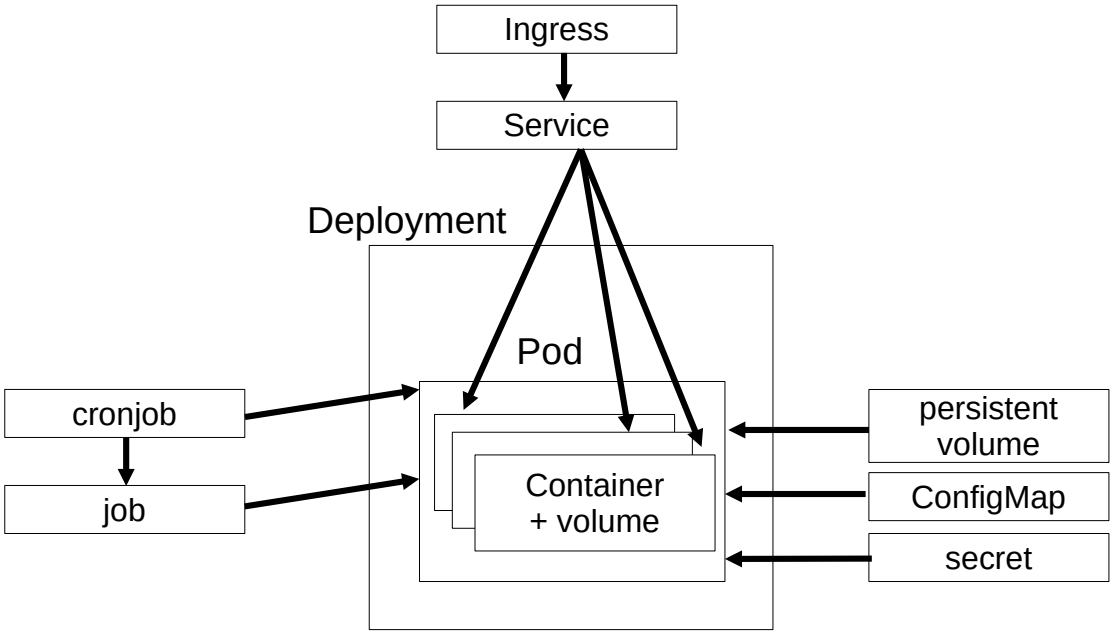
Kubernetes resources

- **Pod:** one or more containers that run the application
- **Deployment:** used to run a scalable application
- **Volume:** represents storage, either as a part of the Pod definition, or as its own resource
- **Service:** a policy that provides access to the application Pods
- **Jobs:** tasks that run as a Pod in the Kubernetes cluster
- **CronJobs:** used to run Jobs on a specific shedule



Kubernetes application creation

GitOps Code Requirements:

- Application updates should happen through the same code as application creation
- Code mus be idempotent

- To meet these requirements in a Kubernetes environment, changes should be applied the declarative way
- The **kubectl apply** command is what should be used thoughout

Running Applications the Declarative Way

- An easy way to run an application in Kubernetes is **kubectl create deploy testapp --image=testimage --replicas=3** and then to view **kubectl get all**
- Although useful, this way doesn't work well in a GitOps environment
- In GitOps the application is defined in a YAML manifest file, and Kubernetes uses an operator to pick up and apply changes in the YAML file automatically
- To generate the YAML manifest file, use **kubectl create deploy testapp --image=testimage --replicas=3 --dry-run=client -o yaml > testapp.yaml**
- Next, push the YAML file to the Git repository and have the Kubernetes operator use **kubectl apply -f testapp.yaml** to apply the code to the cluster

- After defining YAML files, kubectl create -f myapp.yaml can be used to create the application. This command only works if the application doesn't yet exist
- **kubectl apply -f testapp.yaml** is recommended in GitOps
  - if the application doesn't yet exist, it will be updated
  - if the applicaiton already exists, modification will be applied
  - to preview modificaiton sthat wil be applied, use **kubectl diff -f testapp.yaml**
  - each time the application is updated, kubectl apply stores the configuration in the last-applied-configuration annotation, which allows **kubectl diff** to see any change in the manifest file

Kubernetes application creation

Example: Running Application in Declarative way

- **kubectl create deploy testserver --image=nginx**
  ya, it is not declarative, but it just to show, that if you run same command again...
- **kubectl create deploy testserver --image=nginx**
  ... appears message that this deployment already exists
- **kubectl delete deploy testserver**

Declarative way is

- **kubectl create deploy testserver --image=nginx --dry-run=client -o yaml > testserver.yaml**
- **kubectl apply -f testserver.yaml**
- **kubectl get deploy testserver -o yaml | less**
  Let's run same command again ...
- **kubectl apply -f testserver.yaml**
  ... and now message tells us that deployment configured

Example: Running Application

- **source <(kubectl completion bash)**
- **kubectl create deploy webserver --image=nginx --replicas=3 --dry-run=client -o yaml > webserver.yaml**
- **cat webserver.yaml**
- **kubectl apply -f webserver.yaml**
  To see whats happen
- **kubectl get all**
  Let's see pod
- **kubectl describe pod webserver[Tab]**
- edit webserver.yaml and add into **metadata: annotations: environment: qa**
- **kubectl diff -f webserver.yaml**
- **kubectl apply -f webserver.yaml**

Kubernetes accessing Applications

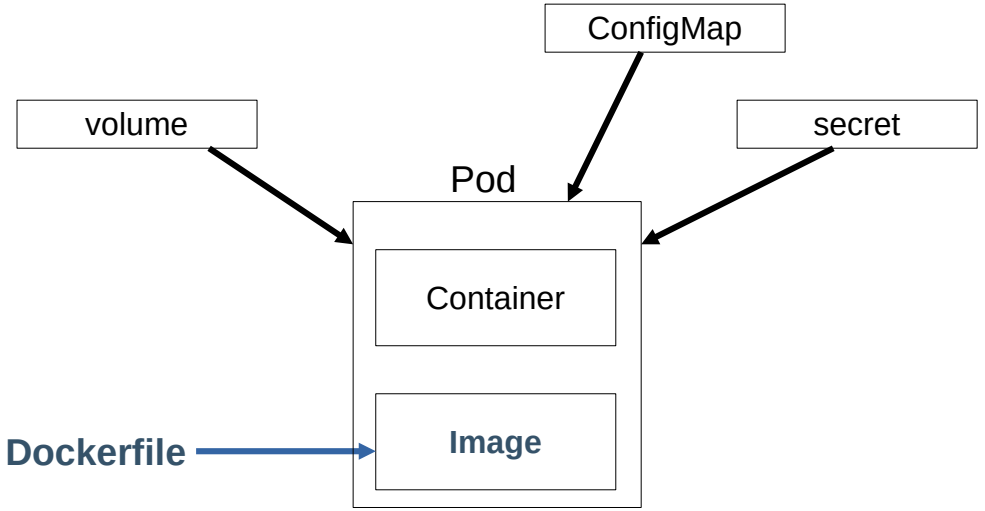
After running an application, it doesn’t automatically become accessible. To access it, different solutions can be used:

- **kubectl port-forward** is for testing puposes only, and exposes a port on the host that runs the **kubctl** client
- Services are Kubernetes objects that provide load balancing to the different Pod instances
- Ingress is an additional Kubernetes object that provides access to HTTP and HTTPS resources

Example: using kubectl port-forward

- Let’s see deployment running with usage of selector-label...
- **kubectl get all --selector app=webserver**
- Then try to check how port-forward works...
- **kubectl port-forward webserver-[Tab] 8080:80**
- **curl localhost:8080**

Decoupling Configuration Files and Variables from Applications



Instruments to decouple infromation from the Application, running in Pod

ConfigMaps. Variables

- ConfigMaps can be used to store variables and configuration in the cloud
- The maximal size is 1MB, if configuration is bigger, it should be provided using volumes
- ConfigMaps can be used in two ways
  - To provide variables
  - To provide configuration files
- The ConfigMap will be addressed from a Pod according to how it is used
  - ConfigMaps containing variables are accessed usin envFrom
  - ConfigMap containing configuration files are mounted
- Secrets are base64 encoded ConfigMaps

Providing Variables with ConfigMaps

- While crating a ConfigMap with **kubectl create cm**, variables can be provided in two ways
  - Using **--from-env-file: kubectl create cm --from-env-file=dbvars**
  - Using **--from-literal: kubectl create cm --from-literal=MYSQL_USER=anna**
- Notice that it's possible to use multiple **--from-literal**, you cannot use multiple **--from-env-file**
- After creating the ConfigMap, use **kubectl set env --from=configmap/mycm deploy/myapp** to use the ConfigMap in your Deployment

Example: Providing Variables with ConfigMaps

- **vim varsfile**
  MYSQL_ROOT_PASSWORD=password
  MYSQL_USER=anna
- **kubectl create cm mydbvars --from-env-file=varsfile**
- **kubectl describe cm mydbvars**
- **kubectl create deploy mydb --image=mariadb --replicas=3**
- **kubectl get all --selector app=mydb**
- **kubectl describe pod mydb-6784929872**
- **kubectl logs mydb-6784929872**
- **kubectl set env deploy mydb --from=configmap/mydbvars**
- **kubectl get all --selector app=mydb**
- **kubectl get deploy mydb -o yaml**

ConfigMaps. Configuration Files

- Configuration files are typically used to provide site-specific information to applications
- To store configuration files in the cloud, ConfigMap can be used
- Use kubectl create cm myconf --from=file=/my/file.conf
- If a ConfigMap is created from a directory, all files in that directory are included in the ConfigMap
- To use the configuration file in an application, the ConfigMap must be mounted in the application
- There is no easy, imperative way to mount ConfigMaps in applications

Mounting a ConfigMap in an Application

- Note: Generate the base YAML code, and add the ConfigMap mount to it later
- In the application manifest, define a volume using the ConfigMap type
- Mount this volume on a specific directory
- The configuration file will appear inside that directory

Example: Using a ConfigMap with a Configuration File

- **echo “hello world!” > index.html**
- **kubectl create cm myindex --from-file=index.html**
- **kubectl describe cm myindex**
- **kubectl create deploy myweb --image=nginx**
- **kubectl edit deployments.apps myweb**

  spec.template.spec
  volumes:
  - name: cmvol
  configMap:
  name: myindex
  -----
  spec.template.spec.containers
  volumeMounts:
  - mountPath: /usr/share/nginx/html
  name: cmvol
- **kubectl describe pd myweb-239857**
- **kubectl exec -it myweb-239857 00 cat /usr/share/nginx/html/index.html**