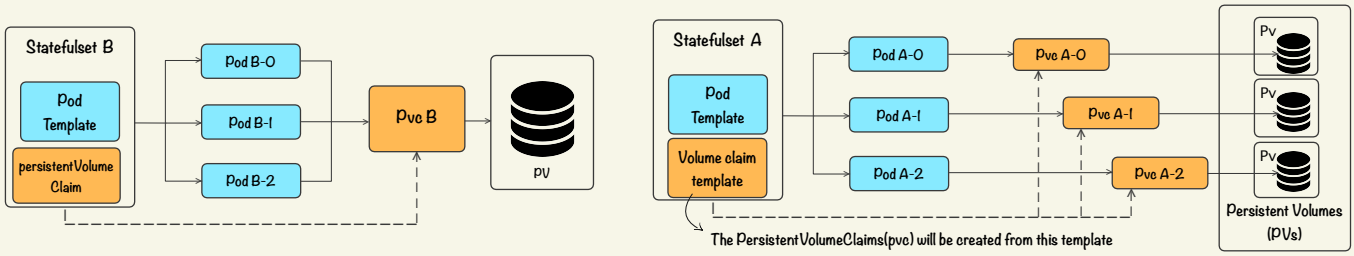
StatefulSets can use two types of storage

Shared Storage

All Pods in the StatefulSet share the same storage volume. Data is available to all Pods.
Good for things like caches, tmp files etc.
Specify a PersistentVolumeClaim template in the sfs spec. All Pods will get a clone of this PVC.
data can be corrupted if multiple Pods write to the same files

Dedicated Storage

Each Pod gets its own PersistentVolume. Data is isolated between Pods.
Good for databases, unique files etc.
Don't specify volumeClaimTemplates. StatefulSet will create a PVC for each Pod
Updating Pods is harder with dedicated storage, may need to coordinate Pod termination to avoid data loss




```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
spec:
  serviceName: mysql-hs
  replicas: 3
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:latest
        env:
          - name: MYSQL_ROOT_PASSWORD
            value: "yourpassword"
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: mysql-persistent-storage
          mountPath: /var/lib/mysql
      volumes:
      - name: mysql-persistent-storage
        persistentVolumeClaim:
          claimName: mysql-pvc
```

The volumeMounts and volumes are defined in the pod template section of the StatefulSet manifest, which means that they will be shared by all pods created by the StatefulSet

serviceName field specifies the name of the Headless Service that controls the network identity of the StatefulSet's Pods and it is a mandatory filed

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mysql-hs
spec:
  ports:
    - port: 3306
  selector:
    app: mysql
  clusterIP: None
```

Pod template

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pvc
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: manual
  resources:
    requests:
      storage: 1Gi
```

The PVC must be created beforehand either manually or through some automated process

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-pv
  labels:
    type: local
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: manual
  nfs:
    path: /srv/nfs/kubedata/pv3
    server: 192.168.49.1
```

The PV and PVC are using the "manual" StorageClass. The PVC has requested a capacity of 1Gi and has been bound to the PV.

```
k get pv,pvc
NAME                                CAPACITY   ACCESS   MODES   RECLAIM POLICY   STATUS   CLAIM                 STORAGECLASS   AGE
persistentvolume/mysql-pv           1Gi        RWM              Retain           Bound    default/mysql-pvc     manual         5h27m

NAME                                      STATUS   VOLUME     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
persistentvolumeclaim/mysql-pvc-sts3      Bound    mysql-pv   1Gi        RWM            manual         5h29m
```

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql-hs
spec:
  selector:
    matchLabels:
      app: mysql
  serviceName: mysql
  replicas: 3
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:latest
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-secret
              key: root-password
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: mysql-shared-storage
          mountPath: /var/lib/mysql
  volumeClaimTemplates:
  - metadata:
      name: mysql-shared-storage
    spec:
      accessModes: [ "ReadWriteMany" ]
      storageClassName: google-storage
      resources:
        requests:
          storage: 10Gi
```

volumeClaimTemplates is specified at the StatefulSet level, not in the pod template

The "volumeClaimTemplates" field in a StatefulSet is used to define persistent volume claims (PVCs) that will be used by the pods in the set for their storage needs. When a pod is created or rescheduled, it will automatically create/claim one of these PVCs and use it for its persistent storage

If you set the storageClassName to the name of a StorageClass that is configured with a dynamic provisioner, Kubernetes will automatically create a new PV based on the specifications defined in the volumeClaimTemplates section

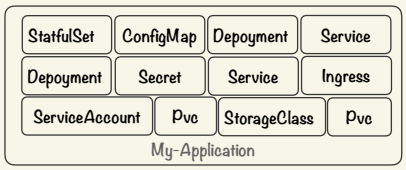
```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: google-storage
provisioner: kubernetes.io/gce-pd
....
```

How to deploy an application in k8s?

An application in Kubernetes typically consists of YAML files that define the k8s resources needed to run the application, such as Deployments, Services, ConfigMaps, and Secrets. You can deploy the application in Kubernetes manually by creating the YAML files and then using the `kubectl apply` command to create the Kubernetes resources on the cluster. Alternatively, you can use deployment tools like Kustomize, Helm, or the Helm Operator to automate the deployment process and simplify the creation of the YAML files. These tools provide a higher-level abstraction for managing Kubernetes resources and can make it easier to deploy and manage complex applications in Kubernetes.

Manual Deployment:

Manually deploying applications in Kubernetes involves creating YAML files that define the k8s resources needed to run the application, such as deployments, services, and config maps. You would then use the kubectl apply command to create those resources on the Kubernetes cluster.




```
kubectl apply -f deployment.yml
kubectl apply -f service.yml
kubectl apply -f statfulset.yml
...
kubectl apply -f serviceaccount.yml
```

This approach can be useful for simple applications or for users who prefer a more hands-on approach, but it can be time-consuming and error-prone for more complex applications

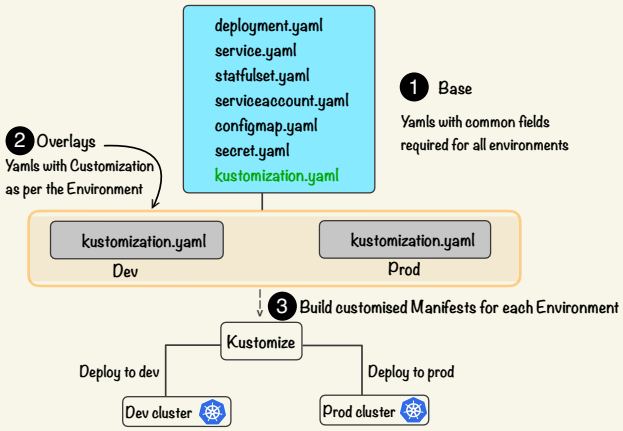
Kustomize

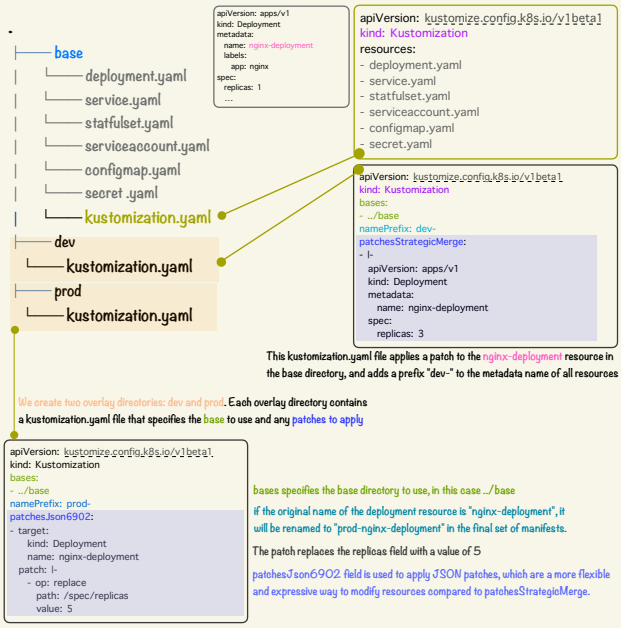
Kustomize is a tool for managing k8s manifest files using a declarative approach. It allows you to define a set of base manifests that define the desired state of your Kubernetes resources, and then apply changes using composition and customization. The basic workflow of Kustomize consists of the following steps:

Create a base directory containing your Kubernetes manifests. This directory represents the desired state of your application or environment

Define a kustomization.yaml file in the base directory. This file specifies the base resources to use, as well as any additional resources that should be added, modified, or removed

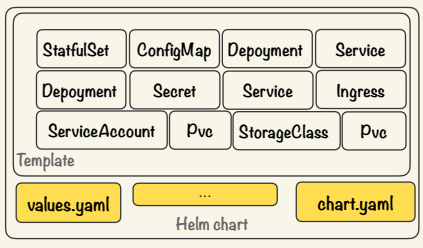
Create overlay directories for each environment or application variant, if needed. These overlay directories contain additional resources or modifications to apply on top of the base resources





Helm

Helm is a widely-used package manager for Kubernetes that simplifies the deployment and management of applications on a k8s cluster. It enables developers to package their applications as charts, which are reusable and shareable bundles that contain all the resources required to deploy an application on a Kubernetes cluster. With Helm, users can easily search for charts, install and upgrade applications, rollback changes, and manage dependencies through a straightforward command-line interface. Additionally, Helm supports versioning, which allows users to track changes to their applications over time and roll back to previous versions if necessary



Helm uses a packaging format called "charts". A chart is a collection of files that describe a related set of kubernetes resources.For example, instead of manually creating deployments, services, and other K8s objects, you can package these into a Helm chart. Then, anyone can easily deploy your application by installing the chart.

A Helm chart typically includes the following files:

Chart.yaml: This is the core file which includes the name, description, and version of the chart. This file is used by Helm to identify the chart and to provide information to the user when installing or upgrading the chart

```
apiVersion: v2
name: wordpress
description: A Helm chart for deploying
WordPress on Kubernetes
version: 1.0.0
appVersion: 5.8.0
maintainers:
  - name: Your Name
    email: your@email.com
```

values.yaml : This file contains the default values for the chart's parameters. These parameters are used in the templates to generate the Kubernetes YAML files. The user can override these values during installation or upgrade using the --set flag or a values file. This file is used to allow users to customize the behavior of the chart without modifying the templates directly.

```
wordpress:
  image: wordpress:5.8.0-php7.4-apache
  imagePullPolicy: IfNotPresent
  replicaCount: 1
```

The values in this file are used by the templates in the templates/ directory to generate the k8s YAML files

To change a value, you can modify the values.yaml file and then run the helm upgrade command

templates/ : This directory contains the Kubernetes YAML files that define the resources to be deployed. These files are usually written in a templating language like Go templating or Helm's own template language. The templates can include placeholders for the values defined in the values.yaml file. The templates can also include logic to conditionally include or exclude resources based on the values of the parameters

The template syntax, enclosed in double curly braces ({{ }}), is used to reference the values specified in values.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Release.Name }}-wordpress
  labels:
    app: wordpress
spec:
  replicas: {{ .Values.wordpress.replicaCount }}
...
    spec:
      containers:
        - name: wordpress
          image: {{ .Values.wordpress.image }}
          imagePullPolicy: {{ .Values.wordpress.imagePullPolicy }}
          ports:
            - name: http
              containerPort: 80
...
```

helpers.tpl : This file contains reusable snippets of code that can be used in the templates. These snippets can be used to simplify the templates and make them more readable. For example, a helper function might generate a random password or generate a unique name for a resource.