Independent Skill Development Mission

Independent Skill Development Mission NXT
EMPOWERING FUTURES, SHAPING TOMORROW

ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION

DevOps & Deployment (Weeks 31-36)

Git Basics (Commit, Branching, Merging)

Chapter 1: Introduction to Git

1.1 What is Git?

Git is a **distributed version control system (VCS)** that helps developers track changes in their code, collaborate efficiently, and manage different versions of a project.

- **Why Use Git?**

✓ **Tracks changes** and maintains a history of edits.
✓ **Supports collaboration** by allowing multiple developers to work on a project.
✓ **Enables branching & merging** for working on different features independently.

- **Common Git Commands**

Command	Description	Example
git init	Initialize a Git repository	git init
git clone	Copy a remote repository	git clone repo_url

git add	Stage changes for commit	git add file.txt
git commit	Save changes to history	git commit -m "Initial commit"
git status	Check changes in the working directory	git status
git branch	Manage branches	git branch feature-branch
git merge	Merge branches	git merge feature-branch

CHAPTER 2: INITIALIZING & SETTING UP GIT

2.1 Configuring Git for First Use

📌 **Set up Git with your username and email:**

git config --global user.name "Your Name"

git config --global user.email "your-email@example.com"

✓ Associates commits with your identity.

📌 **Check Git Configuration:**

git config --list

✓ Displays current Git settings.

2.2 Initializing a New Git Repository

📌 **Create a New Repository Locally:**

mkdir my-project

cd my-project

git init

✓ Initializes an empty .git directory in **my-project**.

📌 **Check Repository Status:**

git status

✓ Displays **untracked or modified files**.

2.3 Cloning an Existing Repository

📌 **Clone a Remote Repository from GitHub:**

git clone https://github.com/user/repository.git

✓ Copies an **existing project** to your local machine.

CHAPTER 3: MAKING CHANGES & COMMITTING CODE

3.1 Staging and Committing Changes

📌 **Add Files to Staging Area:**

git add file.txt

📌 **Add All Files at Once:**

git add .

✓ Stages **all modified and new files**.

📌 **Commit the Staged Changes:**

git commit -m "Added new feature"

✓ Saves the changes with a **message describing the update**.

📌 **View Commit History:**

git log --oneline

✓ Shows **previous commits** in a compact format.

3.2 Undoing Changes (Reset, Checkout, Revert)

📌 **Unstage a File Before Committing:**

git reset file.txt

📌 **Undo the Last Commit (Before Pushing):**

git reset --soft HEAD~1

✓ Moves the commit back to staging **without deleting changes**.

📌 **Revert a Commit After Pushing:**

git revert HEAD

✓ Creates a **new commit to undo previous changes**.

CHAPTER 4: BRANCHING IN GIT

4.1 What is Branching?

Branching allows developers to **work on different features independently** without affecting the main codebase.

🔹 **Why Use Branching?**

✓ Develop **features without breaking the main project**.
✓ Work **collaboratively on different tasks**.
✓ Allows **safe experimentation** with new ideas.

4.2 Creating & Switching Branches

📌 **List All Branches:**

git branch

📌 **Create a New Branch:**

git branch feature-branch

📌 **Switch to the New Branch:**

git checkout feature-branch

📌 **Create & Switch in One Command:**

git checkout -b feature-branch

✓ Creates **a new branch and switches to it immediately**.

4.3 Merging Branches

📌 **Switch to the Main Branch:**

git checkout main

📌 **Merge the Feature Branch into Main:**

git merge feature-branch

✓ Combines **changes from feature-branch into main**.

4.4 Handling Merge Conflicts

🔹 **Merge conflicts occur when multiple changes affect the same file.**

📌 **Example Conflict Resolution in a File:**

<<<<<<< HEAD

console.log("Main branch code");

=======

console.log("Feature branch code");

>>>>>>> feature-branch

✓ Manually **edit the file** to keep the correct changes.

📌 **Mark Conflict as Resolved and Commit:**

git add conflicted-file.txt

git commit -m "Resolved merge conflict"

✓ **Fix conflicts, stage, and commit** the resolved file.

CHAPTER 5: WORKING WITH REMOTE REPOSITORIES (GITHUB)

5.1 Pushing Local Changes to GitHub

📌 **Add Remote Repository URL:**

git remote add origin https://github.com/user/repository.git

📌 **Push Changes to GitHub:**

git push origin main

✓ Uploads local commits to **GitHub's main branch**.

📌 **Push a Specific Branch to GitHub:**

git push origin feature-branch

✓ Uploads **only feature-branch**.

5.2 Pulling Updates from GitHub

📌 **Fetch the Latest Changes:**

git fetch origin

📌 **Merge Remote Changes into Local Code:**

git pull origin main

✓ Ensures your **local branch is up to date**.

5.3 Forking & Pull Requests

🔹 **Forking**: Copies a GitHub repository to your account for independent work.

🔹 **Pull Requests**: Requests to **merge changes from a branch into the main branch**.

📌 **Steps for a Pull Request (PR) Workflow:**

1. **Fork the Repository** on GitHub.
2. **Clone the Forked Repo** to your local machine.
3. **Create a New Branch:** git checkout -b new-feature
4. **Make Changes & Commit:** git commit -m "Added new feature"
5. **Push to GitHub:** git push origin new-feature
6. **Create a Pull Request on GitHub**.

✓ Pull Requests allow **code review before merging** into the main branch.

Case Study: How Microsoft Uses Git for Large-Scale Development

Challenges Faced by Microsoft

✓ Managing **millions of lines of code** across teams.
✓ Handling **simultaneous contributions from thousands of developers**.

Solutions Implemented

✓ Used **branching strategies** like main, develop, and feature branches.
✓ Integrated **automated testing & CI/CD pipelines**.
✓ Implemented **code reviews through pull requests (PRs)**.

🔹 **Key Takeaways from Microsoft's Git Workflow:**
✓ **Branching strategies improve code management.**
✓ **Pull requests ensure quality control before merging.**
✓ **Automated CI/CD improves deployment speed & efficiency.**

📝 Exercise

✅ Create a **Git repository** and initialize a new project.
✅ Make changes, **stage**, and **commit** them.
✅ Create a **new branch**, switch to it, and modify a file.
✅ Merge the branch into main and **resolve merge conflicts** if needed.
✅ Push changes to **GitHub** and create a **pull request**.

Conclusion

✓ **Git enables efficient version control for teams & individuals.**
✓ **Committing, branching, and merging ensure smooth collaboration.**

✓ **Remote repositories allow code sharing & backups.**

✓ **Pull requests and CI/CD workflows improve software development quality.**

WORKING WITH GITHUB PULL REQUESTS

CHAPTER 1: INTRODUCTION TO GITHUB PULL REQUESTS (PRS)

1.1 What is a Pull Request (PR)?

A **Pull Request (PR)** is a request to **merge code changes** from one branch into another in a GitHub repository. PRs allow developers to **collaborate, review, and merge code safely** into the main project.

- **Why Use Pull Requests?**

✓ Enables **code reviews** before merging changes.
✓ Allows **collaboration across teams** on GitHub.
✓ Provides a **history of changes and discussions**.
✓ Ensures **code quality through approvals**.

- **Basic Workflow of a Pull Request:**

1. **Fork or clone a repository** (if working on an open-source project).
2. **Create a new branch** to work on a feature or fix.
3. **Make code changes** and commit them.
4. **Push the branch to GitHub**.
5. **Open a Pull Request** on GitHub.
6. **Request a review from team members**.
7. **Make necessary changes based on feedback**.
8. **Merge the PR into the main branch** once approved.

CHAPTER 2: CREATING A PULL REQUEST ON GITHUB

2.1 Forking a Repository (For Open Source Contributions)

If contributing to an **open-source project**, you need to **fork the repository first**.

📌 **Steps to Fork a Repository:**

1. Go to the GitHub repository you want to contribute to.
2. Click **"Fork"** in the top-right corner.
3. This creates a **copy of the repository** in your GitHub account.
4. Clone it locally:
5. git clone https://github.com/your-username/repository-name.git
6. cd repository-name

2.2 Creating a New Branch

Before making changes, always create a **new branch**.

📌 **Steps to Create a Branch:**

git checkout -b feature-branch

✓ feature-branch is the name of the new branch for development.

2.3 Making Changes & Committing Code

📌 **Modify files and save your changes, then add and commit them:**

git add .

git commit -m "Added a new feature"

✓ git add . stages all changes.

✓ git commit -m saves the changes with a message.

2.4 Pushing the Branch to GitHub

📌 **Send the changes to GitHub:**

git push origin feature-branch

✓ This uploads feature-branch to your GitHub repository.

2.5 Opening a Pull Request on GitHub

📌 **Steps to Create a PR:**

1. Go to your **GitHub repository**.
2. Click the **"Compare & pull request"** button.
3. Ensure the **base branch** (e.g., main) and **feature branch** are correct.
4. Add a **title and description** of your changes.
5. Click **"Create Pull Request"**.

✓ The PR is now **open and ready for review**.

CHAPTER 3: REVIEWING A PULL REQUEST

3.1 How to Review a PR as a Team Member

If you're reviewing a teammate's PR:

1. Go to the **Pull Requests tab** in GitHub.
2. Click on the **PR to review**.

3. Scroll down to see **code changes**.
4. Click **"Files changed"** to add comments or suggestions.
5. Approve, request changes, or reject the PR.

📌 **Approving a PR:**
Click **"Approve"** if the code is ready for merging.

📌 **Requesting Changes:**
If improvements are needed, click **"Request changes"** and add comments.

CHAPTER 4: MERGING A PULL REQUEST

4.1 When to Merge a PR?

✓ When all **reviewers approve the changes**.
✓ When the **code is tested and works correctly**.
✓ When **all checks (CI/CD, tests) pass**.

📌 **Merging a PR in GitHub:**

1. Go to the **Pull Request page**.
2. Click **"Merge pull request"**.
3. Confirm by clicking **"Confirm merge"**.

📌 **After merging, delete the feature branch:**

```
git branch -d feature-branch
```

```
git push origin --delete feature-branch
```

✓ This keeps the **repository clean**.

CHAPTER 5: KEEPING YOUR LOCAL REPOSITORY UPDATED

5.1 Fetching the Latest Changes

After a PR is merged, update your local repository:

git checkout main

git pull origin main

✓ git pull ensures your local repository is **up to date**.

5.2 Rebasing a Branch (If PR is Outdated)

If another PR is merged before yours, **rebase** to prevent conflicts:

git checkout feature-branch

git pull --rebase origin main

✓ This applies **latest changes to your feature branch**.

Case Study: How Facebook Uses Pull Requests for Code Review

Challenges Faced by Facebook

✓ Managing **thousands of daily code changes** across teams.
✓ Ensuring **code quality before deploying updates**.

Solutions Implemented

✓ Used **GitHub Pull Requests** for peer-reviewed code merging.
✓ Implemented **CI/CD pipelines to test PRs before merging**.
✓ Required **approval from senior developers before merging major updates**.

- **Key Takeaways from Facebook's PR Strategy:**

✓ Code reviews improve collaboration and maintainability.

✓ **Automated testing catches errors before merging.**

✓ **Proper branch management prevents conflicts.**

📝 Exercise

✅ Create a **new feature branch** and push changes to GitHub.

✅ Open a **Pull Request (PR) with a proper description**.

✅ Request **code reviews from teammates** before merging.

✅ Merge the PR and **delete the feature branch** after approval.

Conclusion

✓ **Pull Requests allow collaborative development and code reviews.**

✓ **Branching helps keep the main codebase stable.**

✓ **CI/CD tools automate PR testing before merging.**

✓ **Following best practices ensures clean and efficient workflows.**

DEPLOYING ON GITHUB PAGES, NETLIFY, AND VERCEL

CHAPTER 1: INTRODUCTION TO DEPLOYMENT PLATFORMS

1.1 Why Deploy Web Applications?

Deployment makes a **web application accessible online** so users can interact with it. Hosting platforms like **GitHub Pages, Netlify, and Vercel** provide **fast, scalable, and cost-effective deployment solutions**.

🔹 **Comparison of Hosting Platforms:**

Feature	GitHub Pages	Netlify	Vercel
Best For	Static sites (HTML, CSS, JS)	React, Vue, JAMstack	Next.js, Full-stack apps
Backend Support	❌ No backend	⚡ Serverless Functions	⚡ Serverless Functions
Custom Domains	✓ Free	✓ Free	✓ Free
CI/CD (Auto Deployment)	✓ GitHub only	✓ GitHub, GitLab, Bitbucket	✓ GitHub, GitLab, Bitbucket
SSL (HTTPS)	✓ Free	✓ Free	✓ Free

📌 **Choose the best platform based on project requirements:**

✓ Use **GitHub Pages** for simple **static websites**.

✓ Use **Netlify** for **React/Vue** projects with serverless functions.
✓ Use **Vercel** for **Next.js & full-stack applications**.

CHAPTER 2: DEPLOYING ON GITHUB PAGES

2.1 Setting Up GitHub Pages for Static Websites

📌 Step 1: Create a GitHub Repository

1. **Go to GitHub** → Click **New Repository**.
2. Name the repository (e.g., my-website).
3. Initialize with a **README.md** (optional).

📌 Step 2: Upload Your Website Files

- Upload your **HTML, CSS, and JavaScript** files.
- Ensure index.html is in the root directory.

📌 Step 3: Enable GitHub Pages

1. Go to **Settings** → Scroll to **Pages**.
2. Select **Branch: main or master** → Click **Save**.
3. Your site is now live at:
4. https://your-username.github.io/my-website/

✓ GitHub Pages **automatically hosts your static website**!

2.2 Deploying a React App on GitHub Pages

📌 Step 1: Install GitHub Pages Package

npm install gh-pages --save-dev

📌 Step 2: Update package.json

```
"homepage": "https://your-username.github.io/my-react-app",
"scripts": {
  "predeploy": "npm run build",
  "deploy": "gh-pages -d build"
}
```

✓ **"homepage"** sets the GitHub Pages URL.
✓ **"deploy"** pushes the built files to GitHub Pages.

📌 Step 3: Deploy to GitHub Pages

```
git add .
git commit -m "Deploy React App"
git push origin main
npm run deploy
```

✓ Your React App is now live on **GitHub Pages**!

CHAPTER 3: DEPLOYING ON NETLIFY

3.1 Deploying a React App on Netlify

📌 Step 1: Create a Netlify Account

- Sign up at Netlify using **GitHub**.

📌 Step 2: Connect GitHub Repository

1. Click **New Site → Import from GitHub**.
2. Select **your repository** (React project).

📌 Step 3: Set Build Commands

- Build command:
- npm run build
- Publish directory:
- build

📌 Step 4: Deploy & Get Live URL

- Click **Deploy Site**.
- Netlify provides a **free live URL** (e.g., my-app.netlify.app).

3.2 Deploying with Netlify CLI

📌 Step 1: Install Netlify CLI

npm install -g netlify-cli

📌 Step 2: Login to Netlify

netlify login

📌 Step 3: Deploy the React App

npm run build

netlify deploy --prod --dir=build

✓ Your **React app is now deployed on Netlify!**

CHAPTER 4: DEPLOYING ON VERCEL

4.1 Deploying a React App on Vercel

📌 Step 1: Install Vercel CLI

npm install -g vercel

📌 **Step 2: Login to Vercel**

vercel login

📌 **Step 3: Deploy the React App**

vercel

✓ Vercel **automatically detects React and deploys it**.

📌 **Step 4: Get Live URL**

- Vercel assigns a **public URL** (e.g., my-app.vercel.app).

4.2 Deploying a Next.js App on Vercel

📌 **Step 1: Deploy Next.js on GitHub & Connect Vercel**

- Push your **Next.js project** to GitHub.
- Sign in to Vercel → Click **New Project**.
- Select **GitHub Repository**.
- Vercel **automatically detects Next.js**.
- Click **Deploy**.

✓ Vercel provides **automatic deployments on GitHub commits**.

CHAPTER 5: CUSTOM DOMAINS & SSL SETUP

5.1 Adding a Custom Domain on Netlify & Vercel

📌 **Netlify:**

1. Go to **Domain Settings**.

2. Click **Add Custom Domain** → Enter mywebsite.com.
3. Update **DNS settings** in domain provider.

📌 **Vercel:**

1. Go to **Settings** → **Domains**.
2. Click **Add Custom Domain** → Enter mywebsite.com.
3. Configure **DNS settings** with the provider.

✓ Both Netlify & Vercel provide **free SSL (HTTPS)** for security.

Case Study: How GitHub Pages, Netlify & Vercel Power Real-World Apps

Challenges Faced by Developers

✓ Deploying websites quickly **without server management**.
✓ Ensuring **fast content delivery** and scalability.
✓ Handling **automatic updates from GitHub**.

Solutions Implemented

✓ **GitHub Pages hosts open-source documentation** (e.g., React.js Docs).
✓ **Netlify deploys static sites with API integration** (e.g., JAMstack apps).
✓ **Vercel optimizes Next.js apps for real-time updates** (e.g., e-commerce sites).

- **Key Takeaways from Industry Deployment Strategies:**

✓ **Choose the right platform based on app requirements**.
✓ **Use Git-based deployment for continuous integration**.
✓ **Optimize performance with caching & CDNs**.

📝 Exercise

✅ Deploy a **static HTML site** using **GitHub Pages**.

✅ Deploy a **React app** using **Netlify** with automatic GitHub deployment.

✅ Deploy a **Next.js app** on **Vercel** and add a **custom domain**.

✅ Enable **HTTPS** for security on all platforms.

Conclusion

✓ **GitHub Pages is best for static sites** and documentation.

✓ **Netlify is great for React and Vue apps** with API integration.

✓ **Vercel is optimized for Next.js** and full-stack applications.

✓ **Using CI/CD (Continuous Integration/Deployment) improves workflow efficiency.**

SETTING UP A CONTINUOUS INTEGRATION (CI) PIPELINE

CHAPTER 1: INTRODUCTION TO CONTINUOUS INTEGRATION (CI)

1.1 What is Continuous Integration (CI)?

Continuous Integration (CI) is a **development practice** where developers frequently merge their code changes into a shared repository, which is then automatically tested and validated to detect errors early.

- **Why Use CI Pipelines?**

✓ **Early Bug Detection** → Tests run automatically on new code.
✓ **Improved Code Quality** → Ensures consistent coding standards.
✓ **Faster Deployment** → Automates build and testing processes.
✓ **Team Collaboration** → Encourages developers to commit changes frequently.

- **CI vs. CD (Continuous Deployment/Delivery)**

Process	Description	Example Tools
Continuous Integration (CI)	Automatically test and validate code changes	GitHub Actions, Travis CI, Jenkins
Continuous Deployment (CD)	Automatically deploy tested code to production	GitHub Actions, CircleCI, AWS CodeDeploy

CHAPTER 2: CHOOSING A CI/CD TOOL

2.1 Popular CI/CD Tools

🔹 Choosing the Right Tool for Your Project

CI/CD Tool	Best For	Features
GitHub Actions	Open-source projects	Built into GitHub, integrates with workflows
Travis CI	Small to mid-scale projects	Simple configuration, supports open-source projects
CircleCI	Scalable pipelines	Fast builds, parallel testing
Jenkins	Enterprise projects	Highly customizable, requires setup
GitLab CI/CD	GitLab repositories	Integrated directly with GitLab

📌 **We will use GitHub Actions for this tutorial** as it is **easy to set up** and **integrates directly with GitHub repositories**.

CHAPTER 3: SETTING UP A CI PIPELINE WITH GITHUB ACTIONS

3.1 Creating a GitHub Actions Workflow

📌 **Inside your project, create a .github/workflows/ci.yml file:**

```
name: CI Pipeline

on: [push, pull_request]

jobs:
  build:
    runs-on: ubuntu-latest
```

```
steps:
  - name: Checkout repository
    uses: actions/checkout@v3

  - name: Set up Node.js
    uses: actions/setup-node@v3
    with:
      node-version: 16

  - name: Install dependencies
    run: npm install

  - name: Run tests
    run: npm test
```

✓ **Triggers on every push and pull_request**.
✓ Uses **Node.js 16** for consistency.
✓ Runs **npm install and npm test** automatically.

3.2 Adding Unit Tests to Your Project

📌 **Install Jest for testing in Node.js projects:**

```
npm install --save-dev jest
```

📌 **Modify package.json to Include Jest:**

```
"scripts": {
 "test": "jest"
}
```

📌 **Create sum.js with a simple function:**

```
function sum(a, b) {
  return a + b;
}
module.exports = sum;
```

📌 **Create sum.test.js to test the function:**

```
const sum = require('./sum');

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```

📌 **Run tests locally:**

```
npm test
```

✓ Ensures the test passes **before pushing code** to GitHub.

CHAPTER 4: AUTOMATING BUILD AND DEPLOYMENT IN CI/CD

4.1 Deploying to a Web Server Automatically

📌 **Modify .github/workflows/ci.yml to Deploy to a Server**

```
jobs:
```

```
deploy:
  runs-on: ubuntu-latest
  needs: build
  steps:
   - name: SSH into Server & Deploy
     uses: appleboy/ssh-action@master
     with:
       host: ${{ secrets.SERVER_HOST }}
       username: ${{ secrets.SERVER_USER }}
       key: ${{ secrets.SERVER_KEY }}
       script: |
         cd /var/www/myapp
         git pull origin main
         npm install
         npm restart
```

✓ Automatically **logs into the server via SSH**.
✓ Pulls the latest code from GitHub and restarts the app.

📌 **Store SSH Credentials Securely**

1. Go to **GitHub Repository → Settings → Secrets**.
2. Add **SERVER_HOST, SERVER_USER, SERVER_KEY** as **repository secrets**.

4.2 Deploying to AWS S3 or Firebase

📌 Example: Deploying a React App to AWS S3 in CI/CD

```
jobs:
 deploy:
  runs-on: ubuntu-latest
  steps:
   - name: Deploy to AWS S3
     run: |
      aws s3 sync ./build s3://mybucket-name
```

✓ Uses AWS CLI to **sync build files to an S3 bucket**.

📌 Example: Deploying a Web App to Firebase Hosting

```
jobs:
 deploy:
  runs-on: ubuntu-latest
  steps:
   - name: Deploy to Firebase
     run: firebase deploy --token ${{ secrets.FIREBASE_TOKEN }}
```

✓ Deploys to **Firebase Hosting** automatically.

Case Study: How Netflix Uses CI/CD Pipelines

Challenges Faced by Netflix

✓ **Frequent code changes** needed automatic testing & deployment.
✓ **Handling large-scale deployments** without downtime.

Solutions Implemented

✓ Used **CI/CD pipelines** to deploy new features **without breaking services**.
✓ Automated **testing and rollback mechanisms** for failures.
✓ Implemented **blue-green deployment** to minimize downtime.

🔹 **Key Takeaways from Netflix's CI/CD Strategy:**
✓ **Automated testing ensures code quality.**
✓ **Continuous deployment reduces manual intervention.**
✓ **Scalable CI/CD pipelines improve application performance.**

📝 Exercise

✅ Set up a **GitHub Actions workflow** to test and deploy a project.
✅ Add **Jest unit tests** and integrate them into the pipeline.
✅ Configure a **deployment step** to push the app to a remote server.
✅ Use **GitHub Secrets** to store **API keys securely** in your CI/CD pipeline.

Conclusion

✓ **CI/CD pipelines automate testing and deployment, improving efficiency.**
✓ **GitHub Actions simplifies workflow automation.**
✓ **Automated testing ensures code quality before deployment.**
✓ **Secure environment variables protect sensitive credentials.**

HOSTING APPLICATIONS ON AWS EC2 & S3

CHAPTER 1: INTRODUCTION TO AWS HOSTING SERVICES

1.1 What is AWS EC2 & S3?

Amazon Web Services (**AWS**) provides **EC2 (Elastic Compute Cloud)** for hosting applications and **S3 (Simple Storage Service)** for storing static assets.

- **Why Use AWS for Hosting?**

✓ **Scalable** – Supports applications of all sizes.
✓ **Reliable** – High uptime and secure infrastructure.
✓ **Cost-Effective** – Pay-as-you-go pricing.

- **EC2 vs. S3: What's the Difference?**

AWS Service	Purpose	Example Use Case
EC2	Virtual servers to host applications	Deploy backend APIs, web apps
S3	Object storage for static files	Store images, videos, static websites

CHAPTER 2: SETTING UP AN EC2 INSTANCE

2.1 Creating an EC2 Instance

📌 **Steps to Launch an EC2 Instance:**

1. **Log in to AWS Console** → Go to **EC2 Dashboard**.

2. Click **"Launch Instance"**.
3. Choose an **Amazon Machine Image (AMI)**:
    - Ubuntu (Recommended for Node.js, Python).
    - Amazon Linux (Optimized for AWS).
4. Choose an **Instance Type** (Free-tier: t2.micro).
5. Configure **Security Group** (Allow **port 22, 80, and 443** for SSH & HTTP).
6. **Create & Download Key Pair** (.pem file for SSH access).
7. Click **Launch Instance**.

✓ **Instance is now running!**

2.2 Connecting to EC2 via SSH

📌 **Open Terminal and Run:**

```
ssh -i "your-key.pem" ubuntu@your-ec2-public-ip
```

✓ Logs into **EC2 instance using SSH**.

📌 **Update System & Install Node.js (Example for Web Apps):**

```
sudo apt update && sudo apt upgrade -y
sudo apt install nodejs npm -y
```

✓ Prepares the **EC2 instance for application deployment**.

2.3 Deploying a Node.js App on EC2

📌 **Clone a Git Repository into EC2:**

```
git clone https://github.com/your-repo.git
cd your-repo
npm install
```

✓ Downloads and installs **project dependencies**.

📌 **Run Node.js Server:**

```
node server.js
```

✓ **Starts the server,** but it stops when SSH disconnects.

2.4 Running the App in the Background (PM2 Process Manager)

📌 **Install PM2 and Start the Server:**

```
npm install -g pm2
pm2 start server.js
pm2 save
pm2 startup
```

✓ Keeps the **server running even after logout**.

CHAPTER 3: SETTING UP A STATIC WEBSITE ON AWS S3

3.1 Creating an S3 Bucket

📌 **Steps to Set Up S3 for Hosting:**

1. **Go to AWS S3 Dashboard** → Click **"Create Bucket"**.
2. **Enter a Unique Bucket Name**.
3. Choose **"Public Access"** → **Uncheck "Block all public access"**.

4. **Enable Static Website Hosting** in **Bucket Properties**.
5. Upload **HTML, CSS, and JavaScript Files**.
6. Copy the **S3 Website URL** from **Bucket Settings**.

✓ **Your static website is now live!**

3.2 Configuring S3 Bucket for Public Access

📌 **Update Bucket Policy to Allow Public Read Access:**

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::your-bucket-name/*"
    }
  ]
}
```

✓ Allows **public access to website files** in S3.

CHAPTER 4: CONNECTING EC2 WITH A DOMAIN (CUSTOM URL)

4.1 Assigning an Elastic IP to EC2

📌 **Steps to Get a Static IP for EC2:**

1. Go to **EC2 Dashboard** → Click **"Elastic IPs"**.
2. Click **"Allocate New Address"** → Associate it with your **EC2 Instance**.
3. Update DNS settings to point your domain to the **Elastic IP**.

✓ Now, your EC2 server has a **fixed public IP**.

4.2 Setting Up a Custom Domain with Route 53

📌 **Steps to Point a Domain to EC2 (Example: myapp.com)**

1. Go to **AWS Route 53** → Click **"Hosted Zones"**.
2. Click **"Create Record Set"** → Choose **Type: A (Address Record)**.
3. Set **"Value" to Your EC2 Elastic IP**.
4. Save changes and wait **for DNS propagation**.

✓ Now, myapp.com **points to your EC2 server**.

CHAPTER 5: SECURING THE APPLICATION WITH HTTPS (SSL CERTIFICATE)

5.1 Installing Nginx for Reverse Proxy & SSL

📌 **Install Nginx on EC2:**

sudo apt install nginx -y

sudo systemctl start nginx

sudo systemctl enable nginx

✓ Nginx acts as a **reverse proxy** to forward traffic.

📌 **Update Nginx Configuration (/etc/nginx/sites-available/default)**

```
server {
    listen 80;
    server_name yourdomain.com;

    location / {
        proxy_pass http://localhost:3000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

✓ **Forwards requests to Node.js running on port 3000**.

5.2 Installing a Free SSL Certificate (Let's Encrypt)

📌 **Install Certbot & Obtain SSL Certificate:**

sudo apt install certbot python3-certbot-nginx -y

sudo certbot --nginx -d yourdomain.com -d www.yourdomain.com

✓ **Automatically configures HTTPS** for your domain.

📌 **Renew SSL Automatically:**

sudo certbot renew --dry-run

✓ Ensures SSL **auto-renews before expiration**.

Case Study: How Airbnb Uses AWS EC2 & S3 for Hosting

Challenges Faced by Airbnb

✓ Handling **millions of user requests daily**.
✓ Storing and serving **high-resolution images** efficiently.

Solutions Implemented

✓ Used **EC2 to host scalable backend servers**.
✓ Stored **user images & assets in S3 for fast delivery**.
✓ Implemented **CloudFront CDN** to speed up global access.

🔹 **Key Takeaways from Airbnb's AWS Hosting Strategy:**

✓ **EC2 ensures scalable, high-performance applications.**
✓ **S3 provides cost-effective static file storage.**
✓ **SSL (HTTPS) secures user data & improves SEO rankings.**

📝 Exercise

✅ Deploy a **React frontend** on AWS S3.
✅ Set up an **EC2 instance** and deploy a Node.js backend.
✅ Configure **Nginx as a reverse proxy** to serve your API.
✅ Secure your domain with **Let's Encrypt SSL**.

Conclusion

✓ EC2 is ideal for hosting dynamic applications and APIs.
✓ S3 is best for serving static files & websites efficiently.
✓ Custom domains & SSL (HTTPS) improve security and trust.
✓ AWS offers scalable, reliable hosting for full-stack applications.

DEPLOYING FULL STACK APPS WITH FIREBASE

CHAPTER 1: INTRODUCTION TO FIREBASE DEPLOYMENT

1.1 What is Firebase?

Firebase is a **backend-as-a-service (BaaS)** by Google that provides **hosting, authentication, database management, and cloud functions** for full-stack applications.

- **Why Use Firebase for Deployment?**

✓ Provides **free hosting** for web applications.
✓ Supports **continuous deployment with Firebase CLI**.
✓ Offers **backend services like Firestore, Authentication, and Cloud Functions**.
✓ Enables **real-time database syncing and fast global content delivery**.

- **Firebase Services for Full Stack Apps:**

Feature	Description
Hosting	Deploy frontend (React, Vue, Angular) globally
Firestore	NoSQL real-time database for backend storage
Authentication	User login with Google, Email, GitHub, etc.
Cloud Functions	Serverless backend for executing logic

CHAPTER 2: SETTING UP FIREBASE IN A FULL-STACK PROJECT

2.1 Installing Firebase CLI

📌 Step 1: Install Firebase CLI globally

npm install -g firebase-tools

✓ Allows using Firebase commands from the terminal.

📌 Step 2: Login to Firebase

firebase login

✓ Opens a browser window to authenticate your Google account.

2.2 Creating a Firebase Project

1. Go to Firebase Console.
2. Click **"Create a Project"** → Enter project details.
3. Enable **Firestore, Hosting, and Authentication** if needed.

📌 Step 3: Initialize Firebase in the Project

firebase init

✓ Select **Hosting** for frontend deployment.
✓ Choose **Firestore** if using a database.
✓ Select **Functions** for backend logic.

CHAPTER 3: DEPLOYING A REACT FRONTEND TO FIREBASE

3.1 Building the React App

📌 Inside the React project directory, run:

npm run build

✓ Generates a **production-ready version** inside the build/ folder.

3.2 Configuring Firebase Hosting

📌 **Run Firebase initialization and select Hosting:**

firebase init hosting

✓ Set the **public directory** to build.
✓ Choose **"Configure as a single-page app (SPA)"** → Yes.

3.3 Deploying the Frontend to Firebase

📌 **Run Firebase Deploy Command:**

firebase deploy

✓ Firebase generates a **live hosting URL** for your app.

🔹 **Example Output:**

✓ Deploy complete!

Hosting URL: https://your-project-name.web.app

✓ Your **React frontend is now live** on Firebase Hosting!

CHAPTER 4: DEPLOYING A NODE.JS BACKEND WITH FIREBASE FUNCTIONS

4.1 Setting Up Cloud Functions

📌 **Navigate to the backend folder and initialize Firebase Functions:**

firebase init functions

✓ Select **Node.js** as the backend runtime.

✓ Choose **"ES Modules" or CommonJS** when prompted.

4.2 Writing a Cloud Function for an API Endpoint

📌 **Modify functions/index.js to create an Express API:**

```
const functions = require("firebase-functions");

const express = require("express");


const app = express();


app.get("/api/message", (req, res) => {

  res.json({ message: "Hello from Firebase Functions!" });

});


exports.api = functions.https.onRequest(app);
```

✓ Defines an **Express.js API route (/api/message)** inside Firebase Functions.

4.3 Deploying Cloud Functions

📌 **Run the deployment command:**

```
firebase deploy --only functions
```

✓ Firebase generates a **function URL** to call the API.

🔹 **Example Output:**

✓ Deploy complete!

Function URL: https://us-central1-your-project-name.cloudfunctions.net/api

✓ Your **backend API is now live** on Firebase Functions!

CHAPTER 5: CONNECTING REACT FRONTEND TO FIREBASE API

5.1 Fetching Data from Firebase Cloud Functions

📌 **Modify src/App.js in React to Fetch API Data:**

```
import { useState, useEffect } from "react";

function App() {
  const [message, setMessage] = useState("");

  useEffect(() => {
    fetch("https://us-central1-your-project-name.cloudfunctions.net/api/message")
      .then(response => response.json())
      .then(data => setMessage(data.message));
  }, []);

  return <h1>{message}</h1>;
}
```

export default App;

✓ **Retrieves backend data dynamically** into the React app.

CHAPTER 6: IMPLEMENTING FIREBASE AUTHENTICATION

6.1 Enabling Authentication in Firebase Console

1. Go to **Firebase Console → Authentication**.
2. Click **"Sign-in Method"** → Enable **Google, Email/Password, GitHub**, etc.

6.2 Integrating Firebase Authentication in React

📌 **Install Firebase SDK in React:**

npm install firebase

📌 **Set Up Firebase Config (firebaseConfig.js)**

```
import { initializeApp } from "firebase/app";

import { getAuth, GoogleAuthProvider, signInWithPopup } from "firebase/auth";


const firebaseConfig = {

  apiKey: "YOUR_FIREBASE_API_KEY",

  authDomain: "your-project.firebaseapp.com",

  projectId: "your-project-id",

};
```

```
const app = initializeApp(firebaseConfig);
const auth = getAuth(app);
const provider = new GoogleAuthProvider();

export { auth, provider, signInWithPopup };
```

✓ Initializes Firebase Authentication in React.

6.3 Creating a Google Login Button

📌 Modify src/App.js to Add Google Login

```
import { auth, provider, signInWithPopup } from "./firebaseConfig";

function App() {
  const login = async () => {
    const result = await signInWithPopup(auth, provider);
    console.log("User:", result.user);
  };

  return <button onClick={login}>Sign in with Google</button>;
}

export default App;
```

✓ Clicking the button **logs in users via Google**.

Case Study: How Netflix Uses Firebase for Deployment

Challenges Faced by Netflix

✓ Handling **millions of global users** accessing content.
✓ Ensuring **fast and secure authentication**.
✓ Deploying **frontend and backend efficiently**.

Solutions Implemented

✓ Used **Firebase Hosting** for fast content delivery.
✓ Integrated **Firebase Authentication for secure login**.
✓ Used **Firestore for scalable user data storage**.

- **Key Takeaways from Netflix's Deployment Strategy:**

✓ **Firebase simplifies deployment and scaling**.
✓ **Cloud Functions allow serverless backend execution**.
✓ **Real-time databases enable instant content updates**.

📝 Exercise

✅ Deploy a **React app with Firebase Hosting**.
✅ Create a **backend API using Firebase Cloud Functions**.
✅ Implement **Google Authentication in your React app**.
✅ Fetch API data from **Firebase Functions to React frontend**.

Conclusion

✓ **Firebase Hosting provides fast, secure frontend deployment.**
✓ **Cloud Functions allow building serverless Node.js backends.**
✓ **Authentication enables secure user login across devices.**

✓ **Connecting React to Firebase simplifies full-stack development.**

ASSIGNMENT:

DEPLOY A REACT-NODE.JS PROJECT ON NETLIFY & HEROKU

ASSIGNMENT SOLUTION: DEPLOY A REACT-NODE.JS PROJECT ON NETLIFY & HEROKU

Step 1: Setting Up the React Frontend for Deployment

1.1 Preparing React App for Deployment

📌 **Step 1: Navigate to React Project Directory**

cd my-react-app

📌 **Step 2: Build the React App**

npm run build

✓ Generates an optimized **build/ folder** for deployment.

1.2 Deploying React App to Netlify

📌 **Method 1: Deploy Using Netlify Web Interface**

1. **Login to Netlify** at Netlify.
2. Click **New Site** → **Import from GitHub**.
3. Select **React GitHub Repository**.
4. **Set Build Commands**:
5. npm run build
6. Set **Publish Directory** to:
7. build
8. Click **Deploy Site** → Netlify **assigns a live URL** (e.g., myapp.netlify.app).

📌 Method 2: Deploy Using Netlify CLI

npm install -g netlify-cli

netlify login

netlify init

netlify deploy --prod --dir=build

✓ **Deploys the React app directly from the terminal**.

Step 2: Setting Up Node.js Backend for Deployment

2.1 Preparing Node.js App for Deployment

📌 Step 1: Navigate to Backend Project

cd my-node-app

📌 Step 2: Install Required Packages

npm install express cors dotenv mongoose

✓ Installs **Express.js, CORS, dotenv, and MongoDB (if applicable)**.

📌 Step 3: Ensure server.js Has a Dynamic Port

```
const express = require("express");

const app = express();

const cors = require("cors");


app.use(cors());

app.use(express.json());
```

```
app.get("/", (req, res) => {
  res.send("Node.js backend is running!");
});

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

✓ process.env.PORT || 5000 ensures **Heroku assigns a port dynamically**.

📌 Step 4: Add Start Script in package.json

```
"scripts": {
  "start": "node server.js"
}
```

✓ Ensures **Heroku knows how to start the backend server**.

2.2 Deploying Node.js App to Heroku

📌 Step 1: Install Heroku CLI

```
npm install -g heroku
```

📌 Step 2: Login to Heroku

```
heroku login
```

📌 Step 3: Initialize Git Repository

```
git init
git add .
```

git commit -m "Deploy Node.js backend"

📌 Step 4: Create a New Heroku App

heroku create my-node-backend

📌 Step 5: Deploy Backend to Heroku

git push heroku main

✓ Heroku **assigns a live API URL** (e.g., https://my-node-backend.herokuapp.com).

Step 3: Connecting Frontend and Backend

3.1 Update React App to Use Live Backend API

📌 Modify src/config.js in React Project

```
const API_URL = "https://my-node-backend.herokuapp.com";

export default API_URL;
```

📌 Modify API Calls in React Components

```
fetch(`${API_URL}/api/data`)
  .then(response => response.json())
  .then(data => console.log(data));
```

✓ **Ensures frontend communicates with backend properly**.

3.2 Handling CORS Issues

📌 Modify server.js to Enable CORS

```
const cors = require("cors");
```

```
app.use(cors({ origin: "https://myapp.netlify.app" }));
```

✓ **Allows frontend requests from Netlify to Heroku backend**.

Case Study: How Netflix Deploys Full-Stack Apps on the Cloud

Challenges Faced by Netflix

✓ Handling **millions of requests per second**.
✓ Ensuring **fast content delivery globally**.
✓ Scaling **backend API requests efficiently**.

Solutions Implemented

✓ **Frontend hosted on AWS S3 (like Netlify)** for **fast content delivery**.
✓ **Backend API hosted on AWS EC2 (like Heroku)** for **scalability**.
✓ **Database hosted separately** (PostgreSQL, DynamoDB) to **handle user data**.

- **Key Takeaways from Netflix's Deployment Strategy:**

✓ **Cloud hosting ensures high availability**.
✓ **Optimized API calls reduce latency**.
✓ **CI/CD automates deployment and updates**.

📝 **Exercise**

✅ Deploy a **React app on Netlify** using GitHub.
✅ Deploy a **Node.js backend on Heroku** and connect to React.
✅ Configure **CORS to allow API requests from frontend to backend**.
✅ Use **Postman to test API routes hosted on Heroku**.

Conclusion

✓ **Netlify is ideal for frontend deployment** with GitHub integration.
✓ **Heroku is excellent for hosting Node.js backend APIs**.
✓ **CORS must be handled properly for frontend-backend communication**.
✓ **Deploying full-stack apps ensures real-time interaction between frontend and backend**.