Salt - Targeted teardown of instances, clusters or systems

Background

The Salt orchestration conductor is a runner extension developed to provide a single tool to support end to end systems management using a configuration as code data model throughout all phases of product delivery such as provisioning, code and product deployment, configuration and operational management of instances, clusters and full systems within a consistent framework based on Saltstack and Python.

In addition to provisioning new systems, Conductor is and end to end orchestration tool that also provides a single interface to manage systems with tasks like expanding clusters, replacing instances, reporting, system teardown (destroy) and any operational task required.

This document serves to provide information about how to destroy or teardown instances, clusters or systems created with Salt Conductor runner. More information regarding Conductor, Saltstack features and implementation details are beyond the scope of this document.

Summary

Saltstack basic functionality has many facets. Two important ones are Targeting and Grains. Grains are one of many methods that can be used to Target instances aka minions. Salt-minion has its own configuration separate from the master. The Conductor tool is used to create, destroy, upsize, or downsize instances, clusters or systems that it provisioned. Full details about Conductor orchestration tool can be found in Salt - Conductor Orchestration framework.

Targeting salt minions can be done using pillar data or the salt engine (cli and api). This document describes targeting by grains and globs (regex) via cli. The document also describes how to target using the Conductor runner command interface which sits on top of the salt api, for the purpose of destroying/tearing down single instances, multiple instances, clusters, or systems.

Like the create, upsize or downsize actions supported in Conductor, the destroy (aka terminate) action has hooks to allow for environment, team/group or product/role scope actions to be injected before or after the destroy takes place. Tasks such as disable monitoring, remove instance names from dns, remove from service discovery, remove from load balancers, cleanup and archive logs and many others can be performed.

Destroy or Teardown targeted instances, clusters or systems

Reference: Target by grains

Reference: What is Salt

Each sub section that follows provides a Conductor runner command use case and brief explanation.

> **two factor parameter logic in conductor**
> In EVERY command where destroy action is specified, the 'group' option MUST be specified as well as either role or grain. Group is the product.group grain that is also baked into every instance that Conductor provisions. This grain is very important as it is tied to each teams/groups specific deployment and configuration of shared, common or 3rd party implementations.
>
> So the combination group=xxxx and role=xxx (or grain='{}') is the only way a destroy action can be successful (assuming matches are found) using the Conductor. This way no product groups instances with role=xxx can be destroyed in error, group is forced to be specified.
>
> **This technique is two factor parameter logic in Conductor.**

In all use cases, if the group and either role or grain values passed into Conductor do not match any minions, no actions are taken.

Target by role salt grain (instance or instances)

using role grain and required group

```
salt-run conductor.group destroy group=TeamX role=activemq pillarenv=dev
region=us-east-1a
```

Command Deconstruction: (full description shown in this example only)

- *salt-run* - this is the built-in salt engine to invoke a runner. It installs as part of the salt-master package
- *conductor.group* - 'conductor' runner extension and submodule 'group'
- *destroy* - the action [create, destroy, upsize, downsize] <-- the latter two are for clusters
- *group* - the product group where the role resides, this maps to the product.group salt grain on the minions
- *role* - the product/role to target on, this maps to the role salt grain
- *pillarenv* - this specifies the salt pillar environment. The pillar environment is mapped to the branch in the salt pillar repository tree. Can be any defined environment usually mapping to business requirements such as dev/qa/stage/prod etc...There is also a saltenv parameter, but not needed here and will be explained in other documentation. A salt-master can server multiple environments spanning multiple regions, so this is required.
- *region* - the cloud region (aws in this case) where the targeted command will be executed. A salt-master can server multiple environments spanning multiple regions, so this is required.

Role is a core concept in the configuration data model and it is used throughout configuration, state and automation. Role in configuration maps to the 'role' salt grain that is set on instances created with Conductor.

The role grain has uniqueness when multiple teams are deploying different implementations of the same 3rd party (shared) product. In this case the role grain is in the form TeamX.Product.

Example TeamA and TeamB both need to create instances running 3rd party supported product Apache cassandra. Cassandra would be configured in the data model as a 3rd party 'shared' configuration that is re-usable across multiple teams because configuration variables and deltas are externalized for teams to override. In this case TeamA cassandra instances would have role TeamA.cassandra, and TeamB cassandra instances would have role TeamB.cassandra. This also provides a 'namespace' method when coding all this in the configuration data model so we avoid any conflicts.

'role' is a salt grain created on each minion that was created by Conductor by way of Pillar provisioning template configuration. It has been implemented as a required grain. Example here.

The concept of 'role' grain is to identify what function the machine is performing in the salt environment. These terms are interchangeable in most organizations when it comes to Configuration Management:

- role
- profile
- server type

There are other grains created on minions that are closely related to the 'role' grain. These are listed for reference:

- composite.role - a list type gain added when a single instance is configured to perform multiple function. This is very useful when configuring multi-homed instances in smaller scale environments for example. The composite.role grain is checked when issuing a destroy command to the Conductor and specifying the role. So if an instance has multiple roles and one of them is the role specified in the destroy command, it will be included in the action.
- base.role - a grain used when a 3rd party product is the basis of the role. Example, TeamA deploys instances configured to run mongodb. The role grain would be 'TeamA.mongodb', the base.role would be 'mongodb'. This grain can NOT be used to destroy or terminate instances.

Target by node name filter (instance or instances)

using node glob and required group

```
salt-run conductor.group destroy group=TeamX node=sometext pillarenv=dev
region=us-east-1a
```

Command Deconstruction:

The *node* parameter takes a string argument. The above target example will match and terminate any instance in the 'dev' salt environment residing in AWS region us-east-1 in zone A and owned by product.group TeamX that has 'sometext' in the instance Name tag.

> The Conductor also sets the newly provisioned instance local hostname (which in turn is the salt grain 'id') to the same thing that is specified in the cloud map for instance name, which is the Name tag for the instance. So we could update the Conductor code to use the salt grain 'id' instead of the AWS Name tag. This could provide better control over things as humans could change the instance Name tag. If changed to use salt grain 'id', someone changing the AWS Name tag wouldn't affect targeting. Using the salt grains 'id' would be the more "Salty" approach.

Target by cloud.system.id salt grain (environment scope systems/mixed clusters and/or instances)

using salt grain cloud.system.id and required group

```
salt-run conduct.group destroy group=teamX grain='{"cloud.system.id":
4}' pillarenv=qa region=us-east-1a
```

Command Deconstruction:

The grain parameter in the Conductor command interface takes a single entry dictionary for its value. The above target example will match and terminate any instance in the 'qa' salt environment residing in AWS region us-east 1 in zone A and owned by product.group teamX with the 'cloud.system.id' grain set to value 4.

cloud.system.id is an environment scoped salt grain set on any instance that was created as part of a system provisioned using the Conductor. A 'system' is one of three provisioning types supported by Conductor.

This is a dynamically created grain at Conductor runtime.

Target by group.system.id salt grain (group scope systems/mixed clusters and/or instances)

using salt grain cloud.system.id and required group

```
salt-run conduct.group destroy group=teamX grain='{"teamX.system.id":
1}' pillarenv=qa region=us-east-1a
```

Command Deconstruction:

The grain parameter in the Conductor command interface takes a single entry dictionary for its value. The above target example will match and terminate any instance in the 'qa' salt environment residing in AWS region us-east 1 in zone A and owned by product.group teamX with the 'teamX.system.id' grain set to value 1.

teamX.system.id is a group scoped salt grain set on any instance that was created as part of a system provisioned using the Conductor by group=teamX. A 'system' is one of three provisioning types supported by Conductor.

The affective result of this command would be the same as the one above using cloud.system.id as long as the value passed in is accurate. In other words for every system created, each instance of that system would get the cloud.system.id grain set to the next available (conductor logic) , as well as the group specific teamX.system.id next available grain.

This is a dynamically created grain at Conductor runtime.

Target by cloud.product.cluster.id (environment scope product cluster)

using salt grain cloud.product.cluster.id and required group

```
salt-run conduct.group destroy group=teamX
grain='{"cloud.cassandra.cluster.id": 3}' pillarenv=dev
region=us-east-1a
```

Command Deconstruction:

The grain parameter in the Conductor command interface takes a single entry dictionary for its value. The above target example will match and terminate any instance in the 'dev' salt environment residing in AWS region us-east 1 in zone A and owned by product.group teamX that is a cassandra instance with cloud.cassandra.cluster.id 3.

Although cloud.cassandra.cluster.id is environment scope (meaning that the next available increments regardless of product.group when Conductor is allocating it), the above command is still group specific because we always require the "group" parameter when invoking a destroy action with Conductor.

This is a dynamically created grain at Conductor runtime.

Target by productgroup.product.cluster.id (group scope product cluster)

using salt grain cloud.product.cluster.id and required group

```
salt-run conduct.group destroy group=teamX
grain='{"teamX.cassandra.cluster.id": 2}' pillarenv=dev
region=us-east-1a
```

Command Deconstruction:

The grain parameter in the Conductor command interface takes a single entry dictionary for its value. The above target example will match and terminate any instance in the 'dev' salt environment residing in AWS region us-east 1 in zone A and owned by product.group teamX that is a cassandra instance with teamX.cassandra.cluster.id 2.

Fail example

```
salt-run conduct.group destroy group=Foobar
grain='{"teamX.cassandra.cluster.id": 2}' pillarenv=dev
region=us-east-1a
```

The above command would not match any targeted minions because the group parameter passed does not line up with the product group specific grain it's looking for because of two factor parameter logic in Conductor.

This is a dynamically created grain at Conductor runtime using 'next available' technique.

Target by cpid salt grain (special cloud provisioning id, can be instance/s, clusters, or systems)

using salt grain cpid and required group

```
salt-run conduct.group destroy group=teamX grain='{"cpid":
8916868746132874 68}' pillarenv=dev region=us-east-1a
```

Command Deconstruction:

The grain parameter in the Conductor command interface takes a single entry dictionary for its value. The above target example will match and terminate any instance in the 'dev' salt environment residing in AWS region us-east 1 in zone A and owned by product.group teamX that was provisioned as part of the ***32 bit unique cloud provisioning id*** generated on each run of Conductor.

The cpid salt grain is created on each instance provisioning in a single execution of Conductor. cpid is a random 32 bit number created each time Conductor is invoked. It is used to provide Parallelization throughout the Conductor code.

When used in a destroy command, it will tell the Conductor to terminate any instance that was created in that specific execution regardless of the provision type (instance/s, cluster/s, or system).

Addendum

For reference here are a list of commands that can be run to query minions via targeting with grains as noted in this document. Many of the examples use the ping method in the salt module 'test'. I.E. test.ping. This is useful for showing functionality, grains available etc...

The last this is an embedded file showing the results from a multi-grains salt command from the salt-master. The output is yaml, but can be set to json or other formats, with a salt switch in the command.

salt grain target examples

```
GRAIN

salt -G 'role:cassandra' test.ping

salt -G 'cpid:751387512378513' test.ping

salt -G 'cloud.nifi.cluster.id:8' test.ping

salt -G 'teamA.nifi.cluster.id:2' test.ping

COMPOUND (more powerful, can be regex, glob, grain etc...)

salt -C '*foobar* and G@role:cassandra' test.ping (target all nodes with
foobar in the name and role grain set to cassandra)

salt -C '*foobar* and ( G@role:cassandra or G@composite.role:*cassandra*
) and G@product.group:devops' test.ping
(target all devops product group nodes with foobar in name, role grain
cassandra, or composite.role grain includes *cassandra*)
```

The file below was generated from the following salt command issues from the master

```
salt '*' grains.item cpid cloud.system.id cloud.nifi.cluster.id
devops.nifi.cluster.id devops.system.id role
```