Kubernetes Lab Notes: Networking & Persistent Storage 🌟

Table of Contents

---

1. Lab Setup 🏢

What We Have

- **CCE Cluster** = A whole Kubernetes city
- **Node** = An apartment building in the city
- **ECS** = Your house outside the city

Installing kubectl - The City Remote Control 🎮

Q) Why install kubectl in ECS (your house) and not on the Node (apartment building)?

- It’s safer to control the city from your house than from inside an apartment building
- More convenient - you can manage everything from one place

How to Get Your Remote Control Working:

1. Install `kubectl` on your computer
2. Get the special config file (like pairing your remote)
3. Put it in the right place: `~/.kube/config`
4. Test it: `kubectl get nodes` - this should show your buildings

```
# This is like testing your remote control
kubectl get nodes
```

Sample Output:

```
NAME          STATUS   ROLES    AGE   VERSION
cce01-node1   Ready    <none>   15d   v1.25.0
cce01-node2   Ready    <none>   15d   v1.25.0
```

---

2. Pod & Node Networking 📞

What’s a Pod? 🏪

A **Pod** is like a single shop in our city. Each shop has:

- Its own phone number (IP address)
- Can be temporary (opens and closes)

Q) What do you mean by ephemeral?

- “Ephemeral” means **temporary** - like a pop-up shop
- Pods can close and reopen with **new phone numbers**
- This is why we need Services (phone books) later!

Let’s Test Communication

Step 1: Open an Nginx Shop

```
kubectl run nginx --image=nginx --port=80
```

Step 2: Find Its Phone Number

```
kubectl get pods -o wide
```

Sample Output:

```
NAME    READY   STATUS    RESTARTS   AGE   IP
NODE
nginx   1/1     Running   0          10s   10.244.1.2
cce01-node1
```

Phone number = `10.244.1.2`

Step 3: Call the Shop

```
curl 10.244.1.2
```

Q) When we write `curl <IP>` , what does it check?

- `curl` is like making a phone call 📞
- If someone answers (server works), you get the website
- If no one answers, you get an error

Q) What was the purpose of doing all this?

- To prove that every shop (Pod) gets its own phone number (IP)
- And that everyone in the city can call each other directly!

Testing Between Shops

What's Busybox? 🛠️
Q) What is a busybox?

- Busybox is like a **Swiss Army knife** - lots of small useful tools
- Perfect for testing because it's simple and has everything we need

Create a Busybox Shop and Call Nginx:

```
# Enter the busybox shop
kubectl run -it busybox --image=busybox -- /bin/sh

# Inside busybox, call nginx
telnet 10.244.1.2 80
```

Q) What did we achieve by doing this?

- Proved that shops can call each other directly!
- No need for special phone operators

---

3. Service Discovery 📖

The Big Problem 😟

Shops (Pods) keep changing their phone numbers! How do you call “Bob’s Pizza” if their number changes daily?

The Solution: Services! 🎯

A **Service** is like a **city phone book** - it gives permanent numbers to groups of shops.

Creating Multiple Shops (Deployment)

nginx-deploy.yaml:

```
apiVersion: apps/v1  # This tells Kubernetes what
"language" we're speaking
kind: Deployment     # We're creating a shop
manager
metadata:
  name: nginx        # The manager's name
spec:
  replicas: 3        # Create 3 identical shops
  selector:          # The manager looks for shops
with this tag
    matchLabels:
      app: nginx
  template:          # This is the shop blueprint
    metadata:
      labels:
        app: nginx   # All shops get this tag
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

Q) What does `selector` mean?

- It's how the manager finds "his" shops
- "Find all shops with tag `app=nginx` "

Create the Shops:

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

Creating the Phone Book (Service)

nginx-service.yaml:

```
apiVersion: v1
kind: Service          # This is a phone book
metadata:
  name: nginx-svc      # Phone book name
spec:
  selector:
    app: nginx         # List all shops with this tag
  ports:
    - protocol: TCP
      port: 8080       # Phone book number
      targetPort: 80 # Actual shop number
```

Q) What does `selector: app: nginx` mean in the Service?

- "Put all shops with tag `app=nginx` in this phone book"

Create the Phone Book:

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

Sample Output:

```
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP
PORT(S)    AGE
nginx-svc    ClusterIP   10.105.32.105   <none>
8080/TCP   5m
```

Permanent phone number = `10.105.32.105:8080`

Check Who's in the Phone Book:

```
kubectl get endpoints
```

Sample Output:

```
nginx-svc   10.244.1.2:80,10.244.1.3:80,10.244.2.5:80
```

This shows the 3 actual shops behind the phone number!

Making Your Shop Public 🌍

Regular Service = Internal company phone (only inside office)

NodePort Service = Public customer service line (anyone can call)

nginx-expose.yaml:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-svc
spec:
  type: NodePort     # Make it PUBLIC!
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 80
      nodePort: 30144  # Public number
```

Q) What's the difference between normal service and expose.yaml?

- **Normal**: Only people in the city can call
- **NodePort**: Anyone in the world can call!

Test the Public Number:

```
kubectl apply -f nginx-expose.yaml
curl <ANY-NODE-IP>:30144
```

Testing from Another Pod 👥

Q) What do you mean by "using a service to access workload from another pod"?

Let's create a customer and have them order pizza!

Step 1: Create a Customer

```yaml
# client.yaml
apiVersion: v1
kind: Pod
metadata:
  name: clientpod
spec:
  containers:
  - name: clientpod
    image: busybox:1.28.3
    args:
    - /bin/sh
    - -c
    - sleep 30000  # Stay alive for testing
```

Step 2: Go Inside the Customer's Head

```
kubectl apply -f client.yaml
kubectl exec -it clientpod -- /bin/sh
```

Step 3: Look Up Pizza Shop in Phone Book

```
nslookup nginx-svc
```

This finds the permanent phone number

Step 4: Call the Pizza Shop

```
wget -q -O - nginx-svc:8080
```

This successfully orders pizza! 🍕

The Magic: Even if actual pizza shops change, the phone book number stays the same!

---

4. Kubernetes Storage 📦

emptyDir = Shopping Cart 🛒

What it is:

- Temporary storage while shopping
- **Gets emptied when you leave** (pod dies)

empty-pod.yaml:

```
apiVersion: v1
kind: Pod
metadata:
  name: em
spec:
  containers:
  - image: ubuntu
    name: test-container
    volumeMounts:
      - mountPath: /cache  # Where to put cart in
shop
        name: cache-volume
    args:
      - /bin/sh
      - -c
      - sleep 30000
  volumes:
  - name: cache-volume
    emptyDir: {}  # This is the shopping cart
```

Test it:

```
kubectl apply -f empty-pod.yaml
kubectl exec -it em -- /bin/bash

# Inside the container:
echo "Hello from lab!" > /cache/hello.file
```

Q) Why `cat > hello.file<<EOF` ?

- It's a quick way to create a file with content
- Like writing a note and putting it in your cart

hostPath = Writing on the Wall 🏗️

What it is:

- Store data directly on the building
- Stays even if your shop closes
- **Problem**: If shop moves to new building, writing isn't there!

hostPath-pod.yaml:

```
volumes:
- name: hp-volume
  hostPath:
    path: /testdir    # Write on this wall
    type: Directory
```

PV & PVC = Renting Storage Units 🗄️

- **PV (PersistentVolume)** = Actual storage units in city
- **PVC (PersistentVolumeClaim)** = "I want to rent a storage unit!"

Check Available Units:

```
kubectl get pv    # Show available storage units
kubectl get pvc   # Show rental requests
```

Q) What's the difference between `kubectl get pv` and `kubectl get pvc` ?

- `get pv` = See all storage units in city
- `get pvc` = See who wants to rent units

Using a Storage Unit in Your Shop:

```
volumes:
  - name: pvc-volume
    persistentVolumeClaim:
      claimName: cce-efs-import-mypvc  # Your
storage unit
```

Best part: Data survives even if shop closes and reopens!

StatefulSets Explained Like Never Before! 🏦

The Problem: Why Food Trucks Don't Work for Banks 🚚

Imagine you have **3 food trucks** (Regular Pods):

- Taco Truck #1
- Taco Truck #2
- Taco Truck #3

What's the problem?

- They're all **identical** - same tacos, same menu
- If Taco Truck #1 breaks down, a **new identical one** replaces it
- Customers don't care which truck they get tacos from
- **Perfect for tacos, TERRIBLE for banks!**

What ARE StatefulSets? 🏢

StatefulSets are for businesses that NEED to remember things:

Think of **3 bank branches**:

- 🏦 Bank Branch #1 (Downtown)
- 🏦 Bank Branch #2 (Uptown)
- 🏦 Bank Branch #3 (Suburbs)

Each branch is SPECIAL:

- ✅ **Unique name** that NEVER changes
- ✅ **Own vault** (storage) that NEVER mixes with others
- ✅ **Opens/closes in specific order** (very organized!)

Real examples:

- Databases (MySQL, PostgreSQL)
- Where each instance has **DIFFERENT data**
- Applications that need to **remember who they are**

Headless Service = Direct Employee Phone Lines 📞

Regular Service (Main Company Number)

> 📞 `Call: 1-800-COMPANY`
>
> 👋 `You get: ANY available customer service rep`
>
> 🤷 `You don't care WHO answers`

Headless Service (Direct Extensions)

> 📞 Call: Bob's direct line (x101)
>
> 👋 You get: BOB specifically
>
> 📞 Call: Alice's direct line (x102)
>
> 👋 You get: ALICE specifically

Why do banks need this?

- You need to talk to **Bank Manager Bob** specifically
- You need to talk to **Loan Officer Alice** specifically
- Each has **different responsibilities and information**

The Magic File:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: headless
spec:
  clusterIP: None  # ⭐ THIS makes it HEADLESS!
  selector:
    app: tomcat
```

Creating Our Organized Bank System 🏦

Our Bank Blueprint:

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: tomcat-statefulset
spec:
  serviceName: tomcat-headless  # Use direct phone
lines
  replicas: 3                   # Create 3 bank
branches

  volumeClaimTemplates:         # ⭐ Each branch
gets OWN vault!
  - metadata:
      name: www-storage
    spec:
      resources:
        requests:
          storage: 5Gi
```

What Gets Created:

```
tomcat-statefulset-0   www-storage-tomcat-statefulset-0

tomcat-statefulset-0.tomcat-headless...


tomcat-statefulset-1   www-storage-tomcat-statefulset-1

tomcat-statefulset-1.tomcat-headless...


tomcat-statefulset-2   www-storage-tomcat-statefulset-2

tomcat-statefulset-2.tomcat-headless...
```

Key Point: Each branch has its **OWN storage** that **never gets mixed up** with others!

The MAGIC of StatefulSets ✨

What if Disaster Strikes? 🔥

Bank Branch #1 burns down!

```
kubectl delete pod tomcat-statefulset-0
kubectl get pod
```

The MAGIC Happens:

1. **Same Name**: New branch opens as `tomcat-statefulset-0` (NOT a random name!)
2. **Same Vault**: Gets back the EXACT same vault: `www-storage-tomcat-statefulset-0`
3. **All Money Safe**: 💰 All customer data and money is STILL THERE!

This is IMPOSSIBLE with regular food trucks!

Food Truck vs Bank Branch:

Truck #1 breaks	New truck with RANDOM name	New branch with SAME name
Storage	Loses all ingredients	Keeps ALL customer money
Identity	Just another taco truck	Still Bank Branch #1

Testing Our Direct Phone System 📞

Step 1: Create a Phone Tester

```
# Create a special phone book tester
kubectl run dnsutils --image=tutum/dnsutils --
command -- sleep infinity
```

Step 2: Test the Direct Lines

```
# Look up all the direct phone numbers
kubectl exec dnsutils -- nslookup headless
```

What you'll see:

```
🏦 tomcat-statefulset-0.tomcat-headless... → 10.244.1.10
🏦 tomcat-statefulset-1.tomcat-headless... → 10.244.1.11
🏦 tomcat-statefulset-2.tomcat-headless... → 10.244.1.12
```

Now you can call each branch DIRECTLY!

Real-World Examples 🎯

When to Use StatefulSets:

Database Cluster Each database has different data

Message Queue Each node has different messages

Cache Cluster Each instance has different cached data

File Storage Each pod manages different files

When to Use Regular Deployments:

Web Servers All serve same website

API Servers All handle same requests

Microservices All provide same service

Simple Summary 🎪

Food Trucks (Regular Pods) 🚚

- **All identical**
- **Interchangeable**
- **Lose everything if replaced**

- **Good for**: Websites, APIs, stateless apps

Bank Branches (StatefulSets) 🏦

- **Each is unique**
- **Keep their identity**
- **Preserve their data**
- **Good for**: Databases, queues, stateful apps

The Golden Rule:

> Use StatefulSets when your application needs to **remember who it is** and **what data it has**!

Quick Test ❓

Question: Should you use StatefulSet for a MySQL database cluster?
Answer: ✅ **YES!** Each MySQL instance has different data and needs to keep its identity!

Question: Should you use StatefulSet for a website serving static pages?
Answer: ❌ **NO!** All web servers are identical and interchangeable!

The beauty of StatefulSets is that they handle all the complexity of managing stateful applications while keeping everything organized and reliable! 🎉

Quick Summary Cheat Sheet 🎯

Pod Temporary shop Pop-up store

Service Permanent phone book Company phone number

NodePort Public phone number Customer service line

emptyDir Shopping cart Temporary storage

PV/PVC Storage unit rental Long-term storage

Deployment Interchangeable shops Food trucks

StatefulSet Organized branches Bank branches

Headless Service Direct extensions Employee direct lines

Key Takeaways 💡

1. **Pods are temporary** - they come and go with new IPs
2. **Services are permanent** - they provide stable access points
3. **Storage choices matter** - temporary vs permanent vs shared
4. **StatefulSets keep identity** - crucial for databases
5. **Kubernetes manages complexity** - you just describe what you want!

The beauty of Kubernetes is that it handles all the complicated stuff behind the scenes. You just describe your city, and Kubernetes builds and manages it for you! 🎉

Remember: Whether you're running a simple website or a complex database system, Kubernetes has the right tools for the job. Start simple, understand the basics, and gradually work your way up to more advanced concepts! 🚀