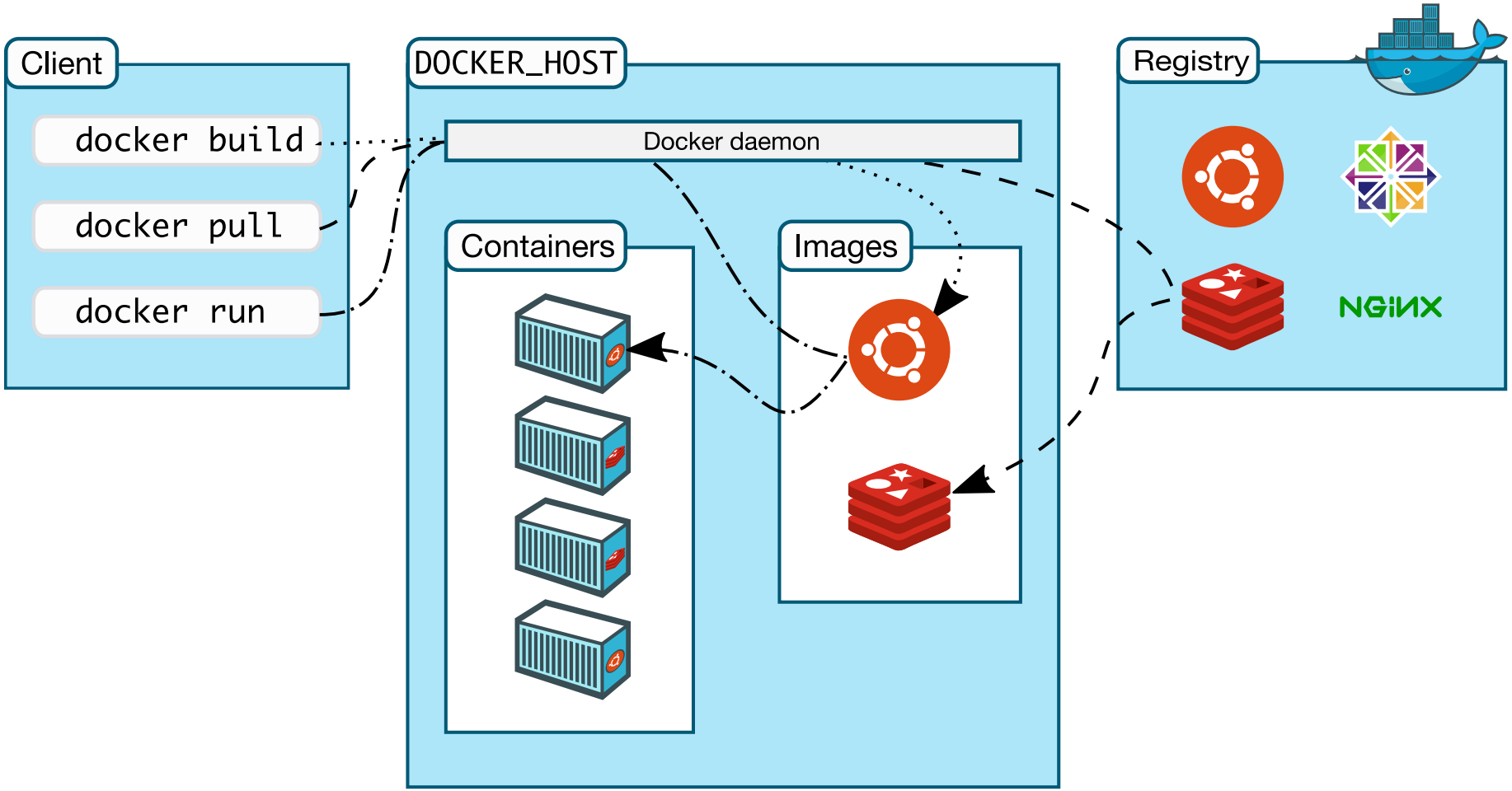
Building Docker containers in GitHub

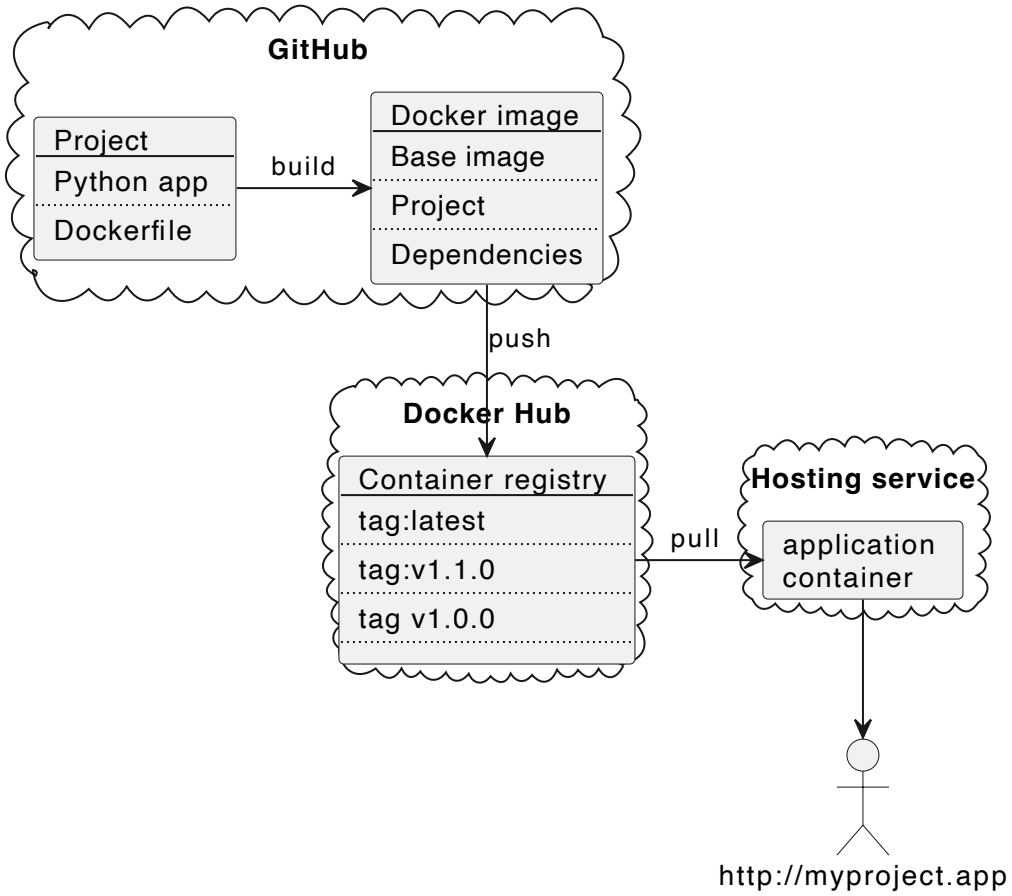
- What are containers?
- Deployment
- Project repository
- The Dockerfile
- Docker build
- GitHub Actions
- Container registry
- Deploy (Google Cloud)
- What next?

Containers are packaged applications that can be run anywhere Docker is available. Docker containers are "lightweight" and can be started very quickly. Sophisticated web services can be built by orchestrating many containers.



https://docs.docker.com/get-started/overview/

For this example we will build one container providing a Python **flask** web server. The project source files are hosted in GitHub. We're also going to need a hosting service to run our container and a *container registry*.



As we are going to deploy directly from a Git repository on GitHub all dependencies must be included in the repo. For a Python project this means a Pip requirements file together with our source files. We also need a Dockerfile.

```
__main__.py
static/*
templates/*
requirement.txt
Dockerfile
```

The Dockerfile describes how to build the container and, optionally, the command to run. Docker is programming language agnostic, so we need to specify the language tools we need, Python and Pip, and the libraries. This is typically done be choosing an appropriate base image.

```
FROM python:3.11-slim-bullseye
ARG VERSION
WORKDIR /app
# Install packages from requirements.txt
COPY requirements.txt .
RUN pip install --no-cache-dir --upgrade pip &&\
    pip install --no-cache-dir --trusted-host pypi.python.org -r requirements.txt
# Copy source code to working directory
COPY flaskserver ./flaskserver

# Default env vars and command.
ENV PORT=5000
ENV VERSION=$VERSION
CMD ["python", "flaskserver"]
```

The Dockerfile describes how to build the container and, optionally, the command to run. Docker is programming language agnostic, so we need to specify the language tools we need, Python and Pip, and the libraries. This is typically done be choosing an appropriate base image.

```
FROM python:3.11-slim-bullseye
ARG VERSION
WORKDIR /app
# Install packages from requirements.txt
COPY requirements.txt .
RUN pip install --no-cache-dir --upgrade pip &&\
    pip install --no-cache-dir --trusted-host pypi.python.org -r requirements.txt
# Copy source code to working directory
COPY flaskserver ./flaskserver

# Default env vars and command.
ENV PORT=5000
ENV VERSION=$VERSION
CMD ["python", "flaskserver"]
```

The Dockerfile describes how to build the container and, optionally, the command to run. Docker is programming language agnostic, so we need to specify the language tools we need, Python and Pip, and the libraries. This is typically done be choosing an appropriate base image.

```
FROM python:3.11-slim-bullseye
ARG VERSION
WORKDIR /app
# Install packages from requirements.txt
COPY requirements.txt .
RUN pip install --no-cache-dir --upgrade pip &&\
    pip install --no-cache-dir --trusted-host pypi.python.org -r requirements.txt
# Copy source code to working directory
COPY flaskserver ./flaskserver

# Default env vars and command.
ENV PORT=5000
ENV VERSION=$VERSION
CMD ["python", "flaskserver"]
```

The Dockerfile is a configuration file for the **docker build** tool. Each instruction in the Dockerfile adds additional layers to the base image. Docker build produces a lot of output, like this -

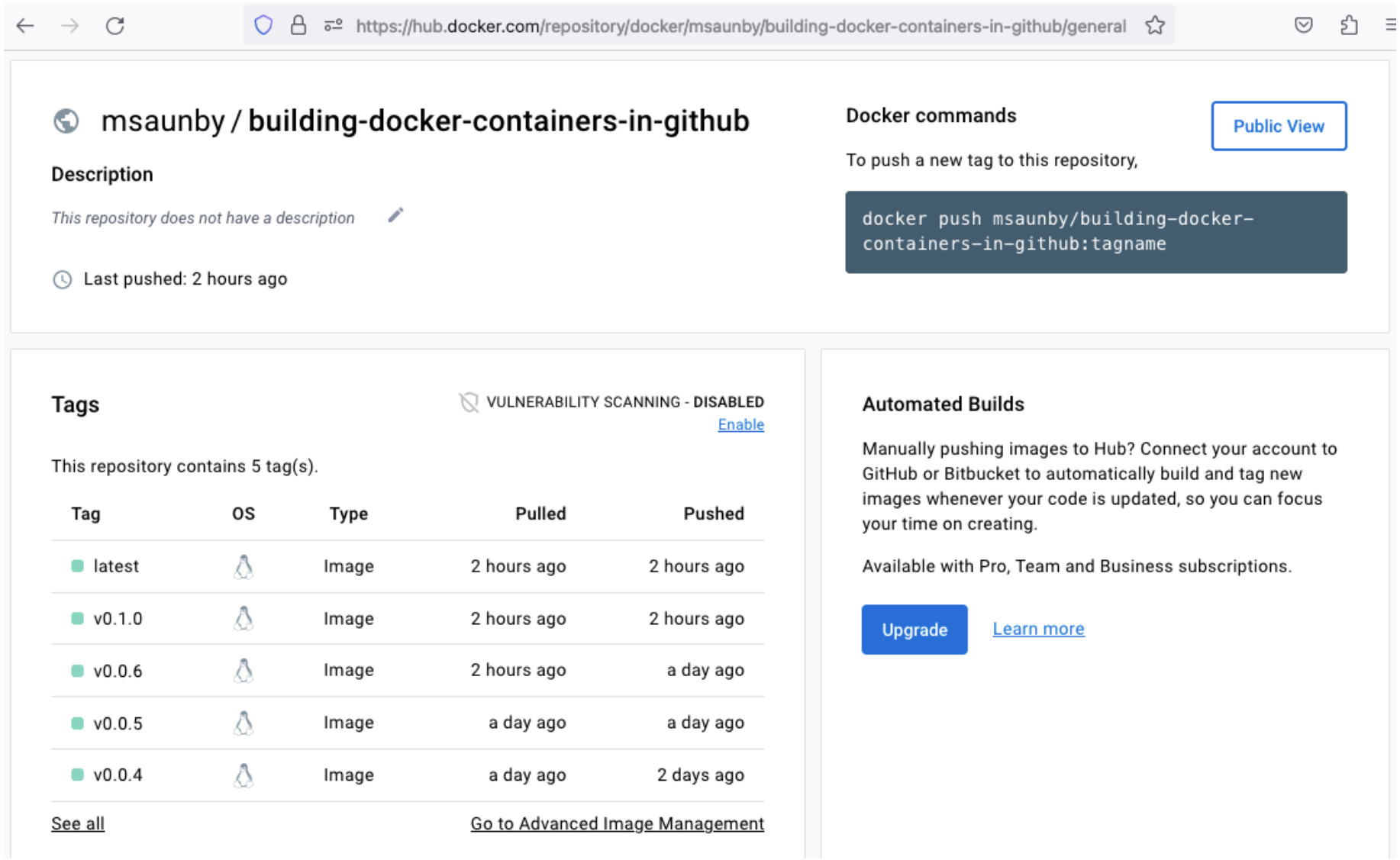
```
Run docker build . --file Dockerfile --tag my-image-name:$(date +%s)
Sending build context to Docker daemon  7.893MB

Step 1/7 : FROM python:3.11-slim-bullseye
3.11-slim-bullseye: Pulling from library/python
bb263680fed1: Pulling fs layer
43900b2bbd7f: Pulling fs layer
...
---> 79e97cd43c08
Step 3/7 : COPY requirements.txt .
 ---> 7c1c862a25e9
Step 4/7 : RUN pip install --no-cache-dir --upgrade pip &&    pip install --no-cache-dir --trus
 ---> Running in e4113ad35f0f
Requirement already satisfied: pip in /usr/local/lib/python3.11/site-packages (22.3.1)
Collecting pip
```

The output in the last slide was copied from a report generated automatically by GitHub. These are produced when a GitHub Action runs.

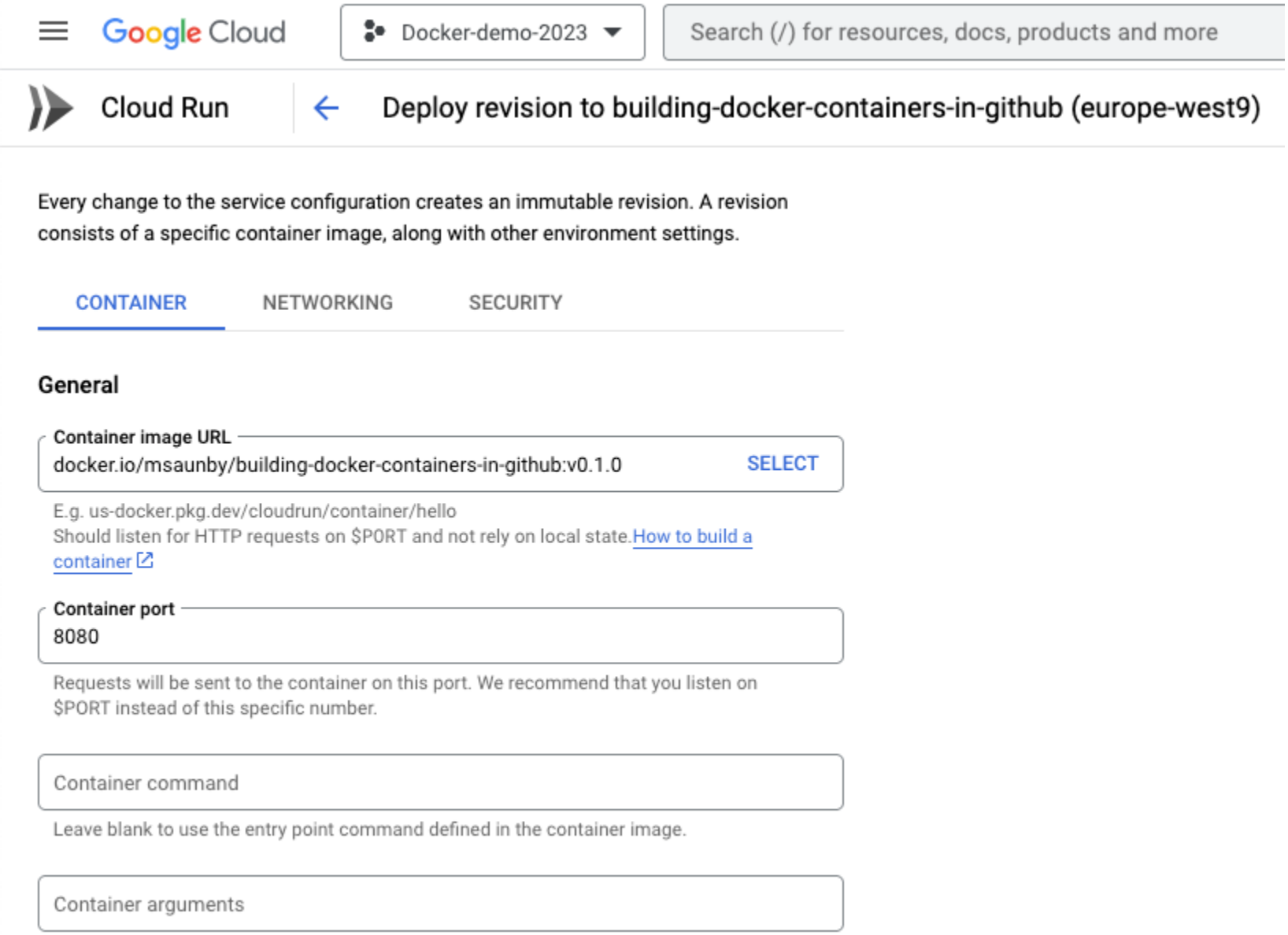
GitHub Actions are automated workflows that use containers to build, test and deploy software. Here's an action to build a Docker container.

```yaml
name: Docker Image CI
on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - name: Build the Docker image
      run: docker build . --file Dockerfile --tag my-image-name:$(date +%s)
```

A registry provides secure storage of container images with version control based on the tags given to images when pushed. The default container registry is Docker Hub. As we are building our container on GitHub a free Docker Hub account is sufficient.



Deploying a container is very simple, we just tell the hosting service the URL of the image to deploy.



https://building-docker-containers-in-github-2w4u2b57sa-od.a.run.app

Google Cloud | Docker-demo-2023 | Search (/) for resources, docs, products and more | Search

Cloud Run | Service details | EDIT AND DEPLOY NEW REVISION | SET UP CONTINUOUS DEPLOYMENT

building-docker-containers-in-github Region: europe-west9 URL: https://building-docker-containers-in-github-2w4u2b57sa-od.a.run.app

METRICS | SLOS | LOGS | REVISIONS | NETWORKING | SECURITY | TRIGGERS | INTEGRATION PREVIEW | YAML

Revisions MANAGE TRAFFIC

Filter Filter revisions

Name	Traffic	Depl	Actions
building-docker-containers-in-github-00012-qiy	100% (to latest)	1 mi	
building-docker-containers-in-github-00009-web	0%	2 ho	
building-docker-containers-in-github-00001-cor	0%	23 h	

building-docker-containers-in-github-00012-qiy

Deployed by mike.saunby@gmail.com using Cloud Console

CONTAINERS | NETWORKING | SECURITY | YAML

General

CPU allocation	CPU is only allocated during request processing
Startup CPU boost	Disabled
Concurrency	80
Request timeout	300 seconds
Execution environment	First generation (default)

Auto-scaling

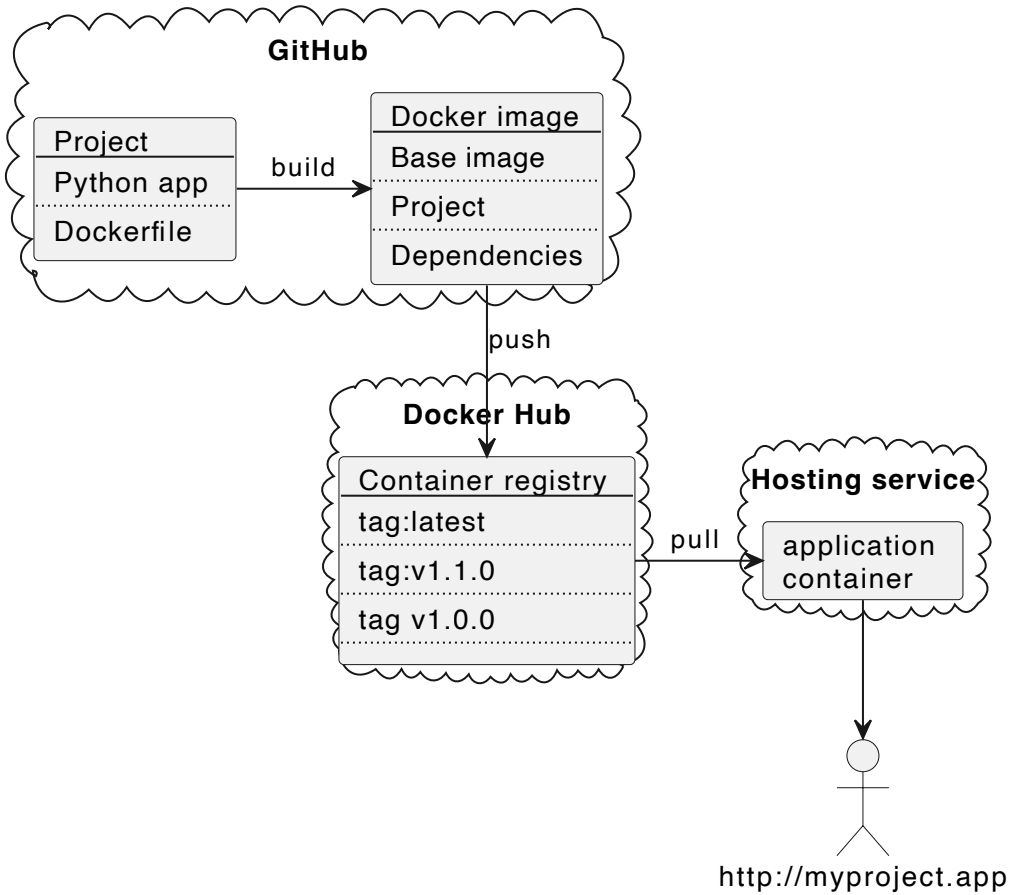
Max. instances	6

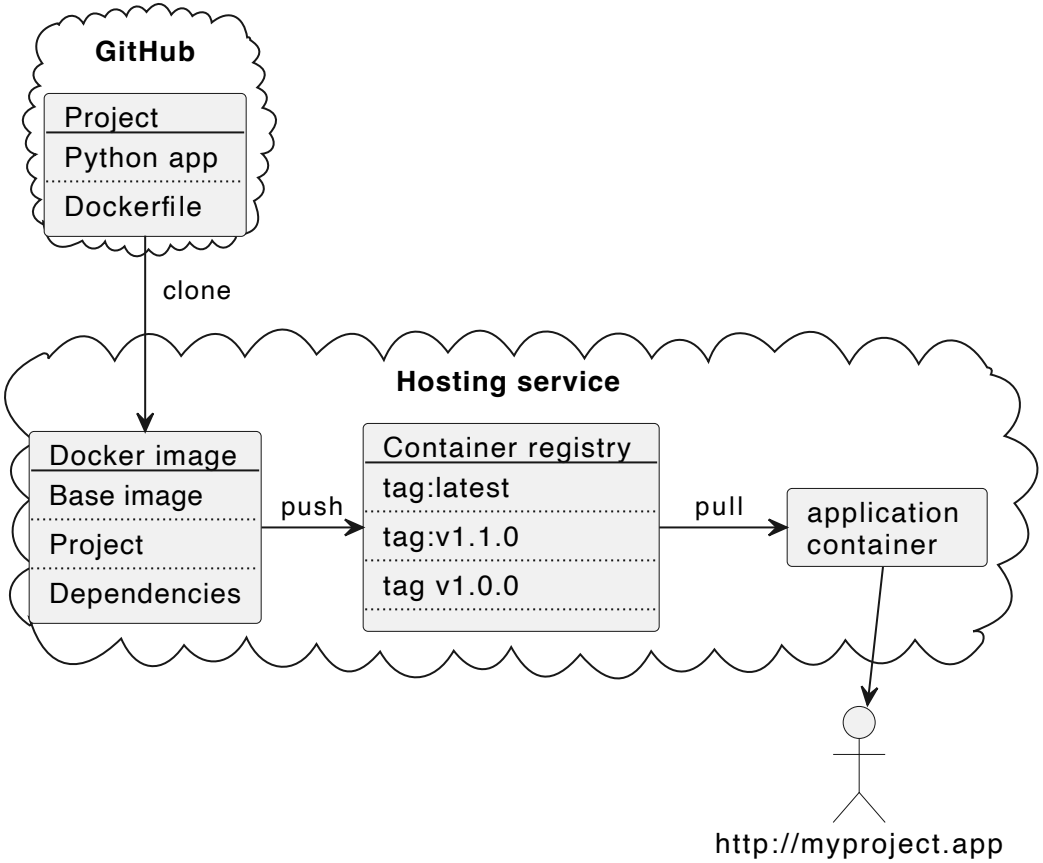
Image URL	docker.io/msaunby/building-docker-containers-in-gith...
Port	8080
Build	(no build information available)
Source	(no source information available)
Command and	(container entrypoint)

- GitHub pull-request already submitted
- Approve the pull-request
- Create a new release
- GitHub action builds and pushes the new image
- Manually update the deployed version in Google Cloud

Typically the last step is also automated, and there would be automated tests of the code before building the container.

Build on GitHub




GitHub
Project
Python app
Dockerfile
clone
Hosting service
Docker image
Base image
Project
Dependencies
push
Container registry
tag:latest
tag:v1.1.0
tag v1.0.0
pull
application container
http://myproject.app

Google Cloud Shell

Click on [>_] for command line tools including git and docker.

```
$ git clone <your repo>
$ cd <repo dir>
$ docker build -t testimg .
....
$ docker run -p 8080:5000 testimg
```

Containers as development environments

"The Visual Studio Code Dev Containers extension lets you use a container as a full-featured development environment."

https://code.visualstudio.com/docs/devcontainers/containers

Docker Compose and Kubernetes

https://docs.docker.com/compose/

https://microk8s.io/