dCache automation at DESY.

on behalf of the DESY dCache Operations Team - especially Christian Voss

Thomas Hartmann

12th int. dCache Workshop

2018.May.28

HELMHOLTZ
RESEARCH FOR GRAND CHALLENGES

What is *automation*?

Automation is

"separation of data and implementation plus version control"

Our (Christian's) Intentions

Automation: make lifes easier for administrators

- admins can centrally manage multiple installations
- admins can concentrate on the installation
- allow easy scaling
  - setting up 10 pools or 100 pools $\sim$ same work
  - make upgrading indenpendent of scale
- make testing and upgrading easier and more fault-tolerant

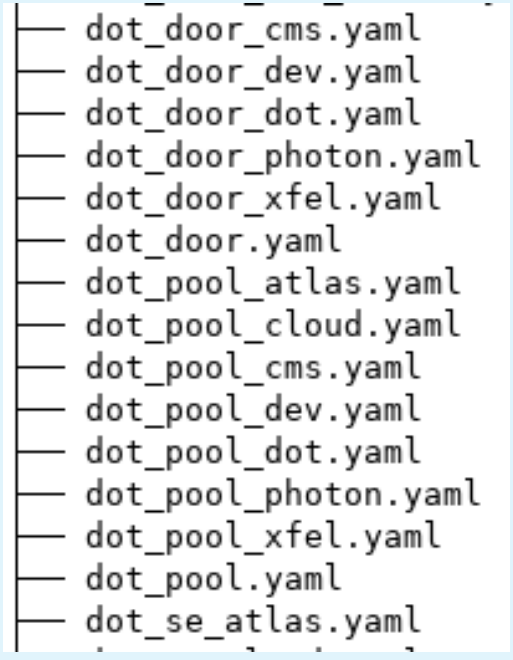
Our Intentions

Automation: make lifes easier for administrators

- admins can centrally manage multiple installations
- admins can concentrate on the installation
- allow easy scaling
  - setting up 10 pools or 100 pools $\sim$ same work
  - make upgrading indenpendent of scale
- make testing and upgrading easier and more fault-tolerant
  1. branch config from the current production branch
  2. make changes $\circlearrowleft$ apply to **test instance**
  3. merge and apply into production

What is the admin supposed to manage

just the data (in our case in stored in Hiera.yamls)



What is the admin supposed to manage

just the data: e.g., common settings for all (CMS) Doors

```
zookeeper::server_name_1 : 'dcache-dir-cms.desy.de'
zookeeper::server_name_2 : 'dcache-core-cms.desy.de'
zookeeper::server_name_3 : 'dcache-se-cms.desy.de'

swcol::additional :
  - dcache-plugin-xrootd-monitor
  - xrootd4j-cms-plugin

dcache::dcache_instance : 'cms'
dcache::fetch_certificates : true
dcache::enable_wlcg: true
dcache::ipv6 : true

dcache::dcache_conf:
   dcache:
     chimera.db.host : 'dcache-dir-cms.desy.de,dcache-core-cms.desy.de'
     dcache.java.memory.direct : '512m'
     dcache.java.memory.heap : '2048m'
   network:
     dcache.net.lan.port.max : '33215'
     dcache.net.lan.port.min : '33115'
     dcache.net.wan.port.max : '23900'
     dcache.net.wan.port.min : '20000'
...
```

What is the admin supposed to manage

Implementation

let an automaton handle the actual implementation

What is the admin supposed to manage

Implementation

let an automaton handle the actual implementation
including all dependencies

What is the admin supposed to manage

Implementation

let an automaton handle the actual implementation
including all dependencies
(and have somebody else already implemented it)

DESY Setup: Basics

Fundament

- manifold IT infrastructure
  $\rightsquigarrow$ automated deployement & management inevitable
- system provisioning: `Foreman`
- software management: `Puppet`
  - version control: `git`

DESY Setup: Hostgroups

Hostgroups as base structure

- organizing services
  - dCache
  - Grid Services
  - ...

Details in modules

- hostgroups include specific modules

e.g. dCache:

- auth
- door
- ...

DESY Setup: dCache

dCache nodes organized by their *roles* aka domains

- *-core-*
  - core, admin
- *-se-*
  - SRM, info, gplazma,...
- *-dir-*
  - namespace, NFS4, cleaner
- *-door-*
  - gridftp/xrootd/dcap/webdav
- *-name##*
  - pools...

(fixed role attribution becoming more relaxed with HA)

dcache_operations hostgroup & module

Hostgroup files

```
hostgroups/hg_dot/manifests/
├── base.pp
├── billing.pp
├── core.pp
├── dir.pp
├── door.pp
├── ha_head.pp
├── head.pp
├── mon.pp
├── pool.pp
└── se.pp
```

- Manage everything not related to dcache
- Manage zookeeper
- Setup dcache_operations module

Module files

```
modules/dcache_operations/manifests
├── array
│   ├── config.pp
│   ├── install.pp
│   └── service.pp
├── array.pp
├── auth
│   ├── admin_acl.pp
│   ├── authorized_keys.pp
│   ├── gplazma.pp
│   ├── grid_vrolemap.pp
│   ├── gss.pp
│   ├── multi_map.pp
│   └── storage_authzdb.pp
├── basic_install.pp
├── config
│   ├── dcache_conf.pp
│   ├── dcache_layout.pp
│   ├── global.pp
│   ├── nfs_exports.pp
│   ├── pool_layout.pp
│   ├── postgres.pp
│   └── resilience_monitoring.pp
├── door_install.pp
├── head_install.pp
├── init.pp
├── pool_install.pp
└── service.pp
```

- Setup for 60 disk storage boxes
- Administrator login
- External access
- dcache configuration and layouts
- Database configuration
- Initialise and wrapper classes

dCache installations

dCache Installations at DESY

- six production installations
  - ATLAS, CMS, XFEL, Cloud, Photon, Misc. (i.e. DESY)
- single test installation
  - Dot

Layout per Installation

- four dedicated head nodes serving the database and services
- varying amount of doors with different protocols
- varying amount of heterogeneous pool hosts

Nodes

Head Nodes: `dcache-{dir,core,se,billing}`

- run `dCache` each with a different layout
- run a `postgres` database each
- run a `zookeeper` on (`dcache-dir`, `dcache-core`, `dcache-se`)
- `NFS`-Server (usually `dcache-dir` only)

Door and Pool Nodes: `dcache-door-NAME{01..99}`

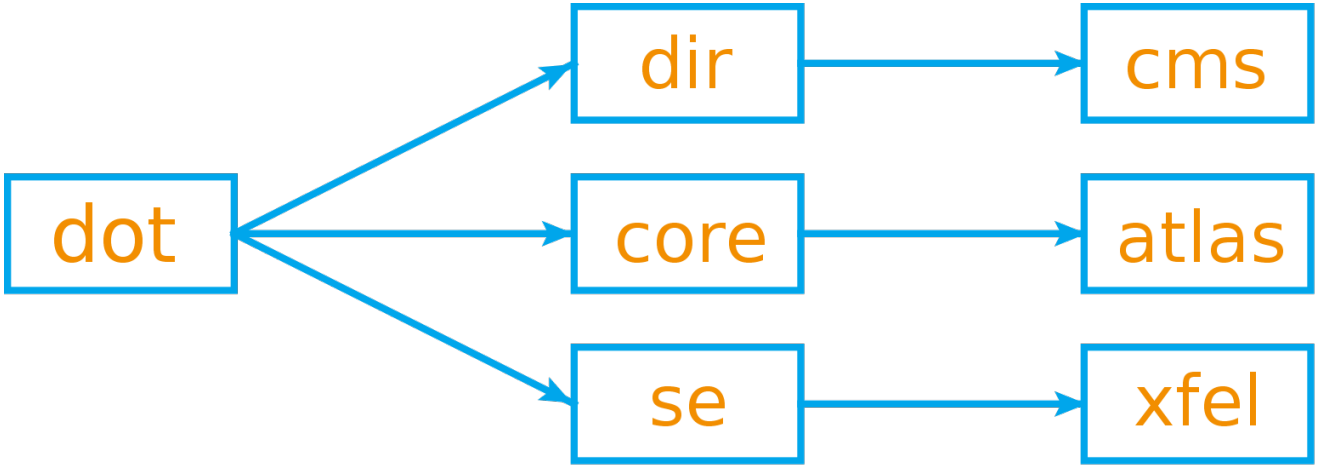
- $\geq$1 doors per protocol
- $>$1 pools per Pool Node RAID set for faster initialization
- easily scale horizontally, *i.e.*, need more storage: ++pool nodes
- fully automated installation and configuration

Implementation Details

Implementation Details

Hostgroup Setup

Take advantage of pseudo-inheritance in our hostgroups



Most fundamental

- Logins to `dCache`
- `dCache` release
- `postgres` release
- `zookeeper` release
- `java` release

Central role

- General part of `dcache.conf`
- General `layout.conf`
- `postgres` configuration
- `zookeeper::id`

Installation specifics

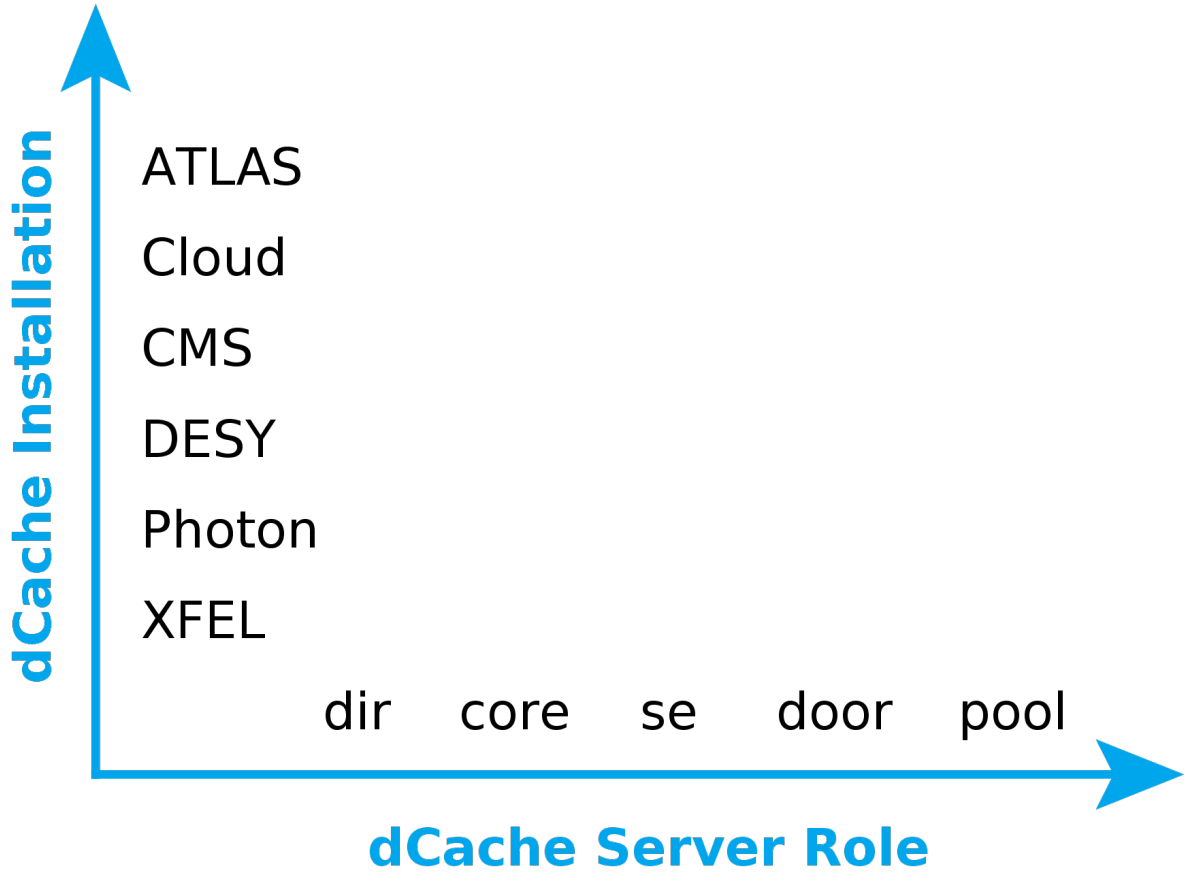
- Additional `dCache` services
- Instance specific variables
- Authorisation

Hostgroup Awareness

Pseudo-inheritance reduces code duplication

> Change `dCache` version in only one place → decrease update workload

Hierarchy can be multi-dimensional

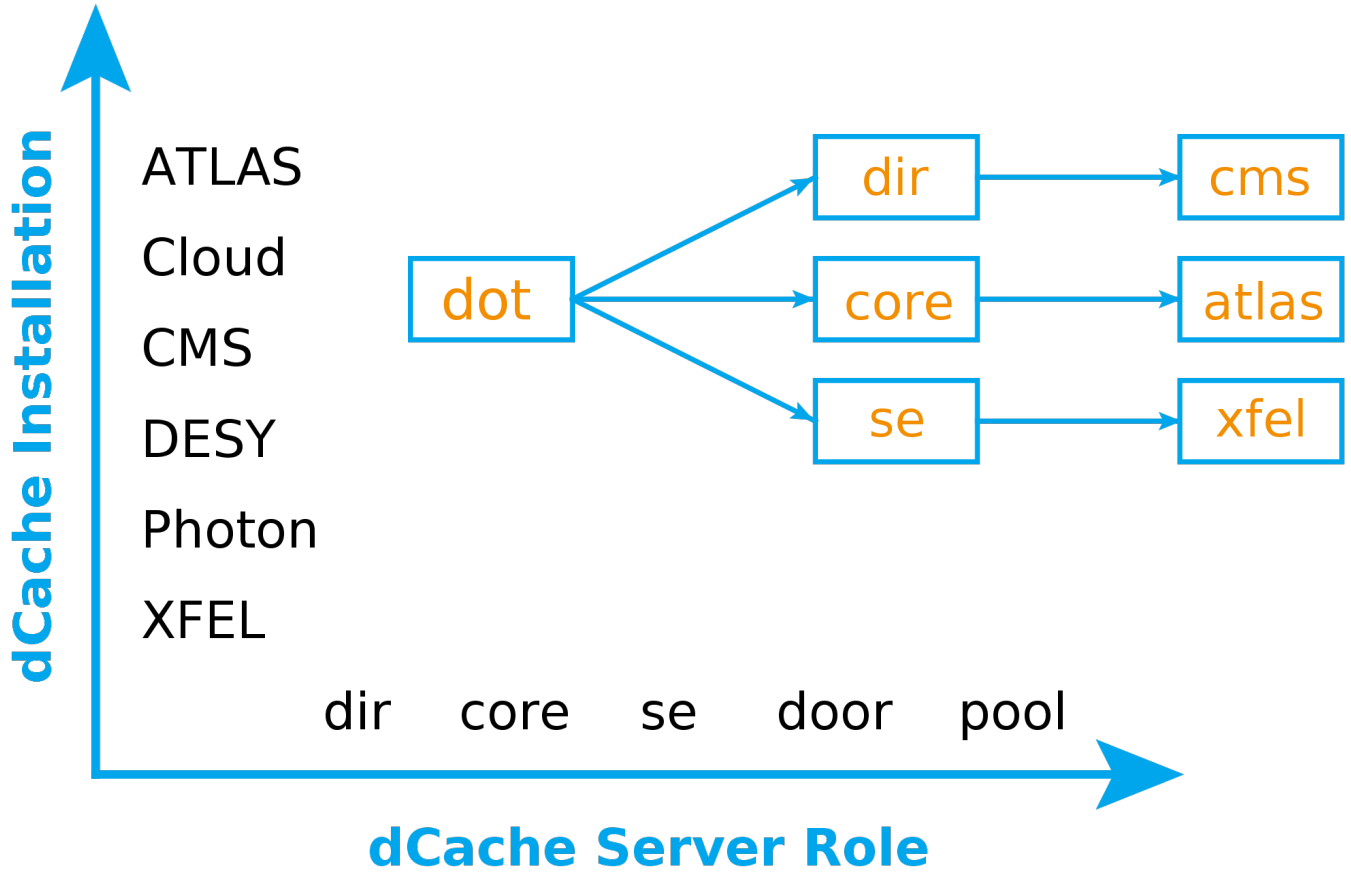


Hostgroup Awareness

Pseudo-inheritance reduces code duplication

> Change `dCache` version in only one place → decrease update workload

Hierarchy can be multi-dimensional

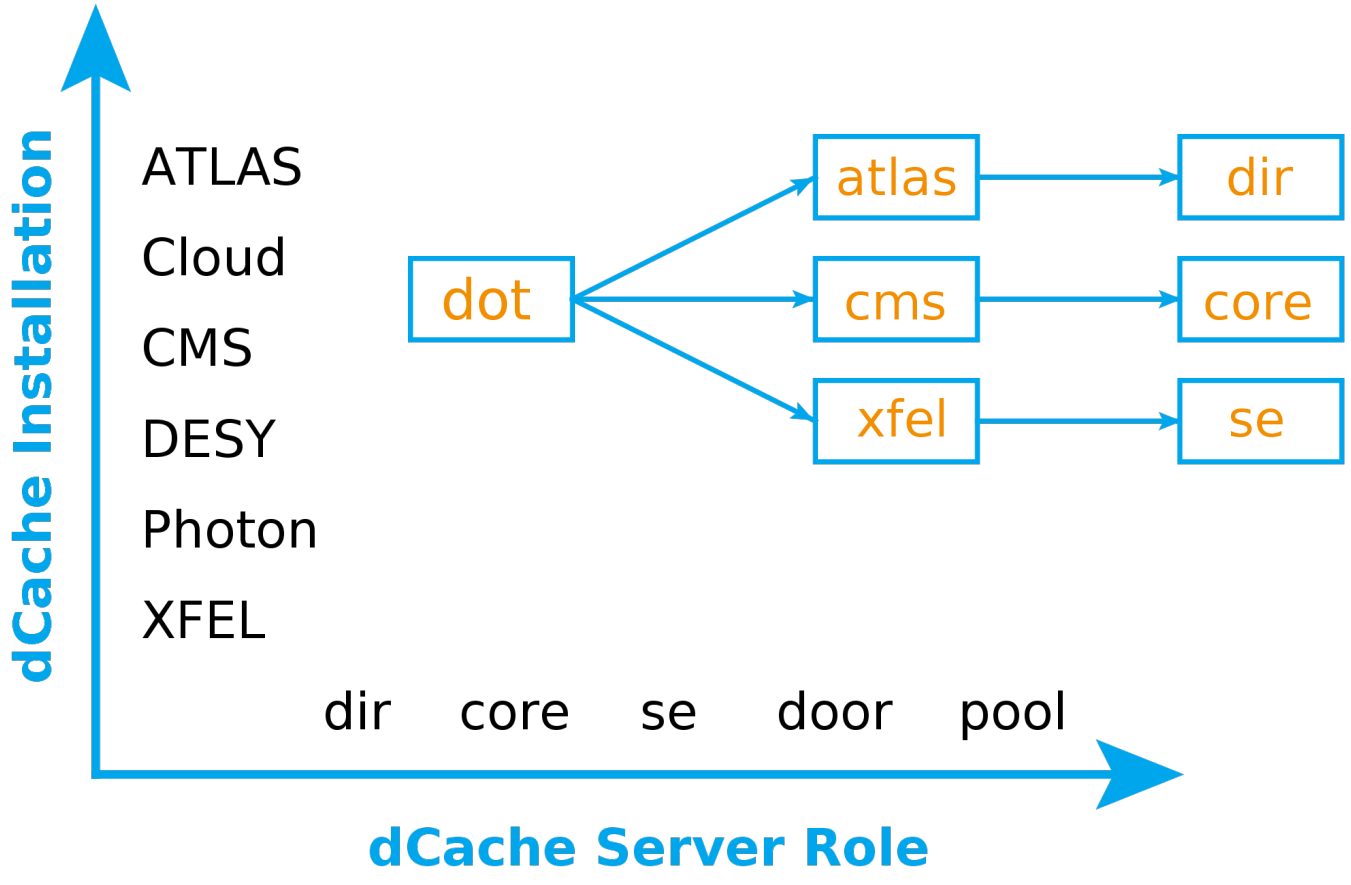


Hostgroup Awareness

Pseudo-inheritance reduces code duplication

> Change `dCache` version in only one place → decrease update workload

Hierarchy can be multi-dimensional

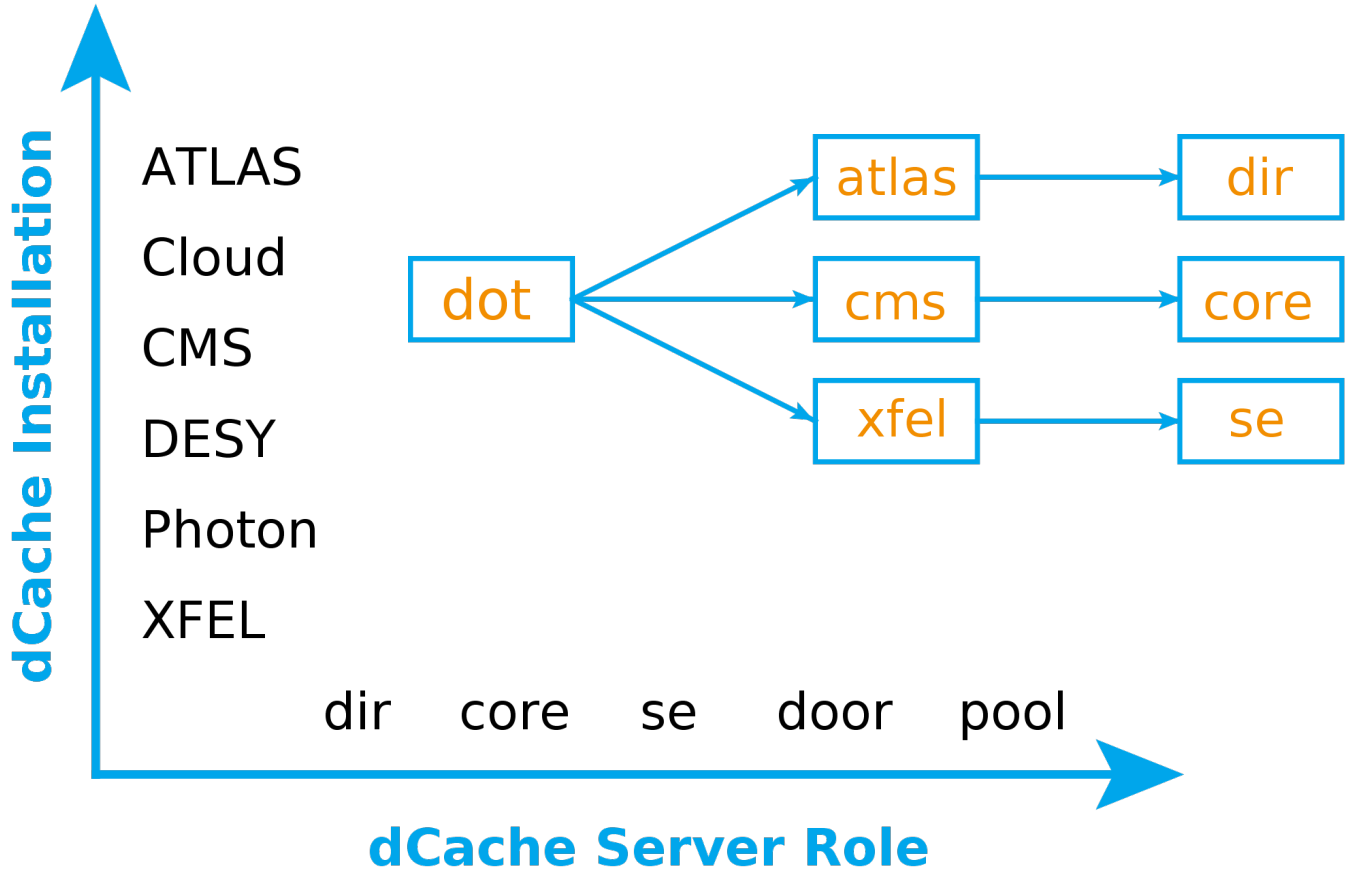


Hostgroup Awareness

Pseudo-inheritance reduces code duplication

> Change `dCache` version in only one place → decrease update workload

Hierarchy can be multi-dimensional → choose your hierarchy wisely



Example

Example: Data vs Implamentation

Example: Data

Admin view: Data in Hiera, e.g.,

```
dcache::dcache_layout:
  domains:
    coreDomain:
      domainsettings:
        dcache.broker.scheme : 'core'
      poolmanager:
    adminDoorDomain:
      admin:
        admin.authz.gid : '1000'
```

Example: Implementation

Implementation in Puppet (*hidden* from the admin),e.g.,

```
...
  if ( $dcache_layout ) {
    class {'::dcache_operations::config::dcache_layout':
      dcache_layout => $dcache_layout,
      ha_mode       => $ha_mode
    }
...
```

```
class dcache_operations::config::dcache_layout ($dcache_layout, $ha_mode){...}
```

```
...
<% @layout_hash['domains'].sort.each do |d_name,d_value| -%>
<% if d_name == "dCacheDomain" -%>
[<%= d_name %>]
<% if !d_value['comments'].nil? -%>
<% d_value['comments'].each do |comment| -%>
# <%= comment %>
<% end -%>
<% end -%>
...
```

Example: Result

Resulting config after Puppet run, e.g.,

```
#####################################################################
##
## DO NOT EDIT
## generated by puppet module :  dcache_operations::config::dcache_layout
## node                       :  dcache-core-cms03.desy.de
##
#####################################################################

[${host.name}_adminDoorDomain]
[${host.name}_adminDoorDomain/admin]
admin.authz.gid=1000

[${host.name}_coreDomain]
dcache.broker.scheme=core
[${host.name}_coreDomain/poolmanager]
```

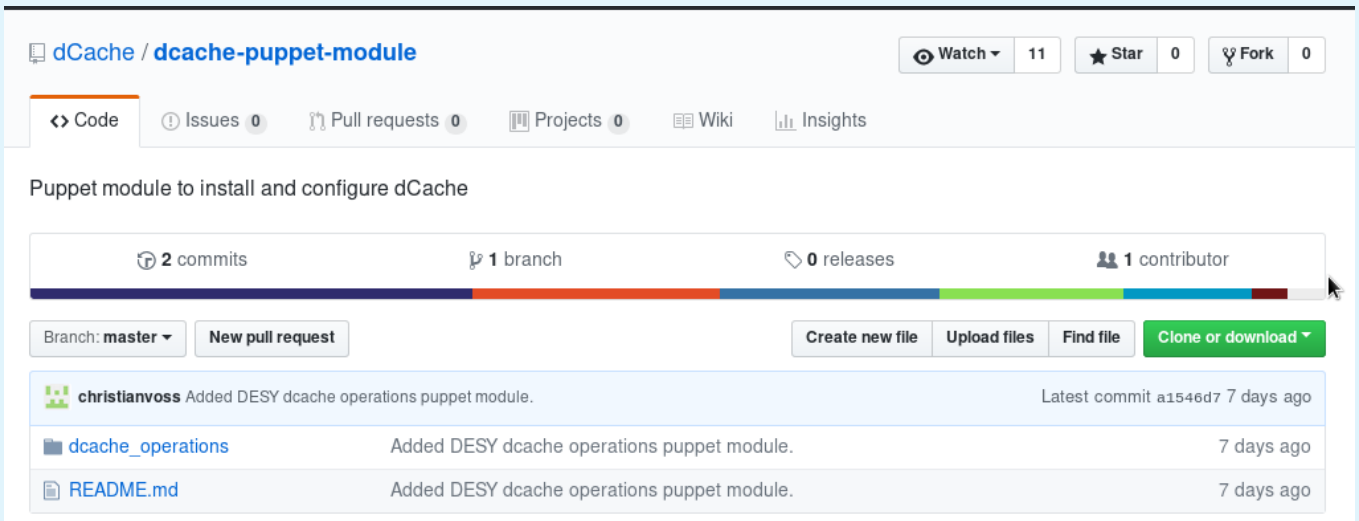
Conclusions

Conclusions

Puppet modules available on Github

Pull from Github

- `https://github.com/dCache/dcache-puppet-module`
- we are interested in you to pull and integrate it into your environment
- ⇝ site-agnostic
  - variables setable via Hiera or directly in Puppet
  - input & comments welcome ☺



Concept of our setup

What do we want to achieve?

- Complete description of all nodes
  - "Those who can't use their head must use their `puppet`."
  - Easy migration to new hardware
  - Easy testing on the dot-instance/OpenStack
  - Comfort during `dcache` updates
- Make it administrator-friendly
  - Avoid administrators having to write `puppet` code
  - Keep `yaml` structure human readable
  - Take care of internal dependencies
  - Have the configuration of the instance in a compact form

Conclusion

Summary

- integrate dCache installations into Puppet/Foreman
- separate data from the actual implementation
- aim for modularity and portability
- administer `dCache` installations primarily by `hiera` variables
- minimise actual logins to head nodes (frequent due to safety)
- minimise load for installing the dumb parts, i.e., doors and pools
- minimise pit falls due to missing dependencies for certain services

Thanks to Christian

Please send some applause to Christian

He did all the work and is now (hopefully) relaxing on his holidays!

Appendix

Appendix

Different Configuration Concepts

Our puppet environment comes with

- foreman integration (setting hostgroups)
- hiera support

Use puppet only

- Write everything in puppet directly

Use hiera variables

- Write generalised puppet code defining external variables
- Set variables in hiera to be used in puppet

Puppet Only Configuration

Puppet comes with predefined resources, e.g. `service`, or 3rd party modules

Example: zookeeper configuration

```
class hg_dot::dir {
   class { 'zookeeper':
             id => '1',
          }
}
```

```
class hg_dot::core {
   class { 'zookeeper':
             id => '2',
          }
}
```

```
class hg_dot::se {
   class { 'zookeeper':
             id => '3',
          }
}
```

```
class hg_dot::dir_atlas {
   class { 'zookeeper':
             id => '1',
             servers => [ 'dcache-dir-atlas', 'dcache-core-atlas', 'dcache-se-atlas'],
          }
}
```

```
class hg_dot::dir_cms {
   class { 'zookeeper':
             id => '1',
             servers => ['dcache-dir-cms', 'dcache-core-cms', 'dcache-se-cms'],
          }
}
```

Use hiera variables

- Using Puppet only does not scale in complex hierarchies
- Solution: hiera coming with our local puppet setup

Example: zookeeper configuration revisited

```
class hg_dot::head {
   class { 'zookeeper':
             servers => {
               1 => hiera("dcache::${dcache_instance}::zookeeper::server_name_1","localhost"),
               2 => hiera("dcache::${dcache_instance}::zookeeper::server_name_2","localhost"),
               3 => hiera("dcache::${dcache_instance}::zookeeper::server_name_3","localhost"),
             }
         }
}
```

- Store values in hiera variables contained in yaml-formatted hostgroup files

Need five variables but only a single puppet class

- zookeeper::id, dcache_instance, zookeeper::server_name{1,2,3}

Hiera yaml Files

How do the hostgroup yaml look like?

> Assume hostgroup dot/dir/cms (assigned via foreman)

Store zookeeper::server_name{1,2,3} in dot.yaml

```
dcache::cms::zookeeper::server_name_1 : 'dcache-dir-cms.desy.de'
dcache::cms::zookeeper::server_name_2 : 'dcache-core-cms.desy.de'
dcache::cms::zookeeper::server_name_3 : 'dcache-se-cms.desy.de'
```

Assign hostgroup and perform foreman match in dot_dir.yaml

```
hostgroups:
        - hg_dot::dir
# dcache-dir is always node 1
zookeeper::id: '1'
```

Assign dCache installation in dot_dir_cms.yaml

```
dcache::dcache_instance : 'cms'
```

Concept of the dot puppet setup

What do we want to achieve?

- Complete description of all nodes
  - "Those who can't use their head must use their `puppet`."
    – Mentality of myself
  - Easy migration to new hardware
  - Easy testing on the dot-instance/OpenStack
  - Comfort during `dcache` updates
- Make it administrator-friendly
  - Avoid administrators having to write `puppet` code
  - Keep `yaml` structure human readable
  - Take care of internal dependencies
  - Have the configuration of the instance in a compact form

Make it administrator-friendly

Avoid administrators having to write puppet code

- Add `NFS`-door to `dcache-core-cms`
  - Necessary to modify `/etc/dcache/layouts/dcache-core-cms03.conf`

```
[${host.name}_nfs4Domain]
nfs.cell.name=nfs4-wn
[${host.name}_nfs4Domain/nfs]
nfs.version=4.1
```

  - Prepare list of hosts to export the `dcache` storage in `/etc/exports`

```
/pnfs/desy.de/cms naf-school*.desy.de(ro,no_dcap)
/pnfs/desy.de grid-mon(ro,no_dcap)
```

  - Start the door
- Could be done by different `puppet` resources
  - Use `concat` to add a line to `/etc/exports`
  - Just a few lines to the `dot_core_cms.yaml` file

Just a few lines to the dot_core_cms.yaml file

- Run-of-the-mill core node → layout defined in dot_core.yaml

```
dcache::dcache_layout:
  domains:
    coreDomain:
      domainsettings:
        dcache.broker.scheme : 'core'
      poolmanager:
    adminDoorDomain:
      admin:
        admin.authz.gid : '1000'
```

- Modify dot_core_cms.yaml

```
dcache::dcache_layout:
  domains:
    nfs4Domain:
      domainsettings:
        nfs.cell.name : 'nfs4-wn'
      nfs:
       nfs.version : '4.1'
```

- The dcache_operations puppet modules merges both entries of dcache::dcache_layout into a single variable

Modified /etc/dcache/layouts/dcache-core-cms03.conf

> After a puppet run the result is

```
#########################################################################
##
## DO NOT EDIT
## generated by puppet module :  dcache_operations::config::dcache_layout
## node                       :  dcache-core-cms03.desy.de
##
#########################################################################

[${host.name}_adminDoorDomain]
[${host.name}_adminDoorDomain/admin]
admin.authz.gid=1000

[${host.name}_coreDomain]
dcache.broker.scheme=core
[${host.name}_coreDomain/poolmanager]

[${host.name}_nfs4Domain]
nfs.cell.name=nfs4-wn
[${host.name}_nfs4Domain/nfs]
nfs.version=4.1
```

For /etc/exports

- Add to dot_core_cms.yaml file

```
dcache::nfs_exports:
  '/pnfs/desy.de/cms':
     'naf-school*.desy.de' : '(ro,no_dcap)'
  '/pnfs/desy.de':
     'grid-mon'            : '(ro,no_dcap)'
```

- After a puppet run the result is

```
##########################################################################
##
## DO NOT EDIT
## generated by puppet module :  dcache_operations::config::nfs_exports
## node                       :  dcache-core-cms03.desy.de
##
##########################################################################
/pnfs/desy.de/cms naf-school*.desy.de(ro,no_dcap)
/pnfs/desy.de grid-mon(ro,no_dcap)
```

For /etc/exports

> Add to dot_core_cms.yaml file

```
dcache::nfs_exports:
  '/pnfs/desy.de/cms':
     'naf-school*.desy.de' : '(ro,no_dcap)'
  '/pnfs/desy.de':
     'grid-mon'            : '(ro,no_dcap)'
```

> After a puppet run the result is

```
##########################################################################
##
## DO NOT EDIT
## generated by puppet module :  dcache_operations::config::nfs_exports
## node                       :  dcache-core-cms03.desy.de
##
##########################################################################
/pnfs/desy.de/cms naf-school*.desy.de(ro,no_dcap)
/pnfs/desy.de grid-mon(ro,no_dcap)
```

What happens inside dcache_operations::config::nfs_exports

Content of dcache_operations

Hostgroup files

```
hostgroups/hg_dot/manifests/
├── base.pp
├── billing.pp
├── core.pp
├── dir.pp
├── door.pp
├── ha_head.pp
├── head.pp
├── mon.pp
├── pool.pp
└── se.pp
```

> Manage everything not related to dcache

> Manage zookeeper

> Setup dcache_operations module

Module files

```
modules/dcache_operations/manifests
├── array
│   ├── config.pp
│   ├── install.pp
│   └── service.pp
├── array.pp
├── auth
│   ├── admin_acl.pp
│   ├── authorized_keys.pp
│   ├── gplazma.pp
│   ├── grid_vrolemap.pp
│   ├── gss.pp
│   ├── multi_map.pp
│   └── storage_authzdb.pp
├── basic_install.pp
├── config
│   ├── dcache_conf.pp
│   ├── dcache_layout.pp
│   ├── global.pp
│   ├── nfs_exports.pp
│   ├── pool_layout.pp
│   ├── postgres.pp
│   └── resilience_monitoring.pp
├── door_install.pp
├── head_install.pp
├── init.pp
├── pool_install.pp
└── service.pp
```

> Setup for 60 disk storage boxes

> Administrator login

> External access

> dcache configuration and layouts

> Database configuration

> Initialise and wrapper classes

Configuration puppet classes

All puppet classes have a similar simple design

```
class dcache_operations::config::dcache_conf ($dcache_conf){

  $dcache_instance = hiera('dcache::dcache_instance')

  $dcache_zookeepers_1 = hiera("dcache::${dcache_instance}::zookeeper::server_name_1","localhost")
  $dcache_zookeepers_2 = hiera("dcache::${dcache_instance}::zookeeper::server_name_2","localhost")
  $dcache_zookeepers_3 = hiera("dcache::${dcache_instance}::zookeeper::server_name_3","localhost")

  file { '/etc/dcache/dcache.conf' :
    content => template('dcache_operations/dcache.conf.erb'),
    backup  => ".backup",
  }


}
```

All magic happens in the template file

Look at dcache_operations/dcache.conf.erb

- Templates are ruby scripts (located in dcache_operations/templates)
- Use variables from the puppet module or do hiera lookups
- Turn this structure

```
dcache::dcache_conf:
   general:
      admin.paths.history : '/var/lib/dcache/admin/admin.history'
   dcache:
      chimera.db.user : dcache'
      dcache.user : 'dcache'
```

into a ruby hash which is iterable

```
##########################################################################
##
## DO NOT EDIT
## generated by puppet module :<%= @name %>
## node :<%= @fqdn %>
##
##########################################################################
dcache.zookeeper.connection = <%= @dcache_zookeepers_1 %>
<% @dcache_conf.each do |section_name,section_properties| -%>
# globals for: '<%= section_name %>'
<% section_properties.each do |property_name,property_value| -%>
<%= property_name %>=<%= property_value %>
<% end -%>
<% end -%>
```

Application of the template

```
################################################################################
##
## DO NOT EDIT
## generated by puppet module :dcache_operations::config::dcache_conf
## node :dcache-dot13.desy.de
##
################################################################################

dcache.zookeeper.connection = dcache-dot13.desy.de

# globals for: 'general'
admin.paths.history=/var/lib/dcache/admin/admin.history

# globals for: 'dcache'
chimera.db.user=dcache
dcache.user=dcache
```

Advantages

- Allows for great flexibility
- Allow for additional automation
- Usually simple scripts but allow for arbitrary large complexity
- Allow for human readable `hiera yaml` files

Example of automation – `dcache` pools

- In most installation there is a heterogeneous storage infrastructure
  - Different number of pools
  - Different pool sizes
  - Pools with tape backend and pools with rotating data
  - Additional requirement for resilience pools and `md{32,34}60` systems

What we need for full automation

- List of pools on the node → for layout file
- Size of each pool → define maximum size minus a gap
- Need to know its purpose, e.g. tape backend
- Setup resilience requirement

Target pool layout file

```
[dcache-photon33-01Domain]
[dcache-photon33-01Domain/pool]
pool.name=dcache-photon33-01
pool.path=/dcache/dcache-photon33-01
pool.wait-for-files=${pool.path}/data
pool.mover.nfs.port.min=24001
pool.mover.nfs.port.max=24001
pool.lfs = precious
pool.tags=hostname=dcache-photon33

[dcache-photon33-02Domain]
[dcache-photon33-02Domain/pool]
pool.name=dcache-photon33-02
pool.path=/dcache/dcache-photon33-02
pool.wait-for-files=${pool.path}/data
pool.mover.nfs.port.min=24002
pool.mover.nfs.port.max=24002
pool.tags=hostname=dcache-photon33

[dcache-photon33-03Domain]
[dcache-photon33-03Domain/pool]
pool.name=dcache-photon33-03
pool.path=/dcache/dcache-photon33-03
pool.wait-for-files=${pool.path}/data
pool.mover.nfs.port.min=24003
pool.mover.nfs.port.max=24003
pool.lfs = volatile
pool.mover.nfs.thread-policy = WORKER_THREAD
pool.tags=hostname=dcache-photon33
```

Set pool name

Set `pool.lfs = precious` for disk only pools

Set nfs ports according to pool number

Set pool tag for resilience

Set `pool.lfs = volatile` for rotating pools

Pool `yaml` Files

```
dcache::pool_list:
  dcache-photon33:
    partner : 'NONE'
    model:
    hsmpools: 'dcache-photon33-02,dcache-photon33-05'
    rotating: 'dcache-photon33-03'
```

- `partner`: Takes into account the setup of the `md{32,34}60` devices
- `model`: `md{32,34}60` or `NetApp` devices, information for raid setup
- `hsmpools`: Pools with tape backend
- `rotating`: Pools from which the oldest files are removed once full

New pools are simply added to the hash in the same manner

```
dcache::pool_list:
  dcache-photon33:
    partner : 'NONE'
    model:
    hsmpools: 'dcache-photon33-02,dcache-photon33-05'
    rotating: 'dcache-photon33-03'
  dcache-photon34:
    partner : 'NONE'
    model:
    hsmpools: 'dcache-photon34-02,dcache-photon34-05'
    rotating: 'dcache-photon34-03'
```

Partial Pool Template Script

- Use variable `pool_layout_hash` from corresponding manifest
- Mount point generated by a custom fact
- Make use of conditions, loops and simple arithmetic

```
<% @pool_layout_hash.sort.each do |pool_name,pool_info| -%>
[dcache-<%= pool_name %>Domain]
[dcache-<%= pool_name %>Domain/pool]
pool.name=dcache-<%= pool_name %>
pool.path=<%= pool_info['MOUNTPOINT'] %>
pool.wait-for-files=${pool.path}/data
<% if !@nfs_port.nil? -%>
pool.mover.nfs.port.min=<%= @nfs_port %>
pool.mover.nfs.port.max=<%= @nfs_port %>
<% @nfs_port = @nfs_port.to_i -%>
<% @nfs_port += 1 -%>
<% end -%>
<% if @hsm_pools.nil? and @rot_pools.nil? -%>
pool.lfs = precious
<% end -%>
<% if !@hsm_pools.nil? and @rot_pools.nil? -%>
<% if !@hsm_pools.include?(pool_name) -%>
pool.lfs = precious
<% end -%>
<% end -%>
```

Partial Pool puppet manifest

Need additional variables including a custom fact

```
class dcache_operations::config::pool_layout ($dcache_layout){
  $layout_hash        = $dcache_layout
  $pool_layout_hash   = $::pools
  $pool_config        = hiera_hash('dcache::pool_list',undef)
  $nfs_port           = hiera('dcache::basic_nfs_port',undef)
  if has_key($pool_config, $hostname) {
    $partner          = $pool_config[$hostname]['partner']
    $hsm_pools        = $pool_config[$hostname]['hsmpools']
    $rot_pools        = $pool_config[$hostname]['rotating']
  }
  if $pool_layout_hash{
     file { 'pool-layout' :
        path    => "/etc/dcache/layouts/$hostname.conf.puppet",
        content => template('dcache_operations/pool.layout.conf.erb'),
        backup  => ".backup",
        } ~>
        exec {'production-layout':
          command => "/bin/cp /etc/dcache/layouts/$hostname.conf.puppet ..."
          creates => "/etc/dcache/layouts/$hostname.conf"
        }
    }
  if $pool_layout_hash {
    file { 'pool-setup-all' :
      path    => "/etc/dcache/all.pools.setup.xml",
      content => template('dcache_operations/pool.setup.conf.erb'),
    }
    file { '/root/distribute_setups.py' :
      source => 'puppet:///modules/dcache_operations/root/distribute_setups.py',
    }
  }
}
```

The ::pools Custom Fact

Series of small ruby scripts parsing the output of `lsblk` into a ruby hash

```
pools => {
  photon33-03 => {
    MOUNTPOINT => "/dcache/dcache-photon33-03",
    SIZE => "9000000000000"
  },
  photon33-04 => {
    MOUNTPOINT => "/dcache/dcache-photon33-04",
    SIZE => "4500000000000"
  },
  photon33-02 => {
    MOUNTPOINT => "/dcache/dcache-photon33-02",
    SIZE => "5000000000000"
  },
  photon33-01 => {
    MOUNTPOINT => "/dcache/dcache-photon33-01",
    SIZE => "4499999983104"
  },
  photon33-05 => {
    MOUNTPOINT => "/dcache/dcache-photon33-05",
    SIZE => "7000346545152"
  }
}
```

- Mount point need for the layout
- Size for the setup file → calculate the actual pools size
- Script for setup distribution obsolete with puppet 4 since loops work

Checking Dependencies

Cells in dCache have some internal dependencies

> Each poolmanager needs access informal for the administrator door

```
if ( $dcache_layout ){
  if has_key($dcache_layout, 'domains'){
    $domains = $dcache_layout['domains']
    $domains.each |$domain_name, $cells| {
      $cells.each |$cell_name, $options| {
        if $name == "poolmanager"{
          notify { 'Configure ssh keys and acl' : }
          class {'::dcache_operations::auth::authorized_keys':
            authorized_keys => $authorized_keys
          }
          class {'::dcache_operations::auth::admin_acl':
            authorized_keys => $authorized_keys
          }
        }
      }
    }
  }
}
```

Human readable hiera code does not translate to pretty puppet code

Complete Picture

- Have a system allowing to administer out `dCache` installation mostly by writing `hiera` variables
- Minimise actual logins to the head nodes (frequent due to safety)
- Minimise load for installing the dumb parts, i.e. doors and pools
- Minimise pit falls due to missing dependencies for certain services
- Shown elements of the whole system but in general most work done with templates

Where to find it

- `puppet/modules/dcache_operations/`
- `puppet/hostgroups/hg_dot/`
- `puppet/hieradata/dot*.yaml`

Suggestions and questions are always welcome