ABOUT SALTSTACK

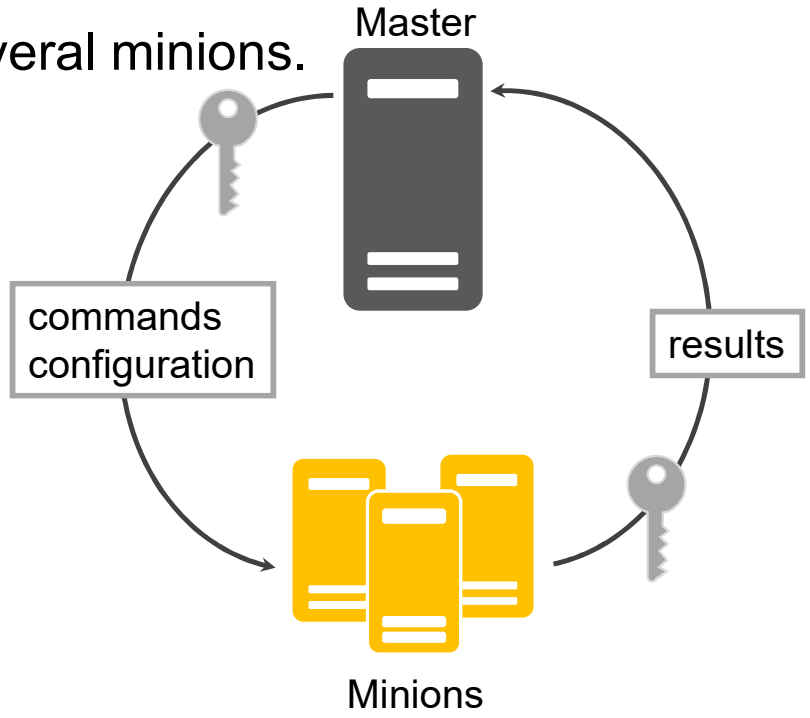
- IT automation tool.
- Competes primarily with Puppet, Chef, Ansible, and StackStorm.
- Event driven infrastructure tool (like StackStorm).
- Python based, open source
- SaltStack is the company. Salt Open is the open source tool. SaltStack Enterprise is the software they sell and support
  - think about SaltStack/Salt Open/SaltStack Enterprise vs Red Hat/Ansible/Tower.

ABOUT SALTSTACK

- Remote execution tool and configuration management system
  - Remote execution engine: run commands on various machines in parallel with a flexible targeting system.
  - Configuration management system: establishes a client-server model to bring infrastructure components in line with a given policy (kind of Ansible playbooks approach)

SALTSTACK ARCHITECTURE

- Client server model
  - Master (server)
  - Minions (agents)
- Kind of hub and spoke with a master controlling several minions.
- Control plan/management plan is 0MQ
  - Asynchronous
  - Publish-Subscribe model
- Encrypted communication (SSH)



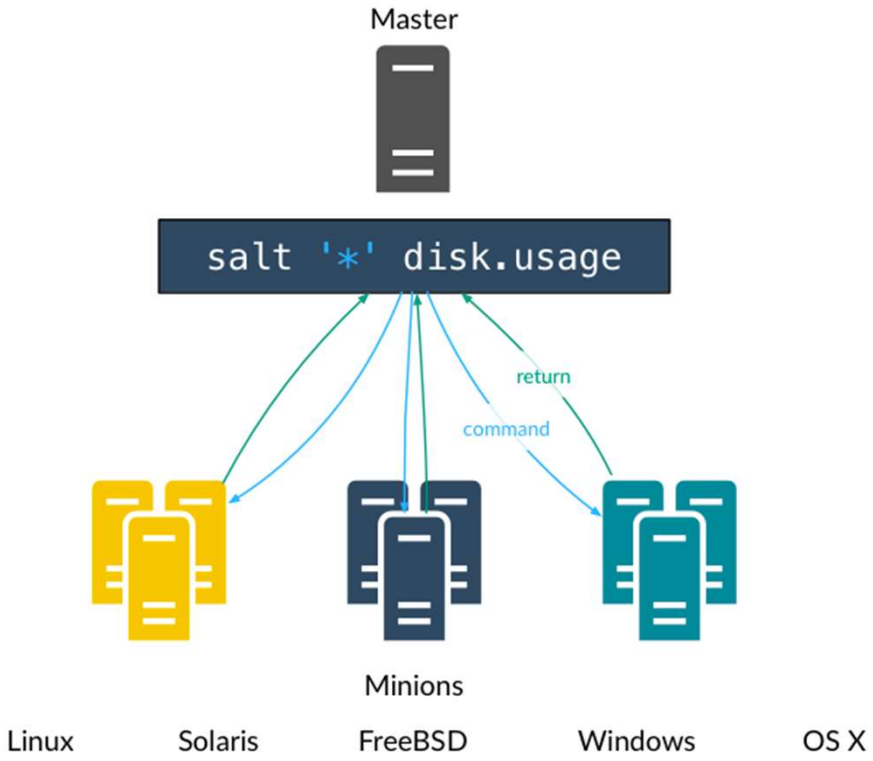
PILLARS AND GRAINS

- Pillars
  - User defined variables
  - Stored on the master
    - Can be stored on a remote git server as well

- Grains
  - Static (read only) information about the managed device such as model, OS version, SN etc.
  - Kind of Puppet/Ansible facts
  - Junos proxy stores the reply of PyEZ facts in grains

EXECUTION MODULES

- Ad hoc commands executed from command line to one or more target minions



STATE FILES

- The core of the Salt State system is the SLS, or **S**a**L**t **S**tate file
- The SLS is a representation of the state in which a system should be in
- Kind of Ansible playbooks
- State files use state moules
- YAML data structure. JINJA templates can be used

```
#more /srv/salt/apache.sls
   install_apache:
     pkg.installed:
       - name: apache
```

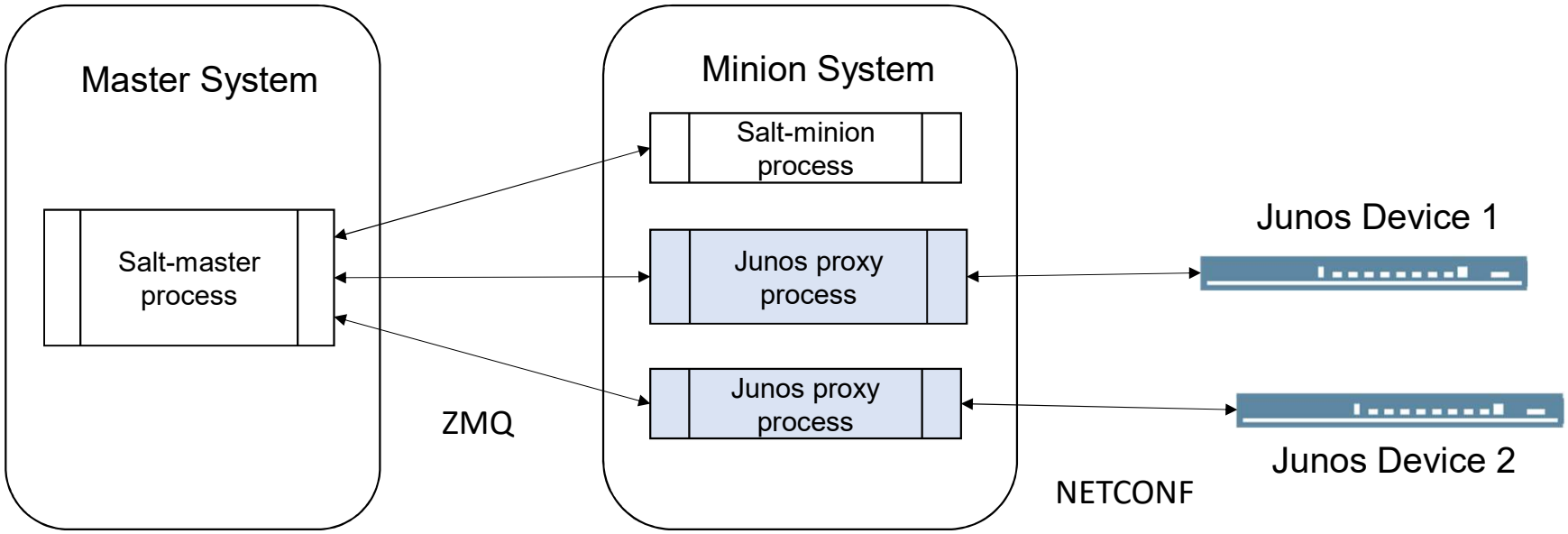
THE TOP FILES

- There are 2 TOP files ☺
- Mapping between groups of machines and the state files that should be applied
- Mapping between groups of machines and the pillar data available to them.

SALT COMPONENTS FOR JUNOS

- Junos proxy controls junos devices without installing a salt minion on device
- Junos execution and state modules
- PyEZ facts stored in grains
- Apart from that there is a junos syslog engine

JUNOS PROXY


Master System
Salt-master process
ZMQ
Minion System
Salt-minion process
Junos proxy process
Junos proxy process
NETCONF
Junos Device 1
Junos Device 2

PILLARS

Provide the details of a device in “pillar” file

Pillar top file

```
base:
  'ex4200-7':
     - ex4200-7-details
  'vsrx01':
     - vsrx01-details
  'vqfx01':
     - vqfx01-details
```

pillars for vqfx01 (vqfx01-details.sls file)

```
proxy:
      proxytype: junos
      host: 192.168.233.158
      username: root
      port: 8331
      passwd: Juniper
```

EXECUTION MODULES FOR JUNOS

- execute commands on junos devices (junos.cli)
- execute an rpc on junos devices (junos.rpc)
- Installs a configuration file into the candidate configuration (junos.install_config)
- gives the difference between the candidate and the current configuration (junos.diff)
- commit a change loaded in the candidate configuration (junos.commit)
- rollback a confifuration (junos.rollback)
- Displays the facts gathered during the connection (junos.facts)
- Copy a file to junos devices (junos.file_copy)
- Installs an image on the device (junos.install_os)
- Shuts down a device (junos.shutdown)
- zeroize a device (junos.zeroize)

https://docs.saltstack.com/en/latest/ref/modules/all/salt.modules.junos.html

EXECUTION MODULES FOR JUNOS

salt "vq*" junos.cli "show version"

salt "vq*" junos.rpc get-software-information

salt "vq*" junos.rpc get-software-information "file1" 'text'

salt "vq*" junos.rpc 'get-interface-information' '/home/ksator/interface.log' 'text' interface_name='lo0' terse=True

salt "vq*" junos.rpc 'get_config' '' 'text' filter='<configuration><system/></configuration>'

FLEXIBLE TARGETING SYSTEM

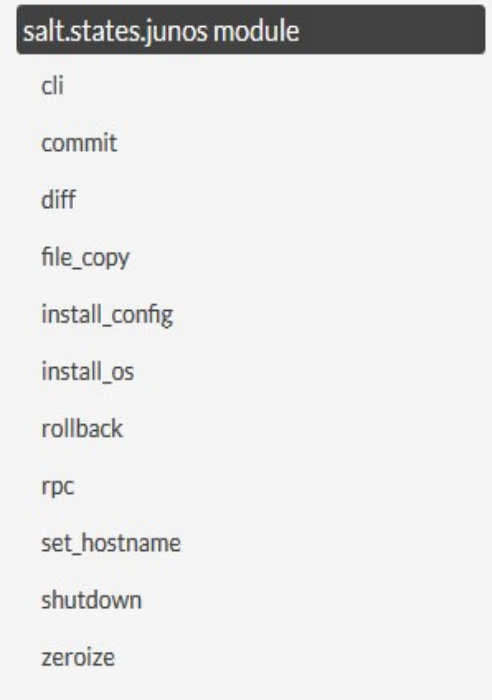
To identify targets, we can use a list, a regex, grains, nodegroups:

```
salt ex4200-7 test.ping
salt -L "ex4200-7, vqfx01" test.ping
salt "ex*" test.ping
salt -G 'junos_facts:model:EX4200-48T' junos.cli "show version“
salt -G 'os_family:junos' test.ping
salt -N group1 test.ping
```

Master configuration file (/etc/salt/master):

```
...
nodegroups:
 group1: 'L@ex4200-7,vqfx01'
 group2:
  - minion_1
  - ex4200-7
 group3: 'G@os_family:junos or minion_1'
...
```

STATE MODULES FOR JUNOS



- execute commands on junos devices (junos.cli)
- execute an rpc on junos devices (junos.rpc)
- Installs a configuration file into the candidate configuration (junos.install_config)
- gives the difference between the candidate and the current configuration (junos.diff)
- commit a change loaded in the candidate configuration (junos.commit)
- rollback a confifuration (junos.rollback)
- Copy a file to junos devices (junos.file_copy)
- Installs an image on the device (junos.install_os)
- Shuts down a device (junos.shutdown)
- zeroize a device (junos.zeroize)

https://docs.saltstack.com/en/latest/ref/states/all/salt.states.junos.html#state-modules-to-interact-with-junos-devices

STATE MODULES FOR JUNOS

state file /srv/salt/install_config2.sls

```
salt://config2.set:
  junos:
    - install_config
    - comment: commit from Salt
    - template_vars:
        dev_name: qefdwfcxwc
```

apply the state file from the master

```
salt "vq*" state.apply install_config2
```

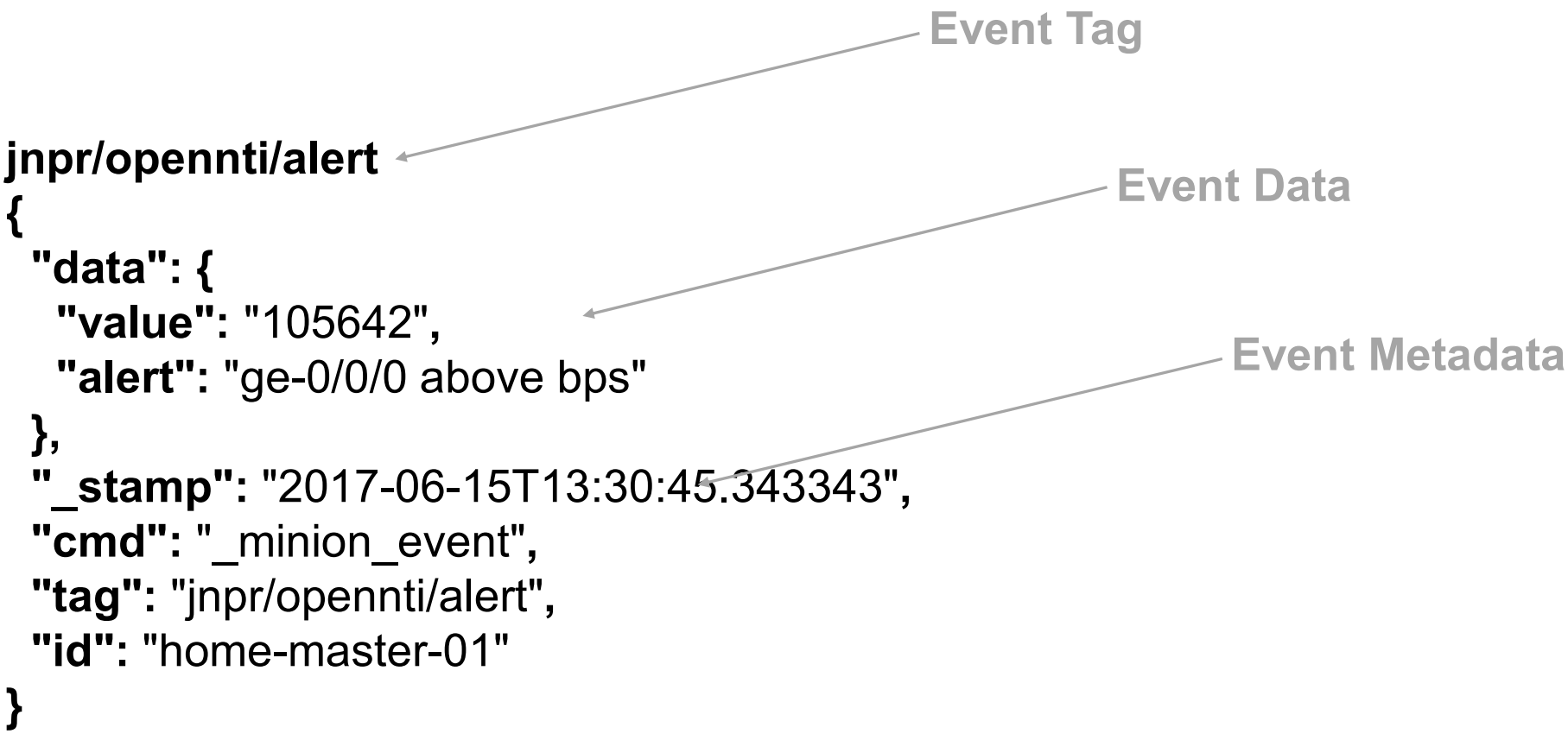
configuration file /srv/salt/config2.set

```
set system host-name {{ template_vars['dev_name'] }}
```

SALT EVENT SYSTEM

- Saltstack uses a pub/sub model to publish events
- Masters, minions, and proxies are attached to an event bus.
- ‘automation backplane’ in the form of an ’Event Bus’.
- Events are data structures which contain a **tag** and a **body**.
  - The **tag** is a high level description or name of the event.
  - The **body** is a dict containing detailed information about the event.

EXAMPLE EVENT

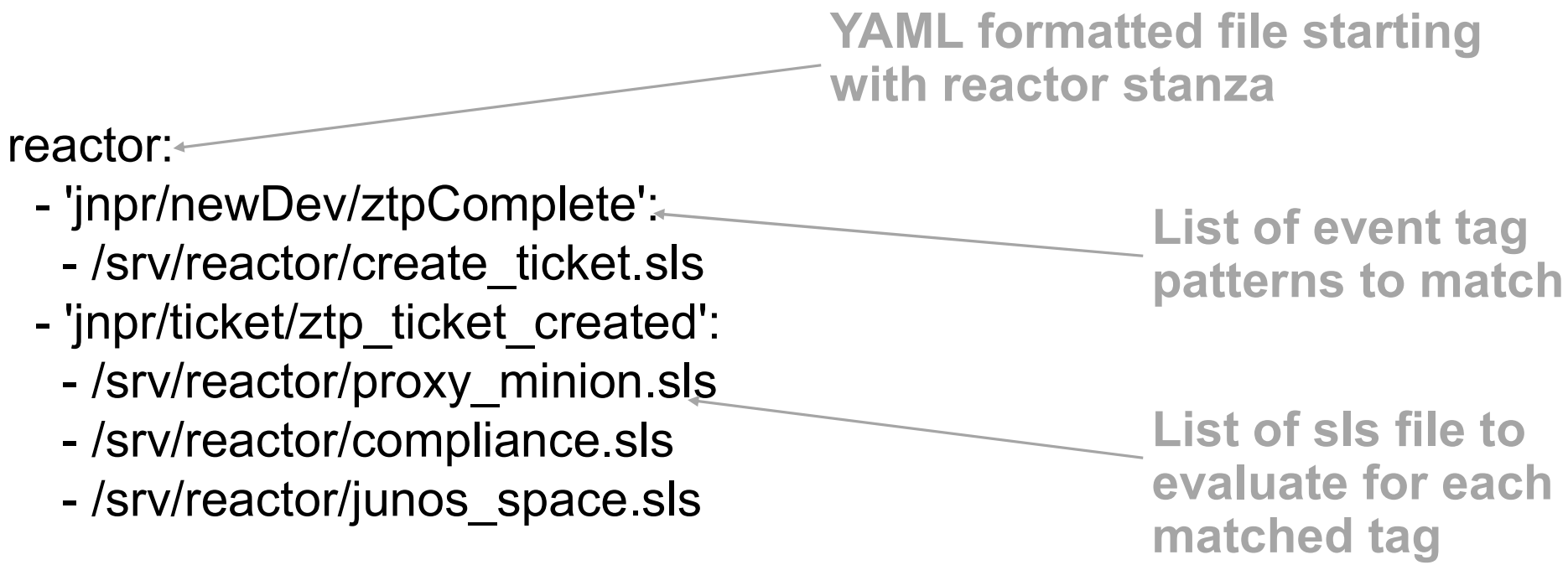


REACTOR SYSTEM

- Watch event bus

- Provides the ability to take actions according to events.
  - This system binds sls files to event tags on the master.
  - These sls files then define reactions.

REACTOR CONFIGURATION



JUNOS SYSLOG ENGINE

- Salt engines are long-running, external system processes managed by Salt.
  - If an engine stops, it is restarted automatically.

- Junos syslog engine:
  - listens to syslog events
  - extracts event information
  - sends it on the master/minion event bus.
  - Control the type of events to be sent.

EXAMPLE USE CASES

- SaltStack can react to junos events
- SaltStack can be also used to automate numerous tasks beyond just provisioning.
- By subscribing to events, SaltStack can launch various actions automatically based on a simple reactor system:
  - Perform a backup of a network device every time a COMMIT operation happens
  - Audit a network device configuration or operational state every time a COMMIT operation happens
  - Push configuration to a device if a compliance audit fails


Thank you