Demonstration content JDL 20170420

Before getting started

Deploy 3 VM and write down their IP:

- master01 (prompt vert) : 10.1.1.____
- minion01 1 CPU 1 GB RAM (prompt jaune): 10.1.1.____
- minion02 2 CPU 1.5 GB RAM (prompt bleu): 10.1.1.____

```
Green
export PS1="\[$(tput bold)\]\[$(tput setaf 2)\][\u@\h \W]\\$ \[$(tput sgr0)\]"
Yellow
export PS1="\[$(tput bold)\]\[$(tput setaf 3)\][\u@\h \W]\\$ \[$(tput sgr0)\]"
Blue
export PS1="\[$(tput bold)\]\[$(tput setaf 6)\][\u@\h \W]\\$ \[$(tput sgr0)\]"
```

Colorize: https://www.kirsle.net/wizards/ps1.html

Add node names in /etc/hosts on each node

For demo disable firewall on master: systemctl stop firewalld

Installation + config

- master and minions

```
yum install https://repo.saltstack.com/yum/redhat/salt-repo-latest-1.el7.noarch.rpm
```

- install salt-master RHEL + config

```
yum -y install salt-master
```

/etc/salt/master

```
file_roots:
  base:
    - /srv/salt/states

pillar_roots:
  base:
    - /srv/salt/pillars
```

- install salt-minion RHEL + config

`yum -y install salt-minion`

/etc/salt/minion

```
master: master01
```

- accept salt-minion keys : `salt-key -a minion*`
- `salt '*' test.ping`
- `salt '*' grains.items`
- targetting based on grain: `salt -G 'mem_total:988' test.ping` (only one should return)
- `salt -G 'num_cpus:1' test.ping`

Remote execution

- remote execution
    - cmd.run
    - iptables.version
    - selinux.getenforce (will show something is missing, install pkg in next step)
    - salt 'minion01' pkg.install policycoreutils-python
    - selinux.getenforce again

Configuration Management

- Create states and pillars directories

`mkdir -p /srv/salt/{states,pillars}`

- manage /etc/motd

/srv/salt/states/motd/init.sls

```
motd:
  file.managed:
    - name: /etc/motd
    - source: salt://motd/motd.jinja
    - template: jinja
```

/srv/salt/states/motd/motd.jinja

```
Hello world
```

/srv/salt/states/top.sls

```
base:
  '*':
    - motd
```

Verif : salt '*' state.show_top

```
minion01:
    ----------
    base:
        - motd
```

`salt '*' state.highstate -v test=True`

`salt '*' state.highstate`

- reminder: `grains.items` and `grains.item grain_name`
- jinja template: use `os` or `mem_total` grains
- declare custom grain with multiple values

```
salt 'minion01' grains.setval ROLE ['frontend','haproxy']
salt 'minion02' grains.setval ROLE ['backend','apache']
```

- use custom grain in MOTD and display raw result first then prettify it with jinja logic

```
Roles :
{% for i in grains['ROLE'] -%}
- {{ i }}
{% endfor -%}
```

- manage service.running postfix

Manage conf (retrieve /etc/postfix/main.cf and add comment at the top)

Manage service without watch statement (will tell service is already running)

Run `tail -f /var/log/maillog`

Add a watch statement and make a change to conf and state.highstate

```
postfix-conf:
  file.managed:
    - name: /etc/postfix/main.cf
    - source: salt://postfix/main.cf.jinja
    - template: jinja

postfix-service:
  service.running:
    - name: postfix
    - enable: True
    - reload: False
    - watch:
      - file: postfix-conf
```

- show content of /var/cache/salt/minion/files/base on minion (cache code)

Pillars

After showing how code is stored on minion and presents security issues, show use of pillars

Put a password in clear text in conf main.cf.jinja and show file in cache again

Now create a pillar

/srv/salt/pillars/postfix/init.sls

```
postfix:
  password: Redlfmskdmlfs
```

/srv/salt/pillars/top.sls

```
base:
  'minion01':
     - postfix
```

salt '*' pillar.data

Replace hardcoded "password=Truc" by "password={{ pillar['postfix']['password'] }}"

Show both syntaxes

```
password={{ pillar['postfix']['password'] }}
password={{ salt['pillar.get']('postfix:password') }}
```

Simple syntax gives error if pillar doesn't exist, other syntax is better

Show on a pillar that doesn't exist:

```
password={{ salt['pillar.get']('postfix:password2', 'valeur par defaut') }}
```

Event bus and reactors

Show events on master bus: `salt-run state.event pretty=True`

Run: `salt 'minion01' test.ping -v`

```
salt/job/20170411144701244715/new   {
    "_stamp": "2017-04-11T18:47:01.245539",
    "arg": [],
    "fun": "test.ping",
    "jid": "20170411144701244715",
    "minions": [
        "minion01"
    ],
    "tgt": "minion0",
    "tgt_type": "glob",
    "user": "root"
}
salt/job/20170411144701244715/ret/minion01  {
    "_stamp": "2017-04-11T18:47:01.351686",
    "cmd": "_return",
    "fun": "test.ping",
    "fun_args": [],
    "id": "minion0",
    "jid": "20170411144701244715",
    "retcode": 0,
    "return": true,
    "success": true
}
```

Send a custom event on the bus from minion:
`salt-call event.send /my/custom/event '{"data": "JDL"}'`

```
/my/custom/event    {
    "_stamp": "2017-04-13T10:03:35.854050",
    "cmd": "_minion_event",
    "data": {
        "__pub_fun": "event.send",
        "__pub_jid": "20170413060330447842",
        "__pub_pid": 3405,
        "__pub_tgt": "salt-call",
        "data": "JDL"
    },
    "id": "minion01",
    "tag": "/my/custom/even"
}
```

Master conf /etc/salt/master:

```
reactor:
  - 'salt/job/*/ret/minion01':
    - /srv/salt/reactors/touch-minion02.sls
```

--> Reactor on minion02 when minion0 returns something

/srv/salt/reactors/touch.sls

```
command_run:
  cmd.cmd.run:
    - tgt: minion02
    - arg:
      - "touch /tmp/touch.txt"
```

salt 'minion01' test.ping

See on minion02 /tmp/touch.txt

Reactor on beacons

`yum install python-inotify` on minions (dep)

/etc/salt/minion.d/beacons.conf

```
beacons:
  inotify:
    /etc/passwd: {}
    interval: 5
```

Run `adduser jdl` on salt-minion and see event bus

```
salt/beacon/minion0/inotify//etc/passwd {
    "_stamp": "2017-04-11T19:08:18.505247",
    "change": "IN_IGNORED",
    "id": "minion01",
    "path": "/etc/passwd"
}
```

Extending Salt (custom grain)

- custom grain from API (httpd)

/var/www/html/index.html

```
{"custom-grain":"custom-value"}
```

/srv/salt/states/_grains/custom.py

```
#!/usr/bin/env python

import requests

def custom_grain():
    grains = {}
    r = requests.get("http://jdl-master")
    grains['zzz_custom'] = r.json()
    return grains
```

salt '*' saltutil.sync_all

Show minion logs

```
[INFO    ] Starting new HTTP connection (1): 10.1.1.17
[DEBUG   ] "GET / HTTP/1.1" 200 34
```

`salt '*' grains.item zzz_custom`

Use custom-grain in motd.jinja

```
{{ grains['zzz_custom']['custom-grain'] }}
```

or better `{{ salt['grains.get']('zzz_custom:custom-grain') }}`

Extending Salt (custom module)

`/srv/salt/states/_modules/jdl.py`

```
#!/usr/bin/env python

import requests

def public_ip():
    r = requests.get('https://ip.wains.be')
    out = r.content
    return out
```

salt '*' saltutil.rsync_all

```
[root@master01 _modules]# salt '*' jdl.public_ip
minion01:
    8.8.8.8
minion02:
    8.8.4.4
```

Use custom module in motd.jinja `{{ salt['jdl.public_ip']() }}`

Salt API

`yum install -y salt-api`

Create local account `testapi` + pwd

Set up API

```
external_auth:
  pam:
    testapi:
      - .*
      - '@wheel'
      - '@runner'
      - '@jobs'

rest_cherrypy:
  port: 8080
  host: 0.0.0.0
  disable_ssl: True
  webhook_url: /hook
  webhook_disable_auth: True
```

`systemctl restart salt-master` (for auth) and `salt-api -l debug`

runner (using Postman)

URL: `http://master01:8080/run`

headers:

- Content-Type: application/x-www-form-urlencoded

`body: client=local&tgt=*&fun=test.ping&username=testapi&password=xxx&eauth=pam`

- test.ping (runner)
- state.highstate (runner)

(no API event on the bus for runners)

webhook (using Postman)

URL: `http://master01:8080/hook/test/jdl`

No headers, no auth

Create reactor:

/etc/salt/master

```
reactor:
  - 'salt/netapi/hook/test/jdl':
    - /srv/salt/reactors/api-hook-jdl.sls
```

/srv/salt/reactors/api-hook-jdl.sls

```
command_run:
  cmd.cmd.run:
    - tgt: minion02
    - arg:
      - "touch /tmp/api-works.txt"
```