

Kubernetes for Newbies

Greg Horie

Overview

- Linux Containers
 - Quick intro / recap
 - System Containers vs. Application Containers
- Kubernetes - What is it?
- Kubernetes Pod
- Kubernetes Deployment
- Kubernetes Service
- Summary
- Future Topics

Linux Containers - Quick Intro / Recap

- A lightweight method for running multiple isolated applications under one Linux host
- Feature - Portable
 - Each container encapsulates its dependencies
 - Reduces conflicts and ensures consistent behavior across environments
- Feature - Efficient
 - Containers share the host's kernel
 - Results in lower overhead compared to traditional virtualization
- Feature - Scalable
 - Containers can be spun up or shut down quickly
 - Enables rapid deployment and scaling of applications
- Feature - Versioning / Rollback
 - Containers support versioning, packaging apps and dependencies together
 - Facilitates easy (automatable) rollbacks

System Containers vs. Application Containers

System Containers:

- Primarily focused on running system-level processes and services
- Serve as lightweight environments for system-level tasks
- Designed to encapsulate and deploy components of the operating system, including system daemons, background services, and other essential processes.
- Examples
 - LXC / LXK

System Containers vs. Application Containers

Application Containers:

- Geared towards encapsulating and deploying specific applications and their dependencies, fostering portability across different environments
- Primarily concerned with encapsulating application-specific resources, optimizing performance and scalability for the application itself
- Examples
 - Docker
 - Containerd
 - Podman
 - CRI-O
 - Kubernetes

Containers - Challenges

- Containers provide an isolated environment where an application can run
- To move these applications from isolated environments to production services requires some "extras"
- Challenges to consider:
 - Shared file systems, configurations, secrets
 - Networking
 - Load balancing
 - Scheduling - i.e. where to provision container workload
 - Distribution - i.e how to deploy the container workload
 - etc.

Kubernetes - What is it?

- K8s = Kubernetes
- K8s is an open-source container orchestration platform
- Automates the deployment, scaling, and management of containerized applications across a set of hosts (cluster)
- Supports load balancing, rolling updates, and application monitoring / scheduling

Kubernetes - History

- Kubernetes was originally developed by Google engineers
 - Written in the Go programming language
 - Based on an internal container orchestration platform called Borg
- Kubernetes was open-sourced by Google in 2014
 - Graduated as a project of the Cloud Native Computing Foundation (CNCF) in 2015
- Kubernetes gained widespread industry adoption
 - By 2017, it became the standard for container orchestration
- Continues to evolve with a strong community contributing to its development and improvement

Kubernetes - Architecture

Control Plane services:

- kube-apiserver: Exposes Kubernetes API for cluster management.
 - kube-controller-manager: Ensures desired cluster state via controllers.
 - kube-scheduler: Assigns pods to nodes based on resources.
 - etcd: Distributed key-value store for cluster data.
-
- ... and many more depending on your environment.

Kubernetes - Architecture

Worker nodes:

1. kubelet Agent running on each node, ensuring containers are running as specified by the Pod.
2. container runtime: Software responsible for running containers (e.g., Docker, containerd).
3. kube-proxy: Maintains network rules and enables communication across the cluster.
4. CNI (Container Network Interface): Plugin system enabling pod networking and communication across nodes.

Minikube Prep

Note - Assumes Docker installed (other options available)

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
```

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 plan
```

```
$ minikube ssh -p newbie-demo -n newbie-demo-m02 # worker
```

Nodes Provisioned

```
$ kubectl get nodes
```

```
$ kubectl describe nodes
```

Namespaces

- Resource Partitioning: Logically partitions cluster resources.
- Isolation: Enables secure sharing of a cluster among multiple users or teams.

Create a namespace:

```
$ kubectl create namespace newbie-ns
```

```
$ kubectl get namespace
```

```
$ kubectl describe namespace
```

Namespace: kube-system

- System Components: Dedicated namespace for essential system components and infrastructure services.
- Critical Operations: Hosts management components (schedulers, controllers, network plugins, etc.) essential for cluster operations.
- Isolation: Helps isolate critical system components from user workloads, enhancing security and maintainability.

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

```
$ kubectl describe namespace kube-system
```

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 inside the container
```

```
$ docker ps # check the containers
```

```
$ kubectl port-forward nginx-pod 8080:80
```


K8S Pod

- Basic Unit -The smallest and simplest unit in the Kubernetes object model.
- Groups Containers - Can host one or more tightly coupled containers
- Single Node - Each Pod scheduled to run on a single node in the cluster.
- Shared Resources - Containers within a Pod share the same IP and ports
 - Enables easy communication over the localhost
- Lifespan - Pods can have a short lifespan
 - Easily created, terminated, and replaced based on the application's requirements.

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 pod nginx-deployment [-o wide] [-o yaml]
```

```
$ kubectl describe pod nginx-deployment
```

```
$ kubectl get pods
```

```
$ kubectl logs ... # using pod names discovered
```

```
$ docker ps # check the containers
```

K8S Deployment

- Scalability -: Scale applications up or down by adjusting replica counts.
 - Automated Load Balancing - Built-in load balancing for even distribution of traffic.
 - Self-healing - Health checks and automatic replacement of unhealthy pods for high reliability.
 - Rolling Updates - Updates without downtime, with quick rollback options.
-
- ... this is where K8S value starts to show

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
```

```
kubectl get endpoints nginx-service
```

```
NODE_IP=$(kubectl get nodes -o  
jsonpath='{.items[0].status.addresses[0].address}')
```

```
NODE_PORT=$(kubectl get service nginx-service -o  
jsonpath='{.spec.ports[0].nodePort}')
```

```
curl $NODE_IP:$NODE_PORT
```

```
kubectl logs <pod-name>
```

K8S Service

- Load Balancing - Efficiently distributes incoming traffic among pods.
- Service Discovery - Provides a stable endpoint for communicating with pods.
- Dynamic Management - Organizes and manages pods dynamically using labels for scalability and updates.
- Communication Flexibility - Facilitates both internal cluster communication and external requests.

Clean-up

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

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


Summary

Possible Future Discussions

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



Backup Slides



Exercise 1 - ...

Setup:

\$

Try:

\$

