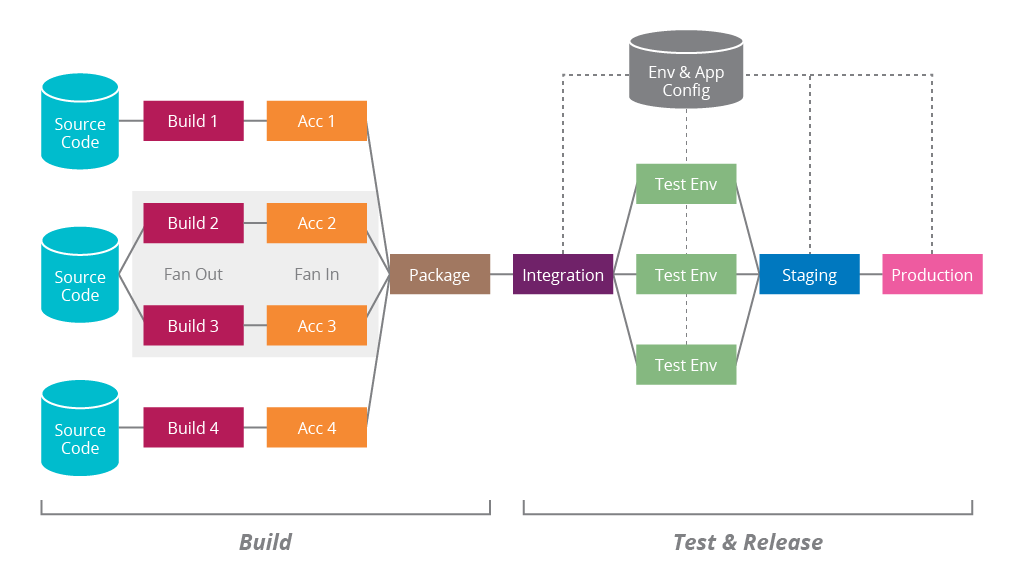
# Lecture 07 – Pipeline Sequences



We know with the power of continuous Integration numerous developers can all work on the same code base, making rapid changes without affecting the work of others. Each merge into a defined branch triggers an automated build-and-test sequence which can provide immediate feedback to developers within minutes. Ideally, **the CI feedback loop should not exceed 10 minutes.**

**Continuous delivery is the next part of the release process** which automates what was once considered a laborious and fragile task to complete. With continuous delivery, we get the benefits of infrastructure provisioning and deployment which may consist of multiple stages. All these processes encapsulate an automated set of steps and with each run, we’re able to produce logs and raise transparency for the entire team.

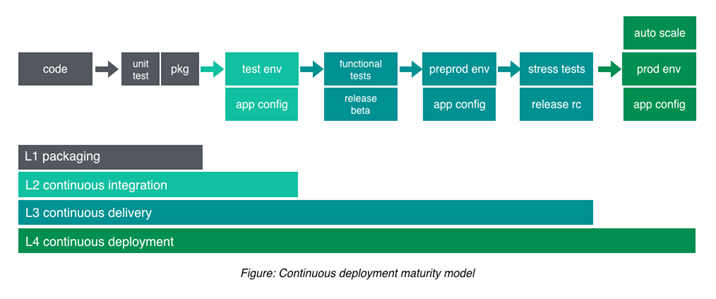
To bring all these pipeline pieces together, it may sound like a lot of overhead, but it really isn’t. In our labs we were able to build our own CI process which takes care of deployments to our static hosted application when new code changes are merged. We can deliver new versions of our application now with only a dozen or so key strokes. If we wanted to take things a step further, we could write a bash script to further simplify things for us. In this week's lab activities, we’ll see how with the power of bash, we can write scripts to handle what small steps we have left to have new versions shipped to production.

We know the stages that software releases must go through. It usually starts with a git push or a code merge into a production-like branch and from there the application is built. Builds can come in the forms of compilations of containerizations. From there the **source is tested with common paradigms like smoke, unit or integration testing**. Finally, after the test suites have been successfully completed, a deployment is made. **Deploys will usually first find its way to a QA environment where it can be qualified for a release to production.**

Failure anywhere in the pipeline will trigger a notification. The notification can come in a variety of forms but most commonly, web hooks will be used for Slack, SMTP or SMS alerts. Depending on which part of the pipeline sequence fails, the responsible team will be made aware of the issue. Otherwise notifications are commonly sent to a team after each successful deploy.

When designing a CI/CD pipeline it’s most wise to take the mentality of maturing sequences slowly. Chasing perfection can be good but one must be realistic with their objective, is proposed solution a must have? Is it intended to solve an apparent problem that will have an immediate boost in productivity? **Do not over-invest in your CI/CD pipeline sequence wise. Ensure that current problems are being solved before the next one is tackled.** Try to leave the theoretical problems for future tasks.

Below is a great model to demonstrate what’s known as a **maturity model. It helps visualize the stages or sequences of a pipeline and the way in which it matures in a linear format**

An engineer will want to assess where the project currently stands and what problems you’re faced with in the immediate future. What problems are currently stalemating work today? Find out what’s blocking the immediate day-to-day tasks and then move up the model. Each sequence in the model must have rationalized intent behind it. It may sound obvious but it’s easy to over think things.

## Step-By-Step Pipeline Sequencing

1. Developer commits a change
   * Alice has been working on a feature request and she has just finished qualifying her work and it’s ready to make a Pull Request for a peer review which is conducted by the rest of her team
   * Bob, John and Stacy have all approved Alice’s work and she’s been notified that the adequate number of reviewers have given their blessing on the code change
   * Alice then goes on to merge her change into a branch which contains the source code repository for what’s currently on production.
2. A build is then triggered
   * The build process kicks in and the source code with Alice's change begins to move through to pipeline
3. An application build takes place
   * Depending on the type of source code that’s moving through the pipeline, this is where compilation takes place. Since Alice is working on a Node application, npm is used to run the build script found in the projects package.json. If the application builds successfully, Alice and her team are notified of the successful build. Should it fail, a notification of the failure is sent out for the team to investigate why and the source code reverts to its last stable state.
4. Test suits are run
   * After the application has been determined that it successfully compiled, the associated test suites are now executed. Since the company, IdeaTree where Alice is employed utilizes unit tests along with a well-defined code convention, both code coverage and linting rules are tested. If the code coverage did not degrade and all unit tests passed, the first test suite gives the green light. The source code is then analyzed to ensure that no newly introduced syntax breaks the convention.
   * If no unit tests break due to Alice’s code change and code coverage does not decline due to a lack of unit tests for the new work, a notification is sent to inform the team of the test cases completing successfully. Should any tests fail, the pipeline process ends and the change is not deployed.
5. The build is delivered to an environment
   * Now that the code changes have been qualified with automated test suites and has been confirmed that it’ll successfully compile, the application is built. Alice’s company, IdeaTree, uses Docker for containerization so the application is now placed in an isolated container and delivered to the QA server where a Quality Assurance checks her work.
   * The QA analyst determines that Alice’s change meets the requirements to fulfill the feature request. Transparency is raised for project managers to make them aware that the feature has been qualified and green lit for a production deployment. It is now to be included in the next release schedule. If Alice’s work did not meet the requirements for the feature, the QA analyst informs Alice that there are some missing requirements, or it introduced unintended behavior which is now affecting a separate part of the application when regression testing is performed. Alice is notified and depending on the application state after the new change was integrated, the Docker build on the server is reverted to the last known working state or Alice starts working on what needs fixing.
6. The application is ready for a production release and deployed where necessary
   * The QA server where Alice’s change lives at IdeaTree has been qualified by an analyst and marked for deployment. A release is scheduled, and her change is bundled in with other feature requests for IdeaTrees next product release.