IoT: Cloud Services

Orchestration & Kubernetes

Container Orchestration

- Orchestrate containers to facilitate microservices architecture
- Provision containers
- Manage container dependencies
- Enable discovery
- Handle container failure
- Scale Containers

Why one to choose?

Kubernetes

Kubernetes is based on Google's and Red Hat's experience of many years working with Linux containers.

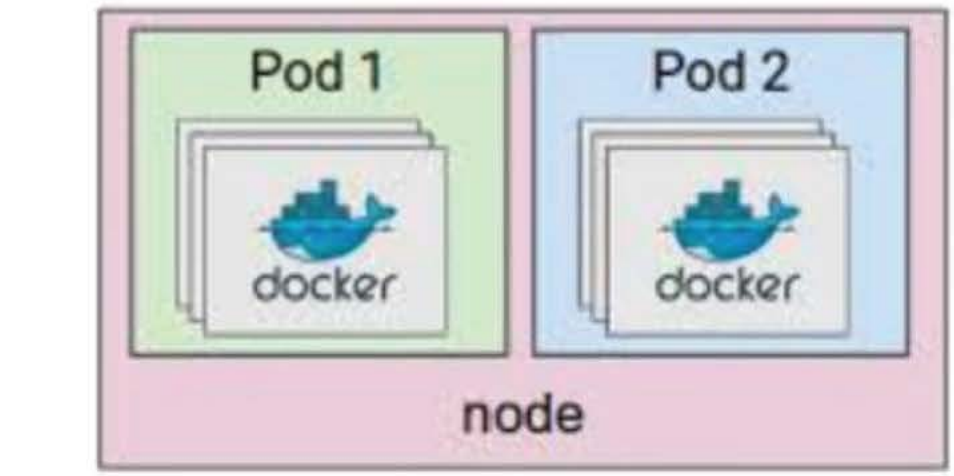
Some abilities:

- Mount persistent volumes that allows to move containers without loosing data,
- it has load balancer integrated
- it uses `Etcd` for service discovery.

Kubernetes uses a different CLI, different API and different YAML definitions.

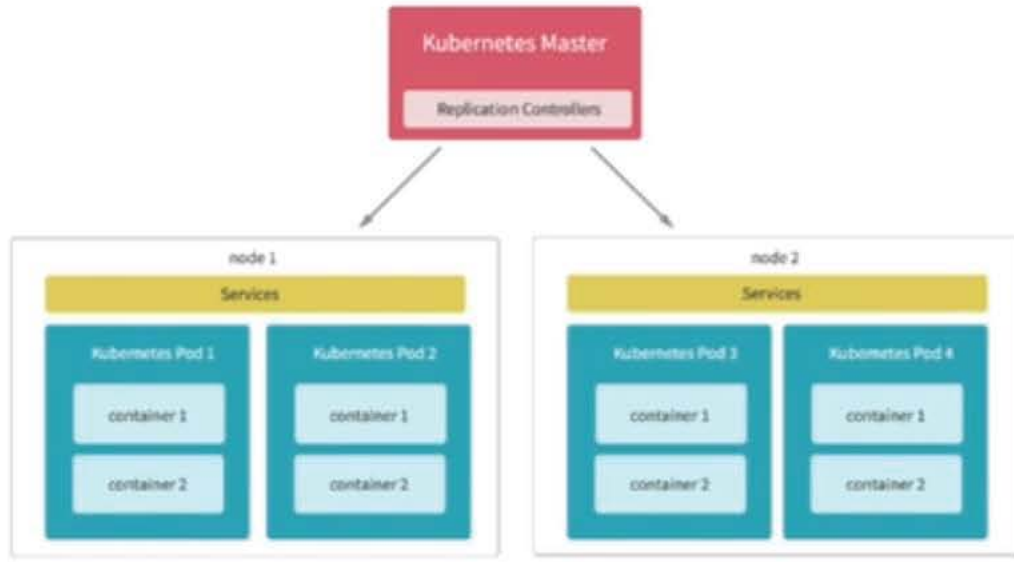
- cannot use Docker CLI

Pods



- ***Pods***:
  - Pods are groups of containers that are deployed and scheduled together.
  - A pod will typically include 1 to 5 containers which work together to provide a service.
  - Kubernetes will run other containers to provide logging and monitoring services.
  - Pods are treated as ephemeral in Kubernetes;

Kubernetes: Services & Replication Controllers



- ***Services***
  - Services are stable endpoints that can be addressed by name.
  - Services can be connected to pods by using label selectors; for example my “DB” service may connect to several “C*” pods identified by the label selector “type”: “Cassandra”.
  - The service will automatically round-robin requests between the pods.

- **Replication Controllers**
  - Replication controllers are used to instantiate pods in Kubernetes
  - They control and monitor the number of running pods (called replicas) for a service.

Kubernetes: Net space and Labels

- ***Flat Networking Space***:
  - Containers within a pod share an IP address, but the address space is “flat” across all pods,
    - all pods can talk to each other without any Network Address Translation (NAT).
  - This makes multi-host clusters much more easy to manage, at the cost of not supporting links and making single host (or, more accurately, single pod) networking a little more tricky.
  - As containers in the same pod share an IP, they can communicate by using ports on the localhost address.
- ***Labels***:
  - Labels are key-value pairs attached to objects in Kubernetes, primarily pods, used to describe identifying characteristics of the object
    - e.g. version: dev and tier: frontend.
  - Labels are not normally unique; they are expected to identify groups of containers.
  - Label selectors can then be used to identify objects or groups of objects, for example all the pods in the frontend tier with environment set to production.

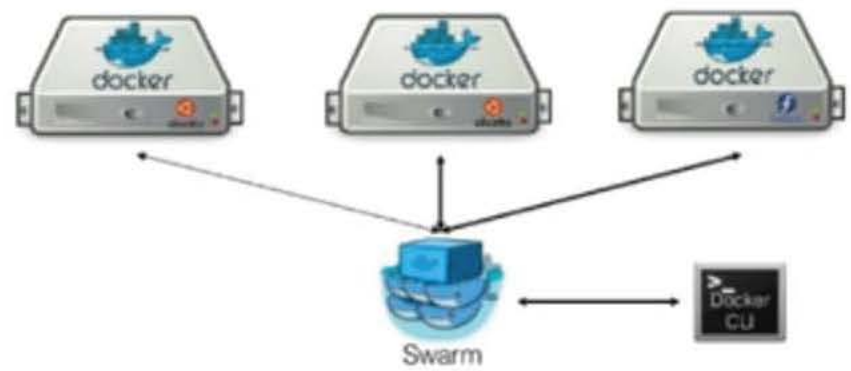
IoT: Cloud Services

Orchestration (next)

Docker Swarm

- A native clustering for Docker.
- Exposes standard Docker API
  - meaning that any tool that you used to communicate with Docker (Docker CLI, Docker Compose, Dokku, Krane, and so on) can work equally well with Docker Swarm.
- Bound by the limitations of Docker API

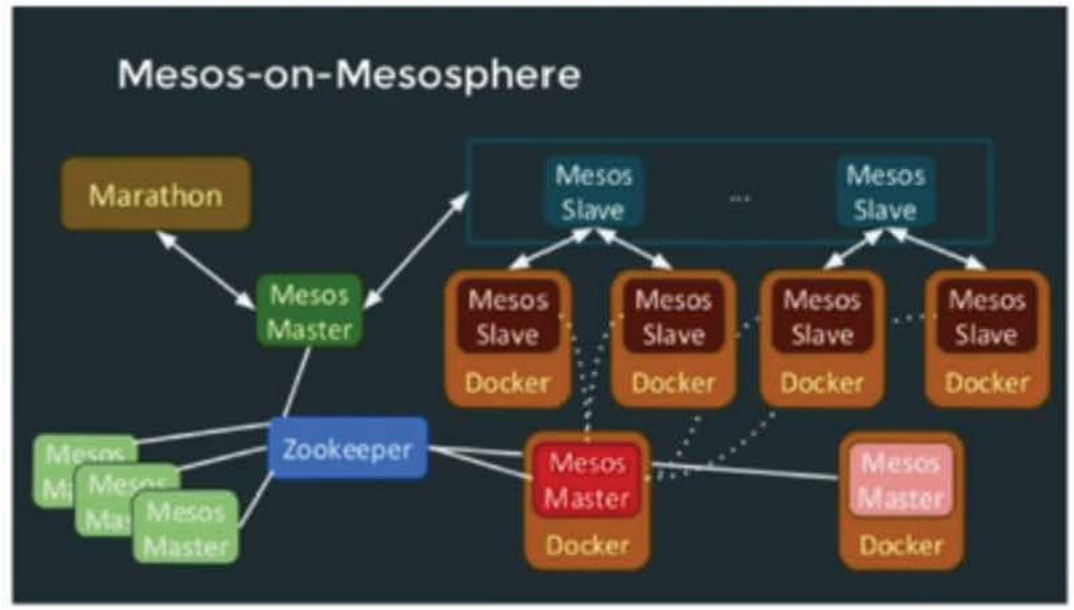
With Docker Swarm



Docker Swarm: Architecture

- Architecture:
  - each host runs a Swarm ***agent*** and one host runs a Swarm ***manager*** (on small test clusters this host may also run an agent).
  - The manager is responsible for the orchestration and scheduling of containers on the hosts.
  - Swarm can be run in a high-availability mode where one of etcd, Consul or ZooKeeper is used to handle fail-over to a back-up manager.
  - There are several different methods for how hosts are found and added to a cluster, which is known as *discovery* in Swarm.
    - By default, *token* based discovery is used, where the addresses of hosts are kept in a list stored on the Docker Hub.

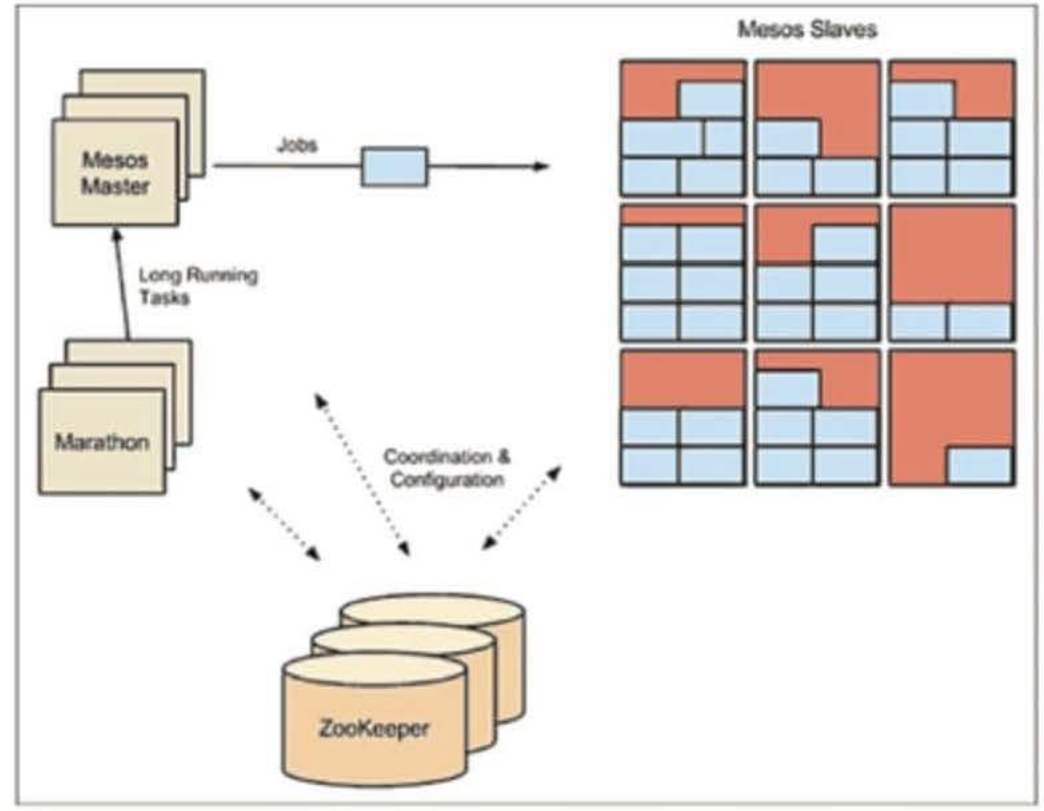
Mesos



- Apache Mesos (https://mesos.apache.org) is an open-source cluster manager. It's designed to scale to very large clusters involving hundreds or thousands of hosts.
  - Mesos supports diverse workloads from multiple tenants; one user's Docker containers may be running next to another user's Hadoop tasks.
- Apache Mesos was started as a project at the University of Berkeley before becoming the underlying infrastructure used to power Twitter and an important tool at many major companies such as eBay and Airbnb.
- With Mesos, we can run many frameworks simultaneously:
  - Marathon and Chronos are the most well known

Mesos Architecture

- ***Mesos Agent Nodes*** – Responsible for actually running tasks. All agents submit a list of their available resources to the master.
  - There will typically be 10s to 1000s of agent nodes.
- ***Mesos Master*** – The master is responsible for sending tasks to the agents.
  - maintains a list of available resources and makes "offers" of them to frameworks.
  - decides how many resources to offer based on an allocation strategy. There will typically be 2 or 4 stand-by masters ready to take over in case of a failure.
- ***ZooKeeper*** – Used in elections and for looking up address of current master.
  - Typically 3 or 5 ZooKeeper instances will be running to ensure availability and handle failures.



Mesos with Marathon: Architecture

- ***Frameworks*** – Frameworks co-ordinate with the master to schedule tasks onto agent nodes. Frameworks are composed of:
  - *executor* process which runs on the agents and takes care of running the tasks
  - *scheduler* which registers with the master and selects which resources to use based on offers from the master.
  - There may be multiple frameworks running on a Mesos cluster for different kinds of task. Users wishing to submit jobs interact with frameworks rather than directly with Mesos.
- **Marathon** is designed to start, monitor and scale long-running applications.
  - Marathon is designed to be flexible about the applications it launches,
    - It can even be used to start other complementary frameworks such Chronos ("cron" for the datacenter).
  - It makes a good choice of framework for running Docker containers, which are directly supported in Marathon.
  - Marathon supports various affinity and constraint rules.
  - Clients interact with Marathon through a REST API.
  - Other features include support for health checks and an event stream that can be used to integrate with load-balancers or for analyzing metrics.

IoT: Cloud Services

Container Conclusion

Containers Pros

Containers are immutable

The OS, library versions, configurations, folders, and application are all wrapped inside the container.

Guarantee that the same image that was tested in QA will reach the production environment with the same behavior.

Containers are lightweight

The memory footprint of a container is small.

Instead of hundreds or thousands of MBs, the container will only allocate the memory for the main process.

Containers are fast

Can start a container as fast as a typical Linux process takes to start.

Instead of minutes, you can start a new container in few seconds.

However, many users are still treating containers just like typical virtual machines and forget that containers have an important characteristic: **Containers are disposable**.

Things to avoid in Docker containers

- Do not store persistent data in containers:
  - containers can be stopped, destroyed, replaced.
- Do not ship the app in pieces:
- Do not create large images
  - Large image is harder to distribute
  - Avoid “updates” (e.g. `yum update`) that downloads may files to a new image
- Do not use single layer image
  - To make an effective layered file system, always create your base image layer for your OS, another layer for the username definitions, another layer for the runtime installation, and another layer for the configuration, and another layer for the app.
  - Makes it easier to recreate, manage and distribute the image
- Do not create images from running containers
  - E.g. do not use “`docker commit`” to create an image. This method is not reproducible

References and Acknowledgements

- Some images have been borrowed from: : http://scm.zoomquiet.io/data/20131004215734/index.html
- Docker Engine and Docker Hub: http://blog.docker.com/category/registry-2/
- Microservices: http://martinfowler.com/articles/microservices.html
- Swarm vs Fleet vs Kubernetes vs Mesos: https://www.oreilly.com/ideas/swarm-v-fleet-v-kubernetes-v-mesos
- 10 things to avoid in docker containers: http://developers.redhat.com/blog/2016/02/24/10-things-to-avoid-in-docker-containers/
- There are more public information that have been used…