

# Hands-on Lab: Deploying Cloud-Native Banking APIs

A Mini-Project for Provisioning Services on AWS

## Target Audience

Python (FastAPI) Developers

AWS Lambda

AWS ECS

Terraform

Docker



Mini Project-Terraform-python.zip

# Agenda & Project Goal

---

## Our Goal Today

Build and deploy two microservices for a Personal Banking System using Infrastructure as Code (IaC)

## What We'll Do in 60 Minutes:

**Step 0:** Prerequisites & Setup - AWS account bootstrapping concepts

**Step 1:** Configure Infrastructure - Use pre-built Terraform module

**Step 2:** Develop the Services - Write minimal API code

**Step 3:** Package the ECS Service - Build and push Docker container

**Step 4 & 5:** Deploy & Test - Run Terraform and test live endpoints

# Mini-Project Architecture

---

We will deploy two simple, read-only API endpoints

## API Endpoint 1: Get Account Balance

`/balance/{accountId}`

**Implementation:** AWS Lambda

**Why?** Cost-effective, scales to zero, and extremely fast for simple tasks

## API Endpoint 2: Get Transaction History

`/transactions/{accountId}`

**Implementation:** AWS ECS (Fargate)

**Why?** Handles complex logic and sustained traffic using familiar container workflow

**Our Tool:** Terraform will define, provision, and manage all required AWS resources for both services

# Mini-Project Architecture (AWS Services)

---

We will deploy two simple, read-only API endpoints



## Lambda Benefits

- ▶ Perfect for short-lived functions
- ▶ Extremely fast for simple tasks
- ▶ Event-driven architecture

## ECS Benefits

- ▶ Handles complex logic well
- ▶ Sustained traffic capability
- ▶ Familiar container workflow

# Step 0: Prerequisites & Account Bootstrapping

---

## What is Account Bootstrapping?

**Preparing an AWS account for secure use by:**

- Setting up IAM users and roles with specific permissions
- Configuring billing alerts
- Enabling necessary services

### For Today's Lab (Time-Saver)

✓ Pre-configured IAM User with necessary permissions provided

## Required Tools & Setup:

### Required Tools

AWS CLI

Terraform

Docker

### Language Environment

**Java:** JDK 17+, Maven or Gradle

**Python:** Python 3.9+, Pip

# Step 1: Configure with Terraform Module

---

No complex Terraform code required - just configuration!

## What You'll Do

- ▶ Open the provided directory structure
- ▶ Edit the **terraform.tfvars** file
- ▶ Fill in your configuration values

```
// terraform.tfvars

project_name    = "personal-banking"
your_name       = "jane-doe" // Use your name to create unique resource names
aws_region      = "us-east-1"
```

**What This Creates:** VPC, subnets, IAM roles, Lambda function, and ECS service - all automatically configured!

# Step 2: Coding the Services

---

Minimal business logic - focusing on deployment, not complex code



## Lambda Handler:

```
# lambda_handler.py
import json

def handler(event, context):
    # In a real app, you'd query a database
    return {
        "statusCode": 200,
        "body": json.dumps({
            "accountId": "123",
            "balance": 5430.50
        })
    }
```

## ECS Service (FastAPI):

```
# main.py
from fastapi import FastAPI

app = FastAPI()

@app.get("/transactions/{account_id}")
def get_transactions(account_id: str):
    # In a real app, you'd query a database
    return [{"txId": "abc", "amount": -50.0}, {"txId": "def", "amount": 1200.0}]
```

# Step 3: Package & Push ECS Container

---

Common steps for both Java and Python developers (ECS service only)

1

Authenticate Docker with AWS ECR:

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin <account-id>.dkr.ecr.us-east-1.amazonaws.com
```

2

Build the Docker Image:

```
docker build -t personal-banking-transactions .
```

3

Tag and Push to ECR:

```
docker tag personal-banking-transactions:latest <account-id>.dkr.ecr.us-east-1.amazonaws.com/personal-banking-transactions:latest
```

```
docker push <account-id>.dkr.ecr.us-east-1.amazonaws.com/personal-banking-transactions:latest
```

**Note:** Terraform module has already created the ECR repository for you!



## Step 4 & 5: Deploy with Terraform & Test!

Moment of Truth! 🚀

1

Initialize Terraform:

```
terraform init
```

2

Plan the Deployment:

```
terraform plan
```

Always review the plan before applying!

3

Apply Configuration:

```
terraform apply
```

4

Test Your APIs:

```
curl <lambda_api_gateway_url>/balance/123
```

Expected: {"accountId": "123", "balance": 5430.50}

```
curl <ecs_load_balancer_url>/transactions/123
```

Expected: [{"txId": "abc", "amount": -50.0}]

# Conclusion & Key Learnings

---



**Congratulations!**



**In under 60 minutes, you successfully:**

## Account Bootstrapping

Learned AWS account preparation concepts

## Infrastructure as Code

Used Terraform for scalable cloud management

## Serverless Deployment

Provisioned AWS Lambda for event-driven tasks

## Container Services

Deployed ECS service with Docker containers

## Language Flexibility

Understood different packaging approaches

## End-to-End Workflow

Complete source-to-cloud deployment

# Q&A and Next Steps

---

Questions? 🤔

## ⚠️ Important: Clean Up

Don't forget to run `terraform destroy` to delete resources and avoid costs!

## 🚀 Next Steps:

- Connect services to a real database (DynamoDB or RDS)
- Add authentication and authorization
- Integrate Terraform into CI/CD pipeline (GitHub Actions, Jenkins)
- Implement monitoring and logging
- Scale to production workloads

**Thank you for your attention! 🙌**