Salt and a bit of Trebuchet

ariel@wikimedia.org

http://primate.net/~ariel/salt/salt-trebuchet.odp

salt in general and at WMF

what is salt?

- a configuration and deployment tool
- a glorified remote command execution tool
- a puppet replacement in python
- a random output generator
- NaCl
- all of the above?

puppet replacement

- LinkedIn uses salt for config and deployment
- now managing 70k servers
- substantial contributions to codebase
- rest api for salt based on tornado
- http://www.slideshare.net/ThomasJackson4/salt-stack-at-web-scale-better-stronger-faster

WMF use case

- remote command execution, replacing ddsh
- ddsh: ssh in parallel. Great in theory.
- the Big Problem: upkeep of lists of machines
- action: remove a host, do something, forget to add it back (frequent!)
- consequence: mw servers missing the latest scap, serving old crap
- not all that fast either

introducing salt

- fast (when we started out)
- puppet-maintained groups via labels applied to servers, called “grains”
- easy to add our own modules 'cause python
- easy to store the result of commands in e.g. redis (“returners”)
- can permit certain hosts to run commands on all minions (tin as deployment server)

executing a command via salt

- all minions listen on one socket for messages (default: 4505)
- master sends out target list and job info on above socket
- minions read message, decide whether they are in the target list
- minions in target list run command
- minions in target list send output to master on one socket (default: 4506)
- master receives returns, publishes on event bus
- command line client reads event bus, waits until all targeted minions respond or timeout exceeded, displays returns for job, exits

The dog that didn't bark

- What's missing from the previous slide?

The dog that didn't bark

- What's missing from the previous slide?
- no guarantee that a specific minion recieved the request
- no guarantee that a specific minion ran the command
- no guarantee that a minion failing to run the command by the time the results were displayed to the user, didn't finish later
- no guarantee that a minion returned output data to the master

the grody underside of pub/sub

- master broadcasts to subscribers that return data asynchronously. No return in time frame?
- minion may be down
- minion process may be hung/not running
- minion may have trouble connecting to master to receive messages, or to return command output
- minion may be overloaded and respond very very slowly
- master doesn't know and can't distinuish between these

batching a command

- salt “*” -b 50 ...
- test.ping to all targeted minions
- collect up the hosts that returned by timeout, either default or user-specified
- send user command out to all minions with target 50 from list
- wait for timeout for command returns, display them
- if any more test.ping returns, add those hosts to end of list
- send user command to next batch of 50, etc

batching test.ping

TOO MANY MINIONS RETURNING AT ONCE

This can also happen during the testing phase, if all minions are addressed at once with

BASH

```
$ salt * test.ping
```

it may cause thousands of minions trying to return their data to the Salt Master open port 4506. Also causing a flood of syn-flood if the Master can't handle that many returns at once.

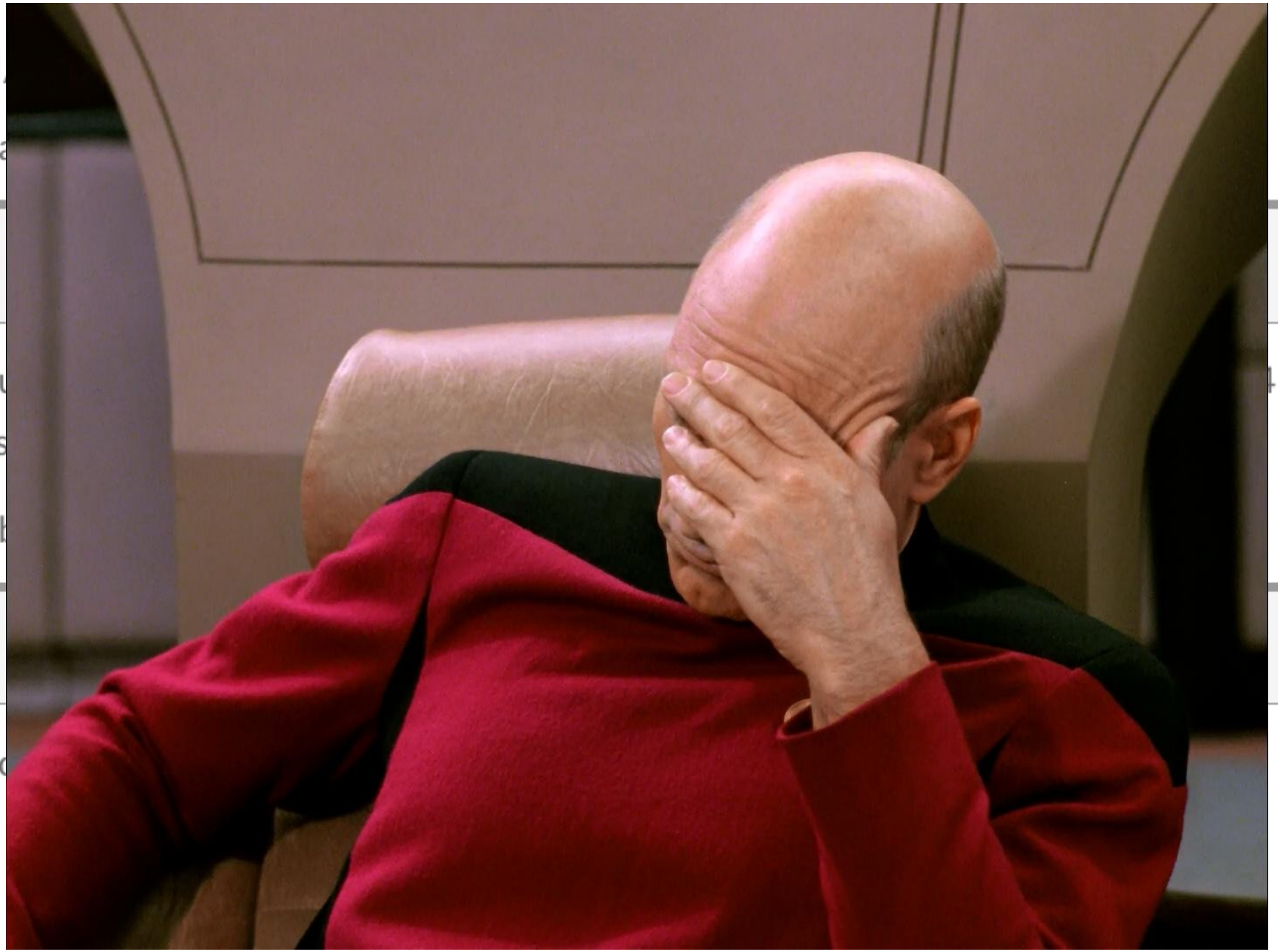
This can be easily avoided with Salt's batch mode:

BASH

```
$ salt * test.ping -b 50
```

This will only address 50 minions at once while looping through all addressed minions.

batching test.ping



TOO M

This can a

BASH

$ salt

it may cau

4506.

Also caus

This can b

BASH

$ salt

This will

Batch test.ping FAIL

- salt '*' -b 50 test.ping will first do...
- salt '*' test.ping
- the very command we know fails.
- fix: generate list of minions by listing all matching keys
- limits: works only for wildcard matches
- limits: unresponsive hosts are included in the batches; this can cause each batch run to wait the full timeout

modules, grains, pillars

what is a salt module?

- collection of functions in a python module that can be run on a minion
- see /usr/lib/python2.7/dist-packages/salt/modules
- user-contributed modules are in /srv/salt/_modules dir on master
- synced to minions via

  salt '*' saltutil.refresh_modules
- saltutil is a module; refresh_modules is one of the funtions available

test module

- /usr/lib/python2.7/dist-packages/salt/modules/test.py
- early version of function ping:
- def ping():

    return True
- salt '*' test.ping
- if module exists and has been succesfully loaded, the minion will return “True” by the usual returner to the master
- salt client reads the returns and displays them. Simple.

test module

writing your own module?

- can be as simple as test.ping
- errors? see if your module is loaded

```
root@palladium:~# salt dataset1001.wikimedia.org sys.list_modules
dataset1001.wikimedia.org:
    - aliases
    - alternatives
    - archive
    - at
    - blockdev
    - buildout
    - cloud
    - cmd
    - composer
    - config
    - cp
    - cron
    - daemontools
    ...
```

standard modules

- test (test.ping)
- cmd (cmd.run)
- grains (grains.items)
- pillars (pillars.items)
- publish (publish.publish)
- many others

salt grains

- label with value on a minion, stored locally
- salt <minionname> grains.setval name value
- or add to /etc/salt/minion config file
- example: palladium minion config file has:

```
grains:
  cluster: misc
  realm: production
  site: eqiad
```

More salt grains

- alternatively: add grains to /srv/salt/_grains on master
- sync to minions by

  salt '*' saltutil.sync_grains on the master
- master keeps a cache of minion grains:

  /var/cache/salt/master/minions/<minionname>/data.p
- binary format file so not too useful to examine directly

Even more salt grains

- salt/grains/core.py contains predefined grains
- examples:

```
biosreleasedate:
        10/21/2011
hwaddr_interfaces:
        ----------
        eth0:
            78:2b:cb:73:fc:44
kernelrelease:
        3.2.0-75-generic
osfinger:
        Ubuntu-12.04
```

Revenge of the son of the salt grain

- grains can also be hardcoded in /etc/salt/grains on minion

  YAML format file. Example:

  ```
  root@palladium:~# more /etc/salt/grains
  rolename:
  - misc::management::ipmi
  - puppetmaster
  trebuchet_master: tin.eqiad.wmnet
  ```

Return of the revenge... salt grain

- grains-based targeting:

  salt -G name:value test.ping

- list grains on a minion:

```
root@palladium:~# salt palladium.eqiad.wmnet grains.ls
palladium.eqiad.wmnet:
    - SSDs
    - biosreleasedate
    - biosversion
    - cluster
    - cpu_flags
    ...
```

salt grains season finale

- built-in grains rely on lspci, dmidecode, ip, /proc...
- other functions besides ls in the grains module:
- items (shows all grains with their values)
- has_value (check if a grain exists with a given value)
- get/setval/delval etc.
- use -C to target hosts with combinations of grains!

salt pillars

- sets of names and values similar to grains
- set on master in /srv/pillars/ in YAML format
- specify which minions get which data, match by grain is typical
- master starts with mandatory top.sls file, follows references to other files
- yes, the salt metaphor is getting a bit much if you ask me

salt pillars close up

```
ariel@palladium:~$ more /srv/pillars/top.sls
base:
  'deployment_server:true':
    - match: grain
    - deployment.repo_config
    - deployment.deployment_config
  'deployment_target:*':
    - match: grain
    - deployment.repo_config
    - deployment.deployment_config
```

salt pillars close up (2)

- for all hosts with grain deployment_server set to true, give them data from deployment.deployment_config, deployment.repo_config
- {"deployment_config": {"parent_dir": "/srv/deployment", "redis": {"db": 0, "host": "tin.eqiad.wmnet", "port": 6379}, "servers": {"codfw": "tin.eqiad.wmnet", "eqiad": "tin.eqiad.wmnet"}}}
- repo name, upstream url, submodules needed? gitfat used?

salt pillars close up (3)

Which hosts should get the data?

```
root@palladium:~# salt -G deployment_server:true cmd.run hostname
tin.eqiad.wmnet:
    tin
mira.codfw.wmnet:
    mira
```

What pillars does tin have? No way to list just pillar names, so...

```
root@palladium:~# result=`salt tin.eqiad.wmnet --out raw pillar.items`
root@palladium:~# python -c "print [k for k in $result[$result.keys()[0]].keys()]"
['repo_config', 'deployment_config', 'master']
```

monitoring salt

monitoring minion status

- don't do: salt-run manage.status

monitoring minion status

- don't do: salt-run manage.status
- nice try, but under the hood:

```
minions = client.cmd('*', 'test.ping', timeout=__opts__['timeout'])

key = salt.key.Key(__opts__)
keys = key.list_keys()

ret['up'] = sorted(minions)
ret['down'] = sorted(set(keys['minions']) - set(minions))
return ret
```

- do: ss -n -t -o state established 'sport = :4505'

salt event monitoring

- what the heck is the master sending/receiving? there's a script for that
- https://raw.githubusercontent.com/saltstack/salt/develop/tests/eventlisten.py
- by default listens for events on master ipc socket in /var/run/salt/
- on a minion: can listen to the minion ipc socket for events fired, won't cover incoming job requests nor outgoing returns

IPC sockets used by master

- located in /var/run/salt/master
- publish_pull.ipc: publisher gets worker messages intended for minions
- workers.ipc: req handler passes commands for master to backend worker
- master_event_pull.ipc: worker announces to master that job to be sent to minions, or sends returns from a job
- master_event_pub.ipc: salt cli scripts listen for events from master
- http://docs.saltstack.com/en/latest/topics/development/architecture.html

Events from test.ping to tin

announce job id on event bus:

self.event.fire_event({'minions': minions}, clear_load['jid'])

Event fired at Thu Aug 13 15:08:04 2015
****
Tag: 20150813150804834175
Data:
{'_stamp': '2015-08-13T15:08:04.860591', 'minions': ['tin.eqiad.wmnet']}

test.ping to tin (2)

dup event sent for backwards compat

self.event.fire_event(new_job_load, 'new_job')

Event fired at Thu Aug 13 15:08:04 2015
****
Tag: new_job
Data:
{'_stamp': '2015-08-13T15:08:04.860868',
 'arg': [],
 'fun': 'test.ping',
 'jid': '20150813150804834175',
 'minions': ['tin.eqiad.wmnet'],
 'tgt': 'tin.eqiad.wmnet',
 'tgt_type': 'glob',
 'user': 'sudo_ariel'}

test.ping to tin (3)

send notification of new job with load, current announcement style

self.event.fire_event(new_job_load, tagify([clear_load['jid'], 'new'], 'job'))

Event fired at Thu Aug 13 15:08:04 2015
****
Tag: salt/job/20150813150804834175/new
Data:
{'_stamp': '2015-08-13T15:08:04.861007',
 'arg': [],
 'fun': 'test.ping',
 'jid': '20150813150804834175',
 'minions': ['tin.eqiad.wmnet'],
 'tgt': 'tin.eqiad.wmnet',
 'tgt_type': 'glob',
 'user': 'sudo_ariel'}

test.ping to tin (4)

minion auths to master

self.event.fire_event(eload, tagify(prefix='auth'))

Event fired at Thu Aug 13 15:08:05 2015
****
Tag: salt/auth
Data:
{'_stamp': '2015-08-13T15:08:05.055560',
 'act': 'accept',
 'id': 'tin.eqiad.wmnet',
  'pub': '-----BEGIN PUBLIC KEY-----\nMIIBIjANBgk.....
  ..... 8\nFQIDAQAB\n-----END PUBLIC KEY-----\n',
 'result': True}

test.ping to tin (5)

old style announcement of command returns

self.event.fire_event(load, load['jid'])

Event fired at Thu Aug 13 15:08:05 2015
****
Tag: 20150813150804834175
Data:
{'_stamp': '2015-08-13T15:08:05.092427',
 'cmd': '_return',
 'fun': 'test.ping',
 'fun_args': [],
 'id': 'tin.eqiad.wmnet',
 'jid': '20150813150804834175',
 'retcode': 0,
 'return': True,
 'success': True}

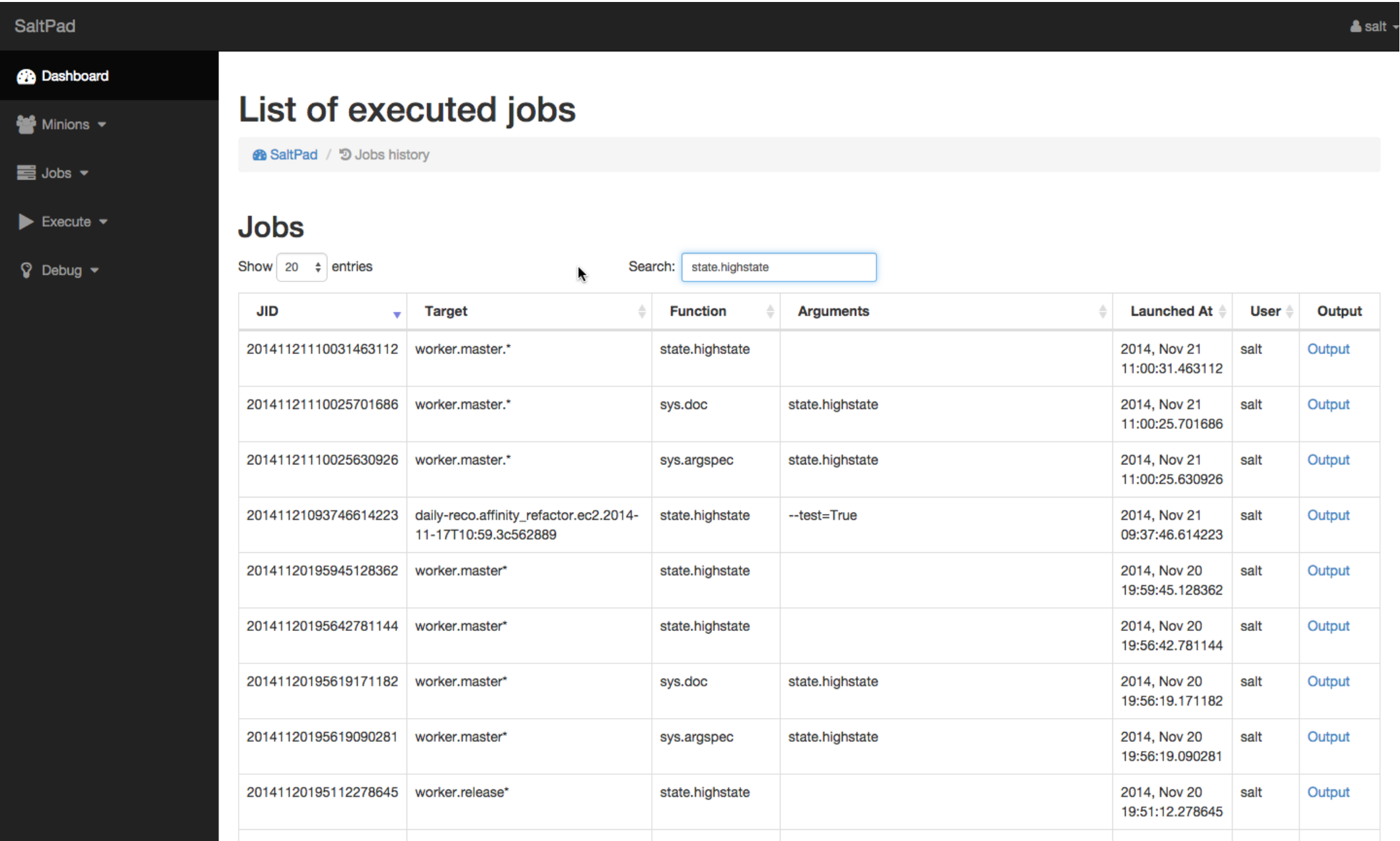
test.ping to tin (6)

return data having arrived and been processed

self.event.fire_event(load, tagify([load['jid'], 'ret', load['id']], 'job'))

```
Event fired at Thu Aug 13 15:08:05 2015
*************************
Tag: salt/job/20150813150804834175/ret/tin.eqiad.wmnet
Data:
{'_stamp': '2015-08-13T15:08:05.092602',
 'cmd': '_return',
 'fun': 'test.ping',
 'fun_args': [],
 'id': 'tin.eqiad.wmnet',
 'jid': '20150813150804834175',
 'retcode': 0,
 'return': True,
 'success': True}
```

“SaltPad” web gui (new)



minion eavesdropping

root@tin:~# python ./eventlisten.py -n minion -i tin.eqiad.wmnet
ipc:///var/run/salt/minion/minion_event_46c5de19aa_pub.ipc
Event fired at Thu Aug 13 13:37:08 2015
****

Tag: module_refresh
Data:
{'_stamp': '2015-08-13T13:37:08.271681'}
Event fired at Thu Aug 13 13:37:09 2015
****

Tag: pillar_refresh
Data:
{'_stamp': '2015-08-13T13:37:08.298031'}

a bit of trebuchet

Salt returners

- once a minion has run a command, what to do?
- return it someplace. default: master, which publishes results on event bus for salt client to read and display to user
- other options? redis memcache cassandra carbon or write your own
- trebuchet uses a custom redis returner

a simple salt returner

```
def returner(ret):
    '''
    Return data to the local syslog
    '''
    syslog.syslog(syslog.LOG_INFO,
                  'salt-minion: {0}'.format(json.dumps(ret)))
```

a simple salt returner

a complicated salt returner

```
def returner(ret):
    '''
        Return data to a redis data store
    '''
    function = ret['fun']
    log.debug('Entering deploy_redis returner')
    log.debug('function: {0}'.format(function))
    if not function.startswith('deploy.'):
        return False
    ret_data = ret['return']
    log.debug('ret_data: {0}'.format(ret_data))
    minion = ret['id']
    timestamp = time.time()
    serv = _get_serv()
    _record_function(serv, func, timestamp, minion, data)
```

trebuchet

- Actually called git-ryan-sartoris-trebuchet-deploy but who's keeping track
- for deploys where the git repo has files in the layout desired for the server install
- integrity checks rely on git (debs/rpms rely on md5, ddsh did no checks)
- two parts: trebuchet (salt module/runner/returner) plus “trigger” (python command line script 'git deploy')

trebuchet moving parts

- deployment server:

  git, apache, redis, grains, deployment repos, special git config per repo, git-deploy script (“trigger” package)

- master:

  salt, pillars, peer_run config, trebuchet module, runner, returner (trebuchet files are pushed out to minions)

- minions:

  grains, python redis bindings, python git bindings

trebuchet moving parts

anatomy of a trebuchet run

- start the deploy: set a special tag in the repo
- fetch: minion logs time with redis for fetch start, does fetch, logs time with redis for completion, ASYNC
- in the meantime git deploy fetch returns whatever it gets to us
- it might be quicker than the deploys and have only a few results instead of all
- checkout: minion logs start time with redis, does checkout, logs completion time with redis
- git deploy checkout doesn't wait but displays what it gets
- user is bewildered

the mystery of trebuchet targets

- how do targets for a given deployment repo get into redis?
- because they reply once to the command
- how do they get removed?
- they don't
- trebuchet doesn't know if the host is down, gone, no longer has that grain, or whatever so manual intervention is the only possibility
- we could insist that grain cache be locally updated on master, then we could force removal of minions from redis that don't match the grain. Worth it? Probably not

performance and scaling

salt keys and auth

- AES encryption with perfect forward secrecy
- any minion key deleted forces key rotation on master. Also rotated every 24 hours by default.
- next job sent by master goes to all minions (pub/sub), they must decrypt to decide if it's for them
- this forces them to re-auth, all of them at once = auth storm
- “ping_on_rotate” config option on master forces re-auth right after key rotation, but still have auth storm issue
- “random_reauth_delay” config, we use this

multiple salt masters

- add more masters in minion configs
- all masters are active, redundancy not scaling
- keys must be accepted/deleted on all masters
- commands may be sent from any master
- minion checks in to all masters on start
- masters must have same key, pillars must be kept in sync across masters

syndics for scaling

- a master-minions go-between
- two components, syndic daemon and salt master daemon on same server
- syndic daemon passes information between local master daemon and remote (real) master
- local master daemon passes information to/from minions and reports back to syndic
- not in use at WMF, might want one per DC

last thoughts

a bit more salt

- http://docs.saltstack.com/en/latest/topics/tutorials/intro_scale.html
  for more about scaling
- salt api: still in flux, included in salt version installed at WMF
- salt-cloud, salt-ssh, states and deployment
- https://github.com/apergos/docker-saltcluster
  To spin up a test cluster on your laptop (might have bitrot)
- ZMQ left for another day

no more salt!

Questions? apergos on irc
Complaints? /dev/null (j/k)
More info? http://docs.saltstack.com