Kubernetes Lab Report: Container Orchestration for Java Applications

1. Setting Up a Kubernetes Cluster

Kubernetes, often referred to as "K8s," is an open-source container orchestration platform designed to automate the deployment, scaling, and operation of application containers. Kubernetes ensures that containers are efficiently managed and run as per defined rules, making it essential in cloud-native and microservices architectures.

Steps:

1. **Install Minikube:** Minikube is a tool that runs a single-node Kubernetes cluster locally on your machine. It's a great starting point for learning Kubernetes.
   - Download and install Minikube
   - Ensure you have Docker or a hypervisor like VirtualBox installed to host the virtualized Minikube environment.
2. **Start Minikube:** Once Minikube is installed, start the cluster by running the following:

   ```
   minikube start
   ```

This command initializes a local Kubernetes cluster with one control plane (master) and one node.

2. Deploying a Java Application in Kubernetes

A **Deployment** in Kubernetes manages the lifecycle of application pods. It ensures that a specified number of replicas are running at any given time, and can also handle rolling updates.

Steps:

1. **Create a Deployment Configuration:** Define how your Java application will be deployed using a YAML file (e.g., `deployment.yaml`):

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: java-app-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: java-app
```

```
  template:
    metadata:
      labels:
        app: java-app
    spec:
      containers:
      - name: java-app
        image: myapp:latest
        ports:
        - containerPort: 8080
```

- **replicas**: The number of pod instances you want Kubernetes to maintain.
- **matchLabels**: Used to link Pods to the deployment.
- **containers**: Specifies the container image (in this case, the Java application) and the port it will expose (8080).

2. **Apply the Deployment:** To create the deployment in Kubernetes, run the command:

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

This command deploys the Java application by running it as three separate pods across the cluster.

3. Exploring Kubernetes Components

Pods:

- **Definition:** Pods are the smallest and simplest Kubernetes objects. A pod can host one or multiple tightly-coupled containers. In our case, each pod will host our Java application.
- **Importance:** They are ephemeral, meaning if a pod fails, Kubernetes ensures a new one is created as a replacement.

Deployments:

- **Definition:** A Deployment is a higher-level Kubernetes abstraction that manages Pods and ReplicaSets. It ensures the right number of Pods are running at all times and handles application updates seamlessly.
- **Importance:** It's crucial for maintaining high availability, rolling updates, and rollback mechanisms.

Services:

- **Definition:** A Service in Kubernetes defines how to expose the Pods to other applications or the outside world. Services enable network access to Pods, decoupling the IP of the Pod from the application.
- **Types of Services:**

- **ClusterIP** (default): Exposes the service internally to the cluster.
- **NodePort**: Exposes the service on each node's IP address at a static port.
- **LoadBalancer**: Exposes the service externally using a cloud provider's load balancer.

Example `service.yaml`: "'yaml apiVersion: v1 kind: Service metadata: name: java-app-service spec: selector: app: java-app ports:
- protocol: TCP port: 80 targetPort: 8080 type: LoadBalancer "'
This exposes the Java application externally on port 80 (which will map to internal 8080).

4. Load Balancing, NodePort, and Ingress

Load Balancer: A **LoadBalancer** service type distributes incoming traffic across multiple Pods, improving availability and scalability.

1. When deploying the service, using `type: LoadBalancer`, Kubernetes automatically sets up external load balancing (on supported cloud providers), distributing traffic evenly to the Pods.

   Command to expose the deployment:

   ```
   kubectl expose deployment java-app-deployment --type=LoadBalancer --name=java-app-servi
   ```

2. The external IP of the LoadBalancer will allow external users to access the Java application.

NodePort: The **NodePort** service exposes the application on a static port on each node in the cluster.

- Use NodePort if you want to expose your application on a specific port on the cluster's nodes. Example:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: java-app-nodeport
spec:
  type: NodePort
  ports:
  - port: 8080
    nodePort: 30007
  selector:
    app: java-app
```

This service exposes your application on port `30007` on each node.

Ingress: **Ingress** is an API object that manages external access to services, typically HTTP. Ingress allows fine-grained control over URL routing, SSL ter-

mination, and load balancing.

1. **Ingress Controller**: First, you need to install an ingress controller in the cluster (e.g., NGINX or Traefik).
2. **Ingress Configuration**: Create an `ingress.yaml` file that defines how to route traffic:

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: java-app-ingress
spec:
  rules:
  - host: java-app.mydomain.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: java-app-service
            port:
              number: 80
```

This configuration will route traffic to the Java application at `java-app.mydomain.com`.

5. Deploying Multiple Instances & Scaling

Kubernetes simplifies scaling applications to handle increased load by adding more replicas.

- **Scaling up/down**: You can scale the number of Pods manually or automatically based on CPU usage.

```
kubectl scale deployment java-app-deployment --replicas=5
```

This command will scale the deployment to 5 Pods.

Personal Takeaways and Summary

Working through Kubernetes and deploying my Java application has deepened my understanding of how modern applications are orchestrated in the cloud. Key insights include:

- **Pods** are transient and lightweight units of deployment. While they can scale horizontally, they are also ephemeral, emphasizing the need for managing state elsewhere.
- **Deployments** ensure my application runs reliably, with the ability to roll back changes and automatically update without downtime.

- **Services** decouple the internal workings of Kubernetes from the external world, offering flexible ways to expose applications.
- **Load Balancing** and **Ingress** are critical for managing traffic across multiple Pods, ensuring fault tolerance and high availability.
- Kubernetes is designed with resilience and scalability in mind, making it perfect for microservices architectures where high availability is non-negotiable.

In conclusion, Kubernetes streamlines complex container management tasks, letting me focus on building robust applications while automating deployment, scaling, and monitoring.