Setup of an AliEn site with containers using Plancton

Matteo Concas (Politecnico di Torino)
matteo.concas@cern.ch

Thursday - November 3rd, 2016

What is Plancton? (1)

- A service that automatically and continuously spawns Docker containers according to the available resources (github.com/mconcas/plancton)
  - It uses Docker APIs through the docker-py module: github.com/docker/docker-py
  - Runs as a **standalone** instance on each machine ➙ each daemon is independent from the others in a cluster
  - Plancton is **stateless**: daemon and its configuration can be updated without affecting running containers
  - **Rolling updates**: it is possible to gradually replace “old” containers with ones based on newer images as soon as they terminate (thanks to Docker)

What is Plancton? (2)

- Suitable both in **opportunistic** and **dedicated** scenarios
- It periodically **measures host usage** and places new containers whenever possible (thresholds are configurable)
  - **Easy to install** and **configure**, (a dry-run instant gratification example)

```
$ sudo pip install plancton                 # Install system-wide
$ sudo plancton-bootstrap --dryrun          # Use default values for a demo
$ watch docker ps -a                        # Check workflow
```

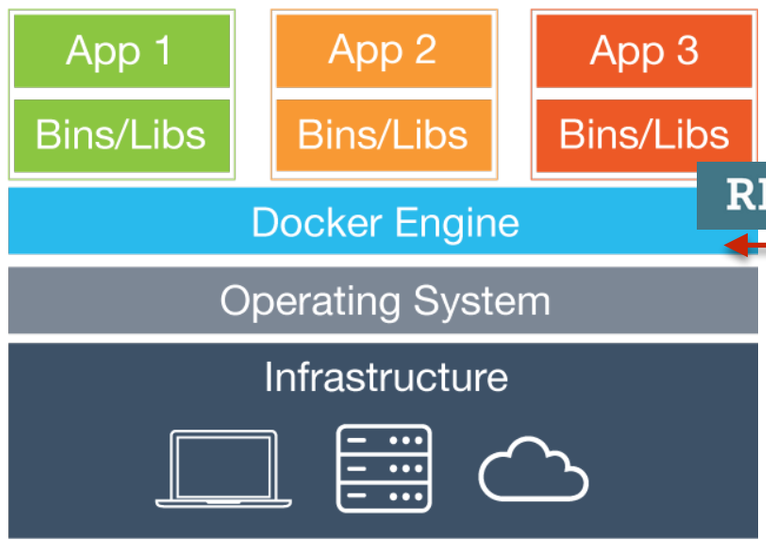
  - Plancton forwards container parameters from its configuration to Docker
  - Plancton pulls config from a GitHub repository of your choice, *e.g.*:

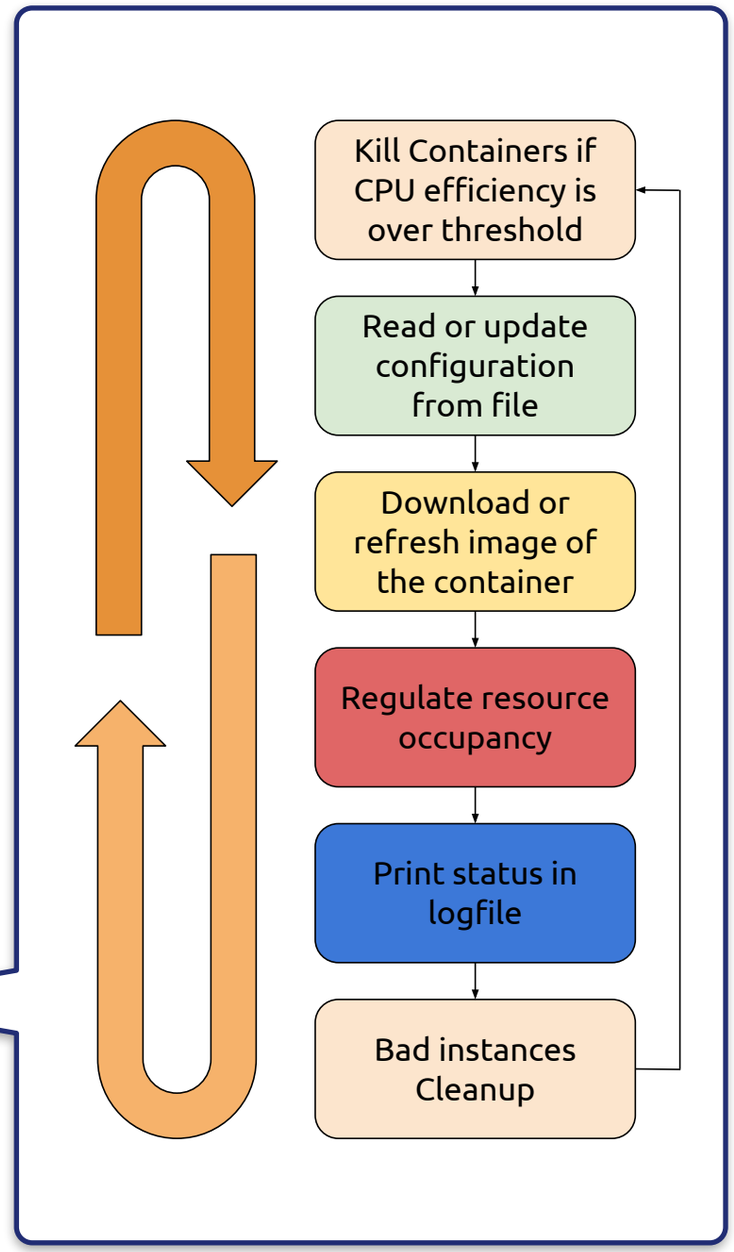
```
# plancton-bootstrap mconcas/plancton-conf:cern-alien/dev
```

- In a dedicated scenario it can be set up to just continuously spawn new containers to replace terminated ones

Plancton: a “simple” container scheduler

- Workflow
  - Check for **available resources** (CPU usage)
  - If enough resources available ➙ **start new container**
  - Other basic **checks** and **cleanup** (clean up exited containers, query status, etc…)
  - Check **overdue containers** (jobs beyond TTL…) and handle misbehaviours





What Plancton is not?

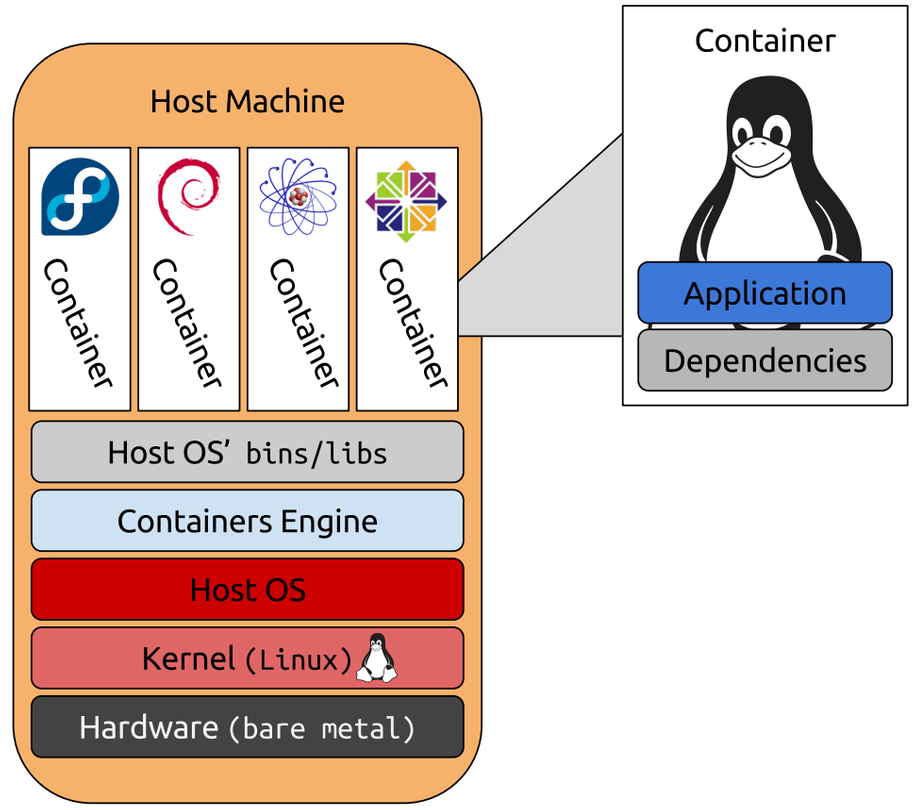
- Plancton is not **Kubernetes**, **Apache Mesos** or **Docker Swarm** and does not aim to replace them!
- It provides a **lightweight** and **easy** way to adapt preexistent batch systems architectures based on pilot jobs, enclosing them within Docker "pilot" containers
- It is **application agnostic**: it does not care about the applications running inside, it does not interact nor manages them ➙ use cases are confined in Docker containers

Core technology: Linux containers

- It has a **short deployment time**, and the possibility to run different Linux environments

- Is it possible to efficiently run jobs inside containers?

  - They offer the possibility to **isolate resources** (RAM, disk, cores, CPU usage)
  - They can provide a **consistent** environment needed for job executions
  - They also constitute a good (but not exclusive)* way to **sandbox each others job executions** ➙ 1 job = 1 container



* The infamous Dirty Cow (CVE-2016-5195) also affected Linux containers: containers do provide an additional security wall in most cases, but they are not an excuse for overlooking other security policies!

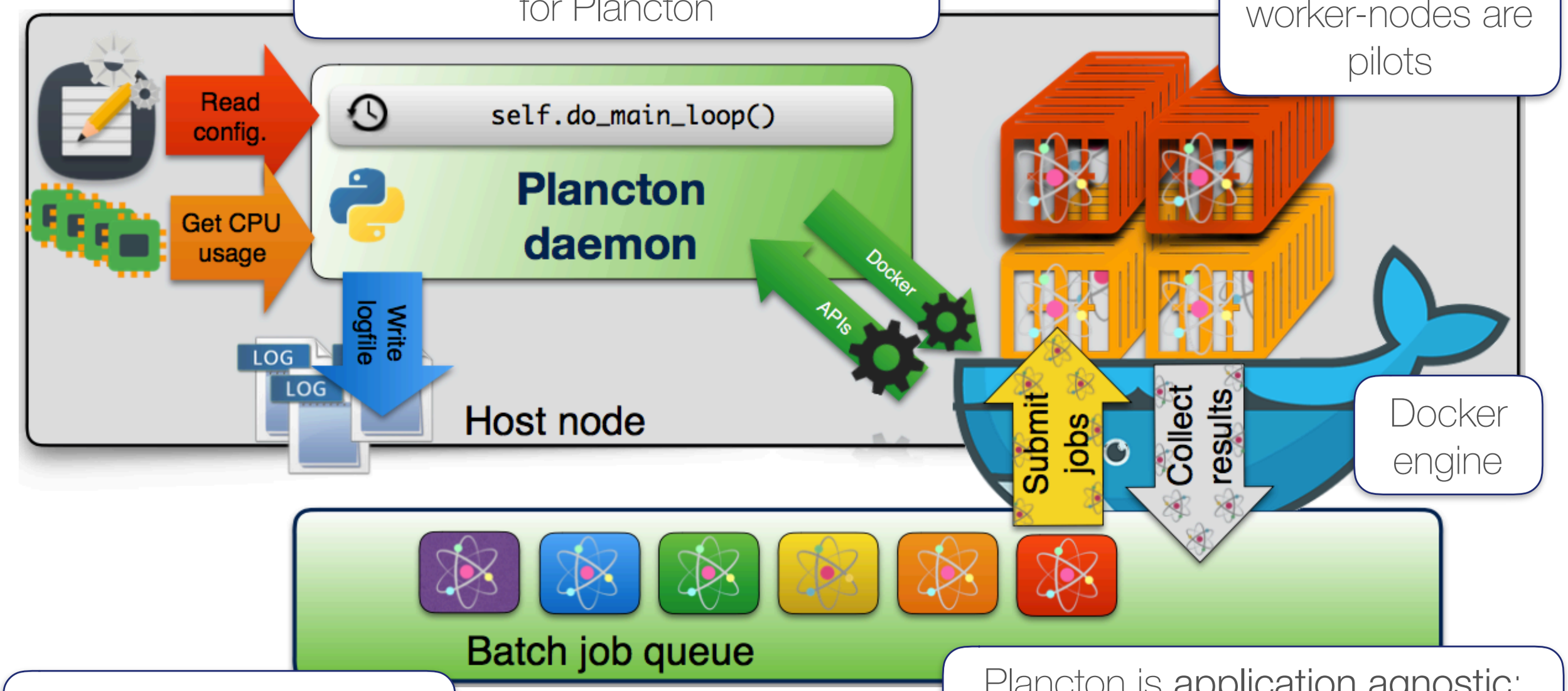
From “pilot jobs” to “pilot containers”

- Short deployment time (~secs) and negligible overhead on launch: it is possible to **use containers as pilots** as it is inexpensive to continuously spawn them
  - The `ENTRYPOINT` of a container is a “pilot” executable: its aim is to start services needed to attach the *worker-node* to the batch system (e.g. HTCondor, Alien-WQ). When the pilot dies, the container dies
  - They act like pilot jobs, the only difference is that the execution is “wrapped” into a correct environment, isolated from the rest of the process tree
  - On a 24-cores machine we can run 24 single-core containers each one executing a job, with the possibility to optionally define resource limits, with a fine granularity

Architecture

There is no centralised management for Plancton

Docker containers worker-nodes are pilots



Docker engine

Users submit their job to local batch system: transparent to users

Plancton is **application agnostic**: It does not need to be aware of the job-scheduling application

Setup of the HLT_DEV cluster with AliEn-WQ and Plancton

Dedicate Grid site: ALICE VOBox+Docker+Plancton

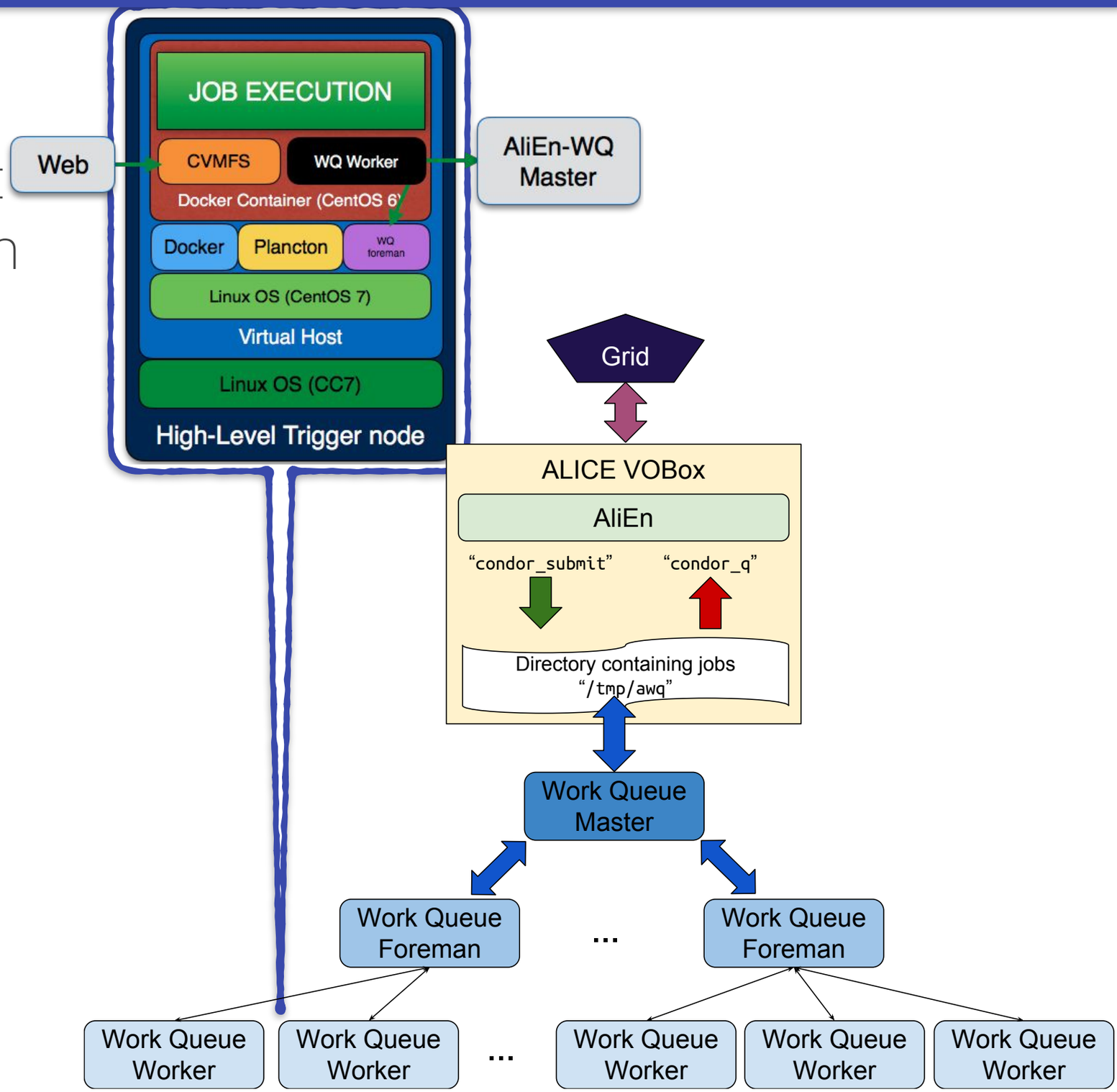
- **Hypothesis**: configure Plancton to use every available resource
  ➙ never kill containers unless they exceed a TTL threshold
- **What**: ALICE Grid site based on Plancton, just for centrally-managed Monte Carlo productions
- **Where**: High Level Trigger development cluster at CERN
  - ~800 cores ➙ ~800 containers
  - 1 job = 1 container
- **How**: AliEn Work Queue backend running on pilot containers
  - Configure the **AliEn VOBOX** as a **Condor** site: fake **condor_q** and **condor_submit** commands will submit jobs to AliEn-WorkQueue instead
  - **Software** provisioning: **CVMFS** mounted from outside the containers (efficient: cache is retained)

Plancton+AliEn+Work Queue integration

- Container image: **alisw/slc6-builder** on DockerHub (a bare CentOS 6 image with Grid worker node packages)
- Jobs are submitted from AliEn to a Work Queue Master
- Containers run single-shot Work Queue Workers
  - A worker connects back to the Work Queue master or to its Foremen (running on the host) and fetches a single AliEn job agent
  - One job per container: worker exits when job agent is done, container is terminated
  - If base container image is updated, new jobs will use it
- **Note:** currently running Plancton inside VMs on the HLT cluster (not natively)

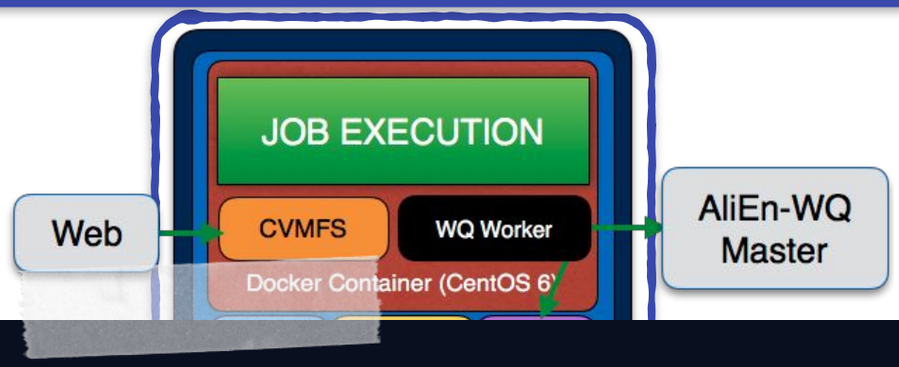
Site architecture

- Monte Carlo jobs "wrapped" in a consistent environment, with limitation of resources
- Minimal configuration: it just does continuous scheduling, TTL control and garbage collection
- Rolling updates: using Plancton and containers for zero-downtime continuous upgrades



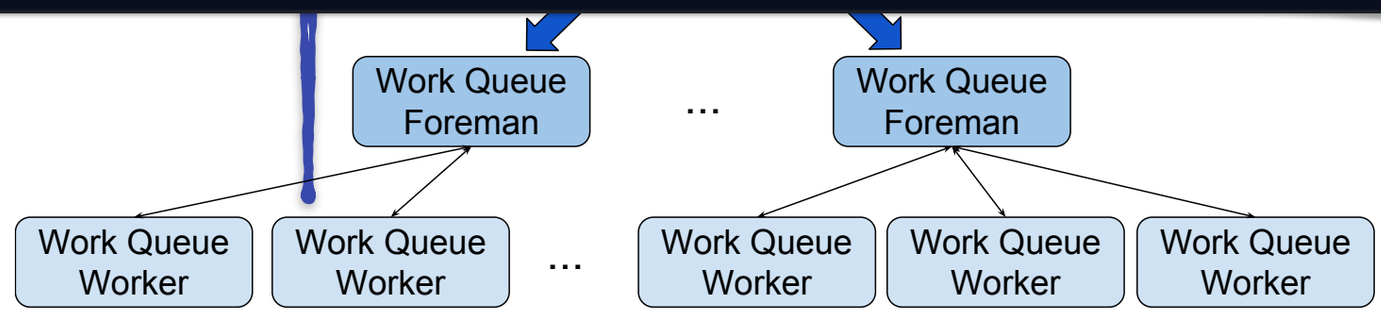
Site architecture

- Monte Carlo jobs “wrapped” in a consistent environment, with limitation of resources



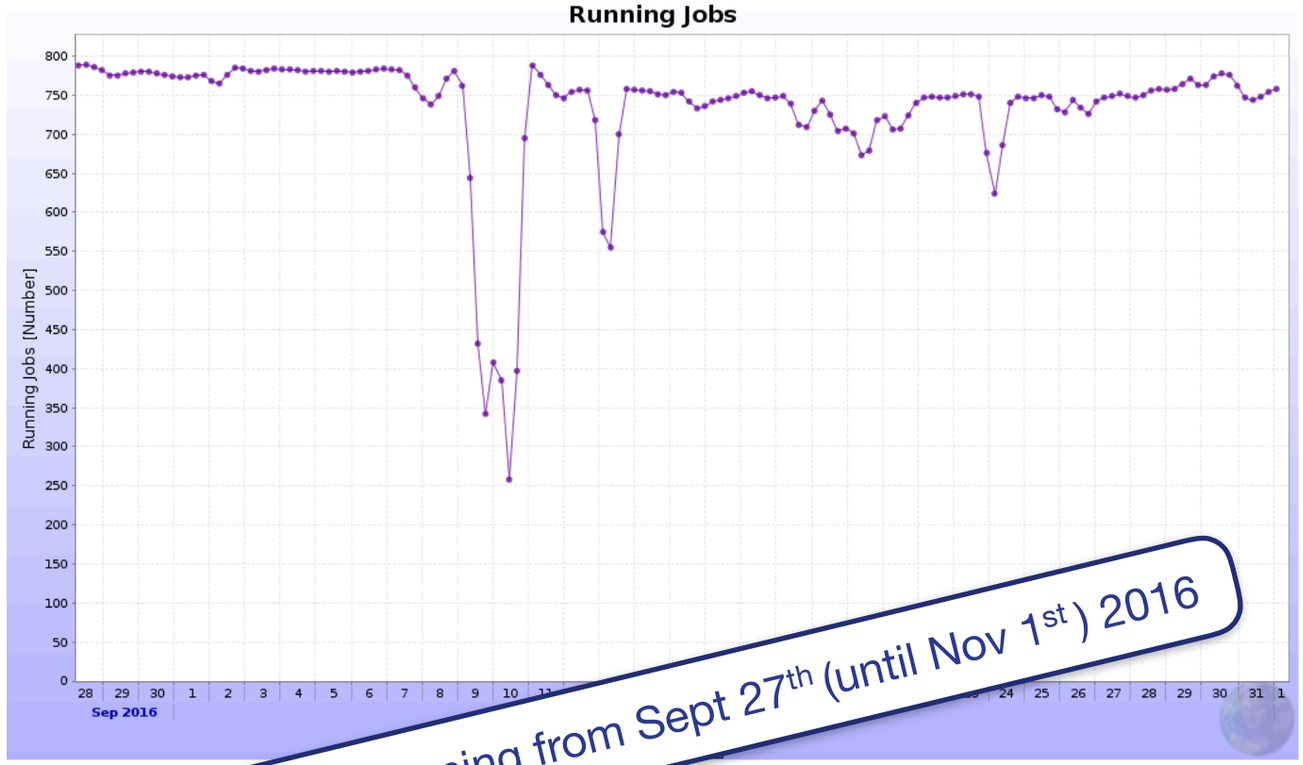

```
---
updateconfig: 10
main_sleep: 10
grace_kill: 100
grace_spawn: 100
cpus_per_dock: 1
max_docks: ncpus
max_ttl: 172800
docker_image: alisw/slc6-builder
docker_cmd:
 - /cvmfs/alice.cern.ch/bin/alienv
 - setenv
 - AliEn-WorkQueue/v1.3-1
 - -c
 - "work_queue_worker --cores 1 --debug all --single-shot --single-task --timeout 60 10.162.223.250 $((RANDOM % 12 + 9080
docker_privileged: False
max_dock_mem: 2750000000
max_dock_swap: 1000000000
binds:
 - /cvmfs:/cvmfs
```

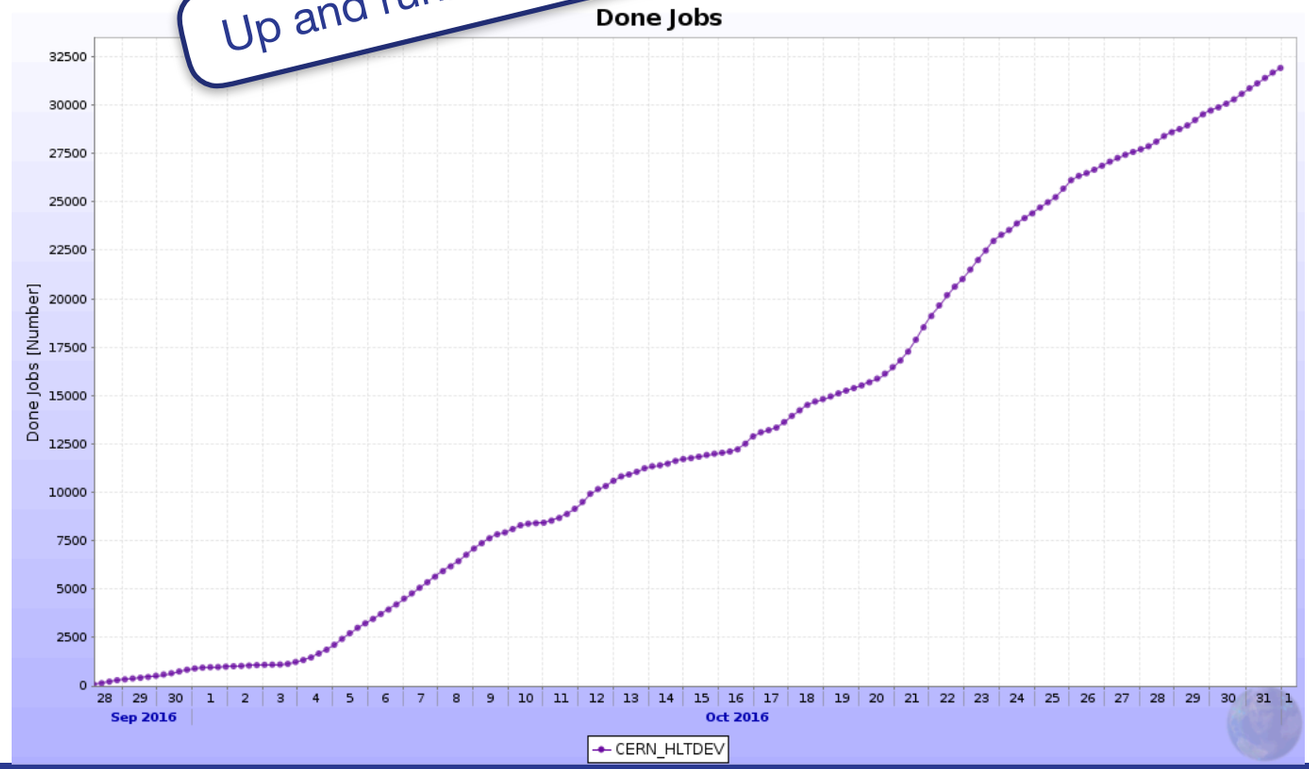
- Plancton and containers for zero-downtime continuous upgrades



Conclusions

- Dedicated Grid site at ALICE HLT_DEV cluster
  - Plancton proven effective
  - Service is up and running: ~30000 jobs successfully done until Nov 1st 2016
  - No difference in job failures since when we switched to containers
  - No impact on CPU efficiency or job timings





Up and running from Sept 27th (until Nov 1st) 2016