sample github jenkins docker terraform aws example with dockerfile

Below is a **simple end-to-end example** showing how GitHub → Jenkins → Docker → Terraform → AWS can work together.
It includes:

- GitHub repo structure
- Jenkins pipeline (`Jenkinsfile`)
- Dockerfile
- Terraform example (ECR + ECS Fargate or EC2)
- Brief explanations

You can copy/paste this into a sample project.

📁 Repository Structure

```css
sample-ci-cd/
│
├── app/
│   ├── main.py
│   └── requirements.txt
│
├── Dockerfile
├── Jenkinsfile
│
└── terraform/
    ├── main.tf
    ├── variables.tf
    └── outputs.tf
```

🐳 Dockerfile

```dockerfile
FROM python:3.10-slim

WORKDIR /app

COPY app/requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY app/ .

EXPOSE 8080

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

🧪 Example Application (`app/main.py`)

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

@app.route("/")
def home():
    return "Hello from Jenkins + Docker + Terraform + AWS!"

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

`app/requirements.txt`:

```ini
flask==2.3.0
```

🏗️ Terraform Example (AWS: ECR + ECS Fargate)

`terraform/main.tf`

```h
provider "aws" {
  region = "us-east-1"
}

resource "aws_ecr_repository" "app_repo" {
  name = "sample-ci-cd-app"
}

resource "aws_ecs_cluster" "app_cluster" {
  name = "sample-ecs-cluster"
}

resource "aws_ecs_task_definition" "app_task" {
  family                   = "sample-task"
  network_mode             = "awsvpc"
  requires_compatibilities = ["FARGATE"]
  cpu                      = "256"
  memory                   = "512"

  container_definitions = jsonencode([
    {
      name  = "app"
      image = "${aws_ecr_repository.app_repo.repository_url}:latest"
      portMappings = [
        {
          containerPort = 8080
          hostPort      = 8080
        }
      ]
    }
  ])
}

resource "aws_ecs_service" "app_service" {
  name            = "sample-app-service"
  cluster         = aws_ecs_cluster.app_cluster.id
  task_definition = aws_ecs_task_definition.app_task.arn
  desired_count   = 1

  launch_type = "FARGATE"

  network_configuration {
    subnets         = ["subnet-1234567890abcdef0"]
    security_groups = ["sg-1234567890abcdef0"]
  }
}
```

🚀 Jenkins Pipeline (`Jenkinsfile`)

This pipeline:

1. Checks out code from GitHub
2. Builds Docker image
3. Logs in to AWS ECR
4. Pushes the image
5. Applies Terraform to deploy

```groovy
pipeline {
    agent any

    environment {
        AWS_REGION = "us-east-1"
        REPO_NAME = "sample-ci-cd-app"
        AWS_ACCOUNT_ID = "123456789012"
        ECR_URL = "${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com/${REPO_NAME}"
    }

    stages {
        stage('Checkout') {
            steps {
                git branch: 'main',
                    url: 'https://github.com/your-user/sample-ci-cd.git'
            }
        }

        stage('Build Docker Image') {
            steps {
                sh """
                  docker build -t ${REPO_NAME}:latest .
                """
            }
        }

        stage('AWS ECR Login') {
            steps {
                sh """
                aws ecr get-login-password --region ${AWS_REGION} | \
                  docker login --username AWS --password-stdin ${AWS_ACCOUNT_ID}.dkr.ecr.$
                """
            }
        }

        stage('Tag & Push Image') {
            steps {
                sh """
                  docker tag ${REPO_NAME}:latest ${ECR_URL}:latest
                  docker push ${ECR_URL}:latest
                """
            }
        }

        stage('Terraform Init') {
            steps {
                sh """
                  cd terraform
                  terraform init
                """
            }
        }

        stage('Terraform Apply') {
            steps {
                sh """
                  cd terraform
                  terraform apply -auto-approve
                """
            }
        }
    }
}
```

🔐 IAM permissions needed for Jenkins

Make sure Jenkins credentials include an IAM user/role with:

- `AmazonEC2ContainerRegistryFullAccess`
- `AmazonECS_FullAccess`
- `AmazonS3FullAccess` (if Terraform state is in S3)
- `IAMFullAccess` (optional, depending on your setup)

✔️ Summary

This sample includes:

- **Dockerfile** for Python app
- **Jenkinsfile** to build & push container + run Terraform
- **Terraform** to deploy ECS Fargate service
- **Git repo structure**

If you want, I can also generate:

✅ CloudFormation version
✅ Terraform for EC2 instead of Fargate
✅ Jenkins declarative → scripted pipeline
✅ GitHub Actions equivalent

Just tell me!