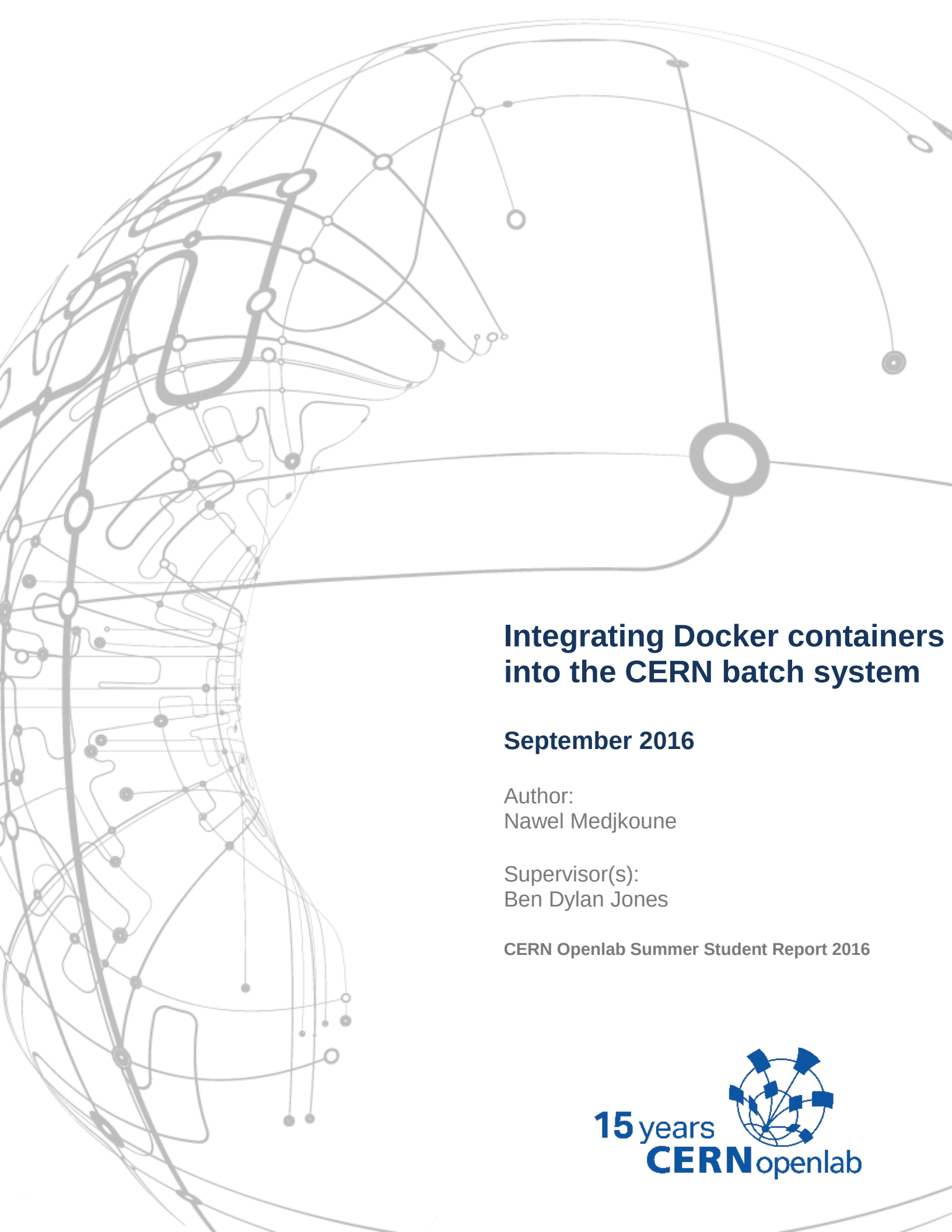

Integrating Docker containers into the CERN batch system

September 2016

Author:
Nawel Medjkoune

Supervisor(s):
Ben Dylan Jones

CERN Openlab Summer Student Report 2016

15 years CERN openlab

Project Specification

The CERN IT Batch service provides computing resources for users at CERN and for the Worldwide LHC Computing Grid. Comprising around 80,000 CPU cores, a large percentage - currently 15,000 - are dedicated to our future batch service powered by HTCondor (https://research.cs.wisc.edu/htcondor/). The new system brings some new features, including the ability to use Docker (https://www.Docker.com) to run jobs in the batch service. Linux container technology such as Docker has become very popular with developers due to ease of packaging and deploying applications. HTCondor has a dedicated “universe” for Docker, and so providing this as part of the batch services should enhance the user experience, and indeed the ease of administering the batch service. The project is to provide and test Docker support in the HTCondor service, and to pilot use cases with advanced user communities such as the ATLAS experiment.

Abstract

Today, the CERN batch system is a facility with more than 10,000 compute nodes. The CERN IT batch service manages this facility and provides computing resources to CERN users and for the Worldwide LHC Computing Grid. The computing power the batch system manages is around 100,000 CPU cores.

In the past, the CERN batch system was running the IBM platform LSF. Currently part of the batch farm has migrated to HTCondor, the future workload management software for the CERN batch system.

Latest versions of HTCondor come with new features, including the ability to run jobs inside Docker containers. We would like to test the Docker support in the batch service to address two issues:

- The inability of the batch submitters to define their own environment of execution, without the intervention of a system administrator
- The underlying OS on the worker nodes are dependent of the job environment. Making updates on those OS may affect the software compatibilities.

In this report, we outline and document the deployment of Docker on the HTCondor worker nodes running CERN CentOS 7, the setting up of the Docker Universe and the creation of job routes that transform incoming jobs in the grid to Docker jobs to be executed in containers. The project includes also the subtask of creating a Scientific Linux CERN 6 (SLC6) Docker image for the grid jobs.

Table of Contents

1 Introduction

1.1 Context

The CERN batch service currently consists of around 100, 000 CPU cores providing a large public computing facility for CERN experiments. This service is used for both CERN local submissions and WLCG grid submissions. The local public queue is open to all CERN users while the grid queues are dedicated to the LHC experiments.

The purpose of the batch system is to process CPU intensive work (which can be for example physics event reconstruction, data analysis and physics simulations), while ensuring that the resources are fairly shared between users.

For a long time, the CERN batch system was running the IBM Platform LSF product for workload management. It is currently migrating to HTCondor.

When a job is submitted in batch, the workload management system (LSF or HTCondor) decides when and in which worker node this job is executed. The job and its input are transferred to the chosen worker node, processed and the output is transferred back to the user.

1.2 Problem

One of the problems that the batch users are facing is that the job environment (dependencies, libraries and configuration files) is not controlled by the job submitter, but rather by what is installed on the worker node and therefore by the system administrator. Besides, running updates on the worker nodes operating systems affects the job environment, which makes it difficult to the sysadmin to run these updates.

While running the jobs on the batch worker nodes, we would like to have an isolated filesystem which allows the submitter to define the job environment and the sysadmin to operate modification on the worker node without affecting the jobs. We would have considered to execute jobs in virtual machines but virtual machines are difficult to maintain, consume a large amount of resources and are very slow.

1.3 Solution

One solution is to use run jobs inside Linux containers using Docker. Docker is an open-source project that automates the deployment of applications inside Linux containers. Docker containers are easy to create and maintain, lightweight and fast. Docker wraps the software in an isolated filesystem that contains everything it needs to run: code, runtime, system tools, and system libraries [1].

The aim of this project is to deploy Docker on the batch system so the jobs will be run inside containers instead of bare metal on the worker node. The submitter has the ability to define its own environment inside the container.

2 Batch and Grid

2.1 Batch system

Batch processing is the execution of a series of programs (called jobs) without manual intervention. A batch system is a system where users do not interact directly with computer. Instead, users submit the job to the system operator. Jobs with similar needs are batched together by the operator and run as a group [2].

The purpose of a batch system is to process CPU intensive work while ensuring that the resources are shared fairly between different users of the system, according to a defined policy. It also ensures keeping a high overall rate of utilization of resources to maximize the efficiency of the organisation [3].

2.2 Grid Computing and WLCG:

A grid is a collection of a large number computer resources from multiple locations connected together in a network to reach a common goal. In the grid computing model, servers or personal computers run independent tasks and are loosely linked by the Internet or low-speed networks. Computers may connect directly or via scheduling systems [4].

In CERN, data flows from the LHC detectors at a very high rate. After filtering, there is 600 MB/s of data that goes out of the detectors, and this data has to be dealt with. It needs to be stored, easily retrieved and analysed. All these tasks require a massive storage capacity and computing power. CERN doesn't have these computing resources on site, so starting from 2012 it turned to grid computing. The Worldwide LHC Computing Grid (WLCG) was created. It is a distributed computing facility arranged in tiers from different locations around the world. The aim of WLCG is to provide storage resources, retrieve and process the data generated by LHC [5] . The LCH uses embarrassingly parallel computing, which is a pattern of computing where no effort is needed

The LHC computing workload is embarrassingly parallel, where little effort is needed to separate the problem into (almost)independent parallel tasks. Those tasks can be submitted to the CERN batch system. Around two thirds of CERN computing power is assigned to the grid, shared between the LHC experiments according to the WLCG/CERN agreements [3].

2.3 CERN batch service

The CERN batch resources are accessible by two ways: local batch system where CERN users with local accounts can submit jobs, and the grid share is provided to WLCG where all users of the grid (local from CERN or remote) can access it to submit jobs in their experiment framework. The jobs are submitted in several queues depending on the experiment for grid, and the maximum CPU time for local jobs. All jobs, regardless of the queue they are submitted to, are considered together on the public share. A fairshare algorithm is executed to determine which job goes next on the batch service.

For a long time, the CERN batch service was running IBM Platform LSF for workload management. Some major scalability concerns raised concerning the number of nodes it can support. Only 6500 nodes are guaranteed by the vendor. Alternative batch systems were investigated to address this issue, and HTCondor was finally chosen. Currently, around 15000 CPU cores were migrated to HTCondor, waiting for a complete migration. Besides the scalability solution, HTCondor offers some new features. One of the interesting features for our project is the ability to use Docker with HTCondor to run jobs in the batch service.

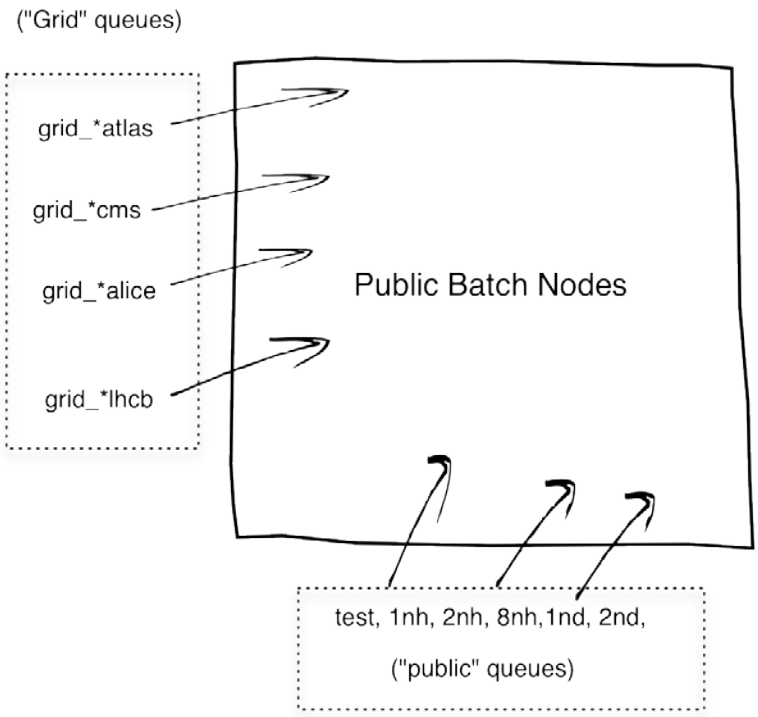

Figure 1- CERN Public batch nodes and accessible queues

3 HTCondor

3.1 Overview

HTCondor is an open-source high throughput distributed batch computing system. HTCondor provides a workload management mechanism, scheduling policy, priority scheme and resource management. Users submit jobs to HTCondor and this latter decides when and where to run them based on the policy defined, monitors their progress and inform the submitter when the execution is complete [6] .

HTCondor allows the user to submit different types of jobs called “universes”. A universe defines an execution environment. The latest version of HTCondor supports 8 different universes. The universe is specified in the job submit description file. Here’s a non-exhaustive list: The **standard** and **Vanilla** universes are for normal jobs. The first provides migration but has more restrictions on the type of jobs, the second does not allow migration but has very few restrictions. The **grid** universe allows to submit jobs to grid resources using the HTCondor interface. The java universe runs jobs written for Java Virtual Machine. The parallel universe is for jobs that require to be run on multiple machines. The Docker universe runs Docker container as an HTCondor Job [7].

HTCondor uses several daemons to manage a pool of machine. A non-exhaustive list is given in following:

- Condor_master: main daemon responsible for keeping the rest of daemons running on each machine of the pool.
- Condor_startd: this daemon is used to advertise attributes about resources available in the machine
- Condor_Starter: responsible for spawning the executed job on a given machine.
- Condor_schedd: Represents resource requests to the HTCondor pool. It is used for selection of resources necessary to run a job
- Condor_shadow: created on the machine where the job was submitted from, it’s responsible for monitoring that machine and manage resources for the job request.
- Condor_collector: is used for collecting information about the status of machines in the HTCondor pool
- Condor_negotioator: responsible for making the match between the job request and the machine capable of running that job
- Condor_job_router: responsible of transforming a job after submission

3.2 Architecture

Depending on the daemons that are running on it, each machine in the HTCondor pool can have one or more than one type:

Submit node: jobs are executed from this node to the HTCondor system

Execute node: a computing node responsible for running jobs that are affected to it

Central Manager: there can be only one central manager per pool. A CM acts as a server: collects information from all machines in pool, and negotiates between the resources available on the machines and the resources requests.

In CERN batch system terms, submit and execute nodes are called **scheduler** and **worker** node respectively.

The next figure illustrates this architecture.

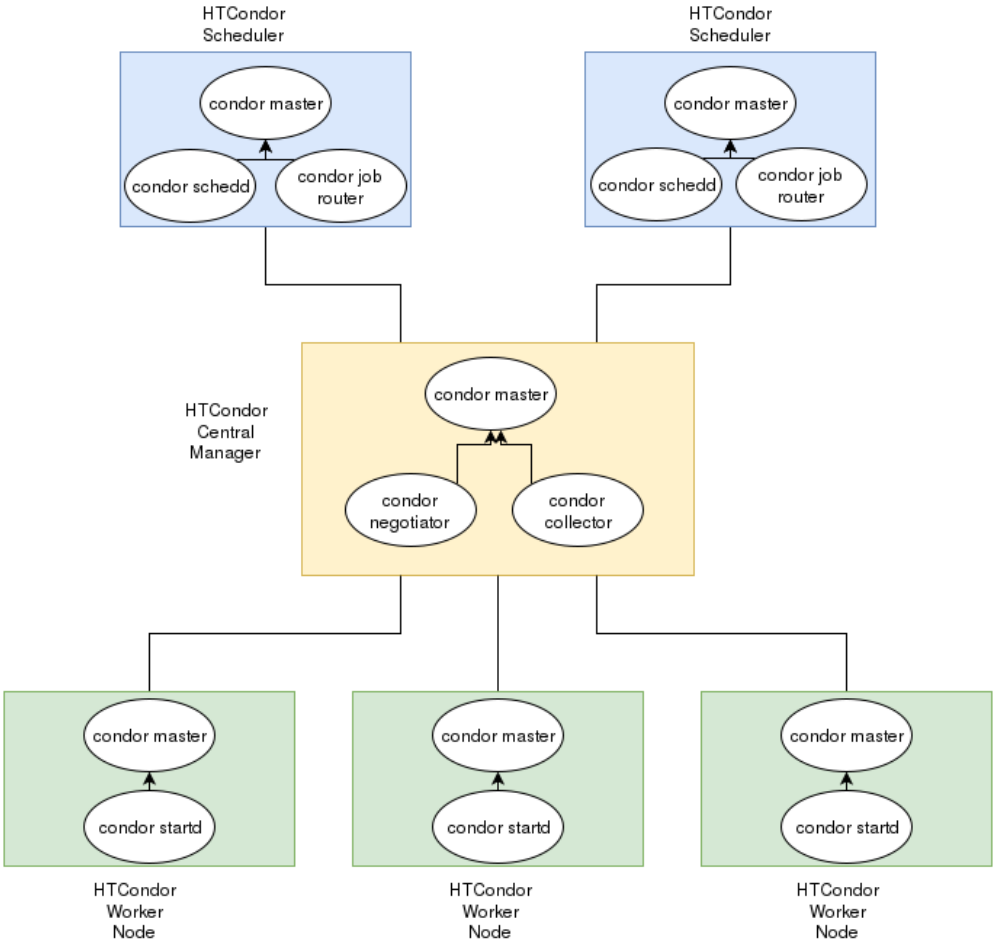

Figure 2Architecture of HTCondor

3.3 HTCondor-CE

HTCondor CE stands for HTCondor Compute Element. A Compute Element is a term used in grids to denote any kind of computing interface (i.e. a job entry for the batch system). A compute element consists of one or more similar machines, managed by a single scheduler/job queue, which is setup to accept and run grid jobs. At the heart of the CE is the *job gateway* software, which is responsible for handling incoming jobs and authorizing them.

HTCondor-CE is a special configuration of the HTCondor software designed to be a job gateway for grids. Once an incoming Grid job is authorized, it is placed in HTCondor-CE scheduler where the job router daemon transforms the job and submits it to the local batch system [8].

4 Docker

4.1 Overview

Docker is an open-source platform that automates the deployment of packaged software inside Linux containers. The software is wrapped in a complete and isolated filesystem that contains all the dependencies and libraries it needs to run. This guarantees the portability of the software on different environments and the isolation provided by a virtualized environment.

Containers running on a single machine share the same operating system kernel with the host, this means only the filesystem at the user level is run in the container. This makes containers very lightweight and start instantly [9]. The next figure illustrate the difference between a virtual machine and a lightweight Docker container.

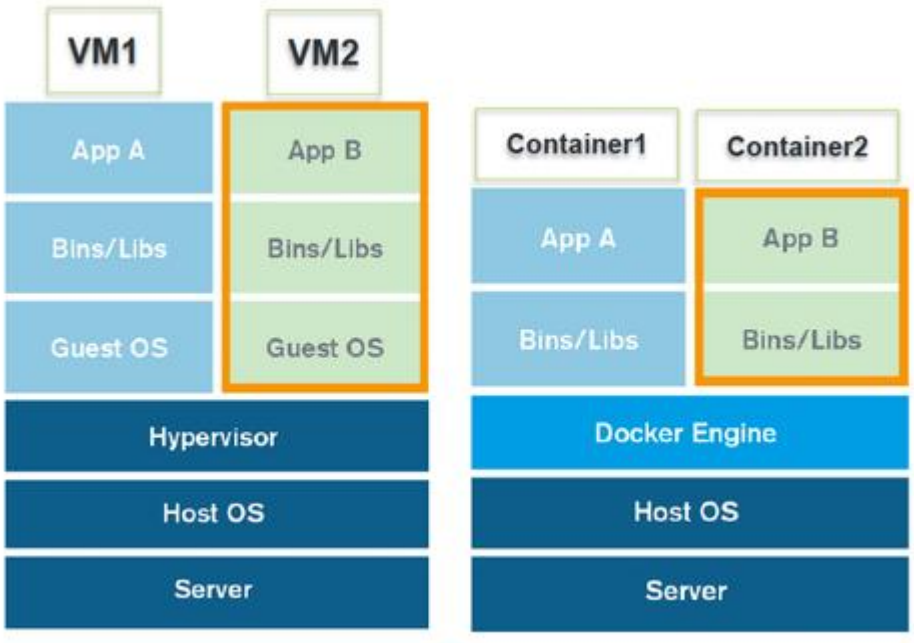

Figure 3Comparison between a virtual machine and a Docker container

Ref: regmedia.co.uk

Containers are launched from Docker images. A Docker image is a read-only file that describes what the container should contain at runtime. Each image is constructed of layers of filesystems. Docker uses the Union File System to combine these layers and form a single coherent filesystem. A container is a running instance of an image. It consists of an operating system (user level filesystem only), user added software and files and meta-data. The image tells Docker what process to run inside the container when it is launched.

Docker containers can be run, started, stopped, moved, and deleted.

4.2 The underlying technology

Docker uses several Linux kernel features to deliver its functionally:

- **Namespaces** are used to provide the isolated workspace named container. When a container is launched, Docker creates a set of namespaces (**pid** for process, **net** for managing network interfaces, **uts** for isolating kernel and version identifiers ….). This makes the container isolated and the processes inside the container do not have access outside of it.
- **Control groups** is used to control the resources that a container can have access to. It is used by Docker to share the resources, set up limits and constraints on them.
- **Union file systems**, or UnionFS, are file systems that operate by creating layers, making them very lightweight and fast. Docker Engine uses union file systems to provide the building blocks for containers.

4.3 Docker registry

A Docker registry holds Docker images. It can be the public Docker Hub (https://hub.Docker.com/) or a private registry. The public Docker registry serves a massive collection of existing images for sue. When a Docker image is created it can be pushed to a Docker registry for reusability. Anyone who has access to that registry can pull the image on its own Docker client and run it [9].

4.4 Data Volumes

A data volume is a specific directory in one or more containers that bypass the Union File System. Data volumes are used for shared or persistent data. They provide the following features:

- Volumes are initialized when a container is created. Data contained in container's base image at the specific mount point are copied.
- Data volumes can be shared and reused among containers and with host filesystem.
- Data volumes are persistent after the container is deleted.

4.5 Docker Universe in HTCondor

Docker universe is one of the several universes existing on HTCondor. It instantiates a Docker image and HTCondor manages the running of the container as an HTCondor Job: it can be scheduled, removed, put on hold ...etc.

In order to execute a Docker job, an execute node must advertises its capability to run Docker universe jobs, and this by having Docker installed and configured for HTCondor.

The image from which the container is instantiated is defined by specifying a Docker image with the submit command **docker_image**. This image must be pre-staged on a Docker hub that the execute machine can access [7].

5 Puppet

CERN uses puppet to configure and manage most of the batch system nodes. Puppet is an open-source configuration management tool produced by Puppet Labs. It is designed to manage the configuration of Unix-like and Microsoft Windows systems using a customizable declarative language. The user describes the resources on the machine and their state. For example: the packages that should be installed, the services that must be running and the configuration files that must be created. This description is done either by using puppet's declarative language or a Ruby DSL.

The puppet code that describes the resources on the node are stored in files called manifests.

The configuration values used by puppet (variables) are stored in Hiera yaml files, a key/value store with a hierarchical search path. These files (manifests and data files) are stored and versioned using git versioning system.

In the next sections, each time a configuration is described, we will give both the bash scripts and their puppet code equivalent [10].

6 Setting up the Docker universe

Giving that HTCondor already has the Docker universe feature, the task of deploying containers into the CERN batch system is completed by setting up correctly the Docker universe.

To reach that goal, the project was done in the following steps:

6.1 Deploying CERN CentOS 7 node

A large majority of nodes in the CERN batch system are running Scientific Linux 6. There is a current wish to upgrade the OS in the worker nodes to CERN CentOS 7 because it offers new features that are being requested by CERN users. In the context of our project, one other reason that leads us to prefer deploying Docker on CentOS 7 is that Docker is no longer supported by RedHat in releases below 7

The first step of our project is to deploy a new worker node running CC7 in the HTCondor workers pool.

Creation of the worker node virtual machine: the configuration for this node is under the bi/condor/testworker hostgroup, the hostname of the machine is condorcker7.cern.ch

```
$ ai-bs-vm --foreman-hostgroup bi/condor/testworker  --foreman-
environment qa  --cc7  condorcker7.cern.ch
```

The configuration of the CC7 node includes installing the batch worker common utilities, the batch support packages and HTCondor.

When configuring HTCondor, the hostnames of the central manager, the schedulers and the CEs are specified. The worker node's hostname must also be added in the certificate mapfile.

6.2 Installing and configuring Docker

In order to set up the Docker universe in the CC7 worker node, the Docker package must first be installed as root

```
$ yum install docker-io
```

Next, the condor user must be added to the Docker unix group so it can run containers

```
$ useradd -G docker condor
```

The docker service is started and condor is reconfigured

```
$ service docker start && condor_reconfig
```

One the docker service is running, we can check on the CC7 worker node if it properly advertised the ClassAdd **HasDocker**

```
$ condor_status -l | grep -i docker
```

The output of this command is:

```
 HasDocker=true
 DockerVersion = "Docker Version 1.6.0, build xxxxx/1.6.0"
```

The puppet equivalent code is giving as following. A Docker puppet module exists and is used to install Docker on the nodes

```
class{'docker':
    docker_users=>['condor'],
    dns=>['137.138.16.5','137.138.17.5','8.8.8.8'],
  }
  ->

  Service['condor']
  ->
  Exec[ '/usr/sbin/condor_reconfig']
```

Next we mount the data volumes from the worker node so they can be used inside the containers. These directories are important for grid submissions: CVMFS, Grid_Security, SSSD and nsswitch

```
DOCKER_VOLUMES = CVMFS, ETC_CVMFS, GRID_SECURITY, SSSD, NSSWITCH
DOCKER_VOLUME_DIR_CVMFS=/cvmfs:/cvmfs:ro
DOCKER_VOLUME_DIR_ETC_CVMFS     = /etc/cvmfs:/etc/cvmfs:ro
DOCKER_VOLUME_DIR_GRID_SECURITY = /etc/grid-security:/etc/grid-
security:ro
DOCKER_VOLUME_DIR_SSSD          = /var/lib/sss/pipes/nss
DOCKER_VOLUME_DIR_NSSWITCH      =
/etc/nsswitch.conf:/etc/nsswitch.conf:ro
DOCKER_MOUNT_VOLUMES = CVMFS, ETC_CVMFS, GRID_SECURITY, SSSD,
NSSWITCH
```

These environment variables are contained in a condor config file created by the following puppet code:

```
 file { '/etc/condor/config.d/35_Docker.config':
    content => template('hg_bi/condor/35_Docker.erb'),
    owner => 'condor',
    group => 'condor',
    mode => 644,
  } -> Exec[ '/usr/sbin/condor_reconfig']
```

6.3 CERN Docker private registry

To allow CERN users to pull images from the CERN Docker private registry, Docker must login to this registry.

The Docker command to login to the registry is the following:

```
$ docker login -u login -u password docker.cern.ch
```

The puppet equivalent is done using the puppet type registry from Docker module:

```
 docker::registry { 'docker.cern.ch':
    username => 'login',
    password => 'password',
    local_user => 'condor',
    email    => 'none',
  }
```

6.4 Docker universe job: submit description file

When a local user wants to submit a Docker job, the submit description file must look like the following:

```
  universe                = docker
  Docker_image            = debian
  executable              = ./some_executable.sh
  arguments               = input1.data
  transfer_input_file    = YES
  should_transfer_files   = YES
  when_to_transfer_output = ON_EXIT
  output                  = out.$(Process)
  error                   = err.$(Process)
  log                     = log.$(Process)
  request_memory          = 100M
  queue 1
```

The user must specify the Docker image. The container does not share a file system with either the execute host or the submit host, except the mounted data volumes. Therefore the user must specify that the files should be transferred. If no executable is given, condor will run the default command in the Docker image.

7 Routing vanilla jobs to Docker jobs

CERN batch farm being part of the WLCG, grid jobs from external submitters are accepted from the HTCondor-CE into the CERN batch system. These jobs uses most of the time the vanilla universe.

We want to execute these vanilla jobs inside containers in the CERN worker nodes. To do this, we use the condor job router feature to transform vanilla jobs to Docker jobs. Condor Job Router is an add-on to the *condor_schedd* that transforms jobs from one type into another according to a configurable policy. The routed job is a copy of the original job, where some attributes are modified [7].

For any upcoming vanilla job, the job router component will do the transformation as following:

- The Docker image is set if not existing

```
copy_DockerImage = "orig_DockerImage"; \
eval_set_DockerImage=ifThenElse(isUndefined(orig_DockerImage),
Docker.cern.ch/linuxsupport/slc6-grid", orig_DockerImage); \
```

- The command for the executable must be prepended with ./ :

```
copy_Cmd = "orig_Cmd"; \
eval_set_Cmd = ifThenElse(regexp("^/", orig_Cmd), orig_Cmd,
strcat("./",orig_Cmd)); \
```

The final full route:

```
JOB_ROUTER_ENTRIES = \
   [ \
    eval_set_environment = debug(strcat("HOME=/tmp
CONDORCE_COLLECTOR_HOST=", CondorCECollectorHost, " ", \
        ifThenElse(orig_environment is undefined,
osg_environment, \
          strcat(osg_environment, " ", orig_environment) \
        ))); \
     MaxJobs = 12000; \
     MaxIdleJobs = 4000; \
     TargetUniverse = 5; \
     name = "Docker_Condor"; \
      [...]
 Requirements = (TARGET.WantExternalCloud =!= True) &&
(TARGET.queue =!= "WantExternalCloud") && (TARGET.queue =!=
"externalcloud") && (TARGET.WantDocker =?= True); \
     copy_Cmd = "orig_Cmd"; \
     eval_set_Cmd = ifThenElse(regexp("^/", orig_Cmd), orig_Cmd,
strcat("./",orig_Cmd)); \
     copy_DockerImage = "orig_DockerImage"; \
     eval_set_DockerImage =
ifThenElse(isUndefined(orig_DockerImage),
```

```
"Docker.cern.ch/linuxsupport/slc6-grid", orig_DockerImage); \
   ] \
```

When a job is submitted to the HTCondor-CE it is transformed to a Docker job and then routed to the local scheduler. The following figure shows this workflow

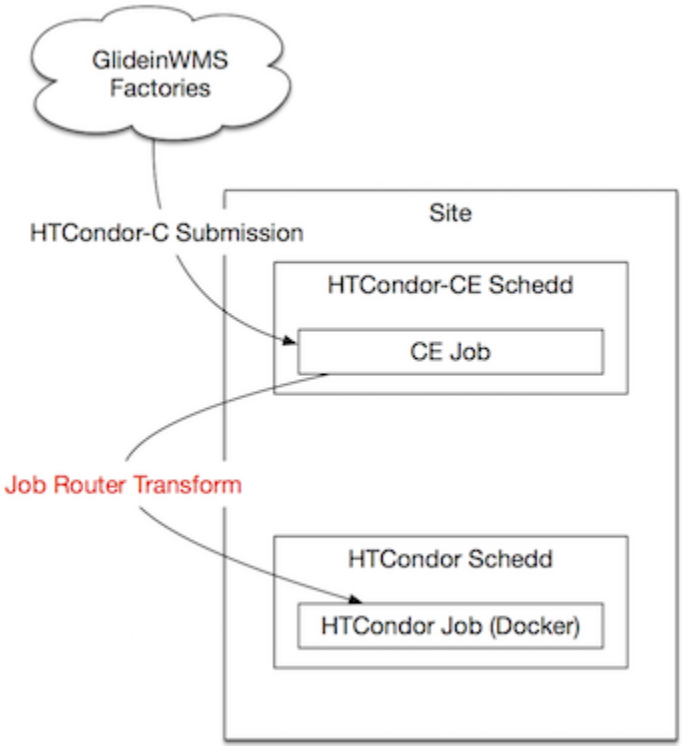

Figure 4 Workflow of a docker job routing in HTCondor CE , ref: djw8605.github.io

8 Building SLC6 images

In order to run the grid jobs in Docker containers, we need to provide a Docker image running SLC6 in which we make some customizations to allow the experiments jobs to be run.

The Docker image is build using Packer. Packer is an open source tool for creating identical machine images for multiple platforms from a single source configuration. One of the outputs of Packer is Docker images. Packer uses json templates files to describe the content and configuration of the image [11].

Building the image includes adding the EMI-3 and batch6 repository, installing all packages related to experiments, installing lcmaps and glexec, adding condor group and condor user and copying the glexec configuration files.

The packer template is given as following:

```
{
"builders":[{
"type": "Docker",
"image": "Docker.cern.ch/linuxsupport/slc6-base",
"export_path": "image.tar",
"pull": "false"
}],
"provisioners":[
{
"type": "file",
"Source": EMI-3-base.repo",
"destination": "/etc/yum.repos.d/EMI-3-base.repo"
},
{
"type": "file",
"source": "/usr/src/packer/batch6-stable.repo",
"destination": "/etc/yum.repos.d/batch6-stable.repo"
},
{
"type": "shell",
"inline": [
"yum -y update",
"yum clean all",
"yum -y install  [ all packages for experiments]”
"Yum -y install lcmaps-plugins-condor-update
lcmaps-plugins-mount-under-scratch
lcmaps-plugins-namespace",
"yum -y install glexec-wn",
"groupadd -r condor",
"useradd -r -g condor -d /var/lib/condor -s /sbin/nologin condor"
]
},
{
"type": "file",
```

```
"source": "/usr/src/packer/glexec.conf",
"destination": "/etc/glexec.conf"
},
{
"type": "file",
"source": "/usr/src/packer/lcmaps-glexec.db",
"destination": "/etc/lcmaps/lcmaps-glexec.db"
},
{
"type": "shell",
"inline": [
"chown root:glexec /etc/glexec.conf",
"chmod 640 /etc/glexec.conf",
"chown root:root /etc/lcmaps/lcmaps-glexec.db",
"chmod 640 /etc/lcmaps/lcmaps-glexec.db"
]
}
],
"post-processors": [
{
"type": "Docker-import",
"repository": "Docker.cern.ch/batch-service/slc6-grid",
"tag": "latest"
     }
     ]
}
```

9 Conclusion

The idea of testing the Docker support in the batch system is to give CERN users the ability to define their own job environments and to allow system administrators to run updates on the worker nodes without affecting the jobs' running environment.

During this work, we tested the deployment of Docker on the HTCondor worker nodes and the setting up of the Docker Universe. Job routes have been created to transform incoming grid jobs to Docker jobs, using an SLC6 Docker image we created with the basic environment needed by the LHC experiments jobs.

The tests of the implemented functionality were successful and will be pushed into production. The usage of the Docker feature will hopefully help the batch service to upgrade the worker nodes OS from SLC6 to CERN CentOS 7 easily.

10 Acknowledgment

I would like to thank my supervisor Ben and all IT-CM-IS section members especially Lorena, Iain, Fernando, Gavin, Carolina and Philippe for their help during this project and for making this experience very pleasant and rewarding.

11 Bibliography

[1] S. Hykes, “Docker (software),” 13 March 2013. [Online]. Available: https://en.wikipedia.org/wiki/Docker_%28software%29.

[2] T. Point, “Operating System, fundamental OS concepts,” 2016. [Online]. Available: http://www.tutorialspoint.com/operating_system/operating_system_tutorial.pdf.

[3] I. Department, “ Batch Service documentation,” CERN, [Online]. Available: http://information-technology.web.cern.ch/services/batch.

[4] J. O. Margaret Rouse, “Tech Target,” March 2015. [Online]. Available: http://searchdatacenter.techtarget.com/definition/grid-computing.

[5] “Worldwide LHC Computing Grid (WLCG),” [Online]. Available: http://wlcg-public.web.cern.ch/.

[6] Thain Douglas, Todd Tannenbaum, and Miron Livny, “Distributed computing in practice: the Condor experience.,” *Concurrency and computation: practice and experience*, pp. 323-356, 2005.

[7] HTCondor, “HTCondorTM Version 8.5.6 Manual,” Center for High Throughput Computing, University of Wisconsin-Madison, [Online]. Available: http://research.cs.wisc.edu/htcondor/manual/v8.5/index.html.

[8] O. S. Grid, “HTCondorCEOverview Documentation/Release3,” 2016. [Online]. Available: https://twiki.grid.iu.edu/bin/view/Documentation/Release3/HTCondorCEOverview.

[9] Docker, “Docker Documentation Page,” [Online]. Available: https://docs.Docker.com/engine/understanding-Docker/.

[10] PuppetLabs, “Puppet Language Documentation Page,” [Online]. Available: https://docs.puppet.com/.