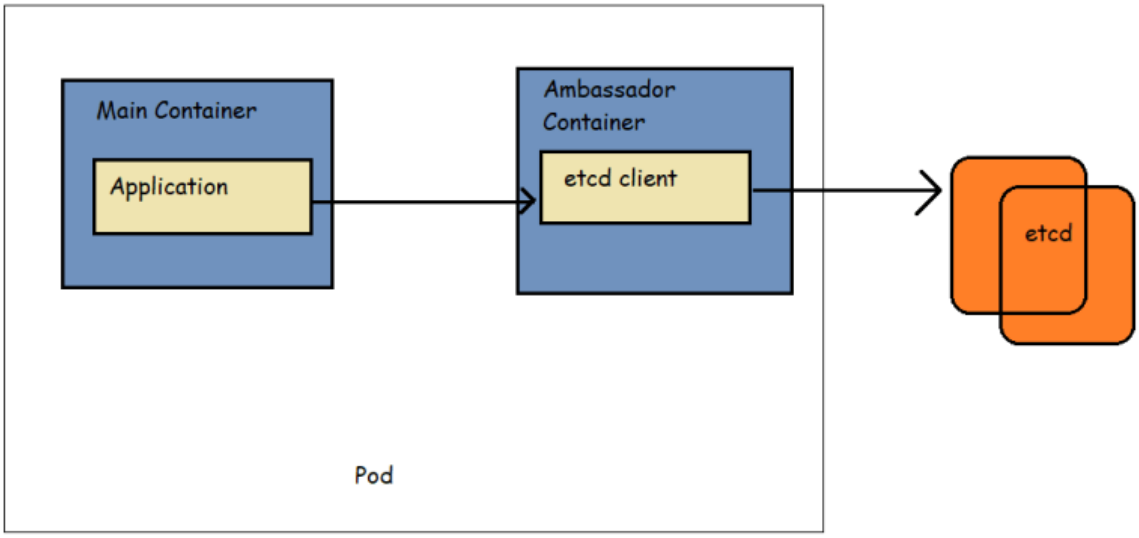
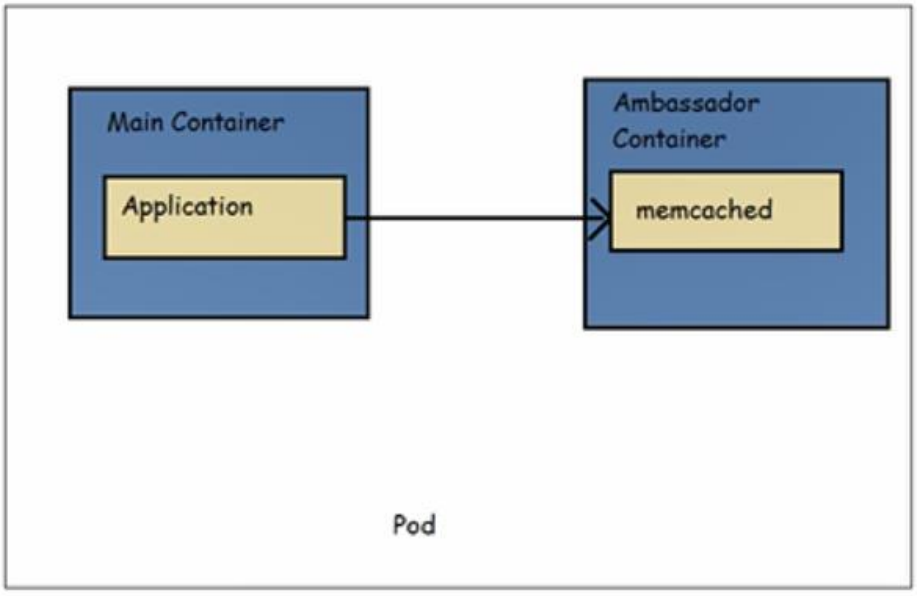
Ambassador

When we build our application, our application needs to communicate with another application. It acts as a one type of entry point for different application (Kind of proxy services)

- Overview: This pattern is a specialized sidecar responsible for hiding the complexity and providing the unified interface for accessing the services
- To demonstrate this pattern, let's say we use a cache for application. In the local developer environment, we want to use Memcached and in production we want to use etcd.
- So, we create an Ambassador container which accesses the Memcached and also the ambassador container which uses local Memcached
- Depending on the environment we can use the Ambassador container.





- Refer below for the sample created

```
---
apiVersion: v1
kind: Pod
metadata:
  app: web-app
  labels:
    app: web
    env: prod
spec:
  containers:
    - image: nginx
      name: main
      env:
        - name: CACHE_URL
          value: http://localhost:9009
      ports:
        - containerPort: 80
          protocol: TCP
    - name: ambassador
      image: qt/etcd-ambassdor
---
apiVersion: v1
kind: Pod
metadata:
  app: web-app
  labels:
    app: web
spec:
  containers:
    - image: nginx
      name: main
      env:
        - name: CACHE_URL
          value: http://localhost:9009
      ports:
        - containerPort: 80
          protocol: TCP
    - name: ambassador
      image: qt/mem-ambassdor
```

Configuration Patterns

- Every application needs to be configured and easy way to do it by storing configurations in code. This approach has side effect of configuration and code living and dying together.
- We still need the flexibility to adapt configuration without recreating application image.

EnvVar Conifiguration

- Config Map(https://kubernetes.io/docs/concepts/configuration/configmap/)
- Use environmental variables is easier way to setup configuration for simple use cases
- Environmental variables are set only before the application starts and we cannot change them later
- Refer Below for the manifest

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: sample-config-map
data:
  # property-like keys; each key maps to a simple value
  username: sqladmin
  password: rootrootroot
  port: "3306"
```

```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: my-main-app
spec:
  containers:
    - image: nginx
      name: main
      env:
        - name: DB_USER
          valueFrom:
            configMapKeyRef:
              name: sample-config-map
              key: username
        - name: DB_PASSWORD
          valueFrom:
            configMapKeyRef:
              name: sample-config-map
              key: password
```

```
PS D:\khajaclassroom\ExpertK8s\envvar> kubectl apply -f .\sampleconfigmap.yaml
configmap/sample-config-map created
PS D:\khajaclassroom\ExpertK8s\envvar> kubectl apply -f .\sampleenv.yaml
pod/my-main-app created
PS D:\khajaclassroom\ExpertK8s\envvar> kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
my-main-app   1/1     Running   0          6s
PS D:\khajaclassroom\ExpertK8s\envvar> kubectl exec -it my-main-app -- /bin/bash
root@my-main-app:/# echo $DB_USER
sqladmin
root@my-main-app:/# echo $DB_PASSWORD
rootroot
```

```
PS D:\khajaclassroom\ExpertK8s\envvar> kubectl apply -f .\sampleconfigmap.yaml
configmap/sample-config-map configured
PS D:\khajaclassroom\ExpertK8s\envvar> kubectl exec -it my-main-app -- /bin/bash
root@my-main-app:/# echo $DB_PASSWORD
rootroot
```

In Above approach variable would be configured during creation of pod, and once pod is created we can't change variable value.

Configuration Resource

- One significant disadvantage of the EnvVar Configuration pattern is that it's suitable for only a handful of variables and simple configurations.
- Often, it's better to keep all the configuration data in a single place.
- Kubernetes has dedicated Configuration Resources that are more flexible than pure environment variables
- These are ConfigMap and Secret Objects for general-purpose configuration and sensitive data respectively
- Once a config Map is creating and holding data, we can use the keys of Config Maps in two ways
    - As a reference for environmental variables. Key is environmental variable
    - As files that are maaped to a volume mounted in a Pod. Key is file name
- The file mounted ConfigMap Volume is updated when the ConfigMap is update via k8s api.
- Refer below for the ConfigMap as volume mount.

```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: my-second-app
spec:
  containers:
    - image: nginx
      name: second
      volumeMounts:
        - name: config-volume
          mountPath: /config
  volumes:
    - name: config-volume
      configMap:
        name: sample-config-map
```

Initial file:

3 envvar/sampleconfigmap.yaml

```
@@ -1,9 +1,10 @@
    apiVersion: v1
    kind: ConfigMap
    metadata:
      name: sample-config-map
    data:
      # property-like keys; each key maps to
      username: sqladmin
-     password: rootrootroot
      port: "3306"
```

Later changes:

```
  apiVersion: v1
  kind: ConfigMap
  metadata:
    name: sample-config-map
  data:
    # property-like keys; each key maps to a simple value
    username: sqladmin
+   password: India*123
    port: "3306"
+   otheruser: qtdevops
```