Here’s a refined version of your notes, structured for clarity and readability. Let me know if you’d like any modifications! 🚀

**Kubernetes ConfigMaps and Secrets - Notes**

**1. Introduction**

* Kubernetes uses **ConfigMaps** and **Secrets** to manage configuration data.
* They help **separate configurations** from application code.
* **ConfigMaps** store **non-sensitive** data (e.g., database ports, URLs).
* **Secrets** store **sensitive** data (e.g., passwords, API keys).

**2. Why Use ConfigMaps?**

* Applications require **configuration settings** (e.g., database port, connection type).
* **Hardcoding configurations is bad practice**—any change requires **redeployment**.
* **ConfigMaps allow**:
  + Storing configurations **externally** from the app.
  + Updating configurations **without rebuilding containers**.

**Example:**

A backend app needs:

* DB\_PORT=3306
* DB\_CONNECTION\_TYPE=TCP

Instead of hardcoding them, use a **ConfigMap**.

**3. Why Use Secrets?**

* Some data is **sensitive** (e.g., DB\_USERNAME, DB\_PASSWORD).
* **ConfigMaps store data in plaintext** → Not safe for secrets.
* **Secrets solve this by**:
  + Encrypting data **at rest** in etcd.
  + Restricting access using **RBAC (Role-Based Access Control)**.

**4. ConfigMaps vs. Secrets**

| **Feature** | **ConfigMap** | **Secret** |
| --- | --- | --- |
| **Purpose** | Non-sensitive data | Sensitive data |
| **Storage** | Plaintext in etcd | Encrypted in etcd |
| **Access Control** | Normal RBAC | Strict RBAC |
| **Use Case** | Database port, URLs | Passwords, API tokens |

**5. Working with ConfigMaps**

**Step 1: Create a ConfigMap**

# configmap.yaml

apiVersion: v1

kind: ConfigMap

metadata:

name: db-config

data:

DB\_PORT: "3306"

Apply it:

kubectl apply -f configmap.yaml

**Step 2: Use ConfigMap in a Pod**

# deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: sample-app

spec:

template:

spec:

containers:

- name: app

env:

- name: DB\_PORT

valueFrom:

configMapKeyRef:

name: db-config

key: DB\_PORT

* The pod now has DB\_PORT=3306 as an **environment variable**.

**Issue: ConfigMap Updates Don't Reflect in Running Pods**

* If DB\_PORT is updated to 3307, the running pods **won't update automatically**.
* **Solution:** Mount the ConfigMap as a **file**.

**Step 3: Mount ConfigMap as a Volume**

spec:

volumes:

- name: db-config-volume

configMap:

name: db-config

containers:

- volumeMounts:

- name: db-config-volume

mountPath: /etc/config

* The **file** /etc/config/DB\_PORT contains the value.
* Changes in the ConfigMap **auto-reflect** in the pod (after a short delay).

**6. Working with Secrets**

**Step 1: Create a Secret**

kubectl create secret generic db-secret --from-literal=DB\_PASSWORD=SuperSecret123

* **Secrets are stored in Base64** (not fully secure).
* For **better security**, use **Sealed Secrets** or **HashiCorp Vault**.

**Step 2: Use Secret in a Pod**

env:

- name: DB\_PASSWORD

valueFrom:

secretKeyRef:

name: db-secret

key: DB\_PASSWORD

Or mount it as a **volume**:

volumes:

- name: secret-volume

secret:

secretName: db-secret

**7. Best Practices**

✅ **ConfigMaps** for **non-sensitive** data.  
✅ **Secrets** for **sensitive** data.  
✅ **Use Volume Mounts** for dynamic updates.  
✅ **Restrict RBAC permissions** on Secrets.  
✅ **Encrypt Secrets externally** (e.g., HashiCorp Vault, Sealed Secrets).

**8. Hands-on Exercise**

🔹 **Create a Secret** for DB\_PASSWORD.  
🔹 **Mount it in a Pod** using:

* Environment variables.
* Volume mounts.  
  🔹 **Test if updates reflect automatically**.

**9. Further Learning**

📌 **Kubernetes Docs:**

* [ConfigMaps](https://kubernetes.io/docs/concepts/configuration/configmap/)
* [Secrets](https://kubernetes.io/docs/concepts/configuration/secret/)

📌 **Security Tools:**

* **HashiCorp Vault** for secret management.
* **Sealed Secrets** to encrypt Secrets in Git.

**Thanks for reading!**

👉 **Follow for more DevOps & Kubernetes content! 🚀**

# \*\*Kubernetes ConfigMaps and Secrets - Notes\*\*

## \*\*1. Introduction\*\*

- ConfigMaps and Secrets in Kubernetes are used to store configuration data.

- They help decouple configuration from application code.

- ConfigMaps store \*\*non-sensitive\*\* data.

- Secrets store \*\*sensitive\*\* data (like passwords, API keys).

---

## \*\*2. Why ConfigMaps?\*\*

- Applications need configuration (e.g., database port, connection type).

- Hardcoding configurations is \*\*bad practice\*\* (changes require redeployment).

- ConfigMaps allow:

- Storing configuration outside the application.

- Updating configurations without rebuilding containers.

### \*\*Example:\*\*

- A backend app needs:

- `DB\_PORT=3306`

- `DB\_CONNECTION\_TYPE=TCP`

- Instead of hardcoding, store these in a \*\*ConfigMap\*\*.

---

## \*\*3. Why Secrets?\*\*

- Some data is \*\*sensitive\*\* (e.g., `DB\_USERNAME`, `DB\_PASSWORD`).

- Storing sensitive data in ConfigMaps is \*\*unsafe\*\* because:

- ConfigMap data is stored in \*\*plaintext\*\* in `etcd`.

- Anyone with `kubectl` access can read it.

- \*\*Secrets solve this by:\*\*

- Encrypting data at rest in `etcd`.

- Restricting access via \*\*RBAC\*\*.

---

## \*\*4. Difference Between ConfigMaps and Secrets\*\*

| Feature | ConfigMap | Secret |

|------------------|----------|--------|

| \*\*Purpose\*\* | Non-sensitive data | Sensitive data |

| \*\*Storage\*\* | Plaintext in `etcd` | Encrypted in `etcd` |

| \*\*Access Control\*\* | Normal RBAC | Strict RBAC |

| \*\*Use Case\*\* | DB port, URLs | Passwords, tokens |

---

## \*\*5. Demo: Using ConfigMaps\*\*

### \*\*Step 1: Create a ConfigMap\*\*

```yaml

# cm.yaml

apiVersion: v1

kind: ConfigMap

metadata:

name: test-cm

data:

DB\_PORT: "3306"

```

Apply:

```sh

kubectl apply -f cm.yaml

```

### \*\*Step 2: Use ConfigMap in a Pod\*\*

```yaml

# deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: sample-app

spec:

template:

spec:

containers:

- name: app

env:

- name: DB\_PORT # Environment variable name

valueFrom:

configMapKeyRef:

name: test-cm # ConfigMap name

key: DB\_PORT # Key in ConfigMap

```

- Now, the pod has `DB\_PORT=3306` as an environment variable.

### \*\*Problem: Updating ConfigMaps\*\*

- If you update `cm.yaml` (e.g., change `DB\_PORT` to `3307`), pods \*\*won’t auto-update\*\*.

- \*\*Solution:\*\* Use \*\*Volume Mounts\*\* instead of environment variables.

### \*\*Step 3: Mount ConfigMap as a File\*\*

```yaml

# deployment.yaml

spec:

volumes:

- name: db-config

configMap:

name: test-cm

containers:

- volumeMounts:

- name: db-config

mountPath: /opt

```

- Now, `/opt/DB\_PORT` contains the value.

- Changes in ConfigMap \*\*auto-reflect\*\* in the pod (after a short delay).

---

## \*\*6. Demo: Using Secrets\*\*

### \*\*Step 1: Create a Secret\*\*

```sh

kubectl create secret generic test-secret --from-literal=DB\_PASSWORD=abc123

```

- Secrets are stored in \*\*Base64\*\* (not secure by default).

- For better security, use \*\*Hashicorp Vault\*\* or \*\*Sealed Secrets\*\*.

### \*\*Step 2: Use Secret in a Pod\*\*

```yaml

env:

- name: DB\_PASSWORD

valueFrom:

secretKeyRef:

name: test-secret

key: DB\_PASSWORD

```

Or mount as a file:

```yaml

volumes:

- name: db-secret

secret:

secretName: test-secret

```

---

## \*\*7. Key Takeaways\*\*

- \*\*ConfigMaps\*\* → Non-sensitive data (e.g., ports, URLs).

- \*\*Secrets\*\* → Sensitive data (e.g., passwords, tokens).

- \*\*Best Practices:\*\*

- Avoid hardcoding configurations.

- Use \*\*Volume Mounts\*\* for dynamic updates.

- Restrict \*\*RBAC\*\* for Secrets.

- Use \*\*external encryption\*\* (Vault) for better security.

---

## \*\*8. Homework\*\*

- Try creating a \*\*Secret\*\* for `DB\_PASSWORD`.

- Mount it in a pod using:

- Environment variables.

- Volume mounts.

- Test if updates reflect automatically.

---

## \*\*9. Further Learning\*\*

- \*\*Kubernetes Official Docs:\*\*

- [ConfigMaps](https://kubernetes.io/docs/concepts/configuration/configmap/)

- [Secrets](https://kubernetes.io/docs/concepts/configuration/secret/)

- \*\*Advanced Security:\*\*

- \*\*Hashicorp Vault\*\* for secret management.

- \*\*Sealed Secrets\*\* for encrypting Secrets in Git.

---

### \*\*Thanks for watching!\*\*

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