# ABOUT CURRENT CONTINUOUS INTEGRATION (CI) AND CONTINUOUS DELIVERY (CD) / CONTINUOUS DEPLOYMENT FLOW.

When I started to working at XXXXX company (we are doing some pretty cool stuff there btw, there was a lot of things to put in place to setup our back-end / front-end layers. The cool thing was the possibility to select the technologies we will use… Let's go for Jenkins, maven/make, JFrog, SonarCube….. other cool stuff then (use tools in which you are convenient)

## CI / CD flow architecture

We have several services (do not know if I should call them microservices…) and I'd like them to follow as the development flow. The idea is to setup a Continuous Integration / Continuous Delivery, Deployment pipeline. This architecture May not look like complete one, but this is something that can have code pushed, tested and deployed automatically several times a day and this setup is improved based on the future requirements.

**GIT PART / CI PART**

1. For the code versioning part, we are using git / GitHub on a daily basis, so I good at configuring and maintaining git tools like GitHub / Git Labs / Bitbucket. Once developer completes his coding and when pushes latest code to dev / feature branch, he will raise a pull request / merge request (PR) from feature branch to respective month's release branch.
2. The above raised PR will code reviewed by architect / Lead / responsible person where at least one of the members must approve a pull request to get it merged to the release branch. (as the number of checks grew, developers have to wait longer and longer for their CI results or code to get merged.)
3. Once the PR is approved by reviewers automatically pipeline job will be triggered in jenkins.
   **Note:** PR review setup is done by gerrit or we have default settings in github / bitbucket to set the default reviewers for PR.

**VERSION CONTROL PHASE: Jenkins Pipeline To check code**

1. In our project the pipeline is always triggered for new PR with open state for code sanity test.
2. Using Blue Ocean plugin, we integrated CI pipeline into our GitHub workflow.
   Learn more about installing Blue Ocean.
3. After a Pipeline is executed, Jenkins will automatically return the build status to GitHub. we can view this status right from a pull request and require it to pass before merging the pull request. This is for status check is a special branch protection checks to ensure that only successfully built and tested code can be merged (sanity test from config management side).
4. If there is a problem or error in code we need to inform the respective developer or in some of our pipeline job automatically a mail will be dropped to respective developer by getting the details from git commit, here Git Changelog Plugin is used to get the log details.
5. If the above pipeline is successful without any errors the code can be merged to respective release branch, then we got main pipeline job which will be triggered for further Deployments to environments such as TEST (T), QA, PRE-PROD(UAT), PROD.
   **Note:** Here developer need to push the code with a proper version tag as our pipeline job build with this tag.

**CONTINUOUS DELIVERY/DEPLOYMENT PHASE: CD with Jenkins Pipeline**

In our project the main pipeline is always triggered after merging PR. The main pipeline job consists of below pipeline stages:

**STAGE 1: Build Stage:**

we have code files/objects from all the features/changes from various branches of the repository from various developers which we merge and needs to be compiled before execution and we build binary from compiled bytecode using build tool, for Java we are using **Maven** & for c **make** as build tool. Here developer changes will go through the version control phase to the build phase, where build is generated by its compiled code. Finally, we create a binary package called Test build, this whole process is called as **build phase.**

**STAGE 2: Testing Stage:**

In this phase, above build is deployed to test environment to run various kind of test cases like below:

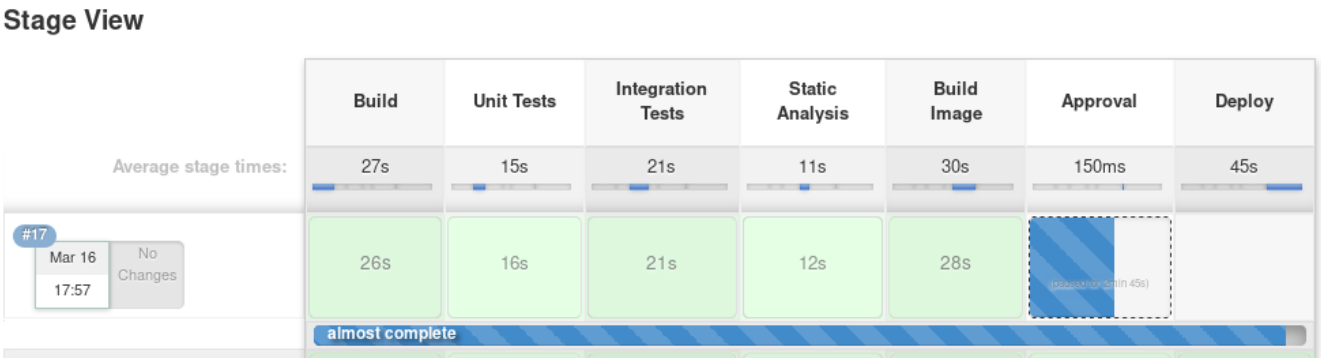
- **STAGE 2: unit test,** where we test the chunk/unit of software, unit by unit.
- **STAGE 3: Integration test,** where all individual units are combined and tested as a single product.
- **STAGE 4: Static Analysis** is the testing and evaluation of an application by examining the code without executing the application.

**STAGE 5: Test Build**

The output from unit and integration tests is a coverage report / release notes, which will be one of the artifacts used by SonarCube server to generate quality metrics and the other one is the application build (Successful Test Build file) which we upload to JFrog repository for future use. If the quality gate defined in the SonarCube server is not met, the pipeline flow will fail.

**STAGE 6: Approval**

If everything went fine, the flow stops its execution and waits for approval. The Jenkins server provides an interface for someone with the right permissions to manually promote the deploy to next stage (this is done using **input** directive of jenkins pipeline).




**STAGE 7: Deployment Stage:**

- After the approval, the pipeline continues to execute the flow and goes on to the next step and for some we configured to deploy directly to live QA for further testing like executing functional, security, performance and UAT – User acceptance test cases.

- We upload the compiled and successfully tested artifact to the JFrog repository. Then we have a new application snapshot ready to be deployed to production after getting approvals from Release management Team & CAB – Change Advisory Board.

**KEY DEFINITIONS**

1. **Continuous Integration (CI)**: is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early. By integrating regularly, you can detect errors quickly, and locate them more easily.

| WHAT YOU NEED (COST) | WHAT YOU GAIN |
|---|---|
| • Your team will need to write automated tests for each new feature, improvement or bug fix.<br>• You need a continuous integration server that can monitor the main repository and run the tests automatically for every new commit pushed.<br>• Developers need to merge their changes as often as possible, at least once a day. | • Less bugs get shipped to production as regressions are captured early by the automated tests.<br>• Building the release is easy as all integration issues have been solved early.<br>• Less context switching as developers are alerted as soon as they break the build and can work on fixing it before they move to another task.<br>• Testing costs are reduced drastically CI server can run hundreds of tests in the matter of seconds.<br>• Your QA team spend less time testing and can focus on significant improvements to the quality culture. |

2. **Continuous Delivery (CD):** Continuous delivery is an extension of continuous integration to make sure that you can release new changes to your customers quickly in a sustainable way. This means that on top of having automated your testing, you also have automated your release process and you can deploy your application at any point of time by clicking on a button.

   with continuous delivery, you can decide to release daily, weekly, fortnightly, or whatever suits your business requirements. However, if you truly want to get the benefits of continuous delivery, you should deploy to production as early as possible to make sure that you release small batches, that are easy to troubleshoot in case of a problem which is called as microservices.

| WHAT YOU NEED (COST) | WHAT YOU GAIN |
|---|---|
| • You need a strong foundation in continuous integration and your test suite needs to cover enough of your codebase.<br>• Deployments need to be automated. The trigger is still manual but once a deployment is started there shouldn't be a need for human intervention. | • The complexity of deploying software has been taken away. Your team doesn't have to spend days preparing for a release anymore.<br>• You can release more often, thus accelerating the feedback loop with your customers. |

- Your team will most likely need to embrace feature flags so that incomplete features do not affect customers in production.
- There is much less pressure on decisions for small changes, hence encouraging iterating faster.

3. **Continuous Deployment:** Continuous deployment goes one step further than continuous delivery. With this practice, every change that passes all stages of your production pipeline is released to your customers. There's no human intervention, and only a failed test will prevent a new change to be deployed to production.

   Continuous deployment is an excellent way to accelerate the feedback loop with your customers and take pressure off the team as there isn't a *Release Day* anymore. Developers can focus on building software, and they see their work go live minutes after they've finished working on it.

**WHAT YOU NEED (COST)**

- Your testing culture needs to be at its best. The quality of your test suite will determine the quality of your releases.
- Your documentation process will need to keep up with the pace of deployments.
- Feature flags become an inherent part of the process of releasing significant changes to make sure you can coordinate with other departments (Support, Marketing, PR...).

**WHAT YOU GAIN**

- You can develop faster as there's no need to pause development for releases. Deployments pipelines are triggered automatically for every change.
- Releases are less risky and easier to fix in case of problem as you deploy small batches of changes.
- Customers see a continuous stream of improvements, and quality increases every day, instead of every month, quarter or year.

4. **Sandbox:** A Development like environment is where developers configure, customize, and use source control to build an image of the application to be promoted to next environment. Developer checks his new code changes in this environment that will follow into next target environments.

5. **Release Notes**: are documents that are distributed with software products, sometimes when the product is still in the development or test state (e.g., a beta release). For products that have already been in use by clients, the release note is delivered to the customer when an update is released.

   **Contents of release note:**
   - **Header** – Document Name (i.e. Release Notes), product name, release number, release date, note date, note version, etc.
   - **Overview** - A brief overview of the product and changes, in the absence of other formal documentation.
   - **Purpose** - A brief overview of the purpose of the release note with a listing of what is new in this release, including bug fixes and new features.
   - **Issue Summary** - A short description of the bug or the enhancement in the release.
   - **Steps to Reproduce** - The steps that were followed when the bug was encountered.
   - **Resolution** - A short description of the modification/enhancement that was made to fix the bug.

- **End-User Impact** - What different actions are needed by the end-users of the application. This should include whether other functionality is impacted by these changes.
- **Support Impacts** - Changes required in the daily process of administering the software.
- **Notes** - Notes about software or hardware installation, upgrades and product documentation (including documentation updates)
- **Disclaimers** - Company and standard product related messages. e.g.; freeware, anti-piracy etc..
- **Contact** - Support contact information.