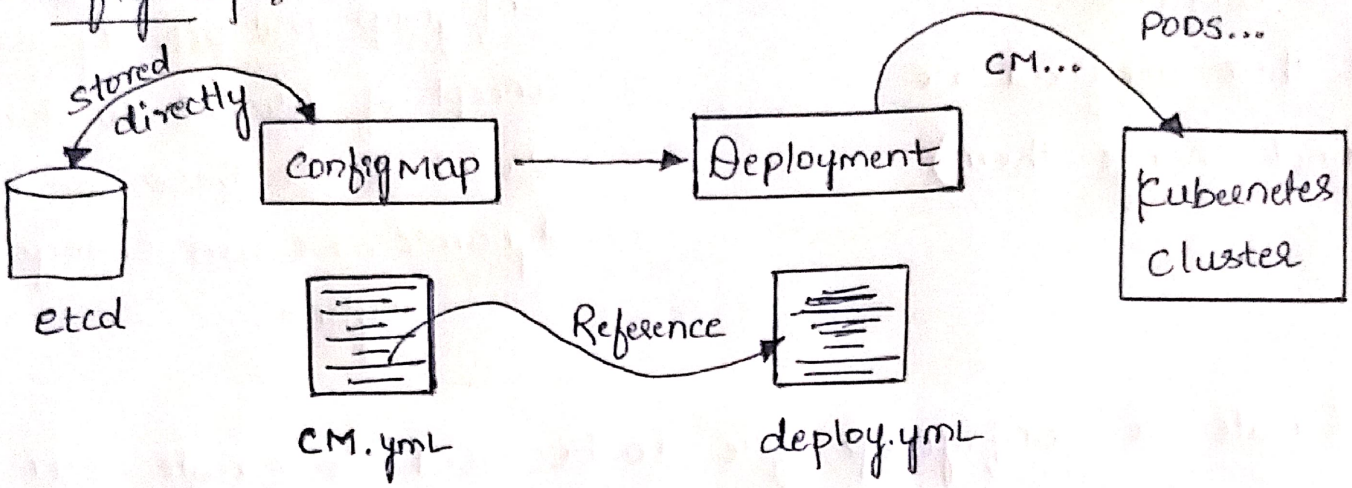
# Day-17

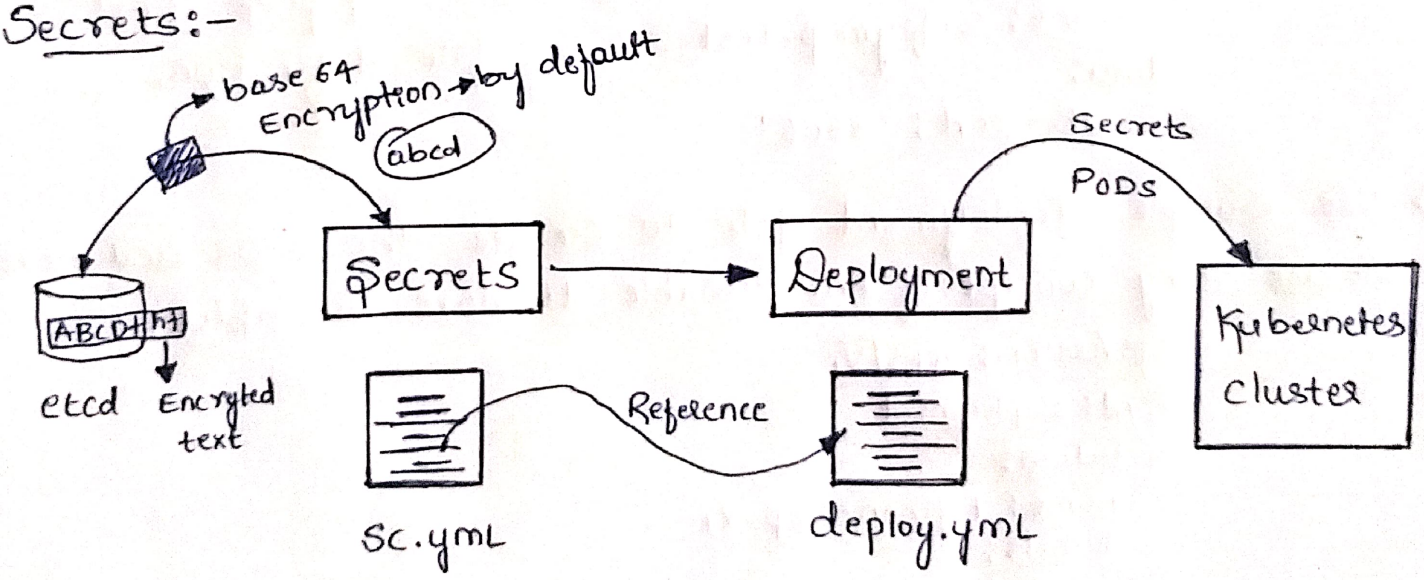
## - Handson ConfigMap And Secrets

### ConfigMap:-

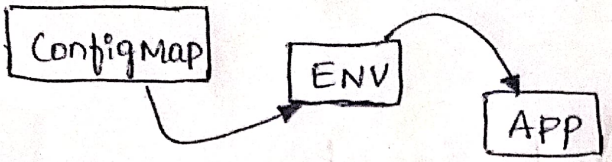



- Usage of configmap is very useful in realtime like DB_PORT, DB_ENDPOINT (Not DB username & password)

### Secrets:-

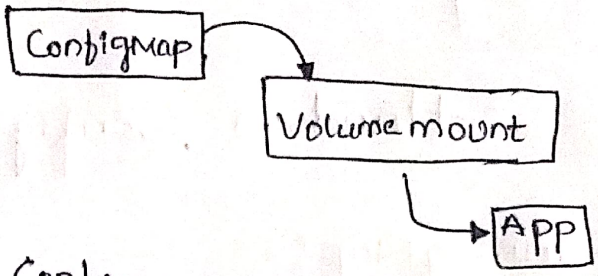



- The usage of Secrets are very specific to the sensitive data such as DB_PWD, DB_USER and may TLS and keys.




Directly creating Env variables on container

(But these are fixed we cannot change them)




ConfigMap will actually use concept of volumes to store variables so these can be dynamic, we can change at anytime.

1) Create a configmap file to be used to create our configmap use below file to generate configmap.

```
apiVersion: v1
kind: Configmap
metadata:
  name: configmap-test
data:
  db-port: "3306"
```

→ Basic configmap object file definition.

2) Use sample deployment file to create the PODS and here we are using concept of Env variable to store variables.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: python-app-v1
  Labels:
    app: python-app-v1
spec:
  replicas: 2
  selector:
    matchLabels:
      app: python-app-v1
  template:
    metadata:
      Labels:
        app: python-app-v1
```

```
Spec:
  Containers:
  - name : python-app-v1
    image: abhishekf5/python-sample-app-demo:v1
    env
     - name: DB_PORT
       valueFrom:
         configMapKeyRef:
          name: configmap-test
          key: db-port
       Ports:
         - containerPort : 8000
```

→ Using concept of Environment Variables.

3) Check the pod status after coming into running state you can go into the container check if Env variable DB_PORT is created or not.

```
# kubectl get po
# kubectl exec -it python-app-v1-56ff87c94c-hxtcz --/bin/bash
# env | grep DB
   DB_PORT=3306
```

→ Env variable is created successfully.

```
# kubectl apply -f cm.yml
# kubectl describe cm/configmap-test
  Name :          configmap-test
  Namespace:      default
  Labels:         <none>
  Annotations:    <none>

  Data
  ====
  db-port :  → key
  ----
  3306 → Configmap → Value.
  BinaryData
  ====
  Events: <none>
```

Q. What if the DB_PORT has been changed to something else?

This won't work and you need to create new pods.

→ To solve this we can have volumes in place.

4) Now, Modify the deployment accordingly below.

```
apiVersion: apps/v1
Kind: Deployment
metadata:
  name: python-app-v1
  labels:
    app: python-app-v1
Spec:
  replicas: 2
  Selectors:
    matchLabels:
      app: python-app-v1
  template:
    metadata:
      Labels:
        app: python-app-v1
    Spec:
      containers:
      - name: python-app-v1
        image: abhishekfs/python-sample-app-demo:v1
        VolumeMounts:
        - name: db-connection
          mountPath: /opt
      ports:
      - containerport: 8000
    Volumes:
      - name: db-connection
        configMap:
          name: configmap-test
```

→ Mounting volume to Container.

→ Creating Volume and linking configmap to volume.

5) Now modify the DB_PORT to "3380" and apply changes to configmap.

```
apiVersion: v1
Kind : Configmap
metadata :
  name : Configmap-test
data :
  db_port : "3380"
```

6) Check the below status for more details.

# kubectl apply -f .\cm.yml → updated configmap.

# kubectl describe cm/configmap-test

```
Name :          Configmap-test
Namespace :     default
Labels :        <none>
Annotations :   <none>

Data
====
db-port :
----
3380 → Value is updated to configmap → Volume → /opt/file

BinaryData
====

Events : <none>
```

# kubectl exec -it python-app-v1-56d988b8fd-t2jjh -- /bin/bash
cat /opt/db-port | more

3380 → Updated PORT without pod restart/ new POD Creation.

1) Secrets are also works same way except the use cases are different just will have a look on secrets as well:

```
apiVersion: v1
Kind: Secret                    → Create a Secrets file
metadata :
   name : Secret-basic-auth
type : kubernetes.io/basic-auth
StringData :
  Username : admin              → UserID/password → Sensitive data
  password : top-secret
```

2) Create the Secrets and check the details of secrets.

$ kubectl get secret secret-basic-auth -o yaml

```
apiVersion: v1
data:
  password: dBANLOP$MU-IEP   → base64 encoded text for password/username
  username: Happy
kind: Secret
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      - - - -
  creationTimestamp: "2024-01-01"
  name: secret-basic-auth
  namespace: default
  resourceVersion: "880"
  uid: 741c14307-6cd7-4419-8545-85b916e3ee50
type: kubernetes.io/basic-auth
```

Kubernetes do some encryption for secrets but in serious note it is very weak to crack so there are external integration you can do to your secrets than using default encryption.

3) Even I took the encryption text and decoded to original username / password.

$ echo -n Happy | base64 --decode (admin)

$ echo -n dBANLOP$MU-IEP | base64 --decode (top-Secret)  → Same credentials that we have injected in Secrets

This is a very good handson on configmaps, Secrets and usage of Volumemounts.