**DOFS Solution Journey**

**1. Introduction**

* “This project demonstrates a serverless, event-driven architecture using AWS services and Terraform, with CI/CD automation through CodePipeline.”

**2. Understanding the Requirements**

* “The goal is to design and deploy an order processing system that takes an API request, validates it, stores it, pushes it to a fulfillment pipeline, and handles failures via DLQ and alerting.”
* “We also need a fully automated CI/CD pipeline using AWS CodePipeline and Terraform.”

**3. Key Assumptions**

* Assume orders are JSON payloads with basic fields: order\_id, product\_id, quantity.
* Simulated 70% success rate for fulfillment.
* Basic monitoring and alerts are set up using SNS (optional for MVP).

**4. Architecture Overview**

Use your **architecture diagram** while speaking. You can display this visually while explaining.

* API Gateway → Lambda (api\_handler) → Step Function
* Step Function calls:
  + validate\_order Lambda
  + store\_order Lambda → DynamoDB (orders)
  + Push to SQS (order\_queue)
* Fulfillment Lambda:
  + Reads from order\_queue
  + Processes with 70% success rate
  + Updates status in DynamoDB
  + Failures go to DLQ → failed\_orders DynamoDB
* CI/CD:
  + GitHub → CodePipeline → CodeBuild → Terraform plan/apply on DEV

**5. Terraform Modules**

* Separate modules for:
  + api\_gateway, lambdas, stepfunctions, dynamodb, sqs, monitoring
* CICD setup under terraform/cicd:
  + CodePipeline
  + CodeBuild
  + IAM Roles
  + S3 backend for Terraform state

**6. Testing Scenarios**

* **Success:** POST /order with valid payload → order gets fulfilled → DynamoDB shows FULFILLED
* **Failure:** 30% of orders fail after retries → go to DLQ → stored in failed\_orders
* CI/CD: Push to GitHub triggers pipeline → provisions infra in DEV

**9. Conclusion**

* “This solution demonstrates scalable, fault-tolerant order fulfillment using AWS Lambda, Step Functions, and SQS—all deployed via Terraform and managed via CI/CD.”