Creating Helm Charts

Introduction:

There are some instances in which a chart for your application might not exist and one must be created; in other cases, you might have a Kubernetes deployment you want to convert into a Helm chart. In this hands-on lab, we will take an existing deployment and convert it into a Helm chart.

Convert the Service Manifest into a Service Template in a New Helm Chart

1. On the first console, create a blob directory called blog:
   mkdir blog

2. Access the directory:
   cd blog

Convert the Service Manifest into a Service Template in a New Helm Chart

1. On the first console, create a `blob` directory called `blog`:
2. `mkdir blog`
3. Access the directory:
4. `cd blog`
5. Run the `touch` and `mkdir` commands to create the minimum necessary scaffolding for the new chart. This includes the `Chart.yaml` and `values.yaml` files as well as the `templates` directory:
6. `touch Chart.yaml`
7. `touch values.yaml`
8. `mkdir templates`
9. View the chart details:
10. `ls -l`
11. Create the `Chart.yaml` file:
12. `vim Chart.yaml`
13. Add the `apiVersion`, `name`, and `version` to the file (this is the minimum data required for *Chart.yaml*):

```
apiVersion: v1
name: blog
```

14. `version: 0.1.0`
15. Save and exit the file.
16. Create the `values.yaml` file:

17. `vim values.yaml`
18. On the second console, view the home directory, which contains a *kubernetes* directory:
19. `ls`
20. Run the `cd` and `ls` commands to open and view the *kubernetes* directory. The directory contains an *application.yaml* file and a *service.yaml* file:
21. `$ cd kubernetes/`
22. `ls`
23. View *service.yaml*:
24. `vim service.yaml`
25. On the first console, use the data from *service.yaml* to add data to *values.yaml*. Update *nodePort* to `30080`:

```
service:
 name: blog
 type: NodePort
 port: 80
 targetPort: 2368
```

26. `nodePort: 30080`
27. On the first console, save *values.yaml*.
28. On the second console, exit out of *service.yaml*.
29. On the second console, run the `cd` command to open the *blog* folder and run the `vim` command to view the *values.yaml* file:
30. `cd ../blog`
31. `vim ./values.yaml`
32. On the first console, open the *templates* directory:
33. `cd templates/`
34. Copy the *service.yaml* file into the *blog* folder's *templates* directory:
35. `cp ~/kubernetes/service.yaml ./`
36. Run the `ls` and `vim` commands to view *service.yaml*.
37. `ls service.yaml`
38. `vim service.yaml`
39. Use the *values.yaml* data on the second console to make *service.yaml* a template on the first console. To do this, update the *service.yaml* file values as follows:

```
apiVersion: v1
kind: Service
```

```
metadata:
    name: {{ .Values.service.name }}
spec:
    type: {{ .Values.service.type }}
    selector:
        app: {{ .Values.service.name }}
    ports:
    -   protocol: TCP
        port: {{ .Values.service.port }}
        targetPort: {{ .Values.service.targetPort }}
        nodePort: {{ .Values.service.nodePort }}
```

41. On the first console, save the template to return to the *templates* directory.
42. On the second console, exit out of *values.yaml* to return to the *blog* folder.
43. On the first console, run the `cd` and `helm show values` commands to view the blog details. At this point, we have a full Helm chart:
44. `cd ~/`
45. `helm show values blog`
46. Verify the manifest's syntax is correct:
47. `helm install demo blog --dry-run`
48. On the second console, run the `cd` and `cat` commands so you can compare the two *service.yaml* files.
49. `cd ../`
50. `cat ./kubernetes/service.yaml`
51. Confirm the *service.yaml* data matches on the first and second consoles, with the exception of the *nodePort* value.
52. After reviewing the *service.yaml* data, `clear` both consoles.
53. `clear`

Convert the Manifest for the Application into a Deployment Template in a New Helm Chart

1. On the second console, view the *application.yaml* file:
2. `vim ./kubernetes/application.yaml`
3. On the first console, view the *blog* folder's *values.yaml* file:
4. `vim ./blog/values.yaml`
5. Below the existing file data, create a new *blog* section by inserting the following values. You can copy these values from the second console:

```
blog:
```

```
  name: blog
  replicas: 1
  image: ghost:2.6-alpine
  imagePullPolicy: Always
```

6. `  containerPort: 2368`
7. On the second console, exit out of the chart.
8. On the first console, save the *values* file.
9. On the second console, view the *blog* folder's *values.yaml* file:
10. `vim ./blog/values.yaml`
11. On the first console, copy the *kubernetes* folder's *application.yaml* file into the *blog* folder's *templates* directory:
12. `cp ./kubernetes/application.yaml ./blog/templates/`
13. View the *application.yaml* file in the *blog* folder's *templates* directory:
14. `vim ./blog/templates/application.yaml`
15. Make *application.yaml* a template by updating the file values as follows:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Values.blog.name }}
  labels:
    app: {{ .Values.blog.name}}
spec:
  replicas: {{ .Values.blog.replicas }}
  selector:
    matchLabels:
      app: {{ .Values.blog.name }}
  template:
    metadata:
      labels:
        app: {{ .Values.blog.name }}
    spec:
      containers:
      - name: {{ .Values.blog.name }}
        image: {{ .Values.blog.image }}
        imagePullPolicy: {{ .Values.blog.imagePullPolicy }}
        ports:
```

16. `        - containerPort: {{ .Values.blog.containerPort }}`

17. After updating the file values, save the template and `clear` the console:
18. `clear`
19. On the second console, exit out of the *values.yaml* file.

Ensure the Manifests Render Correctly and Deploy the Application as a NodePort Application

1. On the first console, run the `helm show values` command to view the *blog* folder's details:
2. `helm show values blog`
3. Run the `helm install` command with the `--dry-run` directive. The manifest should display with the service set to run as a *NodePort* on port *30080* (in the lab video, this step produced an error message because there was a typo in the *application.yaml* file):
4. `helm install demo blog --dry-run`
5. Install and deploy Helm:
6. `helm install demo blog`
7. View the pod details (note that the pod's status is *ContainerCreating*):
8. `kubectl get po`
9. While the container is being created, view the service details (note that the blog service is running on the correct *NodePort* of *30080*):
10. `kubectl get svc`
11. Verify the pod's status is now *Running*:
12. `kubectl get po`
13. On the second console, `exit` out of session so you can view the public IP address for the Kubernetes primary server:
14. `exit`
15. Copy the public IP address of the Kubernetes primary server and paste it into a new browser tab along with the port number: `<PUBLIC_IP_ADDRESS>:30080`. The ghost blog should load: