Hosting your application in a docker container on DigitalOcean

Tools used:
- Docker
- Digital Ocean Container Registry
- Digital Ocean App Platform

Revision: 1

Introduction

When building a web application of any form, the last step in the process is to get it hosted. In this guide, we will look at the process of taking a Python-based API app and getting it to run first locally in a docker container and then finally on Digital Ocean with the container registry and App Runner services.

Although we are using Python, generally most of these steps can be performed for other applications that are running in a docker container.

Sample Python

In this example, we will use a file called **sample_api.py** with the code shown below. Notice on the last line, we are telling Flask to listen on **0.0.0.0** which means accept traffic from any source, if we left this as **localhost**, nobody would be able to connect to the server.

```
from flask import Flask
from flask_restful import Resource, Api
import json


app = Flask(__name__)
api = Api(app)

class GetProducts(Resource):
    def get(self):
        return {'id': '1221'}



api.add_resource(GetProducts, '/getProducts')
```

```
class GetRootPage(Resource):
    def get(self):
        return {'hello': 'root'}



api.add_resource(GetRootPage, '/')



if __name__ == '__main__':
    app.run(host='0.0.0.0',debug=True)
```

Creating the container

The first step is to take a basic Python application to run and create (or download) an associated **Dockerfile** which is the set of instructions to tell the container what to do when we run Docker.

This file is titled **Dockerfile** and does not have any extension such as a .txt on the end. If you are using Windows, a simple way to ensure no extension is added to the file is by adding double quotes around the filename when you save it e.g., “Dockerfile”.

In this Dockerfile, it tells Docker that we want to use the latest **Python** image as our base, we then outline what folder we want to work from, we will just use the **/** directory, and then we tell Docker we want to **copy** our file called **sample_api.py** and place it into the **/** folder in the Docker container.

An important element here is the dependencies. Notice how we are importing the required libraries for the code to work with the **RUN** command that is using pip to install them.

Towards the end, we call **EXPOSE** to open port **5000** to access the container, and then finally the **CMD** tells the container to run **Python** followed by the name of the program we want to run, which in our case is the **sample_api.py** script.

```
#Deriving the latest base image
FROM python:latest


#Labels as key value pair
LABEL Maintainer="me"


# Any working directory can be chosen as per choice like '/' or '/home' etc
# i have chosen /usr/app/src
WORKDIR /

#to COPY the remote file at working directory in container
COPY sample_api.py ./
# Now the structure looks like this '/usr/app/src/test.py'


#CMD instruction should be used to run the software
#contained by your image, along with any arguments.
RUN pip install flask
RUN pip install flask-restful
EXPOSE 5000
CMD [ "python", "./sample_api.py"]
```

Remember to get this to work, this **Dockerfile** must be in the same folder as the Python code we are working on.

Next, open a command prompt in the same folder and run the following command to tell the docker to read the **Dockerfile** and **build** the container. The -t adds a custom **tag** to identify the container, we are calling it **py1.**

There is a dot at the end of this command, remember to include it.

```
docker build -t py1 .
```

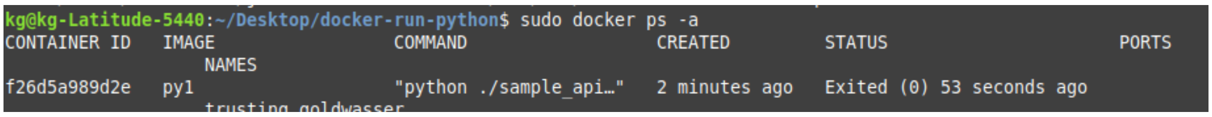
Finally, to test if it is working, we can call the docker **run** command and pass the port we want to map our local machine's port 5000 to, in this case, the container's port 5000. We also use the label **py1** to identify the container.

```
docker run -p 5000:5000 py1
```

To check the process is running, we can open up a terminal and type the following command

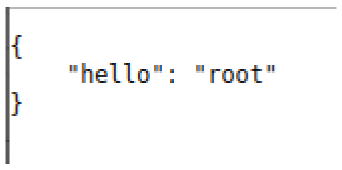
```
docker ps -a
```

This shows us the container ID and when it was launched.



```
kg@kg-Latitude-5440:~/Desktop/docker-run-python$ sudo docker ps -a
CONTAINER ID   IMAGE                COMMAND                  CREATED         STATUS                      PORTS
     NAMES
f26d5a989d2e   py1                  "python ./sample_api…"   2 minutes ago   Exited (0) 53 seconds ago
     trusting_goldwasser
```

Finally to test it, open up a browser and head to http://localhost:5000/ and you should see the following output.



```
{
    "hello": "root"
}
```

Remember

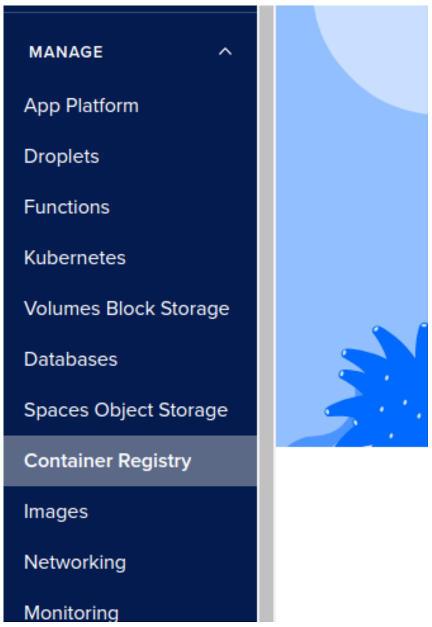
Ensure that you can access the container this way and see the code working correctly. If it does not work on your local machine, it will not work on the cloud server.

Create a docker image registry

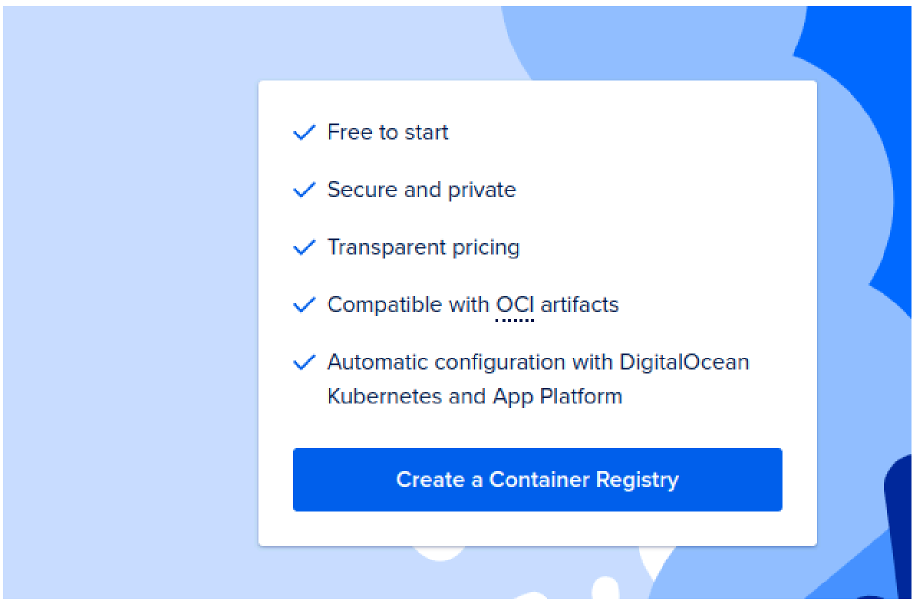
Now that we have tested the container and we know it is working perfectly on our local machine, the next step in the process is to get the container out to the cloud.

This is a two-part process. The first part is to push our container to a **“container registry”**. This registry is a storage place for docker container images. The registry does not run the container, it is simply a repository that we can point to in the future to get our image from.

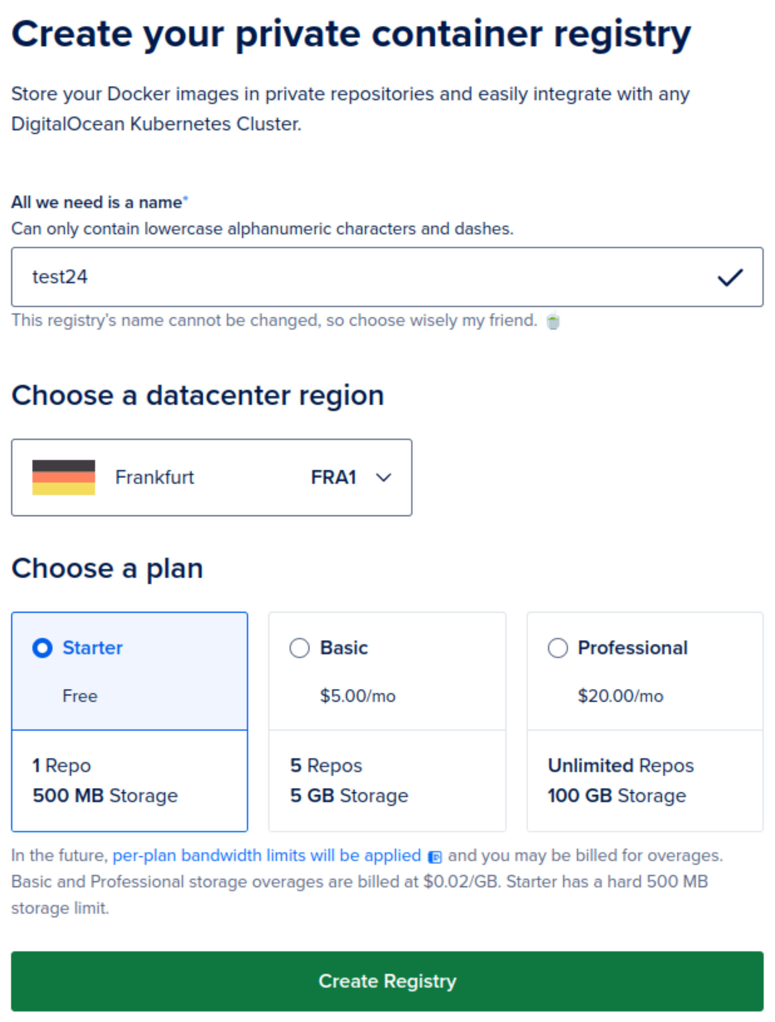
Log into DigitalOcean.com and once you have logged in on the left-hand side you will see an option for the container registry.



Then on the right, click “Create a Container Registry”.


Free to start
Secure and private
Transparent pricing
Compatible with OCI artifacts
Automatic configuration with DigitalOcean Kubernetes and App Platform
Create a Container Registry

Next, provide a **name** for your registry. Then outline where you want the registry hosted. Normally we select whichever location is closest to you. Then finally, click on **Starter Free**, followed by **Create**.



Create your private container registry

Store your Docker images in private repositories and easily integrate with any DigitalOcean Kubernetes Cluster.

All we need is a name*
Can only contain lowercase alphanumeric characters and dashes.

test24 ✓

This registry's name cannot be changed, so choose wisely my friend.

Choose a datacenter region

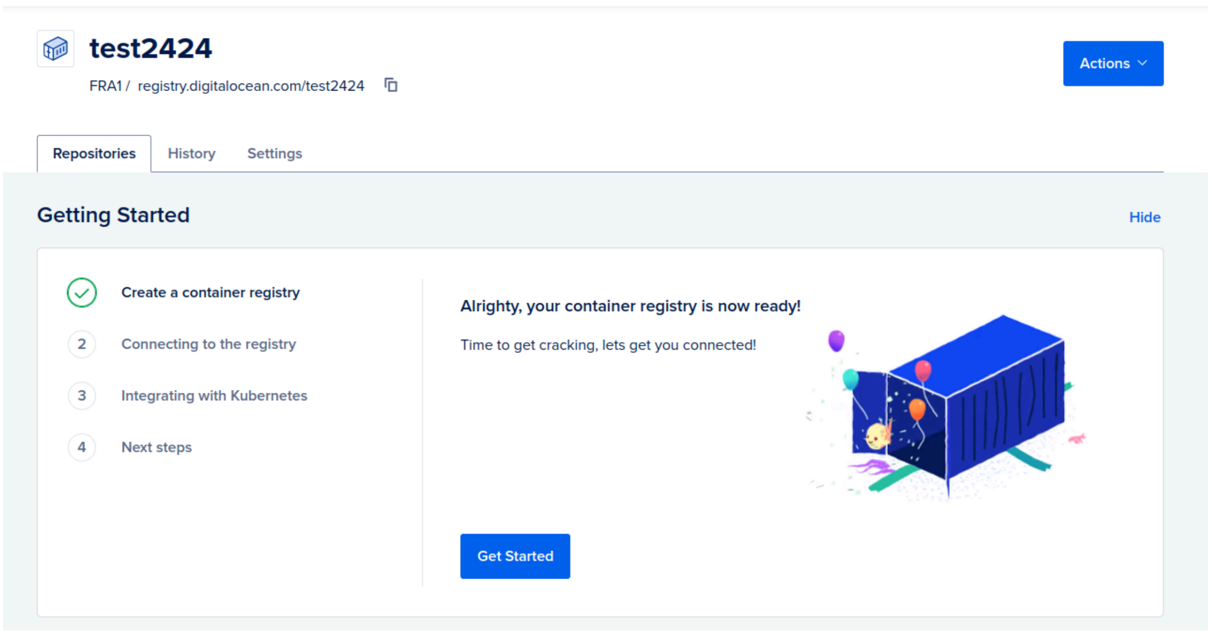
Frankfurt FRA1

Choose a plan

● Starter	○ Basic	○ Professional
Free	$5.00/mo	$20.00/mo
1 Repo 500 MB Storage	5 Repos 5 GB Storage	Unlimited Repos 100 GB Storage

In the future, per-plan bandwidth limits will be applied and you may be billed for overages. Basic and Professional storage overages are billed at $0.02/GB. Starter has a hard 500 MB storage limit.

You will then see the dashboard for the registry. The most important element here is the **registry URL** on the top left, this is the **unique address** for your registry.


test2424
FRA1 / registry.digitalocean.com/test2424
Actions
Repositories
History
Settings
Getting Started
Hide
Create a container registry
2 Connecting to the registry
3 Integrating with Kubernetes
4 Next steps
Get Started

Alrighty, your container registry is now ready!

Time to get cracking, lets get you connected!

We are now ready to move onto the software component of this process, which is outlined when you click “Get started”.

Getting the doctl tool

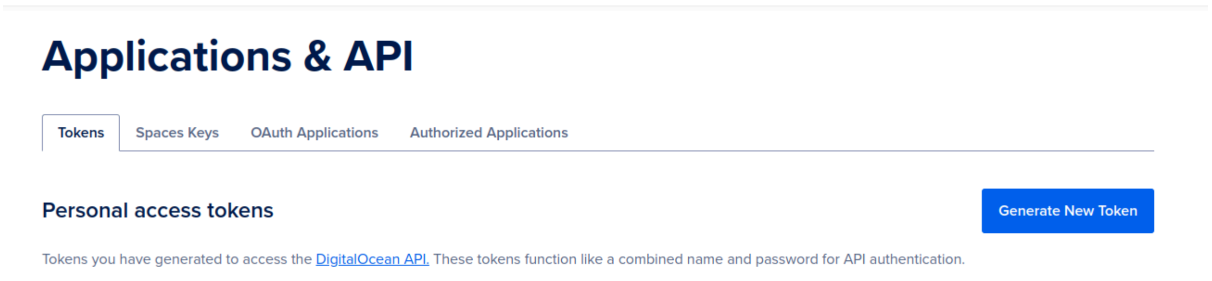
To interact with Digital Ocean, we need a tool titled **doctl.** This tool can be downloaded from their GitHub repository. Under the **assets** section. Download and extract the correct version for your operating system and place the tool into your project folder.

https://github.com/digitalocean/doctl/releases

Creating an API key

For **doctl** to interact with Digital Ocean, we need to create an **API key**. Follow the link below and it will show you what keys have been created and allow you to create a new one.

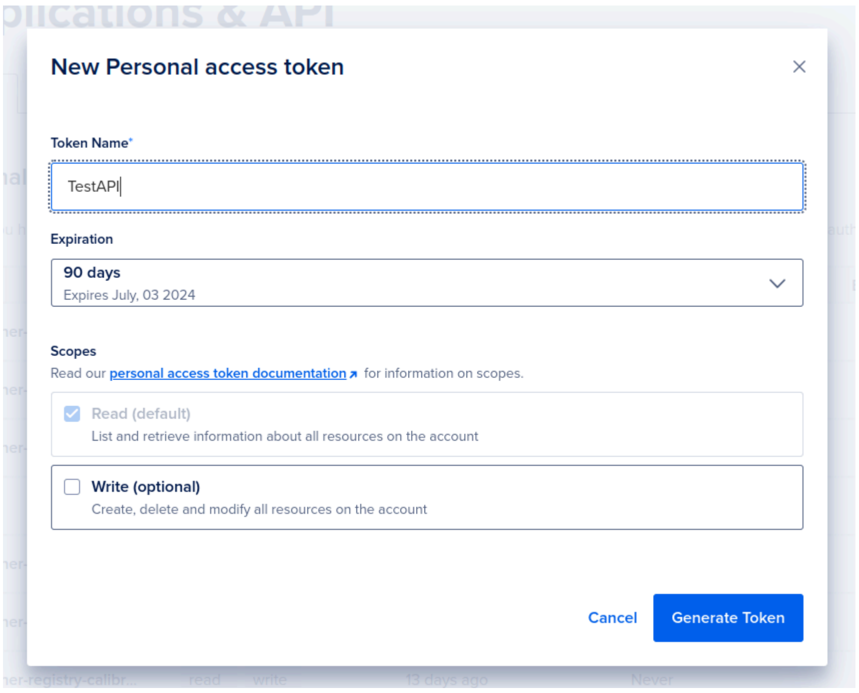
https://cloud.digitalocean.com/account/api/tokens



Tokens you have generated to access the DigitalOcean API. These tokens function like a combined name and password for API authentication.

On this page, click **Generate New Token**. In the window that opens, provide a name, and expiration date.

In addition to this, also check **Write** to allow us to modify resources if needed then click **Generate Token**.


New Personal access token

Token Name*

TestAPI

Expiration

90 days
Expires July, 03 2024

Scopes

Read our personal access token documentation for information on scopes.

✓ Read (default)
List and retrieve information about all resources on the account

☐ Write (optional)
Create, delete and modify all resources on the account

Keep your API key safe, you cannot see it again after it has been created!

An example API key looks like this:

```
Dop_v1_1c8cf0e8a49a9a640f352156d734b663026f3757a3594cb25a6a16d15e697c4
```

Running doctl

Finally, in a command prompt, we will now use this token. Place **doctl** in the same folder as your other source files for convenience. Then run the following command.

```
doctl auth init
```

Then we can run the login process:

```
doctl registry login
```

During this process, it will ask you for your **API key** to validate your account.

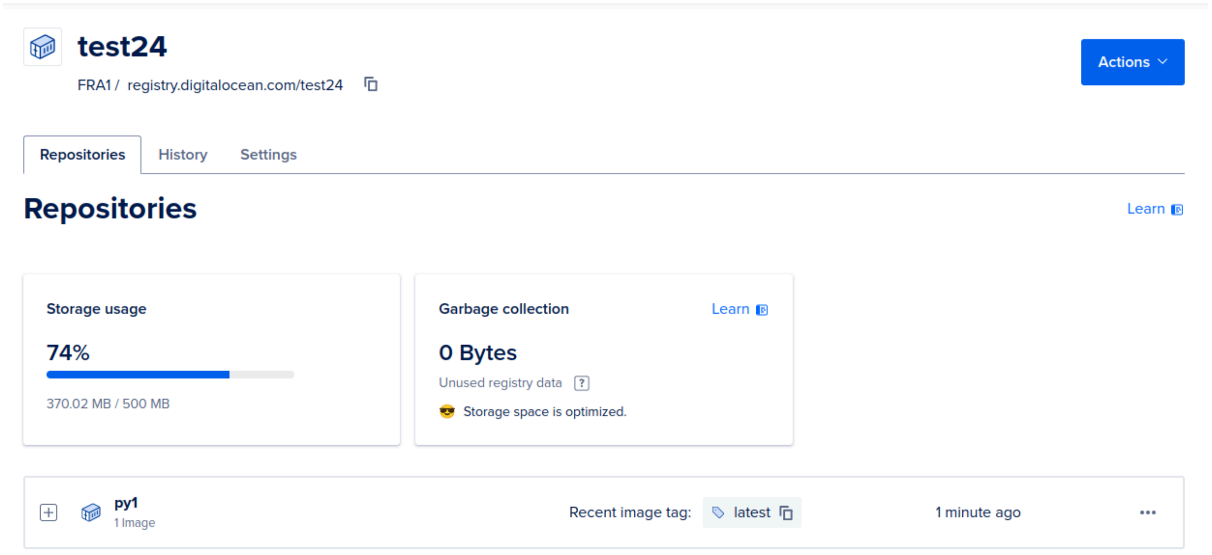
Pushing to the registry

Now we need to push the docker container that we created to the **container registry**. This needs two pieces of information, the first is the **tag** we associated with the container, if you remember earlier when we were running docker to build and run the container, we added the tag py1.

We then run the following command from a terminal, the red is the tag name and the blue is our registry URL.

```
docker tag py1 registry.digitalocean.com/test2424/py1
```

This blue URL is shown on the top left corner of the registry dashboard shown below. This can be accessed by clicking the copy button. Remember it will not append the **tag** shown in red to your URL, you need to manually add this.

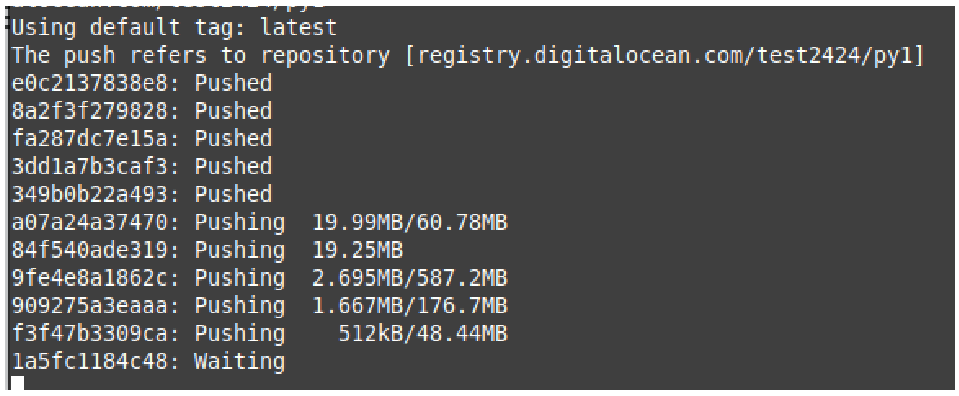


If this command is successful, no error message will be returned from the console where you issued it.

We can then push the container, to do this issue the following command. Again the convention is the same, the blue is your unique registry URL and the red is the tag name we associated with the Docker container we have.

```
docker push registry.digitalocean.com/test2424/py1
```

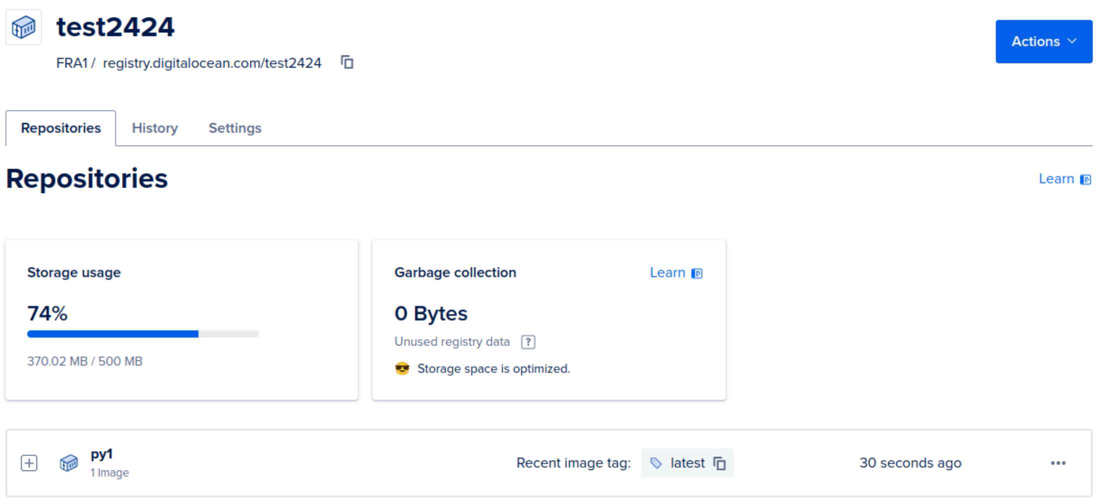
You will see the **push** progress in the terminal as shown below.



```
Using default tag: latest
The push refers to repository [registry.digitalocean.com/test2424/py1]
e0c2137838e8: Pushed
8a2f3f279828: Pushed
fa287dc7e15a: Pushed
3dd1a7b3caf3: Pushed
349b0b22a493: Pushed
a07a24a37470: Pushing  19.99MB/60.78MB
84f540ade319: Pushing  19.25MB
9fe4e8a1862c: Pushing  2.695MB/587.2MB
909275a3eaaa: Pushing  1.667MB/176.7MB
f3f47b3309ca: Pushing    512kB/48.44MB
1a5fc1184c48: Waiting
```

At the end of this process, the terminal will return a hash digest indicating it is finished.

When we return to the container registry dashboard and hit refresh you will see your container image is available at the bottom of the page. In our case, the image was titled **py1**.



We are now ready to create a live cloud environment to deploy the image in the next step.

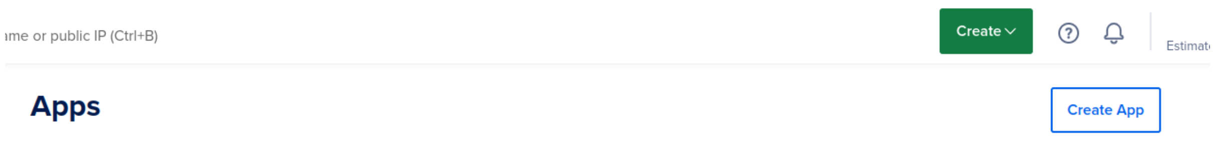
Deploying an instance of your container to the cloud

Now that we moved our container from our machine to the digital ocean container registry, the last step in the process is to deploy an instance of the container. This process is called **provisioning** as you are assigning real resources to your container to make it live to the public.

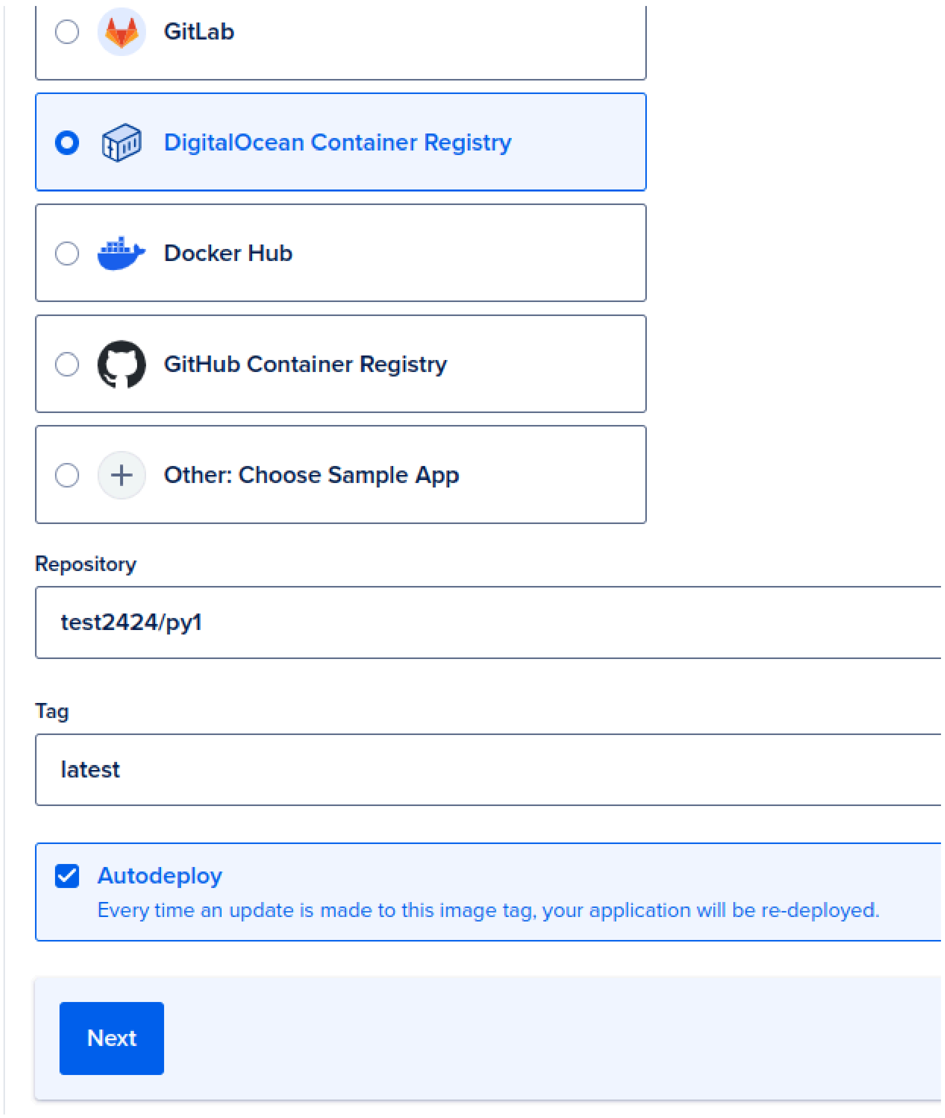
We will use the **App platform** available on Digital Ocean to do this. It is a very quick way to deploy a docker container from your custom registry where we put our image earlier.

First head to the **App Platform** dashboard: https://cloud.digitalocean.com/apps

And then on this dashboard, click **Create** on the right-hand side.



Then select DigitalOcean **Container Registry** as the **source** of the application. You will see references in the dropdown menu to the **py1** container we created earlier. Select **py1** and for the tag, select **latest**. Finally, click **Next**.

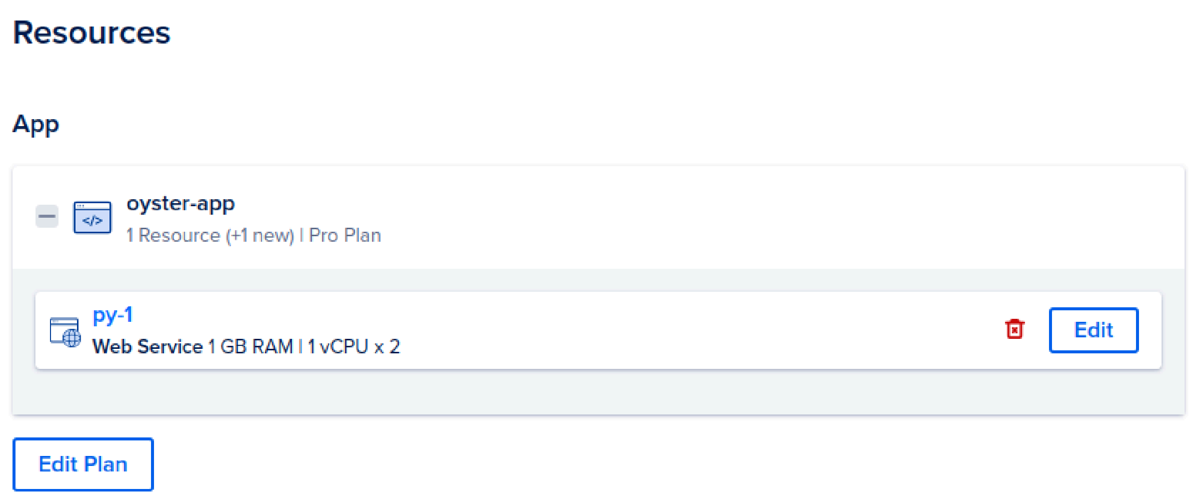


Repository: test2424/py1

Tag: latest

[✓] Autodeploy
Every time an update is made to this image tag, your application will be re-deployed.

After this, we can click **Edit Plan** at the bottom to change the amount of resources that are allocated to the instance.



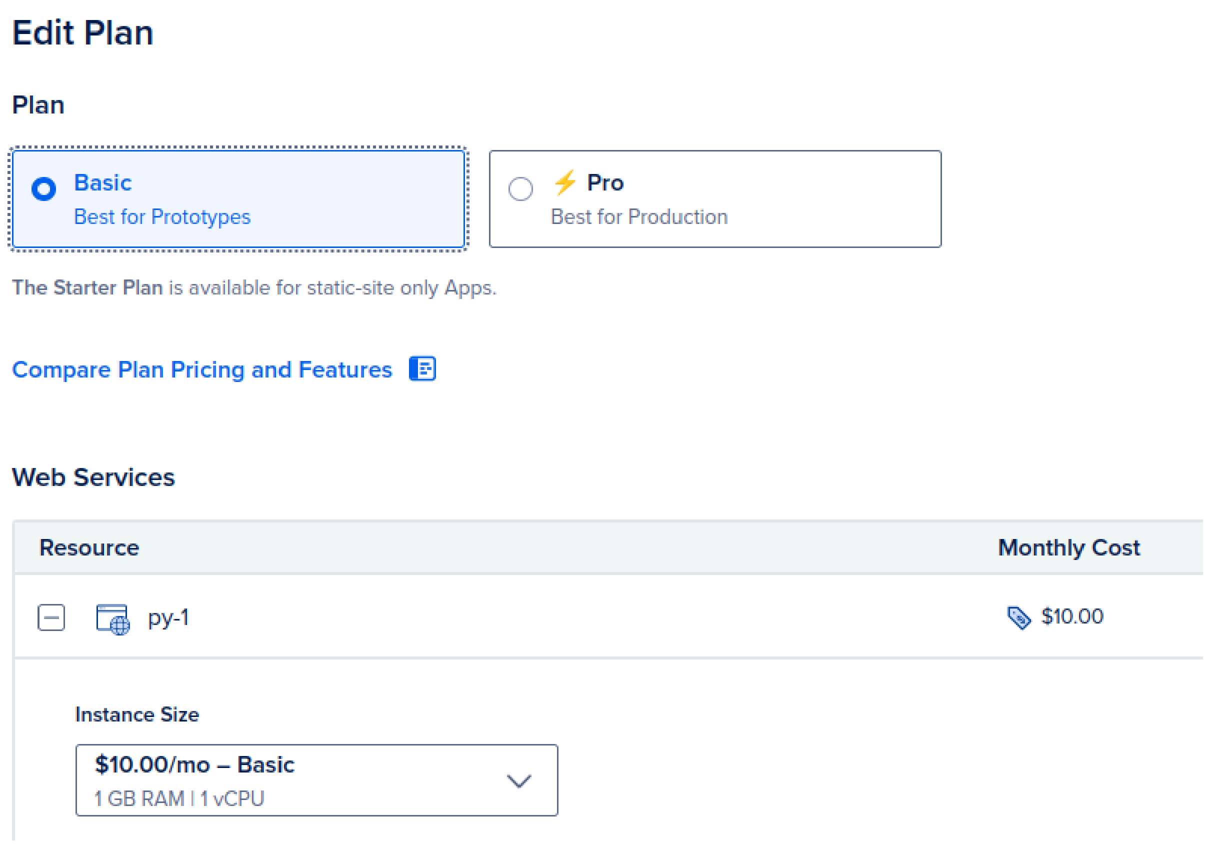
Resources

App

oyster-app
1 Resource (+1 new) | Pro Plan

py-1
Web Service 1 GB RAM | 1 vCPU x 2

Inside this, we can click **Basic** as we do not need too much power for this basic container, and then click **Back**.



Edit Plan

Plan

[✓ Basic
Best for Prototypes]

Pro
Best for Production

The Starter Plan is available for static-site only Apps.

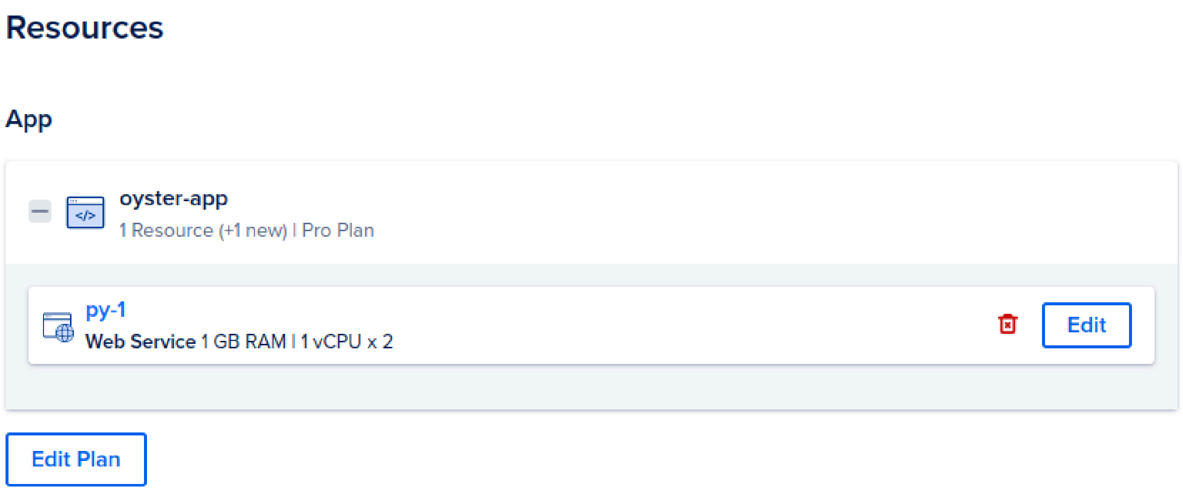
Web Services

Resource	Monthly Cost
py-1	$10.00

Instance Size

$10.00/mo – Basic
1 GB RAM | 1 vCPU

While on the same page after editing the amount of resources it will use, click **Edit** on the right-hand side.



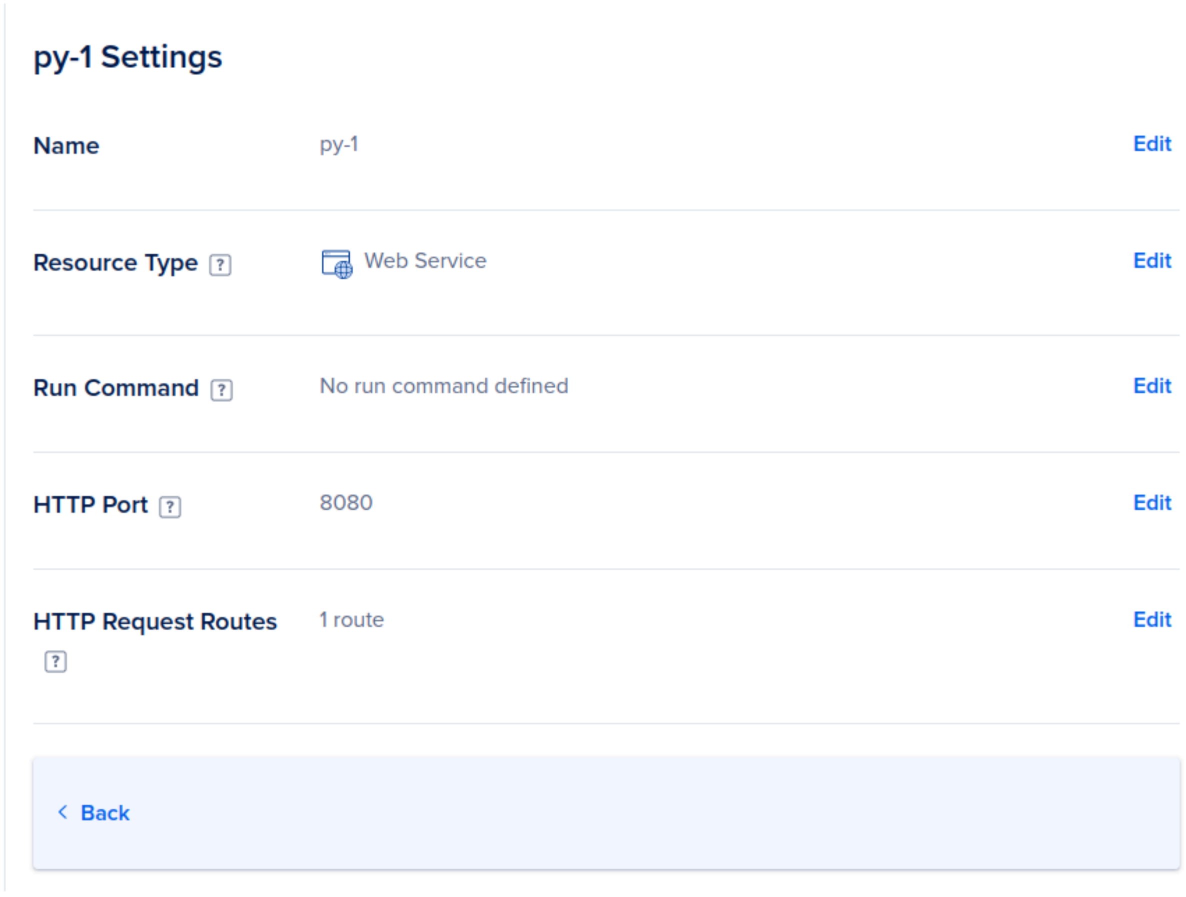
Resources

App

oyster-app
1 Resource (+1 new) | Pro Plan

py-1
Web Service 1 GB RAM | 1 vCPU x 2

When the edit window opens you will see the **default port** is incorrect as our app uses port **5000**. Click “edit” beside the port and change it to **5000**. After this, click **save** and then **back.**



py-1 Settings

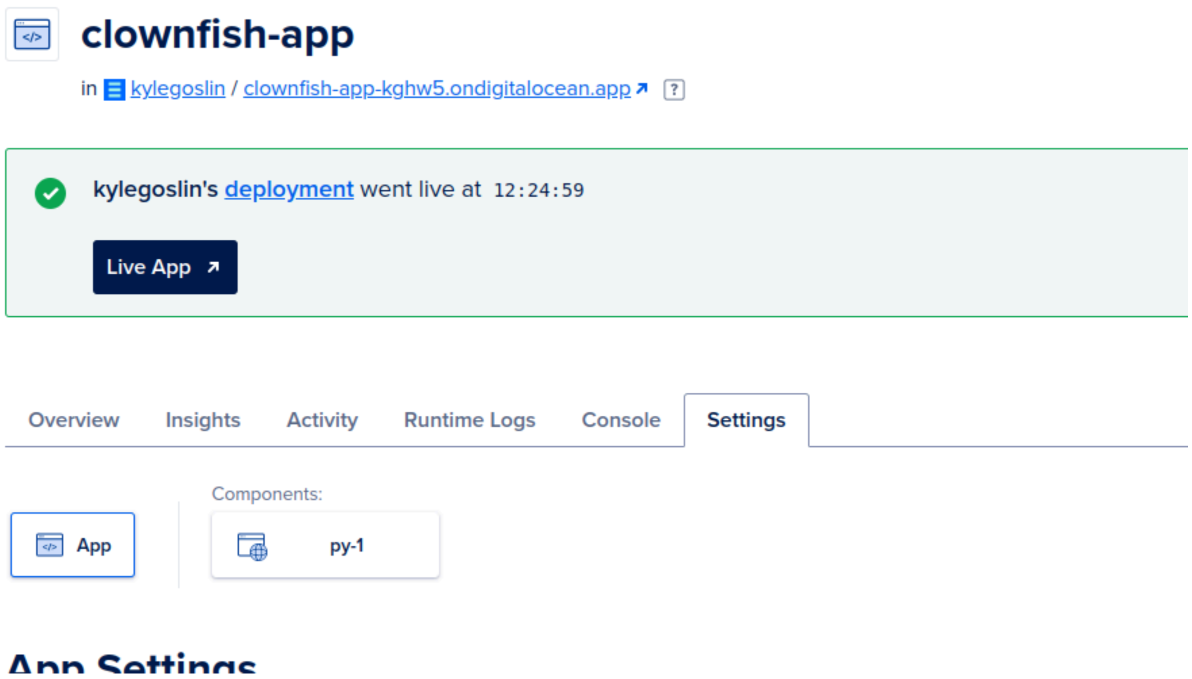
Name	py-1	Edit
Resource Type	Web Service	Edit
Run Command	No run command defined	Edit
HTTP Port	8080	Edit
HTTP Request Routes	1 route	Edit

After this, follow the process to the end, and skip the rest of the options as we do not need any other options, on the very last page click **Create Resource**.

Returning to the dashboard

Then to return to the main dashboard, click here https://cloud.digitalocean.com/apps

You will see your app. It normally takes a few minutes to provision the application. You will then see an option to view the **Live App.** When you click this, your app will open in the browser showing you the URL for your deployed cloud container image.


clownfish-app

in kylegoslin / clownfish-app-kghw5.ondigitalocean.app

kylegoslin's deployment went live at 12:24:59

In my case, the URL is https://clownfish-app-kghw5.ondigitalocean.app/

In the future, if you rebuild and push the same container again, it will automatically update the live version of your application on the App platform making a shorter deployment lifecycle.

Digital Ocean provides $200 in free credit for new accounts, this should last a long time as the container only costs $10 a month to run. To make your free credit last longer, remember to **destroy** old containers from your account when they are not being used.