A container model for resource provision at a WLCG Tier-2

Gareth Roy, Gordon Stewart, David Crooks, Sam Skipsey & David Britton. 
ACAT 2017: 21-25 August, University of Washington
Overview

• Motivation

• Requirements from a Tier-2 perspective

• A worker node container

• Monitoring & Logging

• CI/CD

• Conclusion
UKI-SCOTGRID-GLASGOW

- At present consist of:
  - 6200 CPU cores (with 1760 of that running VAC)
  - 63643 HEPSPEC.
  - 3.8 PB of Storage.
  - 160 Gb/s internal network bandwidth.
  - Primarily supports ATLAS and LHCb, acts as a CMS Tier-3 along with other smaller experiments and local user groups.
• Glasgow examining methods to reduce overall manpower for running a Tier-2.

• At present run both traditional Grid resources as well as a Vac pool (25% or resources)

• Vac pros:
  • Simple.
  • Lightweight.
  • Auto-updating.

• Vac cons:
  • Slow to start VMs.
  • Inflexible resource requirements.
  • Opaque to local monitoring.
  • Scheduling is naive.

• Our goal is to take the good ideas from Vac and bring them into our standard site through the use of containers.
Requirements

• WLCG Requirements:
  • CVMFS, needed for access to experiments software stacks.
  • HEP_OSlibs, for many experiments a standard baseline of installed packages (provides 32-bit compatibility as well as many other required libraries).
  • ca-policy-egi-core/ca-policy-lcg, needed to meet WLCG trust chain requirements.

• Our Requirements:
  • Lightweight resource provision, quick to start and preferably ephemeral.
  • Connects to our existing HTCondor pool to leverage existing infrastructure.
  • Good monitoring that integrates with existing local tools.
  • Use “standard” tools, with broad support base.
Integrating CVMFS with a container (Docker)
Integrating CVMFS

(a)

- Statically mount CVMFS filesystems to /cvmfs via /etc/fstab
- Bind mount into container with volume mounts (-v /cvmfs:/cvmfs)

Pros:
- shared cache.
- simple bind semantics.

Cons:
- all required repositories need to be statically mounted.
- breaks reboot (at least on vanilla centos7).

Promising alternative CVMFS volume driver (https://gitlab.cern.ch/cloud-infrastructure/docker-volume-cvmfs/)
Integrating CVMFS

- Install autosfs within container and run as a service.
- Container needs escalated privileges and fuse device to be mounted ( `--cap-add MKNOD --cap-add SYS_ADMIN --device /dev/fuse` )

**Pros:**
- no CVMFS install required on “hypervisor”.
- allows use of read-only linux distributions (coreos, rancheros).

**Cons:**
- no shared cache, approximate with shared local squid.
- container needs to be run with elevated privileges.
WN Container (with bind mounted CVMFS)

- Total size is 851.9MB.
- Based on centos 6 for compatibility (centos 7 planned).
- Includes HEP_OSlibs as a common baseline for WLCG experiment code.
- Distributed to work nodes via an internal docker registry.
- In future replace HEP_OSlibs with experiment provided singularity container to reduce size.
Caveats

- Uses standard “grid style” pool accounts - would like to move to HTCondor per slot accounts.
- HTCondor configured to use CCB due to private networking between collector and per container startd's.
- In a non-privileged container no access to cgroups or other privileged information so the following parameters need to be unset in our condor configuration:
  - BASE_CGROUP=
  - DISCARD_SESSION_KEYRING_ON_STARTUP=FALSE
- Resource constraints applied to container at run time, not via HTCondor interaction with groups (can cause issues with eviction constraints).
Grid environment via CVMFS

- Using the same method as ATLAS VAC VM’s we load a grid environment from CVMFS, along with all required CA certs and CRLs.
- Reduces image size, and decouples grid middleware from container image.
- In practice the grid environment is loaded via HTCondor at job run time with a custom user job wrapper (c.f. ATLAS VAC payloads)
- Using this process it is possible to create VAC-like containers that can contact experiment pilot factories directly.
A running container

As observed from the “Hypervisor”
Monitoring & Logging

• Local monitoring is essential to enable sites to identify issues and optimise workloads.

• Identified standard tools which can be deployed as containers:
  • For metric gathering uses cAdvisor (stand alone container here, built in when using Kubernetes)
  • For aggregation, gather metrics from each instance via Prometheus.
  • Forward logs from each container via LogSpout
    • Logging is not perfect as stdout grabbed by condor and pilot payloads and not “tee’d” back to stdout.
    • Currently only logging HTCondor logs (as well service container logs e.g. prometheus).
  • Log aggregation using oklog (will be moved to Elasticsearch once site instance becomes available).
cAdviser (via container or built into Kubernetes)
Aggregate Metrics via Prometheus

Prometheus GUI showing metrics for container CPU usage and scrape status for docker_hosts and prometheus endpoints.
Aggregate Metrics via Prometheus

Atlas Multi-Core Payloads
Log aggregation with logspout & oklog

Lightweight log aggregation, with “grep-like” semantics. To scale up oklog could be replaced with an ELK stack.
Future Deployment & Orchestration

WN container  WN container  logspout  cadvisor  pod

kubernetes

cvmfs  docker

Linux (centos 7)

```
[root@kube002 ~]# docker ps -a
CONTAINER ID   IMAGE               COMMAND                  CREATED               STATUS
9e75acb6d825   scotgrid/cntwn     "/cntwn.sh"               7 days ago            Up 7 days
255e86f7a0fc    scotgrid/cntwn     "/cntwn.sh"               7 days ago            Up 7 days
16d90fbbdb4b    scotgrid/cntwn     "/cntwn.sh"               7 days ago            Up 7 days
68ff46f29364    scotgrid/cntwn     "/cntwn.sh"               7 days ago            Up 7 days
b222b002f02ba   gliderlabs/logspout:latest "/bin/logspout syslog" 13 days ago            Up 13 days
83e3531217af    google/cadvisor:latest "/usr/bin/cadvisor -l" 2 weeks ago             Up 2 weeks

[0.0.0.0:9621->9621/tcp  kube002-04
0.0.0.0:9620->9620/tcp  kube002-03
0.0.0.0:9619->9619/tcp  kube002-02
0.0.0.0:9618->9618/tcp  kube002-01
80/tcp  logspout
0.0.0.0:8000->8000/tcp  cadvisor
```
Future Deployment & Orchestration

<table>
<thead>
<tr>
<th>CONTAINER ID</th>
<th>IMAGE</th>
<th>COMMAND</th>
<th>CREATED</th>
<th>STATUS</th>
<th>PORTS</th>
<th>NAMES</th>
</tr>
</thead>
<tbody>
<tr>
<td>9e75ac6b6d825</td>
<td>scotgrid/ntwn</td>
<td>&quot;/ntwn.sh&quot;</td>
<td>7 days ago</td>
<td>Up 7 days</td>
<td>0.0.0.0:9621-9621/tcp</td>
<td>kube002-04</td>
</tr>
<tr>
<td>255e86f7a0fc</td>
<td>scotgrid/ntwn</td>
<td>&quot;/ntwn.sh&quot;</td>
<td>7 days ago</td>
<td>Up 7 days</td>
<td>0.0.0.0:9612-9612/tcp</td>
<td>kube002-03</td>
</tr>
<tr>
<td>16d90fbdb04b</td>
<td>scotgrid/ntwn</td>
<td>&quot;/ntwn.sh&quot;</td>
<td>7 days ago</td>
<td>Up 7 days</td>
<td>0.0.0.0:9614-9614/tcp</td>
<td>kube002-02</td>
</tr>
<tr>
<td>68ff46129364</td>
<td>scotgrid/ntwn</td>
<td>&quot;/ntwn.sh&quot;</td>
<td>7 days ago</td>
<td>Up 7 days</td>
<td>0.0.0.0:9615-9615/tcp</td>
<td>kube002-01</td>
</tr>
<tr>
<td>b222002f002ba</td>
<td>gliderlabs/logspout:latest</td>
<td>&quot;bin/logspout syslog&quot;</td>
<td>13 days ago</td>
<td>Up 13 days</td>
<td>80/tcp</td>
<td>logspout</td>
</tr>
<tr>
<td>83e3531217af</td>
<td>google/cadvisor:latest</td>
<td>&quot;/usr/bin/cadvisor -l&quot;</td>
<td>2 weeks ago</td>
<td>Up 2 weeks</td>
<td>0.0.0.0:8080-8080/tcp</td>
<td>cadvisor</td>
</tr>
</tbody>
</table>
CI/CD

- The goal is to have our compute resources be as ephemeral as possible.
  - Ideally a new container for each job run, to isolate and allow continuous deployment of upgrades.
  - This simplifies security patches etc., as they will be pushed on each container restart.
  - Allows “canary” builds to be deployed to test upgrades and also allows easy rollback by starting previous container.
- To enable this use standard continuous integration tools:
  - GitLab (for revision control).
  - gitlab-runner (for building and pushing containers).
  - private docker registry for distributing images.
  - simple cron for pulling updated images.
CI/CD

admin

```
#456 by 🐧
master → 189bc67b
Atiempting without DIND but bind ...
```

worker node

```
docker pull
```

- **Admin**
  - `git push` → `gitlab` → `gitlab-runner` → `docker build`

- **Worker Node**
  - `docker pull` → `private registry` → `docker push`
Conclusion

- Examined the requirements of a container to run WLCG payloads.
- Created a nominal container and established PoC by running production ATLAS multi-core payloads.
- Integrated monitoring, logging and constructed a CI/CD pipeline to automate updates to containers.

Next Steps:

- Give each container a “lifespan” so they auto-expire.
- Integrate with Kubernetes (or other mechanism) to get auto-container redeploys (via services).