# 1. Clone project (once)

git clone https://github.com/username/repo.git

cd repo

# 2. Make changes in code

# 3. Add and commit

git add .

git commit -m "My changes"

# 4. Pull latest to avoid conflict

git pull origin main

# 5. Push to remote

git push origin main

# 1. Clone your Spring Boot repo

git clone https://github.com/username/springboot-jenkins.git

cd springboot-jenkins

# 2. Make code changes

# (Edit Java files, HTML, etc.)

# 3. Stage and commit changes

git add .

git commit -m "Added new feature"

# 4. Pull latest to avoid conflicts

git pull origin main

# 5. Push changes to GitHub

git push origin main

# 6. Jenkins auto-builds or manually triggered

# - Builds with Maven

# - Deploys JAR using java -jar OR copies to server

# 1. Clone project

git clone https://github.com/username/springboot-jenkins-docker.git

cd springboot-jenkins-docker

# 2. Edit code

# 3. Stage and commit

git add .

git commit -m "Updated feature and Dockerfile"

# 4. Pull latest

git pull origin main

# 5. Push to GitHub

git push origin main

# 6. Jenkins runs:

# - mvn clean package

# - docker build ...

# - docker run ...

**Step-by-Step Explanation in Interview**

✅ **1. Clone and Code**

"First, I clone the Spring Boot project from GitHub and make the necessary code changes in my local development environment."

✅ **2. Commit and Push**

"Once tested locally, I commit the changes and push them back to the main branch, making sure to pull the latest changes before pushing to avoid conflicts."

✅ **3. Jenkins Build Trigger**

"Our Jenkins is integrated with GitHub, so pushing code automatically triggers a Jenkins job."

✅ **4. Maven Build**

"Jenkins runs a Maven build to generate the JAR file of the Spring Boot application."

✅ **5. Docker Build and Run**

"Then Jenkins uses a Dockerfile to build a Docker image from the JAR, tags it, and runs it as a container. We also push the image to Docker Hub for versioning or cloud deployment."

✅ **6. Deployment**

"Finally, Jenkins deploys the Docker container either locally or to a remote server depending on the environment (dev, QA, prod)."

1. Clone the project

git clone https://github.com/username/springboot-jenkins-docker.git

cd springboot-jenkins-docker

1. Make code changes

(Edit Java, HTML, properties, etc. using your IDE)

1. Stage and commit your changes

git add .

git commit -m "Implemented new feature and updated Dockerfile"

1. Pull latest changes from main

git pull origin main

1. Push changes to GitHub

git push origin main

1. Jenkins job is triggered

Inside Jenkins, the following commands are run:

1. Build the JAR

mvn clean package -DskipTests

1. Build Docker image

docker build -t springboot-docker-app .

1. Push image to Docker Hub (optional)

docker tag springboot-docker-app yourdockerhub/springboot-docker-app

docker login

docker push yourdockerhub/springboot-docker-app

1. Run Docker container

docker stop springboot-app || true

docker rm springboot-app || true

docker run -d -p 8080:8080 --name springboot-app springboot-docker-app

"In my project, I worked on a Restaurant Management System where customers can register, log in, and view the menu with prices and special offers. They can place orders for dine-in, takeaway, or delivery. Once the order is placed, it goes to the kitchen through a display system for preparation. The system generates bills with taxes and offers, and customers can pay online using cards, UPI, or wallets. For delivery orders, it assigns delivery staff and allows real-time tracking. The admin panel has role-based access for staff like managers, waiters, and kitchen workers, and it also provides reports for sales and staff performance."

"I worked on a Restaurant Management System where users can sign up, log in, and explore the food menu with prices and offers. They can place orders for eating at the restaurant, takeaway, or home delivery. Once an order is placed, it is shown directly to the kitchen team for preparation. The system automatically creates the bill, adds taxes or discounts, and supports online payments through cards, UPI, or wallets. For delivery, it assigns drivers and allows customers to track their order in real-time. The admin side includes different access levels for staff like managers, waiters, and kitchen staff, and it also gives reports on sales and performance."

"In this project, I developed a Restaurant Management System where customers can create an account, log in, and check the menu with item details and offers. They can place orders for dining in, takeaway, or delivery. Once the order is placed, it is sent to the kitchen so the staff can start preparing it. The system creates the final bill, applies any discounts or taxes, and supports secure online payments. For delivery, it assigns available delivery staff and allows the customer to track the order live. The admin section manages staff access based on roles and provides useful reports like daily sales and staff performance."

"Every day, I usually start by checking emails and any updates from the team, especially from the previous day or night if there were issues or merge requests. Then we have a stand-up meeting where we discuss what we did yesterday, what we plan for today, and any blockers.

After that, I focus on coding tasks — mostly working on new features, bug fixes, or enhancements using Java, Spring Boot, and REST APIs. I write unit tests to ensure code quality and use tools like Postman to test the APIs. I also review other team members' code through pull requests and push my changes to Git after testing.

I frequently check logs, debug issues if something fails, and work with the QA team for testing and deployment. Towards the end of the day, I update the Jira or task board and prepare for the next day's priorities."

"Each day starts with checking emails and team messages to see if there are any urgent issues or tasks. Then we have a stand-up meeting where I share what I worked on yesterday, what I plan to do today, and if I’m facing any problems.

After that, I start my development tasks. I mostly work on building or updating REST APIs using Java and Spring Boot. Since our project uses microservices, I focus on one service at a time, making sure it connects properly with others. I also write unit tests and do proper exception handling.

Once the code is ready, I test it using tools like Postman or Swagger, and then push it to Git. I also review other developers' code through pull requests. We follow CI/CD, so I monitor build status and fix any issues if the pipeline fails.

During the day, I may also check logs using tools like Kibana or Splunk and help debug production or QA issues. At the end of the day, I update Jira with my progress and prepare for the next task."

"நான் தினமும் காலை முதலில் மெயில்களும், டீம் மெசேஜ்களும் பார்ப்பேன் – ஏதாவது முக்கியமான அப்டேட்ஸ் இருக்கா என்று. அதுக்குப்பிறகு, எங்கள் டெய்லி ஸ்டாண்ட்-அப் மீட்டிங்குல், நேற்று என்ன வேலை பண்ணேனும், இன்றைக்கு என்ன பண்ணப்போறேனும், ஏதாவது பிரச்சனை இருந்தா அதையும் ஷேர் பண்ணுவேன்.

அதுக்கப்புறம், Spring Boot & Javaயைப் பயன்படுத்தி REST API-களை டெவலப்பிங் பண்ணுவேன். எங்கள் ஆப்ளிகேஷன் மைக்ரோசர்வீசஸ் அடிப்படையில இருக்குது, எனவே ஒவ்வொரு சர்வீசும் சரியான முறையில் இணைக்கப்பட்டிருக்கும் என்பதை கவனிப்பேன். யூனிட் டெஸ்ட் எழுதுவேன், எரர் ஹேண்ட்லிங் பண்ணுவேன்.

பிறகு, Postman அல்லது Swagger மாதிரி tools மூலம் API-களை டெஸ்ட் பண்ணுவேன். Git-க்கு commit & push பண்ணுவேன். டீம் மெம்பர்கள் code review கொடுத்தா அதையும் பார்த்து approve பண்ணுவேன்.

CI/CD பைப்லைன் பாத்து, build வெற்றிகரமா நடந்திருக்கா, ஏதாவது எரர் இருந்தா அதையும் சரி பண்ணுவேன். லாக் பாக்குவதற்காக Kibana, Splunk மாதிரி tools உபயோகிப்பேன். தினமுடிவில், என் Jira task update பண்ணுவேன்."

"In my daily work, I use Jira to manage and track all my tasks. Each task or user story is created as a ticket in Jira with details like the description, priority, and status. During the daily stand-up, I update the status of my tasks — for example, from 'To Do' to 'In Progress' or 'Done' based on my progress.

Jira helps me plan my work better during sprints. I check the assigned tasks, work on them one by one, and add comments or attach code links if needed. We also use it to track bugs, assign them to the right developer, and follow up until they're resolved.

Overall, Jira helps our team stay organized and gives a clear view of what each member is working on and how the sprint is progressing."

"I use Git daily for version control. Whenever I start a task, I create a new branch from the main branch and do my development there. After completing the changes, I use git add and git commit to save the changes locally, with proper commit messages.

Before pushing the code, I always run git pull to make sure my branch is up to date with the latest changes and to avoid conflicts. Then I push the code using git push and create a pull request (PR) for code review. I also review my teammates' PRs, give comments, and approve them if everything looks good.

If any conflict occurs, I handle it by resolving the differences manually and committing again. Git helps our team collaborate smoothly, keep track of changes, and maintain a clean code history."

"In our project, we use Continuous Integration (CI) to automatically build and test the code whenever someone pushes changes to the Git repository. As soon as I push my code or create a pull request, the CI pipeline gets triggered — usually through tools like Jenkins, GitHub Actions, or GitLab CI.

The pipeline runs automated steps like compiling the code, running unit tests, checking code quality using tools like SonarQube, and verifying if the build is successful. If there are any issues or test failures, we get notified immediately so we can fix them before merging.

This CI process ensures that only clean, tested code goes into the main branch and reduces the chance of bugs in production. It also saves time and makes the development process more reliable and consistent."

"In my project, we use both Continuous Integration (CI) and Continuous Deployment (CD) to automate our development and release process.

With CI, whenever we push code to Git, the pipeline automatically builds the project, runs tests, and checks code quality using tools like SonarQube. Once the build is successful, the Continuous Deployment (CD) part takes over.

CD handles the automatic deployment of the application to different environments — like development, QA, or even production. We use tools like Jenkins, GitHub Actions, or GitLab CI/CD for this. After the code is tested and approved, it gets deployed without any manual steps.

This process helps us release features faster, with fewer errors, and ensures that our code is always in a deployable state."

"I have been working in Agile methodology, where the project is divided into small, manageable parts called sprints, usually 2 weeks long. Every day, we have a daily stand-up meeting where each team member shares what they did yesterday, what they plan to do today, and if they have any blockers.

At the beginning of each sprint, we have a sprint planning meeting where tasks are assigned based on priority. We use tools like Jira to manage these tasks. During the sprint, we focus on delivering working features with proper testing and integration.

At the end of the sprint, we do a sprint review to demo the completed work and get feedback. We also conduct a sprint retrospective to discuss what went well, what can be improved, and how to make the next sprint better.

Agile helps us adapt quickly to changes, deliver features faster, and maintain better communication within the team."

"In Agile, a sprint is a short, fixed period — usually 1 to 2 weeks — where the team works on a set of tasks or user stories. At the beginning of the sprint, we have a sprint planning meeting where we select high-priority tasks from the product backlog and commit to completing them during the sprint.

Throughout the sprint, we have daily stand-up meetings to track progress and discuss any blockers. During the sprint, we focus on completing the selected tasks with development, testing, and integration.

At the end of the sprint, we conduct a sprint review to demo the work done to the Product Owner or client. After that, we do a sprint retrospective to discuss what went well, what didn’t, and how we can improve in the next sprint.

Sprints help us deliver working features regularly and respond quickly to changes in requirements."

"In our project, we follow the Agile model and work in 2-week sprints. At the start of each sprint, we have a **sprint planning meeting** where the Product Owner selects user stories from the backlog based on priority. For example, in one sprint, I worked on creating a new REST API for customer registration in our restaurant management system.

We manage all tasks in **Jira**, where each story is broken into sub-tasks like development, unit testing, and code review. During the sprint, we have daily stand-up meetings to share progress and blockers. I update the Jira board by moving tasks from 'To Do' → 'In Progress' → 'Done'.

At the end of the sprint, we do a **sprint review**, where I demo the completed API with Postman to the team and Product Owner. Then we have a **sprint retrospective** to discuss what went well, any issues we faced, and how to improve in the next sprint.

This sprint cycle helps us deliver features on time, maintain quality, and quickly adapt to changes."