Programming Best Practices

How CI/CD can help you become more efficient

1️⃣ CI - Continuous Integration

> Continuous integration is the practice of integrating all your code changes into the main branch of a shared source code repository early and often, automatically testing each change when you commit or merge them, and automatically kicking off a build.

CI workflows are

- **triggered** either by a `git` event such as a PR merge or a commit, or on schedule
- running on dedicated, **isolated**, **and fresh environments**
- responsible for building artifacts, running your test suites, and reporting any anomalies

Examples: GitHub Actions, Gitlab, Jenkins, Travis, Buildkite, CircleCI

GitHub Actions

GitHub Actions make it easy to automate all your software workflows as it integrates perfectly with GitHub without any configuration effort on the developer's side

✅ Plus, it's free for public GitHub repositories!
A few details to consider:

- An event can be any `git action` (push, new branch, etc.) but also any GitHub-specific event
- Only events on your remote GitHub repository will be considered; if you commit on your local machine but do not push, **nothing will be triggered**
- An action can be really anything: shell command, another GitHub Action, etc.

Documentation

Getting ready
Let's set the stage first. Create an empty folder and initialize it as a repo; this will be our sample repository to work from. Then, we need to create a few other things to use with our GHA.

```
mkdir -p ~/code/basic-ci && cd $_
git init

mkdir tests
touch tests/__init__.py
touch tests/test_sample.py

touch Makefile

touch requirements.txt
```

Add this to your `test_sample.py` file:

```
# pylint: disable-all

import unittest

class TestSample(unittest.TestCase):
    def test_sample(self):
        # We are simply checking whether 42==42!
        self.assertEqual(42, 42)
```

Add this to your `Makefile`:

```
default: pylint pytest

pylint:
    find . -iname "*.py" -not -path "./tests/*" | xargs -n1 -I {}  pylint --output-format=colorized {}; true

pytest:
    PYTHONDONTWRITEBYTECODE=1 pytest -v --color=yes
```

Then, add this to your `requirements.txt` file:

```
# Some example packages
pandas
numpy
seaborn

# Testing packages
pytest
```

Now that we have a test and a Makefile to run it, all you need is to write a **CI configuration file:**

```
git checkout -b ci-github-action-setup
mkdir -p .github/workflows
touch .github/workflows/python-ci.yml
```

```
# python-ci.yml
name: basic CI

on:
  push:
    branches: [ master, main ]
  pull_request:
    branches: [ master, main ]

jobs:
  build-and-run-pytest:

    runs-on: ubuntu-latest

    steps:
    # First step (unnamed here) is to checkout to the branch that triggered the event
    - uses: actions/checkout@v3

    # Second step: install python 3.10
    - name: Set up Python 3.10
      uses: actions/setup-python@v2
      with:
        python-version: "3.10"
```

(1/2)

```
# Third step: install python packages using a requirements file
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip cython wheel
        pip install -r requirements.txt

    # Fourth step: run tests with Pytest
    - name: Run tests
      run: make
```

(2/2)

Then **commit:**

```
git add .github
git commit -m "Configure GitHub Actions CI to run pytest"
git push origin ci-github-action-setup
```

and create a **pull request:**

```
gh pr create --web
```

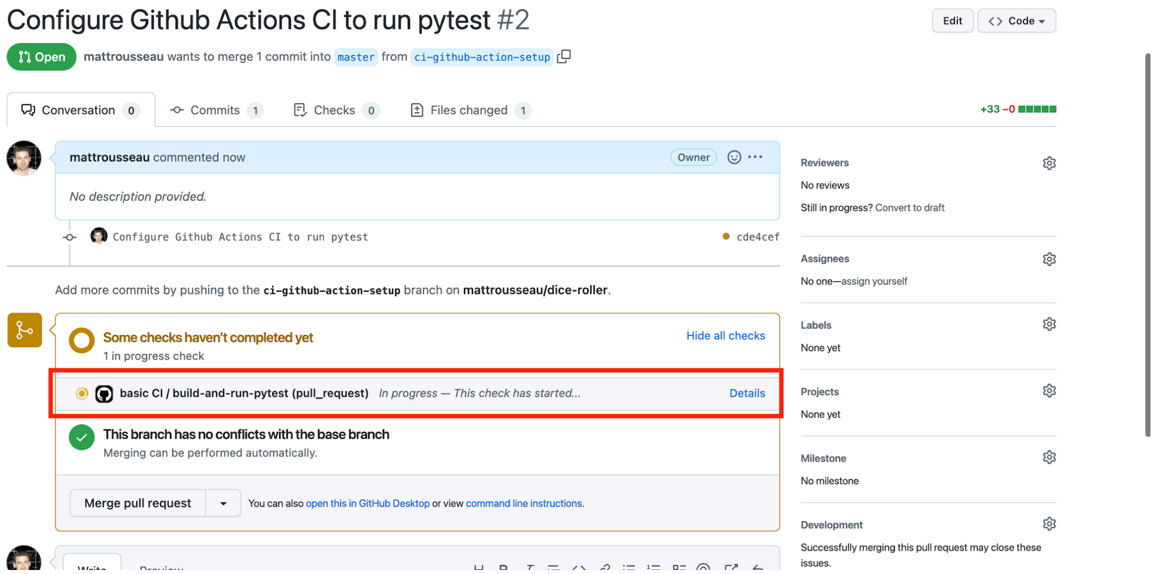
Hot tip: You can also use the Black formatter to format your code as part of the CI workflow!

```
- name: Format with Black
  run: black .
```

Just don't forget to add `black` to the `requirements.txt` file.

View on GitHub

Below the PR description and the list of commits, you will see GitHub Actions running the CI workflow:



Configure Github Actions CI to run pytest #2

Open mattrousseau wants to merge 1 commit into `master` from `ci-github-action-setup`

mattrousseau commented now

No description provided.

`Configure Github Actions CI to run pytest` cde4cef

Add more commits by pushing to the **`ci-github-action-setup`** branch on **mattrousseau/dice-roller**.

Some checks haven't completed yet
1 in progress check

basic CI / build-and-run-pytest (pull_request) *In progress — This check has started...* Details

This branch has no conflicts with the base branch
Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.

Reviewers
No reviews
Still in progress? Convert to draft

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

Development
Successfully merging this pull request may close these issues.

Wait a few seconds, and it should update the status 🎉

Advantages of GitHub Actions

- **Direct feedback** about the build status of the branch, right in GitHub's UI
- Someone pushing some code and forgetting to run the tests locally on their machine will be warned directly on GitHub that they broke the build

Adding tests to a repository and coupling GitHub Actions gives the developer **peace of mind** when adding code. It does so by exercising the whole test suite for every single commit!

2️⃣ CD - Continuous Deployment

> Continuous Deployment means automatically releasing a developer's changes from the repository to production, where it is usable by customers.

Continuous Deployment

- Enables **rapid and reliable delivery of new features** and updates to users by reducing the time and effort required to get code changes from development to production
- **Requires a high level of automation and testing** as well as close **collaboration** between developers and operations teams to ensure that code changes are deployed safely and reliably

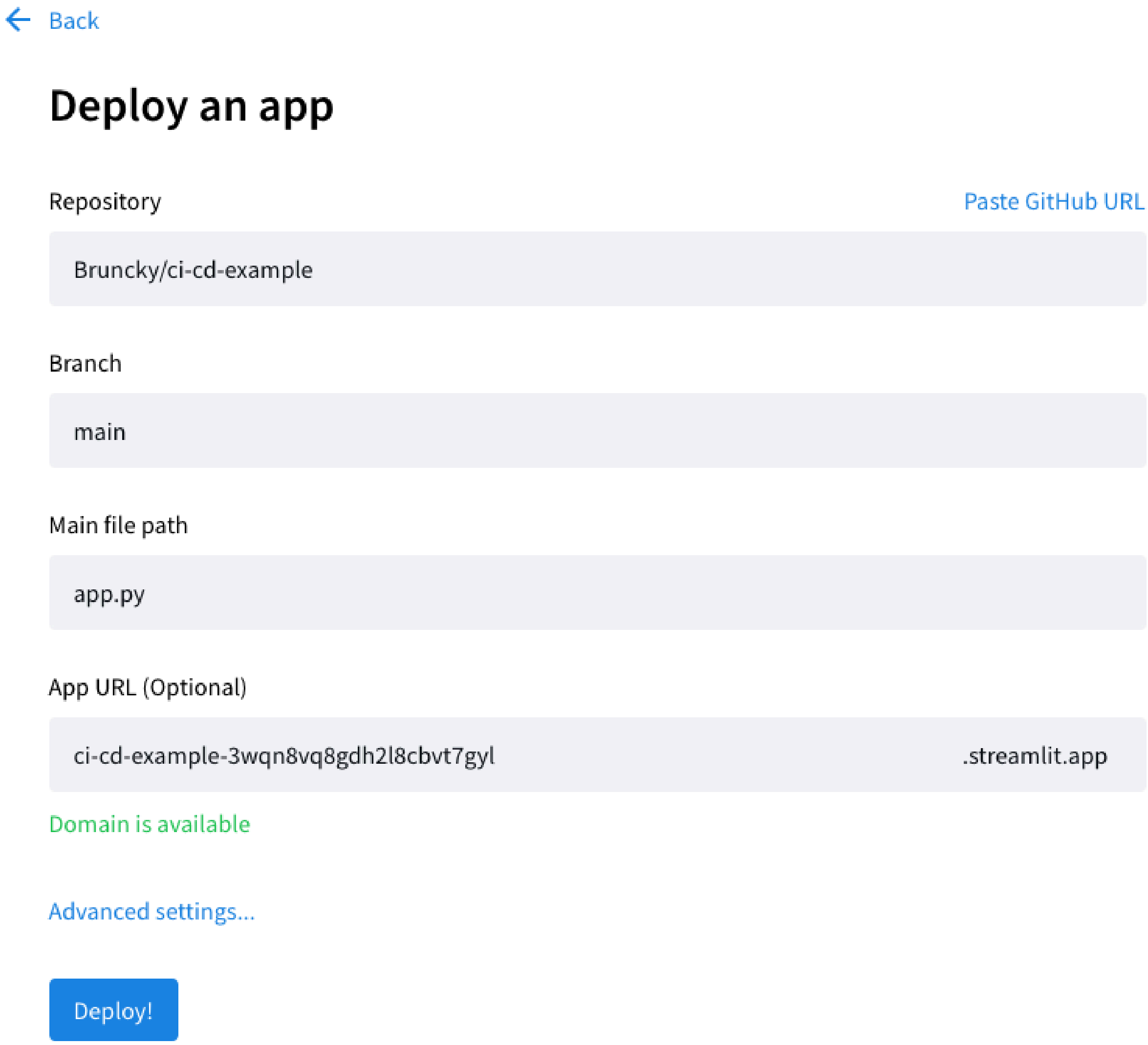
Continuous Deployment with Streamlit/GCP + GitHub Actions

Let's fork this repo so we can take a look at its content and then manipulate it.

The idea here is to learn how to take a simple **dockerized** API and connect it to either Streamlit's or GCP's CD tools.

Streamlit Cloud

Head over to Streamlit Cloud and log in **with GitHub** (the connection with GitHub is important). Once there, click on **"New app"** on the top right. On the next page, select the appropriate values according to your forked repo.



Deploy an app

Repository

Bruncky/ci-cd-example

Branch

main

Main file path

app.py

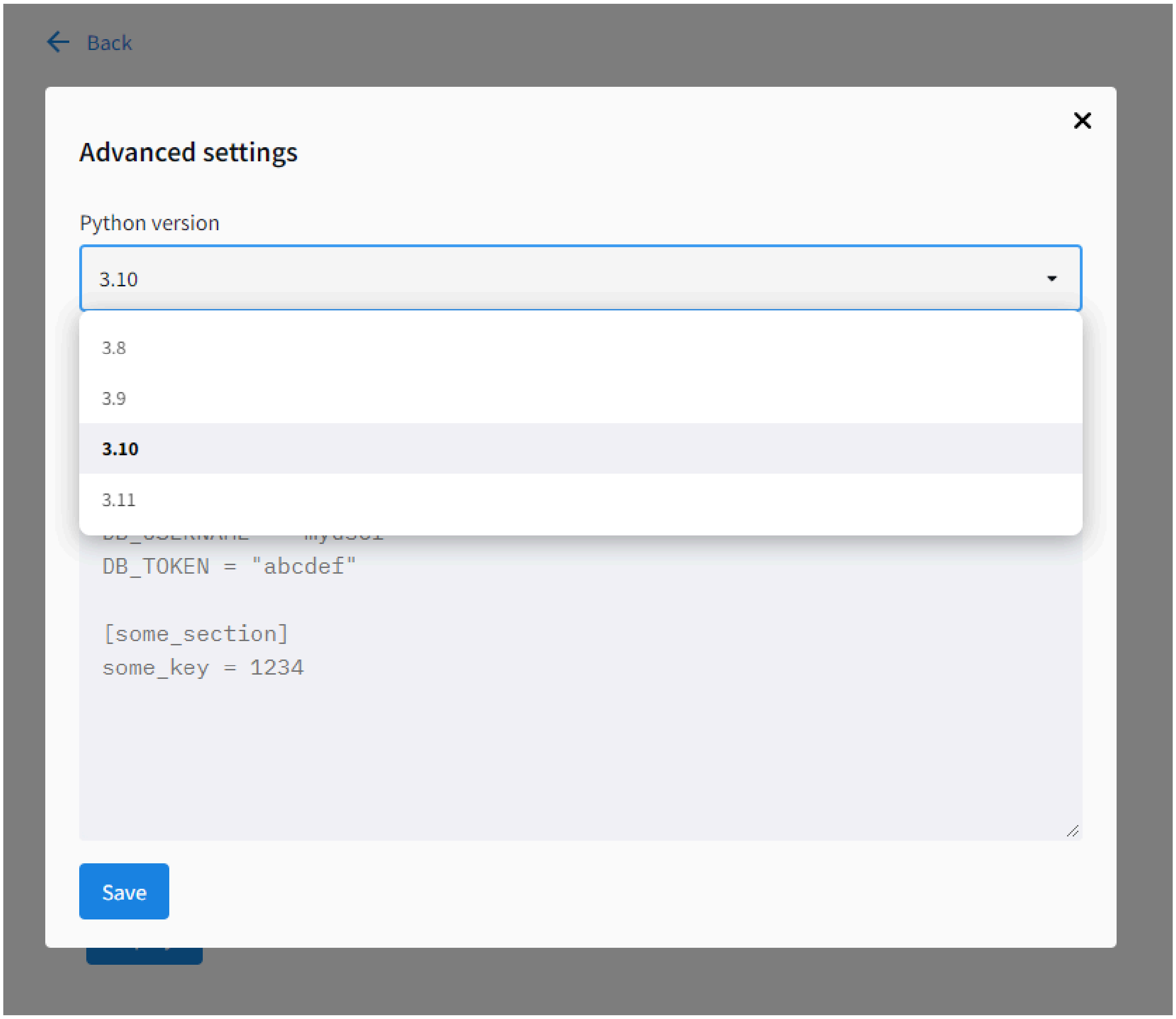
App URL (Optional)

ci-cd-example-3wqn8vq8gdh2l8cbvt7gyl .streamlit.app

Domain is available

- Select the correct repo
- Make sure that the "main file path" includes **subfolders**if relevant

Then, click on **"Advanced settings..."** and select Python 3.10.



Python version: 3.10

```
DB_USERNAME = "myuser"
DB_TOKEN = "abcdef"

[some_section]
some_key = 1234
```

Google Cloud Platform
Reminder from ML Ops

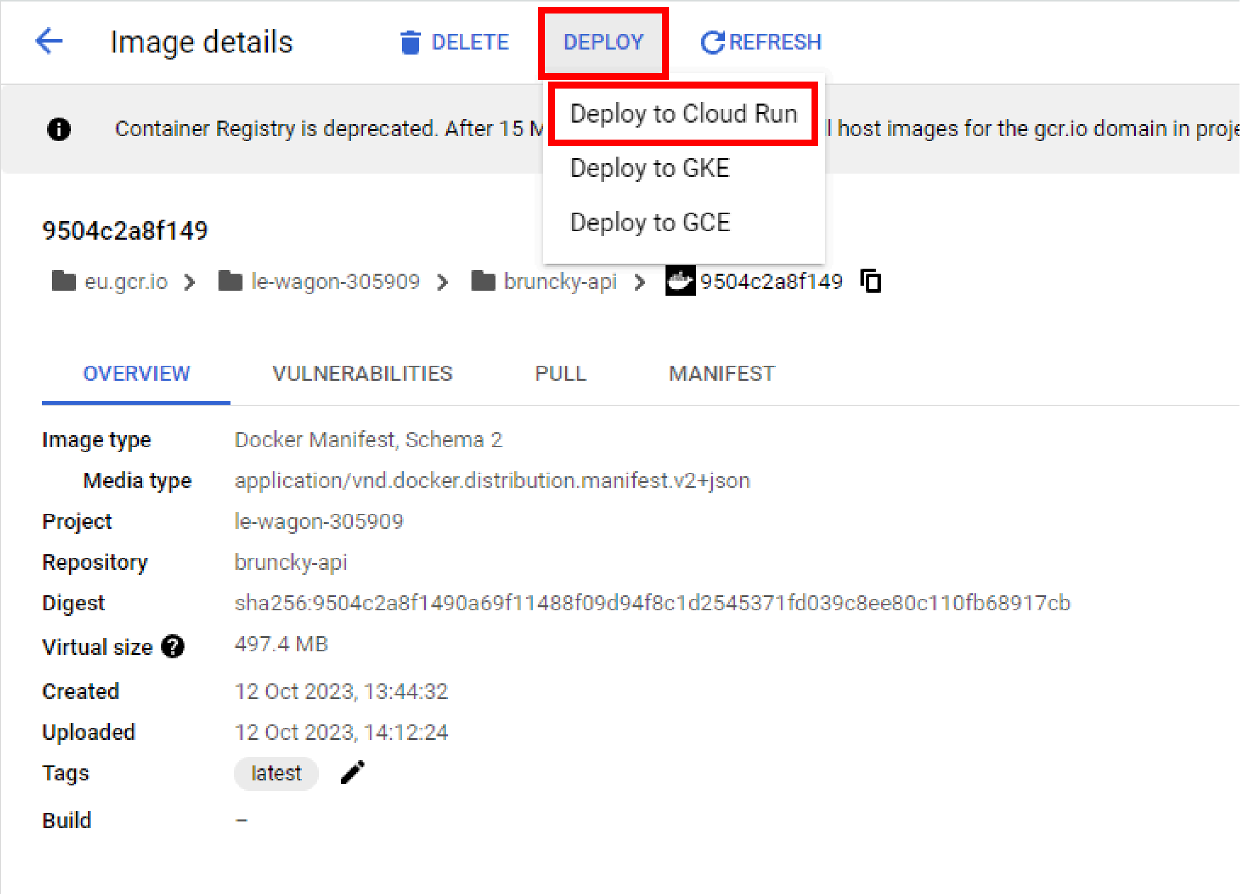
Don't forget to set the proper **environment variables** to make your job easier in the next steps. If you haven't already, make sure you've created your Google Artifact Registry Docker repo!

```
PROJECT="le-wagon-project"
IMAGE="image-name"
REGION="europe-west1"
DOCKER_REPO_NAME="my-docker-repo"
TAG="0.1"
IMAGE_URI=${REGION}-docker.pkg.dev/${PROJECT}/${DOCKER_REPO_NAME}/${IMAGE}:${TAG}
```

With that set, build and push the image from the repo you forked.

```
docker build -t $IMAGE_URI .
docker push $IMAGE_URI
```

Once the image has been pushed, go to GCP's **Artifact Registry** and find the image there. Click on it to inspect its details, and then click on **"Deploy > Deploy to Cloud Run"** at the top.

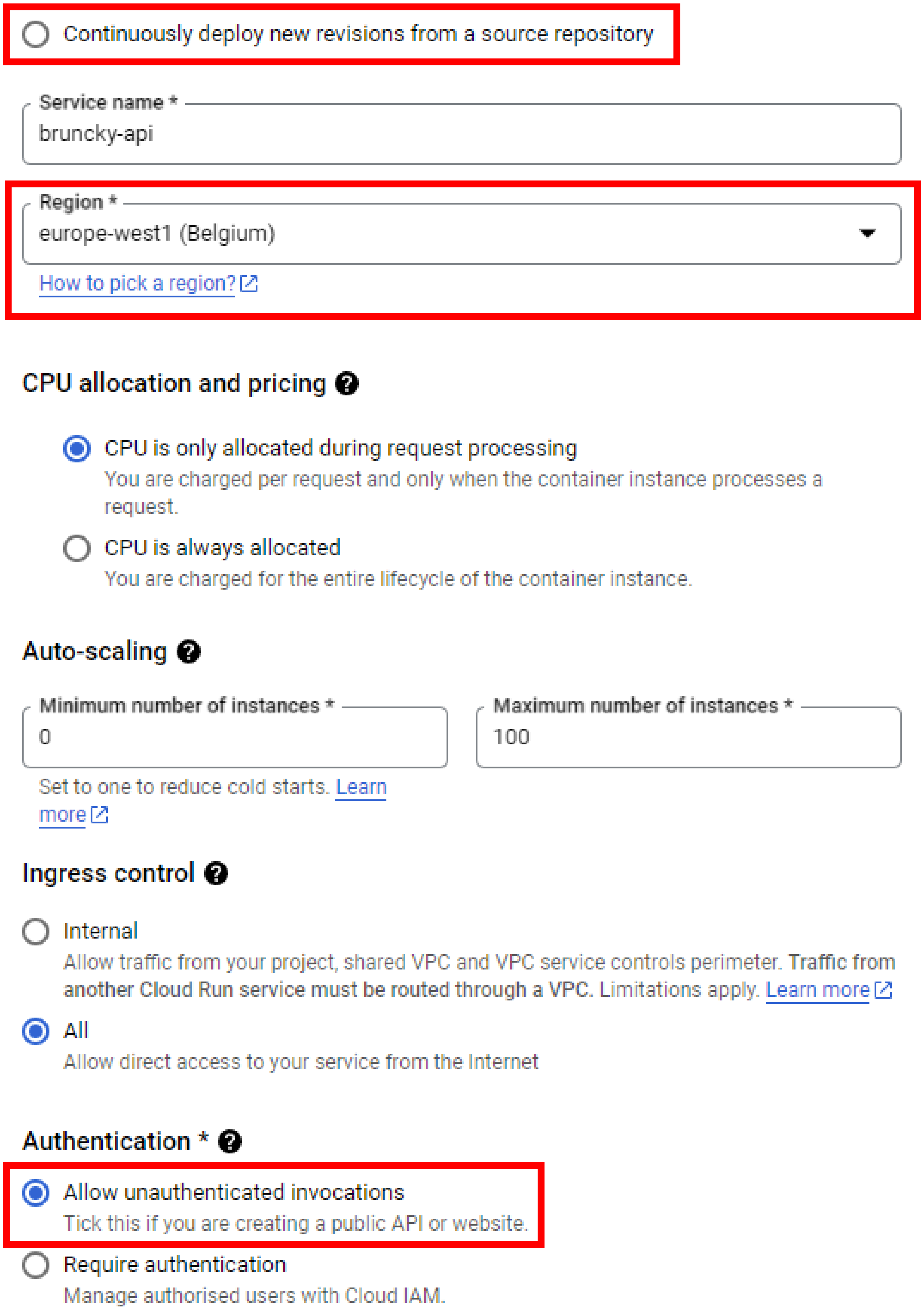


Container Registry is deprecated. After 15 M l host images for the gcr.io domain in proje

9504c2a8f149

Image type	Docker Manifest, Schema 2
Media type	application/vnd.docker.distribution.manifest.v2+json
Project	le-wagon-305909
Repository	bruncky-api
Digest	sha256:9504c2a8f1490a69f11488f09d94f8c1d2545371fd039c8ee80c110fb68917cb
Virtual size	497.4 MB
Created	12 Oct 2023, 13:44:32
Uploaded	12 Oct 2023, 14:12:24
Tags	latest
Build	–

Now, we need to select a few key options to ensure that the CD workflow works.



○ Continuously deploy new revisions from a source repository

Service name *
bruncky-api

Region *
europe-west1 (Belgium)

How to pick a region?

CPU allocation and pricing

● CPU is only allocated during request processing
You are charged per request and only when the container instance processes a request.

○ CPU is always allocated
You are charged for the entire lifecycle of the container instance.

Auto-scaling

Minimum number of instances *
0

Set to one to reduce cold starts. Learn more

Maximum number of instances *
100

Ingress control

○ Internal
Allow traffic from your project, shared VPC and VPC service controls perimeter. Traffic from another Cloud Run service must be routed through a VPC. Limitations apply. Learn more

● All
Allow direct access to your service from the Internet

Authentication *

● Allow unauthenticated invocations
Tick this if you are creating a public API or website.

○ Require authentication
Manage authorised users with Cloud IAM.

- Select **"Continuously deploy new revisions from a source repository"** and set up **Cloud Build** by following the steps
- Make sure that the region is set to `europe-west1 (Belgium)`

- Select **"Allow unauthenticated invocations"**

The **Docker Build** setup is fairly straightforward. It will guide you through installing the **Google Cloud Build**app on your repo, then all you need to do is select the correct repo and tell it to build from the Dockerfile in it.

When you're done, hit **Create** at the bottom!

Let's test it!

That’s it! But is it really working?

Let’s make a very simple change. We will add a timestamp to the response from the API.

```
git checkout -b add-timestamp
```

```
# fast.py
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from datetime import datetime

# [...]

@app.get('/')
def root():
    response = {
        'greeting': 'Servus, griaß di!',    # This is a typical Bavarian greeting ;)
        'timestamp': datetime.now()
    }

    return response
```

```
git add .
git commit -m "Small change to API to check CD"
git push origin add-timestamp
```

Then, open a **Pull Request** and merge to `master`/`main`.

After a while, refresh your GC Run app and check if your change is there! It's also a good idea to check the build process from GCP to know when it's done.

3️⃣ Benefits of CI/CD

- Speeds up time-to-market, increases speed of innovation and ability to compete
- Better product quality, reliability, and faster mean time to resolution
- Higher quality code and operations
- Less manual effort
- Reduces risk and makes rollbacks easier

🚀 Your turn!