Kubernetes for Newbies

Greg Horie

Overview

- Linux Containers
  - Quick intro / recap
- Kubernetes - What is it?
- Kubernetes Architecture
- Live Demo
  - Kubernetes Pod
  - Kubernetes Deployment
  - Kubernetes Service
- Summary
- Future Topics

Linux Containers - Quick Intro / Recap

- A lightweight method for running multiple applications under one Linux host
- **Feature - Portability**
  - Each container encapsulates its dependencies
  - Ensures consistent behavior across environments
- **Feature - Efficiency**
  - Containers share the host's kernel
  - Lower overhead compared to a virtual machine
- **Feature - Scalability**
  - Containers can be spun up or shut down quickly
  - Enables rapid deployment and scaling

Containers - The Challenges

- Container runtime runs on a single host - great for dev, but not prod
- Making containers production-worthy requires more
- Challenges to consider:
  - Redundancy - i.e. multiple hosts in case of failure
  - Networking across redundant hosts
  - Shared file systems, configurations, secrets
  - Load balancing
  - Scheduling workloads
  - etc.

Kubernetes - What is it?

- K8S = Kubernetes
- Kubernetes is Greek for helmsman or pilot
  - Following the container / Docker shipping metaphor
- K8S is an open-source container orchestration platform
- Automates the deployment, scaling, and management of containerized applications across a set of hosts (nodes)

kubernetes

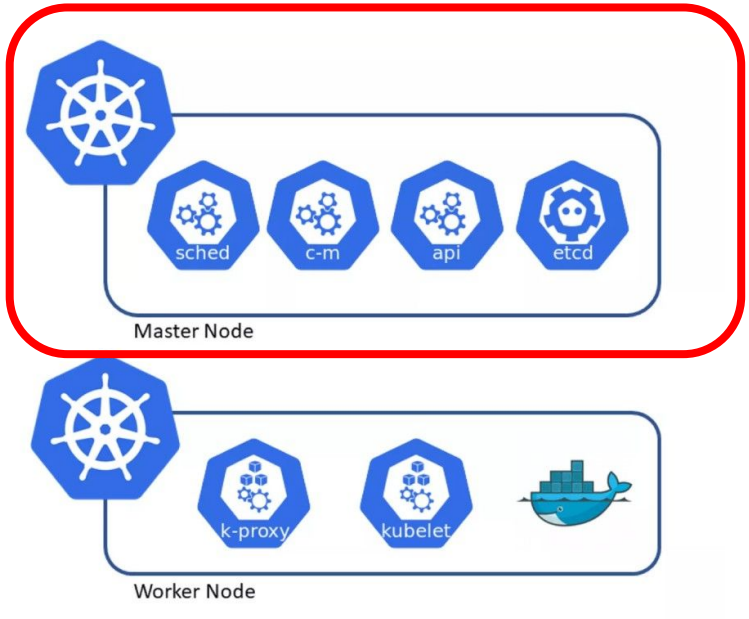
Kubernetes - History

- **2003-2004 - Borg**
  - Early orchestration platform built at Google to manage container-based applications
- **2014 - Kubernetes open-sourced**
  - Google engineers open sourced K8S (based on Borg)
    - Written in the Go programming language
- **2015 - Kubernetes 1.0 released**
  - Google partners with the Linux Foundation
    - Forms the Cloud Native Computing Foundation (CNCF)
  - CNCF goes on to host many open source projects - containerd, prometheus, etcd, etc.
- **2017 - Winner of the container wars**
  - Industry rallies around K8S - Docker, Microsoft AKS, Amazon EKS, etc.
- **Today**
  - Continues to evolve with a strong community contributing to its future

Kubernetes - Architecture

Control plane (master) nodes

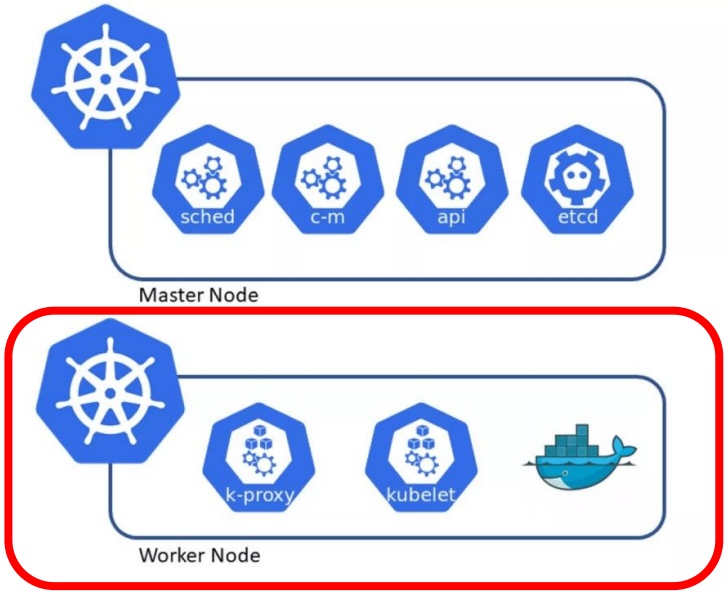
- **kube-apiserver** - Exposes Kubernetes API for cluster management.
- **kube-controller-manager** - Manages desired state via controllers.
- **kube-scheduler** - Assigns Pods to nodes based on resource requirements.
- **etcd** - Distributed key-value store for cluster data.
- … and many more depending on your environment.



Kubernetes - Architecture

Worker nodes

- **kubelet** - Agent running on workers, executing instructions from the control plane.
- **container runtime** - Software responsible for running containers (e.g. Docker).
- **kube-proxy** - Maintains network rules for load balancing inside the cluster.
- **CNI (Container Network Interface)** - Overlay network enabling Pod communications across nodes.



Minikube & Project Prep

Note - Assumes container runtime installed - e.g. Docker

Install minikube

```
$ curl -LO \
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
```

```
$ sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

Install kubectl

```
$ curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
```

```
$ sudo install kubectl /usr/local/bin/kubectl
```

GitHub Repo

```
$ git clone \
  https://github.com/netserf/netsig-presentation-kubernetes-for-newbies.git
```

Minikube Init

Start Cluster

```
$ minikube start --nodes 2 -p newbie-demo

$ minikube status -p newbie-demo

$ minikube ssh -p newbie-demo -n newbie-demo # control plane node
    $ ps -e | grep -E 'kube|etcd'

$ minikube ssh -p newbie-demo -n newbie-demo-m02 # worker node
    $ ps -e | grep -E 'kube|docker'
```

Nodes Provisioned

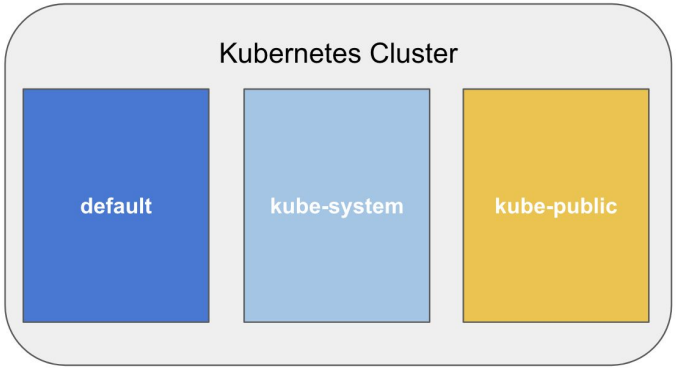
```
$ kubectl get nodes

$ kubectl describe nodes
```

Namespaces

- **Resource Partitioning** - Divides cluster resources into logical groups.
- **Isolation** - Securely share cluster with multiple teams.

Create a namespace:

```
$ kubectl get namespace

$ kubectl create namespace newbie-ns

$ kubectl describe namespace

$ kubectl config set-context \
  newbie-demo --namespace=newbie-ns
```



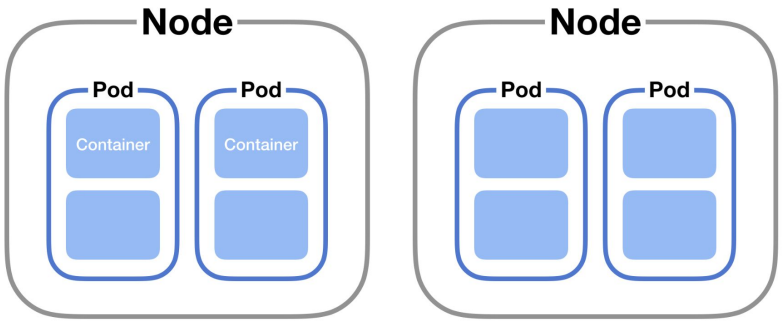
Namespace: kube-system

- **System Components** - Dedicated namespace for management components.
- **Critical Operations** - Hosts management components (schedulers, controllers, network plugins, etc.) essential for cluster operations.
- **Isolated** - Separates critical system components from user workloads.

```
$ kubectl get namespace kube-system

$ kubectl get pods --namespace=kube-system
```

K8S Pod

- **Basic Unit** -The smallest unit in the Kubernetes object model
  - Each Pod containers 1-to-many containers
- **Isolation** - Pod processes and resources are isolated from other Pods
  - Like a mini virtual machine
- **Shared Resources** - Containers within a Pod share the same IP addresses, ports, volumes, configs, etc.

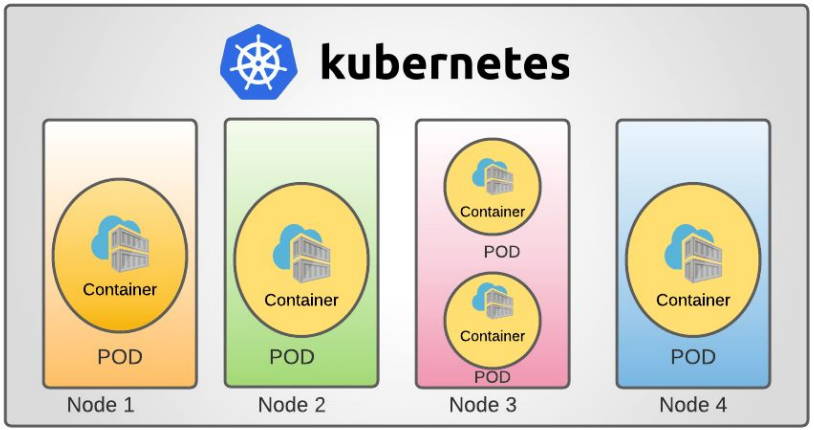


Run a K8S Pod

```
# Create a Pod imperatively

$ kubectl run nginx-pod --image=nginx:latest --restart=Never

$ kubectl delete pod nginx-pod

# Create a Pod declaratively

$ kubectl apply -f k8s/nginx-pod.yaml
```

Take a Look at the Pod

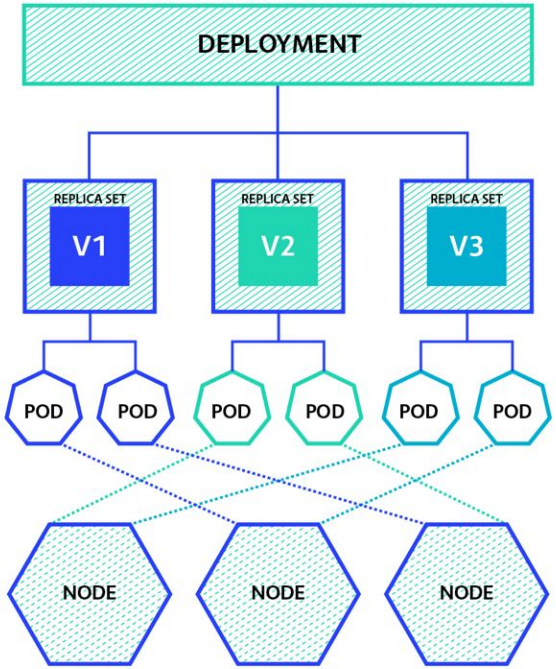
```
$ kubectl get pod nginx-pod [-o wide] [-o yaml]

$ kubectl describe pod nginx-pod

$ kubectl logs nginx-pod

$ kubectl exec -it nginx-pod -- /bin/bash
    $ curl <container-ipaddr>:80

$ kubectl port-forward nginx-pod 8080:80

$ kubectl delete pod nginx-pod
```



K8S Deployment

- **Scalability** - Scale applications up or down by adjusting replica counts.
- **Self-Healing** - Health checks and automatic replacement of unhealthy pods.
- **Rolling Updates** - Updates without downtime, with quick rollback options.

- … this is where K8S value starts to show.



Run a K8S Deployment

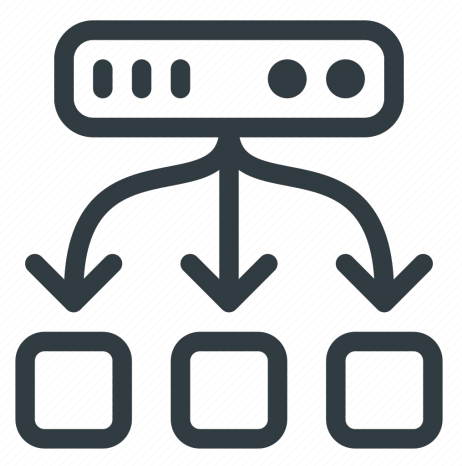
```
# Create a deployment imperatively

$ kubectl create deployment nginx-deployment --image=nginx:latest

$ kubectl scale deployment nginx-deployment --replicas=3

$ kubectl delete deployment nginx-deployment

# Create a deployment declaratively

$ kubectl apply -f k8s/nginx-deployment.yaml
```

Take a Look at the Deployment

```
$ kubectl get deployment nginx-deployment [-o wide] [-o yaml]

$ kubectl describe deployment nginx-deployment

$ kubectl get pods  # look at pods supporting this deployment

$ kubectl logs <pod-name> # use pod names discovered
```

K8S Service

- **Load Balancing** - Distributes incoming traffic among pods.
- **Stable Endpoint** - Provides a single, stable endpoint for communications.
- **Flexible** - Facilitates both internal cluster and external requests.

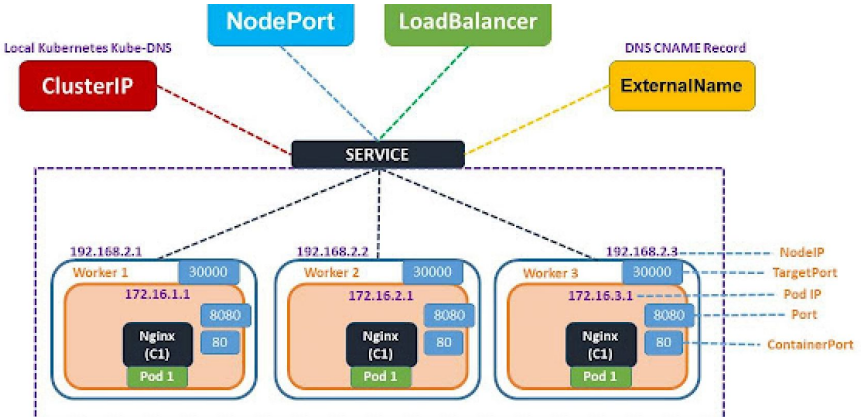


Run a K8S Service

```
# Expose the deployment imperatively

$ kubectl expose deployment nginx-deployment \
  --name=nginx-service --port=80 --type=NodePort

$ kubectl delete service nginx-service

# Expose the deployment declaratively

$ kubectl apply -f k8s/nginx-service.yaml
```

Take a Look at the Service

```
$ kubectl get service nginx-service

$ kubectl describe service nginx-service

$ minikube service nginx-service --url \
  -p newbie-demo -n newbie-ns
    $ curl <url>

$ kubectl logs <pod-name>
```



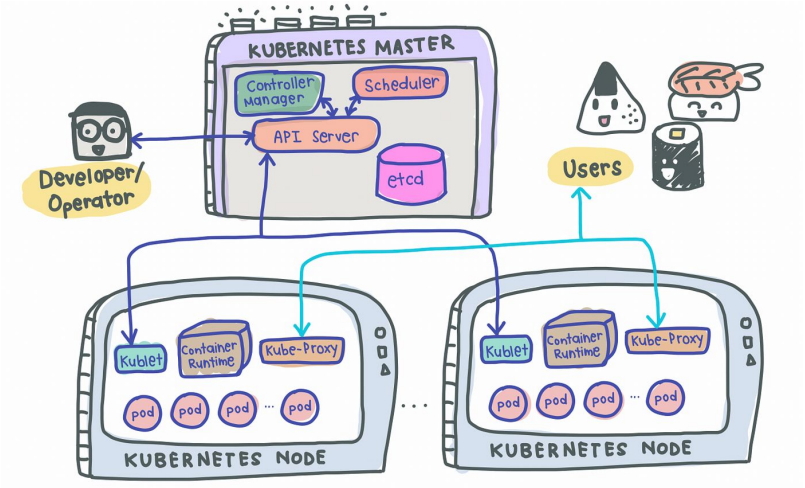
Clean-up

```
$ minikube delete -p newbie-demo
```

```
$ minikube status -p newbie-demo
```

Summary

- Kubernetes is a container orchestration platform
- Organizes machines (nodes) into clusters
- Streamlines the lifecycle management of containerized applications



Possible Future Discussions

- Orchestration
  - K8S ConfigMaps, Secrets, Persistent Volumes
  - K8S Ingress, Gateway
  - Istio Service Mesh
  - Hashicorp Nomad
- Observability
  - Prometheus / Grafana
  - ELK / EFK
  - Loki
- Messaging
  - RabbitMQ / ActiveMQ
- Data Pipelines
  - Airflow / Dagster
- Other ideas welcome!

Backup Slides

System Containers vs. Application Containers

System Containers:

- Running system-level processes and services
    - Like a mini virtual machine
- A lightweight environment for system-level tasks
- Designed to encapsulate and deploy components of the operating system
    - System daemons - systemd, cron, rsyslog, etc.
- No goal to decouple services any more than a traditional VM or bare-metal host
- Examples
    - LXC / LXD

System Containers vs. Application Containers

Application Containers:

- Designed to encapsulate individual applications and their dependencies
    - Fosters portability across different environments
    - Enables a microservices architecture
- Optimizes performance and scalability for the application itself
    - Avoids oversubscribing resources for more efficient use of host resources
- Examples
    - Docker
    - Containerd
    - Podman
    - CRI-O
    - Kubernetes