Salt Runner - Conductor Orchestration Framework

....work in progress....

Overview

Saltstack has many subsystems, together they become Salt ecosystem. These subsystems are accessed via a variety of Salt 'engines'. Salt engines are exposed via executables.

For reference, here are some of the more important Salt engines used in our automation:

- **salt** - the executable running on the salt-master used to run remote salt modules commands and configuration on salt minions. Read about it here
- **salt-minion** - client side daemon. Read about it here
- **salt-master** - master side daemon. Read about it here
- **salt-key** - master/minon public key management. Read about it here
- **salt-call** - the minion side implementation if the 'salt' executable, with some parameter differeneces. Read about it here
- **salt-run** - the "runner" executable, runs on the salt-master. Runners are saltstack extentions developed to perform any saltstack or extended functionality within the Salt framework. Read about it here
- **salt-cloud** - integration with backend cloud providers, runs on master, seamless minion bootstrapping with many cloud providers such as aws, rackspace, azure etc... Read about it here

The full list of Saltstack engines can be found here

Saltstack Runner (Conductor)

This page is somewhat of a launch page for navigating through other wiki documents related to the design and function of Conductor. Conductor is a runner extension module (or collection of modules), thus it is invoked using the ***salt-run*** engine. It is built to serve as an orchestration framework for end to end delivery of products, systems, services, and infrastructure using Salt.

At a high level, the Conductor runner is used to create, destroy or manage any number or configuration of cloud vm's using a data-driven model of configuration. It has 4 top tier actions. Create, upsize, downsize, destroy. Each action has multiple injection points, or orchestration hooks each with support for environment, product group/team, or role-specific tasks in addition to the built-in tasks it performs.

The 'create' action dynamically creates detailed instance, cluster or system cloud configuration files for a specified backend cloud provider, AWS in our case. Conductor 'create' uses ***salt-cloud*** to interface with the backend cloud provider.

These generated cloud configs specify all the metadata Salt and the Conductor runner will need to fully configure and manage the new vm's once they are online. ***Salt-cloud*** seamlessly takes care of all the Salt client bootstrapping, so when AWS (or other) is finished creating the vm's, ***salt-cloud*** bootstraps and applies the metadata we provided via the generated cloud configuration files on all new systems.

Once salt-cloud has completed its work, the process handle is returned back to Conductor. At which time, Conductor will continue with the remaining orchestration tasks leveraging it's ability to target specific minions for the purpose of salt state configuration and any other builtin or orchestration hooks tasks that have been defined and enabled.

Each step in the orchestration such as provisioning instance/s, salt bootstrapping, salt grain and other metadata configuring, installing software, deploying configuration, enabling and anything in between, are all using the same model and structured Pillar data so that nothing is duplicated and everything is re-usable throughout all phases and thereafter. Needless to say, the structured data model, the Conductor orchestrator, and all salt state configuration must follow the same design practices. Conductor is simply the user interface for driving the orchestration of various pieces of a framework. The three top-level pieces of the framework are

- Salt Pillar - structured yaml data driving the variable elements of all automation including systems configuration.
- Salt State - yaml and jinja templated implementation of salt's state and execution modules to configure all machine types and systems.
- Conductor - runner extension providing a model based orchestration and automation to create, destroy, upsize and downsize instances, clusters or systems, as well as setup and teardown cloud infrastructure such as vpc/subnet, load balancers, security groups etc....

This is an open-ended design that provides the conduit to create and manage any system.

What is Saltstack

What is Salt

Scalable, flexible, intelligent IT orchestration and automation for the software-defined data center.

The Salt platform was originally built as an extremely fast and powerful remote execution engine allowing users to execute commands on thousands of remote systems in milliseconds. All Salt functions leverage the asyncronous control of Salt remote execution and it is utilized by both agent-based Salt and agentless Salt SSH.

Spending 5-10 minutes on the page that is linked 'What is Salt' will surely provide better context for Salt, our implementation and the runner described here

Saltstack master side (runners and orchestration states)

add links to detailed documentation on salt-master side functionality

Conductor overview

The Conductor is a custom built Saltstack runner. It is an automation tool that provides the interface for executing automation tasks and managing salt minions through a Saltstack implementation Framework. It provides a consistent and scalable pillar model, consistent and scaleable state design, automation and orchestration for all phase of provisioning (cloud instance and associated cloud infrastructure), deployment and configuration management.

The Conductor salt runner, is actually a runner module, but loads submodules based on supported functionality.

Currently there are two top level modules:

- group
  - the group module provides support for Product Group based provisioning of instances and associated cloud infrastructure such as ELB's, Security groups.
- cloud
  - the cloud module provides support for core cloud infrastructure provisioning such as vpc, subnet, gateways etc...

The Conductor is designed to scale out and support multiple backend cloud providers such as AWS, Rackspace, Azure, Openstack etc... However, AWS is the only fully supported cloud provider.

This document will focus on the Group module of Conductor only. The Cloud module will be documented at a later time and is more of an Administrator module. The Group module however is tied much closer to the teams and product to be deployed.

Conductor has 4 action type (see Conductor In depth). The most complex of the type would be 'Create'.

The create action within the Conductor group module will perform the following high level tasks. There is lots of details behind this, but in general this is what happens when a 'create' action is executed:

- process command line arguments
- start a salt client connector on the salt master
- retrieve Pillar tree based on how Pillar is defined in salt-master config file (/etc/salt/master). We use gitfs and Github as the source for Pillar and state configuration.
- generate salt-cloud configuration files based on backed cloud provider (AWS)
- execute pre-provision hooks (see Orchestration hooks)
- invoke salt-cloud
- process return cloud data
- set any required cluster role grains
- tag any root volumes without tags
- run pre startup state hooks
- run salt states based on instance roles
- run post startup state hooks

Conductor In depth

The following links provide topic specific information about the Conductor salt orchestration runner.

Provisioning Types - describes the 3 top tier provisioning types supported by Conductor in detail. Instance/s, Clusters, Systems

Action Types - describes the 4 top tier actions supported by Conductor runner in detail. Create, Destroy, Upsize, Downsize

Orchestration Hooks - describes the insert steps available during the provisioning and deployment orchestration (aka pipeline), to support injecting environment, product group/team or role specific actions

Conductor Parallelization - describes the technique used to support multiple simultaneous executions of Conductor regardless of environment, product group, roles, provisioning types or shared configuration

Targeted teardown of instances, clusters and systems - describes the technical details, use cases and optional hooks supported in Conductor runner for tearing down cloud vm instances created with Conductor.

Pillar Model - The Framework specific pillar data model. Describes how to configure Pillar in a structured consistent way that scales and can be consumed by our automation.

State Design - The Framework specific state tree model. Describes how to create salt states in a structured consistent way that scales and can be consumed by our automation.

Related technical topics

add links to other Saltstack or implementation specific documentation/diagrams

...

Source Code

There are three repositories that hold the configuration and code our configuration management framework. We may have more depending on whether we build runner extensions or other custom tools that would not fit into the state and pillar repos.

- State repository
- Pillar repository
- Conductor Orchestration runner repository

The state and conductor runner repositories follow the more traditional develop branch and master/release branch model.

The Pillar repository follows a different model due to Salt design. It is used to hold the environment specific configuration for all environment that the Salt system serves. Thus it's branches are mapped to our defined 'environments'. Example, dev, qa, stage, uat, prod etc....

There is some tooling that can be used to populate a new branch (salt environment) with configuration that has been reset. Such things as passwords, accounts etc... that may be different per environment would be reset to a default value. This tooling is in its early stage, but essentially would need to be aware of the Pillar model we are using, so that we can define key value pairs of configuration that would be reset in a new branch.

Syntax Examples

See Action Types

Confusing Diagram

This is the first stab at showing the entire workflow of 'create' action in Conductor with some variable workflow decisions that are built into the framework. This will be revised to a much better diagram (hopefully).

SALTSTACK CONDUCTOR WORKFLOW

Paul Bruno | June 1, 2018