Container orchestration

Antoine Vander Meiren & Charles Vandevoorde

June 10, 2018

ECAM Brussels

Slideshow layout

Orchestration introduction

Orchestration solutions

Kubernetes example

Orchestration introduction

Deployement: The old school way

Deployement: The old school way

1. Manual
2. Scripting
3. Infrastructure automation

IT trends

Micro-services introduces complexity in deployement

Infrastructure as Code (IaC) provides a new paradigm to manage servers

Containers simplify applications packaging

What’s an orchestrator?

- Facilitates complex containers deployement
- Abstracts services mapping on servers
- Container’s management (monitoring, restarting, killing, …)

Why orchestrating?

- Host provisioning
- Containers instantiation
- Failed containers rescheduling
- Containers external interfaces configuration
- Scaling

Orchestration solutions

Existing solutions

- Docker Swarm
- Kurbenetes
- Others

Points of comparaison:

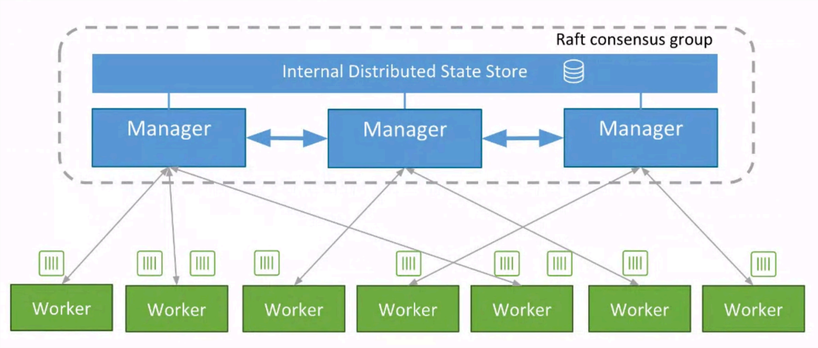
- Added value
- Inconvenient
- Use-cases
- Components

Docker Swarm

- Docker's orchestrator
- Simple to configure
- Well integrated with other Docker solutions
- Use-cases

  Small deployments or experiments

Docker Swarm: Components



Docker Swarm: Components

Manager nodes distributes tasks across the cluster

Worker nodes run Docker images

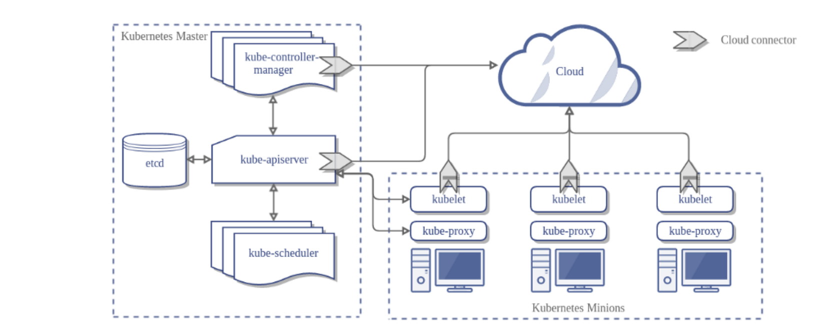
Key-value store keeps configurations and some states distributed across the cluster

Kubernetes

- Google open source solution based on *Borg*
- Highly modulable (network stack, container runtime, ...)
- High learning curve
- Use-cases

  Microservices, medium to large clusters

Kubernetes: Components



Kubernetes: Master's components

API server exposes a front-end to control the Kubernetes cluster

Controllers evaluate and propagate changes

Scheduler dispatchs pods on nodes based on the policy in place

Key-value store (etcd) keeps configurations and some states distributed across the cluster

Kubernetes: Nodes' components

Kubelet controls groups of containers one node

Proxy is a network abstraction

Related work

- **Apache Mesos** is a *distributed systems kernel*
- **Apache Marathon** is an orchestrator built on Apache Mesos
- **Nomad** is another orchestrator built for simplicity
- **OpenShift** is built on top of Kubernetes to provide a self-hosted PaaS container platform
- **OpenStack** is an abstraction of machines providing a self-hosted IaaS

Functionalities

Usual features of an orchestrator:

- **Automatic binpacking**
  Automatically places containers depending resources requirements
- **Self-healing**
  Kill and restarts failed containers
- **Horizontal scaling**
  Scale application based on CPU usage
- **Service discovery**
  Automatically detects devices and services on the fly
- **Load balancing**
  Distributes work among replicas

Functionalities

- **Automated rollouts and rollbacks**
  Automatically undoes failed configuration changes
- **Secret and configuration management**
  Avoids exposing secrets in stack configuration
- **Storage orchestration**
  Abstracts storage (Cloud, Local, Network, ...)
- **Batch execution**
  Manages batch and CI workloads

Summary

Docker Swarm For really small clusters.
Compose well with other Docker tools

Apache Mesos For complex or custom requirements

Kubernetes Default option. Considered as *industry standard*

Kubernetes example

Kubernetes example

Configuration file to:

- Deploy a Redis master store
- Expose a service

Kubernetes example: Service

```
apiVersion: v1 # version of kubernetes API
kind: Service # type of object to define (Service, Pod, ...)
metadata: # name, label or other
  name: redis-master
  labels:
    app: redis
    tier: backend
    role: master
spec: # define the behavior of a service
  ports:
  - port: 6379 # ports exposed by this service
    targetPort: 6379
  selector: # Map a pod with these tags
    app: redis
    tier: backend
    role: master
```

Kubernetes example: Deployment

```
apiVersion: v1
kind: Deployment # pods constructor handler
metadata:
  name: redis-master
spec: # specification of deployment's behavior
  selector: # select target service's labels
    matchLabels:
      app: redis
      role: master
      tier: backend
  replicas: 1 # number of pods to create
  template: # define the pod(s)
    ...
```

Kubernetes example: Deployment

```yaml
metadata:
  labels:
    app: redis
    role: master
    tier: backend
spec:
  containers:
  - name: master
    image: k8s.gcr.io/redis:e2e  # container's image
    resources: # define container's resources
      requests:
        cpu: 100m
        memory: 100Mi
    ports: # port to expose container
    - containerPort: 6379
```

Service utilization

```go
package main

import "github.com/xyproto/simpleredis"

func main() {
    masterPool := simpleredis
        .NewConnectionPoolHost("redis-master:6379")
    defer masterPool.Close()

    // Use the redis db.
}
```