AWS Services Integration with Spring Boot

1. Amazon S3 (File Storage and Retrieval)

Use Case: Store and retrieve files (images, documents).
 Dependency (AWS SDK v2):

```
<dependency>
    <groupId>software.amazon.awssdk</groupId>
    <artifactId>s3</artifactId>
    <version>2.17.28</version>
</dependency>
```

 S3 Service Class:

```
@Service
public class S3Service {
    private final S3Client s3Client;

    public S3Service() {
        this.s3Client = S3Client.builder()
                .region(Region.US_EAST_1)
                .build();
    }

    public String uploadFile(MultipartFile file, String bucketName) throws
IOException {
        String fileName = file.getOriginalFilename();
        s3Client.putObject(
                PutObjectRequest.builder()
                        .bucket(bucketName)
                        .key(fileName)
                        .build(),
                RequestBody.fromBytes(file.getBytes())
        );
        return "File uploaded successfully: " + fileName;
    }
}
```

---

2. Amazon RDS (Relational Database Service)

Use Case: Host MySQL/PostgreSQL databases for Spring Boot apps.

 Add RDS Dependency:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
</dependency>
```

 application.properties:

```
spring.datasource.url=jdbc:postgresql://<RDS_ENDPOINT>:5432/mydb
spring.datasource.username=admin
spring.datasource.password=yourpassword
spring.jpa.hibernate.ddl-auto=update
```

□ Entity and Repository Example:

```
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
}
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {
}
```

---

3. Amazon SQS (Simple Queue Service)

Use Case: Queue messages for decoupling components.

□ Add SQS Dependency:

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-aws-messaging</artifactId>
    <version>2.3.1.RELEASE</version>
</dependency>
```

□ Configuration (application.properties):

```
cloud.aws.credentials.access-key=your-access-key
cloud.aws.credentials.secret-key=your-secret-key
cloud.aws.region.static=us-east-1
cloud.aws.sqs.queue-name=myQueue
```

□ SQS Service:

```
@Service
public class SqsService {
    @Autowired
    private QueueMessagingTemplate queueMessagingTemplate;

    @Value("${cloud.aws.sqs.queue-name}")
    private String queueName;

    public void sendMessage(String message) {
        queueMessagingTemplate.convertAndSend(queueName, message);
        System.out.println("Message sent to SQS: " + message);
    }
}
```

---

4. Amazon DynamoDB (NoSQL Database)

Use Case: Use DynamoDB for fast NoSQL data storage.

 Dependency:

```
<dependency>
    <groupId>software.amazon.awssdk</groupId>
    <artifactId>dynamodb</artifactId>
    <version>2.17.28</version>
</dependency>
```

 DynamoDB Configuration and Service:

```
@Service
public class DynamoDbService {
    private final DynamoDbClient dynamoDbClient;

    public DynamoDbService() {
        this.dynamoDbClient = DynamoDbClient.builder()
                .region(Region.US_EAST_1)
                .build();
    }

    public void saveItem(String tableName, String id, String name) {
        Map<String, AttributeValue> item = new HashMap<>();
        item.put("id", AttributeValue.builder().s(id).build());
        item.put("name", AttributeValue.builder().s(name).build());

        PutItemRequest request = PutItemRequest.builder()
                .tableName(tableName)
                .item(item)
                .build();

        dynamoDbClient.putItem(request);
        System.out.println("Item saved to DynamoDB: " + id);
    }
}
```

---

5. AWS Lambda (Serverless Backend)

Use Case: Run Java functions without provisioning servers.

 Lambda Handler Example:

```
public class LambdaHandler implements RequestHandler<Map<String, String>,
String> {
    @Override
    public String handleRequest(Map<String, String> input, Context context)
{
        return "Hello " + input.get("name");
    }
}
```

 Deploy to Lambda:

- Package Spring Boot app as a **fat JAR** and upload it to AWS Lambda using the **AWS CLI** or AWS Console.

---

6. Amazon API Gateway (Expose REST APIs)

Use Case: Host APIs and integrate with Lambda or EC2.

 Steps:

1. Create API in API Gateway.
2. Configure Lambda as backend integration.
3. Deploy API and test.

---

7. Amazon SNS (Notifications and Alerts)

Use Case: Send email, SMS, or push notifications.

 Add SNS Dependency:

```
<dependency>
    <groupId>software.amazon.awssdk</groupId>
    <artifactId>sns</artifactId>
    <version>2.17.28</version>
</dependency>
```

 SNS Service:

```
@Service
public class SnsService {
    private final SnsClient snsClient;

    public SnsService() {
        this.snsClient = SnsClient.builder()
                .region(Region.US_EAST_1)
                .build();
    }

    public void publish(String message, String topicArn) {
        snsClient.publish(PublishRequest.builder()
                .message(message)
                .topicArn(topicArn)
                .build());
        System.out.println("Notification sent via SNS");
    }
}
```

Best Practices

- **Security:** Use IAM roles for EC2, Lambda, and ECS to manage access.
- **Monitoring:** Use **CloudWatch** for logging and monitoring.
- **Scaling:** Configure Auto Scaling Groups (ASG) for EC2.
- **Caching:** Use Amazon ElastiCache (Redis) for caching.
- **Containerization:** Deploy Spring Boot apps with Docker on ECS or EKS.