HELM

Why Helm

Kubectl is awesome for single files. But when you start talking about go-live there are multiple files that need to go together - a deployment, a service and an ingress is typical. Maybe a HPA and a secret or configmap. Maybe you are fine with deploying these one at a time - but what happens if you mess up and need to rollback ....you will have to do that manually too. This is where helm comes in - helm is a tool that helps you deploy your apps in K8s and while that itself is fine...my personal experience has been how it helps solve the local story vs upper envs. In one of my earlier projects - we did not use Helm , instead we used templated k8s yaml and when the CI/CD system went to deploy to an env...we update the templated variables using Azure Devops variables (or key vault). It worked pretty well but we had two major issues

1. Local story was messed up - if a developer wanted to deploy the yaml templates on his/her k8s instance...they needed to replace the templated placeholders with actual values.
2. In upper environments - there was no way for us to do side by side deployments or even attempt canary deployments.

Getting started with helm - core basics

Best introduction to helm - Donovan Brown. https://www.youtube.com/watch?v=2HPsPOwHOlY

```
#create a directory called helm
helm create helm
```

Concepts

- A **Chart** is a Helm package. It contains all of the resource definitions necessary to run an application, tool, or service inside of a Kubernetes cluster
- A **Repository** is the place where charts can be collected and shared. Think nuget
- A **Release** is an instance of a chart running in a Kubernetes cluster. One chart can often be installed many times into the same cluster. And each time it is installed, a new release is created.
- **Chart.yaml**: A YAML file containing information about the chart
- **values.yaml**: The default configuration values for this chart
- **templates/** : A directory of templates that, when combined with values,will generate valid Kubernetes manifest files.
- **templates/NOTES.txt** : OPTIONAL: A plain text file containing short usage notes

Change a few things in the values.yaml

```
image:
  repository: nishantnepal/k8s-svc-c
  pullPolicy: IfNotPresent
  # Overrides the image tag whose default is the chart appVersion.
  tag: "latest"
```

```
service:
  type: NodePort
  port: 80
```

```
cd .\helm\

# if you want to see the transposed difference
helm template .
# Above will not transform the release name...do this
# Dry run show what it will run
helm install webappc . --dry-run --debug
# and finally to install on local - webappC is my release name
helm install webappc .

kubectl get svc
# you should be able to browse to the site
# http://localhost:30530/chaincallc/configvalue?configKey=DemoAppSettingKey
# http://localhost:30530/chaincallc/configvalue?configKey=MySuperSecretKey
```

That is awesome - changed a few lines and hit a command and we are done. But real apps are a bit different right, we need to add in ingress resource (or modify since that is already built in) and since our app has configurations that differ between environments we need to externalize these. In our case we will choose secrets since we have one regular configuration and one super secure configuration.

- Create a new file `secret.yaml` under the templates directory

```
{{- if .Values.secrets.enabled -}}
{{- $fullName := include "helm.fullname" . -}}
apiVersion: v1
kind: Secret
metadata:
 name: {{ $fullName }}
 labels:
    {{- include "helm.labels" . | nindent 4 }}
type: Opaque
stringData:
  DemoAppSettingKey: "from secret"
  MySuperSecretKey: {{ .Values.secrets.mySuperSecretKey }}
{{- end }}
```

- In `values.yaml` add the following lines

```
secrets:
  enabled: true
```

```
  mySuperSecretKey: "demo"
```

- In `deployment.yaml` add this after the probes

```
envFrom:
            {{- if .Values.secrets.enabled}}
              - secretRef:
                  name: {{ include "helm.fullname" . }}
            {{- end }}
```

- Add Configs
  - Add a `configs` folder under helm
  - Create two files under there `values-template.yaml and values-local.yaml`
  - Add to `values-local.yaml`

```
secrets:
  enabled: true
  mySuperSecretKey: "from local file"
```

- Add to `values-template.yaml`

```
secrets:
  enabled: true
  #can either be replaced using set OR using token replacement in azure devops
  mySuperSecretKey: "#{MY_SUPER_SECRET_KEY}#"
```

- Add `values-local.yaml` to gitignore - **these should contain sensitive values that you dont want checked into source control**

- Test local

```
helm upgrade -f configs/values-local.yaml webappc .
# VERIFY
# http://localhost:30530/chaincallc/configvalue?configKey=MySuperSecretKey
```

- Package and Push to ACR

```
#user name is the client/application id in portal UI or appId in json - this is
the service principal - any account that has access to acr
helm registry login nishant.azurecr.io --username a5da4b0b-bff6-443a-8e6b-
06b70c622cf2 --password XXXX
```

```
#FOR ADO this will be the build id $(Build.BuildNumber)
BUILD_ID=12345
helm package . --app-version="1.19.12346" --version="1.19.12346"
az acr helm push -n nishant webappc-1.19.12345.tgz --force

# ----------------DONT USE START
# HELM V3 is somewhat broken - no option to update chart version and app version
by ci...have to do manually
helm chart save . nishant.azurecr.io/webappc:1.19.$BUILD_ID
helm chart push  nishant.azurecr.io/webappc:1.19.12345
#THIS IS EASIER BUT WILL BE DEPRECATED https://github.com/Azure/azure-
cli/issues/14467
helm package .
az acr helm push -n nishant webappc-0.1.0.tgz --force
# ----------------DONT USE END

# AFTER Connect to aks cluster
# https://www.mytechramblings.com/posts/gitops-with-azure-devops-helm-acr-flux-
and-k8s/

#add repo if not added
helm repo add nishant https://nishant.azurecr.io/helm/v1/repo --username a5da4b0b-
bff6-443a-8e6b-06b70c622cf2 --password XXX
#verify
helm repo ls
#search repo for charts
helm search repo nishant -l
#install or upgrade
helm upgrade --install -f configs/values-cicd.yaml --set
secrets.mySuperSecretKey="from command line" webappc nishant/webappc  --version
1.19.12346
```

Canary Deployments

or - testing in production 😃. Often times, maybe when releasing a new feature, or making some change...you often want to release that in parallel to existing code so that change/effect is limited. Do you need helm for that -probably not...you can use kubernetes for that but Helm does make it easier because of one fundamental aspect of helm - side by side release. First lets talk about what is at the core of canary deployments for kubernetes. Flashback - we know k8s typically consist of pods linked to rs/deployments and connected/exposed via services/ingress. Let hone into that - services keep track of deployments/pods via labels (loosely connected) so for canary (side by side) we want to deploy the canary k8s deployment/pods BUT KEEP THE SAME service and labels -so basically that service will either forward to regular deployment or canary.

Demo

We need to make a couple of changes (this is one approach - not necessarily the best)

1. add `canaryDeployment: false` to `values.yaml`
2. made `service.yaml` conditional by enclosing it into an if block `{{- if not .Values.canaryDeployment -}}`
3. added `values-canary.yaml` config file with the following contents

```
replicaCount: 1
#THIS IS KEY
canaryDeployment: true

autoscaling:
  enabled: false

image:
  #THIS WILL THE NEW TAG
  tag: "latest"

ingress:
  # WE DONT WANT A NEW PUBLIC ENDPOINT
  enabled: false

# resources:
#   limits:
#     cpu: 500m
#     memory: 250Mi
#   requests:
#     cpu: 500m
#     memory: 250Mi

# secrets:
#   enabled: true
#   mySuperSecretKey: "TBD"
```

4. Update in `_helpers.tpl` under `Selector labels` : Comment out the `app.kubernetes.io/instance: {{ .Release.Name }}` selector since we dont want to tie the pods to a release name.

5. Install the canary release - NOTICE the canary config comes after the deployment config. You can make canary independent - i like it this way so that my resources etc remain same but it depends on scenario.
`helm upgrade --install -f configs/values-cicd.yaml -f configs/values-canary.yaml --set secrets.mySuperSecretKey="from canary command line" webappc-canary nishant/webappc --version 1.19.12346`

6. Validate : If you go to the url `http://k8sws-svc-c.20.75.68.97.nip.io/chaincallc/configvalue?configKey=MySuperSecretKey` and hit refresh a couple of times, you should see value fluctuating between the two deployments.

> NOTE: in this example we did a 50/50 split (1 replica each for canary and regular) but in production, you will see canary at a lower percentage - 25% or so.

MISC

https://github.com/helm/helm/issues/8194#issuecomment-635879040
https://www.reddit.com/r/devops/comments/fcol46/how_do_you_usually_deal_with_versioning_of_helm/

ASSIGNEE=$(az aks show -g $RESOURCE_GROUP -n $NAME --query identityProfile.kubeletidentity.clientId -o tsv)