

DevSecOps

Introduction

Name

Total Experience

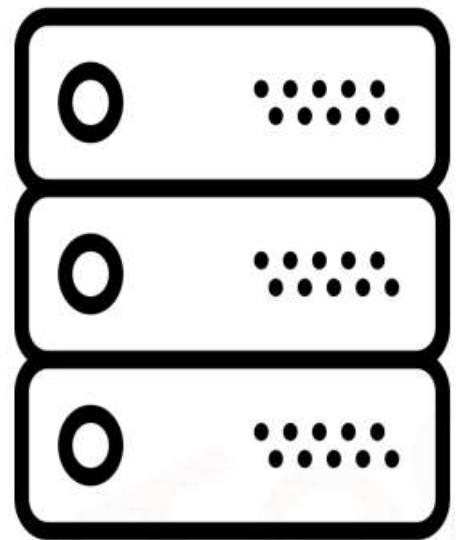
Background – Development / Infrastructure / Database / Network

Experience on monitoring tools

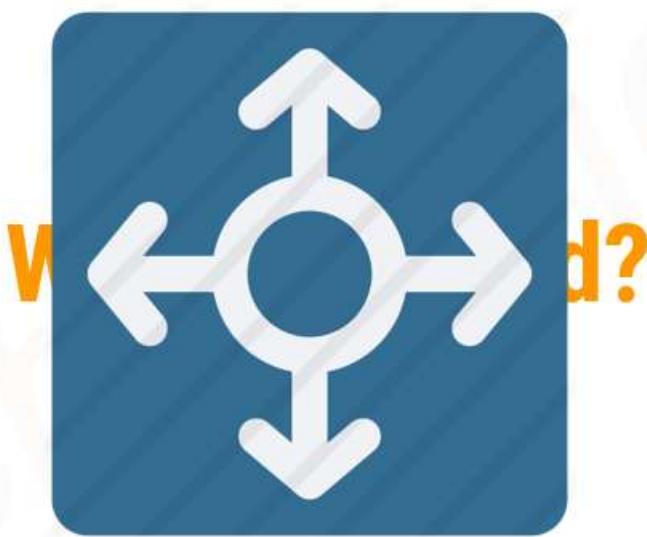
Your Expectations from this training

Module 1: Cloud Fundamentals & DevSecOps

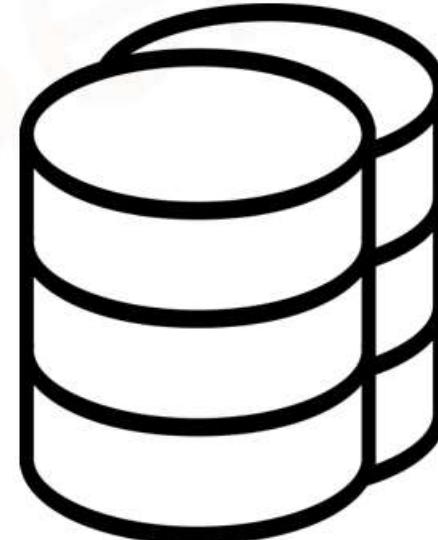
Traditional Datacenters



Servers

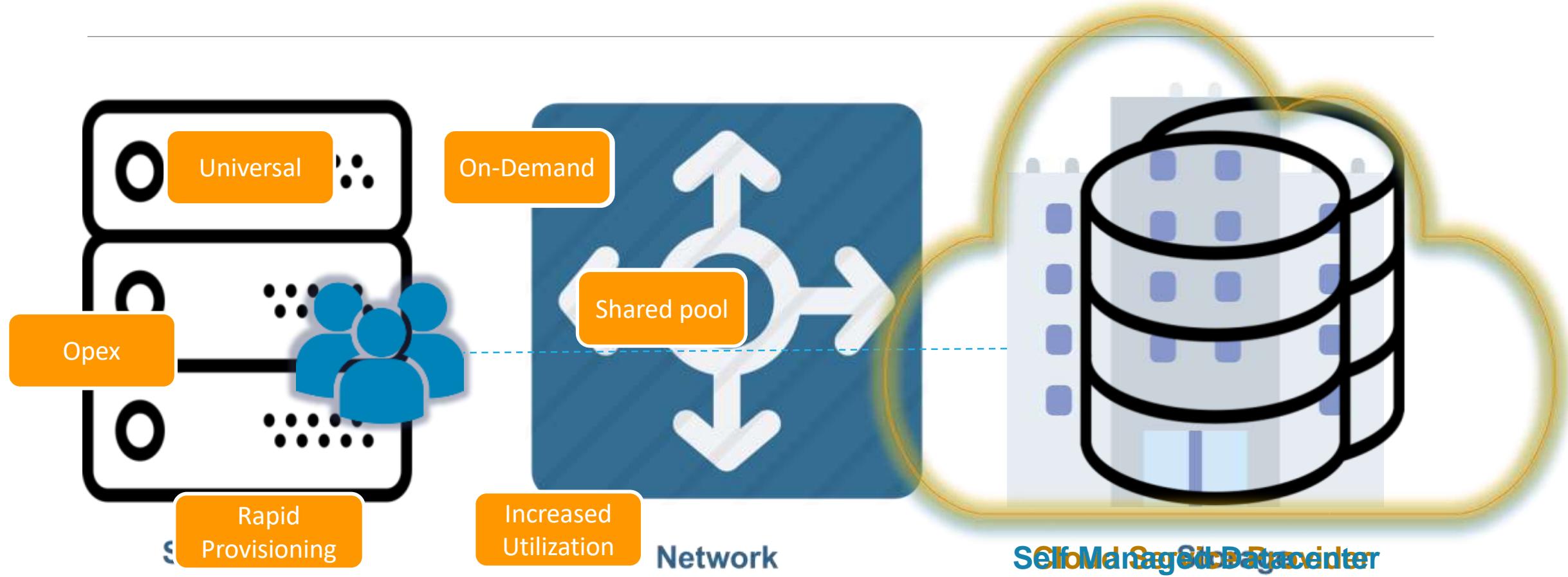


Network

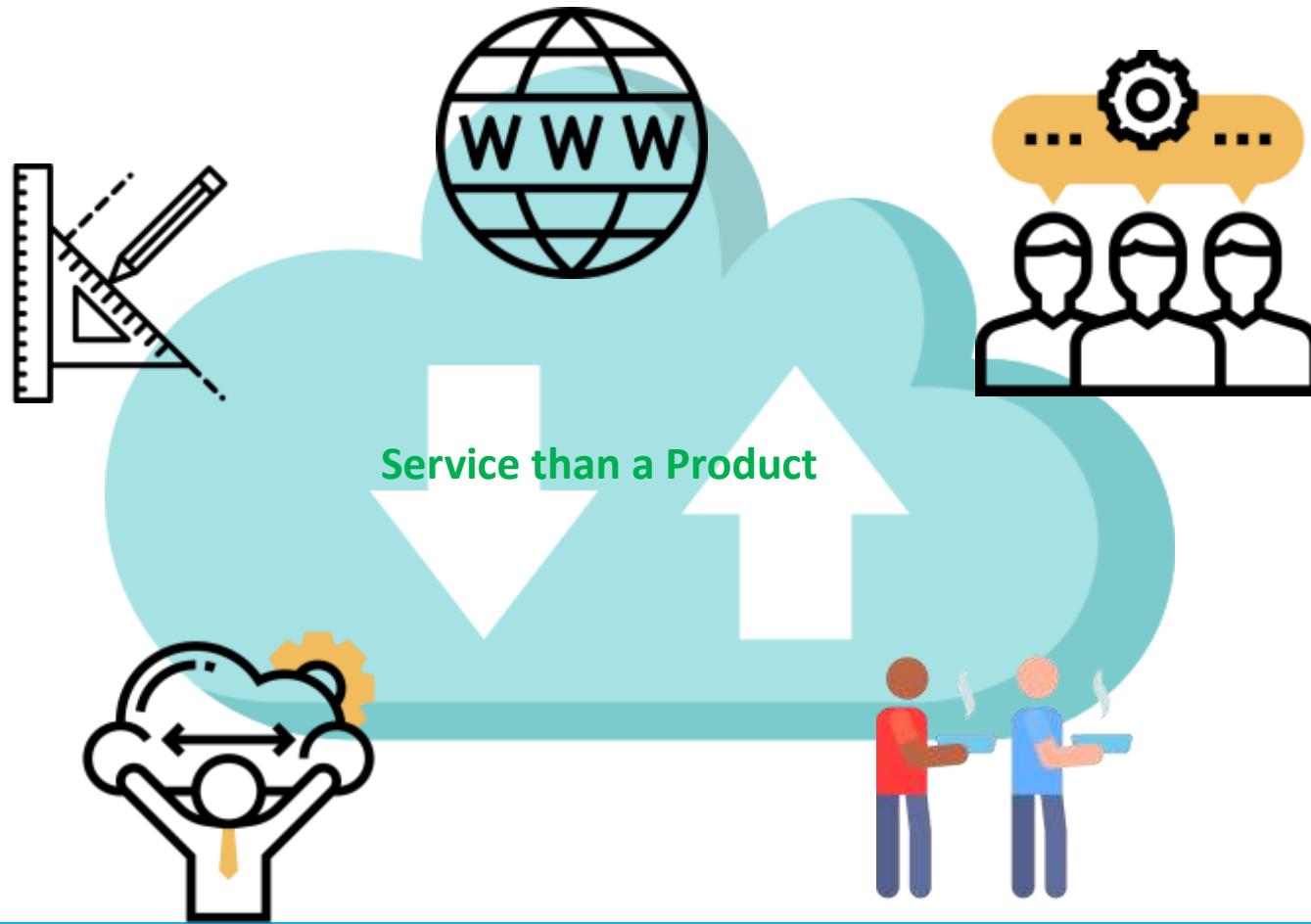


Storage

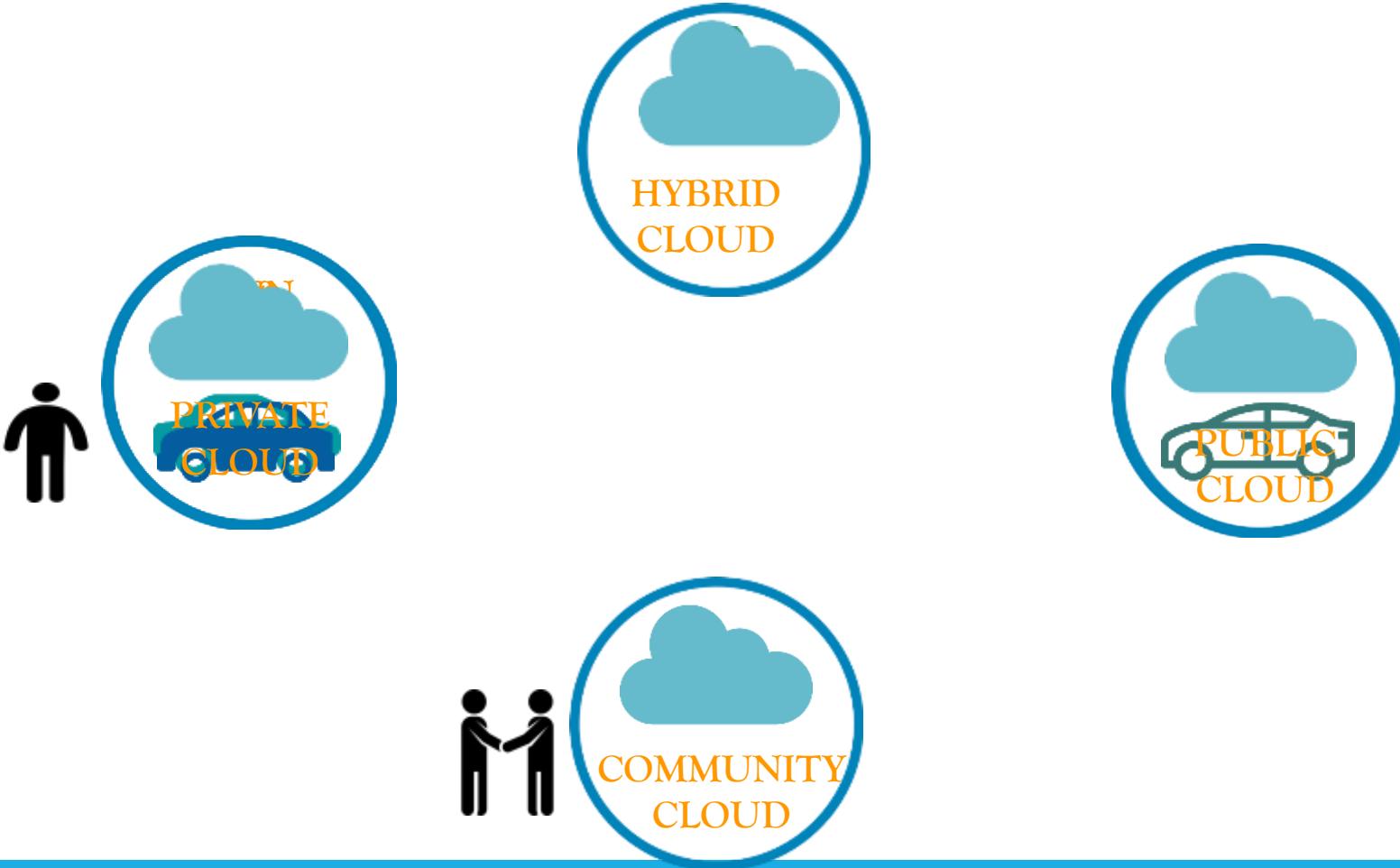
Traditional Datacenters



Characteristics of Cloud



Cloud Deployment Types



Service Models



Car Owned



Car Leased

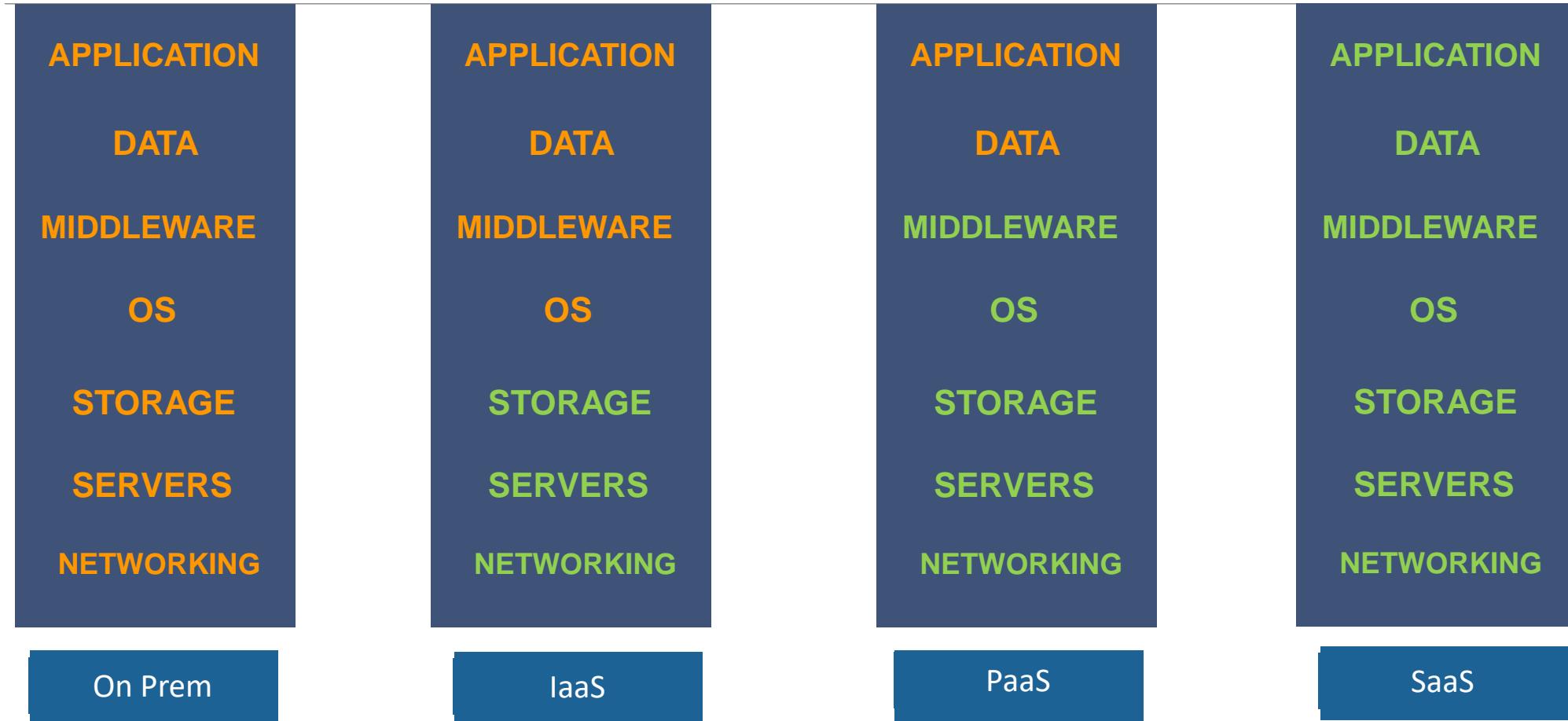


Car Hired

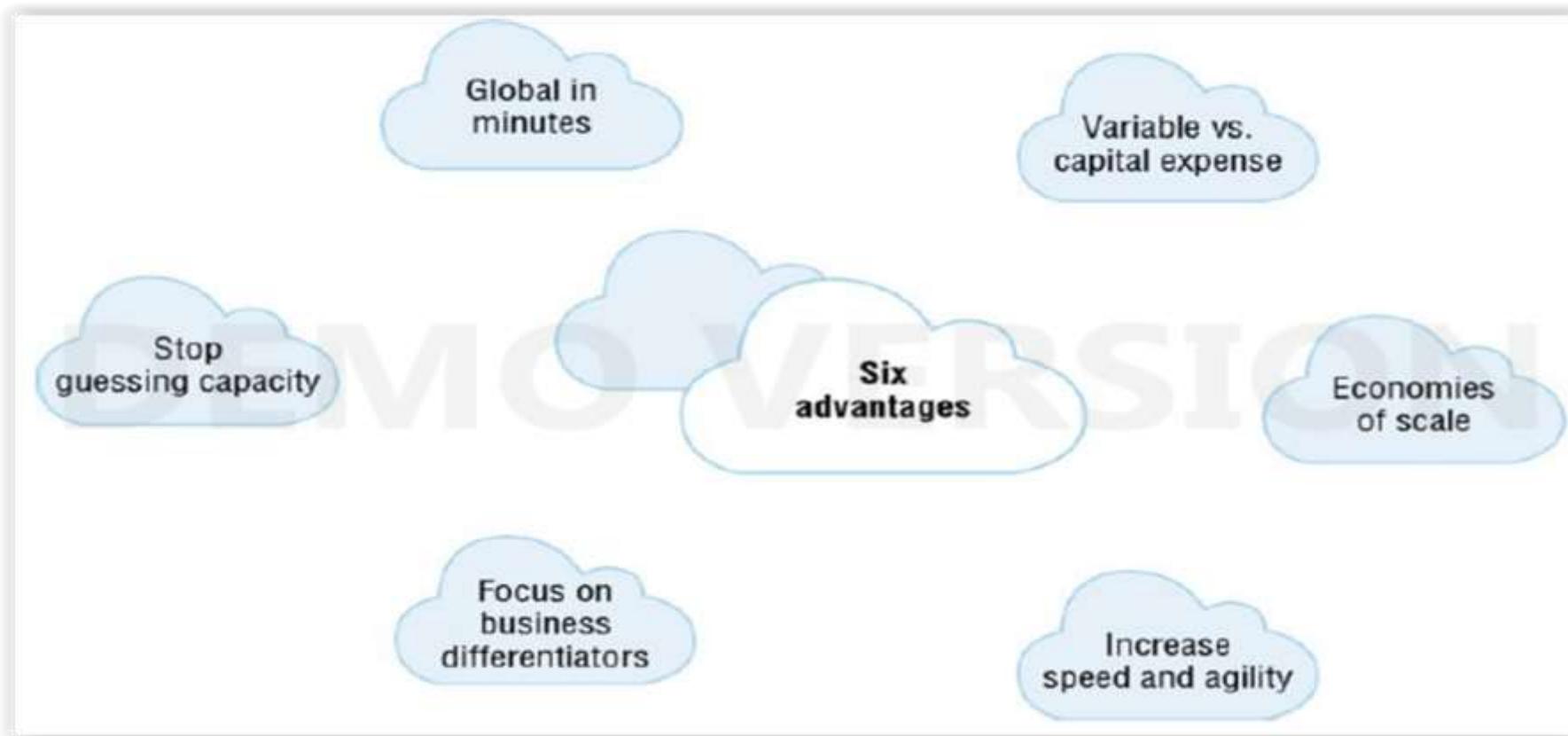


Taxi

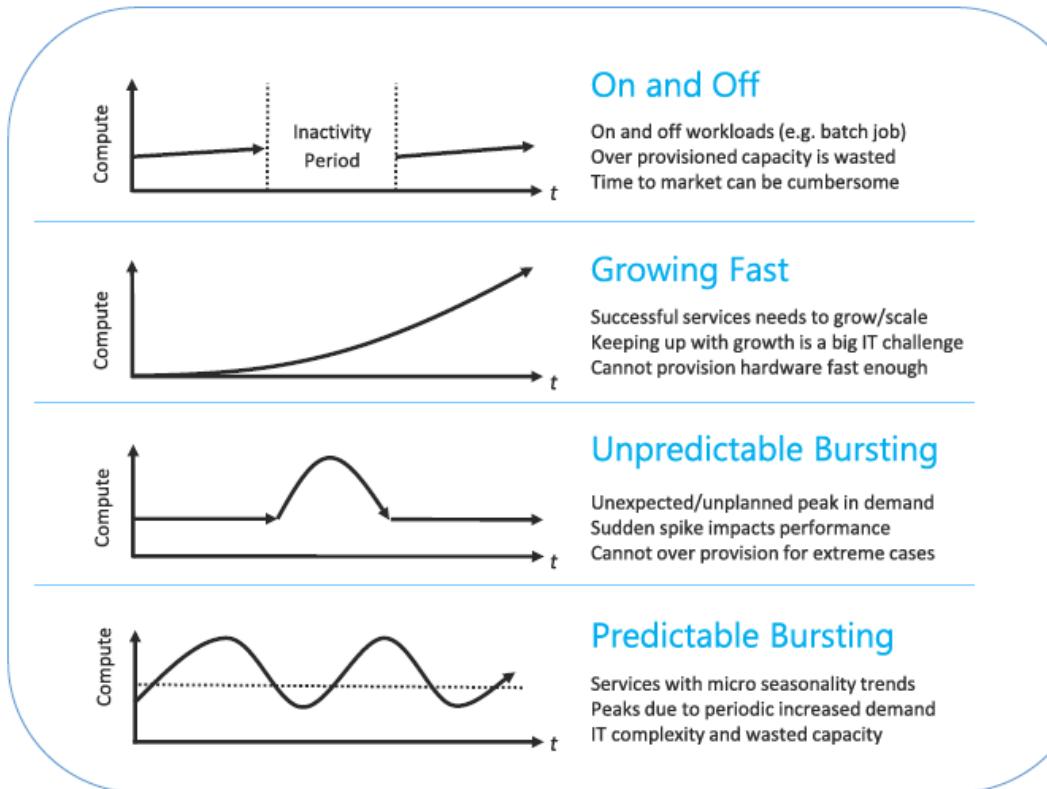
Service Models



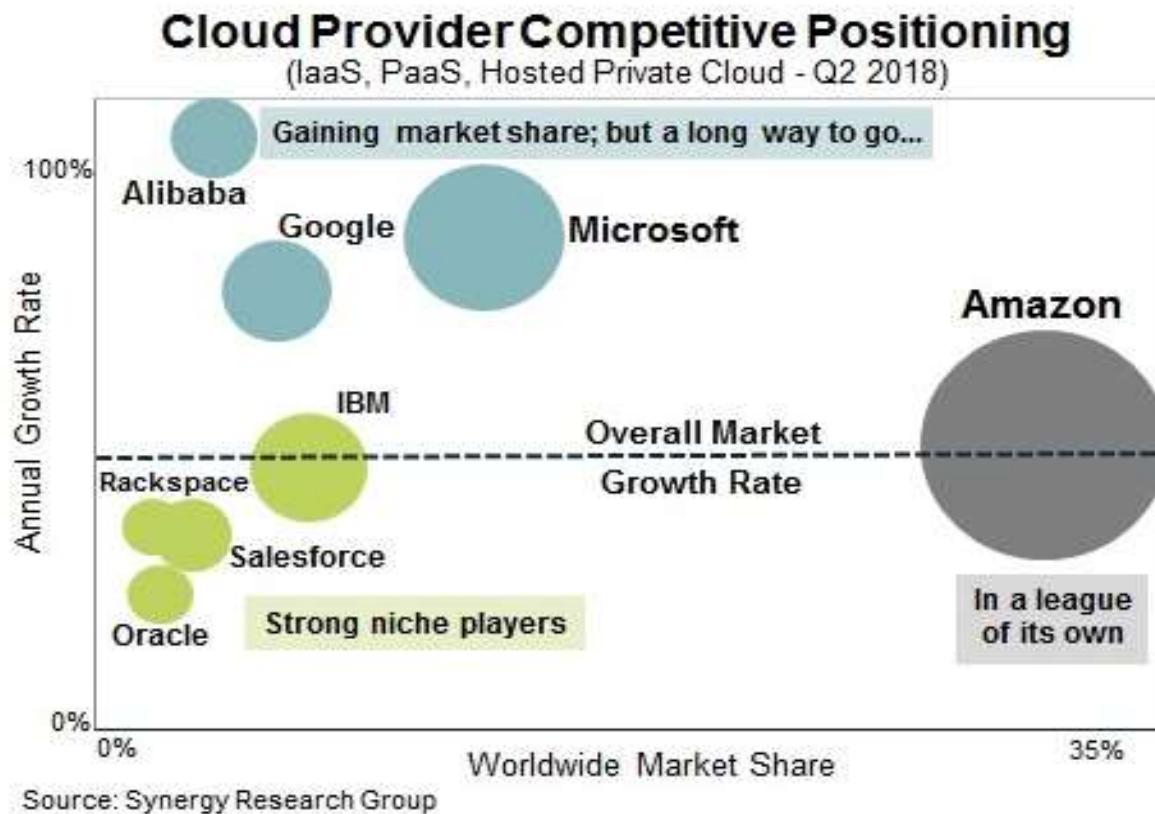
Cloud Benefits



Cloud Major Use Cases



Cloud Players



Amazon Web Services

AWS (Amazon Web Services) is a group of web services (also known as cloud services) being provided by Amazon since 2006.

AWS provides huge list of services starting from basic IT infrastructure like CPU, Storage as a service, to advance services like Database as a service, Serverless applications, IOT, Machine Learning services etc..

Hundreds of instances can be build and use in few minutes as and when required, which saves ample amount of hardware cost for any organizations and make them efficient to focus on their core business areas.

Currently AWS is present and providing cloud services in more than 190 countries.

Well-known for IaaS, but now growing fast in PaaS and SaaS.

Why AWS

Low Cost: AWS offers, pay as you go pricing. AWS models are usually cheapest among other service providers in the market.

Instant Elasticity: You need 1 server or 1000's of servers, AWS has a massive infrastructure at backend to serve almost any kind of infrastructure demands, with pay for what you use policy.

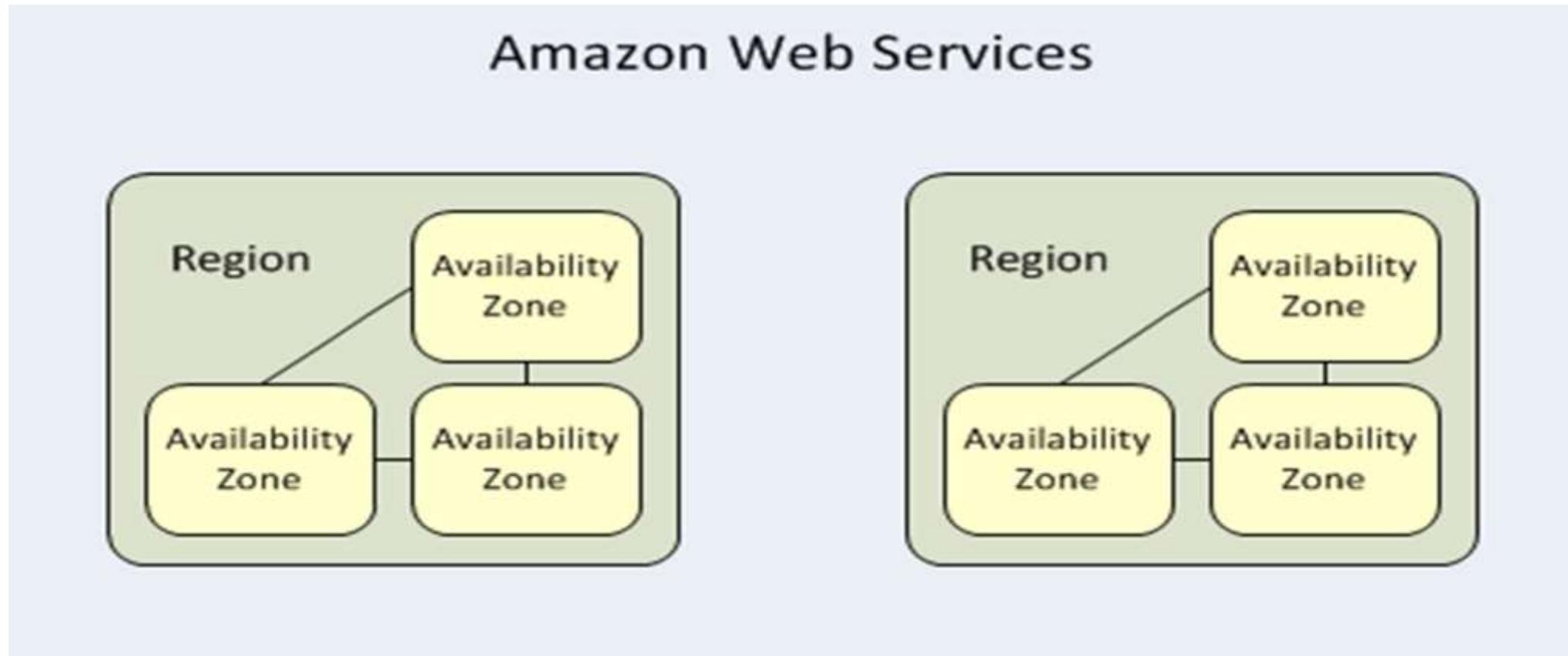
Scalability: Facing some resource issues, no problem within seconds you can scale up the resources and improve your application performance. This cannot be compared with traditional IT datacenters.

Multiple OS's: Choice and use any supported Operating systems.

Multiple Storage Options: Choice of high I/O storage, low cost storage. All is available in AWS, use and pay what you want to use with almost any scalability.

Secure: AWS is PCI DSS Level1, ISO 27001, FISMA Moderate, HIPAA, SAS 70 Type II passed. In-fact systems based on AWS are usually more secure than in-house IT infrastructure systems.

AWS Regions and Availability Zones



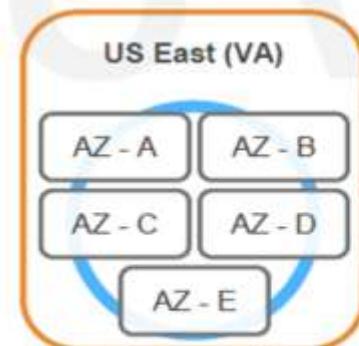
AWS Regions and Availability Zones

At least 2 AZs per region.

Examples:

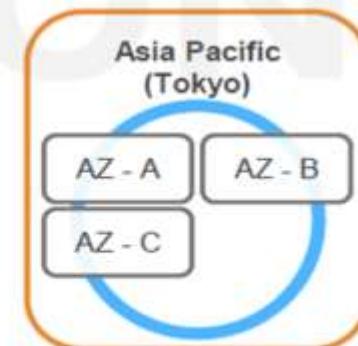
➤ US East (N. Virginia)

- us-east-1a
- us-east-1b
- us-east-1c
- us-east-1d
- us-east-1e



➤ Asia Pacific (Tokyo)

- ap-northeast-1a
- ap-northeast-1b
- ap-northeast-1c



Note: Conceptual drawing only. The number of Availability Zones (AZ) may vary.

AWS Regions and Availability Zones

AWS Regions:

- Geographic Locations
- Consists of at least two Availability Zones(AZs)
- All of the regions are completely independent of each other with separate Power Sources, Cooling and Internet connectivity.

AWS Availability Zones

- AZ is a distinct location within a region
- Each Availability Zone is isolated, but the Availability Zones in a Region are connected through low-latency links.
- Each Region has minimum two AZ's
- Most of the services/resources are replicated across AZs for HA/DR purpose.

AWS Regions and Availability Zones

Current:

22 AWS Regions

69 AZs

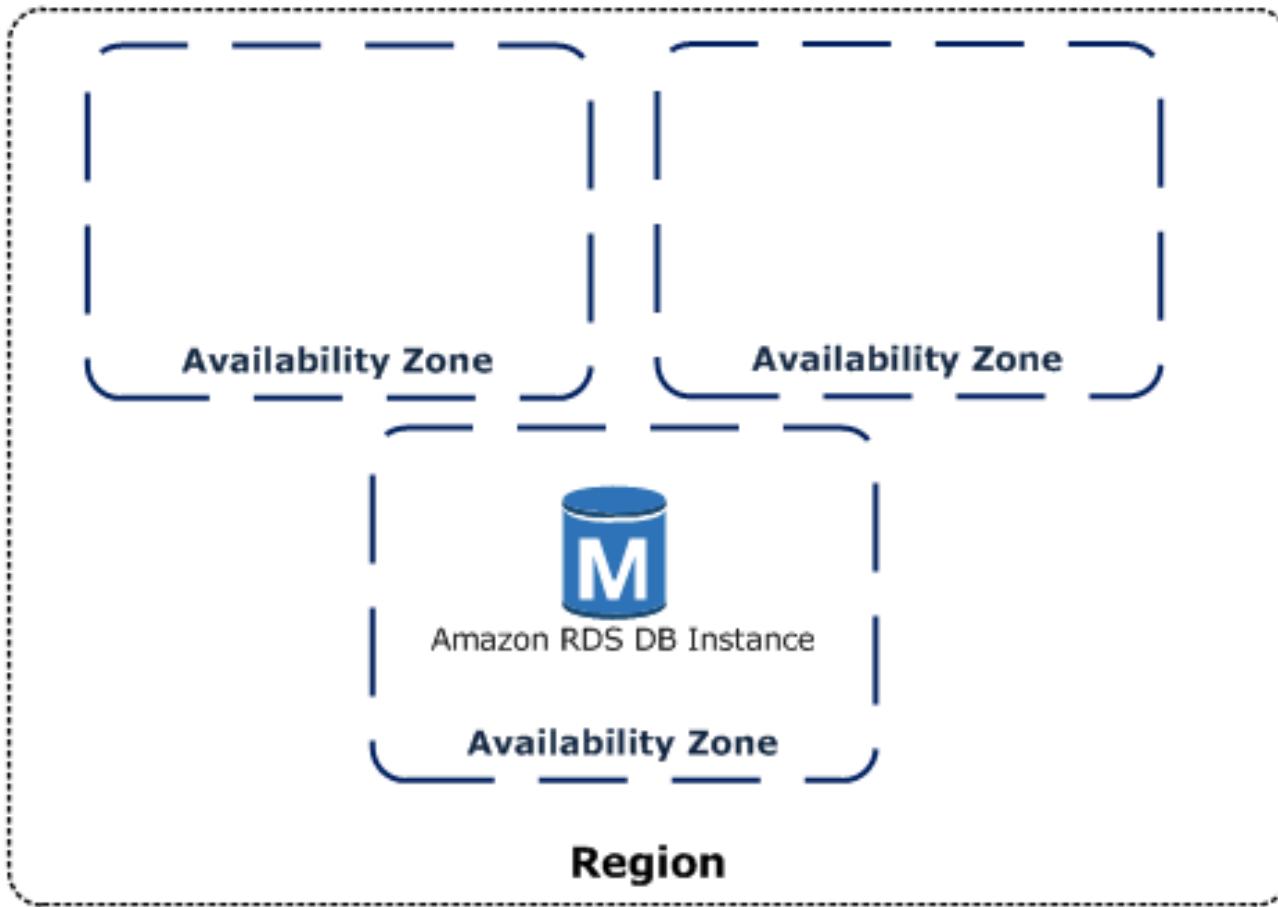
Upcoming:

4 Regions

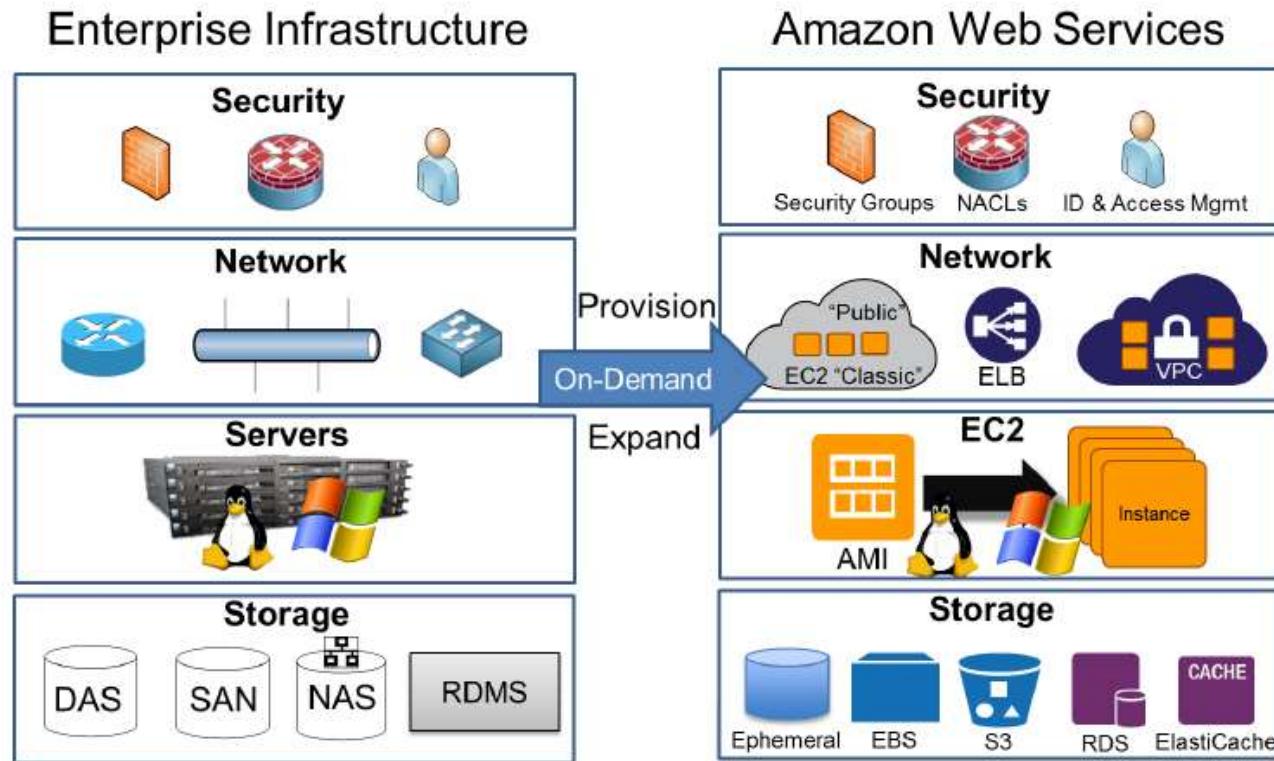
13 AZs



AWS Regions and Availability Zones



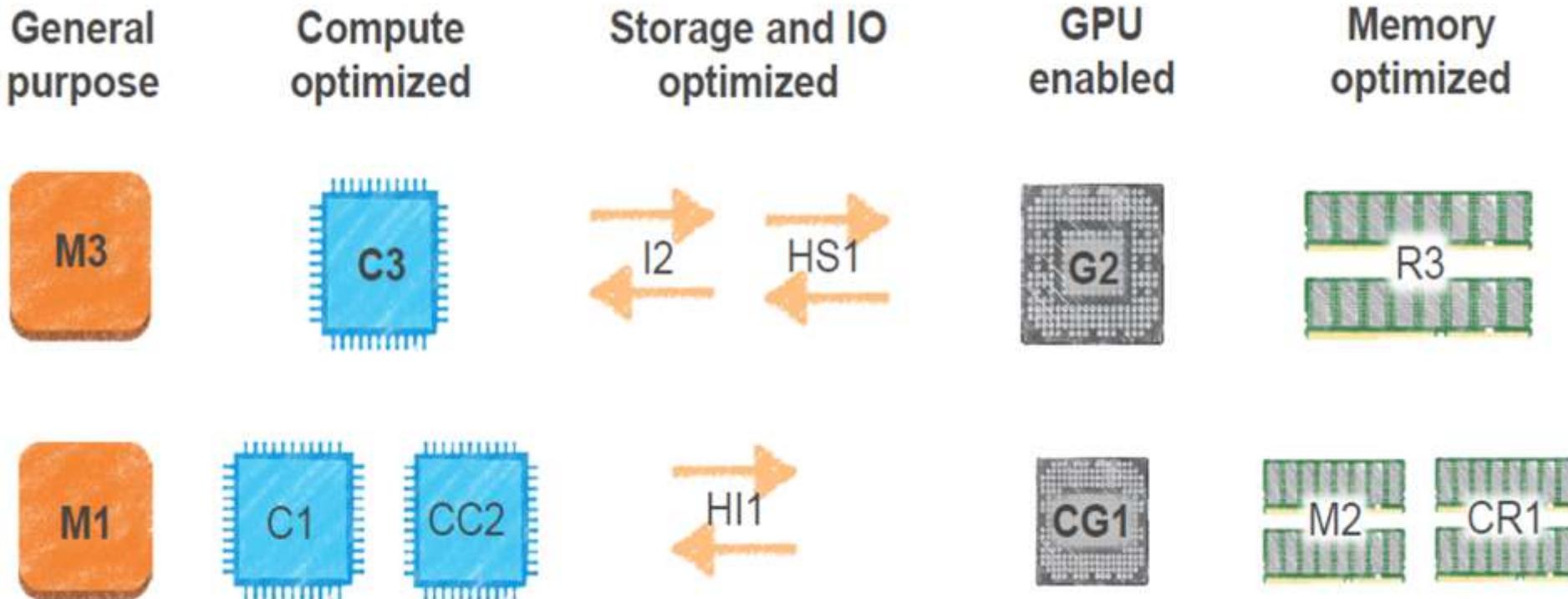
AWS



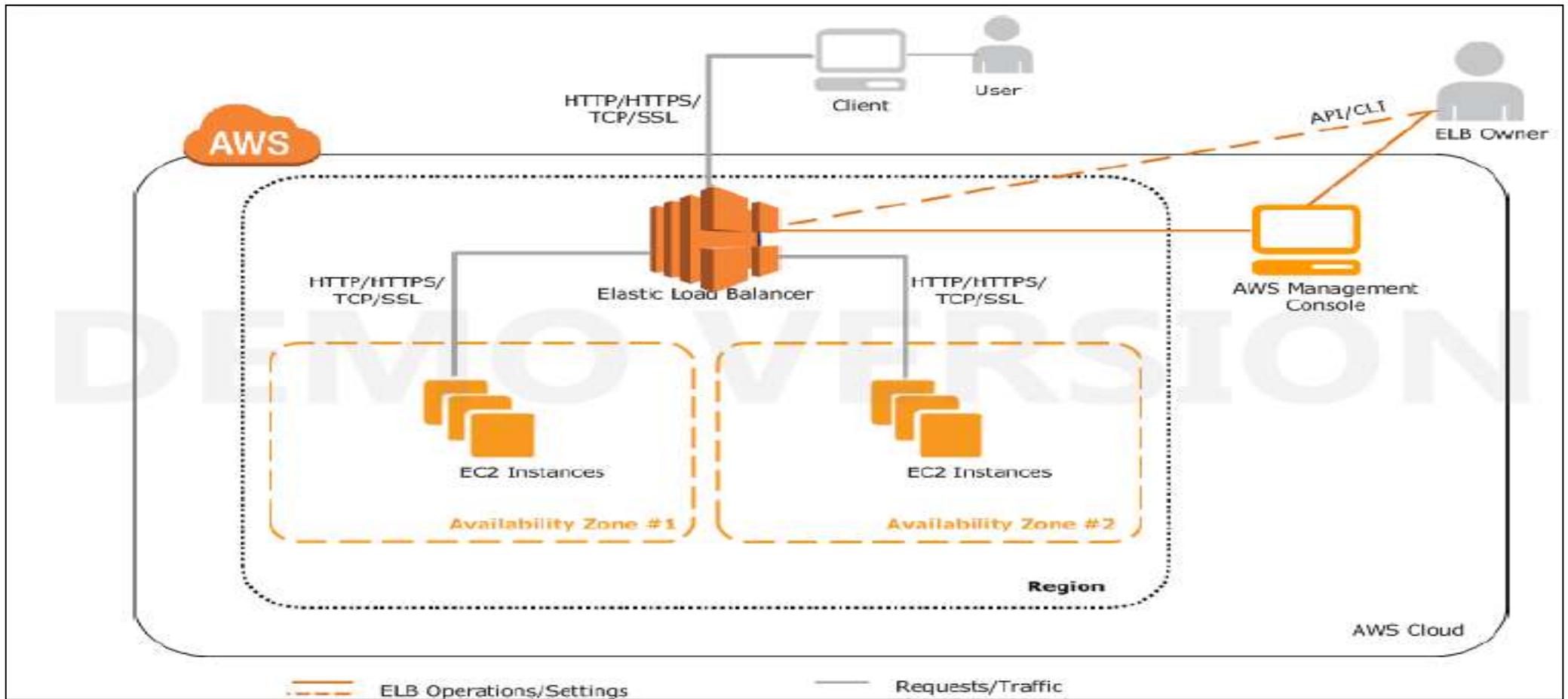
AWS Compute Services

- Amazon EC2 stands for Elastic Compute Cloud, and is the Primary AWS web service.
- Provides Resizable compute capacity
- Reduces the time required to obtain and boot new server instances to minutes
- There are two key concepts to Launch instances in AWS:
 - Instance Type
 - AMI
- EC2 Facts:
 - Scale capacity as your computing requirements change
 - Pay only for capacity that you actually use
 - Choose Linux or Windows OS as per need. You have to Manage the OS and Security of same.
 - Deploy across AWS Regions and Availability Zones for reliability/HA

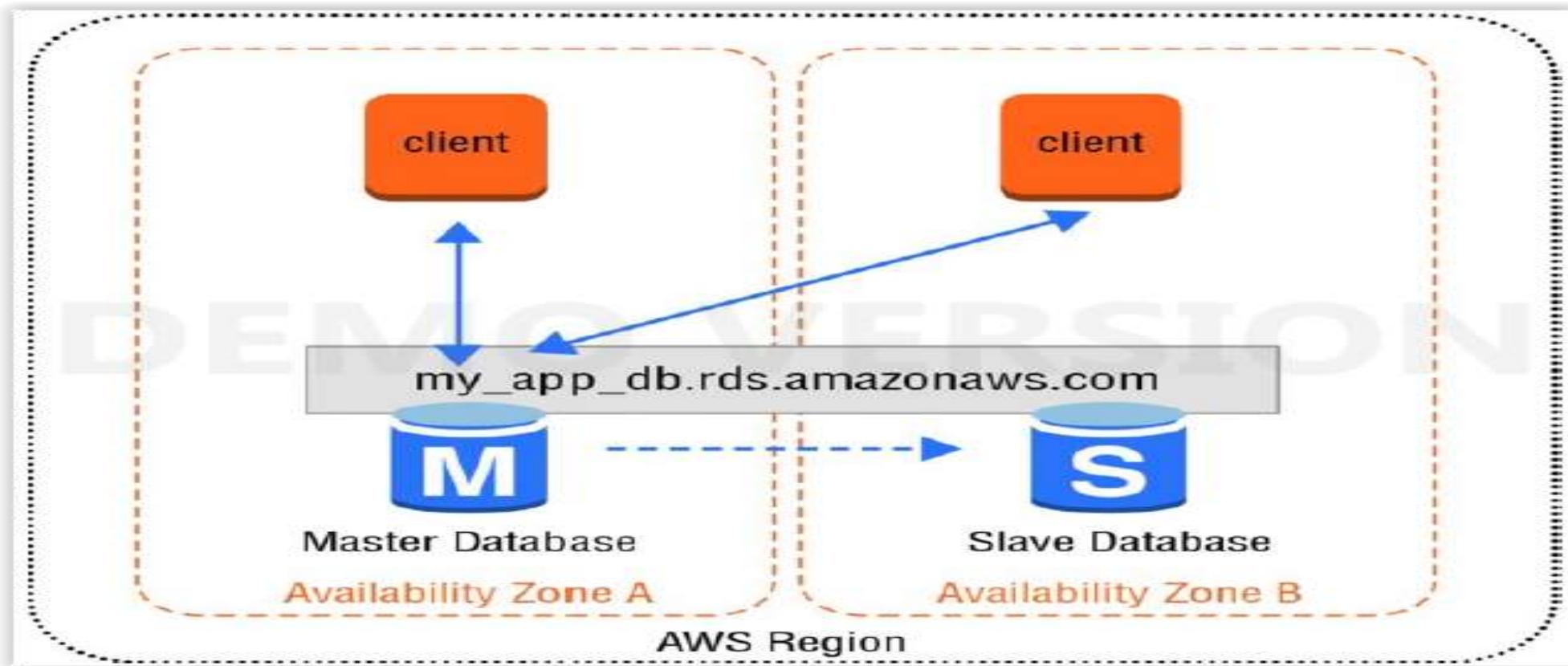
AWS EC2



ELB



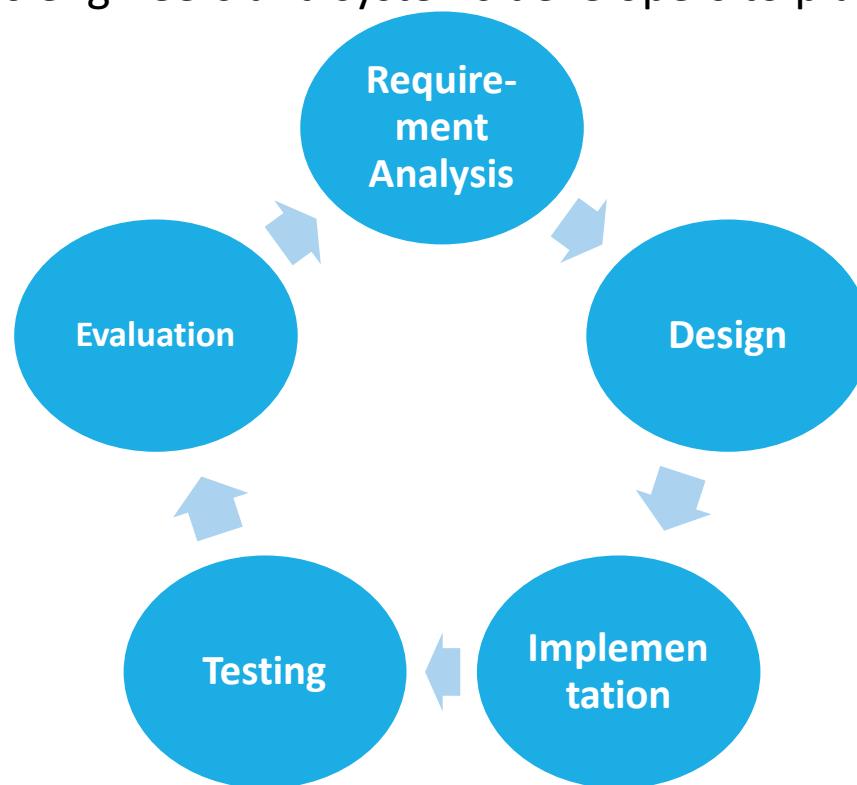
PAAS



What is DevOps

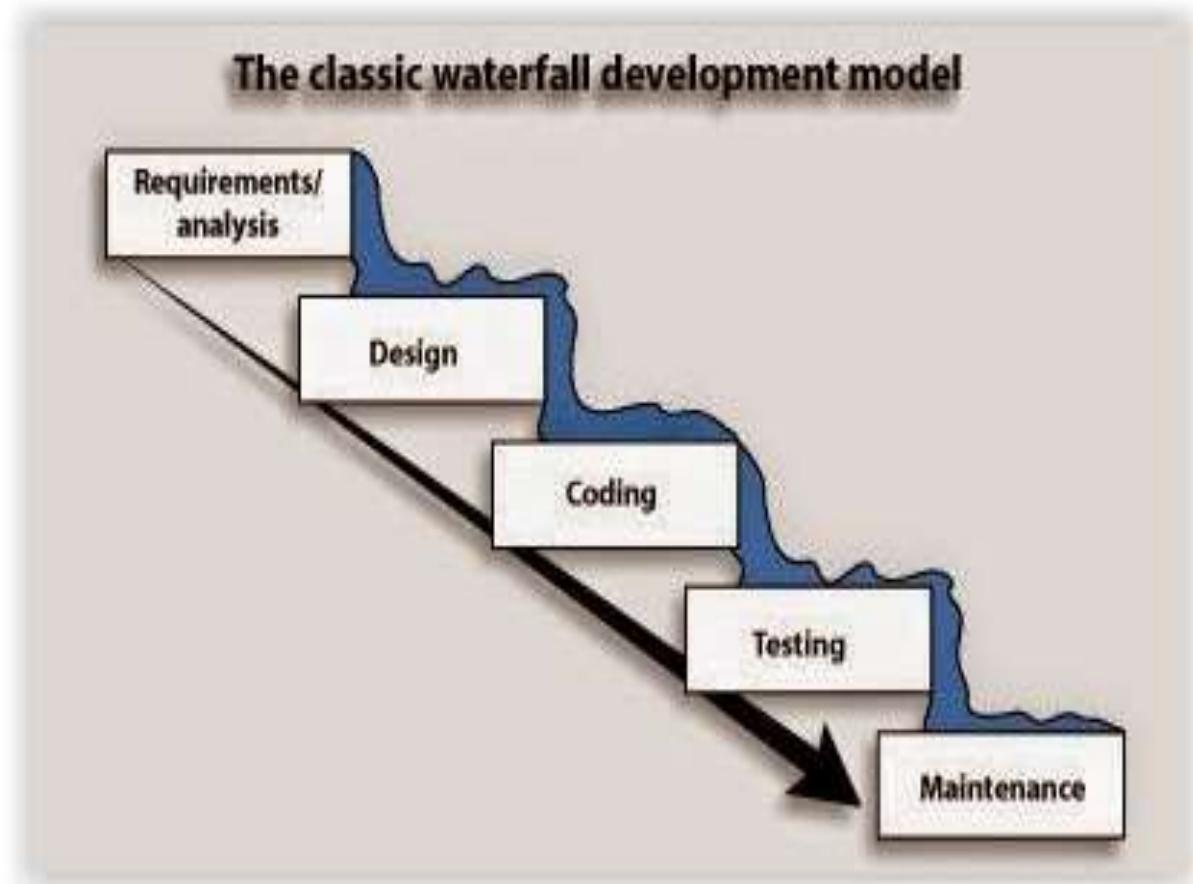
SDLC Model

- A systems development life cycle is composed of **several clearly defined and distinct work phases** which are used by systems engineers and systems developers to plan for, design, build, test, and deliver information systems



Waterfall Model

1. Determine the Requirements
2. Complete the design
3. Do the coding and testing (unit tests)
4. Perform other tests (functional tests, non-functional tests, Performance testing, bug fixes etc.)
5. At last deploy and maintain



Agile

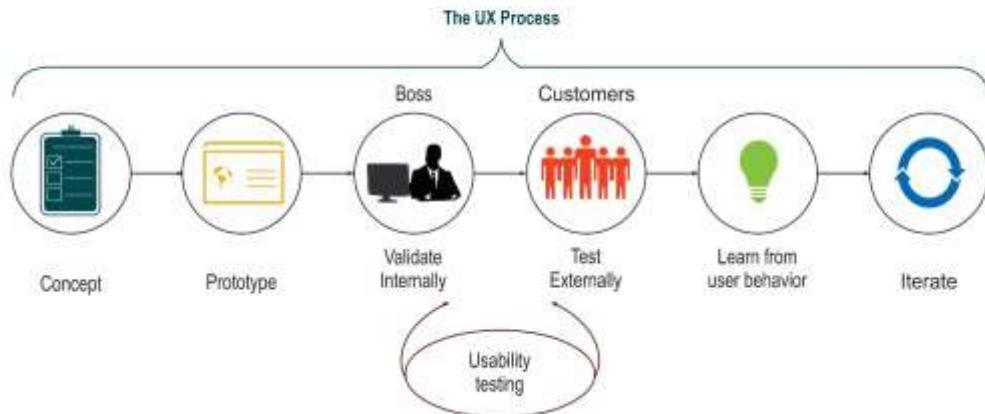
- Shorter release cycle
- Small batch sizes (MVP)
- Cross-functional teams
- Incredibly agile

Agile Methodology



Lean Development

Lean Development (LD)



Not like this...



...instead like this!



- Suddenly ops was the bottleneck (more release less people), again WIP is more!

Challenges

Some of the challenges with the traditional teams of Development and Operations are:



A Typical Case Study

Development Team:

- Monday Morning, the writing of code done, unit tests completed, code delivered to the Integration teams to get the code included in CI builds.
- To get the services tested, a ticket is opened for QA teams

Build/Release/Testing/Integration Team:

- Tuesday Morning, ticket accepted, a tester put an email to the developer asking deployment instructions. There is not automated deployments, developer updated to the tester, lets come online and we will deploy the services to the QA environment together.
- Call started, developer identified the “test environment” is not compatible.
- Tuesday afternoon, a ticket raised in Ops Team with new specifications.

Ops Team:

- Wednesday morning, ticket accepted, specifications checked , a new port open request was identified.
- Ticket raised for Security team, ticket accepted, change approved, port opened, email received by the Ops team the work is done.

A Typical Case Study

Ops Team:

- Identified the provisioning requirements again and started work on building the environment.

Build/Release/Testing/Integration Team:

- Thursday Morning, updates received - the environment is ready. Developer and Tester again on call to deploy new services. Services deployed; tester is running test scripts. Next phase is to run regression test cases. Again a new ticket is raised for new test data with production teams and day ends.

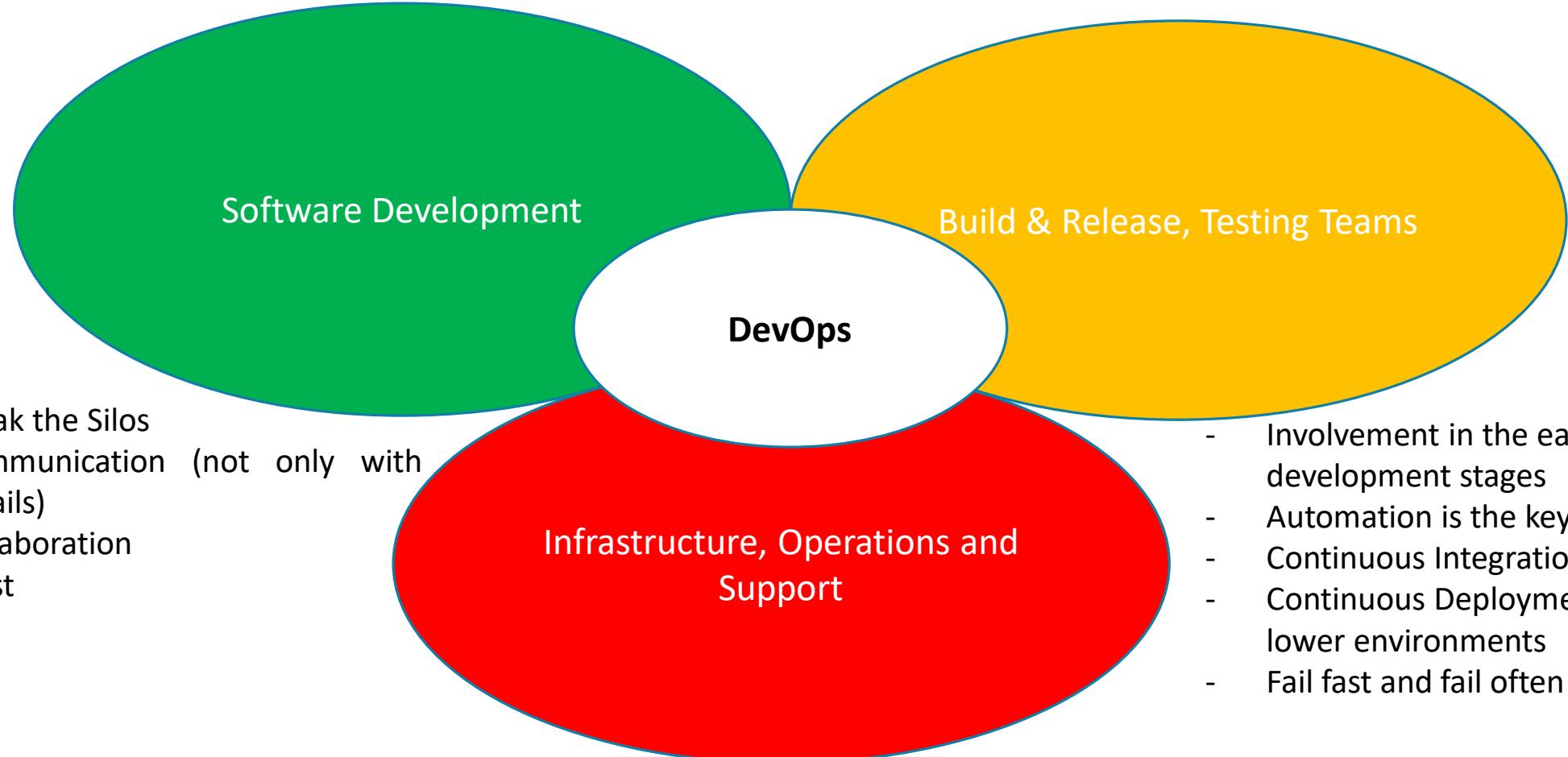
Ops Team:

- Its Friday and the work is not on full swing, ticket accepted but not worked as production team has to complete rest of the works. Somehow the test data is gathered by Friday Evening.

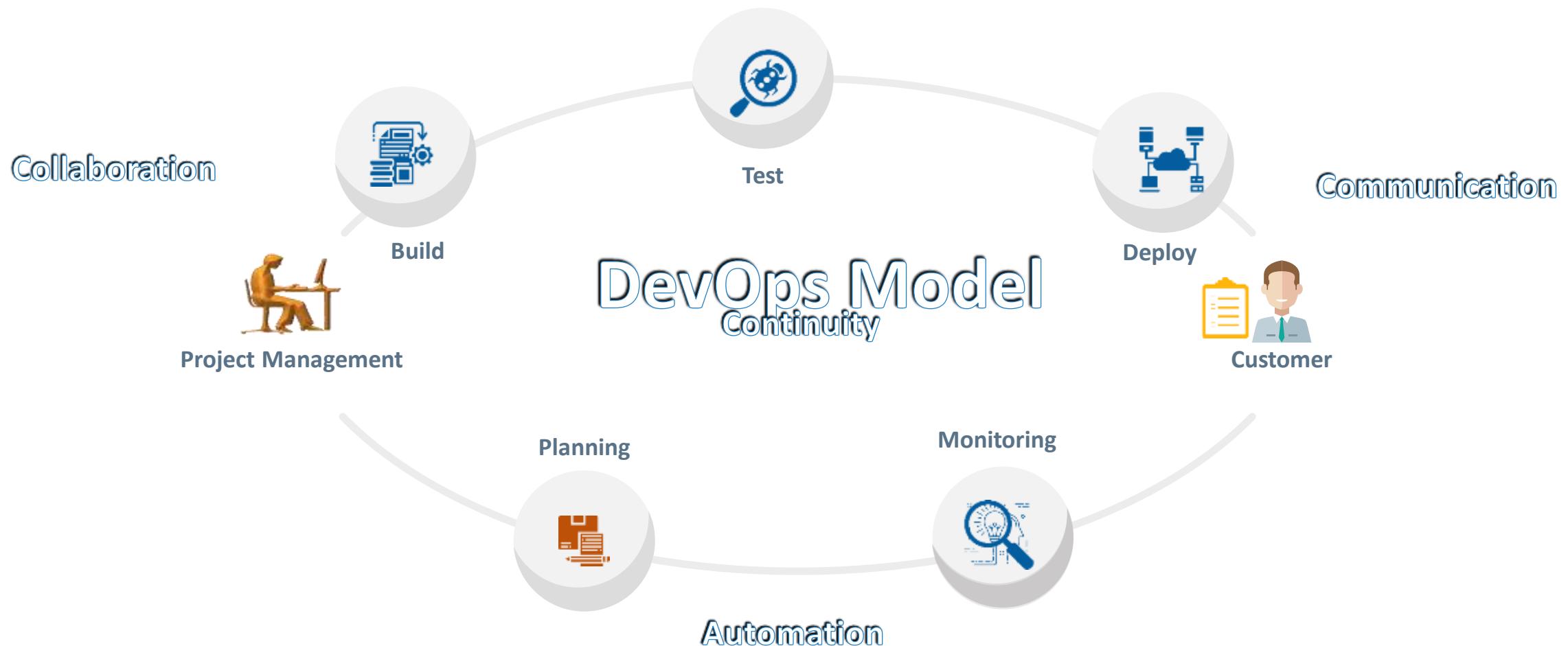
Build/Release/Testing/Integration Team:

- Monday morning, tester gets the data, regression tests run, a defect found, and ticket returned to the development team.

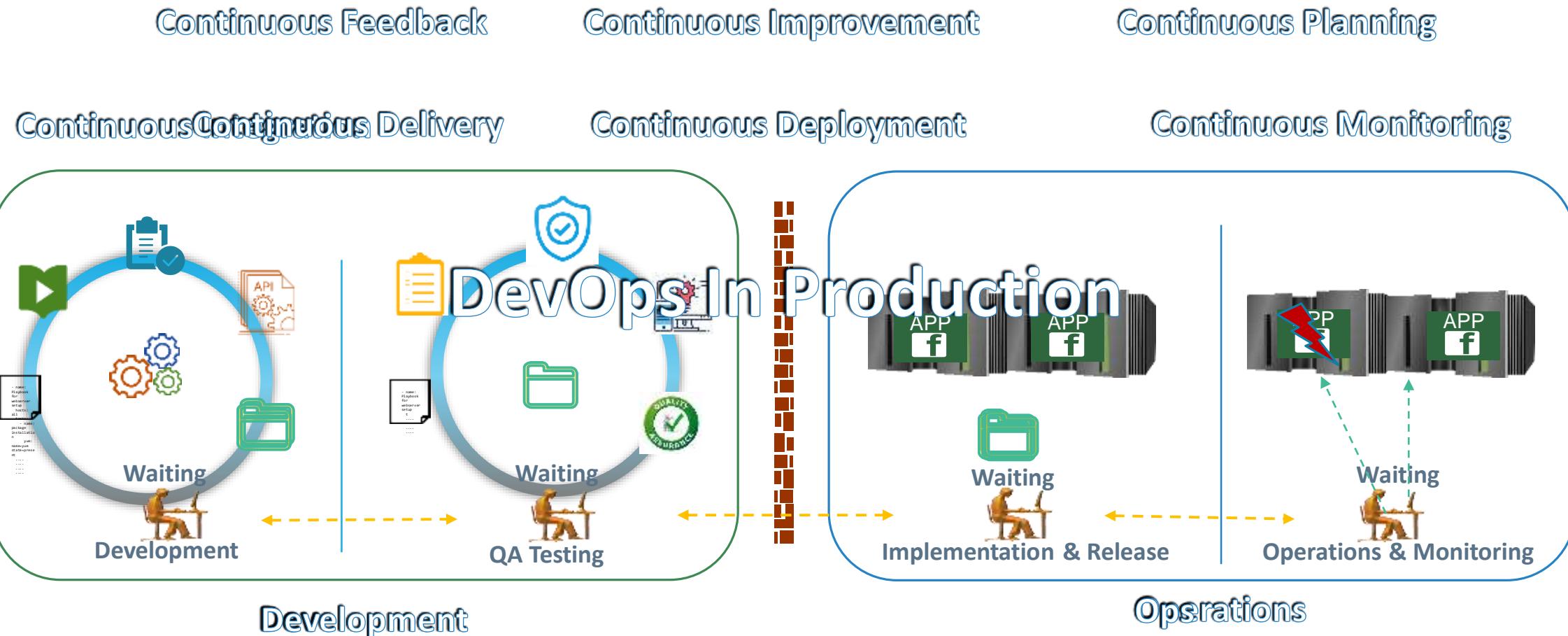
DevOps



DevOps

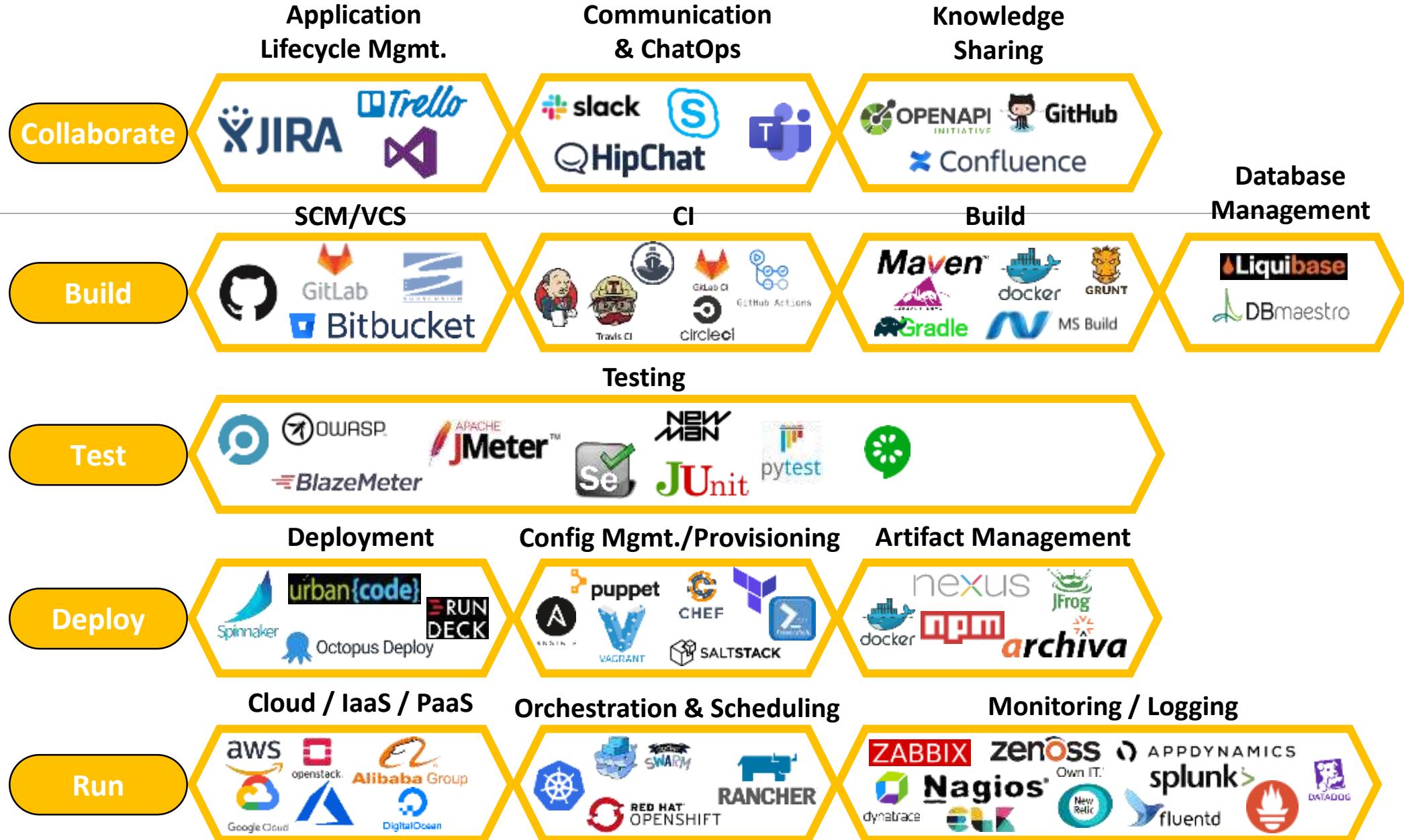


DevOps in Action



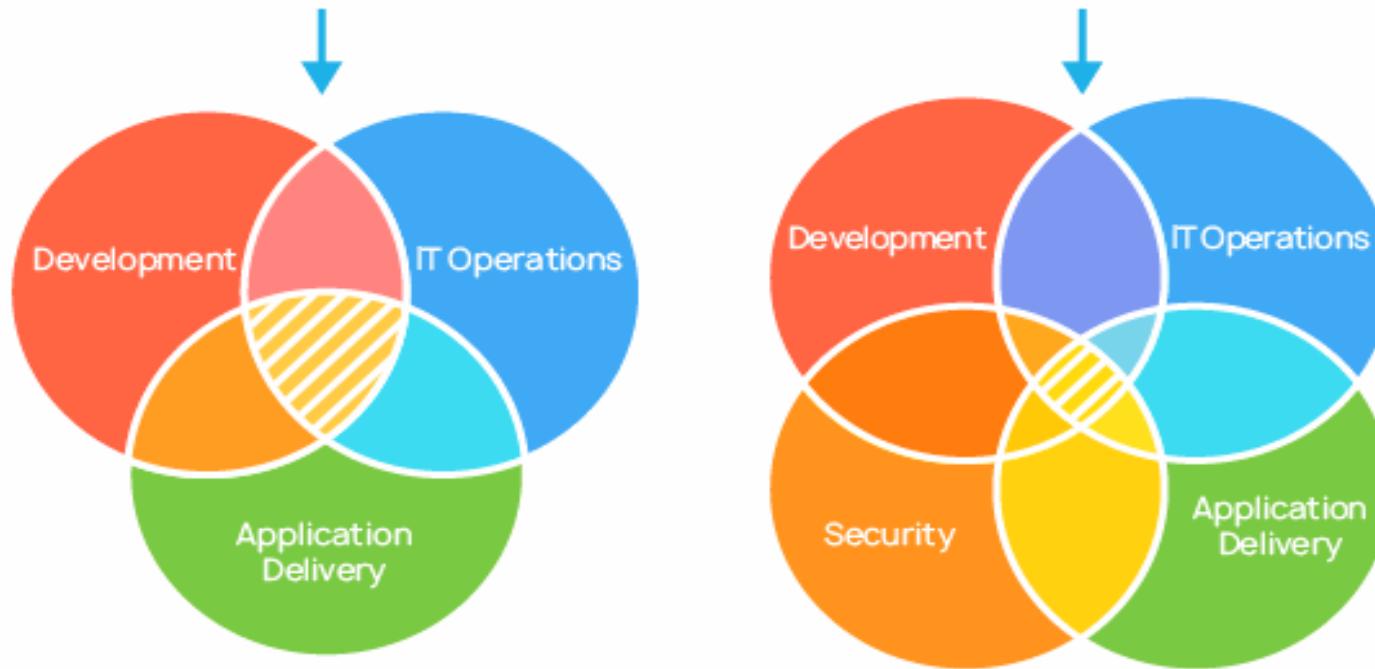
Toolsets

DevOps



DevOps vs DevSecOps

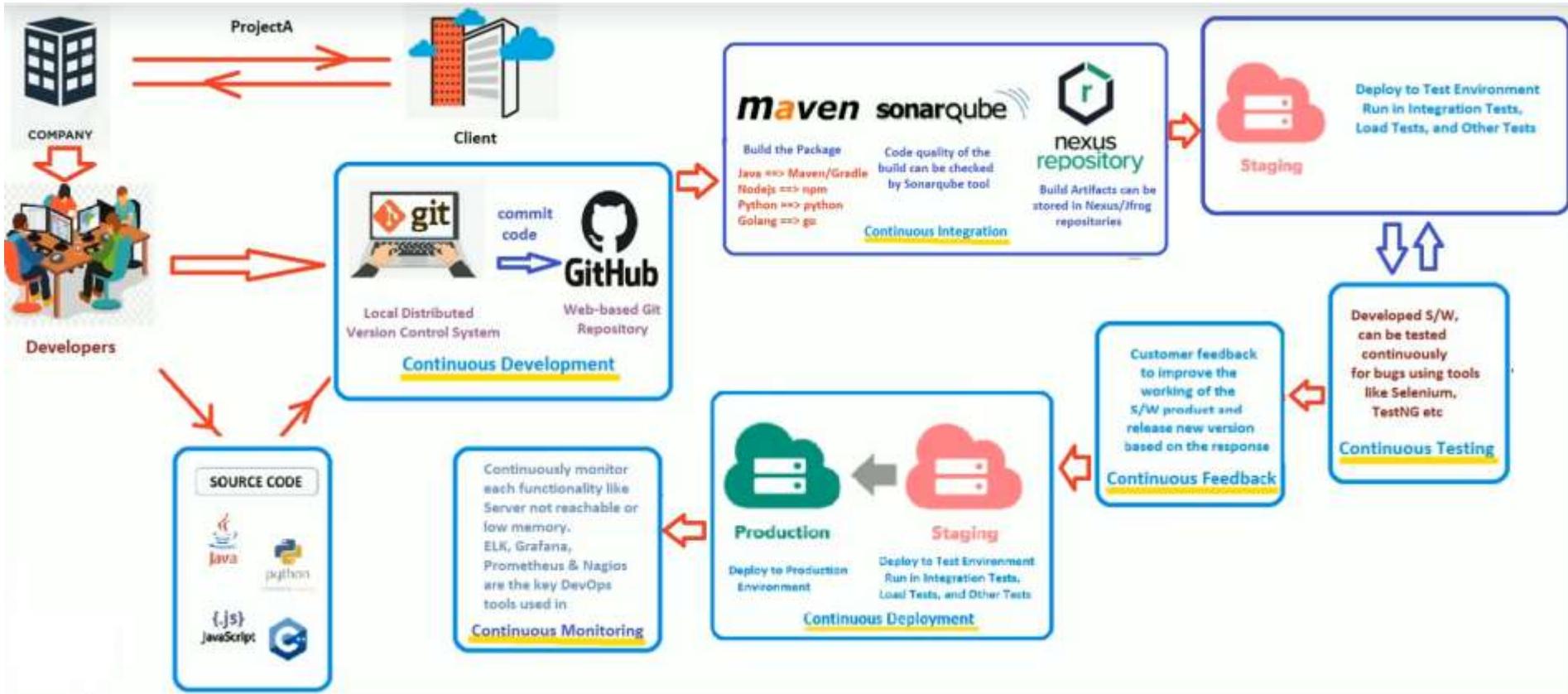
DevOps VS DevSecOps



What is DevOps?

- It is combining software development and IT operations with the aim of shortening software release cycles
- In reality it is all about automating as much as you can (e.g. infrastructure via pipelines). This enables a process for the software to be quickly released. This can involve:
 - Creating/maintaining CI/CD pipelines for development to leverage to release their code
 - Creating/maintaining environments/infrastructure so SDLC can develop/test etc
 - Managing various tech stacks such a version control (Git, SVN), CI/CD (Jenkins, GitLab, GitHub, Azure DevOps), infrastructure (Docker, VMs, Vagrant, Terraform), Cloud (Azure, AWS, GCP) and configuration management (Ansible, Chef)

DevOps Lifecycle



Application Security

According to a report by Web Application Security Statistics, fixing any vulnerability can take **146 days** on average. Are you willing to wait that long?

Maybe you are, but the hacker will not wait. So in the meantime, you can apply some initial security measures on your application to prevent hackers from exploiting any weakness.

DevSecOps

- It is essentially adding security testing to DevOps workflows
- DevSecOps uses Shift Left concept means moving the security testing further “left” as possible
- Identify issues as early as possible
- Cost to find an issue identified in development is ten times cheaper than finding in testing and 100 times cheaper than finding in production.
- So earlier we identify the issue, the cheaper it is.



DevSecOps

We can aid shift left via the implementation of:

- Creation of dedicated security pipelines that development with commit into, this could include:
 - SAST (static application security testing)
 - DAST (dynamic application security testing)
 - SCA (Software composition analysis)
- Ensuring best practices are followed and issues are prioritised and understood
- Push DevSecOps mindset – training development/ product staff in secure design/ principles/ how to develop securely

Traditional Security Tools

Traditional security tools are software programs designed to identify and mitigate security risk within organisation's IT infrastructure.

These tools are often used by dedicated security teams and focus on securing the network servers and endpoints.

For Example:

Firewalls

Antivirus software

Intrusion detection systems

Vulnerability scanners

Limitations of Traditional Security Tools

They tend to be reactive rather than proactive, they are assigned to detect and respond to security threats after they have occurred rather than preventing them from happening in the first place.

They can create silos between different teams. Security teams are often seen as separate from development and operation teams which can lead to a lack of collaboration and communication.

Advantages of DevSecOps

They allow security to be incorporated into every aspect of the software development lifecycle. This means that security becomes the responsibility of everyone involved in the process not just a specialized team.

They encourage collaboration and communication between different teams within an organization. It is a shared responsibility rather than a sole responsibility.

Integration vs Separation

Traditional security tools are often seen as separate from the development process. They are used by dedicated security teams and are focused on securing the network, servers, and endpoints. [DevSecOps tools](#), on the other hand, are integrated into every aspect of the software development lifecycle. They are used by developers, operations teams, and security teams, and focus on securing the application itself.

Automation vs Manual

Traditional security tools often require manual processes, such as running scans or conducting audits. This can be time-consuming and can slow down the development process.

DevSecOps tools, however, automate many of these processes, making it faster and easier for developers to identify and fix security issues.

Proactive vs Reactive

As mentioned earlier, traditional security tools tend to be reactive, meaning they detect and respond to security threats after they have occurred. DevSecOps tools, on the other hand, are proactive, allowing developers to catch and fix security issues before they become a problem.

Key Principles of DevSecOps

In order to begin implementing DevSecOps at your organisation you need six key principles to follow.

- deliver small, frequent releases using agile methodologies
- wherever possible, make use of automated testing
- empower developers to influence security changes
- ensure you are in a continuous state of compliance
- be prepared for threats, always invest in advanced training for your engineers

Security as a Code

- Security as Code is the methodology of codifying security and policy decisions and socializing them with other teams.
- Security testing and scans are implemented into your CI/CD pipeline to automatically and continuously detect vulnerabilities and security bugs.
- Access policy decisions are codified into source code allowing everyone across the organization to see exactly who has access to what resources.
- Adopting Security as Code tightly couples application development with security management, while simultaneously allowing your developers to focus on core features and functionality, and simplifying configuration and authorization management for security teams.
- This improves collaboration between Development and Security teams and helps nurture a culture of security across the organization.

Implementing Security as a Code

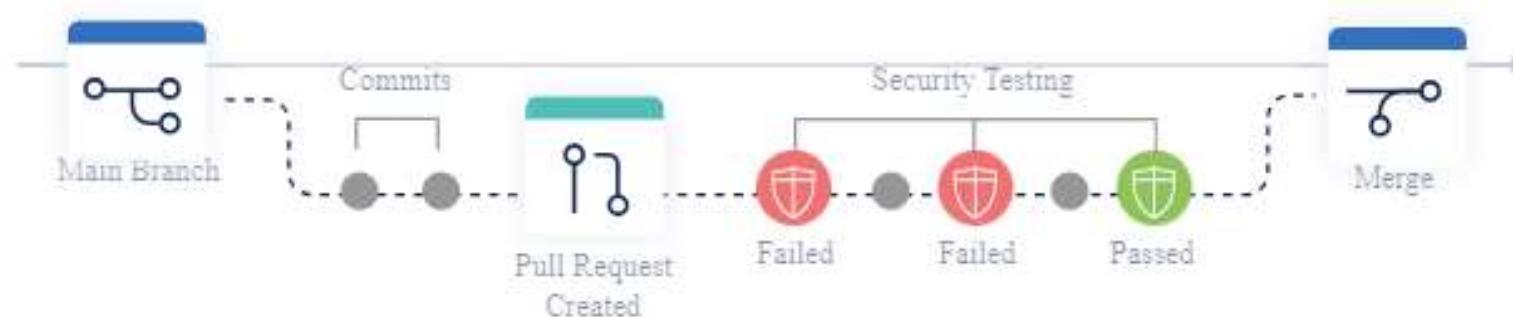
Security as Code generally comes in three different forms:

- security testing
- vulnerability scanning
- access policies

Implementing Security as a Code

1. Security testing

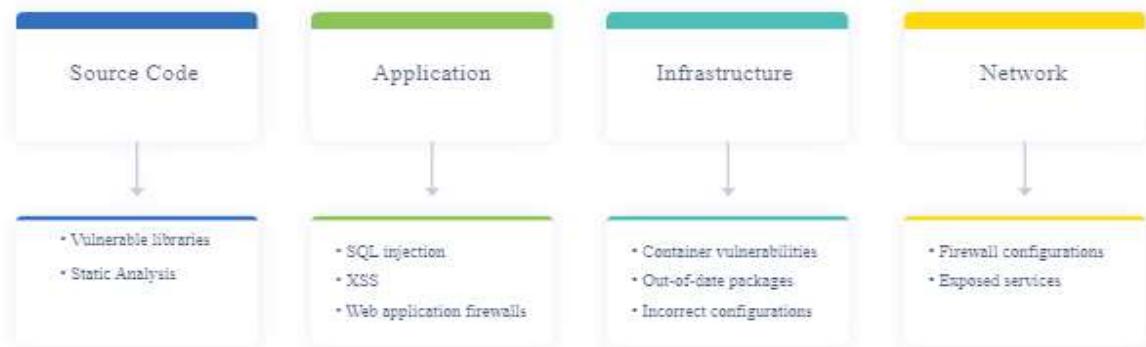
- It expands on best in class coding practices to add to the standard suite of tests to not only include functional and integration testing but also security focused testing.
- Static analysis for security vulnerabilities can be implemented on each commit or pull request.
- Permission boundaries can be checked to verify they cannot be crossed.
- APIs can be tested to ensure they're meeting authentication and authorization requirements.
- Security testing meets your developers where they already are, providing them immediate feedback on each and every commit.



Implementing Security as a Code

2. Vulnerability scanning

- It is applied at every level of your architecture across your pipeline can verify that each section of your application and deployment is secured against known vulnerabilities.
- Source code can be scanned for vulnerable libraries.
- For example, applications can be scanned for susceptibility to XSS and SQL injection.
- Containers can be scanned for vulnerabilities in individual packages and for adherence to best in class practices. Full scanning of test, staging and production environments can be done continuously and automatically.
- Scan early and scan continuously to verify your expected security controls are in place and so that you can find issues sooner rather than later.

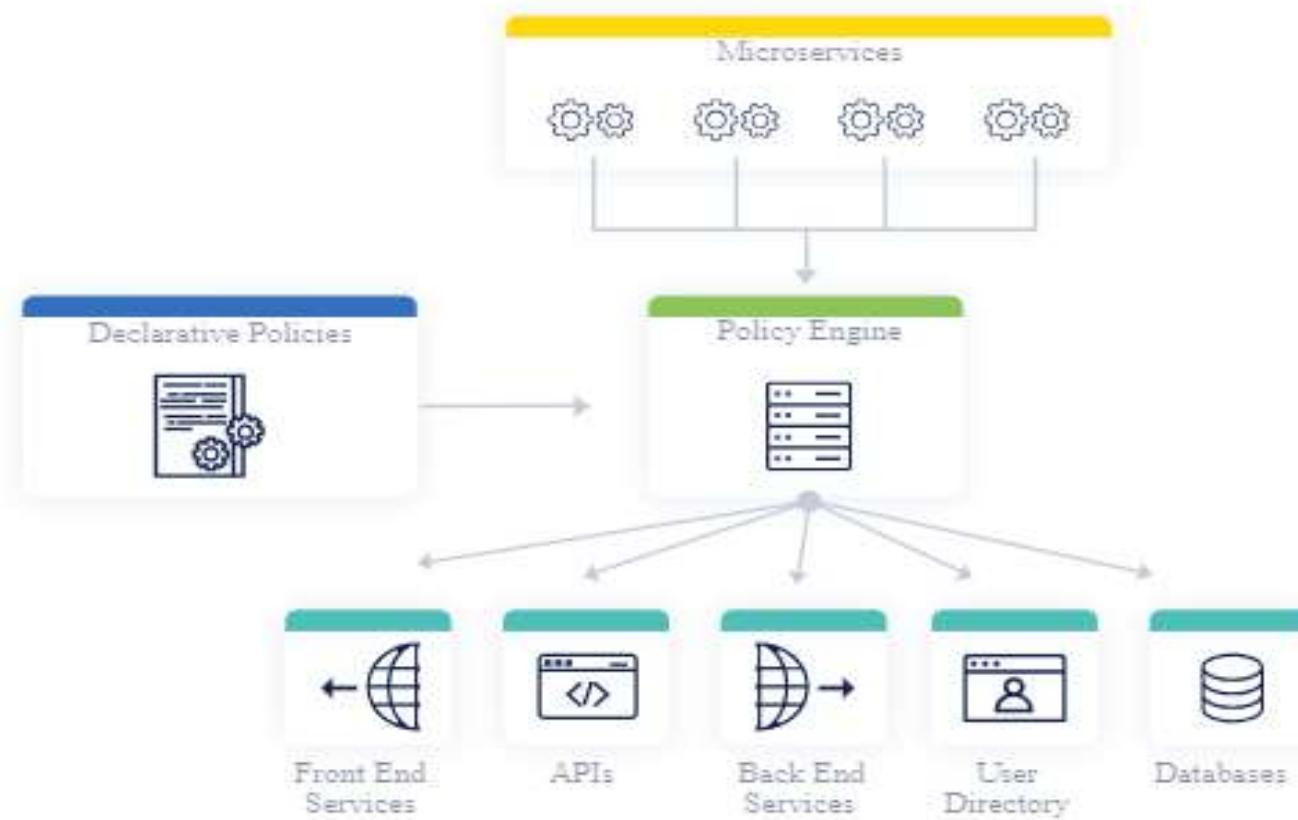


Implementing Security as a Code

3. User and data access policies:

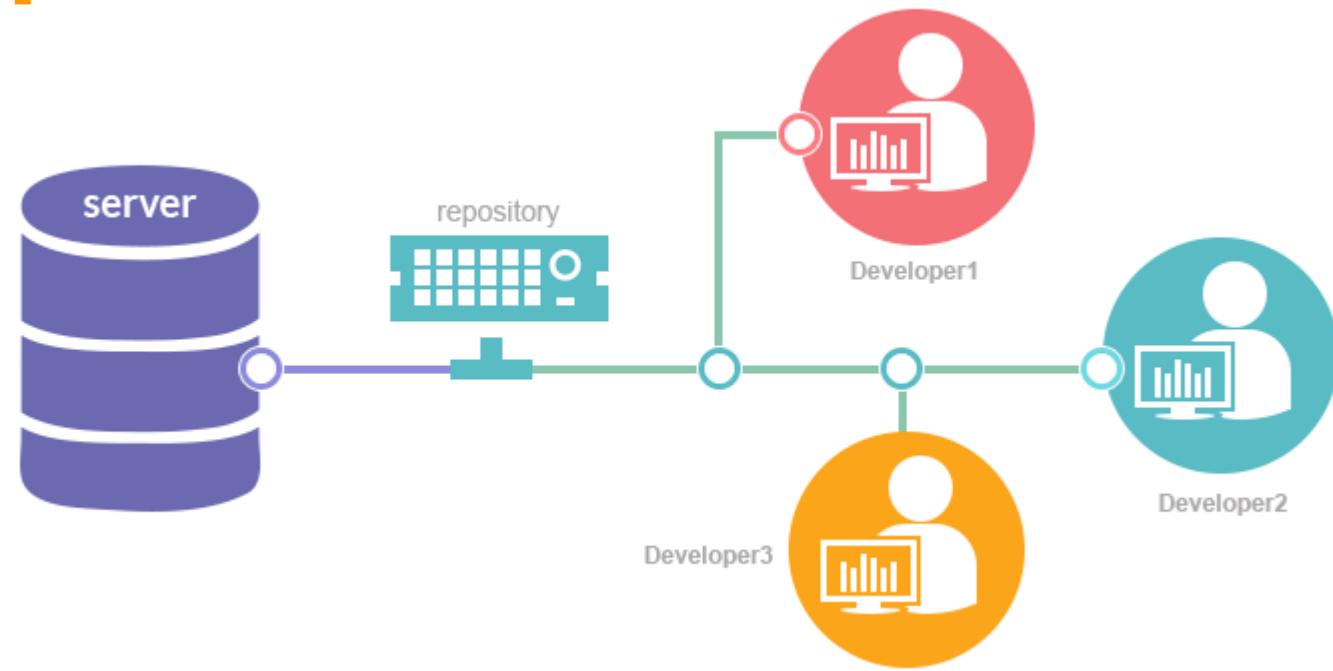
- It codify governance decisions that can then be reviewed by anyone in your organization.
- These policies can be standardized, reducing the toil necessary to constantly monitor and maintain one off requests.
- Authorization can be offloaded to external libraries allowing your Dev teams to focus on core features.
- Security teams now have a central repository to work directly with developers to monitor and review authorization, allowing the entire company to move faster without breaking core security and compliance requirements.

Implementing Security as a Code



Version Control Systems with GIT

What is The Need of Version Control System



What is Version Control System

- Version control (or revision control) is a system that records changes to a file or a group of files and directories over time, so that you can review or go back to specific versions later.
- Quite literally, version control means maintaining versions of your work.
- You may like to think of version control as a tool that takes snapshots of your work across time, creating checkpoints. You can return to those checkpoints any time you want.
- Not only are the changes recorded in these checkpoints, but also information about who made the changes, when they made them, and the reasons behind the changes

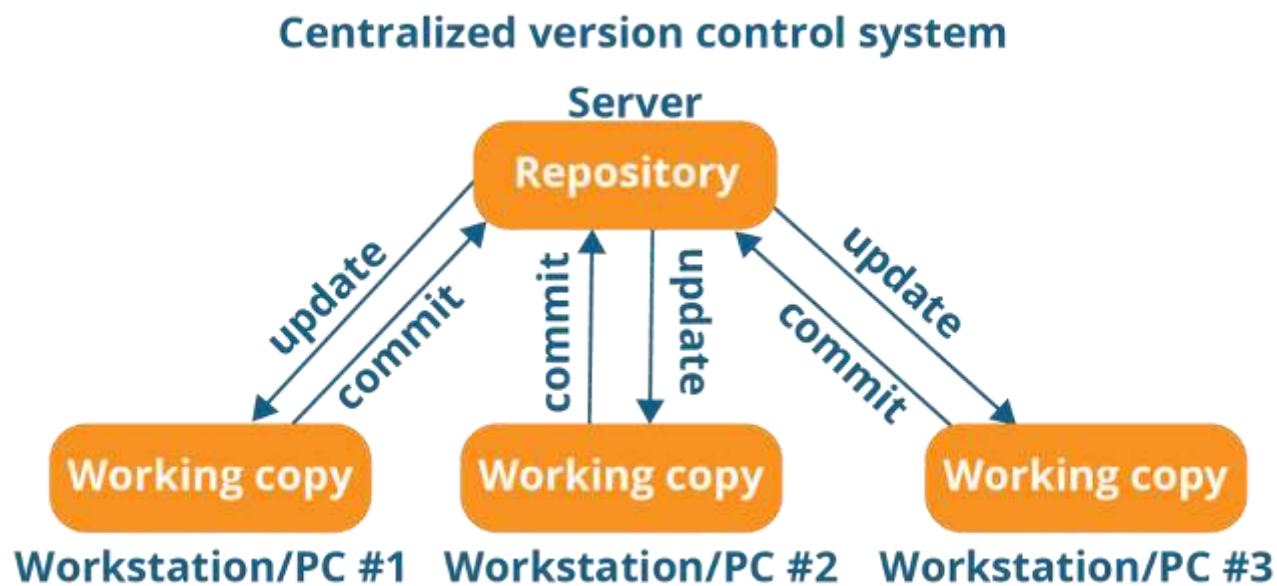
Types of Version Control System

There are two types of version control systems (VCS).

- Centralized version control systems (CVCS)
- Distributed version control systems (DVCS).

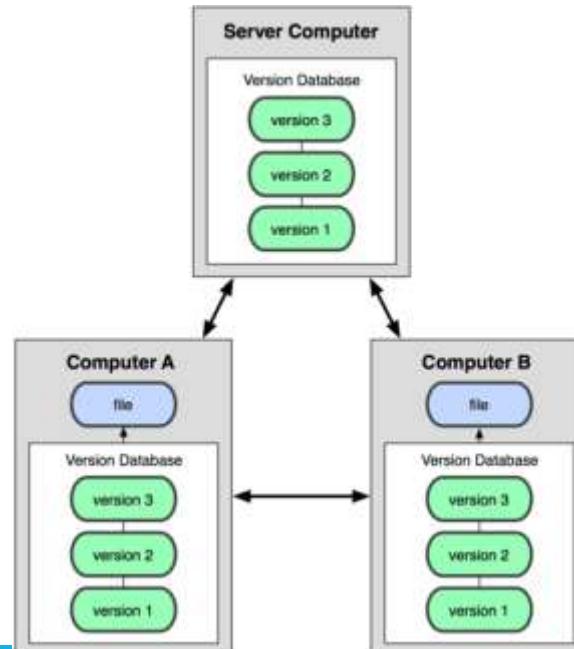
Centralized version control systems (CVCS)

Centralized systems have a copy of the project hosted on a centralized server, to which everyone connects to in order to make changes. Here, the “first come, first served” principle is adopted: if you’re the first to submit a change to a file, your code will be accepted.



Distributed version control systems (DVCS).

In a distributed system, every developer has a copy of the entire project. Developers can make changes to their copy of the project without connecting to any centralized server, and without affecting the copies of other developers. Later, the changes can be synchronized between the various copies.



Advantage of Version Control System

- With a distributed system, you can work on your copy of the code without having to worry about ongoing work on the same code by others.
- Identify the ownership of changes
- you can **sync your repositories among yourselves**, bypassing the central location.
- **Managing access is easier** in distributed systems.



TOP VERSION CONTROL SYSTEMS

Introduction of Git

Git History

Originally authored by Linus Torvalds for development of Linux kernel.

Since 2002 BitKeeper was being used for Linux development.

They revoked the free licencing

- April 5, 2005 - Linus sends out email showing first version
- June 15, 2005 - Git used for Linux version control



What is Git

- Git is a distributed version control system that is used by developers to manage changes to a codebase.
- Git uses a branching model that allows developers to work on different features and changes to the codebase without affecting the main branch.
- Git is fast, scalable, and efficient, making it an essential tool for software development.
- Git is used by millions of developers and companies worldwide, and it can be used with a wide range of programming languages and operating systems.
- Git provides a range of tools and commands for managing changes to the codebase, including committing changes, branching, merging, and resolving conflicts.

Why Git?

- **Branching:** gives developers a great flexibility to work on a replica of master branch.
- **Distributed Architecture:** The main advantage of DVCS is “**no requirement of network connections to central repository**” while development of a product.
- **Open-Source:** Free to use.
- **Integration with CI:** Gives faster product life cycle with even faster minor changes.

Features of Git

- Tracks history
- Free and open source
- Supports non-linear development
- Creates backups
- Scalable
- Supports collaboration
- Branching is easier
- Distributed development

Git repository

A Git repository is the directory where all of your project files and the related metadata resides.

Git records the current state of the project by creating a tree graph from the index(staging area).

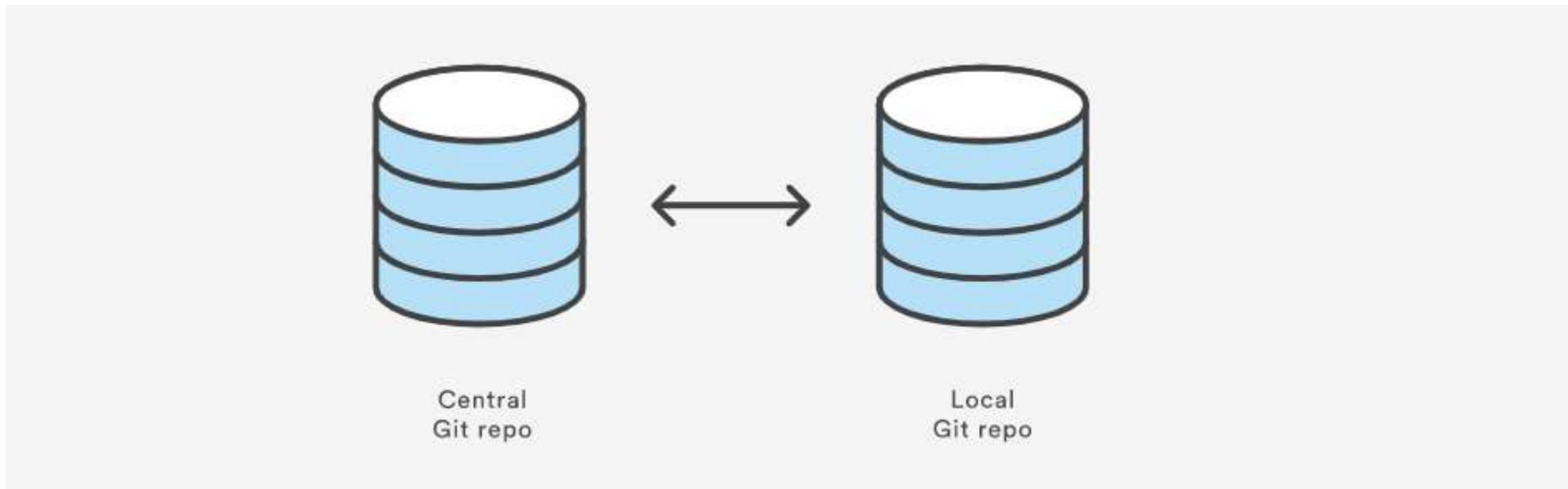
It uses a data structure called the Directed Acyclic Graph (DAG) for managing different commits(each commit is a node in this DAG).

Local Git repository

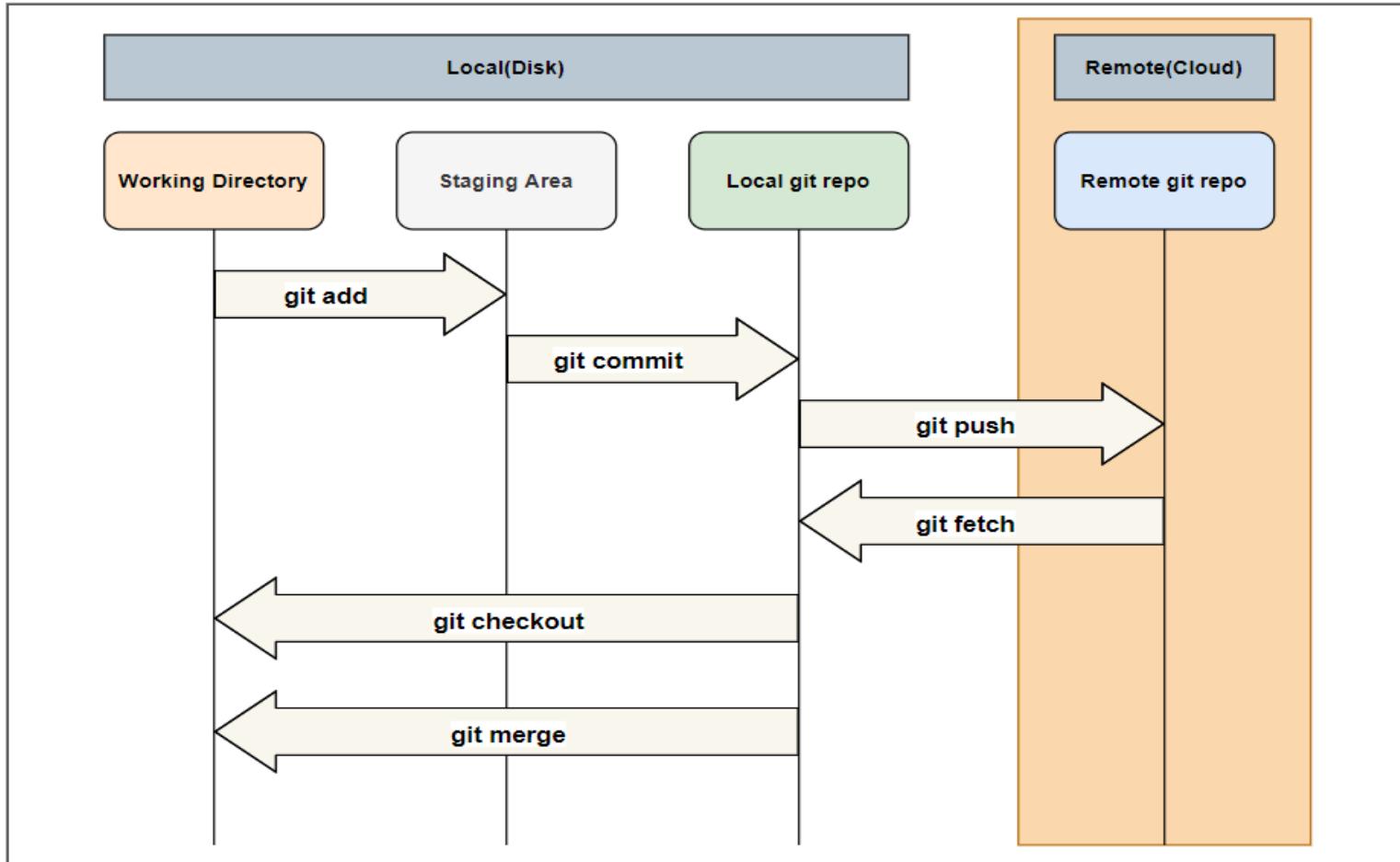
A local Git repository is a storage of your project files and metadata on your computer's disk. It allows you to save versions of your code, which you can access whenever needed.

Remote Git repository

A remote Git repository is a virtual storage of your project on git hosting servers like GitHub or Bitbucket. It has the same copy of your project as you have locally.



Basic Flow of Commands Between Local and Remote Repositories



GIT Architecture

The Working Tree

The Working Tree is the area where you are currently working. It is where your files live. This area is also known as the “untracked” area of git. Any changes to files will be marked and seen in the Working Tree.

The Staging Area (Index):

The Staging Area is when git starts tracking and saving changes that occur in files. These saved changes reflect in the .git directory.

The Local Repository:

The Local Repository is everything in your .git directory. Mainly what you will see in your Local Repository are all of your checkpoints or commits. It is the area that saves everything

Staging Area

- The git staging area is in the middle of the workspace and repository. It is a file which stores the information of the added files. Before a commit, you add changed files or new files. The information of those files will be saved in the file which represents the index or staging area.
- You can add files or remove files from the staging area



Let's Start With Git Basic Commands

Installing Git with Default Packages

- First, use the apt package management tools to update your local package index.
`sudo apt update`
- With the update complete, you can install Git:
`sudo apt install git`
- You can confirm that you have installed Git correctly by running the following command and checking that you receive relevant output.
`git --version`

Setup Git

Before you start using Git, you need to set it up on your local machine. Here's how:

- **Install Git:** If you haven't already, download and install Git from the official website (<https://git-scm.com/downloads>). Once installed, open your terminal or command prompt to verify the installation by typing `git --version`.
- **Configuration:** Set your name and email address for Git to use in your commits using `git config`. This command sets the author name and email address respectively to be used with your commits.

`git config --global user.name "Your Name"`

`git config --global user.email "youremail@example.com"`

git init

git init <directory> : This command creates empty git repo in the specified <directory>. If you run it with no arguments then it initializes the current directory as a git repository.

git init <directory>

After running this command you will find the .git folder in your directory where you initialized the local repository. This contains all objects, refs, and config related to the local repository.

git Clone

- git clone is a Git command used to create a copy of an existing Git repository.
- It essentially allows you to copy a repository, including all its files, branches, commit history, and configurations from a remote repository (such as one hosted on GitHub, GitLab, Bitbucket, etc.) to your local machine

Syntax :

```
git clone <repository_URL> [<directory_name>]
```



- ✓ <repository_URL>: The URL of the remote repository you want to clone.
- ✓ [<directory_name>] (optional): The name of the directory where you want to clone the repository. If not specified, the repository will be cloned into a directory with the same name as the repository itself.

Git Add Command

- git add is a command in Git that adds changes in the working directory to the staging area, preparing them to be included in the next commit.
- It allows you to select specific files or parts of files (with the interactive mode) to be included in the commit.

git add [<file_pathspec>...]

- **git add** moves changes from the working directory to the staging area, preparing them for the next commit. After staging changes using **git add**, you need to commit them using **git commit** to permanently save those changes to the Git history.
- It's possible to add only specific parts of a file to the staging area using the interactive mode.

git add -p filename.txt

This command allows you to interactively choose which changes within the file filename.txt to stage.

Git commit Command

- git commit is a command in Git that records the changes made to the files in the staging area as a new commit in the version history of the repository.

git commit -m "Commit message"

Here **-m "Commit message"**: Flag used to specify a commit message, describing the changes made in the commit. The commit message should be meaningful and concise, summarizing the changes made.

Git commit Command

- After staging changes using **git add**, you use **git commit** to permanently save those changes to the Git repository's history.
- When you run **git commit**, Git opens a text editor (unless the **-m** flag is used to specify the message inline) where you can provide a detailed commit message explaining what changes were made.
- A commit is a snapshot of the changes made to the repository at a particular point in time. Each commit has a unique identifier (hash) and contains information about the changes, including the author, timestamp, and the commit message.
- It's good practice to create commits that represent logical units of work, such as implementing a new feature, fixing a bug, or making specific changes. This helps in maintaining a clear and comprehensible history of the project.
- Commits serve as checkpoints in the project's history, enabling you to track changes, revert to previous states if needed, collaborate with others, and understand the evolution of the codebase.

Git Status Command

- The git status command in Git is used to display the current state of the working directory and the staging area (index), providing information about files that have been modified, staged, or untracked.

Syntax : `git status`

When you run `git status`, you'll see information categorized into three main sections:

1. Changes not staged for commit:

- Lists modified files that have been changed but not yet staged for the next commit.
- Files that are modified but not added to the staging area using `git add` will appear under this section.

2. Changes to be committed:

- Displays files that are staged and ready to be committed.
- Files that have been added to the staging area using `git add` will be listed here.

3. Untracked files:

- Lists files in the working directory that are not tracked by Git.
- These files have not been added to the staging area and are not part of the repository history.

```
On branch main
Your branch is up to date with 'origin/main'.


Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   file1.txt
    modified:   file2.txt


Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:  newfile.txt


Untracked files:
  (use "git add <file>..." to include in what will be committed)
    new_untracked_file.txt
```

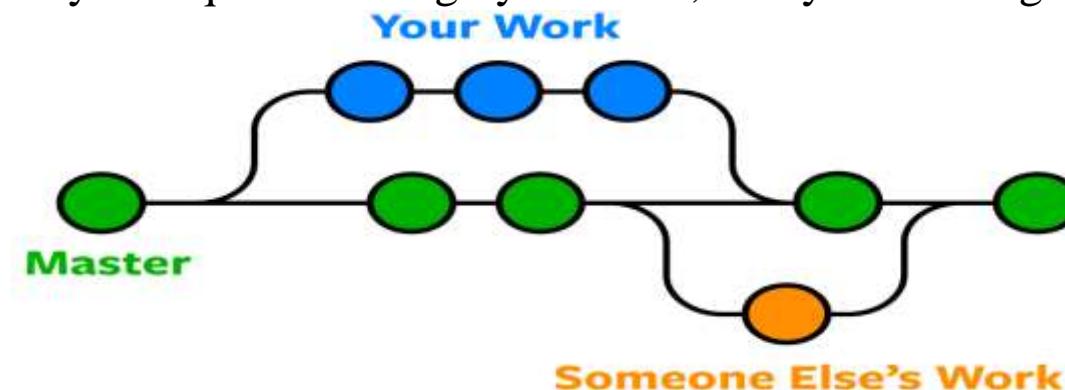
Git diff

- The git diff command in Git is used to show changes between different Git objects, such as commits, branches, files, and the working directory. It provides a detailed view of the differences in content between two points in the Git history

Syntax : **git diff**

Branch in Git

- Git branching is the process of creating multiple, independent lines of development within a single repository. Each branch represents a separate version of the codebase, allowing developers to work on new features, bug fixes, or experiments without affecting the main codebase.
- Once the work on a branch is complete and tested, it can be merged back into the main branch (ideally with a pull request where you request to merge your code, and your colleagues check and approve it), integrating the changes



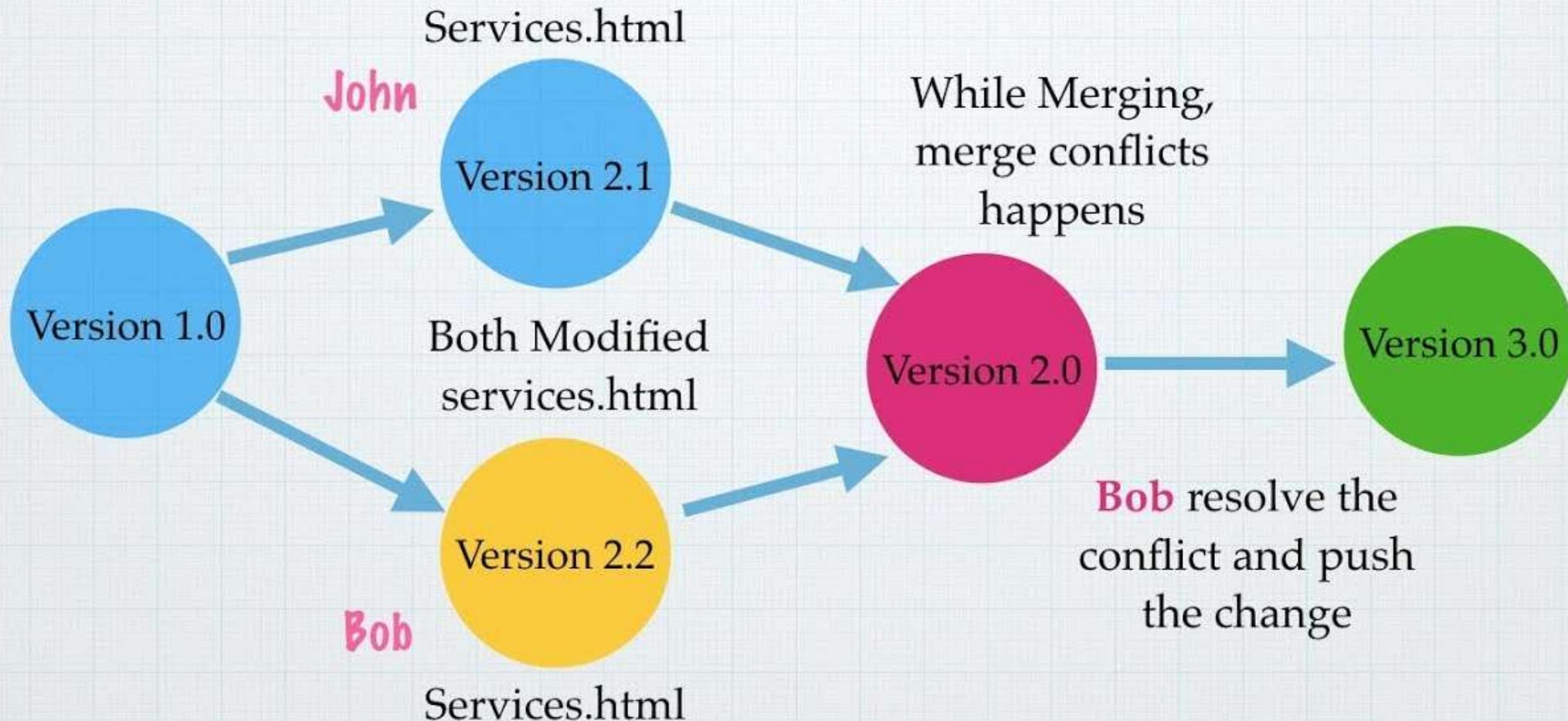
Why Use Branches?

- **Isolated development** — Branches provide a safe environment for you to mess around without worrying about screwing up the main branch.
- **Collaboration** — Multiple developers can work simultaneously on different branches.
- **Easy integration** — we can easily merge our code from different branches back into the main branch for easy integration.
- **Experimentation** — Branches allow developers to experiment with new ideas or techniques without the risk of breaking the main codebase. If you screw up, you can simply delete the branch and start over.
- Code reviews become more focused and manageable.

Git Merge Conflicts

Git merge conflicts occur when there are conflicting changes between two branches that Git is attempting to merge. This typically happens when two branches have made different modifications to the same part of a file, and Git cannot automatically determine which changes to keep. When a merge conflict occurs, Git requests manual intervention to resolve the conflict

Work with merge conflict in Git



Git Stash

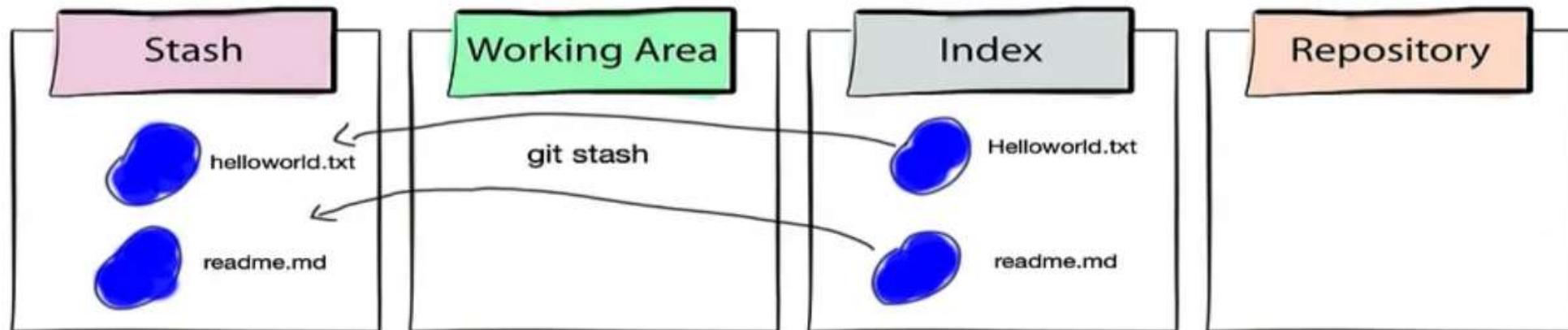
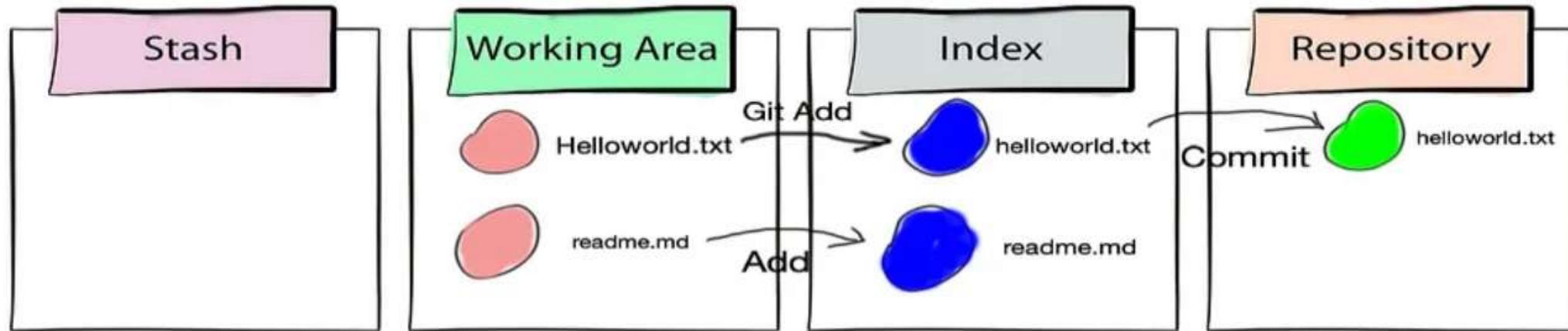
- The **git stash** command is used to temporarily save changes that are not ready to be committed.
- This is useful when you need to switch to a different branch or work on a different task without committing your changes.
- When you run **git stash**, Git will take the changes that you've made to tracked files (i.e., files that are already being tracked by Git) and save them in a "stash" or "temporary commit."
- Git will then revert the working directory to the state it was in when you last committed changes.

Git Stash Commands

- `git stash` → used for send the file from WD to stash
- `git stash apply` → apply the latest stash
- `git stash apply stashid` → used for fetch file from stash to WD
- `git stash show` → to know the status of stash
- `git stash clear` → for clear stash storage
- `git stash list` → list the all files in stash area
- `Git stash push -m "new feature"` → to identify the stash
- `Git stash drop 1`
- `Git stash pop` → `git stash apply + git stash drop`

To handle the merge conflict, first commit the changes then go for `git stash apply stashid` or `Git stash drop 1`

Git Stash



Git Reset and Revert

- The **git reset** command is used to reset the current branch to a specific commit or to unstage changes that have been staged for commit. This means that git reset changes the commit history by removing commits from the branch. If you use git reset to undo a commit, the commit and all its changes will be permanently removed from the branch history. This means that any changes that were made in the removed commit will be lost.
- The **git revert** command is used to create a new commit that undoes the changes made by a previous commit. This means that git revert does not remove any commits from the branch history. Instead, it adds a new commit that contains the changes needed to undo the previous commit. This allows you to keep a complete history of all changes made to the branch.

Git Reset and Revert

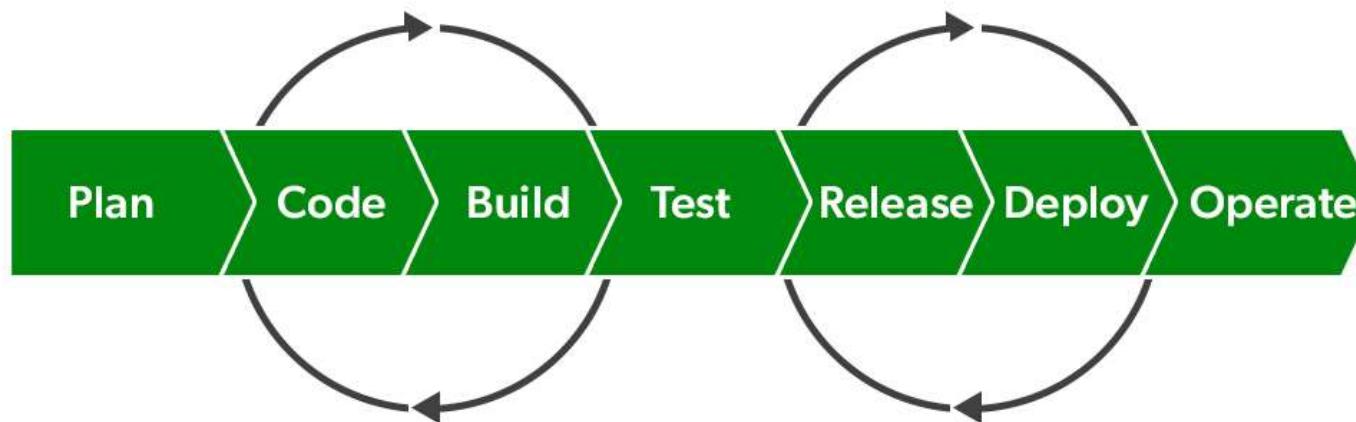
- Git reset file : reset file from staging area to WD
- Git reset –Hard : remove file from staging area and WD
- Git reset commitid : It will reset the commit
- Git revert commit id : it will revert the commit and remove all changes in files also

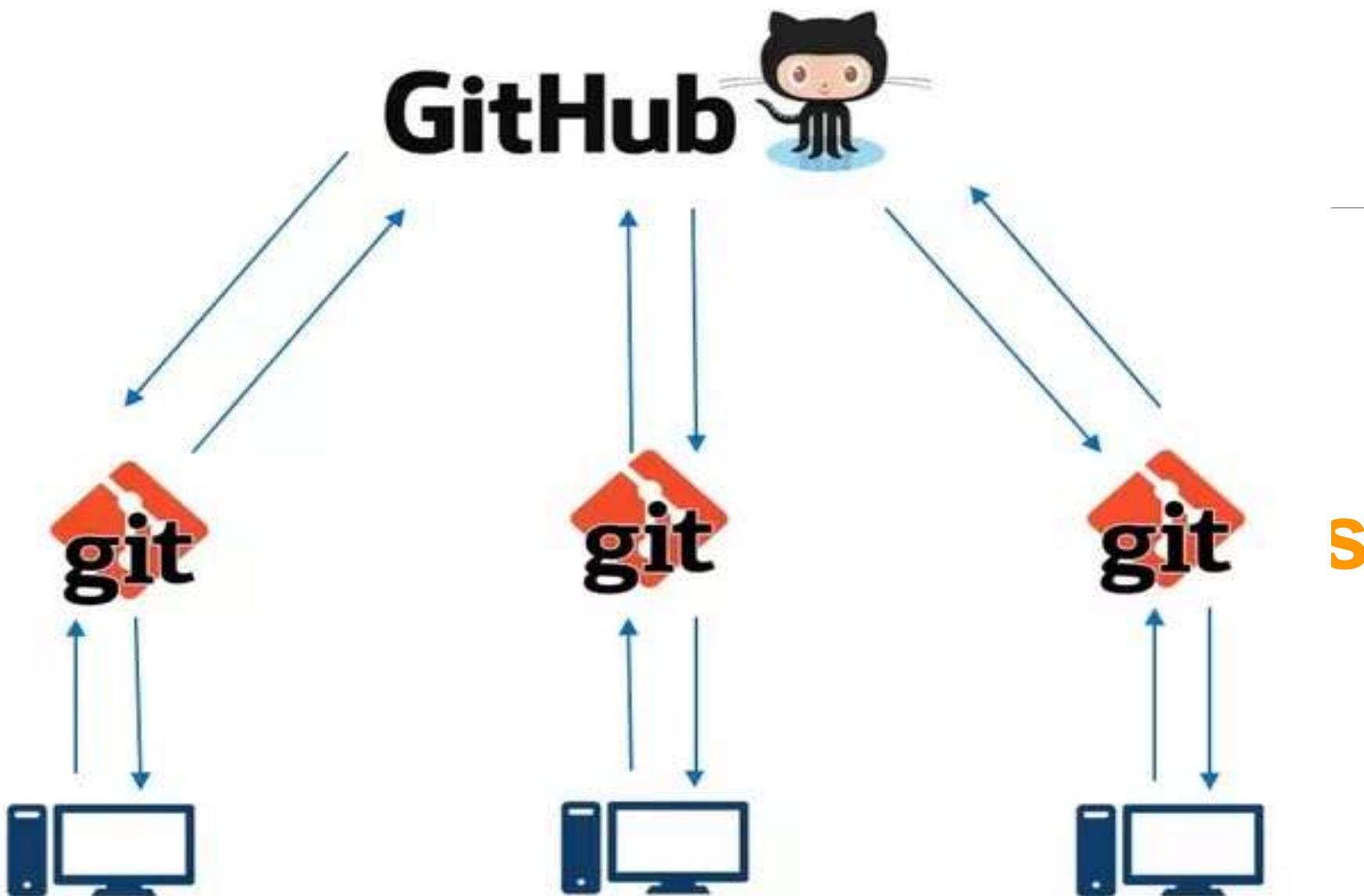
Git Tag

- In Git, a tag is a label that points to a specific commit in a repository's history. It is a way to mark a specific version or release of the code at a certain point in time.
- A Git tag is like a bookmark in a book. It allows you to mark a specific commit and easily refer back to it later. This can be useful for keeping track of releases or marking important milestones in a project's development history.

Git Hub Action

GitHub Actions is a CI/CD (Continuous Integration/ Continuous Deployment) platform for automating the builds, test, and deployment process. Using GitHub actions, we can build and test every pull request in the repository using workflows, or push the merged pull requests to production with workflows





Knowledge Check

What is a branch in Git?

- a) A branch is a separate line of development in a Git repository where changes can be made without affecting the main branch.
- b) A branch is a cloud-based storage service for Git repositories.
- c) A branch is a type of commit used for merging changes from one branch to another in a Git repository.
- d) A branch is a folder within a Git repository where specific files are stored.

Knowledge Check

What is the default branch name for a newly created repository?

- a) Master
- b) Develop
- c) Release
- d) Main

Version Control helps in

It's not just for code, it also helps in

- Backups & Restoration
- Synchronization
- Reverts
- Track Changes
- Most importantly in Parallel Development

Centralized Version Control System

Traditional version control system

- Server with database
- Clients have a working version

Examples

- CVS
- Subversion
- Visual Source Safe

Challenges

- Multi-developer conflicts
- Client/server communication

GIT Installation

GIT comes as default offering for all major Linux flavors

Though you can download the installers from below link for Windows, Mac OS X, Linux, Solaris

- <https://git-scm.com/downloads>
- you can install same with yum too:
 - `yum install git`

Installing GIT Bash on Windows

GIT VERSION

```
[root@user20-master plays]# git --version  
git version 1.8.3.1  
[root@user20-master plays]#
```

Setting Identity in GIT

```
[root@Techlanders ~]# git config --global user.email "Sandeep@Techlanders.com"
[root@Techlanders ~]# git config --global user.name "Sandeep"
[root@Techlanders ~]# git config --global -l
user.name=Sandeep
user.email=Sandeep@Techlanders.com
[root@Techlanders ~]#
```

GIT Repository

A **repository** is usually used to organize a single project.

Repositories can contain folders and files, images, videos, spreadsheets, and data sets – anything your project needs.

Repository is like a unique shared file system for a project.

Initializing a Repository

```
[root@master git]# mkdir /Repo1
[root@master git]# cd /Repo1/
[root@master Repo1]# git init
Initialized empty Git repository in /Repo1/.git/
[root@master Repo1]# ls -lrt /Repo1/.git/
total 12
drwxr-xr-x. 4 root root 31 Dec 19 19:32 refs
drwxr-xr-x. 2 root root 21 Dec 19 19:32 info
drwxr-xr-x. 2 root root 242 Dec 19 19:32 hooks
-rw-r--r--. 1 root root 73 Dec 19 19:32 description
drwxr-xr-x. 2 root root 6 Dec 19 19:32 branches
drwxr-xr-x. 4 root root 30 Dec 19 19:32 objects
-rw-r--r--. 1 root root 23 Dec 19 19:32 HEAD
-rw-r--r--. 1 root root 92 Dec 19 19:32 config
[root@master Repo1]#
```

Adding file to a Repository

```
[root@master Repo1]# git status
# On branch master
# Initial commit - nothing to commit (create/copy files and use "git add" to track)
[root@master Repo1]# echo "File1 content" >> file1
[root@master Repo1]# ll
-rw-r--r--. 1 root root 14 Dec 19 19:35 file1
[root@master Repo1]# git add file1
[root@master Repo1]# git status
# On branch master
# Initial commit
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#       new file:  file1
#
[root@master Repo1]#
```

Checking Repository Status

```
[root@user20-master Repo1]# touch file2
[root@user20-master Repo1]# git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:  file1
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       file2
[root@user20-master Repo1]#
```

Committing changes to a Repository

```
[root@user20-master Repo1]# git add --all
[root@user20-master Repo1]# git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:  file1
#       new file:  file2
#
[root@user20-master Repo1]# git commit -m "Commit one - Added File1 & File2"
[master (root-commit) 9c301cb] Commit one - Added File1 & File2
2 files changed, 1 insertion(+)
create mode 100644 file1
create mode 100644 file2
[root@user20-master Repo1]#
```

Committing changes to a Repository

```
[root@master Repo1]#echo "File2 content added" >file2
[root@master Repo1]#git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:  file2
#
no changes added to commit (use "git add" and/or "git commit -a")
[root@master Repo1]#git commit -am "second commit - Changes done in File2"
[master 77de849] second commit - Changes done in File2
 1 file changed, 1 insertion(+)
[root@master Repo1]#
```

Git Log

```
[root@master Repo1]#git log  
commit 77de8496c39c8d442d8e1212f9f3879a33253a1c  
Author: admin.gagan@gmail.com <admin.gagan@gmail.com>  
Date:  Wed Dec 19 19:48:56 2018 +0000
```

second commit - Changes done in File2

```
commit 9c301cb93733f666e959a87c7a3f61142d1d9f48  
Author: admin.gagan@gmail.com <admin.gagan@gmail.com>  
Date:  Wed Dec 19 19:43:25 2018 +0000
```

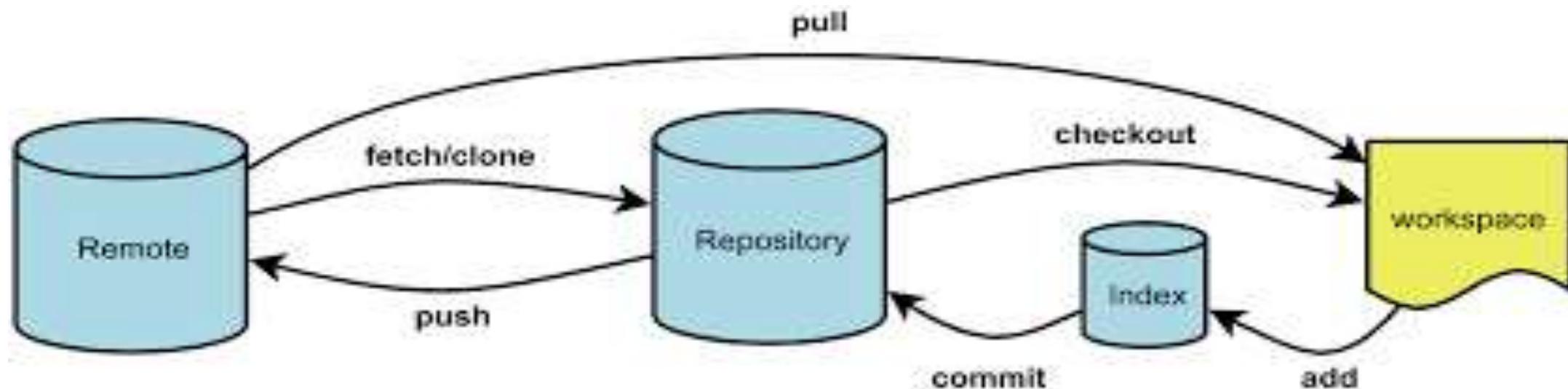
Commit one - Added File1 & File2

```
[root@master Repo1]#
```

Git Diff- Comparing two commits

```
[root@master Repo1]#git diff 9c301cb93733f666e959a87c7a3f61142d1d9f48  
77de8496c39c8d442d8e1212f9f3879a33253a1c  
diff --git a/file2 b/file2  
index e69de29..de51b99 100644  
--- a/file2  
+++ b/file2  
@@ -0,0 +1 @@  
+File2 content added  
[root@master Repo1]#
```

Git Flow



Git Push

```
[root@master Repo1]#git push origin master
Username for 'https://github.com': admin.sandeep@gmail.com
Password for 'https://admin.sandeep@gmail.com@github.com':
Counting objects: 14, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (9/9), done.
Writing objects: 100% (12/12), 1.22 KiB | 0 bytes/s, done.
Total 12 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/adminsandeep/ansibleplaybooks.git
  d9b5ae7..64bad4d master -> master
[root@master Repo1]#
```

Initializing a Remote Repository

```
[root@master Repo1]#git remote add origin https://github.com/adminsandeep/repo1.git
[root@master Repo1]#
[root@master Repo1]#git pull origin master
From https://github.com/adminsandeep/repo1
 * branch      master    -> FETCH_HEAD
Merge made by the 'recursive' strategy.
 README.md |  1 +
 abc       |  0
 ntp.yaml | 13 ++++++
3 files changed, 14 insertions(+)
create mode 100644 README.md
create mode 100644 abc
create mode 100644 ntp.yaml
[root@master Repo1]#
```

Git Pull

```
[root@master Repo1]#ls  
abc file1 file2 ntp.yaml readme5 README.md  
  
[root@master Repo1]#git pull origin dev  
remote: Enumerating objects: 4, done.  
remote: Counting objects: 100% (4/4), done.  
remote: Compressing objects: 100% (2/2), done.  
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0  
Unpacking objects: 100% (3/3), done.  
From https://github.com/admingagan/ansibleplaybooks  
 * branch      dev      -> FETCH_HEAD  
Merge made by the 'recursive' strategy.  
readme6 | 1 +  
1 file changed, 1 insertion(+)  
create mode 100644 readme6  
[root@master Repo1]#ls  
abc file1 file2 ntp.yaml readme5 readme6 README.md  
[root@master Repo1]#
```

Git Clone

```
[root@user20-master git]# git clone https://github.com/admingagan/test.git
Cloning into 'test'...
remote: Enumerating objects: 23, done.
remote: Counting objects: 100% (23/23), done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 23 (delta 5), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (23/23), done.
[root@user20-master git]# cd test
[root@user20-master test]# ll
total 16
-rw-r--r--. 1 root root 10 Dec 20 07:45 Readme
-rw-r--r--. 1 root root 24 Dec 20 07:45 Readme2
-rw-r--r--. 1 root root 57 Dec 20 07:45 readme3
-rw-r--r--. 1 root root 9 Dec 20 07:45 README4
[root@user20-master test]#
```

GIT BRANCH

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git branch
* master
```

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git branch training
```

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git branch
* master
  training
```

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git branch -d training
Deleted branch training (was 74e76df).
```

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git branch
* master
```

GIT CHECKOUT

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git checkout training
Switched to branch 'training'

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (training)
$ git checkout -b training_checkout
Switched to a new branch 'training_checkout'

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (training_checkout)
$ git branch
  master
  training
* training_checkout

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (training_checkout)
$ git log
commit 76e137f900eb33b7eb4a4ac7c81f160af8124697 (HEAD -> training_checkout)
Author: Kulbhushan Mayer <kulbhushan.mayer@thinknyx.com>
Date:   Fri Jun 9 20:38:59 2017 +0530

    commit in branch

commit 74e76dfcaf297ae9b63f950988907cdce8d275de (training, master)
Author: Kulbhushan Mayer <kulbhushan.mayer@thinknyx.com>
Date:   Thu Jun 8 23:16:16 2017 +0530

    second commit

commit 3eaadecbb9972b29b7463822e99b485b9d9b490eb
Author: Kulbhushan Mayer <kulbhushan.mayer@thinknyx.com>
Date:   Thu Jun 8 23:14:06 2017 +0530

    my initial commit
```

GIT MERGE

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (training_checkout)
$ git checkout master
Switched to branch 'master'

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git merge training_checkout
Updating 74e76df..76e137f
Fast-forward
 file.txt | 2 ++
 1 file changed, 2 insertions(+)
```

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git merge training
Auto-merging file.txt
CONFLICT (content): Merge conflict in file.txt
Automatic merge failed; fix conflicts and then commit the result.

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master|MERGING)
$ ls -ltr
total 2
-rw-r--r-- 1 kmayer 197121 30 Jun  9 20:51 file-in-training-branch.txt
-rw-r--r-- 1 kmayer 197121 238 Jun 12 20:47 file.txt
```

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master|MERGING)
$ git status
On branch master
You have unmerged paths.
 (fix conflicts and run "git commit")
 (use "git merge --abort" to abort the merge)

Unmerged paths:
 (use "git add <file>..." to mark resolution)

      both modified:   file.txt
```

GIT MERGE – Resolving Conflicts

```
1 Adding dummy data - Kulbhushan Mayer
2 Second Update
3
4 <<<<< HEAD
5 Making changes in branch for demo
6
7 Making one more change - 09/06/2017 08:44
8
9 Making change at 08:46 PM
10 =====
11 Making change to check merging - 12/06/2017
12 >>>>> training
13
```

REMOVAL OF THE CHANGES FROM THE FILE AND MAKE THEM NECESSARY CHANGES

```
1 Adding dummy data - Kulbhushan Mayer
2 Second Update
3
4
5 Making changes in branch for demo
6
7 Making one more change - 09/06/2017 08:44
8
9 Making change at 08:46 PM
10
11 Making change to check merging - 12/06/2017
12
```

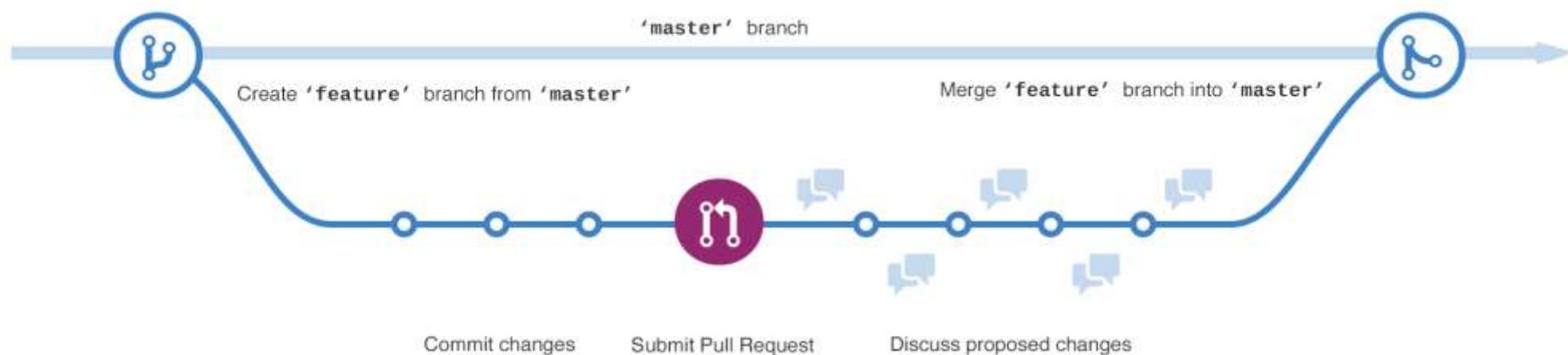
GIT Branch

Here you have:

The master branch

A new branch called feature (because we'll be doing 'feature work' on this branch)

The journey that feature takes before it's merged into master



BitBucket/GitHub/GitLab

What is Bitbucket /GitHub/Gitlab?.

Bitbucket is a Git solution for professional teams. In simple layman language its a UI for Git, offered by Atlassian, similarly we have different available UI solutions from Github (most famous) and Gitlab.

GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

Host in the cloud: free for small teams (till 5 users) and paid for larger teams.

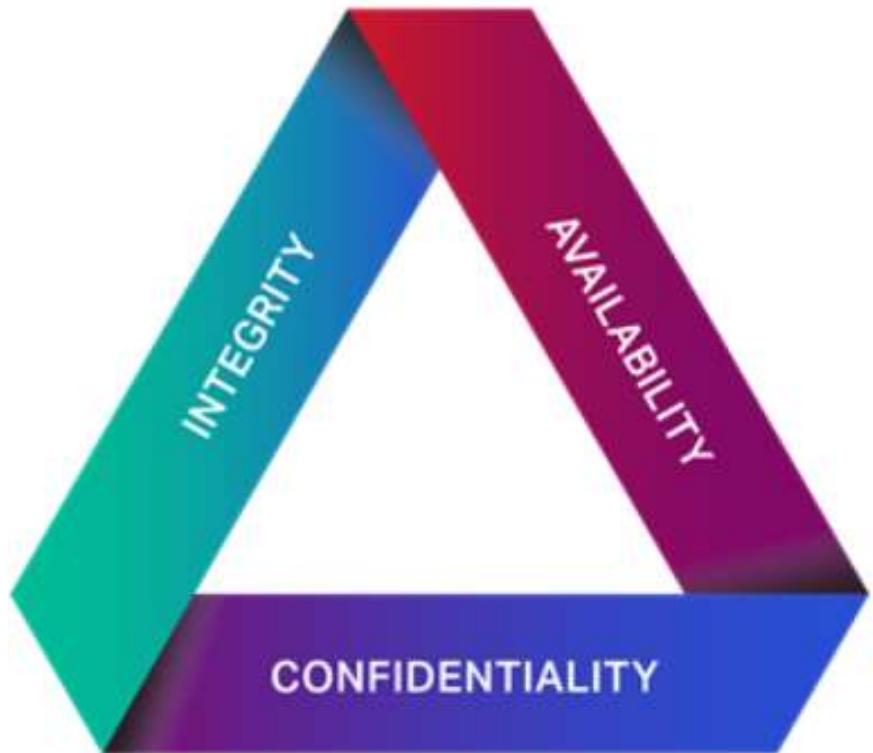
Host on Your server: One-Time pay for most solutions.

Visit “<https://bitbucket.org/>” and click “Get Started” to sign up for free account.

Visit “<https://github.com/>” for Github details

Module 2: Security Culture and Collaboration

Defining Security



CIA Triad:

It's the **confidentiality**, the **integrity** and the **availability** of our information that we're trying to protect.

Defining Security

Confidentiality

- Confidentiality ensures that sensitive information is kept secret and only accessible to authorized individuals or entities.
- It involves measures such as encryption, access controls, and authentication to prevent unauthorized access to sensitive data.
- The goal is to prevent data breaches and unauthorized disclosures of information.

Defining Security

Integrity:

- Integrity focuses on the accuracy and trustworthiness of data and information.
- It ensures that data remains unaltered and reliable throughout its lifecycle.
- Measures to maintain integrity include data validation, checksums, digital signatures, and version control.
- The goal is to protect against unauthorized modifications, data corruption, or tampering.

Defining Security

Availability:

- Availability ensures that information and resources are accessible and usable when needed by authorized users.
- It involves measures to prevent or mitigate disruptions, downtime, or denial of service (DoS) attacks.
- Redundancy, fault tolerance, disaster recovery, and load balancing are strategies used to maximize availability.
- The goal is to ensure that systems and data are accessible and operational, even in the face of failures or attacks.

Common Security Terms

Term	Description
Asset	Anything of value that could be compromised, stolen, or harmed, including information, physical resources, and reputation.
Threat	Any event or action that could potentially cause damage to an asset or an interruption of services.
Attack	The intentional act of attempting to bypass one or more security services or controls of an information system.
Vulnerability	A condition that leaves the system and its assets open to harm.
Exploit	A technique that takes advantage of a vulnerability to perform an attack.
Risk	The likelihood of a threat occurring, as well as its potential damage to assets.
Control	A countermeasure that you put in place to avoid, mitigate, or counteract security risks due to threats or attacks.
Social engineering	The practice of using deception and trickery against human beings as a method of attack.
Defense in depth	The practice of providing security in multiple layers for more comprehensive protection against attack.

Security Culture

- A security culture refers to the collective beliefs, values, attitudes, and behaviors within an organization regarding information security and cybersecurity.
- It reflects the organization's commitment to protecting its information assets, data, systems, and networks.
- Developing a strong security culture is essential for mitigating cybersecurity risks and promoting a proactive and vigilant approach to security across all levels of the organization.

Security Culture

- An information security-positive culture is evident when information is processed in a secure manner by employees, at all times, whilst preserving the integrity, availability and confidentiality of information and abiding by privacy requirements.
- This can suffice only when employees exhibit compliance behaviour in line with regulatory requirements and organisational policies.
- Employees need to have a positive attitude towards the processing of information in order to exhibit acceptable behaviour, which will become the manner in which information is processed over time that postulates in the culture.
- Employees will be able to comply only if they are supported by management and if there are adequate processes and technology safeguards in place to facilitate such compliance.
- A combination of people, process and technology safeguards will, together, aid in inculcating an information security-positive culture. Such a culture can also be referred to as a “healthy” or “strong” information security culture.

Building a Security-First Culture

1. Leadership Support

4. Employee Engagement

7. Integration with Company Values

2. Education and Training

5. Recognition and Rewards

3. Clear Policies and Procedures

6. Open Communication Channel

Building a Security-First Culture

1. Leadership Support

- Ensure that leadership actively supports and promotes a **security-first mindset**. This involves leadership demonstrating a commitment to prioritizing and investing in security measures.
- Leadership commitment sets the tone for the entire organization. When leaders prioritize security initiatives, it sends a clear **message** to employees that security is a fundamental aspect of the organization's values.
- This commitment fosters a culture where security is viewed as a top priority, influencing **decision-making processes** and **resource allocation** throughout the organization.

Building a Security-First Culture

2. Education and Training

- Provide regular **security awareness training** for all employees. This training should cover essential security practices, potential threats, and the importance of safeguarding sensitive information.
- Informed employees are more likely to recognize and respond to security threats. Security awareness training equips employees with the knowledge and skills to identify potential risks and adopt security best practices.
- This **proactive** approach helps create a workforce that is vigilant, knowledgeable, and capable of taking preventive measures, ultimately reducing the likelihood of falling victim to security incidents.

Building a Security-First Culture

3. Clear Policies and Procedures

- Establish and communicate clear security policies and procedures. These guidelines should outline **acceptable behavior, security expectations, and the consequences of non-compliance.**
- Employees understand expectations, reducing the risk of unintentional security breaches.
- Clear policies provide a **framework** for employees to follow, guiding them on how to handle sensitive information, use company resources securely, and comply with regulatory requirements.
- This clarity minimizes the chance of unintentional security lapses and ensures a consistent approach to security across the organization.

Building a Security-First Culture

4. Employee Engagement

- Encourage employees to actively participate in security initiatives. This involves fostering a sense of ownership and shared responsibility for security among all employees.
- Engaged employees are more likely to proactively contribute to a secure work environment. When employees feel invested in the organization's security goals, they are more likely to adopt security practices willingly.
- Actively involving employees in security initiatives promotes a culture where everyone plays a role in maintaining a secure workplace, leading to a more resilient and proactive security posture.

Building a Security-First Culture

5. Recognition and Rewards

- Recognize and reward employees for adhering to security practices. Positive reinforcement can include acknowledgment, incentives, or other forms of recognition.
- Positive reinforcement reinforces desired behaviors. Recognizing and rewarding employees for their commitment to security reinforces the importance of adhering to security practices.
- This creates a positive feedback loop, encouraging continued adherence to security policies and fostering a culture where security-conscious behaviors are valued and recognized.

Building a Security-First Culture

6. Open Communication Channel

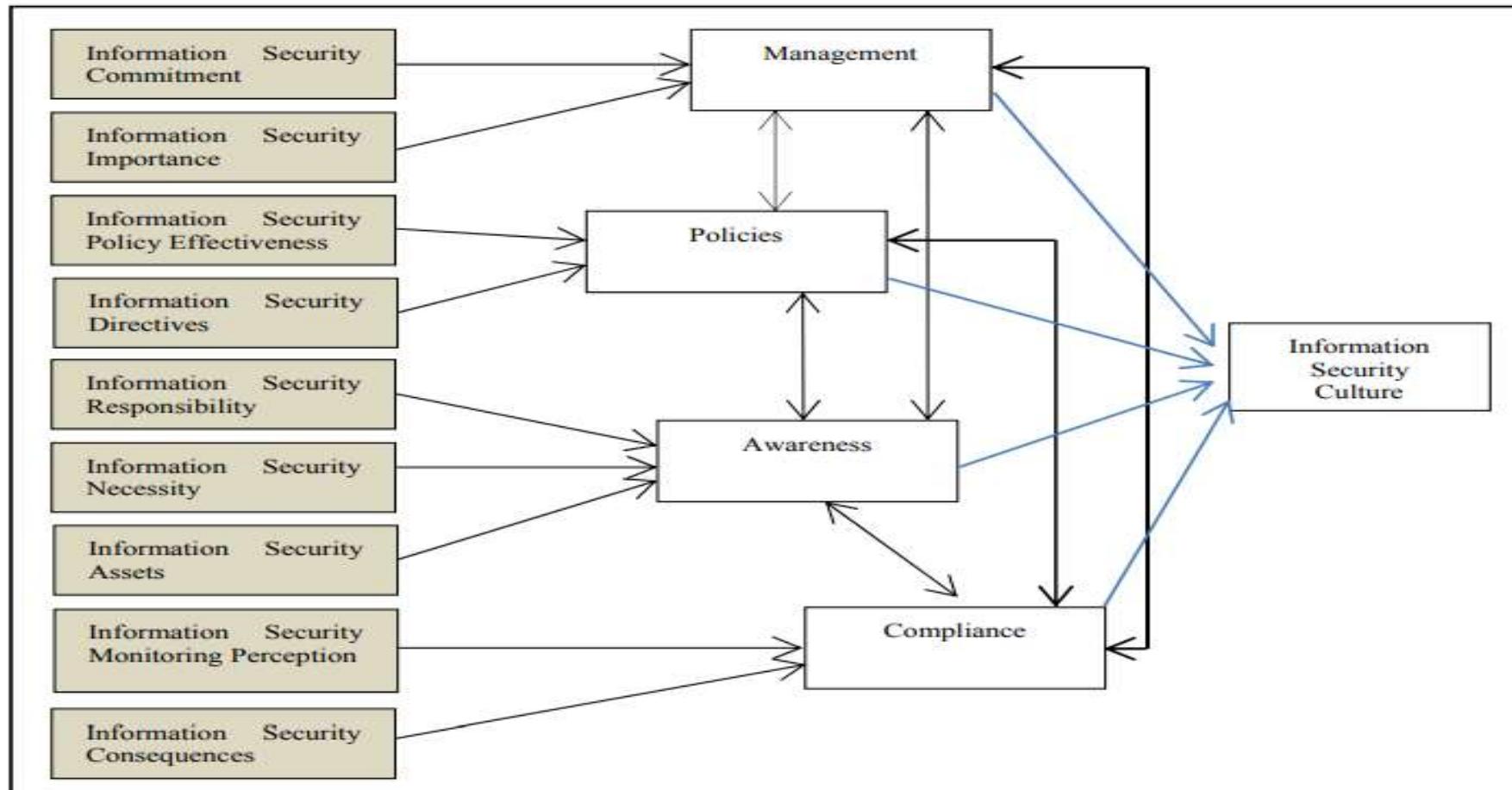
- Foster an environment where employees feel comfortable reporting security incidents or concerns. Establish accessible and confidential channels for reporting security-related issues.
- Early detection and reporting can prevent security incidents from escalating. Creating a culture that encourages open communication about security concerns ensures that potential threats are identified and addressed promptly.
- Early detection allows the organization to implement timely responses and mitigate the impact of security incidents, ultimately contributing to a more secure environment.

Building a Security-First Culture

7. Integration with Company Values

- Align security principles with the organization's core values. This involves embedding security considerations into the broader organizational culture.
- Security becomes an integral part of the company's identity. By aligning security principles with core values, organizations demonstrate a commitment to security as a fundamental aspect of their identity.
- This integration fosters a cohesive culture where security is not seen as an isolated function but as an inseparable component of the organization's overall mission and values. It helps create a sense of shared responsibility for security across the entire organization.

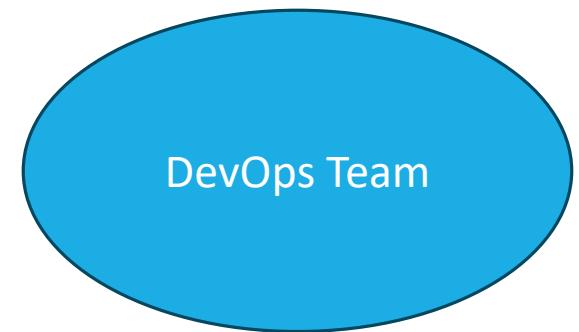
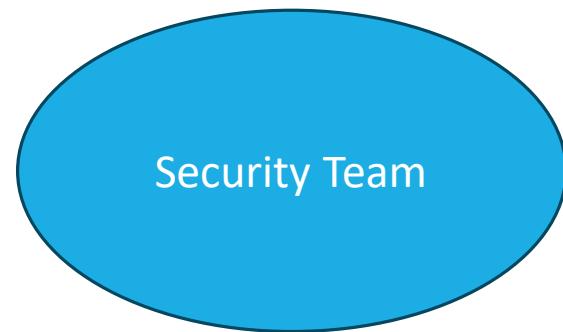
Information Security Culture Model



Collaborative Teams in DevSecOps

Security teams are not capable of ensuring secure development life cycles themselves. We need a collaborative approach.

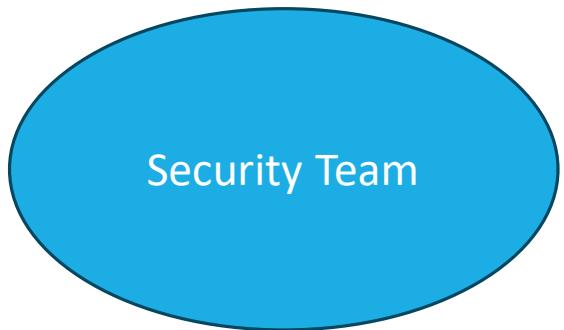
Teams:



Collaborative Teams in DevSecOps

- Different roles involved in the task of DevSecOps adoption, it is quite easy to imagine the difficulty of aligning different teams with different priorities, team sizes, and skillsets to work towards the same goal.
- We need to set the expectations from each team to make the process as easy to digest as possible. We will be looking at the following 5 steps of DevSecOps to make you think of possible ways to increase collaboration in each step.
 1. Threat modeling
 2. Security tests
 3. Prioritization of vulnerabilities
 4. Remediation of vulnerabilities
 5. Monitoring

Security Team



Threat modeling:

- It is shared between security and development teams to prevent vulnerabilities before they arise and it also allows security teams to have a better understanding of the software architecture behind each project to help them with the prioritization of vulnerabilities that are discovered in automated or manual security tests in the later stages.

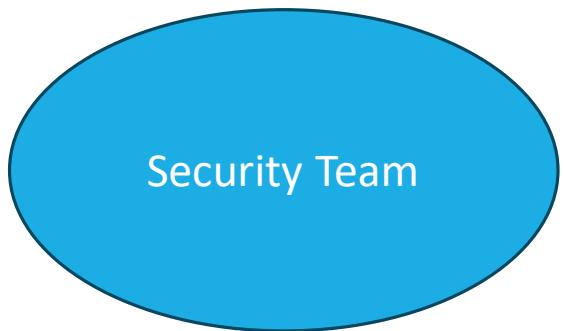
Security Team



Automation:

- In their busy schedule, to free up time for threat modeling activities, security teams need to benefit from automation capabilities of tools.
- Without automation, it is almost impossible for them to keep up with the current speed of development and DevOps and make time for more fruitful activities.
- Once they automate time-consuming and cumbersome tasks, they need to focus their effort on attending threat modeling sessions and the prioritization of vulnerabilities.

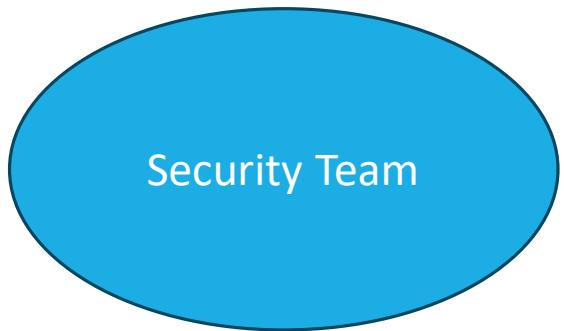
Security Team



SLA:

- Another way security teams can help development teams is by creating SLA levels for different types of vulnerabilities which will make the deadline clear for both parties and set guidelines related to risk acceptance and risk management.

Security Team



Internal Security:

- They can also create internal security libraries and a remediation knowledge base for the use of software developers to help them with the timely remediation of vulnerabilities and to prevent vulnerabilities proactively.

Security Team



Trust and Respect:

- All these activities will help to create a friendly environment where both teams work in harmony and in close collaboration.
- Time spent on these activities will pay off pretty quickly to build trust and earn the respect of development teams.

Software Development Team



Threat Modeling Sessions:

- Developers should actively participate in threat modeling sessions where the team collectively analyzes the software architecture, identifies potential security threats, and devises strategies to mitigate these threats. This proactive approach allows developers to anticipate and prevent vulnerabilities during the design phase.

Software Development Team



Secure Coding Principles:

- Developers are expected to adhere to secure coding principles throughout the software development lifecycle. This includes writing code that is resilient to common security threats such as injection attacks, cross-site scripting (XSS), and other vulnerabilities. Following secure coding practices helps prevent security issues from being introduced during the coding phase.

Software Development Team



Agreed-Upon SLA Levels:

The development team is expected to be responsive in identifying and addressing security vulnerabilities. An SLA defines the timeframe within which identified vulnerabilities must be remediated. This ensures a timely response to security issues and aligns with the DevSecOps principle of integrating security into the development pipeline.

Software Development Team



Timely Remediation:

- Developers play a crucial role in the remediation process. When vulnerabilities are identified through automated security testing or other means, developers should prioritize and fix these issues within the agreed-upon SLA. This ensures that security vulnerabilities are addressed promptly, minimizing the exposure of the software to potential threats.

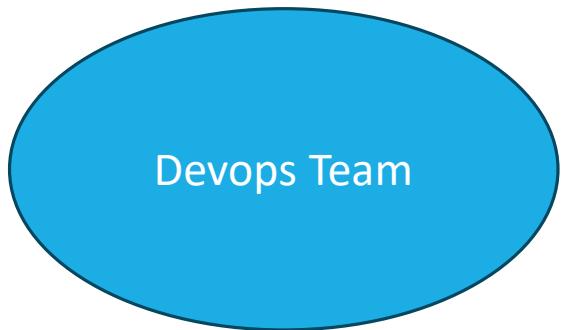
Software Development Team



Share:

- To share how they fix a particular vulnerability by entering a comment on the defect tracker used in the organization.

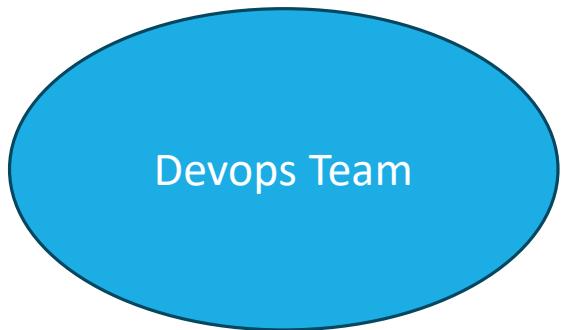
DevOps Team



Pipelines:

- DevOps teams need to be involved in the process to integrate security tests with the pipelines.
- Luckily, with the latest everything-as-code trend, just by adding a few lines of scripts DevOps teams can integrate security tests into pipelines instantly.

DevOps Team



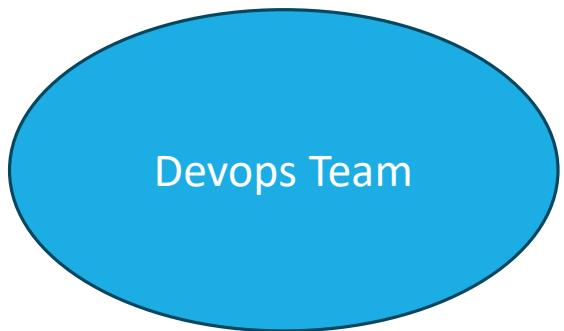
The frequency of scans:

- While some projects can be scanned with each push to the master branch, scans can be triggered by different events in others.

Setting security criteria

- Certain projects with high criticality can be selected and security criteria can be set to break the pipelines automatically when the criteria are not met. This way, security teams can make sure projects with certain vulnerabilities will never make it to production while DevOps teams know why the build has been broken.

DevOps Team



The maximum wait time in pipelines:

Scans of certain projects may take too long and it may not be convenient to halt the pipeline until the scan finishes. For cases like this, maximum wait times can be mutually decided and pipelines can move on while the scan runs in the background after the wait time.

Asynchronous scans:

Certain projects where security is not needed to get in the way of DevOps can also be decided mutually and automated scans can be run asynchronously in these projects without interrupting the pipeline.

What is Communication

- communication is the act of exchanging information from one person to another.
- We can say that communication, is not what you say, but what others understand!
- Though, it is a problem: not always what we think we have transmitted is what others understand. The difficulty to transmit a clear message is called barrier, which can be an influencer through many stages of life.

Communication is the solution, not the problem

- An important point to understand is that technical and non-technical professionals have different ways of communicating, and this same fact can be observed among security and development professionals.
- A clear and effective communication process that could address the aspects of everyone involved in the DevSecOps process.
- A poor communication between the two teams is considered to fall in only one direction, and usually, **one group blames the other**.
- The point is that since communication is a two-way street, if there is no clarity in the sending of the message, the reception is left to a series of assumptions. Eventually we are risking our process.

Communication and Information Sharing

- Cross-Functional Teams
- Transparent Information Sharing
- Joint planning and Design Sessions
- Communication Tools
- Documentation and Knowledge Sharing
- Security Champions
- Integrated tools and Automation
 - Feedback loops
- Incident response planning
- Security Training Programs

Communication and Information Sharing

Cross Functional Teams:

DevSecOps advocates for the formation of cross-functional teams where individuals from development, operations, and security collaborate throughout the entire software development lifecycle.

- Cross-functional teams bring together individuals with diverse skill sets to ensure that the entire spectrum of expertise is represented.
- Team members share the responsibility for both the development and security of applications to foster collective ownership of the end-to-end process.
- Regular interaction means continuous feedback to enable swift adjustments to address security vulnerabilities and operational concerns.

Communication and Information Sharing

Transparent Information Sharing

Transparent information sharing involves open and accessible communication channels that allow teams to share information about security vulnerabilities, incidents, and best practices.

- Utilize collaborative tools and platforms that facilitate real-time communication and information sharing, including chat platforms, documentation systems, and project management tools.
- Conduct regular meetings to discuss ongoing projects, security updates, and any emerging threats. These meetings provide a forum for teams to openly share insights and concerns.
- Maintain comprehensive and accessible documentation that includes security policies, procedures, and incident response plans.

Communication and Information Sharing

Joint Planning and Design Sessions:

- Encourage joint planning and design sessions involving developers and security professionals.
- Collaborative sessions allow for the identification of potential security threats and the establishment of security requirements early in the development process.

Communication and Information Sharing

Communication Tools:

- Utilize collaborative communication tools such as chat platforms, video conferencing, and collaboration platforms.
- Efficient and real-time communication tools facilitate quick information exchange and collaboration among team members, regardless of their physical locations.

Communication and Information Sharing

Documentation and Knowledge Sharing:

- Encourage the documentation of security practices, policies, and procedures.
- Documentation serves as a knowledge base for the team, ensuring that information is shared consistently and can be referenced when needed.

Communication and Information Sharing

Security Champions:

- Appoint security champions within development teams to act as liaisons with the security team.
- Security champions facilitate communication and understanding between developers and security professionals, ensuring that security practices are effectively communicated and adopted.

Communication and Information Sharing

Integrated Tools and Automation:

- Implement integrated tools and automation in the CI/CD pipeline to facilitate information flow.
- Automated tools help streamline the flow of information, such as security scan results, between development, security, and operations teams, enabling quick responses to vulnerabilities.

Communication and Information Sharing

Feedback Loops:

- Establish feedback loops for continuous improvement.
- Regular feedback sessions allow teams to learn from incidents, identify areas for improvement, and adjust security practices accordingly.

Communication and Information Sharing

Incident Response Planning:

- Collaborate on incident response planning and conduct regular drills.
- In the event of a security incident, having a well-coordinated response plan ensures that information is shared promptly, and the incident is mitigated effectively.

Communication and Information Sharing

Security Training Programs:

- Conduct joint security training programs for both development and security teams.
- Shared understanding of security concepts and practices improves communication and collaboration, enhancing the overall security posture.

Importance of Collaboration and communication

Collaboration and communication serve as the foundation for effective DevSecOps practices. Here's why they are crucial for engineers and how they can transform their approach:



Importance of Collaboration and communication

1. Alignment of Objectives:

Collaboration fosters the alignment of objectives among development, security, and operations teams. By working together from the start, teams can understand each other's priorities and integrate security requirements into the development process.

This alignment reduces friction and enhances the overall efficiency of the software delivery lifecycle, enabling engineers to deliver secure and high-quality products.

Importance of Collaboration and communication

2. Early Risk Identification:

Effective communication enables the early identification and mitigation of security risks. When teams openly communicate about potential vulnerabilities or security concerns, they can collectively address them before they escalate into major issues.

This proactive approach enhances the security posture of software systems and helps engineers build robust and resilient applications.

Importance of Collaboration and communication

3. Knowledge Sharing:

Collaboration promotes knowledge sharing among engineers and cross-functional team members. By collaborating with colleagues from different disciplines, engineers can exchange ideas, best practices, and lessons learned.

This knowledge sharing leads to continuous improvement as engineers learn from each other's experiences and apply them to enhance their security practices and technical skills.

Importance of Collaboration and communication

4. Streamlined Incident Response:

In the event of a security incident or breach, collaboration and communication are vital for a swift and effective incident response.

By establishing clear communication channels, incident response teams can coordinate their efforts, share critical information, and respond promptly to mitigate the impact of security incidents.

Engineers play a key role in this process by actively participating in incident response activities and leveraging their technical expertise to resolve issues efficiently.

Importance of Collaboration and communication

5. Enhanced Compliance and Regulatory Standards:

Collaboration and communication facilitate the integration of compliance and regulatory standards into the DevSecOps process.

Engineers working in regulated industries must ensure that their software development practices align with industry regulations.

By involving compliance and governance teams from the early stages, engineers can proactively address compliance requirements, ensuring adherence to standards and building trust with regulators and stakeholders.

Cross Functional Training

Cross functional training or cross-training is a talent development practice where employees learn skills and knowledge that aren't part of their current job description.

Cross functional team training ties in closely with upskilling and internal mobility because it focuses on bringing employees from all different departments and teams together to learn skills not regularly associated with their day jobs.

This way, throughout your organization, you can:

- Share skills and insights
- Break down silos
- Use resources to their maximum potential instead of being stuck in one lane
- Build connections and relationships between team members that help them advance their careers

Benefits of Cross Functional Training



Benefits of Cross Functional Training

Skills gap prevention method:

Cross functional collaboration training helps with building skills and bridging talent gaps (by helping people into new roles upwards or laterally).

- Doing regular skills gap analyses to show you where you're in danger of a shortage
- Cross functional training to help you prevent and close those talent gaps

Benefits of Cross Functional Training

More skill and talent sharing across teams:

- Cross functional training enables you to share skills across departments to strengthen overall knowledge and reduce the need for external help.
- For example, an IT expert can teach the best cybersecurity practices to other departments.
- Cross functional training also helps facilitate a culture of continuous learning.
- People get to share their expertise and use the opportunity to learn from experts about topics they don't know as much about.

Benefits of Cross Functional Training

Increased company resilience and agility/mobility:

- A larger pool of employees with versatile skill sets is an investment in the future.
- With cross functional training, you can keep them in new, modified roles and save money on hiring in the future.
- It gives you a greater number of flexible workers that can bounce across projects and departments, increasing your company's resilience, agility, and employee retention.

Benefits of Cross Functional Training

More trust in the workplace:

Remote workers often report feeling out of place in their company and not trusting their colleagues, so this is a chance to work on it.

Cross functional training facilitates trust in the workplace:

- It's an opportunity to socialize with colleagues and form better relationships
- It provides a higher understanding of everyone's responsibilities, leading to more empathy and better teamwork
- A sense of belonging leads to lower turnover, fewer sick days, and higher performance

Benefits of Cross Functional Training

Goal alignment and better relationships across teams:

- Cross-training is a perfect opportunity to go beyond Slack chatting and task comments to include and connect with everyone.
- It unites the workforce to move together towards the company's goals instead of each department pulling its weight and focusing on its differences.
- When they join forces, experts from different fields have much to offer each other.
- Cross functional training ensures you don't waste good ideas. It also directly contributes to better relationships across departments, leading to a more efficient and fruitful collaboration.

Benefits of Cross Functional Training

Great leadership training for future leaders:

- Leadership training enables you to groom motivated and trustworthy managers who show leadership potential and are familiar with the business instead of looking for a perfect candidate externally.
- Future managers get to test their collaboration and communication skills with colleagues from other departments, not just their closest circle or team.

Benefits of Cross Functional Training

Happier workers with more internal career opportunities:

Instead of leaving the company, employees can turn a fresh chapter internally – everybody wins.

With cross-training, you can help employees experiment with new roles and move around the entire company.

There's no need to confine them within their current job description and its traditional progression. Maybe they want to move on from their role but don't want an upward promotion.

Benefits of Cross Functional Training

Reduced hiring costs:

Cross functional training nurtures a flexible, agile workforce.

Cross-training nourishes your internal talent pool, so you have plenty of great talent to choose from internally, saving you cost and time-to-hire.

Challenges of Cross Functional Collaboration Training

- Employees losing focus on their strengths and responsibilities
 - The training needs to be focused and clear.
- Demotivation due to too much work and/or irrelevant learning
 - Explain to employees why learning about a certain topic or gaining a new skill could help their current performance and which options open which future doors to incentivize them.
- Conflict between employees
 - Prepare communication channels ahead of time, create a process or a standard operating procedure document that dispels uncertainty, and encourage questions.

Introduction of Shift Left Approach

Earlier there was no separate “Testing Phase” for Software development and Testers Role never used to exist at all. Developers used to develop the software, test themselves and make a software release.

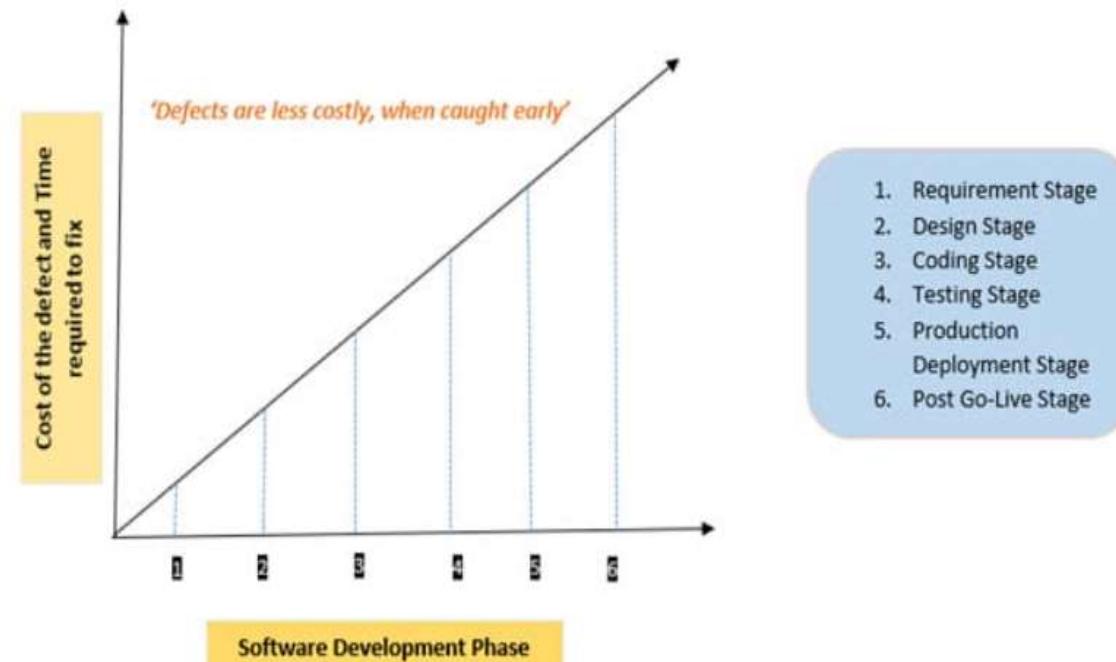
Over a period of time, people have realized the importance of Software Testing and the impact of keeping the ‘**Testing Phase**’ on the extreme right or at the end of the Software Development Lifecycle. This realization happened because the cost of the bug identified towards the extreme right and at the end was very high and enormous effort & too much time was required to fix them.



Introduction of Shift Left Approach

There were cases where after spending so much time and effort on software, due to the crucial bug identified at the end, the mission-critical software could not be released to the market thereby resulting in a huge loss.

Hence, because of the identification of the bug during the last stage either the release was delayed or at times, the software was scrapped by considering the effort required to fix them, which was really not worth it.

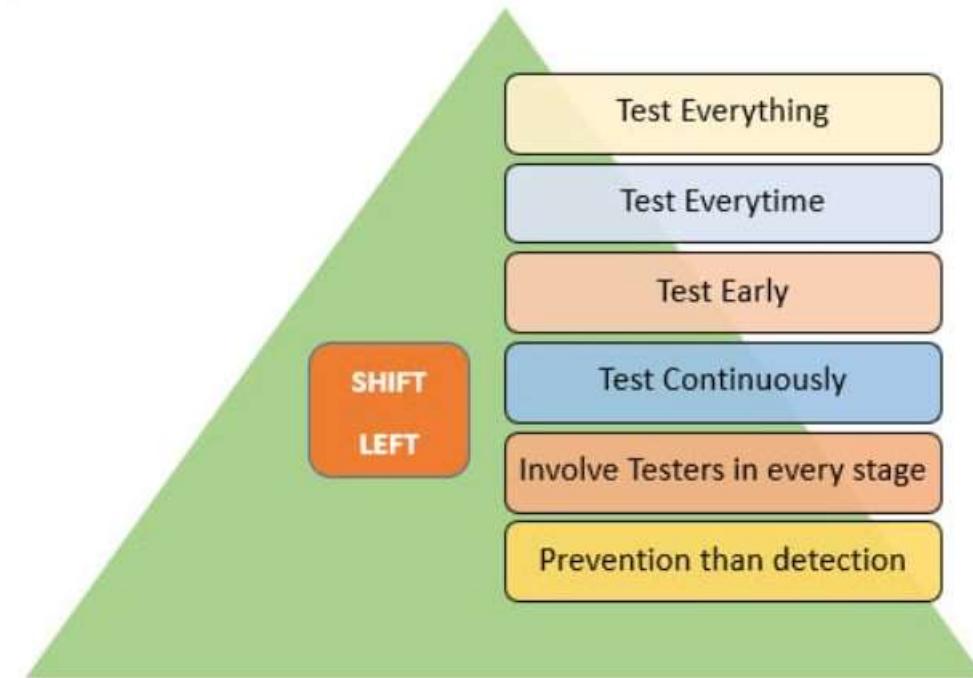


What is Shift Left Testing?

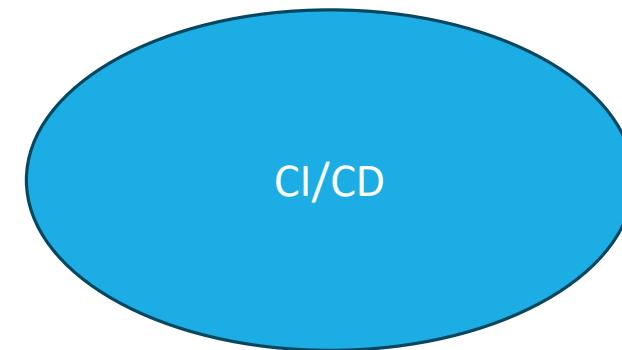
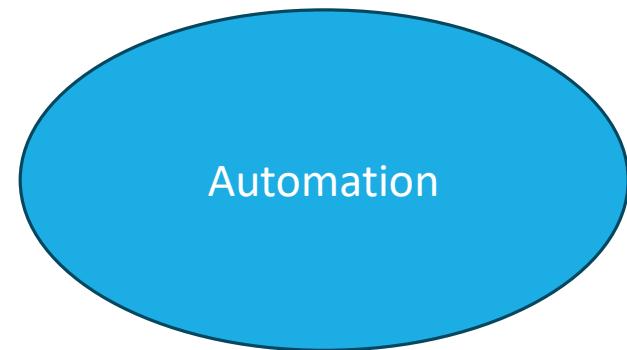
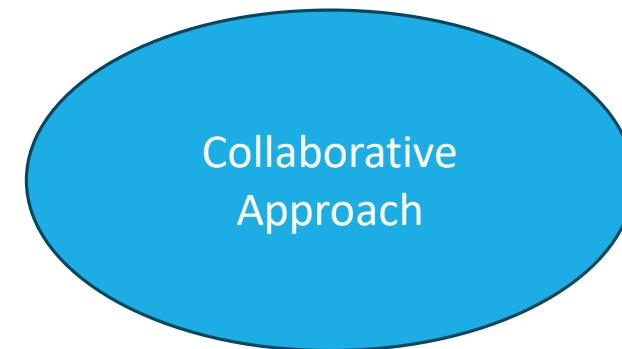
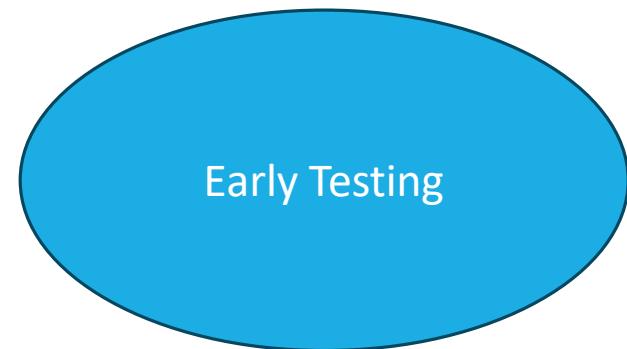
The principle of “Shift left” supports the Testing team to collaborate with all the stakeholders early in the software development phase.

Hence, they can clearly understand the requirements and design the test cases to help the software “Fail Fast” and enable the team to fix all the failures at the earliest.

Allow the testers to understand the requirements, software design, architecture, coding, and its functionality, ask tough questions to customers, business analysts, and developers, seek clarifications and provide feedback wherever possible to support the team.



Key Elements of Shift Left Testing



Shift Left Approach

- Find the defects early thereby reducing the cost of the project.
- Testing continuously again and again to reduce defects in the end.
- Automate everything and improve time to market.
- Focus on customer requirements and improve the customer experience.
- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiations.
- Responding to changes over following a plan.

It's not just about testers

Testers:

Participate actively in the design phase. To test a build, you shouldn't wait for the code. Try to separate client stories into little testable pieces to guarantee that tests are successful and significant.

Developers:

Participate in discussions about test strategy and planning. Your input may help alleviate the testing team's burden and lower your technical debt. Maintain your automated test suites as an essential component of the code base.

It's not just about testers

Product managers:

Make sure your teams have access to the right resources to work together and receive feedback. Set objectives and expectations that are compatible with the shift left strategy. This might entail setting quality objectives for the entire team and anticipating less feature work per sprint.

Infrastructure and deployment teams:

Your deployment pipeline needs to be available early and have a high capacity so test suites can run promptly because testing needs to happen frequently and early.

Module 3:

Risk Assessment and Threat

Modelling

What is a Vulnerability?

A vulnerability is a weakness, flaw or other shortcoming in a system (infrastructure, database or software), but it can also exist in a process, a set of controls, or simply just the way that something has been implemented or deployed.

- Technical Vulnerabilities
- Human Vulnerabilities

Some vulnerabilities are routine

you release something and quickly follow up with a patch for it. The issue with the weakness is when it is unknown or undiscovered to your team. If it's left as-is, this weakness could be vulnerable to some attack or threat.

Examples: Common vulnerabilities such as SQL injection, cross-site scripting (XSS), security misconfigurations, and known software vulnerabilities in third-party libraries.

DevSecOps teams understand that certain types of vulnerabilities frequently appear in software. Recognizing these routine vulnerabilities allows teams to proactively implement measures to prevent, detect, and remediate them.

What is a Threat?

Anything that could exploit a vulnerability, which could affect the confidentiality, integrity or availability of your systems, data, people and more. (Confidentiality, integrity and availability, sometimes known as the CIA triad, is another fundamental concept of cybersecurity.)

when an attacker has the opportunity, capability and intent to bring a negative impact upon your operations, assets, workforce and/or customers.

Examples of this can include:

- Malware
- Ransomware
- Phishing attacks
- more — and the types of threats out there will continue to evolve.

Types of Threats

The main types of threats are :

- **Intentional Threats:** Things like malware, ransomware, phishing, malicious code, and wrongfully accessing user login credentials are all examples of intentional threats. They are activities or methods bad actors use to compromise a security or software system.
- **Unintentional threats:** Unintentional threats are often attributed to human error. For Example misconfigurations, Inadequate Access Controls, data leakage etc
- **Natural threats:** While acts of nature (floods, hurricanes, tornadoes, earthquakes, etc.) aren't typically associated with cybersecurity, they are unpredictable and have the potential to damage your assets.

What is a Risk?

Risk is the probability of a negative (harmful) event occurring as well as the potential of scale of that harm. Your organizational risk fluctuates over time, sometimes even on a daily basis, due to both internal and external factors.

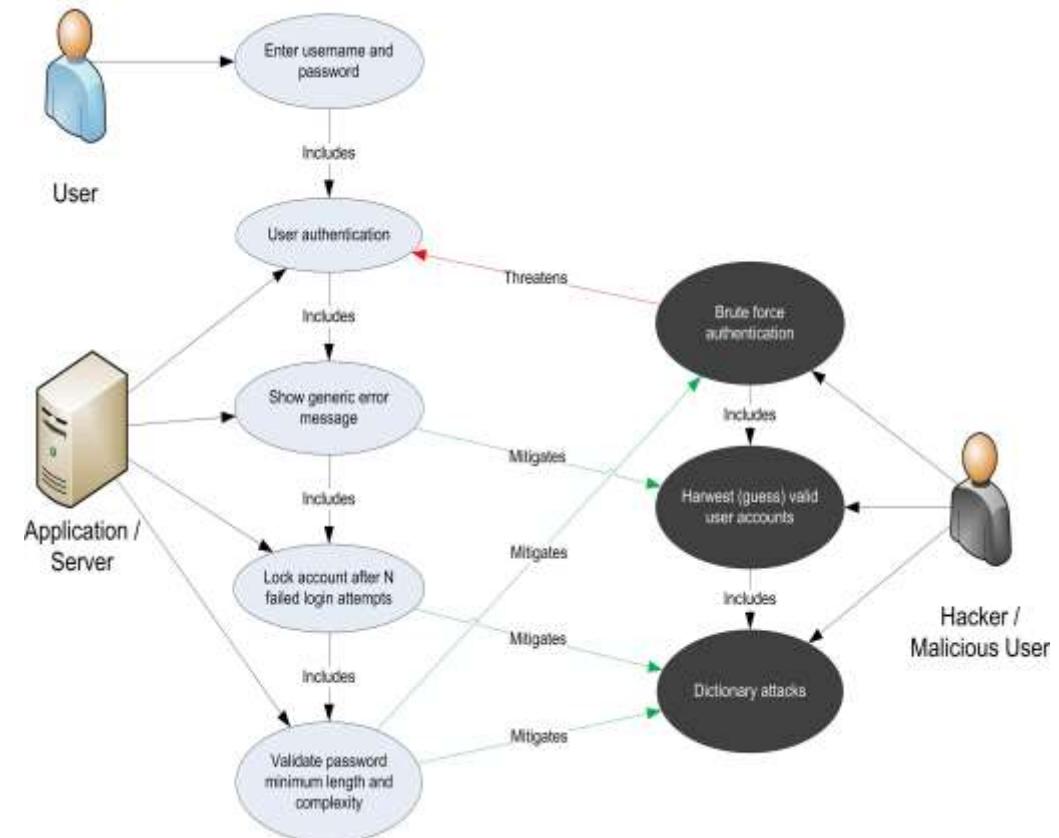
This is another way of looking at risk, albeit a bit simplified:

Vulnerability x Threat = Risk

We can sum up this calculation with the concepts from above: that a single vulnerability multiplied by the potential threat (frequency, existing safeguards, and potential value loss) can give you an estimate of the risk involved. In order for organizations to begin risk mitigation and risk management, you first need to understand your vulnerabilities and the threats to those vulnerabilities.

Threat Modelling

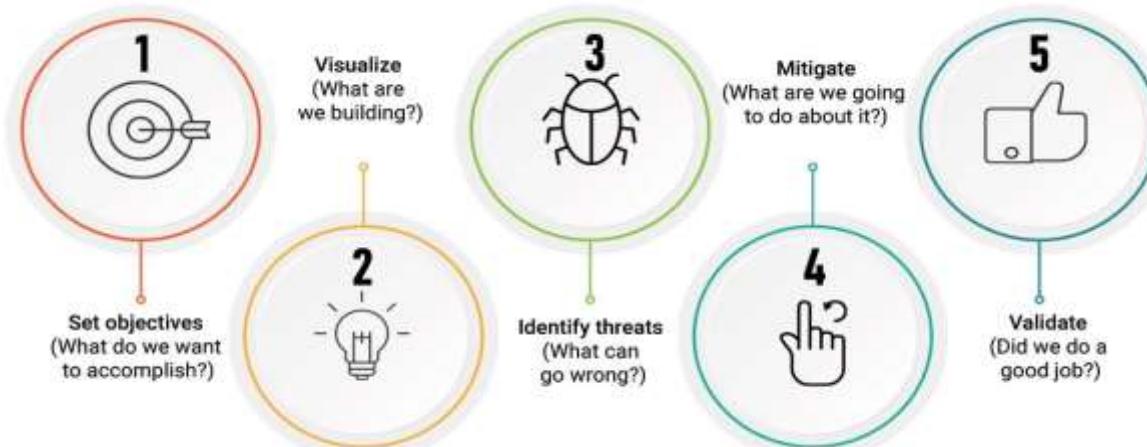
- is a structured representation of all the information that affects the security of an application.
- it is a view of the application and its environment through the lens of security.
- It contains
 - Description of the subject to be modeled
 - Assumptions that can be checked or challenged in the future as the threat landscape changes
 - Potential threats to the system
 - Actions that can be taken to mitigate each threat
 - A way of validating the model and threats, and verification of success of actions taken



Defining Threat Modelling

Threat Modeling is a systematic approach to identify, analyze, and mitigate potential threats and vulnerabilities in a system or application.

- It helps organizations proactively assess security risks and make informed decisions to protect their assets and data.
- Threat Modeling provides a structured framework for understanding the potential threats, their impact, and the countermeasures needed to address them.



Importance of Threat Modelling

Threat Modeling is a critical practice for ensuring the security and resilience of systems and applications. It provides several key benefits that contribute to the overall security posture of an organization.

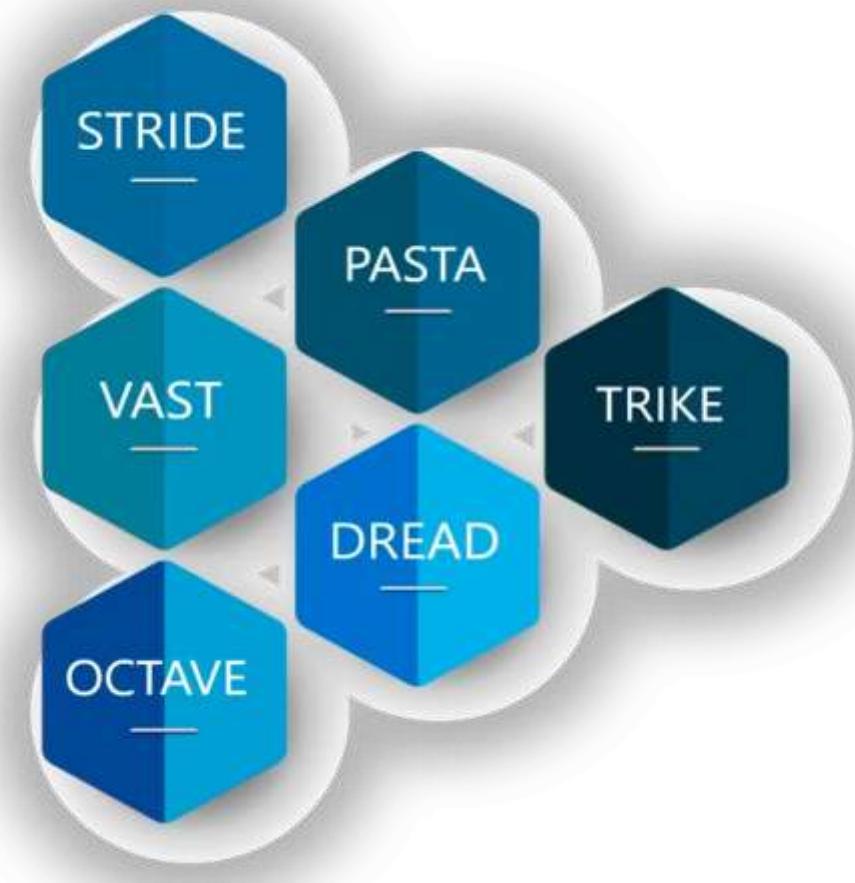
Early Risk Identification:	Prioritization of Security Efforts:	Improved Security Controls:	Integration of Security into the Development Process:	Compliance and Regulatory Requirements:
<ul style="list-style-type: none">• Threat Modeling helps identify potential security risks and vulnerabilities early in the development lifecycle.• By analyzing the system design and architecture, organizations can proactively address these risks before they manifest in real-world scenarios.• Early risk identification enables effective risk mitigation strategies and avoids costly security issues later in the development process.	<ul style="list-style-type: none">• Threat Modeling allows organizations to prioritize security efforts based on the severity and likelihood of identified threats.• By understanding the potential impact of each threat, resources can be allocated efficiently to address the most critical risks.• This ensures that security measures are focused on protecting the most valuable assets and critical components of the system.	<ul style="list-style-type: none">• Through Threat Modeling, organizations can identify and implement appropriate security controls and countermeasures.• By understanding the potential attack vectors and vulnerabilities, security measures can be tailored to mitigate specific threats effectively.• This leads to stronger security defenses, reducing the likelihood of successful attacks and minimizing the potential impact.	<ul style="list-style-type: none">• Threat Modeling promotes a security-focused mindset throughout the development lifecycle.• It encourages collaboration among developers, architects, and security professionals, fostering a shared responsibility for security.• By integrating security considerations from the early stages of development, security becomes an integral part of the system rather than an afterthought.	<ul style="list-style-type: none">• Threat Modeling assists organizations in meeting compliance and regulatory requirements.• It helps identify potential security gaps that may lead to non-compliance and enables the implementation of controls to address those gaps.• This ensures that the system meets the necessary security standards and regulatory obligations.

Approaches to Threat Modelling

Threat Modeling can be approached using various methodologies and techniques, each providing a different perspective on identifying and mitigating potential threats.

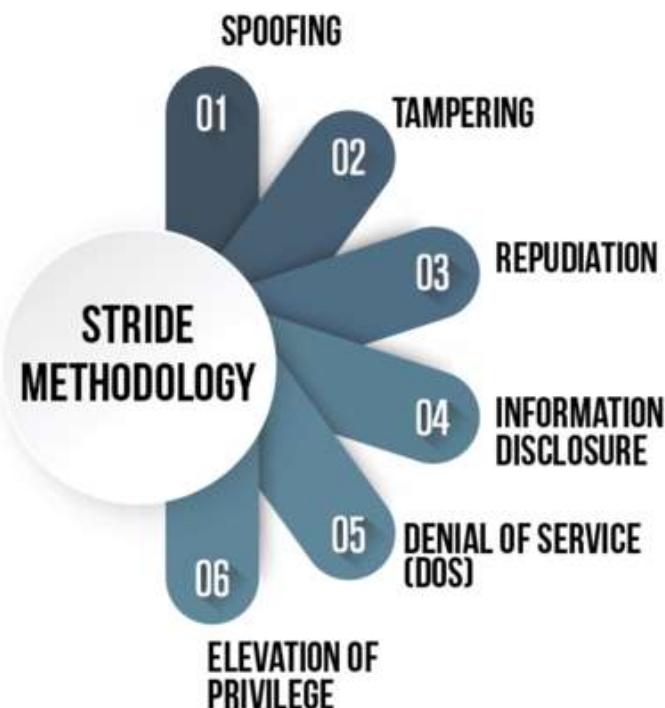
The chosen approach should consider the system's architecture, goals, and the expertise of the team involved in the Threat Modeling process.

Regardless of the approach, the key is to systematically analyze and prioritize threats to ensure that appropriate security measures are implemented to protect valuable assets and data.



Approaches to Threat Modelling: STRIDE

STRIDE is a mnemonic for a set of threats – Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service (DoS), and Elevation of Privilege as described in the table below.



	Type of Threat	What Was Violated	How Was It Violated?
S	Spoofing	Authentication	Impersonating something or someone known and trusted.
T	Tampering	Integrity	Modifying data on disk, memory, network, etc.,
R	Repudiation	Non-repudiation	Claim to not be responsible for an action
I	Information Disclosure	Confidentiality	Providing information to someone who is not authorized
D	Denial of Service (DoS)	Availability	Denying or obstructing access to resources required to provide service
E	Elevation of Privilege	Authorization	Allowing access to someone without proper authorization

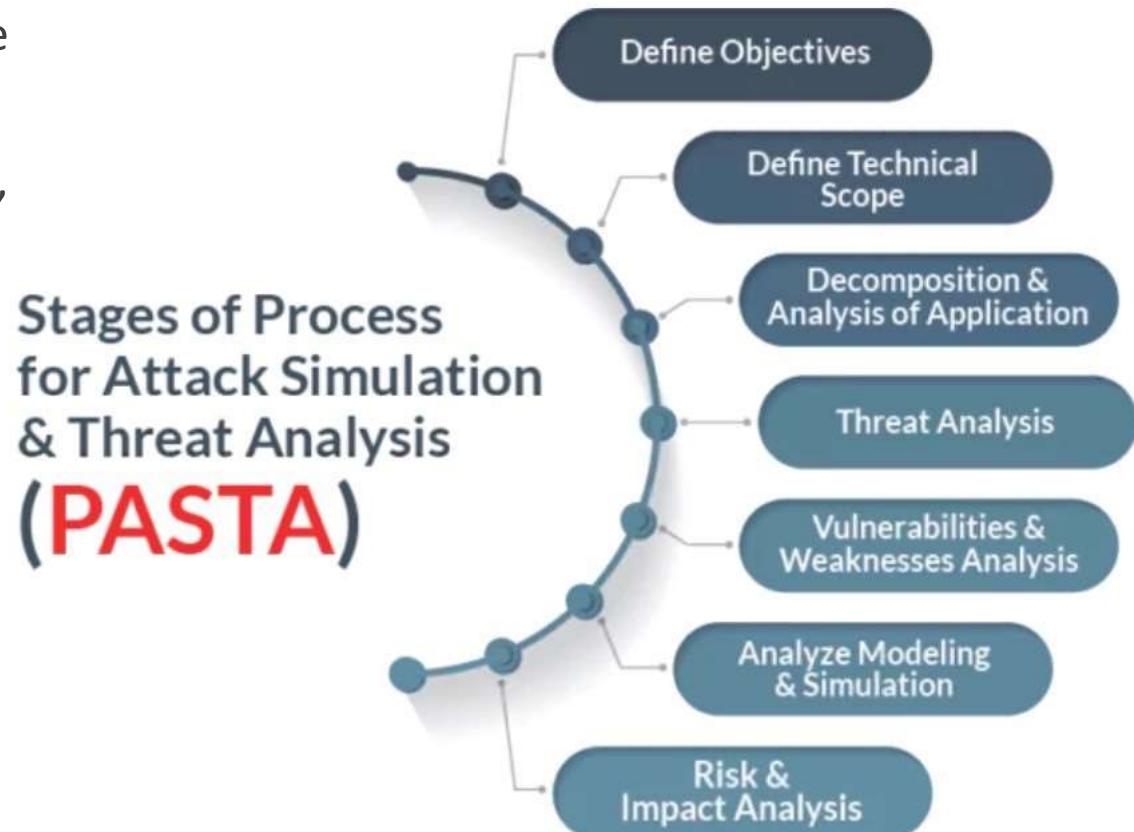
Approaches to Threat Modelling: PASTA

PASTA is a **seven-step methodology** to create a process for simulating attacks to IT applications, analyzing the threats, their origin, the risks they pose to an organization, and how to mitigate them.

The objective of this model is:

- to identify the threat
- enumerate them
- and assign a score

By following this method, the organization can determine the appropriate countermeasures that must be deployed to mitigate the risk.



Approaches to Threat Modelling: TRIKE

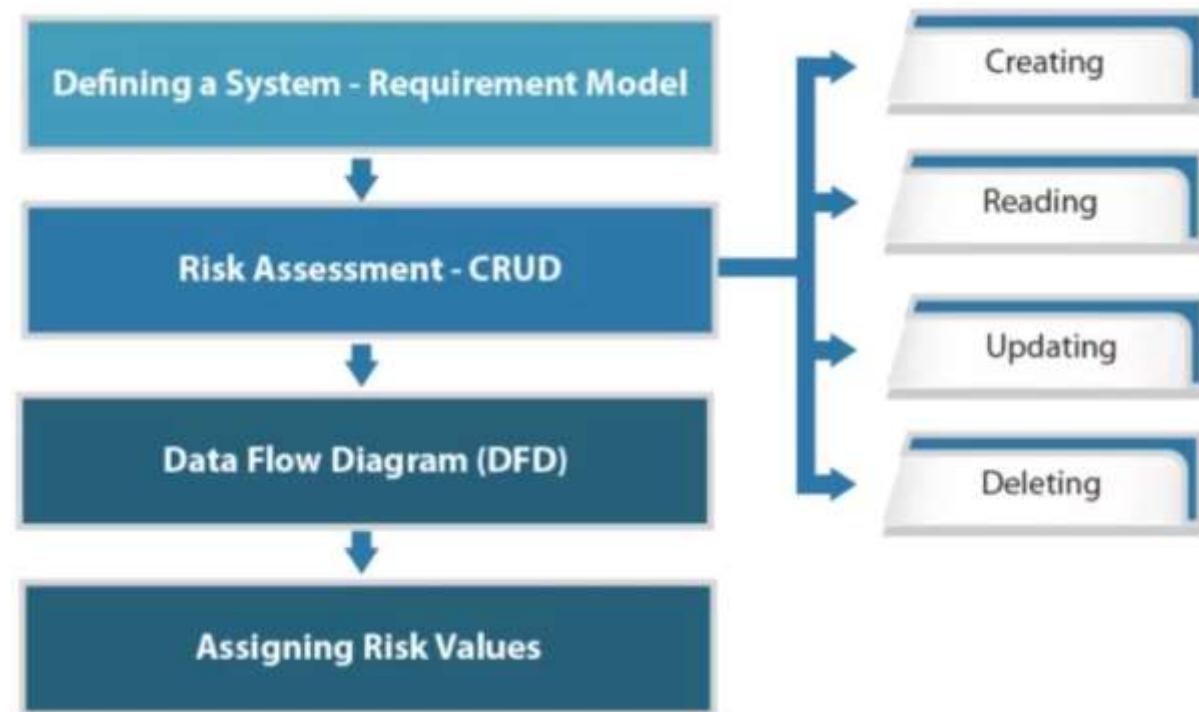
TRIKE is a that an open-source threat modeling methodology that is used for security auditing from a risk management perspective.

TRIKE threat modeling is a fusion of two models namely Requirement Model & Implementations Model.

The requirement model is the base of **TRIKE modeling** that explains the security characteristics of an IT system and assigns acceptable levels of risk to each asset.

The implementation model is a **Data Flow Diagram (DFD)** is created to illustrate the flow of data and the user performed actions within a system.

In this model, threats are analyzed to enumerate and assign a risk value. Based on this, security controls or preventive measures defined to address the threats as per the priority and assigned risks.



Approaches to Threat Modelling: VAST

VAST (Visual, Agile, and Simple Threat) methodology is based on automated threat modeling that covers the software development life cycle across the organization with proper integration with tools and collaboration with all key stakeholders like developers, architects, security professionals, and leaders across the organization.



Automation

- Eliminates Repetition in Threat Modeling
- Ongoing Threat Modeling
- Scaled to Encompass the Entire Enterprise

Integration

- Integration with Tools Throughout the SDLC
- Supports the Agile DevOps

Collaboration

Key Stakeholders Collaboration: App Developers, Systems Architects, Security Team, and Senior Executives

Approaches to Threat Modelling: DREAD

DREAD methodology is used to assess, analyze, and find the probability of risk by rating the threats as described in the image below.



Approaches to Threat Modelling: OCTAVE

OCTAVE (Operationally Critical Threat, Asset, and Vulnerability Evaluation) is an approach to identify, assess, and manage risks to IT assets. This process identifies the critical components of information security and the threats that could affect their confidentiality, integrity, and availability.



Role of Threat Modelling

1. Threat Modeling plays a critical role in **proactive cybersecurity** by identifying and **mitigating potential threats** before they can be exploited.
2. It enables organizations to **prioritize security efforts, design secure architectures, implement proactive countermeasures**, align with compliance requirements, and adapt to evolving threats.
3. By adopting Threat Modeling as a proactive approach, organizations can significantly enhance their overall cybersecurity posture and mitigate risks effectively.

Role of Threat Modelling

Risk Identification
and Prioritization

Designing Secure
Architectures

Enabling
Proactive
Countermeasures

Compliance and
Regulatory
Alignment

Continuous
Improvement and
Adaptation

Risk Assessment

Risk assessment takes a broader perspective by evaluating potential risks and their impact on the organization as a whole.

It involves determining the **likelihood of threats materializing**, estimating the potential impact, and **developing strategies to manage the identified risks**.

Key Components of Risk Assessment

1. Identifying Assets and Business Objectives:

The first step is to identify the assets, systems, and business objectives that require protection. This ensures that risks are assessed within the context of the organization's priorities.

2. Assessing Vulnerabilities and Threats:

An assessment is conducted to evaluate the vulnerabilities and threats that could impact the identified assets. This may involve analyzing historical data, industry reports, or engaging in threat intelligence activities.

Key Components of Risk Assessment

3. Determining Potential Impact:

The potential impact of the identified risks on the organization is evaluated. Factors such as financial loss, reputation damage, regulatory compliance, operational disruptions, and customer trust are considered.

4. Evaluating Likelihood and Probability:

In addition to assessing impact, the likelihood or probability of the identified risks materializing is evaluated. Historical data, industry trends, and expert opinions are taken into account to estimate how likely each risk is to occur.

5. Implementing Risk Mitigation Strategies:

Based on the evaluation of impact and likelihood, risk mitigation strategies are developed and implemented. These strategies may include controls, policies, procedures, or risk transfer mechanisms such as insurance.

Risk Assessment

In a risk assessment for an e-commerce company, potential risks might include credit card fraud, data breaches, or disruption of key services. The assessment would consider the potential impact of each risk, such as financial loss, reputational damage, and operational disruptions.

Mitigation strategies like encryption, secure payment gateways, regular security audits, and incident response planning would be implemented to minimize the identified risks.

Threat Modelling vs Risk Assessment

Aspect	Threat Modeling	Risk Assessment
Focus	Identifying and mitigating specific threats and vulnerabilities.	Evaluating potential risks and their impact on the organization as a whole.
Scope	Narrow – Focused on a specific system, application, or component.	Broad – Considers the organization's overall risks and objectives.
Time Horizon	Typically conducted during the development or change phase of a system.	Ongoing process that considers both current and future risks.
Implementation	Helps design and implement specific countermeasures to mitigate threats.	Guides the selection and implementation of strategies to manage risks.
Primary Objective	Identifying and prioritizing specific threats for proactive mitigation.	Assessing potential risks and their impact for effective risk assessment

Types of Risk Assessment Methodologies

Risk assessment can be either of two types:

Quantitative risk refers to the numerical value of the probability and potential impact of a threat. This type of risk assessment requires data collection and statistical analysis to arrive at those numbers.

Qualitative risk is more subjective, focusing on the characteristics of a threat rather than its numerical value. This type of risk assessment often uses expert opinion to arrive at ratings (usually a low/medium/high scale or something similar) for probability and potential impact.

PIA (Privacy Impact Assessment)

A privacy impact assessment (PIA) is an analysis of how personally identifiable information (PII) is handled to ensure compliance with appropriate regulations, determine the privacy risks associated with information systems or activities, and evaluate ways to reduce the privacy risks. A PIA is both an analysis and a formal document detailing the process and the outcome of the analysis.

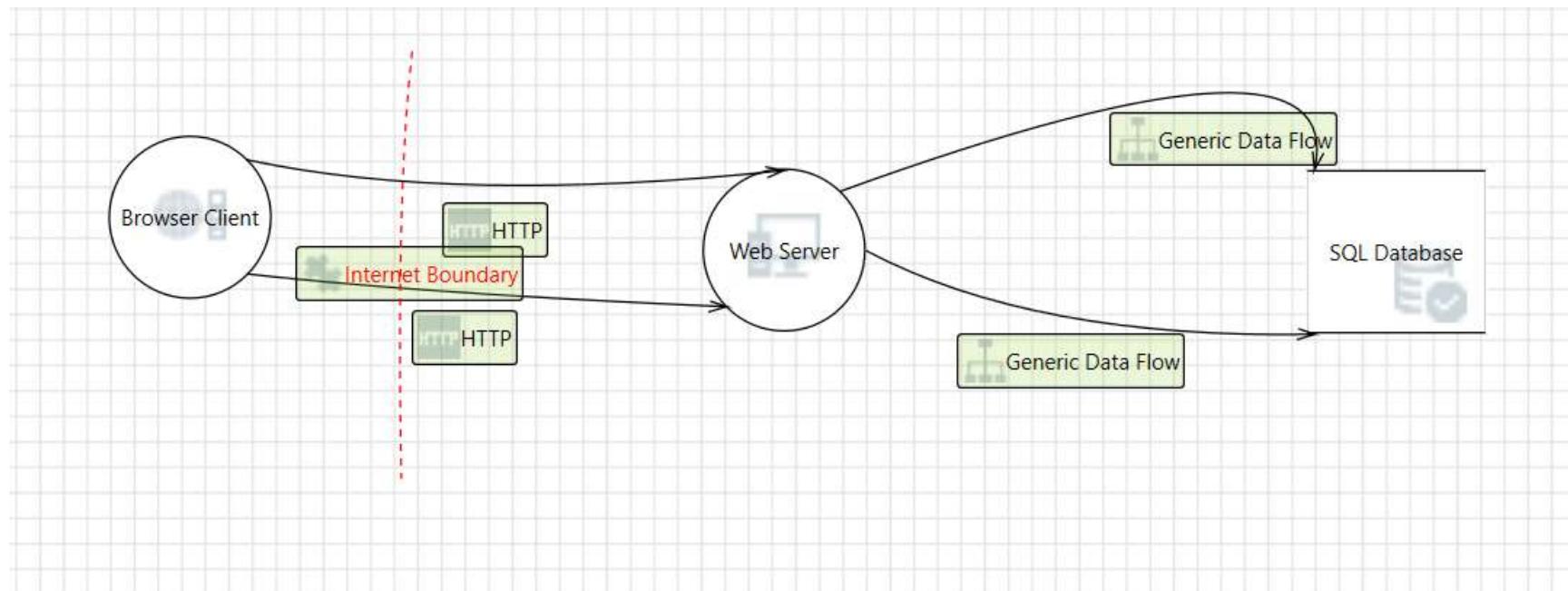
Threat Modelling

We can use Microsoft Threat Modelling Tool 2016 (<https://www.microsoft.com/en-in/download/details.aspx?id=49168>)

How to run a Threat Modelling Tool?

1. Creating a new threat model
2. Opening an existing threat model
3. Converting a threat model from TMT(.tm4) format to the new TMT (.tm7) format
4. Creating a new threat template
5. Opening/Modifying an existing threat template
6. Upgrade threat model to new template

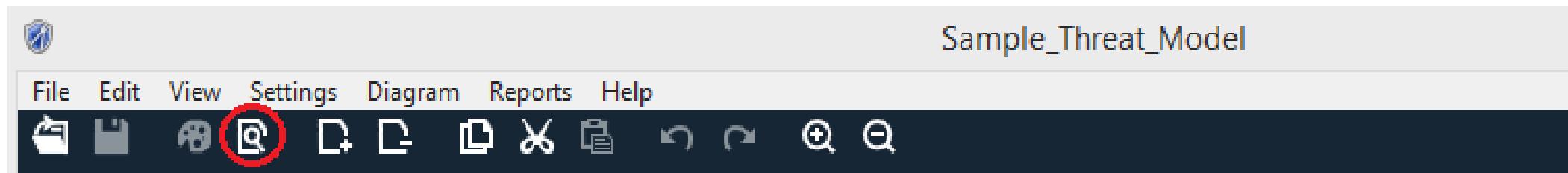
Data Flow Diagram (DFD)



Analyzing Threats

When you have completed your data flow diagram, switch to the Analysis view by using one of the following methods:

- From the View menu, select **Analysis View**.
- Click the **Analysis View** button on the toolbar.



Mitigation Information

For each of your threats, enter information about how to mitigate the threat:

1. Determine if the threat requires mitigation and categorize the mitigation by selecting one of the following options from the Threat Status dropdown list.
 - a) Not Started
 - b) Needs Investigation
 - c) Not Applicable
 - d) Mitigated

Threat Priorities

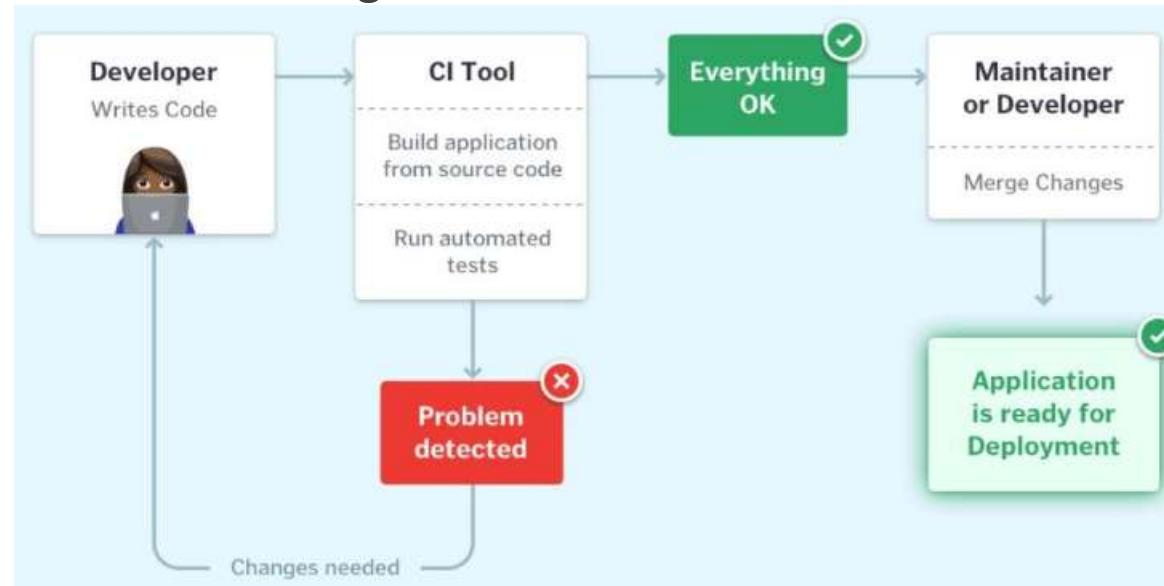
1. Select one of the following threat priorities from the Threat Category dropdown list.
 - a) High (default)
 - b) Medium
 - c) Low

Justification is required for threats in the **Mitigated** or **Not Applicable** states.

Module 4: Security Testing and Continuous Integration

Continuous Integration

- Continuous integration (CI) is the practice of automating the integration of code changes from multiple contributors into a single software project.
- It's a primary DevOps best practice, allowing developers to frequently merge code changes into a central repository where builds and tests then run. Automated tools are used to assert the new code's correctness before integration.



Continuous Integration Tools



Application Security

Application security is the process of developing, adding, and testing security features within applications to prevent security vulnerabilities against threats such as unauthorized access and modification.

- **Authentication**
- **Authorization**
- **Encryption**
- **Logging**
- **Application security testing**

Application Security Controls

Application security controls are techniques to enhance the security of an application at the **coding level**, making it less vulnerable to threats. Many of these controls deal with how the application responds to unexpected inputs that a cybercriminal might use to exploit a weakness.

A programmer can write code for an application in such a way that the programmer has more control over the outcome of these unexpected inputs. Fuzzing is a type of application security testing where developers test the results of unexpected values or inputs to discover which ones cause the application to act in an unexpected way that might open a security hole.

Application Security Testing

Application developers perform application security testing as part of the software development process to ensure there are no security vulnerabilities in a new or updated version of a software application.

A **security audit** can make sure the application is in compliance with a specific set of security criteria. After the application passes the audit, developers must ensure that only authorized users can access it.

In **penetration testing**, a developer thinks like a cybercriminal and looks for ways to break into the application. Penetration testing may include social engineering or trying to fool users into allowing unauthorized access.

Testers commonly administer both **unauthenticated security scans** and **authenticated security scans** (as logged-in users) to detect security vulnerabilities that may not show up in both states.

Software Testing

Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free.

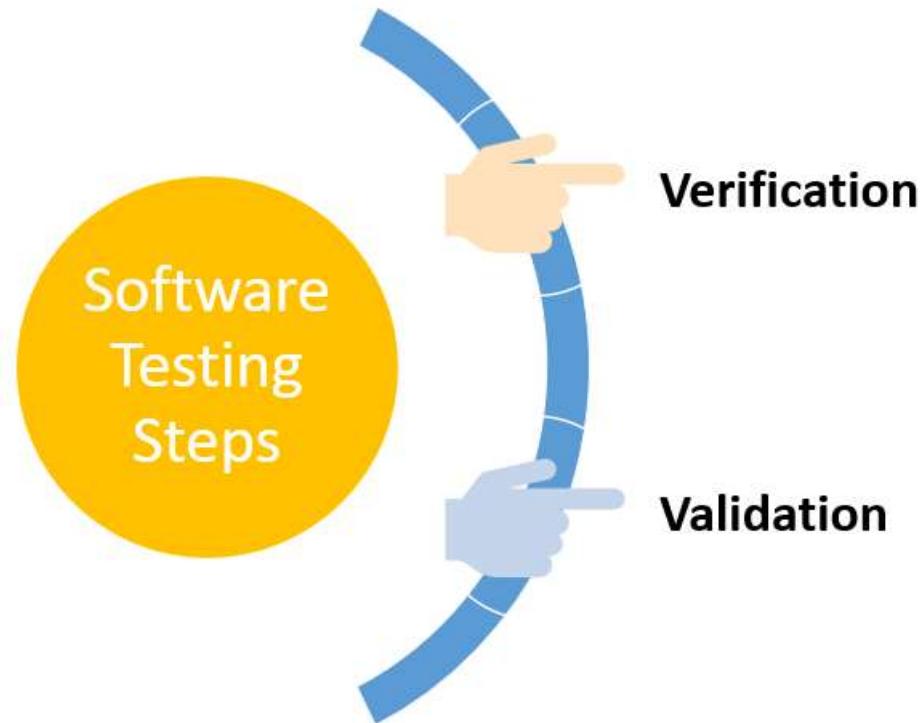
It involves execution of software/system components using **manual or automated tools** to evaluate one or more properties of interest.

The purpose of software testing is to **identify errors, gaps or missing requirements** in contrast to actual requirements.

Why Software testing is important?



Software Testing Steps



Software Testing Types?

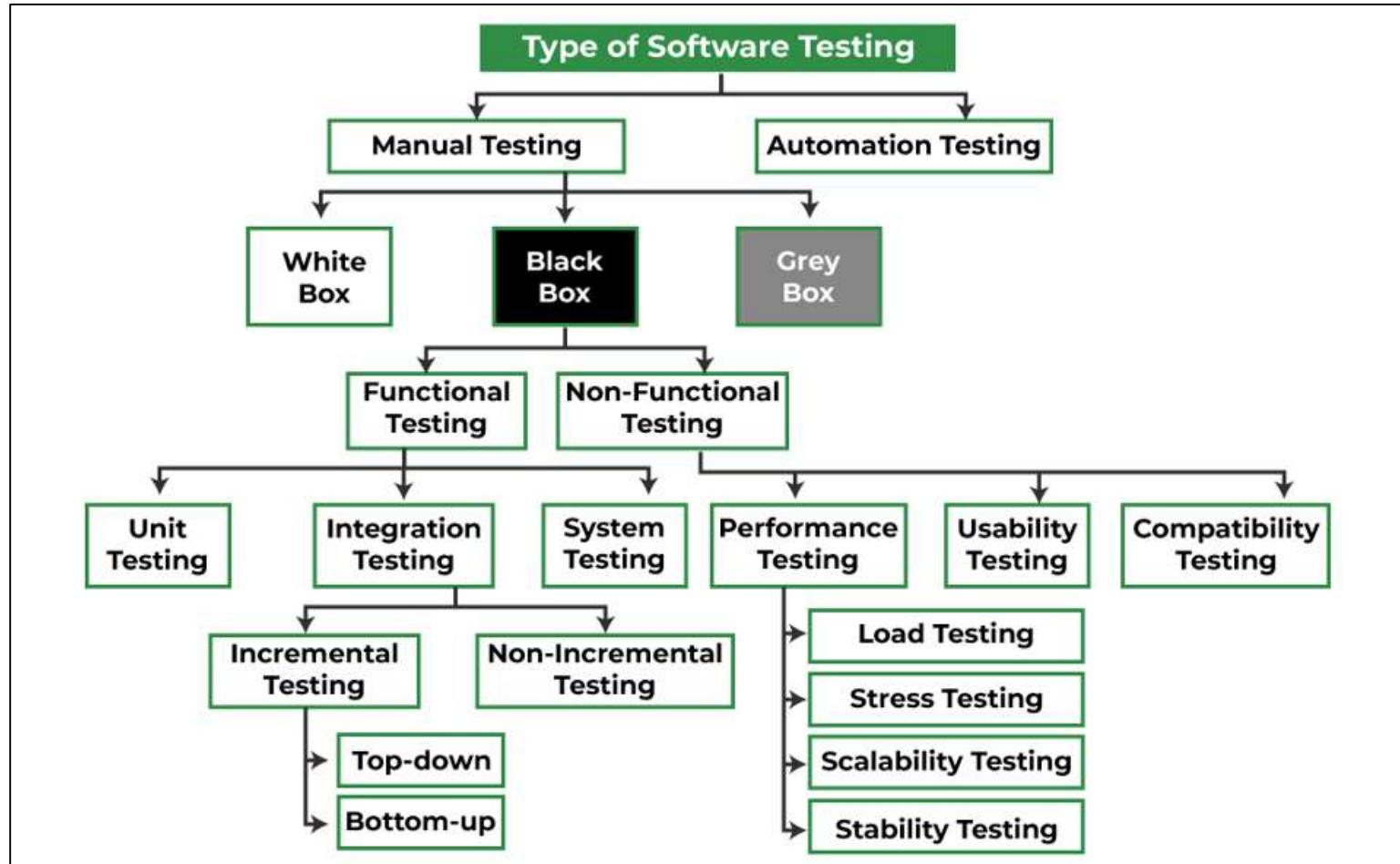
Software Testing can be broadly classified into 3 types:

1. Functional Testing

2. Non-functional Testing

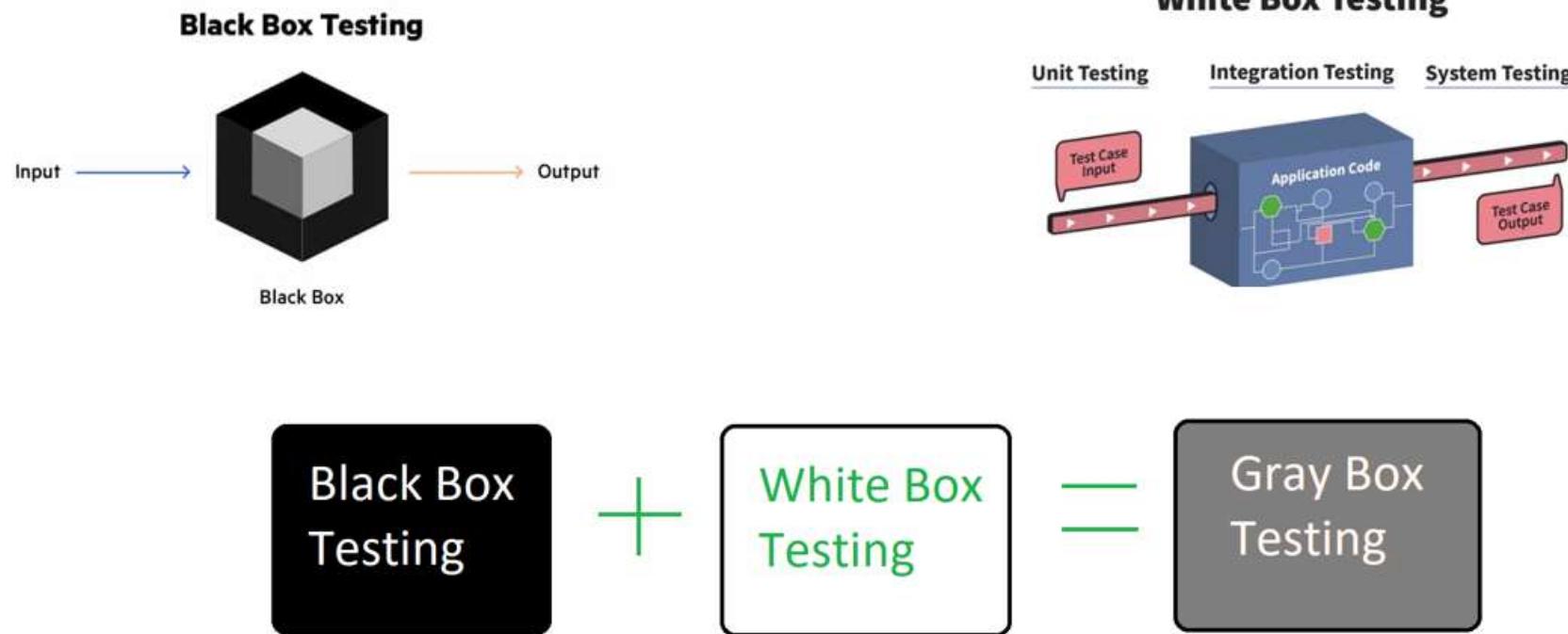
3. Maintenance Testing

Software Testing Types?



Software Testing Types?

Software testing techniques can be majorly classified into two categories:



Black Box vs White Box Testing

Black Box Testing	White Box Testing
The Black Box Test is a test that only considers the external behavior of the system; the internal workings of the software is not taken into account.	The White Box Test is a method used to test a software taking into consideration its internal functioning.
It is carried out by testers.	It is carried out by software developers.
This method is used in System Testing or Acceptance Testing.	This method is used in Unit Testing or Integration Testing.
It is the least time consuming.	It is most time consuming.
It is the behavior testing of the software.	It is the logic testing of the software.
It is also known as data-driven testing, functional testing, and closed box testing.	It is also known as clear box testing, code-based testing, structural testing, and transparent testing.
Black Box Test is not considered for algorithm testing.	White Box Test is well suitable for algorithm testing.

Different Levels of Software testing?

Software level testing can be majorly classified into 4 levels:

- 1. Unit Testing**
- 2. Integration Testing**
- 3. System Testing**
- 4. Acceptance Testing**

Static Testing and Dynamic Testing

- Static Testing involves testing the software without running it
- Dynamic Testing involves executing the program and testing its behavior in different scenarios.

Static Testing vs Dynamic Testing

Features	Static Testing	Dynamic Testing
Definition	A testing technique that reviews and analyzes software documentation, source code, and other artifacts before execution.	A testing technique that evaluates software behavior when it is executed.
Objective	To identify defects and improve software quality early in development.	To validate the software's functionality and behavior in different scenarios.
Types of Testing	Code review, walk-through, and inspection	Unit, integration, system, acceptance, performance, security, and user acceptance testing.
Timing	It begins early in the software development process and continues forever.	Conducted after coding and development are complete.
Results	Includes reviews, walkthroughs, and inspections of the software.	Includes functional, integration, system, performance, and usability tests

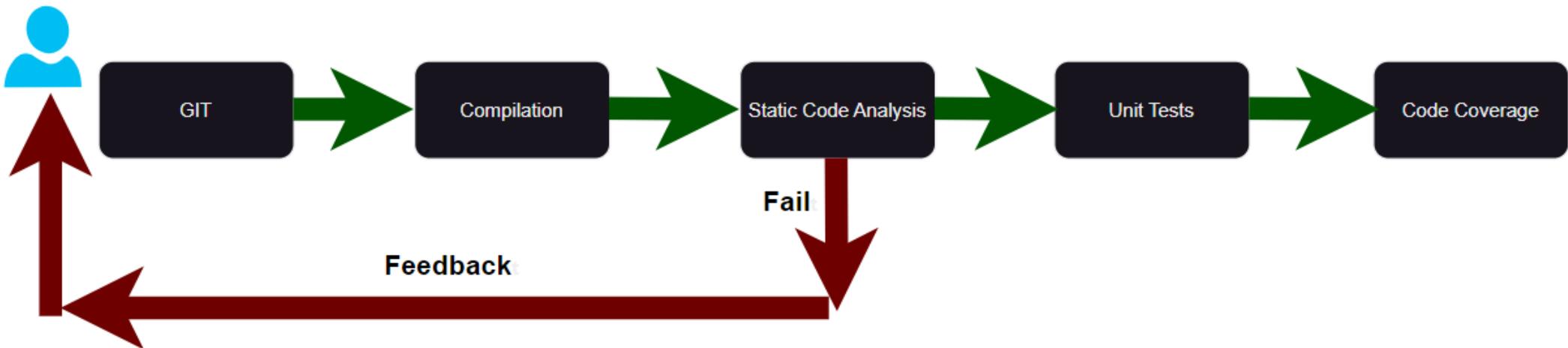
Static Testing Tools

Tool	Language Support	Description
ESLint	JavaScript, TypeScript	Popular tool for JavaScript and TypeScript, focusing on code quality and style.
Pylint	Python	A widely used static analysis tool for Python, emphasizing coding standards and bug detection.
FindBugs	Java	A Java-specific tool that identifies potential bugs and coding issues in Java code.
Cppcheck	C, C++	A tool for C and C++ code that helps identify bugs and coding style issues.
SonarQube	Multiple Languages	A comprehensive platform that supports various languages, offering code quality management, continuous inspection, and security analysis.

SAST

- **Static application security testing**
- Static code analysis is a software testing technique that involves examining the **source code** of a program without executing it.
- The **goal** is to find potential **vulnerabilities, security issues, and code quality** problems early in the development process.
- Static code analysis tools analyze the codebase without running the application and provide feedback to developers on potential issues, helping them identify and fix problems before the software is deployed.
- However, they do have limitations can't catch everything e.g. access control logic issues (authorization)
- They can have a lot of **false positives**
- It is a form of white-box testing (you can see everything)

SAST



Quality of Code

SAST tools are responsible to check the quality of the code before executing the code.

- **Bugs**

SAST tools may find bugs that can lead to security issues, but they might not cover all types of non-security-related bugs.

- **Maintainability**

SAST tools highlight issues such as hard-coded credentials, deprecated functions, or lack of input validation, nested conditions, comments avoided, which could affect maintainability indirectly.

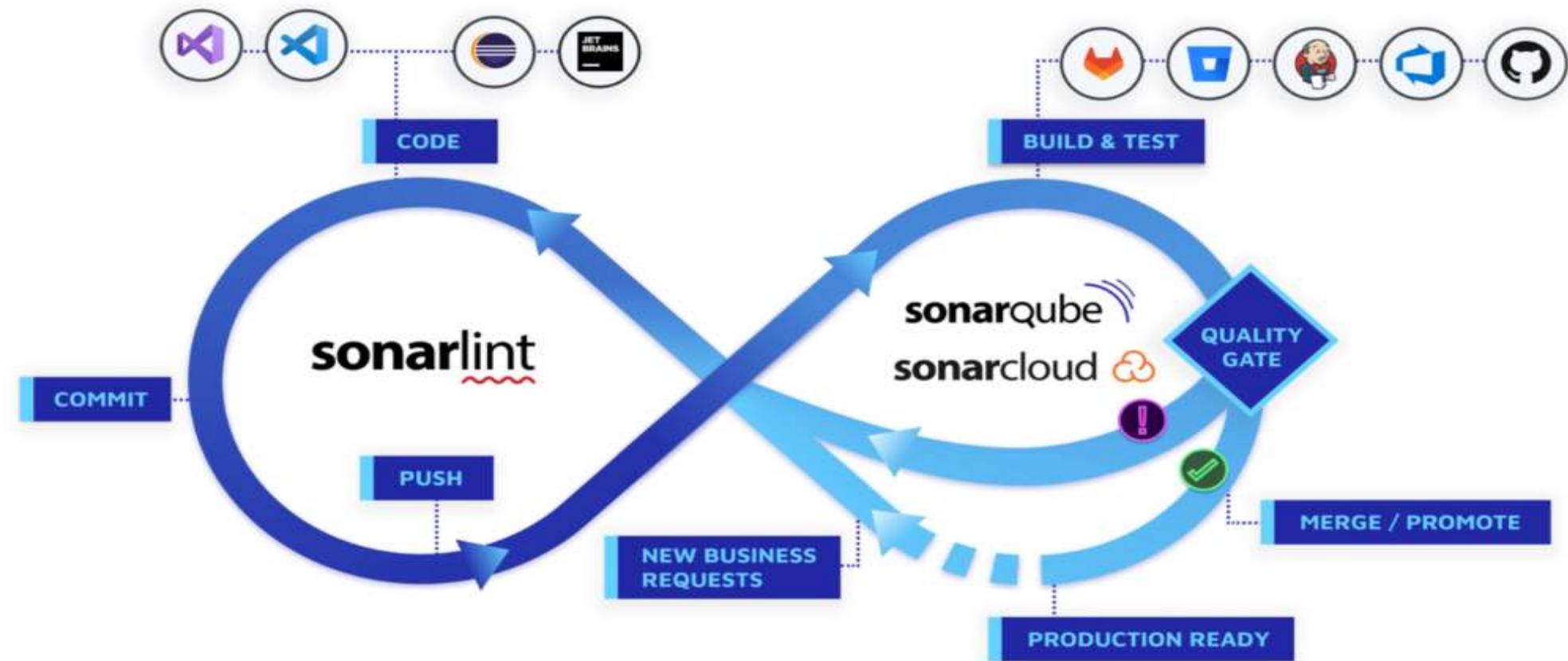
- **Security**

SAST tools can detect issues such as SQL injection, Cross-Site Scripting (XSS), buffer overflows, insecure coding practices, and other security-related flaws.

What is SonarQube

- It is a **Quality Management tool** and **Sonar Source** company introduced this tool.
- SonarQube is a **self-managed, automatic code review** tool that systematically helps you deliver Clean Code.
- SonarQube integrates into your existing workflow and detects issues in your code to help you perform continuous code inspections of your projects.
- The product analyses 30+ different programming languages and integrates into your Continuous Integration (CI) pipeline of DevOps platforms to ensure that your code meets high-quality standards.

What is SonarQube



What is SonarQube

The Sonar solution performs checks at every stage of the development process:

- **SonarLint** provides immediate feedback in your IDE as you write code so you can find and fix issues before a commit.
- **SonarQube's PR analysis** fits into your CI/CD workflows with SonarQube's PR analysis and use of quality gates.
- **Quality gates** keep code with issues from being released to production, a key tool in helping you incorporate the Clean as You Code methodology.
- The **Clean as You Code** approach helps you focus on submitting new, clean code for production, knowing that your existing code will be improved over time.

Releases

Community Edition	Developer Edition	Enterprise Edition	Data Center Edition
<ul style="list-style-type: none">▪ Static code analysis for 19 languages: Java, C#, JavaScript, TypeScript, CloudFormation, Terraform, Docker, Kubernetes, Kotlin, Ruby, Go, Scala, Flex, Python, PHP, HTML, CSS, XML, VB.NET and Azure Resource Manager▪ Detect Bugs & basic Vulnerabilities▪ Review Security Hotspots▪ Track Code Smells & fix your Technical Debt▪ Code Quality Metrics & History▪ CI/CD integration▪ Extensible, with 50+ community plugins	<ul style="list-style-type: none">▪ Support for C, C++, Obj-C, Swift, ABAP, T-SQL and PL/SQL▪ Detection of advanced vulnerabilities including Injection Flaws in Java, C#, PHP, Python, JavaScript, TypeScript▪ Detection of advanced bugs causing runtime errors and crashes in Python & Java▪ Analysis of feature and maintenance branches▪ Pull request analysis and decoration in the following DevOps platforms: GitHub, Bitbucket, Azure DevOps and GitLab	<ul style="list-style-type: none">▪ Support for Apex, COBOL, PL/I, RPG and VB6▪ Portfolio Management & PDF Executive Reports▪ Project PDF reports▪ Security Reports▪ Regulatory Reports to record state & quality of release▪ Project Transfer▪ Parallel processing of analysis reports	<ul style="list-style-type: none">▪ Component redundancy▪ Data resiliency▪ Horizontal scalability

Clean Code

Code is at the core of your software and dictates its behavior and performance. Clean Code ensures that your software works as intended and meets **high standards of quality**.

Definition:

- **Clean Code attribute:** A characteristic that contributes to Clean Code. Attributes are grouped under four main categories: **consistent, intentional, adaptable, and responsible**. Code is considered Clean Code when it has these attributes.
- **Software quality:** A characteristic of software that contributes to its lasting value. There are three software qualities: **security, reliability, and maintainability**.
- **Issue:** A problem in your code that prevents it from being Clean Code. Each issue is linked to one Clean Code attribute which is associated with one or more software qualities, each with a level of severity.

How it works in Sonar?

Issues in code are linked to code attributes.

When an issue is detected, it means that this part of your code is not **consistent**, **intentional**, **adaptable**, **or responsible** enough, and this impacts one or multiple software qualities.

Clean Code Attributes

Issues detected by the Sonar Scanner are impacted by Clean Code attributes that define code health. The characteristics of these attributes contribute to the improvement of your code.

Attributes are classified/grouped into four main categories:

- **Consistent**
- **Intentional**
- **Adaptable**
- **responsible**

Clean Code Attributes

1. Consistent :

The code is written in a uniform and conventional way. All the code looks similar and follows a regular pattern, even with multiple contributors at different times.

Consistent code is:

Formatted: The code presentation is systematic and regular. Non-semantic choices, such as spacing, indentation, and character placement, remain consistent throughout the codebase, maintaining uniformity across files and authors.

Conventional: The code performs tasks with expected instructions. Faced with equally good options, the code adheres to a single choice across all instances, preferring language conventions. This includes using the appropriate programming interfaces and language features.

Identifiable: The names follow a regular structure based on language conventions. The casing, word separators, suffixes, and prefixes used in the identifiers have purpose, without arbitrary differences.

Clean Code Attributes

2. Intentional:

The code is precise and purposeful. Every instruction makes sense, is adequately formed, and clearly communicates its behavior.

Intentional code :

Clear: The code is self-explanatory, transparently communicating its functionality.

Logical: It is free of explicit errors, contradictions, and commands that could be unpredictable or objectionable.

•**Complete:** The code is functional and achieves its implied goals. There are no obviously incomplete or lacking solutions.

•**Efficient:** The code utilizes resources without needless waste. It prioritizes economical options when available, avoiding unnecessary consumption of memory, processor, disk, or network resources.

Clean Code Attributes

3. Adaptable:

The code is structured to be easy to evolve and develop with confidence. It makes extending or repurposing its parts easy, and promotes localized changes without undesirable side-effects.

Adaptable code is:

Focused: The code has a single, narrow, and specific scope. Each unit should have only one concise purpose, without an overwhelming accumulation of instructions or excessive amounts of complexity.

- **Distinct:** The code procedures and data are unique and distinctive, without undue duplication.
- **Modular:** The relationships within the code are carefully managed, ensuring they are minimal and clearly defined.
- **Tested:** The code has automated checks that provide confidence in the functionality. It has enough test coverage which enables changes in implementation without the risk of functional regressions.

Clean Code Attributes

4. Responsible:

The code takes into account its ethical obligations on data, as well as societal norms.

Responsible code is:

Lawful: The code respects licensing and copyright regulation. It exercises the creator's rights and honors other's rights to license their own code.

- **Trustworthy:** The code abstains from revealing or hard-coding private information. It preserves sensitive private information such as credentials and personally-identifying information.
- **Respectful:** The code refrains from using discriminatory and offensive language. It chooses to prioritize inclusive terminology whenever an alternative exists that conveys the same meaning.

Software Qualities

Clean Code leads to software that is **secure, reliable, and maintainable**. These three aspects are called *software qualities* in the Sonar solution, and they contribute to the long-term value of your software.

When an issue is detected in your code, it affects one or more of the three software qualities with a varying level of impact.

Issues

SonarQube raises an issue every time a piece of code breaks a coding rule. The set of coding rules are defined by the associated **quality profile** for each language in the project.

SonarLint helps developers by performing local analyses to check code before pushing it back to the SCM. But in real life, it's not always possible to catch all the issues while you code.

There are three types of issues:

- **Bug:** A coding mistake that can lead to an error or unexpected behavior at runtime.
- **Vulnerability:** A point in your code that's open to attack.
- **Code Smell:** A maintainability issue that makes your code confusing and difficult to maintain.

Issue Severity

The severity of an issue is determined based on its impact on the software qualities and cannot be edited. There are three levels: **high, medium, and low**.

1. High

BLOCKER: Bug with a high probability to impact the behavior of the application in production. For example, a memory leak, or an unclosed JDBC connection are BLOCKERS that must be fixed immediately.

CRITICAL: Either a bug with a low probability to impact the behavior of the application in production or an issue that represents a security flaw. An empty catch block or SQL injection would be a CRITICAL issue. The code must be reviewed immediately.

Issue Severity

2. Medium

MAJOR: A quality flaw that can highly impact the developer's productivity. An uncovered piece of code, duplicated blocks, or unused parameters are examples of MAJOR issues.

3. Low

MINOR: A quality flaw that can slightly impact the developer's productivity. For example, lines should not be too long, and "switch" statements should have at least 3 cases, are both be considered MINOR issues.

INFO: Neither a bug nor a quality flaw, just a finding.

Issue Lifecycle

After creation, issues flow through a lifecycle, taking one of the following statuses:

- **Open**: set by SonarQube on new issues.
- **Confirmed**: set manually to indicate that the issue is valid.
- **Resolved**: set manually to indicate that the next analysis should Close the issue.
- **Reopened**: set automatically by SonarQube when a Resolved issue hasn't actually been corrected.
- **Closed**: set automatically by SonarQube for automatically created issues.

Resolutions

Closed issues will have one of the following resolutions:

Fixed: set automatically when a subsequent analysis shows that the issue has been corrected or the file is no longer available (removed from the project, excluded from a project, or renamed).

Removed: set automatically when the related rule is no longer available. The rule may not be available either because it has been removed from the Quality Profile or because the underlying plugin has been uninstalled.

Resolved issues will have one of the following resolutions:

False Positive: set manually.

Won't Fix: set manually.

Issue Workflow

Issues are automatically closed (Status: Closed) when:

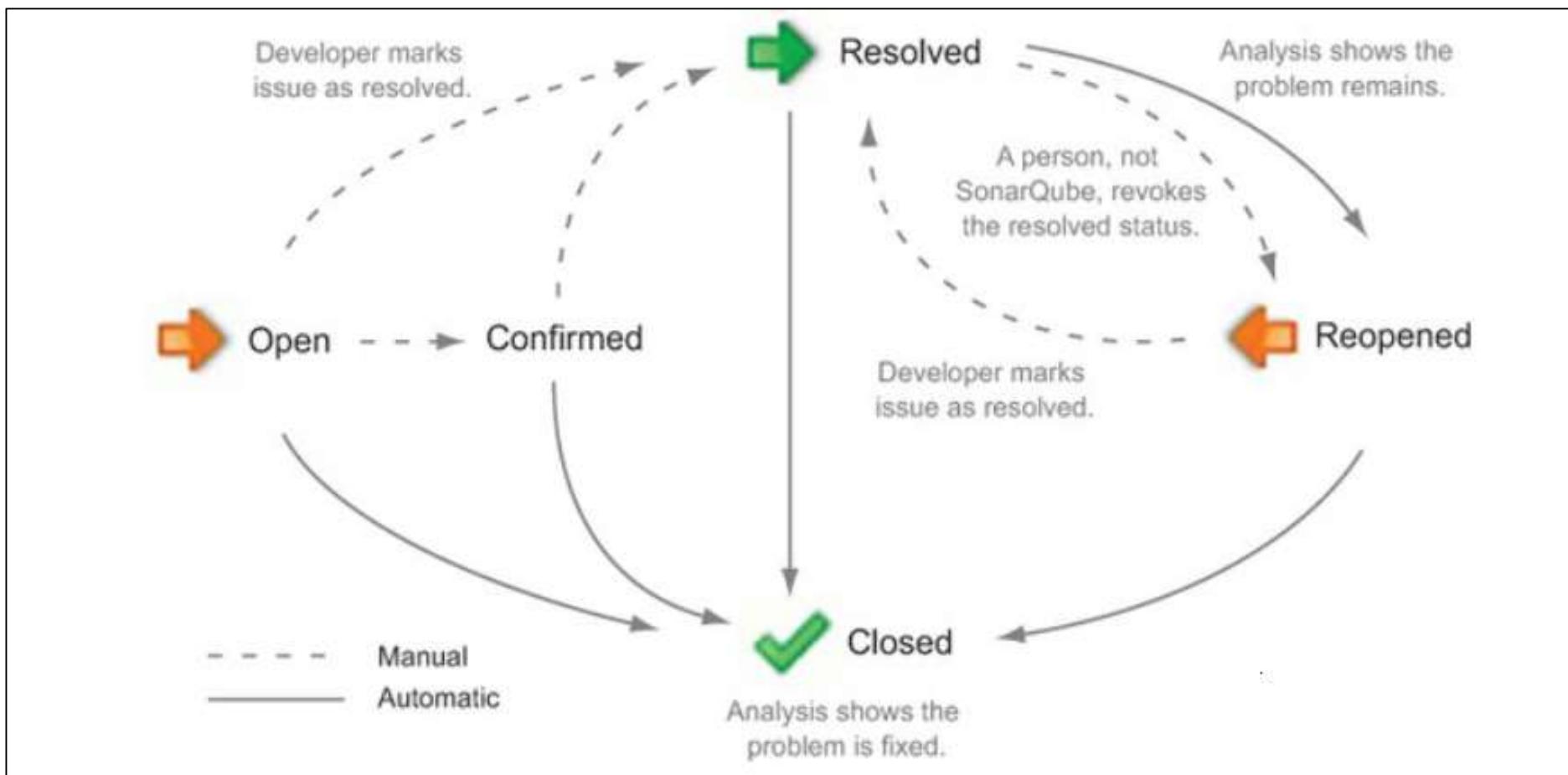
an issue of any status has been properly fixed = Resolution: Fixed

an issue no longer exists because the related coding rule has been deactivated or is no longer available (for example, the plugin has been removed) = Resolution: Removed

Issues are automatically reopened (Status: Reopened) when:

an issue that was manually Resolved as Fixed, and not labeled as False positive, is shown by a subsequent analysis to still exist.

Issue Workflow



Technical Review

Confirm: By confirming an issue, you basically move it out of "Open" status to Confirmed.

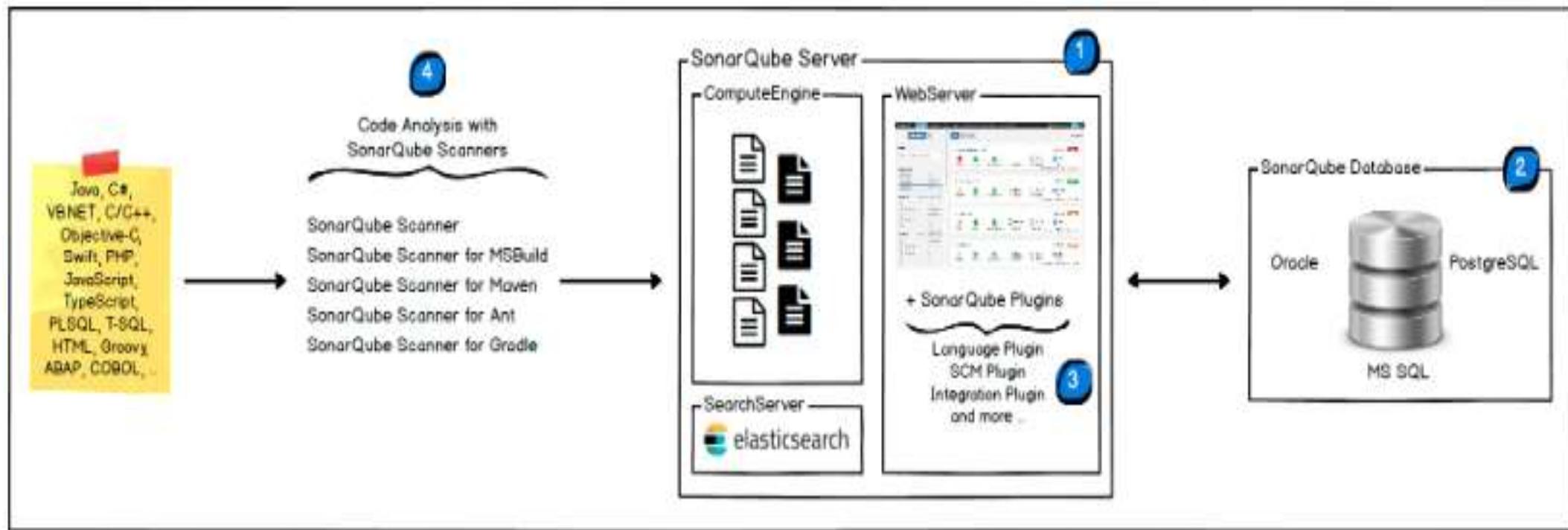
False Positive: Looking at the issue in context, you realize that for whatever reason, this issue isn't actually a problem. So you mark it as a False Positive and move on. This review assignment requires **Administer Issues permission** level on the project.

Severity change: This is the middle ground between the first two options. Yes, it's a problem but it's not as bad a problem as the rule's default severity makes it out to be. Or perhaps it's actually a far worse problem. Either way, you adjust the severity of the issue to bring it in line with what you feel it deserves. This review assignment requires **Administer Issues permission** level on the project.

Won't Fix: Looking at the issue in context, you realize that while it's a valid issue, it's not one that actually needs fixing. In other words, it represents accepted technical debt. So you mark it as Won't Fix and move on. This review assignment requires **Administer Issues permission** level on the project.

Resolve: If you think you've fixed an open issue, you can Resolve it. If you're correct, the next analysis will move it to closed status. If you're wrong, its status will go to re-opened.

SonarQube Architecture



How SonarQube works?

1. One SonarQube Server starting 3 main processes:

- Web Server for developers, managers to browse quality snapshots and configure the SonarQube instance. It is designed in JAVA, HTML, CSS and JS.
- Search Server based on **Elasticsearch** to back searches from the UI
- Compute Engine Server in charge of processing code analysis reports and saving them in the SonarQube Database

2. One SonarQube Database to store:

- the configuration of the SonarQube instance (security, plugins settings, etc.)
- the quality snapshots of projects, views, etc.
- **Default is H2 database.** We can also use Oracle, PostgreSQL, MySQL etc

3. Multiple SonarQube Plugins installed on the server, possibly including language, SCM, integration, authentication, and governance plugins

4. One or more SonarScanners running on your Build / Continuous Integration Servers to analyze projects

How SonarQube works?

1. Developers code in their IDEs and use [SonarLint](#) to run local analysis.
2. Developers push their code into their favourite SCM : git, SVN, TFVC, ...
3. The Continuous Integration Server triggers an automatic build, and the execution of the SonarScanner required to run the SonarQube analysis. The scanner fetches the rules from SonarQube server and apply on the code.
4. The analysis report is sent to the SonarQube Server for processing.
5. SonarQube Server processes and stores the analysis report results in the SonarQube Database, and displays the results in the UI.
6. Developers review, comment, challenge their Issues to manage and reduce their Technical Debt through the SonarQube UI.
7. Managers receive Reports from the analysis. Ops use APIs to automate configuration and extract data from SonarQube. Ops use JMX to monitor SonarQube Server.

SonarQube Prerequisites

- A small scale instance requires at least **2GB of RAM**
- Disk space depends on how much code you analyze with **SonarQube**
- It should be installed on hard drives having excellent read and write performance
- SonarQube and SonarScanner support only 64 bit systems
- It requires Java version 17.

SonarQube Prerequisites

If you're running on Linux, you must ensure that:

- **vm.max_map_count** is greater than or equal to **524288**
- **fs.file-max** is greater than or equal to **131072**
- the user running SonarQube can open at least **131072** file descriptors
- the user running SonarQube can open at least **8192** threads

SonarQube Prerequisites

Set the values in /etc/sysctl.conf file

/etc/sysctl.conf

vm.max_map_count=524288

fs.file-max=131072

ulimit -n 131072

ulimit -u 8192

SonarQube installation

1. Download the community edition from official website.

<https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-10.3.0.82913.zip>

2. Unzip it

```
unzip sonarqube-10.3.0.82913.zip
```

```
mv sonarqube-10.3.0.82913 /opt/sonarqube
```

3. Install java (apt install openjdk-11-jdk)

4. Run the package

```
sh /opt/sonarqube/bin/linux-x86-64/sonar.sh start
```

```
sh /opt/sonarqube/bin/linux-x86-64/sonar.sh status
```

Note: install it as a non-root user

SonarQube installation using Docker

- Install docker
- Create a SonarQube container using below command.

```
docker run -dt -p 9000:9000 --name sonarqube sonarqube
```

Access the container using the machine's **PublicIP:9000** in a browser.

Properties

Server: sonar.properties

Scanner: sonar-scanner.properties

Code: sonar-project.properties

SonarScanner

The SonarScanner is the scanner to use when there is no specific scanner for your build system. The SonarScanner does not support ARM architecture.

Installation:

1. Download the package:

Wget <https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-5.0.1.3006-linux.zip>

2. Unzip and set the path

```
unzip sonar-scanner-cli-5.0.1.3006-linux.zip
```

```
mv sonar-scanner-5.0.1.3006-linux /opt/sonar-scanner
```

```
export PATH=$PATH:/opt/sonar-scanner/bin
```

```
source ~/.bashrc
```

SonarScanner Configuration File

Set the SonarQube path , by default SonarQube works on port 9000

```
vi /opt/sonar-scanner/conf/sonar-scanner.properties
```

```
#Configure here general information about the environment, such as SonarQube server  
connection details for example  
#No information about specific project should appear here
```

```
#----- Default SonarQube server  
sonar.host.url=http://localhost:32768
```

```
#----- Default source code encoding  
#sonar.sourceEncoding=UTF-8
```

SonarQube Project

Create a Project on SonarQube:

The screenshot shows the 'Create a local project' page on the SonarQube interface. The top navigation bar includes links for Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, and More. The main section is titled 'Create a local project'. It contains three input fields: 'Project display name *' with value 'my-project', 'Project key *' with value 'my-project', and 'Main branch name *' with value 'main'. Each input field has a green checkmark icon to its right. Below each input field is a descriptive message. A 'Next' button is located at the bottom left.

sonarqube

Projects Issues Rules Quality Profiles Quality Gates Administration More

Create a local project

Project display name *

my-project

Up to 255 characters. Some scanners might override the value you provide.

Project key *

my-project

The project key is a unique identifier for your project. It may contain up to 400 characters. Allowed characters are alphanumeric, '-' (dash), '_' (underscore), '.' (period) and ':' (colon), with at least one non-digit.

Main branch name *

main

The name of your project's default branch [Learn More](#)

Next

SonarQube Project

Create a Key for this project

Administrator → project → key

This key will be used by Sonar-scanner to connect with SonarQube.

SonarQube Project

Go to My Account → Security

The screenshot shows the SonarQube web interface with the user 'Administrator' logged in. The 'Security' tab is selected in the navigation bar. A yellow box highlights the 'Generate Tokens' section, which includes fields for 'Name' (Enter Token Name), 'Type' (Select Token Type), 'Expires in' (30 days), and a 'Generate' button. A success message states: 'New token "jenkins" has been created. Make sure you copy it now, you won't be able to see it again!' with a copy icon. Below this, a table lists existing tokens:

Name	Type	Project	Last use	Created	Expiration
jenkins	Project	pet	Never	February 6, 2024	March 7, 2024

A yellow box also highlights the 'My Account' link in the top right corner of the header.

Sonar-scanner Config file

Create a config file under the root directory of our code.

sonar-project.properties

```
# must be unique in a given SonarQube instance
sonar.projectKey=my-project
# --- optional properties ---
# defaults to project key
sonar.projectName=my-project
sonar.token=sqp_e1b901c84ea5f9c688fd20f2de8803ca2bc29a1e
# defaults to 'not provided'
#sonar.projectVersion=1.0
# Path is relative to the sonar-project.properties file. Defaults to .
#sonar.sources=.
sonar.java.binaries=.
sonar.exclusions=src/main/resources/**/*.*java
# Encoding of the source code. Default is default system encoding
#sonar.sourceEncoding=UTF-8
```

Sonar-scanner

To run it use the below command under our code directory.

Sonar-scanner

It will use the **sonar-project.properties** to scan the code.

Or

Use below command

```
sonar-scanner      -Dsonar.projectKey=my-project      -Dsonar.sources=.      -Dsonar.host.url=http://3.19.62.29:32768      -  
Dsonar.token=sq..._e1b901c84ea5f9c688fd20f2de8803ca2bc29a1e
```

SonarQube

✓ Quality Gate Passed



Enjoy your sparkling clean code!

New Code Overall Code

Reliability	Maintainability
41 Bugs (C)	43 Code Smells (A)
Security	Security Review
1 Vulnerabilities (E)	2 Security Hotspots (E)
Coverage	Duplications
0.0% Coverage (O)	0.0% Duplications (G)
Coverage on 519 Lines to cover	Duplications on 2.6k Lines

Rules

- SonarQube is a tool where all the rules are stored to check the code quality.
- Rules are related to all the programming languages.
- Rules can be added as plugins which can be free or paid.

To add more rules :

Administration → Marketplace → Plugins

Install the plugin and restart Sonar

Or download the latest plugin

Place jar file to your sonar instance to "[YOUR SONAR PATH]/extensions/plugins", restart Sonar

Rules

SonarQube executes rules on source code to generate issues. There are four types of rules:

- **Code smell (maintainability domain)**
- **Bug (reliability domain)**
- **Vulnerability (security domain)**
- **Security hotspot (security domain)**

Rules status

Rules can have 3 different statuses:

Beta: The rule has been recently implemented and we haven't gotten enough feedback from users yet, so there may be false positives or false negatives.

Deprecated: The rule should no longer be used because a similar, but more powerful and accurate rule exists.

Ready: The rule is ready to be used in production.

Rule Details

- This rule shows “**Why is this an issue?**”
- It shows the profiles the rule is active in and how many open issues have been raised with it.
- We can add existing tags on a rule. Some rules have built-in tags which can not be removed.
- We can extend rule descriptions to let users know how your organization is using a particular rule.

Rule Templates and Custom rules

Rule templates are provided by plugins as a basis for users to define their own custom rules in SonarQube. Select Show Templates Only option from the Template dropdown menu:

To create a custom rule from a template select Create next to the Custom rules heading and enter the following information:

- Name
- Key (auto-suggested)
- Type
- Severity
- Status
- Description (Markdown format is supported)
- The parameters specified by the template

Security Related Rules

Security-injection rules: There's a vulnerability here when the inputs handled by your application are controlled by a user (potentially an attacker) and not validated or sanitized. When this occurs, the flow from sources to sinks (sensitive functions) will be presented. SonarQube uses well-known taint analysis technology on source code which allows, the detection of:

- **CWE-89: SQL Injection**
- **CWE-79: Cross-site Scripting**
- **CWE-94: Code Injection**

Security-configuration rules: Here there is a security issue because when calling a sensitive function, the wrong parameter (for example invalid cryptographic algorithm or TLS version) has been set or when a check (for example, a `check_permissions()` kind of function) was not done or not in the correct order, this problem is likely to appear often when the program is executed (no injected/complex attacks are required unlike in the previous category):

- **CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag**
- **CWE-297: Improper Validation of Certificate with Host Mismatch**
- **CWE-327: Use of a Broken or Risky Cryptographic Algorithm**

Security Hostspots

Security issues are then divided into two categories: **vulnerabilities and hotspots**

A security hotspot **highlights a security-sensitive piece of code** that the developer needs to review. Upon review, you'll either find there is no threat or you need to apply a fix to secure the code.

With hotspots, we want to help developers understand information security risks, threats, impacts, root causes of security issues, and the choice of relevant software protections.

Why are Security Hotspots important?

Understand the risk: Understanding when and why you need to apply a fix in order to reduce an information security risk (threats and impacts).

Identify protections: While reviewing hotspots, you'll see how to avoid writing code that's at risk, determine which fixes are in place, and determine which fixes still need to be implemented to fix the highlighted code.

Identify impacts: With hotspots, you'll learn how to apply fixes to secure your code based on the impact on overall application security. Recommended secure coding practices are included on the hotspots page to assist you during your review.

Vulnerability or Hotspot

The main difference between a hotspot and a vulnerability is the need for review before deciding whether to apply a fix:

- With a **hotspot**, a security-sensitive piece of code is highlighted, but the overall application security may not be impacted. It's up to the developer to review the code to determine whether or not a fix is needed to secure the code.
- With a **vulnerability**, a problem that impacts the application's security has been discovered and needs to be fixed immediately.

Quality Profile

Quality profiles define the **set of rules** to be applied during code analysis.

Every project has a **quality profile** set for each supported language. When a project is analyzed, SonarQube determines which languages are used and uses the active quality profile for each of those languages in that specific project.

Steps:

- Create a quality profile or copy a quality profile to create a new one
- Activate the rules for the profile and attach with the project

Java, 1 profile(s)	Projects ?	Rules	Updated	Used
Sonar way BUILT-IN	DEFAULT	485	1 day ago	4 hours ago
JavaScript, 1 profile(s)	Project			

Create a new quality profile as a replica of "Sonar way". The two profiles will then evolve independently.

- Extend
- Copy

Quality Profile

Built-in and default profiles

SonarQube comes with a built-in quality profile defined for each supported language, called the **Sonar way** profile. The Sonar way activates a set of rules that should be applicable to most projects.

In a newly set up instance, the Sonar way profile is the default for every language (marked with the DEFAULT tag in the interface). The default profile is used for that language if no other profile is explicitly defined at the project level. The default profile for a given language can be changed

Quality Profile

If you have multiple projects, you might also need to have different profiles for each. You might run into the following situations

- You have different technical requirements from one project to another.
- You want to ensure stronger requirements for some of your projects than for others.

New profiles can be created in two ways:

- Copying an existing profile and adjusting the copy.
- Extending an existing profile.

When you extend a profile, you create a child profile that inherits all the activated rules in the parent profile. You can then activate additional rules in the child, beyond those that are inherited.

Quality Profile

Copying a Quality Profile:

When you copy a profile, you clone all activated rules of the original. From here, you independently activate or deactivate rules to fit your needs; your new profile won't inherit changes made to the original profile.

Extending a Quality Profile:

When you extend a profile, you create a child profile that inherits all the activated rules in the parent profile. You can then activate additional rules in the child, beyond those that are inherited.

With an extension, you can only activate rules that are deactivated in the parent. With a copy, you can activate or de-activate any rules you like.

Quality Profile

We can change the severity details of the rules attached to a new profile.

Quality Profile → Select the profile → active rules → access the rule → change

Quality Profiles

[Activate](#)

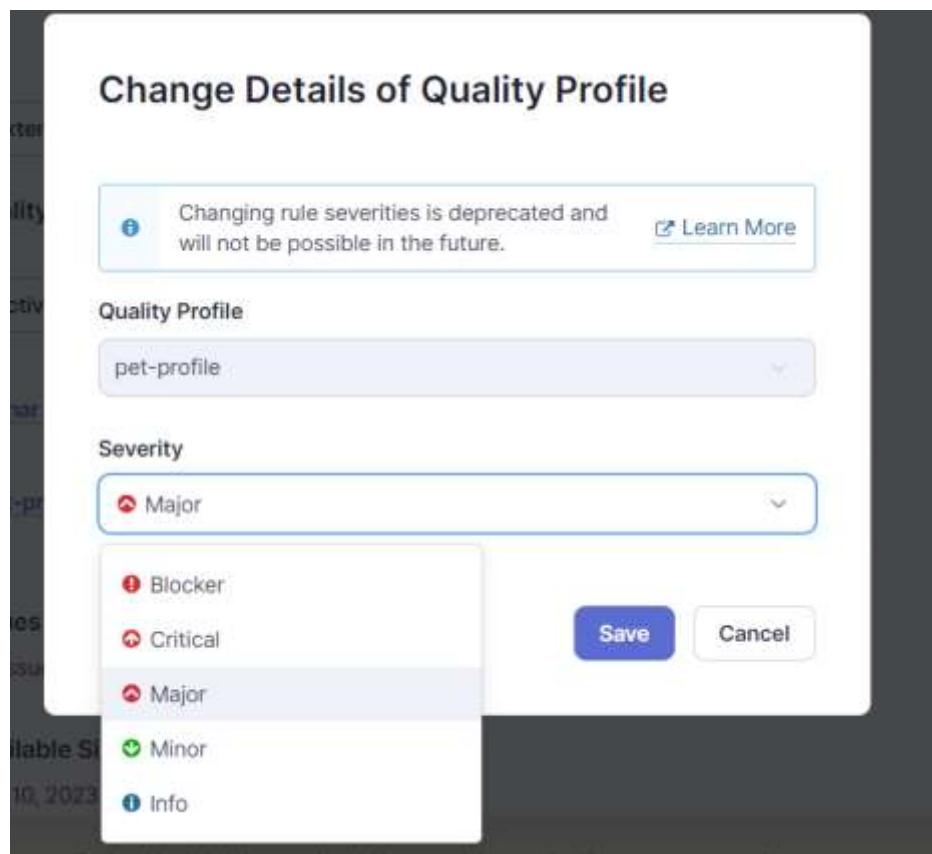
[Sonar way](#) BUILT-IN

[pet-profile](#)

[Change](#) [Deactivate](#)

Quality Profile

Change the severity level



Quality Gates

you might not require the **same code coverage on new code** for web applications as you would for Java applications

You want to ensure stronger requirements on some of your applications

Quality gates enforce a quality policy in your organization by answering one question: is my project ready for release?

You define a set of conditions against which projects are measured. For example:

- No new issues
- Code coverage on new code greater than 80%

Quality Gates

Quality Gates ? [Create](#)

Sonar way	DEFAULT BUILT-IN
Sonar way (legacy)	
pet-quality-gate	⚠

Conditions ? [Add Condition](#)

Conditions on New Code

Metric	Operator	Value	Actions
Issues	is greater than	1	Edit Delete
Security Hotspots Reviewed	is less than	0.0%	Edit Delete
Coverage	is less than	50.0%	Edit Delete
Duplicated Lines (%)	is greater than	2.0%	Edit Delete

Conditions on Overall Code

Metric	Operator	Value	Actions
Bugs	is greater than	1	Edit Delete

Quality Gates

Run code analysis and check the result.

Quality Gate Status  Failed 1 failed condition

41  Bugs is greater than 1

⚠️ The quality gate used by this project does not comply with Clean as You Code.
Fixing [this quality gate](#) will help you achieve a Clean Code state.

[Learn why](#) 

Measures

New Code Overall Code  1 failed condition

 Reliability 41 Bugs 	 Maintainability 43 Code Smells 
 Security 1 Vulnerabilities 	 Security Review 2 Security Hotspots 
 Coverage 0.0% Coverage Coverage on 519 Lines to cover 	 Duplications 0.0% Duplications Duplications on 2.6k Lines 

Authentication

- User-ID and Password
- LDAP (enable it in sonar.properties)
- SSO authentication (enable it in sonar.properties)

Authorization

What you are allowed to do?

Authentication → security → create user

Create a group and attach the user

Attach global permission to the group

Authorization

Administration

Configuration ▾ Security ▾ Projects ▾ System Marketplace

Global Permissions

Grant and revoke permissions to make changes at the global level. These permissions include editing Quality Profiles, executing analysis, and performing global system administration.

All	Users	Groups	Search for users or groups...	Administer System	Administer	Execute Analysis	Create
<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	sonar-administrators System administrators	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> Quality Gates <input checked="" type="checkbox"/> Quality Profiles	<input type="checkbox"/>	<input checked="" type="checkbox"/> Projects
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	sonar-users Every authenticated user automatically belongs to this group	<input type="checkbox"/>	<input type="checkbox"/> Quality Gates <input type="checkbox"/> Quality Profiles	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> Projects
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Anyone <small>DEPRECATED</small> Anybody who browses the application belongs to this group. If authentication is not enforced, assigned permissions also apply to non-authenticated users.	<input type="checkbox"/>	<input type="checkbox"/> Quality Gates <input type="checkbox"/> Quality Profiles	<input type="checkbox"/>	<input type="checkbox"/> Projects
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	dev	<input type="checkbox"/>	<input checked="" type="checkbox"/> Quality Gates <input checked="" type="checkbox"/> Quality Profiles	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> Projects

Connect with PostgreSQL database

Docker-compose.yaml

```
version: '3'
services:
  postgres:
    image: postgres:latest
    environment:
      POSTGRES_USER: sonarqube
      POSTGRES_PASSWORD: sonarqube
      POSTGRES_DB: sonarqube
    volumes:
      - postgres_data:/var/lib/postgresql/data
  sonarqube:
    image: sonarqube:latest
    environment:
      SONARQUBE_JDBC_URL: jdbc:postgresql://postgres:5432/sonarqube
      SONARQUBE_JDBC_USERNAME: sonarqube
      SONARQUBE_JDBC_PASSWORD: sonarqube
    ports:
      - "9000:9000"
    depends_on:
      - postgres
    volumes:
      - sonarqube_data:/opt/sonarqube/data
volumes:
  postgres_data:
  sonarqube_data:
```

Maven Integration

1. Edit maven setting.xml file

```
/usr/share/maven/conf/settings.xml
```

```
<settings>
<pluginGroups> <pluginGroup>org.sonarsource.scanner.maven</pluginGroup> </pluginGroups>
<profiles>
<profile>
<id>sonar</id>
<activation> <activeByDefault>true</activeByDefault> </activation>
<properties> <!-- Optional URL to server. Default value is http://localhost:9000 --> <sonar.host.url>
http://myserver:9000 </sonar.host.url> </properties>
</profile> </profiles>
</settings>
```

Maven Integration

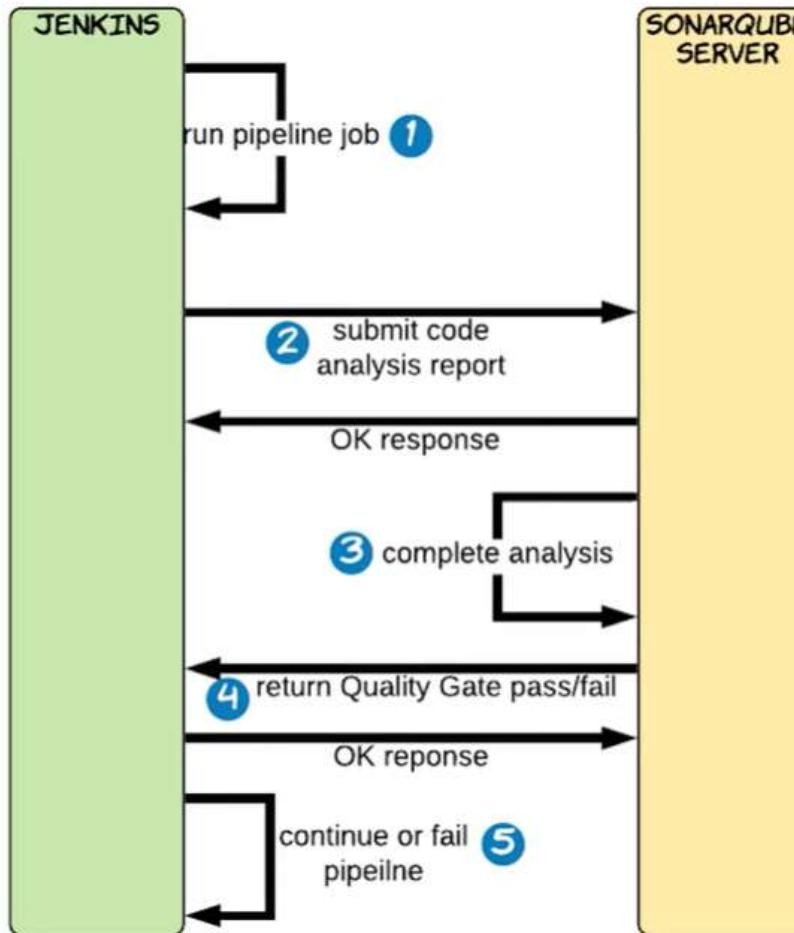
2. Make changes in pom.xml to lock down versions of Mavens

- define property <sonar.skip>true</sonar.skip> in the pom.xml of the module you want to exclude
- use build profiles to exclude some modules (like for integration tests)

3. Create a token (user → my account → security → token)

4. **mvn clean verify sonar:sonar -Dsonar.login=myAuthenticationToken**

Jenkins Integration with SonarQube



Jenkins Integration with SonarQube

1. Install Jenkins
2. Install SonarQube scanner for Jenkins plugins (manage Jenkins → plugins → SonarQube Scanner for Jenkins → install)
3. Add SonarQube server details (manage Jenkins → system → SonarQube Servers → add SonarQube, also add the token to access SonarQube)



Jenkins Integration with SonarQube

4. Configure tool (Manage Jenkins → tools →)

The screenshot shows the Jenkins configuration interface for managing SonarQube Scanner installations. The top navigation bar indicates the page is 'Edited'. The main section is titled 'SonarQube Scanner installations' and contains a single entry for 'sonar-scanner'. This entry includes fields for 'Name' (set to 'sonar-scanner'), a checked checkbox for 'Install automatically', and a dropdown menu for 'Version' (set to 'SonarQube Scanner 5.0.1.3006'). A 'Add Installer' button is also visible at the bottom left.

SonarQube Scanner installations Edited

Add SonarQube Scanner

SonarQube Scanner

Name
sonar-scanner

Install automatically ?

Install from Maven Central

Version
SonarQube Scanner 5.0.1.3006

Add Installer ▾

Jenkins Integration with SonarQube

5. Create a Pipeline (new Item → free style)

Enter an item name

pet
» Required field

 **Freestyle project**
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

 **Pipeline**
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

 **Multi-configuration project**
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

 **Folder**
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

 **Multibranch Pipeline**
Creates a set of Pipeline projects according to detected branches in one SCM repository.

Jenkins Integration with SonarQube

6. integrate with git repo and provide the path

Source Code Management

None

Git [?](#)

Repositories [?](#)

Repository URL [?](#) ✖

`https://github.com/CloudSihmar/pet.git`

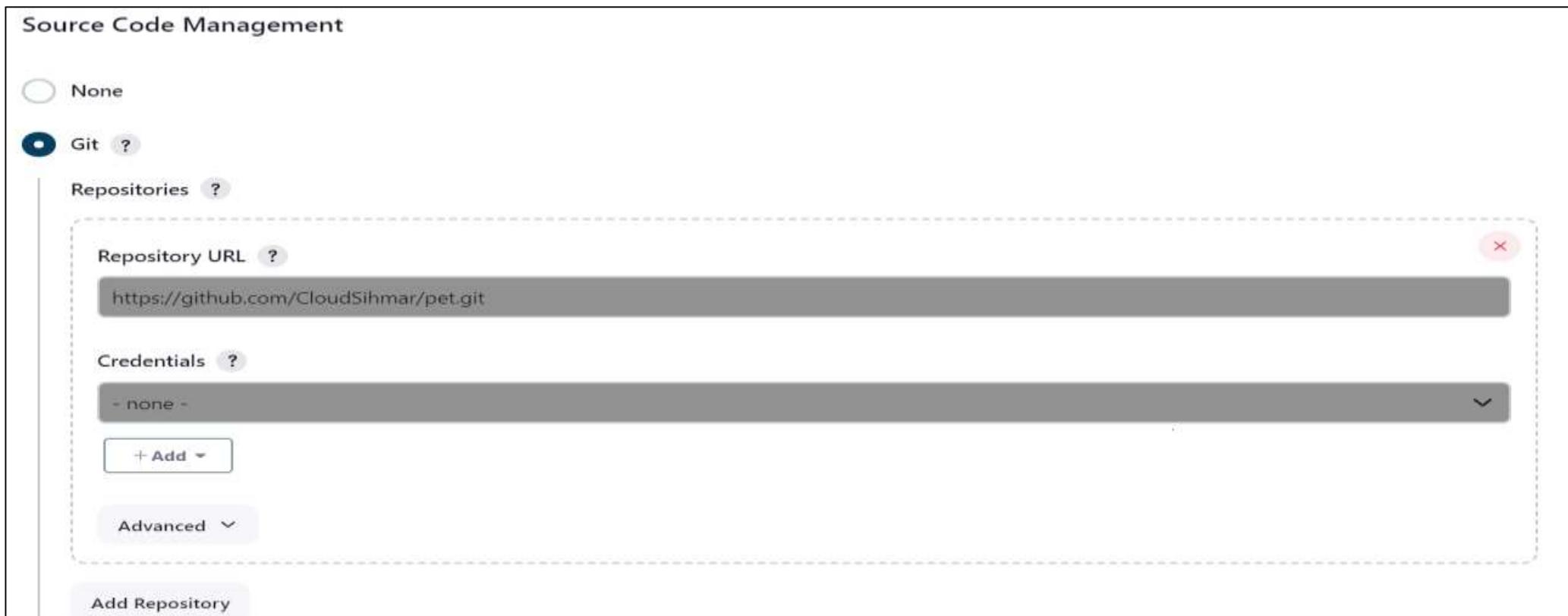
Credentials [?](#)

- none - ▼

+ Add ▾

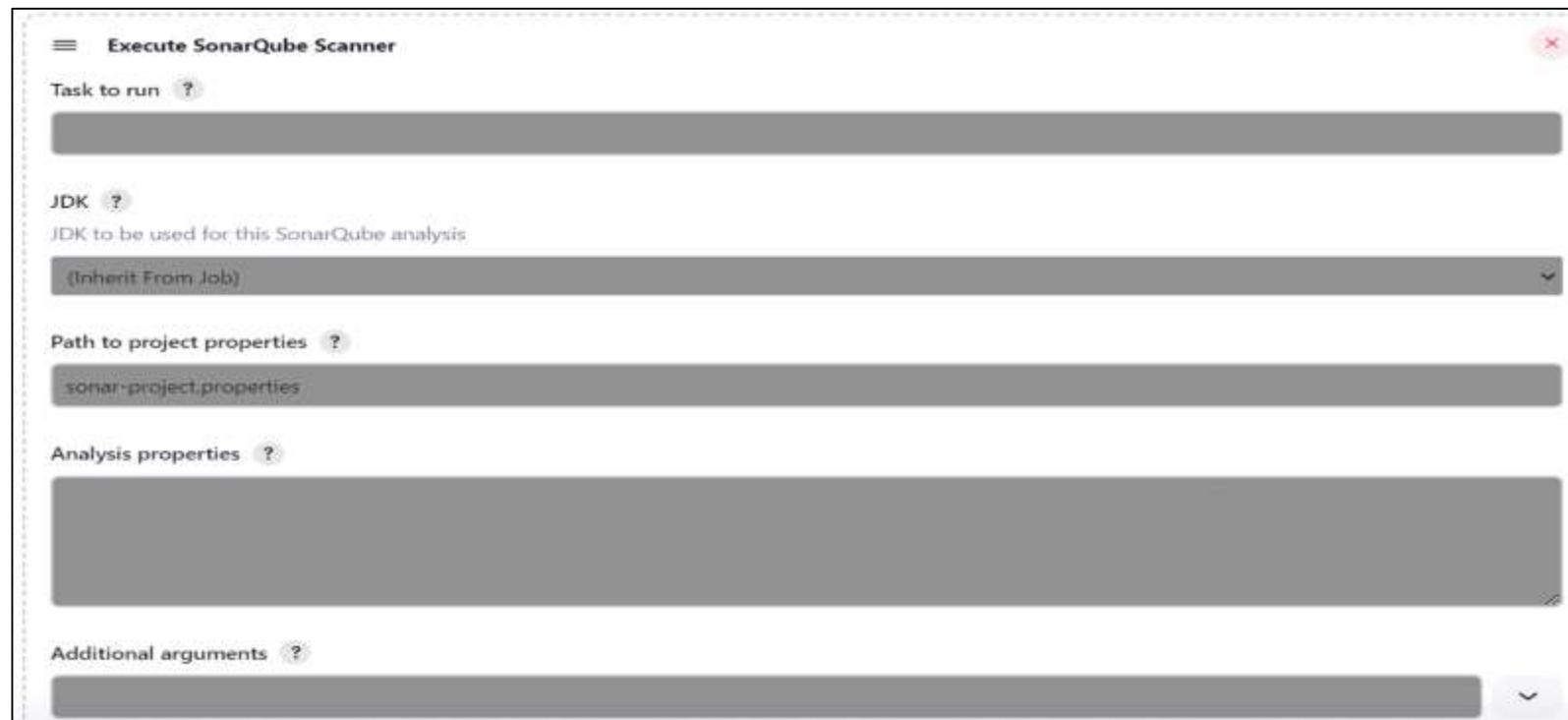
Advanced ▾

Add Repository

A screenshot of the Jenkins configuration interface for source code management. The 'Git' option is selected. A single repository is configured with the URL 'https://github.com/CloudSihmar/pet.git'. There are no credentials defined. The 'Advanced' section is collapsed. At the bottom, there is a button to add more repositories.

Jenkins Integration with SonarQube

7. Add SonarQube build step (build steps → Execute SonarQube Scanner), provide the path of sonar project properties if available in Repo, else provide the properties details in Analysis properties field.



Jenkins Integration with SonarQube

sonar-project.properties

```
# must be unique in a given SonarQube instance
sonar.projectKey=my-project3

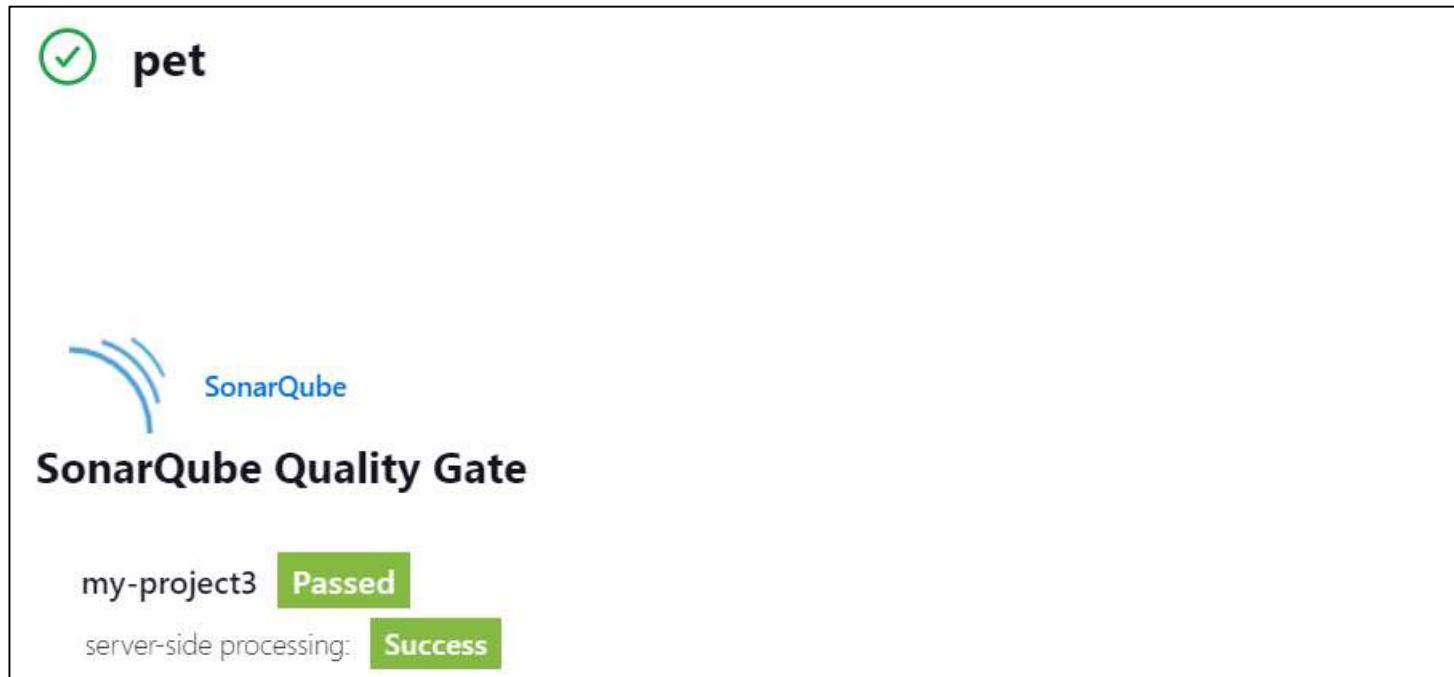
# --- optional properties ---

# defaults to project key
#sonar.projectName=my-project3
#sonar.token=sq_e1b901c84ea5f9c688fd20f2de8803ca2bc29a1e
# defaults to 'not provided'
#sonar.projectVersion=1.0

# Path is relative to the sonar-project.properties file. Defaults to .
#sonar.sources=.
sonar.java.binaries=.
sonar.exclusions=src/main/resources/**/*.*java
# Encoding of the source code. Default is default system encoding
#sonar.sourceEncoding=UTF-8
```

Jenkins Integration with SonarQube

8. Build now and check the status



Jenkins Integration with SonarQube

[Check on SonarQube for more details](#)

The screenshot shows the SonarQube web interface for the project "my-project3". The main navigation bar includes links for Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, More, and a search bar. A banner at the top right indicates that the last analysis has warnings, with a link to see details. The main content area is divided into several sections:

- Quality Gate:** Passed (green checkmark icon).
Icon: A monitor with code and a checkmark.
- Reliability:** 41 Bugs (yellow circle with 'c').
- Maintainability:** 43 Code Smells (green circle with 'A').
- Security:** 1 Vulnerabilities (red circle with 'E').
- Security Review:** 2 Security Hotspots (red circle with 'E').
- Coverage:** 0.0% Coverage (red circle with 'O').
Details: Coverage on 519 Lines to cover.
Sub-section: Unit Tests.
- Duplications:** 0.0% Duplications (green circle with 'G').
Details: Duplications on 2.6k Lines.
Sub-section: Duplicated Blocks.

Below these sections, there are buttons for "New Code" and "Overall Code". The bottom of the page features a footer with links for Project Settings, Project Information, and other navigation options.

Jenkins Integration with SonarCloud

Go to <https://sonarcloud.io/>

Login with GitHub, GitLab or Azure

The screenshot shows the SonarCloud interface for managing security tokens. At the top, there's a navigation bar with links for My Projects, My Issues, Explore, and a search bar. Below the navigation, the user profile 'CloudSihmar' is displayed, along with tabs for Profile, Security (which is selected), Notifications, Organizations, and Appearance.

The main content area is titled 'Security'. It contains a section about generating User Tokens for code scanning or web services. A 'Generate Tokens' button is present, along with a field to 'Enter Token Name'. Below this, a table lists existing tokens, with one entry for 'jenkins'.

To the right, a dark overlay box provides information about upgrading to a Paid Plan to get private projects. It highlights features like a 14-day free trial, strict control over data, and unlimited private projects. Buttons for 'Upgrade your organizations' and 'Learn more' are at the bottom of the overlay.

Name	Last use	Created
jenkins	< 1 hour ago	February 6, 2024

Jenkins Integration with SonarCloud

SonarQube installations

List of SonarQube installations:

Name
SonarCloud

Name

SonarCloud

Server URL

Default is <http://localhost:9000>

<https://sonarcloud.io>

Server authentication token

SonarQube authentication token. Mandatory when anonymous access is disabled.

sonar-cloud-token

+ Add ▾

Advanced ▾



Jenkins Integration with SonarCloud

The screenshot shows the SonarCloud interface for a project named 'pet'. The project is marked as 'PUBLIC' with a star icon. The navigation bar includes links for 'Summary', 'Issues', 'Security Hotspots', 'Measures' (which is selected), 'Code', and 'Activity'. A prominent alert message at the top right states: 'The last analysis has warnings. See details'.

The left sidebar contains sections for 'Overview', 'Main Branch' (with 0 pull requests), 'Pull Requests' (0), 'Branches' (1), 'Information', 'Administration', and a 'Collapse' button.

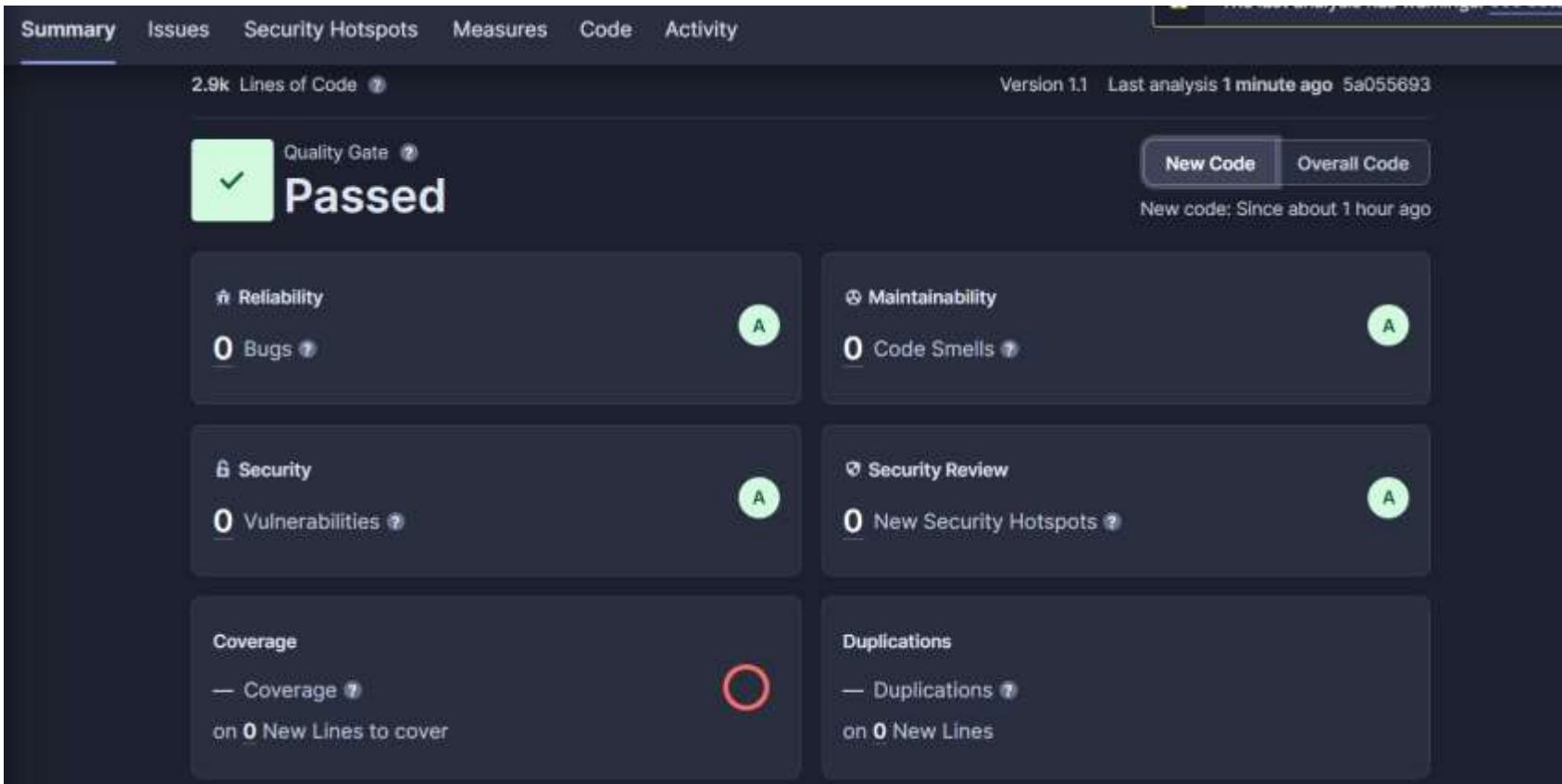
The main content area displays the 'Measures' tab. It features a 'Project Overview' card and a 'Reliability' card. Below these, there's a 'Security' section with 'Overview', 'Overall Code', and a 'Vulnerabilities' section showing 4 findings. The 'Rating' is listed as 7, and the 'Remediation Effort' is 45min. At the bottom of the security section is a 'Security Review' card.

The 'Vulnerabilities' section lists 4 findings across various files:

Vulnerability	Count
push-to-pws	0
src	0
appspec.yml	0
buildspec.yml	0
docker-compose.yml	1
Dockerfile	0
index.html	0
nginx.yaml	3
pom.xml	0
service.yaml	0

At the bottom right of the vulnerabilities list, it says '10 of 10 shown'.

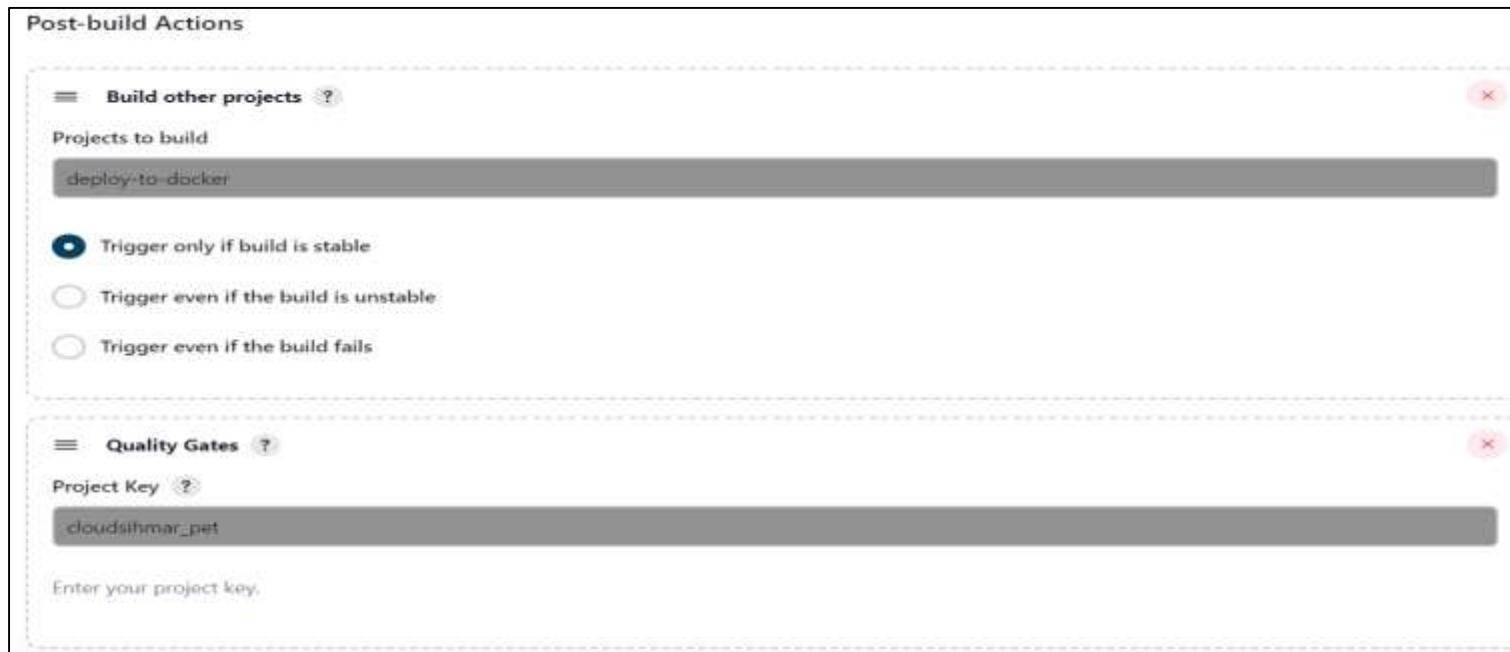
Jenkins Integration with SonarCloud



Jenkins Integration with SonarCloud

Getting Quality Gate status from SonarCloud:

1. Install Quality Gates plugins in Jenkins and configure it
2. Add a post build action and select quality gate



Jenkins Integration with SonarCloud

3. Create a Webhook on SonarQube and provide the Jenkins URL there.

