**Lab5 : GitHub Oauth Setup configuration:**

**GitHub Token** – Added earlier so Backstage backend can fetch public/private repo metadata (e.g., PR lists) from GitHub.  
**GitHub OAuth** – Adding now in next activity so individual users can log in and see their **personalized GitHub data** (e.g., their PRs, review requests, activity).

**Install GitHub Auth in Backend (if not already done)**

yarn --cwd packages/backend add @backstage/plugin-auth-backend-module-github-provider

Important addition :

And add the following dependency to your backend index file:

(venv) root@ip-172-31-14-172:~/backstage-app/raman-idp-portal/packages/backend/src# vi index.ts :

backend.add(import(‘@backstage/plugin-auth-backend-module-github-provider’));

**https://backstage.io/docs/auth/**

**2.1: Create GitHub Oauth App**

Go to: [**https://github.com/settings/developers**](https://github.com/settings/developers) → Oauth Apps → “New Oauth App”

| **Field** | **Value** |
| --- | --- |
| App Name | Backstage Raman |
| Homepage URL | <http://PubIP> :3000 |
| Callback URL | [http:// PubIP](http://localhost) :7007/api/auth/github/handler/frame |

After creating it, copy the **Client ID** and **Client Secret**

**2.2: Add Credentials to app-config.yaml**

In ~/backstage-app/raman-app/app-config.yaml:

yaml

CopyEdit

auth:

providers:

github:

development:

clientId: ${AUTH\_GITHUB\_CLIENT\_ID}

clientSecret: ${AUTH\_GITHUB\_CLIENT\_SECRET}

Now add these in your or .env:

AUTH\_GITHUB\_CLIENT\_ID=your-client-id

AUTH\_GITHUB\_CLIENT\_SECRET=your-client-secret

**LOAD .env file in the backstage env :**

-- Run this in your repo root (where package.json is):

yarn workspace backend add dotenv

-- Load .env in backend

-- In packages/backend/src/index.ts, add this at the very top:

- vi packages/backend/src/index.ts

import path from 'path';

import dotenv from 'dotenv';

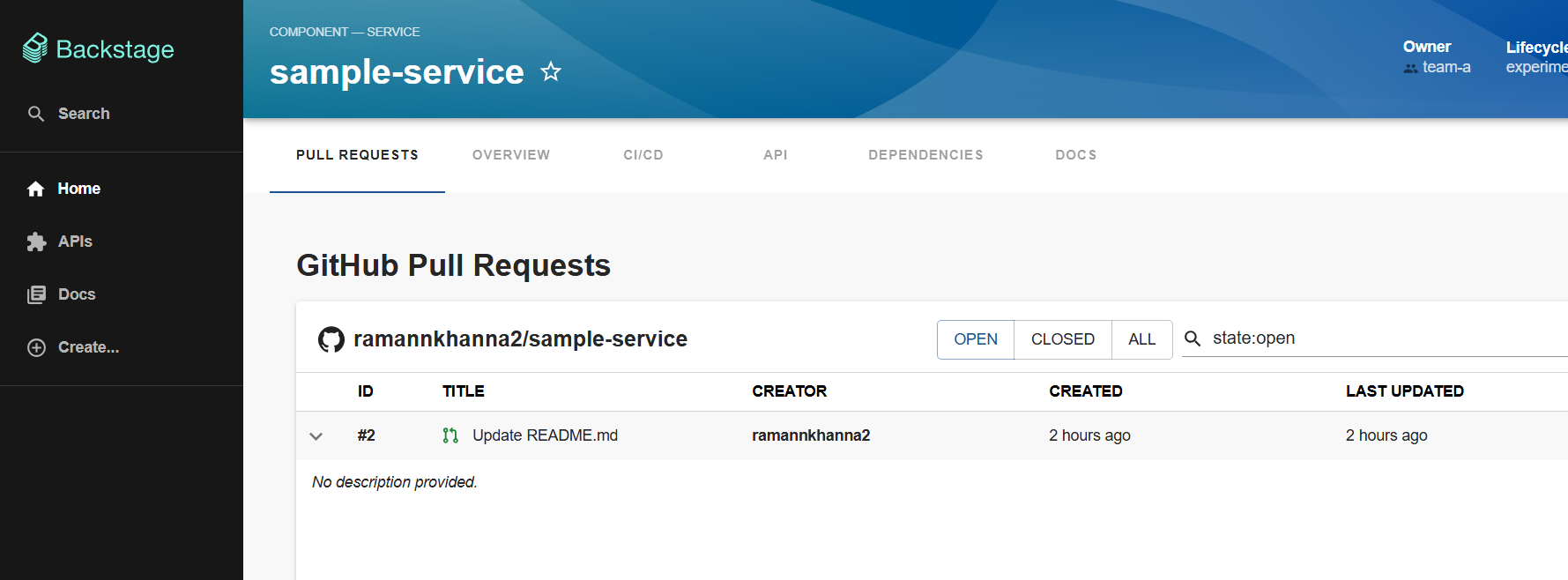
// Load environment variables from root .env file

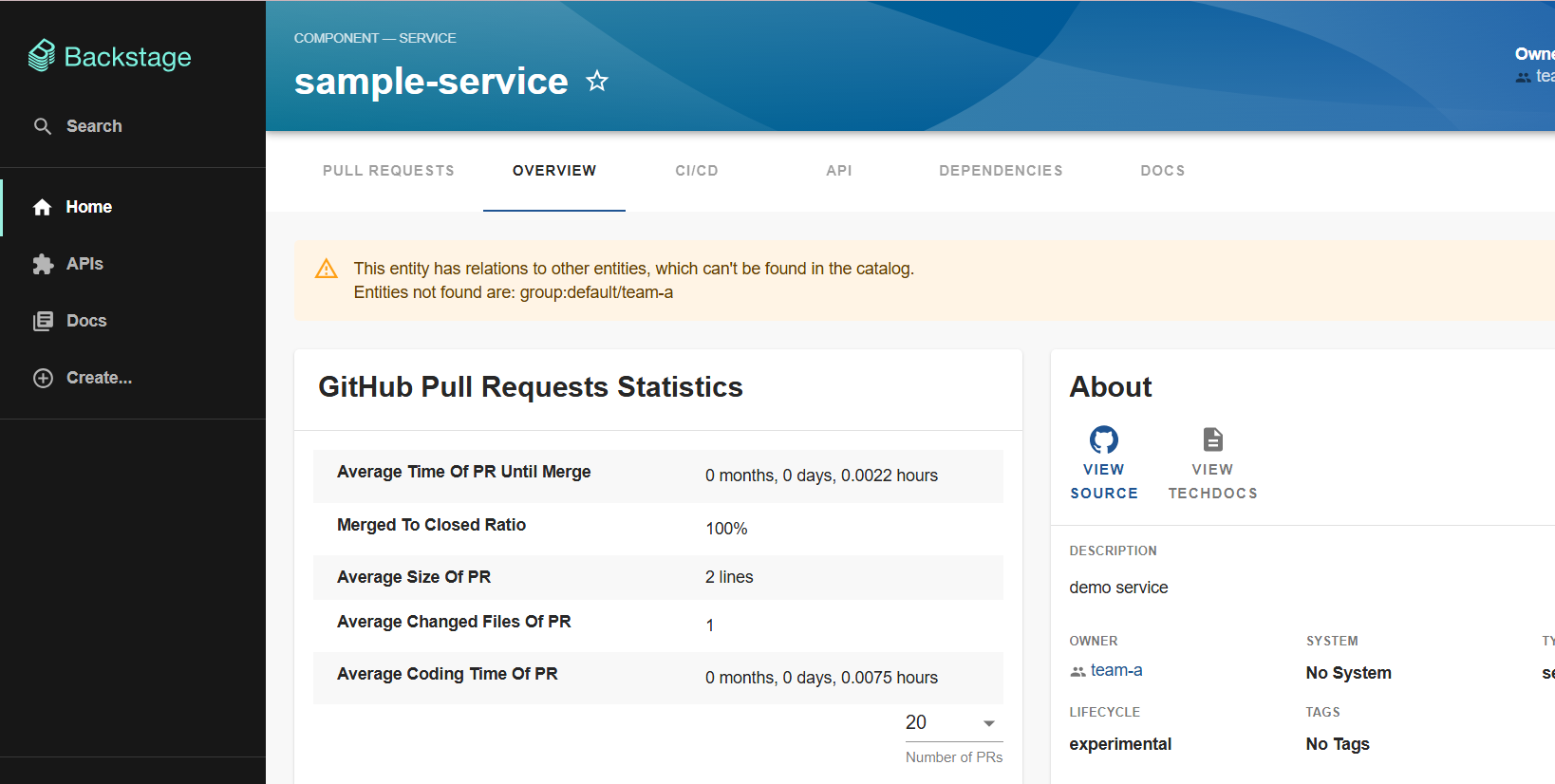
dotenv.config({

path: path.resolve(\_\_dirname, '../../../.env'),

});

* + - Yarn start again





**Lab 6 – Create & Register a Golden Path Flask Service Template in Backstage**

**🎯 Objective**

Build and register a **Golden Path Backstage Software Template** that scaffolds a standardized **Flask microservice** with:

* **Flask API** + /metrics endpoint (Prometheus)
* **Dockerfile** for container builds
* **requirements.txt** for dependencies
* **GitHub Actions CI/CD placeholder**
* **catalog-info.yaml** for auto‑registration in Backstage Catalog

**🧠 Why This Matters**

* **Developers** can create ready‑to‑run, container‑ready Flask services in minutes.
* **Platform team** ensures all new services follow **standardized patterns**.
* **Future labs** (Day 3/Day 4) can build on this for CI/CD, Infra, and Observability.

**📋 Prerequisites**

1. **Day 1** Backstage portal running & accessible.
2. **GitHub Integration** configured in Backstage (app-config.yaml with PAT).
3. **GitHub OAuth** working for repo creation.
4. A GitHub organization where the template repo will live.

**🛠 Step‑By‑Step Guide**

**Step 1 – Create Template Repository**

1. In GitHub:
   * Click **New Repository** → Name:

cpp

CopyEdit

flask-goldenpath-template

* + Visibility: **Public** (easier for workshop)  
    *(In production, use private repos for internal templates.)*
  + Initialize with **README**.

1. Clone the repo locally:

bash

CopyEdit

git clone https://github.com/<org>/flask-goldenpath-template.git

cd flask-goldenpath-template

**Step 2 – Create Folder Structure**

Inside flask-goldenpath-template/:

pgsql

CopyEdit

flask-goldenpath-template/

│── template.yaml # Backstage software template definition

│── skeleton/ # Template source files

│ ├── app.py

│ ├── requirements.txt

│ ├── Dockerfile

│ ├── catalog-info.yaml

│ └── .github/

│ └── workflows/

│ └── ci.yml

**Step 3 – Add Flask Service Code**

📄 **skeleton/app.py**

from flask import Flask

from prometheus\_client import Counter, generate\_latest

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

request\_count = Counter('request\_count', 'Total request count')

@app.route('/')

def home():

request\_count.inc()

return "Hello from ${{ values.name }} service!"

@app.route('/metrics')

def metrics():

return generate\_latest(), 200

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000)

📄 **skeleton/requirements.txt**

flask

prometheus\_client

📄 **skeleton/Dockerfile**

dockerfile

CopyEdit

FROM python:3.10-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY . .

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

📄 **skeleton/.github/workflows/ci.yml**

yaml

CopyEdit

name: CI Pipeline

on:

push:

branches: [ main ]

pull\_request:

branches: [ main ]

jobs:

build:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- name: Set up Python

uses: actions/setup-python@v4

with:

python-version: '3.10'

- name: Install dependencies

run: pip install -r requirements.txt

- name: Run tests

run: echo "No tests yet"

📄 **skeleton/catalog-info.yaml**

apiVersion: backstage.io/v1alpha1

kind: Component

metadata:

name: ${{ values.name }}

description: ${{ values.description }}

annotations:

github.com/project-slug: ${{ values.destinationOwner }}/${{ values.destinationRepo }}

spec:

type: service

owner: ${{ values.owner }}

lifecycle: experimental

**Step 4 – Create Backstage Template Definition**

📄 **template.yaml**

apiVersion: scaffolder.backstage.io/v1beta3

kind: Template

metadata:

  name: goldenpath-flask-service

  title: Golden Path – Flask Service

  description: Scaffold a standardized Flask microservice with Docker, metrics, and CI/CD.

spec:

  owner: development

  type: service

  parameters:

    - title: Service Information

      required: [name, owner, description]

      properties:

        name:

          type: string

          title: Service Name

          description: Unique name for your service

        owner:

          type: string

          title: Owner

          description: Owner team or group

        description:

          type: string

          title: Description

          description: Short description of the service

    - title: Repository Location

      required: [repo]

      properties:

        repo:

          type: string

          title: Repository Name

          description: Name of the new GitHub repository

        repoVisibility:

          type: string

          title: Repository Visibility

          enum: [public, private]

          default: public

  steps:

    - id: fetch

      name: Fetch Template Source

      action: fetch:template

      input:

        url: ./skeleton

        values:

          name: ${{ parameters.name }}

          owner: ${{ parameters.owner }}

          description: ${{ parameters.description }}

          destination:

            repo: ${{ parameters.repo }}

            owner: ramannkhanna2

    - id: wait

      name: Wait for GitHub to index repo

      action: debug:wait

      input:

        seconds: 5

    - id: publish

      name: Publish to GitHub

      action: publish:github

      input:

        #repoUrl: github.com?repo=${{ parameters.repo }}&owner=ramannkhanna2

        repoUrl: github.com?owner=ramannkhanna2&repo=${{ parameters.repo }}

        repoVisibility: ${{ parameters.repoVisibility }}

  output:

    links:

      - title: Repository

        url: ${{ steps['publish'].output.remoteUrl }}

**Replace ramankhanna2 with YOUR\_GITHUB\_ORG** with your actual GitHub org name.

**Step 5 – Commit & Push**

bash

CopyEdit

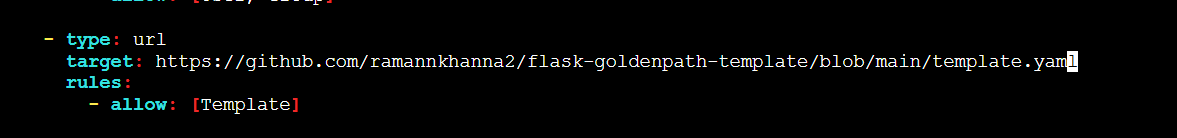
git add .

git commit -m "Golden Path Flask Service Template"

git push origin main

**Step 6 – Register Template in Backstage now not from ui but Static Location Configuration**

1. Go to app-config.yaml
2. **Add a template under catalog section**



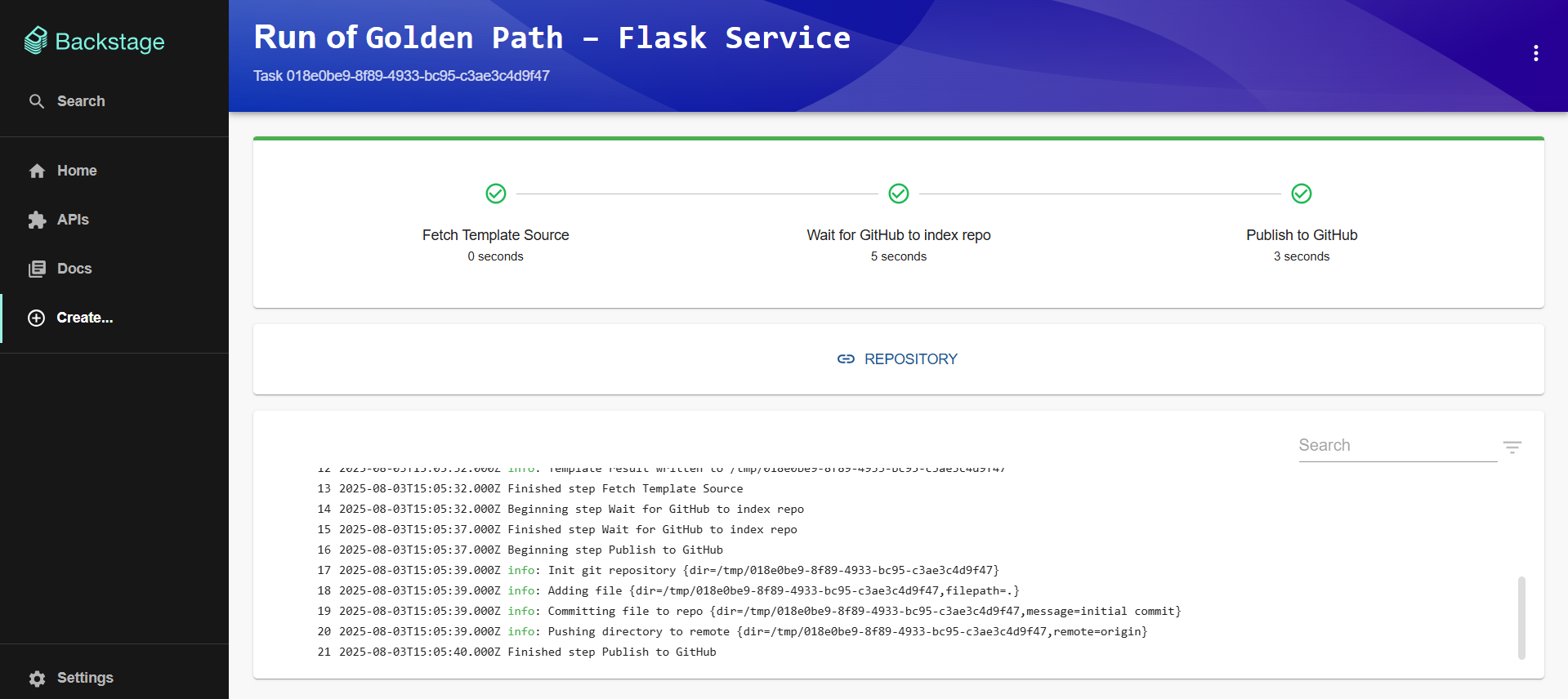
1. Save it
2. Do yarn start agaian

**Step 7 – Test Template**

1. In Backstage **Create…** menu → select **Golden Path – Flask Service**.
2. Fill parameters:
   * **Service Name:** customer-orders-service
   * **Owner:** team-a
   * **Description:** REST API for customer orders
   * **Repository Name:** customer-orders-service
3. Click **Create** → Backstage will:
   * Scaffold code from template
   * Create new GitHub repo
   * Push files

**🔍 Verification**

✅ Template visible in Create menu.  
✅ Selecting it creates a new GitHub repo.  
✅ Repo contains Flask app, Dockerfile, metrics, CI/CD, catalog-info.yaml.  
✅ Service appears in Backstage Catalog.



**ANOTHER TEMPLATE TO RESEARCH FURTHER :**

**https://github.com/ricardoandre97/backstage-software-templates**

**Lab 7 – Test the Golden Path Flask Service Locally with Docker**

**🎯 Objective**

In this lab, you will take the **Flask microservice** scaffolded from your **Golden Path Backstage Template** in **Lab 6**, build a Docker image, run it locally, and validate that:

* The application responds to HTTP requests.
* The /metrics endpoint correctly exposes **Prometheus** metrics.

This ensures the service is production-ready from a containerization standpoint.

**🧠 Why This Matters**

* Confirms that **generated code from Backstage** is **runnable without modification**.
* Validates the **Dockerfile** and **requirements.txt** are correct.
* Ensures **metrics instrumentation** is functional for observability.
* Catches build/runtime issues **before** pushing to a container registry or deploying to Kubernetes.

**📋 Prerequisites**

Before starting, ensure:

1. **Lab 6** is complete:
   * Service scaffolded from Backstage **Golden Path Template**.
   * Code is available locally.
2. **Docker** is installed and running:
   * Test with:

sudo usermod -aG docker ubuntu

newgrp docker

docker --version

1. Your scaffolded service directory contains:

app.py

requirements.txt

Dockerfile

catalog-info.yaml

.github/workflows/ci.yml

**📂 Lab Context**

From Lab 6, your app.py should look similar to:

python

CopyEdit

from flask import Flask

from prometheus\_client import Counter, generate\_latest

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

request\_count = Counter('request\_count', 'Total request count')

@app.route('/')

def home():

request\_count.inc()

return "Hello from {{ values.name }} service!"

@app.route('/metrics')

def metrics():

return generate\_latest(), 200

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000)

💡 {{ values.name }} was replaced during scaffolding with your chosen service name in Lab 6.

**🛠 Step-by-Step Guide**

**Step 1 – Navigate to Your Service Directory**

bash

CopyEdit

cd ~/customer-orders-service

*(Replace customer-orders-service with the folder name generated in Lab 6.)*

**Step 2 – Review Dockerfile**

Check that your Dockerfile matches:

dockerfile

CopyEdit

FROM python:3.10-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY . .

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

📌 **Key Points**

* python:3.10-slim keeps the image small.
* All dependencies are installed from requirements.txt.
* Application starts with:

python app.py

**Step 3 – Build Docker Image**

bash

CopyEdit

docker build -t my-flask-service .

📌 Replace my-flask-service with your preferred image name.

✅ **Expected Output (excerpt)**:

Successfully built <IMAGE\_ID>

Successfully tagged my-flask-service:latest

**Step 4 – Run Container**

bash

CopyEdit

docker run -d --name flaskcon -p 5000:5000 my-flask-service

📌 **Explanation**:

* -d → Run in detached mode.
* --name flaskcon → Name the container.
* -p 5000:5000 → Map container port 5000 to host port 5000.

✅ **Expected Output**:

<CONTAINER\_ID>

**Step 5 – Verify API Endpoint**

bash

CopyEdit

curl http://PubIP:5000/

✅ **Expected Output**:

csharp

CopyEdit

Hello from customer-orders-service service!

*(The service name will match what you entered in Lab 6.)*

**Step 6 – Verify Metrics Endpoint**

bash

CopyEdit

curl http://PubIP:5000/metrics

✅ **Expected Output (excerpt)**:

nginx

CopyEdit

# HELP request\_count Total request count

# TYPE request\_count counter

request\_count 1.0

📌 This confirms **Prometheus metrics** are available.

**Step 7 – View Running Container**

bash

CopyEdit

docker ps

✅ **Expected Output**:

nginx

CopyEdit

CONTAINER ID IMAGE COMMAND STATUS PORTS NAMES

abc123456789 my-flask-service "python app.py" Up 2 minutes 0.0.0.0:5000->5000/tcp flaskcon

**Step 8 – Stop & Remove Container**

bash

CopyEdit

docker stop flaskcon

docker rm flaskcon

**🔍 Verification Checklist**

✅ Docker image builds without errors.  
✅ API endpoint / responds with correct service name.  
✅ /metrics returns Prometheus metrics.  
✅ Container stops/starts cleanly.