**DEVOPS ASSIGNMENT 1**

Deepthi N

4NI19IS027

**INTRODUCTION**

It represents the combination of Development (Dev) and Operations (Ops). It  
has given rise to a movement that advocates bringing developers and operations together  
within teams. This delivers added business value to users more quickly, which makes it  
more competitive in the market.  
DevOps culture is a set of practices that reduce the barriers between developers, who  
want to innovate and deliver faster, and operations, who want to guarantee the stability  
of production systems and the quality of the system changes they make.  
DevOps culture is also the extension of agile processes (Scrum, XP, and so on), which makes  
it possible to reduce delivery times and already involves developers and business teams.  
However, they are often hindered because of the non-inclusion of Ops in the same teams.  
The communication and this link between Dev and Ops allows a better follow-up of  
end-to-end production deployments and more frequent deployments that are of higher  
quality, saving money for the company.  
To facilitate this collaboration and to improve communication between Dev and Ops,  
there are several key elements in the processes that must be put in place, as shown here:

* More frequent application deployments with integration and continuous delivery  
  (called CI/CD).
* The implementation and automation of unitary and integration tests, with a process  
  focused on behaviour-driven design (BDD) or test-driven design (TDD).
* The implementation of a means of collecting feedback from users.  
   Monitoring applications and infrastructure.

**WHAT IS CI/CD?**

Behind the acronyms of CI/CD, there are three practices:

* Continuous integration (CI)
* Continuous delivery (CD)
* Continuous deployment

Continuous Integration:

The "CI" in CI/CD always refers to continuous integration, which is an automation process for developers. Successful CI means new code changes to an app are regularly built, tested, and merged to a shared repository.

Continuous Delivery:

The "CD" in CI/CD refers to continuous delivery and/or continuous deployment, which are related concepts that sometimes get used interchangeably. Both are about automating further stages of the pipeline, but they’re sometimes used separately to illustrate just how much automation is happening.

Continuous delivery usually means a developer’s changes to an application are automatically bug tested and uploaded to a repository (like GitHub or a container registry), where they can then be deployed to a live production environment by the operations team.

Continuous deployment:

Continuous deployment (the other possible "CD") can refer to automatically releasing a developer’s changes from the repository to production, where it is usable by customers. It addresses the problem of overloading operations teams with manual processes that slow down app delivery. It builds on the benefits of continuous delivery by automating the next stage in the pipeline.

Text

Description automatically generated with medium confidence

**PHASES OF CI/CD PIPELINE**

From source code to production, these phases make up the development lifecycle and workflow of the CI/CD pipeline:

* **Build:** This phase is part of the continuous integration process and involves the creation and compiling of code. Teams build off of source code collaboratively and integrate new code while quickly determining any issues or conflicts.
* **Test:** At this stage, teams test the code. Automated tests happen in both continuous delivery and deployment. These tests could include integration tests, unit tests, and regression tests.
* **Deliver:** Here, an approved codebase is sent to a production environment. This stage is automated in continuous deployment and is only automated in continuous delivery after developer approval.
* **Deploy:** Lastly, the changes are deployed, and the final product moves into production. In continuous delivery, products or code are sent to repositories and then moved into production or deployment by human approval. In continuous deployment, this step is automated.

**WHAT ARE THE BENEFITS?**

Automation of software releases — from initial testing to the final deployment — is a significant benefit of the CI/CD pipeline. Additional benefits of the CI/CD process for development teams include the following:

* **Reducing time to deployment through automation**
* **Decreasing the costs associated with traditional software development**
* **Continuous feedback for improvement**
* **Improving the ability to address error detection earlier in the development process**
* **Improving team collaboration and system integration**

**FEATURE FLAGS**

Feature flags are one of the tools that make your CI/CD pipeline work better. They help your team shepherd features through your pipeline safely, quickly, and effectively. They make your team more productive while giving them feedback on how they’re doing at the same time.

Feature flags are a way to change application behaviour without modifying code or deploying a new version. They give you the ability to safely deliver new features to clients quickly with minimal risk.

A flag is a software toggle that enables or disables blocks of code. Think of an “if” statement in your favourite program language that is true only if the related feature flag is enabled. If it’s on, the block of code is available. If it’s off, the code is not. A feature is a set of application behaviour. Depending on how you implement it, it may comprise more than one block of code. So, feature toggle tools have a way to group blocks into a single toggle, giving you a way to associate a single flag with all the code related to a feature.

You can put flags into at least [four categories](https://www.martinfowler.com/articles/feature-toggles.html#CategoriesOfToggles):

1. release
2. experiment
3. ops
4. permissioning

Release flags should have short life cycles tied to the feature they toggle. When the feature is complete, you remove the flag. Experiment toggles can live for a long time since you can repeatedly refine the experiment. Permissioning flags might live forever, so the code they control needs to be designed with the flag in mind.

**COMMON TOOLS**

* [Deploying Jenkins on Red Hat OpenShift](https://cloud.redhat.com/blog/deploying-jenkins-on-openshift-part-1?extIdCarryOver=true&intcmp=7013a000002wBnmAAE&sc_cid=7013a000002DgC5AAK'%5d%5d)
* [Spinnaker](https://spinnaker.io/), a CD platform built for [multi-cloud](https://www.redhat.com/en/topics/cloud-computing/what-is-multicloud) environments.
* [GoCD](https://www.gocd.org/), a CI/CD server with an emphasis on modelling and visualization.
* [Concourse](https://concourse-ci.org/), "an open-source continuous thing-doer."
* [Screwdriver](https://screwdriver.cd/), a build platform designed for CD.

**REFERENCES**

1. <https://www.atlassian.com/continuous-delivery/principles/feature-flags>
2. <https://stackify.com/what-is-cicd-whats-important-and-how-to-get-it-right/>
3. https://www.servicenow.com/products/devops/what-is-cicd.html