Lab 5.1 - Using Helm to Manage the Lifecycle of a Sample Python Application

In this lab we will learn how to manage the lifecycle of a simple Python (**`python3`**) web application on Kubernetes using Helm.

We will start by explaining the source code of the app and what it does. We will then briefly walk through the steps of publishing a container image to a registry so that Kubernetes can fetch our application and run it as a containerized process.

Afterwards, we will start building a Helm chart for this application from scratch, walking through the various files in the chart and how everything works together to manage this application on Kubernetes.

Lastly, we will run various Helm commands to go step-by-step through the full lifecycle of this application.

Setup

This lab should be supplemented by a .zip file containing all of the files referenced in this lab (go to Chapter 1: Introduction > Course Information > Course Resources). Copy this .zip file onto your machine and extract it, then enter the **`workspace/`** directory:

Step1

```
$ unzip ch5_workspace.zip
Archive:  ch5_workspace.zip
   creating: workspace/
```



```
  inflating: workspace/server.py
  inflating: workspace/index.html
  inflating: workspace/Dockerfile
   creating: workspace/quotegen/
  inflating: workspace/quotegen/Chart.yaml
   creating: workspace/quotegen/templates/
  inflating: workspace/quotegen/templates/pre-delete-hook.yaml
  inflating: workspace/quotegen/templates/deployment.yaml
  inflating: workspace/quotegen/templates/service.yaml
  inflating: workspace/quotegen/templates/_helpers.tpl
  inflating: workspace/quotegen/templates/api-test.yaml
  inflating: workspace/quotegen/values.yaml
```

```
$ cd workspace/
```

Note: The filename may be slightly different from “ch5_workspace.zip”. Modify the “unzip” to reference the correct filename.

Step 2

The **docker** command is required to build container images. If running on Ubuntu, use the following command to install Docker:

```
$ sudo snap install docker
docker 19.03.11 from Canonical✓ installed
```

Run the following command to verify that Docker is installed:

```
$ sudo docker info | grep "Server Version"
 Server Version: 19.03.11
```

Step 3

Lastly, we must have access to a container registry to push images to. Using MicroK8s, you can start a local registry at **localhost:32000** by running the following command:

```
$ microk8s enable registry
Enabling the private registry
storage is already enabled
Applying registry manifest
namespace/container-registry created
persistentvolumeclaim/registry-claim created
```



```
deployment.apps/registry created
service/registry created
The registry is enabled
```

You can also use public registries such as Docker Hub, Quay, and others if you have access to that.

The Application

The application itself is a random quote generator called “quotegen”. When a user visits the web application in a browser, they will be met with a thoughtful phrase to ponder upon.

This web app will be comprised of two primary files:

- `index.html`: A static HTML file which requests new quotes.
- `server.py`: A backend web server written in Python that returns great quotes.

Step 4

Here is the source code for `index.html`:

```
<html>
   <head>
       <title>quotegen - worlds first quote generator</title>
       <style>
           body {
               background: black;
           }
           #main {
               color: white;
               text-align: center;
               font-weight: bold;
               font-family: monospace;
               padding-top: 60px;
           }
       </style>
   </head>
   <body>
       <div id="main"></div>
       <script>
           document.addEventListener('DOMContentLoaded', function() {
```



```
                var xhr = new XMLHttpRequest();
                xhr.onreadystatechange = function() {
                    if (xhr.readyState === 4){
                        var data = JSON.parse(xhr.responseText);
                        var elem = document.getElementById('main');
                        elem.style.fontSize = data['size']+'px';
                        elem.innerHTML = data['text'];
                    }
                };
                xhr.open('GET', '/quote');
                xhr.send();
            });
        </script>
    </body>
</html>
```

When this page is loaded in the browser, using JavaScript, we make an AJAX request to an API endpoint to retrieve a JSON payload containing a new quote. Then, the quote is extracted from the JSON body, and displayed on the page for the user.

Step 5

Here is the source code for **server.py**:

```
import os
import json
import random
import http.server
import socketserver

FONT_SIZE = os.environ.get('FONT_SIZE', '20')
RAW_INDEX_CONTENTS = bytes(open('./index.html').read(), 'UTF-8')
QUOTES = [
    'Live, laugh, love.',
    'Keep calm and carry on.',
    'Do what you feel in your heart to be right - for you'll be
criticized anyway.',
    'You may not control all the events that happen to you, but you can
decide not to be reduced by them',
    'Truth is, everybody is going to hurt you; you just gotta find the
ones worth suffering for'
    'If life gives you lemons, make lemonade',
    'You miss 100 percent of the shots you dont take',
```



```
    'Be the change you wish to see in the world',
    'Be kind, for everyone you meet is fighting a hard battle.',
    'Learning to unlearn is the highest form of learning.'
]

def get_quote():
    return {'size': FONT_SIZE, 'text': random.choice(QUOTES)}

class Server(http.server.SimpleHTTPRequestHandler):
    def do_GET(self):
        path = self.path
        if path == '/' or path == '/index.html':
            self.send_response(200, 'OK')
            self.send_header('Content-type', 'text/html')
            self.end_headers()
            self.wfile.write(RAW_INDEX_CONTENTS)
        elif path == '/quote':
            quote = get_quote()
            self.send_response(200, 'OK')
            self.send_header('Content-type', 'application/json')
            self.end_headers()
            self.wfile.write(bytes(json.dumps(quote), 'UTF-8'))
        else:
            self.send_response(404, 'NOT FOUND')

if __name__ == '__main__':
    print('Starting server...')
    socketserver.TCPServer(('0.0.0.0', 8080), Server).serve_forever()
```

This source file starts a web server on port 8080. When it receives an HTTP request for the path `/`, it will simply return the contents of `index.html`. If it receives an HTTP request for the path `/quote`, it will determine a random quote and write it back to the client encoded as JSON.

Containerizing and Publishing the Application with Docker

In order for Kubernetes to run our application as a container, it must first be published to a container registry which Kubernetes has access to.

Once we have Docker up-and-running, we will create a simple `Dockerfile` which will provide instructions for how to package our app as a container image.

Step 6

Here are the contents of **Dockerfile**:

```
FROM python:3-alpine
WORKDIR /app
COPY server.py .
COPY index.html .
EXPOSE 8080
CMD ["python3", "server.py"]
```

This file should be placed directly next to **index.html** and **app.py**.

Step 7

Using Docker, build the image with the reference **localhost:32000/quotes:v1**:

```
$ sudo docker build . -t localhost:32000/quotes:v1
Sending build context to Docker daemon  6.656kB
Step 1/6 : FROM python:3-alpine
3-alpine: Pulling from library/python
df20fa9351a1: Pull complete
36b3adc4ff6f: Pull complete
7031d6d6c7f1: Pull complete
81b7f5a7444b: Pull complete
0f8a54c5d7c7: Pull complete
Digest:
sha256:c5623df482648cacece4f9652a0ae04b51576c93773ccd43ad459e2a195906d
d
Status: Downloaded newer image for python:3-alpine
 ---> 8ecf5a48c789
Step 2/6 : WORKDIR /app
 ---> Running in 525fe7b000f8
Removing intermediate container 525fe7b000f8
 ---> e20e2c3b744e
Step 3/6 : COPY server.py .
 ---> 0e5415a83174
Step 4/6 : COPY index.html .
 ---> 48e60ea78545
Step 5/6 : EXPOSE 8080
 ---> Running in 269551c36b5f
```



```
Removing intermediate container 269551c36b5f
 ---> 24d4dc5c0d1f
Step 6/6 : CMD ["python3", "server.py"]
 ---> Running in 82f756418802
Removing intermediate container 82f756418802
 ---> 059e859346fd
Successfully built 059e859346fd
Successfully tagged localhost:32000/quotes:v1
```

Step 8

You could test to see if your application runs properly:

```
$ sudo docker run -it --rm localhost:32000/quotes:v1
Starting server...
```

(press CTRL+C to exit the running container)

Step 9

Finally, publish the container image to the registry:

```
$ sudo docker push localhost:32000/quotes:v1
The push refers to repository [localhost:32000/quotes]
5dd75bb60e7f: Pushed
008a5aac9e66: Pushed
da5caef63bd4: Pushed
fffdb84c36f2: Layer already exists
50205a7df19a: Layer already exists
ef833453b9c7: Layer already exists
408e53c5e3b2: Layer already exists
50644c29ef5a: Layer already exists
v1: digest:
sha256:76166f6332d33581fec218ea8fe58c6650d3daf2a705775cf9f6544ff46b03d
9 size: 1989
```

The Helm Chart for the Application

Now that we have a container image, we need to create a Kubernetes Deployment object which references this image for the current version of our application.

The Helm chart for the application will generate all of the Kubernetes resources necessary for

running this application, which will allow us to dynamically set not only the version of the container image to run, but also various other things such as the container's table of environment variables.

Step 10

The Helm chart for our application is contained in the **quotegen** directory. Enter this directory and examine the files:

```
$ cd quotegen/
```

Step 11

Checkout the **Chart.yaml** for this chart:

```
$ cat Chart.yaml
name: quotegen
version: 0.1.0
description: worlds first quote generator
```

*Note: The **name:** field of **Chart.yaml** must always match the directory it is contained in.*

Next 12

Next, inspect **values.yaml** containing the default configuration settings for our chart:

```
$ cat values.yaml
image:
  repo: localhost:32000/quotes
  tag: v1

env:
  FONT_SIZE: 20

replicas: 2

service:
  type: NodePort
  port: 30001
```

Step 13

Next, let's create the chart's **templates/** directory:


```
$ mkdir -p templates/
```

Inside the **templates/** directory, we will have five files:

- **_helpers.tpl**: A file with a few partial definitions.
- **deployment.yaml**: A template for a Deployment resource.
- **service.yaml**: A template for a Service resource.
- **api-test.yaml**: A template for a Job to run to test API connectivity.
- **pre-delete-hook.yaml**: A template for a Job to run on the "pre-delete" event.

Take some time to check out the source code for each of these files within the **templates/** directory.

Managing the Application's Lifecycle Using Helm Commands

Step 14

Re-enter the top-level directory for this lab:

```
$ cd ../ # workspace
```

Step 15

We should now have a directory layout that looks like the following:

```
$ tree .
.
├── Dockerfile
├── index.html
├── quotegen
│   ├── Chart.yaml
│   ├── _helpers.tpl
│   ├── api-test.yaml
│   ├── pre-delete-hook.yaml
│   ├── templates
│   │   ├── _helpers.tpl
│   │   ├── api-test.yaml
│   │   ├── deployment.yaml
│   │   ├── pre-delete-hook.yaml
│   │   └── service.yaml
│   └── values.yaml
└── server.py
```



```
2 directories, 13 files
```

Step 16

Now we enter the very first stage in the lifecycle of this application: its first installation. Use Helm to install the `quotegen` chart, creating an initial release:

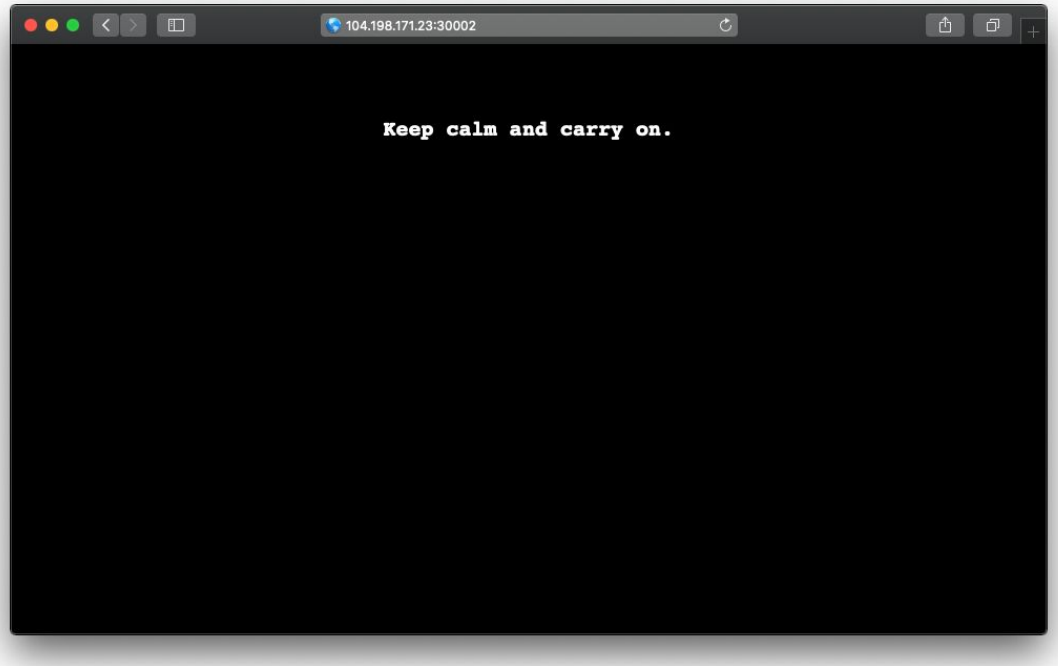
```
$ helm install quotegen-prod quotegen/
NAME: quotegen-prod
LAST DEPLOYED: Wed Jun 17 10:08:13 2020
NAMESPACE: default
STATUS: deployed
REVISION: 1
```

Step 17

Next, let's verify that the application is working properly by executing our test Job:

```
$ helm test quotegen-prod --timeout=30s
NAME: quotegen-prod
LAST DEPLOYED: Wed Jun 17 10:08:13 2020
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE:     quotegen-prod-api-test
Last Started:   Wed Jun 17 10:08:24 2020
Last Completed: Wed Jun 17 10:08:26 2020
Phase:          Succeeded
```

Looks like things appear to be working correctly. The final step is to do some manual quality assurance. Load the app in your web browser and refresh the page a couple times to see a few different quotes:



Note: Keep in mind that in order to access port 30001 on this server, you must open up the firewall to allow inbound access from your IP address to port 30001 as we went over in a Lab 4.1.

Let's say our users aren't happy with font size on the screen. They want it to be bigger (much bigger!). This is something we can easily modify with one of the nested configuration values we have in our chart, **`env.FONT_SIZE`**.

Step 18

Let's upgrade our application to set **`env.FONT_SIZE`** to **80** (for 80 pixels):

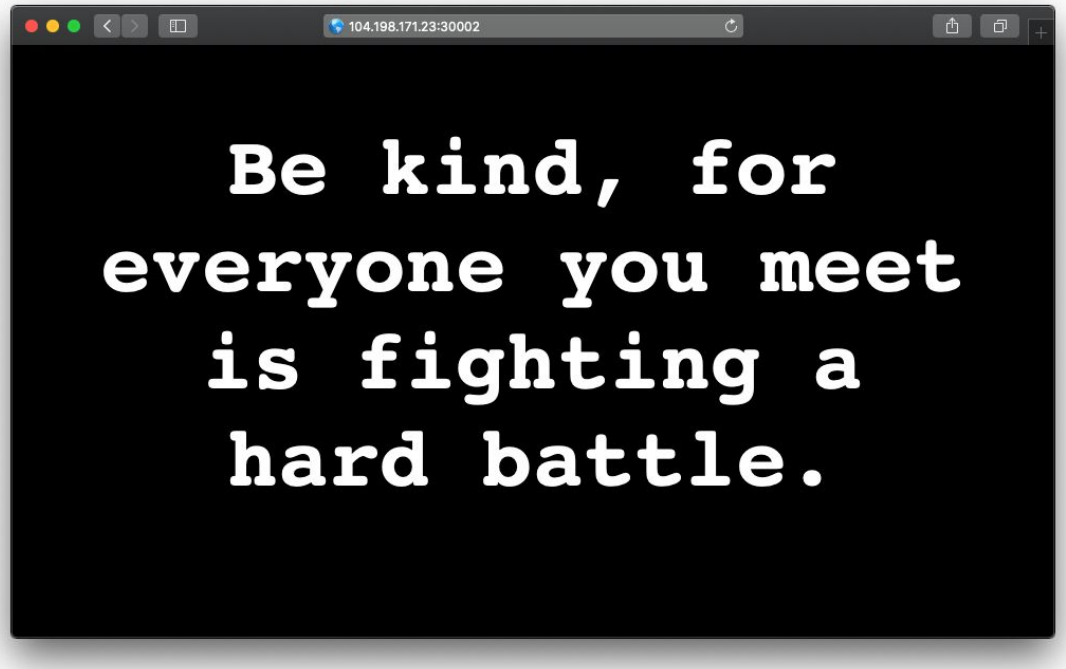
```
$ helm upgrade quotegen-prod quotegen/ --set env.FONT_SIZE=80
Release "quotegen-prod" has been upgraded. Happy Helming!
NAME: quotegen-prod
LAST DEPLOYED: Wed Jun 17 10:09:12 2020
NAMESPACE: default
STATUS: deployed
REVISION: 2
```


Step 19

This will create a second release revision. We can view the current revision using helm status:

```
$ helm status quotegen-prod
NAME: quotegen-prod
LAST DEPLOYED: Wed Jun 17 10:09:12 2020
NAMESPACE: default
STATUS: deployed
REVISION: 2
```

If you navigate to the application in the browser once again, you should now see bigger, better quotes on the screen:



Your boss isn't happy with the quote content. They say that the quotes aren't thought-provoking enough. The quotes need to really inspire people.

Step 20

A change like this will require us to make an update to the Python code. Here's a new version of

server.py with a modified **get_quote()** function:

```
import os
import json
import random
import http.server
import socketserver

FONT_SIZE = os.environ.get('FONT_SIZE', '20')
RAW_INDEX_CONTENTS = bytes(open('./index.html').read(), 'UTF-8')
QUOTES = [
   'Live, laugh, love.',
   'Keep calm and carry on.',
   'Do what you feel in your heart to be right - for you'll be
criticized anyway.',
   'You may not control all the events that happen to you, but you can
decide not to be reduced by them',
   'Truth is, everybody is going to hurt you; you just gotta find the
ones worth suffering for'
   'If life gives you lemons, make lemonade',
   'You miss 100 percent of the shots you dont take',
   'Be the change you wish to see in the world',
   'Be kind, for everyone you meet is fighting a hard battle.',
   'Learning to unlearn is the highest form of learning.'
]

def get_quote():
   text = 'Be very inspired: ' + random.choice(QUOTES)
   return {'size': FONT_SIZE, 'text': QUOTES[0]}


class Server(http.server.SimpleHTTPRequestHandler):
   def do_GET(self):
       path = self.path
       if path == '/' or path == '/index.html':
           self.send_response(200, 'OK')
           self.send_header('Content-type', 'text/html')
           self.end_headers()
           self.wfile.write(RAW_INDEX_CONTENTS)
       elif path == '/quote':
           quote = get_quote()
           self.send_response(200, 'OK')
```



```
            self.send_header('Content-type', 'application/json')
            self.end_headers()
            self.wfile.write(bytes(json.dumps(quote), 'UTF-8'))
        else:
            self.send_response(404, 'NOT FOUND')

if __name__ == '__main__':
   print('Starting server...')
   socketserver.TCPServer(('0.0.0.0', 8080), Server).serve_forever()
```

Step 21

Build and push a new container image with these changes with a new tag, **v2**:

```
$ sudo docker build . -t localhost:32000/quotes:v2
Sending build context to Docker daemon  334.8kB
Step 1/6 : FROM python:3-alpine
 ---> 8ecf5a48c789
Step 2/6 : WORKDIR /app
 ---> Using cache
 ---> e20e2c3b744e
Step 3/6 : COPY server.py .
 ---> 976b42c82d6b
Step 4/6 : COPY index.html .
 ---> 9ec306a9828a
Step 5/6 : EXPOSE 8080
 ---> Running in 658fa87aa621
Removing intermediate container 658fa87aa621
 ---> b8b33d7f06b4
Step 6/6 : CMD ["python3", "server.py"]
 ---> Running in 9676d51b9cc6
Removing intermediate container 9676d51b9cc6
 ---> 13c0f11f0469
Successfully built 13c0f11f0469
Successfully tagged localhost:32000/quotes:v2

$ sudo docker push localhost:32000/quotes:v2
The push refers to repository [localhost:32000/quotes]
ab4875387f73: Pushed
761063c46537: Pushed
da5caef63bd4: Layer already exists
fffdb84c36f2: Layer already exists
```



```
50205a7df19a: Layer already exists
ef833453b9c7: Layer already exists
408e53c5e3b2: Layer already exists
50644c29ef5a: Layer already exists
v2: digest:
sha256:e1ec54d6d4ab83062d98da089a0e046df6c45be1778a87c7fc72e25a4fdea68
7 size: 1989
```

Step 22

Now let's upgrade our application to use this new container image:

```
$ helm upgrade quotegen-prod quotegen/ --reuse-values --set
image.tag=v2
Release "quotegen-prod" has been upgraded. Happy Helming!
NAME: quotegen-prod
LAST DEPLOYED: Wed Jun 17 10:21:32 2020
NAMESPACE: default
STATUS: deployed
REVISION: 3
```

*Note: The **`--reuse-values`** flag used here allows us to merge into the existing set of values, so we do not accidentally reset **`env.FONT_SIZE`**.*

Step 23

Next, let's verify that the application is working properly by executing our test Job:

```
$ helm test quotegen-prod --timeout=30s
NAME: quotegen-prod
LAST DEPLOYED: Wed Jun 17 10:21:32 2020
NAMESPACE: default
STATUS: deployed
REVISION: 3
TEST SUITE:     quotegen-prod-api-test
Last Started:   Wed Jun 17 10:23:08 2020
Last Completed: Wed Jun 17 10:23:18 2020
Phase:          Failed
Error: timed out waiting for the condition
```

Oh no! The test failed! Looks like the same terrible quote is being sent back by the API over and over.

Step 24

Before we try to determine what exactly went wrong with the changes, let's quickly rollback the changes to get to the previous version of the application:

```
$ helm rollback quotegen-prod 2
Rollback was a success! Happy Helming!
```

Step 25

Let's try testing again to make sure things are back to normal:

```
$ helm test quotegen-prod --timeout=30s
NAME: quotegen-prod
LAST DEPLOYED: Wed Jun 17 10:23:54 2020
NAMESPACE: default
STATUS: deployed
REVISION: 4
TEST SUITE:     quotegen-prod-api-test
Last Started:   Wed Jun 17 10:24:25 2020
Last Completed: Wed Jun 17 10:24:27 2020
Phase:          Succeeded
```

Looks like we're in the clear. Phew, that was close. Thankfully Helm came to the rescue there.

Step 26

After inspecting our Python code we find the root cause, a bad line in our **`get_quote()` function**:

```
def get_quote():
   text = 'Be very inspired: ' + random.choice(QUOTES)
   return {'size': FONT_SIZE, 'text': QUOTES[0]}
```

We call your boss to deliver the disappointing news about the failed feature. Before we're able to explain, they tell us the entire project has been cancelled. It turns out the competition launched a bulletproof quote generator just last week. Your company needs to rethink its strategy, and frankly, its entire existence.

So you're instructed to permanently take down the **`quotegen`** app. You poured the last 8 months of your life into this project, but whatever. You suppose you must do what is best for the company.

Step 27

Let's uninstall the application using **`helm delete`**:

```
$ helm delete quotegen-prod
release "quotegen-prod" uninstalled
```

Step 28

Goodbye autogen... But wait! You recall adding a hook to the Helm chart to generate one last quote. A final farewell, per se:

```
$ kubectl get pods | grep pre-delete-hook
quotegen-prod-pre-delete-hook-dm9rr   0/1     Completed     0
35s

$ kubectl logs quotegen-prod-pre-delete-hook-dm9rr
{"size": "20", "text": "Be the change you wish to see in the world"}
```

