I have a ML Model trained using SageMaker pipelines and publish the model in the S3 bucket with the name as output.tar.gz.

I am writing an inference service in Java SpringBoot where I need to download the model from S3 bucket on application startup and bootstrap the service with endpoints listening in 8080 port.

Give me a full implementation steps for the same.

You can treat this as three concerns: S3 download, model extract/load, and Spring Boot startup wiring with REST endpoints. Below is a complete, opinionated implementation you can adapt.[1] [2] [3]

1. Dependencies and config

Use AWS SDK v2 and Apache Commons Compress to handle S3 and tar.gz.[3] [1]

Gradle (build.gradle)

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '3.3.2'
    id 'io.spring.dependency-management' version '1.1.5'
}

group = 'com.example'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '17'

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'

    // AWS SDK v2 – S3
    implementation platform('software.amazon.awssdk:bom:2.25.46')
    implementation 'software.amazon.awssdk:s3'

    // tar.gz extraction
    implementation 'org.apache.commons:commons-compress:1.26.0'
```

```
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

application.yml

```
server:
  port: 8080

model:
  s3:
    bucket: your-bucket-name
    key: your/pipeline/output/output.tar.gz   # key of output.tar.gz
    region: us-east-1
  local:
    dir: /opt/models/current        # where to store and extract
```

2. S3 client and download service

Create a Spring-managed S3 client and a service that downloads the artifact to a local file.[4] [1]

S3Config.java

```
package com.example.inference.config;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;

@Configuration
public class S3Config {

    @Value("${model.s3.region}")
    private String region;

    @Bean
    public S3Client s3Client() {
        return S3Client.builder()
                .region(Region.of(region))
                .credentialsProvider(DefaultCredentialsProvider.create())
                .build();
    }
}
```

ModelDownloadService.java

```
package com.example.inference.model;

import java.nio.file.Files;
```

```
import java.nio.file.Path;
import java.nio.file.StandardCopyOption;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import software.amazon.awssdk.core.sync.ResponseTransformer;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;

@Service
public class ModelDownloadService {

    private static final Logger log = LoggerFactory.getLogger(ModelDownloadService.class)

    private final S3Client s3Client;

    @Value("${model.s3.bucket}")
    private String bucket;

    @Value("${model.s3.key}")
    private String key;

    @Value("${model.local.dir}")
    private String localDir;

    public ModelDownloadService(S3Client s3Client) {
        this.s3Client = s3Client;
    }

    public Path downloadModelTarGz() throws Exception {
        Files.createDirectories(Path.of(localDir));
        Path target = Path.of(localDir, "output.tar.gz");

        log.info("Downloading model from s3://{}/{} to {}", bucket, key, target);

        GetObjectRequest get = GetObjectRequest.builder()
                .bucket(bucket)
                .key(key)
                .build();

        // Download to temp then move (safer)
        Path temp = Files.createTempFile("model-", ".tar.gz");
        s3Client.getObject(get, ResponseTransformer.toFile(temp));
        Files.move(temp, target, StandardCopyOption.REPLACE_EXISTING);

        log.info("Model downloaded to {}", target);
        return target;
    }
}
```

This assumes IAM role or environment credentials are configured for S3 access.[4]

3. Extract output.tar.gz and load model

Use Apache Commons Compress to untar the artifact at startup.[5] [3]

ModelExtractor.java

```
package com.example.inference.model;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.nio.file.Files;
import java.nio.file.Path;

import org.apache.commons.compress.archivers.tar.TarArchiveEntry;
import org.apache.commons.compress.archivers.tar.TarArchiveInputStream;
import org.apache.commons.compress.compressors.gzip.GzipCompressorInputStream;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

@Component
public class ModelExtractor {

    private static final Logger log = LoggerFactory.getLogger(ModelExtractor.class);
    private static final int BUFFER = 8192;

    public Path extractTarGz(Path tarGzPath, Path destDir) throws Exception {
        log.info("Extracting {} to {}", tarGzPath, destDir);
        Files.createDirectories(destDir);

        try (FileInputStream fin = new FileInputStream(tarGzPath.toFile());
             BufferedInputStream in = new BufferedInputStream(fin);
             GzipCompressorInputStream gzIn = new GzipCompressorInputStream(in);
             TarArchiveInputStream tarIn = new TarArchiveInputStream(gzIn)) {

            TarArchiveEntry entry;
            while ((entry = tarIn.getNextTarEntry()) != null) {
                Path destPath = destDir.resolve(entry.getName());
                if (entry.isDirectory()) {
                    Files.createDirectories(destPath);
                    continue;
                }
                Files.createDirectories(destPath.getParent());
                try (BufferedOutputStream out = new BufferedOutputStream(
                        new FileOutputStream(destPath.toFile()), BUFFER)) {
                    byte[] data = new byte[BUFFER];
                    int count;
                    while ((count = tarIn.read(data)) != -1) {
                        out.write(data, 0, count);
                    }
                }
            }
        }
```

```
        log.info("Extraction completed");
        return destDir;
    }
}
```

ModelService.java (placeholder for actual model framework)

You will adapt this based on whether it is XGBoost, PyTorch, etc.; here we just show a simple in-memory stub.[5]

```
package com.example.inference.model;

import java.nio.file.Path;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Service;

@Service
public class ModelService {

    private static final Logger log = LoggerFactory.getLogger(ModelService.class);

    private Path modelRoot;

    // Call at startup after extraction
    public void initFromExtractedDir(Path rootDir) {
        this.modelRoot = rootDir;
        log.info("Model initialized from {}", rootDir);
        // TODO: parse weights/config files and initialize actual model runtime
    }

    public String predict(String input) {
        // TODO: perform real inference using modelRoot
        return "DUMMY_PREDICTION_FOR_" + input;
    }

    public boolean isReady() {
        return modelRoot != null;
    }

    public Path getModelRoot() {
        return modelRoot;
    }
}
```

4. Run initialization logic at application startup

Use `CommandLineRunner` so the app only starts serving once the model is downloaded and loaded. [2] [6]

ModelBootstrapRunner.java

```
package com.example.inference.bootstrap;

import java.nio.file.Path;

import com.example.inference.model.ModelDownloadService;
import com.example.inference.model.ModelExtractor;
import com.example.inference.model.ModelService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class ModelBootstrapRunner implements CommandLineRunner {

    private static final Logger log = LoggerFactory.getLogger(ModelBootstrapRunner.class)

    private final ModelDownloadService downloadService;
    private final ModelExtractor extractor;
    private final ModelService modelService;

    @Value("${model.local.dir}")
    private String localDir;

    public ModelBootstrapRunner(ModelDownloadService downloadService,
                                ModelExtractor extractor,
                                ModelService modelService) {
        this.downloadService = downloadService;
        this.extractor = extractor;
        this.modelService = modelService;
    }

    @Override
    public void run(String... args) throws Exception {
        log.info("Bootstrapping model at application startup");

        Path tarPath = downloadService.downloadModelTarGz();
        Path extractRoot = extractor.extractTarGz(tarPath, Path.of(localDir, "extracted")

        // Initialize model runtime
        modelService.initFromExtractedDir(extractRoot);

        if (!modelService.isReady()) {
            throw new IllegalStateException("Model failed to initialize");
        }

        log.info("Model bootstrap complete; service ready to accept requests");
    }
}
```

`CommandLineRunner` executes after the Spring context is fully initialized but before the app begins serving traffic, which is ideal for this initialization.[6] [2]

5. REST controller on port 8080

Expose basic inference and health endpoints using Spring Web.[7]

InferenceRequest.java

```
package com.example.inference.api;

public class InferenceRequest {
    private String input;

    public String getInput() { return input; }
    public void setInput(String input) { this.input = input; }
}
```

InferenceResponse.java

```
package com.example.inference.api;

public class InferenceResponse {
    private String prediction;

    public InferenceResponse(String prediction) {
        this.prediction = prediction;
    }

    public String getPrediction() { return prediction; }
}
```

InferenceController.java

```
package com.example.inference.api;

import com.example.inference.model.ModelService;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api")
public class InferenceController {

    private final ModelService modelService;

    public InferenceController(ModelService modelService) {
        this.modelService = modelService;
    }

    @PostMapping("/predict")
    public ResponseEntity<InferenceResponse> predict(@RequestBody InferenceRequest reques
        if (!modelService.isReady()) {
            return ResponseEntity.status(503).build();
        }
        String prediction = modelService.predict(request.getInput());
```

```
        return ResponseEntity.ok(new InferenceResponse(prediction));
    }

    @GetMapping("/health")
    public ResponseEntity<String> health() {
        return modelService.isReady()
                ? ResponseEntity.ok("READY")
                : ResponseEntity.status(503).body("MODEL_NOT_READY");
    }
}
```

With `server.port: 8080`, this gives you `POST /api/predict` listening on port 8080 once the model is bootstrapped.

6. IAM, packaging, and deployment notes

- Give the execution environment an IAM role with `s3:GetObject` for the specific bucket/key. [4]
- Externalize `bucket`, `key`, `region`, and `local.dir` via environment variables or profiles for dev/prod.[4]
- Package with `./gradlew bootJar` and run as `java -jar app.jar`; initialization will download and load the model before serving requests.[2]

If you share the framework inside `output.tar.gz` (e.g., XGBoost model file name, PyTorch checkpoint), I can replace the stub `ModelService.predict` with concrete loading and inference code.

⁂

1. https://www.javacodemonk.com/aws-java-sdk-2-s3-file-upload-download-spring-boot-c1a3e072
2. https://reflectoring.io/spring-boot-execute-on-startup/
3. http://java-tweets.blogspot.com/2012/07/untar-targz-file-with-apache-commons.html
4. https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/java_s3_code_examples.html
5. https://www.perplexity.ai/search/2a254285-0956-44d7-b347-9c7621621770
6. https://stackoverflow.com/questions/27405713/running-code-after-spring-boot-starts
7. https://www.perplexity.ai/search/af640e01-39a4-4f06-86f5-c867352d3b7d
8. https://sagemaker.readthedocs.io/en/v2.27.1/frameworks/tensorflow/deploying_tensorflow_serving.html
9. https://docs.aws.amazon.com/sagemaker/latest/dg/inference-pipelines.html
10. https://www.youtube.com/watch?v=UQWjKQe97Ew
11. https://www.youtube.com/watch?v=WIZeMBxtvgM
12. https://www.baeldung.com/java-aws-download-file-s3-url
13. https://www.linkedin.com/posts/syed-sameer-pasha-b5887822a_springboot-java-backenddevelopment-activity-7418688358329053184-MlcU
14. https://helicaltech.com/untar-tar-file-using-java-extract-untar-file-using-java/
15. https://github.com/aws/sagemaker-training-toolkit/issues/82

16. https://stackoverflow.com/questions/73072582/extract-model-saved-in-s3-bucket-as-tar-gz-format-to-sagemaker-notebook-instance
17. https://docs.aws.amazon.com/sagemaker/latest/dg/realtime-endpoints-deploy-models.html