Salting things up in the Devops’ World

Things just got real
whoami

- Juan Manuel Santos
- Team Leader / Principal Technical Support Engineer @ Red Hat
- Organizer of sysarmy / nerdear.la

- Using Salt for a couple of years now, no regrets.
- Or all regrets.
Disclaimer

- I am just a user.
- I do not develop Salt, although I do annoy the team on IRC.
- Only had 72 hours to prepare this.
Why Salt?

- Configuration Management System
- Like Puppet / Chef / Ansible (only better ;)
- Python / YAML / Jinja
- Relatively easy to understand.
- Extremely powerful.
- Allows “root-less” operation (via SSH).
Previously...


- Master / minion
- States / highstates
- Matching
- Grains / Pillar
- Unfortunately, still no Python 3 support:
  - [https://github.com/saltstack/salt/issues/11995](https://github.com/saltstack/salt/issues/11995)
Salt Mine

- Collect arbitrary data on the minions. Ship it to the master.
- Only the most recent data is maintained.
- Data is made available for all minions.

- Grains?
- Mine data is more updated. Grains are mainly static (can be updated manually though).
- If minions need data from other (slower) minions, the mine caches it. It is at least mine-interval fresh.
Salt Mine

● Mine can be populated by either:
  ○ Pillar
  ○ Minion’s configuration file

● In the case of salt-ssh:
  ○ Roster data
  ○ Pillar
  ○ Master configuration
Salt Mine - example

# /srv/pillar/top.sls:
base:
  'G@roles:web':
    - web

# /srv/pillar/web.sls:
mine_functions:
  network.ip_addrs: [eth0]

# /etc/salt/minion.d/mine.conf:
mine_interval: 5
Salt Mine - example

# /srv/salt/haproxy.sls:
haproxy_config:
file.managed:
  - name: /etc/haproxy/config
  - source: salt://haproxy_config
  - template: jinja

# /srv/salt/haproxy_config:
...file contents snipped...>
{% for server, addrs in salt['mine.get']('roles:web', 'network.ip_addrs', expr_form='grain') | dictsort() %}
server {{ server }} {{ addrs[0] }}:80 check
{% endfor %}
...file contents snipped...>
Topologies

- Most common topology:
  - Master → Minion(sssss)

- Alternative topologies?
- Moar masters?
- Segregation?
Topologies

WHAT IF WE TRIED MORE POWER?
Syndic Node

- Intermediate node type, special passthrough one.
- Controls a given set of lower level minions.
- Two daemons running: *salt-syndic* & *salt-master* (optionally but recommended, a *salt-minion* too).

- *salt-syndic* receives ‘orders’ from the **master of masters**.
- *salt-syndic* relays those orders to the local master.
- *salt-master* gets the ‘orders’ and relays them to the lower minions.
- The Syndic node is now a **master of minions**.
Syndic Node

Salt Master (master of masters)

- Salt Minion
- Salt Minion
- Salt Syndic/Master (master of minions)
  - Salt Minion
  - Salt Minion
  - Salt Minion
  - Salt Syndic/Master (master of minions)
    - Salt Minion
    - Salt Minion
Syndic Node - configuration

● On the Syndic node:

```bash
# /etc/salt/master
syndic_master: 10.10.0.1  # may be either an IP address or a hostname

# /etc/salt/minion
# id is shared by the salt-syndic daemon and a possible salt-minion daemon
# on the Syndic node
id: my_syndic
```

● On the master node:

```bash
# /etc/salt/master
order_masters: True
```
Syndic Node - running it

- On the Syndic node:
  - `salt-syndic`

- On the master node:
  - `salt-key -A my_syndic`
Syndic Node

- Different syndics per environment (production, development, QA, etc).
- Different syndics to comply with InfoSec standards.
- We can even do multimaster:
  - [https://docs.saltstack.com/en/latest/topics/tutorials/multimaster.html](https://docs.saltstack.com/en/latest/topics/tutorials/multimaster.html)
The Event System

- Track events.
- That’s it? No.
- Events can be acted upon.
- The Event System is at the base of many other subsystems.

- The event bus is a ZeroMQ PUB interface.
- Every event has a tag.
- Every event has a data structure.
The Event System

- Viewing events:
  
  # salt-run state.event pretty=True

- Sending events to the master:
  
  # salt-call event.send 'myevent/mytag/success' '{success: True, message: "It works!"}'

- Events can also be sent from Python code.
The Event System

- If watching the event bus, this shows up:

    salt/job/20160717180356279472/ret/gantrithor    {
        "_stamp": "2016-07-17T21:03:56.280813",
        [...
        "fun": "event.send",
        "fun_args": [
            "myevent/mytag/success",
            "{success: True, message: "It works!\"}" ]
    }
    "id": "minion_id",
    [...]}
State (Execution) modules vs Runner modules

- Execution modules run on the targeted minions.
- Runner modules run on the master.
- They can be either asynchronous or synchronous.
- Added via `runner_dirs` configuration variable in `/etc/salt/master`.
- Runner modules can be written in pure Python code.
- Convenience: any print statements will generate events.
State (Execution) modules vs Runner modules

def a_runner(outputter=None, display_progress=False):
    print('Hello world')
...

Event fired at Tue Jan 13 15:26:45 2015
***************************
Tag: salt/run/20150113152644070246/print
Data:
{'_stamp': '2015-01-13T15:26:45.078707',
 'data': 'hello',
 'outputter': 'pprint'}
State (Execution) modules vs Runner modules

- You don’t have to forcefully write runner modules.
- Full list: [https://docs.saltstack.com/en/latest/ref/runners/all/index.html](https://docs.saltstack.com/en/latest/ref/runners/all/index.html)
Beacons

- Like in the picture, Salt Beacons serve as a signal.
- Beacons use the Salt Event System to monitor things outside Salt.
- Send notifications (an event) when something changes.
- Are configured via the minion’s configuration file.

- inotify anyone?
- In fact...
Beacons - examples

- inotify

# cat /etc/salt/minion.d/beacons.conf
beacons:
  inotify:
    /etc/resolv.conf:
      auto_add: True
      interval: 30
[...]
Beacons - examples

- Process

```bash
# cat /etc/salt/minion.d/beacons.conf
beacons:
  [...]  
  service:
    process_name:
      onchangeonly: True
    interval: 120
```
Beacons - examples

- Memory usage
- Disk usage
- System load
- Network settings
- ...
- Your own

Salt Reactor
Salt Reactor

- Its job is to “react” (not JS :)
- Trigger actions in response to an event
- So it needs the event system
- Actions → states!
- In reality:
  - *Something happened* → Event → Reactor → Action (state)
- Reactors are defined in the master’s configuration file.
Salt Reactor - associating events to states

- In the master’s configuration file:

```yaml
reactor:
  - 'salt/minion/*/start':  # Match tag "salt/minion/*/start"
    - /srv/reactor/start.sls  # Things to do when a minion starts
    - /srv/reactor/monitor.sls  # Other things to do
  - 'salt/cloud/*/destroyed':  # Glob can be used to match tags
    - /srv/reactor/destroy/*.sls  # Glob can be used to match file names
  - 'myevent/custom/event/tag':  # React to custom event tags
    - salt://reactor/mycustom.sls  # Reactor files can come from the salt fileserver
```
Salt Reactor - Caveats

- State system in the Reactor is limited.
- When compared to the normal state system, things will be missing.
- Grains and pillar are unavailable inside the reactor subsystem.
- Plus, reactor sls are processed sequentially and handled over to a pool of worker threads.

**TL;DR: do not handle logic in reactor states**
- Use them for matching (‘Which minions? Which states?’).
- Call normal Salt states instead and handle the logic there.
- This is due to a ‘disconnect’ between the reactor & master engines (different namespaces).
Salt Reactor - associating events to states

- 'myevent/custom/event/tag': # React to custom event tags
- salt://reactor/mycustom.sls # Reactor files can come from the salt fileserver

# /srv/salt/reactor/mycustom.sls
{% if data['id'] == 'mysql1' %}
state_run:
  local.state.sls:
    - tgt: mysql1
    - arg:
      - a_long_running_and_complex_state
{% endif %}
Salt Reactor - full example

- Need to have minions’ keys automatically accepted.

```bash
# /etc/salt/master.d/reactor.conf:
reactor:
  - 'salt/auth':
    - /srv/reactor/auth-pending.sls
  - 'salt/minion/nice*/start':
    - /srv/reactor/auth-complete.sls
```
Salt Reactor - full example

# /srv/reactor/auth-pending.sls:
{% if not data['result'] and data['id'].startswith('nice') %}
minion_remove:
  wheel.key.delete:
    - match: {{ data['id'] }}

minion_rejoin:
  local.cmd.run:
    - tgt: salt-master.domain.tld
    - arg:
      - ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no "{{ data['id'] }}" 'sleep 10 & & /etc/init.d/salt-minion restart'
{% endif %}
Salt Reactor - full example

[...]

#{ Nice server is sending new key -- accept this key #}
{% if 'act' in data and data['act'] == 'pend' and data['id'].startswith(nice) %}
minion_add:
   wheel.key.accept:
      - match: {{ data['id'] }}
{% endif %}
{% endif %}

#/srv/reactor/auth-complete.sls:
#{ When a Nice server connects, run state.apply. #}
highstate_run:
   local.state.apply:
      - tgt: {{ data['id'] }}
Salt API

- REST API allowing to send commands to a running Salt master server.
- Supports encryption.
- Supports **authentication**.
- Authentication provided via Salt’s External Authentication System.
- Controlled by the **salt-api** daemon.
Salt API - example

# curl -i saltmaster:8000/minions/minion-id

GET /minions/minion-id HTTP/1.1
Host: localhost:8000
Accept: application/x-yaml

HTTP/1.1 200 OK
Content-Length: 129005
Content-Type: application/x-yaml

return:
- minion-id:
  Grains.items:
    ...
Salt API

- /
- /login
- /logout
- /minions
- /jobs
- /run
- /events
- /hook
- /keys
- /ws
- /stats
Salt API - /hook

- Generic webhook entry point.
- Fires events on Salt’s event bus.
- Data is passed as-is to an event.
- Authentication can be explicitly disabled here (think legacy apps).
- This does not mean you can make do without security!
Putting them all together
Recapping

- Events
- Beacons
- Reactor
- API
Recapping

- Beacons
- Salt API
- Salt Event System
- Salt States (SLS)
- Salt Reactor
Recapping

- With great responsibility comes great power.
- If configured properly, Salt can allow for full control of an infrastructure.
- Don’t fear the power; beware of the security risks though.
Possibilities

- Self healing your applications / systems.
- The endless CI cycle of push → build → test → deploy
- Scaling:
  - Both ways (growing and shrinking the environment)
  - Provisioning required.
- Keep environments consistent: react immediately upon changes.
Docs

- [https://docs.saltstack.com/en/latest/](https://docs.saltstack.com/en/latest/)
- #salt @ irc.freenode.net
Q&A

- Twitter: @godlike64
- Freenode: godlike
Thank you!