Writeup :

- A high-level algorithm and description for deploying a Spring Boot application on AWS
- using an EC2 instance:

Algorithm:

- Sign up for an AWS account if you don't have one.
- Prepare your Spring Boot application for deployment by creating a buildable JAR file.
- Create an EC2 instance on AWS to host your application.
- Connect to the EC2 instance using SSH.
- Transfer the Spring Boot JAR file to the EC2 instance.
- Run the Spring Boot application on the EC2 instance.
- Optionally, set up an Elastic Load Balancer (ELB) for high availability and scalability.
- Description:
- Sign up for an AWS account: Create an account on the AWS website if you don't have
- one. This will provide you access to the AWS Management Console.
- Prepare your Spring Boot application: Ensure that your Spring Boot application is ready
- for deployment. This means having a fully functional and buildable JAR file that contains
- your application and all its dependencies.
- Create an EC2 instance: In the AWS Management Console, navigate to the EC2
- dashboard. Click on "Launch Instance" to create a new EC2 instance. Select an appropriate
- Amazon Machine Image (AMI) based on your requirements and configure the instance
- settings, including instance type, security groups, storage, tags, and security keys.
- Connect to the EC2 instance: Once the instance is launched, use SSH to connect to the
- instance from your local machine. This will allow you to access the EC2 instance's
- command-line interface.
- Transfer the Spring Boot JAR file: Use a secure copy method like `scp` to transfer your
- Spring Boot JAR file from your local machine to the EC2 instance. This will enable you to
- deploy your application on the server.
- Run the Spring Boot application: After transferring the JAR file, SSH into the EC2
- instance. Navigate to the directory where you placed the JAR file and run the Spring Boot
- application using the `java -jar` command. This will start your application on the server.
- Optional: Set up an Elastic Load Balancer (ELB): For a more robust setup, you can create
- an Elastic Load Balancer (ELB) to distribute incoming traffic across multiple EC2 instances.
- This provides high availability and scalability to your application.

- By following these steps, you can successfully deploy your Spring Boot application on AWS
- using an EC2 instance. Remember to configure security settings, backups, and other
- necessary configurations based on your application's requirements.
- Deploying a Spring Boot application on AWS using an EC2 instance offers several
- advantages:
  - Flexibility and Scalability: AWS allows you to choose from a wide range of EC2
- instance types, enabling you to select the one that best fits your application's resource
- requirements. Additionally, you can easily scale your application by adding or
- removing instances based on demand.
  - Cost-Effectiveness: AWS offers a pay-as-you-go model, which means you only pay
- for the resources you consume. This cost-effective approach is especially beneficial
- for small to medium-sized businesses and startups, as it eliminates the need for
- upfront infrastructure investments.
  - Easy Setup and Configuration: Launching an EC2 instance on AWS is a
- straightforward process through the AWS Management Console. It comes with pre�configured AMIs that simplify the setup of operating systems and applications.
  - Security: AWS provides various security features, including Virtual Private Cloud
- (VPC) to isolate your resources, security groups to control inbound and outbound
- traffic, and the ability to configure access control using IAM (Identity and Access
- Management).
  - High Availability: By using multiple EC2 instances and load balancers, you can
- achieve high availability for your Spring Boot application. If one instance fails, the
- traffic can be automatically redirected to other healthy instances, ensuring continuous
- operation.
  - Global Reach: AWS has data centers in multiple regions around the world. You can
- deploy your application in different regions to reduce latency and serve users from
- their nearest data center.
  - Elastic Load Balancing: AWS provides Elastic Load Balancers (ELBs), which can
- distribute incoming traffic across multiple EC2 instances. This not only ensures high
- availability but also improves the performance of your application.
  - Monitoring and Analytics: AWS offers various monitoring tools and services like
- CloudWatch, which allow you to track the performance of your EC2 instances and set
- up alarms for any abnormal behavior.
  - Integration with Other AWS Services: AWS provides a wide array of services like
- RDS (Relational Database Service), S3 (Simple Storage Service), and more. These
- services can easily be integrated with your Spring Boot application, making it more
- powerful and feature-rich.

- Automated Deployment and Management: With AWS, you can use tools like AWS
- Elastic Beanstalk or AWS CodeDeploy to automate the deployment of your Spring
- Boot application and manage its lifecycle easily.
- Overall, deploying a Spring Boot application on AWS using an EC2 instance provides a
- robust and scalable infrastructure, along with a wide range of services to enhance the
- performance, security, and management of your application. It is a popular choice for
- developers and businesses looking to leverage cloud computing capabilities effectively.
- Disadvantages:
    - Manual Configuration: Initial setup and configuration of the EC2 instance require
- manual intervention, which can be time-consuming and prone to human errors.
    - Scalability Management: While EC2 instances can be scaled manually or through
- auto-scaling, managing the scaling process can be challenging, especially during
- unexpected traffic spikes.
    - Backup and Disaster Recovery: Managing backups and disaster recovery solutions
- for EC2 instances require additional setup and maintenance. Failure to implement
- proper backup strategies can lead to data loss in case of system failure.
    - Resource Underutilization: Depending on your application's traffic patterns, EC2
- instances may not be fully utilized at all times, leading to resource wastage.
    - Learning Curve: For developers new to AWS or cloud infrastructure, there can be a
- learning curve to understand AWS services, configurations, and best practices.
    - Networking Complexity: Setting up networking, configuring VPCs, subnets, and
- security groups can be complex, especially for applications with intricate networking
- requirements.
    - Security Misconfigurations: Misconfigurations in security groups or IAM roles can
- lead to potential security vulnerabilities, exposing your application and data to risks.
    - Region Selection: Choosing the right AWS region for hosting your EC2 instances is
- essential. Picking the wrong region can result in increased latency and compliance
- issues.
    - Limited Managed Services: While AWS offers a variety of managed services,
- deploying on EC2 requires more manual management compared to serverless options
- like AWS Lambda or container services like AWS Fargate.

- Vendor Lock-In: Deploying a Spring Boot application on AWS using EC2 may lead to vendor lock-in, making it challenging to migrate to another cloud provider in the

- future.
- Despite these disadvantages, many organizations still choose to deploy applications on AWS
- EC2 due to its flexibility, customization options, and ability to host a wide range of
- applications. It is essential to weigh the pros and cons based on your specific application
- requirements, team expertise, and long-term business goals before making a decision
-