Introduction to Kubernetes Part 1

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

Our Goals for Today

- Master Kubernetes components—including Pods, Nodes, Deployments, ReplicaSets, and Services—plus additional objects such as ConfigMaps, Secrets, and Probes.
- Write and modify numerous YAML files for Deployments, Services, and advanced configurations while troubleshooting in real time.
- Get some hands-on practice with kubernetes.
- Learn to add environment variables, resource limits, health probes, autoscaling, etc.

But first, welcome!

Poll:

1. How would you rate your Kubernetes expertise? (Beginner, Intermediate, Advanced)
2. In one word, what does ‘orchestration’ mean to you?

What is Kubernetes

Kubernetes is an open-source platform designed to automate deploying, scaling, and operating application containers. In simple terms, think of it as a system that manages containers, which are like lightweight, portable units that package up your application along with everything it needs to run.

Containers

Containers are a way to package your code and its dependencies together. They allow you to run applications reliably when moved from one computing environment to another.

Orchestration

Orchestration refers to managing these containers automatically. Kubernetes does this by ensuring your application runs the right number of containers, automatically scaling up or down based on demand, and handling updates without downtime.

The Tools You’ll Need

A local Kubernetes cluster. This can be set up using tools such as Minikube, kind, or Docker Desktop’s built-in Kubernetes.

Here is a link to install minikube. If you don’t already have some way to get clusters up and running, it’s a great resource: https://minikube.sigs.k8s.io/docs/

The Tools You’ll Need

The command-line tool called ‘kubectl’ which is used to interact with the cluster.

Try these install tools: https://kubernetes.io/docs/tasks/tools/

The Tools You’ll Need

A code editor, like Visual Studio Code (VS Code), to edit YAML files.

https://code.visualstudio.com/

Instructions for Cluster Setup

If you’re using Minikube, open your terminal and type:

```
minikube start
```

If you prefer kind, use:

```
kind create cluster --name demo-cluster
```

Once your cluster is running, type:

```
kubectl get nodes
```

This command lists the nodes (computers) in your Kubernetes cluster. You should see at least one node listed.

Please now open your terminal and run:

```
kubectl version --client
```

This shows the version of kubectl you’re using.

Kubernetes Architecture Overview

Let’s now look at the overall architecture of Kubernetes. The key components:

- Pods: The smallest deployable unit in Kubernetes. A Pod can contain one or more containers. Think of a Pod as a wrapper for your container(s).
- Nodes: These are the machines (virtual or physical) that run your Pods.
- Deployments: A Deployment is a higher-level concept that manages stateless applications. It tells Kubernetes how many copies of an application you want and handles updates.
- ReplicaSets: A ReplicaSet, managed automatically by a Deployment, ensures that the desired number of Pods are running.
- Services: A Service is used to expose your Pods to other parts of your cluster or even externally.

Quiz

Which component do you think is responsible for replacing a failed Pod automatically? Type your answer in the chat.

Code Example

Now, let’s run this command in your terminal to see all Pods running in all namespaces:

```
kubectl get pods --all-namespaces
```

Creating a Deployment

Now that we’ve covered the basics, it’s time for a hands-on exercise. We’re going to create a Deployment that runs an Nginx web server. I will walk you through creating a YAML file step by step.

But first, what the heck is a YAML??

- YAML is a human-readable data serialization language. Kubernetes uses YAML files to describe objects like Deployments, Services, and more.

Creating the YAML

Let’s create a file named nginx-deployment.yaml. I’ll show you the entire file, and then we’ll go through it section by section.

Challenge: Customize your YAML File

Change the number of replicas from 2 to 3 and update the value of WELCOME_MSG to ‘Hello, [Your Name] from Kubernetes!’ Save the file.

To apply the Deployment, type in your terminal:

```
kubectl apply -f nginx-deployment.yaml
```

Quiz

Which field in the YAML file sets the number of Pods to be created? Type your answer in the chat.

Creating a Service

Now that our Deployment is running, we need to expose it so that we can access the Nginx web server from outside the cluster. We do this using a Kubernetes Service.

Show the YAML File for the Service

Create a file called nginx-service.yaml with the following content…

- apiVersion & kind: “We are creating a Service, so we use apiVersion v1 and kind Service.”
- metadata: “The Service is named ‘nginx-service’ and we add an annotation for extra information.”
- spec.selector: “This tells the Service which Pods to target by matching labels. It will select the Pods that have both app: nginx and tier: frontend.”
- spec.ports: “This defines the port mapping. The Service listens on port 80 and sends traffic to port 80 of the targeted Pods.”
- spec.type: “We’re using NodePort, which makes the service accessible on a port on each Node.”

Apply it!

Now, please open your editor, create the file, and then apply it by running:

```
kubectl apply -f nginx-service.yaml
```

Then run:

```
kubectl get svc
```

to see your new service.

Let me know in the chat how it goes!

Quiz

What does the ‘selector’ field in the Service YAML do?

Challenge

Next, try modifying the service type. Change type: NodePort to type: ClusterIP in the YAML file, save, and reapply it. Then run:

```
kubectl get svc
```

Notice how the external port details change. Again, feel free to let us know what you see in the chat!

Scaling, Rolling Updates & Health Checks

Scaling is one of Kubernetes' core features. It allows you to change the number of Pods running your application on the fly.

Let's scale our Deployment from 3 replicas (if you changed it earlier) to 4.

```
kubectl scale deployment/nginx-deployment --replicas=4
```

Now, run:

```
kubectl get pods
```

to verify that there are now 4 Pods running.

Rolling Updates

Kubernetes allows you to update your application without downtime using rolling updates. This means updating the Pods gradually.

Rolling Updates: Example

Now, let’s change the image version of our Nginx container. Open your nginx-deployment.yaml file and change the image from nginx:latest to nginx:1.19. Save the file, then run:

```
kubectl apply -f nginx-deployment.yaml
```

To monitor the update, run:

```
kubectl rollout status deployment/nginx-deployment
```

Watch the progress and note that the Pods are updated gradually.

Rollback

Once the update is complete, let’s simulate a rollback. Type:

```
kubectl rollout undo deployment/nginx-deployment
```

This command reverts the Deployment to the previous configuration. Verify the rollback with:

```
kubectl rollout status deployment/nginx-deployment
```

Health Checks & Resource Management

Health checks ensure that your Pods are functioning correctly. We’ve already defined readiness and liveness probes in our YAML. Let’s review them.

In your terminal, type:

```
bash kubectl describe pod <pod-name>
```

Replace <pod-name> with the name of one of your Pods (you can get it from *kubectl get pods*).

Advanced Customization & ConfigMaps

Let’s move on to advanced customizations. We’re going to add configuration data to our application using ConfigMaps, which allow you to decouple configuration from your container images.

Creating a ConfigMap

Create a file called configmap.yaml with the following content:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-config
data:
  index.html: |
    <html>
      <body>
        <h1>Welcome to an Advanced Kubernetes Deployment!</h1>
      </body>
    </html>
```

Creating a ConfigMap

Now, apply it with:

```
kubectl apply -f configmap.yaml
```

This ConfigMap stores a custom HTML file.

Updating the Deployment to Use the ConfigMap

Next, update your nginx-deployment.yaml file to mount this ConfigMap as a volume. Under the container definition, add a volumeMounts section and under the Pod spec add a volumes section. The updated part should look like this…

Updating the Deployment to Use the ConfigMap

```
kubectl apply -f nginx-deployment.yaml
```

Code Example

Now, try adding an environment variable from your ConfigMap into your container. Modify the container spec to add an environment variable that references a key from your ConfigMap using the valueFrom field.

Reapply the YAML and check that the environment variable is set by running:

```
kubectl describe pod <pod-name>
```

Customizing Services & Exposure Options

Finally, let’s quickly review how to customize our Service to change how our application is exposed.

Open your nginx-service.yaml file again. Change the service type from NodePort to LoadBalancer if your environment supports it, or try ClusterIP if not. Save and reapply:

```
kubectl apply -f nginx-service.yaml
```

Then run:

```
kubectl get svc
```

and observe the changes in your Service’s external endpoints.

Quiz

What is the difference between a ClusterIP and a NodePort service?

Autoscaling

Let’s take a look at one, more advanced, concept before we log off…

Run the following command to create a Horizontal Pod Autoscaler (HPA) for your deployment:

```
kubectl autoscale deployment/nginx-deployment --cpu-percent=50 --min=2 --max=10
```

Autoscaling

Now, run:

```
kubectl get hpa
```

This command sets up autoscaling based on CPU usage. It will automatically adjust the number of replicas if the CPU usage goes above 50%.

Let’s recap

What command do you use to scale a deployment?

Let’s recap

Name one type of probe used in Kubernetes.