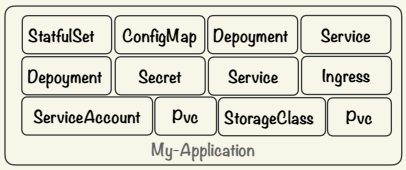
How to deploy an application in k8s?

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

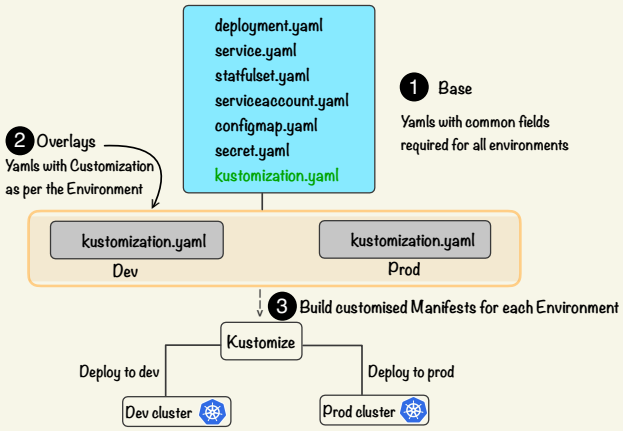
Kustomize

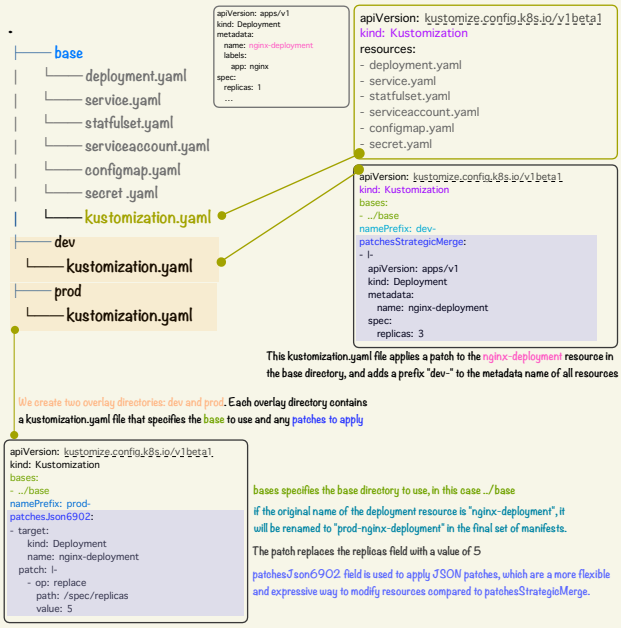
Kustomize is a tool for managing k8s manifest files using a declarative approach. It allows you to define a set of base manifests that define the desired state of your Kubernetes resources, and then apply changes using composition and customization. The basic workflow of Kustomize consists of the following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

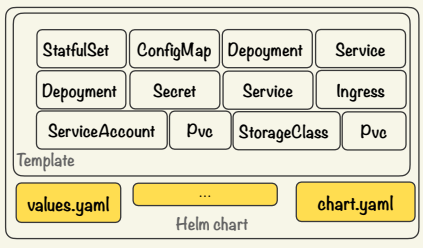
Create overlay directories for each environment or application variant, if needed. These overlay directories contain additional resources or modifications to apply on top of the base resources





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.

Do you want to deploy an application using Helm in Kubernetes? Here are the general steps to follow

Install Helm: You need to install Helm on your local machine or on the cluster where you will be deploying the application. You can follow the official Helm installation guide for your operating system to install Helm.

Add the Helm chart repository: Add the Helm chart repository that contains the application you want to deploy using the helm repo add command. You can specify a name for the repository and the URL of the repository.

Search for the Helm chart: Use the helm search command to search for the Helm chart that contains the application you want to deploy. You can specify the repository name or search all repositories.

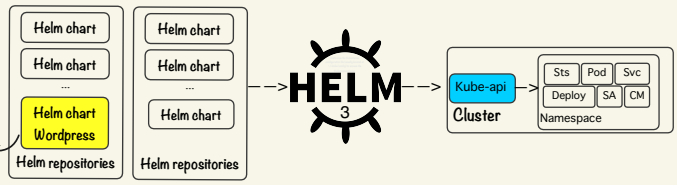
Configure the Helm chart: Create a values.yaml file to configure the Helm chart. This file contains the values that will be used to replace the placeholders in the Kubernetes resource files.

Install the Helm chart: Use the helm install command to install the Helm chart to the Kubernetes cluster. You can specify the release name, namespace, and any other required parameters using the command line or a YAML file

Verify the deployment: After the Helm chart has been installed, you can use kubectl commands to verify that the Kubernetes resources have been created and are running correctly

Upgrade or rollback the Helm chart: If you need to make changes to the application, you can use the helm upgrade command to upgrade the Helm chart. If there are issues with the new version, you can use the helm rollback command to revert to a previous version

How to deploy an application such as WordPress from a Helm repository using Helm?



1 Add the WordPress Helm chart repository:
you need to add the WordPress Helm chart repository to your local Helm installation. You can do this by running the following command:

```
helm repo add bitnami https://charts.bitnami.com/bitnami
```

The Bitnami Helm repository contains a variety of charts for popular applications like WordPress, MySQL.

Update the Repository
This step ensures that Helm has the latest versions of all the charts from the Bitnami repository.

```
helm repo update
```

better to create a Kubernetes namespace for your WordPress installation in order to isolate the resources associated with your WordPress deployment from other resources running in the same Kubernetes cluster

```
kubectl create namespace wordpress
```

2 Customize the WordPress deployment:
Before deploying WordPress, let's customize some values. The default configuration can be obtained using the following command:

```
helm show values bitnami/wordpress
```

You can customize the values in a Helm chart by using the --set flag when you install the chart or by creating a values.yaml file that overrides the default values

- If you want to customize the installation, you can pass additional parameters to the Helm chart using the --set flag. For example, you can set a custom password for the WordPress administrator account by running the following command

```
helm install my-wordpress bitnami/wordpress --namespace wordpress --set wordpressEmail=admin@example.com
```

- WordPress Helm chart comes with a default values file (values.yaml) which contains all the configuration options. We'll create a custom values file (values.yaml) to override some of these defaults, and customize the settings according to ours needs

values.yaml

```
wordpressUsername: myusername
wordpressPassword: mypassword
wordpress:
  persistence:
    size: 20Gi
#mariadb.auth.rootPassword= ROOT_PASSWORD
mariadb:
  auth:
    rootPassword: ROOT_PASSWORD
```

3 Install the chart:
Once you have customized the values, you can install the chart on your Kubernetes cluster using the helm install command. to install the WordPress chart with a release name of my-wordpress, run the following command:

```
helm install my-wordpress bitnami/wordpress --namespace wordpress --create-namespace -f values.yaml --set service.type=NodePort
```

The my-wordpress argument is the name of the release that Helm will use to track the installation, and the --namespace wordpress argument specifies the namespace in which to install WordPress

This command installs WordPress using the values.yaml file and sets the service.type value to NodePort

Verify the chart:

```
helm list -n wordpress
NAME        NAMESPACE   REVISION  UPDATED                                  STATUS    CHART               APP VERSION
my-wordpress wordpress  1         2023-07-28 20:40:28.214200542 +03 +03    deployed  wordpress-16.1.33   6.2.2
```

releaseName — A release is an instance of an application deployed by Helm from a chart

Upgrade or rollback the chart:

```
helm upgrade -f values2.yaml my-wordpress bitnami/wordpress -n wordpress
```

```
helm upgrade [RELEASE] [CHART] [flags]
```

```
helm list -n wordpress
NAME        NAMESPACE   REVISION  UPDATED                                  STATUS    CHART               APP VERSION
my-wordpress wordpress  2         2023-09-26 11:42:00.841703412 +03 +03    deployed  wordpress-17.1.6    6.3.1
```

A revision is a versioned change to the release. Each time a release is installed or upgraded, a new revision is created incrementally (rev 1, 2, 3 etc).

```
helm rollback my-wordpress 1 -n wordpress
```

```
helm rollback RELEASE_NAME REVISION_NUMBER
```

How to get custom values for a helm release?

```
helm get values my-wordpress --revision=2 -n wordpress
USER-SUPPLIED VALUES:
wordpressPassword: "qazwsx"
wordpressUsername: daniele
```

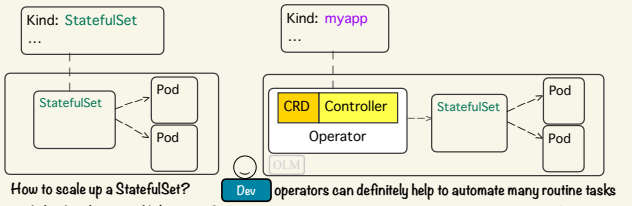
>> operator

operator is a method of packaging, deploying, and managing a specific application or workload on a Kubernetes cluster. Operators are essentially Kubernetes controllers that are designed to automate the deployment and management of complex applications or services

An operator typically consists of custom resources, custom controllers, and a set of Kubernetes objects that are defined to manage the application or workload. The custom resource is a Kubernetes object that represents the desired state of the application or workload, while the custom controller is responsible for ensuring that the actual state of the application or workload matches the desired state.

Managing stateful applications in Kubernetes can be challenging, but operators are particularly well-suited for this task. For example, an operator for a database application might automate tasks such as provisioning new database instances, scaling the database up or down, performing backups and restores, and handling failovers

Operators are typically implemented using the Kubernetes Operator SDK, which provides a set of tools and libraries for building, testing, and deploying operators



Dev — How to scale up a StatefulSet?
How to do leader election in Kubernetes?
How to migrate databases?
..

Dev — operators can definitely help to automate many routine tasks associated with managing complex applications in Kubernetes, & this can free up human operators to focus on more strategic tasks