**Kubernetes Services Deep Dive - Practical Session (Day 37 DevOps Course)**

**Instructor:** Abhishek

**Key Topics Covered**

1. **Kubernetes Services Practical Demo**
   * Load Balancing
   * Service Discovery
   * Exposing Applications (Internal & External Access)
2. **Tools Used:**
   * kubectl
   * **CubeShark** (for traffic visualization)

**1. Setting Up the Demo Environment**

* **MiniKube Cluster** used for the demo.
* Clean up existing resources:
* kubectl delete deploy <deployment-name>
* kubectl delete svc <service-name>
* Verify only the default kubernetes service remains:
* kubectl get all

**2. Creating a Deployment**

* **Purpose:** Deploy a Python Django application.
* **Steps:**
  + Build Docker image:
  + docker build -t python-sample-app-demo:v1 .
  + Create deployment.yaml:
  + apiVersion: apps/v1
  + kind: Deployment
  + metadata:
  + name: sample-python-app
  + spec:
  + replicas: 2
  + selector:
  + matchLabels:
  + app: sample-python-app
  + template:
  + metadata:
  + labels:
  + app: sample-python-app
  + spec:
  + containers:
  + - name: python-app
  + image: python-sample-app-demo:v1
  + ports:
  + - containerPort: 8000
  + Apply deployment:
  + kubectl apply -f deployment.yaml
  + Verify pods:
  + kubectl get pods -o wide # Shows dynamic Pod IPs

**3. Service Discovery & Load Balancing**

**Problem:** Pod IPs change dynamically → Need stable access.  
**Solution:** **Kubernetes Service** (uses labels/selectors).

**Service Types:**

1. **ClusterIP (Default):** Internal access only.
2. **NodePort:** Exposes app on worker node IP.
3. **LoadBalancer:** Assigns external IP (cloud providers).

**4. Creating a NodePort Service**

* **Purpose:** Expose app internally (within the organization).
* **Steps:**
  + Create service.yaml:
  + apiVersion: v1
  + kind: Service
  + metadata:
  + name: python-django-app-service
  + spec:
  + type: NodePort
  + selector:
  + app: sample-python-app # Must match Pod label
  + ports:
  + - port: 80
  + targetPort: 8000
  + nodePort: 30007
  + Apply service:
  + kubectl apply -f service.yaml
  + Access app:
    - **Internally (ClusterIP):**
    - curl -L http://<cluster-ip>:80/demo
    - **NodePort (via Minikube IP):**
    - minikube ip # Get node IP
    - curl -L http://<node-ip>:30007/demo

**5. Exposing App Externally (LoadBalancer)**

* **For cloud providers (AWS/GCP/Azure):**
  + Edit service:
  + kubectl edit svc python-django-app-service
  + Change type: NodePort → type: LoadBalancer.
  + Cloud assigns a public IP (e.g., 35.200.x.x).

**Note:** Minikube requires metallb for LoadBalancer support.

**6. Debugging Service Discovery**

* **Test:** Modify selector in service.yaml (e.g., app: sample-python-ap).
* **Result:** Service fails to route traffic (selector-label mismatch).
* **Fix:** Correct selector → Reapply service.

**7. Load Balancing Demo (Using CubeShark)**

* **Tool:** CubeShark visualizes traffic flow in Kubernetes.
* **Steps:**
  + Install CubeShark:
  + curl -sL https://git.io/cubeshark | sh
  + ./cubeshark tap -a # Monitor all namespaces
  + Access CubeShark UI: http://localhost:8899
  + Send requests:
  + for i in {1..6}; do curl http://<node-ip>:30007/demo; done
  + **Observation:** Requests alternate between Pods (e.g., 172.17.0.5 and 172.17.0.7).

**Key Takeaways**

1. **Service Discovery:** Uses labels/selectors to dynamically find Pods.
2. **Load Balancing:** Distributes traffic across Pods (round-robin).
3. **Exposure Methods:**
   * **NodePort:** Internal access (org-wide).
   * **LoadBalancer:** External access (public IP).
4. **Debugging:** CubeShark helps visualize traffic flow.

**Next Steps**

* Dedicated video on **CubeShark** (advanced traffic analysis).
* Try deploying on **cloud Kubernetes (EKS/GKE/AKS)**.

**Like, Share & Subscribe for more DevOps content!** 🚀

### \*\*Kubernetes Services Deep Dive - Practical Session (Day 37 DevOps Course)\*\*

\*\*Instructor:\*\* Abhishek

---

### \*\*Key Topics Covered\*\*

1. \*\*Kubernetes Services Practical Demo\*\*

- Load Balancing

- Service Discovery

- Exposing Applications (Internal & External Access)

2. \*\*Tools Used:\*\*

- `kubectl`

- \*\*CubeShark\*\* (for traffic visualization)

---

### \*\*1. Setting Up the Demo Environment\*\*

- \*\*MiniKube Cluster\*\* used for the demo.

- Clean up existing resources:

```sh

kubectl delete deploy <deployment-name>

kubectl delete svc <service-name>

```

- Verify only the default `kubernetes` service remains:

```sh

kubectl get all

```

---

### \*\*2. Creating a Deployment\*\*

- \*\*Purpose:\*\* Deploy a Python Django application.

- \*\*Steps:\*\*

- Build Docker image:

```sh

docker build -t python-sample-app-demo:v1 .

```

- Create `deployment.yaml`:

```yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: sample-python-app

spec:

replicas: 2

selector:

matchLabels:

app: sample-python-app

template:

metadata:

labels:

app: sample-python-app

spec:

containers:

- name: python-app

image: python-sample-app-demo:v1

ports:

- containerPort: 8000

```

- Apply deployment:

```sh

kubectl apply -f deployment.yaml

```

- Verify pods:

```sh

kubectl get pods -o wide # Shows dynamic Pod IPs

```

---

### \*\*3. Service Discovery & Load Balancing\*\*

\*\*Problem:\*\* Pod IPs change dynamically → Need stable access.

\*\*Solution:\*\* \*\*Kubernetes Service\*\* (uses labels/selectors).

#### \*\*Service Types:\*\*

1. \*\*ClusterIP (Default):\*\* Internal access only.

2. \*\*NodePort:\*\* Exposes app on worker node IP.

3. \*\*LoadBalancer:\*\* Assigns external IP (cloud providers).

---

### \*\*4. Creating a NodePort Service\*\*

- \*\*Purpose:\*\* Expose app internally (within the organization).

- \*\*Steps:\*\*

- Create `service.yaml`:

```yaml

apiVersion: v1

kind: Service

metadata:

name: python-django-app-service

spec:

type: NodePort

selector:

app: sample-python-app # Must match Pod label

ports:

- port: 80

targetPort: 8000

nodePort: 30007

```

- Apply service:

```sh

kubectl apply -f service.yaml

```

- Access app:

- \*\*Internally (ClusterIP):\*\*

```sh

curl -L http://<cluster-ip>:80/demo

```

- \*\*NodePort (via Minikube IP):\*\*

```sh

minikube ip # Get node IP

curl -L http://<node-ip>:30007/demo

```

---

### \*\*5. Exposing App Externally (LoadBalancer)\*\*

- \*\*For cloud providers (AWS/GCP/Azure):\*\*

- Edit service:

```sh

kubectl edit svc python-django-app-service

```

- Change `type: NodePort` → `type: LoadBalancer`.

- Cloud assigns a public IP (e.g., `35.200.x.x`).

\*\*Note:\*\* Minikube requires `metallb` for LoadBalancer support.

---

### \*\*6. Debugging Service Discovery\*\*

- \*\*Test:\*\* Modify selector in `service.yaml` (e.g., `app: sample-python-ap`).

- \*\*Result:\*\* Service fails to route traffic (selector-label mismatch).

- \*\*Fix:\*\* Correct selector → Reapply service.

---

### \*\*7. Load Balancing Demo (Using CubeShark)\*\*

- \*\*Tool:\*\* CubeShark visualizes traffic flow in Kubernetes.

- \*\*Steps:\*\*

- Install CubeShark:

```sh

curl -sL https://git.io/cubeshark | sh

./cubeshark tap -a # Monitor all namespaces

```

- Access CubeShark UI: `http://localhost:8899`

- Send requests:

```sh

for i in {1..6}; do curl http://<node-ip>:30007/demo; done

```

- \*\*Observation:\*\* Requests alternate between Pods (e.g., `172.17.0.5` and `172.17.0.7`).

---

### \*\*Key Takeaways\*\*

1. \*\*Service Discovery:\*\* Uses labels/selectors to dynamically find Pods.

2. \*\*Load Balancing:\*\* Distributes traffic across Pods (round-robin).

3. \*\*Exposure Methods:\*\*

- \*\*NodePort:\*\* Internal access (org-wide).

- \*\*LoadBalancer:\*\* External access (public IP).

4. \*\*Debugging:\*\* CubeShark helps visualize traffic flow.

---

### \*\*Next Steps\*\*

- Dedicated video on \*\*CubeShark\*\* (advanced traffic analysis).

- Try deploying on \*\*cloud Kubernetes (EKS/GKE/AKS)\*\*.

\*\*Like, Share & Subscribe for more DevOps content!\*\* 🚀