

Introduction to Helm

A Package Manager for Kubernetes



Agenda



- Docker vs Kubernetes
- Helm Introduction
- Helm 2 Vs Helm 3
- Installing Helm
- Helm Charts
- Helm Dependencies
- Helm Templates
- Helm Commands
- Helm Extras





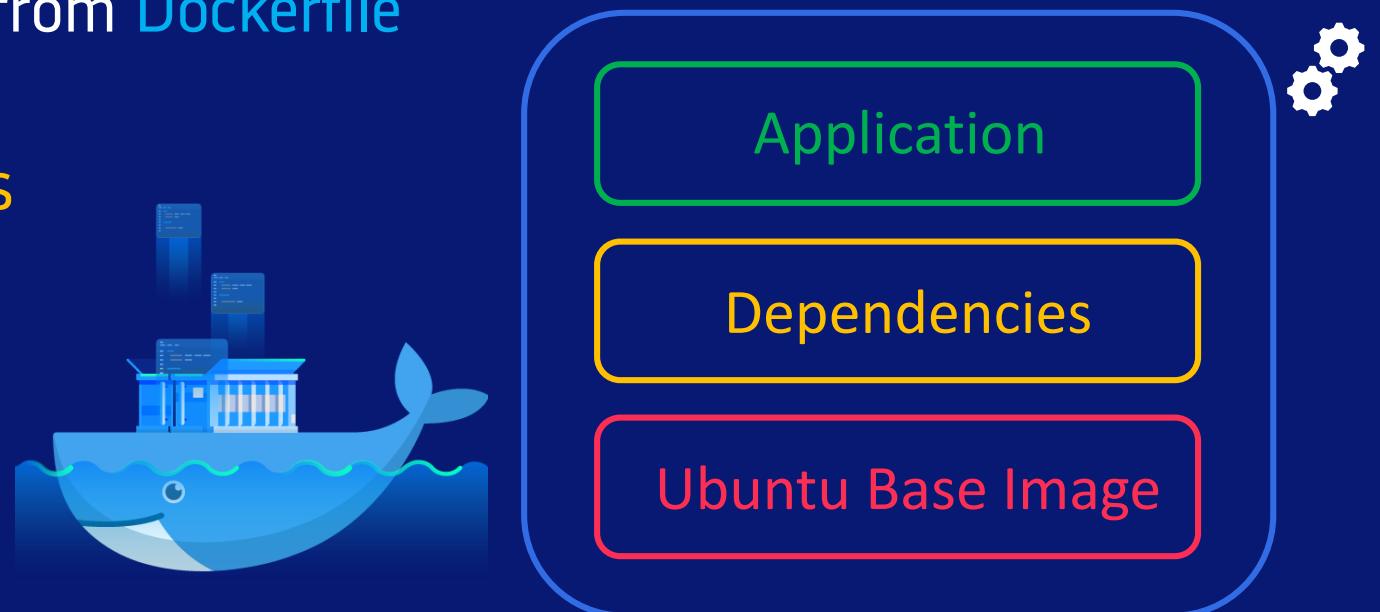
docker®



Docker

- Docker is a containerization engine, which provides blazing fast creation and management of containers. It makes it easy to develop, deploy, and manage applications by using containers
- Container refers to a lightweight, stand-alone, executable package of a piece of software that contains all the libraries, configuration files, dependencies, and other necessary parts to operate the application. They are the running instances of docker images
- Custom docker images can be built from Dockerfile

Ex: Ubuntu + Python + Dependencies



Kubernetes





Kubernetes

- Kubernetes also known as **K8s**, is an open-source Container Management tool
- It provides a container runtime, container orchestration, container-centric infrastructure orchestration, self-healing mechanisms, service discovery, load balancing and container (de)scaling
- Initially developed by Google, for managing containerized applications in a clustered environment but later donated to **CNCF**
- Written in **Golang**
- It is a platform designed to completely manage the life cycle of containerized applications and services using methods that provide predictability, scalability, and high availability.

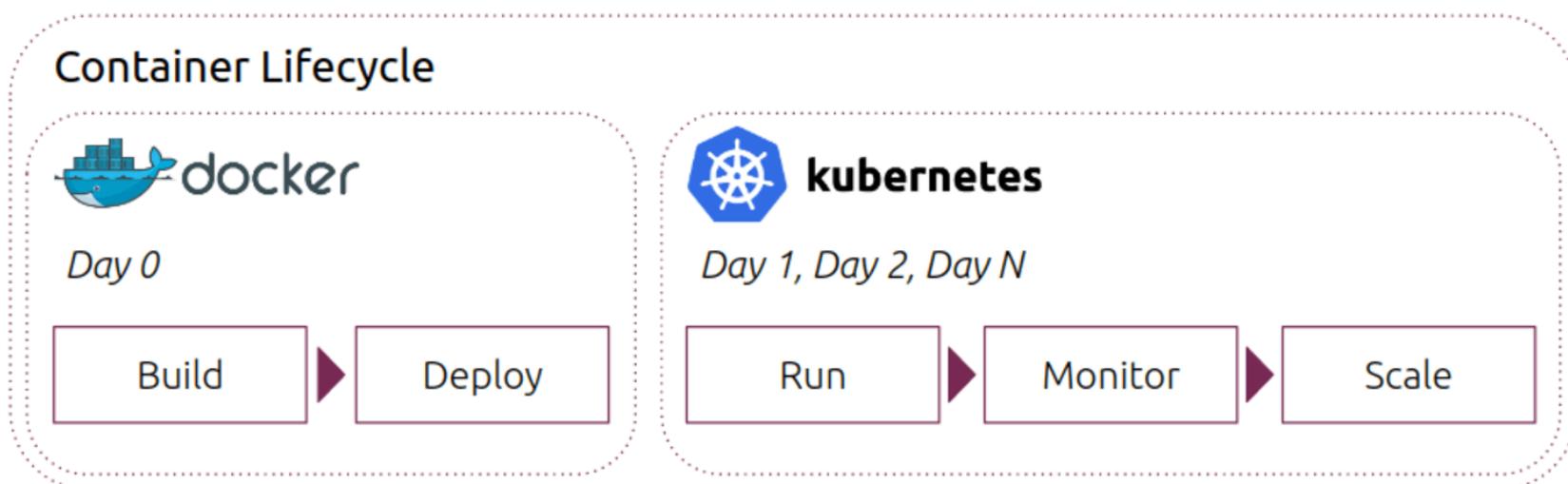




Docker vs Kubernetes

Docker is a computer software that is able to perform operating-system-level-virtualization, popularly known as containerization. It runs containerized applications.

Kubernetes is a container orchestration engine that makes it easy to automate container provisioning, networking, load-balancing, security and scaling applications across all the nodes in a multi host cluster. Kubernetes needs a Container Runtime Interface (CRI) to run containers. Although Docker is by far the most widely used container platform, other CRIs like rkt, containerd, LXC etc., can be used.

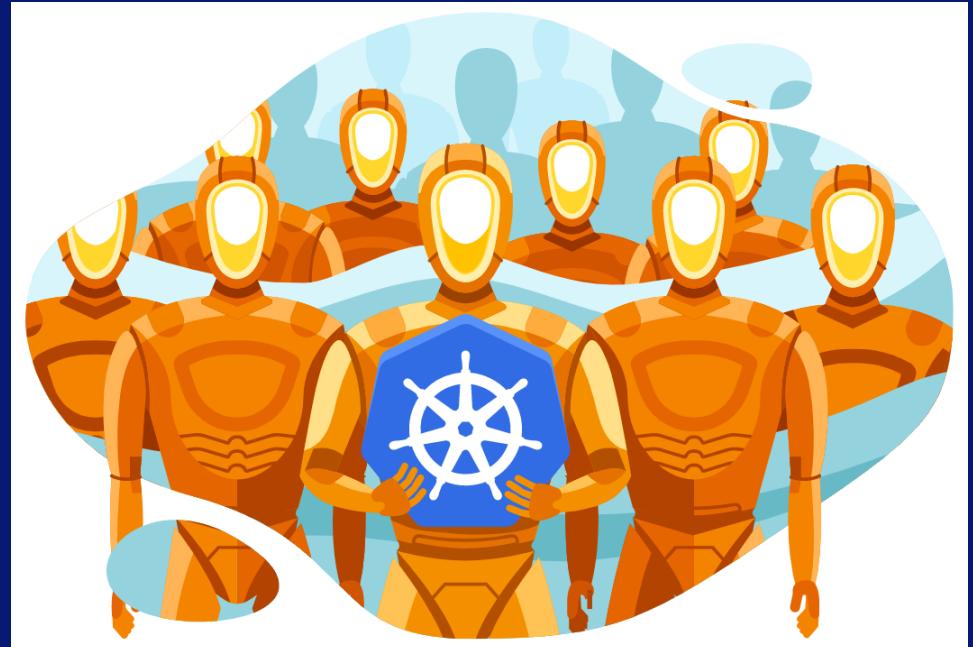


Kubernetes - Overview

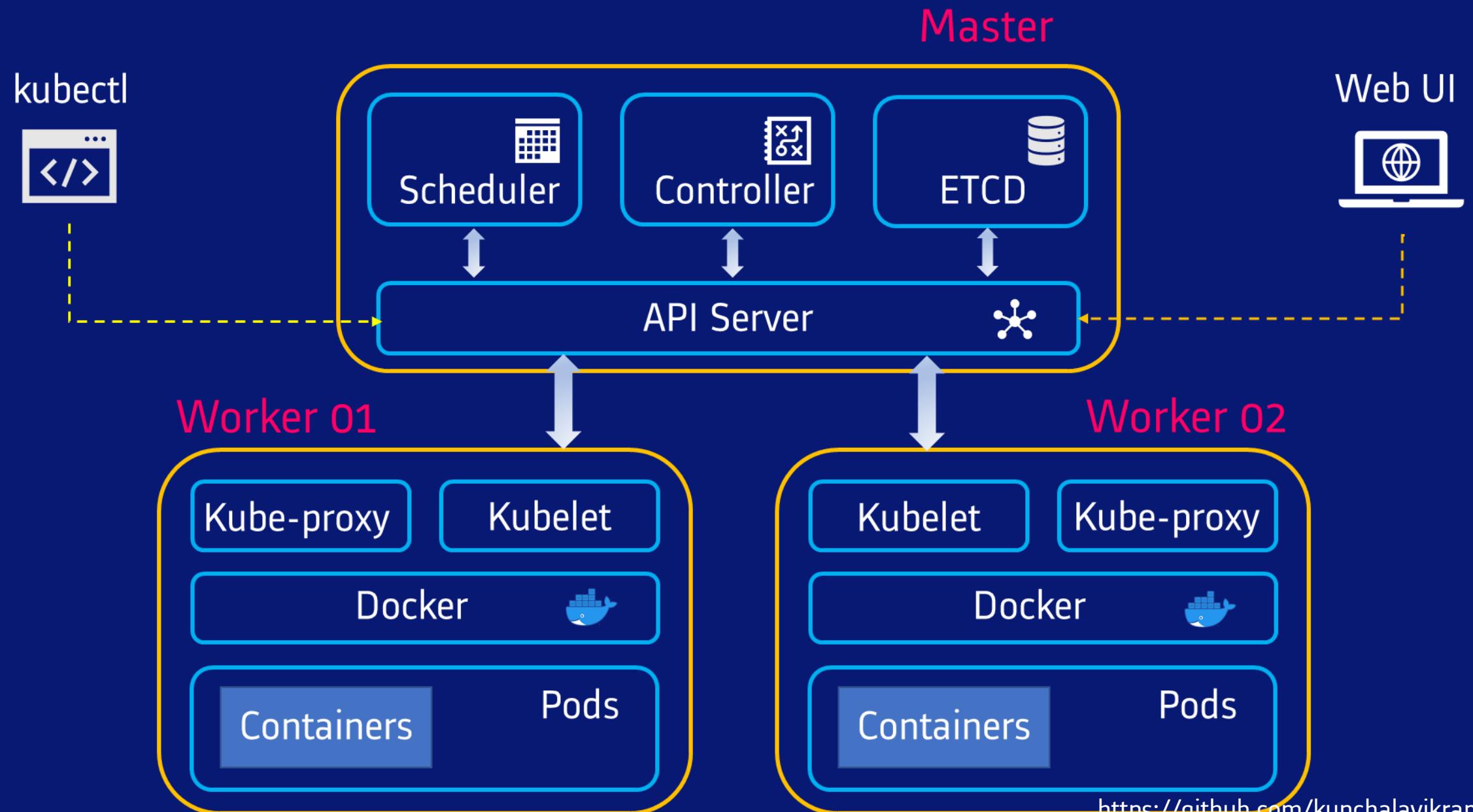
A Kubernetes cluster is a set of physical or virtual machines and other infrastructure resources that are needed to run your containerized applications. Each machine in a Kubernetes cluster is called a **node**.

There are two types of node in each Kubernetes cluster:

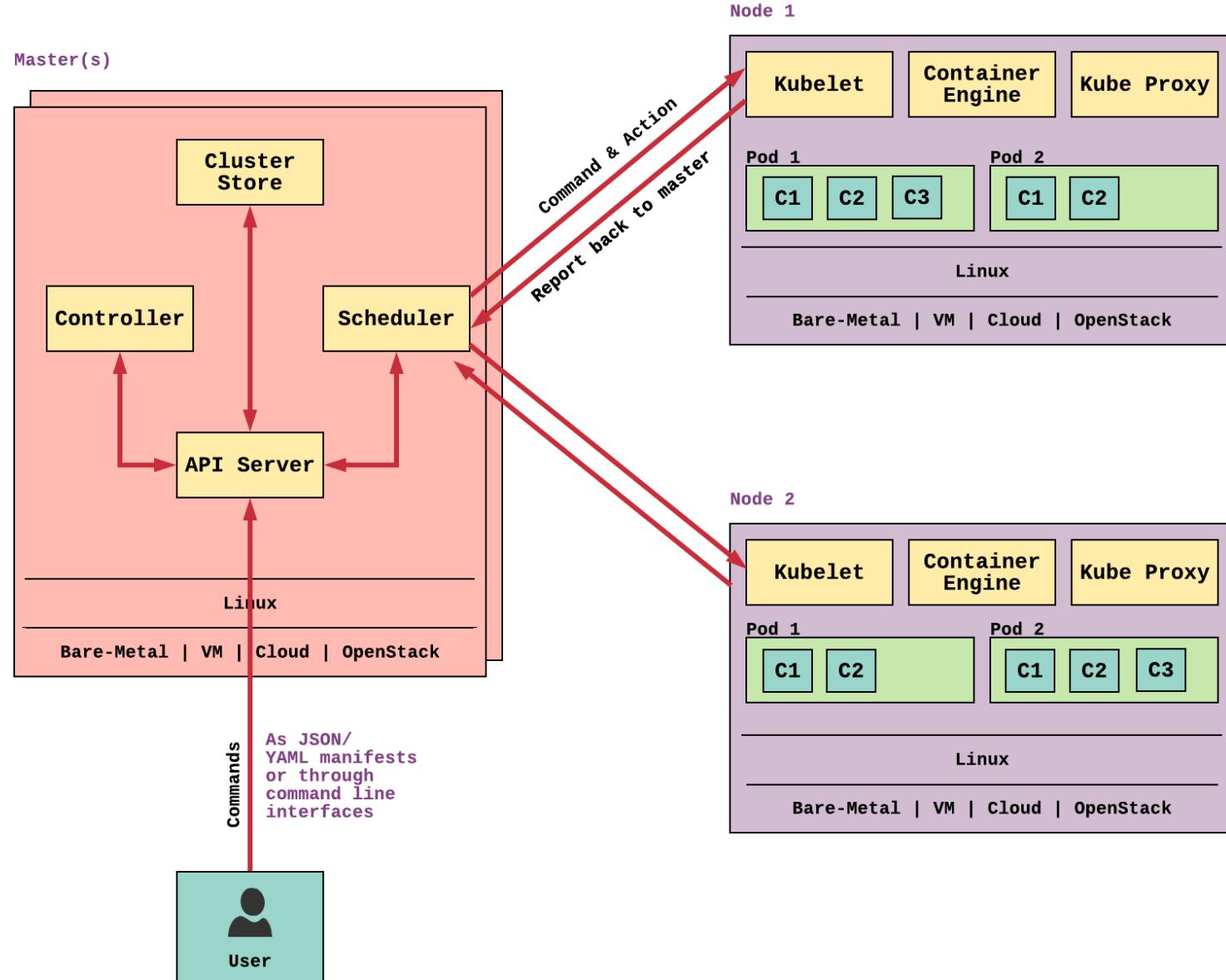
- Master node(s): hosts the Kubernetes control plane components and manages the cluster
- Worker node(s): runs your containerized applications



Kubernetes - Architecture

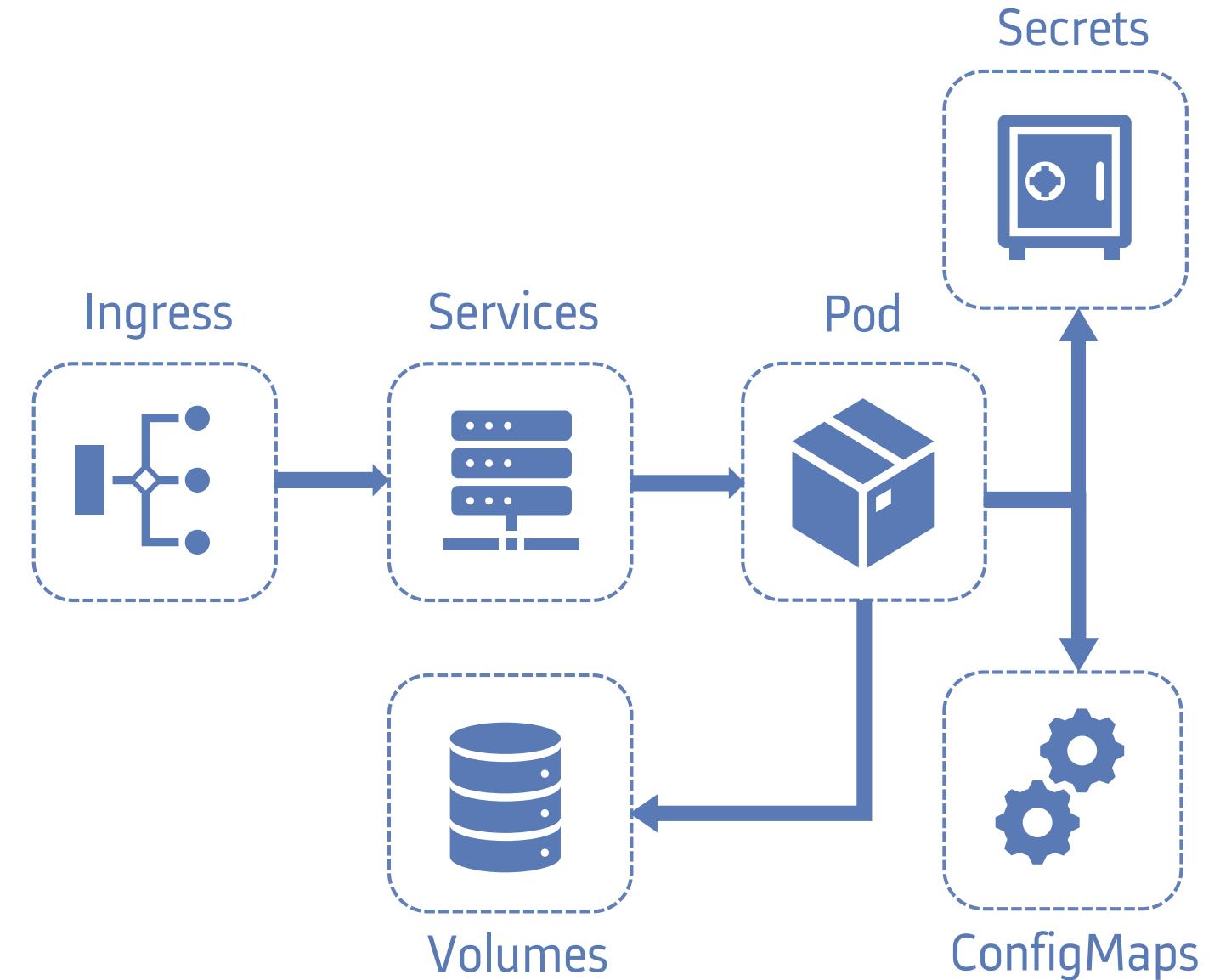


Kubernetes - Architecture



Kubernetes Resources

- Deployments
- Pods
- Services
- Ingress
- ConfigMaps
- Secrets
- Volumes: PV, PVC, storage
- DaemonSets
- RBAC
- Etc.,



Kubernetes – Manifest files

- We use yaml files to describe various objects in Kubernetes and to deployment applications in the cluster.

Deployment and Service yaml files

```
apiVersion: v1
kind: Service
metadata:
  name: connectedcity-service
spec:
  ports:
    - port: 80
      targetPort: 5000
  selector:
    app: connectedcity
```

Application-1
Deployment + ClusterIP service

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: connectedcity-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: connectedcity
  template:
    metadata:
      labels:
        app: connectedcity
    spec:
      containers:
        - name: connectedcity
          image: kunchalavikram/connectedcity:v1
          ports:
            - containerPort: 5000
```

```
apiVersion: v1
kind: Service
metadata:
  name: connectedfactory-service
spec:
  ports:
    - port: 80
      targetPort: 5000
  selector:
    app: connectedfactory
```

Application-2
Deployment + ClusterIP service

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: connectedfactory-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: connectedfactory
  template:
    metadata:
      labels:
        app: connectedfactory
    spec:
      containers:
        - name: connectedfactory
          image: kunchalavikram/connectedfactory:v1
          ports:
            - containerPort: 5000
```

kubectl apply -f <object>.yml

Kubernetes – Manifest files

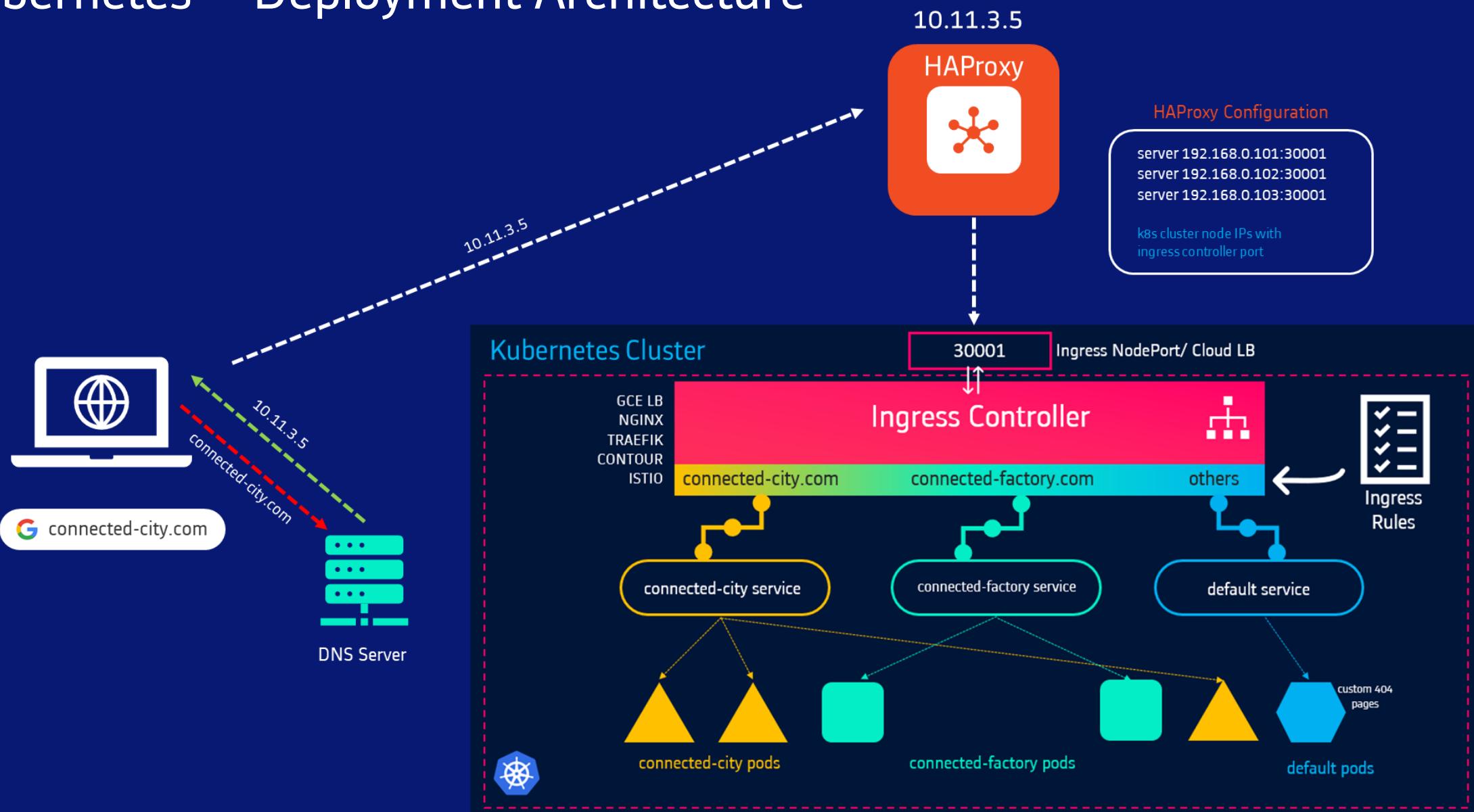
- We use yaml files to describe various objects in Kubernetes and to deployment applications in the cluster.

Ingress Rules yaml file

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-rules
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: connected-city.com
    http:
      paths:
      - backend:
          serviceName: connectedcity-service
          servicePort: 80
  - host: connected-factory.com
    http:
      paths:
      - backend:
          serviceName: connectedfactory-service
          servicePort: 80
```

kubectl apply -f <object>.yml

Kubernetes – Deployment Architecture



Deployment Limitations

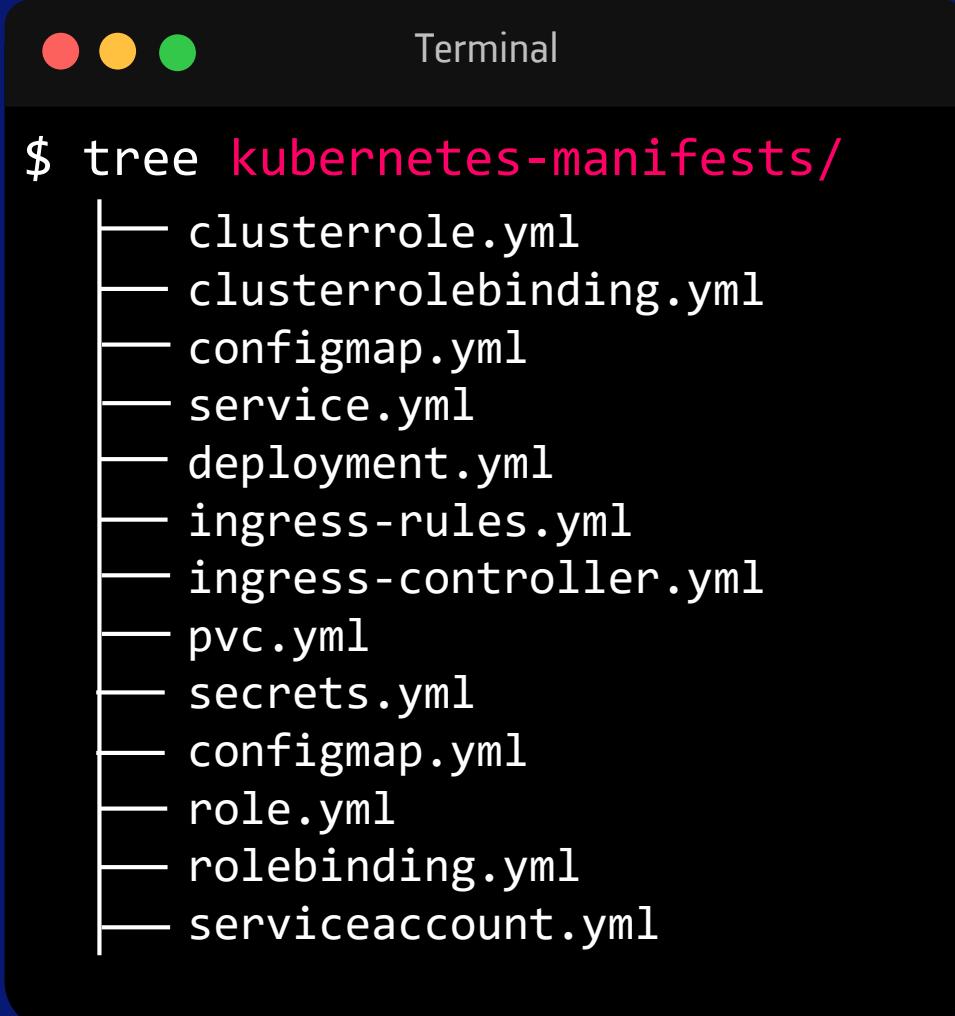
Kubernetes is a powerful tool, however, several challenges can hamper its adoption.

Deployment Complexity

- Deployment and management of applications on Kubernetes can become very complex with all the objects(YAML files) to be managed such as ConfigMaps, Services, Pods, Persistent Volumes. It can be tedious and error prone.

Impaired Developer Productivity

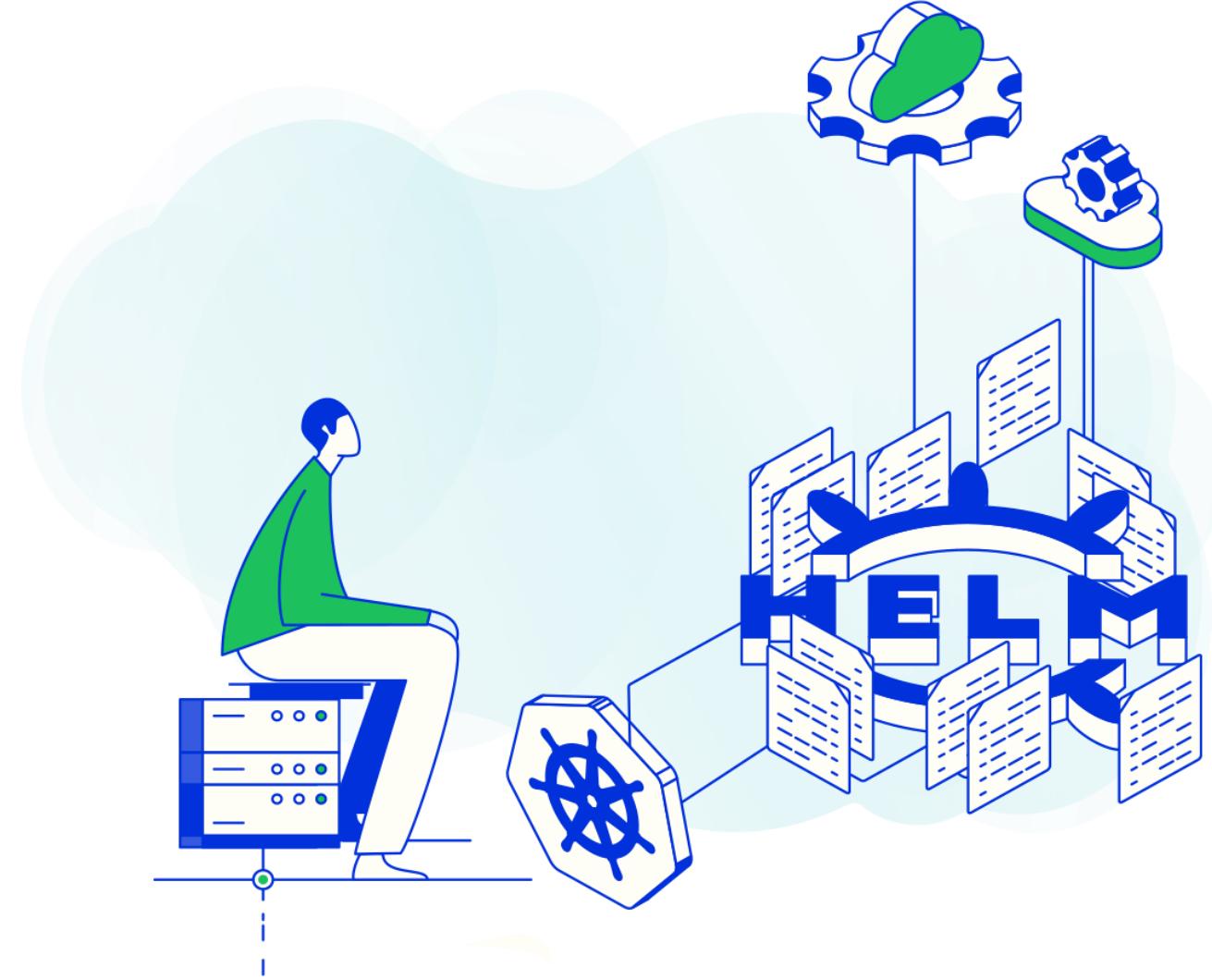
- Developers can spend a lot of time deploying test environments to test code and replicate customer issues.
- With Helm, developers can focus on developing applications instead of deploying dev-test environments.
- Helm Charts – such as MySQL, MongoDB and Postgresql – allow developers to get a working database quickly for their application.



```
$ tree kubernetes-manifests/
├── clusterrole.yml
├── clusterrolebinding.yml
├── configmap.yml
├── service.yml
├── deployment.yml
├── ingress-rules.yml
├── ingress-controller.yml
├── pvc.yml
├── secrets.yml
├── configmap.yml
├── role.yml
├── rolebinding.yml
└── serviceaccount.yml
```

A screenshot of a macOS terminal window titled "Terminal". The window shows the output of the "tree" command applied to a directory named "kubernetes-manifests". The output lists various Kubernetes configuration files, including clusterrole, clusterrolebinding, configmap, service, deployment, ingress-rules, ingress-controller, pvc, secrets, role, rolebinding, and serviceaccount files, each ending in ".yml".

Helm



What is Helm?

- **Helm** is a tool which allows to manage Kubernetes based applications by providing a standard approach for defining how we install, upgrade and uninstall applications
- It is a **package manager** for Kubernetes and the best way to find, share, and use software built for Kubernetes
- Allows developers to easily package, configure, and deploy applications and services onto Kubernetes clusters through **Helm Charts**
- Charts are files that are written in **YAML** and packaged in a particular directory structure which can be versioned within any source control systems. These charts describe a set of Kubernetes resources which makes up the entire application
- Helm is an official Kubernetes project and is part of the Cloud Native Computing Foundation(CNCF)
- Helm can:
 - ✓ Install software
 - ✓ Automatically install software dependencies
 - ✓ Upgrade software
 - ✓ Configure software deployments
 - ✓ Fetch software packages from repositories called **Charts repository**

Helm

Package Managers

	Package manager	Packages
SYSTEM	apt yum	deb rpm
DEVELOPMENT	maven npm pip	jar / java artifacts node modules python packages
KUBERNETES	helm	charts



A package manager is a collection of software tools that automates the process of installing, upgrading, configuring, and removing computer programs for a computer's operating system in a consistent manner.

Helm

Package Managers

apt package manager

apt install nginx
apt upgrade

Linux System

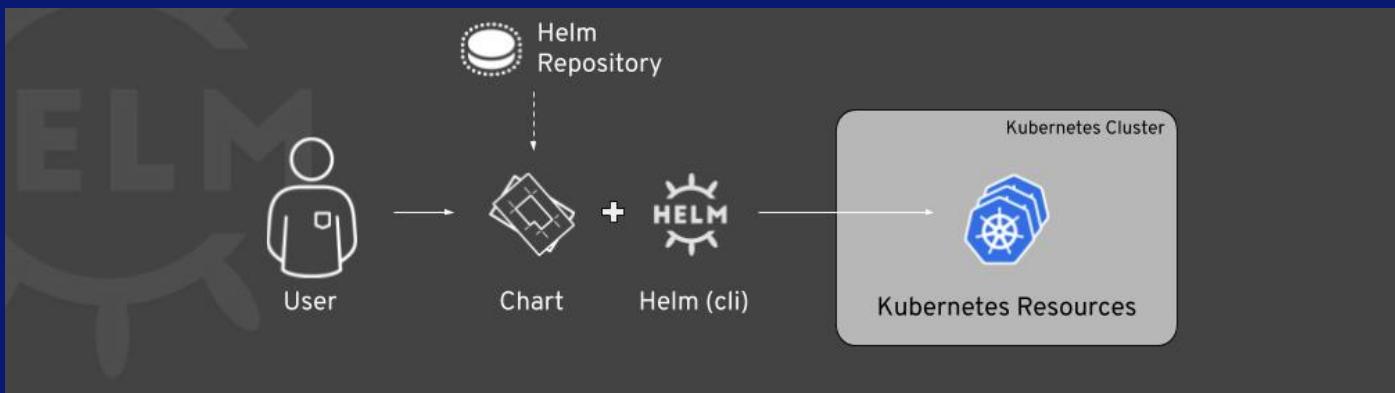
helm packages

helm install my-release bitnami/nginx
helm upgrade my-release bitnami/nginx

Kubernetes cluster

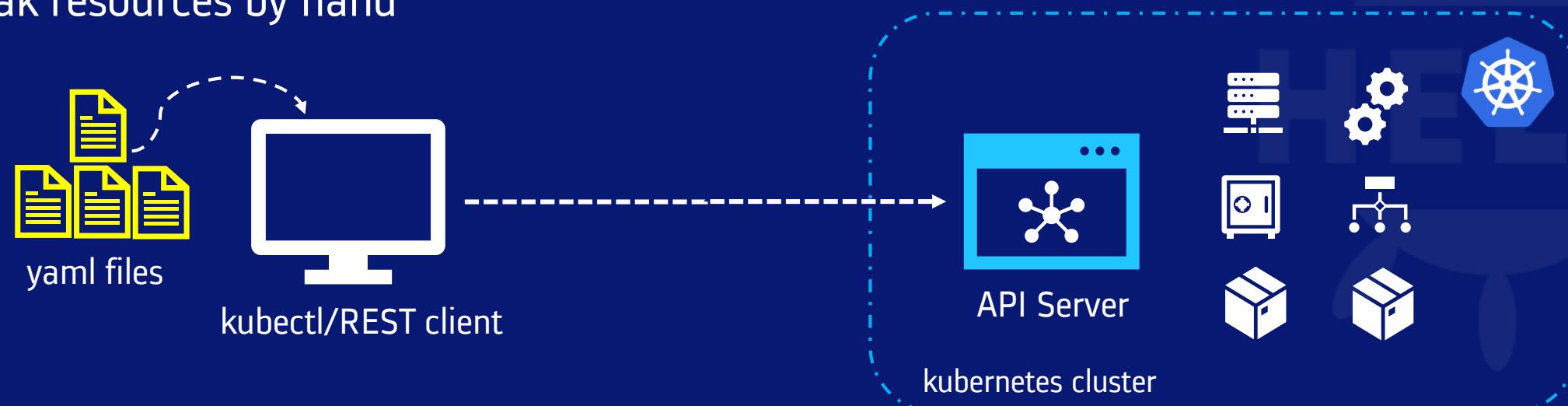
What makes Helm so popular?

- Offers a simple way to package all application resources into one single chart and advertises what you can configure
- Hiding the complexity of running multiple YAML files to deploy the application. Instead we run a single command that install all YAML files at once
- One click install/upgrade/rollback applications
- Easy management of application releases
- Same chart can be used for different environment like dev, staging and prod
- Reusable charts from helm repositories
- Latest version is v3.4.2



Without Helm

- Write and modify large set of Kubernetes manifests
- Do this every time a release is needed
- **No packaging** of files
- Rollbacks are possible at the YAML level. If an application has to be rolled back, all its dependent YAML files/resources are to be rolled back. Since **no version** is maintained for all resources, rollbacks are not easy
- No concept of application rollback since **no application versioning** is done in the cluster
- Figure out your own sharing: Git, offline etc
- Tweak resources by hand

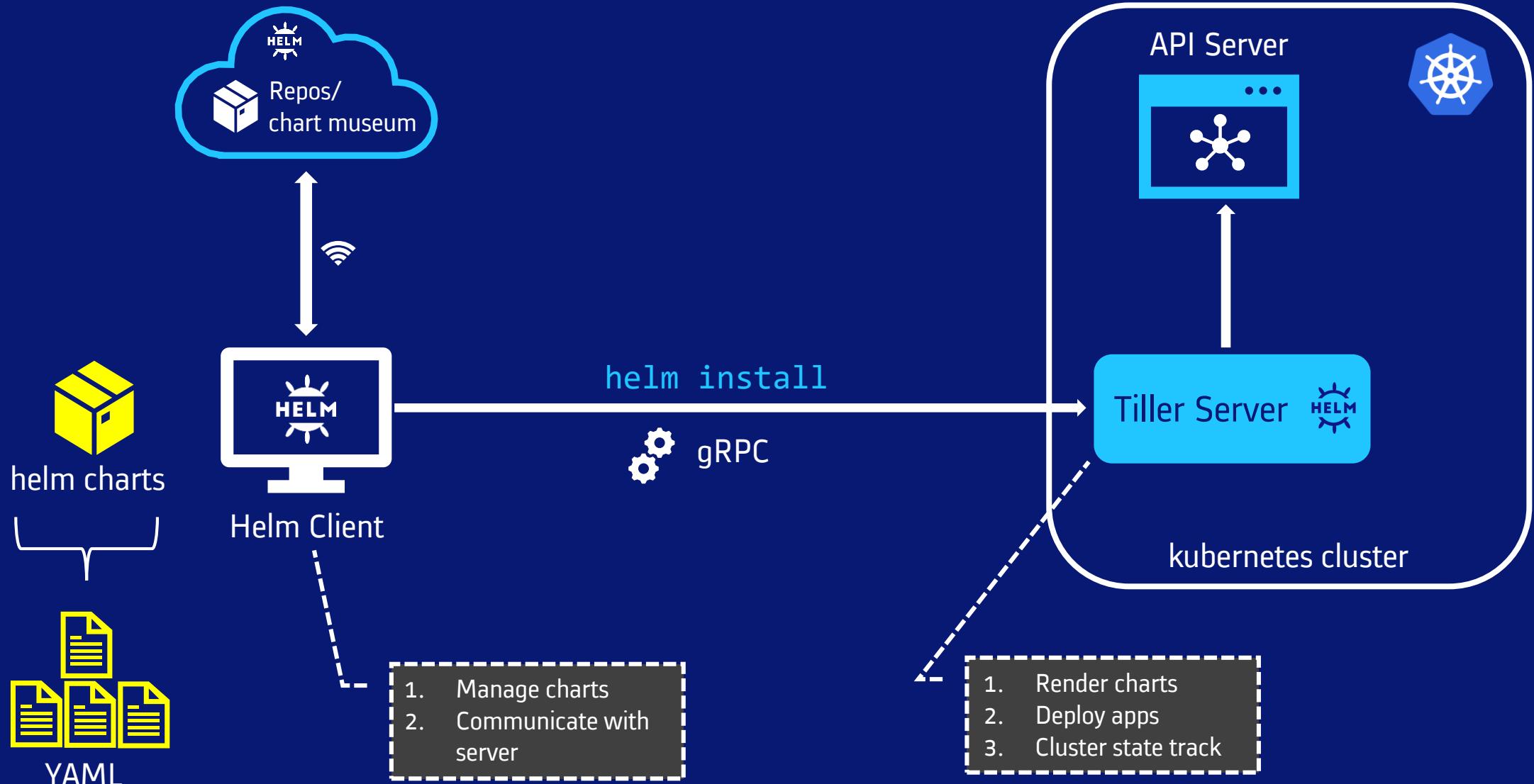


What Helm is NOT

- Full fledged **system** package manager
- Configuration management tool like Ansible, Chef, Puppet etc.
- Kubernetes resource lifecycle controller
- Version control system



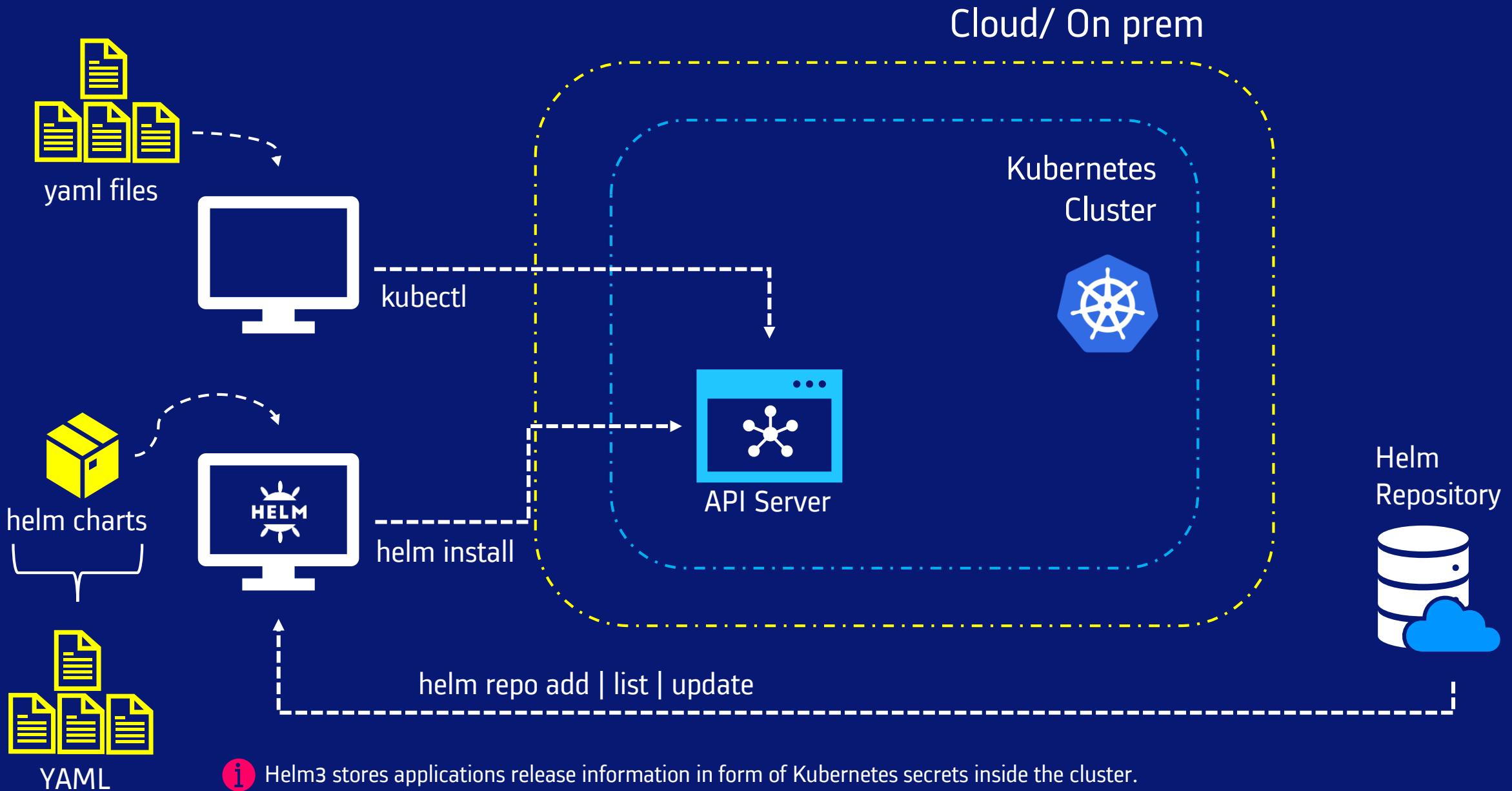
Helm2 Architecture



i Helm2 client uses gRPC protocol to communicate with Tiller server

Helm3 Architecture & Working

<https://github.com/kunchalavikram1427>



Helm3 stores applications release information in form of Kubernetes secrets inside the cluster.

Helm Common terminologies

Helm client/helm

- Helm Client is a command line tool for developing Charts, managing repositories and releases, and interfacing with the Helm Library.

Tiller

- Tiller is the Helm server.
- It interacts directly with the Kubernetes API server to install, upgrade, query, and remove Kubernetes resources. It is installed in the Kubernetes cluster and runs as a pod. **Tiller is removed in Helm3**

Chart

- It contains all of the resource definitions necessary to run an application, tool, or service inside of a Kubernetes cluster. A chart is basically a package of pre-configured Kubernetes resources.

Release

- An instance of a chart running in a Kubernetes cluster
- Same chart can be deployed multiple times in multiple environments

Repository

- Place where charts reside and can be shared with others

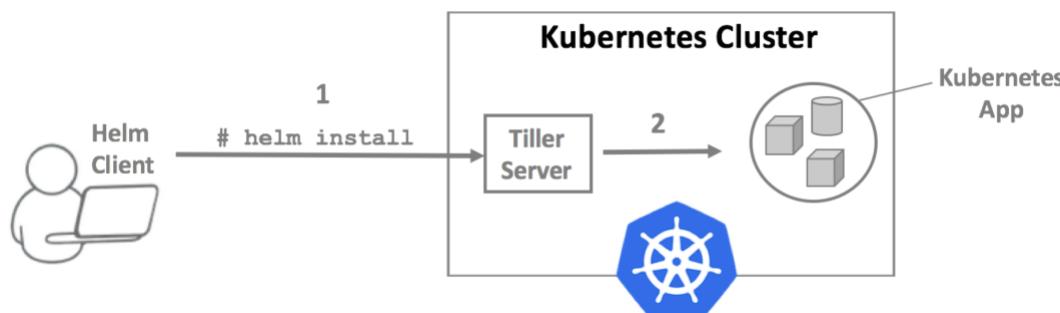
Helm2 Architecture

Helm Client

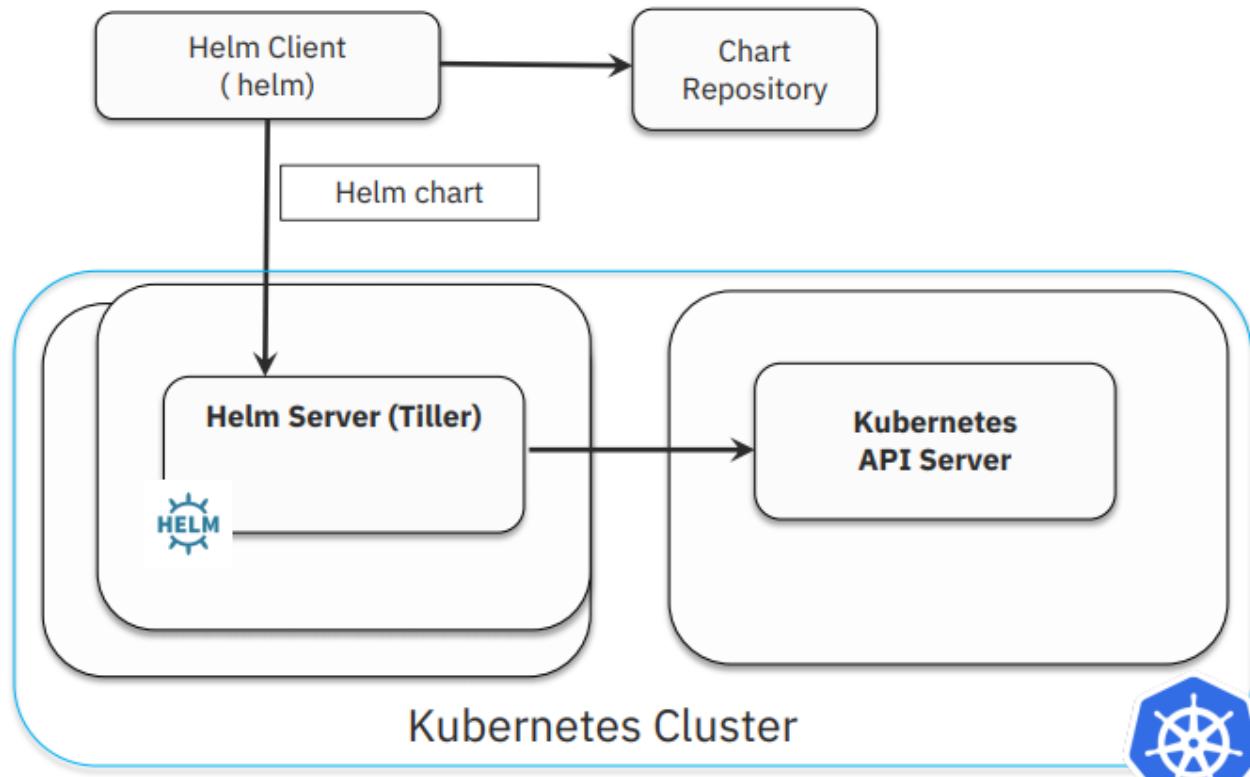
- Command-line client for end users
- Local chart development
- Managing repositories
- Interacts with the Tiller server by sending charts to be installed/upgrade/uninstall

Tiller Server

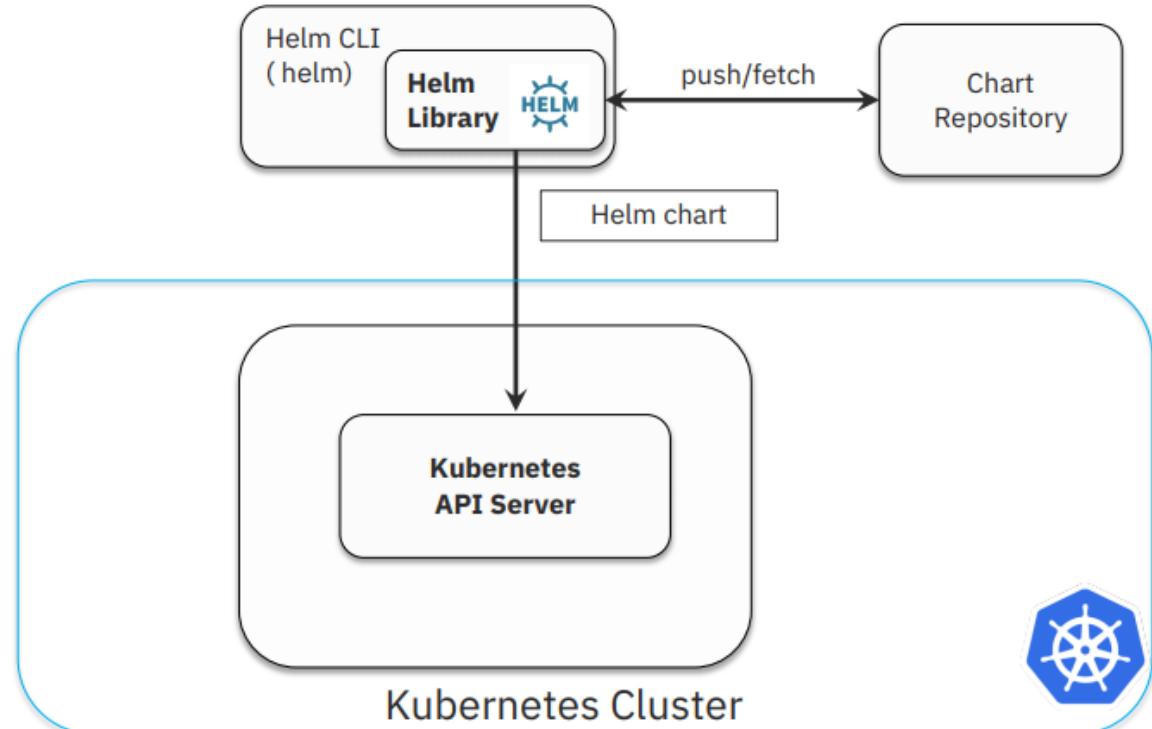
- Listening for incoming requests from the Helm client
- Combining a chart and configuration to build a release: Rendering templates
- Installing charts into Kubernetes, and then tracking the subsequent release
- Upgrading and uninstalling charts by interacting with Kubernetes



Helm2 vs Helm3



v2



v3

Helm3 – What changed from Helm2

Tiller removed in Helm3

- When Helm 2 was developed, Kubernetes did not have role-based access control (RBAC)
- So, Helm had to take care of authorizing who and what actions a user can perform in the cluster
- With Kubernetes 1.6 RBAC is enabled by default, so there is no need for Helm to do the same job which can be done natively by Kubernetes and that's why in Helm 3 tiller is removed completely

Three-way strategic merge patch

- In Helm 2 a patch would be generated by comparing the old manifest against the new manifest
- If someone manually changed the deployment, the helm has no info regarding that and rollbacks would give erroneous results
- In Helm 3, the patch is generated using the old manifest, the live state of the cluster, and the new manifest

Helm3 – What changed from Helm2

Secrets as the default storage driver

- Helm 2 used ConfigMaps to store release information. In Helm 3, Secrets are used instead as the default storage driver

JSON Schema Chart Validation

- A JSON Schema validation can now be forced upon chart values
- With this functionality, we can ensure that values provided by the user follow the schema created by the chart maintainer

Ex: Port number can be made mandatory and without which the chart cannot be installed

Release name is now required

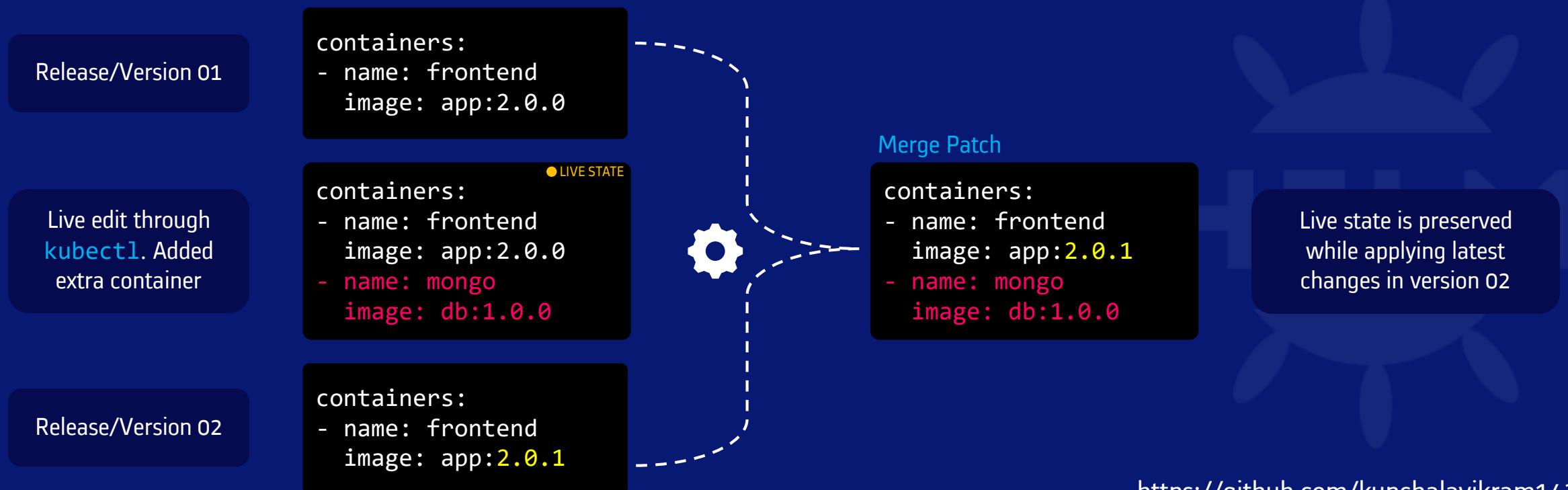
- In Helm 2, if no name was provided, a random release name would be generated.
- Helm 3 will throw an error if no name is provided with helm install



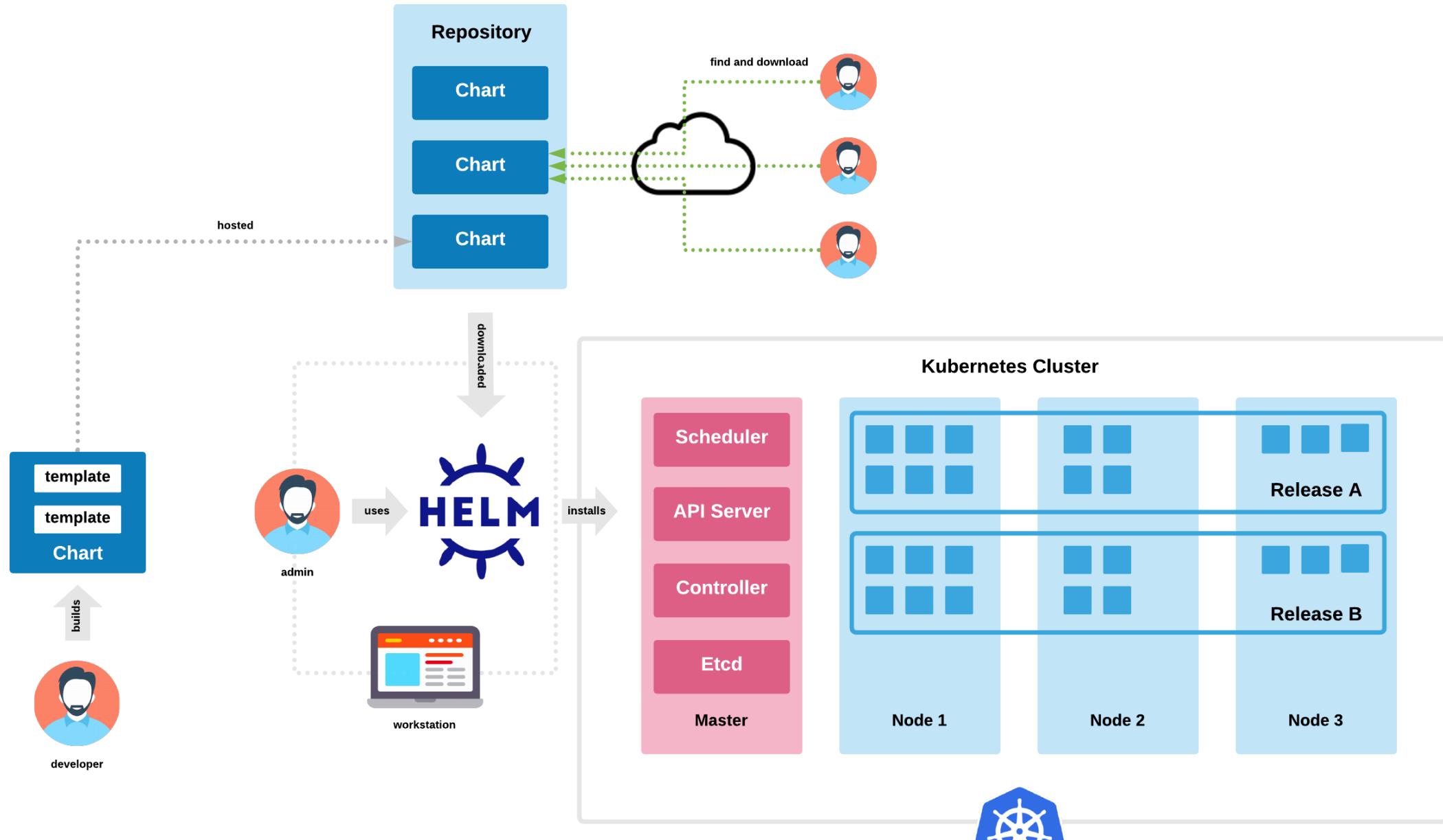
Helm 2 charts are compatible with Helm 3, although the chart structure might differ

Helm: Three-way strategic merge patch

- Helm 3 uses three way strategic merge patch
- It means if any helm operation is to be performed like version upgrade, it compares the most recent manifest chart(v1) against the proposed chart manifest(v2) and also the live state of the cluster
- So, any manual changes made to the cluster using `kubectl` (for example `kubectl edit`), will also be considered and changes are retained in the final release



Helm Architecture



Installing Helm



Installing Helm

From Source

- Download the desired version from <https://github.com/helm/helm/releases>
- Unpack the downloaded file. Ex:
`tar -zxvf helm-v3.0.0-linux-amd64.tar.gz`
- Find the helm binary in the unpacked directory, and move it to its desired destination
`mv linux-amd64/helm /usr/local/bin/helm`

Using Package Managers(apt)

```
curl https://baltocdn.com/helm/signing.asc | sudo apt-key add -
sudo apt-get install apt-transport-https --yes
echo "deb https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list
sudo apt-get update
sudo apt-get install helm
```

Check Helm Installation

Check version

- `helm version`
- `helm version --short`

```
root@k8s-master:/home/osboxes# helm version
version.BuildInfo{Version:"v3.3.0", GitCommit:"8a4aeecc08d67a7b84472007529e8097ec3742105", GitTreeState:"dirty", GoVersion:"go1.14.7"}
root@k8s-master:/home/osboxes# helm version --short
v3.3.0+g8a4aeecc
root@k8s-master:/home/osboxes#
```

Man pages/help

- `helm help`

```
root@k8s-master:/home/osboxes# helm help | more
The Kubernetes package manager

Common actions for Helm:

- helm search:      search for charts
- helm pull:        download a chart to your local directory to view
- helm install:     upload the chart to Kubernetes
- helm list:        list releases of charts
```



Helm defaults directories & options

Default environment variables

`$KUBECONFIG` - set an alternative Kubernetes configuration file (default "`~/.kube/config`")

`$HELM_KUBEAPISERVER` - set the Kubernetes API Server Endpoint for authentication

Default default directories

Operating System	Cache Path	Configuration Path	Data Path
Linux	<code>\$HOME/.cache/helm</code>	<code>\$HOME/.config/helm</code>	<code>\$HOME/.local/share/helm</code>
macOS	<code>\$HOME/Library/Caches/helm</code>	<code>\$HOME/Library/Preferences/helm</code>	<code>\$HOME/Library/helm</code>
Windows	<code>%TEMP%\helm</code>	<code>%APPDATA%\helm</code>	<code>%APPDATA%\helm</code>

- i To view kubectl config file, run: `kubectl config view`

Helm defaults directories & options

Environment information in use by Helm

`helm env`

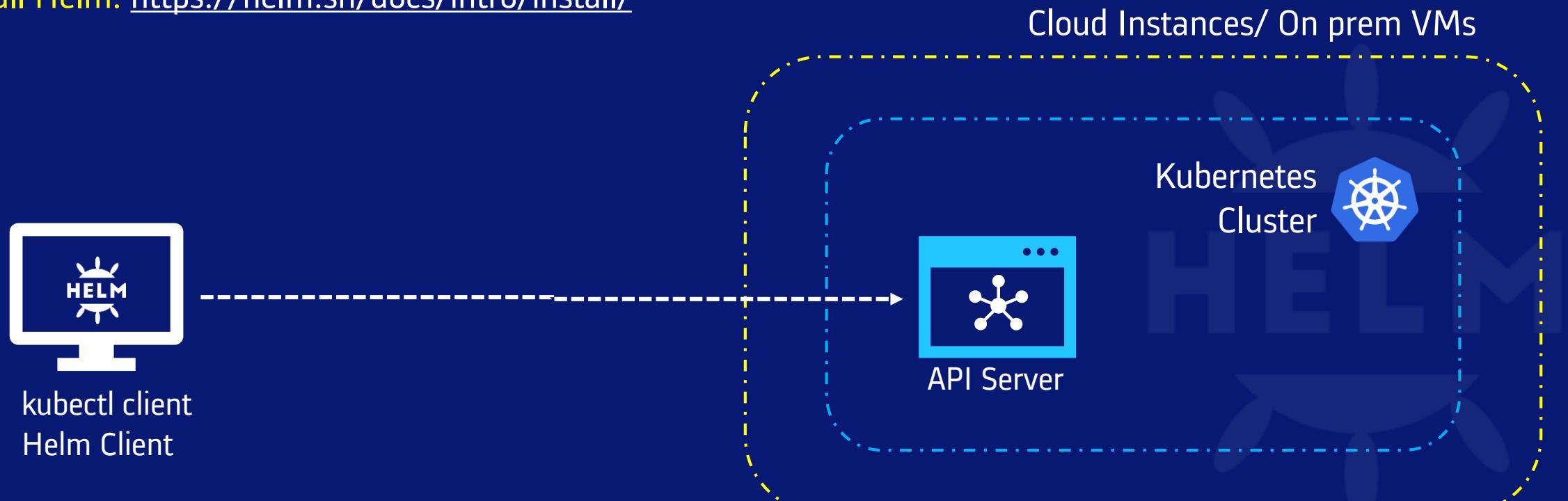
```
root@k8s-master:/home/osboxes# helm env
HELM_BIN="helm"
HELM_CACHE_HOME="/root/.cache/helm"
HELM_CONFIG_HOME="/root/.config/helm"
HELM_DATA_HOME="/root/.local/share/helm"
HELM_DEBUG="false"
HELM_KUBEAPISERVER=""
HELM_KUBECONTEXT=""
HELM_KUBETOKEN=""
HELM_NAMESPACE="default"
HELM_PLUGINS="/root/.local/share/helm/plugins"
HELM_REGISTRY_CONFIG="/root/.config/helm/registry.json"
HELM_REPOSITORY_CACHE="/root/.cache/helm/repository"
HELM_REPOSITORY_CONFIG="/root/.config/helm/repositories.yaml"
root@k8s-master:/home/osboxes#
```



To uninstall Helm completely, remove all the cache files stored in the above shown directories as well, along with removing the helm binary

Kubernetes cluster setup

- Cluster setup:
 - minikube <https://minikube.sigs.k8s.io/docs/start/> (or)
 - kubectl
https://github.com/kunchalavikram1427/Kubernetes_public/blob/master/Bootstrap_K8s_Cluster_Kubeadm.pdf
- Install kubectl utility: <https://kubernetes.io/docs/tasks/tools/install-kubectl/>
- Install Helm: <https://helm.sh/docs/intro/install/>



Helm Repositories



Helm Chart Repos

- Chart repository is a location where packaged charts can be stored and shared
- The official chart repository is maintained by the Kubernetes Charts, but Helm also makes it easy to create and run your own chart repository

```
helm repo add bitnami https://charts.bitnami.com/bitnami  
helm repo update  
helm install my-nginx bitnami/nginx --version 8.2.1
```

The screenshot shows the Artifact Hub website interface. At the top, there is a navigation bar with the logo 'Artifact HUB' (BETA), a search bar labeled 'Search packages', and user options 'SIGN UP' and 'SIGN IN'. Below the header, a breadcrumb link 'Back to "nginx" results' is visible. The main content area features a card for the 'nginx' chart. The card includes a circular icon with the word 'NGINX' inside, the name 'nginx', the organization 'ORG: Bitnami', the repository 'REPO: Bitnami', a star icon with the number '3', and a description 'Chart for the nginx server'. At the bottom of the card, the word 'NGINX' is displayed in large letters, followed by a detailed description of the chart's purpose: 'NGINX (pronounced "engine-x") is an open source reverse proxy server for HTTP, HTTPS, SMTP, POP3, and IMAP protocols, as well as a load balancer, HTTP cache, and a web server (origin server)'. To the right of the description are three buttons: 'INSTALL', 'VALUES SCHEMA', and 'CHANGELOG'.

Helm Chart Repos

```
helm repo add center https://repo.chartcenter.io
helm repo update
helm install ingress-nginx center/kubernetes-ingress-nginx/ingress-nginx
```

The screenshot shows the JFrog ChartCenter interface. At the top, there's a search bar with the query "kubernetes-ingress-nginx/ingress-nginx". Below it, a banner indicates "41,336 versions indexed". To the right of the search bar are links for "Join Slack", "News", "Docs", "Blog", and "Github". A "JFrog" logo is also present.

The main content area displays the "kubernetes-ingress-nginx/ingress-nginx" chart. It includes the following details:

- Name:** kubernetes-ingress-nginx/ingress-nginx
- Chart version:** 3.18.0
- Api version:** v2
- App version:** 0.42.0
- Type:** Application
- Chart Type:** Unknown
- Status:** Active
- Downloads:** 46820

A description below the chart states: "Ingress controller for Kubernetes using NGINX as a reverse proxy...".

On the right side of the chart card, there are two boxes with instructions:

- "Set me up:" followed by the command: `helm repo add center https://repo.chartcenter.io`
- "Install Chart:" followed by the command: `helm install ingress-nginx center/kubernetes-ingress-nginx/ingress-nginx`

At the bottom right of the page, there's a footer with a "Privacy - Terms" link and a small logo.

Helm

Adding helm repos

- `helm repo add stable https://charts.helm.sh/stable`
- `helm repo add bitnami https://charts.bitnami.com/bitnami`
- `helm repo list`
- `helm repo remove <repo-name>`

```
root@k8s-master:/home/osboxes# helm repo add stable https://charts.helm.sh/stable
"stable" has been added to your repositories
root@k8s-master:/home/osboxes# helm repo add bitnami https://charts.bitnami.com/bitnami
"bitnami" has been added to your repositories
root@k8s-master:/home/osboxes# helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "bitnami" chart repository
...Successfully got an update from the "stable" chart repository
Update Complete. * Happy Helm-ing!*
root@k8s-master:/home/osboxes# helm repo list
NAME      URL
stable   https://charts.helm.sh/stable
bitnami  https://charts.bitnami.com/bitnami
root@k8s-master:/home/osboxes#
```

Search for charts

- `helm repo update`
- `helm search repo mysql`

NAME	CHART VERSION	APP VERSION	DESCRIPTION
bitnami/mysql	8.2.3	8.0.22	Chart to create a Highly available MySQL cluster
stable/mysql	1.6.9	5.7.30	DEPRECATED - Fast, reliable, scalable, and easy...
stable/mysqldump	2.6.2	2.4.1	DEPRECATED! - A Helm chart to help backup MySQL...
stable/prometheus-mysql-exporter	0.7.1	v0.11.0	DEPRECATED A Helm chart for prometheus mysql ex...
bitnami/phpmyadmin	8.0.2	5.0.4	phpMyAdmin is an mysql administration frontend
stable/percona	1.2.3	5.7.26	DEPRECATED - free, fully compatible, enhanced, ...
stable/percona-xtradb-cluster	1.0.8	5.7.19	DEPRECATED - free, fully compatible, enhanced, ...
stable/phpmyadmin	4.3.5	5.0.1	DEPRECATED phpMyAdmin is an mysql administratio...
bitnami/mariadb	9.2.0	10.5.8	Fast, reliable, scalable, and easy to use open...
bitnami/mariadb-cluster	1.0.2	10.2.14	DEPRECATED Chart to create a Highly available M...
bitnami/mariadb-galera	5.3.4	10.5.8	MariaDB Galera is a multi-master database clust...
stable/gcloud-sqlproxy	0.6.1	1.11	DEPRECATED Google Cloud SQL Proxy
stable/mariadb	7.3.14	10.3.22	DEPRECATED Fast, reliable, scalable, and easy t...

Hosting own Helm Chart Repository

ChartMuseum

ChartMuseum is an open-source Helm Chart Repository written in Go (Golang), with support for cloud storage backends, including Google Cloud Storage, Amazon S3, Microsoft Azure Blob Storage etc.

```
docker run --rm -it \
-p 8080:8080 \
-v $(pwd)/charts:/charts \
-e DEBUG=true \
-e STORAGE=local \
-e STORAGE_LOCAL_ROOTDIR=/charts \
chartmuseum/chartmuseum:latest
```



Helm Charts

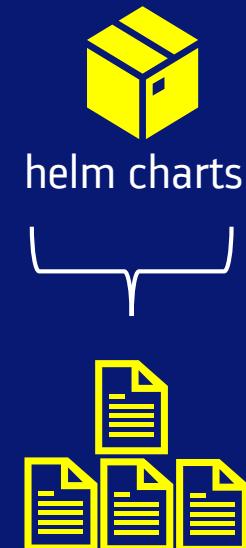


Helm Charts

- Helm Charts are simply Kubernetes YAML manifests combined into a single package(archive) that can be installed in the Kubernetes cluster
- Also contains templates that are rendered into Kubernetes manifest files
- Once packaged, installing a Helm Chart into your cluster is as easy as running a single `helm install`, which really simplifies the deployment of containerized applications
- Can be packaged into versioned objects for sharing (repos). Ex: mychart-0.1.0.tgz, mychart-0.1.1.tgz
- Version numbers follows Semantic Versioning 2.0.0 specification: `major.minor.patch`
- Charts can have dependencies on other charts
- By default, directory name is considered as chart name

Generate chart template

```
helm create example-chart
```



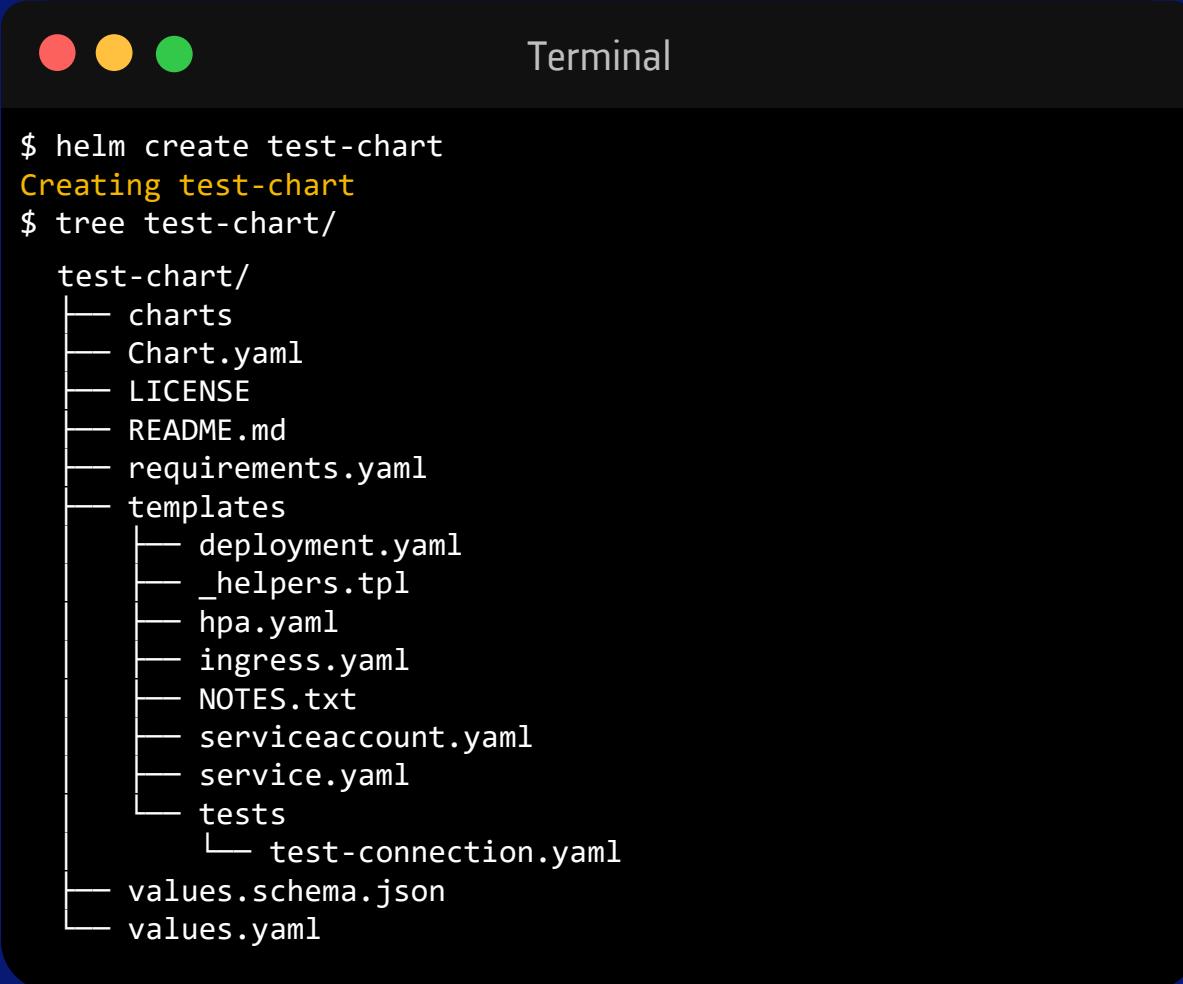
YAML

<https://github.com/kunchalavikram1427>

Helm Charts

Generate chart template

```
helm create test-chart
```



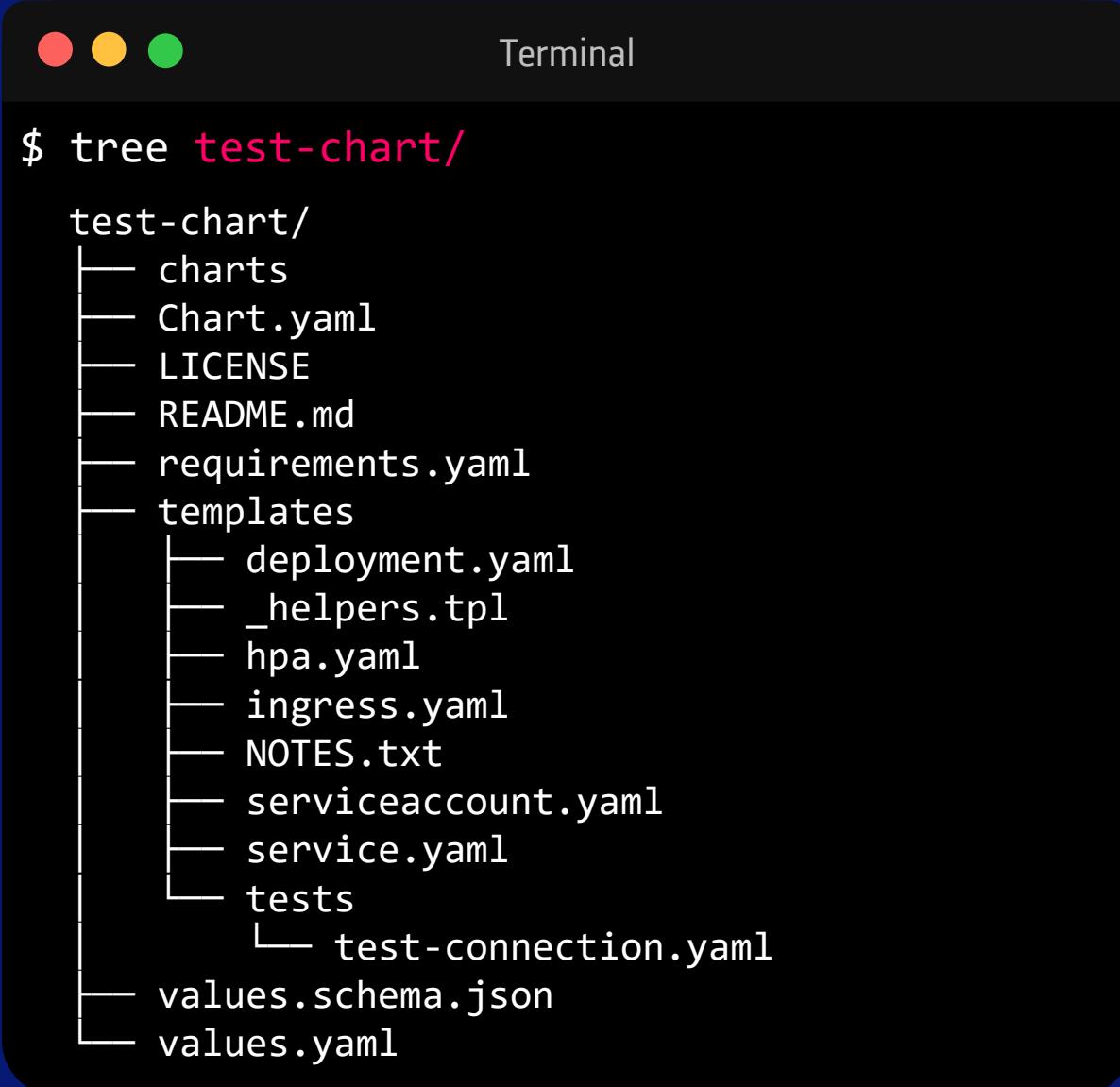
A terminal window titled "Terminal" with three colored window controls (red, yellow, green) at the top. The window contains the following text:

```
$ helm create test-chart
Creating test-chart
$ tree test-chart/
test-chart/
├── charts
├── Chart.yaml
├── LICENSE
├── README.md
├── requirements.yaml
└── templates
    ├── deployment.yaml
    ├── _helpers.tpl
    ├── hpa.yaml
    ├── ingress.yaml
    ├── NOTES.txt
    ├── serviceaccount.yaml
    ├── service.yaml
    └── tests
        └── test-connection.yaml
└── values.schema.json
└── values.yaml
```



Helm Charts

- **helmignore**: holds all the files to ignore when packaging the chart. Similar to `.gitignore` for git
- **Chart.yaml**: holds the metadata about the chart, such as chart name and version, maintainer information, a relevant website, and search keywords
- **Charts**: stores other charts that main chart depends on.
- **Values.yaml**: holds all the values to be injected into the templates. Templates are rendered into kubernetes manifest files
- **Values.schema.json**: validates `values.yaml` file. Removing any required fields will result in an error as JSON schema validation fails. This check happens everytime we run helm install/upgrade/template



A terminal window titled "Terminal" displays the command `$ tree test-chart/` followed by the directory structure of a Helm chart named "test-chart". The structure includes a top-level folder "test-chart" containing "charts", "Chart.yaml", "LICENSE", "README.md", "requirements.yaml", "templates" (which contains "deployment.yaml", "_helpers.tpl", "hpa.yaml", "ingress.yaml", "NOTES.txt", "serviceaccount.yaml", "service.yaml"), "tests" (which contains "test-connection.yaml"), "values.schema.json", and "values.yaml".

```
$ tree test-chart/
test-chart/
├── charts
├── Chart.yaml
├── LICENSE
├── README.md
├── requirements.yaml
├── templates
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── NOTES.txt
│   ├── serviceaccount.yaml
│   └── service.yaml
└── tests
    └── test-connection.yaml

2 directories, 11 files
```

Helm Charts

- **Templates:** holds the actual manifests to be deployed with the chart. For example deployment.yaml, service.yaml, config.yaml and secrets.yaml all in the templated format. They will all get their values from values.yaml from above.
- **LICENSE:** A plaintext license for the chart
- **NOTES.txt file:** prints on the command line when the chart is installed
- **_helpers.tpl:** handles templates
- **README.md:** A readme file with information for users of the chart.
- **requirements.yaml:** A YAML file that lists the chart's dependencies.

 The files whose names begin with an underscore ('_') are assumed to not have a manifest inside, so they are not turned into Kubernetes API resource definitions. However, they are available everywhere within other chart templates for use. These files are conventionally used to store sub-templates and helpers. _helpers.tpl is the default location for small sub-templates.



Terminal

```
$ tree test-chart/
test-chart/
├── charts
├── Chart.yaml
├── LICENSE
├── README.md
├── requirements.yaml
└── templates
    ├── deployment.yaml
    ├── _helpers.tpl
    ├── hpa.yaml
    ├── ingress.yaml
    ├── NOTES.txt
    ├── serviceaccount.yaml
    ├── service.yaml
    └── tests
        └── test-connection.yaml
└── values.schema.json
└── values.yaml
```

Helm Charts

Umbrella Charts

- Umbrella charts are the charts with nested stack of dependencies
- Dependencies are to be mentioned in `requirements.yaml` file at the same level as the `values.yaml` file
- Dependencies can be from a public repository(`helm repo add`) or from a local chart from a file path
- From Helm3 dependencies are moved to `Chart.yaml` file, thus eliminating the need for `requirements.yaml` file

requirements.yaml

```
dependencies:  
  - name: jenkins  
    repository: https://kubernetes-charts.storage.googleapis.com/  
    version: 1.3.6  
  - name: my-local-chart  
    repository: file:///path/to/charts/  
    version: 1.0
```

requirements.yaml

To download all the dependencies into charts directory, run `helm dependency update`

Helm Charts

Chart.yaml

- chart.yaml contains the information of name of chart, version of chart, version of application, dependencies, sources, maintainers, icon etc.,
- **version** indicates chart version and **appVersion** indicates actual application version
- The **apiVersion** field should be v2 for Helm 3 and v1 for Helm 2. However, v1 charts are still installable by Helm 3

```
apiVersion: v1/v2
appVersion: "1.0.0"
description: A Helm chart for demo application
name: demo-application
type: application
version: 0.1.0
sources:
  - https://github.com/user/demo-application-nodejs
maintainers:
  - name: myaccount
    email: myaccount@mycompany.com
icon: https://github.com/blob/master/images/logo.jpg
dependencies:
  - name: mongodb
    version: 7.8.1
    repository: https://kubernetes-charts.storage.googleapis.com
```

Chart.yaml



Refer '**Tags and Condition fields in dependencies**' at <https://helm.sh/docs/topics/charts/> to see how to manage multiple dependencies.

<https://semver.org/spec/v2.0.0.html>

Helm Charts

values.yaml

- `values.yaml` holds all the values to be injected into the templates
- Templates are rendered into kubernetes manifest files using values from `values.yaml` file

```
# Default values for chart.  
# This is a YAML-formatted file.  
# Declare variables to be passed into your templates.  
  
replicaCount: 1  
  
image:  
  repository: nginx  
  pullPolicy: IfNotPresent  
  # Overrides the image tag whose default is the chart appVersion.  
  tag: ""  
  
imagePullSecrets: []  
nameOverride: ""  
fullnameOverride: ""  
  
serviceAccount:  
  # Specifies whether a service account should be created  
  create: true  
  # Annotations to add to the service account  
  annotations: {}  
  # The name of the service account to use.  
  # If not set and create is true, a name is generated using the  
  # fullname template  
  name: ""
```

values.yaml

```
service:  
  type: ClusterIP  
  port: 80  
  
ingress:  
  enabled: false  
hosts:  
  - host: chart-example.local  
    paths: []  
  tls: []  
  # - secretName: chart-example-tls  
  
resources: {}  
# limits:  
#   cpu: 100m  
#   memory: 128Mi  
# requests:  
#   cpu: 100m  
#   memory: 128Mi  
  
autoscaling:  
  enabled: false  
  minReplicas: 1  
  maxReplicas: 100  
  targetCPUUtilizationPercentage: 80  
  # targetMemoryUtilizationPercentage: 80
```

values.yaml

Helm Charts

Pull a chart without installing
`helm pull bitnami/mysql`

```
root@k8s-master:/home/osboxes# helm pull bitnami/mysql
root@k8s-master:/home/osboxes# ls -l mysql-8.2.5.tgz
-rw-r--r-- 1 root root 36831 Jan 18 09:52 mysql-8.2.5.tgz
root@k8s-master:/home/osboxes#
```

`tar -xzf mysql-8.2.5.tgz`

```
root@k8s-master:/home/osboxes# tar -xzf mysql-8.2.5.tgz
root@k8s-master:/home/osboxes# ls -l mysql/
total 136
-rw-r--r-- 1 root root 219 Jan 15 10:08 Chart.lock
drwxr-xr-x 3 root root 4096 Jan 18 09:56 charts
-rw-r--r-- 1 root root 624 Jan 15 10:08 Chart.yaml
drwxr-xr-x 2 root root 4096 Jan 18 09:56 ci
drwxr-xr-x 3 root root 4096 Jan 18 09:56 files
-rw-r--r-- 1 root root 54456 Jan 15 10:08 README.md
drwxr-xr-x 4 root root 4096 Jan 18 09:56 templates
-rw-r--r-- 1 root root 26440 Jan 15 10:08 values-production.yaml
-rw-r--r-- 1 root root 26416 Jan 15 10:08 values.yaml
root@k8s-master:/home/osboxes#
```

 Helm Repo



Local FS

Helm Charts

Start-up order of deployment files

- Helm collects all of the resources in a given Chart and its dependencies, groups them by resource type, and then installs them in the order (shown to the right)
- So secrets/volumes/ConfigMaps are always created prior to the pods mounting them
- While uninstalling the resources, Helm follows the reverse order



- i**
- You can use `initContainers` to add waits to check the dependent microservices are functional before starting other services. Example: MQTT service is up before the dependent java service starts.
 - You can also use chart hooks to execute a Job before/after installing a new chart
https://helm.sh/docs/topics/charts_hooks/

Namespace
ResourceQuota
LimitRange
PodSecurityPolicy
Secret
ConfigMap
StorageClass
PersistentVolume
PersistentVolumeClaim
ServiceAccount
CustomResourceDefinition
ClusterRole
ClusterRoleBinding
Role
RoleBinding
Service
DaemonSet
Pod
ReplicationController
ReplicaSet
Deployment
StatefulSet
Job
CronJob
Ingress
APIService

Helm Charts

Best practices when creating Helm Charts

- Chart name should be lower case letters and numbers. The words can be separated with dashes (-)
- When the directory contains a chart, the directory name must be the same as the chart name.
- When versioning is applied, the version numbers follow the SemVer format.
- YAML files should be indented using two spaces
- The name of a variable should begin with a lowercase letter, and words should be separated using camel case
- You should quote all strings for type conversion. To ensure that integers and floats used as image tags in the `values.yaml` are evaluated as a string, the `quote` function can be used in the file consuming the value to wrap the value in double quotes.

Ex: `{{ .Values.my.value | quote }}`

- Template files should have the extension `.yaml` if they produce YAML output. The extension `.tpl` can be used for template files that produce no formatted content.
- Template file names should use dash notation, not camel case
- For a chart, you should use version ranges instead of referring to an exact version:

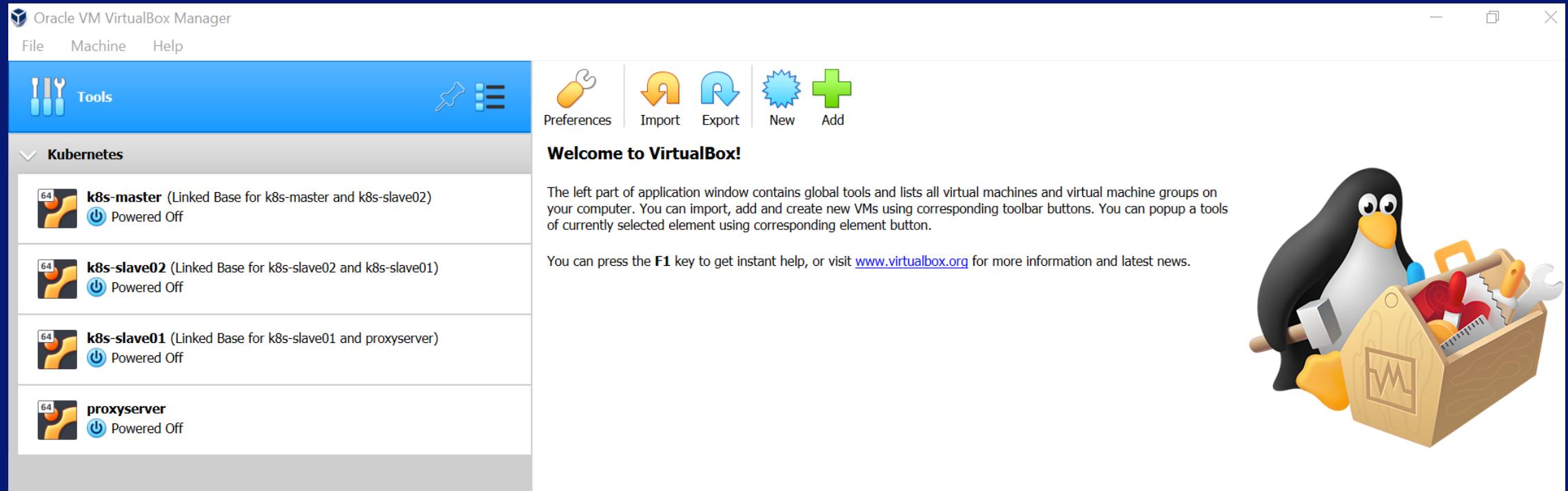
Ex: Version: `~ 1.2.3`

Cluster Setup



Cluster Requirements

- 3 Kubernetes nodes: 1 Master and 2 Slaves. Refer pre-requisites section for setup.
- 1 VM to act as Reverse Proxy (Needed for on-prem. For cloud instances, a service type of LoadBalancer for Ingress controller should do)



This tutorial assumes you are running all the helm charts on on-prem Kubernetes cluster setup using any of the tools like Kubeadm. For running on cloud managed clusters like EKS/AKS/GKE, service types are to be changed accordingly.

Helm

Check the cluster once it is setup...

`kubectl version --short`

```
root@k8s-master:/home/osboxes/Monitoring_tools# kubectl version --short
Client Version: v1.18.3
Server Version: v1.18.3
root@k8s-master:/home/osboxes/Monitoring_tools#
```

`kubectl get nodes -o wide`

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	CONTAINER-RUNTIME
node/k8s-master	Ready	master	52d	v1.18.3	192.168.0.102	<none>	Ubuntu 18.04.3 LTS	5.0.0-23-generic	docker://19.3.11
node/k8s-slave01	Ready	<none>	52d	v1.18.3	192.168.0.103	<none>	Ubuntu 18.04.3 LTS	5.0.0-23-generic	docker://19.3.10
node/k8s-slave02	Ready	<none>	52d	v1.18.3	192.168.0.113	<none>	Ubuntu 18.04.3 LTS	5.0.0-23-generic	docker://19.3.10

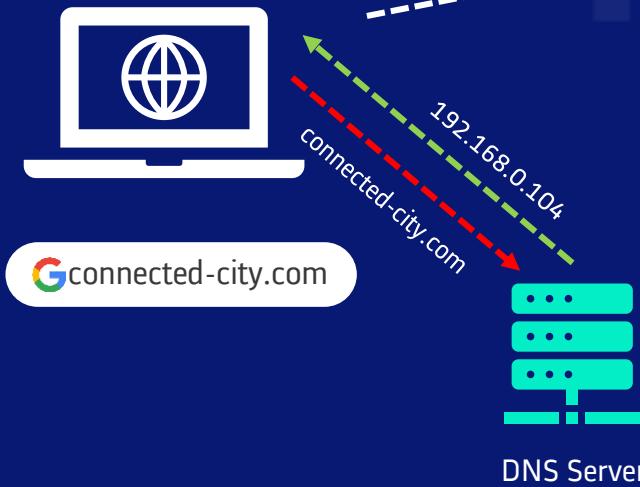
 We need the IPs of all the nodes in the cluster to configure HA Proxy VM later

`kubectl cluster-info`

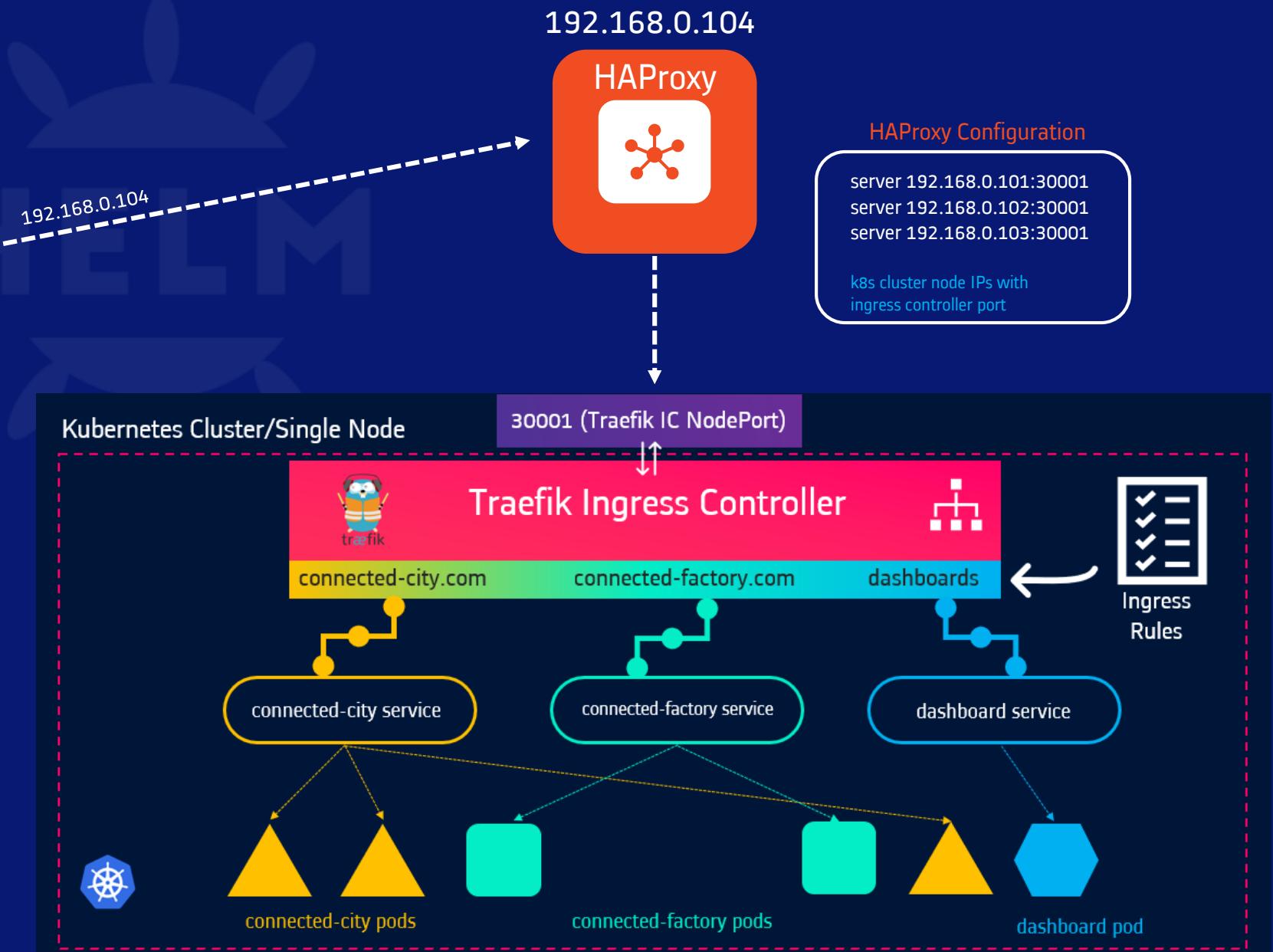
```
root@k8s-master:/home/osboxes# kubectl cluster-info
Kubernetes master is running at https://192.168.50.2:6443
KubeDNS is running at https://192.168.50.2:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
Metrics-server is running at https://192.168.50.2:6443/api/v1/namespaces/kube-system/services/https:metrics-server:/proxy
```

Helm

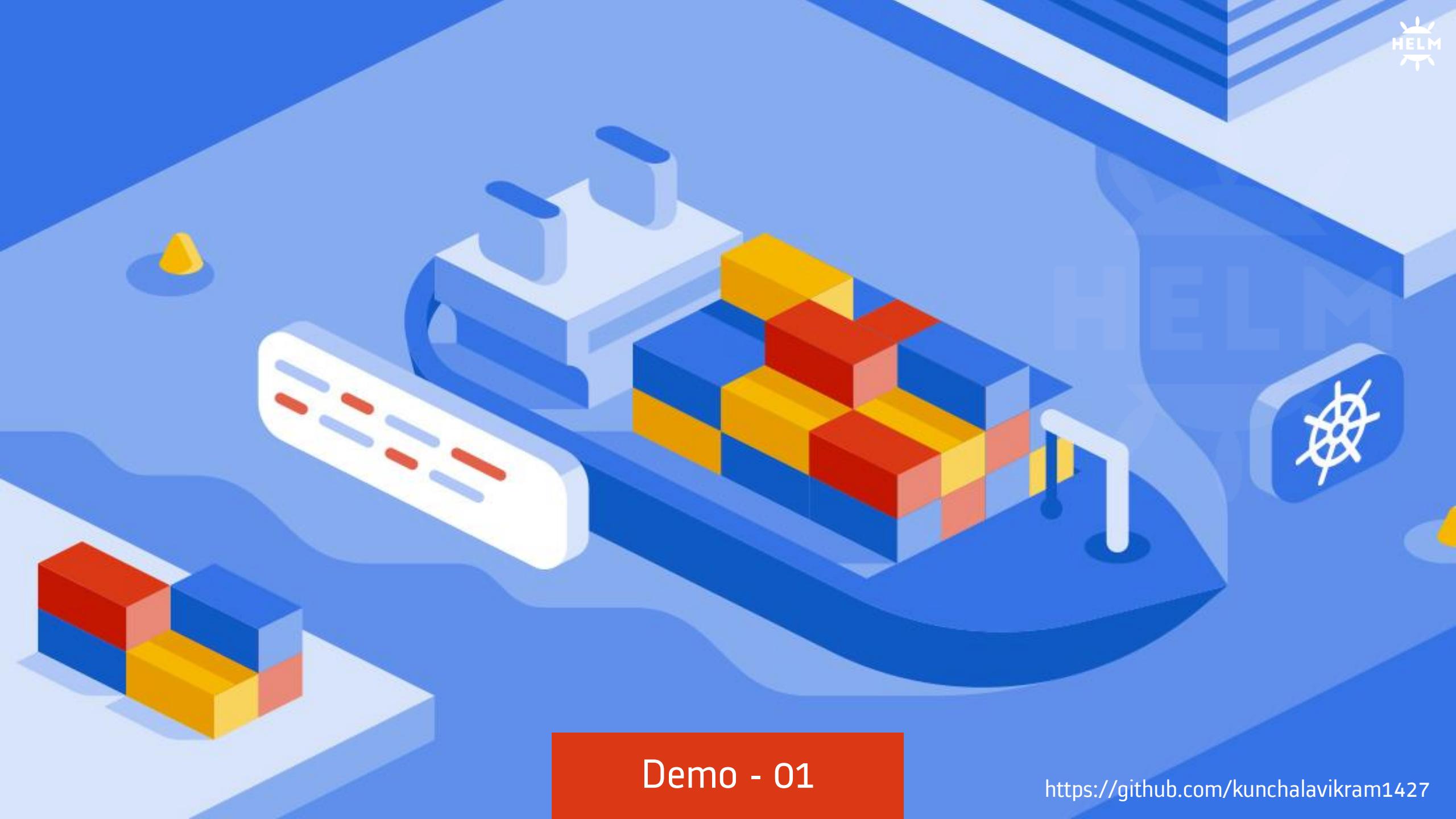
Demo Architecture



- 3-node cluster + 1VM for Reverse Proxy
- 2 front-end applications
- K8s dashboard
- Traefik Ingress Controller
- Helm charts and templates
- Ingress URL based routing



Refer [https://github.com/kunchalavikram1427/Kubernetes_End_to_End_Demo.pdf](https://github.com/kunchalavikram1427/Kubernetes_public/blob/master/end_to_end_demo/Kubernetes_End_to_End_Demo.pdf) for setup instructions



Demo - 01

<https://github.com/kunchalavikram1427>

Helm

Demo – 01: Installing applications using Kubectl

We will install all the applications manually using kubectl command line utility from the master node

```
kubectl create -f kubectl-app-deployment/
```

```
root@k8s-master:/home/osboxes/end_to_end_demo_helm# kubectl create -f kubectl-app-deployment/
deployment.apps/connectedcity-deployment created
service/connectedcity-service created
deployment.apps/connectedfactory-deployment created
service/connectedfactory-service created
ingress.networking.k8s.io/application-ingress-rules created
ingress.networking.k8s.io/kubernetes-web-ui created
deployment.apps/k8dash created
service/k8dash created
serviceaccount/k8dash-sa created
clusterrolebinding.rbac.authorization.k8s.io/k8dash-sa created
root@k8s-master:/home/osboxes/end_to_end_demo_helm# █
```

Helm

Demo – 01: Installing applications using Kubectl

Check all installed resources like pods, replicaset, deployments, services and ingress rules

```
kubectl get pods,svc,deployments,ingress
```

```
kubectl get all -n kube-system | grep k8dash - Dashboard is installed in kube-system namespace
```

```
root@k8s-master:/home/osboxes/end_to_end_demo_helm# kubectl create -f kubectl-app-deployment/deployment.apps/connectedcity-deployment created
service/connectedcity-service created
deployment.apps/connectedfactory-deployment created
service/connectedfactory-service created
ingress.networking.k8s.io/application-ingress-rules created
ingress.networking.k8s.io/kubernetes-web-ui created
deployment.apps/k8dash created
service/k8dash created
serviceaccount/k8dash-sa created
clusterrolebinding.rbac.authorization.k8s.io/k8dash-sa created
root@k8s-master:/home/osboxes/end_to_end_demo_helm#
```

```
root@k8s-master:/home/osboxes/end_to_end_demo_helm# kubectl get all -n kube-system | grep k8dash
pod/k8dash-84978cd779-j9xd6          1/1     Running   0      5m42s
service/k8dash             ClusterIP  10.103.69.97  <none>    80/TCP
deployment.apps/k8dash           1/1     1        1      5m42s
replicaset.apps/k8dash-84978cd779    1        1        1      1      5m42s
root@k8s-master:/home/osboxes/end_to_end_demo_helm#
```

 Traefik ingress controller is already installed in the cluster. If you are using minikube enable the ingress add on by running
minikube addons enable ingress

<https://github.com/kunchalavikram1427>

Helm

Demo – 01: Installing applications using Kubectl

Check all endpoints & Ingress

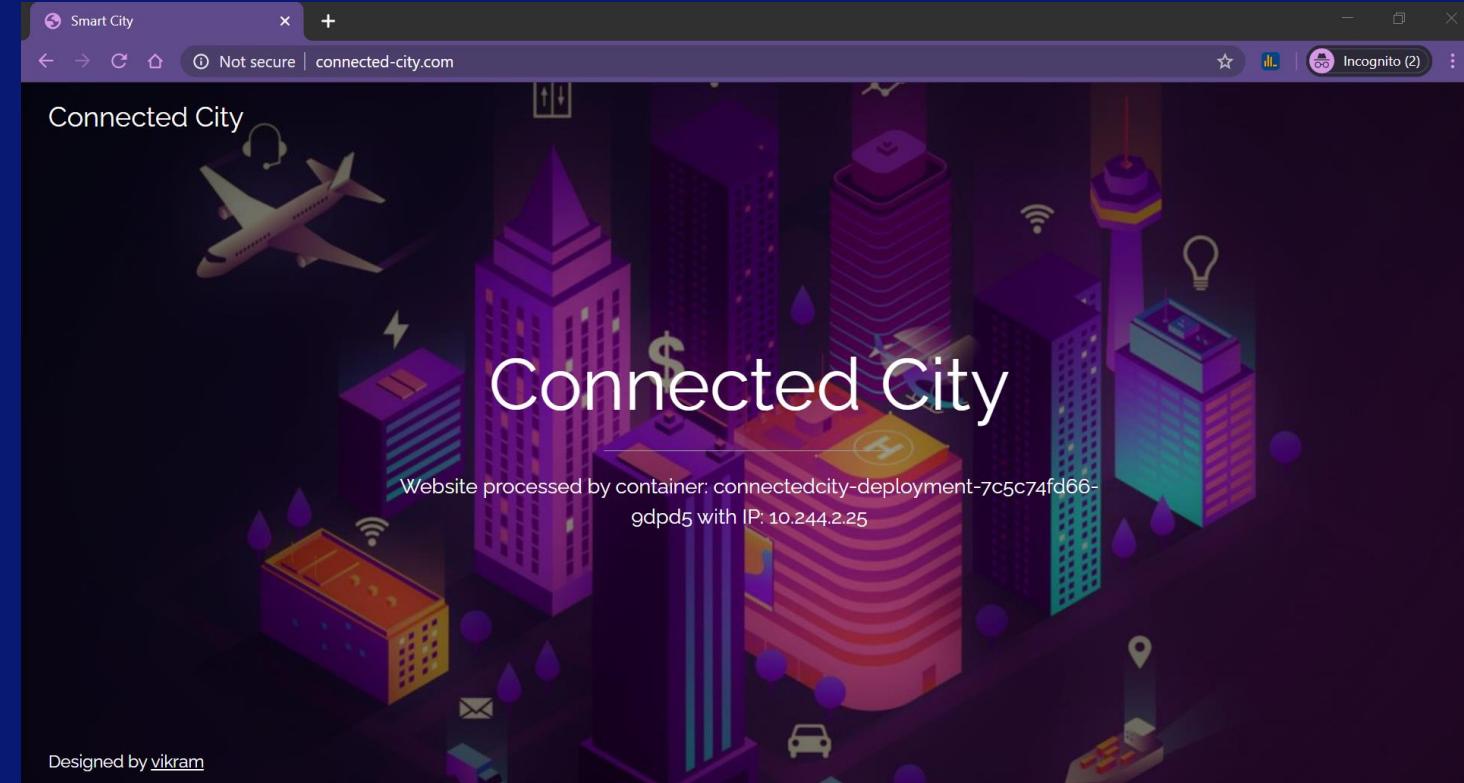
```
kubectl get ep -A
```

```
root@k8s-master:/home/osboxes/end_to_end_demo_helm# kubectl get ep -A
NAMESPACE      NAME           ENDPOINTS
default        connectedcity-service   10.244.0.177:5000,10.244.1.91:5000,10.244.2.85:5000
default        connectedfactory-service 10.244.0.176:5000,10.244.1.92:5000,10.244.2.86:5000
default        kubernetes       192.168.50.2:6443
kube-system    k8dash          10.244.1.93:4654
kube-system    kube-controller-manager <none>
kube-system    kube-dns         10.244.0.170:53,10.244.0.171:53,10.244.0.170:9153 + 3 more...
kube-system    kube-scheduler    <none>
kube-system    metrics-server    192.168.0.105:4443
kube-system    traefik-ingress-service 10.244.2.81:8080,10.244.2.81:80
kube-system    traefik-web-ui     10.244.2.81:8080
root@k8s-master:/home/osboxes/end_to_end_demo_helm#
```

```
kubectl get ing -A
```

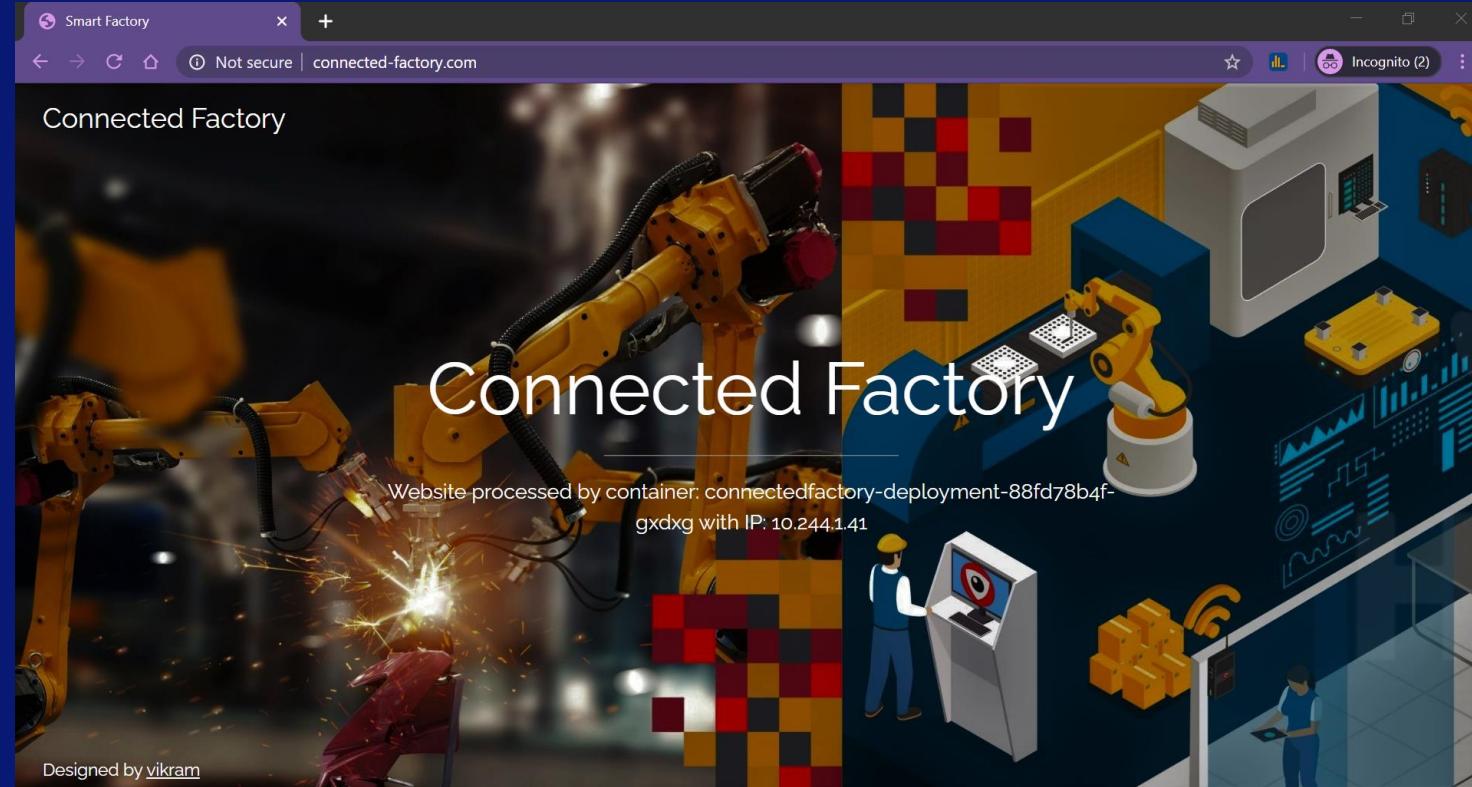
```
root@k8s-master:/home/osboxes/end_to_end_demo_helm# kubectl get ing -A
NAMESPACE      NAME           CLASS      HOSTS                                     ADDRESS   PORTS   AGE
default        application-ingress-rules <none>    connected-city.com,connected-factory.com   80        54m
kube-system    kubernetes-web-ui        <none>    k8s-dashboard.com                         80        54m
root@k8s-master:/home/osboxes/end_to_end_demo_helm#
```

Demo – 01: Installing applications using Kubectl Accessing Dashboards



Since there are 3 instances of frontend applications running, the service load balances the requests.
You can view this effect by refreshing the page and see the actual content changing like POD serving the request and its IP.

Demo – 01: Installing applications using Kubectl Accessing Dashboards



Since there are 3 instances of frontend applications running, the service load balances the requests.
You can view this effect by refreshing the page and see the actual content changing like POD serving the request and its IP.

Helm

Demo – 01: Installing applications using Kubectl

- For accessing k8s dashboard, we need a token that is generated during the deployment

```
kubectl get secret
```

```
kubectl describe secret <k8s-dash-secret-name>
```

```
root@k8s-master:/home/osboxes/end_to_end_demo_helm# kubectl get secret
NAME          TYPE      DATA  AGE
default-token-l6wnb  kubernetes.io/service-account-token  3    205d
k8dash-sa-token-jk42m  kubernetes.io/service-account-token  3    67m
root@k8s-master:/home/osboxes/end_to_end_demo_helm# kubectl describe secret k8dash-sa-token-jk42m
Name:         k8dash-sa-token-jk42m
Namespace:   default
Labels:      <none>
Annotations: kubernetes.io/service-account.name: k8dash-sa
              kubernetes.io/service-account.uid: cc870392-0c1e-4267-8746-db01a26c934d
Type:        kubernetes.io/service-account-token

Data
token:      eyJhbGciOiJSUzI1NiIsImtpZCI6Ij d0MF MwcEdFMj JqbHM5VdLCY2VQanZNNFlWT3VjM1VMVVZUQW9EUVFTVXMifQ .eyJpc3MiOiJrdWJlc m5ldGVzL3Nlc nZpY2VhY2NvdW50Iiwi a3ViZXJuZXRlc y5p
by9zZXJ2aWNl YW Njb3VudC9uYw1lc3BhY2Ui o ijkZwZhdWx0Iiwi a3ViZXJuZXRlc y5pb y9zZXJ2aW NL YW Njb3VudC9z ZW Ny ZXQubmFtZSI6Ims4ZGFzaC1zYS10b2tlbi1qazQybSIsImt1YmVyb mV0ZXMuaW8vc2Vydml
jZWfjY291bnQvc2VydmljZS1hY2NvdW50Lm5hbWUi o ijkRhc2gtc2EiLCJrdWJlc m5ldGVzLm l vL3Nlc nZpY2VhY2NvdW50L3Nlc nZpY2UtYW Njb3VudC51aWQo i ijkYzg3MDM5Mi0wYzFLLTQyNjct0Dc0Ni1kYjAxYT
I2YzkzNGQiLCJzdWIoiJzeXN0ZW06c2VydmljZWfjY291bnQ6ZGVmYXVsdDpr0GRhc2gtc2EifQ .ygedpZL80Qu2MDK1CwDFVsToXcg7VHJnpgbhE33EIvzlqd0fQb1hSyprfsI5oZc0SznGd8d5sZT0r1mXjnsIfW1j5W
HIifqI6xFcXtmemuNMQbuTRdZ1cZcZb64Ld0T687HPcR6SIZ8sc4MfPDRdX1Tbpdtm lJR sSg-ni7vZRBKxMIC T6p4CA0j6k _TF9hYMJ0qUl1VnatPj13K3WEgFhirOr9kaRVK5jBf1FsJuAdeAi hTtPM0rckq0KChXYxUcg
Mq6BLZQnuSr p10_dM2ccEomdp40sEIfzzKODft0_cMsOF-HfRtneYuasgF-UE5YfhEMNnrIXQzMwQp7CB1jmA
ca.crt: 1023 bytes
namespace: 7 bytes
root@k8s-master:/home/osboxes/end_to_end_demo_helm#
```

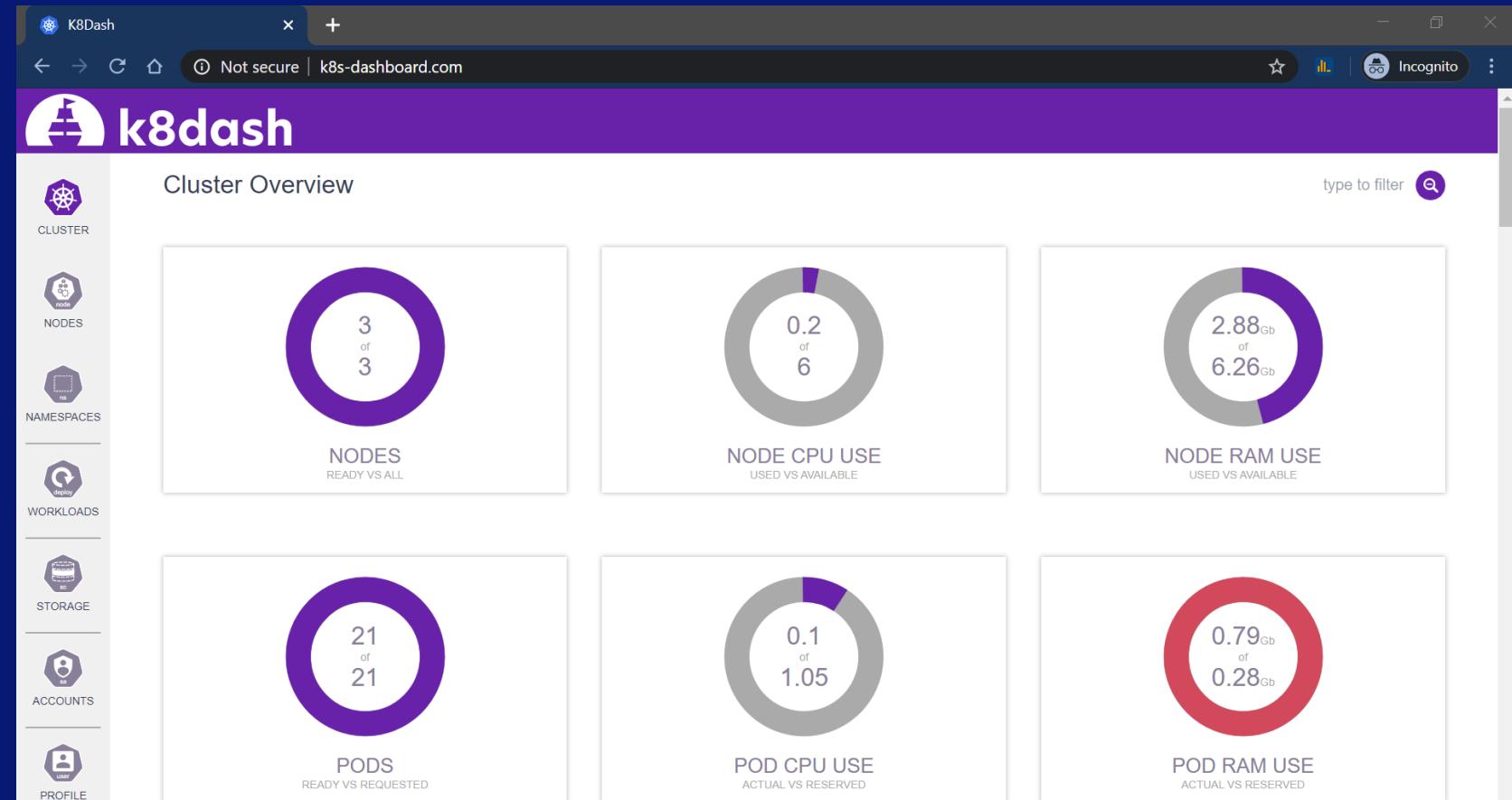


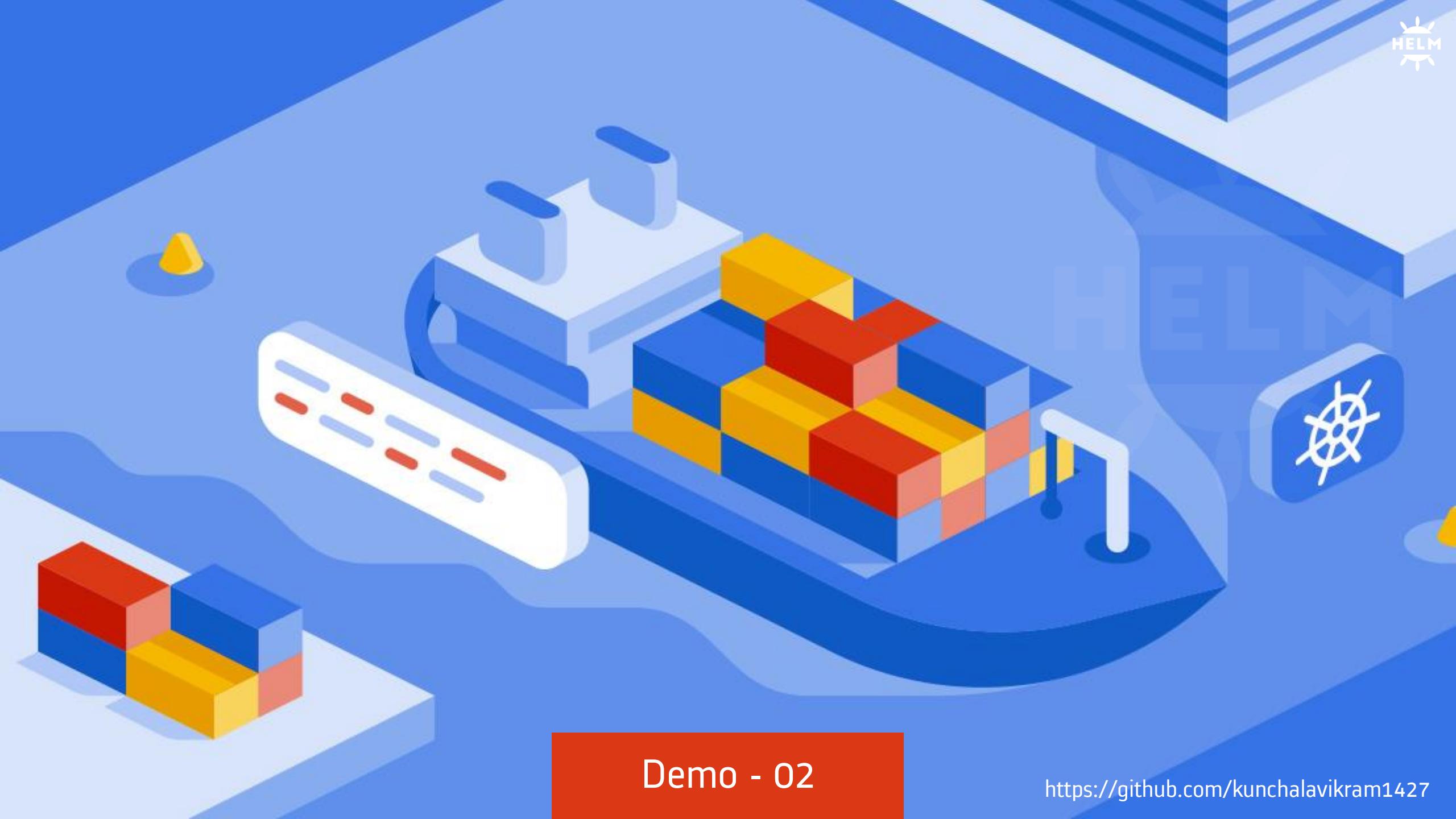
Copy the token to use it at the login page

<https://github.com/kunchalavikram1427>

Helm

Demo – 01: Installing applications using Kubectl Accessing Dashboards





Demo - 02

<https://github.com/kunchalavikram1427>

Helm Charts

Demo – 02: Installing basic chart

Consider below helm charts

Terminal

```
$ tree helm-chart-01-simple
helm-chart-01-simple
└── Chart.yaml
    └── templates
        ├── applications-deployments.yml
        ├── applications-ingress-rules.yml
        ├── applications-services.yml
        ├── k8s-dashboard-deployment.yml
        ├── k8s-dashboard-ingress-rules.yml
        └── k8s-dashboard-service.yml
```

 Both chart are same except for applications shown as sub charts in umbrella chart. In this case, each application will have its own Chart.yaml, values.yaml and can be installed individually. In a 3-tier architecture, we have frontend, backend and database charts under charts directory.

Terminal

```
$ tree helm-chart-02-umbrella
helm-chart-02-umbrella
└── charts
    ├── connectedcity-app
    │   ├── Chart.yaml
    │   └── templates
    │       ├── deployment.yml
    │       ├── ingress.yml
    │       └── service.yml
    ├── connectedfactory-app
    │   ├── Chart.yaml
    │   └── templates
    │       ├── deployment.yml
    │       ├── ingress.yml
    │       └── service.yml
    └── kubernetes-dashboard
        ├── Chart.yaml
        └── templates
            ├── clusterrolebinding.yml
            ├── deployment.yml
            ├── ingress.yml
            ├── serviceaccount.yml
            └── service.yml
```

Umbrella charts

Helm Charts

Install the chart

```
helm install <release-name> <chart-name>
```

```
helm install demo-release helm-chart-01-simple (or)  
helm install demo-release helm-chart-02-umbrella
```

```
root@k8s-master:/home/osboxes/end_to_end_demo_helm# helm install demo-release helm-chart-01-simple  
NAME: demo-release  
LAST DEPLOYED: Sun Jan  3 12:01:20 2021  
NAMESPACE: default  
STATUS: deployed  
REVISION: 1  
TEST SUITE: None  
root@k8s-master:/home/osboxes/end_to_end_demo_helm#
```

Get status of release

```
helm status <release-name>
```

```
helm status demo-release
```

```
root@k8s-master:/home/osboxes/end_to_end_demo_helm# helm status demo-release  
NAME: demo-release  
LAST DEPLOYED: Sun Jan  3 12:01:20 2021  
NAMESPACE: default  
STATUS: deployed  
REVISION: 1  
TEST SUITE: None  
root@k8s-master:/home/osboxes/end_to_end_demo_helm#
```

Helm Charts

Checking the deployed resources

`kubectl get all`

```
root@k8s-master:/home/osboxes/end_to_end_demo_helm# kubectl get all
NAME                                         READY   STATUS    RESTARTS   AGE
pod/connectedcity-deployment-7c5c74fd66-dpdqf   1/1     Running   0          14s
pod/connectedcity-deployment-7c5c74fd66-f25sr   1/1     Running   0          14s
pod/connectedcity-deployment-7c5c74fd66-zdcm2   1/1     Running   0          14s
pod/connectedfactory-deployment-88fd78b4f-bbn9s  1/1     Running   0          14s
pod/connectedfactory-deployment-88fd78b4f-jd5b6  1/1     Running   0          14s
pod/connectedfactory-deployment-88fd78b4f-rchnr  1/1     Running   0          14s

NAME           TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
service/connectedcity-service   ClusterIP   10.110.34.172  <none>        80/TCP      15s
service/connectedfactory-service ClusterIP   10.110.24.165  <none>        80/TCP      14s
service/kubernetes            ClusterIP   10.96.0.1       <none>        443/TCP    3d23h

NAME           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/connectedcity-deployment  3/3     3           3           14s
deployment.apps/connectedfactory-deployment 3/3     3           3           14s

NAME           DESIRED   CURRENT   READY   AGE
replicaset.apps/connectedcity-deployment-7c5c74fd66  3         3         3         14s
replicaset.apps/connectedfactory-deployment-88fd78b4f  3         3         3         14s
root@k8s-master:/home/osboxes/end_to_end_demo_helm#
```

Helm Charts

Upgrade the chart

Update the manifest file with any changes like image tags, replicas and upgrade the chart/release

```
helm upgrade <release-name> <chart>
```

```
helm upgrade demo-release helm-chart-01-simple (or)
```

```
helm upgrade demo-release helm-chart-02-umbrella
```

```
root@k8s-master:/home/osboxes/end_to_end_demo_helm# helm upgrade demo-release helm-chart-01-simple
Release "demo-release" has been upgraded. Happy Helming!
NAME: demo-release
LAST DEPLOYED: Sun Jan  3 12:08:42 2021
NAMESPACE: default
STATUS: deployed
REVISION: 2 ←
TEST SUITE: None
root@k8s-master:/home/osboxes/end_to_end_demo_helm#
```

 Note the REVISION number from the output

Helm Charts

Check revision history

```
helm history <release-name>
```

```
helm history demo-release
```

REVISION	UPDATED	STATUS	CHART	APP VERSION	DESCRIPTION
1	Sun Jan 3 12:01:20 2021	superseded	demo-chart-01-1.0.0	1.0	Install complete
2	Sun Jan 3 12:08:42 2021	deployed	demo-chart-01-1.0.0	1.0	Upgrade complete

Rollback to previous release

```
helm rollback <release-name> <revision-number>
```

```
helm rollback demo-release 1
```

REVISION	UPDATED	STATUS	CHART	APP VERSION	DESCRIPTION
1	Sun Jan 3 12:01:20 2021	superseded	demo-chart-01-1.0.0	1.0	Install complete
2	Sun Jan 3 12:08:42 2021	superseded	demo-chart-01-1.0.0	1.0	Upgrade complete
3	Sun Jan 3 12:12:54 2021	deployed	demo-chart-01-1.0.0	1.0	Rollback to 1

Helm Charts

Show details of a release(manifests, notes, values for a release

helm get all <release-name>

helm get all demo-release

helm get manifest demo-release

 Since we don't have any `values.yaml` file in the chart, the output will be shown as `null/none`

```
root@k8s-master:/home/osboxes/end_to_end_demo_helm# helm get all demo-release
NAME: demo-release
LAST DEPLOYED: Sun Jan  3 12:12:54 2021
NAMESPACE: default
STATUS: deployed
REVISION: 3
TEST SUITE: None
USER-SUPPLIED VALUES:
null

COMPUTED VALUES:
{}

HOOKS:
MANIFEST:
---
# Source: demo-chart-01/templates/k8s-dashboard-deployment.yml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: k8dash-sa
---
# Source: demo-chart-01/templates/k8s-dashboard-deployment.yml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: k8dash-sa
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: k8dash-sa
  namespace: default
---
# Source: demo-chart-01/templates/applications-services.yml
apiVersion: v1
kind: Service
metadata:
  name: connectedcity-service
spec:
```

Helm Charts

List all releases

```
helm list
```

NAME	NAMESPACE	REVISION	UPDATED	STATUS	CHART	APP VERSION
demo-release	default	3	2021-01-03 12:12:54.863672293 -0500 EST	deployed	demo-chart-01-1.0.0	1.0
root@k8s-master:/home/osboxes/end_to_end_demo_helm#						

Uninstall a release

```
helm uninstall <release-name>
```

```
helm uninstall demo-release
```

```
root@k8s-master:/home/osboxes/end_to_end_demo_helm# helm uninstall demo-release
release "demo-release" uninstalled
root@k8s-master:/home/osboxes/end_to_end_demo_helm#
```

 use `helm delete <release-name>` for helm2

Helm Charts

Install the chart in different namespace

```
helm install -n <namespace> <release-name> <chart-name>
```

```
kubectl create ns dev
```

```
helm install -n dev demo-release helm-chart-02-umbrella
```

```
root@k8s-master:/home/osboxes/end_to_end_demo_helm# helm install -n dev demo-release helm-chart-02-umbrella
NAME: demo-release
LAST DEPLOYED: Sun Jan  3 12:48:50 2021
NAMESPACE: dev →
STATUS: deployed
REVISION: 1
TEST SUITE: None
root@k8s-master:/home/osboxes/end_to_end_demo_helm#
```

Uninstall a release from a namespace

```
helm uninstall -n <namespace> <release-name>
```

```
helm uninstall -n dev demo-release
```



Helm Charts

- Helm stores the release information in form of kubernetes secrets

```
kubectl get secrets | grep -i helm
```

```
root@k8s-master:/home/osboxes/end_to_end_demo_helm# kubectl get secrets | grep -i helm
sh.helm.release.v1.demo-release.v1  helm.sh/release.v1          1      13m
sh.helm.release.v1.demo-release.v2  helm.sh/release.v1          1      5m58s
sh.helm.release.v1.demo-release.v3  helm.sh/release.v1          1      107s
root@k8s-master:/home/osboxes/end_to_end_demo_helm#
```

```
kubectl describe secret sh.helm.release.v1.demo-release.v2
```

```
root@k8s-master:/home/osboxes/end_to_end_demo_helm# k describe secret sh.helm.release.v1.demo-release.v2
Name:      sh.helm.release.v1.demo-release.v2
Namespace: default
Labels:    modifiedAt=1609693975
           name=demo-release
           owner=helm
           status=superseded
           version=2
Annotations: <none>
Type:     helm.sh/release.v1

Data
=====
release: 3484 bytes
root@k8s-master:/home/osboxes/end_to_end_demo_helm#
```

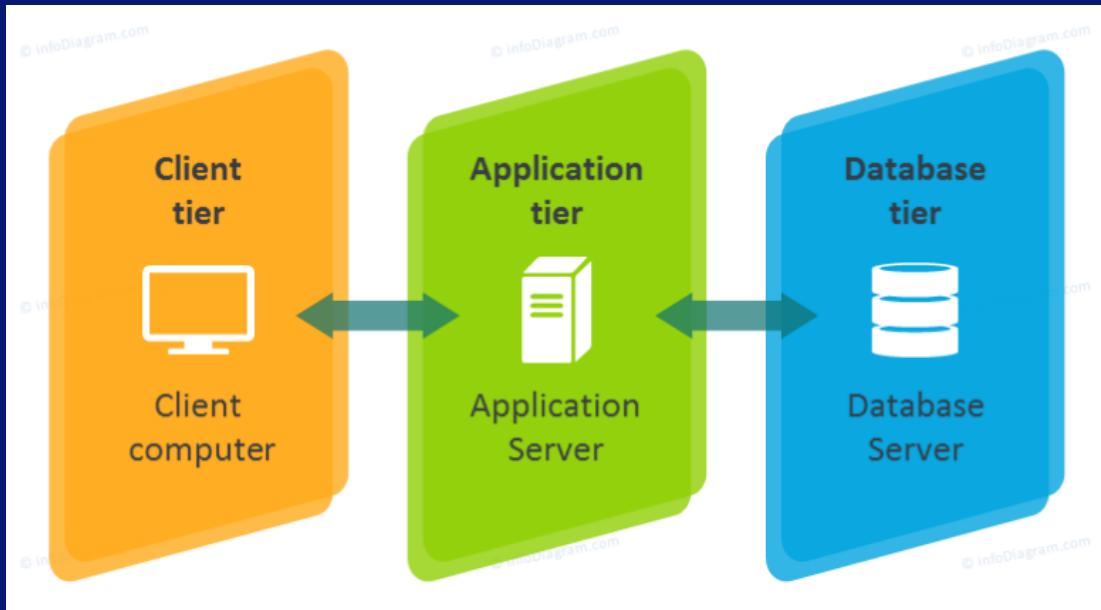
- Decode the secret

```
kubectl get secrets <secret-name> -o jsonpath="{ .data.release }" | base64 -d | base64 -d | gunzip | json_pp
```

Helm Charts

Three-Tier Architecture Charts

Three-tier architecture is a well-established software application architecture that organizes applications into three logical and physical computing tiers: the presentation tier, or user interface; the application tier, where data is processed; and the data tier, where the data associated with the application is stored and managed.



Terminal

```
$ tree web-application
```

```
web-application/
├── charts
│   ├── backend
│   │   ├── Chart.yaml
│   │   ├── templates
│   │   │   ├── backend-service.yaml
│   │   │   ├── backend.yaml
│   │   │   └── _helpers.tpl
│   │   └── ingress.yaml
│   └── values.yaml
├── database
│   ├── Chart.yaml
│   ├── templates
│   │   ├── _helpers.tpl
│   │   ├── mongodb-secret.yaml
│   │   ├── mongodb-service.yaml
│   │   └── mongodb.yaml
│   └── values.yaml
└── frontend
    ├── Chart.yaml
    ├── templates
    │   ├── frontend-configmap.yaml
    │   ├── frontend-service.yaml
    │   ├── frontend.yaml
    │   └── _helpers.tpl
    └── ingress.yaml
    └── values.yaml
    └── Chart.yaml
    └── templates
        └── ingress.yaml
    └── NOTES.txt
    └── values.yaml
```



Helm Dependencies

Helm Charts

Chart Dependencies

- In Helm, one chart may depend on any number of other charts
- These dependencies can be dynamically linked using the `dependencies` field in `Chart.yaml`
- Dependencies can be pulled from a URL or repo name if already added through `helm repo add`

```
apiVersion: v2
name: dependency-example
appVersion: "1.0"
description: A Demo Helm chart for dependencies test
version: 1.0.0
dependencies:
- name: nginx
  version: 8.3.0
  repository: https://charts.bitnami.com/bitnami
- name: wordpress
  version: 10.4.4
  repository: https://charts.bitnami.com/bitnami
```

Chart.yaml

```
apiVersion: v2
name: dependency-example
appVersion: "1.0"
description: A Demo Helm chart for dependencies test
version: 1.0.0
dependencies:
- name: nginx
  version: 8.3.0
  repository: "@bitnami"
- name: wordpress
  version: 10.4.4
  repository: "@bitnami"
```

Chart.yaml

```
root@k8s-master:/home/osboxes/helm# helm repo list
NAME          URL
stable        https://charts.helm.sh/stable
bitnami      https://charts.bitnami.com/bitnami
mirantis     https://charts.mirantis.com
root@k8s-master:/home/osboxes/helm#
```

Helm Charts

List all dependencies of a chart

helm dependency list <chart-name>

helm dependency list helm-dependency-example/

```
root@k8s-master:/home/osboxes/helm# helm dependency list helm-dependency-example/
NAME      VERSION REPOSITORY
nginx    8.3.0   https://charts.bitnami.com/bitnami
mysql    8.2.5   https://charts.bitnami.com/bitnami
```

STATUS
missing
missing

If required dependencies
are not available in the
chart.

```
root@k8s-master:/home/osboxes/helm# helm dependency list helm-dependency-example/
NAME      VERSION REPOSITORY
nginx    8.3.0   https://charts.bitnami.com/bitnami
mysql    8.2.5   https://charts.bitnami.com/bitnami
```

STATUS
ok
ok

If required dependencies
are available in the chart.
Downloaded through helm
dependency update

Helm Charts

Pull chart dependencies

```
helm dependency update <chart-name>
```

```
helm dependency update helm-dependency-example/
```

```
root@k8s-master:/home/osboxes/helm# helm dependency update helm-dependency-example/
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "mirantis" chart repository
...Successfully got an update from the "stable" chart repository
...Successfully got an update from the "bitnami" chart repository
Update Complete. *Happy Helming!*
Saving 2 charts
Downloading nginx from repo https://charts.bitnami.com/bitnami
Downloading wordpress from repo https://charts.bitnami.com/bitnami
Deleting outdated charts
```

```
tree helm-dependency-example/
```

```
root@k8s-master:/home/osboxes/helm# tree helm-dependency-example/
helm-dependency-example/
├── Chart.lock
└── charts
    └── nginx-8.3.0.tgz
        └── wordpress-10.4.4.tgz
└── Chart.yaml

1 directory, 4 files
root@k8s-master:/home/osboxes/helm#
```

Helm Charts

Chart Dependencies: Tags and Conditions

- By default, all dependencies are installed
- Tags and conditions control loading of the dependencies and to be set in parent `values.yaml` file
- Both take Boolean values
- **Conditions always override tags**
- Multiple conditions can be separated by commas and the first condition path that exists wins, ignoring the subsequent ones
- Absence of condition has no effect - it is equivalent to evaluating to `true`
- Multiple tags can be injected as a list. If at least one tag is true, the dependency is installed (logical OR)

- In this example, subchart1 dependency will be installed as it is enabled in `values.yaml` file, even though the front-end tag is set to false (conditions override tags)
- subchart2 dependency will also be installed even though no condition is set for `subchart2.enabled` in `values.yaml` (absence of condition evaluates to true). Also notice that back-end tag is set to true

```
dependencies:
  - name: subchart1
    version: 8.3.0
    repository: https://charts.bitnami.com/bitnami
    condition: subchart1.enabled
    tags:
      - front-end
      - subchart1
  - name: subchart2
    version: 10.4.4
    repository: https://charts.bitnami.com/bitnami
    condition: subchart2.enabled
    tags:
      - back-end
      - subchart2
```

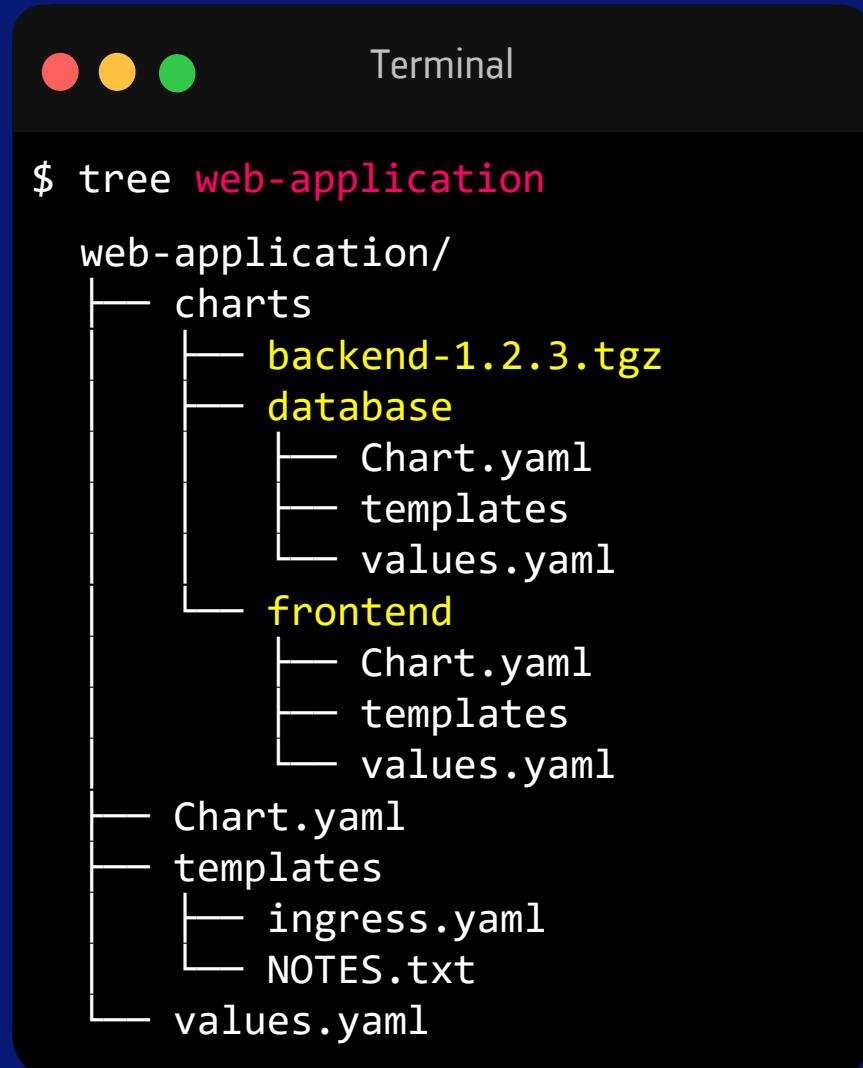
`subchart1:
 enabled: true
tags:
 front-end: false
 back-end: true`

`Chart.yaml`

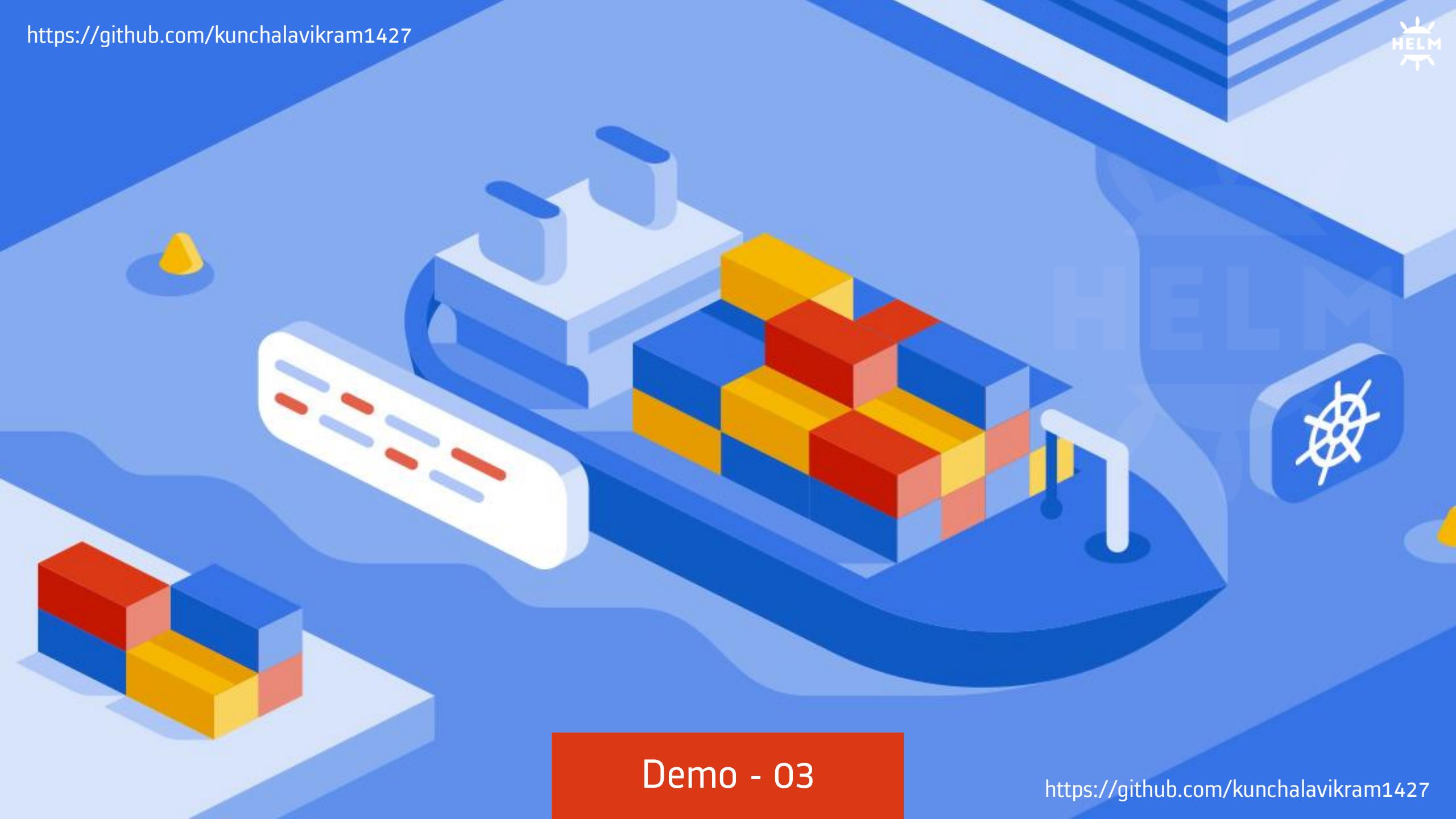
Helm Charts

Managing Dependencies manually via the charts/ directory

- If more control over dependencies is desired, these dependencies can be expressed explicitly by copying the dependency charts into the `charts/` directory
- A dependency can be either a chart archive (`backend-1.2.3.tgz`) or an unpacked chart directory
- Dependency name cannot start with `_` or `..`. Such files are ignored by the chart loader
Ex: `_helpers.tpl` and `.helmignore`



```
$ tree web-application
web-application/
├── charts
│   ├── backend-1.2.3.tgz
│   └── database
│       ├── Chart.yaml
│       ├── templates
│       └── values.yaml
└── frontend
    ├── Chart.yaml
    ├── templates
    └── values.yaml
        ├── Chart.yaml
        ├── templates
        │   └── ingress.yaml
        └── NOTES.txt
    └── values.yaml
```



Demo - 03

Helm Charts

Demo – 03: Chart Dependencies basic chart

```
apiVersion: v2
name: dependency-example
appVersion: "1.0"
description: A Demo Helm chart for dependencies test
version: 1.0.0
dependencies:
  - name: nginx
    version: 8.3.0
    repository: https://charts.bitnami.com/bitnami
    condition: nginx.enabled
    tags:
      - nginx
      - front-end
  - name: mysql
    version: 8.2.5
    repository: https://charts.bitnami.com/bitnami
    condition: mysql.enabled
    tags:
      - mysql
      - back-end
```

Chart.yaml

```
nginx:
  enabled: true
tags:
  front-end: false
  back-end: true
```

values.yaml

Helm Charts

Install the chart

```
helm install demo-rel helm-dependency-example/  
helm list
```

```
root@k8s-master:/home/osboxes/helm# helm install demo-rel helm-dependency-example/  
NAME: demo-rel  
LAST DEPLOYED: Mon Jan 18 13:11:27 2021  
NAMESPACE: default  
STATUS: deployed  
REVISION: 1  
TEST SUITE: None  
root@k8s-master:/home/osboxes/helm# helm list  
NAME          NAMESPACE   REVISION      UPDATED             STATUS        CHART  
demo-rel      default     1            2021-01-18 13:11:27.150474131 -0500 EST  deployed    dependency-example-1.0.0  
root@k8s-master:/home/osboxes/helm#
```

kubectl get pods

```
root@k8s-master:/home/osboxes/helm# kubectl get pods  
NAME           READY   STATUS    RESTARTS   AGE  
demo-rel-mysql-0   0/1     Pending   0          100s  
demo-rel-nginx-5f5bcc5b75-b7vc4  1/1     Running   0          100s  
root@k8s-master:/home/osboxes/helm#
```



Both dependencies are installed as discussed earlier

helm uninstall demo-rel

Helm Charts

Overriding the default values using --set

```
nginx:           values.yaml  
  enabled: true  
tags:  
  front-end: false  
  back-end: true
```

values.yaml doesn't have any condition to enable/disable mysql dependency.
We can override this while installing the chart

```
helm install <release-name> <chart-name> --set <name1>=<value1>,<name2>=<value2>
```

```
helm install demo-rel helm-dependency-example/ --set mysql.enabled=false
```

```
root@k8s-master:/home/osboxes/helm# helm install demo-rel helm-dependency-example/ --set mysql.enabled=false  
NAME: demo-rel  
LAST DEPLOYED: Mon Jan 18 13:19:53 2021  
NAMESPACE: default  
STATUS: deployed  
REVISION: 1  
TEST SUITE: None  
root@k8s-master:/home/osboxes/helm# kubectl get pods  
NAME                      READY   STATUS    RESTARTS   AGE  
demo-rel-nginx-5f5bcc5b75-wzttf   0/1     Running   0          6s  
root@k8s-master:/home/osboxes/helm#
```

 mysql dependency is not installed as it is disabled dynamically

<https://github.com/kunchalavikram1427>

Helm Charts

Overriding the default values using --set

```
ingress:  
  enabled: true  
image:  
  name: nginx  
  tag: 1.0  
labels:  
  - app: nginx  
  - type: application  
service:  
  type: ClusterIP  
  port: 80  
  protocol: TCP
```

values.yaml

```
--set .ingress.enabled=false  
--set .image.tag=1.1  
--set .service.type=NodePort  
--set .service.port=30001
```

```
helm install --set .Values.ingress.enabled=false, --set .Values.service.type=NodePort <chart-name>
```

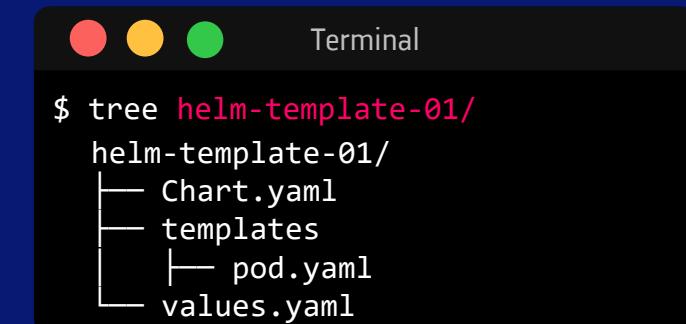
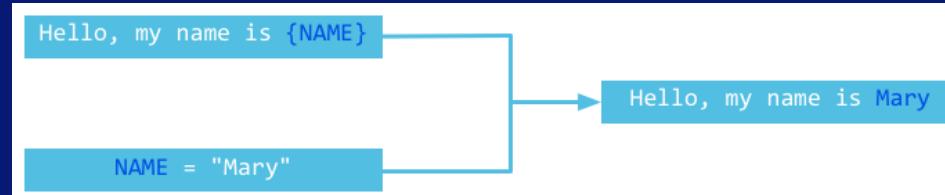
Helm Templates



Helm Templates

- Templates are Kubernetes manifest files that describe the Kubernetes resources
- Helm uses the [Go templating](#) engine by default + functions from “Sprig” lib like default, quote etc.,
- Avoids hardcoded values
- Templates are rendered by Helm template engine into Kubernetes manifest files at the client side before sending it to kubernetes API
- All template files are stored in a chart’s [templates/](#) folder. When Helm renders the charts, it will pass every file in that directory through the template engine
- The values for the templates can be supplied in one of three ways:
 1. [values.yaml](#) file inside of a chart. This file may contain default values
 2. Custom values YAML file to be provided at the command line with the [helm install -f](#) command
 3. Using [--set](#) flag. This set value will replace the value in the values.yaml file

```
$ helm install my-wordpress-prod bitnami/wordpress --version 10.1.4 -f values-production.yaml  
$ helm install --values myvalues.yaml stable/mysql  
$ helm install demo-sql stable/mysql --set db.name='test'
```



```
Terminal  
$ tree helm-template-01/  
helm-template-01/  
├── Chart.yaml  
├── templates  
│   └── pod.yaml  
└── values.yaml
```

Helm Templates

- In order to release the same application in the same cluster, all the resources should have unique name across the cluster
- It becomes tedious to manually write unique names for each resource. Helm templates solves this issues by generating unique names for each object using **templates**
- Here Release name is used to generate unique name for each resource like Pods

```
apiVersion: v1
kind: Pod
metadata:
  name: dev-wordpress-pod
  labels:
    app: wordpress
spec:
  containers:
    - name: wordpress
      image: wordpress:latest
      imagePullPolicy: IfNotPresent
```

pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: prod-wordpress-pod
  labels:
    app: wordpress
spec:
  containers:
    - name: wordpress
      image: wordpress:latest
      imagePullPolicy: IfNotPresent
```

pod.yaml

A template directive is enclosed in {{ and }} blocks.



Go template directive/tag used: {{ .Release.Name }}

We can release the same application in 2 different namespaces without any name conflicts

<https://github.com/kunchalavikram1427>

Helm Templates

Template Directives

- A template directive/tag, is enclosed in {{ and }} blocks, and it is recommended to pad the directive with space at its left and right
- The simplest directive renders a value. A value is a namespaced object, where each dot (.) separates each namespaced element
- A leading dot indicates that we start with the top-most namespace/root for the scope

```
apiVersion: v1                               pod.yaml
kind: Pod
metadata:
  name: {{ .Release.Name }}-wordpress-pod
  labels:
    app: wordpress
spec:
  containers:
  - name: wordpress
    image: wordpress:latest
    imagePullPolicy: IfNotPresent
```

Whitespace Handling

- The spaces inside a directive/tag is irrelevant
- The '{{' and '}}' directive delimiters, without any other modifications, leave the template whitespace surrounding them
- A hyphen '-' placed after the '{{' delimiter or before the '}}' delimiter instructs the rendering engine to trim/drop the whitespace, includes spaces, tabs, newline ('\n') and carriage return ('\r'), around the directive

```
spec:  
  containers:  
    - name: nginx-pod  
      image: nginx:latest  
      ports:  
        - containerPort: {{ .Values.port }}
```

pod.yaml



without -

```
spec:  
  containers:  
    - name: nginx-pod  
      image: nginx:latest  
      ports:  
        - containerPort: 80
```

pod.yaml

```
spec:  
  containers:  
    - name: nginx-pod  
      image: nginx:latest  
      ports:  
        - containerPort: {{- .Values.port }}
```

pod.yaml



with -

```
spec:  
  containers:  
    - name: nginx-pod  
      image: nginx:latest  
      ports:  
        - containerPort:80
```

pod.yaml

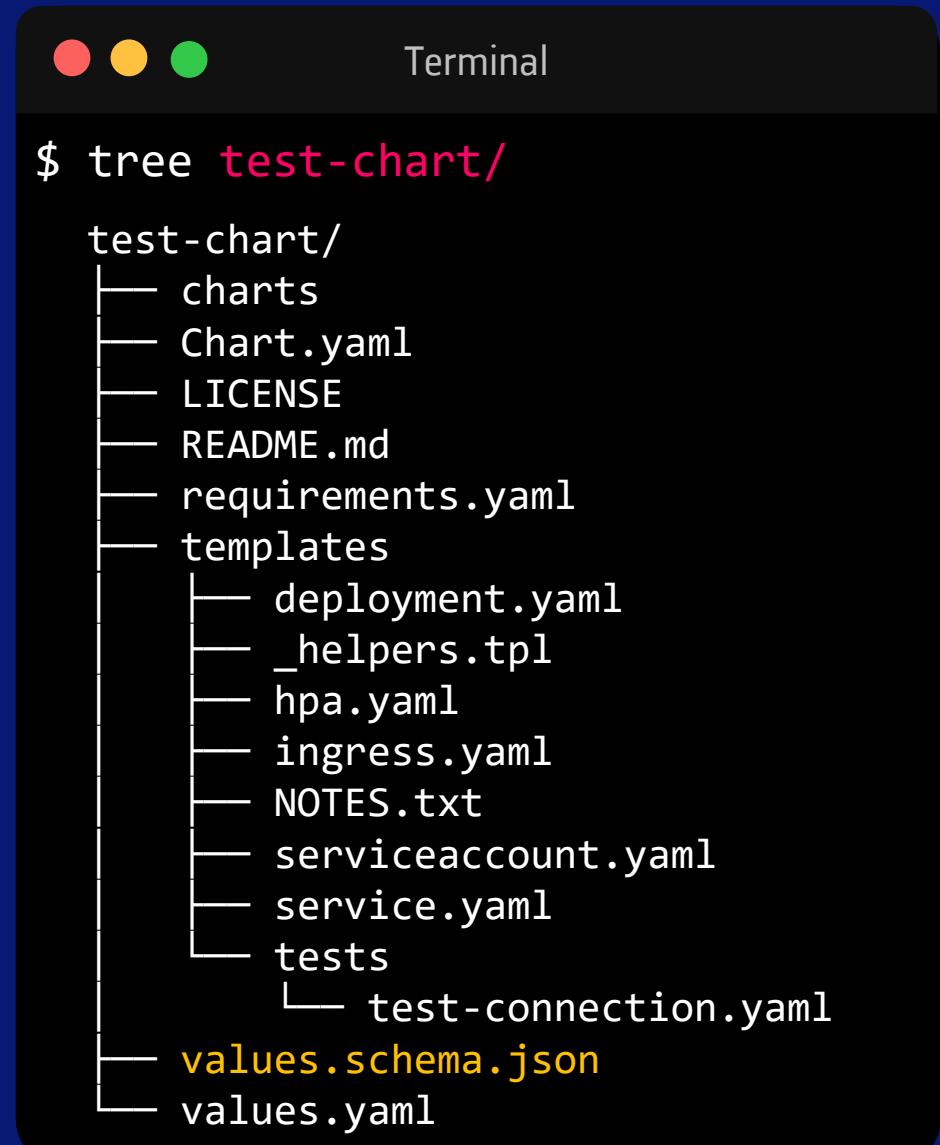
Helm Templates

JSON Schema Files

- Used to validate the values in `values.yaml` file
- Sometimes the chart developer might put restrictions on type of values to be used, mandatory values to be used etc.,
- Failing to adhere to these restrictions/rules will result in chart installation errors
 - Ex: Container Port can be made as a mandatory value
- Validation occurs every time the user runs `helm install/upgrade/template/lint`
- `values.schema.json` file to be located at the root of the chart folder

<https://json-schema.org/draft/2019-09/json-schema-validation.html>

<https://helm.sh/docs/topics/charts/#schema-files>



```
$ tree test-chart/
test-chart/
├── charts
├── Chart.yaml
├── LICENSE
├── README.md
├── requirements.yaml
└── templates
    ├── deployment.yaml
    ├── _helpers.tpl
    ├── hpa.yaml
    ├── ingress.yaml
    ├── NOTES.txt
    ├── serviceaccount.yaml
    └── service.yaml
    └── tests
        └── test-connection.yaml
└── values.schema.json
└── values.yaml
```

<https://github.com/kunchalavikram1427>

Helm Templates

<https://github.com/kunchalavikram1427>



Rendering Templates

```
image:  
  name: nginx  
  tag: latest  
  pullPolicy: IfNotPresent
```

values.yaml



```
apiVersion: v1  
kind: Pod  
metadata:  
  name: myapp-pod  
  labels:  
    app: myapp  
spec:  
  containers:  
  - name: {{ .Chart.Name }}  
    image: "{{ .Values.image.name }}:{{ .Values.image.tag }}"  
    imagePullPolicy: {{ .Values.image.pullPolicy }}  
  ports:  
  - containerPort: 80
```

templates/pod.yaml



```
helm template <chart-name>  
helm template helm-template-01/
```

```
apiVersion: v1  
kind: Pod  
metadata:  
  name: myapp-pod  
  labels:  
    app: myapp  
spec:  
  containers:  
  - name: demo-template-01  
    image: "nginx:latest"  
    imagePullPolicy: IfNotPresent  
  ports:  
  - containerPort: 80
```

pod.yaml



`helm template` renders the template by replacing values from values.yaml file without installing the chart. No k8s cluster needed for this and no release name will be generated

`helm template` cannot replace any values that depends on Release information. Ex: `{{ .Release.Name }}`

Helm Templates

<https://github.com/kunchalavikram1427>



Rendering Templates with Release Information

```
image:  
  name: nginx  
  tag: latest  
  pullPolicy: IfNotPresent
```

values.yaml

```
helm install <release-name> <chart-name> --dry-run --debug  
helm install demo-01 helm-template-01/ --dry-run --debug
```

```
apiVersion: v1  
kind: Pod  
metadata:  
  name: myapp-pod  
  labels:  
    app: myapp  
    release: {{ .Release.Name }}  
spec:  
  containers:  
  - name: {{ .Chart.Name }}  
    image: "{{ .Values.image.name }}:{{ .Values.image.tag }}"  
    imagePullPolicy: {{ .Values.image.pullPolicy }}  
  ports:  
  - containerPort: 80
```

templates/pod.yaml

```
apiVersion: v1  
kind: Pod  
metadata:  
  name: myapp-pod  
  labels:  
    app: myapp  
    release: demo-01  
spec:  
  containers:  
  - name: demo-template-01  
    image: "nginx:latest"  
    imagePullPolicy: IfNotPresent  
  ports:  
  - containerPort: 80
```

pod.yaml



`helm install --dry-run` renders the template by replacing values from values.yaml file by sending the manifests to k8s API server.
A release name will be generated in this case without actual chart installation

Helm Templates

Rendering Templates with Release Information

```
root@k8s-master:/home/osboxes/helm# helm install demo-01 helm-template-01/ --dry-run --debug
install.go:173: [debug] Original chart version: ""
install.go:190: [debug] CHART PATH: /home/osboxes/helm/helm-template-01

NAME: demo-01
LAST DEPLOYED: Tue Jan 19 07:43:33 2021
NAMESPACE: default
STATUS: pending-install
REVISION: 1
TEST SUITE: None
USER-SUPPLIED VALUES:
{}

COMPUTED VALUES:
image:
  name: nginx
  pullPolicy: IfNotPresent
  tag: latest

HOOKS:
MANIFEST:
---
# Source: demo-template-01/templates/pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    release: demo-01
spec:
  containers:
  - name: demo-template-01
    image: "nginx:latest"
    imagePullPolicy: IfNotPresent
    ports:
    - containerPort: 80
```

Helm Templates

Mandatory/Required values

values.yaml

```
image:  
  name: nginx  
  pullPolicy: IfNotPresent
```

templates/pod.yaml

```
apiVersion: v1  
kind: Pod  
metadata:  
  name: myapp-pod  
  labels:  
    app: myapp  
    release: {{ .Release.Name }}  
spec:  
  containers:  
    - name: {{ .Chart.Name }}  
      image: "{{ .Values.image.name }}":{{ required "A valid .Values.image.tag entry required!" .Values.image.tag }}"  
      imagePullPolicy: {{ .Values.image.pullPolicy }}  
      ports:  
        - containerPort: 80
```

```
helm install demo-01 helm-template-01/ --dry-run --debug
```

```
root@k8s-master:/home/osboxes/helm# helm install demo-01 helm-template-01/ --dry-run --debug  
install.go:173: [debug] Original chart version: ""  
install.go:190: [debug] CHART PATH: /home/osboxes/helm/helm-template-01  
  
Error: execution error at (demo-template-01/templates/pod.yaml:11:41): A valid .Values.image.tag entry required!  
helm.go:81: [debug] execution error at (demo-template-01/templates/pod.yaml:11:41): A valid .Values.image.tag entry required!  
root@k8s-master:/home/osboxes/helm#
```



when required values are not set in values.yaml file, Helm throws an error. Alternatively, values can be passed through command line as shown below

```
helm install demo-01 helm-template-01/ --set image.tag=latest --dry-run --debug
```

Helm Templates

Default values

```
values.yaml
```

```
image:
  name: nginx
  tag: latest
  pullPolicy: IfNotPresent
replicas: 2
```

```
templates/pod.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
  labels:
    app: web-development
spec:
  replicas: {{ default 1 .Values.replicas | int }}
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      name: myapp-pod
      labels:
        app: nginx
    spec:
      containers:
        - name: myapp-container
          image: "{{ .Values.image.name }}:{{ .Values.image.tag }}"
          imagePullPolicy: {{ .Values.image.pullPolicy }}
          ports:
            - containerPort: 80
```



Tip: variable quoting – To ensure that integers and floats used as image tags in the `values.yaml` are evaluated as a string, the `quote` function can be used in the file consuming the value to wrap the value in double quotes.
`{{ .Values.image.tag | quote }}`

For integers: `int` function can be used

```
helm install demo-01 helm-template-01/ --dry-run --debug
```

when values are not set in `values.yaml` file, default values are used. The same syntax for default values can also be written as
`{{ .Values.replicas | default 1 | int }}`

Helm Templates

Predefined values

- `Release.Name`: The name of the release (not the chart)
- `Release.Namespace`: The namespace the chart was released to
- `Release.IsUpgrade`: This is set to true if the current operation is an upgrade or rollback
- `Release.Install`: This is set to true if the current operation is an install
- `Release.Revision`: The revision number for this release. On install, this is 1, and it is incremented with each upgrade and rollback
- `Release.Service`: The service that is rendering the present template. On Helm, this is always Helm
- `Chart.Name`: The name of the chart
- `Chart.Version`: The version of chart

```
apiVersion: v1          templates/cm.yaml
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  env: {{ .Release.Namespace }}
```

Helm Templates

Template Functions

- When injecting strings into the template, we ought to quote these strings
- Instead of quoting, we can call the quote function in the template directive

```
apiVersion: v1          templates/cm.yaml
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  url: {{ quote .Values.app.url }}
  type: {{ quote .Values.app.type }}
```

```
colors:                 values.yaml
  - blue
  - red
  - green
image:
  labels:
    - app: web
    - env: prod
```

- {{ (index .Values.image.labels 0).app }}
- {{ (index .Values.image.labels 1).env }}
- {{ index .Values.colors 0 }}



info icon index function is used to access array elements

Helm Templates

Template Pipelines

- Pipelines chain together a series of template commands to express a series of transformations
- Pipelines are an efficient way of getting several things done in sequence

```
apiVersion: v1          templates/cm.yaml
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  url: {{ .Values.app.url | quote}}
  type: {{ .Values.app.type | quote}}
```

Few Pipelines

- {{ .Values.url | upper | quote }}
- {{ .Values.type | repeat 5 | quote }}
- {{ .Values.replicas | default 1 | quote }}
- {{ .Values.secret.db_password | b64enc | quote }}
- {{- default .Chart.Name .Values.nameOverride | trunc 63 | trimSuffix "-" - }}

 In this example, instead of calling quote ARGUMENT, we inverted the order. We sent the argument to the function using a pipeline (|)

Helm Templates

<https://github.com/kunchalavikram1427>



Template Variables

- Variable is a named reference to another object. It follows the form `$name`. Variables are assigned with a special assignment operator: `:=`
- Used in conjunction with `with` and `range`
- Variables are normally not global. They are scoped to the block in which they are declared. However, there is one variable that is always global: `$`. This variable will always point to the root context:
`{{ $.Chart.AppVersion }}`

templates/pod.yaml

```
{-{ $podname := "nginx" -}}
apiVersion: v1
kind: Pod
metadata:
  name: {{ $podname }}-pod
spec:
  containers:
    - name: {{ $podname }}-container
      image: nginx:latest
      ports:
        - containerPort: 80
```

templates/cm.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  {{- $relenv := .Release.Namespace -}}
  {{- with .Values.configmap -}}
  url: {{ .url | quote }}
  db_user: {{ .user | lower | quote }}
  release: {{ $relenv }}
  {{- end -}}
```

Ex: `{{ $fullName := printf "%s-%s" .Release.Name .Chart.Name | trunc 63 | trimSuffix "-" }}`

Helm Templates

Flow Control: Range function

- Range function will "range over" (iterate through) a list
- Range operator sets the scope to ".", so the simplest way to access the current iteration element is with `{{ . }}`

```
values.yaml
data:
  color: 'red'
  shape: 'triangle'
```

```
templates/dep.yaml
metadata:
{{- if .Values.data }}
annotations:
{{- range $key, $value := .Values.data }}
  {{ $key }}: {{ $value | quote -}}
{{- end }}
{{- end}}
```



Iterating over an In-Line List

```
templates/pod.yaml
env:
{{- range list "blue" "red" "green" -}}
- name: {{ . | upper | printf "COLOR_%s" -}}
  value: {{ . -}}
{{- end -}}
```

o/p

```
env:
- name: COLOR_BLUE
  value: blue
- name: COLOR_RED
  value: red
- name: COLOR_GREEN
  value: green
```

Helm Templates

<https://github.com/kunchalavikram1427>



Range function

Iterating over a List

```
colors:          values.yaml
  - blue
  - red
  - green
  - yellow
```



```
env:
{{- range .Values.colors -}}
  - name: {{ . | upper | printf "COLOR_%s" }}
    value: {{ . -}}
{{ end }}
```

templates/dep.yaml

o/p

```
env:
- name: COLOR_BLUE
  value: blue
- name: COLOR_RED
  value: red
- name: COLOR_GREEN
  value: green
- name: COLOR_YELLOW
  value: yellow
```

Iterating over a Map

```
shapes:          values.yaml
  FIRST_SHAPE: square
  SECOND_SHAPE: circle
  THIRD_SHAPE: triangle
```



```
env:
{{- range $key, $value := .Values.shapes -}}
  - name: {{ $key -}}
    value: {{ $value | upper -}}
{{ end }}
```

templates/dep.yaml

o/p

```
env:
- name: FIRST_SHAPE
  value: SQUARE
- name: SECOND_SHAPE
  value: CIRCLE
- name: THIRD_SHAPE
  value: TRIANGLE
```

Helm Templates

Range function

Iterating over a List

```
favorite:      values.yaml
  drink: coffee
  food: pizza
pizzaToppings:
  - mushrooms
  - cheese
  - peppers
  - onions
```



```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
{{- with .Values.favorite }}
  drink: {{ .drink | default "tea" | quote }}
  food: {{ .food | upper | quote }}
{{- end }}
  toppings: |-  

{{- range .Values.pizzaToppings }}
  - {{ . | title | quote }}
{{- end }}
```

templates/cm.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: edgy-dragonfly-configmap
data:
  myvalue: "Hello World"
  drink: "coffee"
  food: "PIZZA"
  toppings: |-  

    - "Mushrooms"
    - "Cheese"
    - "Peppers"
    - "Onions"
```

o/p



In the example above, we used | to indicate a multi-line string. But the content of the string will be followed with a trailing \n. If we want the YAML processor to strip off the trailing newline, we can add a - after the |

Helm Templates

if-else

- if/else can be used to create conditional blocks
- For templates, the operators (eq, ne, lt, gt, and, or, not....) are all implemented as functions

```
db_data:      values.yaml
  env: prod
  db_user: admin
  db_table: test-db
```



templates/cm.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  env: {{ .Values.db_data.env | default "dev" }}
  db_user: {{ .Values.db_data.db_user | lower }}
  {{- if eq .Values.db_data.db_table "test-db" -}}
  db: test-db
  {{- else -}}
  db: app-db
  {{- end -}}
```

o/p

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: test-configmap
data:
  env: prod
  db_user: admin
  db: test-db
```

Helm Templates

if-else

- The operators (eq, ne, lt, gt, and, or, not....) are all implemented as functions

OR

```
{{- if or .Values.myApp.config.local .Values.myApp.config.nfs }}  
...  
{- end }}
```

NOT

```
{{ if not .Values.myApp.something }}  
...  
{- end }}
```

EQUAL

```
{- if eq .Values.myApp.something .Values.myApp.somethingElse }  
...  
{- end }}
```

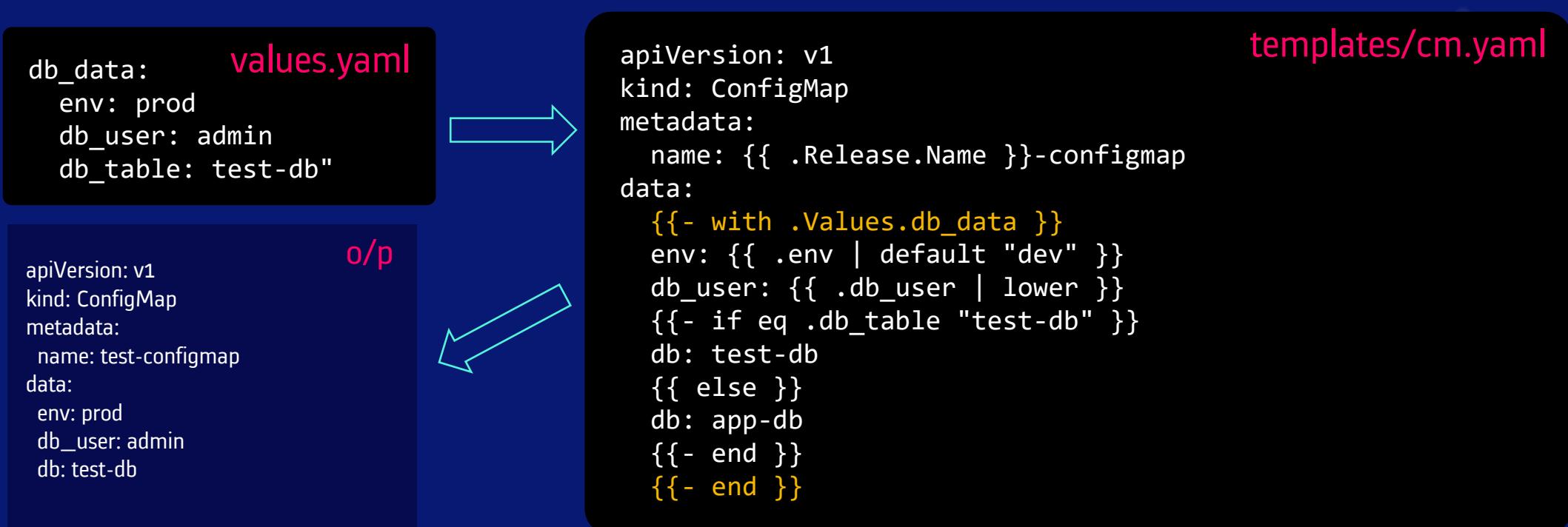
NOT EQUAL

```
{- if ne .Values.myApp.color "blue" }  
...  
{- end }}
```

Helm Templates

Modifying Scope using 'with'

- **With** controls variable scoping & current scope can be changed using it
- **.** is a reference to the current scope as seen earlier
- In the below template, scope is now at **db_data**, so **.Values.db_data.env** can be directly referenced with **.env**. This is because the **with** statement sets **.** to point to **.Values.db_data**
- The **.** is reset to its previous scope after **{ { end } }**



Helm Templates

Named Templates

- Named template/sub-template/embedded template, is a fragment of text that is declared in one file and then rendered in-line into another template, usually defined in a different file, every time it is invoked with the `template` action or the `include` function
- Sub-template is declared with the `define` action
- Sub-template has a name, declared when the sub-template is defined
- Template engine renders the sub-template declaration, when it encounters a `template` action or a `include` function that specify the sub-template name
- Sub-template names, including those declared in subcharts, are global. This means that if two sub-templates have the same name, whichever is loaded last will be the one to be used
- For this reason, is a good practice to name the sub-templates with chart-specific names, and prefix the name of the sub-template with the name of the chart
`{{ define "mychart.mysubtemplate" }}`
- Sub-templates can be declared inside other template files, in helper files, for example `_helpers.tpl`, inside the chart's templates/ directory

Helm Templates

Create a sub-template using 'define'

- **define** action is used to declare the name and the content of the sub-template
- define declaration does not produce output, until the sub-template is rendered with **template** or **include** action
- The indentation of the sub-template body is important
- The **template** action will render the body as is, honouring the existing white space and indentation. So no modifications can be done to a sub-template with **template**
- The **include** function, however, can be used to modify the sub-template like indent the body, by piping it through the **indent** function
- Both **template** and **include** are used to embed the template into manifests

```
 {{ define "MY.NAME" }}  
   # body of template here  
 {{ end }}
```

```
 {{/*  
 Generate basic labels.  
 This is a multi-line comment.  
 */}}  
 {{- define "mychart.mysubtemplate" }}  
 labels:  
   color: blue  
   date: {{ now | htmlDate }}  
 {{- end }}
```

Helm Templates

Embed a sub-template using 'template'

- The 'template' action renders the specified sub-template, with the given scope, in the enclosing template.

```
db_data:      values.yaml
  env: prod
  db_user: admin
  db_table: test-db"
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
  {{- template "mychart.labels" --}}
data:
  {{- with .Values.db_data --}}
    env: {{ .env | default "dev" }}
    db_user: {{ .db_user | lower }}
    {{- if eq .db_table "test-db" --}}
      db: test-db
    {{- else --}}
      db: app-db
    {{- end }}
  {{- end }}"
```

templates/cm.yaml

```
/*
This is a multi-line comment.
Define usage of subtemplate here.
*/
{{- define "mychart.labels" --}}
labels:
  generator: helm
  date: {{ now | htmlDate }}
{{- end }}
```

templates/_helpers.tpl

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: RELEASE-NAME-configmap
  labels:
    generator: helm
    date: 2021-01-21
  data:
    env: prod
    db_user: admin
    db: test-db
```

o/p

Helm Templates

Embed template inside manifests

```
/*  
This is a multi-line comment.  
Define usage of subtemplate here.  
*/}  
{-{ define "mychart.labels" }}  
labels:  
  generator: helm  
  date: {{ now | htmlDate }}  
{- end }  
  
apiVersion: v1  
kind: ConfigMap  
metadata:  
  name: {{ .Release.Name }}-configmap  
  {{- template "mychart.labels" }}  
data:  
  {{- with .Values.db_data }}  
    env: {{ .env | default "dev" }}  
    db_user: {{ .db_user | lower }}  
    {{- if eq .db_table "test-db" }}  
      db: test-db  
    {{ else }}  
      db: app-db  
    {{- end }}  
  {{- end }}  
{{- end }}
```

templates/cm.yaml

Helm Templates

Scope of sub-templates

- If a sub-template includes references to built-in objects(`.Chart.Name`, `.Chart.Version`), those objects need to be reachable from the scope the sub-template is invoked with
- The scope is specified as the second argument of `template` or `include` invocation, after the sub-template name

Syntax:

- `{{ include "<sub-template-name>" <scope> }}`
- `{{ template "<sub-template-name>" <scope> }}`

`{{ include "mychart.mysubtemplate" . }}` – Scope at root of chart

`{{ template "mychart.mysubtemplate" .Values }}` – Scope at values.yaml file

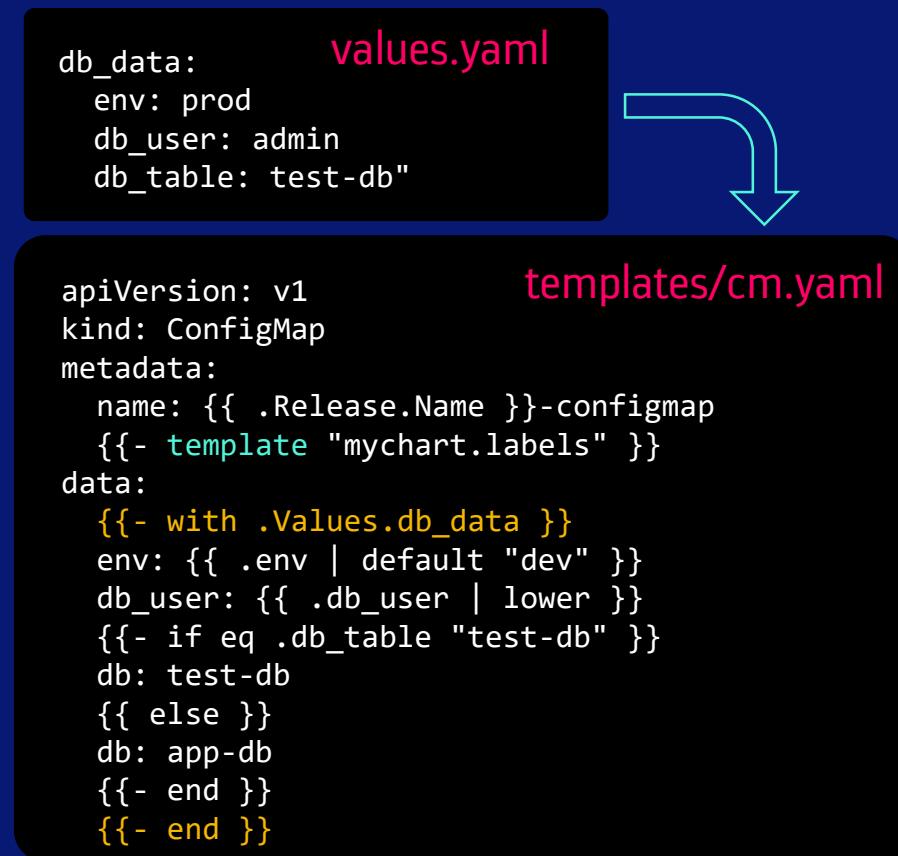
Helm Templates

<https://github.com/kunchalavikram1427>



Scope of sub-templates

- observe that `Chart.Name`, `.Chart.Version` are not replaced from the sub template because they are not in the scope where sub-chart is defined



```
templates/_helpers.tpl  
  
{{- define "mychart.labels" }}  
labels:  
  generator: helm  
  date: {{ now | htmlDate }}  
  chart: {{ .Chart.Name }}  
  version: {{ .Chart.Version }}  
{{- end }}
```

o/p

```
apiVersion: v1 kind: ConfigMap metadata:  
  name: RELEASE-NAME-configmap  
  labels:  
    generator: helm  
    date: 2021-01-21  
    chart: {{ .Chart.Name }}  
    version: {{ .Chart.Version }}  
  data:  
    env: prod  
    db_user: admin  
    db: test-db
```

Helm Templates

<https://github.com/kunchalavikram1427>



Scope of sub-templates

- Scope is now at `.` (dot indicates chart root). Now both `Chart.Name`, `.Chart.Version` are replaced from the sub template
- Observe the dot at the template action below

```
db_data:      values.yaml
  env: prod
  db_user: admin
  db_table: test-db"
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
  {{- template "mychart.labels" . }}
data:
  {{- with .Values.db_data }}
    env: {{ .env | default "dev" }}
    db_user: {{ .db_user | lower }}
    {{- if eq .db_table "test-db" }}
      db: test-db
    {{ else }}
      db: app-db
    {{- end }}
  {{- end }}
```

```
templates/_helpers.tpl
{{- define "mychart.labels" -}}
  labels:
    generator: helm
    date: {{ now | htmlDate }}
    chart: {{ .Chart.Name }}
    version: {{ .Chart.Version }}
{{- end -}}
```

o/p

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: RELEASE-NAME-configmap
  labels:
    generator: helm
    date: 2021-01-21
    chart: demo-chart
    version: 1.0.0
data:
  env: prod
  db_user: admin
  db: test-db
```



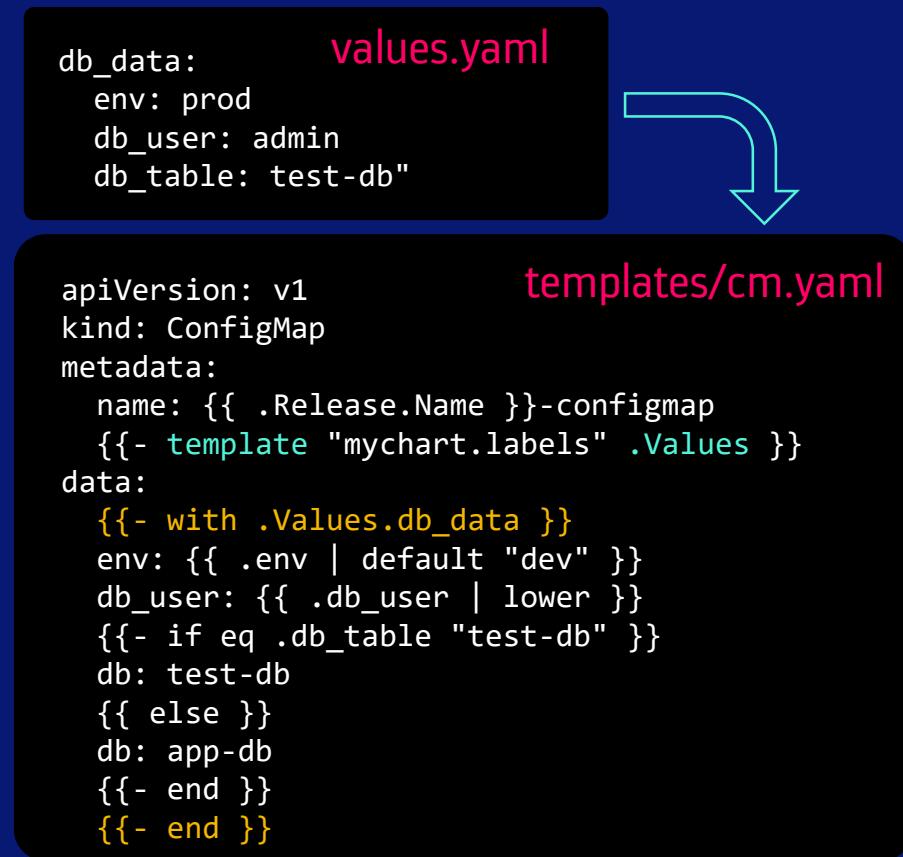
Helm Templates

<https://github.com/kunchalavikram1427>



Scope of sub-templates

- Scope is now at `.Values`
- Works similar to `with` function



templates/_helpers.tpl

```
{{- define "mychart.labels" }}  
  labels:  
    generator: helm  
    date: {{ now | htmlDate }}  
    env: {{ .db_data.env }}  
{{- end }}
```

o/p

```
apiVersion: v1  
kind: ConfigMap  
metadata:  
  name: RELEASE-NAME-configmap  
  labels:  
    generator: helm  
    date: 2021-01-21  
    env: prod  
data:  
  env: prod  
  db_user: admin  
  db: test-db
```

Helm Templates

<https://github.com/kunchalavikram1427>



Embed a sub-template using 'include'

- The `include` function is the preferred alternative to embed a template
- The output of the function can be processed by a pipeline before embedding, hence the indentation can be controlled, unlike in template case

```
db_data:  
  env: prod  
  db_user: admin  
  db_table: test-db"
```

values.yaml

templates/cm.yaml

```
apiVersion: v1  
kind: ConfigMap  
metadata:  
  name: {{ .Release.Name }}-configmap  
  {{- include "mychart.labels" . }}  
data:  
  {{- with .Values.db_data }}  
  env: {{ .env | default "dev" }}  
  db_user: {{ .db_user | lower }}  
  {{- if eq .db_table "test-db" }}  
  db: test-db  
  {{ else }}  
  db: app-db  
  {{- end }}  
  {{- end }}  
  {{- end }}
```

```
templates/_helpers.tpl
```

```
{{- define "mychart.labels" }}  
labels:  
  generator: helm  
  date: {{ now | htmlDate }}  
  chart: {{ .Chart.Name }}  
  version: {{ .Chart.Version }}  
{{- end }}
```

o/p

```
apiVersion: v1  
kind: ConfigMap  
metadata:  
  name: RELEASE-NAME-configmap  
  labels:  
    generator: helm  
    date: 2021-01-21  
    chart: demo-chart  
    version: 1.0.0  
data:  
  env: prod  
  db_user: admin  
  db: test-db
```

Helm Templates

<https://github.com/kunchalavikram1427>



Embed a sub-template using 'include'

- Notice the indentation issues in the final rendered manifest
- The output of the `include` function can be processed by a pipeline before embedding, hence the indentation can be controlled, unlike in template case

```
db_data:          values.yaml
  env: prod
  db_user: admin
  db_table: test-db"
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
  labels:
    {{ template "mychart.app" . }}
data:
  {{- with .Values.db_data -}}
    env: {{ .env | default "dev" }}
    db_user: {{ .db_user | lower }}
    db: {{ .db_table | lower }}
  {{- end -}}
{{ template "mychart.app" . }}
```

templates/_helpers.tpl

```
{{- define "mychart.app" -}}
app_name: {{ .Chart.Name }}
app_version: "{{ .Chart.Version }}"
{{- end -}}
```

templates/cm.yaml

o/p

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: RELEASE-NAME-configmap
  labels:
    app_name: demo-chart
    app_version: "1.0.0" ✘
data:
  env: prod
  db_user: admin
  db: test-db
  app_name: demo-chart
  app_version: "1.0.0" ✘
```

Helm Templates

<https://github.com/kunchalavikram1427>



Embed a sub-template using 'include'

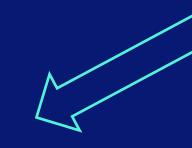
- The output of the `include` function is processed by a pipeline before embedding, hence the indentation is corrected

```
db_data:          values.yaml
  env: prod
  db_user: admin
  db_table: test-db"
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
  labels:
{{ include "mychart.app" . | indent 4 }}
data:
  {{- with .Values.db_data --}}
    env: {{ .env | default "dev" }}
    db_user: {{ .db_user | lower }}
    db: {{ .db_table | lower }}
  {{- end }}
{{ include "mychart.app" . | indent 2 }}
```



templates/cm.yaml



templates/_helpers.tpl

```
{{- define "mychart.app" -}}
app_name: {{ .Chart.Name }}
app_version: "{{ .Chart.Version }}"
{{- end -}}
```



o/p

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: RELEASE-NAME-configmap
  labels:
    app_name: demo-chart
    app_version: "1.0.0" ✓
data:
  env: prod
  db_user: admin
  db: test-db
  app_name: demo-chart
  app_version: "1.0.0" ✓
```

Helm Templates

Comments

Single line:

```
{/* Single line comment */}
```

Multiline:

```
{- /*  
This is a  
multi-line  
comment  
*/ -}
```

Helm Templates

<https://github.com/kunchalavikram1427>



Sub-templates as Variables

- Sub-templates may be used instead of variables, by declaring a template that acts like a variable

```
db_data:          values.yaml
  env: prod
  db_user: admin
  db_table: test-db"
```

templates/_helpers.tpl

```
{{/*
Compute the service name that depends on the release name.
This partial should be inlined every time the service name is needed.
*/}
{{- define "myChart.dbName" -}}
{{- printf "%s-%s" .db_data.env .db_data.db_table -}}
{{- end -}}
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
  labels:
data:
  {{- with .Values.db_data -}}
  env: {{ .env | default "dev" }}
  db_user: {{ .db_user | lower }}
  {{- end -}}
  db: {{ template "myChart.dbName" .Values }}
```

templates/cm.yaml

o/p

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: RELEASE-NAME-configmap
data:
  env: prod
  db_user: admin
  db: prod-test-db ✓
```

Helm Templates

Sprig

- Sprig provides out-of-box many useful functions to be used with Helm charts
- Many template functions for Go templates like String, Math, Encoding, OS, Cryptographic and Security
- Functions like trim, quote, upper, lower, regex, sha, genCA, genCAWithKey, indent etc. are available

String Functions

trim function removes space from either side of a string

```
trim "  hello  "
hello
```

trimSuffix trim just the suffix from a string

```
trimSuffix "-" "hello-"
hello
```

Replace perform simple string replacement.

```
"Hi Helm" |replace " " "-"
Hi-Helm
```

 More functions at <http://masterminds.github.io/sprig/>

Helm Templates

<https://github.com/kunchalavikram1427>



Use of nindent

- While `indent` function indents every line in a given string to the specified indent width, `nindent` also prepends a new line to the beginning of that string

```
image:          values.yaml
  name: nginx
  tag: latest
  pullPolicy: IfNotPresent
replicas: 1
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Release.Name }}-{{ .Chart.Name }}
  labels: {{ include "chart.labels" . | nindent 4 }}
spec:
  replicas: {{ default 1 .Values.replicas | int }}
  selector:
    matchLabels:
      app: {{ .Release.Name }}-{{ .Chart.Name }}
  template:
    [ . . . ]
```

templates/deployment.yaml

```
{{- define "chart.labels" -}}
app: {{- printf "%s-%s" .Release.Name .Chart.Name |
trunc 63 | trimSuffix "-" -}}
{{- end -}}
```

templates/_helpers.tpl

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: RELEASE-NAME-nginx-demo-chart
  labels:
    app: RELEASE-NAME-nginx-demo-chart
spec:
  replicas: 1
  selector:
    matchLabels:
      [ . . . ]
```

o/p

Helm

Helm Plugins

- Helm community maintains lots of custom plugin to ease out day to day operations with Helm

List all installed plugins

```
helm plugin list
```

Install a plugin

```
helm plugin install https://github.com/databus23/helm-diff
```

```
root@k8s-master:/home/osboxes# helm plugin list
NAME      VERSION DESCRIPTION
diff      3.1.3   Preview helm upgrade changes as a diff
```

- Helm diff shows the difference in manifests for 2 different revisions of a release

Helm

Helm Plugins

Show difference between 2 revisions/upgrade of a release

```
helm diff revision <release-name> 1 2
```

Uninstall the plugin

```
helm plugin uninstall diff
```



Terminal

```
root@k8s-master:/home/osboxes/helm# helm install demo-release helm-chart-plugin-d демо/
NAME: demo-release
LAST DEPLOYED: Sat Jan 23 01:20:22 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
root@k8s-master:/home/osboxes/helm# helm upgrade demo-release helm-chart-plugin- демо/
Release "demo-release" has been upgraded. Happy Helming!
NAME: demo-release
LAST DEPLOYED: Sat Jan 23 01:20:33 2021
NAMESPACE: default
STATUS: deployed
REVISION: 2
TEST SUITE: None
root@k8s-master:/home/osboxes/helm# helm diff revision demo-release 1 2
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
  labels:
    app: web-development
spec:
- replicas: 1
+ replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      name: myapp-pod
      labels:
        app: nginx
    [. . .]
```

Helm Templates

NOTES.txt

- The "help text" for your chart and provides instructions to your chart users
- At the end of a `helm install` or `helm upgrade`, Helm can print out a block of helpful information for users
- This information is highly customizable using templates
- Generally, the endpoints, URLs, Ports information are templated in the NOTES.txt file to show users on what URL and Port the application is running/accessible

templates/NOTES.txt

```
Thank you for installing {{ .Chart.Name }}.  
  
Your release is named {{ .Release.Name }}.  
  
To learn more about the release, try:  
  
$ helm status {{ .Release.Name }}  
$ helm get all {{ .Release.Name }}
```



```
helm install test-chart helm-template-01/
```

o/p

```
Thank you for installing demo-chart.  
  
Your release is named test-chart.  
  
To learn more about the release, try:  
  
$ helm status test-chart  
$ helm get all test-chart
```

```
POD_NAME=$(kubectl get pods --namespace {{ .Release.Namespace }} -l "app={{ template "prometheus.name" . }},component={{ .Values.pushgateway.name }}" -o jsonpath="{.items[0].metadata.name}")
```

```
kubectl --namespace {{ .Release.Namespace }} port-forward $POD_NAME 9091
```

Helm Templates

.helmignore file

- The `.helmignore` file is used to specify files you don't want to include in your helm chart
- If this file exists, the `helm package` command will ignore all the files that match the pattern specified in the `.helmignore` file while packaging your application
- This can help in avoiding unnecessary or sensitive files or directories from being added in your helm chart
- The `.helmignore` file supports Unix shell glob matching, relative path matching, and negation (prefixed with !). Only one pattern per line is considered.

.helmignore

```
# comment  
  
# Match any file or path named .git  
.git  
  
# Match any text file  
*.txt  
  
# Match only directories named mydir  
mydir/  
  

```

Helm Templates

Templating Best Practices

```
my-app:  
  config:  
    rbac:  
      enabled: true
```



```
tls_secret: xyz
```



```
secret_name: abc  
Config_Name: xyz
```



```
rbac:  
  enabled: true
```



```
tls:  
  secret: xyz
```



```
secretName: abc  
configName: xyz
```



Helm Charts Revisited

Subcharts and Global Values



Helm Charts

Subcharts and Global Values

- A chart can have dependencies, called **subcharts**, which have their own values and templates
- All subcharts are placed under **charts/** at the root of the chart
- A subchart is considered "stand-alone", which means it can neither depend on its parent chart nor access the values of it
- A parent chart can **override** values for subcharts by mentioning the subchart name in root/main **values.yaml** file
- Helm has a concept of global values that can be accessed by all charts. Use **global** keyword to mark a value as global in root **values.yaml** file

Terminal



```
$ tree web-application
```

```
web-application/
├── charts
│   ├── backend
│   │   ├── Chart.yaml
│   │   ├── templates
│   │   │   ├── backend-service.yaml
│   │   │   ├── backend.yaml
│   │   │   └── _helpers.tpl
│   │   └── ingress.yaml
│   └── values.yaml
├── database
│   ├── Chart.yaml
│   ├── templates
│   │   ├── _helpers.tpl
│   │   ├── mongodb-secret.yaml
│   │   ├── mongodb-service.yaml
│   │   └── mongodb.yaml
│   └── values.yaml
└── frontend
    ├── Chart.yaml
    ├── templates
    │   ├── frontend-configmap.yaml
    │   ├── frontend-service.yaml
    │   ├── frontend.yaml
    │   └── _helpers.tpl
    └── ingress.yaml
    └── values.yaml
    └── Chart.yaml
    └── templates
        └── ingress.yaml
        └── NOTES.txt
    └── values.yaml
```

<https://github.com/kunchalavikram1427>

Helm Charts

Subcharts and Global Values

- without values override and globals

```
values.yaml
db_data:
  env: prod
  db_user: admin
  db_table: admin-db
```

```
charts/backend/
values.yaml
db_data:
  env: dev
  db_user: dev
  db_table: dev-db
```

```
configMap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Chart.Name }}-configmap
data:
  {{- with .Values.db_data }}
  env: {{ .env | default "dev" }}
  db_user: {{ .db_user | lower }}
  db: {{ .db_table | lower }}
  {{- end }}
```

```
charts/backend/templates/configMap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Chart.Name }}-configmap
data:
  {{- with .Values.db_data }}
  env: {{ .env | default "dev" }}
  db_user: {{ .db_user | lower }}
  db: {{ .db_table | lower }}
  {{- end }}
```

helm template helm-chart-subcharts-globals/



```
$ tree helm-chart-subcharts-globals/
helm-chart-subcharts-globals/
├── charts
│   └── backend
│       ├── Chart.yaml
│       └── templates
│           └── configMap.yaml
│               └── values.yaml
└── Chart.yaml
    └── templates
        └── configMap.yaml
            └── values.yaml
```

o/p

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: backend-configmap
data:
  env: dev
  db_user: dev
  db: dev-db
...
apiVersion: v1
kind: ConfigMap
metadata:
  name: demo-chart-main-configmap
data:
  env: prod
  db_user: admin
  db: admin-db
```

Helm Charts

Subcharts and Global Values

- Overriding subchart values by main chart values and using global values

```
db_data:
  env: prod
  db_user: admin
  db_table: admin-db
backend:
  db_data:
    env: test
    db_user: test
    db_table: test-db
global:
  debug: true
```

values.yaml

```
db_data:
  env: dev
  db_user: dev
  db_table: dev-db
```

charts/backend/values.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Chart.Name }}-configmap
data:
{{- with .Values.db_data }}
  env: {{ .env | default "dev" }}
  db_user: {{ .db_user | lower }}
  db: {{ .db_table | lower }}
{{- end }}
  debug: {{ .Values.global.debug }}
```

charts/backend/templates/configMap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Chart.Name }}-configmap
data:
{{- with .Values.db_data }}
  env: {{ .env | default "dev" }}
  db_user: {{ .db_user | lower }}
  db: {{ .db_table | lower }}
{{- end }}
  debug: {{ .Values.global.debug }}
```

configMap.yaml

helm template helm-chart-subcharts-globals/

i For subchart to access newly override values from main values.yaml file, no changes are needed in yaml files as helm automatically manages the scope to point to the correct values.
Ex: `.Values.db_data` need not be changed to `.Values.backend.db_data` in subchart configMap.yaml file

Terminal

```
$ tree helm-chart-subcharts-globals/
helm-chart-subcharts-globals/
├── charts
│   └── backend
│       ├── Chart.yaml
│       ├── templates
│       │   └── configMap.yaml
│       └── values.yaml
└── Chart.yaml
    └── templates
        └── configMap.yaml
            └── values.yaml
```

o/p

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: backend-configmap
data:
  env: test
  db_user: test
  db: test-db
  debug: true
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: demo-chart-main-configmap
data:
  env: prod
  db_user: admin
  db: admin-db
  debug: true
```

Helm

Check the charts for errors

- Use `helm lint` to check the chart for syntax errors
- It's much faster then searching for errors after deployment

```
root@k8s-master:/home/osboxes/helm# helm lint helm-chart-plugin-demo/  
==> Linting helm-chart-plugin-demo/  
[INFO] Chart.yaml: icon is recommended  
  
1 chart(s) linted, 0 chart(s) failed
```

Debugging the chart

- For debugging, use `helm template` or `helm install --debug --dry-run`

Helm

Package and share helm chart

- Use `helm package` to package the chart into .tgz file
- Once packed, create a new directory and move the .tgz file into it
- Run `helm repo index` command to generate the index.yaml
- index.yaml contains metadata about the package, including the contents of a chart's Chart.yaml file. A valid chart repository must have an index file
- Now you can upload the chart and the index file to your chart repository using a sync tool or manually

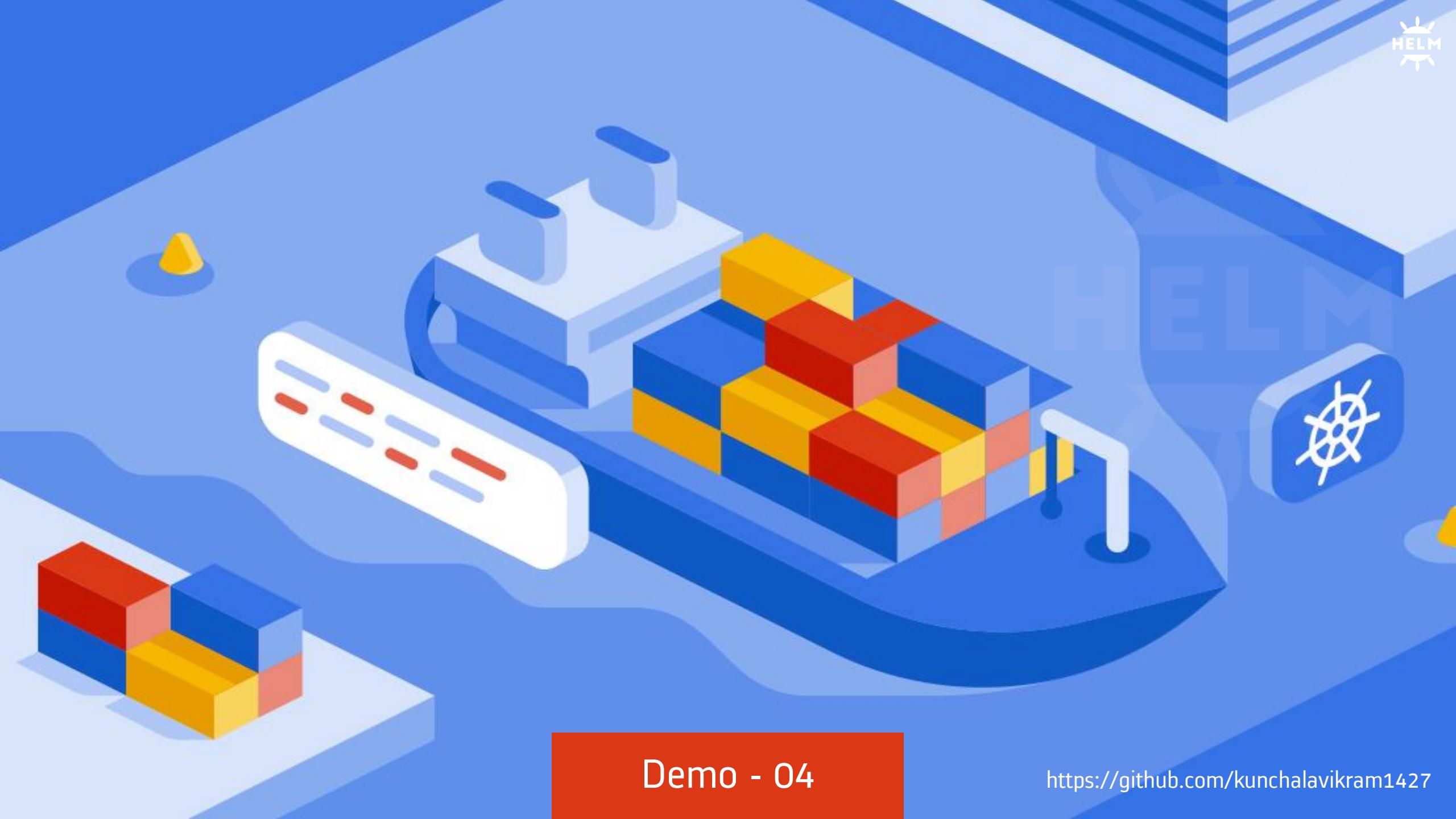
```
helm package helm-chart-plugin-demo
mkdir test-chart
mv nginx-demo-chart-1.0.0.tgz test-chart/
helm repo index test-chart -url <url-name>
```

```
root@k8s-master:/home/osboxes/helm# tree test-chart/
test-chart/
└── index.yaml
    └── nginx-demo-chart-1.0.0.tgz
```

```
$ helm push-artifactory nginx-demo-chart-1.0.0.tgz http://localhost:8081/artifactory/helm-chart-artifactory-repo --username myadminusername --password myadminpass
```

index.yaml

```
apiVersion: v1
entries:
  nginx-demo-chart:
    - apiVersion: v2
      appVersion: "1.0"
      created: "2021-01-2T02:59:01.8152-05:00"
      description: A Demo Helm nginx template
      digest: 9bc0320c1d84da96ef33e5e45
      name: nginx-demo-chart
      type: application
      urls:
        - nginx-demo-chart-1.0.0.tgz
      version: 1.0.0
    generated: "2021-01-23T02:59:01.813831407-05:00"
```



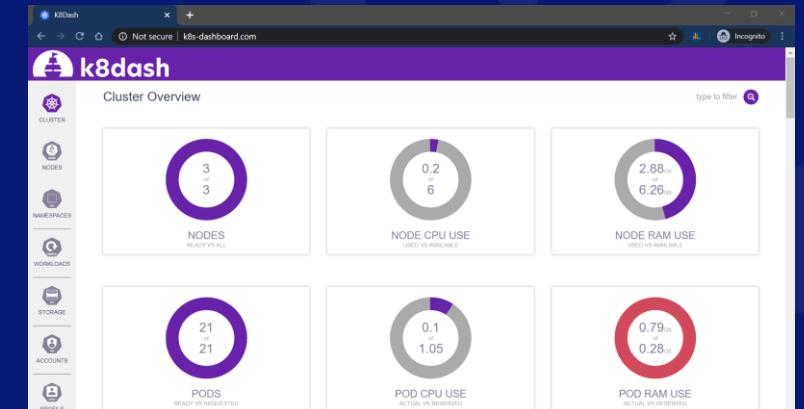
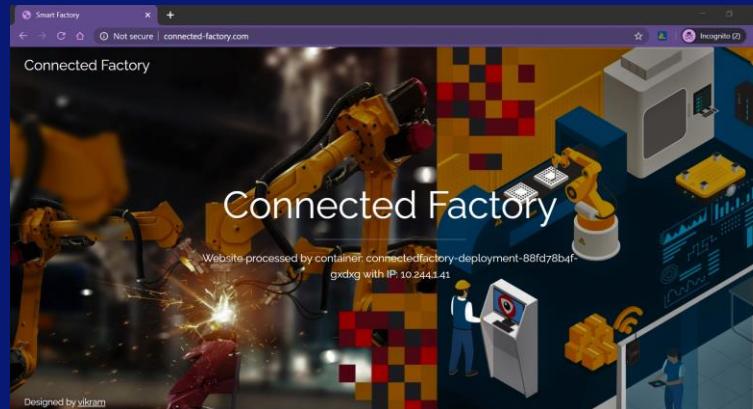
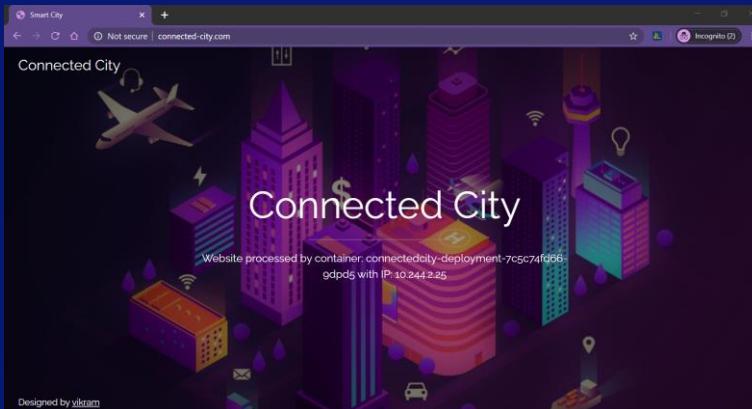
Demo - 04

<https://github.com/kunchalavikram1427>

Helm

Demo -04

- 3 front-end templatized applications
- Metrics-server and Traefik Ingress Controller installation within the same chart
- Helper functions & values files for each subchart and main chart
- NOTES.txt file to show URL info to the users
- Override values of subcharts



Demo -04

Install the chart

```
helm install demo helm-template-final
```

```
NAME: demo
LAST DEPLOYED: Sat Jan 23 03:31:27 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing connected-applications.
```

Your release is named demo.

To learn more about the release, try:

```
$ helm status demo
$ helm get all demo
```

You can access the applications at the following urls :
<http://connected-city.com>
<http://connected-factory.com>
<http://dashboard.com>

To access dashboard, a token is needed to be entered

Get token by running the command:

```
kubectl describe secret $(kubectl get secrets | grep k8s-dashboard-app | awk '{print $1}' )
```

Copy and paste the token in the browser

Have fun !!!



All demo charts are in the github: <https://github.com/kunchalavikram1427/helm>

```
$ tree helm-template-final
```

```
helm-template-final
├── charts
│   ├── connectedcity
│   │   ├── Chart.yaml
│   │   └── templates
│   │       ├── connectedcity-app-deployment.yml
│   │       ├── connectedcity-app-ingress.yml
│   │       ├── connectedcity-app-service.yml
│   │       └── _helpers.tpl
│   └── values.yaml
├── connectedfactory
│   ├── Chart.yaml
│   ├── templates
│   │   ├── connectedfactory-app-deployment.yml
│   │   ├── connectedfactory-app-ingress.yml
│   │   ├── connectedfactory-app-service.yml
│   │   └── _helpers.tpl
│   └── values.yaml
└── kubernetesdashboard
    ├── Chart.yaml
    ├── templates
    │   ├── _helpers.tpl
    │   ├── kubernetes-dashboard-clusterrolebinding.yml
    │   ├── kubernetes-dashboard-deployment.yml
    │   ├── kubernetes-dashboard-ingress.yml
    │   ├── kubernetes-dashboard-serviceaccount.yml
    │   └── kubernetes-dashboard-service.yml
    └── values.yaml
── Chart.yaml
── templates
│   ├── _helpers.tpl
│   ├── metrics-server.yml
│   ├── NOTES.txt
│   └── traefik-ingress-controller.yml
└── values.yaml
```

Helm

Demo -04

Get the token for dashboard

```
kubectl describe secret $(kubectl get secrets | grep k8s-
dashboard-app | awk '{print $1}' )
```

```
NAME: demo
LAST DEPLOYED: Sat Jan 23 03:31:27 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing connected-applications.
```

Your release is named demo.

To learn more about the release, try:

```
$ helm status demo
$ helm get all demo
```

You can access the applications at the following urls :

```
http://connected-city.com
http://connected-factory.com
http://dashboard.com
```

To access dashboard, a token is needed to be entered

Get token by running the command:

```
kubectl describe secret $(kubectl get secrets | grep k8s-dashboard-app | awk '{print
$1}' )
```

Copy and paste the token in the browser

Have fun !!!



All demo charts are in the github: <https://github.com/kunchalavikram1427/helm>

```
root@k8s-master:/home/osboxes/helm# kubectl describe secret
$(kubectl get secrets | grep k8s-dashboard-app | awk '{print $1}' )
Name: demo-k8s-dashboard-app-token-59mvj
Namespace: default
Labels: <none>
Annotations: kubernetes.io/service-account.name: demo-k8s-
dashboard-app
kubernetes.io/service-account.uid: e71b4886-bfb0-
47c3-ac7b-a1133f1ade49
Type: kubernetes.io/service-account-token
Data
=====
token:
eyJhbGciOiJSUzI1NiIsImtpZCI6IlE3a2Z4SDdNb2JhT1VIWnJRVWJQWVc2bER0OG1
aS2J5dThIdFFKdFRZNFEifQ.eyJpc3MiOiJrdWJlcmb5ldGVzL3NlcnZpY2VhY2NvdW5
0Iiwia3VizXJuZXRLcy5pbv9zZXJ2aWN1YWNNjb3VudC9uYW1lC3BhY2UiOijkZWZhDW
x0Iiwia3VizXJuZXRLcy5pbv9zZXJ2aWN1YWNNjb3VudC9zZWNyZXQubmFtZSI6ImR1b
W8tazhZLWRhc2hib2FyZC1hcHAtdG9rZW4tNT1tdmoiLCJrdWJlcmb5ldGVzLmlvL3N1
cnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC5uYW1lIjoiZGVtby1rOHMtZGFzaGJ
vYXJkLWFwcCIiMti1YmVybmV0ZXMuaw8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2
NvdW50LnVpZCI6ImU3MWI0ODg2LWJmYjAtNDdjMy1hYzdilWEExMTMzZjFhZGU0OSiI
nN1YiI6InN5c3RlbTpzZXJ2aWN1YWNNjb3VudDpkZWZhWx00mR1bw8tazhZLWRhc2hi
b2FyZC1hcHAifQ.bgwS8vxd34nGVrpTrrCFwH5HHQ0d279Ps7zESwme5Le4lCREUCRR
hDuvat-N4afC02WF-
M8Y9SIn1t5szHJGjpKeEr0JdTAd8ujvhrdFx9fyM5V7wYhQ8wtoyzLxeCUr57ECVoNG
bkULOA20723gohQwFBgP7pHHI_daUbfvV7kTUcuJXtnqXq0ovQoHZ_GWXIcN-
siOCMJWH4yn6P2qHYExVoI91Ac0ZoBMlpkHD64HZMHsFh5cgwDMkjjs6bSs03y2wRDE
7faxh0j1vP6kZ408G0BMJ82Mj_IkTPuVl1iy_6kGbhjnctUBXeKXwuj7h6Mzv_XkQa
Qy0TBsU7WFQ
```

<https://github.com/kunchalavikram1427>

Helm Commands



Helm Commands

```
# find helm version  
$ helm version --short
```

```
# install tiller in helm 2  
$ helm init
```

```
# list repos  
$ helm repo list
```

```
# add repos:  
$ helm repo add stable https://kubernetes-charts.storage.googleapis.com/  
$ helm repo add stable https://charts.helm.sh/stable  
$ helm repo add bitnami https://charts.bitnami.com/bitnami  
$ helm repo add mirantis https://charts.mirantis.com
```



Helm Commands

```
# update repos index  
$ helm repo update
```

```
# remove repos  
$ helm repo remove <repo-name>
```

```
# search for charts  
$ helm search repo mysql
```

```
# pull a chart without installing  
$ helm pull chartrepo/chartname
```

```
# render the template:  
$ helm template <release-name> <chart>  
$ helm template --debug <chart>  
$ helm template --debug <chart> -f override-values.yaml
```

Helm Commands

```
# deploy a chart/release:
```

```
$ helm install <release-name> <chart> - helm install demo-sql stable/mysql  
$ helm install <release-name> . - deployment from current directory  
$ helm install <release-name> <chart> --dry-run --debug - helm install without actual commit; can generate a release name  
$ helm install chart-name --debug --dry-run  
$ helm install mychart-0.1.0.tgz  
$ helm install my-apache bitnami/apache --version 8.0.2
```

```
# to override values with a values file, use the --values or the --f flag:
```

```
$ helm install <release-name> <chart> --set <some_parameter>=<new_value> - helm install my-nginx nginx --set replicaCount=2 --set service.type=LoadBalancer  
$ helm install --set user.name='student',user.password='password' stable/mysql  
$ helm install banzaicloud-stable/logging-operator --set rbac.enabled=false  
$ helm install <release-name> <chart> --values /path/to/values/file  
$ helm install -f myvalues.yaml chart # to be verified  
$ helm install --set-string long_int=1234567890 myredis ./redis  
$ helm install my-wordpress-prod bitnami/wordpress --version 10.1.4 -f values-production.yaml  
$ helm install --values myvalues.yaml stable/mysql
```

Helm Commands

```
# to deploy the release into a Kubernetes namespace, use the --namespace flag:  
$ helm install --namespace db stable/mysql
```

```
# list all releases:  
$ helm list --all
```

```
# upgrade a release  
$ helm upgrade <release-name> <chart>  
$ helm upgrade -f <values-file> <release-name> <chart-name>  
$ helm upgrade my-apache bitnami/apache --version 8.0.3  
$ helm upgrade --install <release-name> --values <values-file> <chart-name>
```

```
# rollback a release:  
$ helm rollback <release-name> <revision-number>
```



Helm Commands

```
# print release history  
$ helm history <release-name>
```

```
# get/display release status  
$ helm status <release-name>
```

```
# delete a release  
$ helm delete <release-name> - for helm2  
$ helm uninstall <release-name> - for helm3  
$ helm uninstall --keep-history <release-name>
```

```
# force delete a release  
$ helm delete --purge <release> - for helm2
```

```
# get details of a release like manifests etc.,  
$ helm get all <release-name>  
$ helm get manifest <release-name>  
$ helm get manifest demo-app
```



Helm Commands

```
# show all the environment information of helm  
$ helm env
```

```
# shows all values in the values.yaml file  
$ helm inspect values <chart-name>  
$ helm inspect values stable/mysql
```

```
# show list of needed dependencies for a chart and their download status  
$ helm dependency list <chart-name>
```

```
# download dependencies  
$ helm dependency update
```

```
# create a helm chart template layout  
$ helm create <chart-name>
```

```
# generate manifests after replacing values in templates. Can work without a cluster. No Release name generated  
$ helm template <chart-name>
```

Helm Commands

```
# generate manifests after replacing values in templates. helm install without actual commit; can generate a release name  
$ helm install <release-name> <chart> --dry-run --debug  
$ helm install <release-name> <chart> --dry-run --debug 2>&1 | less
```

```
# package a chart into .tgz  
$ helm package <chart-name>
```

```
# push chart to a repo  
$ helm push-artifactory xyz-demofile-0.1.1.tgz http://localhost:8081/artifactory/helm-chart-artifactory-repo --username  
myadminusername --password myadminpass
```

```
# examine a chart for possible issues like syntax errors and runs a series of tests to verify that the chart is well-formed.  
$ helm lint <chart-name>
```

```
# helm plugins  
$ helm plugin list  
$ helm plugin install https://github.com/databus23/helm-diff  
$ helm diff revision <release-name> 1 2  
$ helm plugin uninstall diff
```

Helm Extras

5

`nameOverride` replaces the name of the chart in the `chart.yaml` file, when this is used to construct Kubernetes object names. `fullnameOverride` completely replaces the generated name.

These come from [the template provided by Helm for new charts](#). A typical object in the templates is named

```
name: {{ include "<CHARTNAME>.fullname" . }}
```

If you install a chart with a deployment with this name, and where the `chart.yaml` file specifies

```
name: chart-name ...
```

- `helm install release-name .`, the Deployment will be named `release-name-chart-name`
- `helm install release-name . --set nameOverride=name-override`, the Deployment will be named `release-name-name-override`
- `helm install release-name . --set fullnameOverride=fullname-override`, the Deployment will be named `fullname-override`

The generated `...fullname` template is (one code branch omitted, still from the above link)

```
{{- define "<CHARTNAME>.fullname" -}}
{{- if .Values.fullnameOverride -}}
{{- .Values.fullnameOverride | trunc 63 | trimSuffix "-" -}}
{{- else -}}
{{- $name := default .Chart.Name .Values.nameOverride -}}
{{- printf "%s-%s" .Release.Name $name | trunc 63 | trimSuffix "-" -}}
{{- end -}}
{{- end -}}
```



Helm Extras

VS Code Extension

The screenshot shows the VS Code Extensions Marketplace. A search bar at the top left contains the text "helm". Below it, a list of extensions is displayed:

- Helm Intellisense** 0.8.0 by Tim Koehler: This extension provides intellisense for helm templates. It has 8K installs and 5 stars.
- vscode-helm** 0.4.0 by technosophos: Superseded by Kubernetes extension. It has 41K installs.
- Kubernetes** 1.2.1 by Microsoft: Develop, deploy and debug Kubernetes applications. It has 874K installs and 5 stars.
- Helma Skin** 1.0.18 by Tobi Schäfer: Syntax support for Helma's *.skin files. It has 1K installs.
- File Browser** 0.2.10 by Bodil Stokke: A nicer alternative to the file open dialog. It has 7K installs and 5 stars.
- SAG Container Dev Pack** 1.0.3 by SAG: A collection of essential container development extensions for VS Code product... It has 903 installs.
- Quick Browser** 1.5.0 by Christopher Kahn: A Helm-like file browser for VS Code. It has 891 installs and 5 stars.
- Node.js + DevOps Extension Pack** 0.0.12 by kunchalavikram1427: A collection of DevOps extensions for Node.js. It has 7K installs.

On the right side, a detailed view of the **Helm Intellisense** extension is shown. The extension page includes:

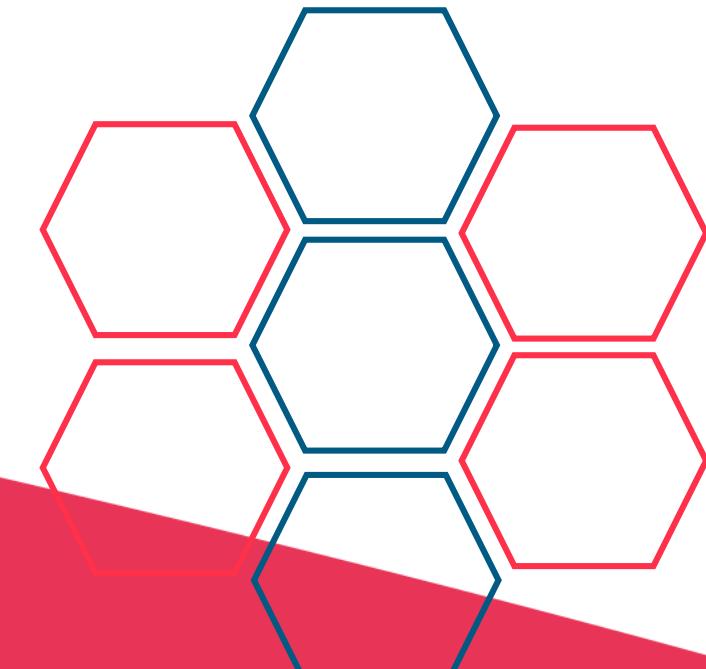
- Helm Intellisense** by tim-kohler.helm-intellisense
- Tim Koehler | 8,086 installs | ★★★★★ 5 stars
- Repository | License | vC
- A description: "This extension provides intellisense for helm templates".
- Buttons: Disable, Uninstall, Install (disabled).
- A note: "This extension is enabled globally."

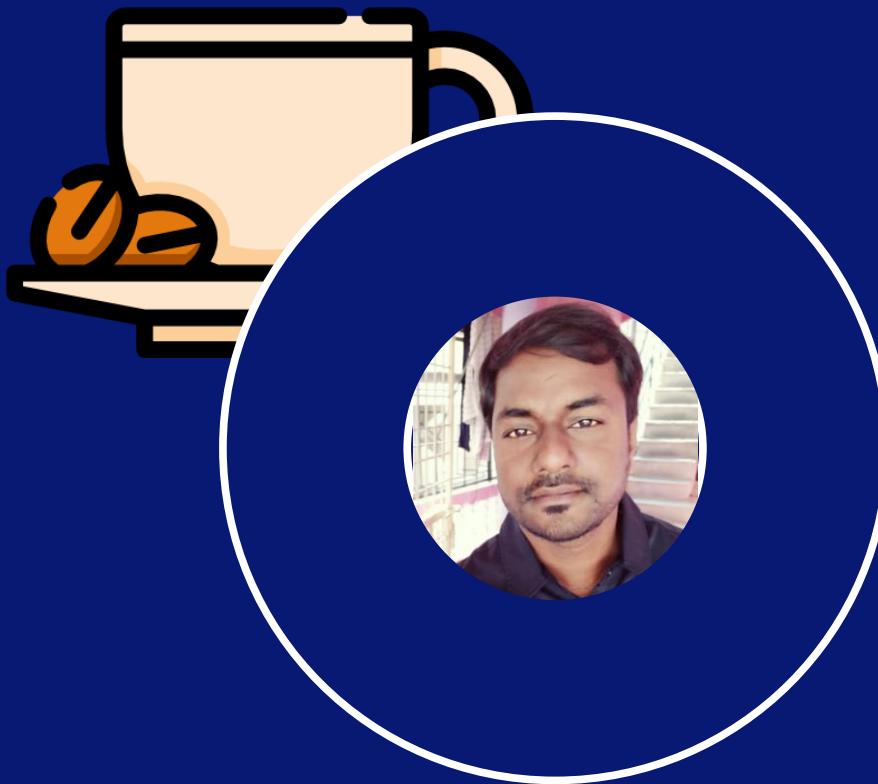
Below the extension details, there is a section titled "Helm-Intellisense" with 8.09K installs. It lists the following features:

- This simple extension provides intellisense for helm-templates. The `values.yaml` file of the chart will be read and evaluated automatically to provide intellisense.
- Autocomplete will also work for all Named Templates defined in the `_helpers.tpl`.
- Compatible with Windows and Linux/Unix.
- Support for custom named value files is provided(see settings section below).
- Lint command to validate correct values templating.
- Support and autocomplete for yaml anchors and labels
- The extension is compatible with the Kubernetes extension.
- Working with language type `yaml` and `helm-template`.

References

- Helm official site
<https://helm.sh/>
- Artifactory
<https://artifacthub.io/packages/search>
- Helm Alternatives
<https://kustomize.io/>
- Helm Online Practice
<https://www.katacoda.com/mjboxboat/courses/kubernetes-fundamentals-2/04-helm>
- Helm tutorials
<https://opensource.com/article/20/5/helm-charts>
<https://www.baeldung.com/kubernetes-helm>
https://kb.novaordis.com/index.php/Helm_Templates
<https://nirmata.com/2020/06/04/why-do-devops-engineers-love-helm/>
<https://docs.bitnami.com/tutorials/create-your-first-helm-chart/>
<https://dzone.com/articles/create-install-upgrade-and-rollback-a-helm-chart-p>
<https://medium.com/stakater/using-chartmuseum-as-a-chart-repository-for-helm-b4d12205f4c9>
<https://www.ibm.com/cloud/architecture/content/course/helm-fundamentals>





If you like my work, Kindly consider supporting me.
This will help me come up with more tutorials on DevOps tools and help
me in launching my own YouTube Channel.
Direct all your support to the below mentioned payment channels.
Your support is highly appreciated.

Check my Github for other tutorials on Docker & Kubernetes.
Planned tutorials: Ansible, Git and Jenkins

Need original presentation PPT? Send an email to below address

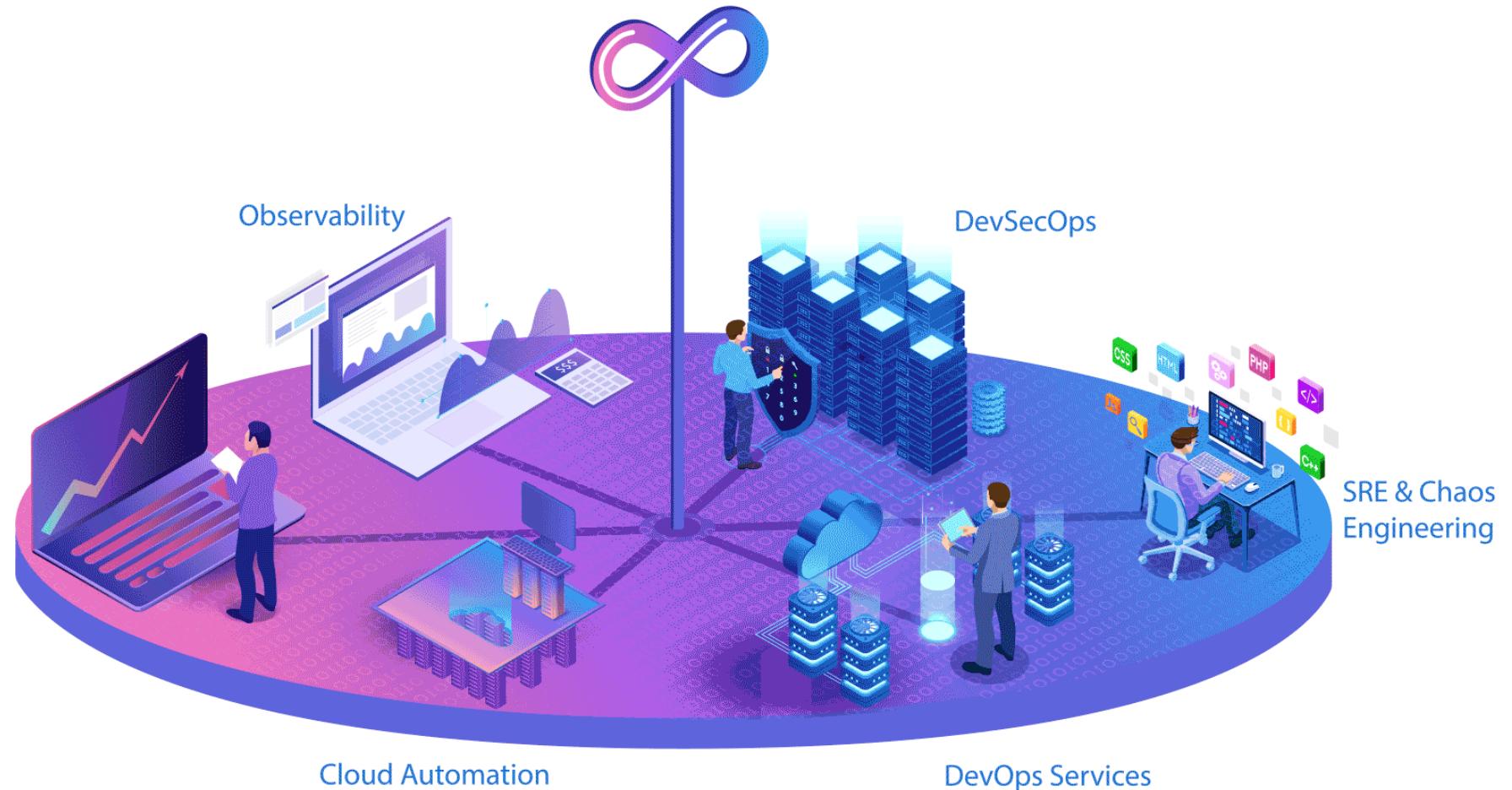
 kunchalavikram1427@gmail.com

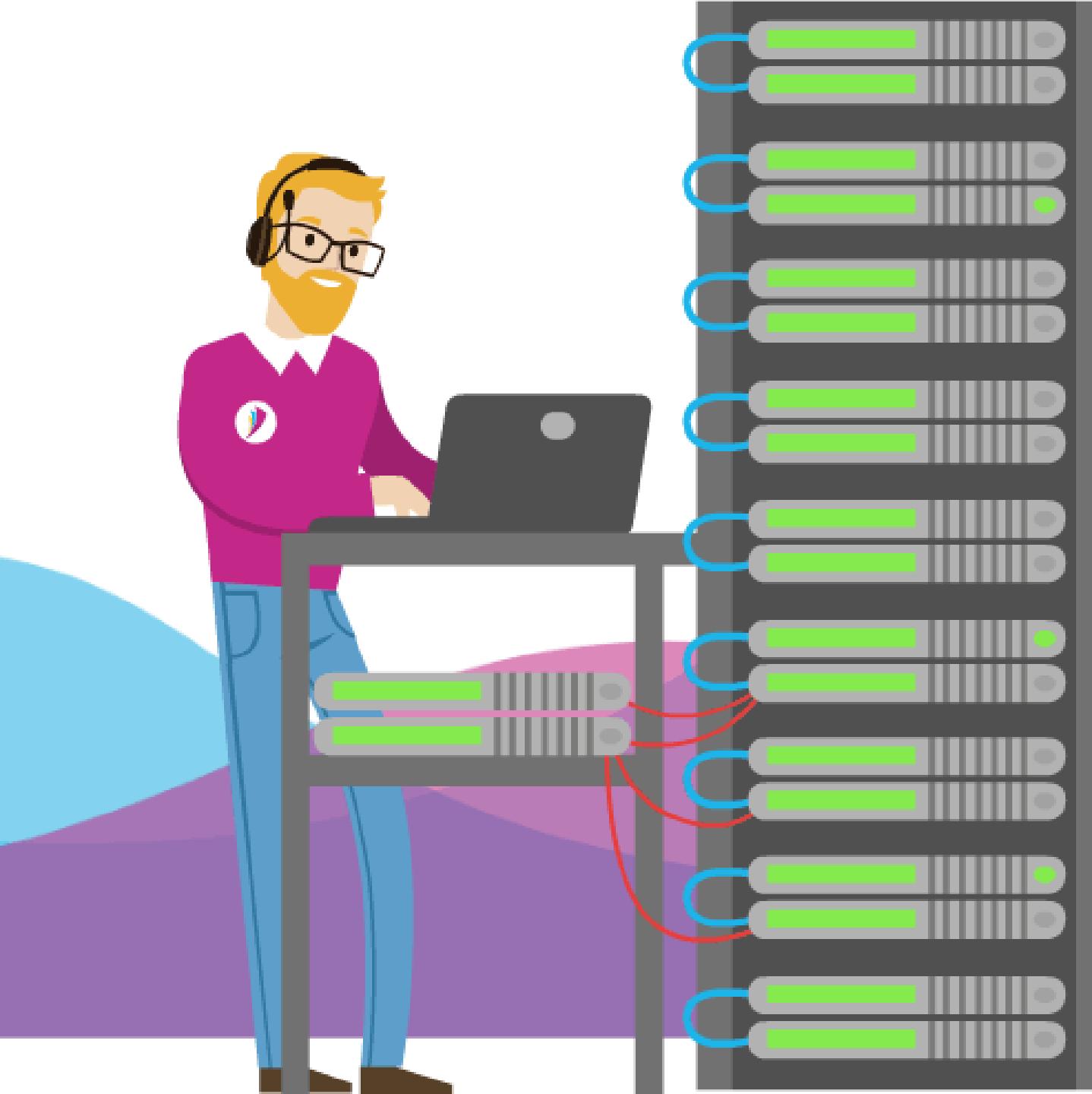
   +91-9885683322
  paypal.me/kunchalavikram

Q&A



Cloud Native
Development
(Microservices)





<https://github.com/kunchalavikram1427>