Perfect — let’s build **Project 1** from absolute beginner level, step by step, with no assumptions.
By the end of this project, you’ll have a **real CI/CD pipeline**, a **containerized app**, and a **cloud deployment** — exactly what a Middle+ DevOps Engineer portfolio needs.

🚀 PROJECT 1: CI/CD Pipeline for a Simple Cloud-Deployed App

Beginner-friendly, step-by-step, no skipping steps

🧩 PHASE 1 — Create a Simple Application (Beginner Level)

You don’t need to be a developer. A tiny API is enough.

1. Create a new GitHub repository

Name it:
devops-hello-api

Inside it, create this folder structure:

```
devops-hello-api/
 ├── app/
 |    └── main.py
 ├── docker/
 ├── .github/
 |    └── workflows/
 └── README.md
```

2. Add a simple Python API

Inside app/main.py:

```
from flask import Flask
app = Flask(__name__)


@app.route("/health")
```

```
def health():
    return {"status": "ok"}


if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8080)
```

3. Add requirements

Create app/requirements.txt:

```
flask
```

🐳 PHASE 2 — Dockerize the Application

4. Create a Dockerfile

Inside the root folder:

```
FROM python:3.10-slim


WORKDIR /app


COPY app/requirements.txt .
RUN pip install -r requirements.txt


COPY app/ .


EXPOSE 8080


CMD ["python", "main.py"]
```

5. Build and test locally

Run:

```
docker build -t hello-api .

docker run -p 8080:8080 hello-api
```

Open browser:
http://localhost:8080/health
You should see:

```
{"status": "ok"}
```

☁️ PHASE 3 — Push Code to GitHub

6. Commit and push

```
git add .

git commit -m "Initial API + Dockerfile"

git push origin main
```

Your repo is now ready for CI/CD.

🔄 PHASE 4 — Build CI Pipeline (GitHub Actions)

7. Create workflow file

Create:
.github/workflows/ci.yml

Add:

```
name: CI Pipeline


on:

  push:

    branches: [ "main" ]

  pull_request:


jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      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 app/requirements.txt

    - name: Run tests
      run: |
        echo "No tests yet"

    - name: Build Docker image
      run: |
        docker build -t hello-api .
```

This pipeline will run automatically on every push.

🏗 PHASE 5 — Push Docker Image to Cloud Registry

You can choose **Azure** (ACR) or **AWS** (ECR).
Since you’re in Dubai and targeting GCC roles, **Azure is perfect**.

8. Create Azure Container Registry

In Azure Portal:

- Create resource group
- Create ACR
- Note the registry name: yourregistry.azurecr.io

9. Add GitHub secrets

In your GitHub repo → Settings → Secrets → Actions:

Add:

- AZURE_CLIENT_ID
- AZURE_TENANT_ID
- AZURE_SUBSCRIPTION_ID
- AZURE_CLIENT_SECRET

10. Update CI to push image

Modify workflow:

```
  - name: Azure Login
    uses: azure/login@v1
    with:
      client-id: ${{ secrets.AZURE_CLIENT_ID }}
      tenant-id: ${{ secrets.AZURE_TENANT_ID }}
      subscription-id: ${{ secrets.AZURE_SUBSCRIPTION_ID }}
      client-secret: ${{ secrets.AZURE_CLIENT_SECRET }}


  - name: Build and Push to ACR
    run: |
      az acr login --name yourregistry
      docker build -t yourregistry.azurecr.io/hello-api:latest .
      docker push yourregistry.azurecr.io/hello-api:latest
```

Now your pipeline builds and pushes images to Azure.

☸ PHASE 6 — Deploy to Kubernetes (AKS)

This step makes your portfolio stand out.

11. Create AKS cluster

In Azure Portal or CLI:

```
az aks create \
  --resource-group my-rg \
  --name my-aks \
  --node-count 1 \
  --generate-ssh-keys
```

12. Get kubeconfig

```
az aks get-credentials --resource-group my-rg --name my-aks
```

13. Create Kubernetes manifests

k8s/deployment.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-api
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello-api
  template:
    metadata:
      labels:
        app: hello-api
    spec:
      containers:
      - name: hello-api
        image: yourregistry.azurecr.io/hello-api:latest
        ports:
        - containerPort: 8080
```

k8s/service.yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: hello-api
spec:
  type: LoadBalancer
  selector:
    app: hello-api
  ports:
  - port: 80
    targetPort: 8080
```

14. Apply manifests

```
kubectl apply -f k8s/
```

Azure will give you a public IP.
Visit:

http://<public-ip>/health

🔁 PHASE 7 — Add CD (Automatic Deployment)

Extend GitHub Actions:

```
  - name: Deploy to AKS
    run: |
      az aks get-credentials --resource-group my-rg --name my-aks
      kubectl set image deployment/hello-api hello-api=yourregistry.azurecr.io/hello-api:latest
```

Now every push → new image → automatic deployment.

📘 PHASE 8 — Add Documentation (Very Important)

In your README.md, include:

- Project purpose
- Architecture diagram
- CI/CD pipeline explanation
- How to run locally
- How deployment works
- Screenshots of:
    - GitHub Actions pipeline
    - AKS service
    - Health endpoint

This makes your repo recruiter-ready.

🎯 RESULT: What You Now Have

By completing Project 1, you've built:

✓ A real API

✓ Dockerized application

✓ CI pipeline

✓ Cloud container registry

✓ Kubernetes deployment

✓ CD automation

✓ Professional documentation

This is **exactly** what a Middle+ DevOps Engineer portfolio needs.

If you want, I can now guide you through:

🔥 **Project 2 — Terraform Cloud Foundation**
or
📦 **I can help you build the GitHub repo structure for Project 1**
or
🧪 **We can start implementing Phase 1 right now**

Tell me what you want to do next.