SOFTWARE ENGINEERING

UNIT - 5

TOPIC – 1

RISK MANAGEMENT

Risk Management in Software Engineering

Risk management is the process of identifying, analyzing, and addressing potential problems that could disrupt a software project.

It ensures the project is completed on time, within budget, and with fewer surprises by reducing the impact of unexpected problems.

Why is Risk Management Important?

Risk management is important because it prepares teams to handle challenges effectively, ensuring a smoother workflow and successful project delivery.

Software projects often face unexpected problems. These might include bugs in the code, delays in getting resources, or changes in customer requirements. Risk management prepares teams to handle these issues smoothly.

Think of it like planning a big event. You think about possible problems like rain or guests arriving late, and you prepare solutions, such as setting up tents or having a backup plan. In software engineering, planning for risks ensures the project stays on track and avoids bigger problems later.

Without risk management, projects can become chaotic. Delays, higher costs, and unhappy stakeholders are common results when risks are ignored. Managing risks means teams are better equipped to face challenges and deliver successful results.

What is Risk Management?

Risk management means planning ahead for problems and taking steps to reduce their impact.

It involves:

1. Identifying what could go wrong.
2. Figuring out how serious the problem could be.
3. Deciding on ways to prevent or fix the problem.

For example, if there's a chance that a software feature won't work on older devices, the team can test early and adjust the feature if needed. By being proactive, the team avoids surprises and ensures the project runs smoothly.

Steps in Risk Management

Risk management steps are the actions teams take to find, assess, and handle problems in a project.

Managing risks is a step-by-step process. Here's how it works:

1. **Identify Risks**: Make a list of everything that could go wrong.
    - Example: The software might not work on all devices, or a team member might leave the project.
2. **Assess Risks**: Decide how likely each problem is and how serious it would be if it happened.
    - Example: A delay in testing might be likely and could slow down the entire project.
3. **Plan for Risks**: Create solutions to avoid or reduce the impact of risks.
    - Example: Train multiple people to handle critical tasks so work doesn't stop if someone leaves.
4. **Monitor Risks**: Keep an eye on risks throughout the project and check for new ones.
    - Example: Regular updates and testing can help spot problems early.
5. **Communicate Risks**: Share information about risks and plans with the team and stakeholders.
    - Example: If a delay is likely, inform everyone involved and explain the solution.

Risk Identification

Risk identification is the process of finding and listing potential problems that could affect the project.

This is the first step in risk management, where teams think about what could go wrong and record all possible risks. The goal is to make sure no risk is overlooked.

How to Identify Risks:

1. **Review Past Projects**: Look at issues faced in similar projects.
    - Example: A common issue might be delays during testing.
2. **Brainstorm with the Team**: Discuss potential problems together.
    - Example: The team might identify that a new tool could cause compatibility issues.
3. **Analyze the Project Plan**: Check for areas that depend on tight schedules or limited resources.
    - Example: A critical task might depend on hardware delivery, which could be delayed.
4. **Consult Experts**: Get insights from people experienced in similar projects.
    - Example: A senior developer might warn about specific technical risks.

Risk Projection

Risk projection, also called risk estimation, is predicting how likely each risk is to happen and how serious it could be.

Steps in Risk Projection:

1. **Describe the Risk**: Clearly explain what the risk is.
    - Example: "There is a risk of delays in hardware delivery."
2. **Estimate Likelihood**: Decide how likely it is for the risk to happen (e.g., high, medium, low).
    - Example: Hardware delays might have a medium likelihood.
3. **Assess Impact**: Determine how much trouble the risk would cause if it happens.

- Example: Hardware delays might have a high impact because they could delay testing.

4. **Prioritize Risks**: Focus on risks that are both likely and have a big impact.
   - Example: A table helps organize risks based on likelihood and impact for easier decision-making.

Risk Description	Likelihood	Impact	Priority
Hardware Delay	Medium	High	High
Minor Code Bugs	High	Low	Medium
Team Member Leaves	Low	High	Medium

SOFTWARE ENGINEERING

UNIT - 5

TOPIC – 2

RMMM AND RMMM PLAN

Principles of Risk Management in RMMM

RMMM (Risk Mitigation, Monitoring, and Management) is a structured approach to handling risks in software projects.

Principles of RMMM:

1. **Focus on High-Priority Risks**: Address risks that are most likely to happen or have the biggest impact first.
    - Example: Start by solving risks that could delay the entire project.
2. **Continuous Monitoring**: Keep track of risks throughout the project lifecycle.
    - Example: Use weekly meetings to review risks and spot new ones.
3. **Prepare Contingency Plans**: Have backup plans ready for critical risks.
    - Example: If a server crashes, switch to backup servers immediately.
4. **Involve the Team**: Ensure everyone understands the risks and the strategies to handle them.
    - Example: Share risk details with all team members so they are prepared.
5. **Document Everything**: Keep records of risks, mitigation plans, and outcomes.
    - Example: Use risk information sheets to track each risk.

Using a Risk Table to Track Risks

A risk table is a tool to organize risks by listing their likelihood, impact, and the plans to handle them.

A risk table simplifies risk tracking by summarizing key information in a clear format. It helps teams quickly see which risks need attention and what steps are planned to manage them.

Example of a Risk Table

Risk Name	Likelihood	Impact	Mitigation Plan
Bugs in Code	High	Medium	Regular code reviews and testing.
Team Member Leaves	Medium	High	Cross-train team members.
Server Downtime	Low	High	Set up backup servers.

Each row represents a specific risk. The "Likelihood" column shows how likely it is to occur (e.g., high, medium, low), and "Impact" shows how much trouble it might cause. The "Mitigation Plan" describes steps to prevent or reduce the risk's effect. This table acts as a quick reference to prioritize risks and ensure the team is prepared.

Risk Refinement

Risk refinement means breaking down large risks into smaller, detailed parts so they are easier to manage and address.

Why is Risk Refinement Important?

Big risks can be overwhelming to deal with. By refining them into smaller parts, teams can understand the root causes, create focused solutions, and monitor progress more effectively.

Steps in Risk Refinement:

1. **Analyze the Risk**: Look at the big risk and identify the specific issues causing it.
    - Example: If the risk is software crashing, the causes might include memory problems, coding errors, or hardware compatibility issues.
2. **Create Specific Solutions**: For each issue identified, plan solutions that directly address it.
    - Example: For memory issues, optimize how the program uses memory.
3. **Monitor Changes**: As the project moves forward, keep refining your understanding of the risk and update your plans as needed.
    - Example: If testing reveals new bugs, include those in the refined risk plan.

Risk Information Sheet

A risk information sheet is a document that records detailed information about a specific risk and how to handle it.

What Does a Risk Information Sheet Include?

1. **Risk Name**: A short title to identify the risk.
   - Example: "Server Crash Risk"
2. **Description**: A clear explanation of the risk.
   - Example: "The server may fail during high traffic periods, leading to downtime."
3. **Likelihood**: How likely the risk is to happen (e.g., high, medium, low).
   - Example: "Medium likelihood during peak usage."
4. **Impact**: How much trouble the risk might cause.
   - Example: "High impact because downtime would delay user transactions."
5. **Mitigation Plan**: Actions to reduce the risk or its effects.
   - Example: "Set up backup servers and monitor server load regularly."
6. **Contingency Plan**: What to do if the risk happens.
   - Example: "Switch traffic to backup servers immediately and notify the team."

Example of a Risk Information Sheet:

Risk Name	Description	Likelihood	Impact	Mitigation Plan
Bug Surge Risk	Risk of increased bugs affecting stability	High	Medium	Regular code reviews and tests
Server Crash	Servers might fail before launch	Medium	High	Backup servers and monitoring
Key Team Absence	Critical member may leave mid-project	Low	High	Cross-training and backups

The risk information sheet ensures that teams have all the necessary details about risks and are prepared to act quickly and effectively.

SOFTWARE ENGINEERING

UNIT - 5

TOPIC – 3

INTRODUCTION TO CLOUD AND ITS SERVICES

Introduction to Cloud Computing

Cloud computing is a way to use computer resources like storage, applications, and processing power over the internet instead of owning and managing physical hardware. It lets you access these resources whenever you need them, and you only pay for what you use.

What is Cloud Computing?

Cloud computing means you can:

- **Store files** like photos, videos, and documents on the internet, so you don't need a big hard drive at home.
- **Run programs** that require a lot of computing power without buying expensive computers.
- **Use software** like email or document editors directly through the internet without downloading or installing them.

Imagine renting a car instead of buying one. You use the car only when needed and pay for it based on how long you use it. Similarly, cloud computing gives you the resources you need without the hassle of owning them.

Features of Cloud Computing

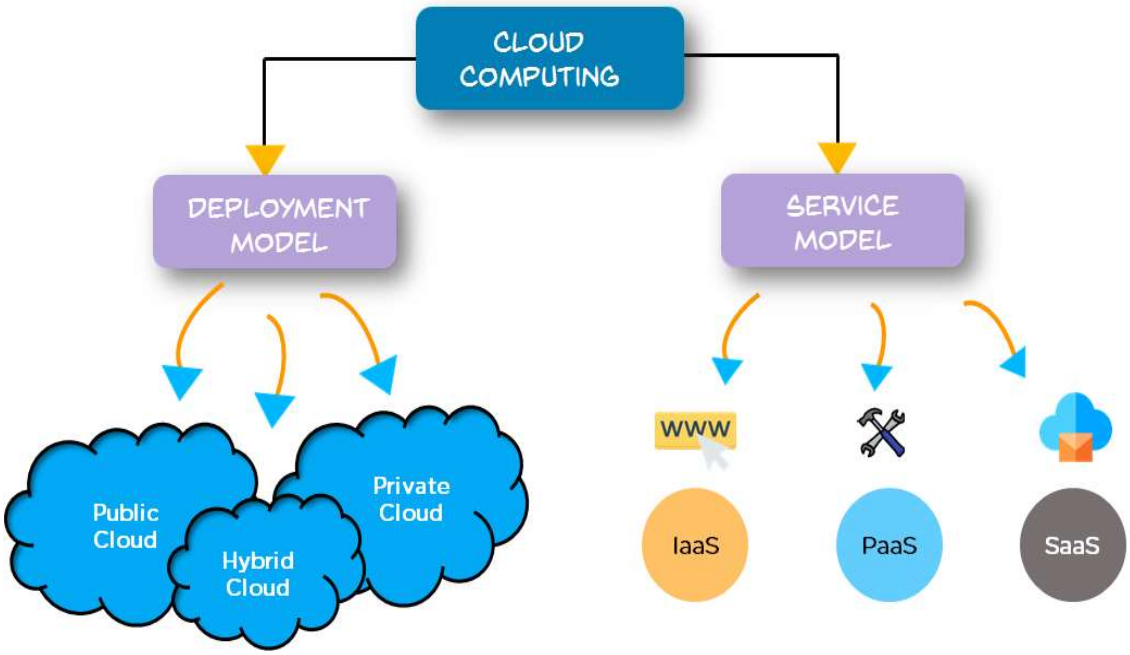
1. **Availability**: Your data and applications are always accessible. Cloud providers work hard to make sure there is little to no downtime.
    - *Example*: Like water from a tap—always ready whenever you need it.
2. **Scalability**: You can adjust the number of resources based on your needs. If you need more storage or computing power, you can get it instantly, and when you don't need as much, you can scale back down.

- *Example*: Like a balloon you can inflate or deflate depending on how much air you need.

3. **Pay-as-you-go**: You only pay for what you use, saving money by not having to invest in expensive hardware upfront.
    - *Example*: Like a phone bill, where you pay for the minutes and data you use.

Types of Cloud Computing

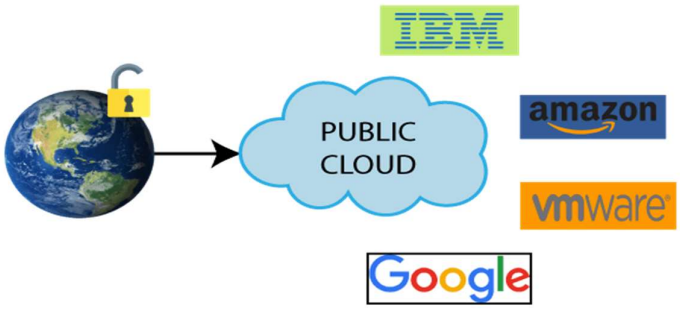
Cloud computing can be categorized based on **how it is deployed** and **the services it offers**.



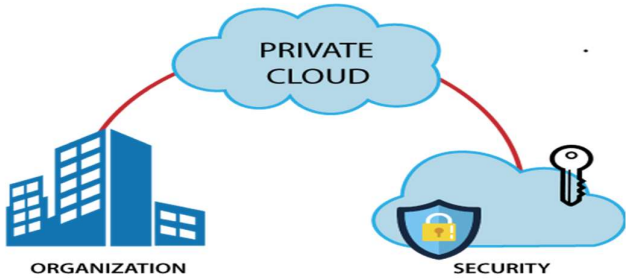
Cloud Deployments

Deployment refers to how and where cloud services are made available. The three types of deployments are:

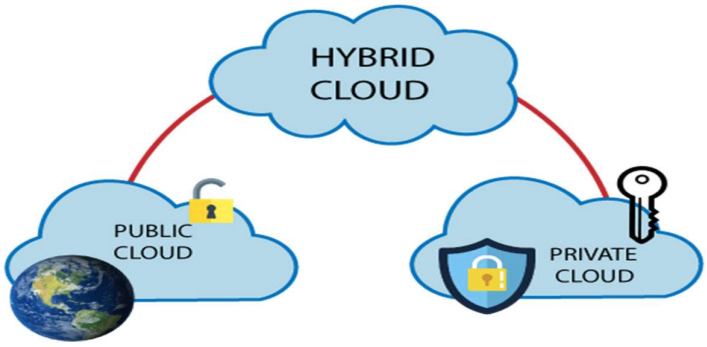
1. **Public Cloud**:
    - Resources like servers and storage are shared by multiple users and managed by companies like Amazon, Microsoft, and Google.
    - *Example*: A public library where anyone can borrow books.



2. **Private Cloud**:
   - Resources are used by a single organization, offering more control and security. It can be managed by the organization or a third-party provider.
   - *Example*: A private library owned by a school, only accessible to students and staff.

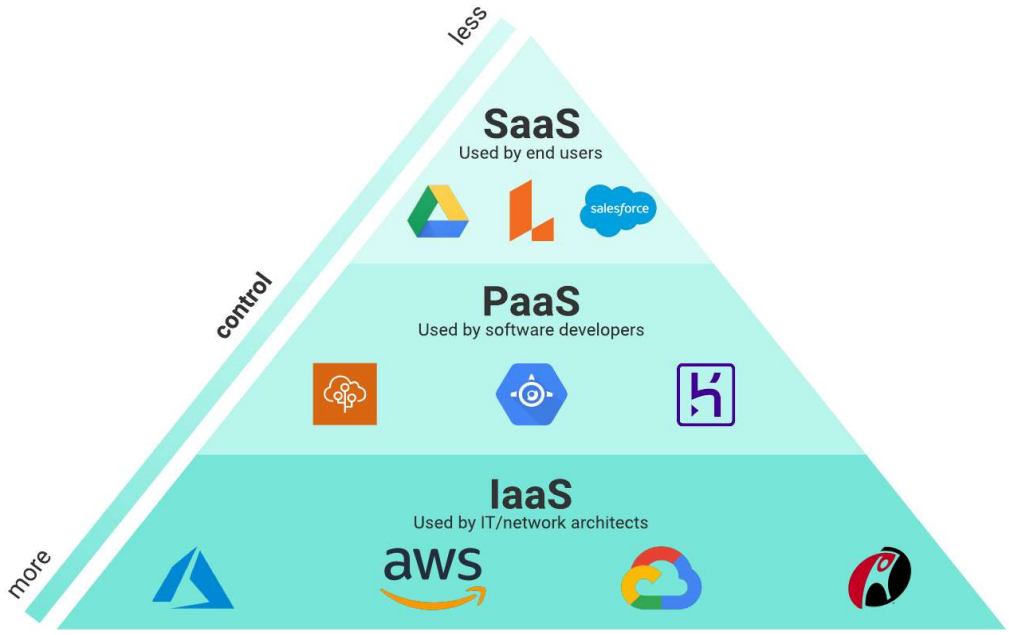


3. **Hybrid Cloud**:
   - A combination of public and private clouds. Organizations can use private clouds for sensitive tasks and public clouds for less critical activities.
   - *Example*: A bakery with a private kitchen for secret recipes and a public shop to sell the goods.



Cloud Services

Cloud services are categorized based on the level of control they provide:



1. **Infrastructure as a Service (IaaS)**:
   - Provides basic resources like virtual machines and storage. You are responsible for managing everything else.
   - *Example*: Renting an empty plot where you can build anything you want.
2. **Platform as a Service (PaaS)**:
   - Offers tools and platforms for developers to build applications without worrying about the underlying hardware.
   - *Example*: A fully equipped kitchen where you can cook without buying the appliances.
3. **Software as a Service (SaaS)**:
   - Provides ready-made software that you can use directly over the internet.
   - *Example*: Using Gmail or Microsoft Word online without installing anything on your computer.

How AWS Helps with Cloud Computing

AWS (Amazon Web Services) is a platform that offers cloud services for building, managing, and scaling applications easily. It removes the need to own physical servers, allowing you to focus on your projects.

Key AWS Services

1. **Amazon EC2**:
   - Let's you rent virtual servers to run applications.
   - *Example*: Like booking a hotel room based on your needs—a single room or a suite.
2. **Amazon S3**:
   - Provides storage space to save and manage your files securely.
   - *Example*: A giant online locker where you can keep your files.
3. **Amazon RDS**:
   - Offers managed databases, so you don't have to worry about setting them up or maintaining them.
   - *Example*: Like hiring an assistant to organize and maintain your files.
4. **Amazon VPC**:
   - Allows you to create a private and secure section of the cloud for your resources.
   - *Example*: A private office within a shared building where only authorized people can enter.
5. **AWS Lambda**:
   - Runs your code automatically when needed without requiring servers.
   - *Example*: A light sensor that turns on a light only when it detects motion.

Steps to Create Cloud Infrastructure Using AWS

Here is how you can set up infrastructure on AWS step by step:

1. **Sign Up**:
   - Go to the AWS website and register for an account. Fill in your details and set up a payment method.

2. **Log In**:
    - Use your account credentials to access the AWS dashboard.
3. **Explore Services**:
    - Familiarize yourself with AWS services like EC2 for virtual servers, S3 for storage, and VPC for networking.
4. **Launch a Virtual Server**:
    - Go to EC2, choose an instance type, select an operating system, and launch your virtual machine.
5. **Set Up Storage**:
    - Use S3 to create a storage bucket where you can upload and manage your files.
6. **Configure Networking**:
    - Use VPC to design a secure network for your application.
7. **Monitor Resources**:
    - Use CloudWatch to track and optimize your usage, ensuring efficient performance.
8. **Secure Access**:
    - Set up user roles and permissions using AWS IAM to control who can access your resources.

Deploying a Web Application

To deploy a static web application using Docker and Nginx, follow these steps:

Steps to Deploy

1. **Push Code to GitHub**:
    - Store your application's code on GitHub. Initialize a Git repository, commit the files, and push them to a GitHub repository.
2. **Create a Virtual Machine**:
    - Launch an EC2 instance with Ubuntu or another operating system. Configure storage and security groups for access.
3. **Clone the Code and Install Docker**:
    - Use Git to clone your application's code from GitHub onto the virtual machine.

- Install Docker on the virtual machine. Docker is a tool that helps package and run your application in containers.

4. **Write and Build a Dockerfile**:
   - Write a Dockerfile to define how your application will run. Build the Docker image using the `docker build` command.
5. **Run the Application**:
   - Use the `docker run` command to create a container from the Docker image. Map it to port 80 so it can be accessed from the internet.
   - Use the public IP address of your EC2 instance to access your web application.

SOFTWARE ENGINEERING LAB

EXERCISE – 8

TOPIC – 1

AWS ACADEMY LEARNING ACCOUNT CREATION

In this exercise, we will be:

- Set up an AWS Academy account to access cloud services for learning purposes.
- Navigate the AWS Academy Learner Lab to activate and explore its features.
- Create a free AWS account by completing email, billing, and phone verification steps.
- Access the AWS Management Console to explore services like EC2 for cloud-based tasks.
- Learn to start, manage, and terminate AWS lab sessions while adhering to usage limits.
- **Note: At every step take screenshots and save in a document**

AWS Academy Account Creation

1. **Check Your Email**
   - Look for an AWS Academy course invitation email.
   - Click on **Get Started**.
2. **Create Your Account**
   - Click **Create My Account**.
   - Set up a **password**.
   - Choose an Indian time zone (e.g., IST).
   - Check all the provided boxes.
   - Scroll down and click **Register**.
3. **Access the Account**
   - After account creation, click on **Account** in the window.
4. **Logout and Relogin Instructions**
   - To log out: Click **Logout**.

- To log back in:
  - Go to Google and search for **AWS Academy Login**.
  - From the search results, look for **Untitled** and click on it.
  - Select **Student Login**.
  - Enter your username and password, then click **Login**.

5. **Navigating the AWS Academy Learner Lab**
   - After logging in, click on **AWS Academy Learner Lab**.
   - Click on **Modules** and scroll down.
   - Select **AWS Academy Learner Lab**.
6. **Agree to Terms & Conditions**
   - Read through the terms and conditions provided.
   - Scroll down and click on **I Agree**.
7. **Start the Sandbox Environment**
   - If there's a **red dot** beside AWS, it indicates that the lab environment is in a stopped state.
   - Click **Start Lab** to activate the environment.
8. **Lab Environment Details**
   - Once the lab starts, you can use it for **4 hours**.
   - If additional time is needed, restart the lab to reset the timer.
   - You are provided with **$50 in free credits**:
     - **Important:** Exceeding this limit will block further AWS access using the same email ID.
   - Once AWS status turns **green**, click on it to access the AWS dashboard.
9. **Perform Exercises**
   - From the AWS dashboard, click on **EC2** (visible under the **Recently Visited** section).
   - Begin your lab exercises from this point.
10. **Ending the Lab**
    - After finishing your tasks, click **End Lab**.
    - Confirm by clicking **Yes**.
    - A **red dot** beside AWS indicates the lab environment has stopped.

The below is for your information, no need to take screenshots from here

Steps to Create a Free Tier AWS Account

1. **Visit the AWS Website**
   - Open aws.amazon.com.
   - Click **Create AWS Account**.
2. **Provide Account Details**
   - Enter your **email ID** and **name**.
   - A verification code will be sent to your email.
   - Enter the verification code on the website.
3. **Set Up Password**
   - Create and confirm a password for your account.
4. **Provide Contact Details**
   - Enter your contact information and agree to the **Terms and Conditions**.
   - Click **Create Account**.
5. **Billing Details**
   - Add your billing information for account verification.
   - AWS will temporarily charge **₹2** for verification purposes, which will be refunded once verification is complete.
6. **Phone Verification**
   - After billing verification, enter your **working contact number**.
   - Choose your **country** and click **Send SMS**.
   - Enter the verification code sent to your phone.
7. **Select a Plan**
   - Choose the **Basic Plan** (free).
   - Click **Complete Sign-Up**.
8. **Access the AWS Management Console**
   - After completing the sign-up, navigate to the **AWS Management Console**.
   - Select your role as **Academic/Researcher** and your interest as **DevOps**.
   - Click **Submit**.

Using the AWS Management Console

- Sign into the AWS console using your **Root User** credentials.
- The console dashboard will appear, where you can access and explore various AWS services.

SOFTWARE ENGINEERING LAB

EXERCISE – 8

TOPIC – 2

PROJECT DEPLOYMENT IN THE AWS CLOUD USING EC2 INSTANCE

In this exercise, we will be:

- Launch a virtual server (EC2 instance) on AWS.
- Install essential tools like Docker, Git, and Nano.
- Create and deploy a simple web application using Docker.
- Access the application online.
- Clean up resources to avoid unnecessary charges.

Note: At every step take screenshots and save in a document

Step 1: Log in to AWS and Go to EC2

In this step, we will log in to our AWS account and access the EC2 service.

1. Log in to your AWS account.
2. On the AWS homepage, click **Services**, then choose **EC2** under **Compute**.

Step 2: Launch an EC2 Instance

Here, we will set up a virtual server to host our web application.

1. Click **Launch Instance**.
2. Configure the settings as follows:
    - **Name**: Enter a name like "MyWebServer" to identify your server.
    - **Application and OS**: Choose **Ubuntu (Free Tier Eligible)**.

- **Instance Type**: Select **t2.micro** (1 CPU, 1 GB RAM).
- **Key Pair**: Create a new key pair, download the `.pem` file, and save it securely.
- **Network**: Enable **Allow HTTP/HTTPS traffic** to make your website accessible.
- **Storage**: Use the default 8 GB.

3. Click **Launch Instance** and wait until the status changes to "Running."

Step 3: Connect to the EC2 Instance

In this step, we will connect to our virtual server.

1. Select your instance, click **Connect**, and copy the **SSH command**.
2. Open **PowerShell** (Windows) or **Terminal** (Mac/Linux) on your computer.
3. Navigate to the folder where your `.pem` file is saved using the `cd` command.
4. Paste the SSH command and press Enter. Type "yes" if prompted.

Step 4: Prepare the Instance

Now, we will prepare the server by installing required tools.

1. **Update the system** to ensure all software is up to date:

```
sudo apt update
```

2. **Install Docker** to package and run our web application:

```
sudo apt-get install docker.io
```

3. **Install Git** to manage and download code:

```
sudo apt install git
```

4. **Install Nano** for editing files directly on the server:

```
sudo apt install nano
```

Step 5: Create Your Web Application

In this step, we will build a simple web page and upload it to GitHub.

1. On your computer, create a file named `index.html` and add the following content:

```
<html>
<head><title>My Webpage</title></head>
<body><h1>Hello from AWS!</h1></body>
</html>
```

2. Initialize Git in the file's folder:

```
git init
git add .
git commit -m "First commit"
```

3. Create a GitHub repository, copy its HTTPS URL, and upload your file:

```
git remote add origin <Your_Repo_URL>
git push -u origin main
```

Step 6: Deploy the Web Application Using Docker

Here, we will deploy the web application to the EC2 instance.

1. On the EC2 instance, clone your GitHub repository:

```
git clone <Your_Repo_URL>
```

2. Create a `Dockerfile` in the project folder using Nano:

```
nano Dockerfile
```

Add the following content:

```
FROM nginx:alpine
COPY . /usr/share/nginx/html
```

Save the file by pressing **Ctrl + O**, then Enter, and exit Nano with **Ctrl + X.**

3. Build and run the Docker container to serve the web application:

```
sudo docker build -t my-web-app .
sudo docker run -d -p 80:80 my-web-app
```

Step 7: Access Your Web Application

In this step, we will view the deployed web page online.

1. Copy the **Public IP Address** of your EC2 instance from the AWS console.
2. Paste it into your browser (e.g., **http://<Public_IP>).**
3. You'll see your web page with the message "Hello from AWS!" displayed.

Step 8: Clean Up

Finally, we will clean up resources to avoid any charges.

1. Stop the running Docker container:

```
sudo docker ps
sudo docker stop <Container_ID>
```

2. Terminate the EC2 instance in the AWS console by selecting it, clicking **Instance State**, and choosing **Terminate Instance**.

SOFTWARE ENGINEERING LAB

EXERCISE – 8

TOPIC – 3

MAVEN WEB PROJECT DEPLOYMENT IN THE AWS CLOUD USING EC2 INSTANCE

In this exercise, we will be:

- Launch an EC2 instance on AWS.
- Install Docker, Git, and Nano.
- Deploy a Maven web project using Docker and expose it on port 9090.
- Access the application online.
- Clean up resources to avoid unnecessary charges.

Note:

1. At every step take screenshots and save in a document.

2. This guide uses JDK 11, which was used for developing the project. If your project was developed with JDK 21, adjust the Dockerfile as instructed in Step 5.

Step 1: Launch an EC2 Instance

In this step, we will set up a virtual server to host our Maven web project.

1. **Log in to AWS**: Access your AWS account and navigate to **Services > Compute > EC2**.
2. **Launch the instance**:
   - **Name**: Enter a descriptive name, e.g., `MavenWebProjectServer`.
   - **AMI**: Select **Ubuntu Server (Free Tier Eligible)**.
   - **Instance Type**: Choose **t2.micro**.
   - **Key Pair**: Create a key pair or use an existing one. Save the `.pem` file securely.
   - **Network Settings**: Enable **Allow HTTP/HTTPS traffic**.

- **Storage**: Use the default size (8 GB).

3. Click **Launch Instance** and wait for the status to change to "Running."
4. **Note down the Public IP Address** from the EC2 dashboard.

Step 2: Connect to the EC2 Instance

In this step, we will connect to the server.

1. Open **PowerShell** (Windows) or **Terminal** (Mac/Linux) and navigate to the folder with the `.pem` file using the `cd` command.
2. Use SSH to connect to the instance:

   ```
   ssh -i "<KeyFile>.pem" ubuntu@<Public_IP>
   ```

   Replace `<KeyFile>` with the `.pem` file name and `<Public_IP>` with your instance's public IP.

3. If prompted, type "yes" to confirm the connection.

Step 3: Prepare the EC2 Server

Now, we will install the necessary tools.

1. **Update the system**:

   ```
   sudo apt update
   ```

2. **Install Docker**:

   ```
   sudo apt-get install docker.io -y
   ```

3. **Install Git**:

   ```
   sudo apt install git -y
   ```

4. **Install Nano** (text editor):

   ```
   sudo apt install nano -y
   ```

Step 4: Clone Your Maven Web Project

In this step, we will download the Maven project from GitHub.

1. Go to your GitHub repository, click **Code > HTTPS**, and copy the URL.
2. Clone the repository:

   ```
   git clone <Your_Repo_URL>
   ```

   Replace `<Your_Repo_URL>` with the copied URL.

Step 5: Create a Dockerfile

We will create a Dockerfile to containerize the Maven project.

1. Navigate to the project folder:

   ```
   cd <Your_Project_Folder>
   ```

   Replace `<Your_Project_Folder>` with the folder name.

2. Open Nano to create the Dockerfile:

   ```
   nano Dockerfile
   ```

3. Add the following content based on the JDK version used during development:
   - For **JDK 11** (used in this guide):

     ```
     FROM tomcat:9-jdk11
     COPY target/*.war /usr/local/tomcat/webapps/
     ```

   - For **JDK 21**:

     ```
     FROM tomcat:9-jdk21
     COPY target/*.war /usr/local/tomcat/webapps/
     ```

4. Save and exit Nano: Press `Ctrl + O`, then `Enter`, and `Ctrl + X`.

Explanation:

- `FROM tomcat:9-jdk11` or `FROM tomcat:9-jdk21` specifies the Tomcat base image with the appropriate JDK version.
- `COPY target/*.war /usr/local/tomcat/webapps/` copies the `.war` file into the webapps directory of Tomcat for deployment.

Step 6: Build and Run the Docker Container

1. **Build the Docker image**:

   ```
   sudo docker build -t maven-web-project .
   ```

2. **Run the container**:

   ```
   sudo docker run -d -p 9090:8080 maven-web-project
   ```

   - `-d:` Runs the container in the background.
   - `-p 9090:8080:` Maps port 9090 on your instance to port 8080 in the container.

Step 7: Configure Security Group for Port 9090

We will ensure the EC2 instance allows traffic on port 9090.

1. In the AWS EC2 dashboard, go to **Security** and click the **Security Group ID**.
2. Add an inbound rule:
   - **Type**: Custom TCP
   - **Port Range**: 9090
   - **Source**: Anywhere (0.0.0.0/0) or your IP.
3. Save the changes.

Step 8: Access the Web Application

We will now test the deployment.

1. Open a browser and navigate to:

```
http://<Public_IP>:9090/<Your_Project_Name>
```

Replace **<Public_IP>** with the instance's public IP and **<Your_Project_Name>** with your Maven project name.

Step 9: Clean Up

Finally, we will stop the container and terminate the instance.

1. **Stop the Docker container**:

   ```
   sudo docker ps
   sudo docker stop <Container_ID>
   ```

   Replace **<Container_ID>** with the container ID.

2. **Terminate the EC2 instance** In the EC2 dashboard, go to **Instance State** and select **Terminate Instance**.