CI/CD Deployment Pipeline – Part 1

This lesson provides insight into the deployment pipeline.

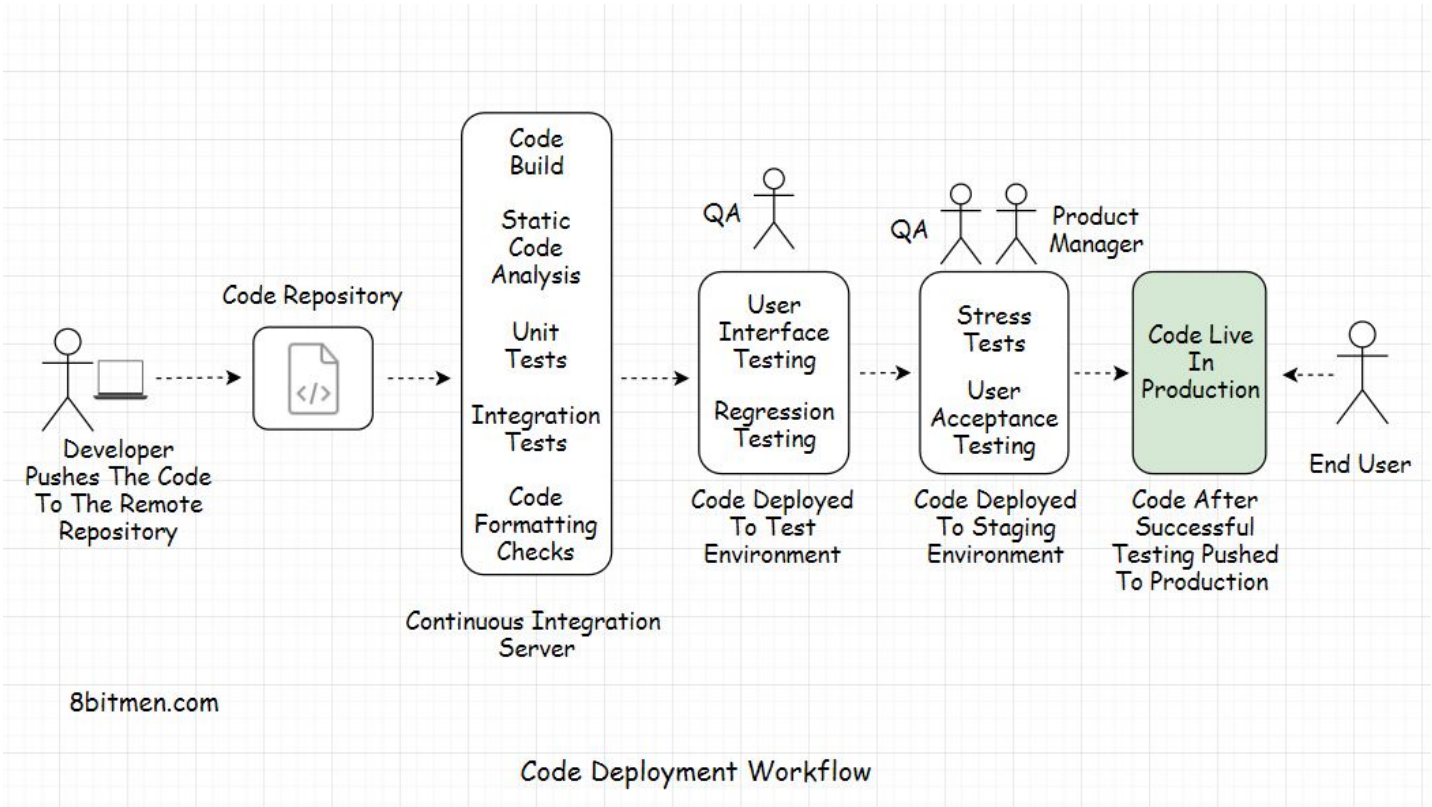
We'll cover the following

- What is a deployment pipeline?
- Deployment workflow

What is a deployment pipeline?

The deployment pipeline is an implementation of the *continuous delivery* and the *continuous deployment* approach. It's a step-by-step specification on how the code should flow from the point it is pushed from the developer's laptop to the remote repository to the point that the code commit is finally deployed to production.

We refer to this structured process of code deployment as the *deployment pipeline* because it enables the code to flow from one starting point to an endpoint. This is the production, going through a series of stages in between, just like the water flows through a pipeline in a well-directed fashion. The in-between stages are the code builds, unit tests, integration tests, code quality checks, and so on.



Businesses can structure their pipeline as per their requirements. There is no standard rule on how a pipeline should be built. The whole point of creating a pipeline is to deploy the code to production as efficiently as possible with minimal human intervention and time investment.

Now, let's go through the deployment workflow to understand the intermediary stages in the deployment pipeline.

Deployment workflow

Here are the general stages that the code goes through in a deployment pipeline:

1. After writing the code, running the build, and testing it on the local machine, the developer pushes the code to the remote repository such as *GitHub*.

2. Once the code is committed to the remote repository, an automated build process is triggered powered by tools like *Jenkins*. *Jenkins* is an open-source automation tool used for *continuous integration*. You will see more on this later. The build process runs several checks on the code in series, such as static code analysis, code quality checks, compliance with the coding standards and formatting set by the organization, and so on.

3. Units tests, integration tests, and other tests also get triggered in the same build to test the code.

4. Once the build is complete, the new patch or the feature developed by the developer may be manually checked by the designated tester on the team. This helps in testing corner cases and regression testing.

5. Once the testing is done, the code is deployed to multiple environments such as *staging* and any other environments set by the engineering team.

6. _ User Acceptance Testing (UAT)_ is generally performed by the product manager in these environments to ensure everything looks good and works as expected. *Load* and *stress tests* are also run in a pre-production environment, which is a replica of the production environment.

7. Once everything looks okay, the code is pushed to production.

These are the common stages you'll find in a deployment pipeline. You can always

These are the common stages you'll find in a deployment pipeline. You can always tweak these based on your requirements.

During the deployment process, if anything fails at any stage, the entire deployment process is killed following a *fail-fast* approach. The system sends notifications to the concerned members of the team for the issue to be fixed immediately.

Developers are advised to only push the code to the remote repo after they perform thorough testing on their local machine. This is done to avoid any build breaks in the pipeline due to the broken code, which is often frowned upon.

If the build fails, fixing the build becomes the main priority of the developer. Also, the build fail email is triggered to everyone on the team, including upper management of the project. As a developer, you definitely do not want that :)

When the deployment pipeline is built from the ground up, it is first set to an environment that is a replica of the production. Once the pipeline is optimized and bug-free, it is directed to the production environment.

The deployment pipeline is commonly known as the *CI/CD pipeline*. *CI* stands for *continuous integration*, and CD stands for *continuous delivery and deployment*.

In the next lesson, let's understand what *continuous integration* means.