Advanced Hands-On Training with SaltStack

Agenda

Master configuration options

- GITFS setup
- Lab init
- Custom modules
- Engine configuration settings

Runners

- Runner introduction
- xMatters runner demo
- Custom runner

Engines

- Engine introduction
- Slack engine runner

Reactors

- Reactor introduction
- Configuration settings

Beacons

- Beacon introduction
- Pillar configuration
- Deployment/testing

Orchestration runner

- Orchestration introduction
- System deployment orchestration

Advanced Hands-On Training with SaltStack

Master configuration options

GITFS

The gitfs backend allows Salt to serve files from git repositories. It can be enabled by adding git to the fileserver_backend list, and configuring one or more repositories in gitfs_remotes. When using gitfs backend each branch and tag becomes a salt environment.

Lab Setup:

Step 1:

Create /etc/salt/master.d/fileserver.conf in a text editor and add the following content.

```
fileserver_backend:
  - roots
  - git

gitfs_remotes:
  - https://github.com/trebortech/suseconf16.git
```

Step 2:

To enable these new settings the master must be restarted.

```
service salt-master restart
```

Step 3:

Now verify that the files are being pulled from gitfs.

```
salt-run fileserver.file_list
```

Advanced Hands-On Training with SaltStack

Additional helpful commands for GITFS

To force the update for fileservers backend.

```
salt-run fileserver.update
```

To debug fileserver connection if files are not being listed put the salt-master into debug mode.

```
service salt-master stop
salt-master -ldebug
```

Advanced Hands-On Training with SaltStack

Lab init

For the lab a state file has been created to load files and configuration into specific directories to support the follow-on labs. Over the course of this session we will review most of these files.

Step 1

The salt master's minion key must be accepted.

```
salt-key -a master1
```

Step 2

To initialize the lab files.

SALT MASTER ONLY

```
salt-call state.sls salt.init.advance_lab1
```

Advanced Hands-On Training with SaltStack

Custom modules

To extend out the current functionality of SaltStack the use of custom modules might be necessary. One way to use custom modules is put the modules into a specific directory and configure SaltStack to load those modules on startup.

The lab initialization script has put the necessary configuration files in place and should be ready to use.

/etc/salt/master.d/base.conf

```
extension_modules: /srv/modules
```

ls /srv/modules

```
engines
runners
reactors
```

Advanced Hands-On Training with SaltStack

Engine configurations

Engines are python processes that are managed by the SaltStack master service. Several engines are available and during our demo we will utilize custom engines located in our modules directory. To enable an engine the specific engine configurations must be put into the master config or the master.d directory.

/etc/salt/master.d/sdb.conf

```
saltconfig:
  driver: sqlite3
  database: /srv/sdb/saltconfig.sqlite
  table: creds
  create_table: True
```

/etc/salt/master.d/engines.conf

```
engines:
  twilio:
      account_sid: sdb://saltconfig/creds?twilio-sid
      auth_token: sdb://saltconfig/creds?twilio-token
      twilio_number: sdb://saltconfig/creds?twilio-number
      interval: 10

  xmatters:
      server: saltstack.na5.xmatters.com
      api: api/integration/1/functions
      uuid: sdb://saltconfig/creds?xmatters-uuid
      user: test
      password: user
      sslVerify: False
```

Advanced Hands-On Training with SaltStack

Runners

Introduction

Salt runners are convenience applications executed with the salt-run command. Salt runners work similarly to Salt execution modules however they execute on the Salt master itself instead of remote Salt minions. Our lab is going to utilize a custom runner that is located in our custom modules directory.

/srv/modules/runners/xmatters.py

```
'''
xMatter runner


:config example:
engines:
  xmatters:
    server: demo.na2.xmatters.com
    api: api/integration/1/functions/
    uuid: 2e....2c-d..b-4..50-9...b-793.....af
    user:
    password:
    sslVerify: False


:depends:
    requests
    json


'''
from __future__ import absolute_import
import logging

log = logging.getLogger(__name__)

try:
    import json
    import requests
    HAS_IMPORTS = True
except ImportError:
```


```
    HAS_IMPORTS = False


def __virtual__():
    if HAS_IMPORTS:
        return True
    return (False, 'The xMatters runner cannot start')


def _send_server(method='POST', opts=None, data=None):

    if opts is None:
        opts = __opts__
    config = opts['engines']['xmatters']
    server = config['server']
    api = config['api']
    uuid = config['uuid']
    user = config['user']
    password = config['password']
    sslVerify = config['sslVerify']
    headers = {'content-type': 'application/json'}

    xm_url = 'https://{0}/{1}/{2}/triggers'.format(
        server,
        api,
        uuid)

    if method == 'POST':
        requestset = requests.post(xm_url,
                                   auth=(user, password),
                                   verify=sslVerify,
                                   data=json.dumps(data),
                                   headers=headers)
    elif method == 'DELETE':
        requestset = requests.delete(xm_url,
                                     auth=(user, password),
                                     verify=sslVerify,
                                     data=json.dumps(data),
                                     headers=headers)
    elif method == 'GET':
        requestset = requests.get(xm_url,
                                  auth=(user, password),
                                  verify=sslVerify,
                                  data=json.dumps(data),
                                  headers=headers)

    return requestset
```

Advanced Hands-On Training with SaltStack


```
def create_event(eventid, priority, message, minionid):
    host_payload = {}
    host_payload['eventid'] = eventid
    host_payload['priority'] = priority
    host_payload['message'] = message
    host_payload['minion id'] = minionid

    ret = _send_server(data=host_payload)

    if ret.ok:
        return "Message sent"
    else:
        return "Error in sending message"
```

xMatters runner demo

The xMatters runner is a very simple runner that we will utilize during our lab. xMatters is a SaaS service that provides messaging services to our application. The current runner only has one method configured and that is to send a message to the configured account. This runner will utilize the REST API service provided by xMatters.

```
salt-run xmatters.create_event eventid=1 priority=1 message='test from me' minionid='minion1'
```

/srv/modules/runners/xmatters.py

```
def create_event(eventid, priority, message, minionid):
    ….
```

Advanced Hands-On Training with SaltStack

Engines

Introduction

Salt Engines are long-running, external system processes that leverage Salt. Salt engines enhance and replace the external processes functionality. Our lab is going to utilize a custom engine that is located in our custom modules directory.

Advanced Hands-On Training with SaltStack

Slack Engine

The slack engine is a great addition to the slack execution module and adds the ability to monitor slack channels. When a message is entered into any slack channel and starts with a “!” the message will be processed. Depending on your engine configuration will depend on if the command can execute.

Slack engine configuration
/etc/salt/master.d/engines.conf
MASTER ONLY
APPEND TO EXISTING FILE

```
engines:
  slack:
      token: sdb://saltconfig/creds?slack
      control: True
      valid_users:
        - rob
      valid_commands:
        - help
        - test.ping
        - state.highstate
      aliases:
        get_device:
          type: runner
          cmd: zenoss.find_device
          help: "get_device device='demobox1'"
        send_event:
          type: runner
          cmd: zenoss.send_event
          help: "send_event summary='Working on this' device='demobox1' severity='Info'"
```

Advanced Hands-On Training with SaltStack

Reactors

Introduction

Salt's Reactor system gives Salt the ability to trigger actions in response to an event. It is a simple interface to watching Salt's event bus for event tags that match a given pattern and then running one or more commands in response.

This system binds sls files to event tags on the master. These sls files then define reactions. This means that the reactor system has two parts. First, the reactor option needs to be set in the master configuration file. The reactor option allows for event tags to be associated with sls reaction files. Second, these reaction files use highdata (like the state system) to define reactions to be executed.

/srv/modules/reactors/twil.sls

```
{% set hal = False %}
{% if "world" in data['body'] %}
{% set mymessage = "I'm sorry, Rob. I'm afraid I can't do that." %}
{% set hal = True %}
{% elif data['images'][0] is defined %}
{% set mymessage = "Message from: *" + data['from'] + "* --> *" + data['body'] + "* " +
data['images'][0] %}
{% else %}
{% set mymessage = "Message from: *" + data['from'] + "* --> *" + data['body'] + "*" %}
{% endif %}

{% if hal %}
'Alert Zenny':
  local.state.sls:
    - tgt: "zenny"
    - expr_form: compound
    - arg:
      - zenny.message
    - kwarg:
        pillar:
          color: 'red'
          message: "{{ mymessage }}"

{% else %}
'Slack notify':
  local.state.sls:
    - tgt: 'master1'
    - expr_form: compound
```


```
    - arg:
      - slack.blast
    - kwarg:
        pillar:
          mymessage: "{{ mymessage }}"
{% endif %}
```

Advanced Hands-On Training with SaltStack

Beacons

Introduction

Beacons let you use the Salt event system to monitor non-Salt processes. The beacon system allows the minion to hook into a variety of system processes and continually monitor these processes. When monitored activity occurs in a system process, an event is sent on the Salt event bus that can be used to trigger a reactor.

Advanced Hands-On Training with SaltStack

Pillar configuration

Salt beacons do not require any changes to the system components that are being monitored, everything is configured using Salt. 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. The beacon system, like many others in Salt, can also be configured via the minion pillar, grains, or local config file.

/srv/pillar/top.sls

```
base:
  '*':
    -   beacons.logins
```

/srv/pillar/beacons/logins.sls

```
beacons:
  btmp: {}
```

Reset minion

MINION ONLY

```
salt-call pillar.items
service salt-minion restart
```

MASTER ONLY

```
salt-run state.event pretty=true tagmatch='*'
```

Advanced Hands-On Training with SaltStack

Orchestration runner

Introduction

The orchestrate runner generalizes the Salt state system to a Salt master context. Whereas the state.sls, state.highstate, et al functions are concurrently and independently executed on each Salt minion, the state.orchestrate runner is executed on the master, giving it a master-level view and control over requisites, such as state ordering and conditionals. This allows for inter minion requisites, like ordering the application of states on different minions that must not happen simultaneously, or for halting the state run on all minions if a minion fails one of its states.

Demo Overview

beacon ->
    eventbus ->
        reactor ->
            orchestration file ->
                xMatters text
                Slack state on minion
                zenny alert

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

```
reactor:
  - 'salt/beacon/*/btmp/':
    - /srv/modules/reactors/logins.sls
```

/srv/modules/reactors/logins.sls

```
# reactor for logins
{% set minionid = data['data']['id'] %}
{% set username = data['data']['user'] %}
{% set fromhost = data['data']['hostname'] %}

{% set color = 'green' %}
{% set message = username + ' failed to log into ' + minionid %}
```


```
invoke_orchestrate_file:
  runner.state.orchestrate:
    - mods: demo.logins
    - pillar:
        minionid: {{ minionid }}
        username: {{ username }}
        fromhost: {{ fromhost }}
        color: {{ color }}
        message: {{ message }}
```

/srv/salt/demo/logins.sls

```
{% set minionid = pillar.get('minionid', '') %}
{% set username = pillar.get('username', '') %}
{% set fromhost = pillar.get('fromhost', '') %}
{% set color = pillar.get('color', '') %}
{% set message = pillar.get('message', '') %}


"Send text via xMatters":
  salt.runner:
    - name: xmatters.create_event
    - eventid: 1
    - priority: 1
    - message: {{ message }}
    - minionid: {{ minionid }}

"Send alert to Slack":
  salt.state:
    - tgt: 'master1'
    - sls:
      - slack.blast
    - pillar:
        mymessage: {{ message }}

'Send update to zenny':
  salt.state:
    - tgt: 'zenny'
    - arg:
      - zenny.message
    - kwarg:
        pillar:
```


```
color: {{ color }}
message: {{ message }}
```