CI/CD Deployment Pipeline – Part 2

This lesson continues the discussion about the deployment pipeline.

We'll cover the following

What is continuous integration?

Continuous integration is a software development practice where all the developers in the team push their code to the remote repository as frequently as possible, as opposed to holding the code that they work on, on their local machine, for a longer period of time.

How does pushing the code so often help?

There are several upsides to the continuous integration approach:

1. In case the developer's machine fails, all the code they have worked on is not lost since it's frequently pushed to the remote repository.

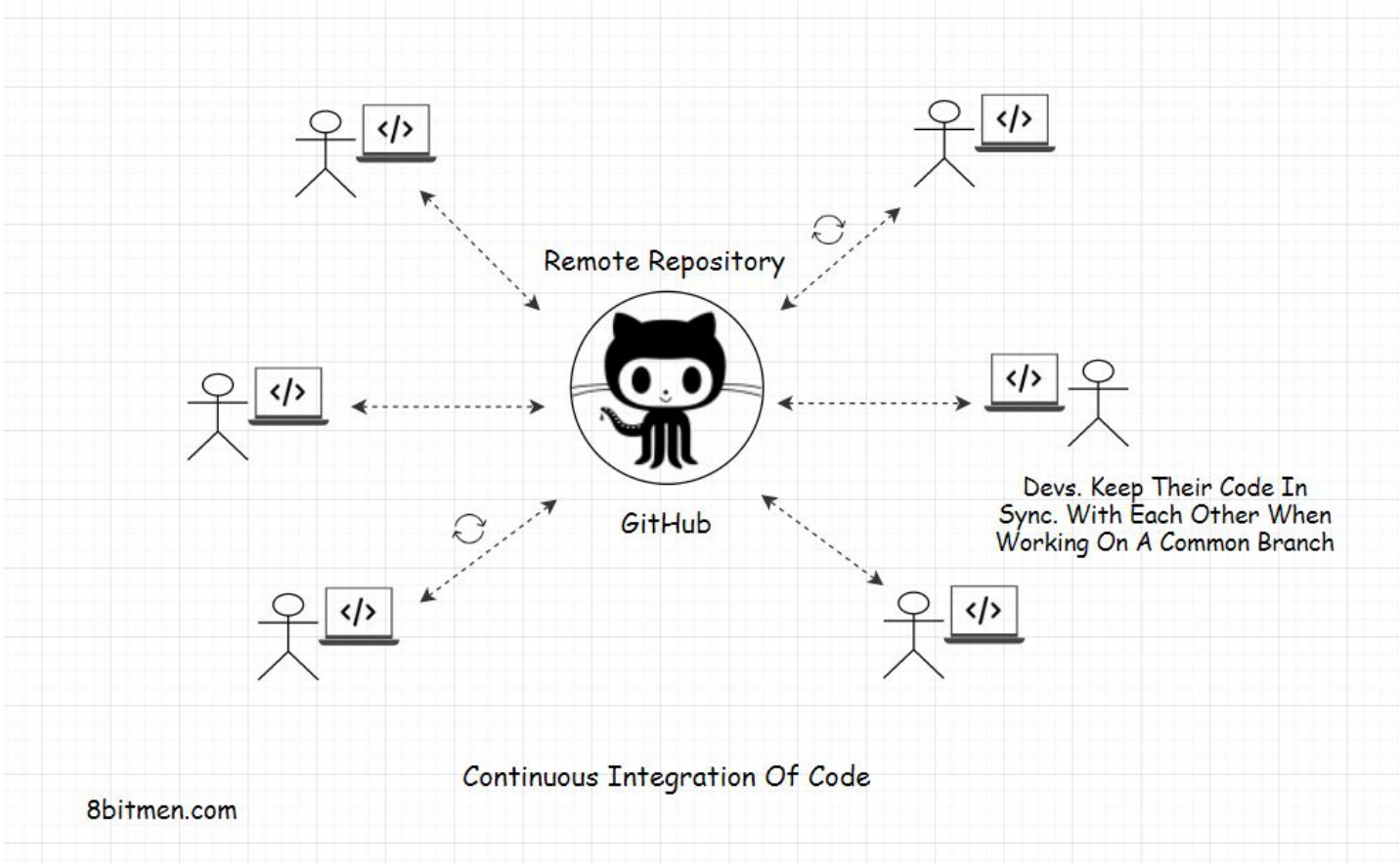
2. Every code push triggers the build and tests; this continually ensures the code quality and its correctness, as opposed to testing a major change at the end. Smaller changes are easy to merge with the *mainline*. Frequently checking in code helps the developer fix the issues early, with ease, and with less debugging.

3. Team members can see your code often and provide early feedback.

4. The biggest upside of continuous integration is avoiding code integration issues. Imagine a scenario, where a new feature is being developed by the team. For the new feature development, a new branch is created from the master and everyone pulls the code from the branch in their local machine to start working on it.

In this scenario, if everyone frequently pushes their code changes to the same branch, it's easy to resolve code conflicts on an ongoing basis when integrating the code. Imagine the situation, when at the end of the sprint, everyone finally pushes their code to the same branch. It would be a merge hell.

n developers in the team would have made *m* changes in the same class, even the same variables. It would be insanely difficult to understand each other's code and eventually reach a consensus. Therefore, it's always a good idea to continually integrate your code to save time and keep everyone on the same page



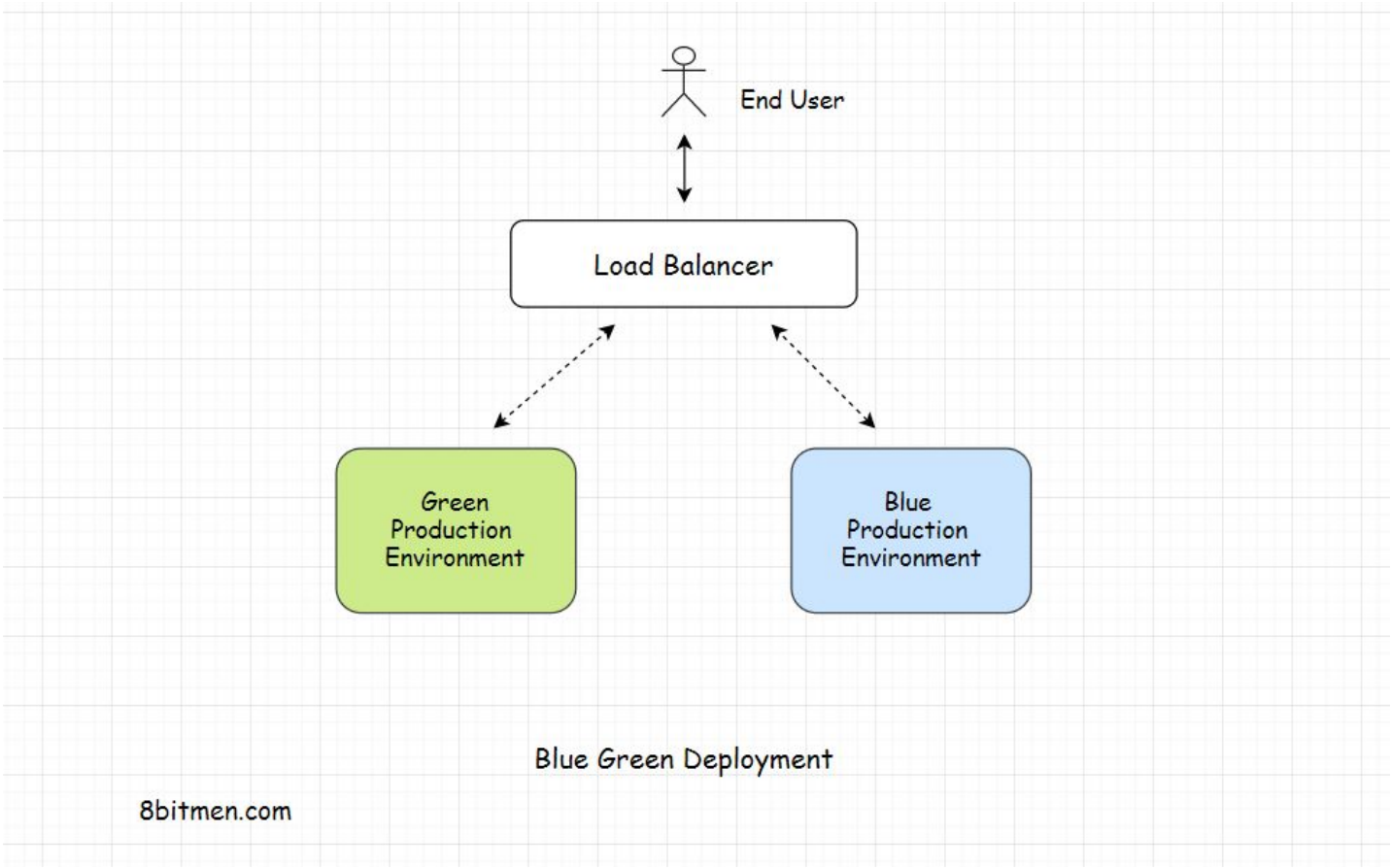
Blue-green deployments

Returning to the deployment pipeline, when the code is deployed to production, engineering teams often create a replica of the production environment. This gives the team two mirror production environments; one is called *blue* and the other *green*. This is done to avoid any sort of service downtime during code deployment.

So, how does this work?

In the blue-green deployment approach, only one production environment is *live*, or handling the traffic on the app, at any point in time. When the code is deployed, it is deployed on the environment that is not live so that the live production environment remains unaffected from the new deployment.

Once everything is tested in the environment in which the code is deployed and the team is satisfied, the traffic is switched to the non-live environment, making it live. The same process is repeated during the next deployment.



> ***A good read***: Micro Deploy, an in-house deployment system that builds, upgrades, and rolls back services at Uber

Well, this concludes our *CI/CD deployment pipeline* lesson. In the next lesson, let's have a look at *continuous monitoring*.