**Lab 1: AI-Augmented Infrastructure Request via Chat Interface**

**🎯 Goal:**

Demonstrate how a developer or platform engineer can use an **AI assistant (e.g., ChatGPT or Bedrock agent)** to request infrastructure resources (e.g., S3 bucket, EC2, or Kubernetes namespace) via natural language, and generate a corresponding **IaC template or API request**.

**🧰 Prerequisites**

| **Tool/Tech** | **Requirement** |
| --- | --- |
| OpenAI/ChatGPT or Amazon Bedrock | API or UI access |
| Terraform CLI (v1.3+) or Pulumi (optional variant) | Installed |
| CLI tools: jq, curl, aws (optional for validation) | Installed |
| Editor (VSCode) | Installed |
| Basic AWS credentials or mock environment | Pre-configured or simulated |
| Prompt engineering worksheet (provided) | Optional template |

**📋 Outline**

**🔩 Step-by-Step Instructions**

**✅ Step 1: Framing the Scenario**

*"You're a developer who needs a new S3 bucket for storing image uploads from a frontend app. You want to request this using natural language and get back a ready-to-deploy Terraform config."*

We know already

* How infra requests typically go through tickets.
* How ChatOps or agentic AI changes this (faster, context-aware).

**✅ Step 2: Open ChatGPT / Bedrock Agent (UI or API)**

Participants open:

* **ChatGPT** (Web UI or API via playground)
* Or **Bedrock Console > Claude/Sonnet UI** (if using Amazon stack)

**✅ Step 3: Make a Natural Language Infra Request**

**type or paste**:

Hi, I need an S3 bucket for storing image uploads from our frontend app. It should:

- Have private access only

- Enable server-side encryption (SSE-S3)

- Versioning enabled

- Enforce naming convention: `teamname-env-purpose` (e.g., `frontend-prod-images`)

Please give me a Terraform configuration.

**✅ Step 4: Validate the AI-Generated IaC**

Typical expected output (Terraform HCL):

resource "aws\_s3\_bucket" "frontend\_prod\_images" {

bucket = "frontend-prod-images"

versioning {

enabled = true

}

server\_side\_encryption\_configuration {

rule {

apply\_server\_side\_encryption\_by\_default {

sse\_algorithm = "AES256"

}

}

}

acl = "private"

tags = {

Name = "frontend-prod-images"

Environment = "prod"

Purpose = "image-uploads"

}

}

* Does it match your original request?
* What is missing (e.g., policy? bucket lifecycle?)

**✅ Step 5: Save and Test the Template (Optional)**

1. Save file as main.tf
2. Run the following:

bash

CopyEdit

terraform init

terraform plan

**✅ Step 6: Expand Use Case (Stretch Task)**

Ask ChatGPT or Bedrock to:

"Add a policy that only allows objects to be uploaded by a specific IAM role."

✅ Compare AI's generated policy with AWS docs.

**✅ Step 7: Reflect and Map to Platform Flow**

* How would this work in an internal developer platform?
* What would be required to productionize this agent? (e.g., validation, approvals, observability)
* Where does this sit in the MVP lifecycle?

**✅ Step 8: Bonus Prompting Practice (Optional)**

Offer advanced tasks:

1. **Multi-resource** prompt:

"Create an S3 bucket and a Lambda function that processes image uploads from it."

1. **Security check** prompt:

"Check if this bucket config violates any security best practices."

1. **Convert Terraform to Pulumi Python**

Useful to show agent flexibility.

**📊 Expected Outcomes**

| **Area** | **Evidence** |
| --- | --- |
| AI understanding | Terraform script reflects user request |
| Prompting skill | Learners refine prompts to get more accurate output |
| IaC familiarity | Learners read and validate S3 config |
| Platform framing | Clear idea of where AI fits in request automation flow |

**Reference Architecture Walkthrough**

Help participants visualize and understand a **modern platform reference architecture** built using:

* Open Source Developer Portal (**Backstage**)
* GitOps-based delivery (**ArgoCD**)
* Golden Paths/templates
* AI augmentations (ChatOps, Code Scaffolding)
* Optional security and observability integration (Vault, OPA, Prometheus, Grafana)

**✅ Step 1: High-Level Reference Architecture**

┌──────────────────────────┐

│ Developer UI │

│ (Backstage Portal UI) │

└────────────┬─────────────┘

│

┌────────────────────────▼────────────────────────┐

│ Developer Golden Path Workflows │

│ (Templates, CI/CD pipelines, deploy scripts) │

└────────────────────────┬────────────────────────┘

│

┌───────────────────────────────────▼────────────────────────────────────┐

│ GitOps Control Plane – Declarative Desired State │

│ (GitHub/GitLab repos, Helm/Kustomize manifests, GitHub Actions) │

└───────────────────────────────────┬────────────────────────────────────┘

│

┌──────────────────────▼──────────────────────┐

│ ArgoCD or FluxCD │

│ (Sync manifests → Kubernetes target) │

└──────────────────────┬──────────────────────┘

│

┌────────────────────────────────────▼────────────────────────────────────┐

│ Runtime Infra: Kubernetes, S3, EKS, Vault, Istio, etc. │

└────────────────────────────────────┬────────────────────────────────────┘

│

┌─────────────────────────▼────────────────────────┐

│ Observability + Security Stack │

│ (Prometheus, Grafana, Loki, Vault, OPA, SAST) │

└──────────────────────────────────────────────────┘

**✅ Step 2: Walk Through Each Layer**

For each layer :

| **Layer** | **Tools** | **Key Role** |
| --- | --- | --- |
| **Developer UI** | Backstage | Central entry point for services, docs, deploy |
| **Golden Paths** | Cookiecutter, Helm, GitHub Actions | Standardized, repeatable workflows |
| **GitOps Control** | GitHub, Helm, Kustomize | Declarative source of truth |
| **GitOps Engine** | ArgoCD/Flux | Continuously syncs state to infra |
| **Runtime Infra** | K8s, AWS, Vault | Where services actually run |
| **Observability + Security** | Prometheus, Vault, OPA | Secure & observable platform foundation |

**Lab 2: Golden Path Demo using Cookiecutter Template**

**🎯 Goal:**

Enable developers to bootstrap new services in a **standardized and reproducible** manner using a **Golden Path** powered by **Cookiecutter**, and integrate CI/CD scaffolding, Docker, Helm, and observability—all from a single template.

**🧪 Summary**

| **Attribute** | **Details** |
| --- | --- |
| 🧑‍🏫 Type | Hands-on, instructor-guided (local or cloud-based terminal) |
| 🧰 Tools Used | Cookiecutter, GitHub CLI, Docker, Helm, GitHub Actions |
| 📁 Provided Inputs | Pre-configured Golden Path template repo |
| 🛠️ Output | Scaffolding of a microservice project with full CI/CD setup |

**✅ Step-by-Step: Create Your Golden Path Cookiecutter Template Repository**

**🧱 1. Directory Structure**

You’ll create a folder structure like this:

cookiecutter-golden-path/

├── {{cookiecutter.project\_slug}}/

│ ├── app/

│ │ └── main.py

│ ├── Dockerfile

│ ├── helm/

│ │ ├── Chart.yaml

│ │ ├── templates/

│ │ │ └── deployment.yaml

│ ├── prometheus\_exporter.py

│ ├── .github/

│ │ └── workflows/

│ │ └── ci.yml

│ ├── README.md

│ ├── requirements.txt

│ └── tests/

│ └── test\_app.py

├── cookiecutter.json

└── LICENSE

**✨ 2. Create the Template**

**👉 Create base directory**

bash

CopyEdit

mkdir cookiecutter-golden-path && cd $\_

**👉 cookiecutter.json**

This defines the prompts users will see when generating a project:

json

CopyEdit

{

"project\_name": "My Microservice",

"project\_slug": "my\_microservice",

"author\_name": "Raman",

"description": "A sample microservice scaffolded using Cookiecutter.",

"port": "8080"

}

**🐍 3. Core Files and Templates**

**🔹 {{cookiecutter.project\_slug}}/app/main.py**

python

CopyEdit

from flask import Flask

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

@app.route("/")

def index():

return "Hello from {{ cookiecutter.project\_name }}!"

**🔹 {{cookiecutter.project\_slug}}/Dockerfile**

dockerfile

CopyEdit

FROM python:3.10-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY . .

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

**🔹 {{cookiecutter.project\_slug}}/requirements.txt**

txt

CopyEdit

flask

prometheus\_client

**🔹 {{cookiecutter.project\_slug}}/prometheus\_exporter.py**

python

CopyEdit

from prometheus\_client import start\_http\_server, Counter

import time

REQUEST\_COUNT = Counter('app\_requests\_total', 'Total app HTTP requests')

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

start\_http\_server({{ cookiecutter.port }})

while True:

REQUEST\_COUNT.inc()

time.sleep(1)

**🔹 {{cookiecutter.project\_slug}}/.github/workflows/ci.yml**

yaml

CopyEdit

name: CI

on: [push, pull\_request]

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 {{ cookiecutter.project\_slug }}/requirements.txt

- name: Run Tests

run: |

pytest {{ cookiecutter.project\_slug }}/tests

**🔹 {{cookiecutter.project\_slug}}/helm/Chart.yaml**

yaml

CopyEdit

apiVersion: v2

name: {{ cookiecutter.project\_slug }}

description: A Helm chart for deploying {{ cookiecutter.project\_name }}

version: 0.1.0

**🔹 {{cookiecutter.project\_slug}}/helm/templates/deployment.yaml**

yaml

CopyEdit

apiVersion: apps/v1

kind: Deployment

metadata:

name: {{ cookiecutter.project\_slug }}

spec:

replicas: 1

selector:

matchLabels:

app: {{ cookiecutter.project\_slug }}

template:

metadata:

labels:

app: {{ cookiecutter.project\_slug }}

spec:

containers:

- name: {{ cookiecutter.project\_slug }}

image: your-docker-image

ports:

- containerPort: {{ cookiecutter.port }}

**{{cookiecutter.project\_slug}}/tests/test\_app.py**

def test\_example():

    assert 1 == 1

* **🔹 {{cookiecutter.project\_slug}}/README.md**

md

CopyEdit

# {{ cookiecutter.project\_name }}

This is a microservice scaffolded using the Golden Path Cookiecutter template.

## Run locally

```bash

python app/main.py

**Metrics endpoint**

bash

CopyEdit

curl http://localhost:{{ cookiecutter.port }}/metrics

yaml

CopyEdit

---

## 🧪 4. \*\*Local Test of the Template\*\*

After setting up your directory, you can test it:

```bash

# Outside the template directory

cookiecutter ./cookiecutter-golden-path

Follow the prompts and a new folder like orders-api will be created with your project scaffolded.

**☁️ 5. Push to GitHub**

Create a public repo:

bash

CopyEdit

cd ..

git init

gh repo create golden-path-cookiecutter-template --public --source=. --push

**Install in a virtual environment (Clean Isolation) :**

If you're working on multiple projects and want to isolate tools:

Apt update -y

sudo apt install python3-venv -y

python3 -m venv venv

source venv/bin/activate

pip install cookiecutter

cookiecutter --version

**🧩 6. Use in Training**

Once published, your learners can now run:

Clone the repository..

**✅ Step 1: Clone & Use the Cookiecutter Template (10 mins)**

(venv) root@ip-172-31-14-172:~# ls

cookiecutter-golden-path snap venv

(venv) root@ip-172-31-14-172:~# cookiecutter ./cookiecutter-golden-path/

📌 When prompted, enter:

* Service name: orders-api
* Port: 8080
* Project slug: orders\_api

🎯 This generates a new local folder: orders-api/

**✅ Step 2: Explore the Generated Golden Path (10 mins)**

From the generated folder, walk participants through:

cd orders-api/

tree -L 2

| **Folder/File** | **Description** |
| --- | --- |
| Dockerfile | Container definition |
| helm/chart/ | Kubernetes Helm chart |
| .github/workflows/ci.yml | GitHub Actions CI/CD pipeline |
| prometheus\_exporter.py | Dummy metric exporter |
| README.md | Pre-filled doc template |
| tests/ | Sample test scaffolding |

**✅ Step 3: Push to GitHub and Trigger CI (15 mins)**

bash

CopyEdit

# Create a new GitHub repo

gh repo create orders-api --public --source=. --push

✅ Navigate to **GitHub Actions tab** – CI pipeline runs automatically.

Expected steps:

* Code lint + test
* Build Docker image (simulated)
* Push to GHCR (or DockerHub – optional)
* Run security check (e.g., trivy, snyk – mocked)
* Check Prometheus exporter presence

**✅ Step 4: Add Observability Touchpoint (10 mins)**

**PROMOTHEUS\_EXPORTER.PY IS CREATING A STANDALONE WEBSERVER(http\_start\_server) FOR MOCKING/SIMULATING METRIC GENERATION IN THIS ACTIVITY , WE WILL INTEGRATE IT INSIDE OUR CODE IN NEXT ACTIVITY WITH THE FLASK app.**

If template includes prometheus\_exporter.py:

bash

CopyEdit

# Run service locally (optional)

python prometheus\_exporter.py

curl http://localhost:8080/metrics

✅ Validate:

* Metrics endpoint exists
* **Returns mock metric** (e.g., service\_up 1)

**✅ Step 5: Deploy to Kubernetes (Optional / Cloud Teams) (15 mins)**

For those with Helm/K8s cluster access

bash

CopyEdit

# Replace values.yaml if needed

helm install orders-api ./helm/chart/ --values values.yaml

kubectl get pods

kubectl get svc

🎯 Validate that the Helm chart works for a quick deploy.

**📊 Expected Outcomes**

| **Outcome** | **Evidence** |
| --- | --- |
| Service bootstrapped from template | orders-api/ folder + GitHub repo |
| CI/CD pipeline in place | GitHub Actions status check |
| Observability baked in | Prometheus /metrics endpoint |

**Lab 3: Test the Golden Path locally**

**✅ Phase 1: Run the Service Locally**

**1️⃣ Install Dependencies**

Since you're inside a venv, just run:

bash

CopyEdit

pip install -r requirements.txt

**2️⃣ Run the Flask App**

The app is in app/main.py. Run it like this:

bash

CopyEdit

python app/main.py

Now, your API should be available at:

curl http://localhost:5000/

Expected response:

text

CopyEdit

Hello from My Microservice!

**3️⃣ Run the Prometheus Exporter (Optional)**

In another terminal:

bash

CopyEdit

python prometheus\_exporter.py

Metrics should now be available at:

bash

curl <http://localhost:8080/metrics>

Expected output includes:

text

CopyEdit

# HELP app\_requests\_total Total app HTTP requests

# TYPE app\_requests\_total counter

app\_requests\_total 3.0

**Lab 4: Test the Golden Path locally by Building and Running Docker Image**

* **Update app/main.yml in coookicutter template.**

from flask import Flask, Response

from prometheus\_client import Counter, generate\_latest, CONTENT\_TYPE\_LATEST

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

REQUEST\_COUNTER = Counter('app\_requests\_total', 'Total number of requests to the root endpoint')

@app.route("/")

def index():

REQUEST\_COUNTER.inc()

return "Hello from {{ cookiecutter.project\_name }}!"

@app.route("/metrics")

def metrics():

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

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

app.run(host="0.0.0.0", port=8080)

**4️⃣ Build Docker Image**

bash

CopyEdit

docker build -t my-microservice .

**5️⃣ Run Container**

bash

CopyEdit

(venv) root@ip-172-31-14-172:~/raman-micro# docker run -d --name mycon -p 8080:8080 my-microservice

You should be able to access:

API: curl http://localhost:8080/

Metrics: curl <http://localhost:8080/metrics>

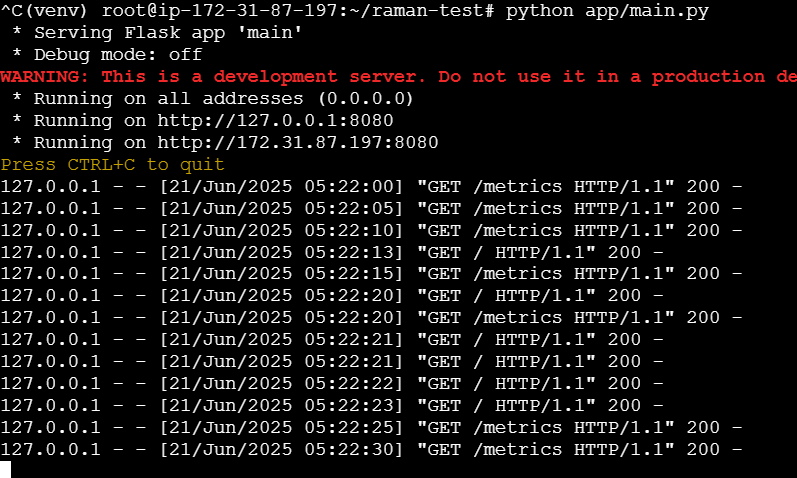
OR

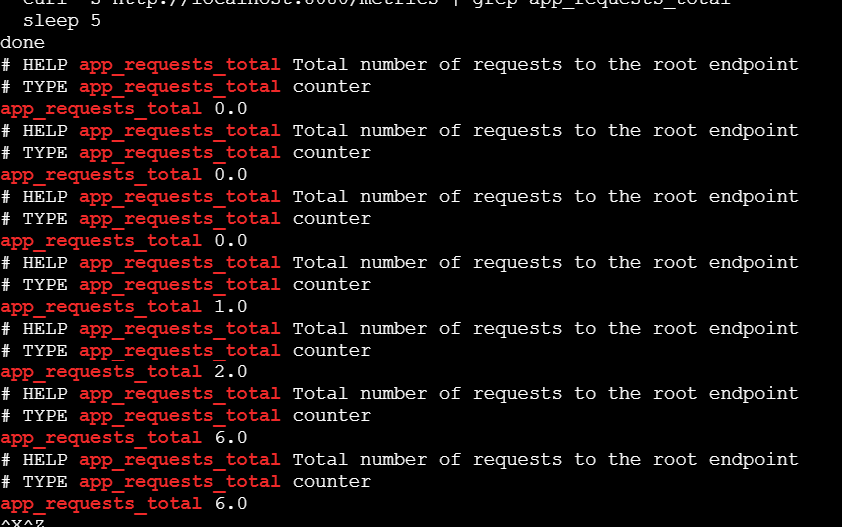
while true; do

curl -s http://localhost:8080/metrics | grep app\_requests\_total

sleep 5

done

****



**Lab 5: Reusable CI/CD Pipeline with GitHub Action**

**🔧 Objective:**

**Build a secure and reusable CI/CD pipeline** with:

* Automated linting
* SBOM generation
* SAST scanning (Bandit)
* Docker build & push (to DockerHub or GHCR)
* Optional Helm lint & deployment job
* Add the .github/workflows/ci.yml file in the cookicuter template :

(venv) root@ip-172-31-87-197:~/cookiecutter-golden-path-test/{{cookiecutter.project\_slug}}# cat .github/workflows/ci.yml

name: {{ cookiecutter.ci\_pipeline\_name }}

on:

workflow\_dispatch:

push:

branches: [ "{{ cookiecutter.default\_branch }}" ]

pull\_request:

branches: [ "{{ cookiecutter.default\_branch }}" ]

jobs:

lint-test-scan:

runs-on: ubuntu-latest

steps:

- name: Checkout Code

uses: actions/checkout@v4

- name: Set up Python

uses: actions/setup-python@v5

with:

python-version: '{{ cookiecutter.python\_version }}'

- name: Install Dependencies

run: |

python -m pip install --upgrade pip

pip install -r requirements.txt

pip install bandit cyclonedx-bom

- name: Run Unit Tests

run: |

pytest tests/

{% if cookiecutter.enable\_security == "yes" %}

- name: Run Bandit Security Scan

run: bandit -r app -f json -o bandit-report.json || true

- name: Generate SBOM

run: cyclonedx-py requirements -i requirements.txt -o sbom.json

{% endif %}

docker-build-push:

needs: lint-test-scan

runs-on: ubuntu-latest

steps:

- name: Checkout Code

uses: actions/checkout@v4

- name: Set up Docker Buildx

uses: docker/setup-buildx-action@v3

- name: Login to DockerHub

uses: docker/login-action@v3

with:

username: ${{ '{{ secrets.DOCKERHUB\_USERNAME }}' }}

password: ${{ '{{ secrets.DOCKERHUB\_TOKEN }}' }}

- name: Build & Push Docker Image

uses: docker/build-push-action@v5

with:

context: .

push: true

tags: ${{ '{{ secrets.DOCKERHUB\_USERNAME }}' }}/{{ cookiecutter.docker\_image\_name }}:{{ cookiecutter.docker\_image\_tag }}

* **Update cookicutter.json file :**

(venv) root@ip-172-31-87-197:~/cookiecutter-golden-path-test# cat cookiecutter.json

{

"ci\_pipeline\_name": "CI Pipeline",

"project\_name": "My Microservice",

"project\_slug": "my\_microsvc",

"author\_name": "Raman",

"description": "A sample microservice scaffolded using Cookiecutter.",

"port": "8080",

"python\_version": "3.11",

"docker\_image\_name": "raman-micro",

"docker\_image\_tag": "latest",

"enable\_security": "yes",

"default\_branch": "main"

}

* **Run the cookicuter template to create raman-micro service with cicd .**

**cookiecutter cookiecutter-golden-path-test/**

**📁 Project Structure Recap (required)**

* **cookiecutter ./cookiecutter-golden-path/**

Ensure your scaffolded project (e.g., raman-micro) looks like this:

raman-micro/

├── .github/

│ └── workflows/

│ └── ci.yaml

├── app/

│ └── main.py

├── Dockerfile

├── requirements.txt

├── tests/

│ └── test\_app.py

├── prometheus\_exporter.py

├── helm/

│ └── Chart.yaml

Note : In general ; cicd ci.yml template resides inside cookiecutter template and renders as per the template configuration.

**🔐 Step 2: Add GitHub Secrets**

Go to your GitHub repo → **Settings → repo Secrets → Actions** and add:

* DOCKERHUB\_USERNAME
* DOCKERHUB\_TOKEN (generate from DockerHub > Account Settings > Security > Access Tokens)

**✅ Step 3: Commit & Push to github as a separate repository:**

cd raman-micro/

403 ls

404 ls -a

405 history

406 git init

407 git commit -m "first commit"

408 git commit -m .

409 git add .

410 git commit -m "first commit"

411 git branch -M main

412 git remote add origin https://github.com/ramannkhanna2/raman-micro.git

413 git push -u origin main

**🔎 Step 4: Watch It Run**

Go to **GitHub → Actions**, you’ll see the pipeline executing:

* 🧪 Unit tests pass
* 🛡️ Bandit scan runs
* 📦 SBOM is generated
* 🐳 Docker image gets pushed to DockerHub

**✅ ✅ ✅ Checkpoints Completion Checklist Golden Path**

**1. Project Bootstrapped from Cookiecutter Template**

* Generated project structure via cookiecutter-golden-path
* Ensured proper layout with:
  + app/main.py
  + requirements.txt
  + Dockerfile
  + tests/test\_app.py
  + prometheus\_exporter.py

✅ *Golden Path project structure created successfully*

**2. Application Verified Locally**

* Ran app via python app/main.py
* Verified / endpoint with:

bash

CopyEdit

curl http://localhost:5000/

* Verified Prometheus /metrics on port 8080

✅ *App works as expected in local development*

**3. Docker Containerization**

* Built Docker image from Dockerfile
* Included Prometheus exporter
* Verified container runs with:

bash

CopyEdit

docker run -p 5000:5000 -p 8080:8080 <your-image>

✅ *Containerized version of app runs successfully*

**4. CI/CD with GitHub Actions**

* Ran multi-step CI pipeline:
  + ✅ Install dependencies
  + ✅ Run Unit Tests via pytest
  + ✅ Run Security Scan with bandit
  + ✅ Generate SBOM with cyclonedx-py
  + ✅ Docker Build & Push to Docker Hub
* ✅ Image successfully tagged and pushed:

bash

CopyEdit

docker.io/<your-username>/raman-micro:latest

✅ *End-to-end CI pipeline executed successfully with SBOM + Security*

**🎯 Summary**

| **Component** | **Status** |
| --- | --- |
| Cookiecutter Setup | ✅ Done |
| Flask App Endpoints | ✅ Tested |
| Prometheus Exporter | ✅ Working |
| Unit Tests | ✅ Passed |
| Bandit Scan | ✅ Clean |
| SBOM Generation | ✅ Done w/ warning (unpinned deps) |
| Docker Build | ✅ Success |
| DockerHub Push | ✅ Done |

**Lab 6: CLI AI Assistant for Template Suggestion**

**🛠️ Step-by-Step Implementation**

**✅ Step 1: Install Dependencies**

You can use a venv or Conda. Then install:

pip install -U langchain langchain-community langchain-huggingface huggingface\_hub

Use openai or huggingface\_hub as LLM wrapper depending on source.

**✅ Step 2: Set Up LangChain LLM Wrapper (HuggingFace Free Model)**

We’ll use a free open-source LLM from HuggingFace (like HuggingFaceHub wrapper with mistralai/Mistral-7B-Instruct-v0.1).

Create a file: cli\_ai\_assistant.py

(venv) root@ip-172-31-14-172:~# cat assistant.py

from langchain\_huggingface import HuggingFaceEndpoint

from langchain\_core.prompts import PromptTemplate

import os

# Set your HF token (it must have access to the Inference endpoint)

os.environ["HUGGINGFACEHUB\_API\_TOKEN"] = "hf\_czNLRvUJBMgu"

# Initialize endpoint (NO use\_cache)

llm = HuggingFaceEndpoint(

repo\_id="mistralai/Mixtral-8x7B-Instruct-v0.1",

temperature=0.5,

max\_new\_tokens=300,

)

prompt = PromptTemplate(

input\_variables=["project\_description"],

template="""

You are a Platform Engineering expert. Given the following description:

"{project\_description}"

Suggest a suitable microservice starter template that includes:

- framework

- observability stack

- testing tools

- containerization (Docker)

Respond in a single paragraph.

"""

)

# Runnable chain

chain = prompt | llm

def main():

print("Developer Template Recommender Assistant")

while True:

user\_input = input("\nDescribe your project (or type 'exit'): ")

if user\_input.lower() == 'exit':

break

response = chain.invoke({"project\_description": user\_input})

print(f"\nSuggested Template:\n{response}\n")

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

main()

**🧪 Example Run**

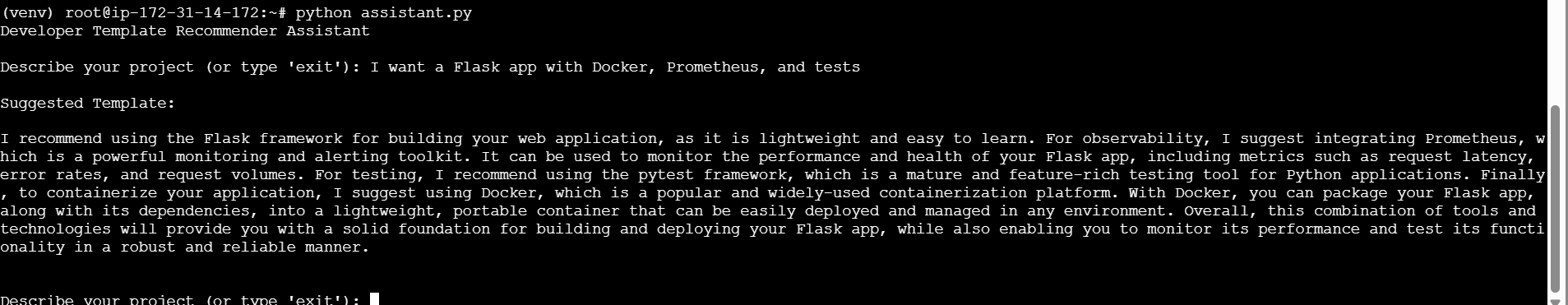
$ python cli\_ai\_assistant.py

Developer Template Recommender Assistant

Describe your project (or type 'exit'): I want a Flask app with Docker, Prometheus, and tests

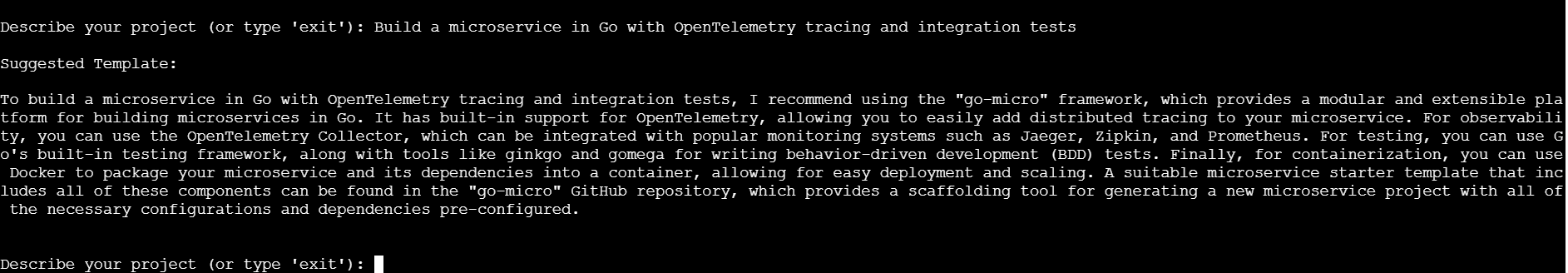
Suggested Template:

Use the `cookiecutter-flask-prometheus-docker` template. It supports Flask scaffolding, Dockerfile, Prometheus metrics and Pytest.

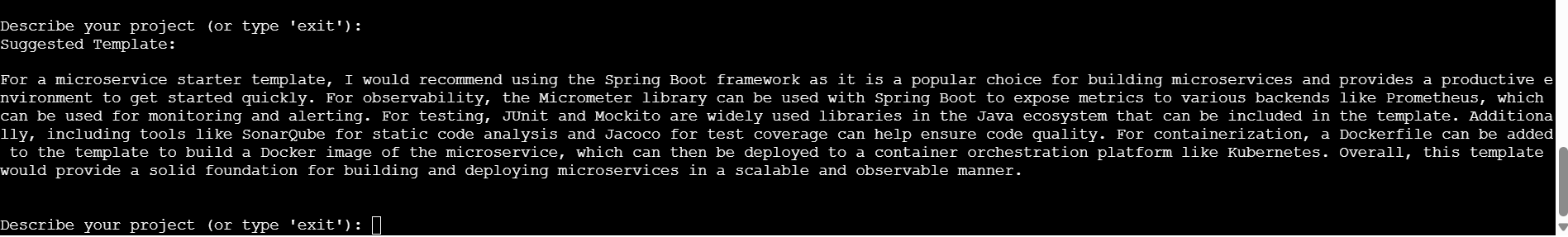


**More tests to get responses from :**

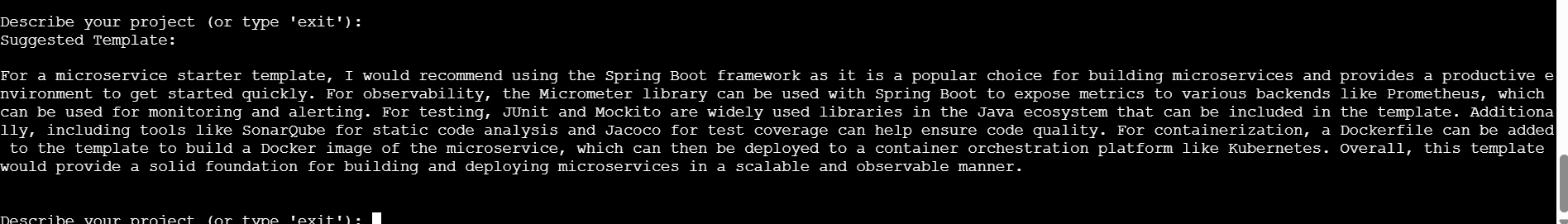
* Build a microservice in Go with OpenTelemetry tracing and integration tests



* Create a data pipeline with Apache Airflow, Docker, and monitoring alerts



* Develop a machine learning service in Python with FastAPI, logging, and Docker



* Below complete code generation applicable only with paide models : gpt 4 , other larger good models.
* **How to enable your assistant to generate complete project templates (code + config + structure)**

**Use a more detailed prompt with project scaffolding instructions**

Your prompt to the LLM should explicitly ask it to **generate the full project structure and key files**, for example:

1.

Generate a complete starter Flask microservice project that includes:

- A `app.py` file with basic Flask routes,

- A Dockerfile to containerize the app,

- Prometheus metrics instrumentation integrated in the app,

- A `tests` folder with pytest test files,

- A `requirements.txt` with necessary dependencies,

- A README.md with instructions.

Output the full content of each file clearly, including directory structure.

2.

You are a Platform Engineering expert. Given the following description:

"{project\_description}"

Suggest a suitable microservice starter template that includes:

- framework

- observability stack

- testing tools

- containerization (Docker)

Respond and Output the full content of each file clearly, including directory structure.

**Lab 7: MVP Golden Path Integration – Link Templates into Platform Design**

**🎯 Objective**

Build and validate a CLI-based AI assistant that:

* Recommends microservice templates based on user prompts
* Uses a HuggingFace-hosted LLM (Mixtral)
* Returns structured JSON responses
* Automatically scaffolds projects via Cookiecutter

It demonstrates **Golden Path Integration** where reusable templates are dynamically suggested and consumed during internal developer workflows.

pip install -U langchain langchain-community langchain-huggingface huggingface\_hub

export HUGGINGFACEHUB\_API\_TOKEN="hf\_czNLRvUJ"

(venv) root@ip-172-31-87-197:~/raman\_micro\_test# cat cli\_ai\_assistant23.py

import os

import json

import traceback

from dotenv import load\_dotenv

from langchain\_core.prompts import PromptTemplate

from langchain\_huggingface import HuggingFaceEndpoint

load\_dotenv()

# ✅ List of Available Golden Path Templates

GOLDEN\_PATH\_TEMPLATES = [

{

"name": "Golden Path Microservice (Flask, Python)",

"stack": ["python", "flask"],

"framework": "Flask",

"docker": True,

"observability": ["Prometheus", "Grafana"],

"test\_frameworks": ["pytest"],

"repo\_url": "https://github.com/ramannkhanna2/cookiecutter-golden-path.git"

}

]

# 🧠 Prompt for Stack Detection

STACK\_DETECT\_PROMPT = PromptTemplate(

input\_variables=["project\_description"],

template="""

Extract the technology stack from this project description.

Respond ONLY in this JSON format:

{{

"stack": ["<tech1>", "<tech2>", ...]

}}

Project Description: {project\_description}

"""

)

# 🔍 Match stack with available templates

def find\_matching\_template(detected\_stack):

for template in GOLDEN\_PATH\_TEMPLATES:

if any(tech.lower() in detected\_stack for tech in template["stack"]):

return template

return None

# 🏗️ Generate fallback scaffold JSON

def fallback\_scaffold(detected\_stack):

stack\_label = ", ".join(detected\_stack)

return {

"error": "❌ No matching template found for the given stack.",

"suggestion": "You can create a new Cookiecutter template for this stack. Here's a sample structure to get you started.",

"template\_scaffold": {

"cookiecutter.json": {

"project\_name": f"My {stack\_label} Microservice",

"project\_slug": f"{'\_'.join(detected\_stack)}\_microservice",

"author\_name": "Your Name",

"description": f"A simple {stack\_label} microservice template with basic observability, Docker, and tests",

"port": "3000"

},

"folder\_structure": [

"{{cookiecutter.project\_slug}}/",

"├── app.main", # generic

"├── Dockerfile",

"├── observability.yaml",

"├── test/",

"│ └── app.test"

]

}

}

def main():

print("🔧 Developer Template Recommender Assistant\n")

print("Hugging Face Token Present:", bool(os.getenv("HUGGINGFACEHUB\_API\_TOKEN")))

try:

llm = HuggingFaceEndpoint(

repo\_id="mistralai/Mixtral-8x7B-Instruct-v0.1",

temperature=0.3,

huggingfacehub\_api\_token=os.getenv("HUGGINGFACEHUB\_API\_TOKEN")

)

except Exception as e:

print("❌ Error initializing LLM:", e)

return

chain = STACK\_DETECT\_PROMPT | llm

while True:

user\_input = input("\n📝 Describe your project (or type 'exit'): ").strip()

if user\_input.lower() == "exit":

break

if not user\_input:

print("⚠️ Please enter a valid description.")

continue

try:

response = chain.invoke({"project\_description": user\_input})

print("\n📦 Raw LLM Output:\n", response)

try:

parsed = json.loads(response)

detected\_stack = [s.lower().strip() for s in parsed.get("stack", [])]

print(f"\n🔍 Detected Stack: {detected\_stack}")

template = find\_matching\_template(detected\_stack)

if not template:

fallback = fallback\_scaffold(detected\_stack)

print(f"\n❌ {fallback['error']}")

print(f"\n💡 {fallback['suggestion']}")

print("\n📁 Suggested Template Structure:")

print(json.dumps(fallback["template\_scaffold"], indent=2))

else:

print("\n✅ Recommended Template:\n")

for k, v in template.items():

print(f"{k}: {v}")

confirm = input("\n➡️ Do you want to scaffold this project using the suggested template? (y/n): ").strip().lower()

if confirm == 'y':

print(f"\n➡️ Scaffolding project using: {template['repo\_url']}")

os.system(f"cookiecutter {template['repo\_url']} --output-dir ./generated-projects")

print("\n✅ Project scaffolded successfully into: ./generated-projects")

except json.JSONDecodeError:

print("❌ Failed to parse LLM response as JSON. Raw output:")

print(response)

print("\n" + "-"\*60 + "\n")

except Exception as e:

print("Exception traceback:")

traceback.print\_exc()

print(f"Error message: {e}")

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

main()

* You can now test it by entering a project description like:

I want a Go microservice using Fiber, containerized with Docker and instrumented via Prometheus

* Above template will end in a suggestion as its not in the git repository which is referenced in prompt template.

I want a Go microservice using Fiber, containerized with Docker and instrumented via Prometheus

Build a Rust microservice using Actix-Web with Docker support and Prometheus metrics

A Java microservice using Spring Boot, Dockerized, and supports distributed tracing using Zipkin

A Node.js API using Express.js, containerized with Docker, and logs exported via Prometheus

**🧱 Next Step (optional) : Add a Golden Path Template for Go + Fiber**

Update your GOLDEN\_PATH\_TEMPLATES like this:

GOLDEN\_PATH\_TEMPLATES = [

{

"name": "Golden Path Microservice (Flask, Python)",

"stack": ["python", "flask"],

"framework": "Flask",

"docker": true,

"observability": ["Prometheus", "Grafana"],

"test\_frameworks": ["pytest"],

"repo\_url": "https://github.com/ramannkhanna2/cookiecutter-golden-path.git"

},

{

"name": "Golden Path Microservice (Go, Fiber)",

"stack": ["go", "fiber"],

"framework": "Fiber",

"docker": true,

"observability": ["Prometheus"],

"test\_frameworks": ["go test"],

"repo\_url": "https://github.com/YOUR\_USERNAME/cookiecutter-go-fiber-template.git"

},

{

"name": "Golden Path Microservice (Node.js, Express)",

"stack": ["node.js", "express"],

"framework": "Express",

"docker": true,

"observability": ["Prometheus"],

"test\_frameworks": ["jest"],

"repo\_url": "https://github.com/YOUR\_USERNAME/cookiecutter-node-express-template.git"

}

]

**OR CREATE A templates.json file as well and put all templates over ther to make the setup more scalable.**

**🧩 Recap: What You’ve Done Till Now**

**✅ Day 1 & Day 2 Summary**

| **Area** | **What You Did** |
| --- | --- |
| **Golden Paths** | Built a cookiecutter-based scaffold template (raman-micro) for microservices |
| **App Stack** | Flask app + Prometheus metrics + Dockerfile + pytest |
| **CLI Assistant** | Created a LangChain-based recommender to generate/scaffold projects |
| **Reusable CI/CD** | Generated starter GitHub Actions pipelines |
| **MVP Golden Path Integration** | Validated Golden Path + pipeline wiring in the platform setup |

✅ At this point: you have a solid service bootstrap + automation layer ready for **developer workflows**.

**Lab 8: Terraform Secured Infra Module**.

Focuses on provisioning secure foundational infrastructure components like VPC, IAM, and S3 using Terraform modules with built-in best practices for security and compliance.

**✅ Objective**

Provision secure AWS infrastructure using Terraform modules that encapsulate best practices:

* VPC with private/public subnets
* S3 buckets with encryption & block public access
* IAM roles with least privilege

**🔧 Tools Required**

* AWS CLI (configured with programmatic access)
* Terraform v1.3+ (v1.5+ recommended)
* Git, Shell
* Prebuilt Terraform modules (from registry or local repo)
* [Optional] VS Code with Terraform plugin

**📁 Folder Structure**

(venv) root@ip-172-31-14-172:~/terraform-secure-infra# tree

.

├── main.tf

├── outputs.tf

├── terraform.tfstate

├── variable.tf

└── versions.tf

**🔹 Step-by-Step Guide**

**🔹 Step 1: Create a New Terraform Project Directory**

bash

CopyEdit

mkdir terraform-secure-infra && cd terraform-secure-infra

(venv) root@ip-172-31-14-172:~/terraform-secure-infra# ls

main.tf outputs.tf terraform.tfstate variable.tf versions.tf

**🔹 Step 2: Define Providers and Versions (s3 backend optional for now)**

**versions.tf**

(venv) root@ip-172-31-14-172:~/terraform-secure-infra# cat versions.tf

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "~> 5.0"

}

}

required\_version = ">= 1.3.0"

}

provider "aws" {

region = "us-east-1"

}

If the backend doesn’t exist yet, initialize with local backend first and switch later.

**Step3 : VPC Module (Secure Network), Secure S3 Bucket (Block Public Access, Enable Encryption) , IAM Module with Least Privilege Role.**

(venv) root@ip-172-31-14-172:~/terraform-secure-infra# cat main.tf

module "vpc" {

source = "terraform-aws-modules/vpc/aws"

version = "5.1.0"

name = "secure-vpc"

cidr = "10.0.0.0/16"

azs = ["us-east-1a", "us-east-1b"]

private\_subnets = ["10.0.1.0/24", "10.0.2.0/24"]

public\_subnets = ["10.0.101.0/24", "10.0.102.0/24"]

enable\_nat\_gateway = true

single\_nat\_gateway = true

tags = {

Environment = "platform-lab"

Owner = "raman"

}

}

resource "aws\_s3\_bucket" "secure\_bucket" {

bucket = "platform-secure-artifacts-${random\_id.id.hex}"

force\_destroy = true

tags = {

Name = "SecureBucket"

Environment = "platform-lab"

}

}

resource "aws\_s3\_bucket\_public\_access\_block" "block\_public" {

bucket = aws\_s3\_bucket.secure\_bucket.id

block\_public\_acls = true

block\_public\_policy = true

ignore\_public\_acls = true

restrict\_public\_buckets = true

}

resource "aws\_s3\_bucket\_server\_side\_encryption\_configuration" "encrypt" {

bucket = aws\_s3\_bucket.secure\_bucket.id

rule {

apply\_server\_side\_encryption\_by\_default {

sse\_algorithm = "AES256"

}

}

}

resource "random\_id" "id" {

byte\_length = 4

}

data "aws\_iam\_policy\_document" "assume\_role\_policy" {

statement {

effect = "Allow"

principals {

type = "Service"

identifiers = ["ec2.amazonaws.com"]

}

actions = ["sts:AssumeRole"]

}

}

resource "aws\_iam\_role" "custom" {

name = "platform-app-role"

assume\_role\_policy = data.aws\_iam\_policy\_document.assume\_role\_policy.json

}

resource "aws\_iam\_role\_policy\_attachment" "attach" {

role = aws\_iam\_role.custom.name

policy\_arn = "arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess"

}

**🔹 Step 6: Variables and Outputs**

**variables.tf** – for reusability (optional)

(venv) root@ip-172-31-14-172:~/terraform-secure-infra# cat variable.tf

variable "region" {

type = string

default = "sa-east-1"

}

**outputs.tf**

(venv) root@ip-172-31-14-172:~/terraform-secure-infra# cat outputs.tf

output "vpc\_id" {

value = module.vpc.vpc\_id

}

output "s3\_bucket" {

value = aws\_s3\_bucket.secure\_bucket.bucket

}

**🔹 Step 7: Initialize and Apply**

bash

CopyEdit

terraform init

terraform plan

terraform apply -auto-approve

**🔹 Step 8: Validate Resources**

**✅ VPC**

* 2 public + 2 private subnets
* NAT gateway in public subnet

**✅ S3**

* Encryption: AES256
* Public Access Blocked

**✅ IAM**

* Role visible in AWS Console with AmazonS3ReadOnlyAccess

**✅ Extra Enhancements (Optional)**

* Add logging and versioning to S3
* Add CloudWatch log group outputs
* Use terraform-docs to generate module README
* Add tfsec or checkov scan for policies

**✅ Deliverables**

| **Component** | **Status** |
| --- | --- |
| VPC with NAT | ✅ |
| Encrypted S3 with public access blocked | ✅ |
| IAM role with least privilege | ✅ |
| Terraform module-based infra | ✅ |