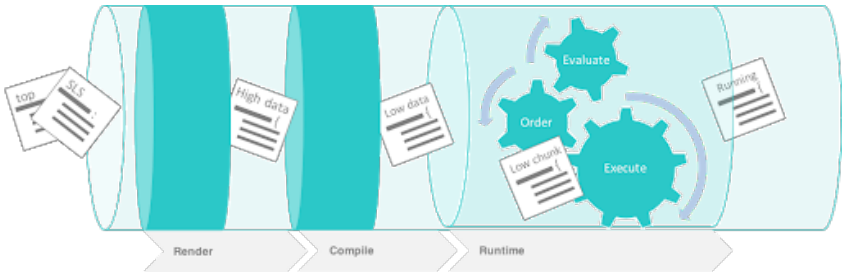
SALTSTACK

Definition
SaltStack, also known as Salt, is a **configuration management and orchestration** tool. It uses a central repository to provision new servers and other IT infrastructure, to make changes to existing ones, and to install software in IT environments, including physical and virtual servers, as well as the cloud.

Key Concepts
Minion: The client software that runs on the managed servers.
Master: The server that controls and manages the Minions.
Grains: System information collected by the Minion, like OS, IP, etc.
States: Declarative descriptions of system configuration.
Pillar: Pillars are used to store dynamic and sensitive information, such as passwords, API keys, and other configuration data. External data passed to Salt Minions.
Top File: Maps systems to specific Salt States.

Salt state processing



Installation
https://docs.saltproject.io/salt/install-guide/en/latest/topics/install-by-operating-system/rhel.html

EXECUTION MODULES LIST
https://docs.saltproject.io/en/latest/ref/modules/all/index.html

STATE LIST
https://docs.saltproject.io/en/latest/ref/states/all/index.html

Master
#!/bin/bash
sudo rpm --import https://repo.saltproject.io/salt/py3/redhat/9/x86_64/SALT-PROJECT-GPG-PUBKEY-2023.pub

curl -fsSL https://repo.saltproject.io/salt/py3/redhat/9/x86_64/latest.repo | sudo tee /etc/yum.repos.d/salt.repo
sudo yum install -y salt-master
sudo systemctl enable salt-master && sudo systemctl start salt-master

Minion
#!/bin/bash
sudo rpm --import https://repo.saltproject.io/salt/py3/redhat/9/x86_64/SALT-PROJECT-GPG-PUBKEY-2023.pub
curl -fsSL https://repo.saltproject.io/salt/py3/redhat/9/x86_64/latest.repo | sudo tee /etc/yum.repos.d/salt.repo
sudo yum install -y salt-minion
echo "master: 172.31.87.87" | sudo tee /etc/salt/minion.d/master.conf
echo "web01" | sudo tee /etc/salt/minion_id
sudo systemctl enable salt-minion && sudo systemctl start salt-minion

Basic Commands
<minion_selection> = 'web*' 'web01' …
salt <minion_selection> <command>
salt '*' test.ping = connectivity test between master and minions.
salt-key = list current machines states and change them.

- List = salt-key -l <status> or -L for ALL without status. (status can be un, acc, rej)
- Accept =salt-key <minion_selection> -a
- Reject = salt-key <minion_selection> -r
- Delete = salt-key <minion_selection> -d
  Note that; for accepting, rejecting, deleting ALL use upper case version of the flag. As an example; -A, -R, -D

salt <minion_selection> cmd.run '<cli_command>' = run cmd command.

salt <minion_selection> grains.items = gather static information about a minion, such as operating system details, hardware information, IP addresses, and more.
salt <minion_selection> grains.ls= Available grains can be listed
salt <minion_selection> saltutil.refresh_grains = refresh the grains.

salt <minion_selection> pillar.items = gather pillar information.
salt <minion_selection> saltutil.refresh_pillar = refresh the pillars.
Target by grains:

- sudo salt -G 'os:Ubuntu' test.ping
- salt -G 'cpuarch:x86_64' grains.item num_cpus
- salt -G 'ec2_tags:environment:*production*' (grains that are nested in a dictionary can be matched by adding a colon for each level that is traversed.)

State file

- State file contains a list of actions using salt modules that a user writes to perform on target minions that is under control by the master.
- Grain and pillars can be used on a state file for conditional module execution.
- The purpose of state file is to keep that data on file and not have to re-run them one by one via cli.

Salt master is configured with the correct file roots. The file roots are directories where Salt looks for SLS files. Check the 'file_roots' configuration in your Salt master configuration file (usually located at /etc/salt/master.d/roots.conf).

Example 'file_roots' configuration (yaml):

```
file_roots:
  base:
    - /srv/salt/base
  dev:
    - /srv/salt/dev
  qa:
    - /srv/salt/qa
pillar_roots:
  base:
    -/srv/salt/pillars/base
  dev:
    -/srv/salt/pillars/dev
  qa:
    -/srv/salt/pillars/qa
```

Feel free to adapt the environment names and directory paths based on your organizational or project requirements.

Top file:

- Defines the salt environment.
- Aggregates a list of minions and runs a list of states in the declared order or lexical order.
- Syncs grains, pillars, and states before running.
- Merge multiple environments to run large list of states.

To see is what is gonna be applied to the state:

They do not apply any changes to the system but gives you a detailed look at the compiled state data.

salt '*' state.show_highstate = more detailed

salt '*' state.show_lowstate = less detailed

salt '*' state.show_low_sls <state_selection> = less detailed with specific state selected.

Note: All the configuration files for the master should be located at /etc/salt/master.d/

Apply:
salt '*' state. highstate

salt 'minion_id' grains.setval <key> <value>
Exp: salt 'web*' grains.setval 'environment' 'dev' -> add a grain value.
salt 'web*' grains.setval 'roles' ['web', 'app1', 'dev'] -> add multiple values
salt 'minion_id' grains.delval 'environment' -> delete a grain value.
salt 'new_minion_id' saltutil.refresh_grains -> refresh the grains. (it does not affect the values that is added by you.)

JINJA… WHAT IS THIS?

It is a template engine for the sls files. It provides you to write your .sls files programmatically likewise other template engines (example popular template engines = ejs, hbs, blade).
https://docs.saltproject.io/salt/user-guide/en/latest/topics/jinja.html
https://jinja.palletsprojects.com/en/3.1.x/

Advanced States with Grains And Pillars

For the pillars you need top.sls files for each unlike the states file.
https://docs.saltproject.io/en/latest/topics/grains/index.html

TESTING

https://github.com/saltstack/kitchen-salt/tree/master

KitchenSalt allows users to replicate an independent infrastructure that is like the production environment locally and execute end-to-end validations against it.

KitchenSalt is built on top of Test Kitchen, which was initially developed to test and validate Chef workflows. It is a provisioner for SaltStack, which permits users to leverage Test Kitchen to validate the SaltStack environment locally without a Salt master or minions.

Testing the packages installed correctly (states applied correctcly) = Testinfra

Beacons
https://docs.saltproject.io/en/latest/topics/beacons/index.html

Beacons are typically enabled by placing a beacons: top level block in /etc/salt/minion or any file in /etc/salt/minion.d/ such as /etc/salt/minion.d/beacons.conf or add it to pillars for that minion. Note that adding the beacons to the pillars is the best practice.

inotify = to follow changes in the files.

salt-run state.event pretty=true -> listening for the events being recorded (event bus of the system)

Reactors
https://docs.saltproject.io/en/latest/topics/reactor/index.html

Reactor states should be kept in the organization. You must have configuration for reactor in the salt master.

File path: /etc/salt/master.d/reactor.conf (we should create this conf file and write the reactor configurations here)

Example reactor.conf (yaml):

```
reactor:
  - 'beacon_tag_id':
    - sls_file_path_to_run1
    - ...
```

NOTE: EVENT SYSTEM, REACTORS AND BEACONS for automatically accept and qualify the keys.

```
# File: base/orch/deploy.sls
{% set servers = salt['pillar.get']('servers', 'test') %}
{% set master = salt['pillar.get']('master', 'salt') %}
create_instance:
  salt.runner:
    - name: cloud.profile
    - prof: cloud-centos
    - provider: cloud
    - instances:
      - {{ servers }}
    - opts:
        minion:
          master: {{ master }}
```