

Installing dCache with Puppet

Alexandra Yurova, DESY Summer Student 2012

September 5, 2012

Abstract

This document contains the description of dCache installation with the use of Puppet tool. Also you can find here some brief information about the main concepts of dCache and Puppet.

Contents

1	Introduction	3
2	What is dCache?	4
2.1	Cells	4
2.2	Domains	4
3	What is puppet?	5
3.1	Agent and master	5
4	Puppet manifest for installing dCache	6
4.1	Ordering	7
4.1.1	Before and Require	7
4.1.2	Notify and subscribe	7
4.2	Configuring Chimera	8
4.3	Creating users and databases for dCache	8
4.4	Creating and configuring pools	8

1 Introduction

The high energy physics experiments produce a sustained stream of data. This data is, while produced, distributed and persistently stored at several dozens of sites around the world. The destination sites are expected to provide the necessary middle-ware, so called Storage Elements, offering standard protocols to receive the data and to store it at the site specific Storage Systems. A major player in the set of Storage Elements is the dCache/SRM system. The system has shown to significantly improve the efficiency of connected tape storage systems, by caching and scheduled staging techniques. Furthermore, it optimizes the throughput to and from data clients as well as smoothing the load of the connected disk storage nodes by dynamically replicating datasets on the detection of load hot spots. The system is tolerant against failures of its data servers which enables administrators to go for commodity disk storage components. Access to the data is provided by various standard protocols. Furthermore the software is coming with an implementation of the Storage Resource Manager protocol (SRM), which is evolving to an open standard for grid middleware to communicate with site specific storage fabrics. For the purpose of working with a huge amount of data one needs to install and configure dCache. But sometimes, the number of machines, where dCache has to be installed is too big. Therefore one can use a technology called Puppet. It's a kind of language, that allows to create configurations for installing and configuring dCache. The main goal of this project was to create such a configuration.

2 What is dCache?

dCache is a system for storing and retrieving huge amounts of data, distributed among a large number of heterogenous server nodes. dCache supports a large set of standard access protocols to the data repository and its namespace. It also provides a coherent service using multiple computers or nodes.

Depending on the Persistency Model, dCache provides methods for exchanging data with backend (tertiary) Storage Systems as well as space management, pool attraction, dataset replication, hot spot determination and recovery from disk or node failures. Connected to a tertiary storage system, the cache simulates unlimited direct access storage space.

2.1 Cells

A cell is dCaches most fundamental executable building block. Even a small dCache deployment will have many cells running. Each cell has a specific task to perform and most will interact with other cells to achieve it.

Cells can be grouped into common types; for example, pools, doors. Cells of the same type behave in a similar fashion and have higher-level behaviour (such as storing files, making files available). Later chapters will describe these different cell types and how they interact in more detail.

There are only a few cells where (at most) only a single instance is required. The majority of cells within a dCache instance can have multiple instances and dCache is designed to allow load-balancing over these cells.

2.2 Domains

A domain is a container for running cells. Each domain runs in its own Java Virtual Machine (JVM) instance, which it cannot share with any other domain. A nodes resources, such as memory, available CPU and network bandwidth, are shared among several domains running on the same node.

dCache comes with a set of domain definitions, each specifying a useful set of cells to run within that domain to achieve a certain goal. These goals include storing data, providing a front-end to the storage, recording file names, and so on. The list of cells to run within these domains are recommended deployments: the vast majority of dCache deployments do not need to alter these lists.

Most cells communicate in such a way that they dont rely on in which domain they are running. This allows a site to move cells from one domain to another or to create new domain definitions with some subset of available cells. Although this is possible, it is rare that redefining domains or defining new domains is necessary. Starting or stopping domains is usually sufficient for managing load.

Domains and services are defined in the layout files. A domain must be defined if services are to run in that domain. Services will be started in the order in which they are defined. Every domain is a Java Virtual Machine that can be started and stopped separately. The layout files define which domains to start and which services to put in which domain.

For example, in our layout file the following domains are defined:

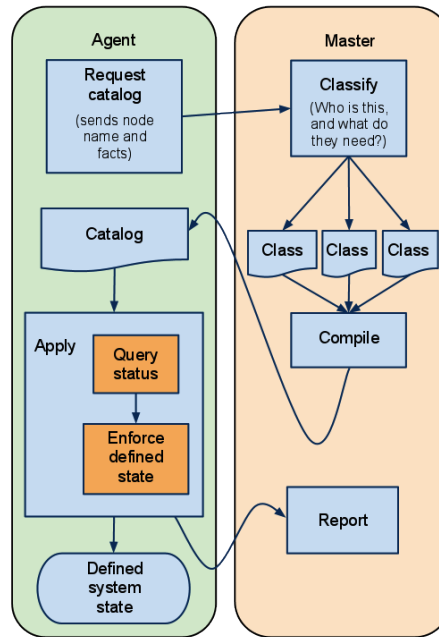
```
[dCacheDomain]
[dCacheDomain/admin]
[dCacheDomain/broadcast]
[dCacheDomain/poolmanager]
[dCacheDomain/loginbroker]
[dCacheDomain/spacemanager]
[dCacheDomain/pnfsmanager]
[dCacheDomain/nfsv3]
[dCacheDomain/cleaner]
[dCacheDomain/acl]
[dCacheDomain/dir]
[dCacheDomain/gplazma]
[dCacheDomain/gsi-pam]
[dCacheDomain/pinmanager]
[dCacheDomain/billing]
[dCacheDomain/srm-loginbroker]
[dCacheDomain/httpd]
[dCacheDomain/topo]
[dCacheDomain/info]
[poolDomain]
[poolDomain/pool]
name=pool1
path=/srv/dcache/p1
```

3 What is puppet?

3.1 Agent and master

Running Puppet in agent/master mode works much the following way: it moves the manifests and compilation to the puppet master server. Agents don't have to have see any manifest files at all, and have no access to configuration information that isn't in their own catalog.

This scheme shows agent, requesting the catalog. which gets compiled and served by a puppet master and applied by the agent.



4 Puppet manifest for installing dCache

We have one puppet agent, that can reach puppet master over the network. We are going to write down the instructions for installing of the dCache into Puppet language. Puppet programs are called manifests and have the file extension .pp. The code of our manifest is included onto class dCache. Classes are singleton collections of resources that Puppet can apply as a unit. They are just something as blocks of code that can be turned on or off. (Thats not the kind of class that is used in object-oriented programming!)

We can concern a systems configuration as a collection of molecules and call them resources. These pieces vary in size, complexity, and lifespan: a user account can be a resource, as can a specific file, a software package, a running service, or a scheduled cron job. Even a single invocation of a shell command can be a resource. In Puppet, every resource is an instance of a resource type and is identified by a title; it has a number of attributes (which are defined by its type), and each attribute has a value.

Here is an example of a representation of resources in puppet, that were used in our configuration.

In this way we describe the file, which is copied from master to agent and will be used further for the installation of the dCache.

```

file { 'single.conf':
  source => puppet:///modules/dcache/single.conf,
  path => '/etc/dcache/layouts/single.conf',
  owner=> 'root',
  group => 'root',

```

```
mode => '644',  
ensure => file,  
}
```

4.1 Ordering

We were writing all the declarations one after another, Puppet might sync them in any order: unlike with a procedural language, the physical order of resources in a manifest doesn't imply a logical order. But some resources depend on other resources. We are going to tell Puppet, which of the resources goes first.

Each resource type has its own set of attributes, but there's another set of attributes, called metaparameters, which can be used on any resource. (They're meta because they don't describe any feature of the resource that you could observe on the system after Puppet finishes; they only describe how Puppet should act.)

4.1.1 Before and Require

`before` and `require` make simple dependency relationships, where one resource must be synced before another. `before` is used in the earlier resource, and lists resources that depend on it; `require` is used in the later resource and lists the resources that it depends on.

These two metaparameters are just different ways of writing the same relationship our example above could just as easily be written like this:

```
file {'/tmp/test1': ensure => present, content => "Hi.", before => Notify['/tmp/test1  
has already been synced.'], }  
notify '/tmp/test1 has already been synced.':
```

4.1.2 Notify and subscribe

A few resource types can be refreshed—that is, told to react to changes in their environment. For a service, this usually means restarting when a config file has been changed; for an `exec` resource, this could mean running its payload if any user accounts have been changed.

The `notify` and `subscribe` metaparameters make dependency relationships the way `before` and `require` do, but they also make refresh relationships. Not only will the earlier resource in the pair get synced first, but if Puppet makes any changes to that resource, it will send a refresh event to the later resource, which will react accordingly.

Using different kinds of dependences we provide the resources are going in required order.

4.2 Configuring Chimera

Chimera is a library providing a hierarchical name space with associated meta data. Where pools in dCache store the content of files, Chimera stores the names and meta data of those files. Chimera itself stores the data in a relational database. We are using PostgreSQL.

4.3 Creating users and databases for dCache

The dCache components will access the database server with the user `srmdcache` which can be created with the `createuser`. Such procedures as creating of the database or user were created in the special modules, which get the name of the database, the owner and the user name as a parameter and can be also used in other cases.

4.4 Creating and configuring pools

dCache will need to write the files it keeps in pools. These pools are defined as services within dCache. Hence, they are added to the layout file of your dCache instance, like all other services. The best way to create a pool, is to use the `dcache` script and restart the domain the pool runs in. The pool will be added to our layout file.

References

- [1] *<http://www.dcache.org/manuals/Book-2.2/>*
<http://docs.puppetlabs.com/>