Share an application image

Aim

We will try to publish our web application from last week, so container services on the cloud can use it.

Motivation

A precondition for continuous integration and continuous delivery (CI/CD) is to publish the image(s) you need for the app to build and run its containers on swarms or Kubernetes clusters.

Requirements

To share Docker images, you have to use a Docker registry. The default registry is Docker Hub and is where all of the images we've used have come from.

> **Docker ID**
>
> A Docker ID allows you to access Docker Hub which is the world's largest library and community for container images. Create a Docker ID for free if you don't have one.

Create a repo

To push an image, we first need to create a repository on Docker Hub.

1. Sign up or Sign in to Docker Hub.
2. Click the **Create Repository** button.
3. For the repo name, use `getting-started`. Make sure the Visibility is `Public`.

   > **Private repositories**
   >
   > Docker also offers private repositories which allows you to restrict content to specific users or teams. This is a paid-for service.

4. Click the **Create** button!

If you look at the image below an example **Docker command** can be seen. This command will push to this repo.

Push the image

First, find the image that you created last week: `docker image ls`. You should see that there is an image called `python-docker` (or whatever you named it then).

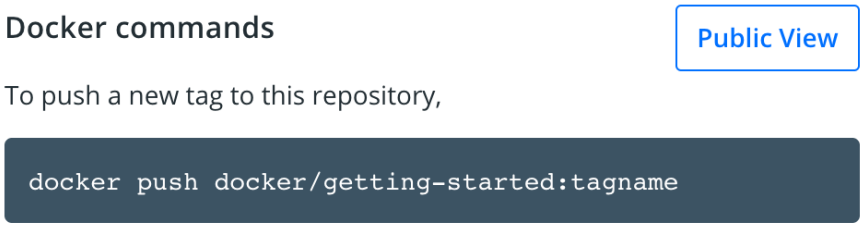

Figure 1: Docker command with push example

1. In the command line, try running the push command you see on Docker Hub. Note that your command will be using your namespace, not "docker".
2. ```
   $ docker push docker/getting-started
   The push refers to repository [docker.io/docker/getting-started]
   An image does not exist locally with the tag: docker/getting-started
   ```

   Why did it fail? The push command was looking for an image named docker/getting-started, but didn't find one. If you run `docker image ls`, you won't see one either.

   To fix this, we need to "tag" our existing image we've built to give it another name.
3. Login to the Docker Hub using the command `docker login -u YOUR-USER-NAME`.
4. Use the `docker tag` command to give the `python-docker` image a new name. Be sure to swap out `YOUR-USER-NAME` with your Docker ID.
5. `docker tag python-docker YOUR-USER-NAME/getting-started`

   Learn more about docker tag.
6. Now try your push command again. If you're copying the value from Docker Hub, you can drop the `tagname` portion, as we didn't add a tag to the image name. If you don't specify a tag, Docker will use a tag called `latest`.
7. `docker push YOUR-USER-NAME/getting-started`

Run the image on a new instance

Now that our image has been built and pushed into a registry, let's try running our app on a brand new instance that has never seen this container image! To do this, we will use Play with Docker.

1. Open your browser to Play with Docker.
   ```

2. Click **Login** and then select **docker** from the drop-down list.

   You can sign up for Play with Docker for free if you have not used it before.

3. Connect with your Docker Hub account.

4. Once you're logged in, click on the **ADD NEW INSTANCE** option on the left side bar. If you don't see it, make your browser a little wider. After a few seconds, a terminal window opens in your browser.

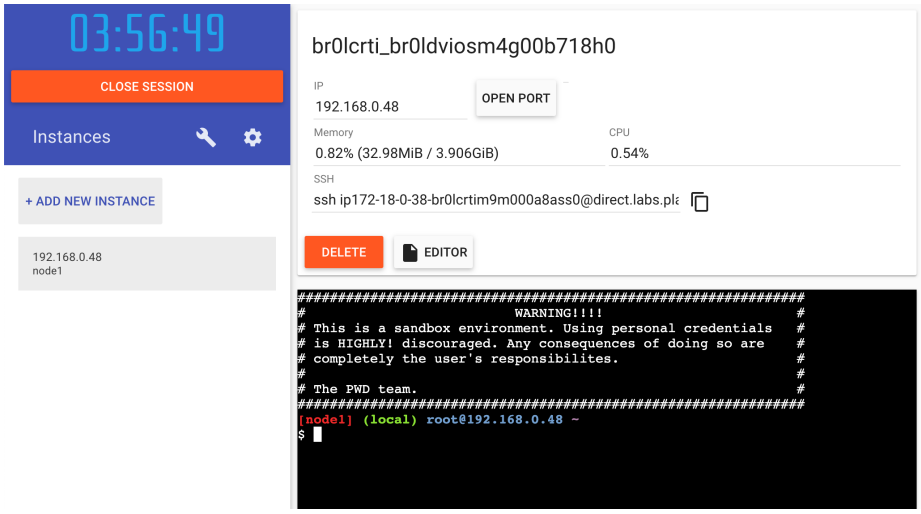

Figure 2: Play with Docker add new instance

5. In the terminal, start your freshly pushed app.

6. `docker run -dp 5000:5000 YOUR-USER-NAME/getting-started`

   You should see the image get pulled down and eventually start up!

7. Click on the 5000 badge when it comes up and you should see the app with your modifications! Hooray! If the 5000 badge doesn't show up, you can click on the "Open Port" button and type in 5000.

Troubleshooting

Using a Mac M1/M2? Then the architecture required by your image will be `arm64` and it will only run on similar processors (thus not on `play-with-docker`).

To build and push your image for alternative architectures use the following command.

> Remember to run it from within the directory containing the relevant `Dockerfile` for the image.

```
docker buildx build --push --platform linux/arm64/v8,linux/amd64 --tag YOUR-USER-NAME/gettir
```

Now run `docker pull YOUR-USER-NAME/getting-started` on the `play-with-docker` terminal and try running a container from the image again.

Bootcamp Challenge

1. In your VSCode IDE, make a change to your Flask application by adding a new route below the existing "hello world" route:

```
@app.route("/goodbye")
def goodbye():
    return {"data": "So long, and thanks for all the fish!"}
```

2. Tag the existing `python-docker` image as `YOUR-USER-NAME/python-docker:v1`,
3. Rebuild the image to include the new route.
4. Push the latest image to Docker Hub.
5. Start a new terminal instance on `play-with-docker` and run your latest image in a container.
6. Check that it works, and includes the newly added route, by adding the path `/goodbye` after the hostname in the address bar
    - (the hostname is what is used for the address when you launch the port from the `play-with-docker` interface, as you did above).