How to build and run OCI containers

A shallow dive on the OCI container configuration and an overview of the available tools

Fosdem Brussels 2018 - Spyros Trigazis @strigazi

whoami

Spyros Trigazis
Computing Engineer at CERN’s cloud team
Project Team Lead of OpenStack/Magnum

OCI containers

A linux container is just process controlled by linux kernel cgroups and namespaces.

OCI containers: defined by the OCI specification

- Runtime-spec: defines the config.json file from which a runtime know to run a container
- Image-spec: defines an OCI Image, consisting of a manifest, an image index (optional), a set of filesystem layers, and a configuration
- Filesystem-bundle: consists of the config.json file and the container root filesystem

Why we need them?

Containers need a supervisor, eg Docker daemon, kubelet, systemd etc

Systemd -> dockerd -> container, doesn’t play very well

Kubernetes implemented its own cri specification

- cri-contianerd: cri implementation based on containerd
- cri-o: cri implementation based on on runc

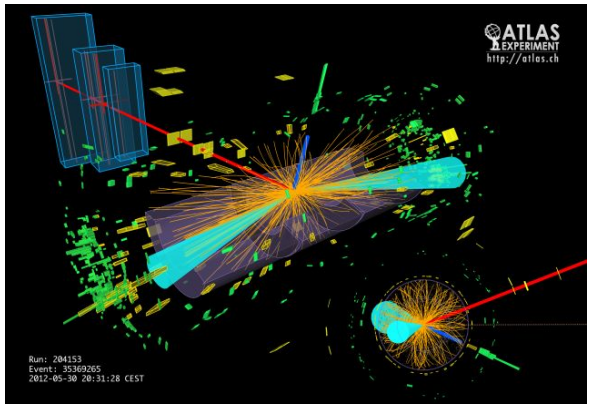
What if you need a service running that docker depends on it?

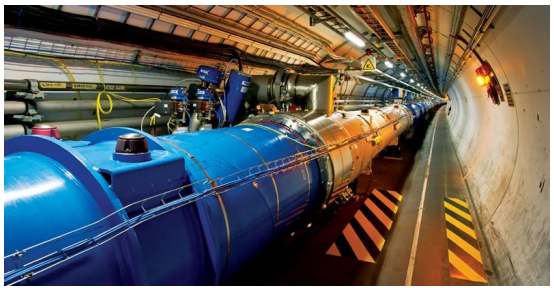
CERN container service

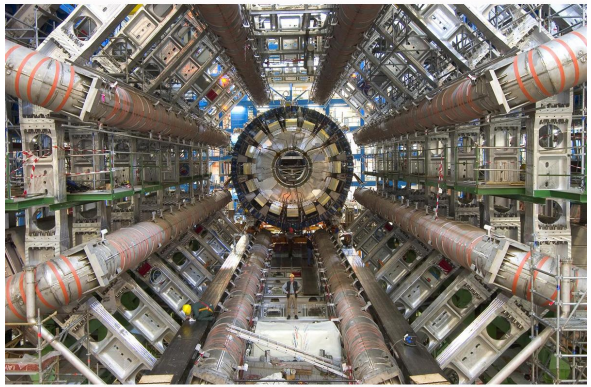
Fosdem Brussels 2018 - Spyros Trigazis @strigazi

CERN Container Use Cases

- Batch Processing
- End user analysis / Jupyter Notebooks
- Machine Learning / TensorFlow / Keras
- Infrastructure Management
  - Data Movement, Web servers, PaaS …
- Continuous Integration / Deployment
- Run OpenStack :-)
- And many others





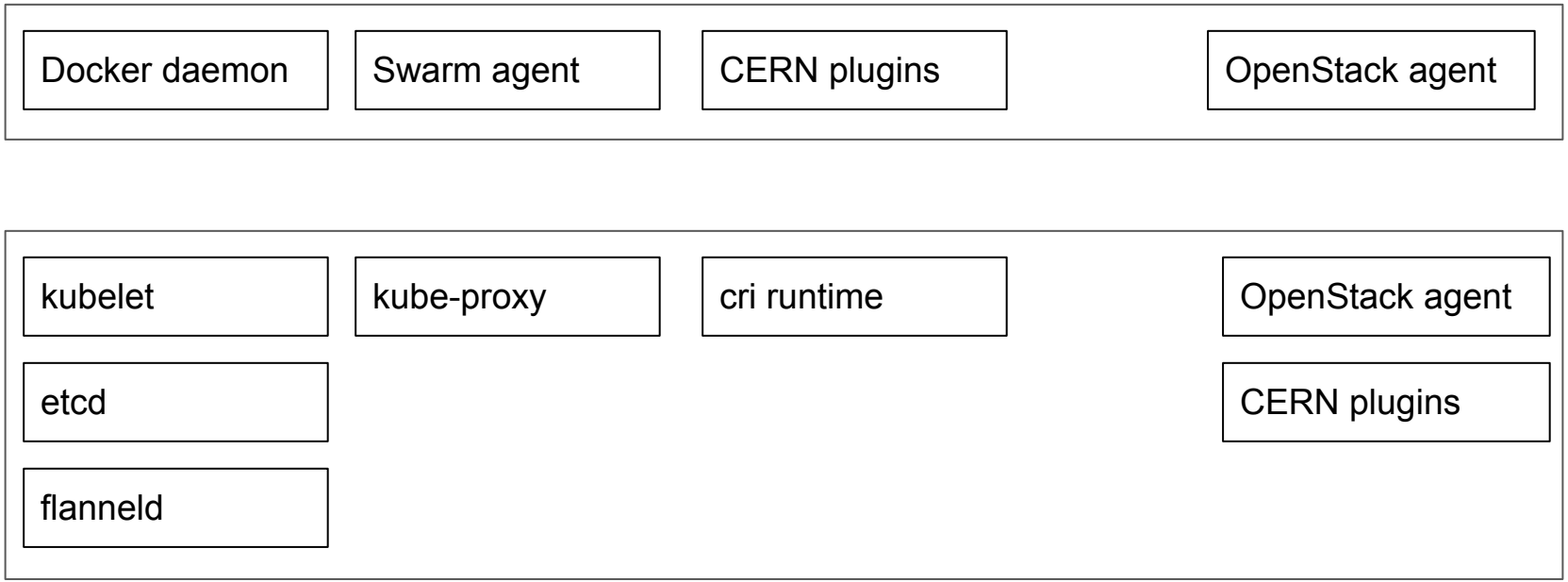


CERN Magnum Deployment

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

```
$ openstack coe cluster create --name myswarmcluster --cluster-template swarm --node-count
100
     ~ 5 mins later
$ openstack coe cluster list
+------+----------------+------------+--------------+----------------------------+
| uuid | name           | node_count | master_count | keypair   | status         |
+------+----------------+------------+--------------+----------------------------+
| .... | myswarmcluster | 100        | 1            | mysshkey | CREATE_COMPLETE |
+------+----------------+------------+--------------+----------------------------+
$ $(openstack coe cluster config myswarmcluster --dir magnum/myswarmcluster)
$ docker info / ps / ...
$ docker service create --mount
'type=volume,volume-driver=cvmfs,source=cms.cern.ch@trunk-previous,destination=/cvmfs/cms.cer
n.ch' busybox sleep 10000
```

Cluster node layout (swarm or kubernetes)



Building OCI containers

Building a system container

Create:

- a dockerfile
- a systemd template
- a list of tmpfiles (if needed)
- a manifest.json for default values(if needed)
- the config.json

https://github.com/projectatomic/atomic-system-containers/blob/master/FILES.md

Building a system container

```
$ git clone https://github.com/projectatomic/atomic-system-containers.git
$ sudo docker build -t strigazi/hello-system-container:latest .
...
$ sudo docker push strigazi/hello-system-container:latest

$ sudo atomic install --system --name hello-world \
> strigazi/hello-system-container:latest
...
Copying config sha256:24908c0e3d9d78d43843834c549b918df24af3ee9bc91c5e8866e4af90f4ca4f
 3.29 KB / 3.29 KB [=======================================================] 0s
Writing manifest to image destination
Storing signatures
Extracting to /var/lib/containers/atomic/hello-world.0
systemctl daemon-reload
systemctl enable hello-world
$ sudo systemctl start hello-world
```

Populating config.json

Namespaces

```
"namespaces": [
     {
          "type": "mount"
     },
     {
          "type": "ipc"
     },
     {
          "type": "uts"
     }
],
```

mount, cgroup, ipc, network, pid, user, uts
http://man7.org/linux/man-pages/man7/namespaces.7.html

Populating config.json

capabilities

```
"capabilities": {
"bounding": [
     "CAP_CHOWN",
     "CAP_DAC_OVERRIDE",
     "CAP_DAC_READ_SEARCH",
     "CAP_FOWNER",
     "CAP_FSETID",
]
```

http://man7.org/linux/man-pages/man7/capabilities.7.html

```
# #get the docker container pid
# grep ^Cap /proc/4037/status
# grep ^Cap /proc/4037/status | awk '{print $2}' | xargs -I val capsh --decode=val
```

Populating config.json

mounts

```
"mounts": [
     {
          "type": "bind",
          "source": "/srv/magnum",
          "destination": "/srv/magnum",
          "options": [
               "rbind",
               "rw",
               "rprivate"
               ]
          },
...
```

Links

https://github.com/opencontainers/runtime-spec
https://github.com/opencontainers/image-spec
https://github.com/projectatomic/atomic-system-containers
http://openstack-in-production.blogspot.be/
https://docs.openstack.org/magnum/latest/

Ping me:

@strigazi
strigazi on Freenode #openstack-containers #atomic #centos-devel #fedora-devel