Databases and Storage System

Prepared By: Aatiz Ghimire, for Herald Center for AI.

Summer, 2025

Learning Objectives

- Understand the core architecture and components of a Kubernetes cluster (Pods, Nodes, Deployments, Services, etc.).
- Learn how to set up and interact with a Kubernetes environment using `kubectl` and a cluster interface.
- Deploy, scale, and update containerized applications within Kubernetes.
- Apply declarative configuration management using YAML manifests.
- Explore real-world Kubernetes environments using tools like Play with Kubernetes (PWK) for hands-on learning.

---

1 Introduction

Kubernetes (K8s) is an open-source platform designed to automate the deployment, scaling, and management of containerized applications. Building on container technologies like Docker, it orchestrates containers across a cluster of nodes, ensuring high availability and scalability [1]. This worksheet utilizes Play with Kubernetes (PWK), a free, browser-based tool, to create a cluster with one master node and two worker nodes, based on a guide by Emre Ozan.

Key Concepts Explained:

- **Pod**: The smallest deployable unit in Kubernetes, wrapping one or more containers that share resources like storage and network. For example, a Pod might run an Nginx container, similar to `docker run nginx`.
- **Deployment**: A controller that manages Pods, ensuring the desired number of replicas are running and enabling updates or rollbacks.
- **Service**: A method to expose Pods to the network, either internally or externally (e.g., via a web browser), acting as a load balancer.
- **Cluster**: A set of nodes managed by Kubernetes, with a master node (control plane) orchestrating tasks and worker nodes executing applications.

PWK Note: PWK provides a 4-hour session with pre-installed Kubernetes. Use `Ctrl+Insert` to copy and `Shift+Insert` to paste commands, as standard copy-paste may not work.

2 Prerequisites

- Access to Play with Kubernetes (requires a Docker Hub or GitHub account).
- Basic Docker knowledge (e.g., running `docker run nginx` to start a container).
- Familiarity with Linux commands (`cd`, `ls`, `nano`) for file navigation and editing.

3 Workshop Agenda

1. Setting Up a Kubernetes Cluster
2. Basic Kubernetes Commands
3. Deploying an Application
4. Scaling and Updating Applications
5. Exposing Applications
6. Hands-On: Deploying a Web App
7. Verifying Cluster Health

4 Setting Up a Kubernetes Cluster

This section guides you through creating a Kubernetes cluster with one master node and two worker nodes using PWK.

4.1 Exercise: Initialize the Cluster

1. **Start Session**: Visit labs.play-with-kubernetes.com, log in with your Docker Hub or GitHub account, and start a new session. You'll see a welcome banner with instructions to add instances.

2. **Create Master Node**:

   - Click `Add New Instance` to create `node1` as the master node.
   - Initialize the cluster by running this command on `node1` (copy with `Ctrl+Insert`, paste with `Shift+Insert`):

```
kubeadm init --apiserver-advertise-address $(hostname -i) --pod-network-cidr 10.5.0.0/16
```

Listing 1: Initialize master

Explanation: This sets up the master node's control plane. The `--apiserver-advertise-address $(hostname -i)` uses the node's internal IP, and `--pod-network-cidr 10.5.0.0/16` reserves a network range for Pods. After execution, copy the displayed `kubeadm join` command for worker nodes.

   - Configure `kubectl` on `node1`:

```
mkdir -p $HOME/.kube
cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
chown $(id -u):$(id -g) $HOME/.kube/config
```

Listing 2: Configure kubectl

Explanation: These commands create a `config` file in `.kube`, enabling `kubectl` to interact with the cluster.

   - Set up networking with Flannel:

```
kubectl apply -f https://raw.githubusercontent.com/flannel-io/flannel/master/Documentation/
    kube-flannel.yml
```

Listing 3: Install Flannel

Explanation: Flannel establishes a network overlay for Pod communication across nodes. Wait about a minute for initialization.

3. **Check Master Status**: Verify cluster components:

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

Listing 4: List pods

Expected Output: Pods like `kube-apiserver` and `kube-flannel-ds` in `kube-system` should be `Running`.

4. **Create Worker Nodes**:

- Click `Add New Instance` twice to create `node2` and `node3`.
- Run the `kubeadm join` command on each worker node:

```
kubeadm join 192.168.0.18:6443 --token 075hg5.ywr4gr7glv5jjglg \
    --discovery-token-ca-cert-hash sha256:4
        db9943f58f1df5da3f16ff758eb41ab83a5329203583af46912fd8a56376774
```

Listing 5: Join worker node

Explanation: This joins workers to the master using a secure token and hash, with 192.168.0.18:6443 as the master's API server.

5. **Verify Cluster**: Check nodes on `node1`:

```
kubectl get nodes
```

Listing 6: List nodes

Expected Output:

```
NAME     STATUS     ROLES    AGE   VERSION
node1    Ready      master   5m    v1.21.0
node2    Ready      <none>   2m    v1.21.0
node3    Ready      <none>   2m    v1.21.0
```

All nodes should be `Ready`.

Task: Set up the cluster and ensure all nodes are `Ready`.
Troubleshooting:

- If nodes are `NotReady`, check Flannel Pods with `kubectl get pods -n kube-system -l app=flannel`. Reapply Flannel YAML if needed.
- If `kubeadm join` fails, verify the command matches the master's output.
- Restart PWK if the session crashes.

5 Basic Kubernetes Commands

Use `kubectl` to manage your cluster.

5.1 Exercise: Exploring kubectl

1. **Check Cluster Info** (on `node1`):

```
kubectl cluster-info
```

Listing 7: Cluster info

Explanation: Displays the API server and control plane endpoints. **Expected Output**: URLs like `kubernetes.default.svc:443`.

2. **List Nodes**:

```
kubectl get nodes
```

Listing 8: List nodes

Explanation: Shows all nodes with their status and roles.

3. **List Pods**:

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

Listing 9: List pods

Explanation: Lists all Pods across namespaces.

Task: Run these commands and note the number of nodes and Pods.

6 Deploying an Application

Deploy a simple Nginx web server.

6.1 Exercise: Create Nginx Deployment

1. **Create Deployment** (on `node1`):

```
kubectl create deployment nginx-app --image=nginx:alpine --replicas=2
```

Listing 10: Create Nginx Deployment

Explanation: Creates a `nginx-app` Deployment with 2 replicas using `nginx:alpine`.

2. **Verify Deployment**:

```
kubectl get deployments
```

Listing 11: List Deployments

Expected Output:

```
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
nginx-app  2/2     2            2           1m
```

3. **List Pods**:

```
kubectl get pods
```

Listing 12: List Pods

Expected Output:

```
NAME                          READY   STATUS    RESTARTS   AGE
nginx-app-abcde12345-xyz   1/1     Running   0          1m
nginx-app-fghij67890-pqr   1/1     Running   0          1m
```

Task: Deploy Nginx and confirm 2 Pods are running.
Troubleshooting:

- Use `kubectl describe pod <pod-name>` if Pods are `Pending`.
- Check internet access for `ImagePullBackOff` errors.

7 Scaling and Updating Applications

Adjust Pod count and update the app.

7.1 Exercise: Scale and Update

1. **Scale Deployment** (on `node1`):

```
kubectl scale deployment nginx-app --replicas=3
```

Listing 13: Scale Deployment

Explanation: Increases replicas to 3 for load balancing. Verify with `kubectl get pods`; expect 3 `Running` Pods.

2. **Update Image**:

```
kubectl set image deployment/nginx-app nginx=nginx:alpine3.20
```

Listing 14: Update image

Explanation: Updates to `nginx:alpine3.20` with rolling updates. Monitor with:

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

Listing 15: Check rollout

Expected Output: `deployment "nginx-app" successfully rolled out`.

Task: Scale to 3 replicas and update the image.

8 Exposing Applications

Make the Nginx app accessible externally.

8.1 Exercise: Create a Service

1. **Create NodePort Service** (on `node1`):

```
kubectl expose deployment nginx-app --type=NodePort --port=80
```

Listing 16: Expose Nginx Service

Explanation: Exposes port 80 on a random high port (30000–32767).

2. **List Services**:

```
kubectl get services
```

Listing 17: List Services

Expected Output:

```
NAME        TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)         AGE
nginx-app   NodePort    10.96.123.45   <none>        80:30800/TCP    1m
```

Note the `NODEPORT` (e.g., 30800).

3. **Access Service**: Click the port link in PWK (e.g., `30800`) to see the Nginx page.

Task: Expose the app and access it via the port link.

9 Hands-On: Deploying a Web App

Deploy a custom Nginx-based web app.

9.1 Exercise: Deploy Web App

1. **Create Deployment YAML** (on `node1`):

```
nano deployment.yaml
```

Listing 18: Create deployment.yaml

Add:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - name: web
        image: nginx:alpine
        ports:
        - containerPort: 80
        volumeMounts:
        - name: html-volume
          mountPath: /usr/share/nginx/html
      volumes:
```

```
      - name: html-volume
        configMap:
          name: web-html
```

Listing 19: Web App Deployment YAML

2. **Create ConfigMap for HTML**:

```
nano configmap.yaml
```

Listing 20: Create configmap.yaml

Add:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: web-html
data:
  index.html: |
    <html>
    <body>
      <h1>Welcome to My Kubernetes App!</h1>
      <p>Deployed on $(date) by studentb.</p>
    </body>
    </html>
```

Listing 21: ConfigMap YAML

Apply it:

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

Listing 22: Apply ConfigMap

3. **Apply Deployment**:

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

Listing 23: Apply Deployment

4. **Expose Deployment**:

```
kubectl expose deployment web-app --type=NodePort --port=80
```

Listing 24: Expose Web Service

5. **Access App**: Check services:

```
kubectl get services
```

Listing 25: List Services

Click the assigned port link (e.g., `30500`) to see your page.

Task: Deploy the web app and verify your custom webpage.
Troubleshooting:

- Use `kubectl describe pod <pod-name>` or `kubectl logs <pod-name>` if Pods fail.
- Verify ConfigMap with `kubectl get configmap`.
- Restart PWK if the port link fails.

10 Verifying Cluster Health

Monitor the cluster's status.

10.1 Exercise: Check Cluster Health

1. **Check Node Status**:

```
kubectl get nodes
```

Listing 26: List nodes

Explanation: Lists all nodes with their status. **Status Meanings**: - `Ready`: Node is operational. - `NotReady`: Investigate with `kubectl describe node <node-name>`.

2. **Check Pod Status**:

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

Listing 27: List pods

Explanation: Shows all Pod statuses. **Status Meanings**: - `Running`: Active. - `Pending`: Awaiting resources. - `CrashLoopBackOff`: Check logs with `kubectl logs`.

3. **Check Events**:

```
kubectl get events
```

Listing 28: Describe events

Explanation: Displays recent cluster events for troubleshooting.

Task: Monitor the cluster and note any non-`Ready` or non-`Running` statuses.

11 Conclusion

You've learned to set up a Kubernetes cluster, deploy apps, scale them, expose them, and monitor health on PWK.

Next Steps:

- Explore advanced features like ConfigMaps, Secrets, and Ingress.
- Try Minikube for local Kubernetes practice.
- Save YAML files externally due to PWK's 4-hour limit.

References

[1] Kubernetes, "What is Kubernetes?", `https://kubernetes.io`, 2023.

[2] Docker Docs, "Play with Kubernetes Overview", `https://docs.docker.com`, 2024.

The - End