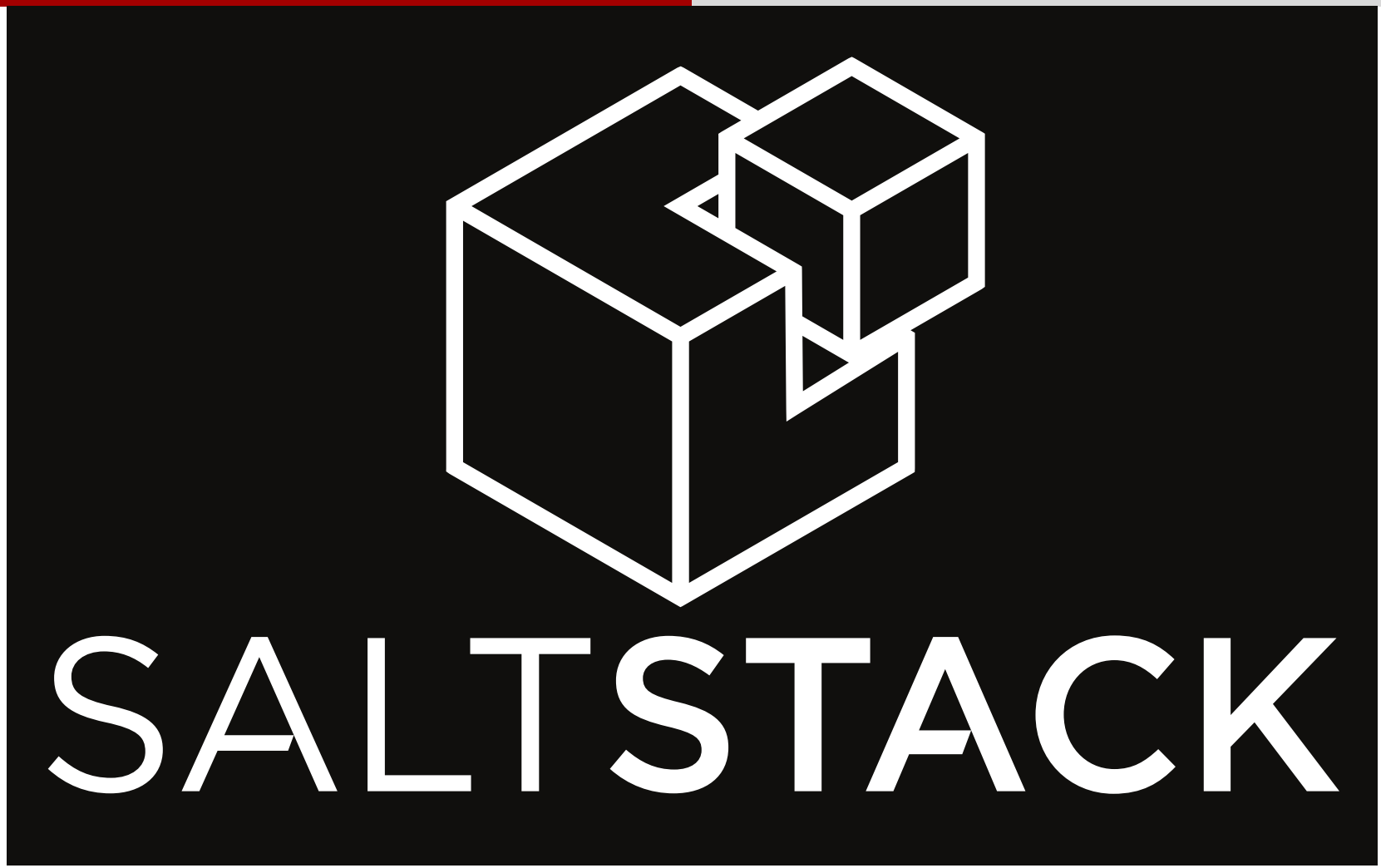
Real-time infrastructure management with Salt

Seth House <seth@eseth.com>

OpenWest Conference 2013

2013-05-03


SALTSTACK

Outline

Salt internals

Master

- `salt-master -d`
- Open two ports (pub/sub & reply channel)

Minions

- `salt-minion -d`
- Connect to the master
- No open ports required
- Listens for pubs from the master

`/etc/salt/minion`:

```
#master: salt
```

Execution modules

- Contain all the functionality

Execution example

```
salt 'web-*' network.interfaces
```

State modules

- Wrap execution modules
- Before-check
- `test=true`
- Call out to execution modules
- After-check

State module example

top.sls:

```
base:
  'web-*':
    - httpd
```

httpd.sls:

```
httpd:
  pkg:
    - installed
```

Outline

Why is Salt fast?

Communication

- ZeroMQ
- msgpack

pub/sub

- Asynchronous
- Minions determine targeting match
- Minions do all the work

Outline

Sharing minion data

<-- Live	-- Recent --	Historic -->
peer interface	Salt Mine	Returners

Peer

/etc/salt/master:

```
peer:
  lb-.*:
    - network.interfaces
```

- Be mindful of data security
- Communication still goes through the master

Peer example

Configuring `haproxy.cfg`:

```
{% for server,ip in
    salt['publish.publish']('
        'web*',
        'network.interfaces',
        ['eth0']).items() %}
server {{ server }} {{ ip[0] }}:80 check
{% endfor %}
```

Salt Mine

/etc/salt/{master,minion}:

```
mine_functions:
    network.interfaces: [eth0]

mine_interval: 60
```

- New in Salt v0.15
- Either master or minion config
- Be mindful of data security

Salt Mine example

Configuring `haproxy.cfg`:

```
{% for server,ip in
    salt['mine.get']('web-*',
        'network.interfaces',
        ['eth0']).items() %}
server {{ server }} {{ ip[0] }}:80 check
{% endfor %}
```

Returners

`/etc/salt/{master,minion}`:

```
redis.db: 0
redis.host: myredis
redis.port: 6379
```

- Minions write directly
- Can be read into Pillar via `ext_pillar`

Returner full-circle example

Collect the data:

```
salt 'web-*' network.interfaces eth0 \
    --return redis_return
```

Fetch the data via a custom `ext_pillar` module.
Use the data:

```
{% for server,ip in
    salt['pillar.get']('web.ip_addrs', {}).items() %}
server {{ server }} {{ ip[0] }}:80 check
{% endfor %}
```

Outline

Events

Fire events

```
salt 'lb-*' event.fire_master \
    refresh_pool loadbalancer
```

Watch for events (manually)

- Some assembly required
- `salt/tests/eventlisten.py`
- Coming soon to `salt-api`

Reactor (react to events)

/etc/salt/master:

```
reactor:
  - loadbalancer:
    - /src/reactor/refresh_pool.sls
```

/src/reactor/refresh_pool.sls:

```
{% if data['type'] == 'refresh_pool' %}
highstate_run:
  cmd.state.highstate:
    - tgt: lb-*
{% endif %}
```

Outline

Schedules

Add events

`/etc/salt/{master,minion}` (or pillar):

```
schedule:
  highstate:
    function: state.highstate
    minutes: 60
```

Stats gathering

```
schedule:
  uptime:
    function: status.uptime
    seconds: 60
    returner: redis
  meminfo:
    function: status.meminfo
    minutes: 5
    returner: redis
```

Outline

What's coming

Salt `v.0.next`

Monitoring states

- Configure inline with existing states
- Individual components are in place
- Needed: glue
- Needed: alerting

Data resolution

- Time-series data
- Thin and/or summarize older and older data
- Free with some returners