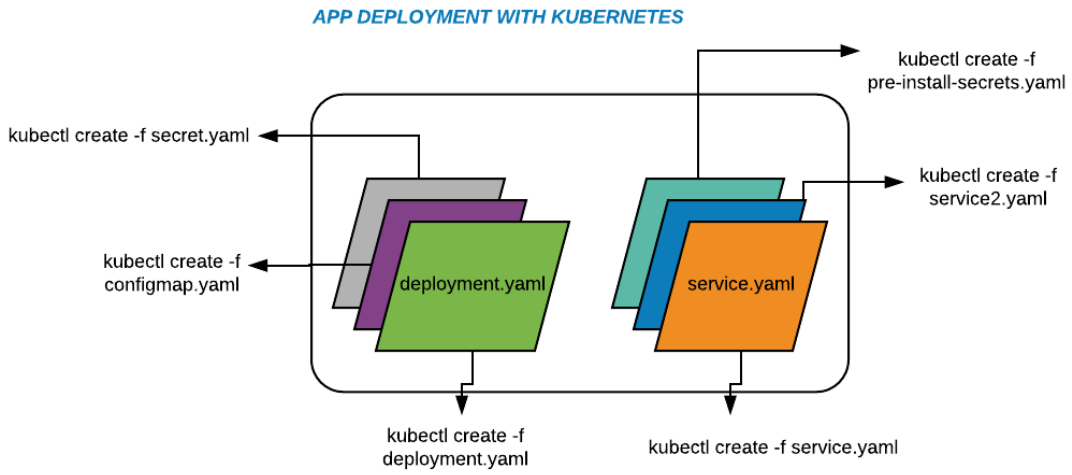
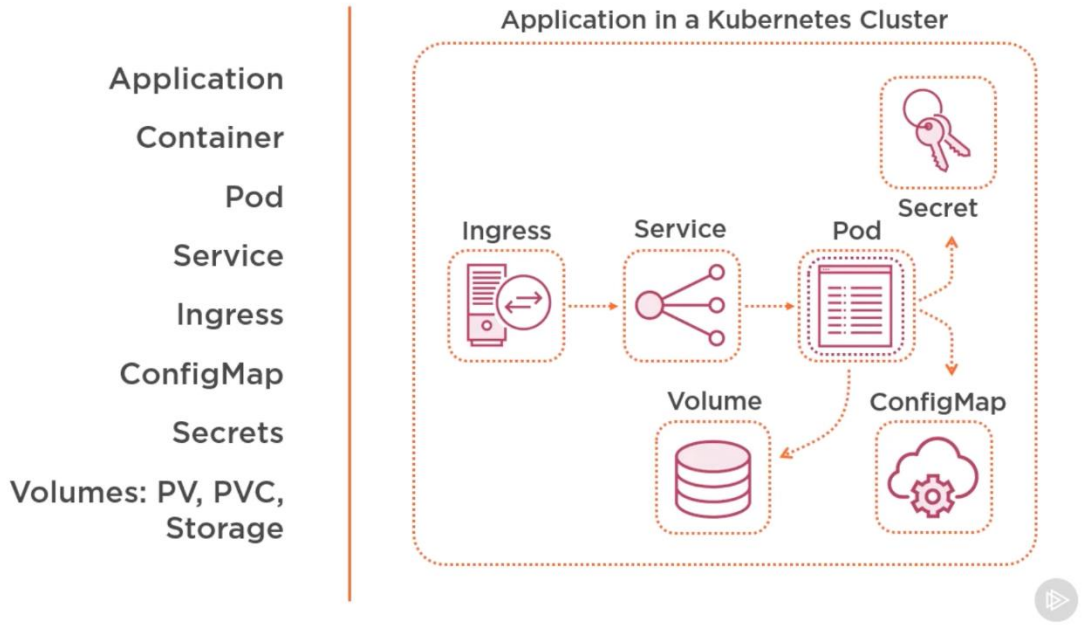
Helm Short notes

Helm and Kubernetes





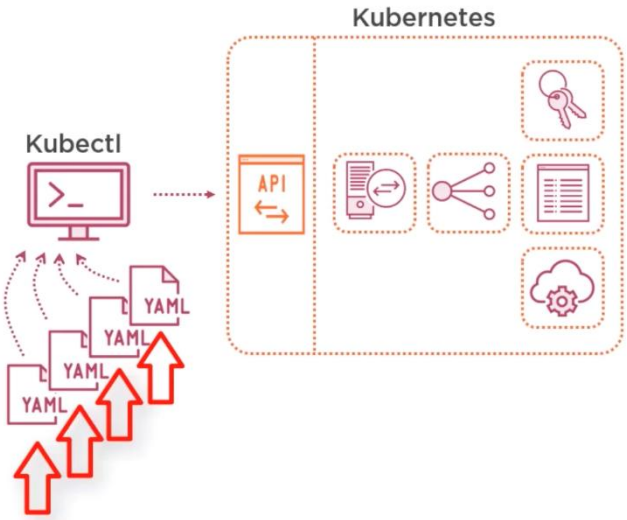
Kubernetes API :

REST Client

Go Client

Kubectl

Limitations :



Kubernetes API :

REST Client

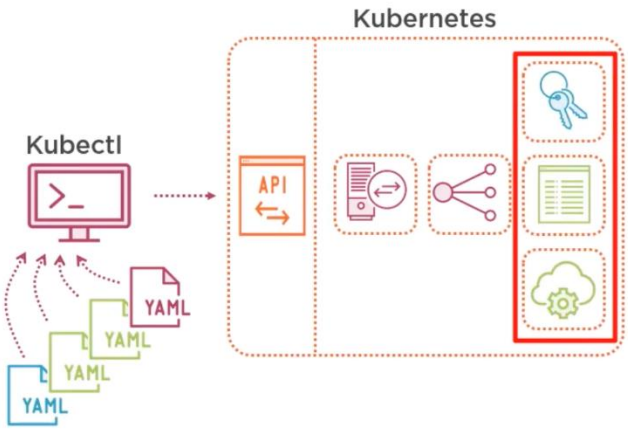
Go Client

Kubectl

Limitations :

Packaging

Versioning



What is Helm?

Package Manager for Kubernetes

To package YAML Files

and distribute them in public and private repositories

Analogy with Other Package Managers

	Package manager	Packages
System	Apt	deb
	Yum	rpm
Dev	Maven	Jar, Ear, ...
	Npm	Node Modules
	Pip	Python packages
Kubernetes	Helm	Charts

Kubernetes Cluster

Need some Deployment?

pod svc

my-app pod my-app service

helm search <keyword>

or Helm Hub:

Discover & launch great Kubernetes-ready apps

Search charts...

1140 charts ready to deploy

Public Registries:

Discover & launch great Kubernetes-ready apps

Search charts...

1140 charts ready to deploy

Private Registries: share in organisations

Kubernetes API :

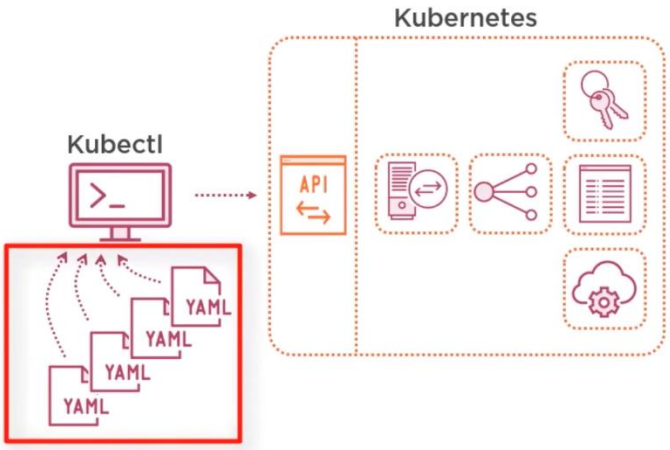
REST Client

Go Client

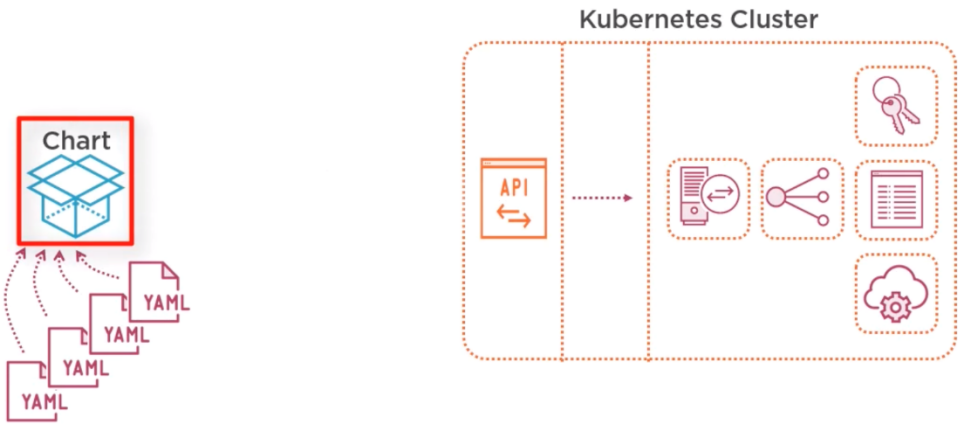
Kubectl

Limitations :

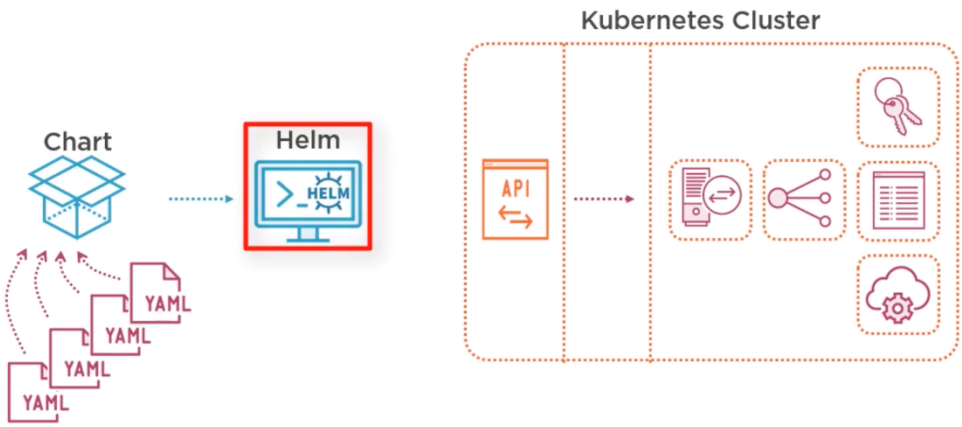
Packaging



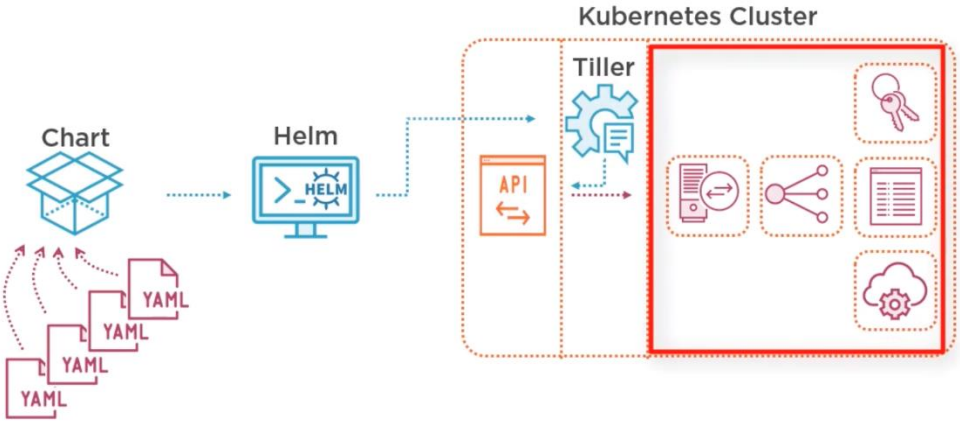
How It Works

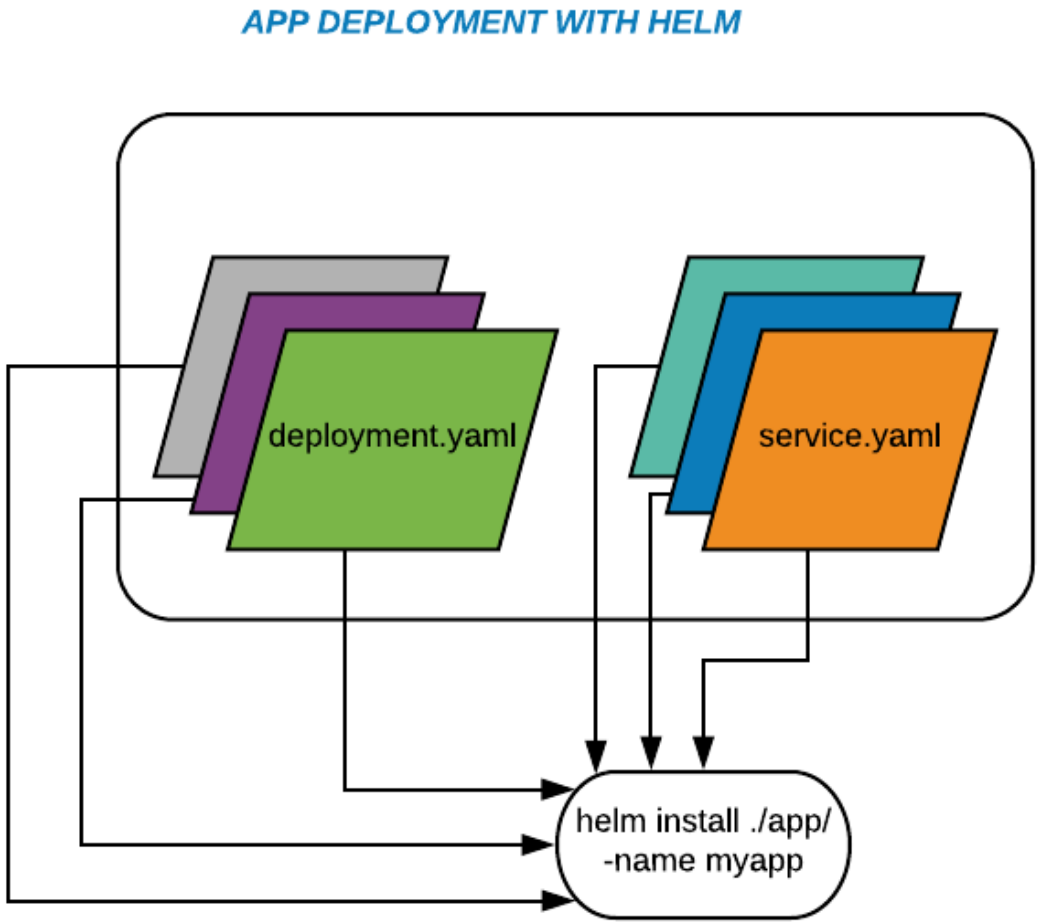


How It Works

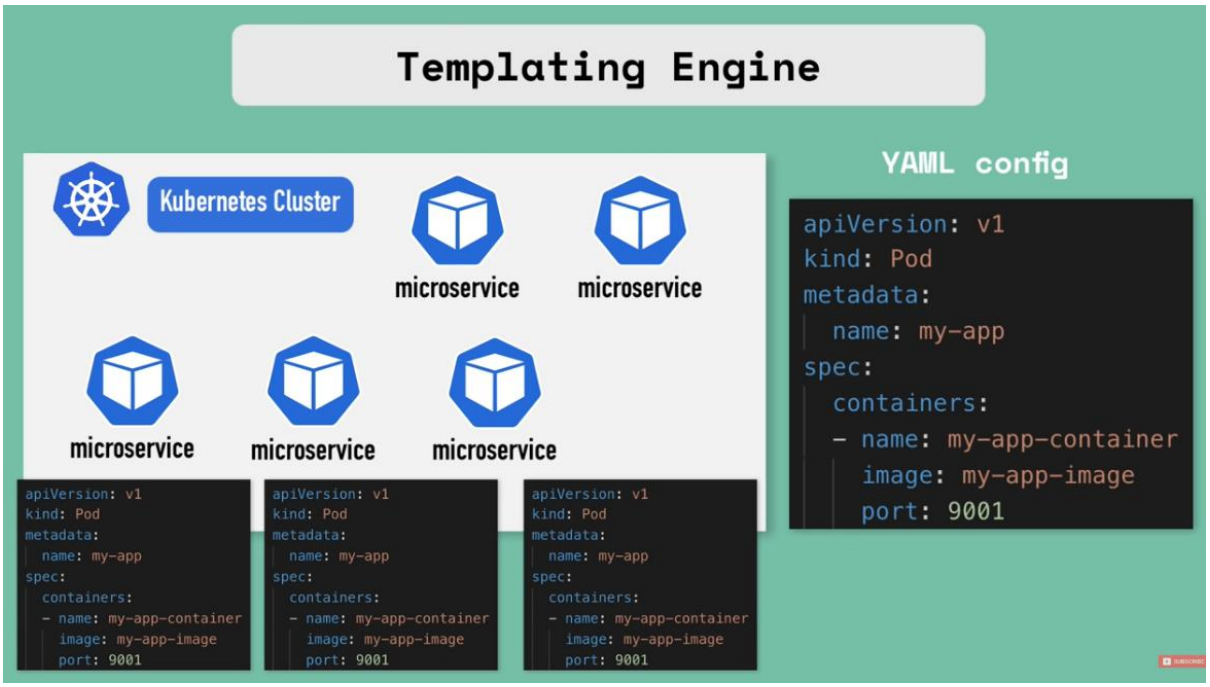


How It Works

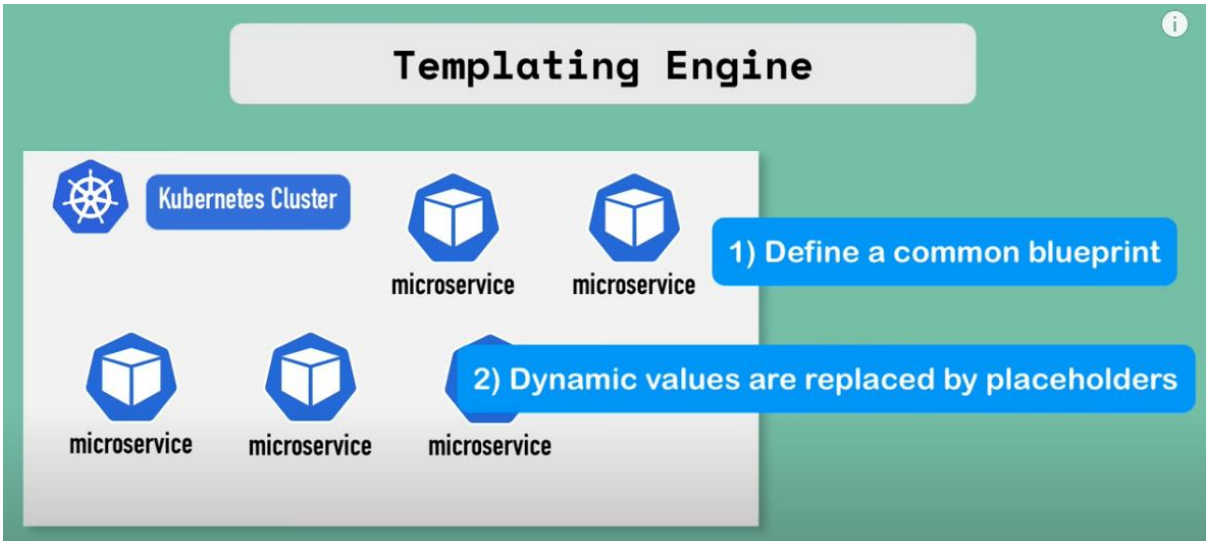

Kubernetes Cluster
Tiller
Chart
Helm
HELM
API
YAML
YAML
YAML
YAML



App deployment without Helm



App Deployment with Helm by Go Templete Engine



Templating Engine

Kubernetes Cluster

microservice

microservice

microservice

microservice

Template YAML config

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

Templating Engine

Kubernetes Cluster

microservice

microservice

microservice

microservice

Template YAML config

```yaml
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

values.yaml

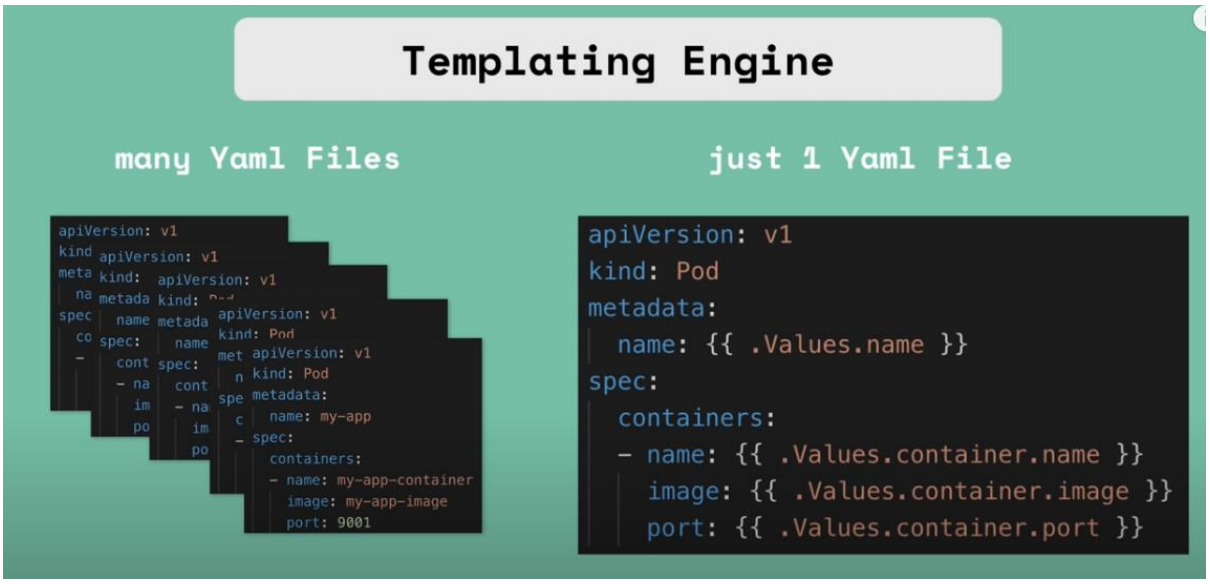
{{ .Values... }}

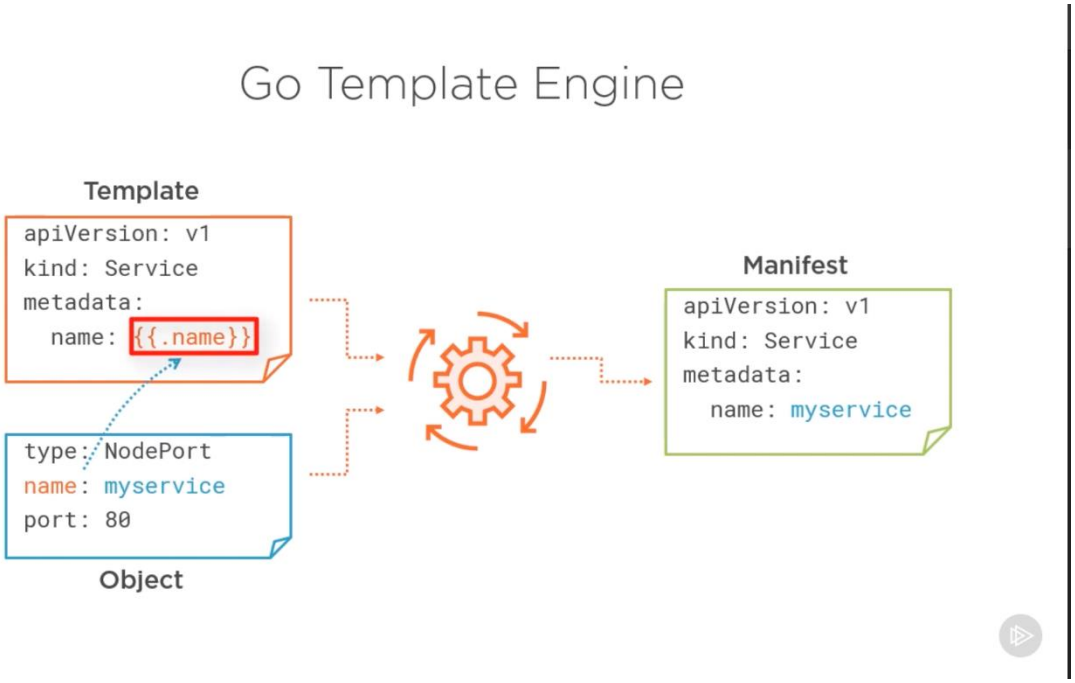
```yaml
name: my-app
container:
  name: my-app-container
  image: my-app-image
  port: 9001
```

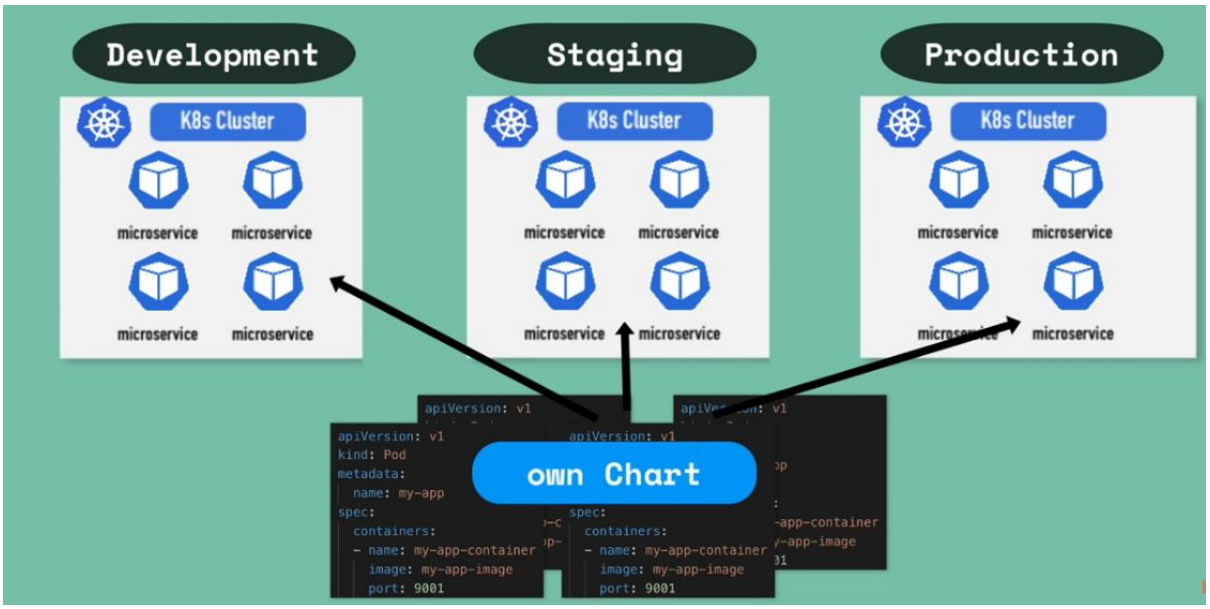
Template YAML config

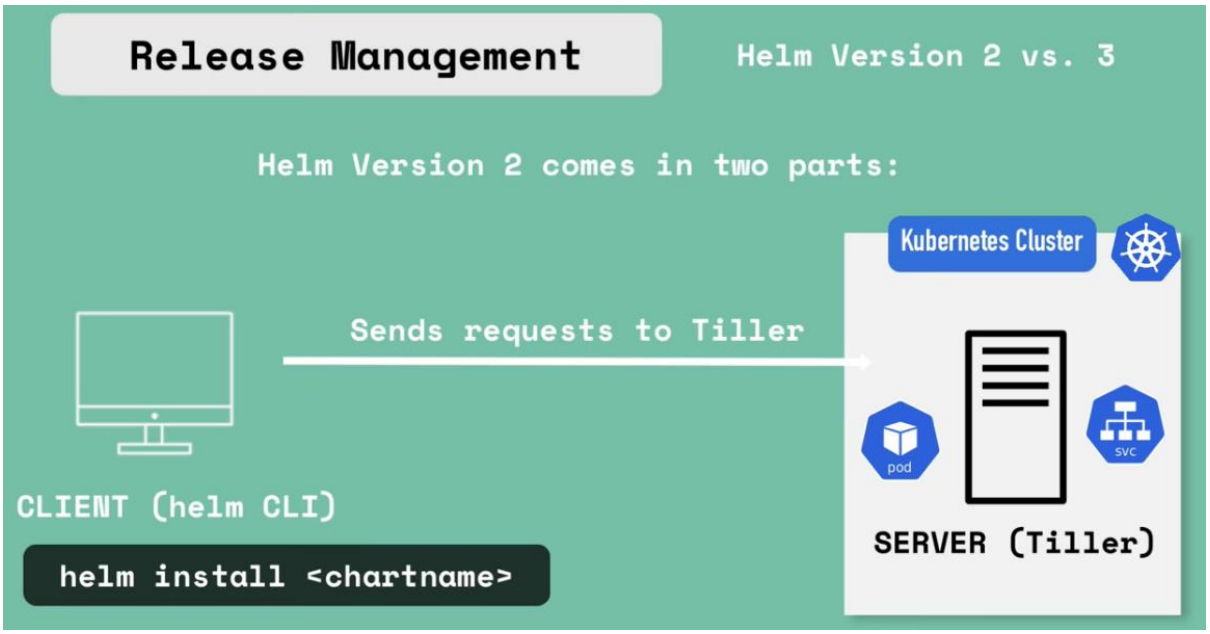
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: {{ .Values.name }}
spec:
  containers:
  - name: {{ .Values.container.name }}
    image: {{ .Values.container.image }}
    port: {{ .Values.container.port }}
```

Values defined either via **yaml file** or with **--set** flag


Templating Engine
many Yaml Files
just 1 Yaml File
apiVersion: v1
kind: Pod
metadata:
name: {{ .Values.name }}
spec:
containers:
- name: {{ .Values.container.name }}
image: {{ .Values.container.image }}
port: {{ .Values.container.port }}


Go Template Engine
Template
apiVersion: v1
kind: Service
metadata:
name: {{.name}}
type: NodePort
name: myservice
port: 80
Object
Manifest
apiVersion: v1
kind: Service
metadata:
name: myservice


Development
Staging
Production
K8s Cluster
K8s Cluster
K8s Cluster
microservice
own Chart
apiVersion: v1
kind: Pod
metadata:
name: my-app
spec:
containers:
- name: my-app-container
image: my-app-image
port: 9001


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

Release Management

Create or change deployment

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

stores copy of configuration

Kubernetes Cluster

SERVER (Tiller)

Release Management

Keeping track of all chart executions:

Revision	Request
1	Installed chart
2	Upgraded to v 1.0.0
3	Rolled back to 1

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

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

Downsides of Tiller

- Tiller has too much power inside of K8s cluster
- Security Issue
- Solves the Security Concern 👍

In Helm 3 Tiller got removed!

Helm Features

Charts

Templates

Dependencies

Repositories

Helm Charts

- Bundle of YAML Files
- Create your own Helm Charts with Helm
- Push them to Helm Repository
- Download and use existing ones

Manage Complexity

Charts describe even the most complex apps, provide repeatable application installation, and serve as a single point of authority.

Easy Updates

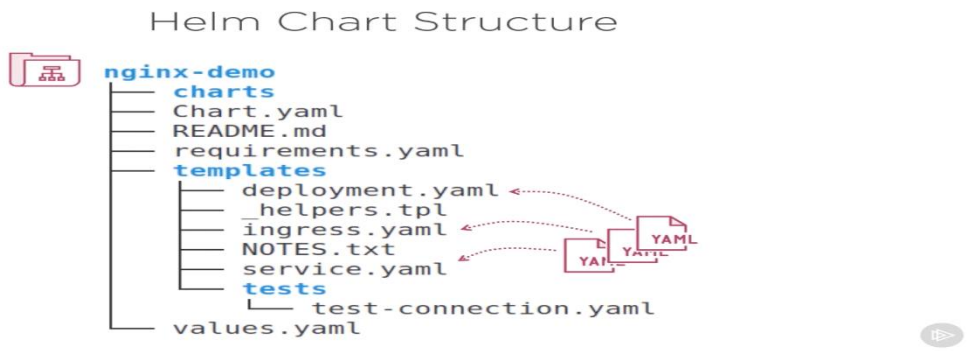
Take the pain out of updates with in-place upgrades and custom hooks.

Simple Sharing

Charts are easy to version, share, and host on public or private servers.

Rollbacks

Use `helm rollback` to roll back to an older version of a release with ease.

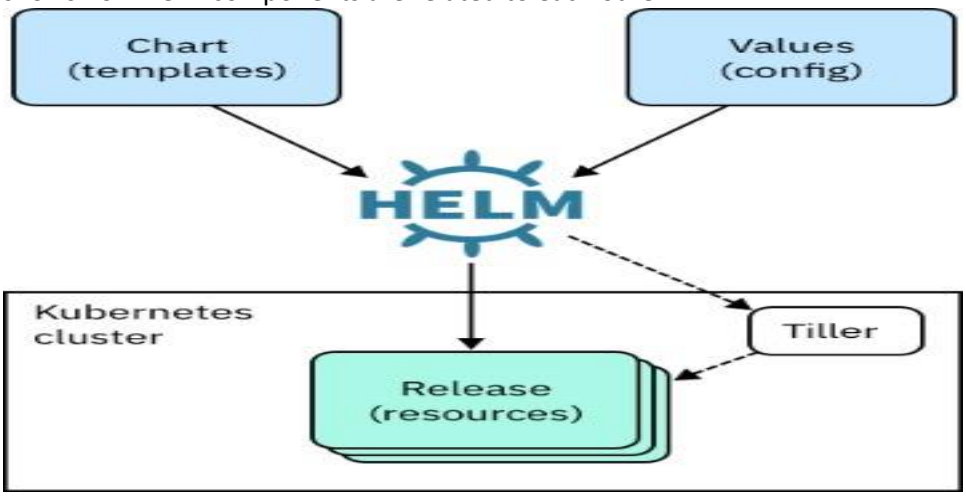



```
chart/
        chart.yaml              # A YAML file containing information about the
                                chart
        LICENSE                 # OPTIONAL: A plain text file containing the
                                license for the chart
        README.md               # OPTIONAL: A human-readable README file
        values.yaml             # The default configuration values for this
                                chart
        values.schema.json      # OPTIONAL: A JSON Schema for imposing a
                                structure on the values.yaml file
        charts/                 # A directory containing any charts upon which
                                this chart depends.
        crds/                   # Custom Resource Definitions
        templates/              # A directory of templates that, when combined
                                with values,will generate valid Kubernetes
                                manifest files.
        templates/ NOTES.txt    # OPTIONAL: A plain text file containing short
                                usage notes
```

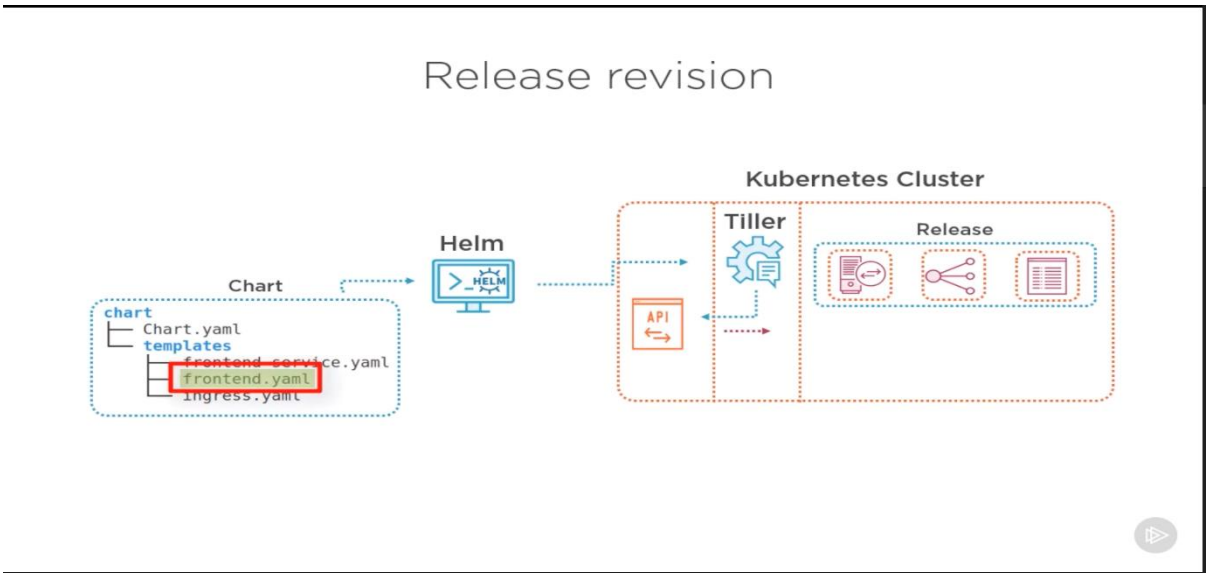
Architecture of Helm

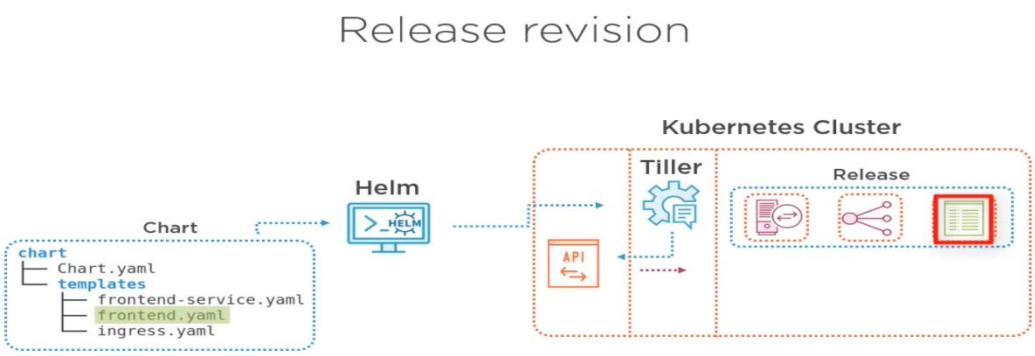
Helm has two elements, a client (Helm) and a server (Tiller).

The server element runs inside a Kubernetes cluster and manages the installation of charts. This diagram shows how Helm components are related to each other:

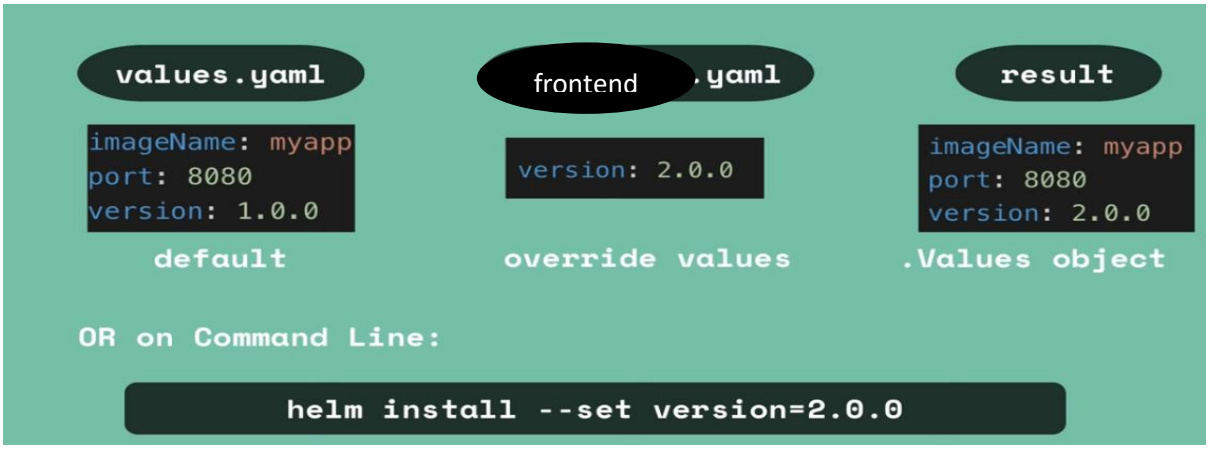


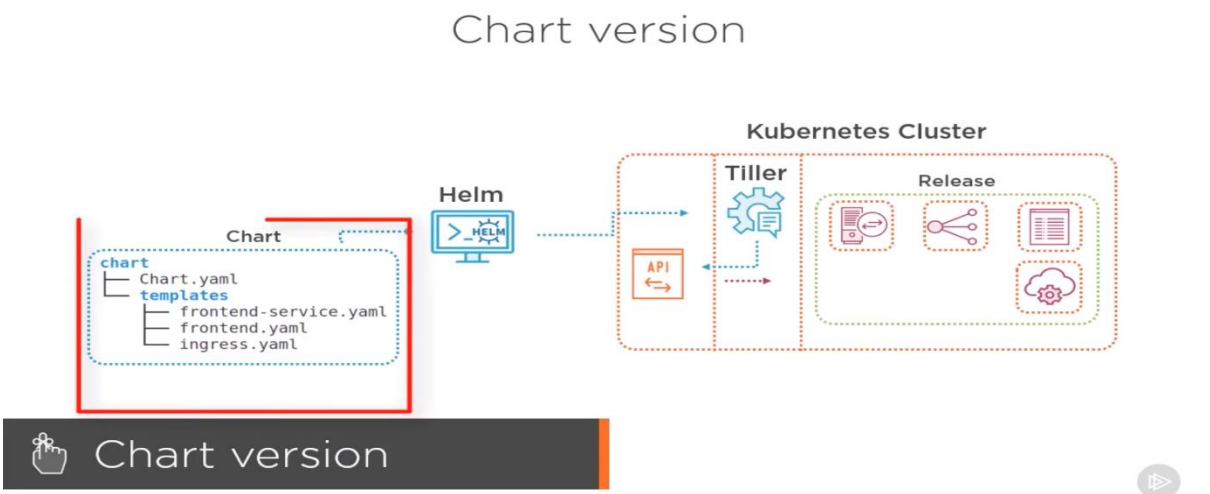
Helm: A command-line interface (CLI) that installs charts into Kubernetes, creating a release for each installation. To find new charts, you search Helm chart repositories.
Chart: An application package that contains *templates* for a set of resources that are necessary to run the application. A template uses variables that are substituted with values when the manifest is created. The chart includes a values file that describes how to configure the resources.
Repository: Storage for Helm charts. The namespace of the hub for official charts is *stable*.
Release: An instance of a chart that is running in a Kubernetes cluster. You can install the same chart multiple times to create many releases.
Tiller: The Helm server-side templating engine, which runs in a pod in a Kubernetes cluster. Tiller processes a chart to generate Kubernetes resource manifests, which are YAML-formatted files that describe a resource. YAML is a human-readable structured data format. Tiller then installs the release into the cluster. Tiller stores each release as a Kubernetes ConfigMap.

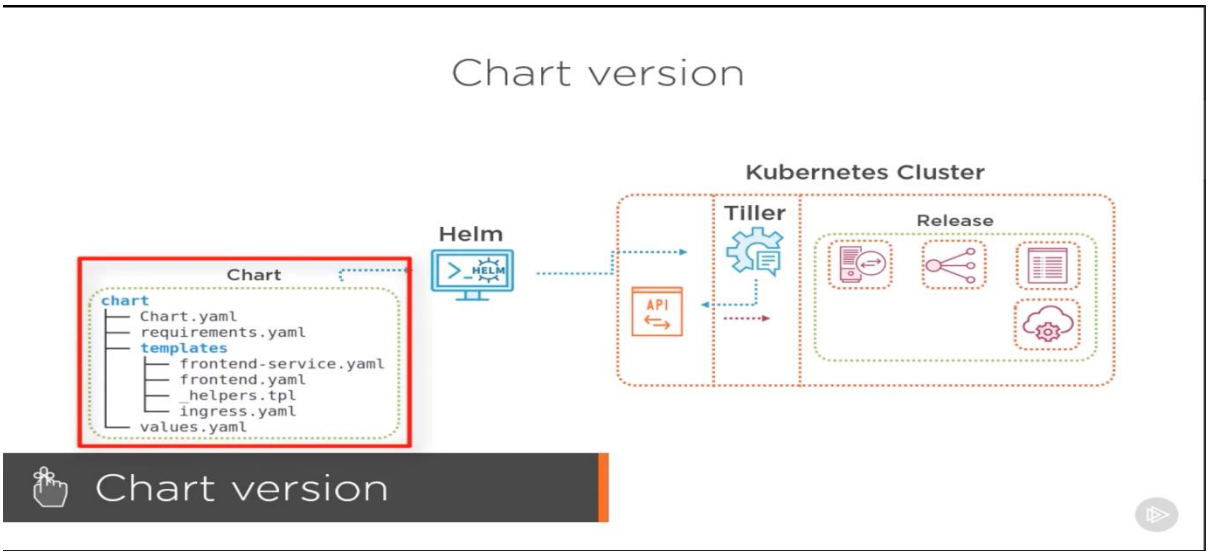




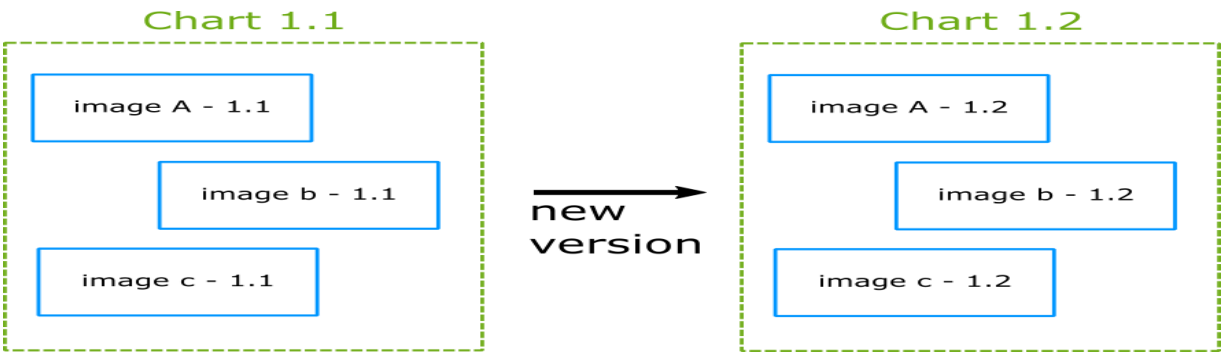
Release revision



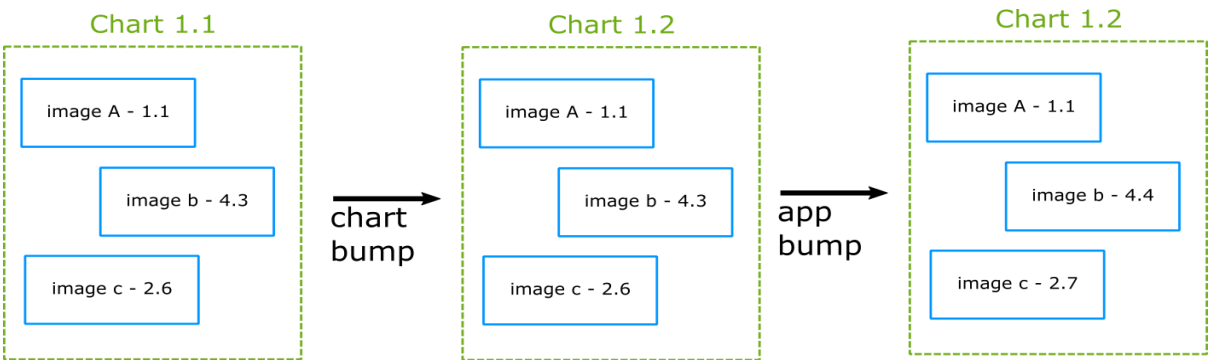




Synced versions in Helm



Independent Helm versioning



Packaging charts

A chart is a directory. A Helm client can use chart directories on the same computer, but it's difficult to share with other users on other computers.

You package a chart by bundling the `chart.yaml` and related files into a .tar file and then installing the chart into a chart file:

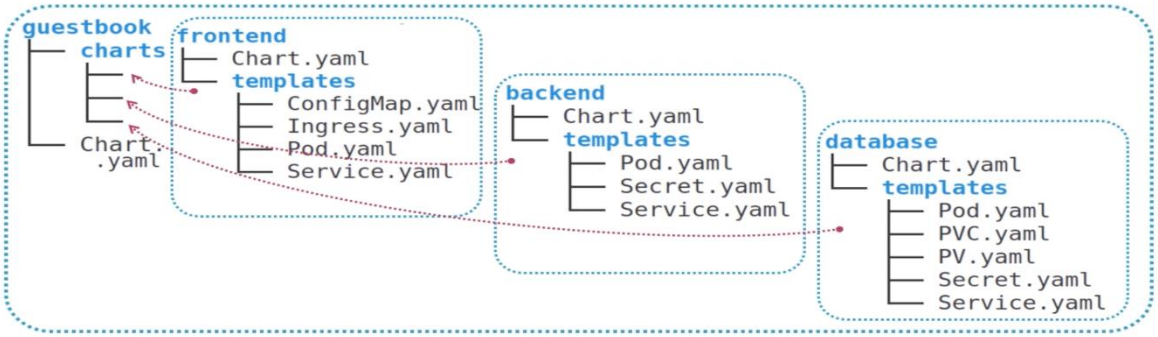
```
$ helm package <chart-path>

$ helm install <chart-name>.tgz
```

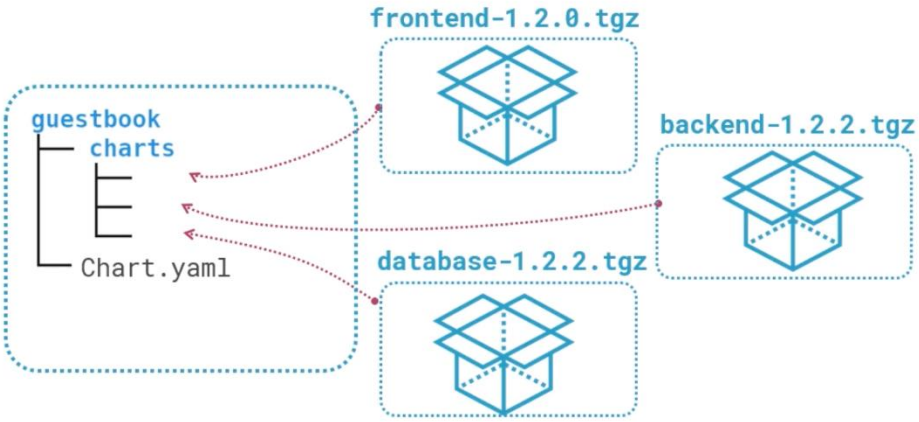
To add a chart to a repository, copy it to the directory and regenerate the index:

```
$ helm repo index <charts-path>  # Generates index of the charts in the repo
```

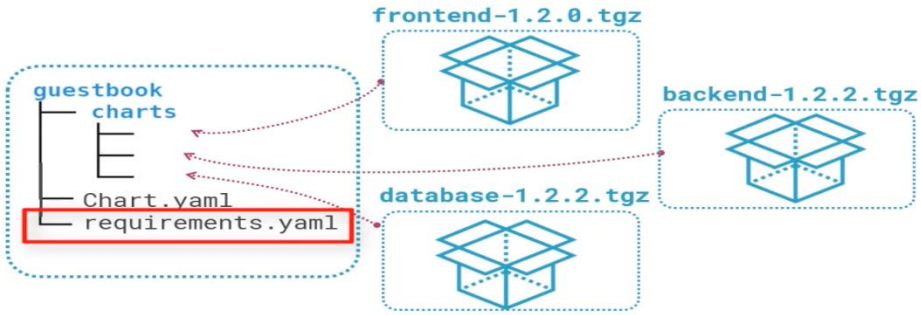
Charts Dependencies



Charts Dependencies



Charts Dependencies

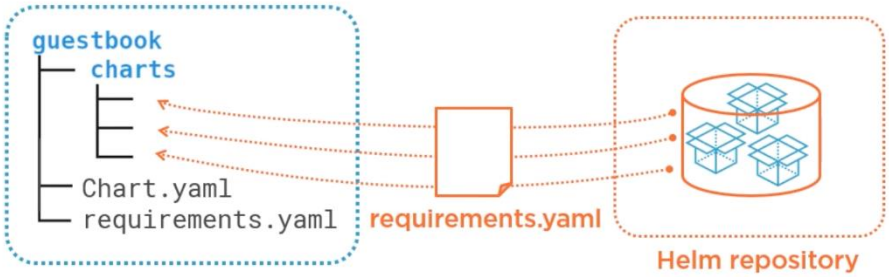


requirements.yaml

requirements.yaml

```yaml
dependencies:
  - name: backend
    version: ~1.2.2
    repository: http://127.0.0.1:8879/charts
  - name: frontend
    version: ^1.2.0
    repository: http://127.0.0.1:8879/charts
  - name: database
    version: ^1.2.2
    repository: http://127.0.0.1:8879/charts
```

Charts Dependencies




```
> helm dependency update guestbook
> helm dependency list guestbook
```

Conditions & Tags

requirements.yaml

```yaml
dependencies:
  - name: backend
    version: ~1.2.2
    repository: http://127.0.0.1:8879/charts
    condition: backend.enabled,global.backend.enabled
    tags:
      - api
  - name: frontend
    version: ^1.2.0
    repository: http://127.0.0.1:8879/charts
  - name: database
    version: ~1.2.2
    repository: http://127.0.0.1:8879/charts
    condition: database.enabled,global.database.enabled
    tags:
      - api
```

values.yaml

```yaml
backend:
  enabled: true
frontend:
database:
  enabled: false
```

Conditions & Tags

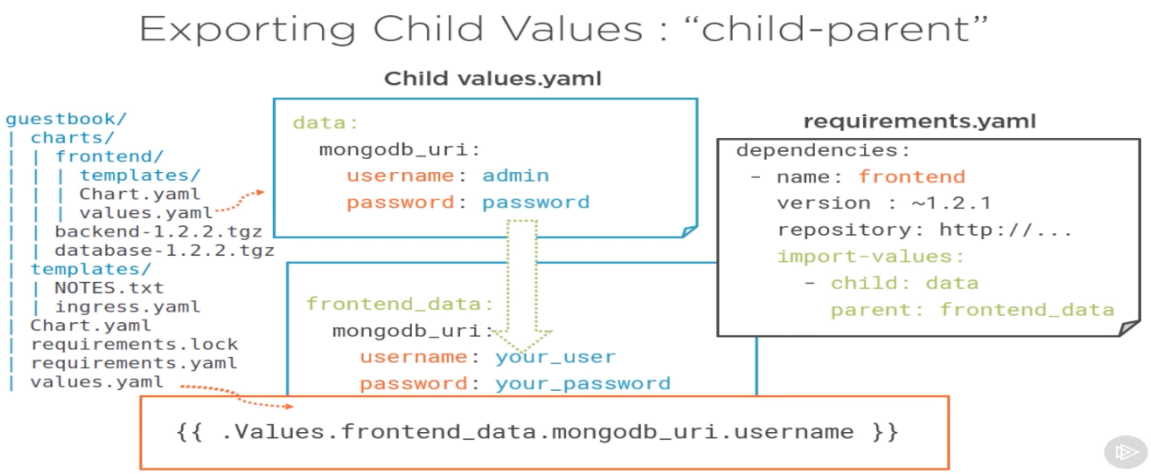
All charts are downloaded

Partial installation with helm install

--set override value.yaml

Conditions override tags

```
> helm dependency update guestbook
> helm install guestbook
> helm install guestbook --set database.enabled=true
> helm install guestbook --set tags.api=false
```



Helm Commands

Action	Command
Install a Release	`helm install [chart]`
Upgrade a Release revision	`helm upgrade [release] [chart]`
Rollback to a Release revision	`helm rollback [release] [revision]`
Print Release history	`helm history [release]`
Display Release status	`helm status [release]`
Show details of a release	`helm get [release]`
Uninstall a Release	`helm delete [release]`
List Releases	`helm list`

Deploy mariadb to Kubernetes using Helm

Install Helm

master $ curl -LO https://storage.googleapis.com/kubernetes-helm/helm-v2.8.2-linux-amd64.tar.gz
master $ mv linux-amd64/helm /usr/local/bin/
master $ helm init
master $ helm repo update

Search for Chart

$helm search mariadb

NAME	CHART VERSION	APP VERSION	DESCRIPTION
stable/mariadb	6.2.0	10.3.15	Fast, reliable, scalable, and easy to use open-...

stable/phpmyadmin 2.2.0 4.8.5 phpMyAdmin is an mysql administration frontend

master $ helm inspect stable/mariadb

Deploy mariadb

$helm install stable/mariadb

List the Release

master $ helm ls

```
NAME          REVISION    UPDATED                    STATUS     CHART        NAMESPACE
modest-zebra  1           Mon May 27 01:01:18 2019   DEPLOYED   mariadb-6.2.0  default
```

See Results

Helm deploys all the pods, replication controllers and services. Use *kubectl* to find out what was deployed.

master $ kubectl get all

```
NAME                                READY   STATUS    RESTARTS   AGE
pod/modest-zebra-mariadb-master-0   0/1     Pending   0          4m
pod/modest-zebra-mariadb-slave-0    0/1     Pending   0          4m

NAME                                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/kubernetes                  ClusterIP   10.96.0.1       <none>        443/TCP   6m
service/modest-zebra-mariadb        ClusterIP   10.107.56.151   <none>        3306/TCP  4m
service/modest-zebra-mariadb-slave  ClusterIP   10.106.168.174  <none>        3306/TCP  4m

NAME                                DESIRED   CURRENT   AGE
statefulset.apps/modest-zebra-mariadb-master   1         1         4m
statefulset.apps/modest-zebra-mariadb-slave    1         1         4m
```

The pod will be in a *pending* state while the Docker Image is downloaded and until a Persistent Volume is available.

master $ kubectl apply -f pv.yaml

```
persistentvolume/pv-volume1 created
persistentvolume/pv-volume2 created
persistentvolume/pv-volume3 created
```

mariadb needs permissions to write

master $ chmod 777 -R /mnt/data*

Once complete it will move into a *running* state. You'll now have a mariadb Cluster running on top of Kubernetes.

The helm could be provided with a friendlier name, such as:

master $ helm install --name my-release stable/mariadb

During installation, the helm client will print useful information about which resources were created, what the state of the release is, and also whether there are additional configuration steps we can or should take.

Now the **mariadb** chart is installed. Note that installing a chart creates a new release object. The release above is named limping-arachnid. (If we want to use our own release name, we can simply use the --name flag on helm install.)
Helm does not wait until all of the resources are running before it exits. To keep track of a release's state, or to re-read configuration information, we can use helm status:
$ helm status modest-zebra-mariadb

Helm delete - Deleting a release
When it is time to uninstall or delete a release from the cluster, use the helm deletecommand:
$ helm delete modest-zebra-mariadb
release " modest-zebra-mariadb " deleted

This will remove the release from the cluster. We can see all of our currently deployed releases with the helm list command:
$ helm list

As we can see from the output above, the limping-arachnid release was deleted.

However, Helm always keeps records of what releases happened. Need to see the deleted releases? helm list --deleted shows those, and helm list --all shows all of the releases (deleted and currently deployed, as well as releases that failed):
$ helm list –deleted
NAME REVISION UPDATED STATUS CHART NAMESPACE
modest-zebra-mariadb 1 Mon May 27 01:16:53 2019 DELETED mariadb-6.2.0 default