**Lab 9: Set up Backstage portal**

**nodejs installation**

We will be installing nodejs to run backstage and we will be installing nodejs version 18  
Why version 18?  
Because due to some core difference in nodejs later versions so we will need to install some other packages (build-essential) too if we want to create the backstage  
so lets start

**Step 1**

becoming root

sudo -i

**Step 2**

updating package manager

apt update -y

**Step 3**

installing nodejs

curl -fsSL https://deb.nodesource.com/setup\_18.x -o nodesource\_setup.sh

bash nodesource\_setup.sh

apt-get install -y nodejs

**Step 4**

enabling corepack

npm install -g corepack

**Step 5**

installing docker official website : <https://docs.docker.com/engine/install/ubuntu/>

# Add Docker's official GPG key:

sudo apt-get install ca-certificates curl

sudo install -m 0755 -d /etc/apt/keyrings

sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc

sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:

echo \

"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \

$(. /etc/os-release && echo "${UBUNTU\_CODENAME:-$VERSION\_CODENAME}") stable" | \

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

sudo apt-get -y install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

**Step 2**

now lets create a folder in this user

mkdir backstage-app && cd backstage-app

**Step 3**

run this command to create our own backstage app

npx @backstage/create-app@latest

after this command it will ask for confirmation and the name of our app type y for confirmation and after that input the name of your app that we want to give

**Step 4**

get into your app that you have created

cd your-app-name

**Step 5**

now lets start our backstage app

yarn start

you will be able to access backstage base app on your localhost:3000 in your web browser

**step 6**

if you are working with a cloud vm then you will have to make some changes in app.config.yaml file

vi app.config.yaml

do these changes  
-- under app:

* in baseurl change localhost to your vms public ip
* under base url add this block

listen:

host: 0.0.0.0

-- under backend:

* same as in app in here too change baseurl localhost --> public ip of vm
* do the same for cors block change origin localhost --> public ip of vm
* under baseurl in listen uncomment host and change it to 0.0.0.0
* the whole file should look something like this

app:

title: Scaffolded Backstage App

baseUrl: http://40.192.33.246:3000

listen:

host: 0.0.0.0

organization:

name: My Company

backend:

# Used for enabling authentication, secret is shared by all backend plugins

# See https://backstage.io/docs/auth/service-to-service-auth for

# information on the format

# auth:

# keys:

# - secret: ${BACKEND\_SECRET}

baseUrl: http://40.192.33.246:7007

listen:

port: 7007

# Uncomment the following host directive to bind to specific interfaces

host: 0.0.0.0

csp:

connect-src: ["'self'", 'http:', 'https:']

# Content-Security-Policy directives follow the Helmet format: https://helmetjs.github.io/#reference

# Default Helmet Content-Security-Policy values can be removed by setting the key to false

cors:

origin: http://40.192.33.246:3000

methods: [GET, HEAD, PATCH, POST, PUT, DELETE]

credentials: true

# This is for local development only, it is not recommended to use this in production

# The production database configuration is stored in app-config.production.yaml

database:

client: better-sqlite3

connection: ':memory:'

# workingDirectory: /tmp # Use this to configure a working directory for the scaffolder, defaults to the OS temp-dir

* yarn start

now access backstage in the web browser at public-ip:3000

**Lab 10: Register Your Microservice in Backstage**

**🎯 Goal:**

Add your service (raman-micro) to Backstage’s Software Catalog so it's visible in the portal.

**🧩 1.1 — Clone Your Repo (Optional, if you want to work locally)**

bash

CopyEdit

git clone https://github.com/ramannkhanna2/raman-micro.git

cd raman-micro

**🧩 1.2 — Add catalog-info.yaml**

Create a file named catalog-info.yaml in the root of your repo. This YAML file tells Backstage about the service — its name, owner, type, GitHub location, etc.

📄 **File**: raman-micro/catalog-info.yaml

yaml

CopyEdit

apiVersion: backstage.io/v1alpha1

kind: Component

metadata:

name: raman-micro

description: A Flask microservice with CI/CD and observability

annotations:

github.com/project-slug: ramannkhanna2/raman-micro

backstage.io/techdocs-ref: dir:.

prometheus.io/scrape: "true"

loki.io/logs: "true"

spec:

type: service

lifecycle: experimental

owner: platform-team

**Explanation of fields**:

| **Field** | **Description** |
| --- | --- |
| name | Display name in Backstage |
| annotations.github.com/project-slug | Enables GitHub integration |
| techdocs-ref | For TechDocs, we’ll use this later |
| prometheus.io/scrape | Lets Backstage know this service emits metrics |
| loki.io/logs | Indicates the service emits logs |
| owner | Should match a group/user in Backstage (platform-team for now) |

**🧩 1.3 — Commit & Push to GitHub**

bash

CopyEdit

git add catalog-info.yaml

git commit -m "Add Backstage catalog metadata"

git push origin main

✅ After this step, the metadata will be available at:

bash

CopyEdit

https://github.com/ramannkhanna2/raman-micro/blob/main/catalog-info.yaml

**🧩 1.4 — Import Service into Backstage UI**

1. Open your Backstage app:  
   🌐 http://localhost:3000 (default)
2. In the sidebar, go to:  
   **Catalog → Register Existing Component**
3. When prompted for the YAML file URL, paste:

bash

CopyEdit

https://github.com/ramannkhanna2/raman-micro/blob/main/catalog-info.yaml

1. Click ✅ **Analyze**. Backstage will show metadata preview.
2. Click ➕ **Import** to add it to the catalog.

**🧩 1.5 — Verify Service Is Listed**

* Go to **Catalog → Components**
* You should now see: raman-micro
* Click it to view:
  + Description
  + Metadata
  + Owner
  + GitHub repo link
  + Placeholder for TechDocs

✅ **Success**: You have now added your microservice to the Backstage UI and registered it in the Software Catalog.

🎯 **Goal**: Add an organizational group (platform-team) and a user (raman) to the Backstage catalog.

**📁 Directory Structure Needed**

You only need two YAML files:

sql

CopyEdit

raman\_micro\_test/

└── backstage\_entities/

├── group-platform-team.yaml ✅ Group

└── user-raman.yaml ✅ User

**1️⃣ Create Group YAML**

📄 backstage\_entities/group-platform-team.yaml

yaml

CopyEdit

apiVersion: backstage.io/v1alpha1

kind: Group

metadata:

name: platform-team

description: Team owning microservices and platform infra

spec:

type: team

members:

- user:raman

**2️⃣ Create User YAML**

📄 backstage\_entities/user-raman.yaml

yaml

CopyEdit

apiVersion: backstage.io/v1alpha1

kind: User

metadata:

name: raman

spec:

profile:

displayName: Raman

email: raman@example.com

memberOf:

- group:platform-team

**3️⃣ Commit and Push to GitHub**

bash

CopyEdit

git add backstage\_entities/user-raman.yaml backstage\_entities/group-platform-team.yaml

git commit -m "Add Backstage user and group entities"

git push

Make sure your files are accessible via **raw GitHub URLs**, like:

* ✅ Group
* ✅ User

**4️⃣ Import Each Entity in Backstage UI**

**🔹 Step-by-step:**

1. Go to your Backstage app:

bash

CopyEdit

http://localhost:3000/catalog-import

1. Paste the **Group URL**:

bash

CopyEdit

https://raw.githubusercontent.com/ramannkhanna2/raman\_micro\_test/main/backstage\_entities/group-platform-team.yaml

👉 Click ✅ **Analyze** → ➕ **Import**

1. Now import the **User URL**:

bash

CopyEdit

https://raw.githubusercontent.com/ramannkhanna2/raman\_micro\_test/main/backstage\_entities/user-raman.yaml

👉 Again, ✅ **Analyze** → ➕ **Import**

**✅ 5️⃣ Verify in Backstage**

* Go to **Catalog → Groups** → You should see platform-team
* Go to **Catalog → Users** → You should see raman
* Click on platform-team → it will show that raman is a member
* Click on raman → it will show membership in platform-team

**📝 Optional: Add Location File to Import Both at Once**

If you want to automate importing both in one go:

📄 backstage\_entities/locations.yaml

yaml

CopyEdit

apiVersion: backstage.io/v1alpha1

kind: Location

metadata:

name: raman-org-entities

description: User and Group for the platform org

spec:

type: url

targets:

- https://raw.githubusercontent.com/ramannkhanna2/raman\_micro\_test/main/backstage\_entities/group-platform-team.yaml

- https://raw.githubusercontent.com/ramannkhanna2/raman\_micro\_test/main/backstage\_entities/user-raman.yaml

Then import this once:

bash

CopyEdit

https://raw.githubusercontent.com/ramannkhanna2/raman\_micro\_test/main/backstage\_entities/locations.yaml

**Lab 11: Adding Docs with TechDocs (Backstage-native Docs)**

📌 With TechDocs, developers can view documentation for any service **directly inside the Backstage UI**, powered by [MkDocs](https://www.mkdocs.org/) and Markdown.

**🧠 What We’ll Do**

| **Task** | **Description** |
| --- | --- |
| 1️⃣ | Add mkdocs.yml and docs/ folder to your repo |
| 2️⃣ | Add basic Markdown documentation |
| 3️⃣ | Ensure your catalog-info.yaml includes the TechDocs annotation |
| 4️⃣ | Configure Backstage to generate and serve docs |
| 5️⃣ | Verify in UI via **View TechDocs** |

**📁 Step 2.1: Add Docs Folder + mkdocs.yml to Your Repo**

In your raman-micro GitHub repo, create the following structure:

pgsql

CopyEdit

raman-micro/

├── catalog-info.yaml

├── mkdocs.yml

└── docs/

└── index.md

**📝 mkdocs.yml**

Create this at the root (raman-micro/mkdocs.yml):

yaml

CopyEdit

site\_name: raman-micro

nav:

- Home: index.md

plugins:

- techdocs-core

**📄 docs/index.md**

Create this at raman-micro/docs/index.md:

markdown

CopyEdit

# Raman Microservice

Welcome to the \*\*raman-micro\*\* documentation.

## 🔧 Features

- Flask-based microservice

- CI/CD via GitHub Actions

- SAST, DAST, SBOM integrated

- Dockerized and Helm-compatible

- Exposes Prometheus metrics

## 🚀 Deployment

Deployed via GitHub Actions workflow.

## 🧪 Observability

Integrated with:

- Prometheus for metrics

- Loki/ELK for logs

- Grafana dashboards

## 👤 Owner

Maintained by \*\*Raman Khanna\*\*.

**✅ Step 2.2: Ensure catalog-info.yaml Has TechDocs Annotation**

It should already have this line (you added it in Step 1):

yaml

CopyEdit

annotations:

backstage.io/techdocs-ref: dir:.

✅ If it’s already there, no change needed. If not, add it.

**🔁 Step 2.3: Commit & Push to GitHub**

Run this from the root of your local raman-micro clone:

bash

CopyEdit

git add mkdocs.yml docs/

git commit -m "Add TechDocs documentation"

git push

**✅ Step 2.5: Open Backstage and View TechDocs**

1. Go to **Catalog > raman-micro**
2. Click **“View TechDocs”** on the top-right

You should now see your rendered index.md contents 🎉

**🟢 Summary**

✅ You now have:

* Markdown docs stored in GitHub
* Rendered natively in Backstage via TechDocs

**Lab 12 : Add Prometheus Metrics & Grafana Dashboards to raman-micro**

**🎯 Goal**

Let’s instrument your raman-micro Flask app to expose metrics in Prometheus format and visualize them in Grafana dashboards — all **linked inside Backstage**.

**🔍 What You’ll Get**

✅ A working /metrics endpoint in Flask  
✅ Prometheus scraping this endpoint  
✅ Grafana showing service metrics  
✅ Dashboard link inside Backstage UI

**📁 Overview of Steps**

| **Step** | **Task** |
| --- | --- |
| 3.1 | Add Prometheus client to Flask |
| 3.2 | Expose metrics on /metrics |
| 3.3 | Add Prometheus config to scrape the service |
| 3.4 | Run Prometheus |
| 3.5 | Add Grafana with dashboard |
| 3.6 | Link Grafana dashboard in Backstage |

**✅ Step 3.1: Add Prometheus Client to Flask**

**📦 1. Install Prometheus Python client**

In your repo (raman-micro):

bash

CopyEdit

pip install prometheus\_client

Also add it to your requirements.txt:

nginx

CopyEdit

prometheus\_client

**🧠 2. Update Flask app**

In app.py or your main Flask file:

python

CopyEdit

from flask import Flask

from prometheus\_client import start\_http\_server, Counter, Histogram, generate\_latest

from prometheus\_client import CONTENT\_TYPE\_LATEST

from flask import Response

import time

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

# Custom metrics

REQUEST\_COUNT = Counter("request\_count", "Total number of requests")

REQUEST\_LATENCY = Histogram("request\_latency\_seconds", "Request latency")

@app.route("/")

def home():

REQUEST\_COUNT.inc()

with REQUEST\_LATENCY.time():

time.sleep(0.1) # simulate processing

return "Hello from Raman Micro!"

# Expose Prometheus metrics at /metrics

@app.route("/metrics")

def metrics():

return Response(generate\_latest(), mimetype=CONTENT\_TYPE\_LATEST)

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

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

✅ Now your service exposes Prometheus-compatible metrics at http://localhost:5000/metrics

**✅ Step 3.2: Run and Test Metrics Endpoint**

bash

CopyEdit

python app.py

In browser or curl:

bash

CopyEdit

curl http://localhost:5000/metrics

You should see output like:

nginx

CopyEdit

# HELP request\_count Total number of requests

# TYPE request\_count counter

request\_count 2.0

...

**🔁 You Need prometheus.yml Because...**

**🧠 prometheus.yml is the configuration for the Prometheus server, which is the metrics collector/scraper.**

This file tells Prometheus:

* What services to scrape (targets)
* How often (scrape\_interval)
* Under what name (job\_name)
* **📦 So, the relationship is:**

| **Component** | **Role** | **Location** |
| --- | --- | --- |
| prometheus\_client in Flask | Exposes /metrics | Inside your app |
| prometheus.yml | Tells Prometheus where and how to scrape | Mounted in Prometheus container |

**✅ Step 3.3: Add Prometheus config :**

Create prometheus.yml config:

root@ip-172-31-14-172:~/raman-micro# cat prometheus.yml

global:

scrape\_interval: 10s

scrape\_configs:

- job\_name: 'raman-micro'

static\_configs:

- targets: ['host.docker.internal:5000']

This tells Prometheus to scrape http://localhost:5000/metrics every 10 seconds.

**✅ Step 3.4: Run Prometheus and grafana :**

docker network create observability-net

# Run Prometheus and connect to network

docker run -d --name prometheus --network observability-net --add-host=host.docker.internal:host-gateway -v "$(pwd)/prometheus.yml":/etc/prometheus/prometheus.yml -p 9090:9090 prom/Prometheus

# Run Grafana and connect to network

docker run -d \

--name grafana \

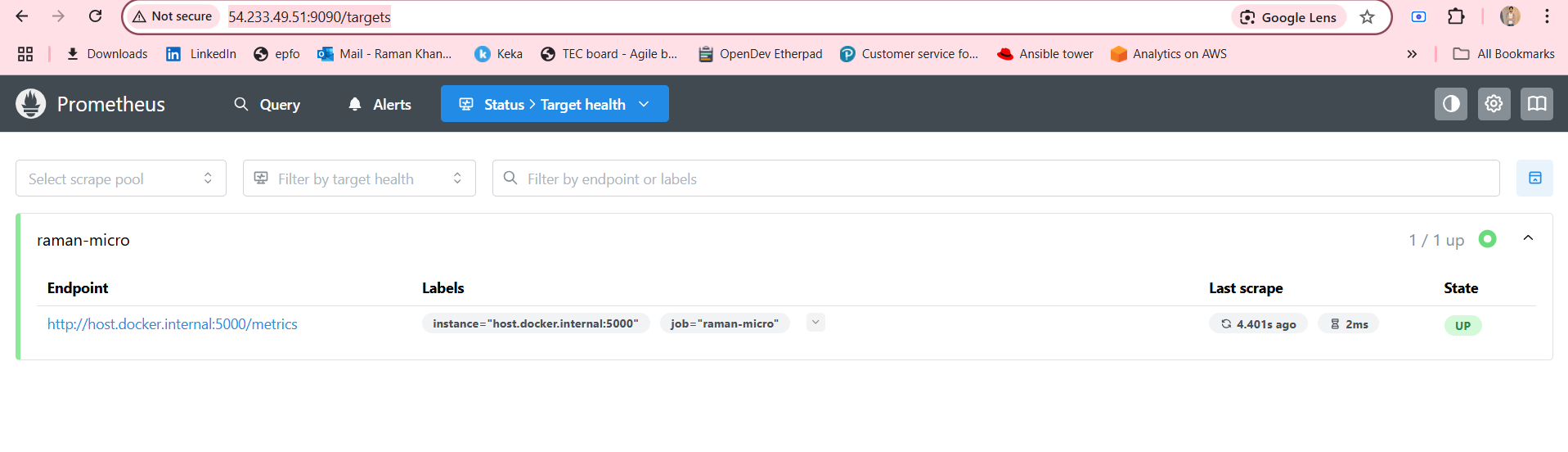
--network observability-net \

-p 3001:3000 \

grafana/Grafana

* Check target health on promethues ui if target point is up or not !!

http://54.233.49.51:9090/targets



You should now see:

* ✅ raman-micro **UP**
* Scrape successful

**📝 Summary**

| **Task** | **Command or Check** |
| --- | --- |
| Flask app running | python3 app/main.py |
| Flask metrics available | curl http://localhost:5000/metrics |
| Prometheus uses host access | --add-host=host.docker.internal:host-gateway |
| Docker network OK | Use same --network observability-net |

1. **Add Prometheus as a data source**:
   * Go to **Gear icon ⚙️ > Data Sources**
   * Click **“Add data source”**
   * Choose **Prometheus**
   * Fill in:

makefile

CopyEdit

Name: Prometheus

URL: http://prometheus:9090

⚠️ Important: This works only because both containers share the observability-net network.

* + Click **Save & Test**

✅ You should see **"Data source is working"**

**🧪 Next: Build a Basic Dashboard**

Let’s visualize your Flask app metrics now.

1. Go to **Dashboards → New → New Dashboard**

**📊 Step 2: Add Metrics Panel**

**🎯 Panel 1: Request Count Over Time**

* **Data source**: Prometheus
* **Query**:

nginx

CopyEdit

request\_count\_total

* **Legend**: Total Requests
* **Visualization**: Time series
* **Title**: Total Requests (request\_count\_total)
* Click **Apply**

**📈 Panel 2: Request Latency**

* Add another panel
* **Query**:

scss

CopyEdit

rate(request\_latency\_seconds\_bucket{le="1.0"}[1m])

* OR visualize just average latency:

css

CopyEdit

rate(request\_latency\_seconds\_sum[1m]) / rate(request\_latency\_seconds\_count[1m])

* **Legend**: Latency
* **Visualization**: Time series
* **Title**: Average Request Latency
* Click **Apply**

**✅ Step 3: Save Dashboard**

* Click the top-right **disk icon**
* Name: raman-micro-dashboard
* Save it

**🧪 Optional: Try Generating Some Traffic**

If you're only seeing request\_count\_total = 1, hit the app a few times:

bash

CopyEdit

curl http://localhost:5000/

Then reload the /metrics and Grafana — you should see request\_count\_total increase, and latency buckets populated.



**Do You Need prometheus\_exporter.py?**

**No — not anymore.**  
You’ve already:  
✅ Integrated Prometheus in your actual Flask microservice  
✅ Exposed /metrics  
✅ Verified Prometheus is scraping real-time app metrics

**✅ You can delete or ignore prometheus\_exporter.py.**

**✅ Final Recommendation**

| **File** | **Keep?** | **Why?** |
| --- | --- | --- |
| app/main.py | ✅ Yes | Your real microservice with embedded metrics |
| prometheus\_exporter.py | ❌ No | Was only for testing/simulation; not needed anymore |

**📝 Summary Table**

| **Component** | **Status** | **Notes** |
| --- | --- | --- |
| Flask app | ✅ | Prometheus client instrumented |
| /metrics | ✅ | Exposes custom & default metrics |
| Prometheus | ✅ | Scrapes every 10s using config |
| Grafana | ✅ | Container running, UI works |
| Data Source | ✅ | Prometheus connected using service name |
| Dashboard | ✅ | Panels for count and latency added |