HELM

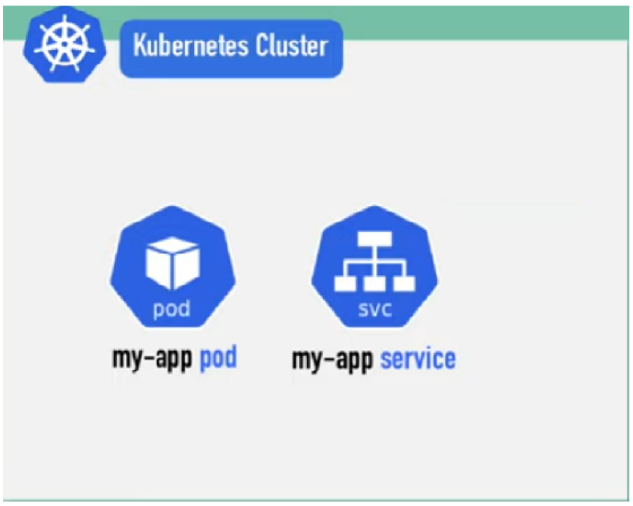
- In order to deploy an application on k8s we need to interact with k8s API to create resources, kubectl is the tool we use to do this
- We express Kubernetes resources in YAML Files
- These files are static in nature.
- Resource files are static:
    - This is the challenge that primarily effects the declarative configuration style of applying YAML resources
    - K8s YAML files are not designed to be parametrized
    - Consider the below two manifests written to deploy two different applications

```
File  Edit  Format  View  Help
apiVersion: v1
kind: Deployment
metadata:
  name: my-k8s-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-k8s-app
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
template:
  metadata:
    labels:
      app: my-k8s-app
  spec:
    containers:
      - image: my-k8s:v1
        name: app
```

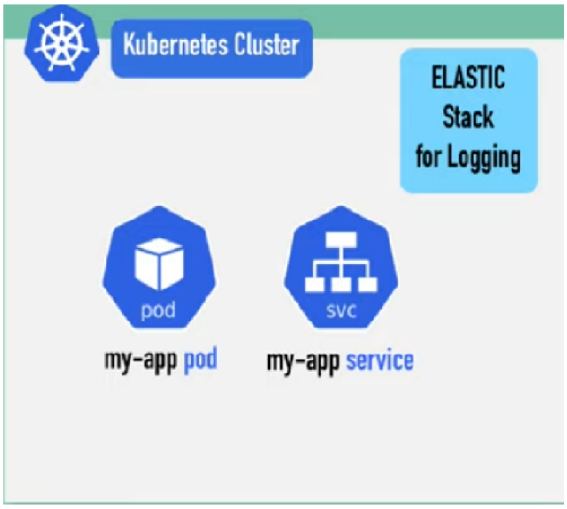
```
File  Edit  Format  View  Help
apiVersion: v1
kind: Deployment
metadata:
  name: your-k8s-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: your-k8s-app
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
template:
  metadata:
    labels:
      app: your-k8s-app
  spec:
    containers:
      - image: your-k8s:v1
        name: app
```

    - In the above image each file is almost exactly the same, but we still cannot parametrize
- Helm to the rescue: Helm is an opensource tool used for packaging and deploying applications on k8s. It is often referred as Kubernetes Package Manager.

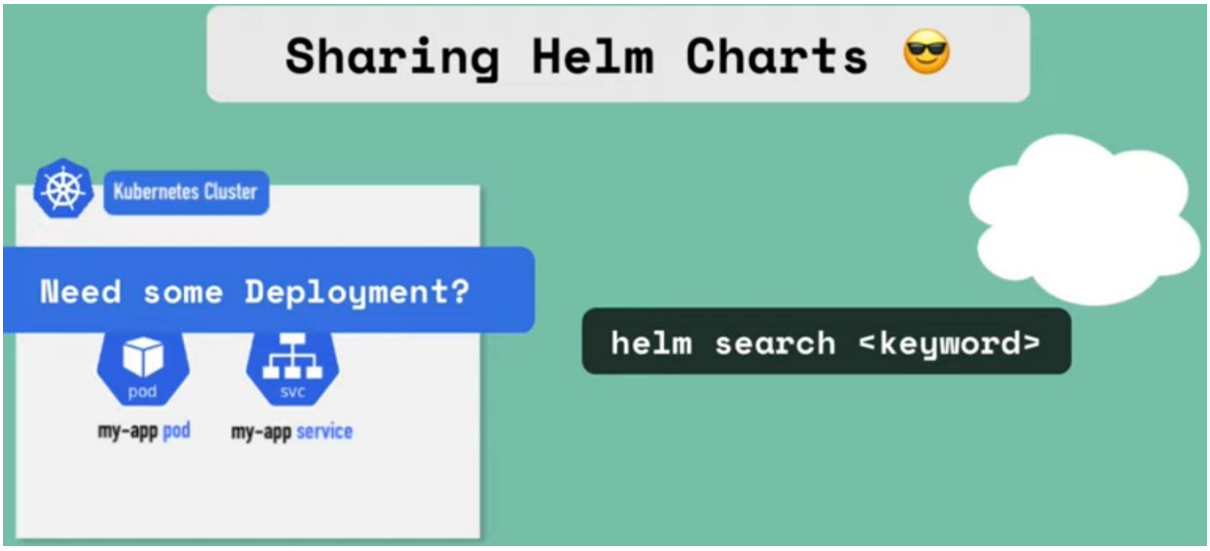
- Suppose you have a Kubernetes cluster, and you want to install ELK as a side car container, then you need to write stateful set, PV, PVC, services, config map and secrets.



Before



After



- Helm was designed to provide an experience similar to that of package manager (apt, yum, dnf etc)
- APT operates on Debian packages and yum/dnf operates on RPM package.
- Helm operates on Charts.
- A Helm Chart contains declarative k8s resource files required to deploy an application
- Helm relies on repositories to provide access to charts
- Chart developers create declarative YAML files, package them into charts and publish them to chart repository
- End users then Helm to search for existing chart to deploy some app on to k8s
- Refer below to view a sample usage of helm chart which installs mysql.
- Link: https://bitnami.com/stack/mysql/helm

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

$ helm install my-release bitnami/mysql
# Read more about the installation in the Bitnami MySQL Stack Chart Github repository
```

Let's try to understand Helm's subcommands

DNF Subcommands	Helm Subcommands	Purpose
install	install	Install an application and its dependencies
upgrade	upgrade	Upgrades an application to newer version
downgrade	rollback	Reverts the application to previous version
remove	uninstall	Delete an application

- The abstracted complexity of k8s resources:
  - Let's assume a developer has been given a task of deploying a MySQL database onto k8s.
  - Developer needs to create resources required to create containers, network and storage
  - With Helm, developer tasked with deploying a mysql database could simply search for MySQL Chart in chart repositories
- Automated Life cycle Hooks:
  - Helm provides the ability to define the life cycle hooks. Lifecycle hooks are actions that take place automatically at different stages of an application's life cycle.
  - Examples:
    - Perform a data backup on an upgrade
    - Restore data on rollback
    - Validate k8s environment prior to installation

Installing Helm Link: https://helm.sh/docs/intro/install/

$ curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
$ chmod 700 get_helm.sh
$ ./get_helm.sh

Configuring Helm

- Helm is a tool with sensible default settings that allow users to be productive without needing to perform a large of post-installation tasks
- With that being said there are several options users can change or enable to modify the Helm's behaviour.
- Adding upstream repositories:
    - Helm provides the **repo** subcommand to allow users to manage configured chart repositories. This subcommand contains additional subcommands
        - **add**: To add a chart repository
        - **list**: To list chart repositories
        - **remove**: To remove the chart repository
        - **update**: To update information on available charts locally from chart repositories
        - **index**: To generate and index file given a directory containing packaged charts
- Example: Lets install MySQL from bitnami repository

Add bitnami repository as upstream

```
qtkhajacloud@cloudshell:~ (expertkubernetes)$ helm repo add bitnami https://charts.bitnami.com/bitnami
"bitnami" has been added to your repositories
qtkhajacloud@cloudshell:~ (expertkubernetes)$ helm repo list
NAME    URL
bitnami https://charts.bitnami.com/bitnami
qtkhajacloud@cloudshell:~ (expertkubernetes)$
```

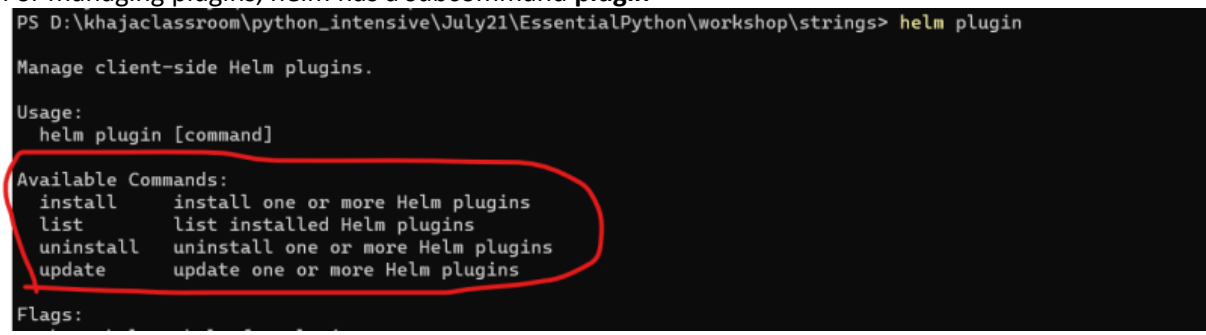
Install MySQL

```
qtkhajacloud@cloudshell:~ (expertkubernetes)$ helm install my-release bitnami/mysql
NAME: my-release
LAST DEPLOYED: Tue Aug  3 14:18:27 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
** Please be patient while the chart is being deployed **

Tip:

  Watch the deployment status using the command: kubectl get pods -w --namespace default

Services:

  echo Primary: my-release-mysql.default.svc.cluster.local:3306

Administrator credentials:

  echo Username: root
  echo Password : $(kubectl get secret --namespace default my-release-mysql -o jsonpath="{.data.mysql-root-password}" | base64 --decode)

To connect to your database:

  1. Run a pod that you can use as a client:

Administrator credentials:

  echo Username: root
  echo Password : $(kubectl get secret --namespace default my-release-mysql -o jsonpath="{.data.mysql-root-password}" | base64 --decode)

To connect to your database:

  1. Run a pod that you can use as a client:

      kubectl run my-release-mysql-client --rm --tty -i --restart='Never' --image  docker.io/bitnami/mysql:8.0.26-debian-10-r0 --namespace default --command -- bash

  2. To connect to primary service (read/write):

      mysql -h my-release-mysql.default.svc.cluster.local -uroot -p my_database


To upgrade this helm chart:

  1. Obtain the password as described on the 'Administrator credentials' section and set the 'root.password' parameter as shown below:

      ROOT_PASSWORD=$(kubectl get secret --namespace default my-release-mysql -o jsonpath="{.data.mysql-root-password}" | base64 --decode)
      helm upgrade --namespace default my-release bitnami/mysql --set auth.rootPassword=$ROOT_PASSWORD
qtkhajacloud@cloudshell:~ (expertkubernetes)$
```

To uninstall MySQL helm uninstall my-release

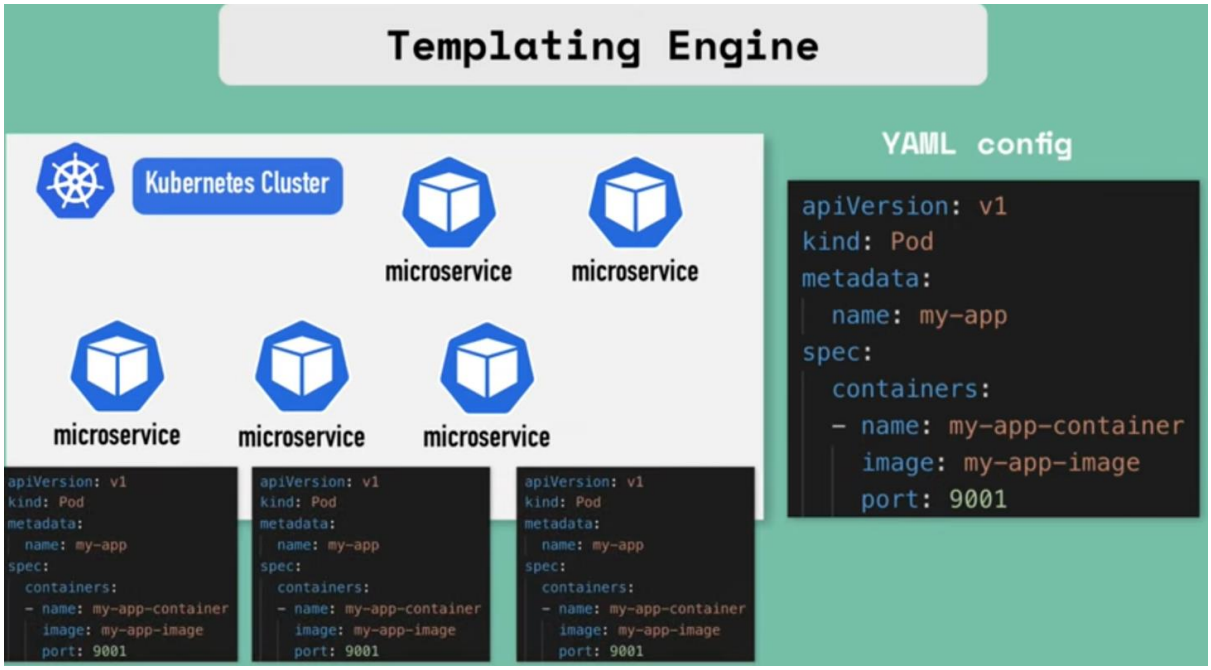
Adding plugins:

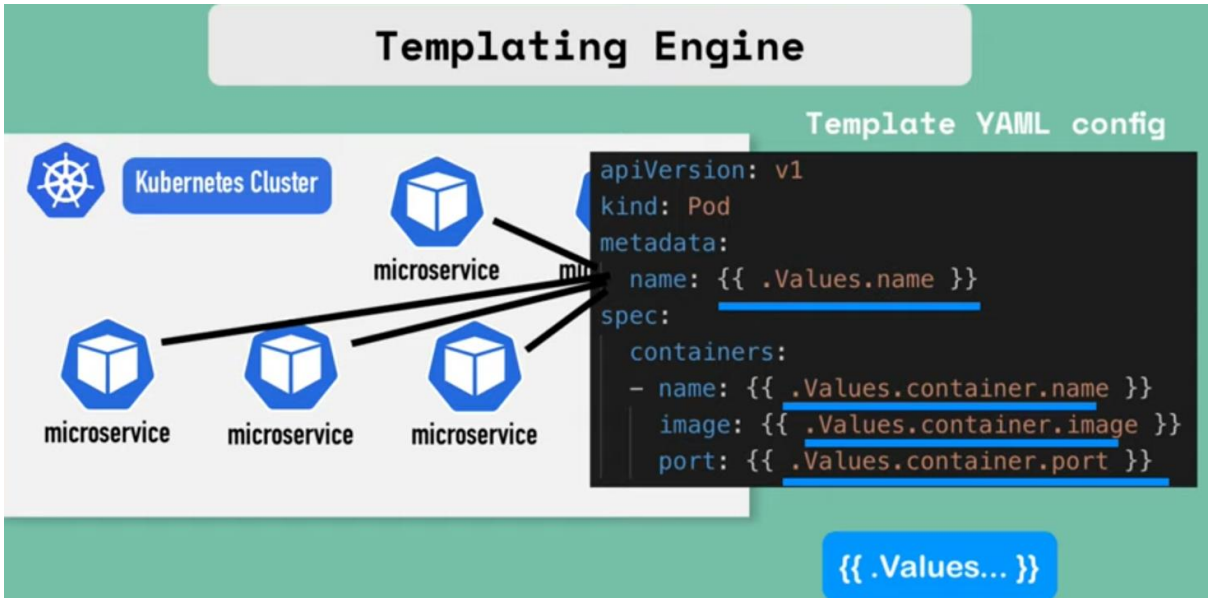
- Plugins are add-on capabilities that can be used to provide additional features to helm.
- For managing plugins, helm has a subcommand **plugin**

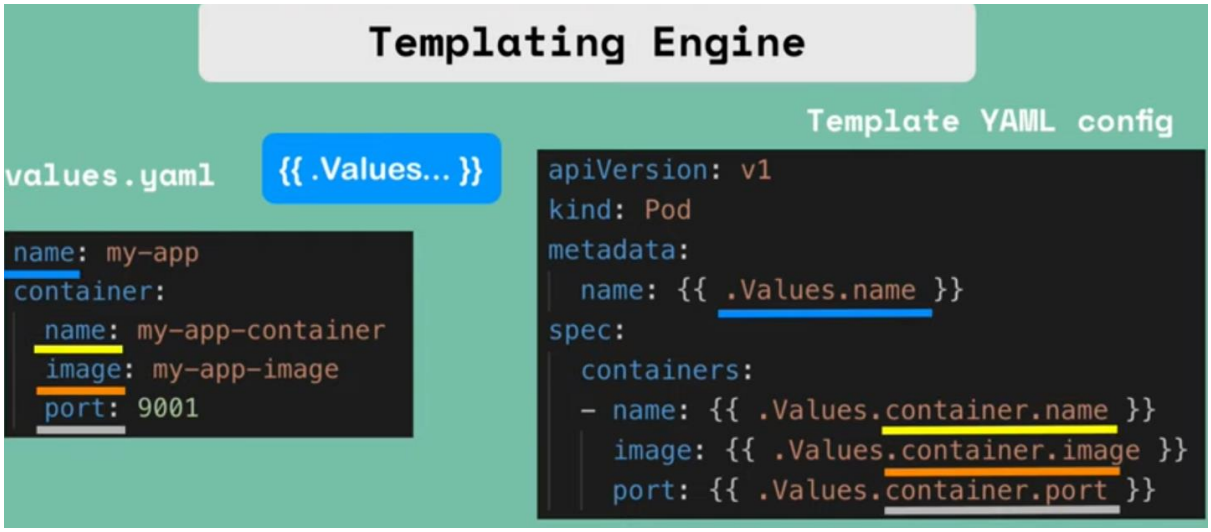


- Refer https://helm.sh/docs/community/related/ for some plugins

- ENVIRONMENT Variables: Helm relies on the existence of externalized environmental variables to configure low-level options
  - XDG_CACHE_HOME: Sets an alternative location for storing cached files
  - XDG_CONFIG_HOME: Sets an alternative location for storing helm configuration
  - XDG_DATA_HOME: Sets an alternative location for storing Helm Data
  - HELM_DRIVER: Sets the backend storage driver
  - HELM_NO_PLUGINS: Disables the plugins
  - KUBECONFIG: Sets an alternative Kubernetes configuration file
- Link- https://helm.sh/docs/helm/helm/
- Helm has the following paths
  - Windows:
    - Cache Path: %TEMP%\helm
    - Configuration Path: %APPDATA%\helm
    - Data Path: %APPDATA%\helm
  - Linux:
    - Cache Path: $HOME/.cache/helm
    - Configuration Path: $HOME/.config/helm
    - Data Path: $HOME/.local/share/helm


Templating Engine
Kubernetes Cluster
microservice
microservice
microservice
microservice
microservice
YAML config
apiVersion: v1
kind: Pod
metadata:
name: my-app
spec:
containers:
- name: my-app-container
image: my-app-image
port: 9001


Templating Engine
Template YAML config
Kubernetes Cluster
microservice
microservice
microservice
microservice
apiVersion: v1
kind: Pod
metadata:
name: {{ .Values.name }}
spec:
containers:
- name: {{ .Values.container.name }}
image: {{ .Values.container.image }}
port: {{ .Values.container.port }}
{{ .Values... }}


Templating Engine
Template YAML config
values.yaml
{{ .Values... }}
name: my-app
container:
name: my-app-container
image: my-app-image
port: 9001
apiVersion: v1
kind: Pod
metadata:
name: {{ .Values.name }}
spec:
containers:
- name: {{ .Values.container.name }}
image: {{ .Values.container.image }}
port: {{ .Values.container.port }}

Templating Engine

many Yaml Files

just 1 Yaml File

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  containers:
  - name: my-app-container
    image: my-app-image
    port: 9001
```

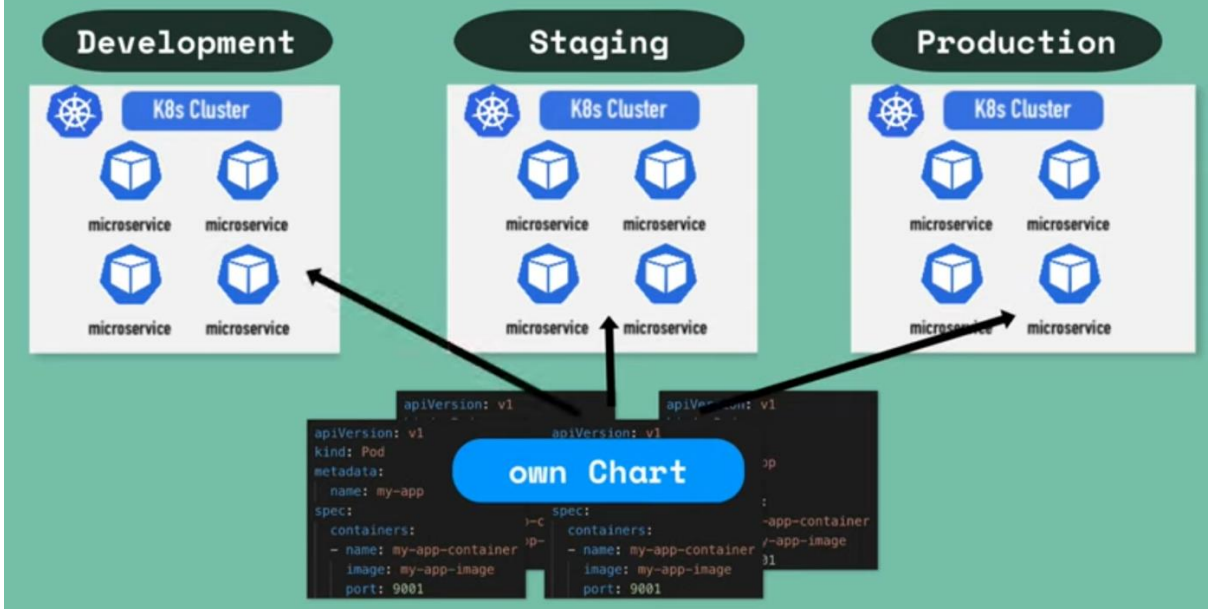
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: {{ .Values.name }}
spec:
  containers:
  - name: {{ .Values.container.name }}
    image: {{ .Values.container.image }}
    port: {{ .Values.container.port }}
```

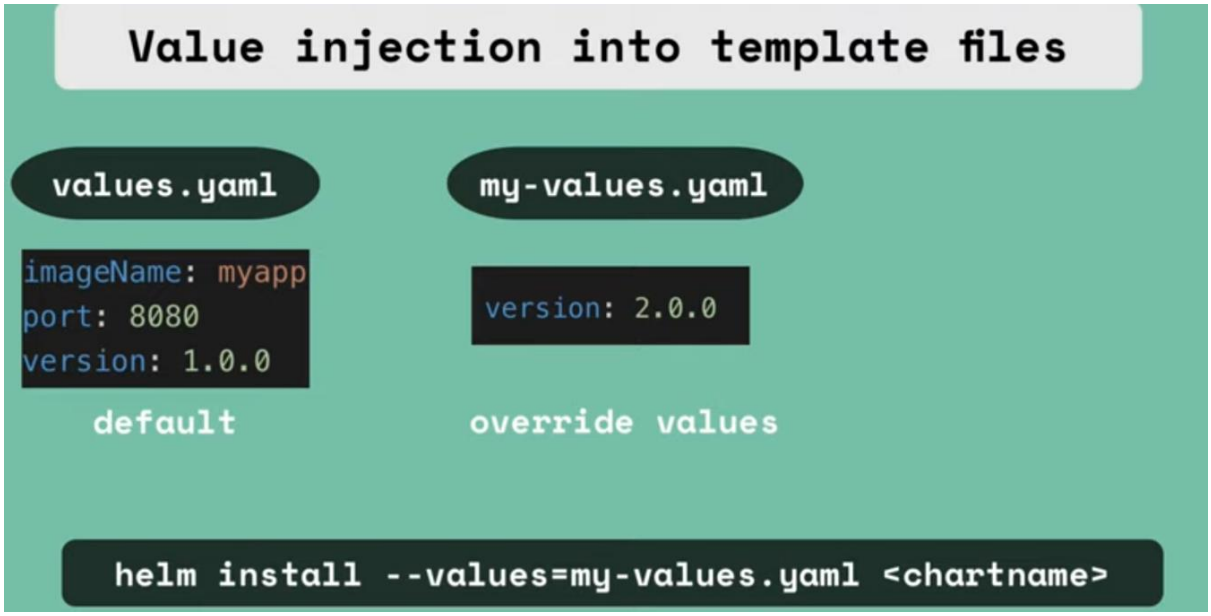
Practical for CI / CD

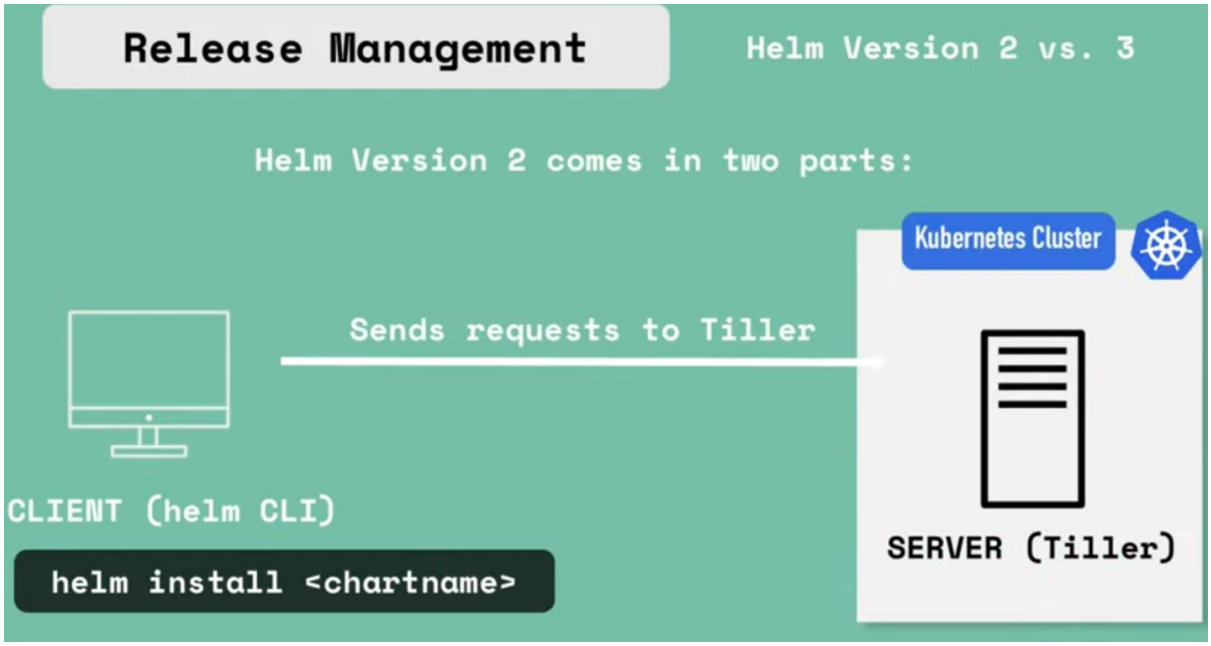
In your Build you can replace the values on the fly

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: {{ .Values.name }}
spec:
  containers:
  - name: {{ .Values.container.name }}
    image: {{ .Values.container.image }}
    port: {{ .Values.container.port }}
```

Same Applications across different environments




Value injection into template files
values.yaml
my-values.yaml
imageName: myapp
port: 8080
version: 1.0.0
version: 2.0.0
default
override values
helm install --values=my-values.yaml <chartname>


Release Management
Helm Version 2 vs. 3
Helm Version 2 comes in two parts:
Kubernetes Cluster
Sends requests to Tiller
CLIENT (helm CLI)
SERVER (Tiller)
helm install <chartname>

Release Management

Create or change deployment

Keeping track of all chart executions:

Revision	Request
1	Installed chart
2	Upgraded to v 1.0.0

- Changes are applied to existing deployment instead of creating a new one

```
helm install <chartname>
helm upgrade <chartname>
```

- Tiller has too much power inside of K8s cluster

CREATE 🤔 DELETE

UPDATE

- Security Issue

In Helm 3 Tiller got removed!

Downsides of Helm

- Tiller has too much power inside of K8s cluster

- Security Issue

In Helm 3 Tiller got removed!

- Solves the Security Concern,
  but makes it more difficult to use