Plancton: an opportunistic distributed computing project based on Docker containers

Matteo Concas, Università di Torino

Thursday, 19 May 2016 - Workshop di CCR

Outline

- The Problem - Underused workstations
- The opportunity
- What is Plancton for
- A disposable pilot approach
- What Plancton is not
- What Plancton is
- Use case example (1) - Volunteer batch facility for ALICE
  - Volunteer container with HTCondor and ALICE software
  - Results
- Use case example (2) - ALICE Grid site
- Conclusions
- Outlook

The problem

- A set of **underused workstations** in a **controlled LAN** (e.g. INFN Torino)
  - Idle for most of the time, rarely utilised at high regime ➙ lots of wasted CPU cycles
  - Occasionally used remotely ➙ in principle can't be turned on/off on demand
  - ~1 host per office ➙ ~10-20 hosts at least in ALICE Torino
  - Relatively powerful hardware (i7 processor, 8~16GB RAM, 0.5-1 TB HDDs …)
  - Linux is installed as main OS, lots of flavours (RHEL/Debian based, *modern* kernels)
  - Decent network ➙ 100Mb/s bandwidth

The ~~problem~~ opportunity

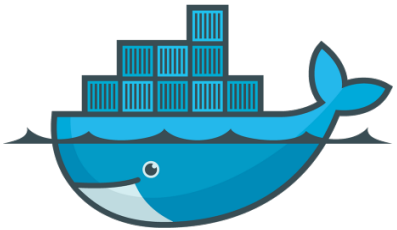
- Workstations owned by physicists belonging to HEP experiments
  - Their software are **voracious of computing resources** and based on distributed high-throughput computing paradigm (event based)
  - They often need to **test their jobs**, software and simulations quickly, on very small datasets ➙ no Grid, but a test farm would be better than testing on laptops
- Idea: build an opportunistic volunteer computing facility based on idle workstations where customers are the same people who share their spare CPU cycles

What is Plancton for

- Providing consistent environments for job execution
- Ensuring isolation between jobs

} Linux containers provide such features

Docker makes Linux containers trivial



- Container deployment for free
- Easy interface for cooking new images
- exposes powerful REST API

A disposable pilot approach

- Thanks to their low deployment times and their low overhead, it is possible to adopt an approach based on pilot containers
  - Container are spawned with a limited time to leave (TTL)
  - They patiently wait for a job; if anything comes it is processed; container exits and it is disposed of at the end → **one job - one container**
  - If no job comes the container exits after a short while without doing anything
- With Virtual Machines this approach is too onerous
  - Plenty of overhead and non-negligible deployment time → disposable approach not convenient

What Plancton is not

- Plancton is not a batch system
- It is s not a tool for scheduling containerised apps on the scale (no Mesos)
  - It scales horizontally because Plancton instances are installed on many hosts and don't communicate with each other: they just spawn containers
- Application and clusterisation logics are not Plancton's business
  - It is completely application-agnostic ➙ provides a general purpose service

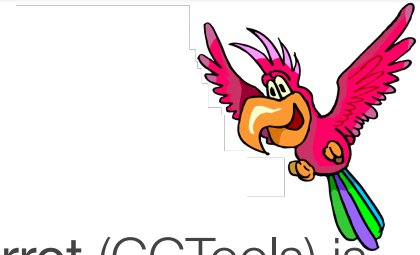
What Plancton is

- It is a simple automation: while(1) on stamina
  - Periodically repeat a simple routine
  - Case-specific requirements are specified through configuration files
- Workflow
  - Check **available resources** (CPU measurement, …)
  - If enough resources available ➙ **start new container** (multiple Docker images possible)
  - Other basic checks and operations (**clean up exited** containers, query status, etc…)
  - Check **overdue** containers (job taking too long, stopped ones…) to handle misbehaviours

Use case example - Volunteer batch facility for ALICE

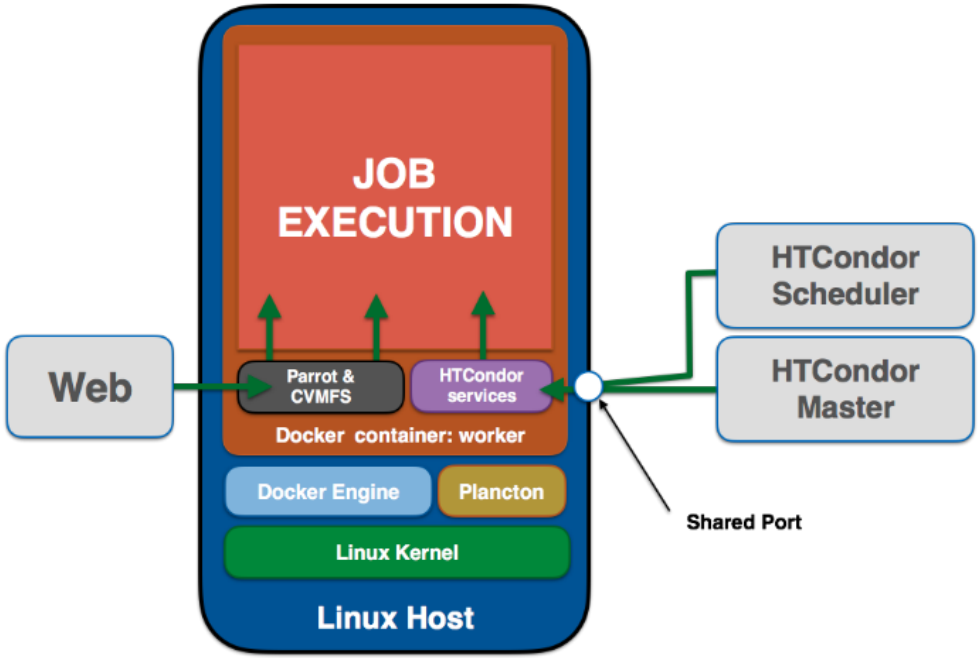
- **Head Node** (dedicated login machine)
  - HTCondor schedd, collector daemons → needed for job submission
- **Host Nodes** (Plancton nodes)
  - Plancton spawns Docker containers designed for volunteer computing
  - Physical nodes require only Plancton and Docker - works in a firewalled environment (no incoming connections)
  - Each Plancton daemon may push metrics for monitoring/accounting purposes (work in progress with Elasticsearch)

Volunteer container with HTCondor and ALICE software



- ALICE software is **published on CVMFS** repository → **Parrot** (CCTools) is used to access data without mounting any device on the container nor on the underlying host
- HTCondor **Shared Port Daemon** and **Condor Connection Broker (CCB)** → No need to change firewall configuration on Plancton Nodes

Volunteer container with HTCondor and ALICE software (2)



Plancton Host

- Summary
  - Applications fully "docked"
  - Extra container config not fitting in Docker image is injected at deploy time (e.g. secrets...)
  - Containers for isolation and known environment chosen with an eye on volunteer/ opportunistic computing: **can we use same design on a dedicated facility?**

Use case example - Results

- Built a small farm on few hosts for test&debug
  - Successfully added virtual nodes from Centro di Calcolo di Torino (the Connection Broker made it possible) ➙ larger test environment for development purposes
  - Plancton works reliably and does not interfere with user's activities

Use case example (2) - ALICE Grid site

- Idea: What if we configured Plancton on a dedicated cluster?
  - Trivial task: configure Plancton to use every available resources and never kill containers to free them
- ALICE Grid site based on Plancton
  - Using AliEn: the ALICE Grid middleware
  - A VOBOX has been configured for tests
  - Using Work Queue for job submission: http://ccl.cse.nd.edu/software/manuals/workqueue.html
  - AliEn Work Queue backend: https://github.com/dberzano/alien-workqueue

Use case example (2) - ALICE Grid site / 2

- Plancton + AliEn + Work Queue integration
  - Containers run **work_queue_worker**
    - It connects back to the Work Queue master (the VOBOX) and fetches jobs
    - One job per container: work_queue_worker exits when job is done
    - Container **plancton/slc6-wq** on Docker Hub
  - Parrot used for CVMFS provisioning
    - Configured with "alien cache" (shared cache that survives containers)
  - AliEn + Work Queue configuration proven to scale to over 10k jobs on opportunistic ALICE HLT cluster
    - Currently uses VMs and OpenStack but Plancton + containers are envisioned for the future

Use case example (2) - ALICE Grid site / 3

- Rolling updates: using Plancton and containers for zero-downtime continuous upgrades
  - Plancton is stateless: can be upgraded without killing containers
  - Plancton always picks the latest version ("force pull") of the specified Docker container, if available
- How to update an existing container image
  - Edit image and `docker push` it
  - All new pilot containers will pick it: Docker takes care of deploying new images and keep existing containers run with the previous version at the same time

Conclusions

- Docker containers as pilot worker nodes are a viable way to deploy consistent environments for running jobs (*e.g.* SLC6 containers running on Ubuntu 16.04…)
- Resource capping and isolation come natural with Docker
- Possible to manage a Grid site with seamless continuous upgrades, much more easily than with Virtual Machines

Outlook

- To test the generality of the tool we are working on porting a **CMS use case** (a simple simulation provided by CMS Torino group, as a proof of concept, complexity comes later)
- Most of this work is close to some activities within the **INDIGO-DataCloud** project; we are in touch with the Torino people working in the project to explore possible collaborations