Automation Testing and DEVOPS Conceptual Training - Session 1

Learn Automation Testing and DEVOPS Concepts in Depth

Session#1: CICD Concept

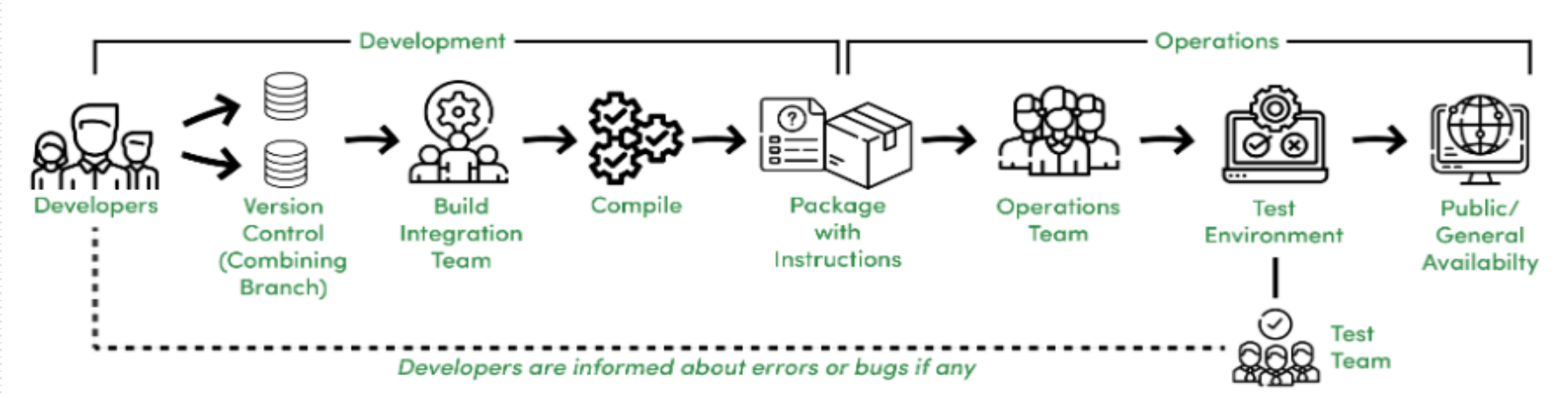
Trainer – Haradhan Pal

Agenda

- Traditional Development and Deployment Process
- Approach to Overcome Challenges in Traditional Process
- What is CICD?
- Stages of CICD
- Benefits of CICD
- Metrics to Measure CI/CD Performance

Traditional Development and Deployment Process

- Before CICD came into picture, traditionally software developers used to write codes for weeks and months in an isolated way, merging their code and handed over the code to the QA team for testing which then handed over the final release to the operations team for deployment. There is lack of collaboration between all these phases i.e. development, testing, and Deployment.
- Developers writes the code and hands over to the deployment team. Now it's up to deployment team to fix the problems that arise during deployment of code or hands over the code altogether back to the development team to fix the bugs. Everywhere manual intervention is required and the whole process used to consume additional time.



- Problem with Traditional Deployment Process:
  - Thorough Manual intervention during compilation/build/deployment process
  - Slowing down the overall process which use to increase overall schedule, efforts and cost
  - No unity, Integrity, and commitment between all involved stakeholders
  - No singular dynamic teamwork and leadership as a central authority
  - No Enough space for new ideas to implement
  - No sufficient space for customer's feedback and the customized final product

Approach to Overcome Challenges in Traditional Process

- To overcome the challenges in traditional process, CICD came into the market with some key features:

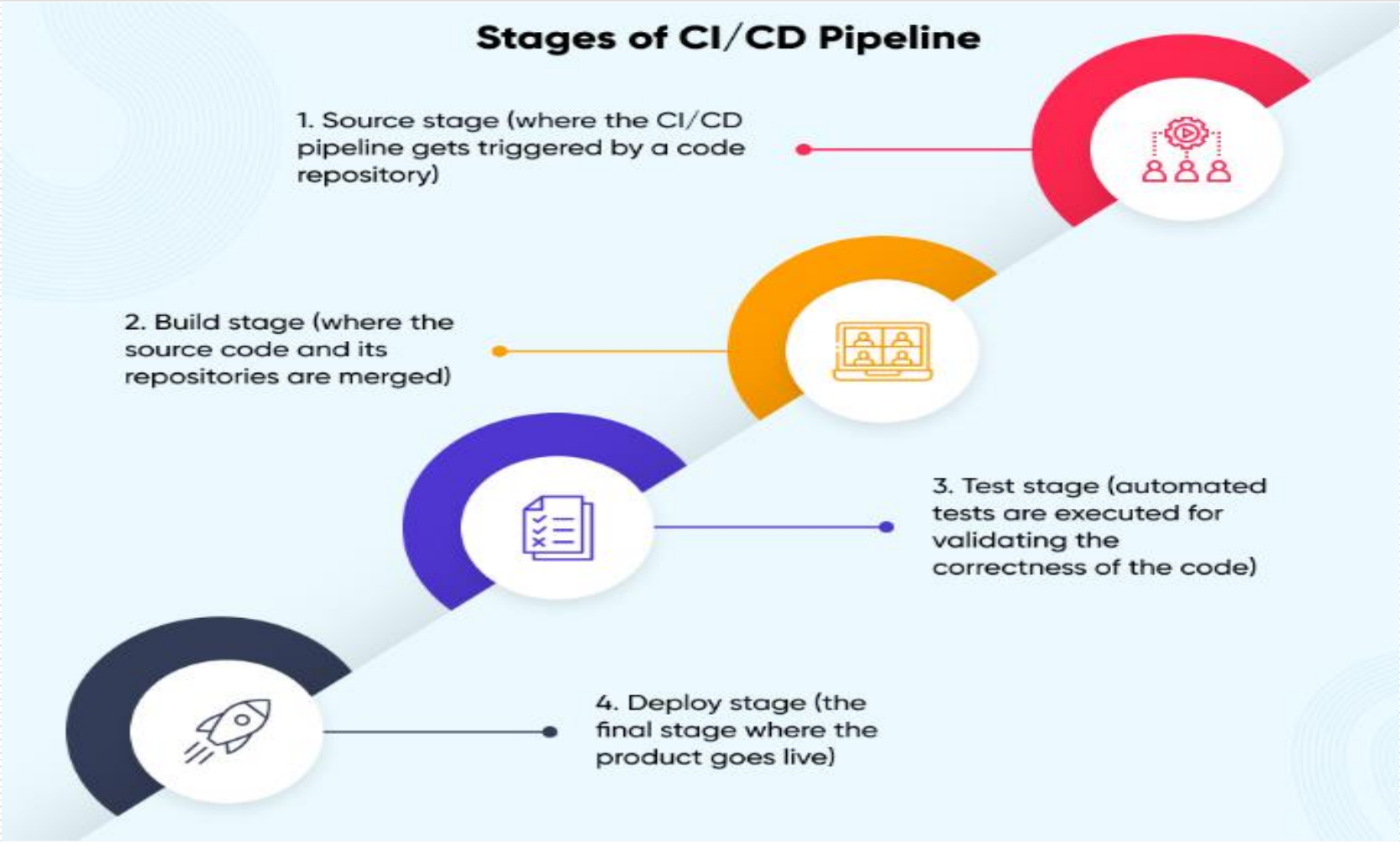
  - Reduce manual interventions for compilation/build/deployment process
  - Centralize the code repository
  - Automated and quick code integration
  - Automated unit/functional testing which reduces test automation cycle time
  - Mitigating Integration Challenges during Software Deployment
  - Build support and enthusiasm among stakeholders in the organization
  - Deployment several times in a day
  - Overall view on delivered application with continuous monitoring approach

What is CICD?

- The acronym CI/CD has a few different meanings. The "CI" in CI/CD always refers to Continuous Integration whereas "CD" refers to Continuous Delivery and/or Continuous Deployment.

- Continuous Integration: Development team members across the globe can integrate and merge their code using SCM (Source code management) tool like git on shared repository like GitHub, GitLab or Bitbucket several number of times a day which need to be validated by automated test scripts (smoke/unit/regression) to ensure the changes haven't breaking the existing functionality or application. If automated testing discovers a conflict between new and existing code, CI makes it easier to fix those defects quickly and often.
- Continuous Delivery: Deployment of code in production like an environment or some staging environment and running automated scripts to ensure the code is ready for release. The goal of continuous delivery is to have a codebase that is always ready for deployment to a production environment. At the end of that process, the operations team is able to deploy an app to production quickly and easily.
- Continuous Deployment: The final stage of a mature CI/CD pipeline is continuous deployment. As an extension of continuous delivery, continuous deployment automates releasing an app to production. it will depend on the customer's need for frequent deployment. Continuous deployment should be the goal of most companies that are not constrained by regulatory or other requirements.

Stages of CICD



Benefits of CICD

- **Cost Reduction:** Automation in the CI/CD pipeline reduces the number of errors that can take place in the many repetitive steps of CI and CD. Doing so also frees up developer time that could be spent on product development as there aren't as many code changes to fix down the road if the error is caught quickly. Another thing to keep in mind: increasing code quality with automation also increases Returns on Investment (ROI).
- **Smaller Code Changes:** Using continuous testing, small pieces of code can be tested as soon as they are integrated into the code repository, allowing developers to recognize a problem before too much work is completed afterward. This works really well for large development teams who work remotely as well as those in-house.
- **Smaller Backlog:** Incorporating CI/CD in any project, development process reduces the number of non-critical defects in backlog. These small defects are detected prior to production and fixed before being released to end-users.
- **Increased Focus on Core Responsibilities:** The development and the operations team (DevOps) can focus on their core competencies. Also, teams do not have to wait for manual sign-offs between them for executing integration, delivery, and deployment — as everything will be automated.
- **Better Code Quality:** A system's chances are reduced, with code being added to the central repository almost daily. These everyday additions make continuous integration and delivery easier as there are small code fragments to deal with, thus less probability of emerging anti-patterns and bugs.
- **Fault Isolations:** Fault isolations combine monitoring the system, identifying when the fault occurred, and triggering its location. Thus, the consequences of bugs appearing in the application are limited in scope. Sudden breakdowns and other critical issues can be prevented from occurring with the ability to isolate the problem before it can cause damage to the entire system.
- **Ensures Faster Time to Market:** The CI/CD pipeline ensures that automation takes over the integration, delivery, and deployment process, which further leads to — a faster time to market. The main objective of CI/CD is to reduce the time to market that otherwise used to take years due to broken processes and minimal collaboration between development and operations. ensure faster time to market.
- **Accurate progress monitoring:** A majority of the CI/CD tools can be used for instrumenting the process and provide a set of metrics that include building, test coverage, failure rates, test fix times, and more. This data can help user to recognize areas causing their pipeline to perform slowly and make improvements as and when required.
- **Maximum End User demand satisfaction:** Implementing CI/CD can facilitate end-user involvement and feedback in the continuous development stage that helps usability modifications. Moreover, it lets user keep their product up-to-date with constant checks for new updates or minor changes. These factors contribute to a high level of user satisfaction.
- **Cloud-based development:** CI/CD pipeline leverages cloud-native tools like Jenkins to automate the DevOps workflow, which initiates the testing process early on. Therefore, changes and new code blocks are automatically deployed in the production environment. This process maintains the continuous flow of new features and updates – an essential aspect of modern software development.

Metrics to Measure CI/CD Performance

- Metrics are essential tool for improving system performance – they help to identify where user can add value and offer a baseline against which to measure the impact of any improvements they make. Performance in CI/CD encompasses both speed and quality. A high performing CI/CD pipeline will allow user to deliver a robust and reliable product.
- The following four metrics have been identified by DevOps Research and Assessment (DORA) as high-level metrics that provide an accurate indication of how well an organization is performing in the context of software development.
  - **Build Duration:** Build duration or build time measures the time taken to complete the various stages of the automated pipeline. Looking at the time spent at each stage of the process is useful for spotting pain points or bottlenecks that might be slowing down the overall time it takes to get feedback from tests or deploy to live.
  - **Development Frequency:** Deployment frequency records the number of times user use their CI/CD pipeline to deploy to production. Deployment frequency was selected by DORA as a proxy for batch size, as a high deployment frequency implies fewer changes per deployment.
  - **Lead Time:** Although lead time (also known as time to delivery or time to market) can be measured as the time from when a feature is first raised until it is released to users, the time involved in ideation, user research and prototyping tends to be highly variable.
  - **Change Failure Rate:** Change failure rate refers to the proportion of changes deployed to production, which result in a failure, such as an outage or bug that requires either a rollback or hotfix.