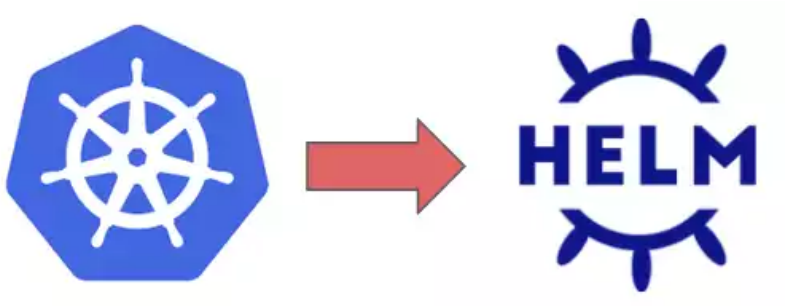
# Convert Kubernetes deployment YAML into Helm Chart YAML

Nov 26, 2020 · 8 min read · HELM CHART

· Last Modified : Nov 26, 2020 Share on: Author : Rahul Wagh




*Convert kubernetes yamls into Helm chart*

In this article we are going to look how can we convert Kubernetes YAMLs into Helm Chart YAMLs.

**Objective 1 : -** At first we are going to create simple Kubernetes deployment(k8s-deployment.yaml)` and in that deployment we are going to deploy a microservice application.

**Objective 2 : -** Secondly we are going to `create service(k8s-service.yaml) for exposing the deployment as a service on NodePort.

**Objective 3 : -** Here we are going to convert `Kubernetes deployment(k8s-deployment.yaml)` and `create service( k8s -service.yaml)` into a Helm Chart YAMls.

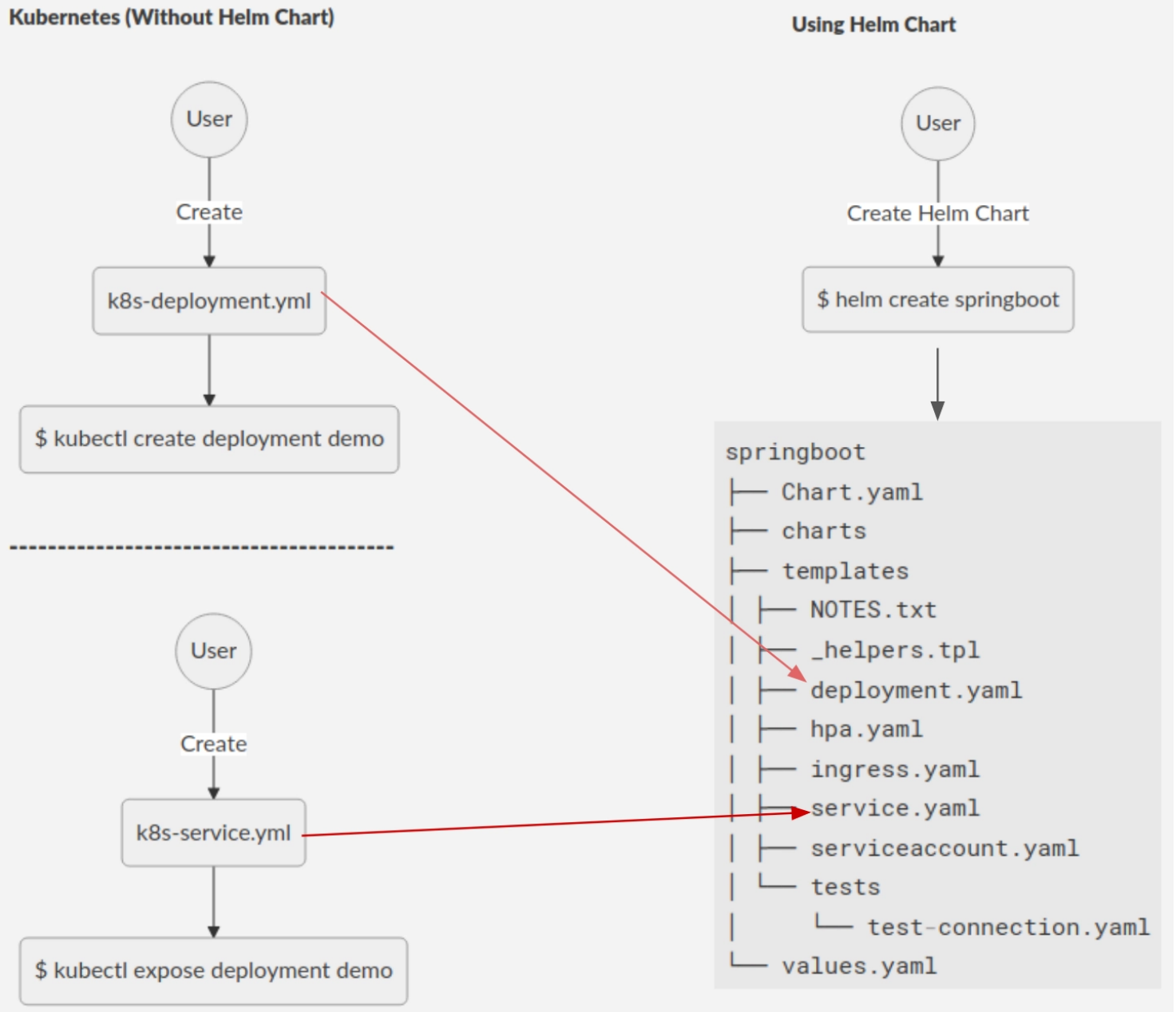
Search...

Categories

TERRAFORM 63 DOCKER 28
KUBERNETES 26 AWS 13
ANSIBLE 11 HELM-CHART 11
BLOGGING 6 SSL 6
SPRING-BOOT 5 QUARKUS 4
GITHUB 3 KUBESPRAY 3
PROMETHEUS-GRAFANA 3
VAGRANT 3
ALL CATEGORIES

Series

# 1. On a high level this is how it looks -




*process flow of converting kubernetes yamls into helm chart*



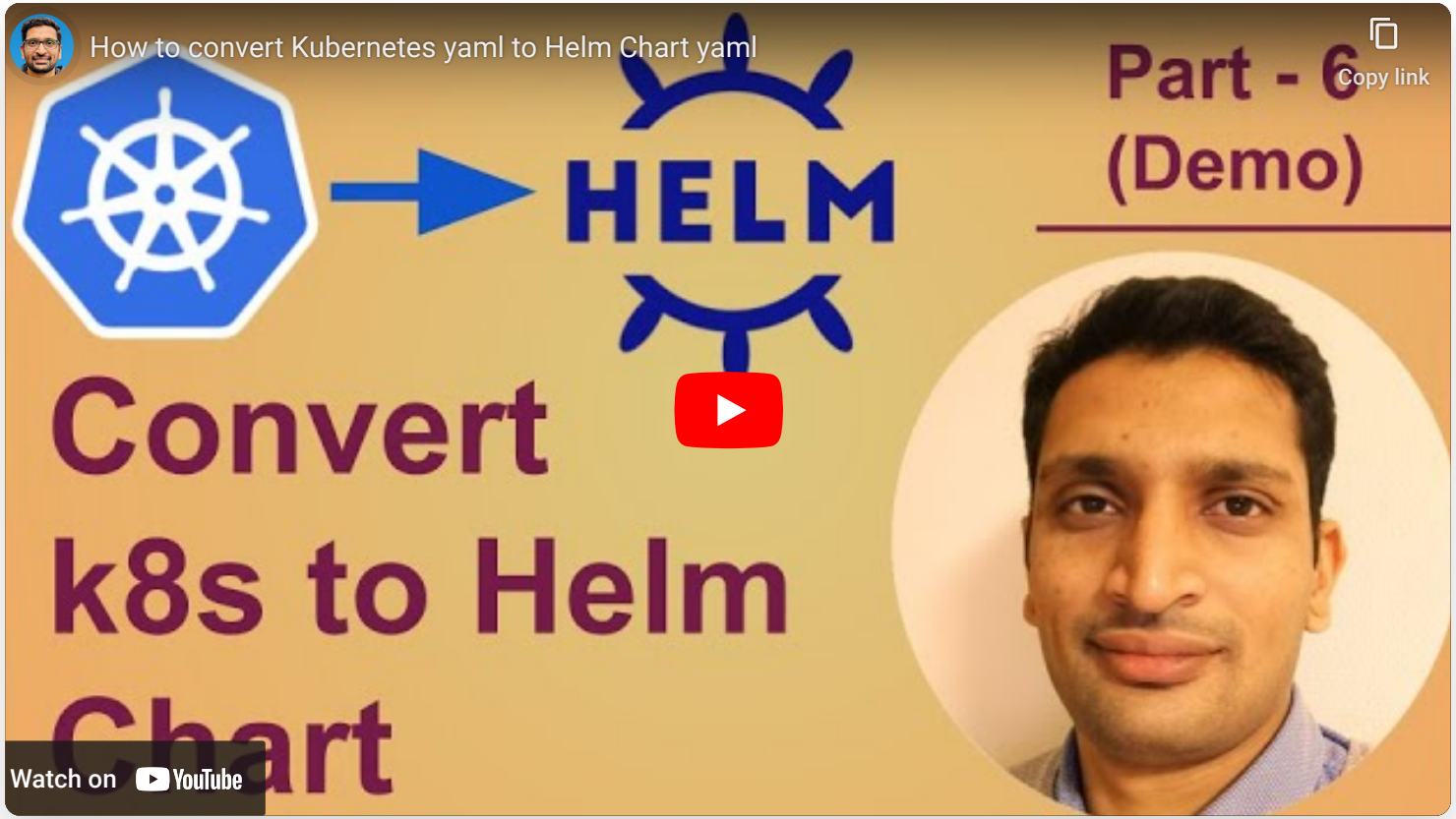
TERRAFORM 61 DOCKER 28
KUBERNETES 27 ANSIBLE 11
HELM-CHART 11 AWS 2
JENKINS 2 LINUX-COMMANDS 1

## Tags

KUBERNETES 18
HELM-CHART 10 BLOGGING 4
QUARKUS 4 DOCKER 3
GITHUB 3 SSL 3
KUBESPRAY 2 SPRING-BOOT 2
ANSIBLE 1 HADOOP 1
INDEX 1 NGINX 1
TERRAFORM 1

## Recent Posts


- Ansible Handlers Explained Real-World Use Cases & Examples
- Securing Sensitive Data in Terraform
- Boost Your AWS Security with Terraform : A Step-by-Step Guide
- How to Load Input Data from a File in Terraform?




## 2.Create kubernetes deployment YAML(k8s-deployment.yaml)

Before we jump to Helm Chart lets create simple YAMLs for kubernetes.

### 2.1 k8s-deployment.yaml

We are going to keep `k8s-deployment.yaml` very simple and we are going to deploy very small microservice

- Can Terraform be used to provision on-premises infrastructure?
- Fixing the Terraform Error creating IAM Role. MalformedPolicyDocument Has prohibited field Resource
- In terraform how to handle null value with default value?
- Terraform use module output variables as inputs for another module?


## Rahul Wagh

Its all about Open Source and DevOps, here I talk about Kubernetes, Docker, Java, Spring boot and practices.

READ MORE

application.

Let's create `k8s-deployment.yaml`

```bash
touch k8s-deployment.yaml
```

Open the deployment YAML into `vi` mode

```bash
vi k8s-deployment.yaml
```

Copy following deployment configs and save

```bash
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: demo
  name: demo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: demo
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: demo
    spec:
      containers:
      - image: rahulwagh17/kubernetes:jhooq-k8s-springboot
```

```
22         name: kubernetes
23         resources: {}
24 status: {}
```

### 2.2 Deploy k8s-deployment.yaml

After creating the `k8s-deployment.yaml` now you need to deploy it inside the kubernetes cluster

Run the following kubectl command to deploy

```BASH
1 kubectl apply -f k8s-deployment.yaml
```

Check the deployment status by running following command

```BASH
1 kubectl get deployment demo
```

It should return the following status on successful deployment

```
1 NAME   READY   UP-TO-DATE   AVAILABLE   AGE
2 demo   1/1     1            1           4m52s
```

## 3. Create kubernetes service YAML(k8s-service.yaml)

After successful deployment, now we need to create service YAML .i.e. - k8s-service.yaml

Let's create `k8s-service.yaml`

```bash
touch k8s-service.yaml
```

Open the service YAML into `vi` mode

```bash
vi k8s-service.yaml
```

Copy following deployment configs and save

```bash
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: demo
  name: demo-service
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 8080
  selector:
    app: demo
  type: NodePort
status:
  loadBalancer: {}
```

Run the following kubectl command to expose the service as `NodePort` on port `8080`

```bash
kubectl apply -f k8s-service.yaml
```

Check the service status by running following command

```bash
kubectl get service demo-service
```

It should return the following status once you expose it successfully

```bash
NAME           TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
demo-service   NodePort   10.233.10.61   <none>        8080:30036/TCP   9s
```

We could tes it by accessing the successfully deployed application on browser

*microservice output after deploying it into kubernetes cluster*

# 4. Convert kubernetes YAML to Helm Chart

Now lets start converting kubernetes(k8s) YAMLs into Helm Chart.

## 4.1 Create your Helm Chart

The first step for this conversion is to create the Helm Chart, so that we can have all the necessary YAMLs generated.

Here is comparision of YAMLs generated by Helm Chart and Kubernetes(k8s) -

| | YAMLs Generated by Helm Chart | Kubernetes(k8s) YAMLs |
|---|---|---|
| 1 | **Chart.yaml** | |
| 2 | **helper.tpl** | |
| 3 | **deployment.yaml** | k8s-deployment.yaml |
| 4 | **hpa.yaml** | |
| 5 | **ingress.yaml** | |
| 6 | **service.yaml** | k8s-service.yaml |
| 7 | **serviceaccount.yaml** | |
| 8 | **test-connection.yaml** | |
| 9 | **values.yaml** | In k8s-deployment.yaml<br>1. replicas: 1<br>2. docker image =`rahulwagh17/kubernetes:jhooq-k8s-springboot` |

Lets create `demo-helm-chart`

```bash
helm create demochart
```

Verify the YAML files generated after running the `helm create` command

```bash
tree demochart
```

It should return you back with following file tree structure

```bash
demochart
├── Chart.yaml
├── charts
├── templates
│   ├── NOTES.txt
│   ├── _helpers.tpl
│   ├── deployment.yaml
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── service.yaml
│   ├── serviceaccount.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml
```

## 4.2 Convert and Update Chart.yaml, deployment.yaml, service.yaml and values.yaml

Follow the instructions for updating the each YAML

### 4.2.1 Chart.yaml

The first YAML which we are converting is `chart.yaml` but it is optional and does not require any change but it would be nice to update some value with regards to your project name.

So update the following values inside your `chart.yaml`

```BASH
apiVersion: v2
name: demochart
description:  Convert Kubernetes(yamls) to Helm Chart
type: application
version: 0.1.0
appVersion: 1.16.0
```

### 4.2.2 deployment.yaml

The next YAML to convert is `deployment.yaml` but here we need to disable the `livenessProbe` and `readinessProbe` because it is very small application and we can verify the application deployment manually.

When we generate the Helm Chart then by default `deployment.yaml` is prefilled/pre-populated with some configs, so we need to update only -

1. containerPort : 8080

```BASH
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "demochart.fullname" . }}
  labels:
    {{- include "demochart.labels" . | nindent 4 }}
spec:
  {{- if not .Values.autoscaling.enabled }}
  replicas: {{ .Values.replicaCount }}
  {{- end }}
  selector:
    matchLabels:
      {{- include "demochart.selectorLabels" . | nindent 6 }}
  template:
    metadata:
      {{- with .Values.podAnnotations }}
      annotations:
```

```
        {{- toYaml . | nindent 8 }}
      {{- end }}
      labels:
        {{- include "demochart.selectorLabels" . | nindent 8 }}
    spec:
      {{- with .Values.imagePullSecrets }}
      imagePullSecrets:
        {{- toYaml . | nindent 8 }}
      {{- end }}
      serviceAccountName: {{ include "demochart.serviceAccountName" . }}
      securityContext:
        {{- toYaml .Values.podSecurityContext | nindent 8 }}
      containers:
        - name: {{ .Chart.Name }}
          securityContext:
            {{- toYaml .Values.securityContext | nindent 12 }}
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag | default .Chart.AppVersic
          imagePullPolicy: {{ .Values.image.pullPolicy }}
          ports:
            - name: http
              containerPort: 8080 #update the port here to 8080
              protocol: TCP
          livenessProbe:
            httpGet:
              path: /
              port: http
          readinessProbe:
            httpGet:
              path: /
              port: http
          resources:
            {{- toYaml .Values.resources | nindent 12 }}
    {{- with .Values.nodeSelector }}
```

### 4.2.3 service.yaml

The next YAML which we need to convert is service.yaml and here we do not need to update anything configs, we can pretty much keep it in the same shape.

```BASH
apiVersion: v1
kind: Service
metadata:
  name: {{ include "demochart.fullname" . }}
  labels:
    {{- include "demochart.labels" . | nindent 4 }}
spec:
  type: {{ .Values.service.type }}
  ports:
    - port: {{ .Values.service.port }}
      targetPort: http
      protocol: TCP
      name: http
  selector:
    {{- include "demochart.selectorLabels" . | nindent 4 }}
```

### 4.2.4 values.yaml

The last YAMLs which is left for conversion is `values.yaml` and here we need to update couple values -

1. **repository : rahulwagh17/kubernetes:jhooq-k8s-springboot**
2. **port : 8080**

Here is how it should look like

```BASH
replicaCount: 1

image:
  repository: rahulwagh17/kubernetes:jhooq-k8s-springboot #update the docker image name
  pullPolicy: IfNotPresent
  # Overrides the image tag whose default is the chart appVersion.
  tag: ""

imagePullSecrets: []
```

```yaml
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: ""

podAnnotations: {}

podSecurityContext: {}
  # fsGroup: 2000

securityContext: {}
  # capabilities:
  #   drop:
  #   - ALL
  # readOnlyRootFilesystem: true
  # runAsNonRoot: true
  # runAsUser: 1000

service:
  type: NodePort
  port: 8080

ingress:
  enabled: false
  annotations: {}
    # kubernetes.io/ingress.class: nginx
    # kubernetes.io/tls-acme: "true"
  hosts:
    - host: chart-example.local
      paths: []
  tls: []
  #  - secretName: chart-example-tls
  #    hosts:
  #      - chart-example.local
```

# 5. Verify the Conversion of YAMLs

Lets verify the conversion of the YAMLs with `helm template` command. This command will show us the `serviceaccount.yaml, service.yaml` and `deployment.yaml` which will be equivalent of the kubernetes(k8s) YAMLs which we generated manually.

```bash
helm template demoHelmChart
```

It should return you back with -

```bash
---
# Source: demochart/templates/serviceaccount.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: RELEASE-NAME-demochart
  labels:
    helm.sh/chart: demochart-0.1.0
    app.kubernetes.io/name: demochart
    app.kubernetes.io/instance: RELEASE-NAME
    app.kubernetes.io/version: "1.16.0"
    app.kubernetes.io/managed-by: Helm
---
# Source: demochart/templates/service.yaml
apiVersion: v1
kind: Service
metadata:
  name: RELEASE-NAME-demochart
  labels:
    helm.sh/chart: demochart-0.1.0
    app.kubernetes.io/name: demochart
    app.kubernetes.io/instance: RELEASE-NAME
    app.kubernetes.io/version: "1.16.0"
```

```
    app.kubernetes.io/managed-by: Helm
spec:
  type: NodePort
  ports:
    - port: 8080
      targetPort: http
      protocol: TCP
      name: http
  selector:
    app.kubernetes.io/name: demochart
    app.kubernetes.io/instance: RELEASE-NAME
---
# Source: demochart/templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: RELEASE-NAME-demochart
  labels:
    helm.sh/chart: demochart-0.1.0
    app.kubernetes.io/name: demochart
    app.kubernetes.io/instance: RELEASE-NAME
    app.kubernetes.io/version: "1.16.0"
    app.kubernetes.io/managed-by: Helm
spec:
  replicas: 1
  selector:
    matchLabels:
...
```

There is one more command `lint` which will tell you if there are any syntactical errors in the YAMls.

BASH
```
helm lint demochart
```

BASH
```
==> Linting demochart
[INFO] Chart.yaml: icon is recommended
```

```
3
4  1 chart(s) linted, 0 chart(s) failed
```

## 6. Run/Install Helm Chart

The final step which we need to do is to install the Helm Chart and verify the rest endpoint so that we can test conversion of YAMLs.

Run the following helm command to install the chart

```BASH
1 helm install k8sToHelmChart demochart
```

> **Read More** -
>
> 1. Helm chart - How to Add/Install plugins
> 2. Getting started with Helm Chart
> 3. Helm chart - WordPress Installation with MariaDB on Kubernetes
> 4. Helm chart - Build you first helm chart with Spring Boot
> 5. Helm Chart - Convert Kubernetes YAML into Helm Chart YAML
> 6. Helm Chart - Pass environment variables
> 7. Helm Chart - Plugin
> 8. Helm Chart - Dry Run Install
> 9. Helm Chart - How to create multiple values files inside helm chart?
> 10. Helmfile - How to use Helmfile for managing helm chart?

## Posts in this series

- How to use Helmfile for managing helm chart?
- How to create multiple values files inside helm chart?
- Pass environment variables into Helm Chart?
- How to fix - Helm install unknown flag --name/Error must either provide a name or specify --generate-name?
- Understanding Helm dry run for template debugging
- How to fix - Error create failed to create Secret invalid metadata.name Invalid value DNS-1123 subdomain must consist of lower case alphanumeric characters - or ., and must start and end with an alphanumeric character (e.g. example.com, regex used for validation is)
- Convert Kubernetes deployment YAML into Helm Chart YAML
- Helm chart - Wordpress Installation with MariaDB on Kubernetes
- Helm chart - How to Add/Install plugins
- Getting Started with Helm Chart
- Building first Helm Chart with Spring Boot Microservices