code validation. There is no guarantee that resulting template will be syntactically correct (Sahara treats it as text). Missing comma in one resource could influence the other resource.
3. Not Sahara's work. Sahara micro-manages the process of infrastructure creation. This is Heat's job.

7.17.2 Proposed change

Use OS::Heat::ResourceGroup for resources inside node group. Each node group will contain only one resource group and specify number of instances needed. Each individual instance of resource group will contain all resources needed for a corresponding sahara instance (nova server, security group, volume, floating ip, etc.).

This change will also prepare ground for node group auto-scaling feature.

Alternatives

None.

Data model impact

None.

REST API impact

None.

Other end user impact

Resulting Sahara stack in Heat will contain nested stacks.

Deployer impact

None.

Developer impact

None.

Sahara-image-elements impact

None.

Sahara-dashboard / Horizon impact

None.

7.17.3 Implementation

Assignee(s)

Primary assignee:
alazarev (Andrew Lazarev)

Other contributors:
None

Work Items

- Add ability to generate separate ResourceGroup for a single instance of node group
- Switch template for node groups to ResourceGroup with specified count of instances
- Update unit tests
- Make sure that Sahara with heat engine still works for all supported configurations

7.17.4 Dependencies

None.

7.17.5 Testing

Manually. CI will also cover changes in heat.

7.17.6 Documentation Impact

None.

7.17.7 References

None.

7.18 Updating authentication to use keystone sessions

https://blueprints.launchpad.net/sahara/+spec/keystone-sessions

Sahara currently uses per access authentication when creating OpenStack clients. This style of authentication requires keystone connections on every client creation. The keystone project has created a mechanism to streamline and improve this process in the form of Session objects. These objects encapsulate mechanisms for updating authentication tokens, caching of connections, and a single point for security improvements. Sahara should migrate its OpenStack client objects to use session objects for all clients.

7.18.1 Problem description

For all OpenStack client instances, sahara uses authentication on a per-client creation basis. For each client object that is requested, a set of credentials are acquired from the context, or the configuration file in the case of admin accounts, which are used to initialize the client object. During this initialization a request is made to the Identity service to determine the user's privileges with respect to the new client.

Sahara must be aware of any changes to the authentication methods for each client as well as any potential security vulnerabilities resulting from the usage of those methods.

This method of authentication does not allow sahara to share any information between clients, aside from the raw credentials. In turn, this introduces brittleness to the sahara/client interface as each authentication relationship must be maintained separately or, worse yet, with a partial shared model.

Having separate interfaces for each client also makes applying security updates more difficult as each client instance must be visited, researched, and ultimately fixed according the specific details for that client.

Although this methodology has served sahara well thus far, the keystone project has introduced new layers of abstraction to aid in sharing common authentication between clients. This shared methodology, the keystoneclient Session object, provides a unified point of authentication for all clients. It serves as a single point to contain security updates, on-demand authentication token updating, common authentication methods, and standardized service discovery.

7.18.2 Proposed change

Sahara should standardize its client authentication code by utilizing keystoneclient Session objects. This change will entail creating a new module, modifying the OpenStack client utility functions, and adding an authentication plugin object to the context.

A new module, `sahara.service.sessions`, will be created to contain utility functions and classes to aid in the creation and storage of session objects. This module will also contain a global singleton for the sessions cache.

`sahara.service.sesssions` will provide a class named `SessionCache` as well as a function to gain the global singleton instance of that class. The `SessionCache` will contain cached session objects that can be reused in the creation of individual OpenStack clients. It will also contain functions for generating the session objects required by specific clients. Some clients may require unique versions to be cached, for example if a client requires a specific certificate file then it may have a unique session. For all other clients that do not require a unique session, a common session will be used.

Authentication for session objects will be provided by one of a few methods depending on the type of session needed. For user based sessions, authentication will be obtained from the keystonemiddleware authentication plugin that is generated with each request. For admin based sessions, the credentials found in the sahara configuration file will be used to generate the authentication plugin. Trust based authentication will be handled by generating an authentication plugin based on the information available in each case, either a token for a user or a password for an admin or proxy user.

The `sahara.context.Context` object will be changed to incorporate an authentication plugin object. When created through REST calls the authentication plugin will be obtained from the keystonemiddleware. When copying a context, the authentication plugin will be copied as well. For other cases the authentication plugin may be set programmatically, for example if an admin authentication plugin is required it can be generated from values in the configuration file, or if a trust based authentication is required it can be generated.

The individual OpenStack client utility modules will be changed to use session and authentication plugin objects for their creation. The sessions will be obtained from the global singleton and the authentication plugin objects can be obtained from the context, or created in circumstances that require more specific authentication, for example when using the admin user or trust based authentication.

The clients for heat and swift do not yet enable session based authentication. These clients should be monitored for addition of this feature and migrated when available.

Alternatives

An alternative to this approach would be to create our own methodology for storing common authentication credentials, but this would be an exercise in futility as we would merely be replicating the work of keystoneclient.

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

7.18.3 Implementation

Assignee(s)

Primary assignee:
Michael McCune (elmiko)

Other contributors:
Andrey Pavlov (apavlov)

Work Items

- create sahara.service.sessions
- modify context to accept authentication plugin
- modify sahara.utils.openstack clients to use sessions
  - cinder
  - keystone
  - neutron
  - nova
- modify admin authentications to use plugin objects
- modify trust authentications to use plugin objects
- create tests for session cache
- create developer documentation for client usage

7.18.4 Dependencies

None

7.18.5 Testing

The tests created for this feature will be unit based, to exercise the code paths and logic points. Functional testing should not be necessary as these authentication methods will be exercised in the course of the standard functional testing.

7.18.6 Documentation Impact

This change will only create documentation within the sahara project. Currently there exists no documentation about client usage within the sahara codebase. This change will add a small section describing how to instantiate clients using the `sahara.utils.openstack` package, with a note about common session authentication.

7.18.7 References

Keystoneclient documentation about using Sessions

How to Use Keystoneclient Sessions (article by Jamie Lennox)

7.19 Manila as a runtime data source

https://blueprints.launchpad.net/sahara/+spec/manila-as-a-data-source

Using manila nfs shares and the mounting features developed for use with job binaries, it should be feasible to use nfs shares to host input and output data. This will allow data to be referenced through local filesystem paths as a another simple alternative to hdfs or swift storage.

7.19.1 Problem description

The work has already been done to support mounting of manila nfs shares at cluster provisioning time or automounting shares for EDP job binaries with a url of the form *manila://<share-id>/path*. Additionally, the Hadoop filesystem APIs already support *file:///path* urls for referencing the local filesystem.

Sahara can build on these existing features by allowing *manila://<share-id>/path* urls for data sources, automounting shares referenced by data sources when necessary, and generating the correct local filesystem urls for EDP jobs at runtime.

Some of the benefits of this approach are:

- parity for job binaries and data sources in regards to manila shares
- the ability to store job binaries and data sources in the same share
- the flexibility to add a new data share to a cluster at any time
- the ability to operate on data from a cluster node using native OS tools
- works on any node where *mount -t nfs ...* is supported
- lays the groundwork for other manila share types in the future

The problem can be divided into three high-level items:

- Add a *manila* data source type with validation of *manila://* urls
- Translate *manila://* urls to *file://* urls for use at job runtime
- Call the existing automounting methods when a data source references a new share

Note, automounting and url translation will only work for manila shares referenced by data source objects. A *manila://* url embedded as a literal in a job config, param, or arg will be ignored. It will not be translated to a *file://* url by Sahara and it will not cause automounting. However, there is a precedent for this -- Sahara currently has other features that are only supported on data source objects, not on literal urls. (It may be possible to remove these limitations in the future through greater use of the unified job mapping interface recently introduced).

7.19.2 Proposed change

A *manila* data source type will be added to the JSON schema for data sources, with appropriate validation of *manilla://* urls.

The existing code in *sahara/service/edp/binary_retrievers/manila_share.py* that supports path name generation and automounting of manila nfs shares for job binaries will be refactored and broken up between *sahara/service/edp/job_utils.py* and *sahara/service/shares.py*. The essential implementation is complete, but this logic needs to be callable from multiple places and in different combinations to support data sources.

Currently, all data source urls are returned to the EDP engines from *get_data_sources()* and *resolve_data_source_references()* in *job_utils.py*. The returned urls are recorded in the job_execution object and used by the EDP engine to generate the job on the cluster. These two routines will be extended to handle manila data sources in the following ways:

- Mount a referenced nfs share on the cluster when necessary. Since an EDP job runs on multiple nodes, the share must be mounted to the whole cluster instead of to an individual instance
- Translate the *manila://* url to a *file://* url and return both urls Since the submission time url and the runtime url for these data sources will be different, both must be returned. Sahara will record the submission time url in the job_execution but use the runtime url for job generation

Alternatives

Do not support *manila://* urls for data sources but support data hosted on nfs as described in

https://review.openstack.org/#/c/210839/

However, these features are complementary, not mutually exclusive, and most of the appartus necessary to make this proposal work already exists.

Data model impact

None

REST API impact

None (only "manila" as a valid data source type in the JSON schema)

Other end user impact

None

Deployer impact

Obviously, if this feature is desired then the manila service should be running

Developer impact

None

Sahara-image-elements impact

None (nfs-utils element is already underway)

Sahara-dashboard / Horizon impact

Sahara needs a manila data source type on the data source creation form

7.19.3 Implementation

Assignee(s)

Primary assignee:
tmckay

Other contributors:
croberts, egafford

Work Items

- Add manila data source type to the JSON schema
- Allow submission time and runtime urls to differ (https://review.openstack.org/#/c/209634/3)
- Refactor binary_retrievers/manila_share.py
- Extend job_utils get_data_sources() and resolve_data_source_references() to handle manila:// urls
- Add manila data source creation to Horizon
- Modify/extend unit tests
- Documentation

7.19.4 Dependencies

https://blueprints.launchpad.net/sahara/+spec/manila-as-binary-store

7.19.5 Testing

Unit tests.

Eventually, as with job binaries, this can be tested with integration tests if/when we have manila support in the gate

7.19.6 Documentation Impact

Discussion of the manila data source type should be added to any sections we currently have that talk about data being hosted in swift of hdfs.

Additionally, we should consider adding information to the Sahara section of the security guide on the implications of using manila data shares.

If the security guide or the manila documentation contains a section on security, this probably can be a short discussion from a Sahara perspective with a link to the security info. If there isn't such a section currently, then probably there should be a separate CR against the security guide to create a section for Manila.

7.19.7 References

7.20 Addition of Manila as a Binary Store

https://blueprints.launchpad.net/sahara/+spec/manila-as-binary-store

Network and distributed filesystems are a useful means of sharing files among a distributed architecture. This core use case makes them an excellent candidate for storage of job binaries.

7.20.1 Problem description

While internal database storage and Swift storage are sensible options for binary storage, the addition of Manila integration allows it as a third option for job binary storage and retrieval. This specification details how this will be implemented.

7.20.2 Proposed change

The scheme `manila` will be added as an option in creation of job binaries. URLs will take the form:

`manila://{share_id}/absolute_path_to_file`

URLs stored as binary locations can be passed through the following validations:

1) The manila share id exists.

2) The share is of a type Sahara recognizes as a valid binary store.

Because manila shares are not mounted to the control plane and should not be, we will not be able to assess the existence or nonexistence of files intended to be job binaries.

We can, however, assess the share type through Manila. For the initial reference implementation, only NFS shares will be permitted for this use; other share types may be verified and added in later changes.

For this binary type, Sahara will not retrieve binary files and copy them into the relevant nodes; they are expected to be reachable through the nodes' own filesystems. Instead, Sahara will:

1) Ensure that the share is mounted to the appropriate cluster nodes (in the Oozie case, the node group containing Oozie server; in the Spark case, the node group containing Spark Master, etc.) If the share is not already mounted, Sahara will mount the share to the appropriate node groups using the mechanism described by blueprint mount-share-api (at the default path) and update the cluster's DB definition to note the filesystem mount.

2) Replace `manila://{share_id}` with the local filesystem mount point, and use these local filesystem paths to build the workflow document or job execution command, as indicated by engine. Job execution can then take place normally.

It is notable that this specification does not cover support of Manila security providers. Such support can be added in future changes, and should not affect this mechanism.

Alternatives

While verification of paths on binary creation would be ideal, mounting tenant filesystems (in the abstract, without a cluster necessarily available) is a prohibitive security concern that outweighs this convenience feature (even if we assume that networking is not an issue.)

We could also create a more general `file://` job binary scheme, either in addition to `manila://` or as a replacement for it. However, this would not particularly facilitate reuse among clusters (without a number of manual steps on the user's part) or allow auto-mounting when necessary.

We could also opt to simply raise an exception if the share has not already been mounted to the cluster by the user. However, as the path to automatic mounting is clear and will be reasonably simple once the mount-share-api feature is complete, automatic mounting seems sensible for the initial implementation.

Data model impact

None.

REST API impact

None.

Other end user impact

None.

Deployer impact

None.

Developer impact

None.

Sahara-image-elements impact

None.

Sahara-dashboard / Horizon impact

Manila will be added as a visible job binary storage option; no other changes.

7.20.3 Implementation

Assignee(s)

Primary assignee:
egafford

Secondary assignee/reviewer:
croberts

Work Items

- Validation changes (URL scheme).
- Integration with mounting feature.
- Creation of "job binary retriever" strategy for Manila (which will mostly no-op, given the strategy above).
- Modification of workflow and execution command code to facilitate this flow.
- Horizon changes (in separate spec).
- Documentation.

7.20.4 Dependencies

None.

7.20.5 Testing

Unit testing is assumed; beyond this, full integration testing will depend on the feasibility of adding a manila endpoint to our CI environment. If this is feasible, then our testing path becomes clear; if it is not, then gated integration testing will not be possible.

7.20.6 Documentation Impact

This feature will require documentation in edp.rst.

7.20.7 References

See https://wiki.openstack.org/wiki/Manila/API if unfamiliar with manila operations.

7.21 API to Mount and Unmount Manila Shares to Sahara Clusters

https://blueprints.launchpad.net/sahara/+spec/mount-share-api

As OpenStack's shared file provisioning service, manila offers great integration potential with sahara, both for shared binary storage and as a data source. While it seems unnecessary to wrap manila's share provisioning APIs in sahara, allowing users to easily mount shares to all nodes of a Sahara cluster in a predictable way will be a critical convenience feature for this integration.

7.21.1 Problem description

Manually mounting shares to every node in a large cluster would be a tedious and error-prone process. Auto-mounting shares that are requested for use in either the data source or binary storage case might be feasible for some use cases. However, outside of our (optional) EDP interface this functionality would never be usable. As such, it is best to provide the user an API for mounting of shares onto Sahara clusters.

7.21.2 Proposed change

This change proposes to expand the node group template, node group, cluster template, and cluster API resources and database objects to contain a "shares" field. As per other node group fields, the cluster template and cluster APIs will allow overwrite of this field (which is particularly critical for composition, given that manila shares represent concrete data rather than abstract resource pools.) At the resource level (for both resource types), this field will be defined by the following jsonschema:

```
"shares": {
    "type": "array",
    "items": {
        "type": "object",
        "properties": {
            "id": {
                "type": "string",
                "format": "uuid"
            },
            "path": {
                "type": ["string", "null"]
            },
            "access_level": {
                "type": ["string", "null"],
                "enum": ["rw", "ro"],
                "default": "rw"
```

(continues on next page)

(continued from previous page)

```
            }
        },
        "additionalProperties": False,
        "required": [
            "id"
        ]
    }
}
```

"id" above refers to the UUID of the manila share to be mounted. It is required.

"path" refers to the local path on each cluster node on which to mount this share, which should be universal across all nodes of the cluster for simplicity. It will default to `/mnt/{share_id}`.

"access_level" governs permissions set in manila for the cluster ips. This defaults to 'rw'.

Because no part of this field requires indexing, it is proposed that the above structure be directly serialized to the database as a TEXT field in JSON format.

Any share specified at the node group level will be mounted to all instances of that node group. Any share specified at the cluster level will be mounted to all nodes of that cluster. At cluster creation, in the case that a specific share id is specified at both the node group and cluster level, the cluster's share configuration (path and access level) will entirely replace the node group's configuration. Any merge of share configurations is seen as needlessly complex and error-prone, and it is our longstanding pattern that cluster-level configurations trump node group-level configurations.

Error cases in this API include:

1. The provided id is not a valid manila share id, as assessed via manilaclient with the user's credentials.
2. The provided path is not a valid, absolute Linux path.
3. Path is not unique (within the set of shares specified for any one node or the set of shares specified for any cluster.)
4. The provided id maps to a manila share type which sahara is not currently equipped to mount.
5. No manila service endpoint exists within the user's service catalog.

On cluster creation (or update, if update becomes an available endpoint,) just after the cluster becomes available and before delegating to the plugin (such that any shares intended for HDFS integration will be in place for the plugin configuration to act upon,) Sahara will execute a share mounting step. For each share, Sahara will take the following steps:

1. Query manila for share information, including share type and defaults.
2. Query the cluster object to find internal ip addresses for all cluster nodes of any node group for which the share should be mounted.
3. For each such node, call to manila to allow access for each ip according to the permissions set on the share.
4. Make a remote call to each qualifying node and mount the share via its mount address as returned from manila.

Steps 1-3 above will be handled via common code in an abstract ShareHandler class. The last step will be delegated to a concrete instance of this class, based on share type as reported by manila, which will

execute appropriate command-line operations over a remote socket to mount the share.

The reference and test implementation for the first revision of this feature will only provide an NFS mounter. An HDFS mounter is the next logical step, but this feature set is already being worked on in parallel to this change and falls outside of the scope of this specification.

Unmounting is a natural extension of this class, but is not covered in this specification.

Alternatives

A more-seamless approach to manila share storage and data sourcing could be attempted, in which no API is exposed to the user, and shares are automatically mounted and unmounted when resources on the share in question are needed (as referenced in a data source URL or binary storage path). However, giving the user the ability to mount and unmount shares at will may allow use cases which we do not anticipate, and particularly in the context of usage of a sahara- provisioned cluster without use of the EDP API, the new API is critical.

It would also be possible to attempt to wrap manila share creation (or even share network creation or network router configuration) in sahara. It seems reasonable, however, to assert that this would be an overstep of our charter, and that asking users to create shares directly through manila will allow them much fuller and up-to-date access to manila's feature set.

On the sahara implementation side, it would be possible to create a new 'share' resource and table, for ease of update and compositional modelling. However, shares will likely never be a top-level noun in sahara; it seems that a field is a better fit for the degree of share management we intend to undertake than an entire resource.

It should be noted that this specification does not attempt to deal with the question of filesystem driver installation across n distributions of Linux and m filesystem types; such an effort is better suited to many specifications and change sets than one. For the first stage of this effort, NFS will be used as the test reference filesystem type.

Note that both binary storage and data source integration are intentionally not handled here. A binary storage specification will build on this spec, but this spec is being posted independently such that the engineers working on data source integration can propose revisions to only the changes relevant to their needs.

Data model impact

A new 'shares' TEXT field will be added to both node groups and node group templates.

REST API impact

A new 'shares' field will be added to the resource for both node groups and node group templates. This field will only allow create functionality in the initial change, as cluster update is currently a sticking point in our API.

Other end user impact

Python-saharaclient will need to be made aware of the new shares field on all supported resources.

Deployer impact

None.

Developer impact

None.

Sahara-image-elements impact

None for the initial change; addition of specialized fs drivers in the future may require image changes.

Sahara-dashboard / Horizon impact

The share mounting feature in Horizon will likely require a separate tab on all affected resources, and is left for a separate spec.

7.21.3 Implementation

Assignee(s)

Primary assignee:
egafford

Secondary assignee/reviewer:
croberts

Work Items

- API Resource modification and call validation.
- DB model modification and testing.
- Manila client integration with Sahara.
- Logical glue code on cluster provisioning.
- ShareMounter abstraction and NFS impl.
- Unit testing.
- Integration testing as feasible (will require manila in CI env for full CI.)
- Update of API WADL site.
- Horizon changes (in separate spec).
- Documentation.

7.21.4 Dependencies

This feature introduces a new dependency on python-manilaclient.

7.21.5 Testing

Unit testing is assumed; beyond this, full integration testing will depend on the feasibility of adding a manila endpoint to our CI environment. If this is feasible, then our testing path becomes clear; if it is not, then gated integration testing will not be possible.

7.21.6 Documentation Impact

This feature will require documentation in features.rst, and will drive changes to the api documentation.

7.21.7 References

See https://wiki.openstack.org/wiki/Manila/API if unfamiliar with manila operations.

7.22 Provide ability to configure most important configs automatically

https://blueprints.launchpad.net/sahara/+spec/recommend-configuration

Now users manually should configure most important hadoop configurations. It would be friendly to provide advices about cluster configurations for users.

7.22.1 Problem description

Now users manually should configure most important hadoop configs, but it's required to have advanced knowledge in Hadoop. Most configs are complicated and not all users know them. We can provide advices about cluster configuration and automatically configure few basic configs, that will improve user experience. Created workaround can extended in future with new confiuguration and advices.

7.22.2 Proposed change

It's proposed to add calculator, which would automatically configure most important configurations in dependency cluster specification: available disk space, ram, cpu, and so on. Such calculator already implemented in Ambari (see [1] and [2]), and we can use it as well. We should have ability to switch off autoconfiguration and if user also manually configured some hadoop config, autoconfiguration also will not be applied.

The following list of configs will be configured, using formulas from [1] and [2]:

- yarn.nodemanager.resource.memory-mb
- yarn.scheduler.minimum-allocation-mb
- yarn.scheduler.maximum-allocation-mb

- yarn.app.mapreduce.am.resource.mb
- yarn.app.mapreduce.am.command-opts
- mapreduce.map.memory.mb
- mapreduce.reduce.memory.mb
- mapreduce.map.java.opts
- mapreduce.reduce.java.opts
- mapreduce.task.io.sort.mb

Also as a simple example we can autoconfigure before cluster validation `dfs.replication` if amout of `datanodes` less than default value.

Also it's required to add new plugin SPI method `recommend_configs` which will autoconfigure cluster configs.

Alternatives

None

Data model impact

It's required to add new column `use_autoconfig` to cluster, cluster_template, node_group, node_group_template, templates_relations objects in DB. By default `use_autoconfig` will be `True`. If `use_autoconfig` is `False`, then we will not use autoconfiguration during cluster creation. If none of the configs from the list above are configured manually and `use_autoconfig` is `True`, then we will autoconfigure configs from list above. Same behaviour will be used for node_groups configs autoconfiguration.

REST API impact

Need to support of switch off autoconfiguration.

Other end user impact

Need to support of switch off autoconfiguration via python-saharaclient.

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

Need to add new checkbox which will allow to swith off autoconfiguration from Horizon during cluster creation/scaling. If plugin doesn't support autoconfig this checkbox will not be displayed. We can use `_info` field at [3] for field.

7.22.3 Implementation

Assignee(s)

Primary assignee:
vgridnev

Other contributors:
sreshetniak

Work Items

Proposed change will consists with following steps:

- Implement new plugin SPI method which will provide configuration advices;
- Add support of this method in following plugins: CDH, Vanilla 2.6.0, Spark (`dfs.replication` only);
- Provide ability to switch on autoconfiguration via UI;
- Provide ability to switch on autoconfiguration via saharaclient;
- Update WADL docs about new feilds objects.

7.22.4 Dependencies

Depends on Openstack requirements

7.22.5 Testing

Unit tests will be implemented for this feature. Sahara CI also can start use autoconfiguration as well.

7.22.6 Documentation Impact

Need to document feature and all rules, which will be used for autoconfiguration.

7.22.7 References

[1] https://apache.googlesource.com/ambari/+/a940986517cbfeb2ef889f0d8a45579b27adad1c/ambari-server/src/main/resources/stacks/HDP/2.0.6/services/stack_advisor.py [2] https://apache.googlesource.com/ambari/+/a940986517cbfeb2ef889f0d8a45579b27adad1c/ambari-server/src/main/resources/stacks/HDP/2.1/services/stack_advisor.py [3] https://github.com/openstack/sahara/blob/master/sahara/service/api.py#L188

7.23 Heat WaitConditions support

https://blueprints.launchpad.net/sahara/+spec/sahara-heat-wait-conditions

Before Heat engine in Sahara Nova was continuously asked for fixed and assigned floating IP and for active SSH connections to VMs. To get rid of such polling mechanism suggested to use Heat WaitConditions feature.

7.23.1 Problem description

Now Sahara checks instances availability via SSH. Wait Condition resource supports reporting signals to Heat. We should report signal to Heat about booting instance.

7.23.2 Proposed change

Add WaitCondition resource to Sahara Heat template.

Alternatives

Using SSH for polling instance accessible.

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

7.23.3 Implementation

Assignee(s)

Primary assignee:
sreshetniak

Work Items

Add Wait Condition support to Sahara

7.23.4 Dependencies

WaitCondition requires pre-installed cloud-init.

7.23.5 Testing

Need to add unit tests for this feature. Integration tests will cover this feature.

7.23.6 Documentation Impact

None

7.23.7 References

http://docs.openstack.org/developer/heat/template_guide/openstack.html#OS::Heat::WaitCondition

7.24 Use Templates for Scenario Tests Configuration

https://blueprints.launchpad.net/sahara/+spec/scenario-test-config-template

Users that want to run the scenario tests available in the Sahara repository need to modify the provided YAML files, which are really template files even if not marked as such. The usability of this process could be improved by extending the test runner to read the templates and perform the substitution of the required variables instead.

7.24.1 Problem description

The scenario tests provided in the Sahara repository are template files even if they are simply marked as YAML files; this is not clear from the README.rst file. Users that want to run them need to manually replace the required variables (the variables are needed because they depend on the environment: host ip, credentials, network type, flavors used, etc). This step needs to be done both by developers and testers, and by wrapper scripts used to run the script on a CI. The repository of the main Sahara CI, sahara-ci-config, contains code which replaces the variables:

https://github.com/stackforge/sahara-ci-config/blob/master/slave-scripts/functions-common.sh#L148

7.24.2 Proposed change

The current template files (mostly under etc/sahara-ci right now) need to be properly identified as templates. The chosen format is Mako, because it is already a dependency for the scenario test runner (runner.py). The files will be marked using a special suffix (file.yaml.mako) and the variables used will be converted in Mako format. The usage of templating would be limited to variable replacement, which means no logic in the templates.

The test runner will continue to handle normal YAML files as usual, in addition to template files.

runner.py will also take a simply INI-style file with the values for the variables used in the template. It will be used by the runner to generate the real YAML files used for the input (in addition to the normal YAML files, if specified).

A missing value for some variable (key not available in the INI file) will raise an exception and lead to the runner termination. This differs from the current behavior of sahara-ci-config code, where a missing values just prints a warning, but given that this would likely bring to a failure, enforcing an early termination limit the resource consumption.

The current sahara-ci-config code allows to specify more details for the replacement variables, like specific end keys where to stop the match for a certain key, but they are likely not needed with a proper choice of names for the variables.

Finally, sahara/tests/scenario/README.rst should be changed to document the currently used variables and the instruction on how to feed the key/value configuration file. The code in sahara-ci-config should be changed as well to create such configuration file and to use the new names of the template files for the respective tests.

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

CI systems which runs tox -e scenario should be updated to use the new filenames.

Developer impact

Developers/QE running tests need to use the new template names.

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

7.24.3 Implementation

Assignee(s)

Primary assignee:
ltoscano

Work Items

The change will be implemented as follow:

1. allow runner.py to use mako template files as input;
2. copy the existing files to the new name and use non-ambiguous variable names for templates;
3. change sahara-ci scripts to use to set the new variables and use the renamed template files;
4. remove the old yaml files;
5. (optional) clean up unnedded code (insert_scenario_value, etc)

Repositories affected: - sahara: 1, 2, 4 - sahara-ci-config: 3, 5

7.24.4 Dependencies

None

7.24.5 Testing

Successful CI run will ensure that the new code did not regress the existing scenarios.

7.24.6 Documentation Impact

sahara/tests/scenario/README.rst needs to be updated.

7.24.7 References

None

7.25 Support of shared and protected resources

https://blueprints.launchpad.net/sahara/+spec/shared-protected-resources

This specification proposes to add ability of creation and modification of shared across tenants and protected from updates objects.

7.25.1 Problem description

Currently all objects created by Sahara are visible only from the tenant in which they were created and not insured from occasional modification or deletion.

7.25.2 Proposed change

This specification proposes to add `is_public` and `is_protected` boolean fields to all Sahara objects that can be accessed through REST API. They will be added to clusters, cluster templates, node group templates, data sources, job executions, jobs, job binaries and job binary internals.

All this objects can be created with enabled `is_public` and `is_protected` parameters which can be updated after creation with corresponding API call. Both of them will be False by default.

If some object has `is_public` field set to `True`, it means that it's visible not only from the tenant in which it was created, but from any other tenants too.

If some object has `is_protected` field set to `True`, it means that it could not be modified (updated, scaled, canceled or deleted) unless this field will be set to `False`. If `is_protected` parameter is set to `True`, object can be modified only if `is_protected=False` will be supplied in update request.

Public objects created in one tenant can be used by other tenants (for example, cluster can be created from public cluster template which is created in another tenant), but to prevent management of resources in different tenants, operations like update, delete, cancel and scale will be possible only from tenant in which object was created.

To control this restrictions, a couple of methods will be implemented in `sahara.service.validation.acl`:

```
def check_tenant_for_delete(context, object)
def check_tenant_for_update(context, object)
def check_protected_from_delete(object)
def check_protected_from_update(object, data)
```

`check_tenant_for_*` will compare tenant_id in context with object tenant_id and if they different, raise an error. But this check should be skipped for periodics as there is no tenant_id in context in this case.

`check_protected_from_delete` will check `is_protected` field and if it's set to True, raise an error. `check_protected_from_update` will additionally check that `is_protected` field wasn't changed to `False` with update data.

This methods will be called mostly in `sahara.db.sqlalchemy.api` inside of update and delete methods that make only db changes. But for cluster_create, cluster_scale, job_execute, job_execution_cancel and job_execution_delete operations they will be called during validation before api calls.

Alternatives

None

Data model impact

Two extra fields `is_public` and `is_protected` will be added to objects listed above.

REST API impact

New API calls will not be added, but existing ones will be updated to support new fields.

Other end user impact

Saharaclient API will be updated to support new fields.

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

`is_public` and `is_protected` checkboxes will be added to `Update` and `Create` panels of each object.

7.25.3 Implementation

Assignee(s)

Primary assignee:
apavlov-n

Work Items

- Adding new fields `is_public` and `is_protected` to objects listed above;
- Implementation of validations, described above;
- Updating saharaclient with corresponding changed;
- Documentation about new features will be added.

7.25.4 Dependencies

None

7.25.5 Testing

Unit tests will be added and a lot of manual testing.

7.25.6 Documentation Impact

All changes will be documented.

7.25.7 References

None

7.26 Running Spark Jobs on Cloudera Clusters 5.3.0

https://blueprints.launchpad.net/sahara/+spec/spark-jobs-for-cdh-5-3-0

This specification proposes to add ability to run Spark Jobs on clusters with CDH (Cloudera Distribution Including Apache Hadoop).

7.26.1 Problem description

Sahara is able to run CDH clusters with running Spark services. However there was no possibility to run Spark jobs on clusters of this type.

7.26.2 Proposed change

The work involves adding a class for running Spark job via Cloudera plugin. Existing Spark engine was changed so that it lets to run Spark jobs with Spark and Cloudera plugins.

Alternatives

Do nothing.

Data model impact

None.

REST API impact

None.

Other end user impact

Required processes: - Master: SPARK_YARN_HISTORY_SERVER - Workers: YARN_NODEMANAGER

Deployer impact

None.

Developer impact

None.

Sahara-image-elements impact

None.

Sahara-dashboard / Horizon impact

None.

7.26.3 Implementation

Assignee(s)

Primary assignee:
Alexander Aleksiyants

Other contributors:
Oleg Borisenko

Work Items

- https://review.openstack.org/#/c/190128/

7.26.4 Dependencies

None.

7.26.5 Testing

- Unit tests to cover CDH engine for working with Spark jobs.
- Unit tests for EDP Spark is now used for Spark Engine and EDP engine.

7.26.6 Documentation Impact

None.

7.26.7 References

None.

7.27 Storm EDP

https://blueprints.launchpad.net/sahara/+spec/storm-edp

This blueprint aims to implement EDP for Storm. This will require a Storm Job Type.

7.27.1 Problem description

Sahara needs an EDP implementation to allow the submission of Storm Jobs.

7.27.2 Proposed change

The implementation of the EDP engine will have 3 basic functions:

- run_job()
- get_job_status()
- cancel_job(kill=False)

This methods are mapped to Storm's: * deploy_toplogy (i.e. storm jar topology-jar-path class ...) * storm list (i.e. storm list) * storm deactivate (i.e storm deactivate topology-name) * storm kill (i.e. storm kill topology-name)

The second part of this implementation is to adapt the UI to allow Storm Job submission.

Alternatives

We may be able to submit Storm jobs as Java Job but it is better for the user to have a specific Storm Job.

Data model impact

None.

REST API impact

None.

Other end user impact

None.

Deployer impact

None.

Developer impact

None.

Sahara-image-elements impact

None.

Sahara-dashboard / Horizon impact

Sahara needs to adapt its UI to allow creation of Storm Jobs. A draft was done by crobertsrh and can be found in https://review.openstack.org/#/c/112408/4

The main changes in the UI will be: * Box for the user to define the main class to be executed * Box for the user to give parameters (if applicable) * Buttons to control job execution (Start, Stop, Kill, View Status) * Since it is possible to have more than one job executing in the same topology the control can be done by job or by topology. In the second case the user will have to choose between the jobs in the topology to control.

7.27.3 Implementation

Assignee(s)

Primary assignee:
tellesmvn

Other contributors:
tmckay (primary review) crobertsrh

Work Items

- Implement Storm Job Type
- Implement EDP engine for Storm
- Implement Unit tests
- Implement integration tests

7.27.4 Dependencies

None.

7.27.5 Testing

First we will implement Unit Tests that follow the example from Spark found in https://github.com/openstack/sahara/blob/master/sahara/tests/unit/service/edp/spark/test_spark.py And also implement the integration tests

7.27.6 Documentation Impact

The documentation needs to be updated with information about Storm EDP and also about Storm Job Type.

7.27.7 References

- *Etherpad <https://etherpad.openstack.org/p/juno-summit-sahara-edp>*
- *Storm Documentation <http://storm.incubator.apache.org/>*

7.28 Storm Scaling

https://blueprints.launchpad.net/sahara/+spec/storm-scaling

This blueprint aims to implement Scaling for Storm.

7.28.1 Problem description

Storm plugin in sahara doesn't have the scaling option implemented yet. This feature is one of the major attractions to building cluster using sahara.

7.28.2 Proposed change

The implementation of the scaling feature following the implementation from Spark plugin.

The implementation will allow users to:

- Scale up a cluster
- Scale down a cluster

Storm is a fairly easy tool to scale. Since it uses Zookeeper as a configuration manager and central point of communication, a new node just needs to configure itself to communicate with the Zookeeper machine and the master node will find the new node. One important point that needs to be taken in consideration is the Storm rebalance action. Once a new node is added, a running topology will not be rescheduled to use the new instance. We are going to call this rebalance action automatically so the user won't have to worry about this call.

Alternatives

None.

Data model impact

None.

REST API impact

None.

Other end user impact

None.

Deployer impact

None.

Developer impact

None.

Sahara-image-elements impact

None.

Sahara-dashboard / Horizon impact

None.

7.28.3 Implementation

Assignee(s)

Primary assignee:
tellesmvn

Other contributors:

Work Items

- Implement Storm Scaling feature
- Implement topology rebalance

7.28.4 Dependencies

None.

7.28.5 Testing

Follow examples on scaling tests from other plugins to implement unit tests for Storm scaling.

7.28.6 Documentation Impact

None.

7.28.7 References

None.

7.29 Support NTP service for cluster instances

https://blueprints.launchpad.net/sahara/+spec/support-ntp

Sahara can support NTP (Network Time Protocol) for clusters instances. NTP is intended to synchronize time on all clusters instances. It can be useful for several services on Cloudera, HDP clusters (see [1] for reference).

7.29.1 Problem description

Sahara should support configuring NTP on cluster instances since it's required to HDP and Cloudera clusters.

7.29.2 Proposed change

First, it's proposed to preinstall ntp daemon on all images via `sahara-image-elements`. For clean images will not configure ntp at all. Also installing ntp on instances of long living clusters can be prevented, as well.

As a second step we should add new common string config option to sahara in `general` section of `cluster configs`, that will allow to specify own NTP server for current cluster. This config option can be supported in all plugins and will allow to install NTP on all cluster instances with specified NTP server. As option, we can allow disabling NTP, at least for fake plugin. So, following plugin options we will have:

1. `NTP_ENABLED`: default value is True, this option is required to allow disabling ntp on cluster instances
2. `NTP_URL`: default value is empty string. So, if user input of this option is empty string, we will use default ntp server from `sahara.conf`. Otherwise, user input will be used.

As the third step, we should provide new config for `sahara.conf` that will to specify default NTP server on current sahara installation. It can useful because default NTP server can be different in different regions. Also it would allow to use NTP server that was installed specially for current lab and current sahara installation.

Second step require to have common options for all plugins to avoid code duplication for all plugins. All options can be added to `plugins/provisioning.py` as well.

Alternatives

We can store `ntp_url` and `ntp_enabled` as cluster column, it's looks like long story for being merged: sahara-side code -> python-saharaclient -> (long long story) horizon support.

Data model impact

We don't need extra migrations, we will store information about NTP server in `general` section of `cluster configs`.

Storing NTP server in separate column in database not really useful, since we can store this information just in cluster configs.

REST API impact

None

Other end user impact

User will have ability to specify own NTP server for current cluster and current sahara installation.

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

It's required to preinstall NTP on all images.

Sahara-dashboard / Horizon impact

Since Sahara already have ability to expose all general config options during cluster-template creation, we don't need extra modifications on Horizon side.

7.29.3 Implementation

Assignee(s)

Primary assignee:
vgridnev

Other contributors:
sreshetniak

Work Items

Proposed change will contain following steps:

1. Install NTP on images in `sahara-image-elements`.
2. Add ability to install NTP on cluster instances and add required config options.
3. Add documentation for feature.

7.29.4 Dependencies

Depends on Openstack requirements

7.29.5 Testing

Feature will covered with integration tests as well.

7.29.6 Documentation Impact

Need to document feature and all config options, which will be used for NTP configuration.

7.29.7 References

[1] http://www.cloudera.com/content/cloudera/en/documentation/core/latest/topics/install_cdh_enable_ntp.html

7.30 Unified Map to Define Job Interface

https://blueprints.launchpad.net/sahara/+spec/unified-job-interface-map

This specification proposes the addition of an "interface" map to the API for job creation, such that the operator registering a job can define a unified, human-readable way to pass in all arguments, parameters, and configurations that the execution of that job may require or accept. This will allow platform-agnostic wizarding at the job execution phase and allows users to document use of their own jobs once in a persistent, standardized format.

7.30.1 Problem description

At present, each of our data processing engines require or may optionally take any of arguments, parameters, configuration values, and data sources (which may be any of 0, 1, or many inputs to 0, 1, or many outputs). This forces users to bear the burden of documenting their own jobs outside of Sahara, potentially for job operators who may not be particularly technical.

A single, human readable way to define the interface of a job, that can be registered at the time of job registration (rather than job execution,) would allow several benefits:

- A more unified UI flow across plugins
- A clean separation of responsibility between the creator of the job (likely a technical user) and the executor of the job
- A means of correcting our current assumptions regarding data sources (where for several plugins we are inappropriately assuming 1 input and 1 output source)

7.30.2 Proposed change

When creating a job, an optional "interface" list may be added to the job json (though we are fundamentally creating a map, presenting a list structure will allow more intuitive ordering and fewer unnecessary error cases.) Each member of this list describes an argument to the job (whether it is passed as a configuration value, a named argument, or a positional argument.)

The interface is described by the following jsonschema object field:

```
"interface": {
    "type": "array",
    "uniqueItems": True,
    "items": {
        "type": "object",
        "properties": {
            "name": {
                "type": "string",
                "minLength": 1
            },
            "description": {
                "type": "string"
            },
            "mapping": {
                "type": "object",
                "properties": {
                    "type": {
                        "type": "string",
                        "enum": ["args", "configs", "params"]
                    },
                    "location": {
                        "type": "string",
                        "minLength": 1
                    }
                },
                "additionalProperties": False,
```

(continues on next page)

(continued from previous page)

```
                "required": [
                    "type",
                    "location"
                ]
            },
            "value_type": {
                "type": "string",
                "enum": ["string",
                         "number",
                         "data_source",
                         "input_data_source",
                         "output_data_source"],
                "default": "string"
            },
            "required": {
                "type": "boolean"
            },
            "default": {
                "type": "string"
            }
        },
        "additionalProperties": False,
        "required": [
            "name",
            "mapping",
            "required"
        ]
    }
}
```

Post-schema validations include:

1) Names must be unique.

2) Mapping is unique.

3) The set of all positional arguments' locations must be an unbroken integer sequence with an inclusive minimum of 0.

4) Positional arguments may not be required, but must be given default values if they are not.

The job execution will also have a simpler interface field definition, described by:

```
"interface": {
    "type": "simple_config"
}
```

New error cases at execution time include:

1) One configuration value or parameter is given two definitions (one through the interface map and one via configs, params, or data sources.)

2) An interface value does not pass validation for the type specified for the field in question.

3) A key in the execution interface map does not equal any key in the job definition's interface map.

4) The specified mapping type is not accepted by the job type being created (for instance, specifying the params type for a Spark job.)

5) An input data source does not contain data.

6) An output data source contains data.

In the case of additional positional values, the positional arguments given in the args list will be appended to the list of interface positional arguments (whether provided or default values.) This will allow for an `*args` pattern, should a plugin permit it.

Params and configs passed via the current mechanism that do not overlap with any key in the execution interface map will be merged and passed to the job as normal. This also applies to $INPUT and $OUTPUT params passed via the input source and output source fields.

Alternatives

In truth, after discussion, it seems that there is not a good alternative to the broad strokes of this plan (save doing nothing). Leaving all configuration of jobs to the execution phase is a real difficulty given that our supported data processing engines simply lack a unified interface. If we wish to create a unified flow, we need to create one; if we want to create one, the job definition phase produces the least user pain, and a simple, flat map is the most legible and flexible thing that can do the job.

Data model impact

A new table will need to be created for storage of interface fields, described by the following DDL (rendered in MySQL syntax for friendliness):

```
CREATE TABLE job_interface_arguments (
    id VARCHAR(36) NOT NULL,
    job_id VARCHAR(36) NOT NULL,
    name VARCHAR(80) NOT NULL, # ex: 'Main Class'
    description TEXT, # ex: 'The main Java class for this job.'
    mapping_type VARCHAR(80) NOT NULL, # ex: 'configs'
    location TEXT NOT NULL, # ex: 'edp.java.main_class'
    value_type VARCHAR(80) NOT NULL, # ex: 'string'
    required BOOL NOT NULL, # ex: 0
    order TINYINT NOT NULL, # ex: 1
    default_value TEXT, # ex: 'org.openstack.sahara.examples.WordCount'
    created_at DATETIME,
    PRIMARY KEY (id),
    FOREIGN KEY (job_id)
        REFERENCES jobs(id)
        ON DELETE CASCADE
);
```

This table will have uniqueness constraints on (job_id, name) and (job_id, mapping_type, location).

Note: While the TEXT type fields above (save Description) could validly be given an upper length limit and stored as VARCHARs, TEXT is safer in the case that a job actually requires an overly long argument,

or is configured with a reasonably massive key. This implementation detail is certainly up for debate re: efficiency vs. usability.

Happily, this change will not require a migration for extant data; the interface fields table has a (0, 1, or many)-to-one relationship to the jobs table, and the existing configs/params/args method of propagating job execution data can continue to function.

REST API impact

The Create Job schema will have a new "interface" field, described above. Each listed exceptional case above will generate a 400: Bad Request error.

This field will also be represented in all GET methods of the Job resource.

The Create Job Execution schema will have a new "interface" field, described above. Each listed exceptional case above will generate a 400: Bad Request error. This field will not be returned on a GET of a job execution object; instead, the final, merged configuration will be returned.

No other impact is foreseen.

Note: I am profoundly open to better options for terminology throughout this document. As "args", "params", and "configs" are already taken, naming of a new option has become difficult. "Interface" and "Interface arguments" seem to me to be the best option remaining in all cases. If you can do one better, please do.

Note: As interface fields will be represented in the data layer as individual records, it would be possible to create an entirely new set of CRUD methods for this object. I believe that course of action to be unnecessarily heavy, however: should the job binary change, the job must be recreated regardless, and a sensible interface need not change for the life of any concrete binary.

Other end user impact

python-saharaclient will require changes precisely parallel to the interface changes described above.

Deployer impact

None.

Developer impact

None.

Sahara-image-elements impact

None.

Sahara-dashboard / Horizon impact

The immediate change does not require a Horizon change. Any UI that utilizes this feature should be represented as a separate blueprint and spec, and will doubtless touch wizarding decisions which are wholly orthogonal to this feature.

7.30.3 Implementation

Assignee(s)

Primary assignee:
egafford

Other contributors:
None

Work Items

1) API updates as specified.
2) DB layer updates as specified.
3) Job execution argument validation and propagation to clusters.
4) Testing.
5) Python-saharaclient updates and testing.

7.30.4 Dependencies

None at present.

7.30.5 Testing

A tempest test will cover injection of each mapping type into jobs (args, configs, params.) This will be tested via a Pig job, as that type may take all of the above. This test will include arguments mapping to both a Swift datasource and an HDFS datasource, to ensure that both URL types are preserved through the flow.

Thorough unit testing is assumed.

7.30.6 Documentation Impact

None that have not already been mentioned.

7.30.7 References

Chat (2014/12/05; begins at 2014-12-05T16:07:55)

7.31 upgrade oozie Web Service API version of sahara edp oozie engine

https://blueprints.launchpad.net/sahara/+spec/update-oozie-client-version

This spec is to upgrade oozie web service api version of oozie engine.

7.31.1 Problem description

Currently sahara oozie server version is 4.1.0, but oozie engine still use v1 oozie web service api which is used in oozie 3.x. By upgrading oozie web service api from v1 to v2, we can add more oozie features into sahara.

7.31.2 Proposed change

change sahara OozieClient job_url and jobs_url from /v1/job/%s, /v1/jobs to /v2/job/%s, /v2/jobs.

for /v2/jobs, remains the same as /v1/jobs

for /v2/job/, there is a difference in the JSON format of job information API,particularly for map-reduce action,no changes for other actions.In v1, externalId and consoleUrl point to spawned child job ID, and exteranlChildIDs is null in map-reduce action. In v2, externalId and consoleUrl point to launcher job ID, and exteranlChildIDs is spawned child job ID in map-reduce action. this exteranlChildIDs can be used for recurrence edp job's child job id.

here are the new oozie features can be added into sahara.

(1)PUT oozie/v2/job/oozie-job-id?action=update, we can update job's definition and properties. (2)GET /oozie/v2/job/oozie-job-id?show=errorlog, we can get oozie error log when job is failed, so we can show user the detail error information. Currently sahara edp engine tells user nothing when job is failed.

so we can add update_job() and show_error_log() into oozie client. details about these two features will be drafted in another spec.

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

7.31.3 Implementation

Assignee(s)

Primary assignee:
luhuichun(lu huichun)

Other contributors:
None

Work Items

- update oozie client in oozie engine

7.31.4 Dependencies

None.

7.31.5 Testing

unit test in edp engine add scenario integration test

7.31.6 Documentation Impact

Need to be documented.

7.31.7 References

oozie web service api http://oozie.apache.org/docs/4.2.0/WebServicesAPI.html

CHAPTER

EIGHT

KILO SPECS

8.1 Add CM API Library into Sahara

https://blueprints.launchpad.net/sahara/+spec/add-lib-subset-cm-api

This specification proposes to add CM API library into Sahara so that the Cloudera plugin will not have to depend on a third party library support.

8.1.1 Problem description

Now the Cloudera plugin is depending on a third party library cm_api (by Cloudera). This library is not python3 compatible, and Cloudera has no resource to fix it. Therefore, it is hard to put this library into requirements.txt file. To fix this issue, we plan to implement a subset of CM APIs and put them into Sahara project, so that the Cloudera plugin will not depend on a third-party library, and we can enable this plugin by default.

8.1.2 Proposed change

Now the CM APIs used in Cloudera plugin include (maybe not all included):

- ApiResource.create_cluster
- ApiResource.get_cluster
- ApiResource.get_all_hosts
- ApiResource.delete_host
- ApiResource.get_cloudera_manager
- ClouderaManager.create_mgmt_service
- ClouderaManager.hosts_start_roles
- ClouderaManager.get_service
- ApiCluster.start
- ApiCluster.remove_host
- ApiCluster.create_service
- ApiCluster.get_service
- ApiCluster.deploy_client_config

- ApiCluster.first_run
- ApiService.create_role
- ApiService.delete_role
- ApiService.refresh
- ApiService.deploy_client_config
- ApiService.start
- ApiService.restart
- ApiService.start_roles
- ApiService.format_hdfs
- ApiService.create_hdfs_tmp
- ApiService.create_yarn_job_history_dir
- ApiService.create_oozie_db
- ApiService.install_oozie_sharelib
- ApiService.create_hive_metastore_tables
- ApiService.create_hive_userdir
- ApiService.create_hive_warehouse
- ApiService.create_hbase_root
- ApiService.update_config
- ApiServiceSetupInfo.add_role_info
- ApiRole.update_config

Those APIs are what we need to implement in our CM APIs. We can create a directory client in plugin cdh directory, and put the lib files in this directory.

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

Deployers will no longer need to install cm_api package anymore.

Developer impact

When new version of CDH is released, if it is incompatible with current used client, or use some new APIs, developer may need to update the client when adding support for new CDH release.

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

8.1.3 Implementation

Assignee(s)

Primary assignee:
ken chen

Other contributors:
ken chen

Work Items

The work items will include:

- Add a directory client in cdh plugin directory, and put lib files under this directory.
- Change all current cm_api imports into using the new client.

8.1.4 Dependencies

None

8.1.5 Testing

Sahara Integration test for CDH plugin is enough.

8.1.6 Documentation Impact

Documents about CDH plugin prerequisites and enabling should be modified, for cm_api is not required any more.

8.1.7 References

https://pypi.python.org/pypi/cm-api/8.0.0

8.2 Add More Services into CDH plugin

https://blueprints.launchpad.net/sahara/+spec/add-cdh-more-services

This specification proposes to add below services into CDH plugins: Flume, Sentry, Sqoop, SOLR, Key-Value Store Indexer, and Impala.

Those services can be added into CDH plugin by using CM API first_run to start them, which can save much effort to prepare and start those services one by one.

8.2.1 Problem description

Now services supported in Sahara CDH plugin is still limited. We want to add more services ASAP. They are Flume, Sentry, Sqoop, SOLR, Key-Value Store Indexer, and Impala.

8.2.2 Proposed change

Since we plan to use first_run to prepare and start those services, we will not need to call other CM APIs for those services in start_cluster() method.

The implementation will need below changes on codes for each service:

- Add process names of the service in some places.
- Add service or process configuration, and network ports to open.
- Add service validation.
- Modify some utils methods, like get_service, to meet more services.
- Some other changes for a few specific services if needed.

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

8.2.3 Implementation

Assignee(s)

Primary assignee:
ken chen

Other contributors:
ken chen

Work Items

The work items will be:

- Change python codes in sahara/sahara/plugins/cdh.
- Add more service resource files in sahara/sahara/plugins/cdh/resources.
- Test and evaluate the change.

8.2.4 Dependencies

None

8.2.5 Testing

Use an integration test to create a cluster.

8.2.6 Documentation Impact

None

8.2.7 References

None

8.3 Add check service test in integration test

https://blueprints.launchpad.net/sahara/+spec/add-service-test-in-integration

This specification proposes to add check services tests in integration test for CDH plugins.Currently we have added zookeeper,HBase,Flume,Sentry,Sqoop, SOLR,Key-Value Store Indexer, and Impala services.

8.3.1 Problem description

Currently we have enabled many new services in CDH plugin.And we want to increase the coverage of the test cases.So we plan to add test cases in the integration test, which will check the availability of those services by using simple scripts like we did in map_reduce_testing.

8.3.2 Proposed change

We plan to write test cases like the way we did in map_reduce_testing. First copy the shell script to the node,then run this script, the script will run basic usage of the services.

The implementation will need below changes on codes for each service:

- Add new cluster template (including all services process) in test_gating_cdh.py
- Add check_services.py (check all services) to check all basic usage of services
- Add shell scripts (check all services)

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

8.3.3 Implementation

Assignee(s)

Primary assignee:
huichun lu

Other contributors:
huichun lu

Work Items

The work items will be:

- Add python codes in `sahara/sahara/tests/integration/tests/gating/test_cdh_gating.py`.
- Add check services scripts files in `sahara/sahara/tests/integrations/ tests/resources`.
- Add check_services.py in `sahara/sahara/tests/integration/tests/`.

8.3.4 Dependencies

None

8.3.5 Testing

Use an integration test to create a cluster.

8.3.6 Documentation Impact

None

8.3.7 References

None

8.4 Add timeouts for infinite polling for smth

https://blueprints.launchpad.net/sahara/+spec/add-timeouts-for-polling

We have infinite polling-processes for smth in sahara code. It would be nice to add timeouts for its execution

8.4.1 Problem description

When creating a cluster, cluster's status may be stuck in Waiting and will always await network if the network configuration is wrong. We can add configurable timeouts for all possible polling processes.

8.4.2 Proposed change

Here you can find list of all polling processes in Sahara code:

For service module:

- volumes._await_attach_volumes
- volumes._create_attach_volume
- volumes._detach_volume
- engine._wait_until_accessible
- engine._await_networks
- direct_engine._await_active
- direct_engine._await_deleted
- edp.job_manager.cancel_job

For utils module:

- openstack.heat.wait_stack_completion

For cdh plugin:

- cloudera_utils.await_agents
- plugin_utils.start_cloudera_manager

For hdp plugin:

- _wait_for_async_request for both plugin versions
- wait_for_host_registrations for both plugin versions
- decommission_cluster_instances for 2.0.6 versionhandler

For spark plugin:

- scaling.decommission_dn

For vanilla plugin:

- hadoop2.scaling._check_decommission
- hadoop2.await_datanodes

- v1_2_1.scaling.decommission_dn
- v1_2_1.versionhandler._await_datanodes

Proposed change would consists of following steps:

- Add new module polling_utils in sahara/utils where would register new timeouts options for polling processes from service and utils modules. Also it would consist with a specific general util for polling. As example it can be moved from https://github.com/openstack/sahara/blob/master/sahara/utils/general.py#L175.
- Add new section in sahara.conf.sample which would consist with all timeouts.
- All plugin specific options would be related only with this plugin and also would be configurable. Also user would have ability to configure all plugin specific options during cluster template creation.

Alternatives

None

Data model impact

This change doesn't require any data models modifications.

REST API impact

This change doesn't require any REST API modifications.

Other end user impact

User would have ability to configure all timeouts separately. Some options would be configurable via sahara.conf.sample, other would be configurable from plugin during cluster template creation.

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

During cluster template creation user would have ability to configure plugin specific options from UI.

8.4.3 Implementation

Assignee(s)

Primary assignee:
vgridnev

Work Items

- Add general polling util to sahara/utils/polling_utils
- Apply this changes to all plugins and sahara engines.

8.4.4 Dependencies

Depends on current Openstack Data Processing Requirements.

8.4.5 Testing

this change would require to add unit tests. Also this change would be tested manually.

8.4.6 Documentation Impact

Required to document this feature in sahara/userdoc/configuration.guide.

8.4.7 References

[1] https://bugs.launchpad.net/sahara/+bug/1402903 [2] https://bugs.launchpad.net/sahara/+bug/1319079

8.5 Authorization Policy Support

https://blueprints.launchpad.net/sahara/+spec/auth-policy

Openstack components are supposed to check user privileges for performed actions. Usually these checks are role-based. See http://docs.openstack.org/developer/keystone/architecture.html#approach-to-authorization-policy. Sahara need to support policies too.

8.5.1 Problem description

OpenStack administrators may want to tune authorization policy for Sahara. There should be a way to restrict some users to perform some of Sahara operations.

8.5.2 Proposed change

Add auth check for all Sahara API endpoints. This could be done in the same way as in other openstack component. There is "policy" module in Oslo library that may do all underlying work.

Proposed content of the policy file:

```
{
    "context_is_admin": "role:admin",
    "admin_or_owner":  "is_admin:True or project_id:%(project_id)s",
    "default": "rule:admin_or_owner",

    "clusters:get_all": "",
    "clusters:create": "",
    "clusters:scale": "",
    "clusters:get": "",
    "clusters:delete": "",

    "cluster-templates:get_all": "",
    "cluster-templates:create": "",
    "cluster-templates:get": "",
    "cluster-templates:modify": "",
    "cluster-templates:delete": "",

    "node-group-templates:get_all": "",
    "node-group-templates:create": "",
    "node-group-templates:get": "",
    "node-group-templates:modify": "",
    "node-group-templates:delete": "",

    "plugins:get_all": "",
    "plugins:get": "",
    "plugins:get_version": "",
    "plugins:convert_config": "",

    "images:get_all": "",
    "images:get": "",
```

(continues on next page)

(continued from previous page)

```
    "images:register": "",
    "images:unregister": "",
    "images:add_tags": "",
    "images:remove_tags": "",

    "job-executions:get_all": "",
    "job-executions:get": "",
    "job-executions:refresh_status": "",
    "job-executions:cancel": "",
    "job-executions:delete": "",

    "data-sources:get_all": "",
    "data-sources:get": "",
    "data-sources:register": "",
    "data-sources:delete": "",

    "jobs:get_all": "",
    "jobs:create": "",
    "jobs:get": "",
    "jobs:delete": "",
    "jobs:get_config_hints": "",
    "jobs:execute": "",

    "job-binaries:get_all": "",
    "job-binaries:create": "",
    "job-binaries:get": "",
    "job-binaries:delete": "",
    "job-binaries:get_data": "",

    "job-binary-internals:get_all": "",
    "job-binary-internals:create": "",
    "job-binary-internals:get": "",
    "job-binary-internals:delete": "",
    "job-binary-internals:get_data": ""
}
```

Separating Sahara users and operators could be the next step.

Alternatives

None.

Data model impact

None.

REST API impact

None.

Other end user impact

None.

Deployer impact

None.

Developer impact

Adding new API will require changing policy rules.

Sahara-image-elements impact

None.

Sahara-dashboard / Horizon impact

None.

8.5.3 Implementation

Assignee(s)

Primary assignee:
alazarev (Andrew Lazarev)

Work Items

- Add policy.py from oslo
- Add config options to control policy file and settings
- Add policy check to all API calls
- Add unit tests
- Add documentation

8.5.4 Dependencies

- Policy module in Oslo.

8.5.5 Testing

- Unit tests
- Manual testing

8.5.6 Documentation Impact

- Feature need to be documented

8.5.7 References

- http://docs.openstack.org/developer/keystone/architecture.html#approach-to-authorization-policy
- http://docs.openstack.org/developer/keystone/api/keystone.openstack.common.policy.html
- http://docs.openstack.org/developer/keystone/configuration.html#keystone-api-protection-with-role-based-access-control-rbac

8.6 CDH HBase Support

https://blueprints.launchpad.net/sahara/+spec/cdh-hbase-support

This specification proposes to add HBase support for CDH Plugin in Sahara.

8.6.1 Problem description

There is no HBase support in current cdh plugin, but Cloudera Manager supports to install this service in a cluster. HBase is a non-relational, distributed database model and can provide BigTable-like capabilities for Hadoop. This service should be supported in Sahara cdh plugin.

8.6.2 Proposed change

The implementation will support CDH 5.0.0. Support Features:

- Install HBase processes in a CDH cluster using cm-api
- Zookeeper must be selected and launched in a cluster first
- Support HMaster and Multiple HRegion processes in a cluster
- Most Configuration Parameters Support in a CDH cluster

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

The end users need to select HMaster and HRegion process in node group templates.

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

It will be required to install the necessary HBase package in the cdh image.

Sahara-dashboard / Horizon impact

None

8.6.3 Implementation

Assignee(s)

Primary assignee:
lu huichun

Other contributors:
weiting-chen

Work Items

The work items can be divided to several parts:

- Investigate HBase service in a CDH cluster via Cloudera Manager(CM)
- Leverage CM-API Client to call functions to install Zookeep via CM
- Test and Evaluate the Concept
- Implement Source Code in Sahara cdh plugin
- Test the Code

8.6.4 Dependencies

Zookeeper process must be installed first in the cluster. HBase needs to use Zookeeper service.

8.6.5 Testing

Writing Unit Tests to basically test the configuration. It also required to have an integration test with the cluster creation.

8.6.6 Documentation Impact

Add HBase in the list and add more information about the configuration.

8.6.7 References

- [1] http://www.cloudera.com/content/cloudera/en/documentation/cdh5/v5-0-0/

8.7 Better Version Management in Cloudera Plugin

https://blueprints.launchpad.net/sahara/+spec/cdh-version-management

This specification proposes to manage different versions of CDH in Cloudera plugin in a better solution.

8.7.1 Problem description

Current situation of CDH version management in CDH plugin:

- We do not have version control for CDH plugin at all.
- Due to backward support requirement, all existing codes need to support CDH version from 5.0.0 to latest (currently it is 5.3.0).

This will cause below issues:

- When new packages are released, we do not know whether our plugin still can work. In fact nobody can ensure that.
- Our work have to be based on a non-static environment.
- Backward-compatibility is desired, but it is not ensured at all. For example, we are not sure tomorrow whether a new released CDH package will be compatible with old plugin codes or configuration files.
- CI-test results are not stable, so we cannot always turn it into voting.
- If new released package version bring issues, it can block all developers even if they do not work on this version.
- When we do not want to support some obsolete versions, we cannot easily remove it.

8.7.2 Proposed change

- Add package version constraints, and give a key packages version list to support CDH5.3.0 (or 5.2.0), and base our development only on this.
- Freeze the package to prevent later CDH release come in.
- Add sub-directories in plugin/cdh, like 5_3_0 for CDH5.3.0 to support different versions of CDH. As we did for vanilla and hdp plugins.
- We also need to do some modifications in sahara-image-elements project to support different versions of CDH image build.
- We need to change sahara-ci-config project to break current cdh tests into several tests for different CDH versions. All new added versions should not break old version tests. For example, current gate-sahara-nova-direct-cdh_ubuntu-aio will be separated into sahara-nova-direct-cdh_5.0_ubuntu-aio and sahara-nova-direct-cdh_5.3.0_ubuntu-aio.
- Break sahara/tests/integration/tests/gating/test_cdh_gating.py into several files, like test_cdh5_0_0_gating.py and test_cdh5_3_0_gating.py to support different CDH versions.
- We need not ensure backward compatibility. E.g., CDH5.3.0 codes are not required to work for CDH5.0.0. We may add some features and codes only for later version of CDH, which was not supported by former CDH version.

- If we want to add more CDH versions in the future, we need to open a BP to do this. For example, if we want to support CDH 5.4.0 in the future, below works are required:
  - Add a directory 5_3_0 including codes to support CDH 5.3.0 in plugin/cdh.
  - Modify scripts in sahara/image-elements/elements/hadoop_cloudera, to add functions to install packages for CDH 5.4.0, while still keeping functions installing packages for former CDH versions.
  - Add a test_cdh5_4_0_gating.py in sahara/tests/integration/tests/gating directory for integration test.
  - Add a new ci-test item in sahara-ci-config, like gate-sahara-nova-direct-cdh_5.4.0_ubuntu-aio. We can set this item as non-voting at the beginning, and after it is tuned and verified ok, we set it back to voting.

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

8.7.3 Implementation

Assignee(s)

Primary assignee:
ken chen

Other contributors:
ken chen

Work Items

The work items will include:

- Change directory structure in sahara/sahara/plugins/cdh to add 5_3_0 and 5_0_0 for CDH5.0.0 and 5.3.0 (We will support those two versions as the first step).
- Retain common codes for each version in the original place, and move version specific codes and files into their own directory.
- Add codes in sahara-image-elements/elements/hadoop-cloudera to support installing different package groups for different CDH versions.
- Add different test items in sahara/tests/integration/tests/gating directory to support different CDH versions
- Add item in sahara-ci-configs project for different CDH versions. At first we mark it as non-voting. After it is verified, we can mark it as voting.
- Test and evaluate the change.

8.7.4 Dependencies

None

8.7.5 Testing

Create clusters of different CDH versions one by one, and do integration tests for each of them.

8.7.6 Documentation Impact

None

8.7.7 References

None

8.8 CDH Zookeeper Support

https://blueprints.launchpad.net/sahara/+spec/cdh-zookeeper-support

This specification proposes to add Zookeeper support for CDH Plugin in Sahara.

8.8.1 Problem description

Currently, Zookeeper isn't supported in the cdh plugin. Zookeeper is a centralized service to provide functions like maintaining configuration, distributed synchronization, and providing group services. It's important to have a Zookeeper to prevent data loss and to avoid a single point failure(SPoF) in a cluster. It has become a basic service to deploy a hadoop cluster in CDH environment.

8.8.2 Proposed change

The implementation will support CDH 5.0.0. Support Features:

- Install Zookeeper Service in a CDH cluster
- Support to run Standalone Zookeeper in a cluster
- Support to run Replicated Zookeepers(Multiple Servers) in a cluster
- To have an option to select Zookeeper in the node group templates
- Most Configuration Parameters Support in a CDH cluster

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

The end users need to select Zookeeper process in their node group templates.

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

It will be required to put the necessary packages in the cdh image.

Sahara-dashboard / Horizon impact

None

8.8.3 Implementation

Assignee(s)

Primary assignee:
ken chen

Other contributors:
weiting-chen

Work Items

The work items can be divided to several parts:

- Investigate Zookeeper service in a CDH cluster via Cloudera Manager(CM)
- Leverage CM-API Client to call functions to install Zookeep via CM
- Test and Evaluate the Concept
- Implement Source Code in Sahara cdh plugin
- Test the Code

8.8.4 Dependencies

None

8.8.5 Testing

Writing Unit Tests to basically test the configuration. It is also required to have an integration test with the cluster creation.

8.8.6 Documentation Impact

Add Zookeeper in the list and add more information about the configuration.

8.8.7 References

- [1] http://www.cloudera.com/content/cloudera/en/documentation/cdh5/v5-0-0/

8.9 Default templates for each plugin

Blueprint: https://blueprints.launchpad.net/sahara/+spec/default-templates

In order to create a basic cluster for any plugin, a user currently has to go through several steps to create an appropriate set of node group and cluster templates. We believe that set of actions should be captured in a default set of templates (per plugin) and loaded into the database ahead of time so that users do not need to repeat some of the most basic steps to provision a simple cluster.

8.9.1 Problem description

Creating even a basic cluster currently requires several steps before the user is able to launch their cluster. In an effort to reduce the amount of effort and time to get a simple cluster running, we propose a set of default templates for each plugin that will be pre-loaded and available for use.

Other potential issues/answers: 1) How do we make the set of templates available for all users/tenants? Today, any given template is only read/writable by one tenant.

A) Implement ACL support for templates (SLukjanov plans to write the spec for this). Having proper ACL support in place will allow us to manage access for read/write across all tenants.

ACL support is not available yet. For the time being, templates will be added per-tenant.

2) Do we allow editing of default templates? I don't think we should allow editing of the default templates since they are shared among all tenants. The flow to "edit" one would be to make a copy of the template and work from there. I propose that each template will be stored with a flag in the database that identifies each default template as being a default template so that we can enforce that users cannot change the default templates.

3) How do we avoid uploading the same default templates each time at startup while still allowing for them to be updated as necessary? We could use a numbering system in the template file names to indicate the version number and store that in the database (perhaps instead of a boolean flag indicating that the template is a default template, we could store an int that is the version number). At startup time, we would go through all of the template files for each plugin and compare the version numbers to the version numbers that are stored in the database.

Sqlalchemy will detect when fields have changed and set the "updated at" field accordingly. Therefore, we can simply attempt to update existing templates whenever the script is run. If the template matches what is in the database, there will be no update.

4) How do we make a json default cluster template reference a json default node group template since we won't know the node group template IDs?

A) CLI util will operate them one-by-one starting with node group templates and then cluster templates. In addition, we could create a few example jobs that could be used for health check.

8.9.2 Proposed change

1) Create sets of default template json files for each plugin.

   A set of default templates will consist of node group templates and/or cluster templates. Whether or not a template file defines a node group template or a cluster template will be determined based on the presence of required fields specified in the Sahara JSON validation code. Node group templates require *flavor_id* and *node_processes*, so presence of these fields implicitly identify a template as a node group template. If it is not a node group template, it is a cluster template. This identification avoids a naming scheme or some other kind of extra labeling to identify a template type.

   Default templates distributed with Sahara will be located in *sahara/plugins/default_templates/*. The code that processes the default templates should use this path as a default starting point but should allow a different starting point to be specified. It should make no assumptions about the directory structure beyond the following:

   - All of the template files in a particular directory should be treated as a set, and cluster templates may reference node group templates in the same set by name.
   - Directories may be nested for logical organization, so that a plugin may define default template sets for each version. Therefore, the code should recurse through subdirectories by default but it should be possible to specify no recursion.

   This design will allow code to process default templates decoupled from explicit knowledge of Sahara plugins, plugin versions, or the structure of plugin directories. The ability to specify a different starting point will allow a user to process particular template sets if the entire set is not desired (only some plugins enabled, for instance), or if an alternate set at a different location is desired. In short, it keeps the processing general and simple.

   In practice, the directories under *sahara/plugins/default_templates* will be named for plugins, and subdirectories will be created for different versions as appropriate.

2) Add a CLI util that can be executed by cron with admin credentials to create/update existing default templates. This utility needs to be able to take some placeholders like "flavor" or "network" and make the appropriate substitutions (either from configs or via command line args) at runtime. The cron job can be optional if we want to force any updates to be triggered explicitly.

   The CLI will take an option to specify the starting directory (default will be *sahara/plugins/default_templates*).

   The CLI will take an option to disable recursion through subdirectories (default will be to recurse).

   At a minimum, the CLI will provide a command to create or update default templates with processing beginning at the designated starting directory.

   The CLI should use the "plugins" configuration parameter from the [database] section to filter the templates that are processed. If the "plugin-name" field of a template matches a plugin name in the "plugins" list, it will be processed. If the "plugins" list is empty, all templates will be processed. It should be possible to override the "plugins" configuration from the command line with a "--plugin-name" option.

   If there is an error during updating or creating templates in a particular set, the CLI should attempt to undo any modifications or creations that were done as part of that set.

   The CLI should also provide a mechanism for deleting default templates, since the *is_default* field will prevent that, should an admin for some reason desire to remove default templates. This can be a simple operation that will remove a single default template by ID. It is not likely to be used very often.

Alternatives

1) The loading process could be done via the REST API if we wanted to have some sort of external process that manages the default templates. That might require changing the API a bit to add endpoints for managing the default templates and seems like a fair bit of unnecessary work since the management of default templates should be something done only within Sahara.

2) Add a hook, possibly in plugins/base:PluginManager for "load_default_templates". This method would be responsible for triggering the loading of the defaults at startup time.

Data model impact

N/A

REST API impact

N/A

Other end user impact

End users will see the default templates show up just like any other template that they may have created.

Deployer impact

N/A

Developer impact

N/A

Sahara-image-elements impact

N/A

Sahara-dashboard / Horizon impact

N/A The default templates will show-up in the UI and look like regular templates.

8.9.3 Implementation

Assignee(s)

Primary assignee:
croberts

Secondary assignee:
tmckay

Work Items

1) Come up with a set of default templates for each plugin. These will probably be json formatted files.

2) Come up with some sort of mechanism to load the templates or ensure that they are already loaded when Sahara starts-up.

3) Update the Sahara documentation.

8.9.4 Dependencies

1) Implementation of the ACL for templates (spec still TBD). This will let us give all users read access to the default templates while still possibly allowing admins to edit the templates.

8.9.5 Testing

Ideally, tests will be added to ensure that a functioning cluster can be started based on each of the default template sets. If that is determined to be too time-consuming per-run, then tests to ensure the validity of each set of templates may be sufficient.

8.9.6 Documentation Impact

The Sahara documentation should be updated to note that the default templates are available for use. Additionally, any future plugins will be expected to provide their own set of default templates.

8.9.7 References

N/A

8.10 Remove support of Hadoop 2.3.0 in Vanilla plugin

https://blueprints.launchpad.net/sahara/+spec/drop-hadoop-2-3-support

At the ATL Design summit it was decided to have 1 OpenStack release cycle management for support of previous Hadoop versions in Vanilla plugin. So it's time to remove 2.3.0 plugin.

8.10.1 Problem description

Current Sahara code contains Vanilla Hadoop 2.3.0 in sources.

8.10.2 Proposed change

The proposed change is to remove all Hadoop 2.3 and any related mentions from Sahara code and sub-projects.

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

Users will not be able to deploy Hadoop 2.3

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

Hadoop 2.3 related elements should be removed

Sahara-dashboard / Horizon impact

None

8.10.3 Implementation

Assignee(s)

Primary Assignee: Sergey Reshetnyak (sreshetniak)

Other Assignees: Sergey Lukjanov (slukjanov) - dib cleanup

Work Items

- Remove Vanilla plugin 2.3.0 from Sahara sources
- Remove DIB stuff related to 2.3.0
- Clear unit/integration tests
- Replace EDP examples with newest version of hadoop-examples
- Documentation changes

8.10.4 Dependencies

None

8.10.5 Testing

None

8.10.6 Documentation Impact

- Documentation must to be updated accordingly

8.10.7 References

None

8.11 Add a common HBase lib in hdfs on cluster start

https://blueprints.launchpad.net/sahara/+spec/edp-add-hbase-lib

HBase applications written in Java require JARs on the HBase classpath. Java actions launched from Oozie may reference JARs stored in hdfs using the *oozie.libpath* configuration value, but there is no standard HBase directory location in hdfs that is installed with Oozie.

Users can build their own HBase directory in hdfs manually from a cluster node but it would be convenient if Sahara provided an option to build the directory at cluster launch time.

8.11.1 Problem description

HBase applications written in Java require the Hbase classpath. Typically, a Java program will be run this way using /usr/bin/hbase to get the classpath:

```
java -cp `hbase classpath`:MyHbaseApp.jar MyHBaseApp
```

Java jobs launched from EDP are Oozie actions, and there is no way to set an extra classpath value. Instead, the Oozie solution to this problem is to create a directory of JAR files in hdfs and then set the *oozie.libpath* configuration property on the job to that location. This causes Oozie to make all of the jars in the directory available to the job.

Sahara currently supports setting the *oozie.libpath* configuration on a job but there is no existing collection of HBase JARs to reference. Users can log into a cluster node and build an HBase directory in hdfs manually using bash or Python. The steps are relatively simple:

- Run the `hbase classpath` command to retrieve the classpath as a string
- Separate the string on the `:` character
- Prune away all paths that do not end in **.jar**
- Upload all of the remaining paths to the designated directory in hdfs

However, it would be relatively simple for Sahara to do this optionally at cluster creation time for clusters that include HBase services.

Note that the same idea of a shared hdfs directory is used in two different but related ways in Oozie:

- the *Oozie sharelib* is a pre-packaged collection of JARs released and supported as part of Oozie and referenced from a job by setting the *oozie.use.system.libpath* configuration parameter to *True*. Sahara already sets this option for all Ooozie-based jobs. The official Oozie sharelib changes over time, and Oozie uses a timestamp naming convention to support upgrades, multiple versions, etc.
- the ability to create an hdfs directory containing JAR files and reference it from a job with the *oozie.libpath* configuration parameter is open to anyone. This is what is being proposed here. This change in no way touches the official *Oozie sharelib*. If Oozie ever adds HBase JARs to the system sharelib, we probably will no longer need this feature.

8.11.2 Proposed change

Create a class that can be shared by any provisioning plugins that support installing the HBase service on a cluster. This class should provide a method that runs remote commands on a cluster node to:

- Run the `hbase classpath` command to retrieve the classpath as a string
- Separate the string on the **:** character
- Prune away all paths that do not end in **.jar**
- Upload all of the remaining paths to pre-determined directory in hdfs

A code sample in the reference section below shows one method of doing this in a Python script, which could be uploaded to the node and executed via remote utilities.

The HBase hdfs directory can be fixed, it does not need to be configurable. For example, it can be "/user/sahara-hbase-lib" or something similar. It should be readable by the user that runs Hadoop jobs on the cluster. The EDP engine can query this class for the location of the directory at runtime.

An option can be added to cluster configs that controls the creation of this hdfs library. The default for this option should be True. If the config option is True, and the cluster is provisioned with an HBase service, then the hdfs HBase library should be created after the hdfs service is up and running and before the cluster is moved to "Active".

A job needs to set the *oozie.libpath* value to reference the library. Setting it directly presents a few problems:

- the hdfs location needs to be known to the end user
- it exposes more "Oozie-ness" to the end user. A lot of "Oozie-ness" has leaked into Sahara's interfaces already but there is no reason to add to it.

Instead, we should use an *edp.use_hbase_lib* boolean configuration parameter to specify whether a job should use the HBase hdfs library. If this configuration parameter is True, EDP can retrieve the hdfs location from the utility class described above and set *oozie.libpath* accordingly. Note that if for some reason an advanced user has already set *oozie.libpath* to a value, the location of the HBase lib should be added to the value (which may be a comma-separated list).

Alternatives

- Do nothing. Let users make their own shared libraries.
- Support Oozie Shell actions in Sahara.

  Shell actions are a much more general feature under consideration for Sahara. Assuming they are supported, a Shell action provides a way to launch a Java application from a script and the classpath can be set directly without the need for an HBase hdfs library.

  Using a Shell action would allow a user to run a script. In a script a user would have complete control over how to launch a Java application and could set the classpath appropriately.

  The user experience would be a little different. Instead of just writing a Java HBase application and launching it with *edp.use_hbase_lib* set to True, a user would have to write a wrapper script and launch that as a Shell action instead.

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

We may want a simple checkbox option on the UI for Java actions to set the *edp.use_hbase_libpath* config so that users don't need to add it by hand.

8.11.3 Implementation

Assignee(s)

Primary assignee:
huichun

Other contributors:
tmckay

Work Items

8.11.4 Dependencies

8.11.5 Testing

An EDP integration test on a cluster with HBase installed would be great test coverage for this since it involves cluster configuration.

Unit tests can verify that the oozie.libpath is set correctly.

8.11.6 Documentation Impact

We need to document how to enable creation of the shared lib at cluster creation time, and how to configure a job to reference it

8.11.7 References

Here is a good blog entry on Oozie shared libraries in general.

http://blog.cloudera.com/blog/2014/05/how-to-use-the-sharelib-in-apache-oozie-cdh-5/

Here is a simple script that can be used to create the lib:

```
#!/usr/bin/python
import sys
import os
import subprocess

def main():
    subprocess.Popen("hadoop fs -mkdir %s" % sys.argv[1], shell=True).wait()
    cp, stderr = subprocess.Popen("hbase classpath",
                                  shell=True,
                                  stdout=subprocess.PIPE,
                                  stderr=subprocess.PIPE).communicate()
```

(continues on next page)

(continued from previous page)

```
    paths = cp.split(':')
    for p in paths:
        if p.endswith(".jar"):
            print(p)
            subprocess.Popen("hadoop fs -put %s %s" % (os.path.realpath(p),
                                                       sys.argv[1]),
                                                       shell=True).wait()

if __name__ == "__main__":
    main()
```

8.12 [EDP] Add Oozie Shell Action job type

https://blueprints.launchpad.net/sahara/+spec/add-edp-shell-action

Oozie shell actions provide a great deal of flexibility and will empower users to easily customize and extend the features of Sahara EDP as needed. For example, a shell action could be used to manage hdfs on the cluster, do pre or post processing for another job launched from Sahara, or run a data processing job from a specialized launcher that does extra configuration not otherwise available from Sahara (ie, setting a special classpath for a Java job).

8.12.1 Problem description

Today if a user finds a limitation in Sahara EDP that prevents them from performing desired processing they have a few choices:

- Log into the cluster nodes with ssh using the specified keypair and perform processing by hand. This may not be available to all users.
- Write a Java application to do the custom processing and run it as a Java job in EDP. This can work, however we as developers know that Java is not often used as a scripting language; it's a little too heavyweight and not everyone knows it.
- Modify the Sahara source. A savvy user might extend Sahara EDP themselves to get the desired functionality. However, not everyone is a developer or has the time to understand Sahara enough to do this.
- Submit a bug or a blueprint and wait for the Sahara team to address it.

However, the existence of shell actions would empower users to easily solve their own problems. With a shell action, a user can bundle files with a script written in bash, Python, etc and execute it on the cluster.

Here is a real-world example of a case that could be easily solved with a shell action:

https://blueprints.launchpad.net/sahara/+spec/edp-add-hbase-lib

In the above blueprint we are calling for additional features in Sahara as a convenience for users, but with a shell action a user could solve this problem on their own. A simple bash script can be written to launch a Java application like this:

```
#!/bin/bash
java -cp HBaseTest.jar:`hbase classpath` HBaseTest
```

In this case a user would associate the script and the Java application with the shell job as job binaries and Sahara would execute the script.

In a similar case, Sahara EDP itself uses a Python wrapper around spark-submit to run Spark jobs. A shell action makes these kinds of launchers easily available to end users.

8.12.2 Proposed change

Add a *Shell* job type that is implemented by the Oozie EDP engine. (It is possible that other EDP engines, such as the Spark or Storm engines, could share a basic shell command implementation for such jobs but that would be another CR).

Shell jobs can use the existing *mains* and *libs* fields in a job execution. The script identified in *mains* will be the script that Sahara runs and the files in *libs* will be supporting files bundled in the working directory by Oozie.

As with other job types, shell actions will support *configs* and *args* passed in the existing *job_configs* field of a job execution. Values in *configs* are specified in the Oozie workflow with **<configuration>** tags and will be available in a file created by Oozie. The *args* are specified with the **<argument>** tag and will be passed to the shell script in order of specification.

In the reference section below there is a simple example of a shell action workflow. There are three tags in the workflow that for Sahara's purposes are unique to the *Shell* action and should be handled by Sahara:

- **<exec>script</exec>** This identifies the command that should be executed by the shell action. The value specified here will be the name of the script identified in *mains*. Technically, this can be any command on the path but it is probably simpler if we require it to be a script. Based on some experimentation, there are subtleties of path evaluation that can be avoided if a script is run from the working directory
- **<file>support.jar</file>** This identifies a supporting file that will be included in the working directory. There will be a <file> tag for every file named in *libs* as well as the script identified in *mains*.

  (Note that the <file> tag can be used in Oozie in any workflow, but currently Sahara does not implement it at all. It is necessary for the shell action, which is why it's mentioned here. Whether or not to add general support for <file> tags in Sahara is a different discussion)
- **<env-var>NAME=VALUE</env-var>** The env-var tag sets a variable in the shell's environment. Most likely we can use the existing *params* dictionary field in *job_configs* to pass env-var values even if we want to label them as "environment variables" in the UI.

Alternatives

Do nothing.

Data model impact

This change adds a new job type, but since job types are stored as strings it should not have any data model impact.

REST API impact

Only a change in validation code for job type

Other end user impact

None

Deployer impact

There may be security considerations related to *Shell Action Caveats*, bullet number 2, in the URL listed in the reference section.

It is unclear whether or not in Sahara EDP the user who started the TaskTracker is different than the user who submits a workflow. This needs investigation -- how is Sahara authenticating to the Oozie client? What user is the Oozie server using to deploy jobs?

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

We would need a new form for a Shell job type submission. The form should allow specification of a main script, supporting libs, configuration values, arguments, and environment variables (which are 100% analogous to params from the perspective of the UI)

8.12.3 Implementation

Assignee(s)

Primary assignee:
egafford

Other contributors:
tmckay

Work Items

- Investigate user issue mentioned above (who is the user that runs shell actions in Sahara and what are the implications?)
- Add a Shell job type and an implementation in the Oozie EDP engine components under the *workflow_creator* directory
- Update job validation routines to handle the Shell job type
- Add an integration test for Shell jobs
- Update the EDP documentation to describe the Shell job type
- Add a UI form for Shell job submission

8.12.4 Dependencies

None

8.12.5 Testing

- Unit tests to cover creation of the Shell job
- Integration tests to cover running of a simple shell job

8.12.6 Documentation Impact

The EDP sections of the documentation need updating

8.12.7 References

http://blog.cloudera.com/blog/2013/03/how-to-use-oozie-shell-and-java-actions/

A simple Shell action workflow looks like this:

```
<workflow-app xmlns='uri:oozie:workflow:0.3' name='shell-wf'>
  <start to='shell1' />
  <action name='shell1'>
      <shell xmlns="uri:oozie:shell-action:0.1">
          <job-tracker>${jobTracker}</job-tracker>
          <name-node>${nameNode}</name-node>
          <configuration>
              <property>
                <name>mapred.job.queue.name</name>
                <value>default</value>
              </property>
          </configuration>
          <exec>doit.sh</exec>
          <argument>now</argument>
          <env-var>VERSION=3</env-var>
          <file>HBaseTest.jar</file>
```

(continues on next page)

(continued from previous page)

```
          <file>doit.sh</file>
      </shell>
      <ok to="end" />
      <error to="fail" />
  </action>
  <kill name="fail">
      <message>oops!</message>
  </kill>
  <end name='end' />
</workflow-app>
```

8.13 JSON sample files for the EDP API

https://blueprints.launchpad.net/sahara/+spec/edp-api-json-samples

Provide sample JSON files for all EDP Sahara APIs to facilitate ease of use by command-line users.

8.13.1 Problem description

As an End User who prefers the Sahara CLI to its UI, I want a set of pre-constructed example JSON payloads for the Sahara EDP API so that I can easily learn the expected API signatures and modify them for my use.

8.13.2 Proposed change

Example JSON payloads will be added to the directory `sahara/etc/edp-examples/json-api-examples/v1.1`, with a subpath for each relevant manager (data_source, job, job_binary, job_binary_internals, and job_execution.) It is intended that examples for future API versions will follow this path structure.

Each file will be named after the pattern: `method_name.[variety.]json`, where variety is optional and will be used to describe independently useful variations in payloads for one method (as in varying engines underlying data sources or processing jobs.)

Alternatives

A tool could conceivably be created to generate template payloads from the jsonschemata themselves. However, as the core use of this change is to provide immediately available, semantically valid payloads for ease of adoption, it is proposed that providing raw examples will better meet the perceived user need.

It would also be possible to package these examples directly with the python-saharaclient repository, an option which has much to recommend it. However, as these examples are globally useful to any non-UI interface, as they are reliant on the jsonschemata in the core repository for testing, and as the extant etc/edp-examples path is a natural home for them, placing them in the sahara repository itself seems indicated.

Data model impact

None.

REST API impact

None.

Other end user impact

None, though it is intended that the payloads may be used via the python-saharaclient.

Deployer impact

None.

Developer impact

None.

Sahara-image-elements impact

None.

Sahara-dashboard / Horizon impact

None.

8.13.3 Implementation

Assignee(s)

Primary assignee:
egafford

Other contributors:
tmckay (primary review)

Work Items

- Payload generation: data_source
- Payload generation: job
- Payload generation: job_binary
- Payload generation: job_binary_internals
- Payload generation: job_execution
- Addition of schema validation unit tests for all the above.

8.13.4 Dependencies

None.

8.13.5 Testing

After discussion with croberts and tmckay, it is proposed that integration testing is in this case unnecessary; these examples constitute documentation. While exhaustive testing is possible in this case, the resultant bloat of the CI build would be disproportionate to the utility of the change.

Unit testing will validate that these resources pass schema validation for their intended APIs.

8.13.6 Documentation Impact

This task is itself a documentation effort. A README.rst will be provided in place, in keeping with pre-existent etc/edp-examples.

8.13.7 References

- https://etherpad.openstack.org/p/kilo-summit-sahara-edp

8.14 [EDP] Add options supporting DataSource identifiers in job_configs

https://blueprints.launchpad.net/sahara/+spec/edp-data-sources-in-job-configs

In some cases it would be convenient if users had a way to pass a DataSource object as a job configuration value to take advantage of the data codified in the object instead of copying and specifying that information manually. This specification describes options that allow users to reference DataSource objects in configuration values, parameters, and arguments in a JobExecution record.

8.14.1 Problem description

With the exception of Java and Spark all job types in Sahara take a single input DataSource object and a single output DataSource object. The DataSource object ids are passed as part of a job execution request and Sahara configures the job based on the information in the objects. Sahara assumes that jobs at runtime will consume the path information using a fixed set of parameters and configuration values. This works well in the general case for the constrained job types supported by Oozie/hadoop (Hive, Pig, MapReduce) but there are cases where DataSource objects currently may not be used:

- Java and Spark job types do not require DataSources since they have no fixed arg list. Currently input and output paths must be specified as URLs in the `args` list inside of `job_configs` and authentication configs must be manually specified.
- Hive, Pig, and MapReduce jobs which use multiple input or output paths or consume paths through custom parameters require manual configuration. Additional paths or special configuration parameters (ie anything outside of Sahara's assumptions) require manual specification in the `args`, `params`, or `configs` elements inside of `job_configs`.

Allowing DataSources to be referenced in `job_configs` is an incremental improvement that gives users the option of easily using DataSource objects in the above cases to specify IO.

8.14.2 Proposed change

Add optional boolean config values on a JobExecution that cause Sahara to to treat values in `job_configs` as potential names or uuids of DataSource objects. This applies to configuration values, parameters, and arguments for all job types for maximum flexibility -- Hive and Pig jobs use parameters to pass values, MapReduce uses configuration values, and Java and Spark use arguments.

If Sahara resolves a value to the name or uuid of a DataSource it will substitute the path information from the DataSource for the value and update the job configuration as necessary to support authentication. If a value does not resolve to a DataSource name or uuid value, the original value will be used.

Note that the substitution will occur during submission of the job to the cluster but will *not* alter the original JobExecution. This means that if a user relaunches a JobExecution or examines it, the original values will be present.

The following non mutually exclusive configuration values will control this feature:

- edp.substitue_data_source_for_name (bool, default False)

  Any value in the `args` list, `configs` dict, or `params` dict in `job_configs` may be the name of a DataSource object.

- edp.substitute_data_source_for_uuid (bool, default False)

  Any value in the `args` list, `configs` dict or `params` dict in `job_configs` may be the uuid of a DataSource object.

This change will be usable from all interfaces: client, CLI, and UI. The UI may choose to wrap the feature in some way but it is not required. A user could simply specify the config options and the DataSource identifiers on the job execution configuration panel.

Alternatives

A slightly different approach could be taken in which DataSource names or uuids are prepended with a prefix to identify them. This would eliminate the need for config values to turn the feature on and would allow individual values to be looked up rather than all values. It would be unambiguous but may hurt readability or be unclear to new users.

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None required. However, I can imagine that the UI might benefit from some simple tooling around this feature, like checkboxes to enable the feature on a Spark or Java job submission panel.

8.14.3 Implementation

Assignee(s)

Primary assignee:
<tmckay>

Other contributors:
<croberts>

Work Items

- Support in Sahara
- Document
- Support in the UI (optional, it will work without additional work)

8.14.4 Dependencies

8.14.5 Testing

Unit tests

8.14.6 Documentation Impact

We will need to document this in the sections covering submission of jobs to Sahara

8.14.7 References

8.15 Enable Swift resident Hive tables for EDP with the vanilla plugin

https://blueprints.launchpad.net/sahara/+spec/edp-hive-vanilla-swift

vanilla1 plugin supports Hive but Hive can't access Swift. vanilla2 plugin not supports Hive. This proposal aims that Hive can process the table that stored in Swift and add hive support to vanilla2 plugin.

8.15.1 Problem description

When Hive processes table that stored in Swift, *hiveserver* has to access Swift. But *hiveserver* cannot read SwiftFS configuration from Hive query. *hiveserver* reads configurations from xml files only (core-site.xml/hive-site.xml). Therefore *hiveserver* doesn't know the authentication info and can't access Swift.

In vanilla2 plugin, doesn't implemented hive support code.

8.15.2 Proposed change

hiveserver reads configuration at startup time only and doesn't read configuration by hive query. Therefore sahara have to pass the swift authentication info to *hiveserver* through hive-site.xml before launching *hiveserver*.

When Hive enabled cluster created, Sahara creates keystone TRUST and Swift proxy user (cluster-scope). And Sahara writes swift proxy user and TRUST to hive-site.xml before *hiveserver* started.

This will enable *hiveserver* to read a authentication info at startup-time. And when hive query arrived, *hiveserver* can access the swift with cluster-scoped TRUST.

When cluster terminated, Sahara removes TRUST and proxy user before cluster's database entry removed. If error on removing a TRUST, cluster goes error status and not removed. Therefore there is no orphaned proxy user.

In vanilla2 plugin, implement hive support code in reference to vanilla1.

Alternatives

1. Adds auth config field (*fs.swift.service.sahara.username*, *fs.swift.service.sahara.password*) to hive-default.xml. And end-user inputs username/password for cluster configurations. I think this alternative is very inconvenience.

2. Fix hive-server to read a configuration from hive query.

Data model impact

Database schema: No change

Cluster config: Change. But no affects others
cluster-config is dict. Adds internal key and stores swift proxy user info and TRUST-id. This key doesn't send to client.

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

8.15.3 Implementation

Assignee(s)

Primary assignee:
k.oikw (Kazuki OIKAWA)

Other contributors:
None

Work Items

- Implement proxy user / TRUST creation feature when cluster creates
- Implementation of sanitizing cluster-config
- Implementation of writing proxy user and TRUST to hive-site.xml
- Implement hive support code in vanilla2

8.15.4 Dependencies

- https://blueprints.launchpad.net/sahara/+spec/edp-swift-trust-authentication

8.15.5 Testing

We will add a integration test. This test checks whether Hive can process the table that stored in the swift executes successfully.

8.15.6 Documentation Impact

None

8.15.7 References

None

8.16 [EDP] Improve Java type compatibility

https://blueprints.launchpad.net/sahara/+spec/edp-improve-compatibility

Currently, EDP MapReduce (Java type) examples must add modifications to be able to use from a java action in an Oozie workflow.

This bp aims that users can migrate from other Hadoop cluster to Sahara without any modifications into their applications.

8.16.1 Problem description

Users need to modify their MapReduce programs as below:

- Add *conf.addResource* in order to read configuration values from the <*configuration*> tag specified in the Oozie workflow:

```
// This will add properties from the <configuration> tag specified
// in the Oozie workflow.  For java actions, Oozie writes the
// configuration values to a file pointed to by ooze.action.conf.xml
conf.addResource(new Path("file:///",
                          System.getProperty("oozie.action.conf.xml")));
```

- Eliminate *System.exit* for following restrictions of Oozie's Java action. e.g. *hadoop-examples.jar* bundled with Apache Hadoop has been used *System.exit*.

First, users would try to launch jobs using examples and/or some applications executed on other Hadoop clusters (e.g. Amazon EMR). We should support the above users.

8.16.2 Proposed change

We will provide a new job type, called Java EDP Action, which overrides the Main class specified by *main_class*. The overriding class adds property and calls the original main method. The class also catches an exception that is caused by *System.exit*.

Alternatives

According to Oozie docs, Oozie 4.0 or later provides the way of overriding an action's Main class (3.2.7.1). The proposing implementation is more simple than using the Oozie feature. (We will implement this without any dependencies of Oozie library.)

Data model impact

None

REST API impact

None

Other end user impact

Users will no longer need to modify their applications to use EDP.

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

sahara-dashboard / horizon needs to add this new job type.

8.16.3 Implementation

Assignee(s)

Primary assignee: Kazuki Oikawa (k.oikw)

Other contributors: Yuji Yamada (yamada-yuji)

Work Items

- Add new job type (*Java.EDP*)
  - *Java.EDP* will be subtype of *Java*
  - Implement of uploading jar file of overriding class to HDFS
  - Implement of creating the *workflow.xml*
- Implement the overriding class

8.16.4 Dependencies

None

8.16.5 Testing

We will add a integration test. This test checks whether WordCount example bundled with Apache Hadoop executes successfully.

8.16.6 Documentation Impact

If EDP examples use this feature, the docs need to update.

8.16.7 References

- Java action in Oozie http://oozie.apache.org/docs/4.0.0/WorkflowFunctionalSpec.html#a3.2.7_Java_Action

8.17 [EDP] Add a new job-types endpoint

https://blueprints.launchpad.net/sahara/+spec/edp-job-types-endpoint

Add a new job-types endpoint that can report all supported job types for a Sahara instance and which plugins support them.

8.17.1 Problem description

There are currently two problems around job types in Sahara that impact user experience:

- The current */jobs/config-hints/<job_type>* endpoint is not adequate for providing configuration hints because it does not take into account plugin type or the framework version. This endpoint was meant to give users a list of likely configuration values for a job that they may want to modify (at one point the UI incorporated the hints as well). Although some config is common across multiple plugins, hints need to be specific to the plugin and the framework version in order to be useful. The endpoint does not take a cluster or plugin argument and so hints must be general.
- A user currently has no indicator of the job types that are actually available from a Sahara instance (the UI lists them all). The set of valid job types is based on the plugins loaded for the current instance. Furthermore, not all job types will be available to run on all clusters launched by the user because they are plugin dependent.

These problems should be solved without breaking backward compatibility in the REST API.

8.17.2 Proposed change

Add a new endpoint that will indicate for the running Sahara instance which job types are supported by which versions of which plugins. Optionally, plugin-and-version-specific config hints will be included for each supported job type.

Because config hints can be very long, they will not be included in a response by default. A query string parameter will be used to indicate that they should be included.

The endpoint will support the following optional query strings for filtering. Each may be used more than once to query over a list of values, for example *type=Pig&type=Java*:

- **type** A job type to consider. Default is all job types.
- **plugin** A plugin to consider. Default is all plugins.
- **version** A plugin version to consider. Default is all versions.

The REST API method is specified in detail below under *REST API impact*.

We will need two new optional methods in the *Plugin SPI*. This information ultimately comes from the EDP engine(s) used by a plugin but we do not want to actually allocate an EDP engine object for this so the existing **get_edp_engine()** will not suffice (and besides, it requires a cluster object):

```
@abc.abstractmethod
def get_edp_job_types(self, versions=[]):
    return []

@abc.abstractmethod
def get_edp_config_hints(self, job_type, version):
    return {}
```

These specific methods are mentioned here because they represent a change to the public *Plugin SPI*.

Alternatives

Fix the existing */jobs/config-hints* endpoint to take a cluster id or a plugin-version pair and return appropriate config hints. However, this would break backward compatibility.

Still add an additional endpoint to retrieve the supported job types for the Sahara instance separate from config hints.

However, it makes more sense to deprecate the current config-hints interface and add the new endpoint which serves both purposes.

Data model impact

None

REST API impact

Backward compatibility will be maintained since this is a new endpoint.

GET /v1.1/{tenant_id}/job-types

Normal Response Code: 200 (OK)

Errors: none

Indicate which job types are supported by which versions of which plugins in the current instance.

Example

request

```
GET http://sahara/v1.1/775181/job-types
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
    "job_types": [
        {
            "name": "Hive",
            "plugins": [
                {
                    "description": "The Apache Vanilla plugin.",
                    "name": "vanilla",
                    "title": "Vanilla Apache Hadoop",
                    "versions": {
                        "1.2.1": {}
                    }
                },
                {
                    "description": "The Hortonworks Sahara plugin.",
                    "name": "hdp",
                    "title": "Hortonworks Data Platform",
                    "versions": {
                        "1.3.2": {},
                        "2.0.6": {}
                    }
                }
            ]
        },
        {
            "name": "Java",
            "plugins": [
                {
                    "description": "The Apache Vanilla plugin.",
                    "name": "vanilla",
                    "title": "Vanilla Apache Hadoop",
                    "versions": {
                        "1.2.1": {}
                    }
                },
                {
                    "description": "The Hortonworks Sahara plugin.",
                    "name": "hdp",
                    "title": "Hortonworks Data Platform",
                    "versions": {
                        "1.3.2": {},
                        "2.0.6": {}
                    }
                }
            ]
        },
        {
            "name": "MapReduce",
            "plugins": [
                {
```


(continues on next page)

(continued from previous page)

```
                    "description": "The Apache Vanilla plugin.",
                    "name": "vanilla",
                    "title": "Vanilla Apache Hadoop",
                    "versions": {
                        "1.2.1": {}
                    }
                },
                {
                    "description": "The Hortonworks Sahara plugin.",
                    "name": "hdp",
                    "title": "Hortonworks Data Platform",
                    "versions": {
                        "1.3.2": {},
                        "2.0.6": {}
                    }
                }
            ]
        },
        {
            "name": "MapReduce.Streaming",
            "plugins": [
                {
                    "description": "The Apache Vanilla plugin.",
                    "name": "vanilla",
                    "title": "Vanilla Apache Hadoop",
                    "versions": {
                        "1.2.1": {}
                    }
                },
                {
                    "description": "The Hortonworks Sahara plugin.",
                    "name": "hdp",
                    "title": "Hortonworks Data Platform",
                    "versions": {
                        "1.3.2": {},
                        "2.0.6": {}
                    }
                }
            ]
        },
        {
            "name": "Pig",
            "plugins": [
                {
                    "description": "The Apache Vanilla plugin.",
                    "name": "vanilla",
                    "title": "Vanilla Apache Hadoop",
                    "versions": {
                        "1.2.1": {}
```

(continues on next page)

(continued from previous page)

```
                    }
                },
                {
                    "description": "The Hortonworks Sahara plugin.",
                    "name": "hdp",
                    "title": "Hortonworks Data Platform",
                    "versions": {
                        "1.3.2": {},
                        "2.0.6": {}
                    }
                }
            ]
        }
    ]
}
```

The job-types endpoint returns a list. Each item in the list is a dictionary describing a job type that is supported by the running Sahara. Notice for example that the *Spark* job type is missing.

Each job type dictionary contains the name of the job type and a list of plugins that support it.

For each plugin, we include the basic identifying information and then a *versions* dictionary. Each entry in the versions dictionary has the name of the version as the key and the corresponding config hints as the value. Since this example did not request config hints, the dictionaries are empty.

Here is an example of a request that uses the plugin and version filters:

Example

request

```
GET http://sahara/v1.1/775181/job-types?plugin=hdp&version=2.0.6
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
    "job_types": [
        {
            "name": "Hive",
            "plugins": [
                {
                    "description": "The Hortonworks Sahara plugin.",
                    "name": "hdp",
                    "title": "Hortonworks Data Platform",
                    "versions": {
                        "2.0.6": {}
                    }
                }
            ]
        },
```

(continues on next page)

(continued from previous page)


```
        {
            "name": "Java",
            "plugins": [
                {
                    "description": "The Hortonworks Sahara plugin.",
                    "name": "hdp",
                    "title": "Hortonworks Data Platform",
                    "versions": {
                        "2.0.6": {}
                    }
                }
            ]
        },
        {
            "name": "MapReduce",
            "plugins": [
                {
                    "description": "The Hortonworks Sahara plugin.",
                    "name": "hdp",
                    "title": "Hortonworks Data Platform",
                    "versions": {
                        "2.0.6": {}
                    }
                }
            ]
        },
        {
            "name": "MapReduce.Streaming",
            "plugins": [
                {
                    "description": "The Hortonworks Sahara plugin.",
                    "name": "hdp",
                    "title": "Hortonworks Data Platform",
                    "versions": {
                        "2.0.6": {}
                    }
                }
            ]
        },
        {
            "name": "Pig",
            "plugins": [
                {
                    "description": "The Hortonworks Sahara plugin.",
                    "name": "hdp",
                    "title": "Hortonworks Data Platform",
                    "versions": {
                        "2.0.6": {}
                    }
```


(continues on next page)

(continued from previous page)

```
                }
            ]
        }
    ]
}
```

Here is another example that enables config hints and also filters by plugin, version, and job type.

Example

request

```
GET http://sahara/v1.1/775181/job-types?hints=true&plugin=hdp&version=1.3.
↪2&type=Hive
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
    "job_types": [
        {
            "name": "Hive",
            "plugins": [
                {
                    "description": "The Hortonworks Sahara plugin.",
                    "name": "hdp",
                    "title": "Hortonworks Data Platform",
                    "versions": {
                        "1.3.2": {
                            "job_config": {
                                "args": {},
                                "configs": [
                                    {
                                        "description": "Reduce tasks.",
                                        "name": "mapred.reduce.tasks",
                                        "value": "-1"
                                    }
                                ],
                                "params": {}
                            }
                        }
                    }
                }
            ]
        }
    ]
}
```

This is an abbreviated example that shows imaginary config hints.

Other end user impact

The python-saharaclient should be extended to support this as well:

```
$ sahara job-types-list [--type] [--plugin [--plugin-version]]
```

Output should look like this (not sure where else to specify this):

```
+--------------------+----------------------------------+
| name               | plugin(versions)                 |
+--------------------+----------------------------------+
| Hive               | vanilla(1.2.1), hdp(1.3.2, 2.0.6) |
| Java               | vanilla(1.2.1), hdp(1.3.2, 2.0.6) |
| MapReduce          | vanilla(1.2.1), hdp(1.3.2, 2.0.6) |
| MapReduce.Streaming | vanilla(1.2.1), hdp(1.3.2, 2.0.6) |
| Pig                | vanilla(1.2.1), hdp(1.3.2, 2.0.6) |
+--------------------+----------------------------------+
```

Since config hints can return so much information, and description fields for instance can contain so much text, how to support config hints through the python-saharaclient is TBD.

As noted above, the *Plugin SPI* will be extended with optional methods. Existing plugins that support EDP will be modified as part of this change.

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

The UI will be able to take advantage of this information and filter the job types available to the user on the forms. It will also be able to make use of config hints.

8.17.3 Implementation

Assignee(s)

Primary assignee:
tmckay

Other contributors:
none

Work Items

- Add basic endpoint support with optional methods in the plugin SPI
- **Implement the methods for each plugin that supports EDP**
  This can be done as a series of separate small CRs
- Add support to python-saharaclient
- Update documentation

8.17.4 Dependencies

None

8.17.5 Testing

- Unit tests
- Tempest tests for API

8.17.6 Documentation Impact

It should be added to the REST API doc.

8.17.7 References

8.18 Enable Spark jobs to access Swift URL

https://blueprints.launchpad.net/sahara/+spec/edp-spark-swift-integration

Spark uses Hadoop filesystem libraries to dereference input and output URLs. Consequently, Spark can access Swift filesystem URLs if the Hadoop Swift JAR is included in the image and a Spark jobs Hadoop configuration includes the necessary Swift credentials. This specification describes a method of transparently adding Swift credentials to the Hadoop configuration of a Spark job so that the job source code does not need to be altered and recompiled to access Swift URLs.

8.18.1 Problem description

Spark jobs may access Swift URLs assuming the cluster image includes the Hadoop Swift JAR. To do this, the jobs Hadoop configuration must include the necessary Swift credentials (*fs.swift.service.sahara.username* and *fs.swift.service.sahara.password*, for example).

As with Oozie Java actions, job source code may be modified and recompiled to add the necessary configuration values to the jobs Hadoop configuration. However, this means that a Spark job which runs successfully with HDFS input and output sources cannot be used "as is" with Swift input and output sources.

Sahara should allow users to run Spark jobs with Swift input and output sources without altering job source code.

8.18.2 Proposed change

This change follows the approach developed by Kazuki Oikawa in https://blueprints.launchpad.net/sahara/+spec/edp-improve-compatibility for Java compatibility.

A new configuration value will be added to Sahara, *edp.spark.adapt_for_swift*. If this configuration value is set to True on a Spark job, Sahara will run a wrapper class (SparkWrapper) instead of the original class indicated by the job. The default for this configuration value will be False.

Sahara will generate a *spark.xml* file containing the necessary Swift credentials as Hadoop configuration values. This XML file will be uploaded to the master node with the JAR containing the SparkWrapper class, along with the other files normally needed to execute a Spark job.

Saharas Spark launcher script will run the SparkWrapper class instead of the jobs designated main class. The launcher will pass the name of the XML configuration file to SparkWrapper at runtime, followed by the name of the original class and any job arguments. SparkWrapper will add this XML file to the default Hadoop resource list in the job's configuration before invoking the original class with any arguments.

When the jobs main class is run, its default Hadoop configuration will contain the specified Swift credentials.

The permissions of the job dir on the master node will be set to 0x700. This will prevent users other than the job dir owner from reading the Swift values from the configuration file.

The sources for the SparkWrapper class will be located in the sahara-extra repository under the *edp-adapt-for-spark* directory. This directory will contain a *pom.xml* so that the JAR may be built with maven. Maintenance should be light since the SparkWrapper class is so simple; it is not expected to change unless the Hadoop Configuration class itself changes.

Currently, the plan is to build the JAR as needed and release it with Sahara in *service/edp/resources/edp-spark-wrapper.jar*. Alternatively, the JAR could be hosted publically and added to Sahara images as an element.

Alternatives

There is no known way to supply Swift credentials to the Hadoop filesystem libraries currently other than through configuration values. Additionally, there is no way to add values to the Hadoop configuration transparently other than through configuration files.

This does present some security risk, but it is no greater than the risk already presented by Oozie jobs that include Swift credentials. In fact, this is probably safer since a user must have direct access to the job directory to read the credentials written by Sahara.

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

The DIB image for Spark will need to change to include the Hadoop Swift JAR. There is an existing element for this, this is trivial.

Additionally, we still need a solution to a compatibility issue with the *jackson-mapper-asl.jar*.

This can be patched by including an additional JAR on the cluster nodes, so we can conceivably bundle the extra jar with the Spark image as a patch. Alternatively, the issue might be resolved by upgrading the CDH version (or Spark assembly version) used in the image.

Sahara-dashboard / Horizon impact

The Spark job submission form should have an input (checkbox?) which allows a user to set *edp.spark.adapt_for_swift* to True. Default should be False.

We dont want to assume that just because Swift paths are passed as arguments to a Spark job that Sahara should run the wrapper. The job itself may handle the Swift paths in its own way.

8.18.3 Implementation

Assignee(s)

Primary assignee: tmckay

Other contributors: croberts

Work Items

- Add a simple SparkWrapper class. This is different than the wrapper class developed for Oozie Java actions.
- Update Spark image to include the Hadoop Openstack JAR
- Find a solution to the jackson issue
- Update the UI
- Implement handling of the *edp.spark.adapt_for_swift* option in Sahara. This includes generation and upload of the extra XML file, upload of the additional utility jar, and alteration of the command generated to invoke spark-submit
- Updated node configuration All nodes in the cluster should include the Hadoop *core-site.xml* with general Swift filesystem configuration. Additionally, spark-env.sh should point to the Hadoop *core-site.xml* so that Spark picks up the Swift configs and *spark-defaults.conf* needs to set up the executor classpath. These changes will allow a user to run Spark jobs with Swift paths manually using *spark-submit* from any node in the cluster should they so choose.

8.18.4 Dependencies

https://blueprints.launchpad.net/sahara/+spec/edp-improve-compatibility

8.18.5 Testing

Unit tests and integration tests for Spark jobs will identify any regressions introduced by this change.

Once we have Spark images with all necessary elements included, we can add an integration test for Spark with Swift URLs.

8.18.6 Documentation Impact

Any user docs describing job submission should be updated to cover the new option for Spark jobs.

8.18.7 References

8.19 Storage of recently logged events for clusters

https://blueprints.launchpad.net/sahara/+spec/event-log

This specification proposes to add event logs assigned to cluster.

8.19.1 Problem description

It will be more user friendly have event log assigned to cluster. In this case users will have the ability to see the steps performed to deploy a cluster. If there is an issue with their cluster, users will be able to see the reasons for the issues in the UI and won't be required to read the Sahara logs.

8.19.2 Proposed change

The new feature will provide the following ability:

- For each cluster there will be an event log assigned to it. The deployer will have the ability to see it in Horizon. In that case the user will have the ability to see all the steps for the cluster deploying.
- The deployer will have the ability to see the current progress of cluster provisioning.

Alternatives

None

Data model impact

This change will require to write event log messages to the database. It's good idea to store events messages in database in similar manner, in which we store cluster data, node groups data and so on.

Plugins should provide a list of provisioning steps and be able to report status of the current step. All steps should be performed in linear series, and we will store events only for the current step. All completed steps should have the duration time stored in the database. There are no reasons to store events for successfully completed steps, so they will be dropped periodically.

If an error occurs while provisioning cluster we will have error events saved for the current step. Also we should store for each error event error_id, which would help for admins to determine reasons of failures in sahara logs.

We should have new a database object called ClusterEvent, which will have the following fields:

- node_group_id;
- instance_id;
- instance_name;

- event_info;
- successful;
- provision_step_id;
- id;
- created at;
- updated at;

Also we should have a new database object called ClusterProvisionStep, which will have the following fields:

- id;
- cluster_id;
- step_name;
- step_type;
- completed;
- total;
- successful;
- started_at;
- completed_at;
- created_at;
- updated_at;

Fields `step_name` and `step_type` will contain detail info about step. `step_name` will contain description of the step, for example `Waiting for ssh` or `Launching instances`. `step_type` will contain info about related process of this step. For example, if we creating new cluster this field will contain `creating` and for scaling some cluster this field will contain `scaling`. So, possible values of this field will be `creating`, `scaling`, `deleting`. Also we should add ability to get main provisioning steps from Plugin SPI for each `step_type` as dictionary. For example, expected return value:

```
{
   "creating": [
       "Launching instances",
       "Waiting for ssh",
       ....
   ]
   "scaling": [
       ....
   ]
   "deleting": [
       ....
   ]
}
```

Cluster should have new field: * provisioning_progress

This field will contain list with provisioning steps, which should provide ability to get info about provisioning steps from cluster. We should update this list with new steps every time we start new process with cluster (creating/scaling/deleting). Provision steps should updated both from plugin and infra, because some steps are same for all clusters.

REST API impact

Existing GET request for a cluster should be updated with completed steps info, and short info for the current step. For example, we will have following response: "Launching instances completed 1000 out of 1000 in 10 minutes", "Trying ssh completed: 59 out of 1000". Also response should be sorted by increasing of value `created_at`.

```
{
   "cluster": {
       "status": "Waiting",
       ....
       "provisioning_progress": [
          {
            "id": "1",
            "cluster_id": "1111",
            "step_name": "Launching instances",
            "step_type": "creating",
            "completed": 1000,
            "total": 1000,
            "successful": "True",
            "created_at": "2013-10-09 12:37:19.295701",
            "started_at": 36000000,
            "completed_at": 18000000,
            "updated_at": "2013-10-09 12:37:19.295701",
          },
          {
            "id": "2",
            "cluster_id": "1111",
            "step_name": "Waiting for ssh",
            "step_type": "creating",
            "completed": 59,
            "total": 1000,
            "successful": None,
            "created_at": "2013-10-09 12:37:19.295701",
            "started_at": 18000000,
            "completed_at": None,
            "updated_at": "2013-10-09 12:37:19.295701",
          }
       ]
       ....
   }
}
```

In case of errors:

```
{
   "cluster": {
       "status": "Waiting",
       ....
       "provisioning_progress": [
          {
            "id": "1",
            "cluster_id": "1111",
            "step_name": "Launching instances",
            "step_type": "creating",
            "completed": 1000,
            "total": 1000,
            "successful": "True",
            "created_at": "2013-10-09 12:37:19.295701",
            "started_at": 36000000,
            "completed_at": 18000000,
            "updated_at": "2013-10-09 12:37:19.295701",
          },
          {
            "id": "2",
            "cluster_id": "1111",
            "step_name": "Waiting for ssh",
            "step_type": "creating",
            "completed": 59,
            "total": 1000,
            "successful": False,
            "created_at": "2013-10-09 12:37:19.295701",
            "started_at": 18000000,
            "completed_at": None,
            "updated_at": "2013-10-09 12:37:19.295701",
          }
       ]
       ....
   }
}
```

Also in these cases we will have events stored in database from which we can debug cluster problems. Because first steps of cluster provision are same, then for these steps infra should update `provisioning_progress` field. Also for all plugin-related steps plugin should update `provisioning_progress` field. So, new cluster field should be updated both from infra and plugin.

New endpoint should be added to get details of the current provisioning step: GET /v1.1/<tenant_id>/clusters/<cluster_id>/progress

The expected response should looks like:

```
{
    "events": [
          {
             'node_group_id': "ee258cbf-4589-484a-a814-81436c18beb3",
             'instance_id': "ss678cbf-4589-484a-a814-81436c18beb3",
```

(continues on next page)

(continued from previous page)

```
            'instance_name': "cluster-namenode-001",
            'provisioning_step_id': '1',
            'event_info': None,
            'successful': True,
            'id': "ss678cbf-4589-484a-a814-81436c18eeee",
            'created_at': "2014-10-29 12:36:59.329034",
        },
        {
            'cluster_id': "d2498cbf-4589-484a-a814-81436c18beb3",
            'node_group_id': "ee258www-4589-484a-a814-81436c18beb3",
            'instance_id': "ss678www-4589-484a-a814-81436c18beb3",
            'instance_name': "cluster-datanode-001",
            'provisioning_step_id': '1',
            'event_info': None,
            'successful': True,
            'id': "ss678cbf-4589-484a-a814-81436c18eeee",
            'created_at': "2014-10-29 12:36:59.329034",
        },
        {
            'cluster_id': "d2498cbf-4589-484a-a814-81436c18beb3",
            'node_group_id': "ee258www-4589-484a-a814-81436c18beb3",
            'instance_id': "ss678www-4589-484a-a814-81436c18beb3",
            'instance_name': "cluster-datanode-001",
            'provisioning_step_id': '2',
            'event_info': "Trying to access failed: reason in sahara logs
                           by id ss678www-4589-484a-a814-81436c18beb3",
            'successful': False,
            'id': "ss678cbf-4589-484a-a814-81436c18eeee",
            'created_at': "2014-10-29 12:36:59.329034",
        },
    ]
}
```

Event info for the failed step will contain the traceback of an error.

Other end user impact

None

Deployer impact

This change will takes immediate effect after it is merged. Also it is a good idea to have ability to disable event log from configuration.

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

This change will add section in Horizon at page with event logs /data_processing/clusters/cluster_id. At this page it will be possible to see main provisioning steps, and current progress of all of it. Also we would have an ability to see events of current provisioning step. In case of errors we will be able to see all events of the current step and main reasons of failures.

8.19.3 Implementation

Assignee(s)

Primary assignee:
vgridnev

Other contributors:
slukjanov, alazarev, nkonovalov

Work Items

This feature require following modifications:

- Add ability to get info about main steps of provisioning cluster from plugin;
- Add ability to view progress of current provisioning step;
- Add ability to specify events to current cluster and current step;
- Add periodic task to erase redundant events from previous step;
- Add ability to view events about current step of cluster provisioning;
- Sahara docs should be updated with some use cases of this feature;
- Saharaclient should be modified with new REST API feature;
- New cluster tab with events in Horizon should be implemented;
- Add unit test to test new features of events.

8.19.4 Dependencies

Depends on OpenStack requirements

8.19.5 Testing

As written in Work Items section this feature will require unit tests

8.19.6 Documentation Impact

As written in Work Items section this feature will require docs updating with some use cases of feature

8.19.7 References

8.20 Exceptions improvement

https://blueprints.launchpad.net/sahara/+spec/exceptions-improvement

This specification proposes to add identifiers for every raised Sahara exception, so that they can be easily found in logs.

8.20.1 Problem description

Now it's hard to find an error in logs, especially if there are a lot of errors of the same type which occurs when a lot of operations are executed simultaneously. They can produce a bunch of similar exceptions (error code doesn't help in this kind of situation).

It would be nice to have an opportunity to find exceptions by unique identifiers. This identifiers will be found in Horizon tab with events that will be implemented in this spec: https://review.openstack.org/#/c/119052/.

8.20.2 Proposed change

Support Features:

- Every error that has been raised during the workflow will have, besides of error message, uuid property, whereby error can be found in logs easily.

For example, NotFoundException will leave in logs:

```
NotFoundException: Error ID: 7a229eda-f630-4153-be03-d71d6467f2f4
Object 'object' is not found
```

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

8.20.3 Implementation

Assignee(s)

Primary assignee:
apavlov-n

Other contributors:
sreshetniak

Work Items

- Adding ability to generate unique identifiers for SaharaException class
- Change messages of Sahara exceptions so that all of them contain identifier.

8.20.4 Dependencies

None

8.20.5 Testing

None

8.20.6 Documentation Impact

None

8.20.7 References

None

8.21 Use first_run to One-step Start Cluster

https://blueprints.launchpad.net/sahara/+spec/first-run-api-usage

This specification proposes to use cm_api method first_run to start cluster for CDH plugin in Sahara, instead of current a batch of methods.

8.21.1 Problem description

Now in CDH plugin method start_cluster, a lot of methods defined in cloudera_utils.py is used to configure/prepare services to be started. Those methods include cm_api methods in their body, and check the return status. E.g., cu.format_namenode will call cm_api ApiService.format_hdfs.

However, this way is not preferred by Cloudera. The much easier way is using a single method first_run to most of those work. In fact, in Cloudera Manager first_run is also used to do the final step of deploying a cluster.

Changing current start_cluster codes into using first_run method will benefit by:

- Leave work up to Cloudera Manager to itself, instead of manually doing it.
- Simplify the work of adding more service support.
- Avoid possible errors generated by future CM changes.

8.21.2 Proposed change

The implementation will change start_cluster to call first_run, and remove the other part of work can be done by first_run from the method body.

For detail, it will be like following:

In deploy.py, possible start_cluster method may be like:

```
def start_cluster(cluster):
    cm_cluster = cu.get_cloudera_cluster(cluster)
    """ some pre codes """
    cu.first_run(cluster)
    """ some post codes """
```

Current methods used to configure CDH components, like _configure_spark, _install_extjs, and most part of start_cluster body can be removed.

In cloudera_utils.py, first_run can be defined as (just for example):

```
@cloudera_cmd
def first_run(cluster):
    cm_cluster = get_cloudera_cluster(cluster)
    yield cm_cluster.first_run()
```

Methods for configuring CDH components, like create_yarn_job_history_dir, create_oozie_db, install_oozie_sharelib, create_hive_metastore_db, create_hive_dirs can be removed.

Alternatives

Current way works at this stage, but it increases complexity of coding work to add more services support to CDH plugin. And, when CM is upgraded in the future, the correctness of current codes cannot be assured. At the end, the first_run method to start services is recommended by Cloudera.

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

It will be easier for developers to add more CDH services support.

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

8.21.3 Implementation

Assignee(s)

Primary assignee:
ken chen

Other contributors:
ken chen

Work Items

The work items will be:

- Change current deploy.py and cloudera_utils.py in the way above.
- Test and evaluate the change.

8.21.4 Dependencies

None

8.21.5 Testing

Take an integration test to create a cluster.

8.21.6 Documentation Impact

None

8.21.7 References

None

8.22 Enable HDFS NameNode High Availability with HDP 2.0.6 plugin

https://blueprints.launchpad.net/sahara/+spec/hdp-plugin-enable-hdfs-ha

Extend HDP 2.0.6 plugin to include the setup and configuration of the HDFS NameNode High Availability after creating, configuring and starting the cluster.

8.22.1 Problem description

Hadoop clusters are created with a single NameNode which represents a SPOF (Single Point Of Failure). If the NameNode becomes unavailable, the whole cluster becomes unavailable and we have to wait for the NameNode to come back up before using the cluster again. NameNode High Availability was introduced in Hadoop 2.0.0 and integrated into HDP as of the 2.0.0 release. The High Availability is achieved by having 2 NameNodes, an active NameNode and a stand by one. When the active fails the standby steps in and act as the cluster's NameNode. NameNode's High Availability can be configured manually on clusters deployed with HDP 2.0.6 plugin in Sahara through Ambari, but the process is long, tedious and error prone.

End users might not have the necessary skills required to setup the High Availability and deployers might prefer to deploy highly available clusters without manually configuring each one.

HDFS NameNode High Availability (Using Quorum Journal Manager (QJM)) uses Journal nodes to share HDFS edits between the active and the standby namenodes. The journal nodes ensure that the two namenodes have the same set of HDFS edits, but do not ensure the automatic failover. The automatic failover requires Zookeeper servers and Zookeeper Failover Controllers (ZKFC). A typical cluster with HDFS NameNode High Availability uses at least three (or a an odd number greater than three) journal nodes and a zookeeper cluster with three or five zookeeper servers. The Zookeeper Failover Controllers are installed on the servers acting as active and standby namenodes. The setup removes the secondary namenode (which is usually installed on a different server than the one hosting the namenode) and installs a second namenode process. The journal nodes and zookeeper servers can be installed on the same servers running the active and standby (old secondary namenode) namenodes. This leaves us with 2 journal nodes and 2 zookeeper servers. An additional server is all what's needed with journal node and zookeeper server to setup a minimally viable Hadoop cluster with HDFS NameNode High Availability. (For more info : http://hadoop.apache.org/docs/r2.3.0/hadoop-yarn/hadoop-yarn-site/HDFSHighAvailabilityWithQJM.html)

8.22.2 Proposed change

The 'Create Node Group Template' wizard will introduce a new process 'JOURNALNODE' to the list of available processes for HDP 2.0.6 plugin. Other processes necessary for HDFS HA are either already included in the list (NAMENODE, ZOOKEEPER_SERVER and SECONDARY_NAMENODE) or will be automatically setup (Zookeeper Failover Controllers).

The 'Launch Cluster' wizard for HDP 2.0.6 plugin will include a checkbox 'Enable HDFS HA'. This option will default to False and will be added to the cluster object.

The verification code will verify the necessary requirements for a Hadoop cluster and a set of additional requirements in the case where 'Enable HDFS HA' is set to True. The requirements include : NAMENODE and SECONDARY_NAMENODE on different servers At least three journal nodes on different servers At least three zookeeper servers on different servers

Upon successful validation the cluster will be created and once it's in the 'Active' state Availability' and if 'Enable HDFS HA' is True the service will instruct the plugin to start configuring the NameNode High Availability. The cluster will be set in 'Configuring HDFS HA' state and the plugin will start the configuration procedure. The procedure starts by the plugin stopping all the services and executing some preparation commands on the server with the namenode process (through the hadoopserver objects). Then the plugin installs and starts the journal nodes using Ambari REST API (POST, PUT, WAIT ASYNC). Next the configuration is updated using Ambari REST API (PUT), other services including Hive, Oozie and Hue might require configuration update if they are installed. Finally some more remote commands are executed on the namenodes and the Zookeeper Failover Controllers are installed and started and the SECONDARY_NAMENODE process is deleted. The plugin will return and the cluster will be set back in the 'Active' state.

Alternatives

Manual setup through Ambari web interface

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

Developers of subsequent versions of the HDP plugin should take into account this option and the added functionality. The procedure is not likely to change in newer versions of HDP as it uses Ambari's API which stays intact with newer versions.

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

A checkbox 'Enable HDFS HA' in the 'Launch Cluster' wizard when the user chooses HDP 2.0.6 plugin.

8.22.3 Implementation

Assignee(s)

Primary assignee:
abbass-marouni

Work Items

- Add a new attribute to cluster-level configs to indicate whether HA is enabled or not.
- Add new service classes to HDP 2.0.6 for Journal nodes and Zookeeper Failover Controllers
- Add new remote methods to hdp hadoopserver.py for remote commands
- Add new methods to generate new configurations according to cluster configuration

8.22.4 Dependencies

None

8.22.5 Testing

Unit Test service classes Unit Test new cluster specs Integration Test cluster creation with HA Integration Test cluster creation without HA

8.22.6 Documentation Impact

Update documentation to reflect new changes and to explain new options.

8.22.7 References

None

8.23 Indirect access to VMs

https://blueprints.launchpad.net/sahara/+spec/indirect-vm-access

This blueprint proposes one more way for Sahara to manage VMs. Management could be done via VM that works as proxy node. In this case access from controller should be given for one VM only.

8.23.1 Problem description

Currently there are several ways to give Sahara access to VMs:

1. flat private network
   - not secure
   - doesn't work with neutron
2. floating IPs
   - all nodes need to have floating IPs
     - floating IPs are limited resource
     - floating IPs are usually for external world, not for access from controller
   - all nodes need to be accessible from controller nodes
     - it is more complicated in HA mode
   - access to data nodes should be secure
3. net_ns
   - hard to configure
   - can be inappropriate
   - doesn't work in HA mode
5. tenant-specific proxy node (https://review.openstack.org/#/c/131142/)
   - proxy setting is for the whole system (template based)
   - proxy can't be configured for a specific cluster
   - proxy node needs to be spawned and configured manually
4. agents
   - not implemented yet
   - require external message queue accessible from VMs and controllers

- require maintenance of agents

So, there can be cases when none of listed approaches work.

8.23.2 Proposed change

This blueprint proposes one more way to access VMs by Sahara. Sahara will use one of spawned VMs as proxy node and gain access to all other nodes through it. Access to VM that has proxy node role could be gained using any of methods above.

Sahara will understand which node to use as a proxy node by "is_proxy" field of nodegroup. If this nodegroup contains several instances the first one will be used as proxy (this leaves space for load balancing).

So, proposed workflow:

1. Nodegoup object is extended with "is_proxy" field, horizon is changed accordingly.
2. User selects "is_proxy" checkbox for one of node groups (manager, master or separate one). If proxy is used for separate node group it could be with really small flavor.
3. Sahara spawns all infrastructure
4. Sahara communicates with all instances via node with proxy role. Internal IPs are used for communication. This removes restriction that all nodes must have floating IP in case if floating network used for management. In case with proxy node this restriction will be applied to proxy node only.

Pros:

1. we need external access to only one VM.
2. data nodes could be isolated

Cons:

1. Indirect access could be slower.
2. Loss of proxy means loss of access to entire cluster. Intellectual selection is possible, but not planned for the first implementation.

Implementation will extend global proxy implemented at https://review.openstack.org/#/c/131142/. For indirect access Paramiko-based analogy of "ssh proxy nc host port" command will be used. Paramiko implementation will allow to use private keys from memory.

Note, proxy command can still be used to access proxy instance.

Implementation details

Sahara uses two ways of access to instances:

1. SSH
2. HTTP

SSH access

For ssh access one more layer of ssh will be added. Old code:

```
_ssh.connect(host, username=username, pkey=private_key, sock=proxy)
```

New code:

```
_proxy_ssh = paramiko.SSHClient()
_proxy_ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())

_proxy_ssh.connect(proxy_host, username=proxy_username,
                   pkey=proxy_private_key, sock=proxy)

chan = _proxy_ssh.get_transport().open_session()
chan.exec_command("nc {0} {1}".format(host, SSH_PORT))

_ssh.connect(host, username=username, pkey=private_key, sock=chan)
```

HTTP access

Http access will be implemented in a similar way with ProxiedHTTPAdapter. SshProxySocket class will be implemented that corresponds to netcat socket running on remote host.

Note, if proxycommand present, it will be passed to paramiko directly without involving NetcatSocket class.

Alternatives

This blueprint offers one more way to access VMs. All existing ways will remain unchanged.

Data model impact

None

REST API impact

New boolean field "is_proxy" in nodegroup and nodegroup template objects.

Other end user impact

None

Deployer impact

One more deployment option to consider.

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

Checkbox in nodegroup template edit form.

8.23.3 Implementation

Assignee(s)

Primary Assignee:

Andrew Lazarev (alazarev)

Work Items

- Sahara core changes
- Python client changes
- Horizon changes
- Doc changes

8.23.4 Dependencies

- Global proxy implementation (https://review.openstack.org/#/c/131142/)

8.23.5 Testing

Manually

8.23.6 Documentation Impact

The feature needs to be documented.

8.23.7 References

None

8.24 Plugin for Sahara with MapR

https://blueprints.launchpad.net/sahara/+spec/mapr-plugin

https://blueprints.launchpad.net/sahara/+spec/mapr-image-elements

This specification proposes to add MapR plugin with MapR Distribution of Hadoop in Sahara.

8.24.1 Problem description

The MapR Distribution for Apache Hadoop provides organizations with an enterprise-grade distributed data platform to reliably store and process big data. MapR packages a broad set of Apache open source ecosystem projects enabling batch, interactive, or real-time applications. The data platform and the projects are all tied together through an advanced management console to monitor and manage the entire system.

MapR is one of the largest distributions for Hadoop supporting more than 20 open source projects. MapR also supports multiple versions of the various individual projects thereby allowing users to migrate to the latest versions at their own pace. The table below shows all of the projects actively supported in the current GA version of MapR Distribution for Hadoop as well as in the next Beta release.[1]

8.24.2 Proposed change

MapR plugin implementation will support Hadoop 0.20.2 and Hadoop 2.4.1. Plugin will support key Sahara features:

- Cinder integration
- Cluster scaling/decommission
- EDP
- Cluster topology validation
- Integration with Swift

Plugin will be able to install following services:

- MapR-FS
- YARN

- Oozie (two versions are supported)
- HBase
- Hive (two versions are supported)
- Pig
- Mahout
- Webserver

MapR plugin will support the following OS: Ubuntu 14.04 and CentOS 6.5.

MapR plugin will support the following node types:

- Nodes Running ZooKeeper and CLDB
- Nodes for Data Storage and Processing
- Edge Nodes

In a production MapR cluster, some nodes are typically dedicated to cluster coordination and management, and other nodes are tasked with data storage and processing duties. An edge node provides user access to the cluster, concentrating open user privileges on a single host.

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

MapR plugin uses specific pre-installed images with MapR local repository files.

Sahara-dashboard / Horizon impact

None

8.24.3 Implementation

Assignee(s)

Primary assignee:
aosadchiy

Other contributors:
ssvinarchuck

Work Items

- Add implementation of plugin for bare images.

8.24.4 Dependencies

Depends on OpenStack requirements.

8.24.5 Testing

- Add unit tests to Sahara to cover basic functionality of plugin
- Add integration tests to Sahara

8.24.6 Documentation Impact

MapR plugin documentation should be added to the plugin section of Sahara docs.

8.24.7 References

- [1] https://www.mapr.com/products/apache-hadoop
- [2] http://doc.mapr.com/display/MapR/Home
- [3] https://www.mapr.com/
- [4] https://github.com/mapr

8.25 Refactor MapR plugin code

https://blueprints.launchpad.net/sahara/+spec/mapr-refactor

MapR plugin's code should be refactored to support easy addition of new services and releases

8.25.1 Problem description

Current plugin implementation has several weaknesses: * Declarative nature of Service * Code complexity caused by usage of plugin-spec.json * Almost all actions are implemented as sequence of utility functions calls * Not clear separation of behaviour to modules and classes * Some code is redundant

8.25.2 Proposed change

- Extract Service entity to separate class with customizable behaviour
- Move provisioning logic from utility modules to specialized classes
- Remove redundant code

MapR plugin implementation delegates all operations to it's counterparts in VersionHandler interface. VersionHandler interface mimics plugin SPI with additional methods get_context(cluster) and get_services(). ClusterContext object returned by get_context wraps cluster object passed as argument and provides additional information about cluster as well as utility methods related to wrapped cluster.

Service definitions resides in 'sahara.plugins.mapr.services' package instead of plugin-spec.json which is completely removed now. Each service definition represents particular version of service.

Alternatives

Leave code "as-is"

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

Developers of subsequent versions of the MapR plugin should take into account this changes.

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

8.25.3 Implementation

Assignee(s)

Primary assignee:
aosadchiy

Other contributors:
ssvinarchuk

Work Items

1) Extract Service entity to separate class with customizable behaviour
2) Move provisioning logic from utility modules to specialized classes
3) Remove redundant code

8.25.4 Dependencies

None

8.25.5 Testing

Existing integration tests are sufficient

8.25.6 Documentation Impact

None

8.25.7 References

None

8.26 Clean up clusters that are in non-final state for a long time

https://blueprints.launchpad.net/sahara/+spec/periodic-cleanup

This spec is to introduce periodic task to clean up old clusters in non-final state.

8.26.1 Problem description

For now it is possible that sahara cluster becomes stuck because of different reasons (e.g. if sahara service was restarted during provisioning or neutron failed to assign floating IP). This could lead to clusters holding resources for a long time. This could happen in different tenants and it is hard to check such conditions manually.

Related bug: https://bugs.launchpad.net/sahara/+bug/1185909

8.26.2 Proposed change

Add "cleanup_time_for_nonfinal_clusters" parameter in "periodic" section of configuration.

Based on this configuration periodic task will search clusters that are in non-final state and weren't updated for a given time.

Term "non-final" includes all cluster states except "Active" and "Error".

"cleanup_time_for_nonfinal_clusters" parameter will be in hours. Non-positive value will indicate that clean up option is disabled.

Default value will be 0 to keep backward compatibility (users don't expect that after upgrade all their non-final cluster will be deleted).

'updated_at' column of 'clusters' column will be used to determine last change. This is not 100% accurate, but good enough. This field is changed each time cluster status is changed.

Alternatives

Add such functionality to external service (e.g. Blazar).

Data model impact

None.

REST API impact

None.

Other end user impact

None.

Deployer impact

None.

Developer impact

None.

Sahara-image-elements impact

None.

Sahara-dashboard / Horizon impact

None.

8.26.3 Implementation

Assignee(s)

Primary assignee:
alazarev (Andrew Lazarev)

Other contributors:
None

Work Items

- Implement feature
- Document feature

8.26.4 Dependencies

None.

8.26.5 Testing

Manually.

8.26.6 Documentation Impact

Need to be documented.

8.26.7 References

- https://bugs.launchpad.net/sahara/+bug/1185909

8.27 Support multi-worker Sahara API deployment

https://blueprints.launchpad.net/sahara/+spec/sahara-api-workers

Add support of multi-worker deployment of Sahara API.

8.27.1 Problem description

Currently Sahara API uses one thread with wsgi application. This means that API can service only one request at a time. Some requests (e.g. cluster scale) require synchronous requests to Sahara engine (using message queue). This means that Sahara API will not be able to service other requests until full round trip finished.

Also, multi-threaded solution gives much more options for performance tuning. It could allow to utilize more server CPU power.

Most of OpenStack services support several workers for API.

8.27.2 Proposed change

The ideal solution for that would be migration to Pecan and WSME (https://etherpad.openstack.org/havana-common-wsgi) with multi-threading support. Although this will require a lot of work and there is no much pressure to do that.

This spec suggests simple solution of specific problem without much refactoring of existing code.

So, the solution is: 1. Leave current wsgi implementation 2. Leave current socket handling 3. Run wsgi server in several threads/processes 4. Implement only children processes management, leave all existing code as is.

Children processes management will include:

1. Handling of children processes, restart of dead processes
2. Proper signals handling (see https://bugs.launchpad.net/sahara/+bug/1276694)
3. Graceful shutdown
4. Support of debug mode (with green threads instead of real threads)

Things that will NOT be included: 1. Config reload / API restart

Alternatives

Migrate to Pecan and WSME first.

Implementation details

Most OpenStack services use deprecated oslo wsgi module. It has tight connections with oslo services module.

So, there are three options here:

1. Use deprecated oslo wsgi module. (Bad, since module is deprecated)
2. Use oslo services module, but write all wsgi stuff ourselves (or copy-paste from other project).
3. Write minimum code to make server start multi-worker (e.g. see how it is implemented in Heat).

I propose going with the option 3. There is no much sense spending resources for code, that will be replaced anyway (we will definitely migrate to Pecan some day).

Data model impact

None.

REST API impact

None.

Other end user impact

None.

Deployer impact

One more configuration parameter.

Developer impact

None.

Sahara-image-elements impact

None.

Sahara-dashboard / Horizon impact

None.

8.27.3 Implementation

Assignee(s)

Primary assignee:
alazarev (Andrew Lazarev)

Other contributors:
None

Work Items

- Implement feature
- Document feature

8.27.4 Dependencies

None

8.27.5 Testing

Manually. Probably CI could be changed to run different tests in different modes.

8.27.6 Documentation Impact

Need to be documented.

8.27.7 References

None.

8.28 New style logging

https://blueprints.launchpad.net/sahara/+spec/new-style-logging

Rewrite Sahara logging in unified OpenStack style proposed by https://blueprints.launchpad.net/nova/+spec/log-guidelines

8.28.1 Problem description

Now log levels and messages in Sahara are mixed and don't match the OpenStack logging guidelines.

8.28.2 Proposed change

The good way to unify our log system would be to follow the major guidelines. Here is a brief description of log levels:

- Debug: Shows everything and is likely not suitable for normal production operation due to the sheer size of logs generated (e.g. scripts executions, process execution, etc.).
- Info: Usually indicates successful service start/stop, versions and such non-error related data. This should include largely positive units of work that are accomplished (e.g. service setup, cluster start, successful job execution).
- Warning: Indicates that there might be a systemic issue; potential predictive failure notice (e.g. job execution failed).
- Error: An error has occurred and an administrator should research the event (e.g. cluster failed to start, plugin violations of operation).
- Critical: An error has occurred and the system might be unstable, anything that eliminates part of Sahara's intended functionality; immediately get administrator assistance (e.g. failed to access keystone/database, plugin load failed).

Here are examples of LOG levels depending on cluster execution:

- Script execution:

```
LOG.debug("running configure.sh script")
```

- Cluster startup:

```
LOG.info(_LI("Hadoop stack installed successfully."))
```

- Job failed to execute:

```
LOG.warning(_LW("Can't run job execution {job} (reason: {reason}")).format(
            job = job_execution_id, reason = ex)
```

- HDFS can't be configured:

```
LOG.error(_LE('Configuring HDFS HA failed. {reason}')).format(
          reason = result.text)
```

- Cluster failed to start:

```
LOG.error(_LE('Install command failed. {reason}')).format(
          reason = result.text)
```

Additional step for our logging system should be usage of pep3101 as unified format for all our logging messages. As soon as we try to make our code more readable please use {<smthg>} instead of {0} in log messages.

Alternatives

We need to follow OpenStack guidelines, but if needed we can move plugin logs to DEBUG level instead of INFO. It should be discussed separately in each case.

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

8.28.3 Implementation

Assignee(s)

Primary assignee:
starodubcevna

Work Items

- Unify existing logging system
- Unify logging messages
- Add additional logs if needed

8.28.4 Dependencies

None

8.28.5 Testing

None

8.28.6 Documentation Impact

None

8.28.7 References

https://blueprints.launchpad.net/nova/+spec/log-guidelines https://www.python.org/dev/peps/pep-3101/

8.29 HTTPS support for sahara-api

https://blueprints.launchpad.net/sahara/+spec/sahara-support-https

Most OpenStack services support running server supporting HTTPS connections. Sahara must support such way too.

8.29.1 Problem description

There are two common ways to enable HTTPS for the OpenStack service:

1. TLS proxy. Proxy communicates with user via HTTPS and redirects all requests to service via unsecured HTTP. Keystone is configured to point on HTTPS port. Internal port is usually closed for outside using firewall. No additional work to support HTTPS required on service side.
2. Native support. Service can be configured to expect HTTPS connections. In this case service handles all security aspects by itself.

Most OpenStack services support both types of enabling SSL. Sahara currently can be secured only using TLS proxy.

8.29.2 Proposed change

Add ability to Sahara API to listen on HTTPS port.

Currently there is no unified way for OpenStack services to work with HTTPS. Process of unification started with sslutils module in oslo-incubator. Sahara could use this module to be on the same page with other services.

Alternatives

Copy-paste SSL-related code from other OpenStack project.

Data model impact

None

REST API impact

None

Other end user impact

- python-saharaclient should support SSL-related options

Deployer impact

One more option to consider.

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

Add SSL-related parameters to pass to python client.

8.29.3 Implementation

Assignee(s)

Primary assignee:
alazarev (Andrew Lazarev)

Other contributors:
None

Work Items

- Implement feature in Sahara
  - Import sslutils
  - Configure WSGI server to HTTPS
- Add support to python-saharaclient
- Add support to devstack
- Add documentation

8.29.4 Dependencies

- sslutils module from oslo-incubator

8.29.5 Testing

Devstack doesn't have HTTPS testing for now. It looks like manual testing is the only option.

8.29.6 Documentation Impact

Need to be documented.

8.29.7 References

None

8.30 Scenario integration tests for Sahara

https://blueprints.launchpad.net/sahara/+spec/scenario-integration-tests

For now the Sahara project has not functional integration tests. We need to create new integration tests that will be more flexible.

8.30.1 Problem description

Current integration tests allow to test only limited number of test scenarios and cluster configurations that are hardcoded in code of tests. In many cases we should test various configurations of Sahara clusters but current integration tests don't have this functionality. Also we have many copy-pasted code in test files and this code requires large work for its refactoring.

8.30.2 Proposed change

It is proposed to create new integration tests that will be more flexible. Test scenarios will be defined in YAML files and it is supposed this approach will provide more flexibility in testing. The usual scenario will have the following look:

```
credentials:
    os_username: dev-user
    os_password: swordfish
    os_tenant: devs
    os_auth_url: http://os_host:5000/v2.0
    sahara_url: http://sahara_host:8336/v1.1  # optional

network:
    type: neutron  # or nova-network
    private_network: private  # for neutron
    auto_assignment_floating_ip: false  # for nova-network
    public_network: public  # or floating_ip_pool for nova-network

clusters:
    - plugin_name: vanilla
      plugin_version: 2.6.0
      image: some_id
      node_group_templates:  # optional
        - name: master
          node_processes:
            - namenode
            - resourcemanager
          flavor_id: '3'
        - name: worker
          node_processes:
            - datanode
            - nodemanager
          flavor_id: '3'
      cluster_template:  # optional
          name: vanilla
          node_group_templates:
              master: 1
              worker: 3
      scenario:  # optional
          - run_jobs
          - scale
          - run_jobs

    - plugin_name: hdp
      plugin_version: 2.0.6
      image: some_id
```

Minimal scenario will look the following way:

```
clusters:
```

(continues on next page)

(continued from previous page)

```
    - plugin_name: vanilla
      plugin_version: 2.6.0
      image: some_id
```

Full scenario will look the following way:

```
concurrency: 3

credentials:
    os_username: dev-user
    os_password: swordfish
    os_tenant: devs
    os_auth_url: http://os_host:5000/v2.0
    sahara_url: http://sahara_host:8336/v1.1  # optional

network:
    type: neutron  # or nova-network
    private_network: private  # for neutron
    auto_assignment_floating_ip: false  # for nova-network
    public_network: public  # or floating_ip_pool for nova-network

clusters:
      # required
    - plugin_name: vanilla
      # required
      plugin_version: 2.6.0
      # required (id or name)
      image: some_id
      node_group_templates:  # optional
        - name: master
          node_processes:
            - namenode
            - resourcemanager
          flavor_id: '3'
          description: >-
              Some description
          volumes_per_node: 2
          volumes_size: 2
          node_configs:
              HDFS:
                  dfs.datanode.du.reserved: 10
          security_groups: ~
          auto_security_group: true
          availability_zone: nova
          volumes_availability_zone: nova
          volume_type: lvm
        - name: worker
          node_processes:
            - datanode
            - nodemanager
```

(continues on next page)

(continued from previous page)

```
        flavor_id: 3
    cluster_template:  # optional
        name: vanilla
        description: >-
            Some description
        cluster_configs:
            HDFS:
                dfs.replication: 1
        node_group_templates:
            master: 1
            worker: 3
        anti_affinity: true
    cluster:
        name: test-cluster
        is_transient: true
        description: >-
            Cluster description
    scaling:
        - operation: resize
          node_group: worker
          size: 4
        - operation: add
          node_group: worker
          size: 2
    scenario:  # optional
        - run_jobs
        - scale
        - run_jobs
    edp_jobs_flow: example
    retain_resource: true  # optional

edp_jobs_flow:
    example:
      - type: Pig
        main_lib:
            source: swift
            path: path_to_pig_script.pig
        input_datasource:
            type: swift
            source: etc/edp-examples/edp-pig/top-todoers/data/input
        output_datasource:
            type: hdfs
            destination: /user/hadoop/edp-output
        configs:
            dfs.replication: 1
      - type: Java
        additional_libs:
            - type: database
              source: |
```

(continues on next page)

(continued from previous page)

```
                etc/edp-examples/.../hadoop-mapreduce-examples-2.4.1.jar
        configs:
            edp.java.main_class: |
                org.apache.hadoop.examples.QuasiMonteCarlo
        args:
            - 10
            - 10
```

After we described test scenario in YAML file we run test as usual. The python test code will be generated from these YAML files.

We will use the Mako library to generate the python code. The generated code will look like:

```
from sahara.tests.scenario import base


class vanilla2_4_1TestCase(base.BaseTestCase):
    @classmethod
    def setUpClass(cls):
        super(vanilla2_4_1TestCase, cls).setUpClass()
        cls.credentials = {
            'os_username': 'dev-user',
            'os_password': 'swordfish',
            'os_tenant': 'devs',
            'os_auth_url': 'http://172.18.168.5:5000/v2.0',
            'sahara_url': None
        }
        cls.network = {
            'type': 'neutron',
            'public_network': 'net04_ext',
            'auto_assignment_floating_ip': False,
            'private_network': 'dev-network'
        }
        cls.testcase = {
            'image': 'sahara-juno-vanilla-2.4.1-ubuntu-14.04',
            'plugin_name': 'vanilla',
            'retain_resources': False,
            'class_name': 'vanilla2_4_1',
            'edp_jobs_flow': [
                {
                    'configs': {
                        'dfs.replication': 1
                    },
                    'output_datasource': {
                        'type': 'hdfs',
                        'destination': '/user/hadoop/edp-output'
                    },
                    'input_datasource': {
                        'type': 'swift',
                        'source':
```

(continues on next page)

(continued from previous page)

```
                        'etc/edp-examples/edp-pig/top-todoers/data/input'
                    },
                    'main_lib': {
                        'type': 'swift',
                        'source':
                        'etc/edp-examples/edp-pig/top-todoers/example.pig'
                    },
                    'type': 'Pig'
                },
                {
                    'type': 'Java',
                    'args': [10, 10],
                    'additional_libs': [
                        {
                            'type': 'database',
                            'source':
                            'etc/edp-examples/hadoop2/edp-java/'
                            'hadoop-mapreduce-examples-2.4.1.jar'
                        }
                    ],
                    'configs': {
                        'edp.java.main_class':
                        'org.apache.hadoop.examples.QuasiMonteCarlo'
                    }
                }
            ],
            'scenario': ['run_jobs', 'scale', 'run_jobs'],
            'plugin_version': '2.4.1'
        }

    def test_plugin(self):
        self.create_cluster()
        self.check_run_jobs()
        self.check_scale()
        self.check_run_jobs()


class hdp2_0_6TestCase(base.BaseTestCase):
    @classmethod
    def setUpClass(cls):
        super(hdp2_0_6TestCase, cls).setUpClass()
        cls.credentials = {
            'os_username': 'dev-user',
            'os_password': 'swordfish',
            'os_tenant': 'devs',
            'os_auth_url': 'http://172.18.168.5:5000/v2.0',
            'sahara_url': None
        }
        cls.network = {
```

(continues on next page)

(continued from previous page)

```
            'type': 'neutron',
            'public_network': 'net04_ext',
            'auto_assignment_floating_ip': False,
            'private_network': 'dev-network'
        }
        cls.testcase = {
            'image': 'f3c4a228-9ba4-41f1-b100-a0587689d4dd',
            'plugin_name': 'hdp',
            'retain_resources': False,
            'class_name': 'hdp2_0_6',
            'edp_jobs_flow': None,
            'scenario': ['run_jobs', 'scale', 'run_jobs'],
            'scaling': [
                {
                    'operation': 'resize',
                    'size': 5,
                    'node_group': 'worker'
                }
            ],
            'plugin_version': '2.0.6'
        }

    def test_plugin(self):
        self.create_cluster()
        self.check_run_jobs()
        self.check_scale()
        self.check_run_jobs()
```

Mako template will look the following way:

```
from sahara.tests.scenario import base

% for testcase in testcases:
    ${make_testcase(testcase)}
% endfor

<%def name="make_testcase(testcase)">
class ${testcase['class_name']}TestCase(base.BaseTestCase):
    @classmethod
    def setUpClass(cls):
        super(${testcase['class_name']}TestCase, cls).setUpClass()
        cls.credentials = ${credentials}
        cls.network = ${network}
        cls.testcase = ${testcase}

    def test_plugin(self):
        self.create_cluster()
    % for check in testcase['scenario']:
        self.check_${check}()
    % endfor
```

(continues on next page)

(continued from previous page)

```
</%def>
```

By default concurrency will be equal to number of cpu cores. This value can be changed in YAML file.

We are going to use new integration tests for CI as soon as they are completely implemented.

Alternatives

We can use current integration tests for further testing Sahara but they don't have sufficient coverage of Sahara use cases.

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

Developers will be able much better to test their changes in Sahara.

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

8.30.3 Implementation

Assignee(s)

Primary assignee:
sreshetniak

Other contributors:
ylobankov slukjanov

Work Items

The work items will be the following:

- Add python code to sahara/tests
- Add examples of test scenarios
- Add documentation for new integration tests

8.30.4 Dependencies

Mako tempest-lib

8.30.5 Testing

None

8.30.6 Documentation Impact

We need to add a note about new tests in Sahara documentation.

8.30.7 References

None

8.31 Creation of Security Guidelines Documentation

https://blueprints.launchpad.net/sahara/+spec/security-guidelines-doc

As Sahara increases in functionality and complexity, the need for clear, concise documentation grows. This is especially true for security related topics as they are currently under-represented. This specification proposes the creation of a document that will provide a source for security related topics, guides, and instructions as they pertain to Sahara.

8.31.1 Problem description

There is currently a distinct lack of centralized, security related, documentation for Sahara. Several features have been implemented to address security shortfalls and they could use broadened discussions of their application and proper usage. Additionally there is no current documentation which discusses the specific procedures for securing the individual plugin technologies at use within Sahara.

8.31.2 Proposed change

This specification proposes the creation of Sahara specific documentation addressing the security concerns of users and operators. This documentation will cover current features, best practices, and guidelines for installing and operating Sahara.

The documentation generated by this effort will be included in the OpenStack Security Guide[1]. Bug patches will be generated against the current OpenStack manuals, and the OpenStack Security Group will be engaged with respect to finalizing and including the documentation.

The process will be broken down into sub-chapter sections that will make up a Sahara chapter for the OpenStack Security Guide. Initially these sub-chapters will include; Sahara controller installs, current feature discussions, and plugin specific topics.

The information provided is intended to be updated as new methodologies, plugins, and features are implemented. It will also be open to patching through the standard OpenStack workflows by the community at large.

Alternatives

Creation of a separate document managed by the Sahara team outside the purview of the OpenStack manuals, and distributed through the Sahara documentation. This solution would be non-ideal as it creates an alternate path for OpenStack manuals that is outside the expected locations for end users.

Creation of a separate document as above with the exception that it will be maintained with the other OpenStack manuals. This option might be more plausible than the previous, but it still suffers from the problem of creating an alternate location for security related guidelines that is separate from the official manual. It also bears the burden of creating a new project within the manuals infrastructure.

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

8.31.3 Implementation

Assignee(s)

Primary assignee:
mimccune (Michael McCune)

Other contributors:
None

Work Items

- Create a bug against the OpenStack manuals for Sahara chapter inclusion
- Create the documentation and change requests for the following sub-chapters:
  - Sahara controller install guides, with security related focus
  - Feature discussions and examples
  - plugin specific topics
    * Hadoop
    * Spark
    * Storm

8.31.4 Dependencies

None

8.31.5 Testing

Format testing as provided by the security guide project.

8.31.6 Documentation Impact

This specification proposes the creation of hypertext and PDF documentation.

8.31.7 References

[1]: http://docs.openstack.org/security-guide/content/ch_preface.html

8.32 Spark Temporary Job Data Retention and Cleanup

https://blueprints.launchpad.net/sahara/+spec/spark-cleanup

Creates a configurable cron job at cluster configuration time to clean up data from Spark jobs, in order to ease maintenance of long-lived clusters.

8.32.1 Problem description

The current Spark plugin stores data from any job run in the /tmp directory, without an expiration policy. While this is acceptable for short-lived clusters, it increases maintenance on long-running clusters, which are likely to run out of space in time. A mechanism to automatically clear space is needed.

8.32.2 Proposed change

On the creation of any new Spark cluster, a script (from sahara.plugins.spark/resources) will be templated with the following variables (which will be defined in Spark's config_helper module and thus defined per cluster):

- Minimum Cleanup Seconds
- Maximum Cleanup Seconds
- Minimum Cleanup Megabytes

That script will then be pushed to /etc/hadoop/tmp_cleanup.sh. In the following cases, no script will be pushed:

1) Maximum Cleanup Seconds is 0 (or less)
2) Minimum Cleanup Seconds and Minimum Cleanup Megabytes are both 0 (or less)

Also at cluster configuration time, a cron job will be created to run this script once per hour.

This script will iterate over each extant job directory on the cluster; if it finds one older than Maximum Cleanup Seconds, it will delete that directory. It will then check the size of the set of remaining directories. If there is more data than Minimum Cleanup Megabytes, then it will delete directories older than Minimum Cleanup Seconds, starting with the oldest, until the remaining data is smaller than Minimum Cleanup Megabytes or no sufficiently aged directories remain.

Alternatives

Any number of more complex schemes could be developed to address this problem, including per-job retention information, data priority assignment (to effectively create a priority queue for deletion,) and others. The above plan, however, while it does allow for individual cluster types to have individual retention policies, does not demand excessive maintenance or interface with that policy after cluster creation, which will likely be appropriate for most users. A complex retention and archival strategy exceeds the intended scope of this convenience feature, and could easily become an entire project.

Data model impact

None; all new data will be stored as cluster configuration.

REST API impact

None; operative Spark cluster template configuration parameters will be documented the current interface allows this change.

Other end user impact

None.

Deployer impact

None.

Developer impact

None.

Sahara-image-elements impact

None.

Sahara-dashboard / Horizon impact

None. (config_helper.py variables will be automatically represented in Horizon.)

8.32.3 Implementation

Assignee(s)

Primary assignee:
egafford

Work Items

- Creation of periodic job
- Creation of deletion script
- Testing

8.32.4 Dependencies

None

8.32.5 Testing

Because this feature is entirely Sahara-internal, and requires only a remote shell connection to the Spark cluster (without which many, many other tests would fail) I believe that Tempest tests of this feature are unnecessary. Unit tests should be sufficient to cover this feature.

8.32.6 Documentation Impact

The variables used to set retention policy will need to be documented.

8.32.7 References

https://blueprints.launchpad.net/sahara/+spec/spark-cleanup

8.33 Specification Repository Backlog Refactor

https://blueprints.launchpad.net/sahara/+spec/add-backlog-to-specs

This specification proposes the refactoring of the *sahara-specs* repository to inform a new workflow for backlogged features. This new workflow will increase the visibility of specifications that have been approved but which require implementation.

8.33.1 Problem description

Currently the *sahara-specs* repository suggests a layout that creates "implemented" and "approved" sub-directories for each release. This pattern could create duplicates and/or empty directories as plans that have been approved for previous releases but have not been completed are moved and copied around as necessary.

Additionally, this pattern can increase the barrier for new contributors looking for features to work on and the future direction of the project. By having a single location for all backlogged specifications it will increase visibility of items which require attention. And the release directories can be devoted entirely to specifications implemented during those cycles.

8.33.2 Proposed change

This change will add a backlog directory to the specs directory in the repository, and refactor the juno directory to contain only the specs implemented in juno.

It will also update the documentation to clearly indicate the usage of the backlog directory and the status of the release directories. This documentation will be part of the root readme file, a new document in the backlog directory, and a reference in the main documentation.

This change is also proposing a workflow that goes along with the repository changes. Namely that any work within a release that is either predicted to not be staffed, or that is not started at the end of a release should be moved to the backlog directory. This process should be directed by the specification drafters as they will most likely be the primary assignees for new work. In situations where the drafter of a specification feels that there will be insufficient resources to create an implementation then they should move an approved specification to the backlog directory. This process should also be revisited at the end of a release cycle to move all specifications that have not been assigned to the backlog.

Alternatives

Keep the directory structure the way it is currently. This does not improve the status of the repository but is an option to consider. If the directory structure is continued as currently configured then the release directories will need to create additional structures for each cycle's incomplete work.

Refactor each release directory to contain a backlog. This is very similar to leaving things in their current state with the exception that it changes the names in the directories. This change might add a small amount of clarification but it is unlikely as the current names for "implemented" and "approved" are relatively self explanatory.

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

The main impact will be for crafters of specifications to be more aware of the resource requirements to implement the proposed changes. And potentially move their specifications based on those requirements.

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

8.33.3 Implementation

Assignee(s)

Primary assignee:
mimccune (Michael McCune)

Work Items

- Create the backlog directory and documentation.
- Clean up the juno directory.
- Add references to backlog in the contributing documentation.

8.33.4 Dependencies

None

8.33.5 Testing

None

8.33.6 Documentation Impact

The "How to Participate" document should be updated to make reference to the backlog directory as a place to look for new work.

8.33.7 References

Keystone is an example of another project using this format for backlogged work. Examples of the documentation for the backlog directory[1] and the root documentation[2] will be used as reference in the creation of Sahara specific versions.

[1]: https://github.com/openstack/keystone-specs/tree/master/specs/backlog [2]: https://github.com/openstack/keystone-specs

8.34 Storm Integration

https://blueprints.launchpad.net/sahara/+spec/storm-integration

This blueprint aims to implement Storm as a processing option in Sahara. Storm is a real time processing framework that is highly used in the big data processing community.

8.34.1 Problem description

Sahara today is only able to process big data in batch. Storm provides a easy setup stream processing. Having storm integrated with Sahara gives sahara a new feature, so not only users will be able to process batch but also real time data.

8.34.2 Proposed change

- The implementation is divided in three steps:
  - First we need to implement Storm Plugin.
    - Identify Storm configuration files and manage their creation via sahara;
    - Create the plugin itself to manage Storm deploy;
  - Second, we need to create a new Job Manager for storm, following Trevor McKay's refactoring
  - Last we will create a new image that comes with storm installed.
- Node Groups:

- Storm has two basic components **Nimbus** and **Supervisor**, Nimbus is the master node, hosts the UI and is the main node of the topology. Supervisors are the worker nodes. Other than that, storm needs only zookeeper to run, we need have it as well. The basic Node Groups that we will have are:
  - Nimbus;
  - Supervisor;
  - Nimbus and Supervisor;
  - Zookeeper;
  - Nimbus and Zookeeper;
  - Supervisor and Zookeeper;
  - Nimbus, Supervisor and Zookeeper

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

Sahara-image-elements may be the place to create storm image, it will need deeper investigation but looks like the best place to create and publish storm images.

Sahara-dashboard / Horizon impact

The first changes needed here is to show the configuration options of Storm when this type of job is selected. There will be also necessary to have a deploy job where the user can submit the topology jar and the command line to run it. As for now, I don't see other changes in the UI, the jobs are very simple and need no special configurations.

8.34.3 Implementation

Assignee(s)

Primary assignee:

- tellesmvn

Work Items

The implementation is divided in three steps:

- First we need to implement Storm Plugin.
  - Storm plugin is similar to Spark plugin. The implementation will be based on Spark's but necessary changes will be made.
  - Storm doesn't rely on many configuration files, there is only one needed and it is used by all nodes. This configuration file is written in YAML and it should be dynamically written in the plugin since it needs to have the name or ip of the master node and also zookeeper node(s). We will need PYYAML to parse this configuration to YAML. PYYAML is already a global requirement of OpenStack and will be added to Sahara's requirement as well.
  - The plugin will run the following processes:
    * Storm Nimbus;
    * Storm Supervisor;
    * Zookeeper.
- Second, we need to create a new Job Manager for storm, following Trevor McKay's refactoring
  - This implementation is under review and the details can be seen here: https://review.openstack.org/#/c/100678/
- Last we will create a new image that comes with storm installed.
  - This part is yet not the final decision, but it seems to me that it is better to have a prepared image with storm than having to install it every time a new cluster is set up.

8.34.4 Dependencies

- https://review.openstack.org/#/c/100622/

8.34.5 Testing

I will write Unit Tests, basically to test the write configuration file and more tests can be added as needed.

8.34.6 Documentation Impact

We will need to add Storm as a new plugin in the documentation and write how to use the plugin. Also a example in sahara-extra on how to run a storm job will be provided

8.34.7 References

- *Wiki <https://wiki.openstack.org/wiki/HierarchicalMultitenancy>*
- *Etherpad <https://etherpad.openstack.org/p/juno-summit-sahara-edp>*
- *Storm Documentation <http://storm.incubator.apache.org/>*

8.35 Support Cinder API version 2

https://blueprints.launchpad.net/sahara/+spec/support-cinder-api-v2

This specification proposes to add support for the second version of the Cinder API, which brings useful improvements and will soon replace version one.

8.35.1 Problem description

Currently Sahara uses only version 1 of Cinder API to create volumes. Version two, however, brings useful features such as scheduler hints, more consistent responses, caching, filtering, etc.

Also, Cinder is deprecating version 1 in favor of 2, so supporting both would make switching easier for users.

Use cases:

- As a developer I want to be able to pass scheduler hints to Cinder when creating clusters, in order to choose volumes more precisely and achieve improved performance.
- As a developer I want filtering into my requests to Cinder to make queries lighter.
- As a deployer I want to be able to choose between legacy Cinder API v1 and newer v2 API.

8.35.2 Proposed change

The implementation will add a configuration option, cinder_api_version, that will be defaulted to:

```
cinder_api_version=1
```

but can be changed to

```
cinder_api_version=2
```

by modifying sahara.conf.

The client() method in sahara/utils/openstack/cinder.py will either return clientv1() or clientv2() depending on the configuration.

Alternatives

Wait for Cinder API v1 to be deprecated and switch abruptly to v2.

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

- If the deployer wants to keep using Cinder API version 1, nothing has to be done.
- If the deployer wants to upgrade to version 2, the cinder_api_version option in sahara.conf should be overwritten.

Developer impact

Developers can read CONF.cinder_api_version to know what API version is being used.

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

8.35.3 Implementation

Assignee(s)

Primary assignee:
adrien-verge

Work Items

- Add a configuration option: cinder_api_version.
- Put some magic in sahara/utils/openstack/cinder.py to pick the correct Cinder client depending on the configuration.

8.35.4 Dependencies

None

8.35.5 Testing

Same as for v1 API.

8.35.6 Documentation Impact

None

8.35.7 References

- https://wiki.openstack.org/wiki/CinderAPIv2

8.36 Support Cinder availability zones

https://blueprints.launchpad.net/sahara/+spec/support-cinder-availability-zones

Extend API to support specifying Cinder availability zones where to create volumes.

8.36.1 Problem description

It can be desirable to assign Cinder availability zones to node groups, in order to have fine-grained cluster volumes topologies.

Use case:

- As a end user I want namenode volumes to be spawned in the regular AZ and datanode volumes in a high-performance `ssd-high-io` AZ.

8.36.2 Proposed change

It adds a new `volumes_availability_zone` property in NodeGroup and NodeGroupTemplate objects. When set, it modifies the direct and Heat engines to force creating of volumes into the right AZ.

Alternatives

None

Data model impact

This change will add `volumes_availability_zone` columns in sahara database, next to `volumes_per_node` and `volumes_size`. Impacted tables are `node_groups`, `node_group_templates` and `templates_relations`.

A database migration will accompany this change.

REST API impact

Each API method which deals with node groups and node groups templates will have an additional (and optional) `volumes_availability_zone` parameter, which will be taken into account if `volumes_per_node` is set and non-zero.

Example:

```
{
  "name": "cluster1",
  "node_groups": [
    {
      "name": "master",
      "count": 1,
    },
    {
      "name": "worker",
```

(continues on next page)

(continued from previous page)

```
      "count": 3,
      "volumes_per_node": 1,
      "volumes_size": 100,
      "volumes_availability_zone": "ssd-high-io"
    }
  ]
}
```

Other end user impact

python-saharaclient should be modified to integrate this new feature.

Deployer impact

Needs to migrate DB version using:

```
sahara-db-manage --config-file /etc/sahara/sahara.conf upgrade head
```

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

sahara-dashboard should be modified to integrate this new feature.

8.36.3 Implementation

Assignee(s)

Primary assignee:
adrien-verge

Work Items

- Add a `volumes_availability_zone` property in NodeGroup and NodeGroupTemplate objects.
- When a user specifies the (optional) volumes_availability zone, check its existence.
- In the direct engine, include the `volumes_availability_zone` argument in the call to cinder.client().volumes.create().
- In the Heat engine, add the `availability_zone` property in sahara/resources/volume.heat.

8.36.4 Dependencies

None

8.36.5 Testing

- Test cluster creation with `volumes_availability_zone` specified.
- Test cluster creation with wrong `volumes_availability_zone` specified.

8.36.6 Documentation Impact

Documentation will need to be updated, at sections related to node group template creation and cluster creation.

8.36.7 References

None

8.37 Support Nova availability zones

https://blueprints.launchpad.net/sahara/+spec/support-nova-availability-zones

Extend API to support specifying Nova availability zones where to spawn instances.

8.37.1 Problem description

It can be desirable to assign Nova availability zones to node groups, in order to have fine-grained clusters topologies.

Use cases:

- As a end user I want namenode instances to be spawned in the regular `nova` AZ and datanode instances in my high-performance `nova-highperf` AZ.
- As a end user I want instances from node-group A to be all together in the `nova-1` AZ, separated from instances from node-group B in `nova-2`.

8.37.2 Proposed change

The proposed change is already implemented at [1].

It adds an `availability_zone` property in NodeGroup and NodeGroupTemplate objects. When set, it modifies the direct and Heat engines to force spawning of instances into the right AZ.

Alternatives

None

Data model impact

This change will add `availability_zone` columns in the sahara database (`node_groups`, `node_group_templates` and `templates_relations` tables).

A database migration will accompany this change.

REST API impact

Each API method which deals with node groups and node groups templates will have an additional (and optional) `availability_zone` parameter.

Other end user impact

python-saharaclient should be modified to integrate this new feature.

Deployer impact

Needs to migrate DB version using:

```
sahara-db-manage --config-file /etc/sahara/sahara.conf upgrade head
```

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

sahara-dashboard should be modified to integrate this new feature.

8.37.3 Implementation

Assignee(s)

Primary assignee:
adrien-verge

Work Items

The proposed change is already implemented at [1].

- Add an `availability_zone` property in NodeGroup and NodeGroupTemplate objects.
- When a user specifies the (optional) availability zone, check its existence.
- In the direct engine, include the `availability_zone` argument in the call to nova.client().servers.create().
- In the Heat engine, add the `availability_zone` property in sahara/resources/instance.heat.

8.37.4 Dependencies

None

8.37.5 Testing

- Test cluster creation without availability zone specified.
- Test cluster creation with availability zone specified.
- Test cluster creation with wrong availability zone specified.

8.37.6 Documentation Impact

Documentation will need to be updated, at sections related to node group template creation and cluster creation.

8.37.7 References

- [1] https://review.openstack.org/#/c/120096

8.38 Spec - Add support for filtering results

Blueprints:

Sahara service: https://blueprints.launchpad.net/sahara/+spec/enable-result-filtering

Client library: https://blueprints.launchpad.net/python-saharaclient/+spec/enable-result-filtering

Horizon UI: https://blueprints.launchpad.net/horizon/+spec/data-processing-table-filtering

As result sets can be very large when dealing with nontrivial openstack environments, it is desirable to be able to filter those result sets by some set of field constraints. This spec lays out how Sahara should support such queries. Changes will be required in the API, client library, and in the Horizon UI.

8.38.1 Problem description

The original use case for this change came up when we wanted to provide a way to filter some of the tables in the UI. In order to make that filtering possible, either the UI has to perform the filtering (inefficient since large result sets would still be returned by the api) or the api/client library need to provide support for such filtering. Upon review of other api/client libraries, it looks like most, if not all, provide filtering capabilities. Sahara should also make filtering possible.

8.38.2 Proposed change

In order to make filtering possible, each of the "list()" methods in the client library should be extended to take an optional (default=None) parameter, "search_opts". "search_opts" should be a dict that will include one or more {field:query} entries. Pagination can also be handled through search_opts by allowing for the setting of page/page_size/max results.

In addition, the REST api needs to be extended for the list() operations to support query syntax. This will require changes to how Sahara currently does its database queries to include query options. Currently, they are set to just return all entries for each call.

Alternatives

Filtering could be done entirely in the UI, but that is a wasteful way to do it since each library/api call will still return the full result set. Also, any users of the REST api or the client library would not benefit from this functionality.

It would also be possible to just implement filtering at the client library level, but for the same reasons that doing it in the UI is undesirable, this approach is also suboptimal.

Data model impact

No changes to the model itself should be required. Only changes to how we query the database are needed.

REST API impact

The rest api methods for the list operations will need to be changed to support query-style parameters. The underlaying database access methods will need to be changed from the basic "get all" functionality to support a ***kwargs* parameter (this is already done in cluster_get_all(), but needs to be done for the other methods).

There are a pair of special cases to be considered for filtering on the Job Executions table. The "job" and "cluster" columns actually contain information that are not part of the job execution object. For filters on those fields, a field value of "job.name" or "cluster.name" should be passed down. That will trigger the database query to be joined to a filter on the name property of either the job or cluster table.

Additionally, there is special handling required to search on the Job Executions status field. This is because status is not a proper field in the Job Execution itself, but rather it is part of the "info" field. To handle this, a bit of manual filtering will be done against the info.status field.

Other end user impact

Users of the REST API, client library and Horizon UI will be able to filter their result sets via simple queries.

Deployer impact

These changes should not affect deployment of the Sahara service.

Developer impact

No developer impact.

Sahara-image-elements impact

No sahara-image-elements impact.

Sahara-dashboard / Horizon impact

Once the REST API and client library changes are in place, the Horizon UI will be modified to allow for filtering on the following tables: Clusters, Cluster Templates, Node Group Templates, Job Executions, Jobs. It would also be possible to filter any of our tables in the future without any other API/client library changes.

8.38.3 Implementation

Assignee(s)

Primary assignee:
croberts (Horizon UI)

Other contributors:
tmckay (REST API/sahara.db.api) croberts (Client Library)

Work Items

These may be able to be worked on in parallel, but they must be released in the order they are given below.

1. REST API implementation
2. Client library implementation
3. Horizon UI implementation

8.38.4 Dependencies

None

8.38.5 Testing

There should be tests added against the REST interface, client library and Horizon UI to test the filtering capabilities being added.

8.38.6 Documentation Impact

Docs for the REST API will need to be updated to reflect the querying functionality.

Docs for the client library will need to be updated to reflect the querying functionality.

The dashboard user guide should be updated to note the table filtering functionality that will be added.

8.38.7 References

Sahara service: https://blueprints.launchpad.net/sahara/+spec/enable-result-filtering

Client library: https://blueprints.launchpad.net/python-saharaclient/+spec/enable-result-filtering

Horizon UI: https://blueprints.launchpad.net/horizon/+spec/data-processing-table-filtering

8.39 Spec - Add support for editing templates

Blueprints:

Sahara Service: https://blueprints.launchpad.net/sahara/+spec/support-template-editing

Horizon UI: https://blueprints.launchpad.net/horizon/+spec/data-processing-edit-templates

Currently, once a node group template or a cluster template has been created, the only operations available are "copy" or "delete". That has unfortunate consequences on a user's workflow when they want to change a template. For example, in order to make even a small change to a node group template that is used by a cluster template, the user must make a copy of the node group template, change it, save it, and then create a new (or copy) cluster template that uses the new node group template. Ideally, a user could just edit the template in place and move on.

8.39.1 Problem description

Sahara currently lacks implementation for the update operation for both node group and cluster templates. In order to provide an implementation for these operations, the following issues must be addressed.

1. What to do with existing clusters that have been built by using a template that is being edited. Would we automatically scale a cluster if a cluster template was changed in a way that added/removed nodes? Would we shut down or start up processes if a node group template changed to add/remove processes?

I propose that for this iteration, a user may not change any templates that were used to build a currently running cluster. This is consistent with the rule currently in place to not allow deletion of templates that were are used by a currently active cluster. A future iteration could remove that restriction for both delete and edit. A user could still make a copy of a template and edit that if they wanted to start a new cluster with the altered version of the template.

2. Make sure that all cluster templates that are dependant upon an edited node group template will pick-up the changes to the node group template.

This is only a problem if we go the route of allowing templates used by an active cluster to be edited. In that case, we would need to change the existing templates to reference the newly created template.

8.39.2 Proposed change

The put (update) method will be implemented for both node group and cluster templates. Currently, the stubs are there, but they just return "Not Implemented".

I think that the simplest method of sending an updated template is to send the entire new template rather than just a diff. It could be slightly inefficient to do it this way, but avoids the need to build-in the diff logic in the UI. The current client library methods for update() seem to be anticipating that sort of implementation. If we went to sending a diff, we may need to adjust the client library.

Alternatives

Updates could be achieved by doing a delete/add combination, but that is not a very good solution to this problem since any given node group or cluster template could be referenced by another object. Deleting and adding would require updating each referencing object.

If we need to be able to edit templates that are used by an active cluster, the edited version of the template will retain the ID and name of the original template. Prior to overwriting the original template, it will be saved with a new ID and some form of "dotted" name to indicate the version ("origname.1"). All running clusters would be changed to reference the original template with the new ID.

Data model impact

N/A

REST API impact

N/A. The update operation are already defined in the API, but they are not yet implemented.

Other end user impact

The Horizon UI will be updated to include "Edit" as an operation for each node group and cluster template. The edit button will appear in the list of operations in each row of the table.

The python-saharaclient library already defines the update() methods that will be used for implementing this feature in the Horizon UI. No changes to python-saharaclient are anticipated to support this feature.

Deployer impact

N/A

Developer impact

N/A

Sahara-image-elements impact

N/A

Sahara-dashboard / Horizon impact

Horizon will be updated to include an "edit" button in each row of both the node group and cluster templates tables. That edit button will bring up the edit form (essentially the "create" form, but with all the values pre-populated from the existing template). Clicking on "Save" from the edit form will result in a call to the node group/cluster template update() method in the python-saharaclient library.

8.39.3 Implementation

Assignee(s)

Primary assignee:
croberts

Other contributors:
tmckay

Work Items

Horizon UI: croberts Sahara service: tmckay

8.39.4 Dependencies

N/A

8.39.5 Testing

There will be unit and integration tests added in Sahara.

Horizon will have a test added to the node group and cluster template panels to verify that the appropriate forms are generated when edit is chosen.

8.39.6 Documentation Impact

The Sahara UI user guide should be updated to note the availability of edit functionality for templates.

8.39.7 References

N/A

8.40 Cinder volume instance locality functionality

https://blueprints.launchpad.net/sahara/+spec/volume-instance-locality

This specification proposes to add an opportunity to have an instance and an attached volumes on the same physical host.

8.40.1 Problem description

Currently there is no way to specify that volumes are attached to the same physical host as the instance. It would be nice to have this opportunity.

8.40.2 Proposed change

This feature could be done with Cinder InstanceLocalityFilter which allows to request creation of volumes local to instance. It would increase performance of I/O operations.

There will be several changes:

- Boolean field `volume_local_to_instance` will be added to every node group template, node group and templates relation. This field will be optional and `False` by default.
- If `volume_local_to_instance` is set to `True`, Cinder volumes will be created on the same host.
- If `volume_local_to_instance` is `True`, all instances of the node group should be created on hosts with free disk space >= `volumes_per_node` * `volumes_size`. If it cannot be done, error should be occurred.

Alternatives

None

Data model impact

`volume_local_to_instance` field should be added to `node_groups`, `node_group_templates`, `templates_relations`.

REST API impact

- API will be extended to support `volume_local_to_instance` option. `volume_local_to_instance` is optional argument with `False` value by default, so this change will be backward compatible.
- python client will be updated

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

New field to select `volume_local_to_instance` option during node group template creation will be added.

8.40.3 Implementation

Assignee(s)

Primary assignee:
apavlov-n

Work Items

- Adding ability to create instance and volumes on the same host;
- Adding ability to create instance on appropriate host;
- Updating documentation;
- Updating UI;
- Updating python client;
- Adding unit tests.

8.40.4 Dependencies

None

8.40.5 Testing

Unit tests will be added.

8.40.6 Documentation Impact

Documentation will be updated. Will be documented when this feature can be used and when it cannot. Also will be noted how to enable it on Cinder side.

8.40.7 References

- http://docs.openstack.org/developer/cinder/api/cinder.scheduler.filters.instance_locality_filter.html

CHAPTER

NINE

JUNO SPECS

9.1 Make anti affinity working via server groups

https://blueprints.launchpad.net/sahara/+spec/anti-affinity-via-server-groups

Server groups is an openstack way to implement anti affinity. Anti affinity was implemented in Sahara before server groups were introduced in nova. Now it is time to replace custom solution with the common one.

9.1.1 Problem description

Direct engine uses manual scheduler hints for anti affinity.

Heat engine has limited anti affinity support and also uses scheduler hints (https://bugs.launchpad.net/sahara/+bug/1268610).

Nova has generic mechanism for this purpose.

9.1.2 Proposed change

Proposed solution is to switch both engines to implementation that uses server groups.

Current server group implementation has limitation that each server could belong to a single server group only. We can handle this constraint by having one server group per cluster. In this case each instance with affected processes will be included to this server group. So, with such implementation there will be no several affected instances on the same host even if they don't have common processes. Such implementation is fully compliant with all documentation we have about anti affinity.

We need to keep backward compatibility for direct engine. Users should be able to scale clusters deployed on Icehouse release. Sahara should update already spawned VMs accordingly. Proposed solution - Sahara should check server group existence during scaling and update whole cluster if server group is missing.

We don't care about backward compatibility for heat engine. It was in beta state for Icehouse and there are other changes that break it.

Alternatives

We can implement anti affinity via server groups in heat engine only. We will stop support of direct engine somewhere in the future. So, we can freeze current behavior in direct engine and don't change it until it is deprecated and removed.

Data model impact

None

REST API impact

None

Other end user impact

Anti affinity behavior changes in case of several involved processes. Before the change there was possible situation when several instances with affected (but different) processes are spawned on the same host. After the change all instances with affected processes will be scheduled to different hosts.

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

9.1.3 Implementation

Assignee(s)

Primary assignee:
alazarev

Work Items

Implement change

9.1.4 Dependencies

None

9.1.5 Testing

Will be covered by current integration tests.

9.1.6 Documentation Impact

Need note in upgrade notes about anti affinity behavior change.

9.1.7 References

- Team meeting minutes: http://eavesdrop.openstack.org/meetings/sahara/2014/sahara.2014-08-21-18.02.html

9.2 Append to a remote existing file

https://blueprints.launchpad.net/sahara/+spec/append-to-remote-file

Sahara utils remote can only create a new file and write to it or replace a line for a new one, but it can't append to an existing file. This bp aims to implement this feature.

9.2.1 Problem description

When managing remote files, sahara can only create new files and replace lines from existing one. The feature to append to an existing file doesn't exist and it is necessary.

9.2.2 Proposed change

Implement this feature following the idea of the write_to_file method The code is basically the same, the change will be the method of opening the file. Write uses 'w' we need to use 'a'.

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

9.2.3 Implementation

Assignee(s)

Primary assignee:

- tellesmvn

Work Items

The implementation is very basic, the idea is similar to the write_file, the necessary change is to open the remote file in append mode.

9.2.4 Dependencies

None

9.2.5 Testing

None for now.

9.2.6 Documentation Impact

None

9.2.7 References

None

9.3 Plugin for CDH with Cloudera Manager

https://blueprints.launchpad.net/sahara/+spec/cdh-plugin

This specification proposes to add CDH plugin with Cloudera Distribution of Hadoop and Cloudera Manager in Sahara.

9.3.1 Problem description

Cloudera is open-source Apache Hadoop distribution, CDH (Cloudera Distribution Including Apache Hadoop). CDH contains the main, core elements of Hadoop that provide reliable, scalable distributed data processing of large data sets (chiefly MapReduce and HDFS), as well as other enterprise-oriented components that provide security, high availability, and integration with hardware and other software. Cloudera Manager is the industry's first end-to-end management application for Apache Hadoop. Cloudera Manager provides many useful features for monitoring the health and performance of the components of your cluster (hosts, service daemons) as well as the performance and resource demands of the user jobs running on your cluster. [1]

9.3.2 Proposed change

CDH plugin implementation will support Cloudera Manager version 5 and CDH version 5.

Plugin will support key Sahara features:

- Cinder integration
- Cluster scaling
- EDP
- Cluster topology validation
- Integration with Swift
- Data locality

Plugin will be able to install following services:

- Cloudera Manager
- HDFS
- YARN
- Oozie

CDH plugin will support the following OS: Ubuntu 12.04 and CentOS 6.5. CDH provisioning plugin will support mirrors with CDH and CM packages.

By default CDH doesn't support Hadoop Swift library. Integration with Swift should be added to CDH plugin. CDH maven repository contains Hadoop Swift library. [2]

CDH plugin will support the following processes:

- MANAGER - Cloudera Manager, master process
- NAMENODE - HDFS NameNode, master process
- SECONDARYNAMENODE - HDFS SecondaryNameNode, master process
- RESOURCEMANAGER - YARN ResourceManager, master process
- JOBHISTORY - YARN JobHistoryServer, master process
- OOZIE - Oozie server, master process
- DATANODE - HDFS DataNode, worker process
- NODEMANAGER - YARN NodeManager, worker process

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

CDH plugin must be support vanilla images and images with Cloudera packages. For building pre-installed images with Cloudera packages use specific CDH elements.

Sahara-dashboard / Horizon impact

None

9.3.3 Implementation

Assignee(s)

Primary assignee:
sreshetniak

Other contributors:
iberezovskiy

Work Items

- Add implementation of plugin
- Add jobs in Sahara-ci
- Add integration tests
- Add elements to Sahara-image-elements for building images with pre-installed Cloudera packages

9.3.4 Dependencies

Depends on OpenStack requirements, needs a *cm_api* python library version 6.0.2, which is not present in OS requirements. [3] Need to add *cm_api* to OS requirements. [4]

9.3.5 Testing

- Add unit tests to Sahara to cover basic functionality of plugin
- Add integration tests to Sahara

9.3.6 Documentation Impact

CDH plugin documentation should be added to the plugin section of Sahara docs.

9.3.7 References

- [1] http://www.cloudera.com/content/cloudera/en/products-and-services/cdh.html
- [2] https://repository.cloudera.com/artifactory/repo/org/apache/hadoop/hadoop-openstack/
- [3] https://pypi.python.org/pypi/cm-api
- [4] https://review.openstack.org/#/c/106011/

9.4 Specification for integration Sahara with Ceilometer

https://blueprints.launchpad.net/sahara/+spec/ceilometer-integration

It is impossible to send notifications from Sahara to Ceilometer now. Sahara should have an ability to send notifications about cluster modifications, for example about creating/updating/destoroying cluster.

9.4.1 Problem description

New feature will provide the following ability:

- Sending notifications to Ceilometer about cluster modifications, which can help for user to achive some information about clusters which he have: number of active clusters in each moment and etc.

9.4.2 Proposed change

Change will consist of following modifications:

- Adding to Sahara an ability to send notifications.
- Adding to Sahara sending notificitions in such places, where cluster is modified
- Adding to Ceilometer an ability to pull notifications from Sahara exchange and to parse it.

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

9.4.3 Implementation

Assignee(s)

Primary assignee: vgridnev

Other contributors: slukjanov

Work Items

- Add notification sender in Sahara
- Add Ceilometer parser
- Add unit tests

9.4.4 Dependencies

Depends on OpenStack requirements

9.4.5 Testing

There will be:

- unit tests in Sahara
- unit tests in Ceilometer

9.4.6 Documentation Impact

Need modifications in Ceilometer documentation here:

- [1] http://docs.openstack.org/developer/ceilometer/measurements.html
- [2] http://docs.openstack.org/developer/ceilometer/install/manual.html

9.4.7 References

- [1] https://review.openstack.org/#/c/108982/

9.5 Store Sahara configuration in cluster properties

https://blueprints.launchpad.net/sahara/+spec/cluster-persist-sahara-configuration

It is important to know Sahara version and configuration for proper cluster migration and preventing dangerous operations.

9.5.1 Problem description

Now Sahara has no way to know conditions when cluster was created. Cluster operations after Openstack upgrade or switching infrastructure engine could be dangerous and cause data loss.

9.5.2 Proposed change

Store main Sahara properties (version, type and version of infrastructure engine, etc) to cluster DB object. This will allow to prevent dangerous operations and notify user gracefully.

Alternatives

Don't store any information about Sahara settings. Always assume that settings didn't change and we have current or previous release of OpenStack.

Data model impact

New field in cluster object. Probably this should be dictionary with defined keys.

REST API impact

We can expose information about Sahara settings to user. Or not.

Other end user impact

Some error messages could become more descriptive.

Deployer impact

None

Developer impact

More options to handle errors.

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

9.5.3 Implementation

Assignee(s)

Primary assignee:
alazarev

Work Items

Implement change

9.5.4 Dependencies

None

9.5.5 Testing

Manual

9.5.6 Documentation Impact

None

9.5.7 References

None

9.6 Security groups management in Sahara

https://blueprints.launchpad.net/sahara/+spec/cluster-secgroups

It is not acceptable for production use to require default security group with all ports open. Sahara need more flexible way to work with security groups.

9.6.1 Problem description

Now Sahara doesn't manage security groups and use default security group for instances provisioning.

9.6.2 Proposed change

Solution will consist of several parts:

1) Allow user to specify list of security groups for each of node groups.
2) Add support of automatic security group creation. Sahara knows everything to create security group with required ports open. In the first iteration this will be security group with all exposed ports open for all networks.

Alternatives

Creation of security groups by Sahara could be done in several ways. Ideally Sahara should support separation between different networks and configuration on what to allow and what is not.

Data model impact

1) List of security groups need to be saved in each node group.
2) Flag indicating that one of security groups is created by Sahara
3) List of ports to be opened. It need to be stored somewhere to provide this information to provisioning engine.

REST API impact

Requests to create cluster, nodegroup, cluster template and nodegroup template will be extended to receive security groups to use. Also option for automatic security group creation will be added.

Other end user impact

None

Deployer impact

In some cases there will be no need to configure default security group.

Developer impact

Plugin SPI will be extended with method to return required ports for node group.

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

New field to select security group in all create screens.

9.6.3 Implementation

Assignee(s)

Primary Assignee: Andrew Lazarev (alazarev)

Work Items

- Allow user to specify security groups for node group
- Implement ability of security group creation by Sahara

Both items require the following steps:

- Implement in both engines (heat and direct engine)
- Test for nova network and neutron
- Update documentation
- Update UI
- Create integration test

9.6.4 Dependencies

None

9.6.5 Testing

Feature need to be covered by integration tests both for engine and UI.

9.6.6 Documentation Impact

Feature need to be documented.

9.6.7 References

None

9.7 Move the EDP examples from the sahara-extra repo to sahara

https://blueprints.launchpad.net/sahara/+spec/edp-move-examples

Moving the Sahara EDP examples from the sahara-extra repo to the sahara repo accomplishes several things:

- It eliminates code duplication since the examples are actually used in integration tests
- It removes an element from the sahara-extra repo, thereby moving us closer to retiring that repo and simplifying our repo structure
- It puts examples where developers are more likely to find it, and makes it simpler to potentially bundle the examples with a Sahara distribution

9.7.1 Problem description

The goal is to create one unified set of EDP jobs that can be used to educate users and developers on how to create/run jobs and can also be used as jobs submitted during integration testing.

9.7.2 Proposed change

Under the sahara root directory, we should create a new directory:

```
sahara/edp-examples
```

The directory structure should follow a standard pattern (names are not important per se, this is just an illustration):

```
subdirectory_for_each_example/
  README.rst (what it is, how to compile, etc)
  script_and_jar_files
  src_for_jars/
  how_to_run_from_node_command_line/ (optional)
  expected_input_and_output/ (optional)
hadoop_1_specific_examples/
  subdirectory_for_each_example
hadoop_2_specific_examples/
  subdirectory_for_each_example
```

The integration tests should be modified to pull job files from the sahara/edp-examples directory.

Here are some notes on equivalence for the current script and jar files in `sahara-extra/edp-examples` against `sahara/tests/integration/tests/resources`:

```
pig-job/example.pig == resources/edp-job.pig
pig-job/udf.jar == resources/edp-lib.jar
wordcount/edp-java.jar == resources/edp-java/edp-java.jar
```

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

Examples won't be found in the sahara-extra repo any longer. We should perhaps put a README file there that says "We have moved" for a release cycle.

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

9.7.3 Implementation

Assignee(s)

None as yet

Work Items

The problem has several components:

- Move the examples to the sahara repository
- Merge any jobs used by the integration tests into the new examples directory to create one comprehensive set
- Provide source code and compilation instructions for any examples that currently lack them
- Make the integration tests reference the new directory structure
- Delineate which, if any, examples work only with specific Hadoop versions. Most examples work on both Hadoop 1 and Hadoop 2 but some do not. Version-specific examples should be in a subdirectory named for the version

9.7.4 Dependencies

None

9.7.5 Testing

Testing will be inherent in the integration tests. The change will be deemed successful if the integration tests run successfully after the merging of the EDP examples and the integration test jobs.

9.7.6 Documentation Impact

If our current docs reference the EDP examples, those references should change to the new location. If our current docs do not reference the EDP examples, a reference should be added in the developer and/or user guide.

9.7.7 References

None

9.8 [EDP] Refactor job manager to support multiple implementations

https://blueprints.launchpad.net/sahara/+spec/edp-refactor-job-manager

The job manager at the core of Sahara EDP (Elastic Data Processing) was originally written to execute and monitor jobs via an Oozie server. However, the job manager should allow for alternative EDP implementations which can support new cluster types or new job execution environments.

9.8.1 Problem description

To date, the provisioning plugins released with Sahara all support deployment of an Oozie server. Oozie was a logical choice for the early releases of Sahara EDP because it provided commonality across several plugins and allowed the rapid development of the EDP feature.

However, Oozie is built on top of Hadoop Mapreduce and not every cluster configuration can or should support it (consider a Spark cluster, for example, where Hadoop Mapreduce is not a necessary part of the install). As Sahara supports additional cluster types and gains a wider install base, it's important for EDP to have the flexibility to run on new cluster configurations and new job execution paradigms.

The current implementation of the job_manager has hardcoded dependencies on Oozie. These dependencies should be removed and the job manager should be refactored to support the current Oozie implementation as well as new implementations. The job manager should select the appropriate implementation for EDP operations based on attributes of the cluster.

9.8.2 Proposed change

Sahara EDP requires three basic operations on a job:

- Launch the job
- Poll job status
- Terminate the job

Currently these operations are implemented in job_manager.py with explicit dependencies on the Oozie server and Oozie-style workflows.

To move these dependencies out of the job manager, we can define an abstract class that contains the three operations:

```
@six.add_metaclass(abc.ABCMeta)
class JobEngine(object):
    @abc.abstractmethod
    def cancel_job(self, job_execution):
        pass

    @abc.abstractmethod
    def get_job_status(self, job_execution):
        pass

    @abc.abstractmethod
    def run_job(self, job_execution):
        pass
```

For each EDP implementation Sahara supports, a class should be derived from JobEngine that contains the details. Each implementation and its supporting files should be contained in its own subdirectory.

Logic can be added to the job manager to allocate a JobEngine based on the cluster and/or job type, and the job manager can call the appropriate method on the allocated JobEngine.

As much as possible the JobEngine classes should be concerned only with implementing the operations. They may read objects from the Sahara database as needed but modifications to the job execution object should generally be done in the job_manager.py (in some cases this may be difficult, or may require optional abstract methods to be added to the JobEngine base class).

For example, the "cancel_job" sequence should look something like this:

1) "cancel_job(id)" is called in job_manager.py
2) The job manager retrieves the job execution object and the cluster object and allocates an appropriate job engine
3) The job manager calls engine.cancel_job(job_execution)
4) The engine performs necessary steps to cancel the job
5) The job manager traps and logs any exceptions
6) The job manager calls engine.get_job_status(job_execution)
7) The engine returns the status for the job, if possible
8) The job manager updates the job execution object in the Sahara database with the indicated status, if any, and returns

In this example, the job manager has no knowledge of operations in the engine, only the high level logic to orchestrate the operation and update the status.

Alternatives

It may be possible to support "true" plugins for EDP similar to the implementation of the provisioning plugins. In this case, the plugins would be discovered and loaded dynamically at runtime.

However, this requires much more work than a refactoring and introduction of abstract classes and is probably not realistic for the Juno release. There are several problems/questions that need to be solved:

- Should EDP interfaces simply be added as optional methods in the current provisioning plugin interface, or should EDP plugins be separate entities?
- Do vendors that supply the provisioning plugins also supply the EDP plugins?
- If separate EDP plugins are chosen, a convention is required to associate an EDP plugin with a provisioning plugin so that the proper EDP implementation can be chosen at runtime for a particular cluster without any internal glue
- For clusters that are running an Oozie server, should the Oozie EDP implementation always be the default if another implementation for the cluster is not specified? Or should there be an explicitly designated implementation for each cluster type?
- It requires not only a formal interface for the plugin, but a formal interface for elements in Sahara that the plugin may require. For instance, an EDP plugin will likely need a formal interface to the conductor so that it can retrieve EDP objects (job execution objects, data sources, etc).

Data model impact

This change should only cause one minor (optional) change to the current data model. Currently, a JobExecution object contains a string field named "oozie_job_id". While a job id field will almost certainly still be needed for all implementations and can be probably be stored as a string, the name should change to "job_id".

JobExecution objects also contain an "extra" field which in the case of the Oozie implementation is used to store neutron connection info for the Oozie server. Other implementations may need similar data, however since the "extra" field is stored as a JsonDictType no change should be necessary.

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

9.8.3 Implementation

Assignee(s)

Primary Assignee: Trevor McKay

Work Items

9.8.4 Dependencies

None

9.8.5 Testing

Testing will be done primarily through the current unit and integration tests. Tests may be added that test the selection of the job engine.

9.8.6 Documentation Impact

None

9.8.7 References

None

9.9 [EDP] Add a Spark job type (instead of overloading Java)

https://blueprints.launchpad.net/sahara/+spec/edp-spark-job-type

Spark EDP has been implemented initially using the Java job type. However, the spark semantics are slightly different and Spark jobs will probably continue to diverge from Java jobs during future development. Additionally, a specific job type will help users distinguish between Spark apps and Java mapreduce jobs in the Sahara database.

9.9.1 Problem description

The work involves adding a Spark job type to the job type enumeration in Sahara and extending the dashboard to allow the creation and submission of Spark jobs. The Sahara client must be able to create and submit Spark jobs as well (there may not be any new work in the client to support this).

Existing unit tests and integration tests must be repaired if the addition of a new job type causes them to fail. Unit tests analagous to the tests for current job types should be added for the Spark job type.

Integration tests for Spark clusters/jobs will be added as a separate effort.

9.9.2 Proposed change

Changes in the Sahara-api code:

- Add the Spark job type to the enumeration
- Add validation methods for job creation and job execution creation
- Add unit tests for the Spark job type
- Add the Spark job type to the job types supported by the Spark plugin
  - Leave the Java type supported for Spark until the dashboard is changed
- Add config hints for the Spark type
  - These may be empty initially

Changes in the Sahara dashboard:

- Add the Spark job type as a selectable type on the job creation form.
  - Include the "Choose a main binary" input on the Create Job tab
  - Supporting libraries are optional, so the form should include the Libs tab
- Add a "Launch job" form for the Spark job type
  - The form should include the "Main class" input.
  - No data sources, as with Java jobs
  - Spark jobs will share the edp.java.main_class configuration with Java jobs. Alternatively, Spark jobs could use a edp.spark.main_class config
  - There may be additional configuration parameters in the future, but none are supported at present. The Configuration button may be included or left out.
  - The arguments button should be present

Alternatives

Overload the existing Java job type. It is similar enough to work as a proof of concept, but long term this is probably not clear, desirable or maintainable.

Data model impact

None. Job type is stored as a string in the database, so there should be no impact there.

REST API impact

None. The JSON schema will list "Spark" as a valid job type, but the API calls themselves should not be affected.

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

Described under Proposed Change.

9.9.3 Implementation

Assignee(s)

Primary assignee:
Trevor McKay (sahara-api)

Other contributors:
Chad Roberts (dashboard)

Work Items

Additional notes on implementation of items described under Proposed Change:

- The simple addition of JOB_TYPE_SPARK to sahara.utils.edp.JOB_TYPES_ALL did not cause existing unit tests to fail in an experiment
- Existing unit tests should be surveyed and analagous tests for the Spark job type should be added as appropriate
- sahara.service.edp.job_manager.get_job_config_hints(job_type) needs to handle the Spark job type. Currently all config hints are retrieved from the Oozie job engine, this will not be correct.
  - Related, the use of JOB_TYPES_ALL should probably be modified in workflow_creator.workflow_factory.get_possible_job_config() just to be safe
- sahara.service.edp.job_utils.get_data_sources() needs to treat Spark jobs like Java jobs (there are no data sources, only arguments)
- service.validations.edp.job.check_main_libs() needs to require a main application jar for Spark jobs and allow supporting libs as optional
- service.validations.edp.job_executor.check_job_executor()
  - Spark requires edp.java.main_class (or edp.spark.main_class)
  - check_edp_job_support() is called here and should be fine. The default is an empty body and the Spark plugin does not override this method since the Spark standalone deployment is part of a Spark image generated from DIB
- Use of EDP job types in sahara.service.edp.oozie.workflow_creator should not be impacted since Spark jobs shouldn't be targeted to an Oozie engine by the job manager (but see note on get_job_config_hints() and JOB_TYPES_ALL)
- The Sahara client does not appear to reference specific job type values so there is likely no work to do in the client

9.9.4 Dependencies

This depends on https://blueprints.launchpad.net/sahara/+spec/edp-spark-standalone

9.9.5 Testing

New unit tests will be added for the Spark job type, analogous to existing tests for other job types. Existing unit and integration tests will ensure that other job types have not been broken by the addition of a Spark type.

Integration tests for Spark clusters should be added in the following blueprint, including tests for EDP with Spark job types

https://blueprints.launchpad.net/sahara/+spec/edp-spark-integration-tests

9.9.6 Documentation Impact

The User Guide calls out details of the different job types for EDP. Details of the Spark type will need to be added to this section.

9.9.7 References

None.

9.10 [EDP] Add an engine for a Spark standalone deployment

https://blueprints.launchpad.net/sahara/+spec/edp-spark-standalone

The Spark provisioning plugin allows the creation of Spark standalone clusters in Sahara. Sahara EDP should support running Spark applications on those clusters.

9.10.1 Problem description

The Spark plugin uses the *standalone* deployment mode for Spark (as opposed to Spark on YARN or Mesos). An EDP implementation must be created that supports the basic EDP functions using the facilities provided by the operating system and the Spark standalone deployment.

The basic EDP functions are:

- **run_job()**
- **get_job_status()**
- **cancel_job()**

9.10.2 Proposed change

The Sahara job manager has recently been refactored to allow provisioning plugins to select or provide an EDP engine based on the cluster and job_type. The plugin may return a custom EDP job engine object or choose a default engine provided by Sahara.

A default job engine for Spark standalone clusters can be added to Sahara that implements the basic EDP functions.

Note, there are no public APIs in Spark for job status or cancellation beyond facilities that might be available through a SparkContext object instantiated in a Scala program. However, it is possible to provide basic functionality without developing Scala programs.

- **Engine selection criteria**

  The Spark provisioning plugin must determine if the default Spark EDP engine may be used to run a particular job on a particular cluster. The following conditions must be true to use the engine

  - The default engine will use *spark-submit* for running jobs, therefore a cluster must have at least Spark version 1.0.0 to use the engine.
  - The job type must be Java (initially)

- **Remote commands via ssh**

  All operations should be implemented using ssh to run remote commands, as opposed to writing a custom agent and client. Furthermore, any long running commands should be run asynchronously.

- **run_job()**

  The run_job() function will be implemented using the *spark-submit* script provided by Spark.

  - *spark-submit* must be run using client deploy mode
  - The *spark-submit* command will be executed via ssh. Ssh must return immediately and must return a process id (PID) that can be used for checking status or cancellation. This implies that the process is run in the background.
  - SIGINT must not be ignored by the process running *spark-submit*. Care needs to be taken here, since the default behavior of a process backgrounded from bash is to ignore SIGINT. (This can be handled by running *spark-submit* as a subprocess from a wrapper which first restores SIGINT, and launching the wrapper from ssh. In this case the wrapper must be sure to propagate SIGINT to the child).
  - *spark-submit* requires that the main application jar be in local storage on the node where it is run. Supporting jars may be in local storage or hdfs. For simplicity, all jars will be uploaded to local storage.
  - *spark-submit* should be run from a subdirectory on the master node in a well-known location. The subdirectory naming should incorporate the job name and job execution id from Sahara to make locating the directory easy. Program files, output, and logs should be written to this directory.
  - The exit status returned from *spark-submit* must be written to a file in the working directory.
  - *stderr* and *stdout* from *spark-submit* should be redirected and saved in the working directory. This will help debugging as well as preserve results for apps like SparkPi which write the result to stdout.
  - Sahara will allow the user to specify arguments to the Spark application. Any input and output data sources will be specified as path arguments to the Spark app.

- **get_job_status()**

  Job status can be determined by monitoring the PID returned by run_job() via *ps* and reading the file containing the exit status

  - The initial job status is PENDING (the same for all Sahara jobs)
  - If the PID is present, the job status is RUNNING
  - If the PID is absent, check the exit status file - If the exit status is 0, the job status is SUCCEEDED - If the exit status is -2 or 130, the job status is KILLED (by SIGINT) - For any other exist status, the job status is DONEWITHERROR
  - If the job fails in Sahara (ie, because of an exception), the job status will be FAILED

- **cancel_job()**

  A Spark application may be canceled by sending SIGINT to the process running *spark-submit*.

  - cancel_job() should send SIGINT to the PID returned by run_job(). If the PID is the process id of a wrapper, the wrapper must ensure that SIGINT is propagated to the child

  - If the command to send SIGINT is successful (ie, *kill* returns 0), cancel_job() should call get_job_status() to update the job status

Alternatives

The Ooyala job server is an alternative for implementing Spark EDP, but it's a project of its own outside of OpenStack and introduces another dependency. It would have to be installed by the Spark provisioning plugin, and Sahara contributors would have to understand it thoroughly.

Other than Ooyala, there does not seem to be any existing client or API for handling job submission, monitoring, and cancellation.

Data model impact

There is no data model impact, but a few fields will be reused.

The *oozie_job_id* will store an id that allows the running application to be operated on. The name of this field should be generalized at some point in the future.

The job_execution.extra field may be used to store additional information necessary to allow operations on the running application

REST API impact

None

Other end user impact

None. Initially Spark jobs (jars) can be run using the Java job type. At some point a specific Spark job type will be added (this will be covered in a separate specification).

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

9.10.3 Implementation

Assignee(s)

Trevor McKay

Work Items

- implement default spark engine selection in spark provisioning plugin
- implement run
- implement get_job_status
- implement cancel
- implement launch wrapper
- implement unit tests

9.10.4 Dependencies

None

9.10.5 Testing

Unit tests will be added for the changes in Sahara.

Integration tests for Spark standalone clusters will be added in another blueprint and specification.

9.10.6 Documentation Impact

The Elastic Data Processing section of the User Guide should talk about the ability to run Spark jobs and any restrictions.

9.10.7 References

9.11 [EDP] Using trust delegation for Swift authentication

https://blueprints.launchpad.net/sahara/+spec/edp-swift-trust-authentication

Sahara currently stores and distributes credentials for access to Swift objects. These credentials are username/password pairs that are stored in Sahara's database. This blueprint describes a method for using Keystones trust delegation mechanism in conjuction with temporary proxy users to remove the storage of user credentials from Sahara's purview.

9.11.1 Problem description

Sahara allows access to job binaries and data sources stored in Swift containers by storing a user's credentials to those containers. The credentials are stored in Saharas database and distributed to cluster instances as part of a jobs workflow, or used by Sahara to access job binaries. The storage of user credentials in Sahara's database represents a security risk that can be avoided.

9.11.2 Proposed change

A solution to using credentials for access to Swift objects is to generate a Keystone trust between the user with access to those objects and a Sahara proxy user. The trust would be established based on the users membership in the project that contains the Swift objects. Using this trust the Sahara proxy user could generate authentication tokens to access the Swift objects. When access is no longer needed the trust can be revoked, and the proxy user removed thus invalidating the tokens.

A proxy user will be created per job execution, and belong to a roleless domain created by the stack administrator for the express purpose of containing these new users. This domain will allow Sahara to create users as needed for the purpose of delegating trust to access Swift objects.

Sahara will generate usernames and passwords for the proxy users when they are created. These credentials will be used in conjuction with the trust to allow Swift accress from the cluster instances.

General breakdown of the process:

1. On start Sahara confirms the existence of the proxy domain. If no proxy domain is found and the user has configured Sahara to use one, then Sahara will log an error and resume in backward compatibility mode.
2. When a new job that involves Swift objects is executed a trust is created between the context user and a newly created proxy user. The proxy user's name and password are created by Sahara and stored temporarily.
3. When an instance needs access to a Swift object it uses the proxy user's credentials and the trust identifier to create the necessary authentication token.
4. When the job has ended, the proxy user and the trust will be removed from Keystone.

Detailed breakdown:

Step 1.

On start Sahara will confirm the existence of the proxy domain specified by the administrator in the configuration file. If no domain can be found and the user has configured Sahara to use one, then it will log an error and resume in backward compatibility mode. The domain will be used to store proxy users that will be created during job execution. It should explicitly have an SQL identity backend, or another backend that will allow Sahara to create users.(SQL is the Keystone default)

If the user has configured Sahara not to use a proxy domain then it will fall back to using the backward compatible style of Swift authentication. Requiring usernames and passwords for Swift access.

Step 2.

Whenever a new job execution is issued through Sahara a new proxy user will be created in the proxy domain. This user will have a name generated based on the job execution id, and a password generated randomly.

After creating the proxy user, Sahara will delegate a trust from the current context user to the proxy user. This trust will grant a "Member" role by default, but will allow configuration, and impersonate the

context user. As Sahara will not know the length of the job execution, the trust will be generated with no expiration time and unlimited reuse.

Step 3.

During job execution, when an instance needs to access a Swift object, it will use the proxy user's name and password in conjuction with the trust identifier to create an authentication token. This token will then be used to access the Swift object store.

Step 4.

After the job execution has finished, successfully or otherwise, the proxy user and trust will be removed from Keystone.

A periodic task will check the proxy domain user list to ensure that none have become stuck or abandoned after a job execution has been completed.

Alternatives

Three alternatives have been discussed regarding this issue; using Swifts TempURL mechanism, encrypting the Swift credentials, and distributing tokens from Sahara to the cluster instances.

Swift implements a feature named TempURL which allows the generation of temporary URLs to allow public access to Swift objects. Using this feature Sahara could create TempURLs for each Swift object that requires access and then distribute these URLs to the cluster instances in the job workflow.

Although TempURLs would be low impact in terms of the work required to implement they have a few major drawbacks for Saharas use case. When creating a TempURL an expiration date must be associated with the URL. As job lengths in Sahara cannot be predictively determined this would mean creating indefinite expiration dates for the TempURLs. A solution to this would be deleting the Swift object or changing the authentication identifier associated with the creation of the TempURL. Both of these options present implications that run beyond the boundaries of Sahara. In addition the TempURLs would need to be passed to the instances as ciphertext to avoid potential security breaches.

Another methodology discussed involves encrypting the Swift credentials and allowing the cluster instances to decrypt them when access is required. Using this method would involve Sahara generating a two part public/private key that would be used to encrypt all credentials. The decryption, or private, part of the key would be distributed to all cluster nodes. Upon job creation the Swift credentials associated with the job would be encrypted and stored. The encrypted credentials would be distributed to the cluster instances in the job workflow. When access is needed to a Swift object, an instance would decrypt the credentials using the locally stored key.

Encrypting the credentials poses a lower amount of change over using Keystone trusts but perpetuates the current ideology of Sahara storing credentials for Swift objects. In addition a new layer of security management becomes involved in the form of Sahara needing to generate and store keys for use with the credentials. This complexity adds another layer of management that could instead be relegated to a more appropriate OpenStack service(i.e. Keystone).

Finally, another possibilty to achieve the removal of credentials from Sahara would be for the main controller to distribute preauthorized tokens to the cluster instances. These tokens would be used by the instances to validate their Swift access. There are a few implementation problems with this approach that have caused it to be abandoned by a previous version of this blueprint.

Keystone tokens have a default lifespan of one hour, this value can be adjusted by the stack administrator. This implies that tokens must be updated to the instances at least once an hour, possibly more frequently. The updating process proves to add hindrances that are difficult to overcome. Update frequency is one,

but resource contention on the instances is another. A further examination of this method shows that the updates to the Hadoop Swift filesystem component will create a disonant design surrouding the injestion of configuration data. In sum this methodology proves to be more fragile than is acceptable.

Data model impact

The job execution model currently stores username and password information in a field that is a dictionary. There will be no changes to the model, but the trust identifier will need to be stored in addition to the username and password.

Once the credentials have been passed to the cluster the only values that need be stored are the user id and trust identifier. These would need to be used to destroy the trust and user after job execution. The proxy user's password is only needed during the creation of the job execution and will be distributed to the instances, but long term storage is not necessary.

REST API impact

The proxy usernames and trust identifiers should be sanitized from the job execution output.

Other end user impact

Users will no longer need to enter credentials when adding Swift data sources to their jobs.

The users OpenStack credentials will need to have sufficient privileges to access the Swift objects they add.

From the python-saharaclient, developers will no longer need to enter credential_user or credential_pass when making a requests to create data sources.

Keystone deployments that use LDAP backed domains will need to be configured as recommended by the Keystone group, using domain based configurations. This ensures that new domains created will be backed by SQL.

Deployer impact

A deployer will need to be aware of the Keystone configuration with respect to the default identity backend. They will also need to create the proxy domain and provide the Sahara service user with enough access to create new users in that domain.

Developer impact

Developers will no longer need to pass credentials when creating data sources.

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

For backward compatibility the username and password fields should be left in the Swift data source forms and views, but they should allow the user to enter blank data.

9.11.3 Implementation

Assignee(s)

Primary assignee:

- Michael McCune

Other contributors:

- Trevor McKay

Work Items

- Domain detection and configuration option
- Proxy user creation/destruction
- Trust acquisition/revocation
- Workflow update
- Swift file system component update to use trust identifier
- Periodic proxy user removal task
- Documentation
- Tests

9.11.4 Dependencies

This feature will require the usage of Keystone v3 with the OS-TRUST mechanism enabled.

The Horizon forms for Swift data sources will need to allow blank entries for username and password.

The Hadoop Swift file system component will need to be updated as well.

9.11.5 Testing

The current tests for Swift based objects will need to be modified to remove the usage of username/password credentials. Otherwise these tests should prove that the trust method is working properly.

Tests for situations where a users Keystone access do not permit permission for the Swift objects they are adding should be implemented.

The Swift file system component will need it's tests modified to use trust identifiers for scoping of the authentication tokens.

9.11.6 Documentation Impact

The documentation for usage of Swift storage will need to have references to the object credentials removed. Additionally there should be documentation added about the impact of a user not having access to the Swift sources.

The proxy domain usage should be documented to give stack administrators a clear understanding of Sahara's usage and needs. This should also note the impact of Keystone configurations which do not provide default SQL identity backends.

9.11.7 References

Original bug report https://bugs.launchpad.net/sahara/+bug/1321906

Keystone trust API reference https://github.com/openstack/identity-api/blob/master/v3/src/markdown/identity-api-v3-os-trust-ext.md

9.12 Improve error handling for provisioning operations

https://blueprints.launchpad.net/sahara/+spec/error-handling-in-provisioning

Currently provisioning error handling is sprayed across the whole provisioning code. This spec is to unify error handling and localize it in one place.

9.12.1 Problem description

Currently we have two problems connected with error handling in provisioning part:

1) The code incorrectly handles situations when cluster was deleted by user during provisioning. In that case an arbitrary error might be raised in many places.

2) The code performs rollback only in certain places, while it could be done for any provisioning/scaling phase.

The following CR: https://review.openstack.org/#/c/98556 mostly fixes issue #1, but it is full of duplicate code.

9.12.2 Proposed change

The following solution is proposed instead which requires architectural changes, but rather reliably fixes both problems:

1) For both cluster creation and scaling move error handling logic to the very top functions inside ops.py file. Once exception is caught properly process it:
    a) if cluster object does not exists in DB, that means that user deleted the cluster during provisioning; handle it and return
    b) if cluster object exists, log it and perform rollback
2) Do not do any checks if cluster exists outside of ops.py, except places where processing might hang indefinitely without the check.

We can employ the following rollback strategy:

For cluster creation: if anything went wrong, kill all VMs and move cluster to the Error state.

For cluster scaling: that will be long. Cluster scaling has the following stages:

1) decommission unneeded nodes (by plugin)
2) terminate unneeded nodes and create a new ones if needed (by engine). Note that both scaling up and down could be run simultaneously but in different node groups.
3) Configure and start nodes (by plugin)

My suggestion what to do if an exception occurred in the respective stage:

1) move cluster to Error state
2) kill unneeded nodes (finish scale down). Also kill new nodes, if they were created for scale up.
3) move cluster to Error state

In cases #1 and #3 it is dangerous to delete not decommissioned or not configured nodes as this can lead to data loss.

Alternatives

Keep supporting current code. It is not elegant but works good enough.

Data model impact

None

REST API impact

None

Other end user impact

Data provisioning logic will be changed a lot. This could lead to behavior change in case of errors on different stages.

Deployer impact

None

Developer impact

Provisioning engine API will be extended with "rollback_cluster" method.

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

9.12.3 Implementation

Assignee(s)

Primary assignee:
alazarev

Other contributors:
dmitrymex

Work Items

1) Implement change
2) Test that cluster provisioning and rollback works on all feature matrix we have.

9.12.4 Dependencies

None

9.12.5 Testing

Provisioning could be tested manually and by CI. It is much harder to test rollback. Even current code is not tested well (e.g. https://bugs.launchpad.net/sahara/+bug/1337006).

9.12.6 Documentation Impact

None

9.12.7 References

None

9.13 Move Sahara REST API samples from /etc to docs

https://blueprints.launchpad.net/sahara/+spec/move-rest-samples-to-docs

Initially this idea was raised during discussions about moving/releasing/versioning of Sahara's subprojects. REST samples are slightly outdated and common documentation is a good place to have them there to keep them up-to-date.

9.13.1 Problem description

Today REST API samples are outdated and don't reflect current state of changes made in Sahara api since Havana release. Also it's not obvious to find those samples in Sahara sources. The goal is to update current state of samples and move it to http://docs.openstack.org/

9.13.2 Proposed change

Create a new page in docs:

```
sahara/doc/source/restapi/rest_api_samples.rst
```

Move all JSON examples from sahara/etc/rest-api-samples to the new page. Simple description for each example should be added before JSON code blocks.

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

Examples won't be found in the sahara/etc dir any longer.

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

9.13.3 Implementation

Assignee(s)

Primary Assignee: Alexander Ignatov

Work Items

The following steps should be done:

- Move REST samples from sahara/etc to a new page in docs
- Update samples in docs according to current state of Sahara api
- Remove sahara/etc/rest-api-samples directory in Sahara sources

9.13.4 Dependencies

None

9.13.5 Testing

None

9.13.6 Documentation Impact

New page with information about REST samples will appear in the Sahara docs.

9.13.7 References

None

Unimplemented backlog specs:

CHAPTER

TEN

BACKLOG

This directory is for specifications that have been approved but not yet implemented. If you are interested in implementing one of these specifications submit a review request moving it to the appropriate release.

The updated review should reflect the new primary assignee. Please maintain all previous assignees in the list of Other Contributors.

If a specification has been partially implemented, the document in the backlog will contain information of what has been completed.

10.1 Support for Boot from Volume

This specification proposes to add boot from volume capability to Sahara.

10.1.1 Problem description

Sahara engine provisions VMs using a Glance image directly. In most installations this means that the image is copied as a local file the Nova-compute host and used as a root disk for the VM.

Having the root disk as a plain file introduces some limitations:

- Nova VM live migration (or Host evacuation) is not possible.
- Root disk performance may significantly suffer if QCOW2 format is used.

Having root disk replaced with a bootable Volume backed by a distributed backend or local disk storage will solve the limitation listed above.

10.1.2 Proposed change

The ability to boot from volume still requires an image to serve as source. This means that Volume base provisioning will require 2 steps.

- Create a bootable volume from a registered Sahara image.
- Boot a VM with the block_device_mapping parameter pointing to the created volume.

Volume based provisioning requires the volume size to be set explicitly. This means that Sahara should take care about this parameter. The proposed way to set the bootable volume size is take the root disk size of the flavor being used for the VM.

If the user selects the Node Group to be provisioned from the volume, he should set the boot_from_volume flag to True.

The volume based provisioning is different from image based and implies the following change.

The image parameter from the instance template should be removed.

The instance Heat template should have a new section with the block device mapping.

```
block_device_mapping: [{
 device_name: "vda",
 volume_id : {
   get_resource : bootable_volume },
 delete_on_termination : "true" }
]
```

The resource group definition should have a volume added by the following template:

```
bootable_volume:
  type: OS::Cinder::Volume
  properties:
    size: <size derived from the flavor>
    image: <regular Sahara image>
```

Alternatives

Alternatively the user may be allowed to chose an existing volume to boot from. This however cannot guarantee that the provided volume is suitable for the cluster installation. Sahara also requires a username metadata to be able to log into VM. This metadata is only stored in images right now.

Data model impact

Node Group Template, Node Group and Templates relation objects should now have a boolean boot_from_volume field.

REST API impact

Convert and Boot from Volume flag should be added to all endpoints responsible for Node Group manipulation.

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

An option should be added to the Node Group create and update forms.

10.1.3 Implementation

Assignee(s)

Primary assignee:
Nikita Konovalov nkonovalov@mirantis.com

Work Items

- Implement boot from volume support at the backend with the db migration and Heat templates update.
- Add boot_from_volume flag support in python-saharaclient
- Add boot_from_volume flag support in sahara-dashboard

10.1.4 Dependencies

None

10.1.5 Testing

- Unit test coverage in sahara and python-saharaclient repositories.
- Integration test coverage in sahara-tests framework.

10.1.6 Documentation Impact

- REST API documents should be updated.
- General user documentation should describe the behavior introduced by the boot_from_volume flag.

10.1.7 References

None

10.2 Revival of "Chronic EDP" work

https://blueprints.launchpad.net/sahara/+spec/enable-scheduled-edp-jobs https://blueprints.launchpad.net/sahara/+spec/recurrence-edp-job https://blueprints.launchpad.net/sahara/+spec/add-suspend-resume-ability-for-edp-jobs

10.2.1 Problem description

Three specs about time-related EDP job submission are partially completed. The work could be revived.

10.2.2 Proposed change

With reference to existing specs/patches, revive any or all of the 3 specs.

Alternatives

Keep the specs in their current state, either abandoned or partially complete.

Data model impact

Refer to existing specs.

REST API impact

Refer to existing specs.

Other end user impact

Refer to existing specs.

Deployer impact

Refer to existing specs.

Developer impact

Refer to existing specs.

Image Packing impact

Refer to existing specs.

Sahara-dashboard / Horizon impact

Refer to existing specs.

10.2.3 Implementation

Assignee(s)

Primary assignee:
None

Other contributors:
None

Work Items

- Enumerate and verify what has already been done
- Decide what work should be revived
- Clean up doc of exisitng work
- Follow steps outlined in existing specs for new work

10.2.4 Dependencies

None

10.2.5 Testing

Refer to existing specs.

10.2.6 Documentation Impact

Refer to existing specs.

10.2.7 References

None

10.3 Two step scaling with Heat engine

https://blueprints.launchpad.net/sahara/+spec/heat-two-step-scaling

In case of failure during cluster scaling Heat will rollback deletion of all resources. After that Sahara will ask Heat to delete them anyway. "Rollback deletion and delete after that" step looks unnecessary.

10.3.1 Problem description

The following scenario happens when Sahara with Heat engine failed to scale cluster:

1. User requested cluster scaling
2. Sahara runs hadoop nodes decommissioning
3. Sahara runs heat stack update with both added and removed nodes and rollback_on_failure=True
4. If 3 failed Heat returns all deleted nodes back
5. Sahara runs heat stack update with removed nodes only
6. Heat removes nodes one more time

So, at step 4 Heat restores nodes that will be deleted later anyway.

10.3.2 Proposed change

The described problem could be avoided by scaling in two steps. So, resulting flow will look like this:

1. User requested cluster scaling
2. Sahara runs hadoop nodes decommissioning
3. Sahara runs heat stack update with removed resources only and rollback_on_failure=False
4. Sahara runs heat stack update with new resources and rollback_on_failure=True

In this case if step 4 failed Heat will not try to restore deleted resources. It will rollback to the state where resources are already deleted.

If step 3 fails there is nothing Sahara can do. Cluster will be moved to 'Error' state.

Alternatives

Do nothing since issue appears only in rare scenario of failed scaling down.

Data model impact

None.

REST API impact

None.

Other end user impact

None.

Deployer impact

None.

Developer impact

None.

Sahara-image-elements impact

None.

Sahara-dashboard / Horizon impact

None.

10.3.3 Implementation

Assignee(s)

Primary assignee:
alazarev (Andrew Lazarev)

Other contributors:
None

Work Items

- Perform changes
- Test that rollback on cluster scale failure works as expected

10.3.4 Dependencies

None.

10.3.5 Testing

Manually.

10.3.6 Documentation Impact

None.

10.3.7 References

None.

Other components:

CHAPTER

ELEVEN

SAHARA CLIENT

11.1 SaharaClient CLI as an OpenstackClient plugin

https://blueprints.launchpad.net/python-saharaclient/+spec/cli-as-openstackclient-plugin

This specification proposes to create SaharaClient CLI as an OpenstackClient plugin.

11.1.1 Problem description

Currently SaharaClient CLI has a lot of problems and is not so attractive as wanted to be. It should be refactored or recreated from the start.

11.1.2 Proposed change

New SaharaClient CLI will be based on OpenstackClient that brings the command set for different projects APIs together in a single shell with a uniform command structure.

OpenStackClient provides first class support for the several services. The other ones (including Data Processing service) may create an OpenStackClient plugin.

Proposed Objects:

- plugin
- image
- data source
- job template
- job
- job binary
- node group template
- cluster template
- cluster

Proposed Commands:

All commands will have a prefix `dataprocessing`. Arguments in `[]` are optional, in <> are positional.

For Plugins:

```
plugin list [--long]
plugin show <plugin>
plugin configs get <plugin> <version> [--file <filepath>] # default name of
# the file is <plugin>
```

Detailed description of a plugin contains too much information to display it on the screen. It will be saved in file instead. If provided file exists, data will not be rewritten.

Columns for `plugin list`: name, versions. Columns for `plugin list --long`: name, versions, title, description. Rows for `plugin show`: name, versions, title, description.

For Images:

```
image list [--tags <tag(s)>] [--long]
image show <image>
image register <image> <username> [--description <description>]
image unregister <image(s)>
image tags set <image> <tag(s)>
image tags add <image> <tag(s)>
image tags remove <image> <tag(s)>
```

`tags set` will replace current image tags with provided ones. `tags remove` will support passing `all` to `tags` argument to remove all tags.

Columns for `image list`: name, id, username, tags. Columns for `image list --long`: name, id, username, tags, status, description. Rows for `plugin show`: name, id, username, tags, status, description.

For Data Sources:

```
data source create <name> <type> <url> [--password <password>]
    [--username <user>] [--description <description>]
data source list [--type <type>] [--long]
data source show <datasource>
data source delete <datasource(s)>
data source update <datasource> [--name <name>] [--type <type>]
    [--url <url>] [--password <password>] [--username <user>]
    [--description <description>]
```

Columns for `data source list`: name, id, type. Columns for `data source list --long`: name, id, type, url, description. Rows for `data source show`: name, id, type, url, description.

New CLI behavior in case of Node Group Templates, Cluster Templates and Clusters creation will be pretty much the same as in Horizon, but additionally will allow to create them from json. It doesn't let to mark some arguments as required for successful creation, but it could be done in help strings.

For Job Binaries: Job Binaries and Job Binary Internals will be combined

```
job binary create <name> [--data <filepath>] [--description <description>]
    [--url <url>] [--username <username>] [--password <password>]
job binary list [--name <name-regex>]
job binary show <job-binary>
job binary update <job-binary> [--description <description>] [--url <url>]
    [--username <username>] [--password <password>]
```

(continues on next page)

(continued from previous page)

```
job binary delete <job-binary(ies)>
job binary download <job-binary> [--file <filepath>]
```

Columns for `job binary list`: name, id. Columns for `job binary list --long`: name, id, url, description. Rows for `job binary show`: name, id, url, description.

For Node Group Templates:

```
node group template create [--name <name>] [--plugin <plugin>]
    [--version <version>] [--flavor <flavor>] [--autoconfigs]
    [--node-processes <node-processes>] [--floating-ip-pool <pool>]
    [--proxy-gateway] [--configs <filepath>] [--json <filepath>]
    # and other arguments except of "image-id"
node group template list [--plugin <plugin>] [--version <version>]
    [--name <name-regex>] [--long]
node group template show <node-group-template>
node group template configs get <node-group-template> [--file <filepath>]
    # default name of the file is <node-group-template>
node group template update <node-group-template> ... [--json <filepath>]
    # and other arguments the same as in create command
node group template delete <node-group-template(s)>
```

Columns for `node group template list`: name, id, plugin, version. Columns for `node group template list --long`: name, id, plugin, version, node-processes, description. Rows for `node group template show`: name, id, plugin, version, node-processes, availability zone, flavor, is default, is proxy gateway, security groups or auto security group, if node group template contains volumes following rows will appear: volumes per node, volumes local to instance, volumes mount prefix, volumes type, volumes availability zone, volumes size, description.

For Cluster Templates:

```
cluster template create [--name <name>] [--description <description>]
    [--node-groups <ng1:1,ng2:2>] [--anti-affinity <node-processes>]
    [--autoconfigs] [--configs <filepath>] [--json <filepath>]
cluster template list [--plugin <plugin>] [--version <version>]
    [--name <name-regex>] [--long]
cluster template configs get <cluster-template> [--file <filepath>]
    # default name of the file is <cluster-template>
cluster template show <cluster-template>
cluster template update <cluster-template> ... [--json <filepath>]
    # and other arguments the same as in create command
cluster template delete <cluster-template(s)>
```

Plugin and its version will be taken from node group templates.

Columns for `cluster template list`: name, id, plugin, version. Columns for `cluster template list --long`: name, id, plugin, version, node groups (in format name:count), description. Rows for `cluster template show`: name, id, plugin, version, node groups, anti affinity, description.

For Clusters:

```
cluster create [--name <name>] [--cluster-template <cluster-template>]
    [--description <description>][--user-keypair <keypair>]
    [--image <image>] [--management-network <network>] [--json <filepath>]
    [--wait]
cluster scale [] [--wait]
cluster list [--plugin <plugin>] [--version <version>]
    [--name <name-regex>] [--long]
cluster show <cluster>
cluster delete <cluster(s)> [--wait]
```

If `[--wait]` attribute is set, CLI will wait for command completion. Plugin and its version will be taken from cluster template.

Columns for `cluster list`: name, id, status. Columns for `cluster list --long`: name, id, url, description. Rows for `cluster show`: name, id, anti affinity, image id, plugin, version, is transient, status, status_description, user keypair id, description.

For Job Templates (Jobs):

```
job template create [--name <name>] [--type <type>]
    [--main-binary(ies) <mains>] [--libs <libs>] [--description <descr>]
    [--interface <filepath>] [--json <filepath>]
job template list [--type <type>] [--name <name-regex>] [--long]
job template show <job-template>
job template delete <job-template>
job template configs get <type> [--file <file>] # default file name <type>
job types list [--plugin <plugin>] [--version <version>] [--type <type>]
    [--hints] [--file <filepath>] # default file name depends on provided
    # args
```

`job types list` and `job template configs get` outputs will be saved in file just like `plugin configs get`.

Columns for `job template list`: name, id, type. Columns for `job template list --long`: name, id, type, libs(ids), mains(ids), description. Rows for `job template show`: name, id, type, libs(ids), mains(ids), description.

For Jobs (Job Executions):

```
job execute [--job-template <job-template>] [--cluster <cluster>]
    [--input <data-source>] [--output <data-source>] [--args <arg(s)>]
    [--params <name1:value1,name2:value2>]
    [--configs <name1:value1,name2:value2>]
    [--interface <filepath>] [--json <filepath>] [--wait]
job list [--long]
job show <job>
job delete <job(s)> [--wait]
```

Columns for `job list`: id, cluster id, job id, status. Columns for `job list --long`: id, cluster id, job id, status, start time, end time Rows for `job show`: id, cluster id, job id, status, start time, end time, input id, output id

If `[--wait]` attribute is set, CLI will wait for command completion.

Besides this, there are a bunch of arguments provided by OpenstackClient, that depends on chosen command plugin. For example, there is a help output for `plugin list` command:

```
(openstack) help dataprocessing plugin list
usage: dataprocessing plugin list [-h] [-f {csv,html,json,table,value,
                                            yaml}]
                                  [-c COLUMN] [--max-width <integer>]
                                  [--quote {all,minimal,none,nonnumeric}]
                                  [--long]

Lists plugins

optional arguments:
-h, --help            show this help message and exit
--long                List additional fields in output

output formatters:
output formatter options

  -f {csv,html,json,table,value,yaml}, --format {csv,html,json,table,value,
                                                  yaml}
                      the output format, defaults to table
  -c COLUMN, --column COLUMN
                        specify the column(s) to include, can be repeated

table formatter:
  --max-width <integer>
                        Maximum display width, 0 to disable

CSV Formatter:
  --quote {all,minimal,none,nonnumeric}
                        when to include quotes, defaults to nonnumeric
```

Alternatives

Current CLI code can be refactored.

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

11.1.3 Implementation

Assignee(s)

Primary assignee:
apavlov-n

Work Items

- Creating OpenstackClient plugin for SaharaClient
- Commands implementation for each object, described in "Proposed change" section
- Updating documentation with corresponding changes
- Old CLI will be deprecated and removed after some time

11.1.4 Dependencies

None

11.1.5 Testing

Every command will be provided with unit tests.

11.1.6 Documentation Impact

Documentation about new CLI usage will be written.

11.1.7 References

OpenstackClient documentation about using Plugins OpenstackClient documentation about Objects and Actions naming

11.2 CLI: delete by multiple names or ids

https://blueprints.launchpad.net/python-saharaclient/+spec/cli-delete-by-multiple-names-or-ids

In sahara-cli you can only delete an object by name or id. In several os clients, such as nova and heat, you can delete an object by providing a list of ids or names. This blueprint is about adding this feature in sahara-cli.

11.2.1 Problem description

The CLI does not include deletion of objects by list of names or ids.

11.2.2 Proposed change

- Long term solution

Our long term goal is to have the sahara-cli more consistent with other os clients.

Current CLI usage for cluster-delete:

```
sahara cluster-delete [--name NAME] [--id <cluster_id>]
```

Nova CLI usage for delete:

```
nova delete <server> [<server> ...]
```

In nova-cli, and other os clients, you pass directly the id(s) or the name(s) of the items you want to delete. We can refactor sahara-cli to remove the --name and --id arguments. So in long term the usage of sahara cli will be:

```
sahara cluster-delete <cluster> [<cluster> ...]

Positional arguments:
 <cluster>  Name or ID of cluster(s).``
```

- Short term solution

Note that the CLI refactoring will take substantial time, so as short term solution, can temporary use --names and --ids for all delete verbs of the CLI. And once the CLI will be refactored, we will remove all --name(s) and --id(s) arguments.

So the proposed change implies to add --names and --ids arguments which consist of a Comma separated list of names and ids:

```
sahara cluster-delete [--name NAME] [--id cluster_id]
                      [--names NAMES] [--ids IDS]
```

Alternatives

No short term solution and just depend on the CLI refactoring to provide this feature.

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

11.2.3 Implementation

Update CLI methods *_delete in saharaclient/api/shell.py

Assignee(s)

Primary Assignee: Pierre Padrixe (stannie)

Work Items

- Add delete by list of names or ids in the CLI
- Once the CLI is refactored, remove --name(s) --id(s) arguments

11.2.4 Dependencies

- For long term solution: we depend on the refactoring of the CLI
- For short term solution: none

11.2.5 Testing

Update the tests to delete list of names and ids

11.2.6 Documentation Impact

Documentation of the CLI needs to be updated

11.2.7 References

None

CHAPTER

TWELVE

SAHARA IMAGE ELEMENTS

12.1 Add an option to sahara-image-create to generate bare metal images

https://blueprints.launchpad.net/sahara/+spec/sahara-bare-metal-images

Bare metal image generation is supported by disk-image-create but there is no option exposed in sahara-image-create that allows the generation of sahara images for bare metal. If sahara is to support bare metal deployments, we must have sahara bare metal images.

12.1.1 Problem description

Users should have a simple option to generate a sahara bare metal image. This option should be applicable to all platforms and all plugins. The default behavior of sahara-image-create should remain unchanged -- it should generate VM images if the option is not enabled.

To generate a sahara bare metal image, the "vm" element needs to be left out of the element list and the "grub2", "baremetal", and "dhcp-all-interfaces" elements should be added.

12.1.2 Proposed change

Add a "-b" command line option that sets a boolean flag indicating bare metal image generation. If this option is set, add "grub2 baremetal dhcp-all-interfaces" to the list of elements passed to disk-image-create and prevent the "vm" element from being passed.

Do not bother making the list of baremetal elements modifiable from the shell. It's unlikely that capability will be needed.

If the "-b" command line option is not set, no bare metal elements will be added to the element list and the "vm" element will not be removed.

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

No impact beyond the change to sahara-image-create itself.

Sahara-dashboard / Horizon impact

None

12.1.3 Implementation

Assignee(s)

Primary assignee:
tmckay

Work Items

A single patch to diskimage-create.sh

12.1.4 Dependencies

None

12.1.5 Testing

Manually test the generation of bare metal images for all OSs and plugins. Image generation should succeed and generate .vmlinuz, .initrd, and .qcow2 files. However, successful generation doesn't guarantee that they will actually work (for instance in Kilo, Fedora and CentOS 6 images will not boot correctly in ironic)

For this change, it is enough that the image generation completes.

12.1.6 Documentation Impact

The "-b" option should be mentioned where we discuss image generation.

12.1.7 References

None

CHAPTER

THIRTEEN

SAHARA TESTS

13.1 Add an API to sahara-scenario for integration to another frameworks

https://blueprints.launchpad.net/sahara-tests/+spec/api-for-scenario-tests

Sahara scenario is the main tool for testing Sahara and we can provide API for using this framework in another tools/frameworks/tests.

13.1.1 Problem description

When you perform testing of Sahara from another framework, you can't reuse created scenarios and need to create similar code in another framework with new scenarios for Sahara. I think, that we can implement simple Python API for running Sahara scenarios from default templates. It will be useful, for example in frameworks with destructive tests or something else.

13.1.2 Proposed change

Refactor current *runner.py* module, pull out code in *sahara_tests/scenario/utils.py* for reusing it in API methods.

I propose to move code with:

- creation of tmp files,
- working with .mako,
- running tests,
- merging/generation of scenario files.

In *runner.py* we should leave only calls of this methods, for more convenient usage. The next step is adding the directory *api* in *sahara_tests/scenario* with files for implementing API.

I think, we need to add file *api.py* with methods for external usage:

- def run_scenario(plugin, version, release=None)

and *base.py* with preparing of files for running and auxiliary methods in the future.

In future we can separate one scenario into steps:

- create node group template;
- create cluster template;

- create cluster;
- create data sources;
- create job binaries;
- perform EDP;

Alternatives

None

Data model impact

None

REST API impact

None

Other end user impact

None

Deployer impact

None

Developer impact

None

Sahara-image-elements impact

None

Sahara-dashboard / Horizon impact

None

13.1.3 Implementation

Assignee(s)

Primary assignee:
esikachev

Work Items

- move methods from *runner.py*
- add the ability to run tests via API
- add the job for testing API

13.1.4 Dependencies

None

13.1.5 Testing

We can create separate job on Sahara-ci with custom script for checking API calls and correct performing of actions.

13.1.6 Documentation Impact

The API implementation should be mentioned in Sahara-tests docs.

13.1.7 References

None

13.2 Feature sets for scenario tests

The runner and the templates for scenario tests do not provide an easy way to specify the dependency between a certain item (i.e. an EDP job for a certain cluster) and its configuration (i.e. credentials, the definition of the job). This proposal tries to address the problem.

13.2.1 Problem description

A key feature of sahara-scenario is the support for test templates They support parameters which allow tester to describe different test scenarios without writing multiple copies of test templates which differs only by few arguments. This flexibility is somehow limited when testing features like S3 support. Testing S3 integration requires few additional details to be specified when configuring the tests:

- the credentials required to access the S3 API;
- the definitions of the new EDP jobs which uses S3;
- for each cluster where the jobs needs to be executed, the name of the EDP job.

The first two items can be easily added to new files that can be specified as argument for the sahara-scenario command (for example *credentials_s3.yaml.mako* and *edp_s3.yaml.mako*). The keys that they contain (*credentials* and *edp_jobs_flow*) are merged together by the runner. Their content may also be added directly to the existing YAML files (*credentials.yaml.mako* and *edp_s3.yaml.mako*) but that would mean adding a long list of default values for all the arguments in the template.

The third item is more complicated to model, because there is no easy way to override an item in a *cluster* element, which is a list, not a dictionary.

While it could be possible to introduce a command line parameter to override the items in a specific cluster, that would still leave up to the user to remember to specify all the required bits to enable S3 testing.

A more general solution to the problem is the definition of feature sets for testing.

13.2.2 Proposed change

The user of sahara-scenario would only need to pass an argument like:

```
sahara-scenario ... --feature s3 ...
```

If the *-p* and *-v* arguments are specified, for each *feature* argument *sahara-scenario* will include the files *credentials_<feature>.yaml.mako* and *edp_<feature>.yaml.mako*, if they exist.

In addition, from the list of EDP jobs specified for all enabled clusters, all the items marked with *feature: s3* will be selected.

This means that items without the *feature* tag will always be executed, while items with *feature* will be executed only when the associated feature(s) are selected.

The initial implementation will focus on EDP jobs, but other items may benefit from the tagging.

Alternatives

Use conditional statements in the YAML file. But the author of this spec requested multiple times to keep the YAML files free of business logic and purely declarative instead when the *sahara-scenario* initial spec was discussed.

Another possible solution is the duplication of the YAML templates, which is going against maintainability and easiness of usage.

Data model impact

None.

REST API impact

None.

Other end user impact

None.

Deployer impact

The new test runner will be backward compatible with the old templates, but new templates will require the new runner, but this should not be a problem.

Developer impact

None.

Image Packing impact

None.

Sahara-dashboard / Horizon impact

None.

13.2.3 Implementation

Assignee(s)

Primary assignee:
ltoscano

Work Items

- extend the runner to accept the *feature* argument
- add new EDP jobs and add the feature marker to the EDP jobs which need it, extending the existing attempt of S3 testing (see *References*.) which is an early attempt to solve the issue.

13.2.4 Dependencies

None.

13.2.5 Testing

The new argument and the merging of values will be covered by unit tests. The regression testing of will be covered by the *sahara-tests-scenario* jobs in the gate.

13.2.6 Documentation Impact

The new arguments and its usage will be documented from the user and the test writer point of view.

13.2.7 References

Initial work to support S3 testing:

- https://review.openstack.org/610920
- https://review.openstack.org/590055