Basic Deployment Pipeline

The deployment pipeline starts when developers commit code to version control. The pipeline is triggered by continuous integration (CI), where the code is compiled, unit tests are run, code analysis is done, and installers are created. If tests pass, the code is turned into binaries and stored in an artifact repository like Nexus.

Next, the pipeline prepares for automated acceptance testing, which can be run in parallel to speed up feedback. Once this testing is successful, the code is automatically deployed to environments like user acceptance testing (UAT), capacity testing, and production.

The operations team manages automated deployment scripts. Testers can see the status of release candidates and deploy builds with the press of a button. The overall goal is to provide quick feedback and track the code's progress through each pipeline stage

Deployment pipeline practices..

- **Build the binary only once**: To avoid inconsistencies, the code is compiled into binaries (e.g., .jar, .NET assemblies) only once during the first stage. These binaries are stored and reused across all stages of testing and deployment, ensuring there are no unintentional differences between environments. Verifying the integrity of the binaries ensures the same artifact is deployed everywhere.

- **Deploy the same way to every environment**: Deployments should follow the same process regardless of the environment (development, testing, production). Using configuration files or external services for environment-specific settings allows the same deployment scripts to work everywhere, minimizing errors that arise from differences in the deployment process.

- **Smoke test the deployments**: After deploying, a smoke test checks that the application can start and run its essential functions. It quickly verifies that the application and critical services like databases or external APIs are functioning properly, giving the team confidence before proceeding with further testing.

- **Deploy into a copy of production**: Testing should be done in environments that closely resemble production to ensure that issues are caught before the real deployment. This includes having similar infrastructure, OS configurations, application stacks, and valid data. Any significant differences between the test environment and production can lead to undetected bugs.

- **Instant propagation of changes through the pipeline**: In a continuous integration setup, when a new change is committed, it triggers the pipeline stages (build, unit tests, etc.) automatically. The process ensures that each code change moves quickly through testing and deployment, providing fast feedback. If any issue arises, it can be detected early in the pipeline, reducing the chance of deploying broken code.

- **Stop the line if any part fails**: If there’s a failure in any stage of the pipeline (build, tests, or deployment), the entire team stops working on new tasks and focuses on fixing the issue. This ensures that problems are addressed immediately, preventing further issues from building up and maintaining the quality of the deployment pipeline.