University of BRISTOL

Container Orchestration

Dr Dan Schien – COMSM0010

Lecture 8

bristol.ac.uk

Admin

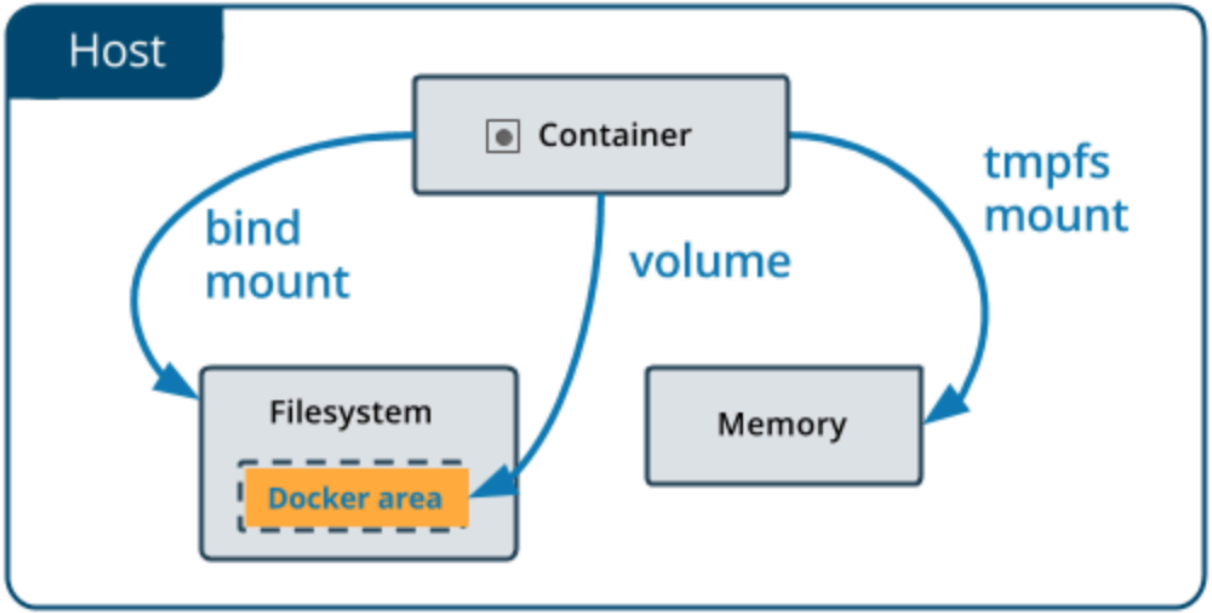
Goals

- Be familiar with the main concepts underlying Kubernetes

Backflash – Containers

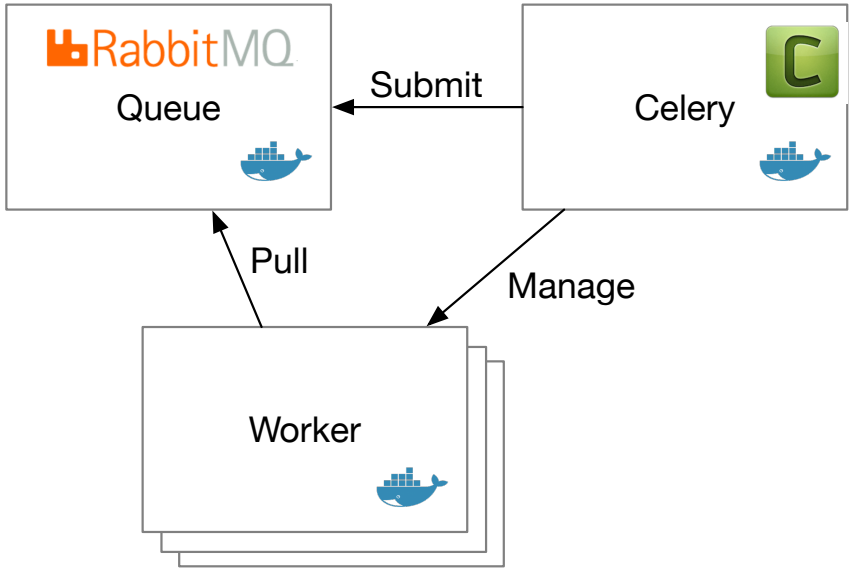
- Reproducability, Portability, Flexibility, Isolation
- System consisting of
  - Linux containers
  - Container runtimes
  - Container images
  - Container storage
  - Container registries
  - Container engines
  - Container image builders

Docker Volumes



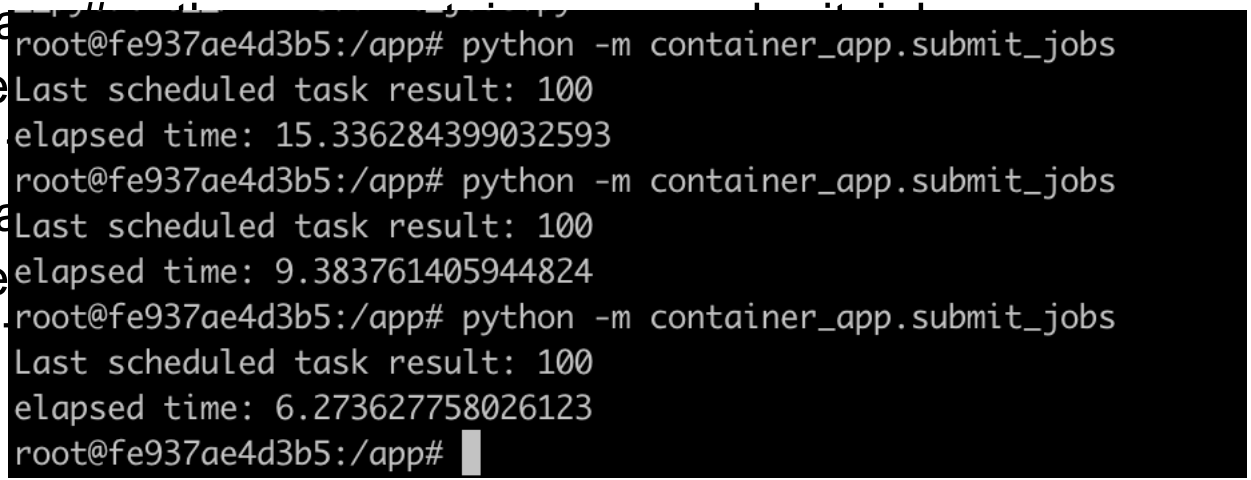
Demo

- Simple application with multiple workers that speed up work
- https://github.com/dschien/coms0010_docker_basic



- docker run --rm --name rabbit --env RABBITMQ_DEFAULT_USER=admin --env RABBITMQ_DEFAULT_PASS=mypass rabbitmq:latest
- docker run --link rabbit -v $(pwd):/app worker
- docker exec -i -t XXX /bin/bash

root@fe937ae4d3b5:/app# python -m container_app.submit_jobs
Last scheduled task result: 100
elapsed time: 15.336284399032593
root@fe937ae4d3b5:/a
Last scheduled task re
elapsed time: 9.38376
root@fe937ae4d3b5:/a
Last scheduled task re
elapsed time: 6.27362



Orchestration

- Running Containers at scale requires management tools
- Manage networking, volumes, infrastructure
- Automate
  - Fault tolerance, self-healing
  - Auto-scaling on demand
  - DevOps
  - Update/rollback without downtime
- Mesos, Docker Swarm, Kubernetes

Kubernetes History

kubernetes

- Origins as a Google project mid-2014 (Google “Borg”)
- 1.0 release in July 2015
- Google with the Linux Foundation formed the Cloud Native Computing Foundation (CNCF) and handed over Kubernetes
- Now part of the Cloud Native Computing Foundation project
- Abbreviated as “k8s”, Greek for “helmsman” or “pilot”

Features

- Automatic scheduling of work based on resource usage and constraints
- Self-healing: automatic replacement and rescheduling of failed containers
- Service discovery and Load balancing
- Automated rollouts and rollbacks

Kubernetes Runtime Objects

- Pods
- Deployments
- Services

Pod

- Set of one or more containers that act as a unit and are scheduled onto a node together
- Share a local network and can share file-system volumes

Deployments

- Describe “desired state” through declarative updates for pods and ReplicaSets
- Encapsulates ReplicaSets
  - Balances the number of scheduled and running pods (kills and creates)
- Managed via
  - Spec: describes desired state (What)
  - Monitors status = current state
  - Template (how)
- What you can do
  - Create a Deployment to rollout a ReplicaSet
  - Declare the new state of the Pods
  - Rollback to an earlier Deployment revision
  - Scale up the Deployment to facilitate more load

A Deployment Yaml File

Run with:
`> kubectl create deployment.yml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - name: web-pod-port
        - containerPort: 80
```

Label

Implicit ReplicaSet

selector

PodTemplateSpec

bristol.ac.uk

Services

- Defines networking to access pods consistently
- Expose pods to external world
- Create groupings of pods that can be referred to by name
- Unique IP address and a DNS hostname (by default cluster scope only)
- Pods in Service are load balanced
- Environment variables containing the IP address of each service in the cluster are injected into all containers

Service config file

Run with:
```
> kubectl create service.yml
```

```
kind: Service
apiVersion: v1
metadata:
  name: web-app
spec:
  selector:
    app: web
  ports:
  - protocol: TCP
    port: 80
    targetPort: web-pod-port
```

Named port on target pods

ServiceTypes

- Influences networking configuration
- ClusterIP
  - Default
  - Service is discoverable/routable only within the cluster
  - kube-proxy watches API service and updates pod IPTables on change
- NodePort
  - Exposes the service on each Node's IP at a static port (the NodePort)
  - Access the service via <NodeIP>:<NodePort>
  - The simplest way to make your service externally accessible
  - Alternatively, use better **Ingress** (currently in beta)
- More at https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services---service-types

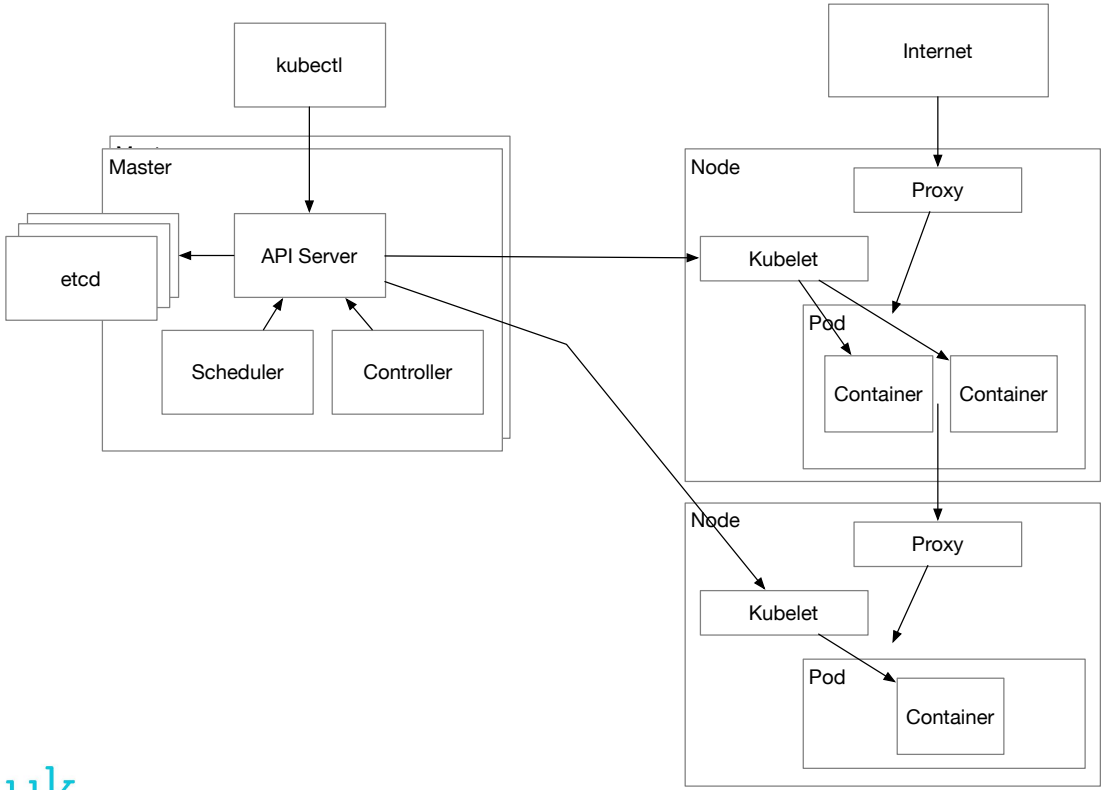
Kubernetes Components

- Master
- Etcd
- Nodes
  - Pods
  - Kubelet
  - Kube Proxy
- Kubectl
  - local cli to manage cluster
  - configured to know cluster target for commands

Nodes

- Run work in pods
- Pods are the scheduling unit
  - Contain one or more containers
  - scheduled together on the same host
  - Mount the same external storage (Volumes)
- Container Runtime
- Kubelet
  - Agent that communicates with Master
- kube-proxy
  - Network agent
  - Manages overlay network routes

Components Architecture



Deployment:
Pod1:
- C ‘A’
- C ‘B’
Replicas: 3
Pod2:
- C ‘C’
Replicas: 2

kubectl

API Server

etcd

Scheduler

Controller

Worker Node

Kubelet

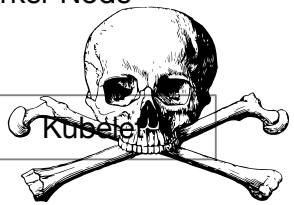
P1R1

P2R1

P1R2

Worker Node

Kubelet



Worker Node

Kubelet

P1R3

P2R2

Master

- Manages the cluster state
- Subcomponents
  - API Server
    - Gateway from outside
  - Controller
    - regulates the state from current to desired state
  - Scheduler
    - schedules pods to worker nodes, taking into account constraints
- Possibly redundant
- Writes to etcd
  - Distributed reliable key-value store
  - https://github.com/coreos/etcd

Review

- Kubernetes is Container Orchestration
- Automates
  - Rollout
  - Self-healing
  - Scaling
  - Service discovery
- Master, Worker, kubectl
- Pods, Deployments, Services

Next Week

- Demo