📘 EXECUTION REPORT

Dynamic Python Execution via Java, S3, and SageMaker (Part 2 → Part 6)

✅ PHASE 1 — PROVE JAVA → SAGEMAKER WORKS (FOUNDATION)

Goal: Prove that Java can trigger a SageMaker Training Job successfully.

No UI.
No dynamic code.
Just **proof of integration**.

STEP 1.1 — Understand SageMaker Script Mode (Key Concept)

For a SageMaker training job, **ALL 3 are mandatory**:

1. 📦 **Code** (Python script)
2. 🧠 **Entry point** (what file SageMaker runs)
3. 🐳 **Container** (Docker image that knows how to run Python)

We chose:

- **Container**: SageMaker managed Scikit-learn image
- **Entry point**: `numpy_job.py`
- **Code location**: S3 (`code.tar.gz`)

STEP 1.2 — Prepare S3 Code Package (CRITICAL)

On local machine

Create file:

```
numpy_job.py
```

Example content:

```
import numpy as np

a = np.array([1,2,3])
b = np.array([4,5,6])
print("Sum:", a + b)
```

Create tar file (MANDATORY)

```
tar -czf code.tar.gz numpy_job.py
```

Verify contents:

```
tar -tzf code.tar.gz
```

✅ Output MUST be:

```
numpy_job.py
```

Upload to S3

```
aws s3 cp code.tar.gz s3://sagemaker-python-scripts-shivanshi/code.tar.gz
```

STEP 1.3 — Java Code to Invoke SageMaker (STATIC SCRIPT)

📍 File:

```
src/main/java/com/shivanshi/aws/RunPythonFromS3.java
```

EXACT code used at this stage:

```
package com.shivanshi.aws;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sagemaker.SageMakerClient;
import software.amazon.awssdk.services.sagemaker.model.*;

import java.util.Map;
```

```
public class RunPythonFromS3 {

    public static void main(String[] args) {

        SageMakerClient client = SageMakerClient.builder()
                .region(Region.US_EAST_1)
                .build();

        CreateTrainingJobRequest request =
                CreateTrainingJobRequest.builder()
                        .trainingJobName("numpy-s3-job-" +
System.currentTimeMillis())

.roleArn("arn:aws:iam::780167008601:role/AmazonSageMaker-ExecutionRole-
JavaS3")
                        .algorithmSpecification(
                                AlgorithmSpecification.builder()
                                        .trainingImage(
                                          "683313688378.dkr.ecr.us-east-
1.amazonaws.com/sagemaker-scikit-learn:1.2-1-cpu-py3"
                                        )

.trainingInputMode(TrainingInputMode.FILE)
                                        .build()
                        )
                        .hyperParameters(Map.of(
                                "sagemaker_program", "numpy_job.py",
                                "sagemaker_submit_directory",
                                "s3://sagemaker-python-scripts-
shivanshi/code.tar.gz"
                        ))
                        .inputDataConfig(
                                Channel.builder()
                                        .channelName("training")
                                        .dataSource(
                                                DataSource.builder()
                                                        .s3DataSource(

S3DataSource.builder()

.s3DataType(S3DataType.S3_PREFIX)

.s3Uri("s3://sagemaker-python-scripts-shivanshi/")

.s3DataDistributionType(

S3DataDistribution.FULLY_REPLICATED
                                                                                )

.build()
                                                        )
                                                        .build()
                                        )
                                        .build()
                        )
                        .resourceConfig(
                                ResourceConfig.builder()
```

```
.instanceType(TrainingInstanceType.ML_M5_LARGE)
                                        .instanceCount(1)
                                        .volumeSizeInGb(5)
                                        .build()
                        )
                        .stoppingCondition(
                                StoppingCondition.builder()
                                        .maxRuntimeInSeconds(600)
                                        .build()
                        )
                        .outputDataConfig(
                                OutputDataConfig.builder()
                                        .s3OutputPath("s3://sagemaker-python-
scripts-shivanshi/output/")
                                        .build()
                        )
                        .build();

        client.createTrainingJob(request);
        client.close();

        System.out.println("Training job started successfully.");
    }
}
```

STEP 1.4 — IAM ISSUE & FIX (CRITICAL TURNING POINT)

❌ Problem

Training job failed with:

```
403 Forbidden - HeadObject
```

🔍 Root Cause

IAM role had **Permissions Boundary SET**, blocking:

- `s3:GetObject`
- `s3:ListBucket`

✅ Fix

IAM → Role → Permissions boundary → **REMOVE**

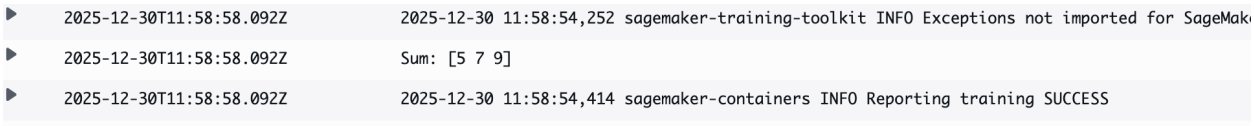
After fix:

```
Permissions boundary: NOT SET
```

✅ RESULT OF PHASE 1

✔ Java → SageMaker integration WORKS
✔ Script Mode configured correctly
✔ S3 tar structure correct
✔ IAM permissions fixed
✔ Training job completes
✔ Logs show:

```
Sum: [5 7 9]
Reporting training SUCCESS
```



✅ PHASE 2 — PART 4: UPLOAD CODE FROM UI TO S3

Goal: Take Python code as TEXT and upload it to S3.

No SageMaker changes yet.

STEP 2.1 — Create AiCodeService.java

📍 Path:

```
src/main/java/com/shivanshi/aws/AiCodeService.java
```

FULL FILE (copy-paste exactly):

```
package com.shivanshi.aws;

import software.amazon.awssdk.core.sync.RequestBody;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
```

```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class AiCodeService {

    private static final String BUCKET_NAME =
            "sagemaker-python-scripts-shivanshi";

    private void validateAiCode(String code) {
        if (code == null || !code.contains("#ai")) {
            throw new RuntimeException("Code must contain #ai hashtag");
        }
    }

    private File writeCodeToTempFile(String code) throws IOException {
        File file = File.createTempFile("ai_code_", ".py");
        FileWriter writer = new FileWriter(file);
        writer.write(code);
        writer.close();
        return file;
    }

    public String uploadCodeToS3(String code) throws IOException {

        validateAiCode(code);

        File file = writeCodeToTempFile(code);
        String key = "ai-code/" + System.currentTimeMillis() + ".py";

        S3Client s3 = S3Client.builder()
                .region(Region.US_EAST_1)
                .build();

        s3.putObject(
                PutObjectRequest.builder()
                        .bucket(BUCKET_NAME)
                        .key(key)
                        .build(),
                RequestBody.fromFile(file)
        );

        return "s3://" + BUCKET_NAME + "/" + key;
    }
}
```

STEP 2.2 — Test Upload (Part 4 Validation)

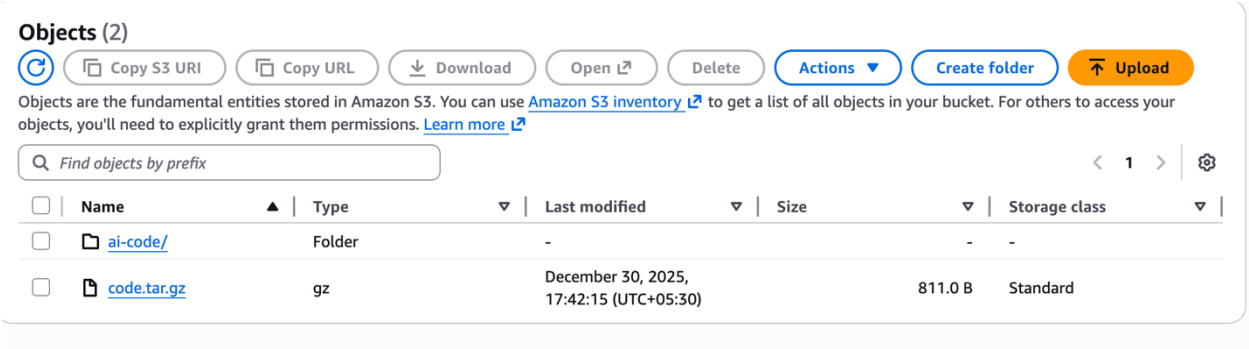
Create:

📍

```
TestUpload.java
package com.shivanshi.aws;

public class TestUpload {

    public static void main(String[] args) throws Exception {

        String codeFromUi =
                "print('Hello from AI code')\n" +
                "#ai";

        AiCodeService service = new AiCodeService();
        String s3Path = service.uploadCodeToS3(codeFromUi);

        System.out.println("Uploaded to: " + s3Path);
    }
}
```

✅ Result

- Console prints S3 path
- `.py` file visible in S3 bucket



➡️ PART 4 COMPLETE

✅ PHASE 3 — PART 5: DYNAMIC EXECUTION (REAL REQUIREMENT)

This is where **dynamic execution** was implemented correctly.

STEP 3.1 — Why numpy_job.py was NOT enough

At this point:

- SageMaker always ran `numpy_job.py`
- Uploaded UI code was **not executed**

This did NOT prove dynamic execution.

So we introduced a **runner**.

STEP 3.2 — Create runner.py (LOCAL MACHINE)

📍 Same folder where tar was created earlier

```
import os
import subprocess
import boto3

print("Runner started")

s3_path = os.environ.get("USER_SCRIPT_S3_PATH")
if not s3_path:
    raise Exception("USER_SCRIPT_S3_PATH not set")

bucket = s3_path.replace("s3://", "").split("/")[0]
key = "/".join(s3_path.replace("s3://", "").split("/")[1:])

local_file = "/opt/ml/code/user_code.py"

s3 = boto3.client("s3")
s3.download_file(bucket, key, local_file)

print("Downloaded user code, executing...")
subprocess.run(["python", local_file], check=True)
print("Runner finished")
```

STEP 3.3 — Replace code.tar.gz Contents

```
tar -czf code.tar.gz runner.py
tar -tzf code.tar.gz
```

✅ MUST output:

```
runner.py
```

Upload:

```
aws s3 cp code.tar.gz s3://sagemaker-python-scripts-shivanshi/code.tar.gz
```

STEP 3.4 — Update SageMaker Entry Point

In `RunPythonFromS3.java`

Change:

```
"sagemaker_program", "numpy_job.py"
```

To:

```
"sagemaker_program", "runner.py"
```

STEP 3.5 — Add Dynamic Invocation Method

Add inside **`RunPythonFromS3.java`**

```
public static void runWithUserScript(String scriptS3Path) {

    SageMakerClient client = SageMakerClient.builder()
            .region(Region.US_EAST_1)
            .build();

    CreateTrainingJobRequest request =
            CreateTrainingJobRequest.builder()
                    .trainingJobName("ai-job-" + System.currentTimeMillis())
                    .roleArn("arn:aws:iam::780167008601:role/AmazonSageMaker-
ExecutionRole-JavaS3")
                    .algorithmSpecification(
                            AlgorithmSpecification.builder()
                                    .trainingImage(
                                      "683313688378.dkr.ecr.us-east-
1.amazonaws.com/sagemaker-scikit-learn:1.2-1-cpu-py3"
                                    )

.trainingInputMode(TrainingInputMode.FILE)
                                    .build()
                    )
                    .hyperParameters(Map.of(
                            "sagemaker_program", "runner.py",
                            "sagemaker_submit_directory",
                            "s3://sagemaker-python-scripts-
shivanshi/code.tar.gz"
                    ))
                    .environment(Map.of(
                            "USER_SCRIPT_S3_PATH", scriptS3Path
                    ))
                    .resourceConfig(
```

```
                                ResourceConfig.builder()

.instanceType(TrainingInstanceType.ML_M5_LARGE)
                                        .instanceCount(1)
                                        .volumeSizeInGb(5)
                                        .build()
                        )
                        .stoppingCondition(
                                StoppingCondition.builder()
                                        .maxRuntimeInSeconds(600)
                                        .build()
                        )
                        .outputDataConfig(
                                OutputDataConfig.builder()
                                        .s3OutputPath("s3://sagemaker-python-
scripts-shivanshi/output/")
                                        .build()
                        )
                        .build();

    client.createTrainingJob(request);
    client.close();
}
```

STEP 3.6 — Connect Part 4 → Part 5

In `AiCodeService.java`

```
public String uploadAndRun(String code) throws Exception {

    String scriptS3Path = uploadCodeToS3(code);
    RunPythonFromS3.runWithUserScript(scriptS3Path);
    return scriptS3Path;
}
```

STEP 3.7 — FINAL PROOF TEST

```
String codeFromUi =
        "print('HELLO FROM UI')\n" +
        "print(10 + 20)\n" +
        "#ai";

service.uploadAndRun(codeFromUi);
```

✅ CloudWatch Logs

```
Runner started
Downloaded user code, executing...
HELLO FROM UI
30
```

```
Runner finished
```

➡️ **DYNAMIC EXECUTION CONFIRMED**

📘 EXECUTION REPORT (CONTINUED)

PART 6 — Converting Dynamic SageMaker Execution into a REST API

🔰 CONTEXT BEFORE PART 6

At the end of **Part 5**, the system was already capable of:

- Taking Python code as text
- Validating `#ai`
- Uploading it to S3
- Passing the S3 path to SageMaker
- Executing the **exact uploaded file dynamically** using `runner.py`
- Verifying output in CloudWatch logs

This was confirmed by logs such as:

```
Runner started
Downloaded user code, executing...
HELLO FROM UI
30
Runner finished
```

At this point:

- ❌ No REST API existed
- ❌ Execution was triggered only via Java `main()` classes

- ❌ No Postman / HTTP interface

🎯 OBJECTIVE OF PART 6

Convert the **existing dynamic execution flow** into a **RESTful web service**, so that:

```
HTTP Request
 → Java REST API
 → S3 upload
 → SageMaker execution
 → Python output
```

🪜 STEP 6.1 — Initial Assumption & Failure

Initial assumption

We assumed the project was already a Spring Boot project.

Action taken

A REST controller was added directly.

STEP 6.1.1 — REST Controller Creation

📍 Location:

```
src/main/java/com/shivanshi/aws
```

📄 File created:

```
AiCodeController.java
```

Code pasted:

```
package com.shivanshi.aws;

import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api")
```

```
public class AiCodeController {

    private final AiCodeService aiCodeService = new AiCodeService();

    @PostMapping("/run-ai-code")
    public String runAiCode(@RequestBody String code) throws Exception {

        aiCodeService.uploadAndRun(code);

        return "AI code submitted successfully";
    }
}
```

STEP 6.1.2 — ERROR OBSERVED

Immediately after creating the controller, Eclipse showed errors:

```
@RestController cannot be resolved
@PostMapping cannot be resolved
org.springframework cannot be resolved
```

STEP 6.1.3 — ROOT CAUSE ANALYSIS

The project at this stage was:

- ✅ Java project
- ✅ AWS SDK project
- ❌ **NOT a Spring Boot project**

There was:

- no Spring Boot parent
- no `spring-boot-starter-web`
- no Spring context

Hence, REST annotations could not resolve.

🪜 STEP 6.2 — Convert Project into Spring Boot Project

This step involved **ONLY Maven configuration**, no Java logic change.

STEP 6.2.1 — Modify `pom.xml`

📍 File:

`pom.xml`

Action taken

- Entire `pom.xml` was replaced
- Spring Boot parent added
- Spring Boot Web dependency added
- AWS dependencies preserved
- Spring Boot Maven plugin added

✅ FINAL `pom.xml` (EXACT)

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             https://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.0</version>
        <relativePath/>
    </parent>

    <groupId>com.shivanshi.aws</groupId>
    <artifactId>sagemaker-s3-runtime</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <properties>
        <java.version>11</java.version>
    </properties>

    <dependencies>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
```

```
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>s3</artifactId>
            <version>2.25.60</version>
        </dependency>

        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>sagemaker</artifactId>
            <version>2.25.24</version>
        </dependency>

        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>regions</artifactId>
            <version>2.25.24</version>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

STEP 6.2.2 — Maven Update

Command executed:

```
Maven → Update Project
```

Result:

- Spring dependencies downloaded
- REST annotations resolved
- Red errors disappeared

🪜 STEP 6.3 — Spring Boot Build Error & Fix

New error during build:

```
Unable to find a single main class
Found:
RunPythonFromS3
TestUpload
TestUploadAndRun
```

STEP 6.3.1 — ROOT CAUSE

Spring Boot requires **one** entry point annotated with:

```
@SpringBootApplication
```

But the project had:

- multiple `main()` methods
- no Spring Boot application class

STEP 6.3.2 — Create Spring Boot Entry Point

📍 Location:

```
src/main/java/com/shivanshi/aws
```

📄 File created:

```
SagemakerApplication.java
```

Code pasted:

```
package com.shivanshi.aws;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SagemakerApplication {

    public static void main(String[] args) {
        SpringApplication.run(SagemakerApplication.class, args);
    }
}
```

STEP 6.3.3 — Rebuild Project

Command executed:

```
mvn clean package
```

Result:

```
BUILD SUCCESS
```

Spring Boot JAR created successfully.

🪜 STEP 6.4 — Run Spring Boot Application

Application started using:

```
SagemakerApplication.java
```

Console output confirmed:

```
Tomcat started on port 8080
Started SagemakerApplication
```

This validated:

- Spring Boot runtime is active
- REST controllers are registered
- HTTP server is listening

🪜 STEP 6.5 — Postman Test (FINAL VALIDATION)

STEP 6.5.1 — Postman Configuration

Method

```
POST
```

URL

```
http://localhost:8080/api/run-ai-code
```

Headers

```
Content-Type: text/plain
```

Body (raw → Text)

```
print("HELLO FROM POSTMAN")
print(100 + 200)
#ai
```

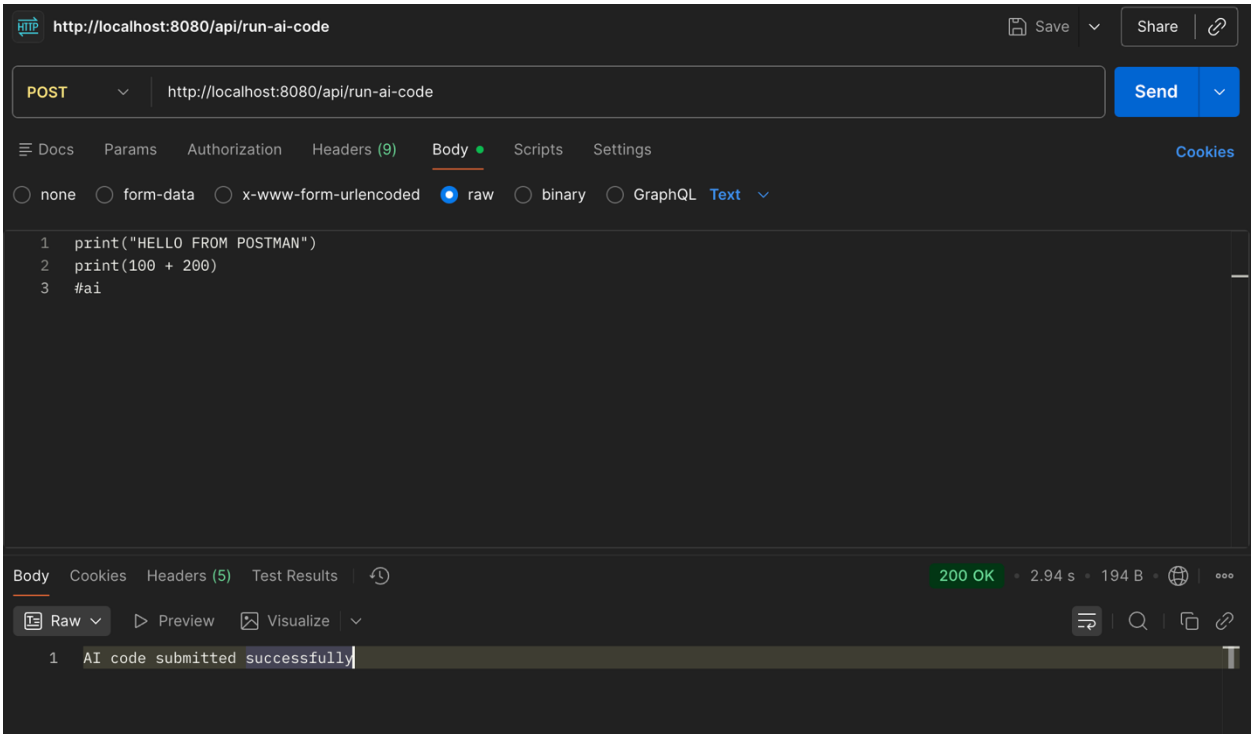
STEP 6.5.2 — Execution Flow Triggered

Postman → REST API → Java → S3 → SageMaker → Python

STEP 6.5.3 — Observed Results

Postman response:

```
AI code submitted successfully
```



Spring Boot console:

```
Uploaded user code to: s3://...
SageMaker job started for user script.
```

SageMaker / CloudWatch logs:

```
Runner started
Downloaded user code, executing...
HELLO FROM POSTMAN
300
Runner finished
```

2025-12-30T13:08:35.846Z	Runner started
2025-12-30T13:08:35.846Z	User script S3 path: s3://sagemaker-python-scripts-shivanshi/ai-code/1767099963023.py
2025-12-30T13:08:35.846Z	Downloaded user code, executing...
2025-12-30T13:08:35.846Z	HELLO FROM POSTMAN
2025-12-30T13:08:35.847Z	300
2025-12-30T13:08:35.847Z	Runner finished

Given Below is some wiring of files made during execution of all the tasks

---

✅ PHASE 1 — PROVE JAVA → SAGEMAKER WORKS (FOUNDATION)

Goal
Prove that Java can trigger a SageMaker Training Job successfully.

- No UI
- No dynamic code
- Pure integration proof

STEP 1.1 — Understand SageMaker Script Mode (Key Concept)

For a SageMaker training job, **ALL 3 are mandatory**:

1. 📦 Code (Python script)
2. 🧠 Entry point (what file SageMaker runs)
3. 🐳 Container (Docker image that knows how to run Python)

Chosen configuration

- Container: SageMaker managed Scikit-learn image
- Entry point: `numpy_job.py`
- Code location: S3 (`code.tar.gz`)

📂 FILES CREATED / USED (PHASE 1)

File	Location	Purpose
`numpy_job.py`	Local machine	Static Python script for initial validation
`code.tar.gz`	Local → S3	Package required by SageMaker Script Mode
`RunPythonFromS3.java`	Java project	Triggers SageMaker training job

STEP 1.2 — Prepare S3 Code Package (CRITICAL)

Local file created

```
numpy_job.py

import numpy as np

a = np.array([1,2,3])
b = np.array([4,5,6])
print("Sum:", a + b)
```

Create tar file (MANDATORY)

```
tar -czf code.tar.gz numpy_job.py
tar -tzf code.tar.gz
```

✅ Output MUST be:

```
numpy_job.py
```

Upload to S3

```
aws s3 cp code.tar.gz s3://sagemaker-python-scripts-shivanshi/code.tar.gz
```

STEP 1.3 — Java Code to Invoke SageMaker (STATIC SCRIPT)

📍 File

```
src/main/java/com/shivanshi/aws/RunPythonFromS3.java
```

Purpose

- First working Java → SageMaker execution
- Static Python execution only

(code unchanged — already present in your report)

STEP 1.4 — IAM ISSUE & FIX (CRITICAL TURNING POINT)

Problem

```
403 Forbidden - HeadObject
```

Root Cause

IAM Role had **Permissions Boundary SET**, blocking:

- `s3:GetObject`
- `s3:ListBucket`

Fix

```
IAM → Role → Permissions boundary → REMOVE
```

✅ Permissions boundary: **NOT SET**

✅ RESULT OF PHASE 1

✔ Java → SageMaker integration WORKS
✔ Script Mode configured correctly
✔ S3 tar structure correct
✔ IAM permissions fixed
✔ Training job completes

CloudWatch logs:

```
Sum: [5 7 9]
Reporting training SUCCESS
```

✅ PHASE 2 — PART 4: UPLOAD CODE FROM UI TO S3

Goal
Take Python code as **TEXT**, validate it, and upload it to S3.

No SageMaker changes yet.

📂 FILES CREATED / USED (PART 4)

File	Location	Purpose
`AiCodeService.java`	Java project	Validate + upload Python code
`TestUpload.java`	Java project	Local test for S3 upload

STEP 2.1 — Create `AiCodeService.java`

📍 Path:

```
src/main/java/com/shivanshi/aws/AiCodeService.java
```

Purpose

- Validate `#ai`
- Write Python code to temp file
- Upload `.py` file to S3
- Return S3 path

(code unchanged — already present in your report)

STEP 2.2 — Test Upload (Part 4 Validation)

📍 File created:

`TestUpload.java`

Purpose

- Confirm code is uploaded to S3 correctly

✅ RESULT

✔ Console prints S3 path
✔ `.py` file visible in S3 bucket

➡️ PART 4 COMPLETE

✅ PHASE 3 — PART 5: DYNAMIC EXECUTION (REAL REQUIREMENT)

This phase converts **static execution** into **true dynamic execution**.

📂 FILES CREATED / MODIFIED (PART 5)

File	Type	Purpose
`runner.py`	NEW (local → S3)	Executes uploaded Python code
`code.tar.gz`	REPLACED	Now contains only `runner.py`

`RunPythonFromS3.java`	MODIFIED	Dynamic SageMaker invocation
`AiCodeService.java`	MODIFIED	Glue between upload + execution
`TestUploadAndRun.java`	NEW	End-to-end validation

STEP 3.1 — Why `numpy_job.py` Was NOT Enough

- SageMaker always ran the same script
- Uploaded UI code never executed
- This **did NOT satisfy dynamic execution**

Solution: **introduce a runner**

STEP 3.2 — Create `runner.py` (LOCAL MACHINE)

Purpose

- Download uploaded UI code from S3
- Execute it dynamically inside SageMaker

STEP 3.3 — Replace `code.tar.gz` Contents

```
tar -czf code.tar.gz runner.py
tar -tzf code.tar.gz
```

✅ Output:

```
runner.py
```

Upload:

```
aws s3 cp code.tar.gz s3://sagemaker-python-scripts-shivanshi/code.tar.gz
```

STEP 3.4 — Update SageMaker Entry Point

In `RunPythonFromS3.java`

Change:

```
numpy_job.py
```

To:

`runner.py`

STEP 3.5 — Add Dynamic Invocation Method

Added:

`runWithUserScript(String scriptS3Path)`

Purpose

- Pass uploaded S3 path into SageMaker
- Enable true dynamic execution

STEP 3.6 — Connect Part 4 → Part 5

In `AiCodeService.java`

`uploadAndRun(String code)`

STEP 3.7 — FINAL PROOF

CloudWatch logs:

```
Runner started
Downloaded user code, executing...
HELLO FROM UI
30
Runner finished
```

➡️ **DYNAMIC EXECUTION CONFIRMED**

📘 PHASE 4 — PART 6: REST API + POSTMAN

📂 FILES CREATED / MODIFIED (PART 6)

File	Purpose

`AiCodeController.java`	REST API endpoint
`pom.xml`	Convert project to Spring Boot
`SagemakerApplication.java`	Spring Boot entry point

STEP 6.1 — REST Controller Creation

📍 File:

`AiCodeController.java`

Purpose:

- Expose POST API
- Trigger existing dynamic execution flow

STEP 6.2 — Convert Project to Spring Boot

Root cause

- Project was Java + AWS SDK
- Not a Spring Boot project

Fix

- Replace `pom.xml`
- Add Spring Boot parent
- Add `spring-boot-starter-web`

STEP 6.3 — Spring Boot Entry Point

📍 File created:

`SagemakerApplication.java`

Purpose:

- Single Spring Boot entry
- Resolve multiple `main()` conflict

STEP 6.4 — Run Application

Logs confirmed:

```
Tomcat started on port 8080
Started SagemakerApplication
```

STEP 6.5 — Postman Test (FINAL VALIDATION)

Request

```
POST http://localhost:8080/api/run-ai-code
Content-Type: text/plain
```

Body

```
print("HELLO FROM POSTMAN")
print(100 + 200)
#ai
```

Observed Results

- Postman response: `AI code submitted successfully`
- CloudWatch logs:

```
HELLO FROM POSTMAN
300
```

➡️ **PART 6 COMPLETE**

🧾 FINAL FILE INVENTORY (COMPLETE)

```
src/main/java/com/shivanshi/aws
├── AiCodeController.java
├── AiCodeService.java
├── RunPythonFromS3.java
├── SagemakerApplication.java
├── TestUpload.java
└── TestUploadAndRun.java
```

External:

```
runner.py
code.tar.gz
```