HPC configuration management using Salt

Overview of Salt used for our clusters Baobab and Yggdrasil

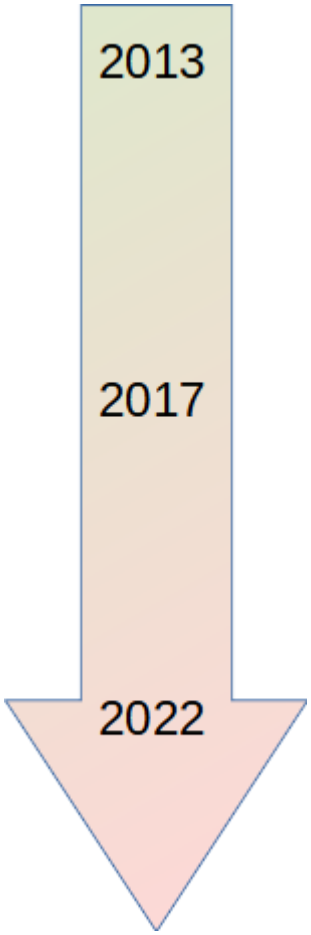
HPC-CH 19 05 2022

Yann Sagon

Goal

- We ensure that every part of the cluster is in the desired state.
- Reproducible and documented
- One central reference (everything in salt)
- We can reinstall a node, a server from scratch in a couple of minutes

History



- Install nodes from PXE
- Copy config files from a repo to nodes
- Only for compute nodes, no servers

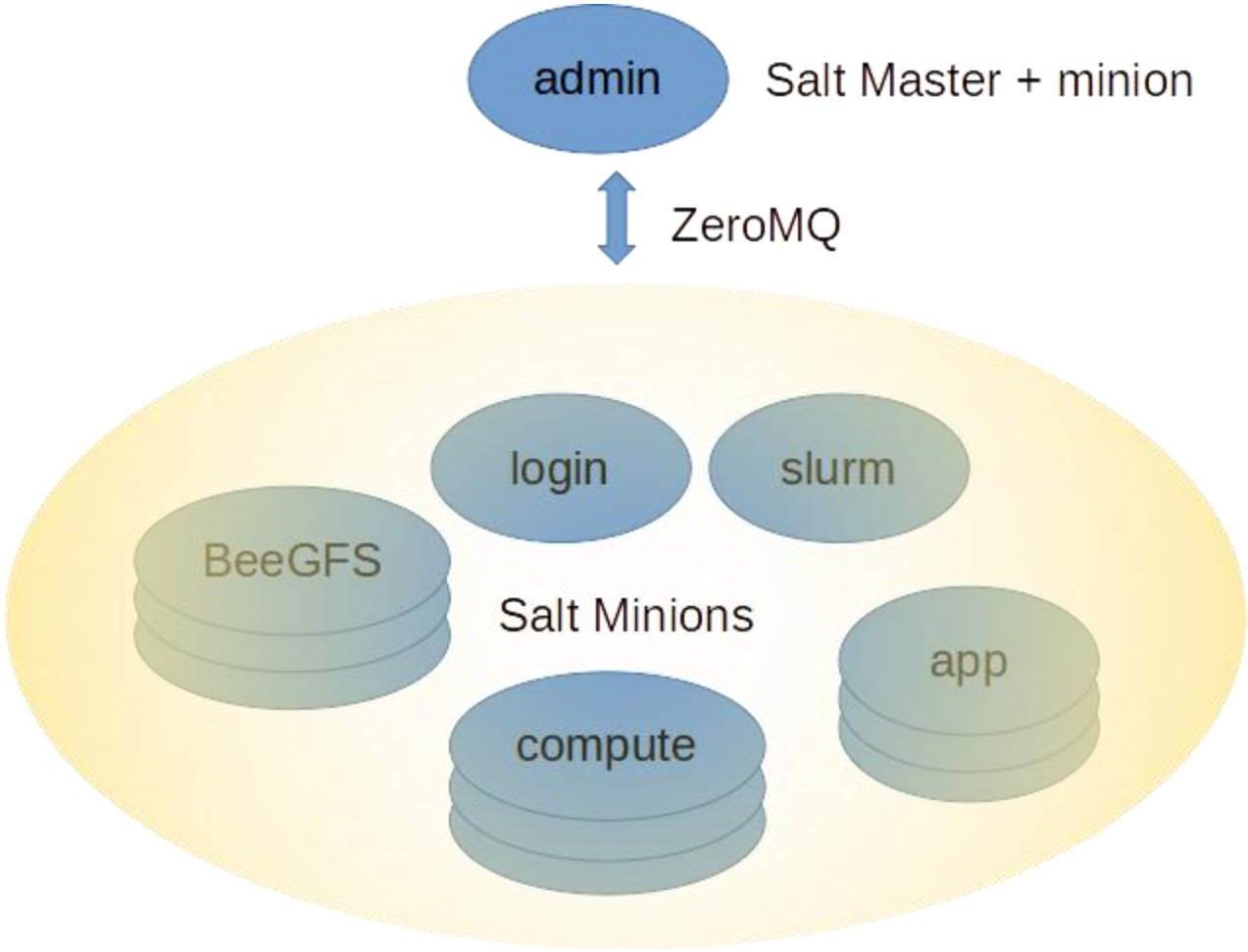
- Installed Salt as our configuration management server
- Moved config files to Salt states
- Parametrized config using templates, pillars
- Managed services

- Almost everything in Salt

What is Salt

- Salt is a client-server configuration management system.
- Central reference
- Files written in yaml + jinja (templates)
- Can scale to thousands of servers
- Abstraction from OS (pkgs, services network etc.)
- Customizable in Python
- License apache

HPC cluster infrastructure



Pillar

- Data specific to the installation are stored in pillars.
- Pillars data are consumed by states.
- GPG encoded secrets can be stored.
- Supports git repo
- Pillar support merging. Order:
  - Global
  - Roles
  - minions

In practice: pillar

```
ipmi:
  nodes:
    password: |
      -----BEGIN PGP MESSAGE-----
      Version: GnuPG v2.0.22 (GNU/Linux)

      hQGMA3QZI5D3Ev9gAQv8DfSBhhtlN1FmOdVCkcf0qym/SZS0Ez6XBzM5GCT/X0+B
      a/JHt[redacted]h+j0bq4RRvESsoc/WLfW87dyOSzb9PjH0I0diH3B[redacted]dbY3dHB
      lfZVq[redacted]jpw7kGilI5kd431ZIoPbxLo2LuRFlipjeUFF4K67[redacted]oNxgfRW
      aH0Dc[redacted]bqOcatT5x[redacted]dEmY7XOpIB2LZYM3z0UW1bMKLW7
      T1R5Ms6WS4bd+q59i0P5Q[redacted]HQPskjPd63[redacted]HgKpIttFXwk+
      cj3X7aa+Sm7w43CFI4E+yHc[redacted]ctBjDV4QfHMcIY4el/[redacted]iQyV3T6SSehh
      qcw/8O7zTaCIB5KCATFDFWTuexMBWPrQQ1tcO/3YxgrNbur+i05XhQqwTE6PhiAN
      n++GPKkeGT2YpaBSjDsXbiJxEXzTBoolHq[redacted]9ESQyvk6hKKIJOMYNplWqMR
      Xx5OZX3r5NjRBE2RRMeS0kUBKnbuulnckt[redacted]2OHiU6BZgH+4Qdw6VSCbrQJ
      N88aUHENADJDZmx7kask4iKlwHi3ZlKzVZfwFumsqPXiPu/+CdY=
      =heex
      -----END PGP MESSAGE-----
```

Cyphered

```
(baobab)-[root@admin1 ~]$ salt-call pillar.get ipmi:nodes:password
local:
    [redacted]
```

clear text

Grains

- Grains are information that a minion **have**
  - Hostname:
  - OS version:
  - CPU :
  - Custom (in Python): mellanox_card

- Can be used in states to do conditional stuff
- Can be used to target minions

Grains

```
(baobab)-[root@admin1 ~]$ salt "gpu031.baobab" grains.get cpu_model
gpu031.baobab:
    AMD EPYC 7742 64-Core Processor
(baobab)-[root@admin1 ~]$ salt "gpu031.baobab" grains.get mellanox_card
gpu031.baobab:
    ConnectX-5
(baobab)-[root@admin1 ~]$ salt "gpu031.baobab" grains.get osmajorrelease
gpu031.baobab:
    7
```

State

- Salt states are a «recipe» that once executed will set the minion in a given state.
- States should not be specific to the installation.
- Many states availables for states:
  - file management
  - packages management, version lock
  - mount points
  - users creation
  - custom: apache, mysql, cvmfs etc

State

- Can process templates
- Modular: state.substate
- Supports git repo
- Target minions: per grains, hostname
- All states to be applied to a minion with dependencies: highstate

In practice: state

Pillar

```
- name: cpu001
  sn: N-12.12.101
  ethernet:
    mac: '00:1e:67:4f:99:b0'
  management:
    mac: '00:1e:67:4f:99:b2'
  comment: ""

- name: cpu002
  sn: N-12.12.102
  ethernet:
    mac: '00:1e:67:4f:91:a0'
  management:
    mac: '00:1e:67:4f:91:a2'
  comment: ""
```

State dnsmasq.hosts

```
dnsmasq_dhcp_hosts:
    file.managed:
        - name: /etc/dnsmasq.d/hosts.conf
        - user: root
        - group: root
        - mode: 0644
        - template: jinja
        - source: salt://{{ tpldir }}/templates/etc/dnsmasq.d/hosts.conf
        - context:
            nodes: {{ salt['pillar.get']('nodes_' ~ ClusterName) }}
        - watch_in:
            - service: dnsmasq_service
        - require:
            - pkg: dnsmasq_pkg
```

Template

```
# ethernet: nodes
{% for node in nodes_active -%}
{%   if ( node.name is match('^(cpu|gpu|chassis).*') ) and
       ( node.ethernet is defined ) -%}
dhcp-host={{ node.ethernet.mac }},{{ node.name }},set:internal_nodes
{%   endif -%}
{% endfor %}
```

result

```
# ethernet: nodes
dhcp-host=00:1e:67:4f:99:b0,cpu001,set:internal_nodes
dhcp-host=00:1e:67:4f:91:a0,cpu002,set:internal_nodes
dhcp-host=00:1e:67:4f:9a:10,cpu003,set:internal_nodes
```

Salt highstate

```
base:
  '(app|gpu|login|master|monitor|node|scratch|server).*':
    - match: pcre
    - admin
    - hosts
   [... ]
  '(node|gpu).*':
    - match: pcre
    - autofs
  '(node|gpu|login).*':
    - match: pcre
    - pkgs
    - beegfs_client
  "login1.cluster":
    - galaxy
  '(server|scratch).*':
    - match: pcre
    - beegfs_meta
    - beegfs_storage
```

Salt highstate

```
[root@master ~]# salt node002.cluster state.highstate --state-output=changes --state-verbose=False test=True
node002.cluster:
----------
          ID: systemd_daemon-reload
    Function: cmd.run
        Name: systemctl daemon-reload
      Result: None
     Comment: Command "systemctl daemon-reload" would have been executed
     Started: 11:08:47.879536
    Duration: 0.68 ms
     Changes:
----------
          ID: hosts_files
    Function: file.recurse
        Name: /etc
      Result: None
     Comment: #### /etc/hosts.equiv ####
              The file /etc/hosts.equiv is set to be changed
     Started: 11:08:48.027571
    Duration: 209.01 ms
     Changes:
----------
          ID: pkgs_compute
    Function: pkg.installed
      Result: None
     Comment: The following packages would be installed/updated: tig
     Started: 11:08:52.146443
    Duration: 52.193 ms
     Changes:
----------
          ID: rpcgssd
    Function: service.disabled
      Result: None
     Comment: Service rpcgssd set to be disabled
     Started: 11:08:53.961754
    Duration: 24.607 ms
     Changes:

Summary for node002.cluster
--------------
Succeeded: 321 (unchanged=4)
Failed:      0
--------------
Total states run:     321
Total run time:     8.116 s
```

Salt highstate

```
                     @@ -517,3 +524,5 @@
                      node234i.cluster
                      node235i.cluster
                      node236i.cluster
                     +node237i.cluster
                     +node238i.cluster
----------
          ID: pkgs_compute
    Function: pkg.installed
      Result: True
     Comment: The following packages were installed/updated: tig
              The following packages were already installed: boost-python, stress, git, gnuplot, java-1.8.0-ope
ity, xorg-x11-server-Xvfb, opensm-libs, zsh
     Started: 11:11:52.996307
    Duration: 5667.6 ms
     Changes:
              ----------
              tig:
                  ----------
                  new:
                      2.4.0-1.el7
                  old:

Summary for node002.cluster
--------------
Succeeded: 321 (changed=3)
Failed:      0
--------------
Total states run:     321
Total run time:    19.114 s
[root@master ~]# salt node002.cluster state.highstate --state-output=changes --state-verbose=False test=False
```

node (re) installation workflow

- Add the node details to the inventory in pillar
  - This generates entries for DNS, DHCP
- node boot using PXE
  - unattended minimalist CentOS installation using kickstart (in Salt!)
- Salt installation in ks postinstallation script and basic admin states applied
- Reboot, highstate executed
- Reboot, a couple of applications launched as slurm jobs.
- If the final check is ok, the node is in production.

Future

- Add SLURM information to inventory
- Test the hightstate on regular basis
- Use environment (prod, dev etc).
- Access pillar and states from git directly

Cons

- Can be difficult to «revert» a state.
  - Ex: If you don’t install a package already installed, not erased.
- Not super fast

links

- Salt states reference: https://docs.saltproject.io/en/latest/ref/states/all/index.html

- Usefull tool in adition to Salt:
  - Parallel command execution: https://clustershell.readthedocs.io/en/latest/
  - IPMI console: https://github.com/dun/conman

Thanks!

Questions ?