Getting Started with Salt
Manage tens of thousands of servers and communicate with each system in seconds

Pablo Suárez Hernández
Senior Software Engineer SUSE Manager
SUSE Linux GmbH
psuarezhernandez@suse.com

Eric Jackson
Senior Software Developer Distributed Storage
SUSE Linux GmbH
ejackson@suse.com
Contents

- What is Salt? Why Salt?
- Salt Master and Minions
- Grains and Pillar
- States and Modules
- Salt Mine
- Runners
- Reactor
- Other Topics
What is Salt?

“A configuration management system which provides a high-speed communication working with large number of systems in a highly scalable environment”

- Remote execution engine
- Configuration management system
- Apply defined states
- Enable high-speed communication
- Provisioning and Orchestration
- Easy to extend. Salt Mine
- Python!
# Why Salt?

<table>
<thead>
<tr>
<th></th>
<th>Salt</th>
<th>Ansible</th>
<th>Chef</th>
<th>Puppet</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial release</strong></td>
<td>2011</td>
<td>2012</td>
<td>2009</td>
<td>2005</td>
</tr>
<tr>
<td><strong>Configuration Language</strong></td>
<td>YAML</td>
<td>YAML</td>
<td>Ruby/DSL</td>
<td>DSL</td>
</tr>
<tr>
<td><strong>Template Language</strong></td>
<td>Jinja2</td>
<td>Jinja2</td>
<td>ERB</td>
<td>ERB</td>
</tr>
<tr>
<td><strong>Core Language</strong></td>
<td>Python</td>
<td>Python</td>
<td>Ruby</td>
<td>Ruby</td>
</tr>
<tr>
<td><strong>Agent-less</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Why Salt?

“We had 10,000 lines of Puppet code and reduced that to roughly 1,000 in Salt” - Ryan Lane, Lyft

“We are running around 70,000 minions. When you keep doubling boxes like this, one does not simply scale” - Thomas Jackson, LinkedIn

“It is not about configuring Linux or Windows machine or configuring an operations or deployment. It is about development, testing, production and heterogeneity across all that.” - Ross Gardler, Microsoft Azure
How to run Salt?

- Checkout from GitHub: https://github.com/saltstack/salt/ (current release 2016.3.3)

- Install from SLE & openSUSE repositories: (2015.8.7 and 2016.3.2)

- Multiple running environments:
  - Master-less configuration
  - No agent needed!
  - master – minion
  - “dumb” devices (salt-proxy)
  - Multiple masters (syndic)
Using Salt without a master server (master-less)

“We can use Salt locally to execute commands and configurations”

```
# salt-call --local test.ping
# salt-call --local pkg.install apache2
# salt-call --local service.enable apache2
# salt-call --local state.apply mystate
```
No agent: salt-ssh

Master

Commands configuration

SSH

Client machine

No Agents!

Results
Salt Master and Minions

Ports: 4505-4506

Master

ØMQ

Minion

Minion

Minion
The Salt Transport: ØMQ

Salt uses ZeroMQ as default transport protocol:

PUB channel (4505 tcp):
- Implements a pub/sub socket. Encrypted payload is published by master which includes minion targeting

• REQ channel (4506 tcp):
- Implements a req/rep sockets. Fetching files and returning jobs from minions to master.

Also supports:
- TCP transport
- RAET transport
Salt in “dumb” devices (salt-proxy)
Salt with multiple masters (salt-syndic)
Master
Salt Master

The main daemon for managing the bus and components.

Installation:
zypper install salt-master

Configuration:
/etc/salt/master
/etc/salt/master.d/*.conf
Salt Master

Useful settings:

  log_level: warning|debug
  state_output: full|mixed

Files:

  /var/cache/salt/master/*
  /var/log/salt/master
  /run/salt/master
Setting up a Salt Master

- `/etc/salt/master`

**(single environment)**

```yaml
# Files and state roots
file_roots:
  base:
    - /srv/salt

# Pillar data roots
pillar_roots:
  base:
    - /srv/pillar
```

**(multiple environments)**

```yaml
file_roots:
  dev:
    - /srv/salt/dev/
  prod:
    - /srv/salt/prod/
  qa:
    - /srv/salt/qa/

pillar_roots:
  base:
    - /srv/pillar
  prod:
    - /srv/pillar/prod
  dev:
    - /srv/pillar/dev
```
Salt Minion

The agent running on each host.

Installation:
zypper install salt-minion

Configuration:
/etc/salt/minion
/etc/salt/minion.d/*.conf
/etc/salt/minion_id
Salt Minion

Useful settings:
  log_level: warning|debug
  master: salt

Files:
  /var/cache/salt/minion/*
  /var/log/salt/minion
  /run/salt/minion
Salt Minion

A minion is not part of the Salt cluster until the master accepts its key.

Useful commands (to run in Master):

- `salt-key -A`
- `salt-key -A -y`
- `salt-key -L`

```bash
# salt-key -L
Accepted Keys:
minion0
minion1
Denied Keys:
Unaccepted Keys:
minion2
Rejected Keys:
```
Starting playing with Salt

**Useful commands:**
- `salt '*' test.ping`
- `salt '*' cmd.run "date +%s.%N"`
- `salt-run jobs.active`
- `salt '*' saltutil.kill_job <jid>`
- `salt-run manage.up`
- `salt-run state.event pretty=True`

```bash
# salt '*' test.ping
minion0:
  True
minion1:
  True
```
Grains
Salt Grains

Data provided by the minion. Static for the life of the minion.

Commands
- salt 'node1*' grains.items
- salt 'node1*' grains.get kernelrelease

State file
{% set kernel = grains['kernelrelease'] | replace('-default', '') %}
kernel:
  cmd.run:
    - name: “echo Kernel is {{ kernel }}”
Salt Grains

Custom grains can override core grains.

Precedence:
/etc/salt/grains
/etc/salt/minion
/srv/salt/_grains

Caveats
• Not immediately available, salt '*' saltutil.sync_grains
Pillar
Salt Pillar

Pillar files are essentially YAML files with an sls extension. Centralized configuration data filtered to each minion by the master.

Location

- /srv/pillar
- /srv/pillar/top.sls
- Subdirectories do not affect data structure

/srv/pillar/ceph/cluster/node1.domain.sls
  cluster: ceph
  roles:
  - master
Salt Pillar

Display all pillar data for a specific minion.

Commands

```bash
salt 'node1*' pillar.items
node1.domain:
    ----------
    cluster:
        ceph
    roles:
        - master
```
Salt Pillar

Using pillar data.

Commands
   salt 'node1*' pillar.get roles

State file

tgt: {{ pillar.get('master_minion') }}
tgt: {{ salt['pillar.get']('master_minion') }}
{% set FAIL_ON_WARNING = salt['pillar.get']('FAIL_ON_WARNING', 'True') %}
{% for role in salt['pillar.get']('rgw_configurations', [ 'rgw' ]) %}
   mon_initial_members = {{ salt['pillar.get']('mon_initial_members') | join(', ') }}
{% endfor %}
Salt Pillar

Pillar data is not automatically available.

Commands

```
salt 'node1*' saltutil.pillar_refresh
    node1.domain:
        True

salt 'node1*' saltutil.refresh_pillar
    node1.domain:
        True
```
The Pillar “top.sls” file

- /srv/pillar/top.sls

  base: '*':
    - common

  environment

  targeting

  sls file

- /srv/pillar/common.sls

  vim:
    pkg.installed

  /etc/hosts:
    file.managed:
      -source: salt://hosts

- /srv/pillar/common.sls

  foo: bar

  {% if grains['os_family'] == "Suse" %}
  suseToken: token-only-in-suse-minions
  {% endif %}

“SLS files are JINJA2 templates rendering YAML output. We can render output using pillars, grains or even salt functions”
States
Salt States

States are synonymous with the Salt state file. The extension is sls.

Location:
/srv/salt/top.sls
/srv/salt/anything.sls
/srv/salt/project/thing1.sls
/srv/salt/project/thing2/init.sls

Caveat:
• Not all sls files are the same
Salt States

Template

/etc/ceph/ceph.client.admin.keyring:
  file.managed:
    - source:
      - salt://ceph/.../ceph.client.admin.keyring
    - user: root
    - group: root
    - mode: 600

Package and Service

lrbd:
  pkg.installed:
    - pkgs:
      - lrbd

enable lrbd:
  service.running:
    - name: lrbd
    - enable: True
Salt States

Includes

include:
  - .keyring

start mds:
  service.running:
    - name: ceph-mds@{{ grains['host'] }}
    - enable: True

No operation

# No pools are created by default
	nop:
    test.nop
Salt States

/srv/salt/anything.sls:
  salt '*' state.apply anything
  salt '*' state.sls anything

/srv/salt/project/thing1.sls:
  salt '*' state.apply project.thing1

/srv/salt/project/thing2/init.sls:
  salt '*' state.apply project.thing2
Salt States

Highstate can be considered a default setting for a salt cluster.

/srv/salt/top.sls:
  salt '*' state.highstate

Layman definitions:

• Highstate: simply processed, human friendly output
• Lowstate: “compiled”, Salt master friendly output
The States “top.sls” file

- /srv/salt/top.sls

- /srv/salt/common.sls

- /srv/salt/apache.sls

base: ‘*’:
  - common

‘webserver*’:
  - apache
  - webnode

‘G@os_family:SUSE’:
  - certs

vim:
  pkg.installed

/etc/hosts:
  file.managed:
    -source: salt://hosts

apache:
  pkg.installed: []
  service.running:
    - require:
      - pkg: apache
State Modules
State Modules

“Salt State modules allows you to define custom configuration states for almost every component of your system”

frank:
  mysql_user.present:
    - host: localhost
    - password: bobcat

server1:
  host.present:
    - ip: 192.168.0.42
    - names:
      - server1
      - florida
  /dev/sda:
    lvm.pv_present
  my vg:
    lvm.vg_present:
      - devices: /dev/sda
  lvroot:
    lvm.lv_present:
      - vgname: my_vg
      - size: 10G
      - stripes: 5
      - stripesize: 8K

- alias
- apache
- archive
- bower
- cloud
- disk
- dockerng
- file
- grafana
- htpasswd
- iptables
- mount
- lvm
- pkg
- mysql_user
- network
- rsync
- selinux
- service
- snapper
- sysctl
- supervisord
- virt
- user
- group
...
Salt State Module Example

```
/srv/salt/_states/mystatemodule.py

def __virtual__():
    return 'fwmod' if 'network.connect' in __salt__ else False

def check(name, port=None, **kwargs):
    ret = {'name': name, 'result': True, 'changes': {}, 'comment': ''}
    if 'test' not in kwargs:
        kwargs['test'] = __opts__.get('test', False)
    # check the connection
    if kwargs['test']:
        ret['result'] = True
        ret['comment'] = 'The connection will be tested'
    else:
        results = __salt__['network.connect'](host, port, **kwargs)
        ret['result'] = results['result']
        ret['comment'] = results['comment']
    return ret
```
Salt State Module Example

Commands

salt 'node1*’ saltutil.sync_states
salt 'node1*’ state.apply testgoogle

State file

testgoogle:
    fwmod.check:
        - name: "google.com"
        - port: 80
        - proto: 'tcp'
Execution Modules
Salt Execution Modules

“Execution modules allows you to manage and configure almost every component of your system”

<table>
<thead>
<tr>
<th>Command</th>
<th>Minion 0</th>
<th>Minion 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>salt '*' cmd.run date</code></td>
<td>Mon Sep 5 12:29:04 UTC 2016</td>
<td>Mon Sep 5 12:29:03 UTC 2016</td>
</tr>
<tr>
<td><code>salt '*' service.restart apache2</code></td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td><code>salt 'minion0' pkg.version vim</code></td>
<td>7.4.326-2.62</td>
<td></td>
</tr>
</tbody>
</table>
Salt Module

Python scripts that return useful values or perform custom actions.

Uses:
- Alter behavior of a state file (e.g. wait or retry)
- Encapsulate logic, simplify Jinja
- Retrieve dynamic information from minions

Commands
- `salt 'node1*' keyring.secret`
- `salt 'node1*' freedisks.list`
Salt Module Example

/srv/salt/_modules/keyring.py

def secret(filename):
    if os.path.exists(filename):
        with open(filename, 'r') as keyring:
            for line in keyring:
                if 'key' in line and ' = ' in line:
                    key = line.split(' = ')[1].strip()
                    return key

    key = os.urandom(16)
    hdr = struct.pack('<hih', 1, int(time.time()), 0, len(key))
    return base64.b64encode(hdr + key)

def file(component, name=None):
    if component == "osd":
        return "/srv/salt/ceph/osd/cache/bootstrap.keyring"

    if component == "igw":
        return "/srv/salt/ceph/igw/cache/ceph." + name + ".keyring"

    if component == "mds":
        return "/srv/salt/ceph/mds/cache/" + name + ".keyring"

    if component == "rgw":
        return "/srv/salt/ceph/rgw/cache/" + name + ".keyring"
Salt Module Example

Commands
- `salt 'node1*' saltutil.sync_modules`
- `salt 'node1*' keyring.secret`
- `salt 'node1*' keyring.file mds node1`

State file
```yaml
{% set keyring_file = salt['keyring.file']['mds', grains['host']] %}
{{ keyring_file}}:
  file.managed:
    - source:
    ...
    - context:
      secret: {{ salt['keyring.secret']|(keyring_file) }}
```
Mines
Salt Mine

Cache the output of a module. Complements grains.

Few steps:
• Write a module
• Synchronize to minions
• Configure minions

Commands
    salt 'node1*' freedisks.list
    salt-call mine.get 'node1*' freedisks.list
    salt 'node1*' mine.get 'node*' freedisks.list
Salt Mine Example

/srv/salt/_modules/freedisks.py

def list():
    drives = []
    for path in glob('/sys/block/*/device'):
        base = os.path.dirname(path)
        device = os.path.basename(base)
        # Skip partitioned drives
        partitions = glob(base + '/'+ device + '*')
        if partitions:
            continue
        # Skip removable media
        removable = open(base + '/removable').read().rstrip('
')
        if (removable == '1'):
            continue
        rotational = open(base + '/queue/rotational').read().rstrip('
')
        hardware = _hwinfo(device)
        hardware['device'] = device
        hardware['rotational'] = rotational
        drives.append(hardware)
    return drives

# Skip removable media
    removable = open(base + '/removable').read().rstrip('
')
    if (removable == '1'):
        continue
    rotational = open(base + '/queue/rotational').read().rstrip('
')
    hardware = _hwinfo(device)
    hardware['device'] = device
    hardware['rotational'] = rotational
    drives.append(hardware)
return drives
def _hwinfo(device):
    results = {}
    cmd = "/usr/sbin/hwinfo --disk --only /dev/{}/".format(device)
    proc = Popen(cmd, stdout=PIPE, stderr=PIPE, shell=True)
    for line in proc.stdout:
        m = re.match("  \([^\:]+\): (.*)", line)
        if m:
            if (m.group(1) == "Capacity"):
                c = re.match("(\d+ \w+) \((\d+) bytes\)", m.group(2))
                if c:
                    results[m.group(1)] = c.group(1)
                    results['Bytes'] = c.group(2)
                else:
                    # Remove double quotes
                    results[m.group(1)] = re.sub(r"", '', m.group(2))
    return results
Salt Mine Example

24: IDE 200.0: 10600 Disk
[Created at block.245]
...
Model: "HGST HTS721010A9"
Vendor: "HGST"
Device: "HTS721010A9"
Revision: "A3J0"
Serial ID: "JR100X6P2PWRME"
Device File: /dev/sdc
...
Size: 1953525168 sectors a 512 bytes
Capacity: 931 GB (1000204886016 bytes)
Config Status: cfg=new, avail=yes, need=no, active=unknown
Attached to: #14 (SATA controller)
Salt Mine Example

/srv/salt/_modules/freedisks.py

Sync'ing:
• salt '*' saltutil.sync_module
• salt '*' saltutil.sync_all

State file
load modules:
module.run:
  - name: saltutil.sync_all
  - refresh: True
Salt Mine Example

/srv/salt/.../mine_functions/init.sls:

configure_mine_functions:
  file.managed:
    - name: /etc/salt/minion.d/mine_functions.conf
    - source: salt://ceph/mine_functions/files/mine_functions.conf

manage_salt_minion_for_mine_functions:
  service.running:
    - name: salt-minion
    - watch:
      - file: configure_mine_functions
Salt Mine Example

/etc/salt/minion.d/mine_functions.conf

mine_functions:
  freedisks.list: []
Runners
Salt Runner

A runner is conceptually similar to a stand-alone script. Unlike state files or modules, the runner executes on the master.

Uses:

• Query salt state
  salt-run jobs.active
  salt-run manage.up
• Updating external systems
  salt-run queue.insert myqueue abc
  salt-run network.wol 52:54:00:1D:62:F7
Salt Runner

Uses Continued:

- Coordinate complex states across groups of minions
  
  salt-run state.orchestrate orch.gateways

- Custom runners
  
  salt-run filequeue.add queue=master complete
  
  salt-run validate.pillar

- Jinja conditionals
  
  {% if salt['saltutil.runner']('filequeue.check', name='complete', queue='master') == True %}
  
  True
  
{% endif %}
Salt Runner: state.orchestrate

The state.orchestrate runner has a different sls file format that allows targeting of minions.

installation:
salt.state:
  - tgt: 'I@roles:gateways'
  - tgt_type: compound
  - sls: gateways.packages
Salt Runner: state.orchestrate

Imagine a redundant gateway setup where the application retrieves the configuration from an external source.
Salt Runner: state.orchestrate

The dilemma is the configuration only needs to be stored once for all gateways. The import command is only available after package installation.
Salt Runner: state.orchestrate

The manual steps are

• Install packages on each gateway
  zypper in mypackage
• Initialize the external source
  Import configuration
• Start application on each gateway
  systemctl enable myservice
  systemctl start myservice
Salt Runner: state.orchestrate

The salt states are

- **Install packages**
  - *zypper in mypackage*
- Configure packages
  - Import configuration
- Start application
  - systemctl enable myservice
  - systemctl start myservice

```
/srv/salt/gateways/package/init.sls
```

installation:

pkg.installed:
  - pkgs:
    - mypackage
Salt Runner: state.orchestrate

The salt states are

• Install packages
  zypper in mypackage
• **Configure packages**
  Import configuration
• Start application
  systemctl enable myservice
  systemctl start myservice

```
/srv/salt/gateways/import/init.sls

configure:
  cmd.run:
    - name: "client-cli {{ salt['pillar.get']('mysettings') }}"
    - shell: /bin/bash
```
Salt Runner: state.orchestrate

The salt states are

- Install packages
  zypper in mypackage
- Configure packages
  Import configuration
- **Start application**
  systemctl enable myservice
  systemctl start myservice

```
/srv/salt/gateways/start/init.sls

enable:
  service.running:
    - name: myservice
    - enable: True

restart:
  module.run:
    - name: service.restart
    - m_name: myservice
```
Salt Runner: state.orchestrate

/srv/salt/orch/gateways.sls:

installation:
  salt.state:
    - tgt: "@roles:gateways"
    - tgt_type: compound
    - sls: gateways.package

configure:
  salt.state:
    - tgt: "{{ salt.saltutil.runner('select.one_minion', roles='gateways') }}"
    - sls: gateways.import

start:
  salt.state:
    - tgt: "@roles:gateways"
    - tgt_type: compound
    - sls: gateways.start
Salt Runner: state.orchestrate

To execute the orchestration runner

salt-run state.orchestrate orch.gateways
salt-run state.orch orch.gateways
Reactor
Salt Reactor

Listens for salt events and triggers an action

Examples:
• Putting all minions into highstate
• Coordinating actions with reboots
Salt Reactor

Reactor is part of the salt master. The configuration only allows simple globbing.

/etc/salt/master.d/reactor.conf

reactor:
  - 'salt/minion/*/start':
    - /srv/salt/reactor/highstate.sls
Salt Reactor

Reactor sls files are different.

/srv/salt/reactor/highstate.sls

highstate_run:
  cmd.state.highstate:
    - tgt: {{ data['id'] }}
Salt Reactor

Reactors sls files are different.

/srv/salt/reactor/highstate.sls

highstate_run:
  cmd.state.highstate:
    - tgt: {{ data['id'] }}
Salt Reactor

Caution:

• No circular dependency checking across events
  – Reactor starts an orchestration which results in firing events that starts an orchestration...

• No synchronization primitives
  – Tasks run by the master are separate processes. Race conditions will happen

• No state machine
  – An error stops the workflow. No automatic retries.
Other Topics
Other Topics

Topics not covered:

• Queues
• Tornado
• Thorium
• Engines
• Beacons
• Salt-api
• Renderers
Salt Projects

SUSE Manager:
- https://www.suse.com/products/suse-manager

SUSE Software Defined Storage:
- Ceph
Recommended Sessions

Hear from a Customer:
SUSE Manager 3 & Salt at Tyson Foods (CAS91938) - Weds 3pm & Fri 10:15am

Try a Hands-on Session:
SUSE Manager for Smarties (HO91268) – Tues 2pm & Weds 4:45pm
Advanced Hands-on with SaltStack (HO91449) – Weds 10am & Thurs 2pm
Extending Salt for Fun and Profit (HO92263) – Tues 4:30pm & Thurs 2pm

Learn More:
Managing Configuration Drift and Auditing with Salt (TUT89994) – Weds 2pm
Questions?