**Lab: 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 |

**Lab: 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: 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 LAB , WE WILL INTEGRATE IT INSIDE OUR CODE IN NEXT LAB 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: 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: 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>

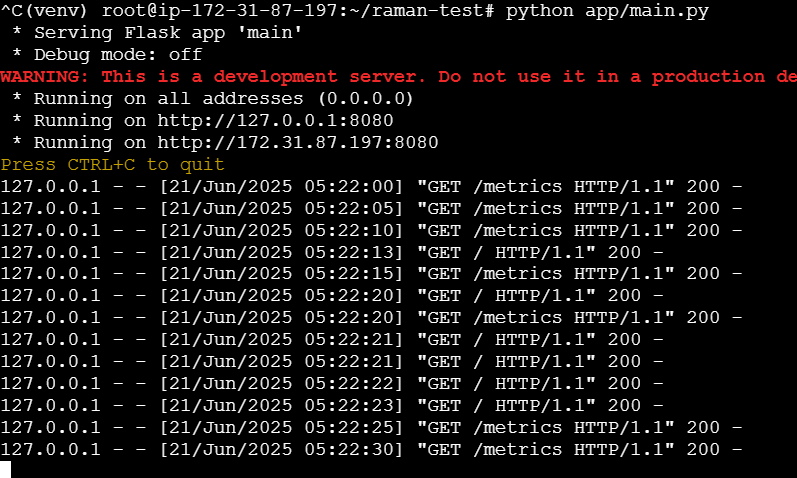
OR

while true; do

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

sleep 5

done

****

