Lab 6.1 - Hosting a Simple Chart Repository with Python

The Repository Index (index.yaml)

At the core of each chart repository is a file called `index.yaml`. This is known as the **repository index**. This file contains a list of all available charts in the repository, and all of their versions. This file should be accessible at the relative root of the chart repo URL.

Step 1

Let's use an example from Chapter 4, the public chart repository provided by Bitnami. You may recall you ran the following command to add the repository:

```
$ helm repo add bitnami https://charts.bitnami.com/bitnami
"bitnami" has been added to your repositories
```

Behind the scenes, Helm is fetching the repository index from **https://charts.bitnami.com/bitnami/index.yaml**.

Step 2

Let's see what this file looks like if we download it directly:

```
$ curl -s https://charts.bitnami.com/bitnami/index.yaml | less
apiVersion: v1
entries:
  airflow:
```

```
  - apiVersion: v1
    appVersion: 1.10.10
    created: "2020-07-02T07:45:13.7728364Z"
    description: Apache Airflow is a platform to programmatically
author, schedule
      and monitor workflows.
    digest:
d72687d8ca8f85411d7273ad64cee41afb19b007dae0acf1c0d50e96333be2e5
    engine: gotpl
    home: https://airflow.apache.org/
    icon:
https://bitnami.com/assets/stacks/airflow/img/airflow-stack-110x117.pn
g
    keywords:
    - apache
    - airflow
    - workflow
    - dag
    maintainers:
    - email: containers@bitnami.com
      name: Bitnami
    name: airflow
    sources:
    - https://github.com/bitnami/bitnami-docker-airflow
    urls:
    - https://charts.bitnami.com/bitnami/airflow-6.3.5.tgz
    version: 6.3.5
  - apiVersion: v1
    appVersion: 1.10.10
    created: "2020-06-28T21:39:48.486454019Z"
    description: Apache Airflow is a platform to programmatically
author, schedule
      and monitor workflows.
    digest:
508f6c22d45963bdcdfbfb26c1b2b1ae5eab2d17522bfccf6f241aabc226bde1
    engine: gotpl
    home: https://airflow.apache.org/
    icon:
https://bitnami.com/assets/stacks/airflow/img/airflow-stack-110x117.pn
g
    keywords:
```

```
    - apache
    - airflow
    - workflow
    - dag
    maintainers:
    - email: containers@bitnami.com
      name: Bitnami
    name: airflow
    sources:
    - https://github.com/bitnami/bitnami-docker-airflow
    urls:
    - https://charts.bitnami.com/bitnami/airflow-6.3.4.tgz
    version: 6.3.4
…
generated: "2020-07-02T08:39:50.316281483Z"
```

The whole file is over 4 MB, so it has been shortened for this example. The most important part of this file is the **`entries`** section. This is a map which maps unique chart names in this repository to a list of their available versions. For example, the example above shows two (2) versions of the “airflow” chart. Let’s look as one of these versions for the “airflow” chart:

```
  - apiVersion: v1
    appVersion: 1.10.10
    created: "2020-07-02T07:45:13.7728364Z"
    description: Apache Airflow is a platform to programmatically
author, schedule
      and monitor workflows.
    digest:
d72687d8ca8f85411d7273ad64cee41afb19b007dae0acf1c0d50e96333be2e5
    engine: gotpl
    home: https://airflow.apache.org/
    icon:
https://bitnami.com/assets/stacks/airflow/img/airflow-stack-110x117.pn
g
    keywords:
    - apache
    - airflow
    - workflow
    - dag
    maintainers:
    - email: containers@bitnami.com
```

```
      name: Bitnami
    name: airflow
    sources:
    - https://github.com/bitnami/bitnami-docker-airflow
    urls:
    - https://charts.bitnami.com/bitnami/airflow-6.3.5.tgz
    version: 6.3.5
```

Does this look familiar? That's because the majority of this information is gathered from the chart's **Chart.yaml** file when the index is created. One section in each entry that is unique to repository indexes is the `urls` section. This contains a list of available places to download this specific chart version (usually just a single location is provided).

When you run `helm install bitnami/airflow`, Helm will first determine the latest semantic version of the chart based on the cached version of the index (`6.3.5`). After this, Helm will attempt to download the chart based on the contents of the url field of the entry (i.e. **https://charts.bitnami.com/bitnami/airflow-6.3.5.tgz**). After the chart has been downloaded, Helm then proceeds to install it.

Helm provides commands for building your own `index.yaml` which you can use to host a chart repository.

Step 3

First, let's create a test chart:

```
$ helm create mychart
Creating mychart
```

Step 4

Next, let's create a directory which will contain all of our chart tarballs:

```
$ mkdir -p lab/
```

Step 5

Now let's package the `mychart` chart into this directory:

```
$ helm package mychart/ -d lab/
Successfully packaged chart and saved it to: lab/mychart-0.1.0.tgz
```

Step 6

Next, all you need to do is run **`helm repo index`**, pointing to the directory containing this chart:

```
$ helm repo index lab/
```

Step 7

This should create a new **`index.yaml`** file inside the directory containing a single entry for the **`mychart`** chart:

```
$ cat lab/index.yaml
apiVersion: v1
entries:
  mychart:
  - apiVersion: v2
    appVersion: 1.16.0
    created: "2020-06-02T11:50:33.588597349Z"
    description: A Helm chart for Kubernetes
    digest:
64be7ad4e2ab08a57229c83f7f6b34f324a5a16305817d549052d20c69633ee0
    name: mychart
    type: application
    urls:
    - mychart-0.1.0.tgz
    version: 0.1.0
generated: "2020-06-02T11:50:33.58767312Z"
```

Updating the Index with New Chart Versions

As you iterate on your chart, you will want to make sure to regenerate the repository index to contain the latest versions.

Step 8

Let’s make a slight modification to the **`mychart`** chart, just bumping the version to **`0.2.0`**:

```
$ sed -i 's/^version:.*$/version: 0.2.0/' mychart/Chart.yaml
```

Step 9

Now, let’s create another tarball for this new chart version:

```
$ helm package mychart/ -d lab/
Successfully packaged chart and saved it to: lab/mychart-0.2.0.tgz
```

Step 10

Now, all we need to do to regenerate our index file is to re-run the **`helm repo index`** command:

```
$ helm repo index lab/
```

Step 11

You will see that our index file is now larger, with a new entry for the latest chart version:

```
$ cat lab/index.yaml
apiVersion: v1
entries:
  mychart:
  - apiVersion: v2
    appVersion: 1.16.0
    created: "2020-06-02T11:55:00.70151907Z"
    description: A Helm chart for Kubernetes
    digest:
2abb7e97678c65a5c671dc20f3c54d9e48834810304b77f77aa95d4d0bdace2b
    name: mychart
    type: application
    urls:
    - mychart-0.2.0.tgz
    version: 0.2.0
  - apiVersion: v2
    appVersion: 1.16.0
    created: "2020-06-02T11:55:00.700913049Z"
    description: A Helm chart for Kubernetes
    digest:
64be7ad4e2ab08a57229c83f7f6b34f324a5a16305817d549052d20c69633ee0
    name: mychart
    type: application
    urls:
    - mychart-0.1.0.tgz
    version: 0.1.0
generated: "2020-06-02T11:55:00.700108583Z"
```

In some instances you might not have access to all of the chart package (`.tgz`) files at the time which you are trying to generate the index.

For example, you might upload charts to your chart repository storage as part of a CI/CD pipeline. Instead of downloading every single chart version to generate the index, you can use the current version of the index and simply append to it. This can be achieved using the **`--merge`** flag.

Step 12

To demonstrate this, let's create a second chart called **`otherchart`**:

```
$ helm create otherchart
Creating otherchart
```

Step 13

Next, we will create a new workspace directory called `workspace` and package the new chart into it:

```
$ mkdir -p workspace/
$ helm package otherchart/ -d workspace/
Successfully packaged chart and saved it to:
workspace/otherchart-0.1.0.tgz
```

Step 14

Finally, run **`helm repo index`** using the **`--merge`** option to build a new repository index containing the new chart package based on the existing file:

```
$ helm repo index --merge lab/index.yaml workspace/
```

Step 15

Now if we inspect this new index, we will see all three (3) chart versions (**`mychart-0.1.0`**, **`mychart-0.2.0`**, and **`otherchart-0.1.0`**):

```
$ cat workspace/index.yaml
apiVersion: v1
entries:
  mychart:
  - apiVersion: v2
    appVersion: 1.16.0
    created: "2020-06-02T11:55:00.70151907Z"
    description: A Helm chart for Kubernetes
    digest:
2abb7e97678c65a5c671dc20f3c54d9e48834810304b77f77aa95d4d0bdace2b
    name: mychart
    type: application
    urls:
    - mychart-0.2.0.tgz
    version: 0.2.0
  - apiVersion: v2
    appVersion: 1.16.0
    created: "2020-06-02T11:55:00.700913049Z"
    description: A Helm chart for Kubernetes
    digest:
64be7ad4e2ab08a57229c83f7f6b34f324a5a16305817d549052d20c69633ee0
    name: mychart
    type: application
    urls:
    - mychart-0.1.0.tgz
    version: 0.1.0
  otherchart:
  - apiVersion: v2
    appVersion: 1.16.0
    created: "2020-06-02T12:13:06.685534862Z"
    description: A Helm chart for Kubernetes
    digest:
660492730dd4dc154dd44a7b4580abdd1d7f915bb2c04d13559f24716161a9b1
    name: otherchart
    type: application
    urls:
    - otherchart-0.1.0.tgz
    version: 0.1.0
generated: "2020-06-02T12:13:06.684391664Z"
```

Standing Up a Web Server with Python

The only thing left to do in hosting a chart repository is to leverage a web server to host `index.yaml` and the chart `.tgz` files.

In a production environment, you might use a performant web server such as Nginx: https://nginx.org/.

For the sake of this lab, we will simply use Python’s built-in, command-line web server (Python 3).

Step 16

Prior to starting the web server, make sure all files are in the proper location. Let’s move the `.tgz` packages for the **`mychart`** chart into the `workspace` directory which contains our most up-to-date index file:

```
$ mv lab/*.tgz workspace/
$ ls workspace/
index.yaml  mychart-0.1.0.tgz  mychart-0.2.0.tgz  otherchart-0.1.0.tgz
```

Next, open a new terminal window in the current directory.

Step 17

To start a Python web server to statically serve these files at http://localhost:8000 (http://127.0.0.1:8000), open a new terminal window to the current directory and run the following commands:

```
$ cd workspace/
$ python3 -m http.server --bind 127.0.0.1 8000
Serving HTTP on 127.0.0.1 port 8000 (http://127.0.0.1:8000/) ...
```

Note: If at any time you wish to stop the web server, just use CTRL+C.

Your chart repository is now ready to be used with Helm.

Using Your Chart Repository with Helm

Open the original terminal (the one not running your web server).

Now that our chart repository is up-and-running, we can start to use it from the Helm CLI.

Step 18

The first step is to add our chart repository as a new local repository with a unique identifier (**`lab`**):

```
$ helm repo add lab http://localhost:8000
"lab" has been added to your repositories
```

As we do this, in our other terminal running the Python web server, you should see the incoming request for **index.yaml** being logged:

```
127.0.0.1 - - [02/Jun/2020 12:30:03] "GET /index.yaml HTTP/1.1" 200 -
```

The repository index was downloaded by Helm and cached locally.

Step 19

Next, let's install one of our charts from the repo (**mychart**):

```
$ helm install lab-demo lab/mychart
NAME: lab-demo
LAST DEPLOYED: Thu Jul  2 12:34:41 2020
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l
"app.kubernetes.io/name=mychart,app.kubernetes.io/instance=lab-demo"
-o jsonpath="{.items[0].metadata.name}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl --namespace default port-forward $POD_NAME 8080:80
```

If we look again at the logs of our web server, we should see another incoming request, this time for **mychart-0.2.0.tgz**:

```
127.0.0.1 - - [02/Jun/2020 12:34:41] "GET /mychart-0.2.0.tgz HTTP/1.1"
200 -
```

Why did Helm choose to use version `0.2.0` of the **mychart** chart vs. version `0.1.0`? This is because Helm is able to compare semantic versions and automatically choose the latest chart if no specific version is specified.

Step 20

If we wanted to, we could deploy the old version of the **mychart** chart by specifying the **--version** flag:

```
$ helm install lab-demo-010 lab/mychart --version 0.1.0
NAME: lab-demo-010
```

```
LAST DEPLOYED: Thu Jul  2 12:39:05 2020
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l
"app.kubernetes.io/name=mychart,app.kubernetes.io/instance=lab-demo-01
0" -o jsonpath="{.items[0].metadata.name}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl --namespace default port-forward $POD_NAME 8080:80
```

This time we should see an incoming request in our web server logs for `mychart-0.1.0.tgz`:

```
127.0.0.1 - - [02/Jun/2020 12:39:05] "GET /mychart-0.1.0.tgz HTTP/1.1"
200 -
```

Next, let's see what happens when we update the repository index again.

Step 21

Make another modification to the `mychart` chart, bumping the version to `0.3.0`:

```
$ sed -i 's/^version:.*$/version: 0.3.0/' mychart/Chart.yaml
```

Step 22

Next, package it into the `workspace` directory and regenerate the index:

```
$ helm package mychart/ -d workspace/
Successfully packaged chart and saved it to:
workspace/mychart-0.3.0.tgz
$ helm repo index workspace/
```

Step 23

Now our repository index should contain a total of four (4) entries (`mychart-0.1.0`, `mychart-0.2.0`, `mychart-0.3.0`, and `otherchart-0.1.0`):

```
$ cat workspace/index.yaml | grep version: | wc -l
4
```

So our chart repository has an update available for us. Our Helm release we installed titled

lab-demo is based on version **0.2.0** of the **mychart** chart, but now there is a **0.3.0** available.

Step 24

Let's try to make the upgrade:

```
$ helm upgrade lab-demo lab/mychart --version 0.3.0
Error: failed to download "lab/mychart" (hint: running `helm repo
update` may help)
```

What went wrong? If we inspect the logs of our Python web server, we see no request for **mychart-0.3.0.tgz** as we might expect.

The problem is that we did not update the locally-cached version of the repository index. When we run **helm install**, Helm is determining available versions based on the contents of the **index.yaml** file that was downloaded when we ran **helm repo add**.

Step 25

If we want to update the local version of the repository index, we need to run **helm repo update**:

```
$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "lab" chart repository
Update Complete. ⎈ Happy Helming!⎈
```

Now if we look at the web server logs, we should see an additional request for **index.yaml**:

```
127.0.0.1 - - [02/Jun/2020 12:52:16] "GET /index.yaml HTTP/1.1" 200 -
```

Step 26

Now we can retry our original command to upgrade to version **0.3.0**:

```
$ helm upgrade lab-demo lab/mychart --version 0.3.0
Release "lab-demo" has been upgraded. Happy Helming!
NAME: lab-demo
LAST DEPLOYED: Thu Jun  2 12:53:25 2020
NAMESPACE: default
STATUS: deployed
REVISION: 2
```

```
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l
"app.kubernetes.io/name=mychart,app.kubernetes.io/instance=lab-demo"
-o jsonpath="{.items[0].metadata.name}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl --namespace default port-forward $POD_NAME 8080:80
```

Nice! We were able to download and install version `0.3.0` since we fetched the newer version of the index which contains it.

If we inspect out web server logs one last time, we should see the incoming request for:

```
127.0.0.1 - - [02/Jun/2020 12:53:25] "GET /mychart-0.3.0.tgz HTTP/1.1"
200
```

That's it! You should now have a thorough understanding of how to operate chart repositories.

Step 27

Lastly, let's do some cleanup. Exit the Python web server using CTRL+C, and delete the Helm releases we created:

```
$ helm delete lab-demo lab-demo-010
release "lab-demo" uninstalled
release "lab-demo-010" uninstalled
```