Exercise 3.1: Deploy a New Application

Overview

In this lab we will deploy a very simple **Python** application, test it using **Podman**, ingest it into Kubernetes and configure probes to ensure it continues to run. This lab requires the completion of the previous lab, the installation and configuration of a Kubernetes cluster.

Note that **Podman** and **crictl** use the same syntax as **docker**, so the labs `should` work the same way if you happen to be using Docker instead.

Working with A Simple Python Script

1. Install python on your cp node. It may already be installed, as is shown in the output below.

```
student@cp:~$ sudo apt-get -y install python3
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
python3 is already the newest version (3.8.2-0ubuntu2).
The following package was automatically installed and is no longer required:
  libnuma1
Use 'sudo apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

2. Locate the python binary on your system.

```
student@cp:~$ which python3
```

```
/usr/bin/python3
```

3. Create and change into a new directory. The Podman build process pulls everything from the current directory into the image file by default. Make sure the chosen directory is empty.

```
student@cp:~$ mkdir app1
student@cp:~$ cd app1
student@cp:~/app1$ ls -l
```

```
total 0
```

4. Create a simple python script which prints the time and hostname every 5 seconds. There are six commented parts to this script, which should explain what each part is meant to do. The script is included with others in the course tar file, you may consider using the **find** command used before to find and copy over the file.

   While the command shows **vim** as an example other text editors such as **nano** work just as well.

```
student@cp:~/app1$ vim simple.py
```

simple.py

```python
#!/usr/bin/python3
## Import the necessary modules
import time
import socket

## Use an ongoing while loop to generate output
while True :

## Set the hostname and the current date
  host = socket.gethostname()
  date = time.strftime("%Y-%m-%d %H:%M:%S")

## Convert the date output to a string
  now = str(date)

## Open the file named date in append mode
## Append the output of hostname and time
  f = open("date.out", "a" )
  f.write(now + "\n")
  f.write(host + "\n")
  f.close()

## Sleep for five seconds then continue the loop
  time.sleep(5)
```

5. Make the file executable and test that it works. Use `Ctrl-C` to interrupt the `while` loop after 20 or 30 seconds. The output will be sent to a newly created file in your current directory called `date.out`.

```
student@cp:~/app1$ chmod +x simple.py
student@cp:~/app1$ ./simple.py
```

```
^CTraceback (most recent call last):
  File "./simple.py", line 42, in <module>
    time.sleep(5)
KeyboardInterrupt
```

6. Verify the output has node name and timedate stamps.

```
student@cp:~/app1$ cat date.out
```

```
2025-02-06 03:34:46
cp
2025-02-06 03:34:51
cp
2025-02-06 03:34:56
cp
<output_omitted>
```

7. Create a text file named `Dockerfile`.

Very Important

The name is important: it cannot have a suffix.

We will use three statements, `FROM` to declare which version of Python to use, `ADD` to include our script and `CMD` to indicate the action of the container. Should you be including more complex tasks you may need to install extra libraries, shown commented out as `RUN pip install` in the following example.

```
student@cp:~/app1$ vim Dockerfile
```

Dockerfile

```
FROM docker.io/library/python:3
ADD simple.py /
## RUN pip install pystrich
CMD [ "python", "./simple.py" ]
```

8. As sometimes happens with open source projects the upstream version may have hiccups or not be available. When this happens we could make the tools from source code or install a binary someone else has created. For time reasons we will install an already created binary. Note the command is split on two lines. If you type it on one line there is no need for the backslash. Also notice the **cp** command and the trailing slashes, which may change the copy behaviour.

   Should the following binary not be available, you can search for an existing, and recent binary.

```
student@cp:~/app1$ curl -fsSL -o podman-linux-amd64.tar.gz \
https://github.com/mgoltzsche/podman-static/releases/latest/download/podman-linux-amd64.tar.gz

student@cp:~/app1$ tar -xf podman-linux-amd64.tar.gz

student@cp:~/app1$ sudo cp -r podman-linux-amd64/usr podman-linux-amd64/etc /
```

9. Build the container. The output below shows the end-build as necessary software was downloaded. You will need to use **sudo** in order to run this command. After the four step process completes the last lines of output should indicate success. Note the dot (.) at the end of the command indicates the current directory.

   The **podman** command has been built to replace all of the functionality of **docker**, and should accept the same syntax. As with any open source, fast changing project there `could` be slight differences. You may note the process takes almost a minute to finish, with a pause or two in output. Some may alias **docker** to **podman**.

   Choose to use the `docker.io` version of python.

```
student@cp:~/app1$ sudo podman build -t simpleapp .
```

```
STEP 1/3: FROM python:3
Resolved "python" as an alias (/etc/containers/registries.conf.d/000-shortnames.conf)
Trying to pull docker.io/library/python:3...
Getting image source signatures
Copying blob 0c6b8ff8c37e done
Copying blob 724cfd2dc19b done
Copying blob 808edda3c2e8 done
Copying blob 461bb1d8c517 done
Copying blob e6d3e61f7a50 done
Copying blob 412caad352a3 done
Copying blob 1bb6570cd7ac done
Copying blob aca06d6d45b1 done
Copying blob 678714351737 done
Copying config dfce7257b7 done
Writing manifest to image destination
Storing signatures
STEP 2/3: ADD simple.py /
--> 0e0bd8b3671
STEP 3/3: CMD [ "python", "./simple.py" ]
COMMIT simpleapp
--> 5b825f8d5f3
Successfully tagged localhost/simpleapp:latest
5b825f8d5f3a69660f2278a7119716e2a0fc1a5756d066800429f8030f83e978
```

10. Verify you can see the new image among others downloaded during the build process, installed to support the cluster, or you may have already worked with. The newly created `simpleapp` image should be listed first.

```
student@cp:~/app1$ sudo podman images
```

```
REPOSITORY                 TAG         IMAGE ID      CREATED        SIZE
localhost/simpleapp        latest      11d4607c72e0  7 seconds ago  1.04 GB
docker.io/library/python   3           58a8f3dcd68a  3 days ago     1.04 GB
<output_omitted>
```

11. Use **sudo podman** to run a container using the new image. While the script is running you won't see any output and the shell will be occupied running the image in the background. After 30 seconds use **ctrl-c** to interrupt. The local `date.out` file will not be updated with new times, instead that output will be a file of the container image.

```
student@cp:~/app1$ sudo podman run localhost/simpleapp
```

```
^CTraceback (most recent call last):
  File "./simple.py", line 24, in <module>
    time.sleep(5)
KeyboardInterrupt
```

12. Locate the newly created `date.out` file. The following command should show two files of this name, the one created when we ran simple.py and another under `/var/lib/containers` when run via a podman or crio container.

```
student@cp:~/app1$ sudo find / -name date.out
```

```
/home/student/app1/date.out
/var/lib/containers/storage/overlay/
0dea104afa098f608dff06b17b2196d0ba12d09a775243781862abf016e3378a/diff/date.out
```

13. View the contents of the `date.out` file created via Podman. Note the need for **sudo** as Podman created the file this time, and the owner is `root`. The long name is shown on several lines in the example, but would be a single line when typed or copied.

```
student@cp:~/app1$ sudo tail \
 /var/lib/containers/storage/overlay/
0dea104afa098f608dff06b17b2196d0ba12d09a775243781862abf016e3378a/diff/date.out
```

```
2025-02-06 03:39:11
08a97648dfa4
2025-02-06 03:39:16
08a97648dfa4
2025-02-06 03:39:21
08a97648dfa4
```