DevOps

The term Devops was introduced in 2007 - 2009 by patrick Debois, Gene kim and John willis, and its represents the combination of Development (Dev) and Operations (Ops).

DevOps culture reduce the barrier b/w developers, who wants to innovate and deliver fast, on the other hand operations, who want to guarantee the stability of production systems and the quality of the system.

- It is an extension of Agile system / processes.

To facilitate the collaboration and improve communication b/w Dev and Ops, There are several key elements

- [More frequent Application deployment with integration and continous delivery (called CI/CD)]
- The implementation and automation of unitary and integration tests, with a process focused on Behavior - Driven Design (BDD) or test - Driven Design (TDD)

Product owner + DEV + Quality Analyst

- The implementation of a means of collecting feedback from users
- Monitoring applications and infrastructure

The DevOps movement is based on three axes :

- <u>The culture of collaboration</u> :- DevOps, no longer seperates team Developers and team Ops ~~and so on~~ ~~but on the contrary~~, these people are brought together to deliver added value to the product as quickly as possible.

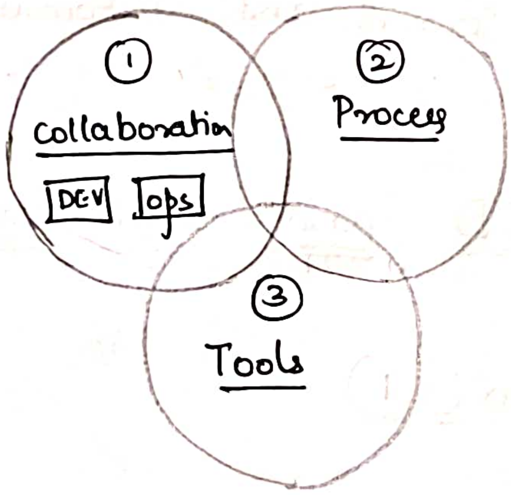
- <u>Processes</u> :- To expect rapid deployment, these teams must follow development processes from agile methodologies. These processes should not only be integrated into the development workflow with continuous integration but also into the development workflow with continous delivery and deployment.

- → There are several phases in DevOps process
  - * The planning and prioritizing of functionalities.
  - * Development
  - * Continous integration and delivery
  - * Continous deployment
  - * Continous monitoring.

• Tools :- The choice of tools and products used by teams is very important in DevOps.

The creation and updating of the infrastructure and integrate the code into a code manager; this is called Infrastructure as code

The DevOps culture



DevOps is the union of people, process and products to enable continous delivery of value to our end users.

The benefits of establishing a DevOps culture within an enterprise are as follows

- Better collaboration and communication in teams,

~~which~~

- Shorter lead times to production, resulting in better performance and end user satisfaction.
- Reduced infrastructure costs of IaC.
- Significant time saved with iterative cycles that reduce application errors and automation tools.

Implementing CI/CD and Continuous deployment

* Continous Integration (CI)

Continous Integration is a software development practice where members of a team integrate their work frequently... Each integration is verified by an automated build to detect integration errors as quickly as possible.

that is, CI is an automatic process that allows you to check the completeness of an application's code every time a team member makes a change. This verification must be done as quickly as possible.

Implementing CI

- To set up CI, it is necessary to have a Source code Manager (SCM) that will allow the centralization of the code of all members. This code manager can be Git, SVM, TFVC etc.

- It is also necessary to have an automatic build manager (CI server) that supports continous integration such as jenkins, Gitlab CI, Azure Pipeline etc.

- Each team member will work on the application code dialy, iteratively and incrementally. Each task or feature must be partitioned from other development with the use of branches.

- Regularly members archeve and commit their code. This will, therefore, be integrated into the rest of the code of the application with all the other commits of the other members.

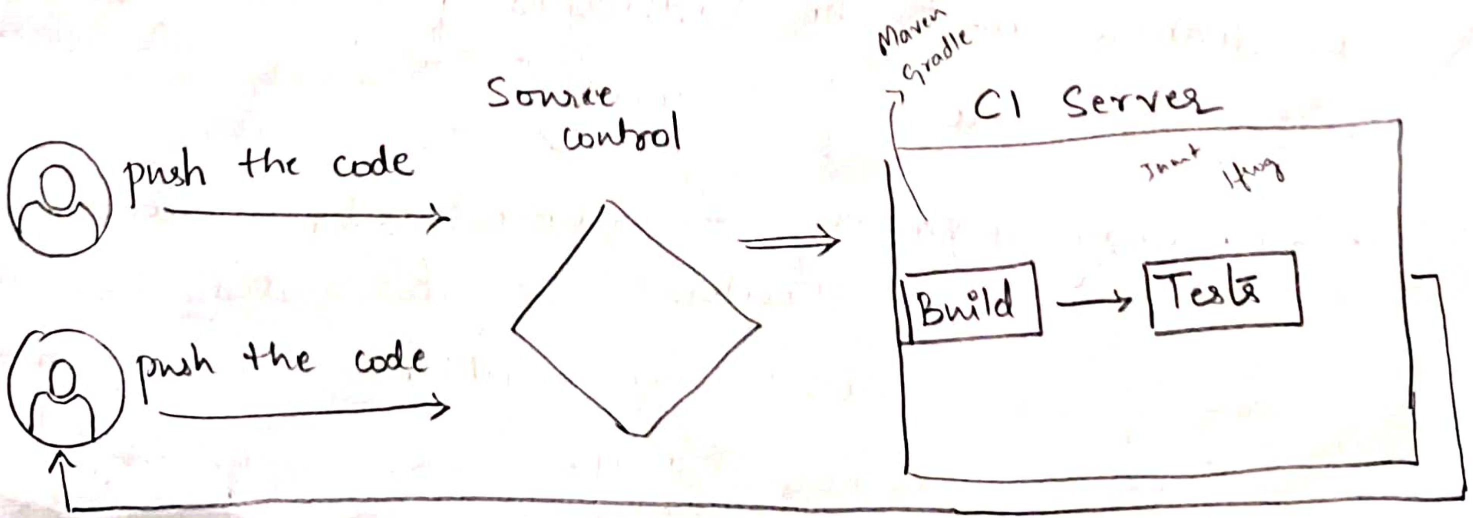
This integration of all the commit is the starting point of the CI process.

- Build the application package - compilation, file transformation,
- Perform unit tests.

are the 2 steps which performed, once ~~its~~ commit is triggered.

- This CI process must be optimized as soon as possible so that it can run fast and developers can have quick feedback on the integration of their code.

- With an optimized and complete CI process, the developer can quickly fix their problem and improve their code or discuss it with the ~~the~~ rest of the team and commit their code for a new integration



Continous delivery (CD)

Once continous integration has been successfully completed, the next step is to deploy the application automatically in one or more non production environments, which is called staging. This process is called continuous delivery (CD).

CD often starts with an application package prepared by CI, which will be installed according to a list of automated tasks. These tasks can be of any type: unzip, stop and restart service, copy files, replace configuration, and so on.

unlike CI, CD aims to test the entire application with all of its dependencies.

In prattice, today it is very common to link CI with CD in an integration environment; that is CI deploys at the same time in an environment.

It is very important that the package generated during CI and that will be deployed during CD is the same one that will be installed on all environments, and this should be the case until

There may be configuration file 'transformations that
production, differ depending on the environment, but the
application code must remain unchanged.
This immutable, unchangeable character of the code is the only guarantee that the application verified in an environment will be of the same quality as the version deployed in the previous environment and the same one that will be deployed in the next environment.

The tools set up for CI/CD are often completed with others solutions, which are as follows.

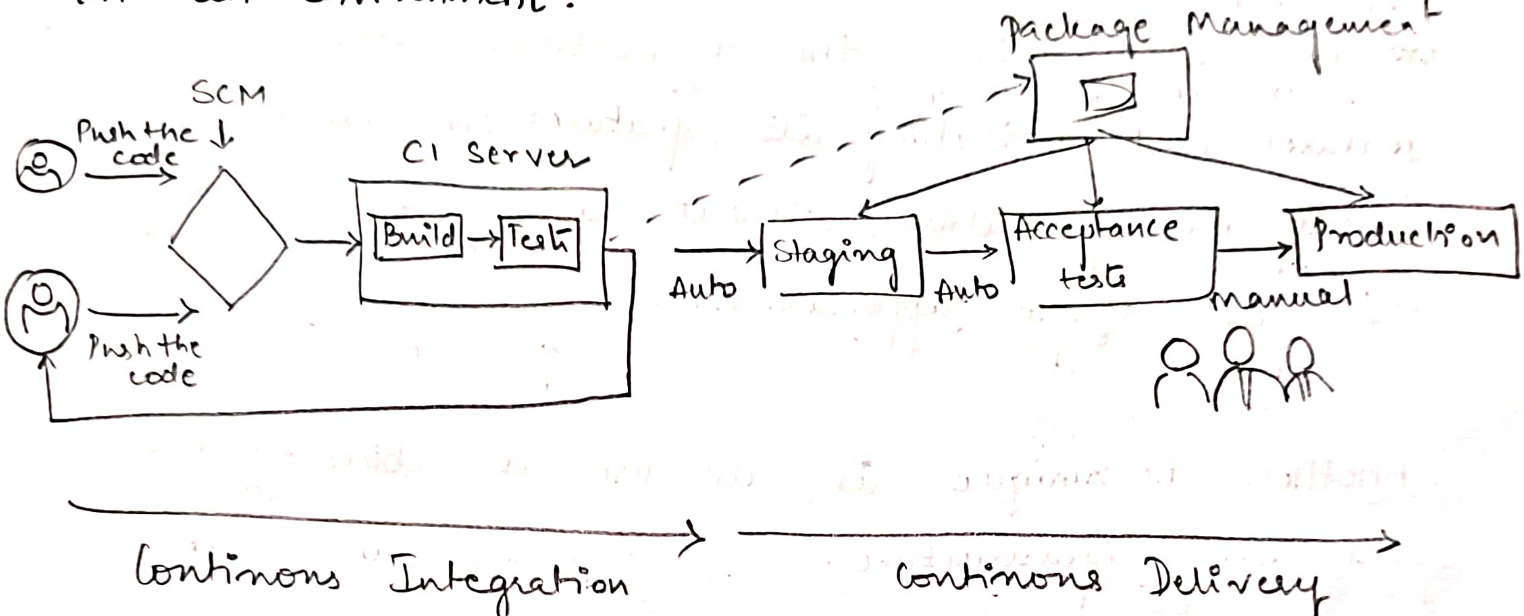
- A package manager :- This constitutes the storage space of the packages generated by CI and recovered by CD. These managers must support feeds, versioning and different types of packages.

- A configuration manager :- This allows you to manage configuration changes during CD

In CD, the deployment of the application in each staging environment is triggered as follows.

- It can be triggered automatically, following a successful execution on a previous environment.

• It can be triggered manually, for sensitive environments such as the production environment, following a manual approval by a person responsible for validating the proper functioning of the application in an environment.



Continous Deployment

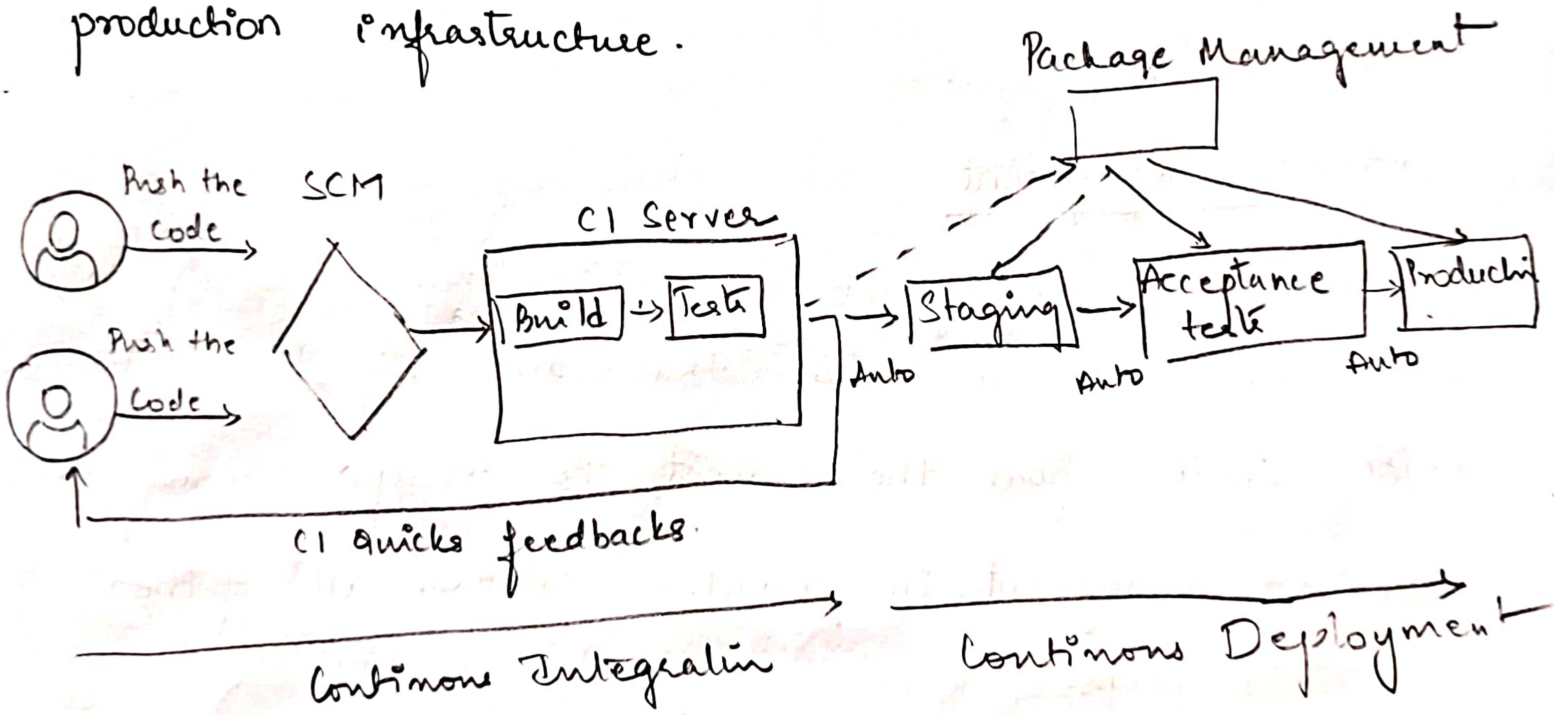
Continuous deployment is an extension of CD, but this time, with a process that automates the entire CI/CD pipeline from the moment the developer commits their code to deployment in production through all of the verification steps.

This requires wide coverage of tests (unit, functional, integration, performance and so on) for the application, and the successful execution of these tests is sufficient to validate the proper functioning of the application with all of these dependencies.

The continous deployment process must also take into account all of the steps to restore the application in the event of a production problem.

Continuous deployment can be implemented ~~with the use~~ by encapsulating the application's functionalities in features and activating its features on demand, directly in production, without having to redeploy the code of the application.

Another technique is to use a blue-green production infrastructure.



The preceding diagram is almost the same as that of CD, ~~but~~ but with the difference that it depicts automated end to end deployment.

Understanding IaC practices

IaC is the process of writting the code of the provisioning and configuration steps infrastructure components to automate its deployment in a repeatable and consistent manner.

The benefits of IaC

1) The standardization of infrastructure configuration reduces the risk of error.

2) The code that describes the infrastructure is versioned and controlled in a source code manager.

3) The code is integrated into CI/CD pipelines.

4) Deployments that make infrastructure changes are faster and more efficient.

5) There's better management, control and a reduction in Infrastructure costs.

IaC languages and tools

The languages and tools used to code the infrastructure can be of different types ; that is , scripting and declarative types.

Scripting types

These are scripts such as Bash, Powershell , or any other language that use the different clients provided by the cloud provider ; For Example , you can script the provisioning of an Azure infrastucture with the Azure CLI or Azure PowerShell.

For example

- Using the Azure CLI, we have the following

```
az group create -location westeurope -name
                                    MyAppResourcegroup
```

- Using Azure Powershell , we have the following.

```
New-AzResourceGroup -Name MyAppResourcegroup
                          -Location westeurope
```

The problems with these language and tools is that they require a lot of lines of code because we need to manage the different states of the manipulated resources and it is necessary to write all of the steps of the creation or update of the desired infrastructure.

Declarative types

These are languages in which it is sufficient to write the state of the desired system or infrastructure in the form of configurations and properties. For example, Terraform, Ansible, Powershell DSC, puppet and chef. The user only has to write the final state of the desired infrastructure and the tool takes care of Applying it.

For example

The following is the terraform code that allows you to define the desired configuration of an Azure resource group.

```
resource "azurerm_resource_group" "myrg" {
    name     = "My App Resource Group"
    location = "West Europe"

    tags = {
        environment = "Bookdemo"

    }.
}
```

In this example, if you want to add or modify a tag, just modify the tags property in the preceeding code Terraform will do the update itself.

Another example, ~~of~~ ~~you~~ ~~would~~ that allows you to install and restart nginx on a server using Ansible.

```
- hosts : all
  tasks :
- name : install and check nginx latest version
  apt : name = nginx   state = latest
- name : start nginx
service :
  name : nginx
  state : started .
```

And to ensure that the services is not installed just change the preceding code, with service as an absent value and the state property with the stopped value.

```
---
 - hosts : all
   tasks :
    name : stop nginx
   service :
     name ; nginx
    state : stopped
    -name : check nginx is not installed
     apt : name =nginx state =absent .
```