Container orchestration and Kubernetes

COSC349—Cloud Computing Architecture

David Eyers

Learning objectives

- Explain required **container orchestration** functionality
  - Kubernetes is the dominant tool, so a good point of reference
- Describe etcd's role in **managing container clusters**
  - Its history within Container Linux (was CoreOS) gives context
- Appreciate **rapid change** in features of cloud tools
  - Also that tool functionality may partially or completely overlap

Container orchestration

- Containers are useful, but need to be managed
- Numerous container orchestration systems emerged:
  - **Docker swarm**—(now) built-in simple Docker cluster manager
  - **Docker compose**—means to specify multi-container “stacks”
  - **Kubernetes**—focus of today’s lecture…
  - **Apache Mesos**—also supports non-containerised workloads
  - **OpenShift**—as discussed previously
- Google has used containers in production for years
- First though, we examine an OS to host containers…

RHEL / Fedora CoreOS

- Previously Container Linux, previously previously CoreOS
- **Linux distribution** intended only to run containers
  - No software package manager: `/usr` **is read-only**
  - Can be started using **network boot**
  - **Security updates** are applied monolithically
    - Can schedule rolling reboot of cluster machines
- Originally ran Docker containers, subsequently rkt, etc.
  - Previously used as a Docker layer in CS lab environment
- Can't run container host cluster without coordination
  - … so developed and provides etcd for coordination

Kubernetes

- Project emerged from Google in 2014; v1.0 released 2015
  - Related to Borg—Google's (secret) internal container scheduler
    - K8s is implemented in Go, in contrast to Borg's C++
- A number of key types of objects
  - **Pods**—tightly coupled set of containers; smallest unit of scheduling
  - **Services**—set of pods grouped behind load balancer
  - **Volumes**—persistent storage; can share between containers
  - **Namespaces**—*e.g.*, same device names in dev, test & production
  - **ConfigMaps** and **Secrets**—runtime configuration parameters
- Kubernetes works with many container technologies

Kubernetes pods

- Pods are the basic unit of **application execution**
  - Common case is to have one container in a pod
  - Multiple containers in a pod tightly couple them
    - Should be used when those containers share local resources
- Pods are assigned an **IP address**, for networking
  - All containers within the pod share that address and its ports
- Pods provide **app. storage** (volumes) to containers
- Pods usually created by controllers, and not directly
  - *e.g.*, controller types: Deployment, StatefulSet, DaemonSet

Stateless versus stateful applications

- **Stateless applications scale easily**: just start more pods
  - *e.g.*, web servers presenting read-only workloads
- **Stateful apps are more difficult**, *e.g.*:
  - Databases having primary and secondary instances
  - Distributed components that spread state over instances
- **Kubernetes controllers** select stateless / stateful
  - *e.g.*, storage is handled differently for stateful applications
    - volume can be unique for a given instance of a pod in a set
    - otherwise volumes are shared across all instances of pods in a set

Architecture of Kubernetes

- Master node is logically centralised control point
  - **API server**—allows Kubernetes cluster to be controlled
  - **controller manager**—checks replication; nodes are up
  - **scheduler**—allocates pods waiting to run to nodes
  - **etcd**—consistent repository of configuration information
- Kubernetes Nodes run pods, but also:
  - **Kube-Proxy**—provides network services; leveraging OS facilities
  - **cAdvisor**—provides statistics about container resource use
  - **Kubelet**—checks on health of containers within a pod

Kubernetes Scheduler

- Scheduling is a **multi-factor optimisation** problem
  - Tradeoff between global (slow) & local (may not be optimal)
- K8s Scheduler is not global; uses multiple phases:
  - **P1**: find nodes that can run pods without resourcing violations
  - **P2**: score which plan appears to be best, choose best score
- Will try to place pods on nodes with available space…
  - … otherwise force pods onto nodes & kill some existing pods
  - **Killed pods may be replicas** not currently being used much

etcd—consistent, distributed key-value DB

- etcd was developed to support **CoreOS coordination**:
  - needed to reliably do rolling OS reboots without breaking apps
- Similar objective to Apache ZooKeeper, but…
  - … uses the **Raft consensus algorithm** instead of Paxos
    - inspired by Google Chubby—an internal database project
  - Implemented in Go instead of Java, with API using HTTP+JSON
  - Can operate with a **smaller resource investment** than ZKs
- Allows registration of watches on keys (and directories)
  - Build applications that trigger reconfiguration on DB changes

Kubernetes as a Service

- K8s can manage your containers, but how to set it up?
  - IaaS needs for the **VMs running master and the nodes**
- Amazon offer a range of options:
  - **AWS Fargate** provides a complete container service
  - **AWS EKS** provides control plane; you set up K8s nodes on EC2
  - **Use EC2** to deploy all of the components for full control
- Cloud providers' container services are very similar
  - Can deploy containers onto clusters in multiple clouds
  - Tools like Rancher provide for **multi-K8s cluster management**

Terraform versus Rancher, Kubernetes, *etc.*

- Rancher can help **deploy Kubernetes over bare metal**
  - Rancher also unifies monitoring & security management tools
- However, you **may need specific infrastructure** nodes
  - *e.g.*, configuring a **GPGPU node** on Amazon for deep learning
  - Containers can use this hardware, but container managers?
- **Terraform** is a level below tools like Rancher, it:
  - can **effect deep IoC impacts**—allows preview of its plans
  - can thus easily provision at level of particular GPU instance
    - … then pass control of software to a container manager

Why it's so hard to pick a "winner"

- Tools can **manage each other**—it's all just software!
- All are **churning rapidly** in what's provided, e.g.:
  - Rancher's original functionality replaced by Docker Swarm
  - CoreOS Linux's original fleet functionality replaced by K8s
- Which to use? Consider your and **your team's time**
  - Look to see whether a new tool can optimise your processes
  - ... but only when taking into account the cost of transition
- Aim to have IoC and continuous integration pipelines
  - All future tooling is likely to **move in the direction of IoC**