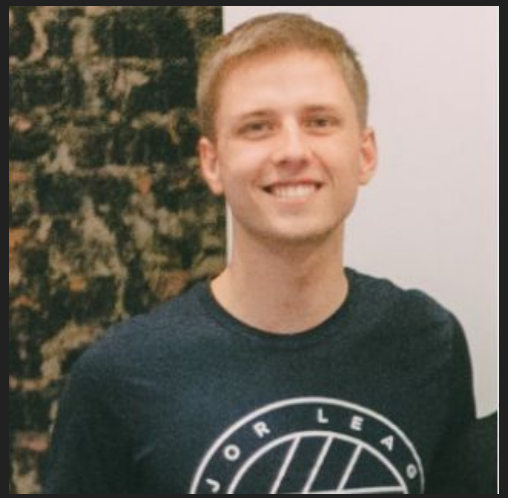
Introduction to Kubernetes Part 2

GHW Cloud Week - March 2025

Welcome!

I’m Stephen, your MLH Coach.



- Been hacking for almost ten years now!
- Super excited for GHW Cloud Week

Don’t forget to check in…
https://events.mlh.io/events/12336

First, a quick poll for today

Which advanced topic are you most excited to learn about today?

1. Advanced Networking
2. Persistent Storage
3. Stateful Applications

Let’s quickly review some key takeaways from Part 1…

- We learned about the core Kubernetes components—Pods, Nodes, Deployments, ReplicaSets, and Services.
- We created and modified Deployments and Services using YAML files.
- We practiced scaling, rolling updates, health checks, and even worked with ConfigMaps to inject configuration into our Pods.

Detailing the Networking Components

Now, we'll explore advanced networking concepts in Kubernetes. We're going to focus on Services, Ingress, and load balancing:

- ClusterIP: The default Service type that exposes the service on an internal IP.
- NodePort: Opens a specific port on all nodes so that the service can be accessed externally.
- LoadBalancer: Integrates with cloud provider load balancers to expose your service externally.
- Ingress: A collection of rules that allow inbound connections to reach the cluster services.

Quiz

Who can tell me the key difference between ClusterIP and NodePort? Type your answer in the chat.

Quiz

True or False: An Ingress resource is itself a load balancer.

Configuring Networking Components

Let’s now put theory into practice. We’ll first create a Service that uses a ClusterIP and then modify it to use a NodePort.

Create a Basic Service YAML File

Open your code editor and create a file called advanced-service.yaml

The YAML defines a Service that selects Pods labeled with app: nginx and tier: frontend. It listens on port 80 and routes traffic to port 80 of the Pods. Right now, it’s set to ClusterIP, meaning it’s only accessible within the cluster.

Create a Basic Service YAML File

Save the file and apply it using:

kubectl apply -f advanced-service.yaml

Then run:

kubectl get svc advanced-service

Configuring an Ingress Controller

Next, we're going to set up an Ingress to route external traffic to our Service.

Note: This demo assumes you have an Ingress Controller installed. If you're using Minikube, you can enable the Nginx Ingress Controller by running:

minikube addons enable ingress

Create an Ingress YAML File

Create a file called nginx-ingress.yaml

This Ingress resource defines a rule that directs traffic from the host nginx.local to our Service named advanced-service on port 80. The annotation rewrites the URL path so that the Service receives the correct path.

Apply the Ingress

Use: *kubectl apply -f nginx-ingress.yaml*

Now, if you’re using Minikube, you can run:

minikube tunnel

and add an entry in your hosts file (or use DNS) so that nginx.local points to your cluster IP.

Persistent Storage & Volumes

Introduction to Persistent Volumes and Claims

Now we'll move on to storage.

- In Kubernetes, data persistence is managed via Persistent Volumes (PVs) and Persistent Volume Claims (PVCs).
- A PV is a piece of storage in the cluster.
- A PVC is a request for storage by a user.

Introduction to Persistent Volumes and Claims

To emphasize:

- Persistent Volume (PV): The storage resource (could be local, NFS, cloud storage, etc.)
- Persistent Volume Claim (PVC): The request made by your application.
- Pod: Uses the PVC to access the storage.

Quiz: What advantage does using a PVC give over ephemeral storage in a Pod?

Live YAML Demo – Creating a PVC

Create a PVC YAML File

Create a file named *pvc.yaml*

This YAML file requests a persistent volume of 1Gi with the ‘standard’ storage class. The access mode ReadWriteOnce means that the volume can be mounted as read-write by a single node.

Apply the PVC

Use:

kubectl apply -f pvc.yaml

Then run:

kubectl get pvc nginx-pvc

to confirm it’s bound.

Quiz

What is the purpose of a PVC?

Deploying a Stateful Application

Deploying a StatefulSet

StatefulSets are used for applications that require persistent identities and stable storage, such as databases. Let’s deploy a simple stateful application using a StatefulSet.

Create a StatefulSet YAML File

Create a file named statefulset.yaml.

This StatefulSet creates 2 replicas of an Nginx Pod. Notice that it defines a volumeClaimTemplates section which creates a PVC for each Pod automatically. The serviceName ‘nginx’ is used for stable network identity.

Apply the StatefulSet

kubectl apply -f statefulset.yaml

Then, run:

kubectl get pods -l app=nginx

to see your stateful Pods.

RBAC and Network Policies

Exploring RBAC (Role-Based Access Control)

Security is critical. Kubernetes uses Role-Based Access Control (RBAC) to manage permissions within the cluster. RBAC defines what actions users or service accounts can perform.

Create a Simple Role YAML File

Create a file named *role.yaml*

This role, named pod-reader, allows reading of pods in the default namespace.

Create a RoleBinding YAML File

Now create rolebinding.yaml to bind this role to a user (or service account)

This RoleBinding grants the ‘example-user’ the permissions defined in the pod-reader role.

Applying these files

To apply these files, use:

kubectl apply -f role.yaml

kubectl apply -f rolebinding.yaml

Then, check:

kubectl get rolebinding read-pods-binding -o yaml

Network Policies

Network Policies allow you to control the traffic flow at the IP address or port level between Pods. They are crucial for securing inter-Pod communication.

Create a Simple Network Policy

Create a file named *network-policy.yaml*

This policy denies all ingress and egress traffic for Pods in the namespace. You can then create more specific policies to allow only certain traffic.

Apply the network policy:

kubectl apply -f network-policy.yaml

Then, run:

kubectl get networkpolicy

Quiz

What is the purpose of a Network Policy?

Thank you, and stay tuned next time for part 3!

Next stream coming up…

Stay tuned for our stream, next:

Today in Global Hack Week Day 4

12:00 - 13:00 ET

https://events.mlh.io/events/12329