

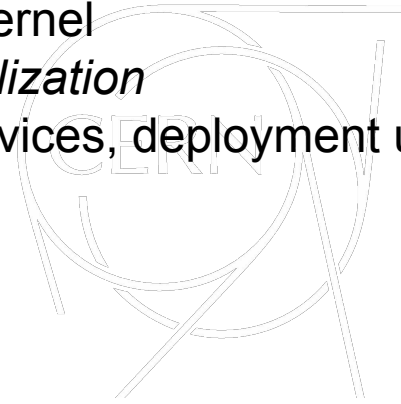


Integrating Containers in the CERN Private Cloud

Ricardo Rocha
(on behalf of the CERN Cloud team)

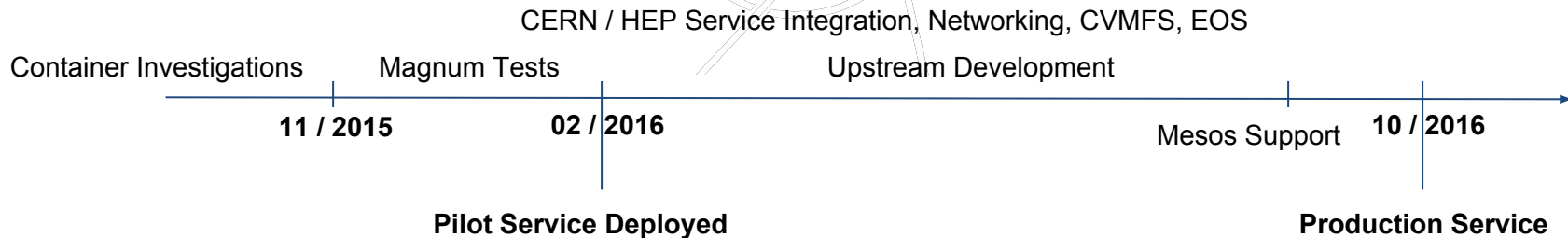
Why containers?

- *Isolation*, via kernel namespaces and cgroups
- *Performance*, same kernel
- *Improved resource utilization*
- *Ease of use*, microservices, deployment units, image repositories



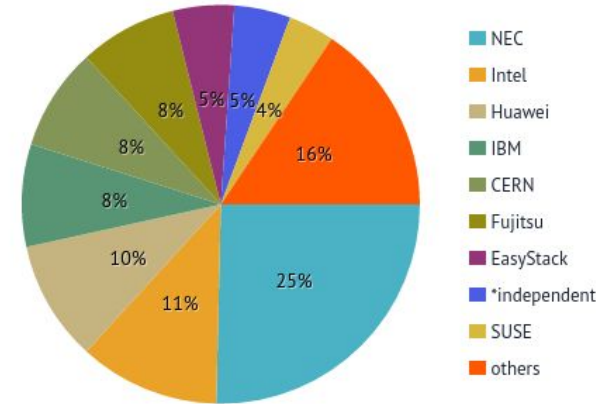
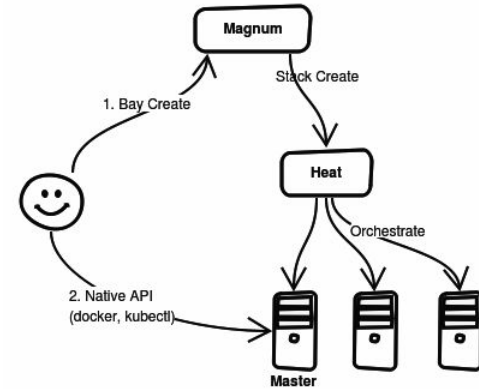
Goals and Timeline

- Integrate containers in the CERN cloud
 - Shared identity, networking integration, storage access, ...
- Agnostic to container orchestration engines
 - Docker Swarm, Kubernetes, Mesos
- Fast, easy to use



OpenStack Magnum

- OpenStack container project
- Orchestrate deployment of container clusters
- Key features
 - Swarm, Kubernetes, Mesos support
 - Client access using native clients / APIs
 - Cluster scaling
 - Lifecycle operations



Example Usage

- Clusters are described by *cluster templates*
- Shared/public templates for most common setups, customizable by users

```
$ magnum cluster-template-list
+-----+-----+
| uuid | name |
+-----+-----+
| .... | swarm |
| .... | swarm-ha |
| .... | kubernetes |
| .... | kubernetes-ha |
| .... | mesos |
| .... | mesos-ha |
+-----+-----+
```

```
$ magnum cluster-template-show swarm
...
| coe | swarm |
| master_flavor_id | m1.small |
| flavor_id | m1.small |
| server_type | vm |
| image_id | fedora-23-atomic |
| labels | {} |
| network_driver | docker |
```

Example Usage

- Create a cluster in a single command (no matter what size)

```
$ magnum cluster-create --name myswarmcluster --cluster-template swarm --node-count 100

$ magnum cluster-list
+-----+-----+-----+-----+-----+
| uuid | name           | node_count | master_count | status           |
+-----+-----+-----+-----+-----+
| .... | myswarmcluster | 100        | 1            | CREATE_COMPLETE |
+-----+-----+-----+-----+-----+

$ $(magnum cluster-config myswarmcluster --dir magnum/myswarmcluster)

$ docker info / ps / ...
$ docker run --volume-driver cvmfs -v atlas.cern.ch:/cvmfs/atlas -it centos /bin/bash
[root@32f4cf39128d /]#
```

Example Usage

- Scale your cluster (up)

```
$ magnum cluster-update myswarmcluster replace node_count=200
```

- Scale your cluster (and down...)

```
$ magnum cluster-update myswarmcluster replace node_count=5
```

- Support for built-in orchestration tools
 - Docker Compose, Kubernetes, Marathon/DCOS

Use Cases

- Example: Spark on Mesos

```
$ magnum cluster-create --name myspark --cluster-template mesos --node-count 20

$ magnum cluster-show myspark | grep api_address
| api_address | 137.138.7.77 |

$ spark-shell --master mesos://zk://137.138.7.77:2181/mesos
scala> val NUM_SAMPLES = 1000
      val count = sc.parallelize(1 to NUM_SAMPLES).map{i =>
        val x = Math.random()
        val y = Math.random()
        if (x*x + y*y < 1) 1 else 0
      }.reduce(_ + _)
      println("Pi is roughly " + 4.0 * count / NUM_SAMPLES)
Pi is roughly 3.142532
```

Use Cases

- Example: File Transfer Service

```
$ magnum cluster-create --name fts --cluster-template kubernetes --node-count 20  
$ $(magnum cluster-config fts --dir magnum/fts)  
$ kubectl create -f fts-server.yaml
```

```
$ magnum cluster-create --name fts --cluster-template swarm --node-count 20  
$ $(magnum cluster-config fts --dir magnum/fts)  
$ docker-compose fts-server.yaml
```

<https://indico.cern.ch/event/505613/contributions/227329/>

Deploying Container Based FTS on Openstack Magnum

Andrea Manz, Alejandro Alvarez Ayllón, María Arsuaga Ríos, Ricardo Brito Da Rocha

FTS3 is the service responsible for globally distributing the majority of the LHC data across the WLCG infrastructure. It is a low level data movement service, responsible for reliable bulk transfers of files from one site to another while allowing participating sites to control the network resource usage.

Current deployment of FTS

FTS3 has been designed to be easy to scale horizontally just adding more VMs to a server. Each instance is a clone running the exact same set of services, with state persisted via a shared MySQL database.

The service splits the load evenly across VMs, and when a node enters or exits, the queue is readjusted accordingly, so each instance runs a similar amount of transfers.

Dockerized FTS

A single FTS3 VM deployment has been split in 3 different containers (Server/Staging, REST, Web Mon) automatically built via GitLab CI and published to GitLab Registry available at CERN. Both Server/Staging and REST containers have been integrated with the Monitoring Infrastructure at CERN, based on Fume/ES/Kibana, so logs are automatically published and metrics are available on dedicated dashboards.

Issues with the current deployment

While we scale all services in parallel, the fact is that the scaling needs of each one is different.

For instance, we may need to add a new VM to sustain the amount of parallel transfers required to process the queue, but there may be no staging operations pending, or the amount of HTTP requests may actually remain constant.

Therefore, we are scaling services that do not need to scale, those additional services will consume resources (mainly RAM) that could be used to run more transfers in parallel.

Here we evaluate the containerization of FTS as a possible way of independently scaling services, making the overall operation cheaper in terms of computing resources.

Magnum

Magnum is an OpenStack API service making container orchestration engines such as Docker Swarm, Kubernetes, and Apache Mesos available as first class resources in OpenStack.

Magnum uses Heat to orchestrate an OS image which contains Docker and Kubernetes and runs that image in either virtual machines or bare metal in a cluster configuration.

The installation at CERN is using VMs for the moment and TLS support. It has been extended with specific customizations for HEP, like CVMFS and EOS support (under integration).

Links

- <https://wiki.cern.ch/display/fts/fts3dockerize-ftsd3>
- <https://wiki.cern.ch/display/fts/fts3kubernetes-ftsd3>
- <https://wiki.cern.ch/display/fts/fts3mesos-ftsd3>
- <https://wiki.cern.ch/display/fts/fts3monitoring-ftsd3>

Docker Swarm

First tests were performed with Docker 1.10 and Docker Swarm 1.2.3 deployed via Magnum. FTS3 configuration and x509 certificates have been installed directly on the Bays nodes and mounted via Docker volume s.

We managed to run the 3 FTS container types without issues, but we missed some functionalities implemented only in Docker 1.12 not yet available at CERN via Magnum (Docker Service in particular). Therefore we moved ahead our testing to Kubernetes.

Kubernetes

A Kubernetes (v 1.2.0) cluster deployed via Magnum allowed us to:

- perform scaling of the FTS Server/Staging component according to our needs. A first component in Python was developed integrating Kubernetes REST API to autoscale based on the FTS transfer queues;
- set up a unique REST API entry point via Kubernetes Services and HA-Proxy. The FTS REST containers could be scaled over the cluster transparently to users;
- write logs to Persistent Volumes (locally but EOS integration is also foreseen);
- test rolling upgrades

Future Work

The scaling at the level of the cluster VMs performed in Magnum via Heat is the next item on the list, together with new tests with Swarm when the latest features will be available. A mixed VM + Docker deployment is now possible but we aim first at reaching the same confidence at operating our VM cluster before eventually moving to a full dockerized FTS service in production.

Future developments FTS side, which will allow further isolation of the Server, Staging, Optimizer and Scheduler functionalities are foreseen: a micro-service architecture is crucial in order to benefit from the Swarm/Kubernetes Orchestration.

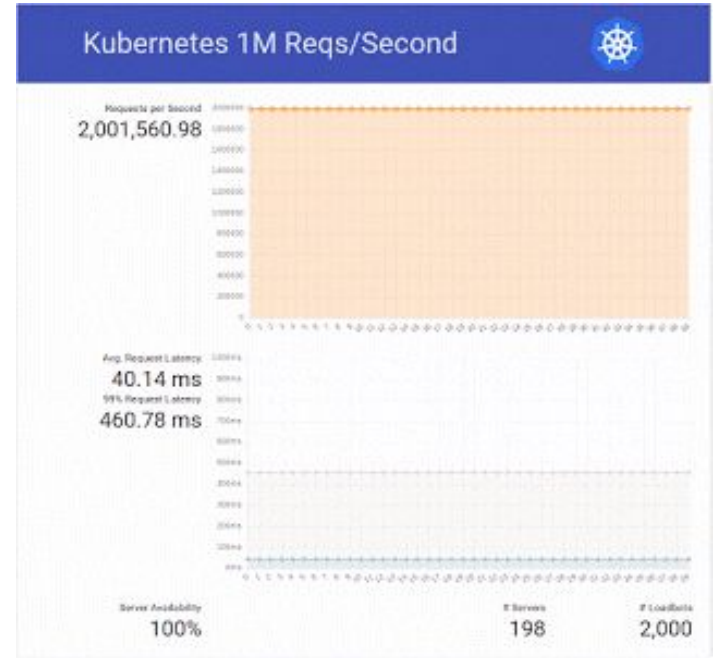
<https://fts3-service.web.cern.ch> fts-devel@cern.ch

Use Cases

- And many others used to collect requirements...
 - Continuous Integration / Deployment
 - Swan / MyBinder / Jupyter Notebooks
 - ML / TensorFlow
 - ...
- These have triggered a lot of the work we've done in the last months
 - Collaboration with Rackspace via CERN OpenLab
 - Indigo DataCloud project
 - <https://indico.cern.ch/event/505613/contributions/2227435/>

Performance

- How fast is cluster deployment?
- How does it scale with cluster size?
- How does a cluster itself scale?
- First try (May 16):
 - Kubernetes 1 million reqs/sec
 - Suboptimal latency
 - But we got to **2 million reqs/sec**

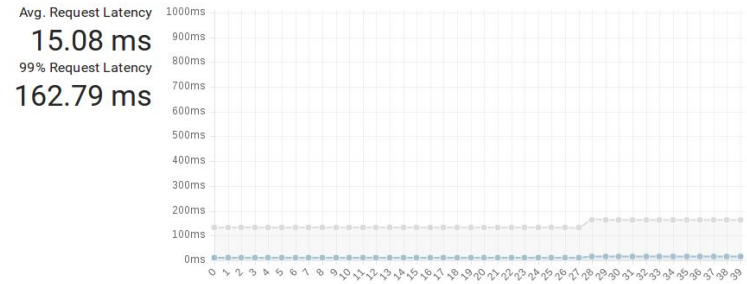
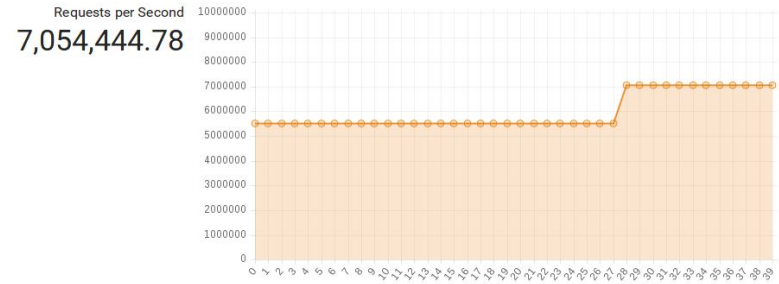


Performance

- Second try (Aug 2016)
 - Much **better latency**
 - Managed **7 million requests / sec**
- And an analysis of cluster deployments

Cluster Size (Nodes)	Deployment Time (min)
2	2.5
32	4
128	5.5
512	14
1000	23

Kubernetes 1M 10M Reqs/Second



Server Availability 100% # Servers 500 # Loadbots 9,449



Conclusion

- Production end of October 2016
- Swarm, Kubernetes, Mesos one click away
- Integration with common HEP services
 - CVMFS, EOS
- Extensive scalability tests
 - Clusters of 1000 nodes, millions of requests / sec
- Ongoing Work
 - Node Groups
 - Lifecycle Operations

<http://clouddocs.web.cern.ch/clouddocs/containers/index.html>