dCache, Salt and Reclass

A.Pickford

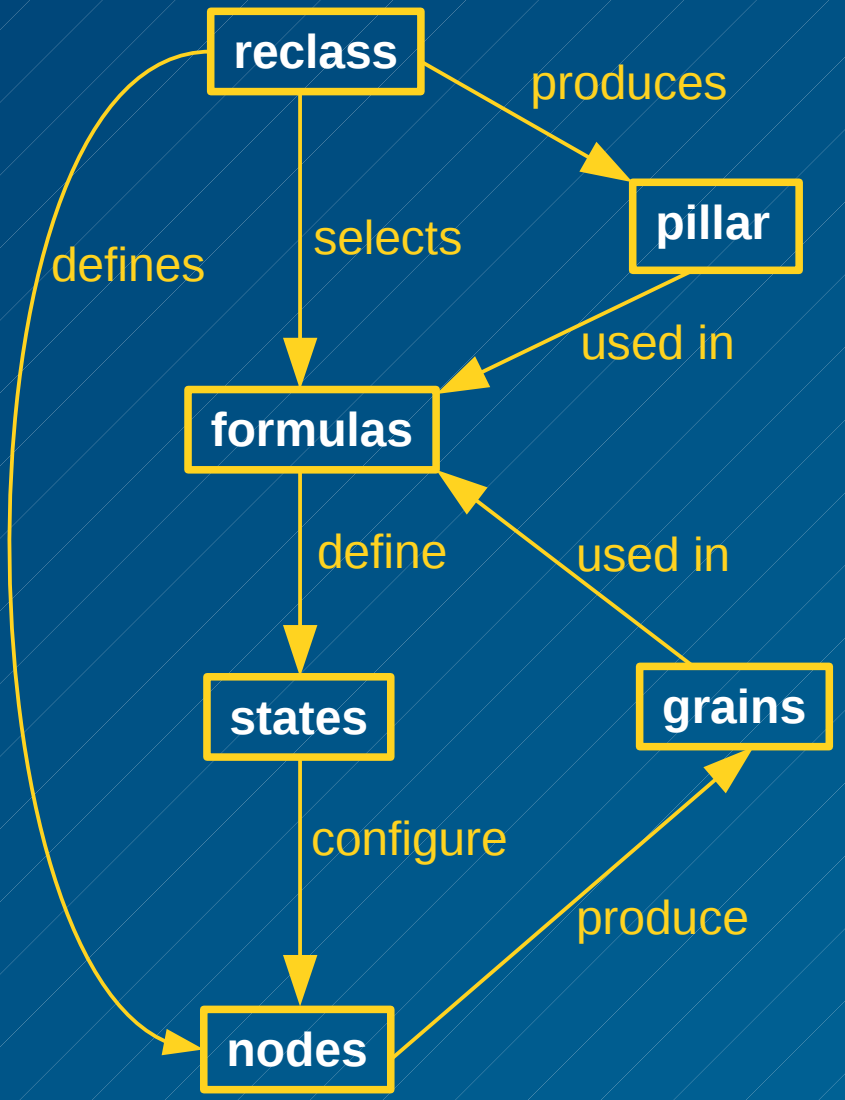
History

- quattor
  - used at Nikhef since early datagrid days
  - our installation
    - not updated since 2012
    - not compatible with centos 7
  - update to latest quattor or change
- considered ansible, chef, puppet, quattor upgrade, salt + others
  - difficult to really evaluate without running a full system
  - evaluated on
    - the usual: features, documentation, extensibility, ease of use
    - and also: programming language, personal prejudice
  - went with salt/reclass

Nikhef

Salt

- master / minion(s) configuration management
- coded in python, easily extended
- test mode, see what will happen
- minion controlled by defined states
  - logically bundled into formulas
  - implicitly ordered by dependencies
  - define how the node should be
    - file contents
    - installed software
    - running services
  - usually written in yaml + jinja2
- static configuration data (node pillar)
  - dictionary of values
  - yaml files or external source
    - we use reclass



Nikhef

Reclass

- external node classifier
  - defines nodes through class inheritance
  - override values further up the tree
  - yaml files plus reference syntax - ${}
- urls:
  - originally from reclass.pantsfullofunix.net
  - our extended fork github.com/AndrewPickford/reclass
  - and github.com/salt-formulas/reclass [develop branch]
- Nikhef implementation
  - uses four hierarchies: cluster, hardware, os, role
  - nodes belong to each level of a hierarchy
    - similar to the hierarchy of evolutionary clades (but we have four separate hierarchies)
    - eg os.linux.redhat.centos.7 (os/linux/redhat/centos/7/init.yml)
  - lose coupling between hierarchies

Nikhef

Reclass Node Definitions

```
# storage node – vaars-08.nikhef.nl.yml
classes:
  - cluster.ndpf.testbed.dcache
  - hardware.vm.xen.standard
  - os.linux.redhat.centos.7
  - role.server.dcache.plain.storage
environment: pre-prod
parameters:
  _hardware_:
    xen_pool: piet
    xen_sr: low-iops-4
    network_interfaces:
      eth0:
        mac_address: 0a:4e:c2:ab:61:75
        address: 194.171.97.117
        ipv6_address: 2001:610:120:e030::75

# door node – vaars-06.nikhef.nl.yml
classes:
  - cluster.ndpf.testbed.dcache
  - hardware.vm.xen.standard
  - os.linux.redhat.centos.7
  - role.server.dcache.plain.door
environment: pre-prod
parameters:
  _hardware_:
    xen_pool: piet
    xen_sr: low-iops-5
    network_interfaces:
      eth0:
        mac_address: 0a:4e:c2:ab:61:73
        address: 194.171.97.115
        ipv6_address: 2001:610:120:e030::73
```

- Each node is defined in a node file
  - yaml format
  - list of additional files (classes) to load
  - node environment (maps to salt environment)
  - parameters unique to the node
  - node files contain only a small amount of data
    - the class files contain most of the config data
    - very simple to add additional nodes

Nikhef

Cluster Classes I

```
# cluster/ndpf/testbed/dcache/init.yml
classes:
  - cluster.ndpf.testbed
exports:
  dcache:
    cluster: ${_cluster_:dcache_cluster}
parameters:
  _cluster_:
    name: dcache testbed
    dcache_cluster: testbed
    dcache_version: 3.1
    dcache_carbon_server: ${_cluster_:monitoring_satellite}
    dcache_database_hosts:
      alarms: vaars-07.nikhef.nl
      billing: vaars-07.nikhef.nl
      chimera: vaars-06.nikhef.nl
      pinmanager: vaars-06.nikhef.nl
      spacemanager: vaars-06.nikhef.nl
    dcache_database:
      password: XXXXXXXXXX
    dcache_nfs_allowed_ipv4:
      - ${_site_:network_groups:ipv4:ndpf}
      - ${_site_:networks:ipv4:stbcnet}
    dcache_nfs_allowed_ipv6:
      - ${_site_:network_groups:ipv6:ndpf}
      - ${_site_:networks:ipv6:stbcnet}
    dcache_remote_users:
      - andrewp
    dcache_zookeeper_connection: vaars-05.nikhef.nl:2181
    postgresql_version: 9.6
    postgresql_admin_password: XXXXXXXXXX
    zookeeper_members:
      - host: vaars-05.nikhef.nl
        id: 1
```

classes to load evaluated before this class
- classes are loaded once when first seen

exported parameters, available to other nodes

reference

reference value

which nodes do what in the cluster

networks allowed nfs access
- references, protects against typos, standardises network definitions

passwords, config files must be private

Nikhef

Cluster Classes II

```
# cluster/ndpf/testbed/init.yml
classes:
  - cluster.ndpf
parameters:
  _cluster_:
    name: testbed
    monitoring_satellite: vaars-03.nikhef.nl
    autofs:
      master:
      mapfiles:
        dcache:
          location: vaars-06.nikhef.nl:/dcache
    ~nameservers:
      - 194.171.97.130
      - 194.171.97.131
...


# cluster.ndpf
classes:
  - cluster
  - site.networks
  - site.networks.trusted
parameters:
  _cluster_:
    login_networks_ipv4:
      - ${_site_:networks:ipv4:pubgrid}
    login_networks_ipv6:
      - ${_site_:networks:ipv6:pubgrid}
...
```

each class loads the class above it in the hierarchy

cluster:name defined here, but overridden in cluster.ndpf.testbed.dcache

value referenced by _cluster_:dcache_carbon_server

autofs config for testbed clients to access dcache

override (ie replace not append) the list of nameservers - the testbed uses testbed nameservers

classes define _site_:networks keys

very general configs
- such as where to allow logins from, applies to the whole of the ndpf cluster not just the testbed

Nikhef

Role Classes I

```
# role/server/dcache/plain/door/init.yml
classes:
  - role.server.dcache.plain
  - role.server.dcache._gplazma.gplazma-nfs
  - role.server.dcache._iptables.nfs-restricted
parameters:
  _role_:
    name: dCache door
    dcache_nfs_domain_memory:
      heap: 4096m
      direct: 2048m
  dcache:
    domains:
      nfs:
        name: ${_system_:name}nfs
        options:
          'dcache.java.memory.heap': ${_role_:dcache_nfs_domain_memory:heap}
          'dcache.java.memory.direct': ${_role_:dcache_nfs_domain_memory:direct}
        services:
          gplazma:
            'gplazma.cell.name': gplazmaNFS
            'gplazma.cell.consume': gplazmaNFS
            'gplazma.configuration.file': '\${dcache.paths.etc}/gplazma-nfs.conf'
          nfs:
            'nfs.service.gplazma': gplazmaNFS
            'nfs.version': 4.1
            'nfs.db.connections.max': 200
            'nfs.cell.max-message-threads': 32
            'nfs.cell.max-messages-queued': 4000
            'nfs.service.pool.timeout': 30000
            'nfs.namespace-cache.time': 5
            'nfs.namespace-cache.size': 1024
...
```

default values for java memory config
- referenced here
- can be overridden by later classes before the references are evaluated

domains on this node

domain name
- overrides key name

services in this domain

escaped $, not a reference, instead a a literal ${

configuration options for the service

Nikhef

Role Classes II

```
# role/server/dcache/_iptables/nfs-restricted.yml
parameters:
  ipset:
    sets:
      dcache_nfs_v4:
        family: inet
        members: ${_cluster_:dcache_nfs_allowed_ipv4}
  iptables:
    ipv4:
      chains:
        INPUT:
          rules:
            - rule: '-m set --match-set dcache_nfs_v4 src -j DCACHE-NFS'
        DCACHE-NFS:
          rules:
            - rule: '-p tcp --dport 2049 -j ACCEPT'
            - rule: '-p tcp --dport 33115:33145 -j ACCEPT'
```

ipset config, creates a list based on the _cluster_:dcache_nfs_allowed_ipv4 key defined in cluster.ndpf.testbed.dcache

use the ipset list in iptables to allow traffic in only fromallowed machines to the required ports

these rules get merged with all other ipset/iptables rules

query export parameters of other nodes
- evaluates to a dictionary of node name, ipv4 address pairs
- eg: { vaars-06.nikhef.nl: 194.171.97.115
  vaars-08.nikhef.nl: 194.171.97.117
  …. }

```
# role/server/dcache/_iptables/allow_dcache_cluster.yml
parameters:
  ipset:
    sets:
      dcache_cluster_v4:
        family: inet
        members: $[exports:host:ip_address:ipv4 if exports:dcache:cluster == self:_cluster_:dcache_cluster]
  iptables:
    ipv4:
      chains:
        INPUT:
          rules:
            - rule: '-m set --match-set dcache_cluster_v4 src -j DCACHE-CLUSTER-INPUT'
```

Nikhef

Service Class

```
# role.server.dcache
classes:
  - role.server
  - service.dcache
parameters:
  _role_:
    dcache_admins: ${_cluster_:admins}
...
```

set the list of dcache admins to the list of cluster admins

```
# service/dcache/init.yml
applications:
  - dcache
parameters:
  dcache:
    layout: ${_system_:name}
    nfs_domain: ${_cluster_:domain}
    pool_size_units: K
    zookeeper:
      connection: ${_cluster_:dcache_zookeeper_connection}
    database_hosts: ${_cluster_:dcache_database_hosts}
    database:
      user: dcache
      password: ${_cluster_:dcache_database:password}
    pool_setup: false
```

add the dcache formula to the list of formulas to use when running salt

by default the pool setup part of the dcache state doesn't run
- pure paranoia: not enough experience with salt at the time to trust it not to reformat partitions

Nikhef

Reclass Output

```
dcache:
  admin_data_pillar: site:admins
  database:
    password: XXXXXXXXXXXX
    user: dcache
  database_hosts:
    alarms: vaars-07.nikhef.nl
    billing: vaars-07.nikhef.nl
    chimera: vaars-06.nikhef.nl
    pinmanager: vaars-06.nikhef.nl
    spacemanager: vaars-06.nikhef.nl
  databases:
  - chimera
  - pinmanager
  - spacemanager
  domains:
    ftp:
      name: vaars-06ftp
      services:
        ftp:
          ftp.authn.hostcert.cert: /etc/ssl/localcerts/dcache-cert.pem
          ftp.authn.hostcert.key: /etc/ssl/localcerts/dcache-key.pem
          ftp.authn.protocol: gsi
          ftp.authz.readonly.gsi: 'false'
          ftp.authz.readonly.kerberos: 'true'
          ftp.authz.readonly.plain: 'true'
          ftp.limits.clients: 1000
          ftp.net.port.gsi: 2811
          ftp.service.gplazma: gplazmaftp
        gplazma:
          gplazma.authzdb.file: ${dcache.paths.etc}/authzdb
          gplazma.cell.consume: gplazmaftp
          gplazma.cell.name: gplazmaftp
          gplazma.configuration.file: ${dcache.paths.etc}/gplazma-xrootd.conf
 ...
```

- partial reclass output for testbed storage node
  - dictionary of key value pairs
  - passed into salt as the pillar data for the node

Nikhef

Salt Formula

```
# dcache/init.sls
{%- if pillar.dcache is defined %}
include:
- dcache.server
{%- if pillar.dcache.logstash is defined %}
- dcache.logstash
{%- endif %}
{%- if pillar.dcache.graphite_monitoring is defined %}
- dcache.graphite
{%- endif %}
{%- endif %}
```

state files mostly writen in yaml + jinja2

only do something if dcache pillar key is defined

include sub state file

```
# dcache/map.jinja
{%- set dcache = salt['grains.filter_by']({
    'default': {
        'enabled': False,
        'service': 'dcache-server',
        'user': 'dcache',
        'group': 'root',
        'conf_dir': '/etc/dcache',
        'logstash_dir': '/etc/logstash',
    },
    'RedHat': {
        'pkgs': [ 'dcache', 'java-1.8.0-openjdk' ],
    },
}, merge=salt['pillar.get']('dcache'), base='default') %}
```

salt filter_by function
- merge pillar data on top of given keys
- first merge in the 'default' key (base option)
- second merge in an os specific dictionary (or the 'default' entry if not present)
- finally merge in the pillar data in the dcache key

Nikhef

Server States (Packages, Service, Files)

```
# dcache/server.sls
{%- from "dcache/map.jinja" import dcache with context %}

dcache_packages:
  pkg.installed:
  - names: {{ dcache.pkgs }}

dcache_service:
  service.running:
  - enable: true
  - name: {{ dcache.service }}
  - watch:
    - pkg: dcache_packages
{%- if dcache.pools is defined %}
{%- set pool = dcache.pools[0] %}
  - onlyif: "test `test -d '{{ pool.dir }}/{{ pool.name }}' >/dev/null;echo $?` -eq 0"
{%- endif %}

dcache_conf:
  file.managed:
  - name: {{ dcache.conf_dir }}/dcache.conf
  - source: salt://dcache/files/dcache.conf
  - template: jinja
  - user: {{ dcache.user }}
  - group: {{ dcache.group }}
  - mode: 640
  - watch_in:
    - service: dcache_service
```

merge pillar data with default values and put result into a jinja variable

install dcache packages

ensure dcache service is running
- but not if pools are defined and the first pool does not exist

manage files using file templates
- destination
- template name (from file server on salt master)
- restart dcache service if file changes

values from mapfile defaults, overwritable by pillar data

Nikhef

File Template - dcache.conf

```
# salt://dcache/files/dcache.conf
{%- from "dcache/map.jinja" import dcache with context -%}

dcache.layout = {{ dcache.layout }}
nfs.domain = {{ dcache.nfs_domain }}
dcache.zookeeper.connection = {{ dcache.zookeeper.connection }}

dcache.db.user = {{ dcache.database.user }}
dcache.db.password = {{ dcache.database.password }}

alarms.db.host = {{ dcache.database_hosts.alarms }}
billing.db.host = {{ dcache.database_hosts.billing }}
chimera.db.host = {{ dcache.database_hosts.chimera }}
pinmanager.db.host = {{ dcache.database_hosts.pinmanager }}
spacemanager.db.host = {{ dcache.database_hosts.spacemanager }}

dcache.java.memory.heap = {{ dcache.java.memory.heap }}
dcache.java.memory.direct = {{ dcache.java.memory.direct }}
```

template file
- master provides the template
- minion parses file through jinja before writing to disc

```
dcache.layout = vaars-06
nfs.domain = nikhef.nl
dcache.zookeeper.connection = vaars-05.nikhef.nl:2181

dcache.db.user = dcache
dcache.db.password = XXXXXXXXXXX

alarms.db.host = vaars-07.nikhef.nl
billing.db.host = vaars-07.nikhef.nl
chimera.db.host = vaars-06.nikhef.nl
pinmanager.db.host = vaars-06.nikhef.nl
spacemanager.db.host = vaars-06.nikhef.nl

dcache.java.memory.heap = 1024m
dcache.java.memory.direct = 1024m
```

Nikhef

Server States (Database, Pools)

```
{%- for name in dcache.get('databases', []) %}
dcache_setup_database_{{ name }}:
  cmd.run:
  - name: dcache database update
  - require:
    - pkg: dcache_packages
  - onchanges:
{%- for db in dcache.databases %}
      - postgres_database: postgresql_database_localhost_{{ db }}
{%- endfor %}
  - watch_in:
    - service: dcache_service
{%- endfor %}

{%- if dcache.pool_setup %}
{%- for pool in dcache.get('pools', []) %}
dcache_pool_create_{{ pool.name }}:
  cmd.run:
  - name: dcache pool create {{ pool.dir }}/{{ pool.name }}
{{ pool.name }} {{ pool.name }}Domain
  - onlyif: "test `test -d '{{ pool.dir }}/{{ pool.name }}'
            >/dev/null;echo $?` -eq 1"
  - require:
    - pkg: dcache_packages
  - require_in:
    - file: dcache_layout_conf
    - service: dcache_service

dcache_pool_config_{{ pool.name }}:
  file.managed:
  - name: {{ pool.dir }}/{{ pool.name }}/setup
  - source: salt://dcache/files/pool-setup
  - template: jinja
  - mode: 644
  - required:
    - cmd: dcache_pool_create_{{ pool.name }}
{%- endfor %}
{%- endif %}
```

run dcache database update to setup databases, no simple statefull way to know if the command should be run

only if the database has been just created in the postgresql formula
- cross formulas, not ideal
- updates have to be handled manually

only run pool setup states if flag is set

use dcache pool create cmd, otherwise need to work out what command does

only if the pool directory does not exist
- use the existence of the directory as a state test

manage the pool layout file

Nikhef

Summary

- automated most of the dcache installation:
  - software packages, services, monitoring, intinal database setup
  - pool creation (states need to be run manually)
- still manual:
  - upgrades
- keeps testbed and production systems as similar as possible
  - reliably test changes
- formula on github
  - https://github.com/NDPF/nikhef-formula-dcache

Nikhef