Application Deployment

Application Deployment

For deploying larger or configurable applications there are 3 popular ways in Kubernetes

- Kompose
  Convert docker compose to Kubernetes
- Helm
  Package manager for Kubernetes
- Kustomize
  Kubernetes native configuration management

Kompose

CONVERT DOCKER COMPOSE TO KUBERNETES

Kompose

Kompose allows you to easily convert from docker-compose file to Kubernetes yaml

Steps:

- Install Kompose from https://kompose.io
- Place your docker-compose file in a directory
- In that directory execute:
  `kompose convert`
- Apply your newly created yaml files to Kubernetes

Helm

PACKAGE MANAGER FOR KUBERNETES

What is Helm?

Helm helps you manage Kubernetes applications — Helm Charts help you define, install, and upgrade even the most complex Kubernetes application.

Charts are easy to create, version, share, and publish — so start using Helm and stop the copy-and-paste.

The latest version of Helm is maintained by the **CNCF** - in collaboration with **Microsoft**, **Google**, **Bitnami** and the **Helm contributor community**.

The purpose of Helm

Helm is a tool for managing Kubernetes packages called charts. Helm can do the following:

- Create new charts from scratch
- Package charts into chart archive (tgz) files
- Interact with chart repositories where charts are stored
- Install and uninstall charts into an existing Kubernetes cluster
- Manage the release cycle of charts that have been installed with Helm

For Helm, there are three important concepts:

- The chart is a bundle of information necessary to create an instance of a Kubernetes application.
- The config contains configuration information that can be merged into a packaged chart to create a releasable object.
- A release is a running instance of a chart, combined with a specific config.

Components

Helm CLI

- Locally installed client tool (like kubectl)
- Allows us to communicate easier with Tiller service
- Create our own charts
- Managing repositories
- Working with deployments

Tiller

- Helm service running in Kubernetes cluster
- Installation + configuration of applications via charts
- Upgrading, uninstalling deployments

Install Helm cli first

Windows

- Easiest is via Chocolatey
- `choco install kubernetes-helm`

Mac

- `brew install kubernetes-helm`

Getting started: helm init (v2!)

```
multipass@microk8s-vm:~$ sudo helm init
$HELM_HOME has been configured at /home/multipass/.helm.

Tiller (the Helm server-side component) has been installed into your Kubernetes Cluster.

Please note: by default, Tiller is deployed with an insecure 'allow unauthenticated users' policy.
To prevent this, run `helm init` with the --tiller-tls-verify flag.
For more information on securing your installation see: https://docs.helm.sh/using_helm/#securing-your-helm-installation
```

kubectl get pods --namespace kube-system

```
PS >kubectl get pods --namespace kube-system
NAME                                          READY   STATUS    RESTARTS   AGE
coredns-f7867546d-v4bjs                       1/1     Running   4          5d8h
heapster-v1.5.2-844b564688-mcjgd              4/4     Running   97         5d8h
hostpath-provisioner-65cfd8595b-hjqqk         1/1     Running   1          2d9h
kubernetes-dashboard-7ddbdf6949-mlj8w         1/1     Running   0          40h
monitoring-influxdb-grafana-v4-6b6954958c-mlv7t   2/2     Running   11         5d8h
tiller-deploy-75f6c87b87-mnl76                1/1     Running   0          3m31s
```

Helm init (v2)

If we are running Helm inside a RBAC enabled cluster we need to setup something more secure
More info on locking down access:
https://helm.sh/docs/using_helm/#tiller-and-role-based-access-control

Setup a service account first
(Next slide will show the manifest)

Run init with this account:

```
helm init --service-account tiller
```

If Helm init failed, you might need to uninstall

```
helm reset --force
```

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: tiller
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: tiller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: tiller
    namespace: kube-system
```

Example install

Run:

```
helm install stable/wordpress --name handsonakswp
```

If you get an error, refresh your repo:

Helm repo update

This will automatically install wordpress including persistent storage, database, service, loadbalancer config
(ps loadbalancer works only in cloud, replace by ingress on microk8s)

Looking at our installation

kubectl get services

```
PS >kubectl get services
NAME                    TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)                      AGE
handsonakswp-mariadb    ClusterIP      10.0.131.156   <none>          3306/TCP                     4m
handsonakswp-wordpress  LoadBalancer   10.0.181.8     40.118.88.179   80:30565/TCP,443:31190/TCP   3m59s
kubernetes              ClusterIP      10.0.0.1       <none>          443/TCP                      17h
```

Get the password for user ‘user’

```
$(kubectl get secret --namespace default handsonakswp-wordpress
-o jsonpath="{.data.wordpress-password}" | base64 --decode)
```

Chart file structure

```
wordpress/
  Chart.yaml          # A YAML file containing information about the chart
  LICENSE             # OPTIONAL: A plain text file containing the license for the chart
  README.md           # OPTIONAL: A human-readable README file
  requirements.yaml   # OPTIONAL: A YAML file listing dependencies for the chart
  values.yaml         # The default configuration values for this chart
  charts/             # A directory containing any charts upon which this chart depends.
  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
```

Chart.yaml structure

```
apiVersion: The chart API version, always "v1" (required)
name: The name of the chart (required)
version: A SemVer 2 version (required)
kubeVersion: A SemVer range of compatible Kubernetes versions (optional)
description: A single-sentence description of this project (optional)
keywords:
  - A list of keywords about this project (optional)
home: The URL of this project's home page (optional)
sources:
  - A list of URLs to source code for this project (optional)
maintainers: # (optional)
  - name: The maintainer's name (required for each maintainer)
    email: The maintainer's email (optional for each maintainer)
    url: A URL for the maintainer (optional for each maintainer)
engine: gotpl # The name of the template engine (optional, defaults to gotpl)
icon: A URL to an SVG or PNG image to be used as an icon (optional).
appVersion: The version of the app that this contains (optional). This needn't be SemVer.
tillerVersion: ">2.0.0" (optional)
```

Dependencies

Either deployed in Charts folder

Or in requirements.yaml

```
dependencies:
  - name: apache
    version: 1.2.3
    repository: http://example.com/charts
  - name: mysql
    version: 3.2.1
    repository: http://another.example.com/charts
```

Run to download into charts: `helm dependency update`

Templates

Templates are used to generate manifest files

Can use variables either through installation or from defaults file

```
#values.yaml
title: "My WordPress Site" # Sent to the WordPress template

mysql:
  max_connections: 100 # Sent to MySQL
  password: "secret"

apache:
  port: 8080 # Passed to Apache
```

Templates

To create the template structure

```
helm create <chart-name>
```

Build the package:

```
Helm package <chart-name>
```

Kustomize

Kustomize

Kustomize let’s your create Helm-like installations, but in a simpler way

If you have complex requirements probably Helm will suit you better

Kustomize helps you fine-tune and patch existing yaml configs

Install via https://kustomize.io/

Or

Use `kubectl apply -k`

Create kustomization.yaml

```
~/someApp
├── deployment.yaml
├── kustomization.yaml
└── service.yaml
```

```
kustomize build ~/someApp | kubectl apply -f -
```

Create overlays

```
~/someApp
├── base
│   ├── deployment.yaml
│   ├── kustomization.yaml
│   └── service.yaml
└── overlays
    ├── development
    │   ├── cpu_count.yaml
    │   ├── kustomization.yaml
    │   └── replica_count.yaml
    └── production
        ├── cpu_count.yaml
        ├── kustomization.yaml
        └── replica_count.yaml
```

```
kustomize build ~/someApp/overlays/production | /
  kubectl apply -f –
```

```
kubectl apply -k ~/someApp/overlays/production
```