Deploying a Game on Kubernetes Cluster!

In this project, I successfully deployed a game on a Kubernetes cluster, gaining hands-on experience with container orchestration and cloud-native application deployment. The process involved several crucial steps: creating a deployment, exposing it via a service, making necessary adjustments to the service, and finally accessing the game externally. Kubernetes is a powerful platform for managing containerized applications, and this project allowed me to explore its features for deployment, scaling, and monitoring applications in a Kubernetes environment. Below are the detailed steps I took to deploy and manage the game on the cluster, along with some insights into the tools and concepts used throughout the process.

1. Creating the Deployment:

I started by creating a deployment named "app" using the following command:

```
controlplane $ kubectl create deployment app --image=jayshrilandge/mario:game
deployment.apps/app created
```

This command pulls the container image from Docker Hub and deploys it on the Kubernetes cluster. Kubernetes then manages the application, ensuring scalability, replication, and updates. It abstracts the infrastructure, providing efficient orchestration for containerized applications. The deployment allows for declarative configuration, meaning any changes can be managed easily by updating and applying the configuration without manually managing containers.

Once the deployment is created, Kubernetes automatically ensures that the desired number of replicas are running, providing high availability for the application. If a pod fails, Kubernetes automatically replaces it to maintain the desired state. Additionally, the deployment simplifies application updates by allowing rolling updates, ensuring that the application is updated without downtime. This approach helps in maintaining consistency and reliability in production environments.

2. Checking the Deployment and Pods:

After successfully creating the deployment, it was important to verify that the deployment and the pods were functioning correctly. I used the following commands:

```
controlplane $ kubectl get deployment
NAME   READY   UP-TO-DATE   AVAILABLE   AGE
app    1/1     1            1           12m
```

This command provides an overview of the deployment's status, showing whether it is running as expected and if the desired number of pods are created. It confirms that Kubernetes has successfully deployed the containerized game and that the cluster is aware of it.

```
controlplane $ kubectl get pod
NAME                   READY   STATUS    RESTARTS   AGE
app-85f9cbcfb6-xgt4q   1/1     Running   0          5m20s
```

This command checks the status of the pods running the game within the cluster. Pods are the smallest deployable units in Kubernetes, containing one or more containers. By checking the pod status, I ensured that the containerized game was running properly and could troubleshoot any deployment issues.

3. Exposing the Deployment:

To make the deployed game accessible externally, I exposed the deployment by creating a service with the following command:

```
controlplane $ kubectl expose deployment app --port=8080
service/app exposed
```

This command creates a Kubernetes service that acts as a bridge between the external network and the pod(s) running the game. By exposing the deployment, Kubernetes ensures that the game can be accessed through port 8080. The --port=8080 option specifies the port on which the game will be exposed to the outside world. Services in Kubernetes provide stable networking, load balancing, and a consistent way to access the pods, regardless of how many are running. This step is crucial for ensuring that the game is available to external users, as the service directs network traffic to the appropriate pods, ensuring smooth communication

4. Checking the Service:

After exposing the deployment, I checked the service configuration using the following command:

```
controlplane $ kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
app          NodePort    10.107.195.136  <none>        8080:32613/TCP   72s
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP          26d
```

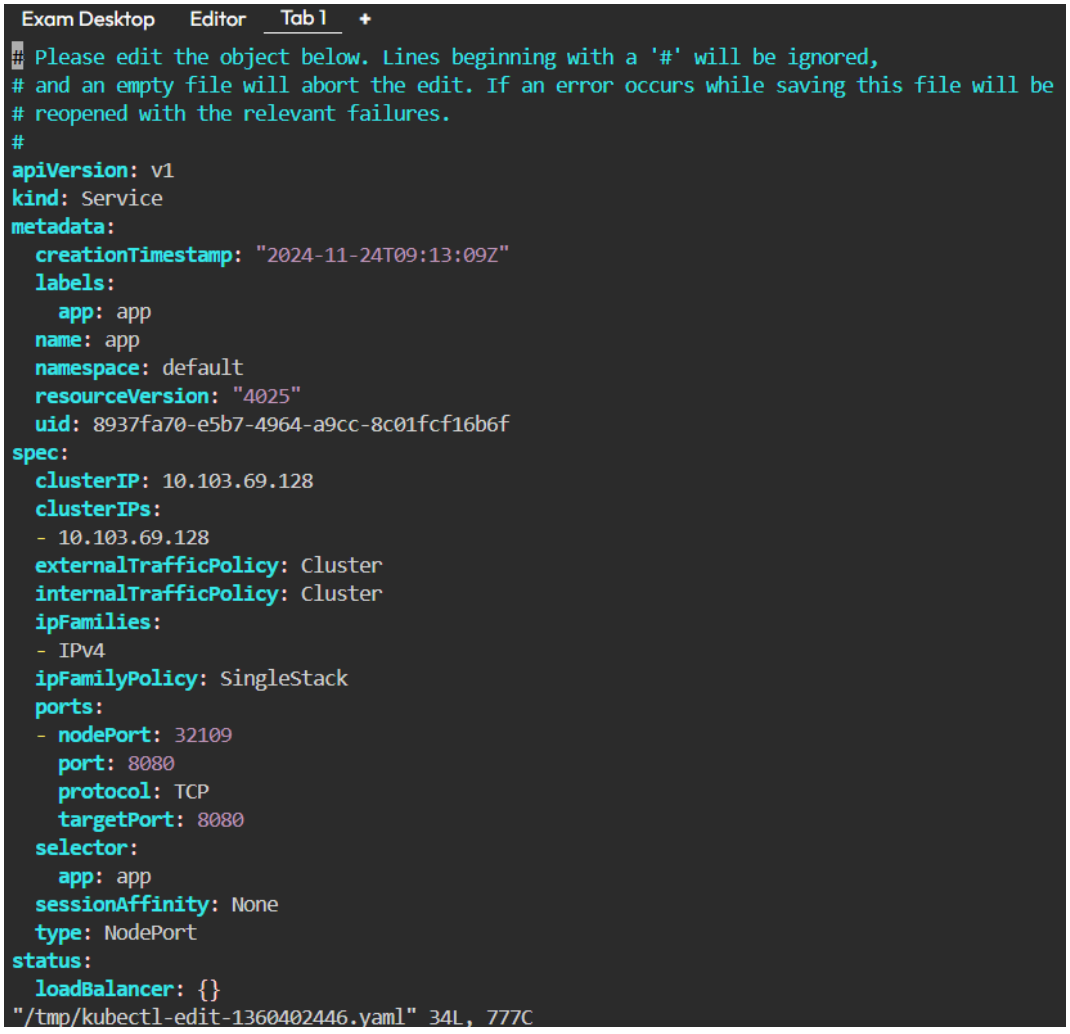
This command provides details about the service, such as its name, type, and the exposed ports. It also shows the external IP address, which is essential for accessing the game from outside the Kubernetes cluster. By reviewing this information, I confirmed that the game was successfully exposed on port 8080 and properly configured for external access. The service's external IP allows external users to interact with the game, ensuring it is accessible as intended. Additionally, this step ensures that the Kubernetes service is properly linked to the deployment, enabling smooth communication between the application and external users. Finally, I verified the correctness of the port mapping to ensure the game was accessible through the correct endpoint.

5. Editing the Service:

While the initial service setup exposed the game, I needed to make some adjustments to the service configuration for more specific requirements. I used the following command to edit the service:

```
controlplane $ kubectl edit svc app
service/app edited
```

This command opens the service definition in an editor, allowing me to modify configurations such as changing the exposed port, adjusting the service type (e.g., NodePort), or making other adjustments as needed.

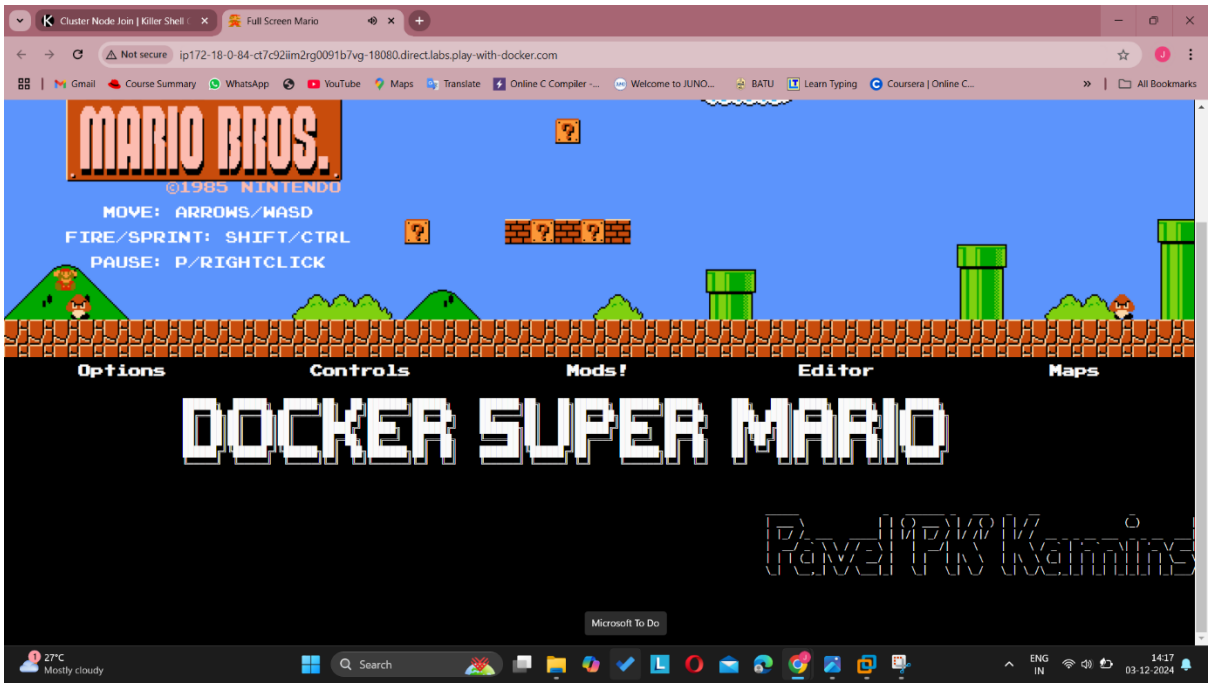


By modifying the service type to "NodePort", I enabled the game to be accessible through an external port on the cluster's nodes. This flexibility is one of the advantages of Kubernetes services, as they allow fine-grained control over how applications are accessed, both internally and externally. Editing the service configuration allowed me to

tailor the deployment to meet the requirements and ensure the game was accessible as intended.

6. Accessing the Game:

After configuring the service, I was able to access the game externally using the exposed port (8080). The game was now accessible through the node's IP address and the assigned port, allowing anyone to play the game directly through their web browser.



I took a screenshot of the game running successfully on the Kubernetes cluster as evidence of the successful deployment. Accessing the game externally demonstrated that all configurations—deployment, service exposure, and port assignment—were correctly set up and functioning.

Conclusion:

This project provided practical experience with Kubernetes and container orchestration, enhancing my skills in deploying and managing scalable applications. It deepened my understanding of Kubernetes' features, and I'm eager to explore more about container management and DevOps in future projects.