DevOps Writeup

# 1.0 Introduction

## 1.1 Lifecyclus

The lifecycle of an application:

* First we build the application
* What if the developer is done with development?
* What if the application runs in production
* What is in between, before and after?

Diagram

Description automatically generated

Flow of an application = From planning the building to planning for when the application is live

* What steps are in between?
* What has to happen during these steps

Diagram

Description automatically generated

## 1.2 Application flow

The flow of an application goes as followed:

* ~~Business requirement~~
* ~~Making of user stories~~
* ~~Creating wireframes and prototypes~~
* Development – creating new feature
* Development – Create Unit tests for new code
* Development – Install dependencies
* Ops - running test environment

QA – Responsible for tests

Ops – responsible for deploying application environment

Developer – Responsible for creating the application

Take feedback to new cycle

Wall of confused: A scenario in which development and Devops have fundamentally different thoughts

Developers wanting change while Devops want stability

Difference between functional and non-functional requirements:

* Functional requirements are:
  + Specification features
  + Prioritization of features
* Use automatic tests:
  + Automated functional tests
  + Integrated build process
* Aimed at business usage:
  + Bugs
  + evolution
* Non functional requirements are
  + Useful troubleshooting logs
  + Deployment
  + Undo-redo operations
* Non functional tests:
  + Automatic technical tests
  + Integrated build process
  + Continuous development
* Technical operation
  + performance and technical problems

Functional requirements describe what the system should do

Non functional requirements describe how the system works

# 2.0 Agile & Devops, The three ways

## 2.1 Agile & Devops

There are different concepts to take note of:

* Agile: Agile is that there is no one way to do something
* Waterfall: Maps out a project into distinct phases with each a beginning phase and an end phase
* Devops: Increases the organizations ability to deliver applications and services at high velocity
* Pipeline: Set of automated processes and tools used by developers and Devops
* Flow: An end-to-end manufacturing chain of software to run lines of code in production
* Feedback: Modification or control of a process or system
* Swarming: When as much team members as possible work on the same priority item
* Technical debt: The accumulation of sub optimal technical decision making made over the lifetime of an application

Why is Agile and Devops not the same:

* Devops is a practice bringing development and operations teams together
* Agile focusses on collaboration, customer feedback and small rapid release
* Devops focuses on constant testing and delivery
* Agile focusses on constant changes

A picture containing text, clipart

Description automatically generated

Different focus points within Devops for:

* developers: end user performance, analytics, code quality and code errors
* operations engineers: Availability, performance, user complains and performance analytics

## 2.2 The Three ways

The three ways:

* The first way: identify the component and make business idea a reality – Principles of flow
* The second way: Continuing to monitor, test and handle feedback  
  - Principles of feedback
* The third way: Continuous improvement, seeking to improve every process – Principles of continuous improvement

Principals of flow:

* work flowing from left to right
* Passing work through the different teams
* Each phase in Devops focusses on closing the loop between development and operations

Principals of feedback:

* Goes from right to left
* Providing feedback in a fast consistent way
* Quickly address and identify problems

Principals of continuous learning:

* Using the feedback to improve the process
* Implementing changes might be challenging in the organizational culture we work in

A testing pyramid is a concept that groups software tests into three different categories:

* UI/exploratory tests
* Integration tests
* Unit tests

Plus and minus points from testing pyramid

A positive is that you will have a nice way to test a negative is that it will take a lot of time

# 3.0 Source control

General concept:

* Source control: Practice of tracking and managing changes to code
* Git: Freely distributed source control system
* GitHub: Web based version-control platform for developers
* Flow: End to end manufacturing chain of software
* (Feature) Branch: Unique set of code changes with an unique name
* Merge: A way of putting a forked history back together again
* Merge conflict: When different changes are made on the same lines of a file
* Pull request: Lets you tell others about your changes you’ve pushed
* Commit: An individual change to a file
* Deploy: This is when you launch the code stored on a server of your own
* Main: The branch where all changes eventually go back to
* Markdown (md): Plaintext format of writing structured documents
* Single source of truth: Storing all code, configuration and other digital assets where everyone can access them

Why is source control important and for what is it used:

* To maintain a single source of truth
* To facilitate collaboration
* To accelerate release velocity
* It allows multiple developers at once
* It stores your data so when a drive failure occurs your code isn’t gone

Basic GIT commands:

* Create a new branch  
  | git branch new  
  | git checkout -b new
* Move to other branch  
  | git checkout new
* See all branches  
  | git branch -a
* Push new branch to git  
  | git push origin new
* Pull all branches from git  
  | git fetch –all
* Merge a branch named new with the branch you are in (after merging you still need to pull)  
  | git merge new
* Deleting a branch  
  | git branch -d new

A good git commit message contains clear feedback of changes

The git flow goes as followed:

* A developer creates a feature in a separate branch
* Developer pushed the branch to central repo
* Developer creates a pull request via Github
* The rest of the team goes over the code, gives feedback etc
* The responsible of the repo merges the changes to the main branch

A branching strategy defines how a team uses branches to achieve a level of concurrent development

Github flow is how we work with GitHub  
Diagram

Description automatically generated

We use main for deployment and develop to merge our changes

# 4.0 Flows -CI & CD concepts

The concepts:

* Pipeline: Ensures all code in version control is built and tested
* Continuous integration: Automating the integration of code changes from multiple contributors
* Continuous delivery: Ability to get changes of all types
* Build: The stage where the application is compiled
* Artifact: A by-product of the software development process
* Compiling code: transform source code to machine code
* build once deploy anywhere: an agile approach to creating new applications
* vendor agnostic: an organization that is open to all viable and established solutions
* vendor centric:
* Jenkins: Used to reliably build, test and deploy software

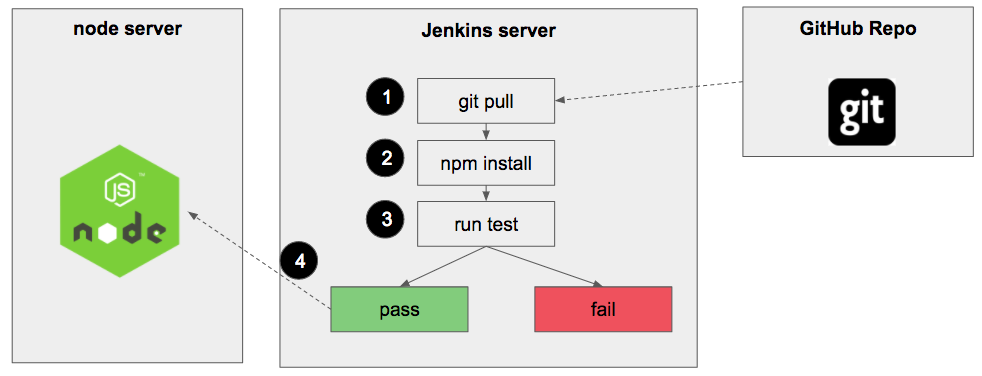
Phases in continuous integration:

* Unit test
* Code analysis
* test coverage

Stages in CI pipeline:

* Build
* Test
* Release
* Deploy

Global view on a CI pipeline in NodeJS



Different code testing:

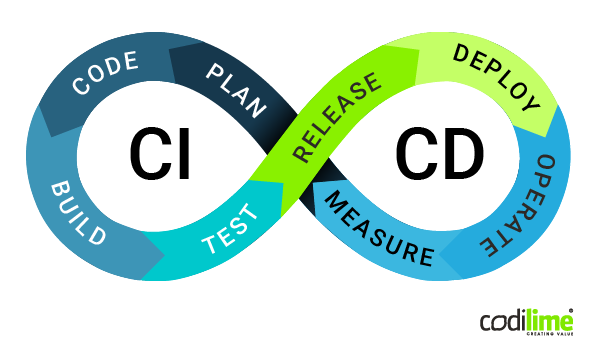
* Unit tests, automized tests

Reusing artifacts is essential for reusing your code and improving the build process

Phases in continuous delivery:

* Build
* Unit test
* Code Quality analysis
* Integration test
* Acceptance test
* Store versioned Binary Artifact

Phases of CD pipeline



Global view on a CD pipeline in NodeJS

Diagram

Description automatically generated

CI is simpler, its about using the right tools.  
CD is more complicated and is about processes that have to happen after code is integrated

We use this to integrate code into a mainline code base

**Jenkins helps in automate building and testing systems to the integration work**

# 5.0 CI in Jenkins

Define concepts

* Pipeline: Ensures all code in version control is built and tested
* Continuous integration: Automating the integration of code changes from multiple contributors
* Continuous delivery: Ability to get changes of all types
* Infrastructure as code: Being able to reproduce and/or restore a full infrastructure based on recipes and automation
* Stage view: Progress of each stage
* Build attempt: Jenkins tries to build the application
* Groovy: A language that offers the ability to do what Java can
* Pipeline script: A file named Jenkinsfile that supports implementing and integrating continuous delivery
* Stages: the build, test, and deploy processes all come together in a stage
* Steps a step is a single command that preforms a single action
* Agents: a machine, or container, which connects to a Jenkins controller and executes tasks when directed by the controller
* Post run steps: Can be set to only execute when build has a specific status
* Workspace: Where Jenkins builds your project

Freestyle projects are for simple jobs for a project, pipeline is better to set up a CD pipeline or to define the deploy the pipeline as code

A Jenkins pipeline is designed to perform the tasks when directed by the master

Plugins are used to extend the Jenkins build system

How does Jenkins integrate with Git? Go to Jenkins dashboard, click on “Manage Jenkins.” Now follow these steps- Manage Plugins -> 'Available' tab -> Enter Git in search bar and filter -> Install required plugin. After the installation, all you need to do is click on “Configure System” and go to the 'GitHub' section.

# 6.0 DTAP & environments on demand

Define concepts:

* DTAP: Phased approach to software testing and deployment
  + Development
  + Test
  + Acceptance
  + Production
* Development: Environment where software is developed
* Test: Environment where tests are done
* Acceptance/acceptation: When team decides product is ready it will be deployed to acceptance
* Production: Environment where product needs to be ready
* Continuous deployment: approach to development where the product has automated implementations
* Configuration management: A process for establishing and maintaining consistency in a products performance
* Application release automation: the process of packing and deploying an application or updating
* Docker: Allows developers to package application into containers
* Mutable: This means that a running container can be changed
* Immutable: This means that a running container never changes

To prevent a risk of killing a system

Configuration management is responsible for establishing consistency of a products attributes

Configuration management tools:

* Solar Winds
* Auvik
* CFEngine

A mutable environment can be changed easily but has also more security risks

# 7.0 Flows & feedback – Integrated testing

Define concepts:

* functional testing: Test if software works as required
* non-functional testing: Test of non functional like performance and reliability
* integration testing: testing different units as a combined entity
* unit testing: Testing a separate small piece of code, a function
* regression testing: Test to check if software still works as expected after change
* performance testing: Check system under particular workload
* security testing: Test for potential flaws in the security
* load testing: Test for multiple users accessing the program continuously
* stress testing: Deliberately intense testing to determine stability under high loads

Diagram

Description automatically generated

Manual testing takes a lot of resources away!

We can test API’s by doing calls and analysing these with Postman

To automate web apps functionally we can use Selenium

Automatization is important to spare time

# 8.0 Flows & feedback – Monitoring and reporting

Define concepts:

* monitoring: Overseeing the entire development process
* metrics: Data points to reveal performance of Devops software
* log aggregation: The process of collection, standardizing and consolidating log data
* snmp: Networking protocol used for management and monitoring of network connected devices
* Mib: Collection of information used for managing devices in the network
* agent software: Agent software is responsible for monitoring the other network devices on the network
* application performance management: Allows Devops to monitor and review backend applications
* proactive monitoring: Automated monitoring
* predictive monitoring: Monitoring irregularities before they develop
* AI ops: Artificial intelligence for IT Operators
* low risk releases: Releasing with the lowest possible risk attached
* rolling deploys: Release strategy that staggers development across multiple stages
* blue/green deploys: Change management strategy for releasing software code
* canary deploys: roll out code/features to a subset of users for tests
* A/B testing: Testing for reasons like usability, popularity and noticeability

In Devops things like network activity, performance and errors can all be logged

Some monitoring tools used in Devops:

* Sensu
* Librato
* Splunk

Monitor alerting is used to provide a health signal, gives a performance baseline and helps to notify when things go wrong

In Devops you can monitor for performance of a system but also network traffic and for irregularities in a system

Classic is the older generation of deployment in which supports continuous integration and continuous delivery to continuously test, build and deploy code, this is done by defining a pipeline

Modern deployment is a set of automated processes that allow bot developer and operators to work cohesively to build and deploy code to a production environment

You can use metrics to link them to abnormalities my looking at their uptime

# 9.0 Integration exercise

Define concepts:

* yaml: A data serialization language used for config files
* webhooks: Automated messages sent from apps when something happens
* cloud: used to access the same files and applications from almost any device
* environment variables: A dynamic-named value that effects the way a running process will behave
* secrets: Used for managing access permissions

Github actions is a continuous integration that allows you to automate your build, test and deployment pipeline

Heroku is used to deploy, manage and scale modern apps

Both Github actions and Jenkins allow you to create workflows automatically, build, test, release and deploy code

But Jenkins uses stages to run a collection of steps

Github Actions uses jobs to group one or more steps or individual commands

Environmental variables are changed so it can be used to modify the systems behaviour