Salt

A new approach to infrastructure management

Martin Höfling Munich, 2014-04-01

TNG TECHNOLOGY CONSULTING

Overview

Introduction

First Steps

Salt State System

Beyond Salt States

What is Salt?

Salt ...

- ... started as remote execution environment
- ... provides configuration management
  - with dynamic reconfiguration
- ... enables cloud provisioning

Salt is ...

- ... written in Python
- ... OpenSource (Apache License)
- ... available for all major platforms

Competitors / Alternatives

- Puppet
- Chef
- Ansible
- Cfengine
- ...

Salt is based upon...

- 2.6 ≤ python-version < 3.0
- ZeroMQ
- YAML (default markup language)
- Jinja2 (default template engine)

How to get started?

- Salt Documentation: http://docs.saltstack.com/
- IRC: Freenode at #salt
- Code: https://github.com/saltstack

Overview

Introduction

First Steps

Salt State System

Beyond Salt States

Configuration Management Set-ups

- Standalone
- Master - Minion ("default")
- Multi Master / Cascades

Connect Master and Minion

Minion Config

```
master: master-hostname-or-ip
id: minion-name #(optional)
grains: ... #(optional)
```

Master Config

```
file_roots:
  base:
    - /vagrant/base/salt
pillar_roots:
  base:
    - /vagrant/base/pillar
```

When the daemons are running...

... use salt-key to accept minion on master.

Remote Execution

```
root@precise64:~# salt '*' test.ping
minion:
    True
minion-manual:
    True
```

salt parameters:

- '*': target
- test.ping: module.function
- function parameters (optional)

Example with parameters

```
root@precise64:~# salt 'minion-manual' group.add tng 3456
minion-manual:
    True
```

Full list of built-in modules.

Node Targeting via Grains

Grains are static bits of information that a minion collects about the system when the minion first starts.

Minion Config

```
master: 192.168.50.49
id: minion-manual
grains:
      node_location: ufo
```

Targeting minions via Grains

```
root@precise64:~# salt -G 'node_location:ufo' test.ping
minion-manual:
    True
```

Listing Grains

```
root@precise64:~# salt '*' grains.items
```

Salt Pillar ...

... (not only) for sensitive data.

Pillar is an interface to offer global values distributed to specific minions.

Pillar should be used to store sensitive data!

Pillar data is only available for the targeted minion specified by the matcher type.

Sample Pillar Data

pillar/top.sls

```
base: # the environment for which the pillar data is valid
  'minion': # superusers information is only transferred to minion
    - superusers
  'minion-manual': # special-superusers is only transferred to minion-ma
    - special-superusers
```

pillar/superusers.sls

```
superusers:
  - martin
  - willi
```

pillar/special-superusers.sls

```
superusers:
  - martin
  - otto
```

Overview

Introduction

First Steps

Salt State System

Beyond Salt States

Salt States: Simple Example

example.sls (YAML)

```
mc: # name parameter
  pkg: # state module
    - latest # state function

the-vim-package: # id
  pkg.latest: # state module + state function
    - name: vim # name parameter
    - refresh: True # keyword parameter refresh
```

Apply State

```
root@precise64:~# salt 'minion' state.sls example
```

The Code:

salt/states/pkg.py

Salt States: More Complex Example

locale.sls

```
en_US.UTF-8:
  locale.system

locales:
  pkg:
    - latest

/etc/locale.gen:
  file.managed:
    - require:
      - pkg: locales
    - contents: |
        en_US.UTF-8 UTF-8

/usr/sbin/locale-gen:
  cmd.wait:
    - watch:
      - file: /etc/locale.gen
```

Salt States: Jinja2 Templating

example-jinja.sls

```
{% for package in ['vim', 'mc']%}
{{ package }}:
  pkg:
    - latest
{% endfor %}
```

Salt States: Using Grain Information

example-grains.sls

```
apache:
  pkg.installed:
    {% if grains['os'] == 'RedHat' %}
    - name: httpd
    {% elif grains['os'] == 'Ubuntu' %}
    - name: apache2
    {% endif %}
```

Salt States: Using Pillar Information

```
include:
  - example-users

{% for username in salt['pillar.get']('superusers') %}
usermod {{ username }}:
  cmd.run:
    - name: usermod --append --groups sudo {{ username }}
    - require:
      - sls: example-users
{% endfor %}
```

Collection of States: The High-state

Sample top.sls file

```
base:
  '*':
    - locale

  'minion':
    - example

  'minion-manual':
    - example-pillar
```

Highstate is executed via:

```
root@precise64:~# salt '*' state.highstate
```

Overview

- Introduction
- First Steps
- Salt State System
- Beyond Salt States

Provisioning

- Vagrant plugin (testing the deployment)
- Salt Cloud (manage cloud instances)

Salt Cloud: Loadtest example

Cloud Configuration /etc/salt/cloud

```
providers:
  ec2-europe:
    minion:
      master: ip-1-2-3-4
    ssh_interface: private_ips
    id: ABCDEFGHIKJL   # EC2 api_access_key_id
    key: 'W+ABCDEFGHIKJL'  # EC2 api_secret_access_key
    keyname: mapconv-cloud
    securitygroup: Loadtest
    private_key: /srv/cloud/mapconv-cloud
    provider: aws
    location: eu-west-1
    availability_zone: eu-west-1b
```

Salt Cloud: Loadtest example

Profile Configuration /etc/salt/cloud.profiles

```
websocket-backend:
  provider: ec2-europe
  image: ami-eecd2899
  size: Small Instance
  ssh_username: ubuntu
  sudo: True
  script: Ubuntu
  grains:
    mapconv_type: websocket-backend

stable-instance:
  provider: ec2-europe
  image: ami-78cf2a0f
  size: Medium Instance
  ssh_username: ubuntu
  sudo: True
  script: Ubuntu
  grains:
    mapconv_type: all-in-one
```

Salt Cloud: Loadtest example

Infrastructure map file loadtest.map

```
Example map file loadtest.map
mongodb:
  - mongodb.loadtest
redis:
  - redis.loadtest
loadbalancer:
  - loadbalancer.loadtest
http-backend:
  - http-backend-1.loadtest
  - http-backend-2.loadtest
    [...]
  - http-backend-8.loadtest
websocket-backend:
  - websocket-backend-1.loadtest
  - websocket-backend-2.loadtest
    [...]
  - websocket-backend-8.loadtest
```

Setup Cloud Instances

```
salt-cloud -m loadtest.map
```

Salt Reactor System

Master Configuration

```
# register reaction on host_announce
reactor:
  - 'host_announce':
    - /srv/reactor/host_announce.sls
```

host_announce.sls

```
distribute_hosts:
  cmd.hosts.add_host:
    - tgt: '*'
    - arg:
      - {{ data['data']['ip'] }}
      - {{ data['data']['hostname'] }}
```

Triggered on minion

```
salt-call event.fire_master '{"ip":"1.2.3.4", "hostname":"newminion"}' 'h
```

Further Features

- Peer communication (Minion to minion)
- Salt Mine (dynamic gathering of minion data)
- File Server (also via git)
- Returners (MongoDB, MySQL, Syslog, ...)
- Master Cascades / Passthrough Minion (Salt Syndic)
- Scheduling
- Basic UI (Halite)

THE END

Questions?