Set up a Docker registry

Because a swarm consists of multiple Docker Engines, a registry is required to distribute images to all of them. You can use the [Docker Hub](https://hub.docker.com/) or maintain your own. Here’s how to create a throwaway registry, which you can discard afterward.

1. Start the registry as a service on your swarm:

$ docker service create --name registry --publish 5000:5000 registry:2

1. Check its status with docker service ls:

$ docker service ls

ID NAME REPLICAS IMAGE COMMAND

l7791tpuwkco registry 1/1 registry:2@sha256:1152291c7f93a4ea2ddc95e46d142c31e743b6dd70e194af9e6ebe530f782c17

Once it reads 1/1 under REPLICAS, it’s running. If it reads 0/1, it’s probably still pulling the image.

1. Check that it’s working with curl:

$ curl http://localhost:5000/v2/

{}

Create a Web For Registry

sudo service create –name registryweb \

-d \

-e ENV\_DOCKER\_REGISTRY\_HOST=ENTER-YOUR-REGISTRY-HOST-HERE \

-e ENV\_DOCKER\_REGISTRY\_PORT=ENTER-PORT-TO-YOUR-REGISTRY-HOST-HERE \

-p 8080:80 \

konradkleine/docker-registry-frontend:v2

Create the example application

The app used in this guide is based on the hit counter app in the [Get started with Docker Compose](https://docs.docker.com/compose/gettingstarted/) guide. It consists of a Python app which maintains a counter in a Redis instance and increments the counter whenever you visit it.

1. Create a directory for the project:

$ mkdir stackdemo

$ cd stackdemo

1. Create a file called app.py in the project directory and paste this in:

from flask import Flask

from redis import Redis

app = Flask(\_\_name\_\_)

redis = Redis(host='redis', port=6379)

@app.route('/')

def hello():

count = redis.incr('hits')

return 'Hello World! I have been seen {} times.\n'.format(count)

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host="0.0.0.0", port=8000, debug=True)

1. Create a file called requirements.txt and paste these two lines in:

flask

redis

1. Create a file called Dockerfile and paste this in:

FROM python:3.4-alpine

ADD . /code

WORKDIR /code

RUN pip install -r requirements.txt

CMD ["python", "app.py"]

1. Create a file called docker-compose.yml and paste this in:

version: '3'

services:

web:

image: 127.0.0.1:5000/stackdemo

build: .

ports:

- "8000:8000"

redis:

image: redis:alpine

Note that the image for the web app is built using the Dockerfile defined above. It’s also tagged with 127.0.0.1:5000 - the address of the registry created earlier. This will be important when distributing the app to the swarm.

Test the app with Compose

1. Start the app with docker-compose up. This builds the web app image, pull the Redis image if you don’t already have it, and create two containers.

You will see a warning about the Engine being in swarm mode. This is because Compose doesn’t take advantage of swarm mode, and deploys everything to a single node. You can safely ignore this.

$ docker-compose up -d

WARNING: The Docker Engine you're using is running in swarm mode.

Compose does not use swarm mode to deploy services to multiple nodes in

a swarm. All containers will be scheduled on the current node.

To deploy your application across the swarm, use `docker stack deploy`.

Creating network "stackdemo\_default" with the default driver

Building web

...(build output)...

Creating stackdemo\_redis\_1

Creating stackdemo\_web\_1

1. Check that the app is running with docker-compose ps:

$ docker-compose ps

Name Command State Ports

-----------------------------------------------------------------------------------

stackdemo\_redis\_1 docker-entrypoint.sh redis ... Up 6379/tcp

stackdemo\_web\_1 python app.py Up 0.0.0.0:8000->8000/tcp

You can test the app with curl:

$ curl http://localhost:8000

Hello World! I have been seen 1 times.

$ curl http://localhost:8000

Hello World! I have been seen 2 times.

$ curl http://localhost:8000

Hello World! I have been seen 3 times.

1. Bring the app down:

$ docker-compose down --volumes

Stopping stackdemo\_web\_1 ... done

Stopping stackdemo\_redis\_1 ... done

Removing stackdemo\_web\_1 ... done

Removing stackdemo\_redis\_1 ... done

Removing network stackdemo\_default

Push the generated image to the registry

To distribute the web app’s image across the swarm, it needs to be pushed to the registry you set up earlier. With Compose, this is very simple:

$ docker-compose push

Pushing web (127.0.0.1:5000/stackdemo:latest)...

The push refers to a repository [127.0.0.1:5000/stackdemo]

5b5a49501a76: Pushed

be44185ce609: Pushed

bd7330a79bcf: Pushed

c9fc143a069a: Pushed

011b303988d2: Pushed

latest: digest: sha256:a81840ebf5ac24b42c1c676cbda3b2cb144580ee347c07e1bc80e35e5ca76507 size: 1372

The stack is now ready to be deployed.

Deploy the stack to the swarm

1. Create the stack with docker stack deploy:

$ docker stack deploy --compose-file docker-compose.yml stackdemo

Ignoring unsupported options: build

Creating network stackdemo\_default

Creating service stackdemo\_web

Creating service stackdemo\_redis

The last argument is a name for the stack. Each network, volume and service name is prefixed with the stack name.

1. Check that it’s running with docker stack services stackdemo:

$ docker stack services stackdemo

ID NAME MODE REPLICAS IMAGE

orvjk2263y1p stackdemo\_redis replicated 1/1 redis:3.2-alpine@sha256:f1ed3708f538b537eb9c2a7dd50dc90a706f7debd7e1196c9264edeea521a86d

s1nf0xy8t1un stackdemo\_web replicated 1/1 127.0.0.1:5000/stackdemo@sha256:adb070e0805d04ba2f92c724298370b7a4eb19860222120d43e0f6351ddbc26f

Once it’s running, you should see 1/1 under REPLICAS for both services. This might take some time if you have a multi-node swarm, as images need to be pulled.

As before, you can test the app with curl:

$ curl http://localhost:8000

Hello World! I have been seen 1 times.

$ curl http://localhost:8000

Hello World! I have been seen 2 times.

$ curl http://localhost:8000

Hello World! I have been seen 3 times.

Thanks to Docker’s built-in routing mesh, you can access any node in the swarm on port 8000 and get routed to the app:

$ curl http://address-of-other-node:8000

Hello World! I have been seen 4 times.

1. Bring the stack down with docker stack rm:

$ docker stack rm stackdemo

Removing service stackdemo\_web

Removing service stackdemo\_redis

Removing network stackdemo\_default