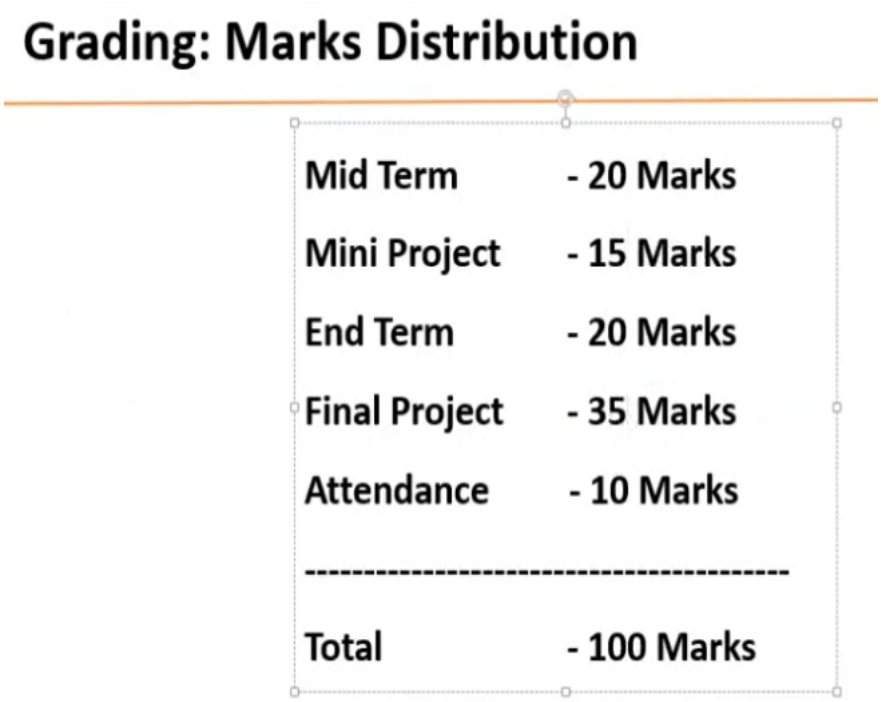
11/01/22: Lecture



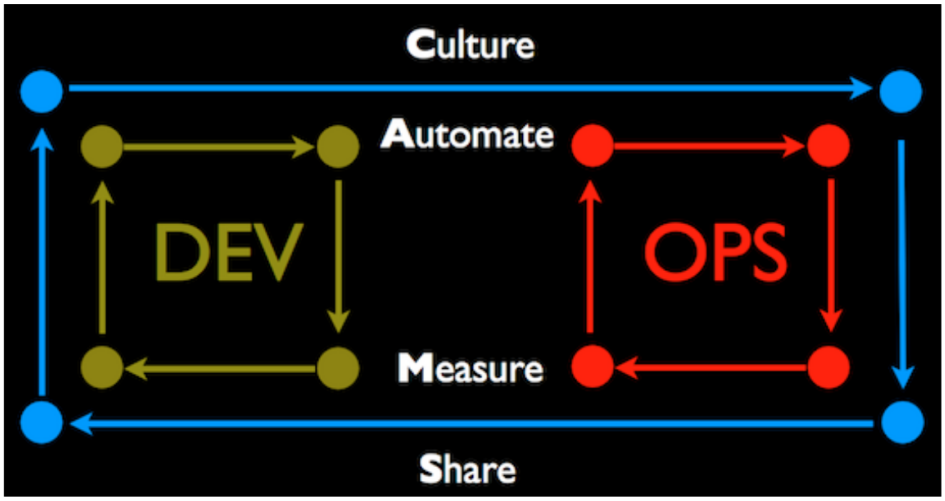
Goal is to develop a full CI/CD pipeline that goes from the Git repository to deployment on a server.

Important Questions

Why do we need DevOps?
User management related questions (in Docker I guess)
Why do we need distributed architecture?
Docker cgroups and namespaces, why and what?

Module 1: Introduction

DevOps



Complete collaboration between Developers and Operation Engineers -> DevOps.
It is a methodology/ideology/model for project management like Waterfall, Agile, etc.
This is why a term like DevOps Engineer doesn't make much sense because you don't have stuff like Agile Engineer or Waterfall Engineer.
Ofcourse this is all subjective and as always you can still find job postings for DevOps engineers as well :)

Software Engineering

When a Customer explains how their project is to be done, the way it's interpreted by the Project Leader and people further down the hierarchy keeps changing and eventually we end up getting a product that isn't even close to what the Customer wanted.

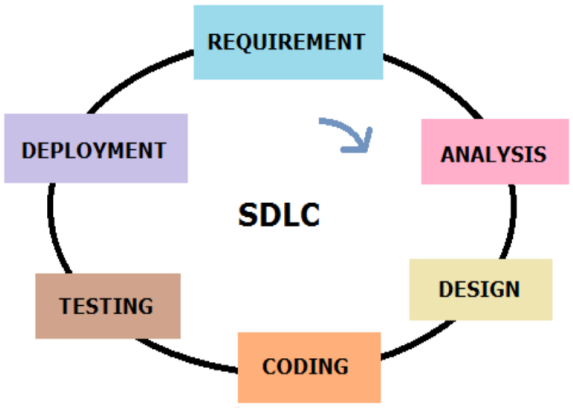
And the bigger twist is that the Customer themselves is not completely sure of what they want and it is up to you, the IT team to properly interpret what is actually that the Customer wants. Basically, we need really good project management and analysis and all that good SDLC stuff.

- Software Engineering - Software engineering encompasses a process, a collection of methods (practice) and an array of tools that allow professionals to build high quality computer software.

- Domains -Business, science and engineering

- Types of Software - System Software (BSP, OS), OSS, Application Software, Engineering or Scientific Software, Embedded Software, Web Application and AI Software.

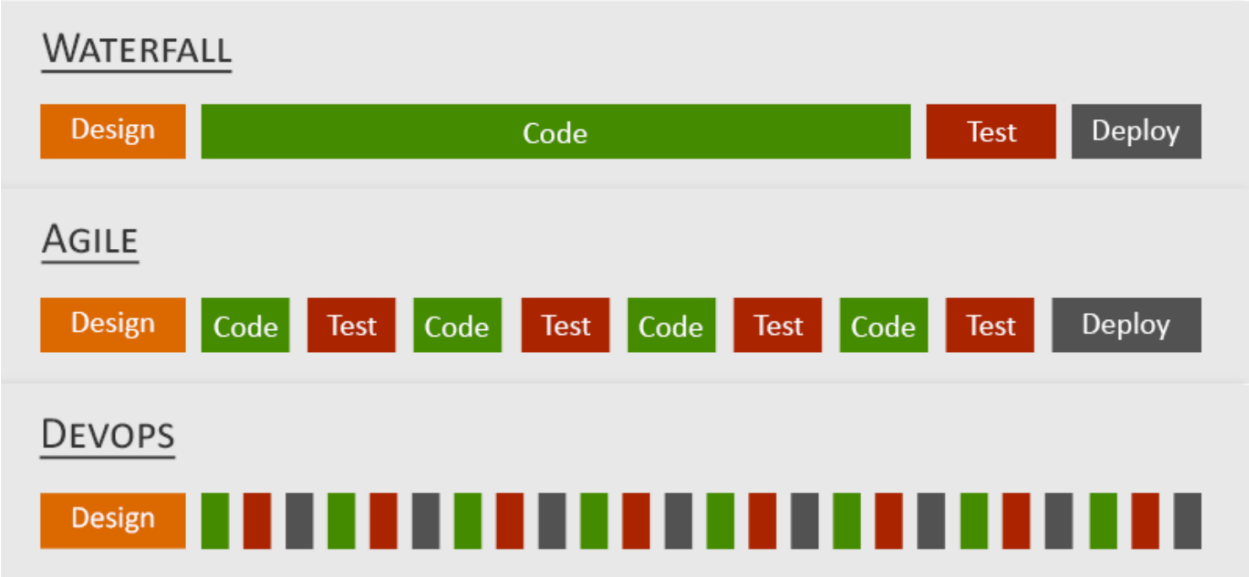
- SDLC



You should be getting flashbacks about the details of each component in SDLC as was covered in the ESD course because that's required here as well.

Summary:
i) High level requirements specifications like SRS document.
ii) Analysis of each specification and nailing down the architectures required
iii) Designing of individual diagrams for each component
iv) Converting that to coding
v) Unit testing, system testing, integration testing, user acceptance testing
vi) Deployment
This continues in a cycle.

Waterfall vs Agile vs DevOps



Basically, the development cycle also includes deployment (which is on the operations side of things) now and everything is done at a much, much faster rate with smaller workload (only few user stories at a time).

Note: The Agile representation here is talking about only one sprint as typically deployment happens at the end of a sprint (which can go from 1 week to multiple weeks or months), which is still different from DevOps as any testing phase is followed by a Deployment in DevOps.

Note 2: In DevOps, after the testing phase, the product is delivered and ready to be deployed, whether it is actually deployed or not is dependent on the use case and product requirements at the time. But this convenience is not present in Agile as readiness for deployment is only achieved at the end of a sprint.

The benefit of adding deployment as part of the development cycle is that with each tested component, you as well as the end-user can immediately see the product response with the updated features.

DevOps is basically the CI/CD pipeline as each element is automated here. Once the updated code has been pushed, it'll be automatically tested against the specified test cases and once testing has been successfully done, the updated code will be automatically deployed.

The CI/CD pipeline can also include the user acceptance testing phase, all we have to do is just add the relevant UAT test cases in the pipeline and even that will be automated here.
Similarly, we can add or remove any required phases in the pipeline and provide the relevant information like test cases in case we're adding a testing phase.

CI = Continuous Integration

Integration refers to integration of updated code with a Version Control System (VCS) like Git.

Periodic Table of DevOps Tools

[Image Link] because it'll be too blurry otherwise.
* The table is subject to change at a rapid pace :(

Corporate Culture

Reliability Engineering is the same as the DevOps team, i.e. training developers on the Operations side and having them manage both departments.

Challenges to IT Agility

1. Everything needs software.
2 Software runs on a server to become a service
3. Delivering a service from inception to its users is too slow and error-prone
4. There are internal friction points.
5. Defects released into production.
6. Inability to diagnose production issues quickly
7. Problems appear in some environments only; Blame shifting or finger pointing
9. Long delays while dev, QA, ops team waits on resource/response from other teams.
10. Releases slip or fail - Delay causes money loss.
11. Traditional Thinking: Conflicting Perspectives and Lesser Collaboration

Dev's job is to add new features; Ops' job is to keep the site stable and fast.

- Delivering a service is too slow because everything is done manually (development, testing, deployment).
- Internal friction points, i.e. points within the organization itself where the product's functioning is challenged. For example, the development environment successfully runs the updated product but because of different infrastructure being used in the testing environment, it is not able to successfully run the updated product and the blame game begins on who is at fault here which is what point 7 is talking about.

Brief History of DevOps

- In 2009, at the O'Reilly Velocity Conference, two Flickr employees—John Allspaw, senior vice president of technical operations, and Paul Hammond, director of engineering—deliver a seminal talk known as "10+ Deploys per Day: Dev and Ops Cooperation at Flickr." (Flickr is an image- and video-hosting website and web services suite)
- The talk is an energetic presentation in which John and Paul act out the classic "finger point" problem of Dev versus Ops. Whey they brought together to a roomful of developers and operations teams. The usual blame was "It's not my code, it's your machines!" or it is not my machine, it is your code created the problem.
- They explained that the only sensible way to build, test and deploy workable new software is to make development and operations transparent and integrated.
- The talk becomes widely credited with showing the world what development-operations collaboration can achieve.
- Viewing the presentation from Belgium via streamed video, Debois who is an independent IT consultant inspired to organize his own conference, called Devopsdays in Belgium.

13/01/22: TA Session

Version Control System (VCS): Git

Why do we need a VCS?

- If you are a software developer and working in a project team, your main job is to develop code for a specific module in your project. Initially you may develop a code locally in your system. If your project code is getting bigger and bigger then you should maintain the code carefully.

- Suppose you are working on a code and after making many changes you realize that you have really messed up or the current version of your code may have some issues and now you would like to revert to the last good version of your project. How would you do that?

- if you are not maintaining copies of the various versions of your code then you will be in trouble. So how do you revert to the previous working version of your project code?

- In other scenario, if you are working in a project team and develop a module with a group of team members, every time you work on the project, you should know exactly what has already been completed, added, changed and so on. How would you know who made the changes, to which files were changed. Otherwise, you may end up working on something that is already been finished.

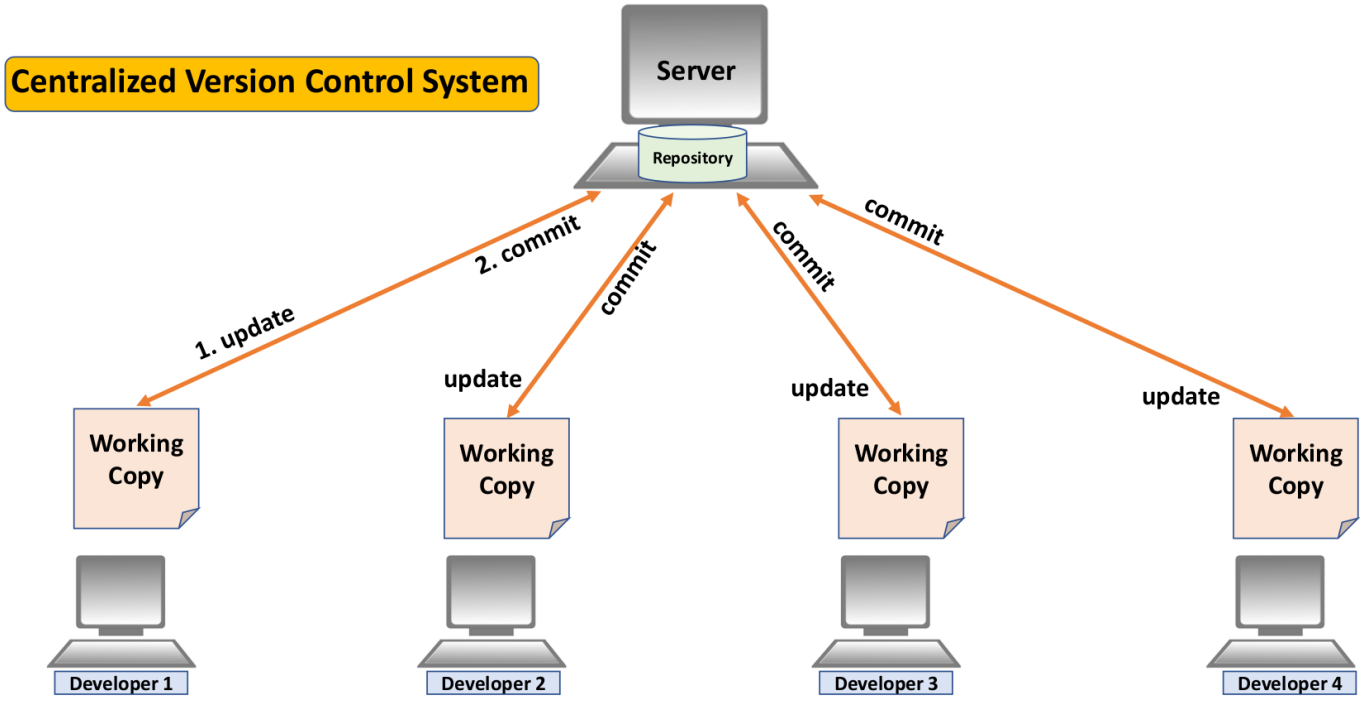
- Now, how do you share your recent code update to your team members? One way is that you can compress your code with all the necessary artifacts and send every updates through email then others can start to work with the module. If a large group of developers share the code updates through mail then someone should collate all the updates into the whole project code. This process will become a hectic task and error prone.

- This is an inefficient way to share the code updates to the team. Do we have any better solution to share the code to others and maintain history of each and every updates in a systematic way? The solution is Version Control System.

Centralized VCS

- Version control system is the management of changes to collection of information like documents, project code.
- It can track collaborative changes to a project, so a developer is aware of the recent changes in the project code and he can access the most recent version of the project.
- Also developers can view all the past versions of the code and the difference between them.
- There are two major types of version control model namely
  - 1. Centralized
  - 2. Distributed

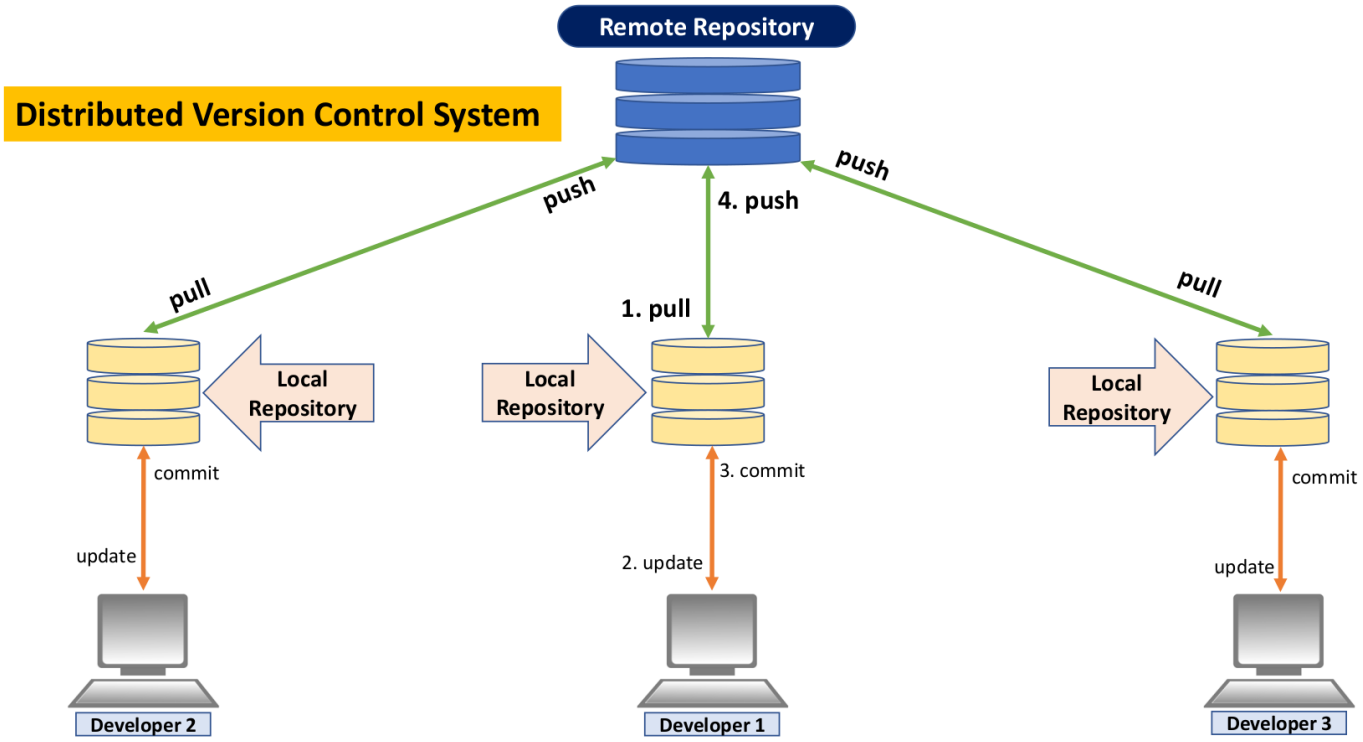
The **centralized version control system** is working as a client-server model. Here we have one centralized server and a localized repository filesystem that is accessible by a number of clients. The advantage of this centralized model are simplicity and ease of use. One of the examples of this model is Subversion.



- Google doc is an example of centralized version control system because google doc only keeps one file for the word document, and everyone works on that one file. Therefore, all the changes and modifications will be stored and get reflected on that one common file.
- In Subversion there is only one repository filesystem, usually stored on a network server. Whenever you want to work on your project, you can connect the central file system through network connection. So you cannot do any work on your project without a network connection. If anything happened to the central filesystem you may lose your data and or history of the changes.
- So the disadvantages are the Subversion is completely dependent on the functionality of a single server, which can become a bottleneck and may affect the performance due to heavy network traffic and reliability of backups.

Distributed VCS

- Distributed version control systems takes a peer-to-peer approach to version control, as opposed to the client–server approach of centralized systems.
- In the distributed model, all developers have their own local filesystem, and changes between file system are implemented locally on their machines. The advantages of this model are faster access, ability to work offline and does not rely on a single location for backups. Example for this model are GitHub and Mercurial.
- Git has some advantages over other distributed version control systems like Mercurial.
  - git is faster than mercurial for network operations such as downloading and uploading project files to the file server.
  - Git's approach to branches are more powerful than Mercurial.
  - git is more powerful for larger projects.
  - Git is one of the most widely-used popular version control system in use today. For example, teams at Amazon and Microsoft have adopted git as their version control system for many of their projects.
  - Git is Open Source and you can create a public or private file system which can be accessible by your project team.

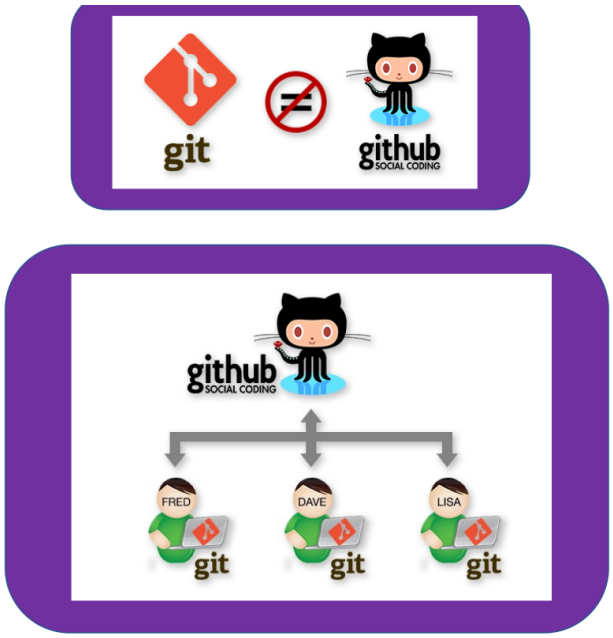


Git

- Git is invented by Linus Torvalds.
- The latest Linux kernel-4.13 has 60,538 files and over 24 million lines of code.
- Approximately 3,500 new lines of code are added into the Linux kernel source code every day.
- 1,681 developers are involved from 225 companies. New version of the Linux kernel is released in every quarter.
- To maintain such length code which is being worked upon by thousands of developers there was a need of a tool which could manage such a length code well and make collaborations between developers across different places and companies efficient and easy.
- Git is used to maintain Linux kernel. Git makes collaboration as quick and painless as possible.
- If Git helps to maintain a Linux kernel project of that magnitude operating smoothly, you can imagine how easy git can make collaboration on your projects.

Git vs GitHub

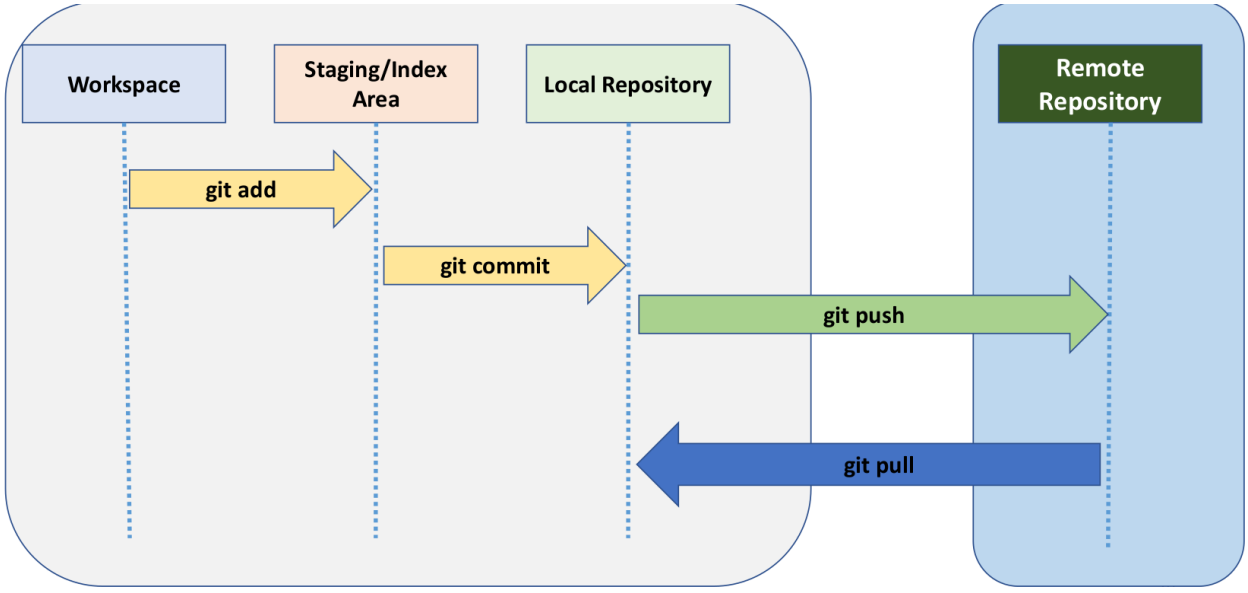
- Git is a distributed version control system, it is a tool to manage your project source code history.
- Whereas Github is a web based, git file hosting service which enables us to showcase/share our projects and files to others.



Git Installation + Demo + GitHub Demo

Demo for how to install git and use it to commit to the local repository.
Then use this to commit to a remote GitHub repository.
Learn this on your own :)

Git Workflow



Git remote and git push

```
root@IITB:/git/project1# git remote add origin https://github.com/BThangaraju/Project1.git
root@IITB:/git/project1# git push -u origin master
Username for 'https://github.com': BThangaraju
Password for 'https://BThangaraju@github.com':
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 645 bytes | 322.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/BThangaraju/Project1.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
root@IITB:/git/project1#
```

git remote add origin <URL>

- This command gathers all the committed files from your local repository and uploads them to our repository we created on github.

git push -u origin master

- This command will push all our local changes to our online repository that is our repository on Github.

"Git push" tells git that we want to upload our files and development history from the repository on our local computer to a repository hosted on github

"-u origin master" means that we want to upload the "master" version of our commits to a repository on github called "origin"

Git branches

- A branch is the fundamental means of launching a separate line of development with a software project. In git, you can create many branches resulting in many different lines of development within a git repository. The branch management in git is lightweight and simple to learn.
- Each team in a project can work on a different branch simultaneously. When you are ready and if you want to merge some specific branches or all the branches with master branch you can do it easily.
- To start off with, git will automatically create a master copy (or "branch" in git terms) of your project when you create a repository. This master copy is called the "master" branch.
- If you want to do parallel development with the existing project code, without making any changes to our master branch then you can create different branches based on your need and each branch will have the same copy of the master branch project source code.
  - For example, you can create different branches for different team members. Or you can create different branches for the different features that you are adding to the project.

Merging branches

- **Merging different branches into master branch**
- Now, we can use the command **#git merge branchname to** merge branch1 into the master and then merge team2 branch into the master branch.
- ***#git merge branch1***
- ***#git merge branch 2***
- Now if we want to the see the difference between the lines of code from branc1 and master we will use the command
- ***#git diff master..branch1***
- *This* command will highlight the added lines into the calculator.java file.

```
root@IITB:/git/branch_demo# git merge team1
Updating 742e5b3..f816c7f
Fast-forward
 HelloWorld.java | 1 +
 1 file changed, 1 insertion(+)
root@IITB:/git/branch_demo# git merge team2
Auto-merging HelloWorld.java
Merge made by the 'recursive' strategy.
 HelloWorld.java | 1 +
 1 file changed, 1 insertion(+)
root@IITB:/git/branch_demo# javac HelloWorld.java
root@IITB:/git/branch_demo# java HelloWorld
Hello World from Master Branch !
Hello World from team2 branch !
root@IITB:/git/branch_demo# more HelloWorld.java
/* This is Hello World Java proram - team1 added the comment */
public class HelloWorld {
    public static void main( String[] args ) {
            System.out.println( "Hello World from Master Branch !" );
            System.out.println( "Hello World from team2 branch !" );
            System.exit( 0 );
        }
}
root@IITB:/git/branch_demo#
```

Managing Conflict

When a team is working on the same project on the same files but from different branches conflicts might arise maybe two people end up changing the same lines of code in a given file.

Git will be confused about which one of the conflicting changes it should consider for merging. And in such cases when there are merge conflict, the merge will fail, and we won't be able to merge one branch into another branch.

If branch1 and branch2 modified the same line in the calculator.java program then it will create conflict when you want to merge the branches into the master branch.

```
root@IITB:/git/branch_demo# git diff master..team1
diff --git a/HelloWorld.java b/HelloWorld.java
index bd89cbc..5b4f403 100644
--- a/HelloWorld.java
+++ b/HelloWorld.java
@@ -1,6 +1,7 @@
 public class HelloWorld {
     public static void main( String[] args ) {
             System.out.println( "Hello World!" );
+            System.out.println( "Hello World! program modified by team1" );
             System.exit( 0 ); //success
         }
 }
root@IITB:/git/branch_demo# git diff master..team2
diff --git a/HelloWorld.java b/HelloWorld.java
index bd89cbc..9511e84 100644
--- a/HelloWorld.java
+++ b/HelloWorld.java
@@ -1,6 +1,7 @@
 public class HelloWorld {
     public static void main( String[] args ) {
             System.out.println( "Hello World!" );
+            System.out.println( "Hello World! program modified by team2" );
             System.exit( 0 ); //success
         }
 }
root@IITB:/git/branch_demo#
```

```
root@IITB:/git/branch_demo# git checkout master
Switched to branch 'master'
root@IITB:/git/branch_demo# git merge team1
Updating 99e18cb..dee8ea9
Fast-forward
 HelloWorld.java | 1 +
 1 file changed, 1 insertion(+)
root@IITB:/git/branch_demo# git merge team2
Auto-merging HelloWorld.java
CONFLICT (content): Merge conflict in HelloWorld.java
Automatic merge failed; fix conflicts and then commit the result.
root@IITB:/git/branch_demo#
```

- The first merge between master and branch1 is working fine but the second merge with team2 creates conflict.
- When you open the calculator.java file, we can see the conflict details and you can decide whether you want to keep both the changes or remove any modifications created by branch2.

```
public class HelloWorld {
    public static void main( String[] args ) {
        System.out.println( "Hello World!" );
<<<<<<< HEAD
        System.out.println( "Hello World! program modified by team1" );
=======
        System.out.println( "Hello World! program modified by team2" );
>>>>>>> team2
        System.exit( 0 ); //success
    }
}
```


• To remove a conflict we should use our best judgments to resolve branch conflicts.

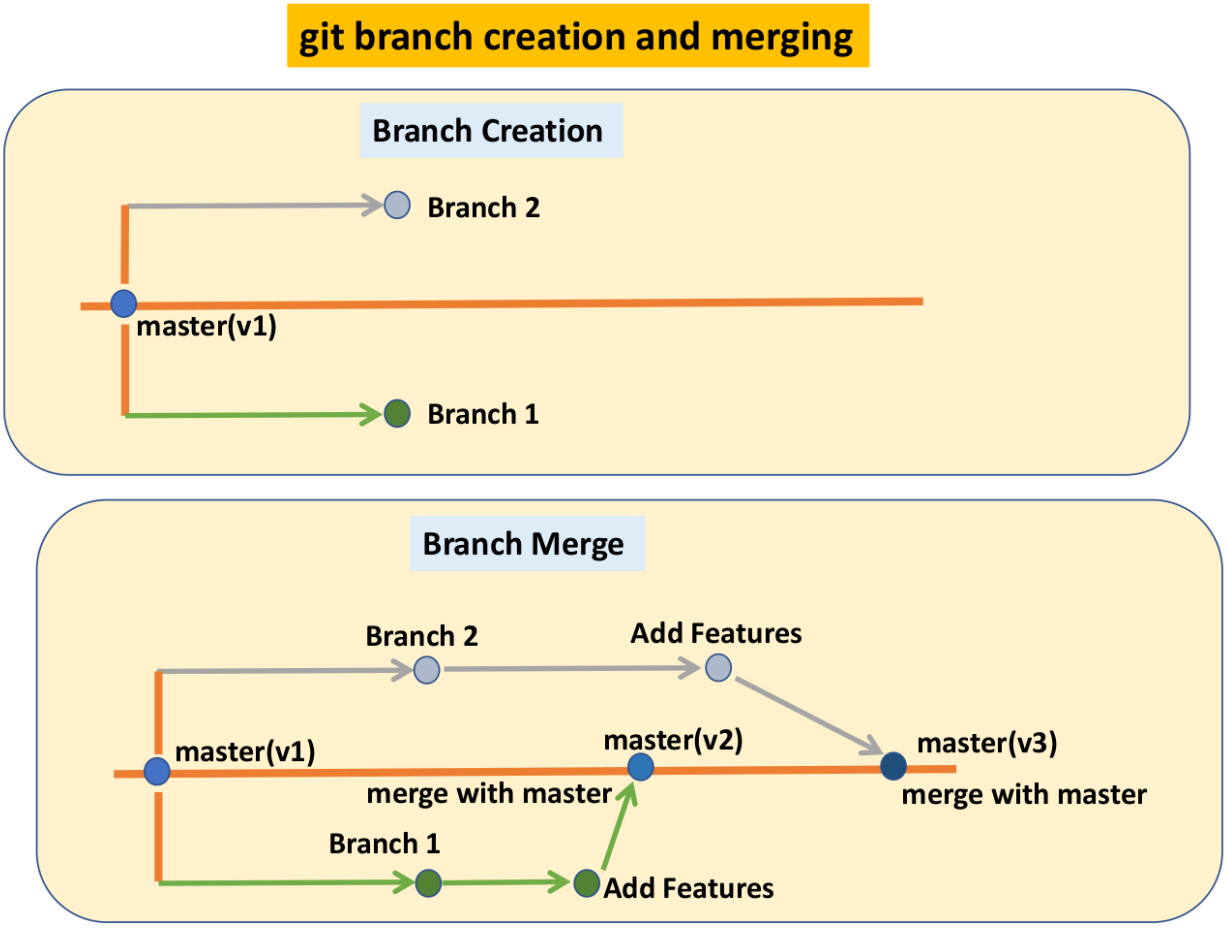
We should use our best judgment to determine which branch is “correct” and which branch is “faulty”

• Now removed the HEAD and team2 lines and save the file and then commit the file to keep modifications done by team1 and team2.

Then add and commit the calculator.java file into the master branch.

When you compile and execute the program, it will be working fine and can see the modifications done by both branch1 and branch 2

```
root@IITB:/git/branch_demo# vim HelloWorld.java
root@IITB:/git/branch_demo# 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:   HelloWorld.java

no changes added to commit (use "git add" and/or "git commit -a")
root@IITB:/git/branch_demo# git add HelloWorld.java
root@IITB:/git/branch_demo# git commit -m "Resolved the conflict"
[master ae6036c] Resolved the conflict
root@IITB:/git/branch_demo# javac HelloWorld.java
root@IITB:/git/branch_demo# java HelloWorld
Hello World!
Hello World! program modified by team1
Hello World! program modified by team2
root@IITB:/git/branch_demo#
```



Git clone –working with remote repo

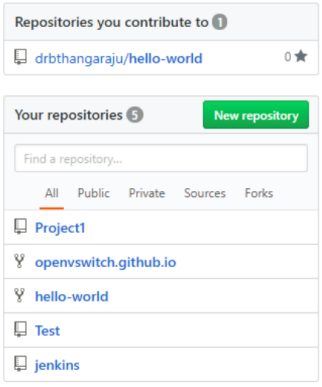
A clone is a copy of a repository. A clone contains all the objects from the original. Each clone is an independent and autonomous repository and a symmetric peer of the original.

• **git clone <URL> -** The command git clone creates a new, local Git repository on your computer and copies all the files, commit histories, commit messages, branches, and etc.

Git fork

- If you want to work with someone projects or if I want to collaborate with any open source projects then I need to go their github web page and fork their project into my github account.
- For example, in our demo, I will fork OVS project from https://github.com/openvswitch/openvswitch.github.io
- OVS (Open v Switch) is an open source implementation of a distributed virtual multilayer switch.
- OVS is to provide a switching stack for hardware virtualization environments, while supporting multiple protocols and standards used in computer networks.
- In the following page, click fork button

Now I will go to my github account and check the forked repository.

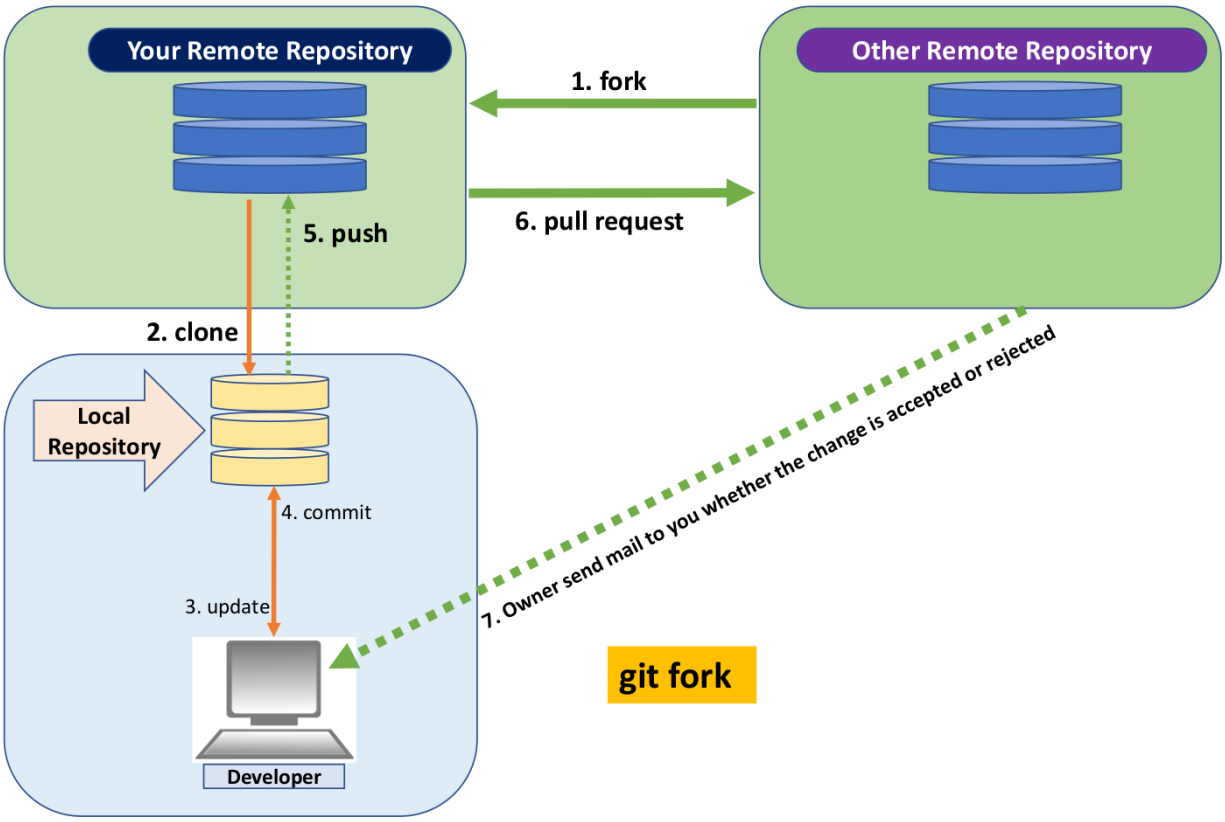


The **openvswitch** open source project is in my remote repository.

You can do the modifications or add any new changes into the project after cloning into the local repository and push the changes into your remote repository.

If you want to contribute back to the openvswitch project, you can create **pull request** with propose file change details.

The project maintainer of the openvswitch will decide whether will accept or reject your change proposal.



Advanced Git usage

https://onlywei.github.io/explain-git-with-d3/#rebase
13 Advanced (but useful) Git Techniques and Shortcuts - YouTube

18/01/22: Lecture

Basic DevOps terms

Dev: People involved in developing the product.
Ops: System engineers, admins, operations staff, release engineers, DBAs, network engineers and security professionals.
Agile Software Development: Collaboration of customers, product management, developers and QA to fill in the gaps and rapidly iterate towards a better product.
DevOps: Extending Agile principles beyond the boundary of “the code” to the entire delivered service.
Goal of DevOps - Spans the entire delivery pipeline.

- Improved development frequency
- Faster time to market
- Lower failure rate of new release

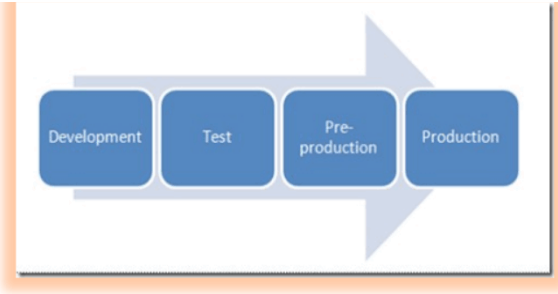
- Shortened lead time between fixes
- Faster mean time to recovery in the event of a new release crashing

Lead time - Time elapsed between the identification of a requirement and its fulfillment.
Mean time to failure - Length of time a system or application is expected to last in operation.

SDLC Journey

[Most of the things here will seem like a repeat of what has been discussed before but it will help with the overall clarity of the subject topic of DevOps]

- **Goal is to make this journey smooth, quick and without hiccups.**
- **What is constant and variable in this SDLC Journey?**



- As the development team reaches code complete, then deploy the code to the test cluster where the QA team goes through a test pass.
- After meeting the test pass exit criteria, roll the code into an intermediary staging environment, which is called “Pre-production”.
- This staging environment matches our production hardware as close as possible and use it to do final acceptance testing.
- This model, helps ensure the highest quality release when finally deploy to Production.

Consider the above given SDLC, in it after the “Development” phase, the code is essentially complete and so that’s the constant part whereas the hardware that the code is deployed on can keep changing throughout the Testing, Pre-production and Production phases, so that’s the variable part.

The Pre-production phase is essentially simulating the production environment as closely as possible which is called the staging environment and doing stuff like user acceptance testing, alpha/beta testing, etc. whereas the Test phase is for doing unit, integration and system testing.

Need of Change vs Fear of Change

Each department defines its goals based on the division of work. Development and operations are two distinct departments. Often, these departments act like silos because they are independent of each other. The development department may be measured by its speed in creating new features, whereas the operations department may be judged by server uptime and application response time.

Development team struggle for change, whereas operations teams struggle for stability. The conflict between development and operations is caused by a combination of conflicting motivations, processes and tooling, as a result, isolated silos evolve.

In a nutshell, the conflict between development and operations is :

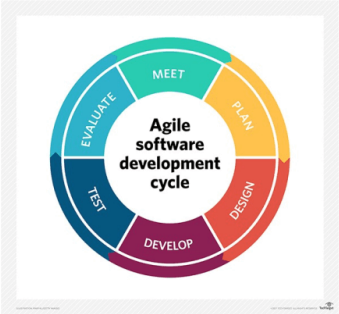
1. **Need for change**: Development produces changes (e.g: new features, bug fixes and work based on change requests). They want their changes rolled out to production.
2. **Fear of change**: Once the software is delivered, the operations department wants to avoid making changes to the software to ensure stable conditions for the production systems.

Agile Methodology + DevOps

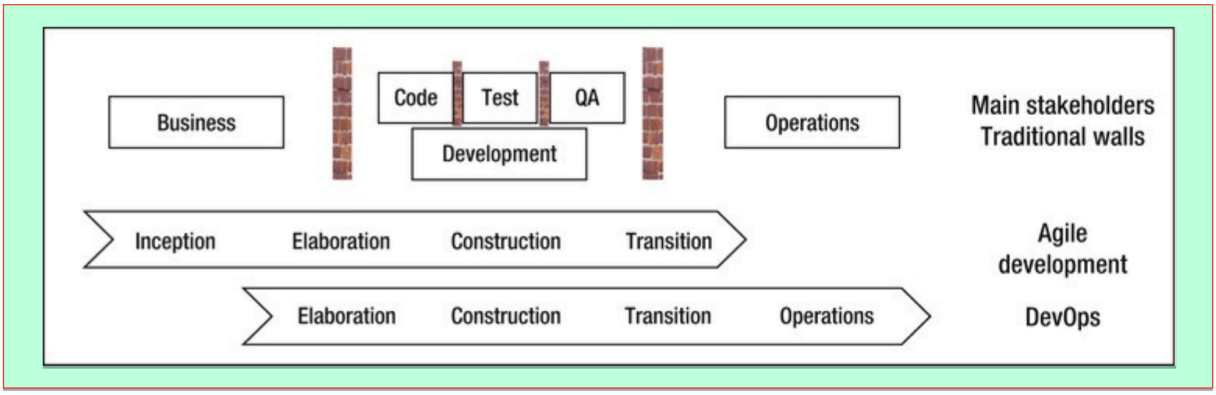
- Developers primarily focus on accelerating the creation of new features by, for instance, adopting Agile methodologies. The Agile movement has brought together programmers, testers, and business representatives.
- Conversely, operations teams are isolated groups that maintain stability and enhance performance by applying practices such as the Information Technology Infrastructure Library (ITIL).
- The principles of Agile methods are focused on defining, building and constructing software.
- DevOps helps development teams increase the frequency of application updates. Agile adoption traditionally omits operations, obstructing the delivery pace.
- DevOps removes this obstruction by applying agile values to the deployment, environment configuration, monitoring and maintenance tasks.

Agile SDLC

1. Inception: In this interval, the vision of the system is developed, a project scope is defined, and the business case is justified.
2. Elaboration: In this interval, requirements are gathered and defined, risk factors are identified, and a system architecture is initialized.
3. Construction: In this interval, the software is delivered to the user.
4. Operations: In this interval, the software is available to the user and is maintained by the operations team.
5. Transition phase is from development to operations.



Traditional vs Agile vs DevOps - Again



- **Agile breaks the wall between Business and Development team.**
- **DevOps breaks the wall between Development and Operations team.**
- **DevOps centers on the concept of sharing: sharing ideas, issues, processes, tools and goals.**

DevOps doesn't include the "Inception" phase mostly because discussion with the business team is typically not considered under the DevOps umbrella however sir did mention that atleast some members of the DevOps team will be present in the Inception meeting where these discussions take place so it is what it is.

Continuous Delivery and Deployment

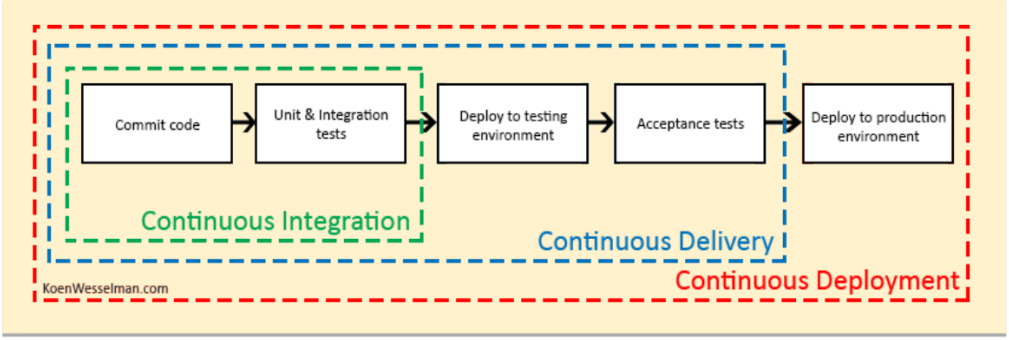
Prerequisite SDLC terms

Error: When a developer makes a mistake while writing code, it's an error.
Defect: When an error is caught by the testing team, it's a defect.
Bug: When a defect is accepted by the dev team as a problem, then it's a bug.

Failure: When a particular build is not upto the specified requirements document then it's a failure.

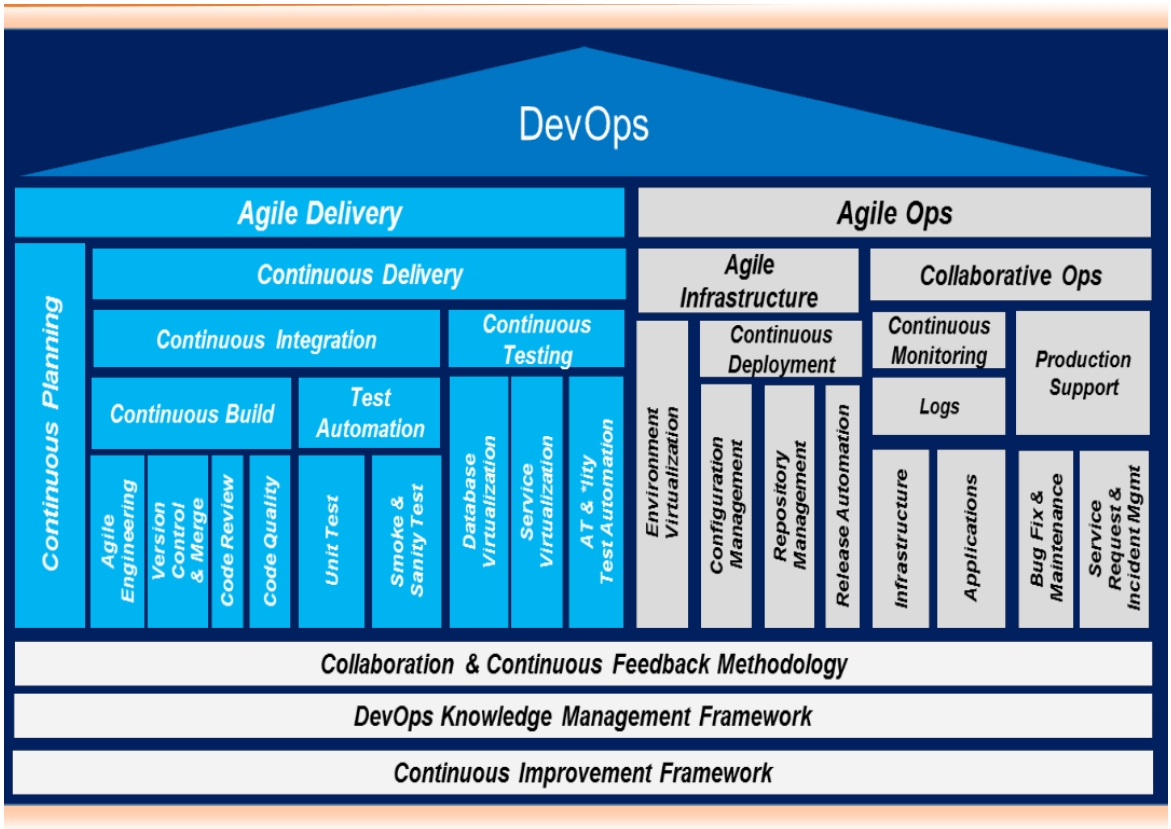
Verification: Whether output of one phase conforms to its previous phase.
Validation: Whether a fully developed system conforms to its SRS document.

Unit testing: Testing of individual units/modules of code.
Integration testing: Testing of integration of two or more units
(as per the ESD course Integration referred to integration of third party frameworks and APIs and Integration testing referred to testing of this integration, so idk what's the actual version).
System testing: Testing the entire system end-to-end
User acceptance testing: System testing done by the customer themselves.



- In Continuous Delivery, the full software delivery life cycle automated till the last environment before production, so that you are ready at any time to deploy automatically to production.
- So, continuous Delivery doesn't mean every change is deployed to production ASAP. It means every change is proven to be deployable at any time
- In Continuous Deployment, you go one step further; you actually automatically deploy to production. The difference is really just whether there is *an automatic trigger or a manual trigger*.

DevOps Architecture



This diagram shows you the entire picture of DevOps and each larger block is the parent of the smaller blocks that come below it.
For example, Continuous Integration consists of Continuous Build + Test Automation.
This concept repeats till you reach the super-parent block of DevOps.

-lity Test => Non-functional testing that tests stuff like reliability, scalability, performance, security, etc.

Smoke Testing is a software testing technique performed post software build to verify that the critical functionalities of software are working fine. It is executed before any detailed functional or regression tests are executed. The main purpose of smoke testing is to reject a software application with defects so that the QA team does not waste time testing broken software applications.

In Smoke Testing, the test cases chose to cover the most important functionality or component of the system. The objective is not to perform exhaustive testing, but to verify that the critical functionalities of the system are working fine.

For example: A typical smoke test would be – Verify that the application launches successfully, Check that the GUI is responsive … etc.

Sanity testing is a kind of Software Testing performed after receiving a software build, with minor changes in code, or functionality, to ascertain that the bugs have been fixed and no further issues are introduced due to these changes. The goal is to determine that the proposed functionality works roughly as expected. If the sanity test fails, the build is rejected to save the time and costs involved in a more rigorous testing.

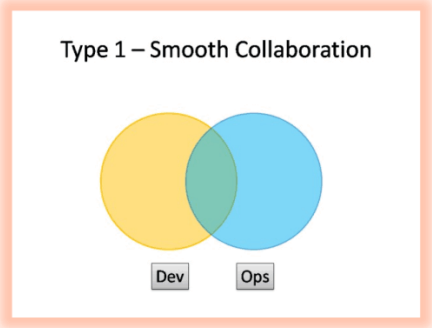
The objective is “not” to verify thoroughly the new functionality but to determine that the developer has applied some rationality (sanity) while producing the software. For instance, if your scientific calculator gives the result of 2 + 2 =5! Then, there is no point testing the advanced functionalities like sin 30 + cos 50.

DevOps Models

- The primary goal of any DevOps setup within an organization is to **improve the delivery of value for customers and the business**.
- So, different companies might need different team structures for effective Dev and Ops collaboration.
- So what is the appropriate team structure for DevOps? Clearly, there is no magic conformation or team topology which will suit every organization.
- However, it is useful to characterize a small number of different models for team structures, some of which suit certain organizations better than others.
- Models:
  - 1. Smooth Collaboration
  - 2. Fully Embedded
  - 3. Infrastructure as a Service
  - 4. DevOps as a Service
  - 5. Temporary DevOps Team

Smooth Collaboration Model

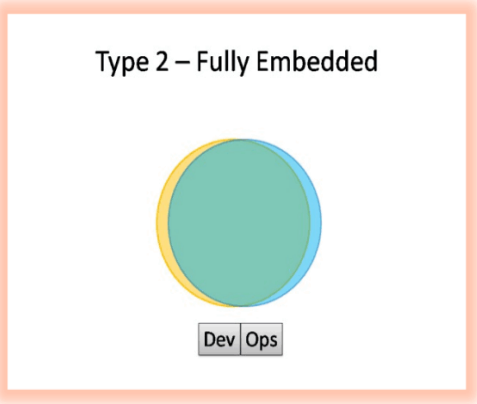
- This type has smooth collaboration between Dev teams and Ops teams, each specializing where needed, and also sharing where needed.
- Dev and Ops must have effective shared goal that may be 'Delivering Reliable, or Frequent Changes'.
- Ops team must be comfortable working with Devs and get to grips with test-driven coding and Git, and Devs must take operational features seriously and seek out Ops people for input into logging implementations, and so on.
- Type 1 is suitable for an organization with strong technical leadership and the potential effectiveness is high.



[The Venn diagrams in these models are really helpful in understanding the essence of what a model is talking about]

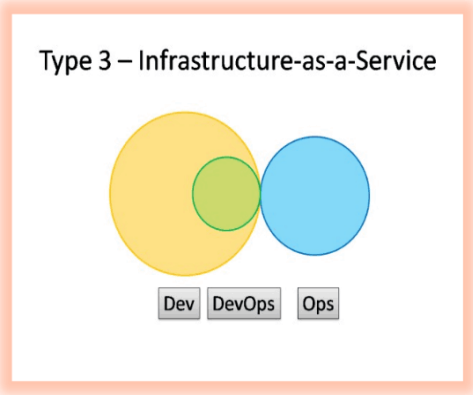
Fully Embedded Model

- Where operations people have been fully embedded within product development teams in Type 2 topology.
- There is so little separation between Dev and Ops that all people are highly focused on a shared purpose.
- The *Fully Embedded* topology might also be called '**NoOps**', as there is no distinct or visible Operations team.
- This type is used by companies like Netflix and Facebook with a single web-based product have achieved.
- It is suitable for companies with a single main web-based product or service and it's potential effectiveness is also high.



Infrastructure as a Service - IaaS Model

- This may be useful for Companies with a traditional IT Operations department which will not change rapidly. Or the companies run all their applications in the public cloud (Amazon EC2, OpenStack, Azure, etc.).
- This type helps to treat Operations as a team who simply provide the elastic infrastructure on which applications are deployed and run; the internal Ops team is thus directly equivalent to Amazon EC2, or Infrastructure-as-a-Service.
- A team (perhaps a virtual team) within Dev then acts as a source of expertise about operational features, metrics, monitoring, server provisioning, etc., and probably does most of the communication with the IaaS team. This team is still a Dev team.
- This type is suitable for Companies with several different products and services, with a traditional Ops department or the companies whose applications run entirely in the public cloud. The Potential effectiveness of this type is MEDIUM.

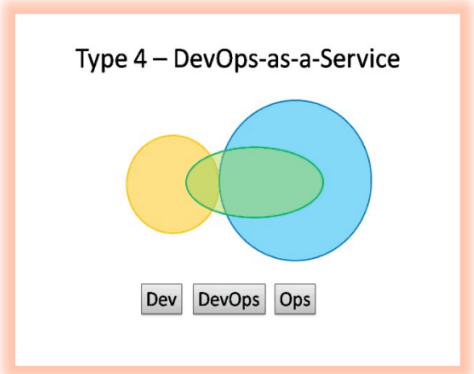


33

Do note that in this case you are essentially outsourcing your Ops team and they are not your employee even though your employees will work in collaboration with them which is why this is not the same as the Smooth Collaboration model.

DevOps as a Service Model

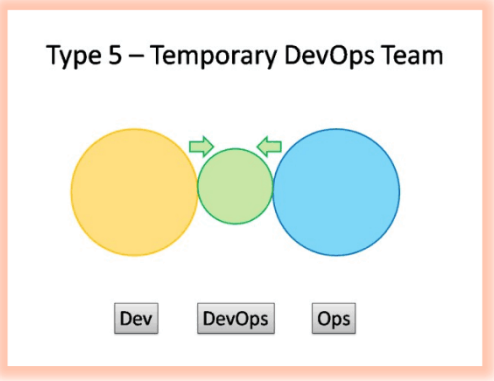
- This type is suitable for some organizations, particularly smaller ones, may not have the budget, experience, or employees to take a lead on the operational aspects of the software they produce.
- The Dev team might then reach out to a service provider like OpenStack (Cloud Computing Company) to help them build test environments and automate their infrastructure and monitoring, and advise them on the kinds of operational features to implement during the software development cycles.
- So, *DevOps-as-a-Service* topology could be a useful and realistic way for a small organization or a team in a project to learn about automation, monitoring, and configuration management, and later move towards a Type 1 for *Smooth Collaboration* model when they grow and add more team members for operational focus.
- This type is good for smaller teams or organizations with limited experience of operational issues and it's Potential effectiveness is MEDIUM.



34

Temporary DevOps Team Model

- The members of the temporary team will 'translate' between Dev-speak and Ops-speak. If you start to see the value of bringing Dev and Ops together, then the temporary team has a real chance of achieving its aim.
- This type can be used as a precursor of type 1 (smooth collaboration) and it's potential effectiveness is initially low and once it's transformed to Type 1 then its potential effectiveness is high.



- Exactly which DevOps team structure or topology will suit an organization depends on several things. The discussed topologies so far, are meant as a reference guide or experimental for assessing which patterns might be appropriate in your project.
- But in reality, a combination of more than one pattern, or one pattern transforming into another, will be the best approach.

Components of Software Delivery

$$T_{DELIVERY} = \frac{T_{PLAN} + T_{DESIGN} + T_{DEVELOP} + T_{BUILD} + T_{DEPLOY} + T_{TEST} + T_{FIX} + T_{RELEASE} + T_{EVALUATE}}{\%\ TRUST\ (0\text{-}1)}$$

- $T_{DELIVERY}$ is optimized by minimizing the time for each of the tasks required to complete the delivery. They estimate the time to do each part and then the total is the sum of the parts.
- *Productivity is inversely proportional to the number of dependencies in a release.*
- ***Trust is inversely proportional to the number of dependencies in a release.***
- The important theme of DevOps is that the entire development-to-operations lifecycle must be viewed as one end-to-end process.

DevOps Principles

1. DevOps is the practice of operations and development engineers participating together in the entire service lifecycle, from design through the development process to production support.

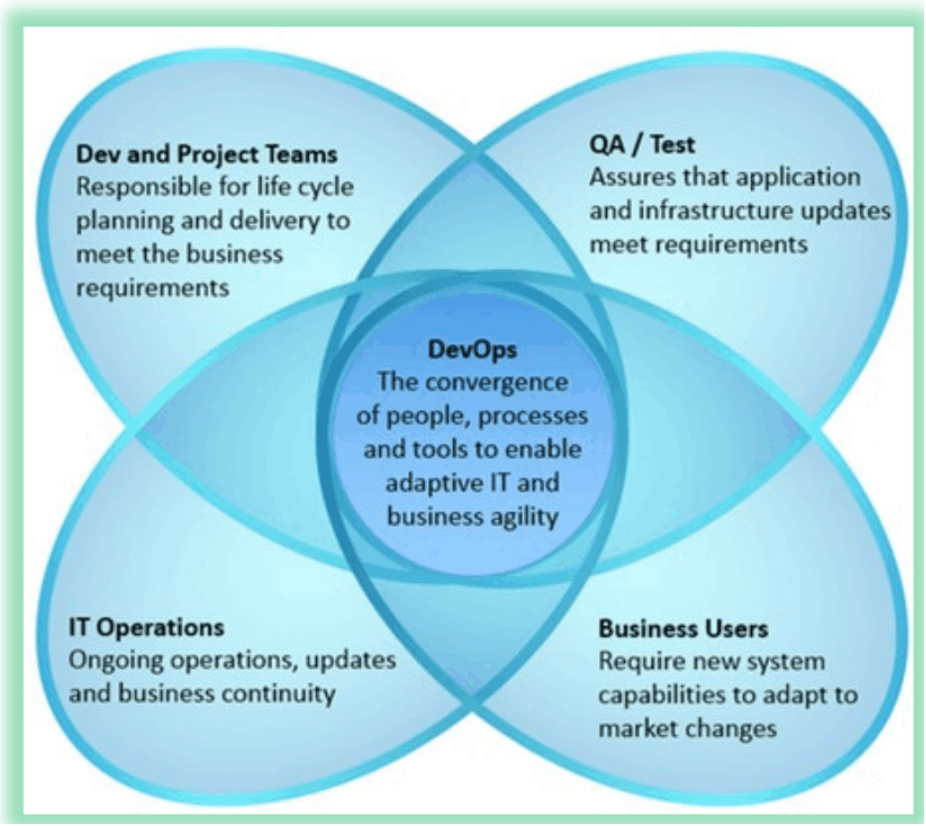
2. DevOps is also characterized by operations staff making use of many of the same techniques/tools as developers for their work.

3. DevOps principles involves CALMS:
    Culture - People -> Process -> Tools
    Automation - Infrastructure as Code
    Lean - Small batch sizes, focus on value for end-user
    Measure - Measure everything & show improvements
    Sharing - Collaboration between Dev and Ops

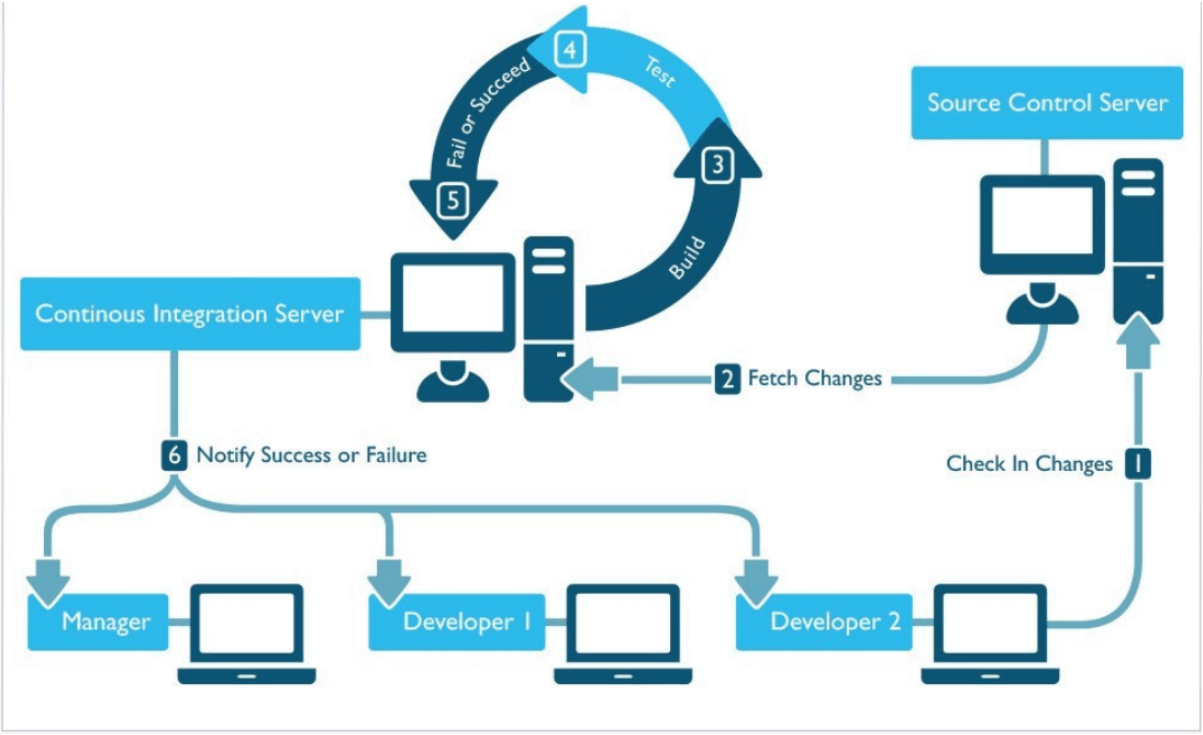
DevOps Practices

1. Version control for all.
2. Automated testing
3. Proactive monitoring and metrics
4. Configuration management
5. Continuous Integration/Deployment/Delivery
6. Virtualization/Cloud/Containers
7. Toolchain approach

Roles and Responsibilities of Teams



SDLC Automation



Removing DevOps Barriers

Barriers as in, “Dev team want change, Ops team want stability” and other good stuff like that.

Culture: Respect -if there is only one thing you do

- Don't stereotype
- Respect other people's expertise, opinions and responsibilities.
- Don't just say "No"
- ***Don't hide things***

Developers: Talk to ops about the impact of your code:

- what metrics will change, and how?
- what are the risks?
- what are the signs that something is going wrong?

This means Dev needs to work this out before talking to ops.

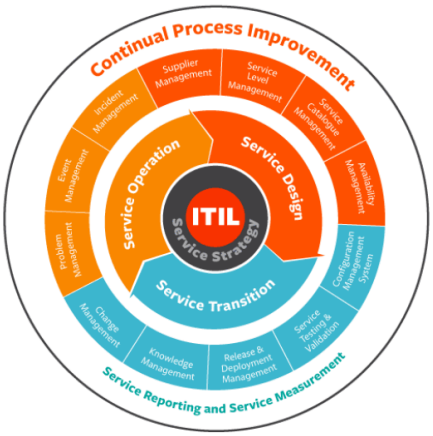
The Ops team needs to trust the Dev team to involve them on feature discussions.
The Dev team needs to trust the Ops team to discuss infrastructure changes.
Everyone needs to trust that everyone else is doing their best for the business.

Ops team should be transparent and give the Devs team access to systems.
Healthy attitude about failure.
Avoiding blame, i.e. no fingerpointing
Require Ops who think like Devs and Devs who think like Ops

Key Benefits of all this,
Faster release of apps with automation of integrated build, test and deployment process.
Increase developer and operational efficiency by managing your Infrastructure as Code.
Improve customer experience with immediate feedback loops and continuous improvement.

Popular Myths/Misconceptions related to DevOps

- **DevOps is Development + Operations**
  - Modern application lifecycle has many other teams involved like Business , Senior Management, partners, Testers , Compliance etc.
- **DevOps is all about using tools & Automation**
  - Surprise !!! There is No DevOps tool !!! You have tools for CI/CD and Operations. Tools and automation are part of DevOps culture.
- **DevOps is a Skill - DevOps Engineer , DevOps Manager**
  - Replace the word “DevOps” with Agile . Does it make sense ? Like Agile , you can have DevOps Team not an DevOps Engineer or Manager
- **DevOps disrupts existing processes like ITIL**
  - It complements & enhances these processes. DevOps still uses the same principles but with an enhanced scope.



ITIL, an acronym for Information Technology Infrastructure Library, is a set of detailed practices for IT service management that focuses on aligning IT services with the needs of business

- **Everybody can / should do 10 deploys a day (or any other exotic number) with DevOps**
  - No , only if there is a business need and supporting eco system.
- **Is for the cloud or web shops only**
  - A DevOps approach is applicable to any infrastructure but a cloud infra gives you an additional benefit
- **It is THE solution – a silver bullet**
  - Thinking a DevOps strategy will solve all issues in simple manner. It’s a collaborative & Iterative approach to solving issues pertaining to people , process and technology to achieve streamlined Flow
- **Finally, Only Agile Projects can do DevOps**
  - It is applicable to any development model.

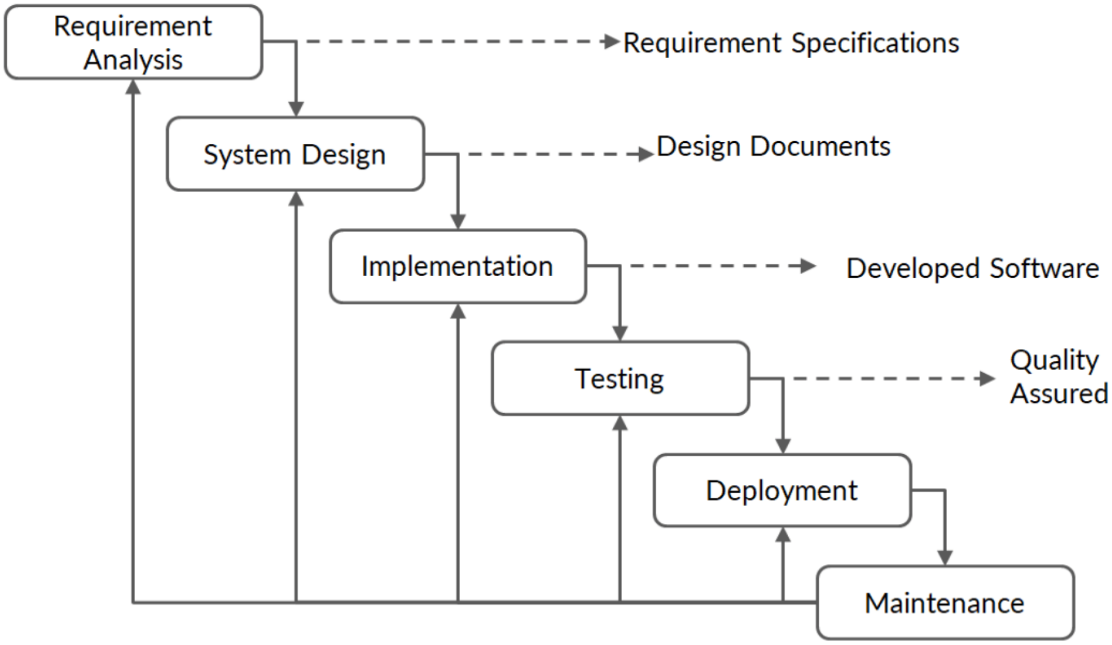
20/01/22: TA Session

Advanced Git + GitHub usage with Hands-on Pull Requests + Forks + Review and a bit of an overview of GitHub Actions and Marketplace and how GitHub can be used for the full CI/CD pipeline.

25/01/22: Lecture

Module 2: CI/CD

SDLC



Requirement Specifications => SRS document (Software Requirement Specification)

Introduction to CI/CD

- In software engineering, Continuous Integration (CI) is defined as a process of integrating all developers' code to a shared repository in a version-control system (VCS) frequently.

-The term CI was first proposed by Grady Booch in 1991.

- CI is supposed to be used in conjunction with automated build, test and QA.

- A build server compiles the code and reports the results to the developers on a regular basis or even after each commit.

- Unit testing in the developer's local environment before committing to the mainline helps avoid one developer's work-in-progress while breaking another developer's copy.

- Along with running the unit and integration checks, you can check the quality of your code and profile performance and format the documentation from the source code, thereby facilitating QA processes.

- This quality check application mainly aims to enhance software quality and reduce delivery time continuously.

- CI is connected closely with Continuous Delivery or Continuous Deployment which is also called the CI/CD pipeline.

- Continuous Delivery ensures that an incremental feature of a software product checked in on the mainline is ready to deliver to the end user.

- Continuous Deployment makes the deployment process fully automated.

[More explanation on CI/CD has already been covered in module 1]
[Traditional vs Agile vs DevOps - Has already been covered twice in module 1]

Continuous Integration

- CI in its simplest form, involves a tool that continuously monitors VCS for any new changes.

- Whenever any new changes are pushed, this tool starts compiling and testing your application.

- If a bug is found or the code fails, the developers are immediately notified to fix the issue.

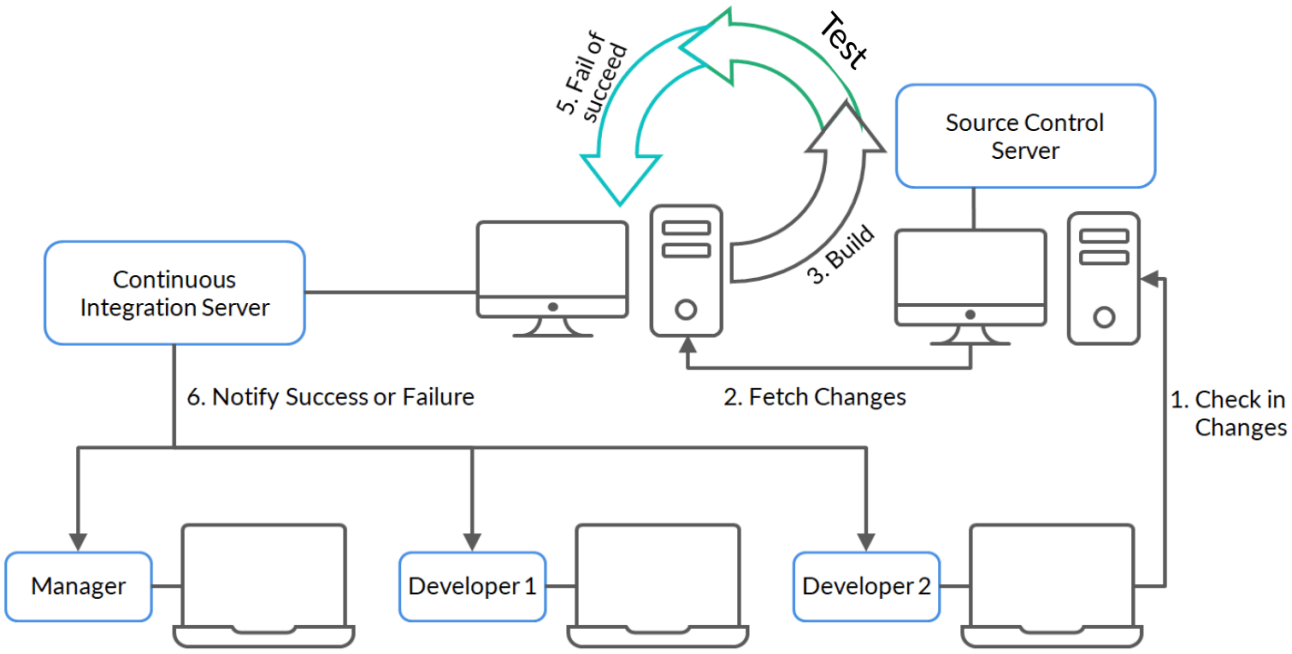
- In DevOps the typical ideology is that "If you are going to fail, then fail fast so that we can immediately fix it", so catching bugs or failures as fast as possible is the game here.

- If you integrate it with automated end-to-end acceptance checks, CI can serve as a feedback method, offering a straightforward image of the current state of development efforts.

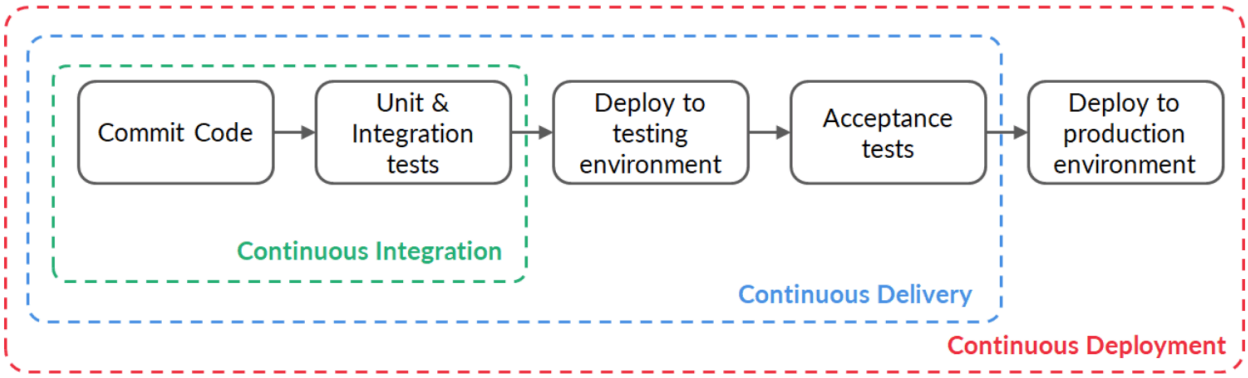
- It will allow you to deploy the latest version of your application either automatically or as a one-click process.

Diagram

[This same diagram was shown before as well but this colour scheme looks better :)]



CI/CD Overview



This was done already but here you go again :(

Benefits of CI/CD

- Automation: ensures build, test, QA, delivery and deployment
- Improves consistencies and quality of code
- Delivers new features quickly to the end user
- Increases product visibility
- Helps avoid manual errors
- Helps fix any issues that may arise
- Reduces cost and labour

Introduction to Jenkins

- It is a free and open-source automated server.
- It can be used to automate all types of software development tasks such as building, testing and delivering or deploying software.
- Jenkins can be installed via native system packages, Docker or it can be run standalone on any computer.

Installing pre-installed and pre-configured Jenkins

This can be done by installing the official Docker image (container) for Jenkins.
Go to “Docker Hub” website and search for jenkins and that has the instructions for installing the Docker image for Jenkins and you can quickly get Jenkins up and running on your local system. But ofcourse you’ll need Docker for this, so first learn that and then come back to this.

History of Jenkins

- Jenkins is the outcome of an innovative developer, Kohsuke Kawaguchi, who began this project as a hobby under the name Hudson in late 2004 while working at Sun Microsystems.
- In 2009, Sun Microsystems was acquired by Oracle. At the end of 2010, tensions arose between the Hudson developer community and Oracle. Initially, the problem was with the Java.net infrastructure, which later worsened due to issues related to Oracle's claim to the Hudson trademark.
- In January 2011, the Hudson developer community decided to rename the project as Jenkins. Subsequently, they migrated the original Hudson code base to a new GitHub project and continued their work there.
- After this, the majority of users joined the Jenkins developer community, migrating to Jenkins.

So, Hudson is the Enterprise edition that belongs to Oracle and Jenkins is the Community edition that’s fully open source.

Features of Jenkins

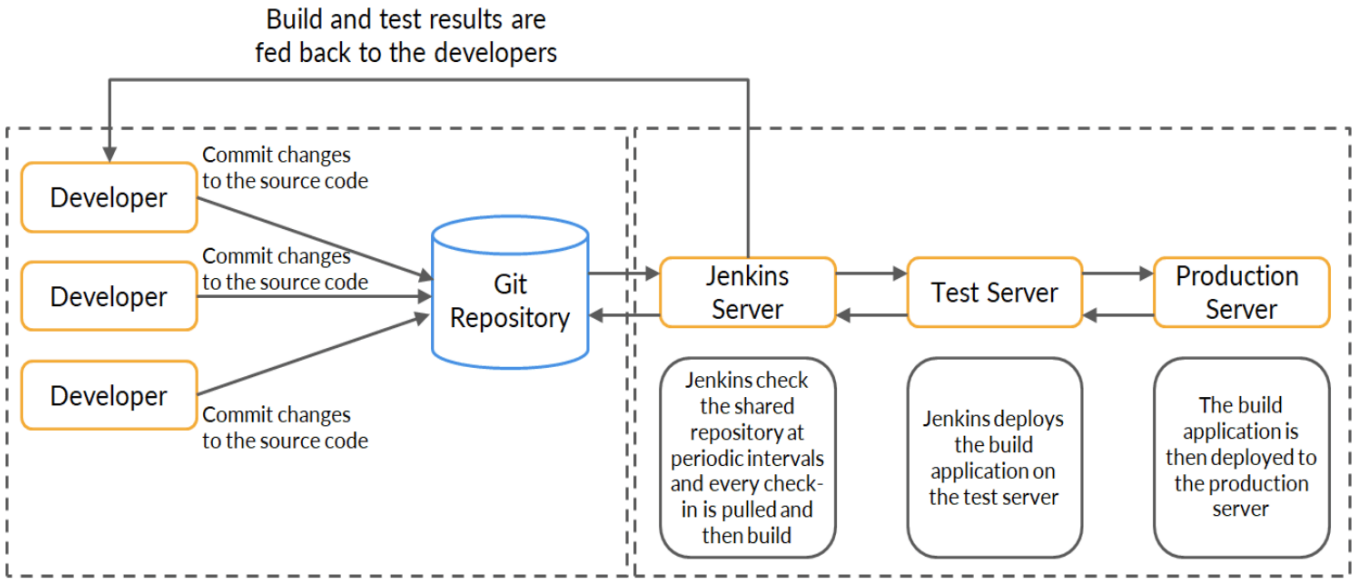
- Open-source tool written in Java (so it is OS independent).
- Very user-friendly tool with a simple, intuitive and visually appealing user interface.
- Very low learning curve (but for complex stuff, you’ll require significant practice).
- An extremely flexible tool and hundreds of open-source plugins are available, with more coming up every week.
- These plugins cover everything from VCS, build tools, code quality metrics and build notifiers to integration with external systems, UI customisations and much more.

- The Jenkins community is a large, dynamic, reactive and welcoming bunch with passionate champions, active mailing lists, IRC (Internet Relay Chat) channels (basically glorified WhatsApp group :P) and a very vocal blog and Twitter account.
- The pace of development is quite fast with the latest new features, bug fixes and plugin updates being released on a weekly basis.
- **Jenkins is a Java application so it is platform independent.**

Comparison of CI Tools

	Jenkins	TeamCity	Bamboo	Travis	Circle	Codeship
Pricing	Free	$299-$1999	$10-$800	$69-$489	$50-$3150	$75-$1500
Operating system	Windows, Linux, macOS, any Unix-like OS	Windows, Linux, macOS, Solaris, FreeBSD and more	Windows, Linux, macOS, Solaris	Linux, macOS	Linux, iOS, Android	Windows, macOS
Hosting	On premise/cloud	On premise	On premise/ Bitbucket as cloud	On premise/cloud	Cloud	Cloud
Container support	✓	✓	✓	✓	✓	Yes for Pro version
Plugins	*****	****	**	****	***	****
Docs and support	Adequate	Good	Good	Poor	Good	Poor
Learning curve and usability	Easy	Medium	Medium	Easy	Easy	Easy
Use case	For big Projects	For enterprise needs	For Atlassian integrations	For small projects and startups	For fast development and high budget	For any project

Jenkins Architecture



Jenkins Installation

Install Java
sudo apt-get update
sudo apt install -y openjdk-11-jdk

To check: java - - version
openjdk version "11.0.11" 2021-04-20
OpenJDK Runtime Environment (build 11.0.11+9-Ubuntu-0ubuntu2.18.04)
OpenJDK 64-Bit Server VM (build 11.0.11+9-Ubuntu-0ubuntu2.18.04, mixed mode, sharing)

Install Jenkins
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

Install ca-certificates (man 8 update-ca-certificates)
sudo apt install ca-certificates

Install Jenkins
sudo apt-get update
sudo apt-get install jenkins

To check Jenkins version: vim /var/lib/jenkins/config.xml
To copy admin password: sudo cat /var/lib/jenkins/secrets/initialAdminPassword

22

The “wget” command is fetching the public GPG (GNU Privacy Guard) key for Jenkins which ensures that what we are downloading is legit and the “apt-key add” command adds this public key to our system.

The “sudo sh -c” command adds the location of Jenkins binary stable version to the sources list which is where the apt-get commands look at.
This is why the “sudo apt-get install jenkins” command even works.

Then fetch IP address using “ifconfig” command and then go to “IPAddress:8080” in the browser to visit Jenkins dashboard.
Push the admin password in that dashboard where it asks for the admin password.
Then install the plugins that you want, I just did “Install suggested plugins”

Then you create a Jenkins account and you can proceed using the instructions.

Create a New project by clicking the “New Item” button on the left-side menu.
Follow the steps on there and make a Freestyle project with “echo “Hello World”” as the build action. Other than that it’s all up to you to get familiar with using the Jenkins system.

Know the status of Jenkins: sudo service jenkins status.
To start Jenkins: sudo service jenkins start.
To stop Jenkins: sudo service jenkins stop.
To restart Jenkins: sudo service jenkins restart. (stop and start)

Jenkins home directory:

```
ubuntu@ip-172-31-81-117:~$ sudo su - jenkins
jenkins@ip-172-31-81-117:~$ pwd
/var/lib/jenkins
jenkins@ip-172-31-81-117:~$
```

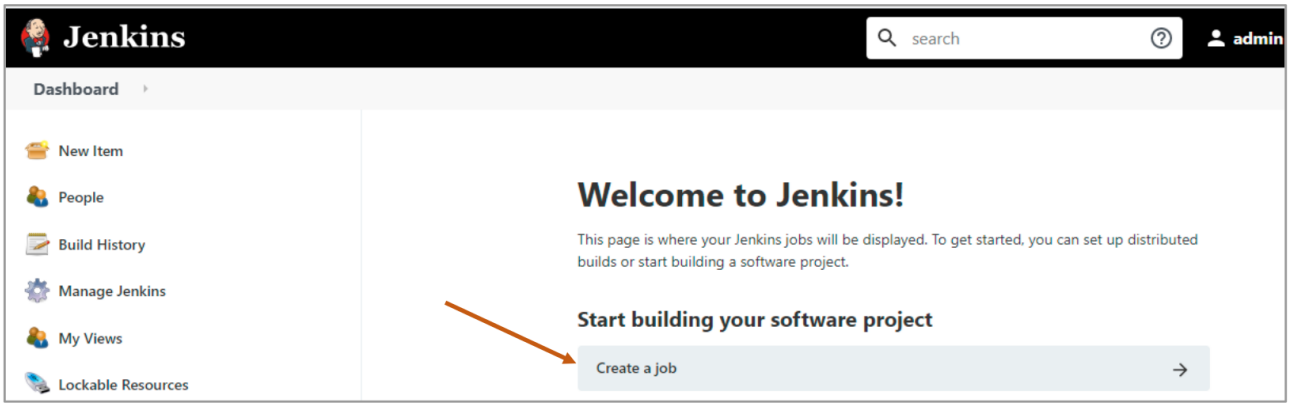
Jenkins home directory: /var/lib/jenkins
It contains the details of the Jenkins server configuration, which is used to configure in the Manage Jenkins. The core configuration files are stored in config.xml.

It contains two important subdirectories: **jobs and workspace**.
Jobs directory contains configuration details of the build job in config.xml file.
Jenkins builds the given project in the workspace directory. Each project has its own directory in the workspace.

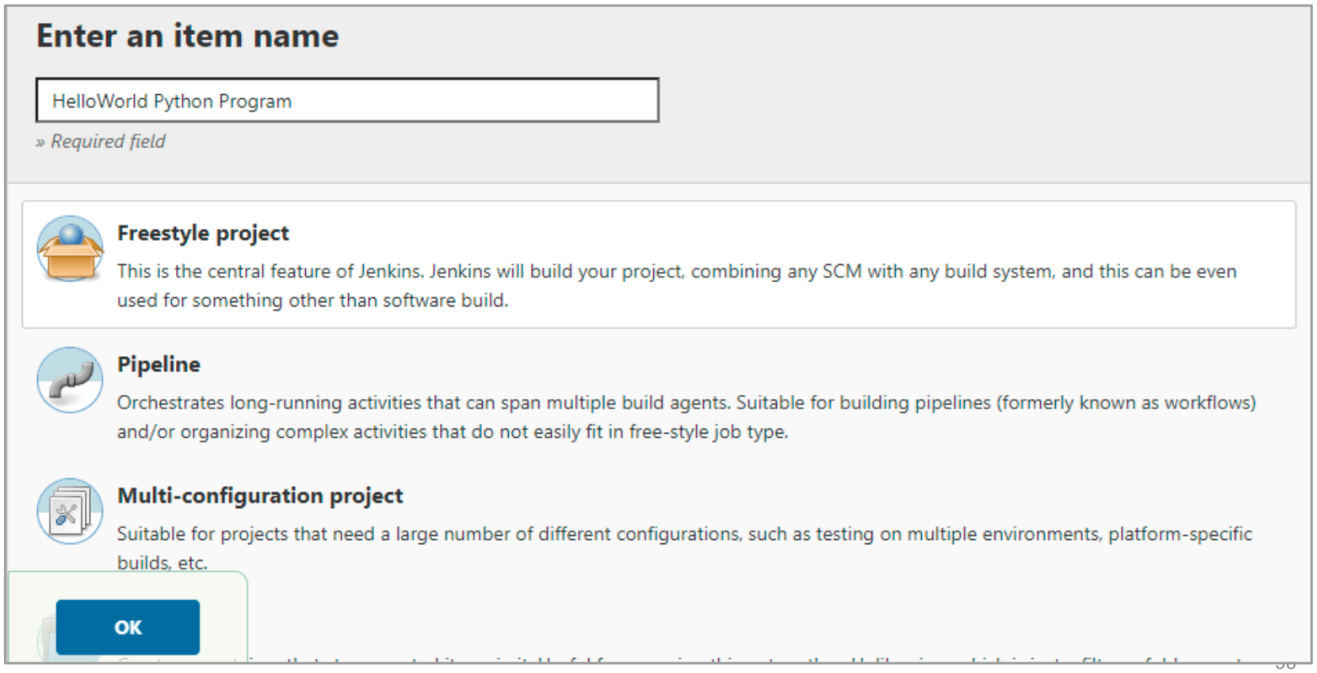
Jenkins Hands-on

Jenkins Job

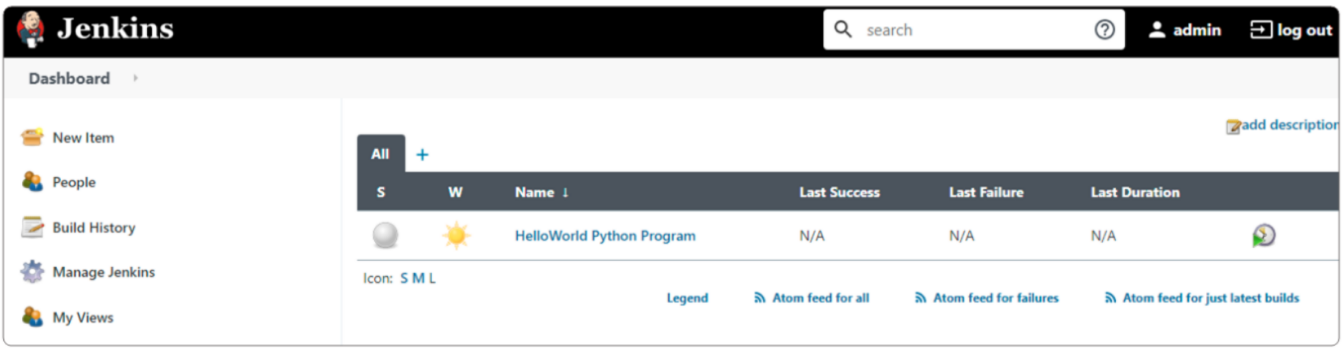
Click to Create a Job



Freestyle Project

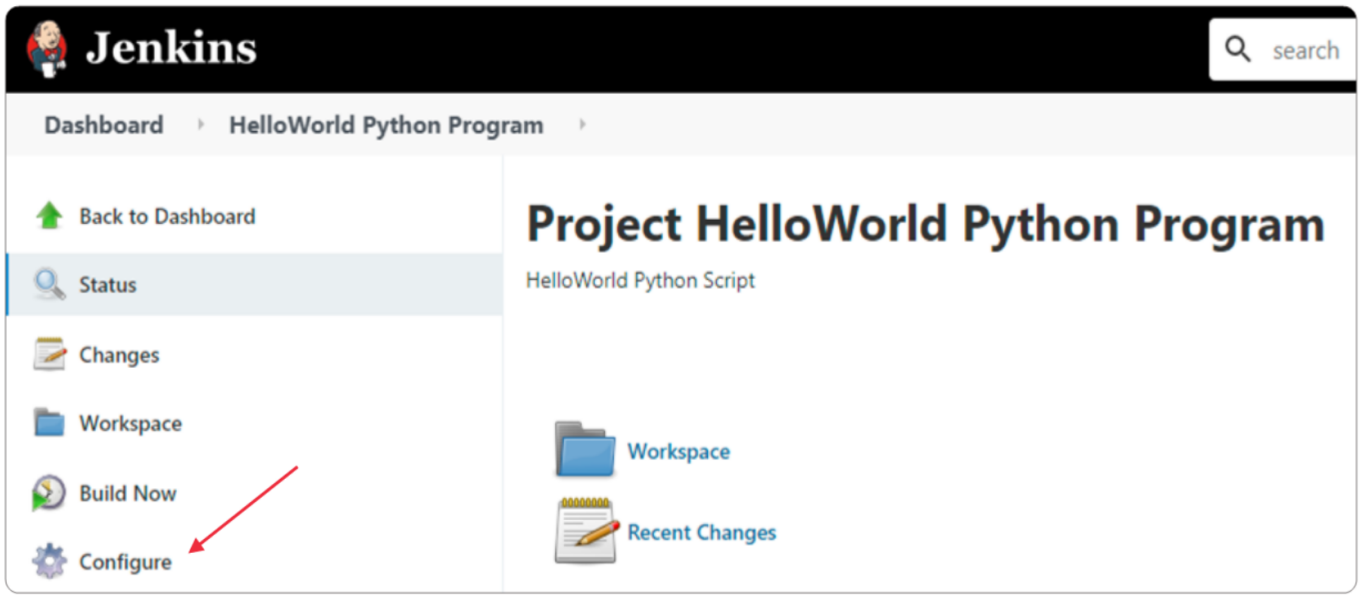


Example: HelloWorld Python Program

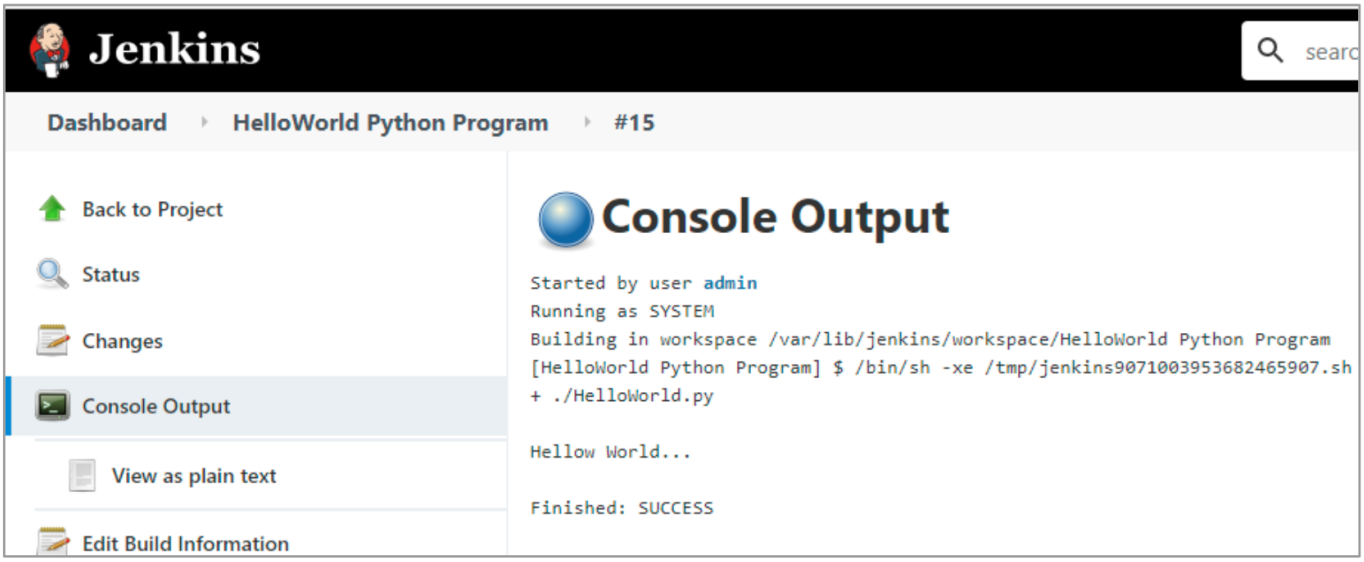



```
#! /usr/bin/python3
# This Phython program will print Hello World...
print("Hello World ...\n")
```

Build the Project



Console Output



Check Log Files

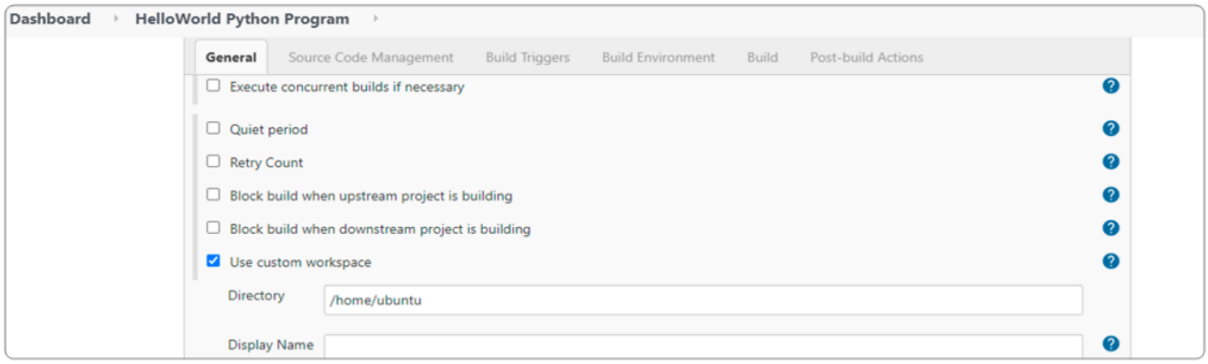
```
ubuntu@ip-172-31-81-117:/var/lib/jenkins/jobs/HelloWorld Python Program/builds$ ls
1  10  2  3  4  5  6  7  8  9  legacyIds  permalinks
ubuntu@ip-172-31-81-117:/var/lib/jenkins/jobs/HelloWorld Python Program/builds$ cd 4
ubuntu@ip-172-31-81-117:/var/lib/jenkins/jobs/HelloWorld Python Program/builds/4$ ls
build.xml  changelog.xml  log
ubuntu@ip-172-31-81-117:/var/lib/jenkins/jobs/HelloWorld Python Program/builds/4$ cat log
Started by user ha:////4O1Xwh1EUGVcmahxd0uDY/VpeIbwK+Jgej0qfQuwtGnlAAAAlx+LCAAAAAAAAP9b85aBtbiIQTGjNKU4P08vOT+v
1x6OyILUoJzMv2y+/JJUBAhiZGBgqihhk0NSjKDWzXb3RdlLBUSYGJk8GtpzUvPSSDB8G5tKinBIGIZ+sxLJE/ZzEvHT94JKizLx0a6BxUmjGOU
gZu/dLi1CL9xJTczDwAj6GcLcAAAAA=admin
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/HelloWorld Python Program
[HelloWorld Python Program] $ /bin/sh -xe /tmp/jenkins10839759464246091234.sh
+ echo Hello World
Hello World
Finished: SUCCESS
ubuntu@ip-172-31-81-117:/var/lib/jenkins/jobs/HelloWorld Python Program/builds/4$
```

Place the py file wherever you want.
While creating “New Item” in Jenkins, select Build Action as “Executing shell script”, in the text box that opens, type, “cd /absolute/path/to/py/file” and on next line type “python3 py_file_name.py”
And run “Build Now”, then you can see output in the “Console Output” section.

Using a Custom Workspace for Job

Basically the directory for the Job’s Workspace is now provided by you (no other notable changes as such).

Custom Workspace




```
ubuntu@ip-172-31-81-117:/var/lib/jenkins/jobs/HelloWorld Python Program/builds$ ls
1  10  2  3  4  5  6  7  8  9  legacyIds  permalinks
ubuntu@ip-172-31-81-117:/var/lib/jenkins/jobs/HelloWorld Python Program/builds$ cd 9
ubuntu@ip-172-31-81-117:/var/lib/jenkins/jobs/HelloWorld Python Program/builds/9$ cat log
Started by user ha:////4O1Xwh1EUGVcmahxd0uDY/VpeIbwK+Jgej0qfQuwtGnlAAAAlx+LCAAAAAAAAP9b85aBtbiIQTGj
1x6OyILUoJzMv2y+/JJUBAhiZGBgqihhk0NSjKDWzXb3RdlLBUSYGJk8GtpzUvPSSDB8G5tKinBIGIZ+sxLJE/ZzEvHT94JKizL
gZu/dLi1CL9xJTczDwAj6GcLcAAAAA=admin
Running as SYSTEM
Building in workspace /home/ubuntu
[ubuntu] $ /bin/sh -xe /tmp/jenkins2121288571684403099.sh
+ ./HelloWorld.py

Hellow World...

Finished: SUCCESS
ubuntu@ip-172-31-81-117:/var/lib/jenkins/jobs/HelloWorld Python Program/builds/9$
```

Custom Workspace: Console Output

Note: Sometimes for executing particular shell scripts in this job, the ".sh" file's owner must be changed to Jenkins and "rwx" permissions must be given to the owner before executing the job.

You can place the py file directly into the Custom Workspace directory and Jenkins will directly recognize it without any “cd” command to change the working directory.

Running Jenkins remotely

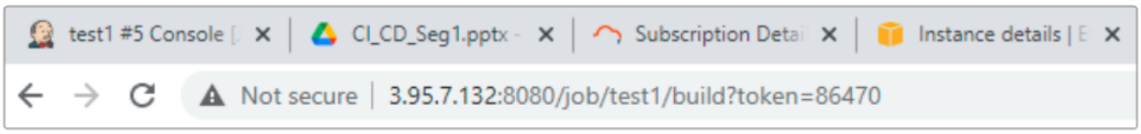
Build Jenkins Job Remotely

Step 1: Choose the project.
Step 2: Configure – Build – enable Trigger builds remotely.
Step 3: Modify: JENKINS_URL/job/test1/build?token=TOKEN_NAME
http://3.95.7.132:8080/job/test1/build?token=86470.
Step 4: Open a browser and enter: http://3.95.7.132:8080/job/test1/build?token=86470
Step 5: Verify the build number.

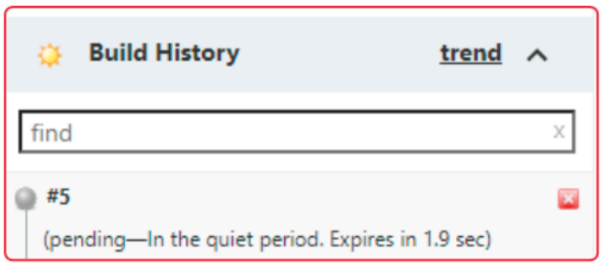
Type a Authentication Token yourself and this will be a secret key of sorts in order to trigger the remote build.
JENKINS_URL can be found by doing ifconfig on your device and then you can use this IP to trigger the build from any place, in my case it was,
http://192.168.1.106:8080/job/PythonBuildProgram/build?token=86470
You will have to login to your Jenkins admin user account in order to trigger the build.

Trigger Build Job by URL

Open a browser, type the URL and enter



Check Build History in the Jenkins Server



Build by Remote Host – Check Console Output

Manual Build

Console Output

```
Started by user admin
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/test1
[test1] $ /bin/sh -xe /tmp/jenkins17988033407760365594.sh
+ cd ../../
+ echo The total number of directories in the /var/lib/jenkins
The total number of directories in the /var/lib/jenkins
+ wc -l
+ grep ^d
+ ls -Rl
676
Finished: SUCCESS
```

Trigger Build Remotely

Console Output

```
Started by remote host 122.181.192.30
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/test1
[test1] $ /bin/sh -xe /tmp/jenkins1040342982316183531.sh
+ cd ../../
+ echo The total number of directories in the /var/lib/jenkins
The total number of directories in the /var/lib/jenkins
+ wc -l
+ grep ^d
+ ls -Rl
678
Finished: SUCCESS
```

So you can thus use the remote URL to trigger the build even if you don't have access to the machine that hosts the build files.

This help snippet from the Jenkins Build menu should enlighten you

“Enable this option if you would like to trigger new builds by accessing a special predefined URL (convenient for scripts).

One typical example for this feature would be to trigger a new build from the source control system's hook script, when somebody has just committed a change into the repository, or from a script that parses your source control email notifications.

You'll need to provide an authorization token in the form of a string so that only those who know it would be able to remotely trigger this project's builds.

This is most useful when your Jenkins instance grants read access to this job to anonymous users.

When that's not the case, Jenkins will reject requests sent to the trigger URL even when the correct token is specified.

To solve this, the HTTP requests need to be authenticated as a user with the necessary read permission for the job — but then you could probably just grant this user the permission to build this anyway.

Another option is to use the Build Token Root Plugin, that provides additional URL endpoints to trigger builds using this token, and doesn't require the otherwise necessary Overall/Read and Job/Read permissions to do so.”

01/02/22: Lecture

Job Chaining

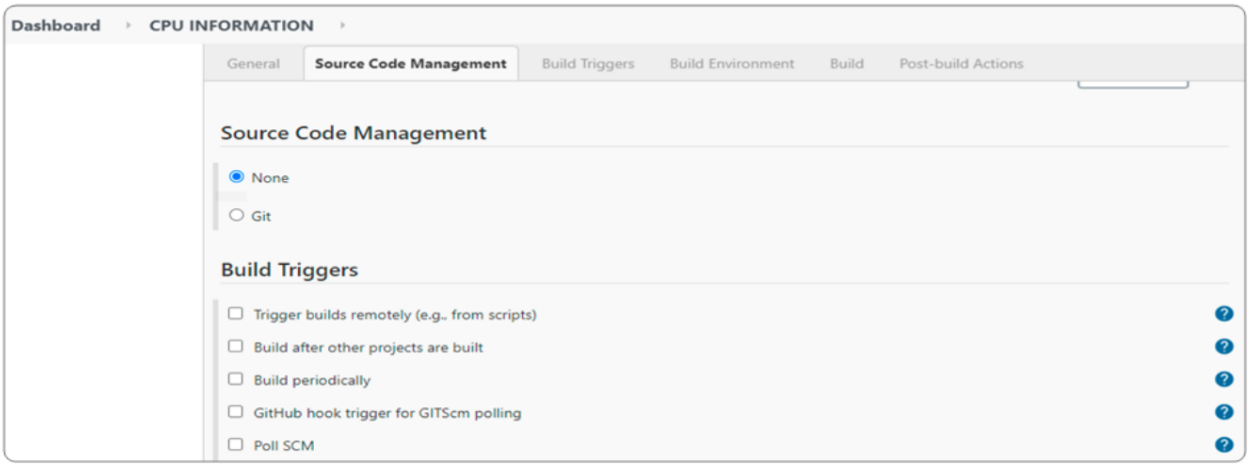
- Create 3 Jobs:
  - Job1: CPU INFORMATION
  - Job2: RAM INFORMATION
  - Job3: DISK INFORMATION

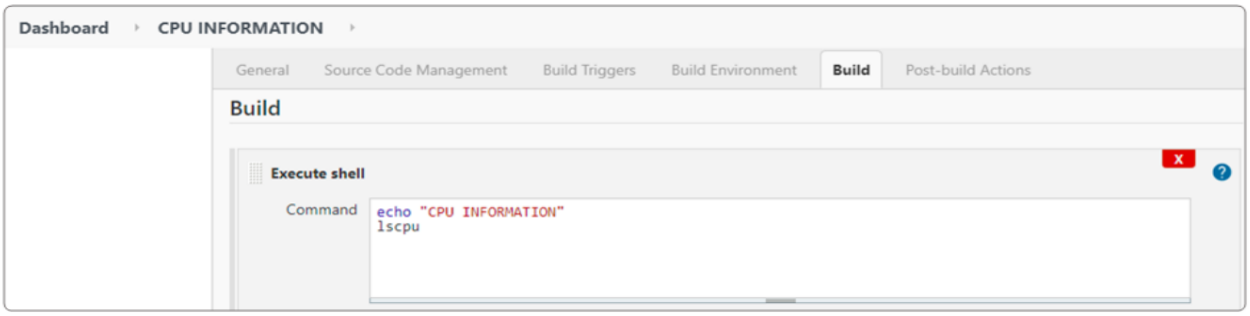
- Job2 will start after Job1 build is successful
- Job3 will start after job2 build is successful

So in summary,

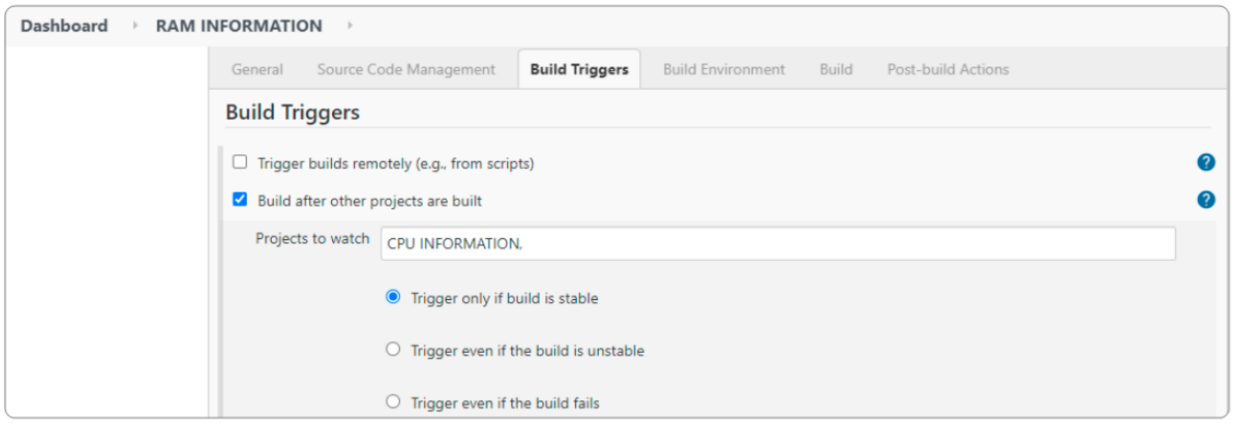
- Job1 has 2 Downstream Jobs
- Job2 has 1 Upstream Job and 1 Downstream Job
- Job3 has 2 Upstream Jobs

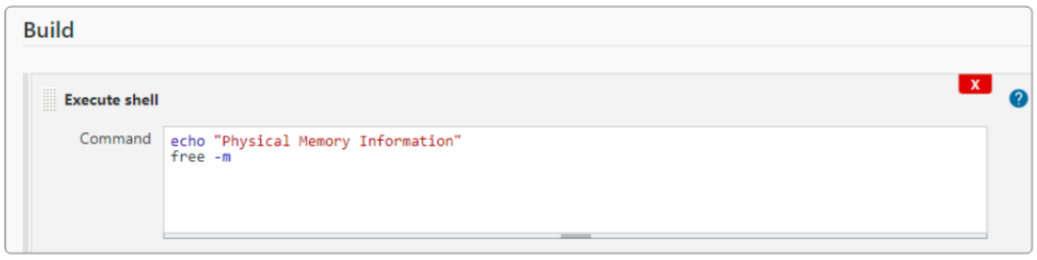
Job1 Configuration



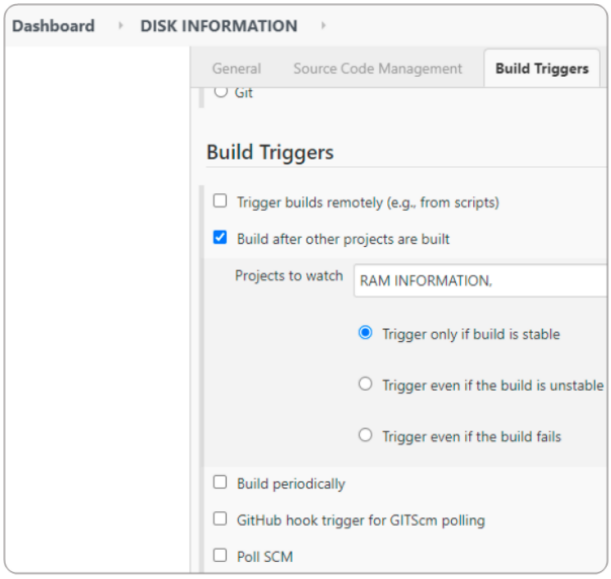


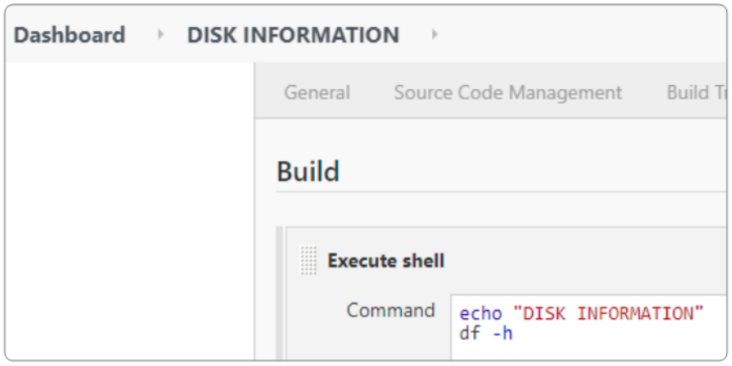
Job2 Configuration





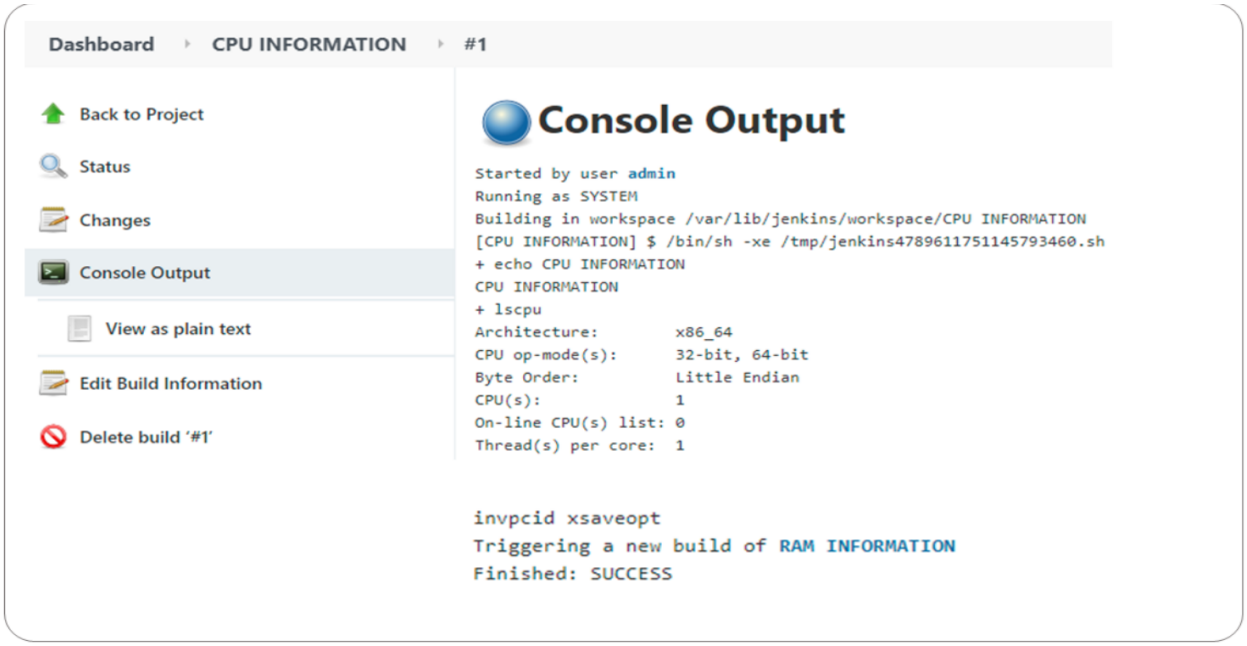
Job3 Configuration

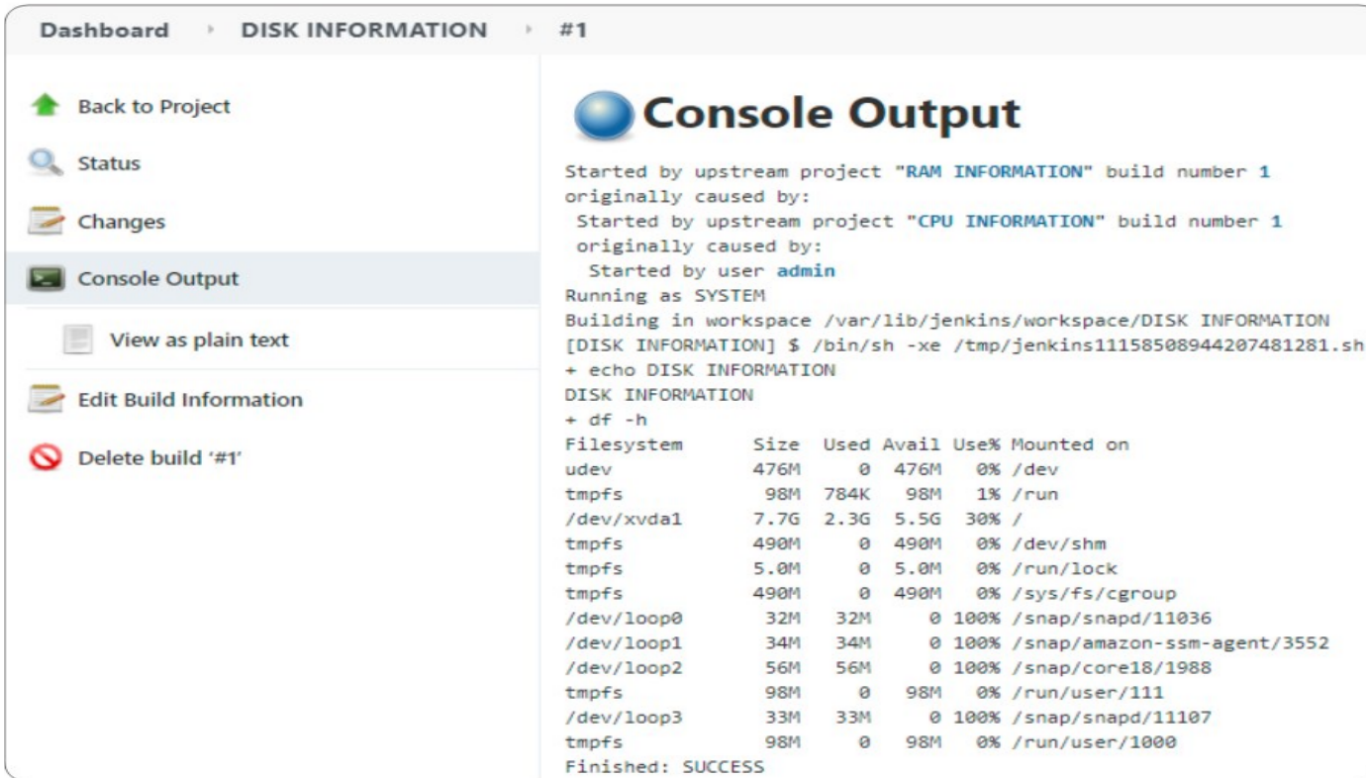




Now Building Job 1 “CPU INFORMATION” will trigger a Build of Job 2 “RAM INFORMATION” and Job 3 “DISK INFORMATION”.

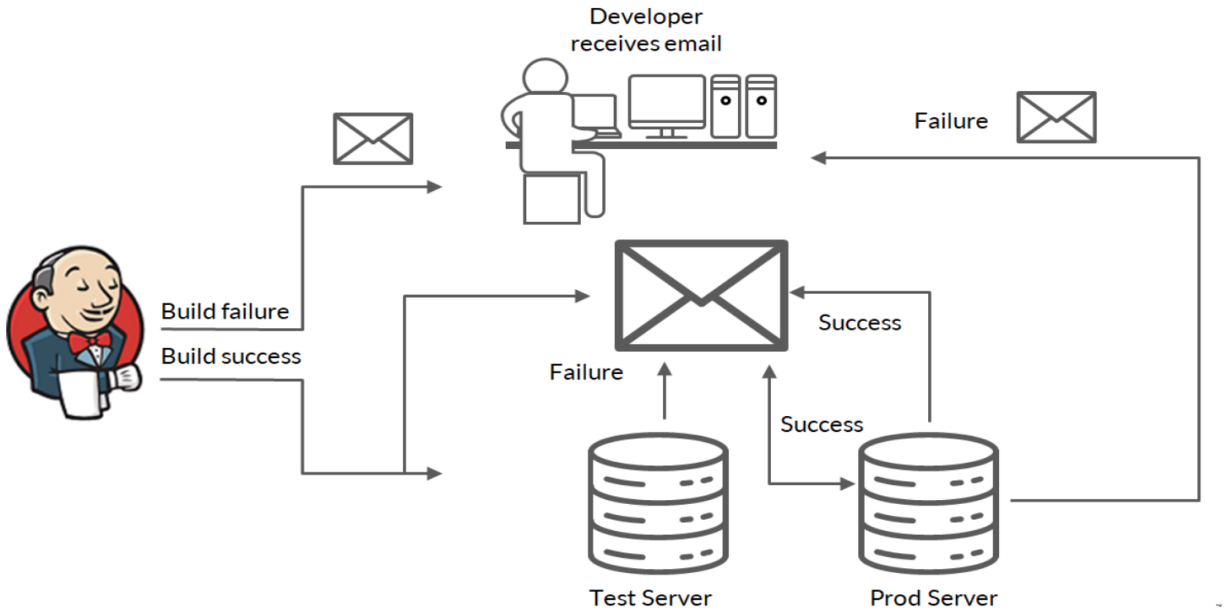
You can see how Building “CPU Information” triggers a build of “RAM Information” and how “DISK Information” is triggered by other upstream jobs.





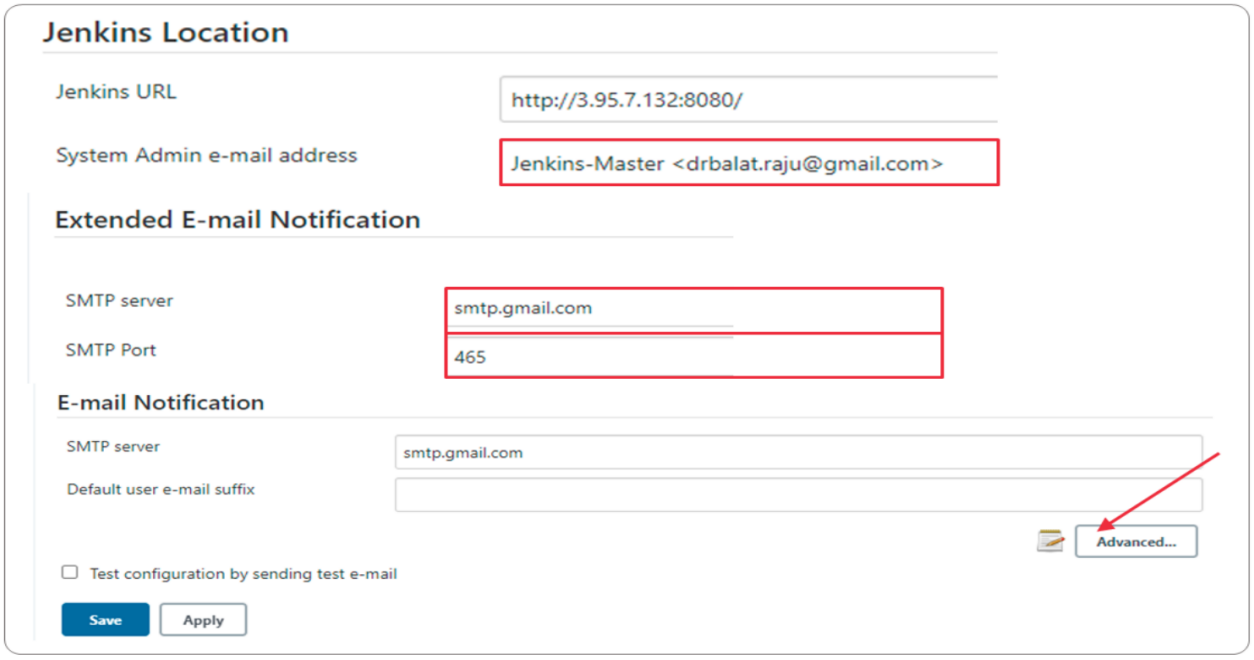
Email Notifications

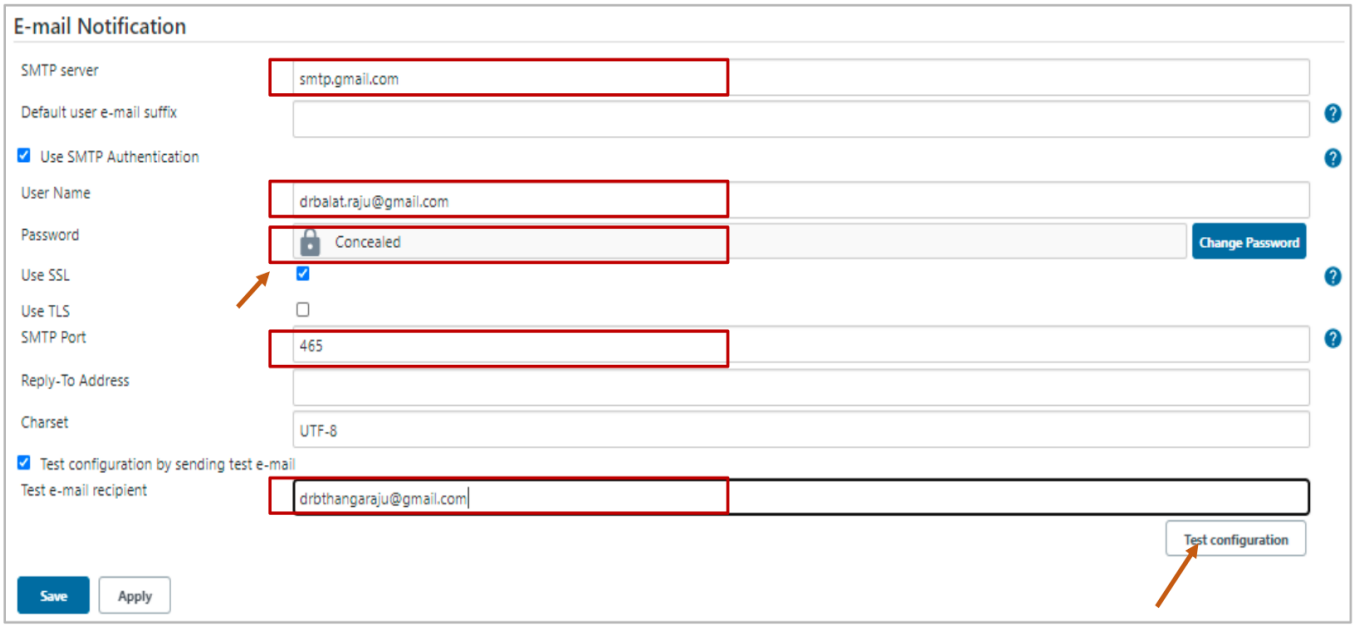
- If the build failed or the build status changed from failed to success, then Jenkins has a provision to send an email notification to the given recipients.



- Install Email Extension Plugin and Mailer Plugin (check whether they are already installed as well).

Email Configuration Setup

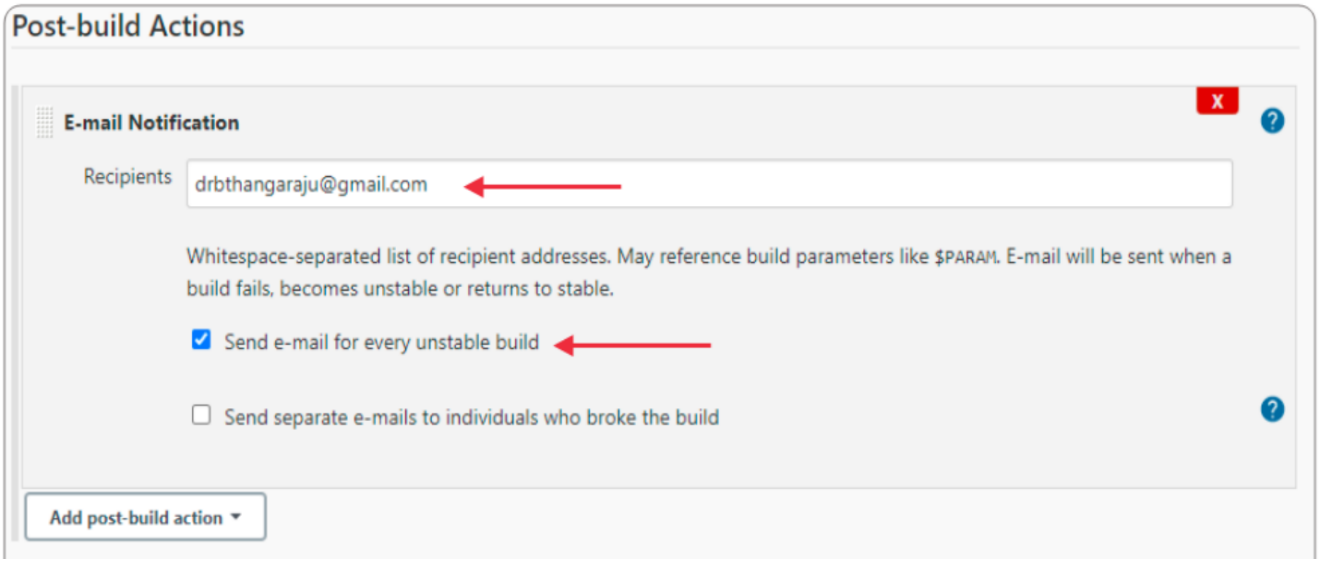




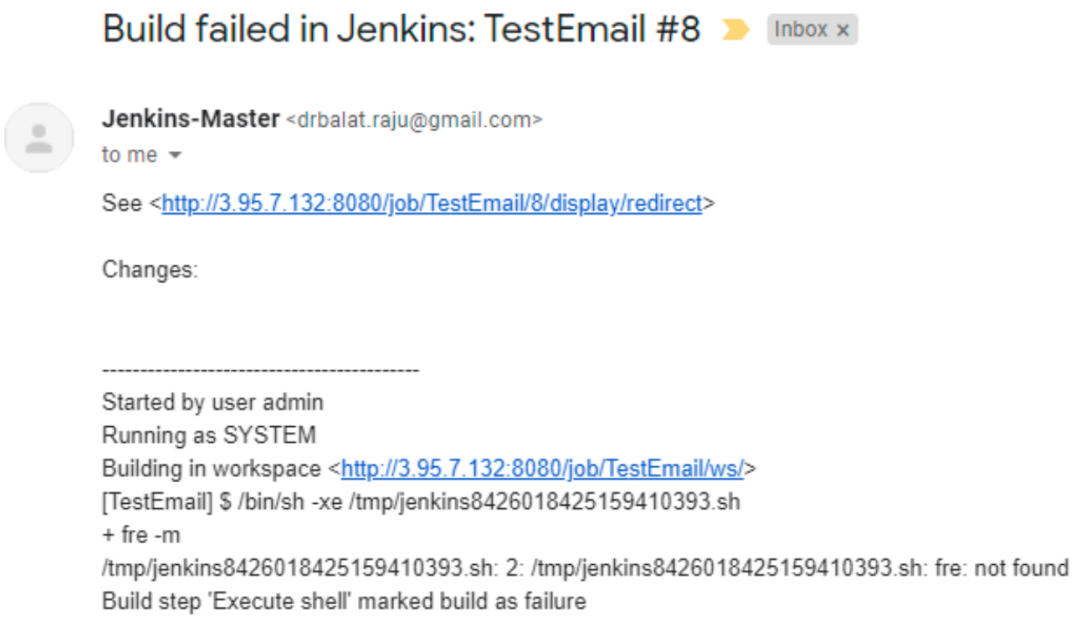
Ensure that all of these settings are exactly the same. Email addresses for both test and master email should be gmail one because for using other domains you'll need to specify the corresponding SMTP server, so better just keep Gmail addresses.

If your Gmail account has 2FA then you'll need to set up an App Password and copy and paste that password in Jenkins under the E-mail Notification section.

Now Configure your Jenkins Project to setup Email Notifications,



The email sent will also specify the Console Output of the build



08/02/22: Lecture

Parameterized Jenkins Jobs

Save this shell file as “c.sh” in workspace

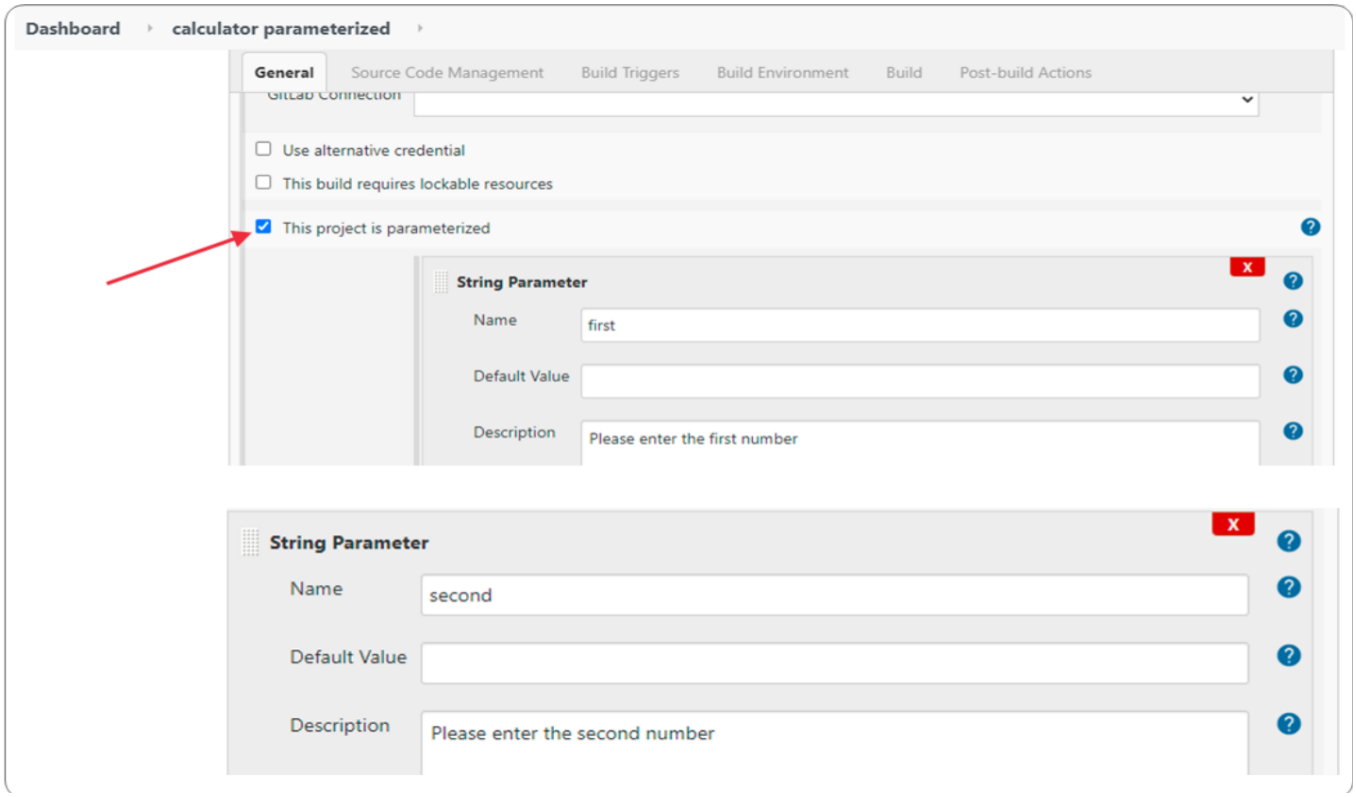
```
# !/bin/bash
# A simple calculator shell program

a=$1
b=$2

echo "Enter Choice :"
echo "1. Addition"
echo "2. Subtraction"
echo "3. Multiplication"
echo "4. Division"
ch=$3

case $ch in
  1)res=`expr $a + $b`
  ;;
  2)res=`expr $a - $b`
  ;;
  3)res=`expr $a \* $b`
  ;;
  4)res=`expr $a / $b`
  ;;
esac
echo "Result : $res"
```

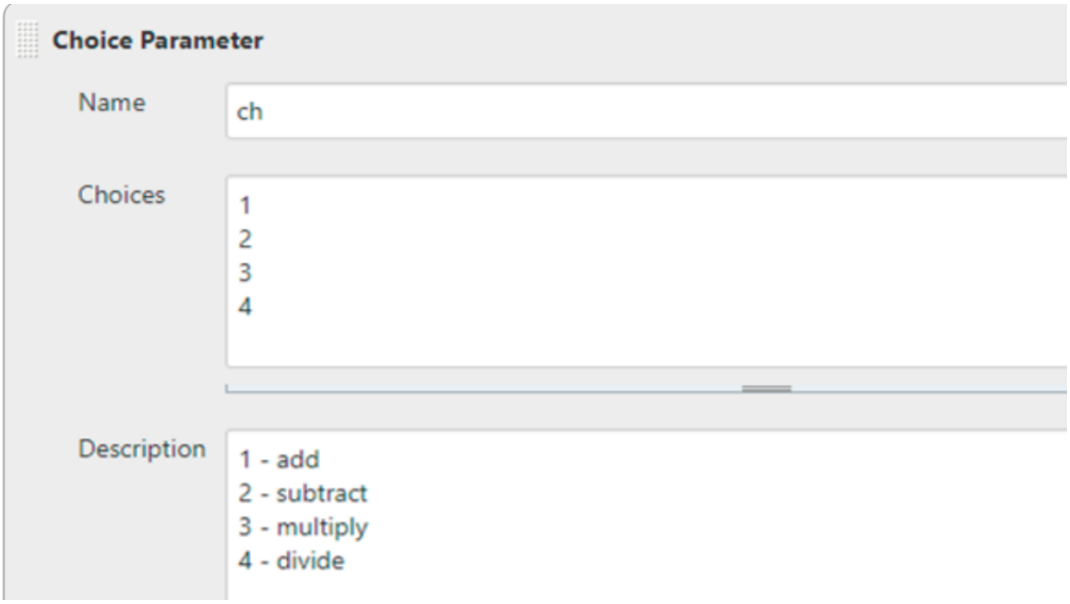
Step1. Create a freestyle project: **calculator parameterized**

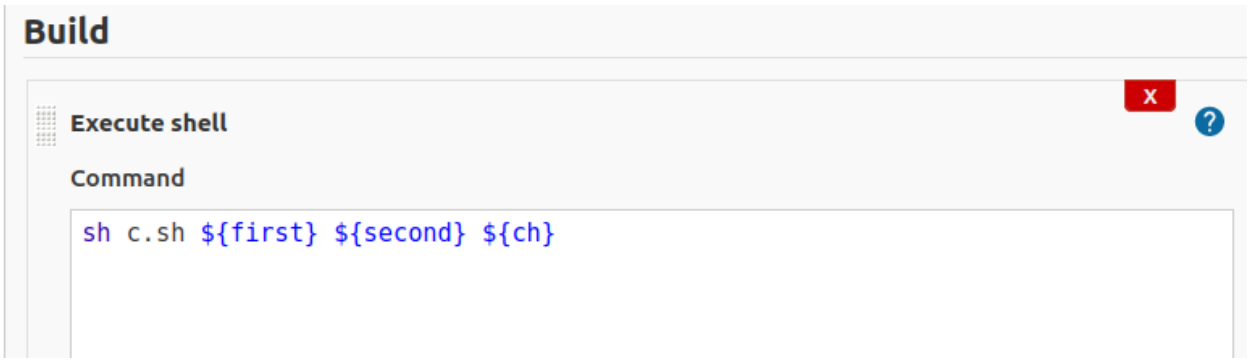
Step 2. Develop a simple calculator shell program in the default workspace

Step 3. During execution, you can choose the a, b and choice through Jenkins ‘This project is parameterized’ option

67







After this run “Build With Parameters” and provide the parameter values in the menu that shows up.

While defining parameters for the job, the parameter types available can be useful to check out as there are quite a number of parameter types like Run, Credentials, File, etc.

User Management in Jenkins

- In a typical project environment, many employees working on a project can access the Jenkins server to run their build or test jobs. This can create security and authorization issues.

- So, you need to give every Jenkins user appropriate permission to enable the Jenkins server's safety and security.

- In Jenkins, different configuration options are available to enable, edit or disable various security features.

- By default, anonymous users have no permissions and logged in users have complete control, i.e. any created user is given admin privileges by default.

- Jenkins admin manages these users based on their roles. Jenkins provides capabilities to add users, edit users and provide different roles to each user. For this, Jenkins provides a role-based authentication plugin.

- The ‘Configure Global Security’ option helps a Jenkins administrator to enable, configure or disable key security features to the Jenkins environment.

Security Realm

- To configure authentication and authorization schemes in Jenkins, you need to use Security Realm and Authorization configurations.

- Security Realm informs the Jenkins environment how and from where to pull user information.

- Authorization configuration informs the Jenkins environment about which users can access which aspects of Jenkins and to what extent.

- The Security Realm/authentication specifies who can access the Jenkins environment, whereas Authorization specifies what they can access.

- Matrix-based security allows the administrator a granular control over assigning users:
    - Provides the most security and flexibility
    - **Recommended for production environments**
    [Not really required for our environments where its just one admin user, i.e. us]

Role-Based Authentication Strategy

- Used to add a new role-based mechanism to manage users’ permission.

- Roles can be assigned to users and user groups.

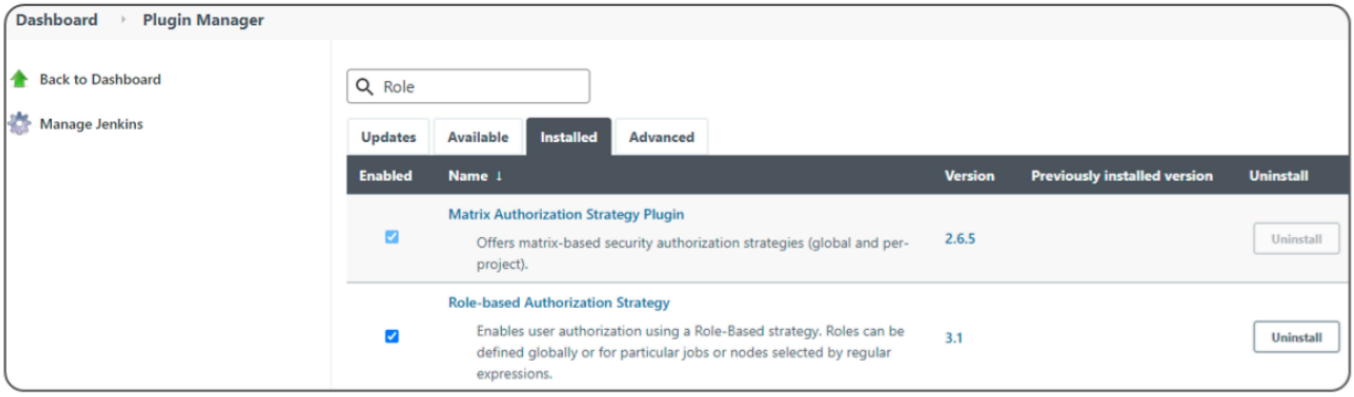
- Global Roles: Admin, Developer, Tester, QA and Anonymous

- Allow to set node-wise permissions: Agent, Job, Run, View and SCM [Will learn more about these later]

- Project/Item roles: Allow additional access control for each project separately in the Project configuration screen and give access control to specific users to access only the specified projects.

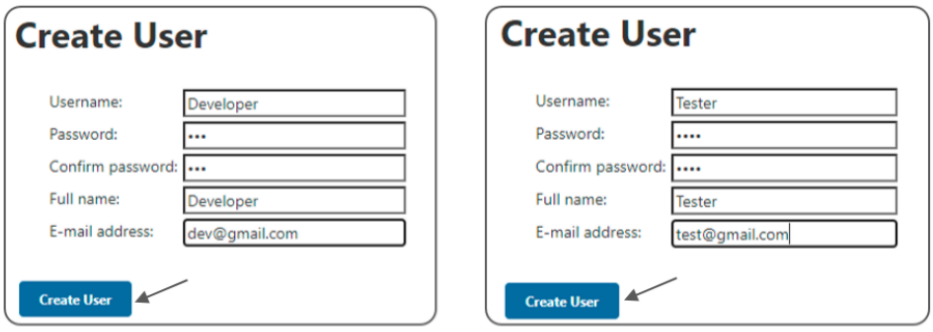
- Agent roles: Set node-related permissions

Required Plugins



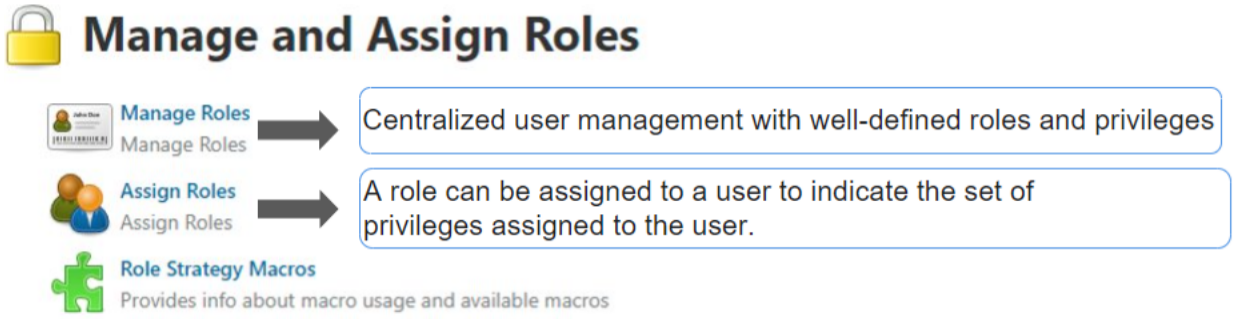
Demo

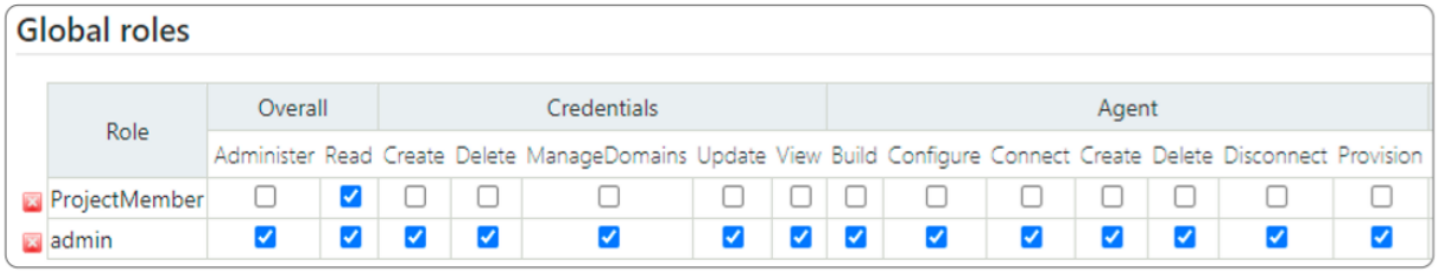
- Create two users: Developer and Tester



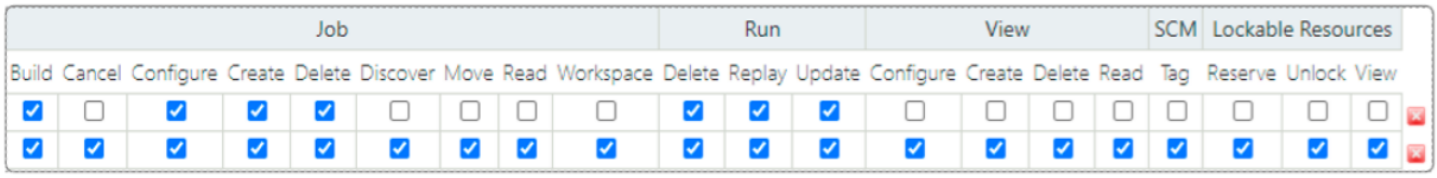
- Configure Global Security: Enable Role-based Authentication strategy

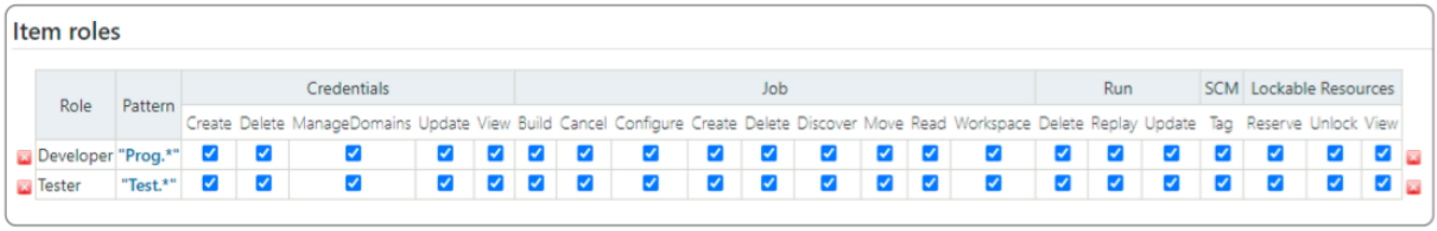
- Manage and Assign roles:
    - ○ Manage roles: Create a Global Roles - add ProjectMember and enable only required access.
    - ○ Item roles: Create two roles - Developer and Tester. Enable all the options.
    - ○ Manage and Assign roles: Add Developer and Tester as ProjectMember
    - ○ Item roles: Developer and Tester users should be assigned to Developer and Tester roles, respectively. Set pattern as Prog.* for Developer and Test.* for Tester



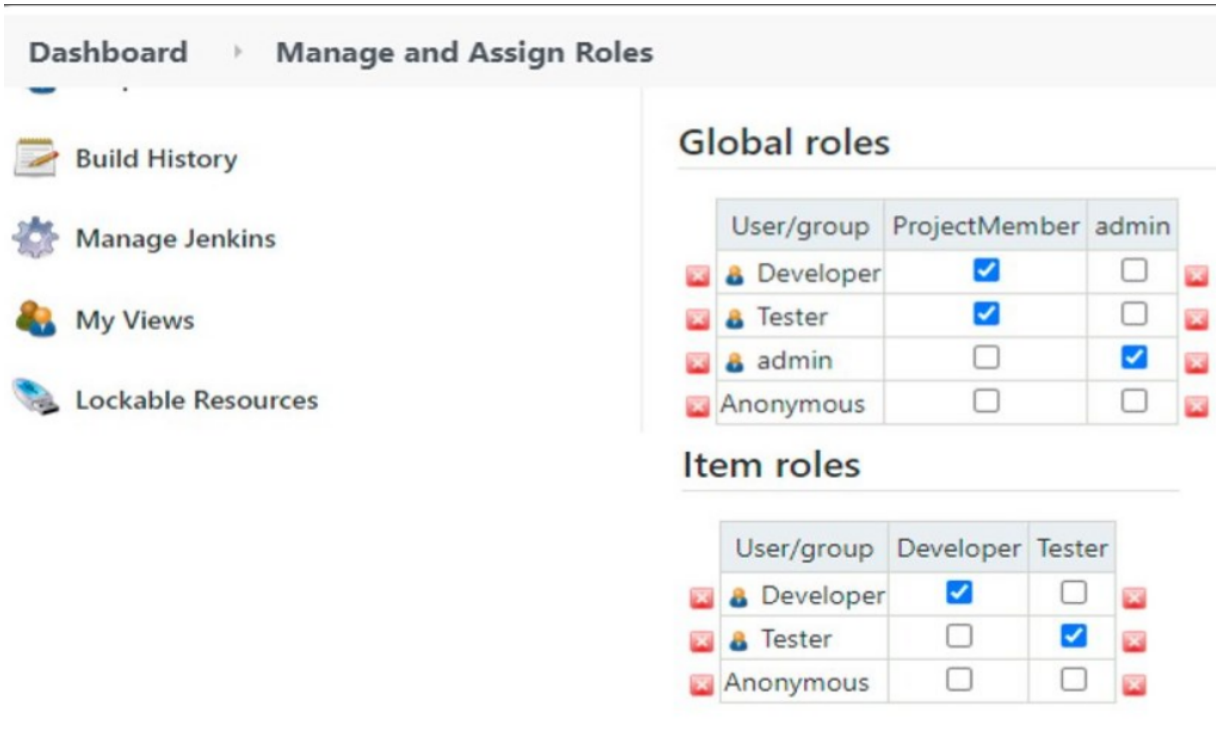


Add ProjectMember role in Global roles and enable required permissions





Add Developer and Tester roles and set pattern as Prog.* for Developer and Test.* for Tester.



- Create two projects: Program1 and TestProject1 (no special configuration required, just that **their names should start with Prog and Test**).
- Login as Developer or Tester and view/build/create/delete the Projects.

i) Developers can view only the jobs starting with “Prog”
ii) Developers can build the available Job.
iii) Developers can create a new development job, but the job name should start with Prog.
iv) Developers cannot see: Tester projects, configure the system or manage plugins.

The same is true for Tester role but their jobs should start with “Test”

[Do all of these while you’re logged in to the Admin account, don’t even be mistake login to one of the other accounts and set the permissions]

Take some time to understand the difference between Global and Item Roles. Global roles deal with Jenkins configuration permissions and everything else in general whereas Item roles only deal with permissions related to jobs.

A user can have multiple Global roles as well as Item roles but it is preferred to design Global roles in such a way that a user will only have to be assigned to one Global role only.

Integrating Jenkins with Git

Install Git plugins,

Poll Source Code Management (SCM) vs Build Periodically

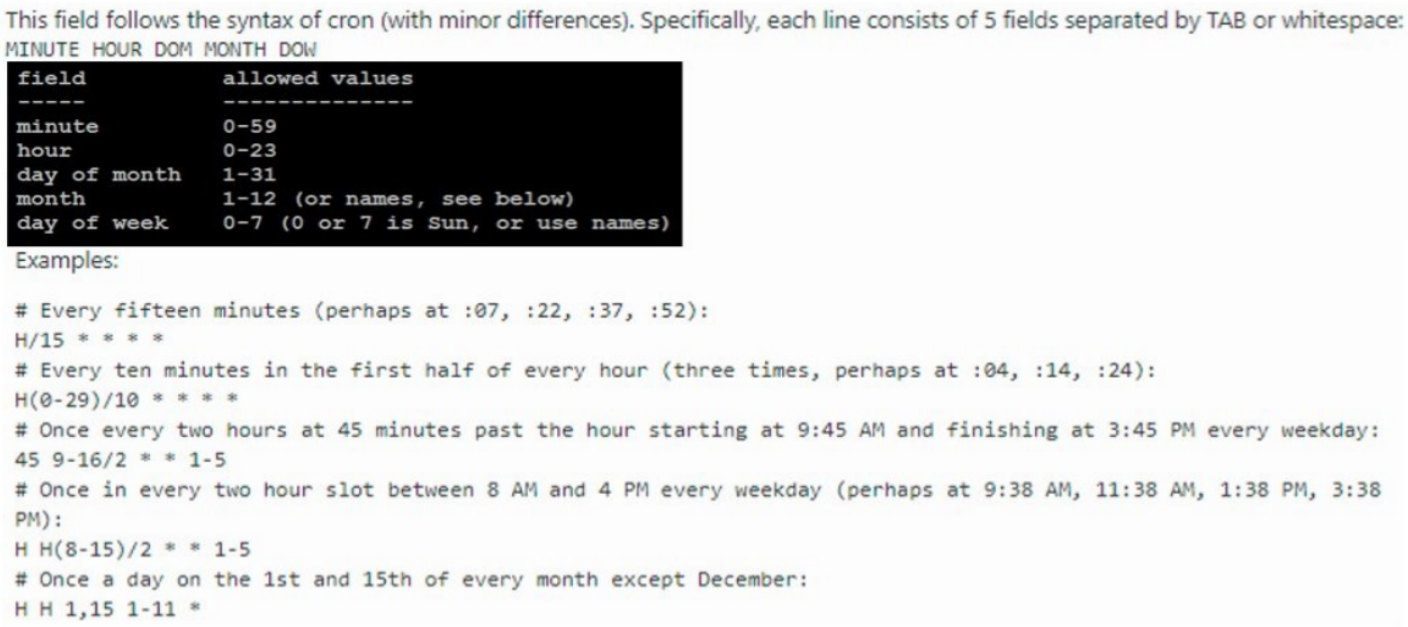
Build Periodically
Jenkins builds periodically even if there are no changes in the project.

Poll SCM
- For every minute (we can specify custom time duration), Jenkins periodically polls the GitHub repo to check whether any new commits were made.
- If there are any changes pushed since the last build, then Jenkins automatically builds the project.

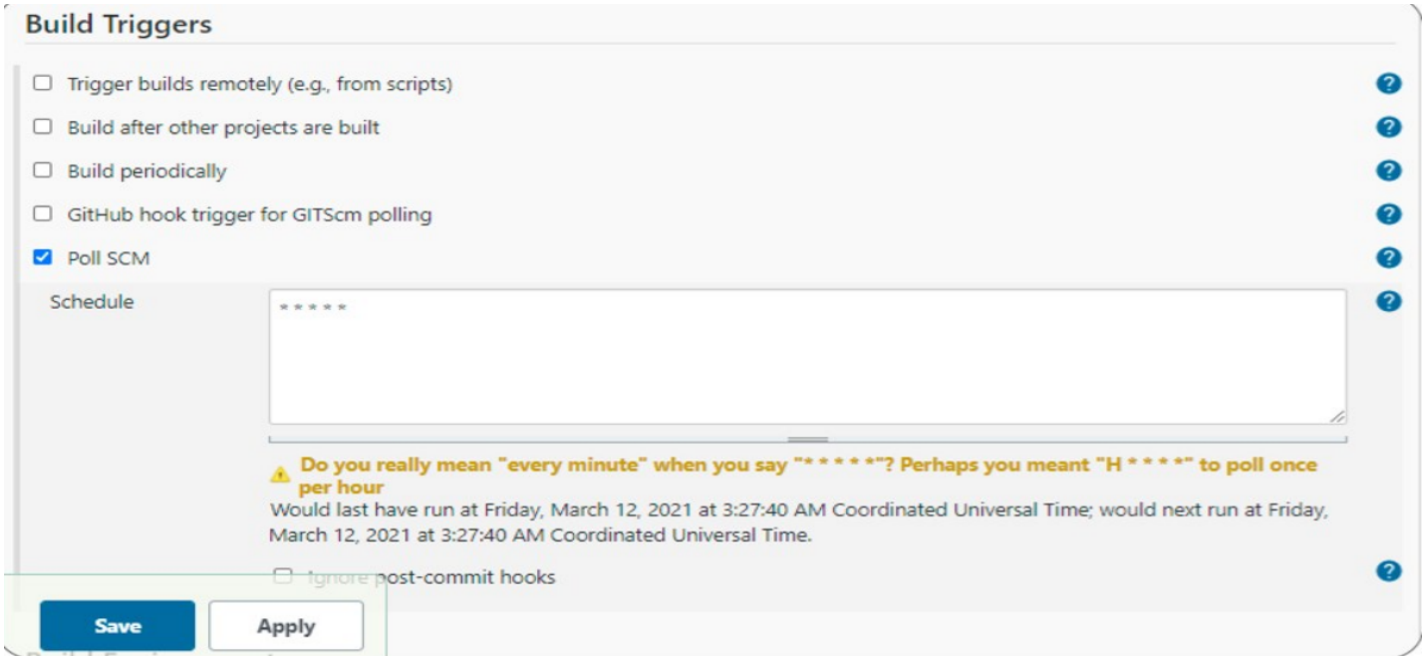
How to?

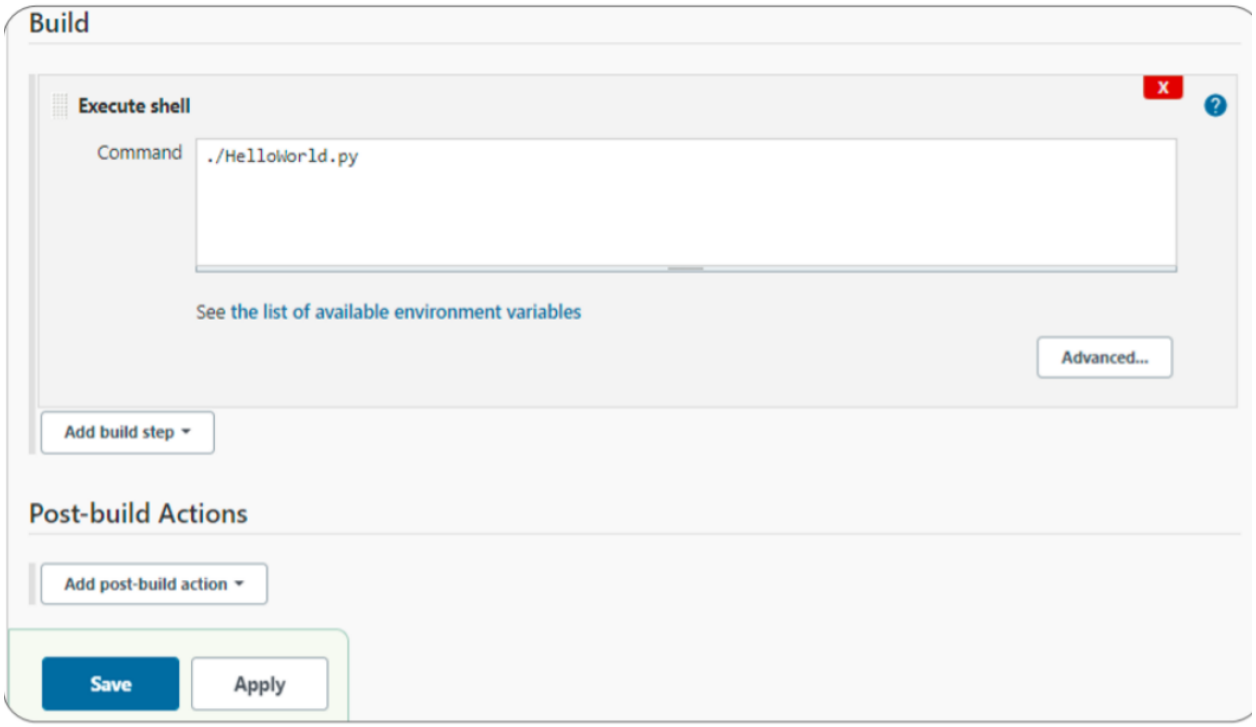
- Copy your repo's GitHub URL
- While creating a new Jenkins job or editing Configurations of an existing one,
Select Git in the Source Code Management section and enter the copied GitHub URL in the given Repository URL option.
[For private repositories you will have to specify the GitHub password (only Private Access Token works now) in the Credentials tab]
Do note that if you are using custom workspace, then you will have to give "Jenkins" user the ownership of that workspace in order for any of this to work,
"sudo chown jenkins -R custom_jenkins_workspace/"
jenkins = Username, -R = To recursively set permissions for all files and folders
custom_jenkins_workspace = Custom workspace folder name

Select Poll SCM and enter the appropriate value which follows the below syntax
Refer to the man page, "man 5 crontab" for more info.

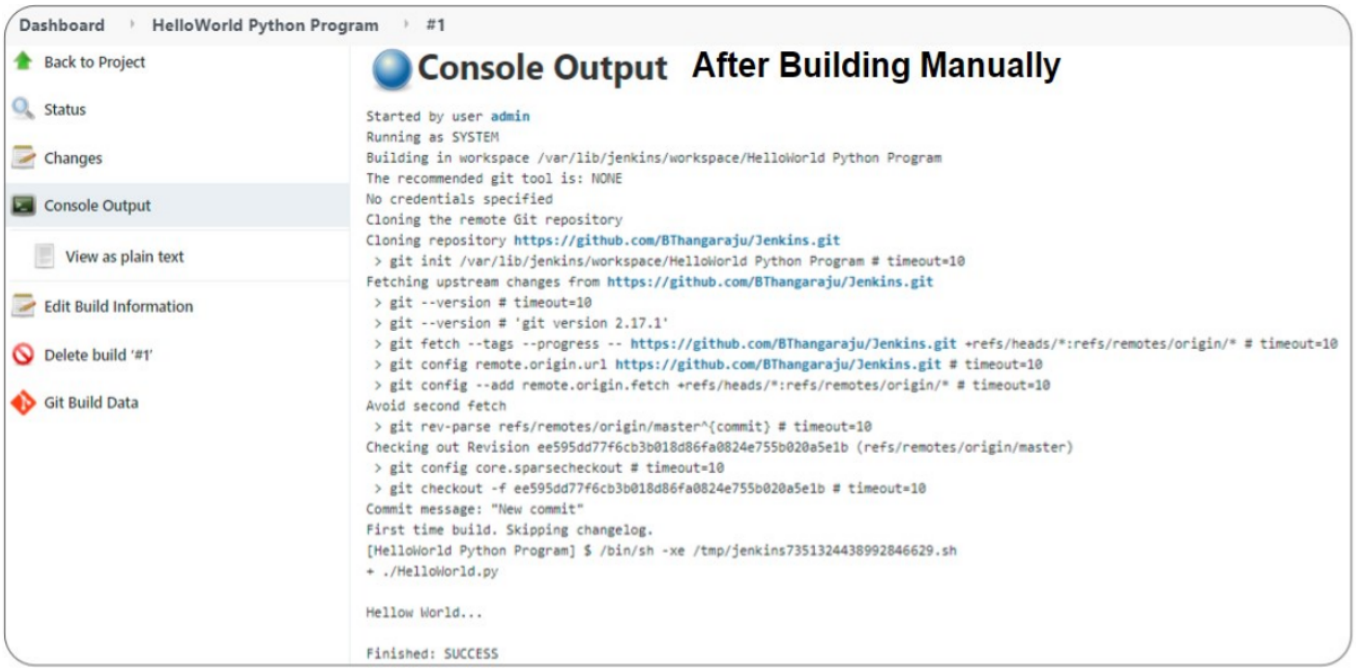


You can also choose "Build Periodically" in the below menu, but we're going ahead with Poll SCM.





Now manually build the project and see the integration in action.



15/02/22

GitHub Hook Trigger for GitSCM Polling

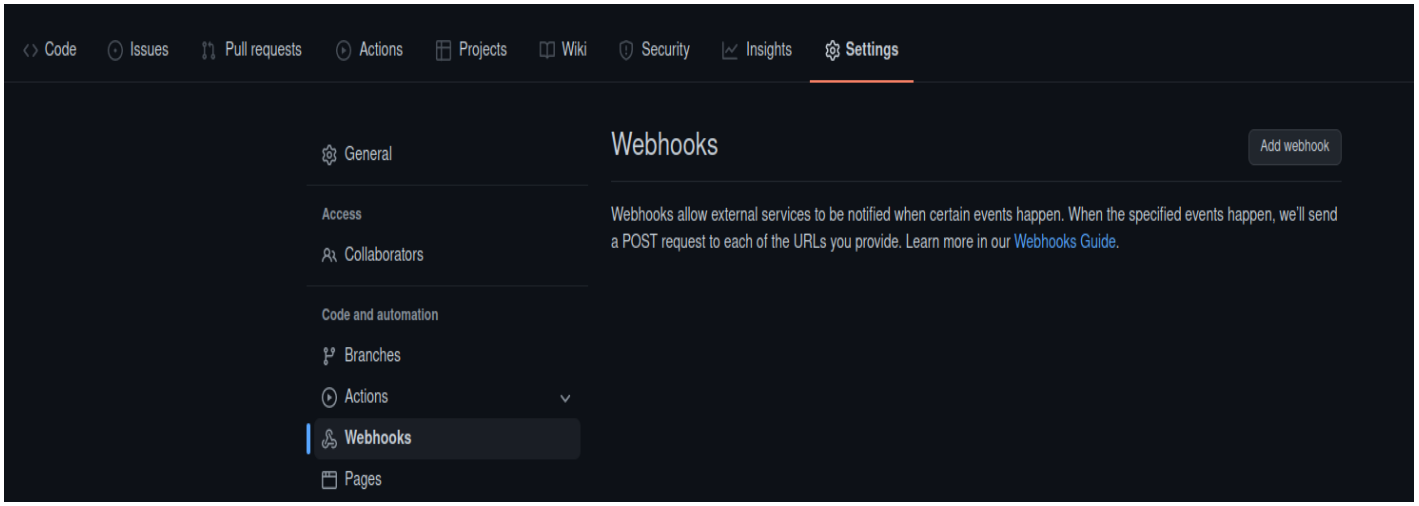
This does the same thing as the Poll SCM option but polling works from the GitHub endpoint. We set up a Webhook at the Github repository that we want to be polled by Jenkins.

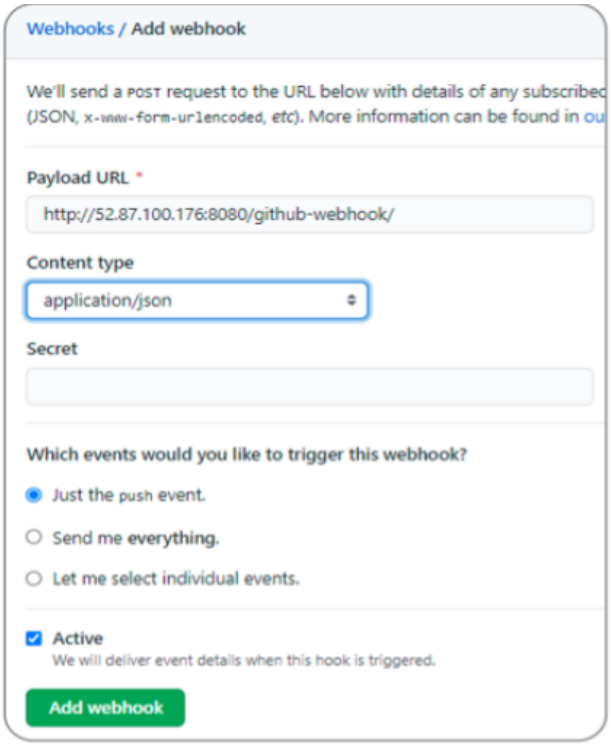
This is the definition of a Github Webhook,
Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide.

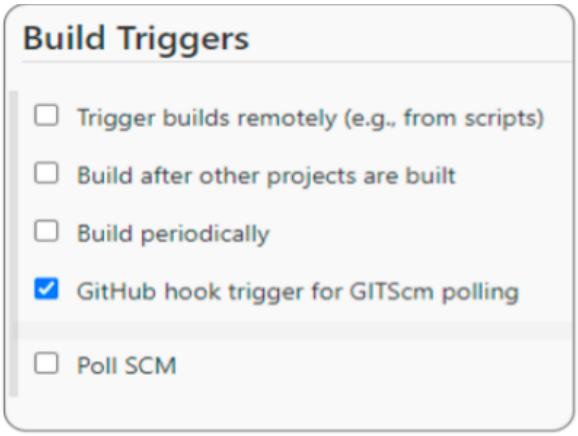
So when a commit happens in a given Github repository, the Jenkins webhook is triggered and Jenkins is alerted of this event and so it polls the Github repo to detect whether any changes have been made in the repository or not and if there are any changes, it will build the repo.

You can see why this is a more efficient mechanism than the normal "Poll SCM" option.
Even the Jenkins Configure menu states this as a warning under the "Poll SCM" option,
"Note that this is going to be an expensive operation for CVS, as every polling requires Jenkins to scan the entire workspace and verify it with the server. Consider setting up a "push" trigger to avoid this overhead, as described in this document"

Configuring GitHub Webhook for Jenkins







The Payload URL should point to, http://<JENKINS_URL>:8080/github-webhook/
“Just the push event” => When a push is made to GitHub repo, trigger the webhook which means Jenkins would poll the repo at that point.

NOTE: This only works if you have a public IP address like AWS URL, but in this case, you have Jenkins on your local system and this will not work as the IP is not public. Therefore you need to setup ngrok server in order to convert your localhost into a public IP address.

ngrok - secure introspectable tunnels to localhost

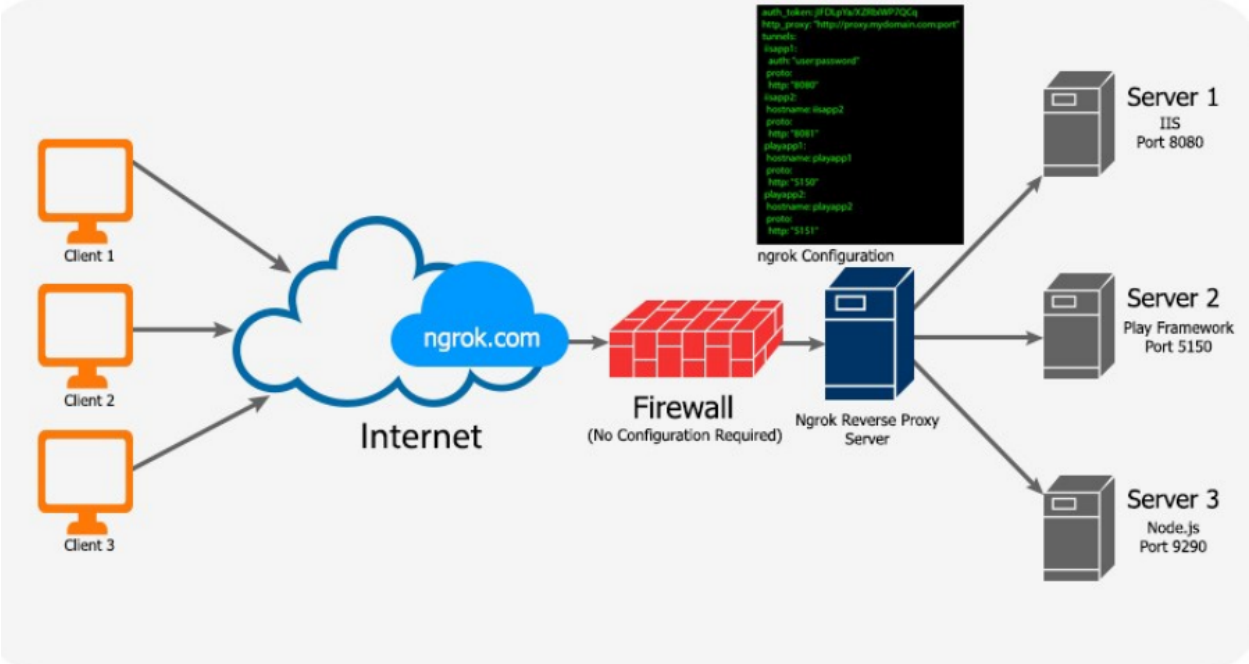
Ngrok exposes local servers behind NATs (Network Address Translation) and firewalls to the public internet over secure tunnels.

ngrok provides a real-time web UI where you can introspect all HTTP traffic running over your tunnels.

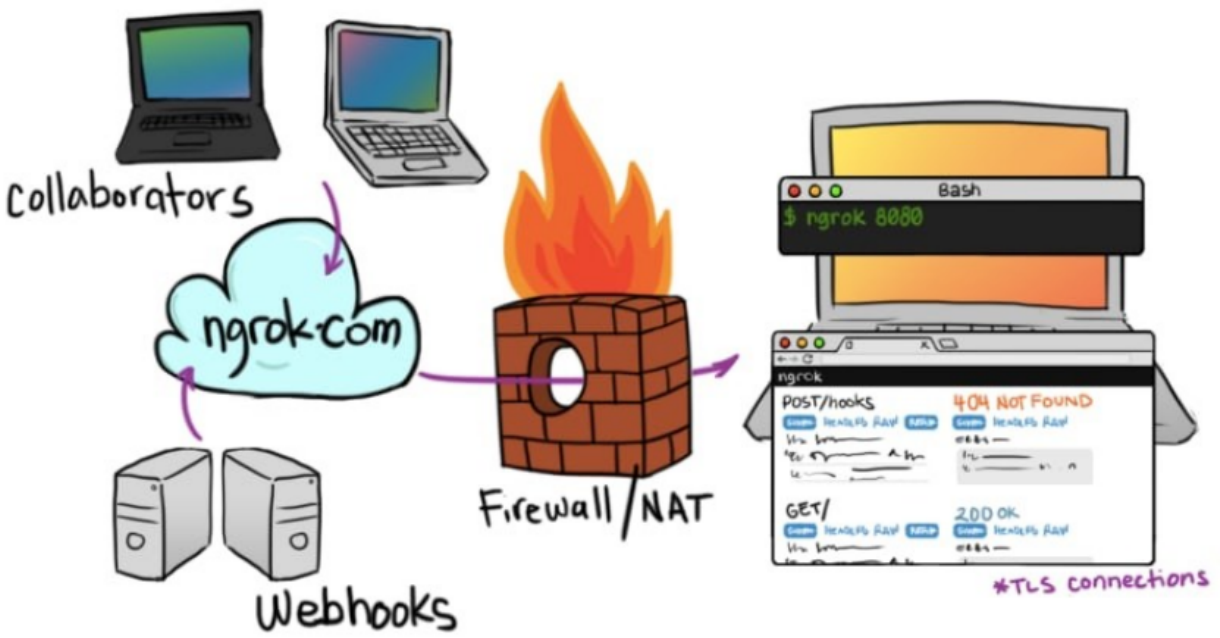
ngrok allows you to expose a web server running on your local machine to the internet. Just tell ngrok what port your web server is listening on.

Example: Expose a web server on port 80 of your local machine to the internet.
$ngrok http 8080
For a different port number, for ex: 5001 then:
$ngrok http https://localhost:5001

ngrok Architecture



That looks more formal and below image ngl explains everything much better :P



ngrok Installation

1. Sign up in https://ngrok.com/
2. Download ngrok from: https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.tgz
3. Then extract ngrok from the terminal: $sudo tar xvzf ~/Downloads/ngrok-stable-linux-amd64.tgz -C /usr/local/bin
4. Copy Authtoken from: https://dashboard.ngrok.com/get-started/your-authtoken
5. Add Authtoken: $ngrok authtoken <token>
6. Execute $ngrok http 8080; copy the public ip address for your local host.

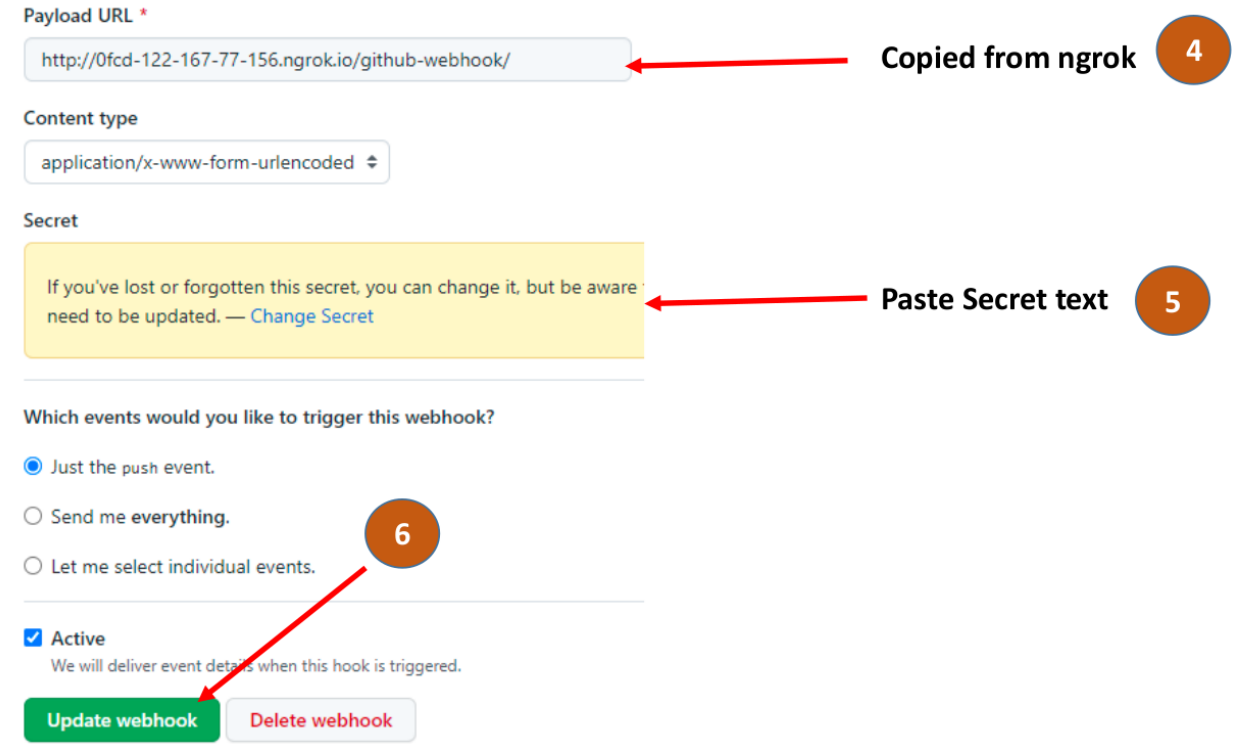
```
ngrok by @inconshreveable                                       (Ctrl+C to quit)

Session Status                online
Account                       Thangaraju (Plan: Free)
Version                       2.3.40
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://5336-122-167-77-156.ngrok.io -> http://localhost:8080
Forwarding                    https://5336-122-167-77-156.ngrok.io -> http://localhost:8080

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

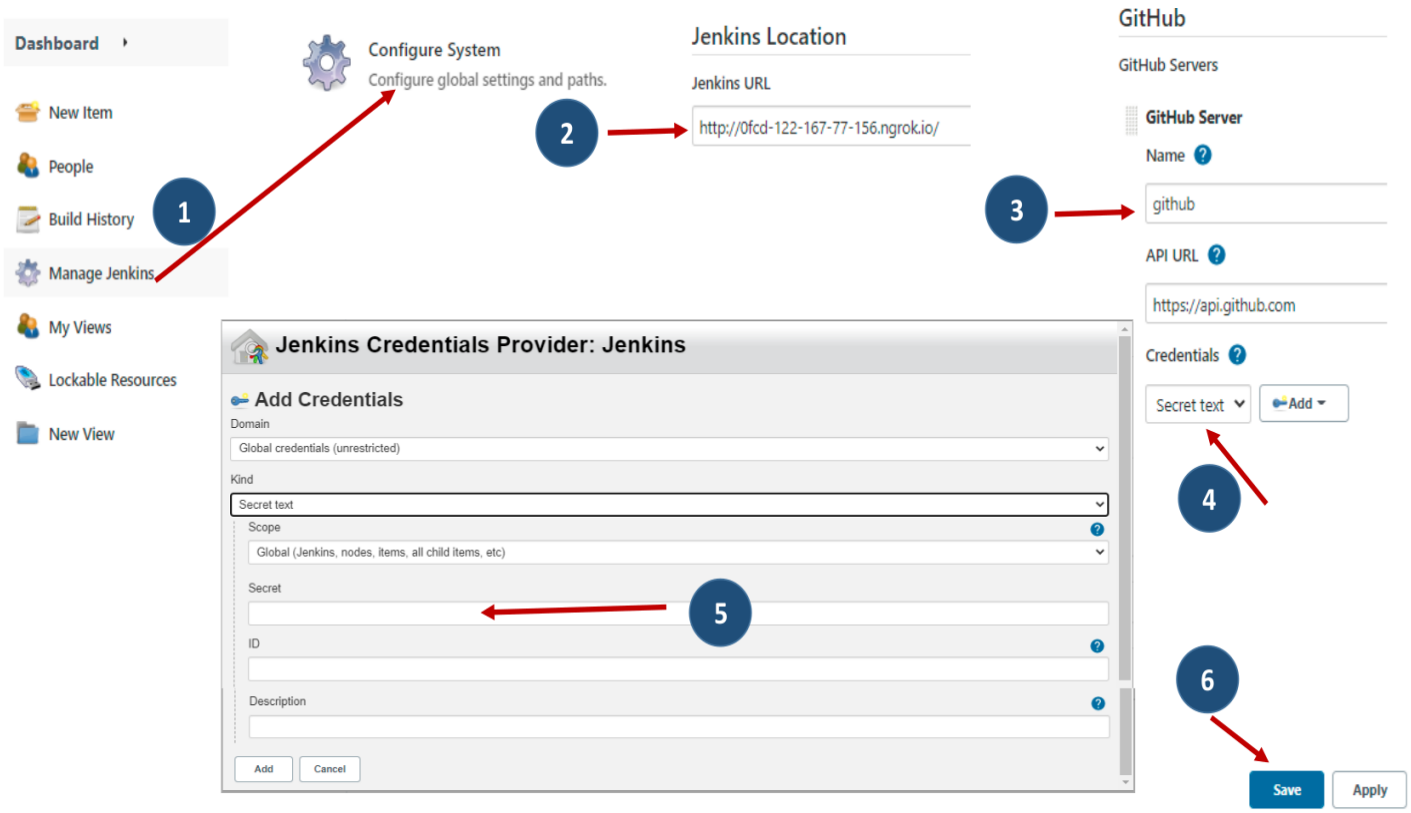
Now we can finally setup the Webhook for Jenkins as described before.

To ensure that the communication between localhost and the ngrok server is secure, we use the Authtoken and to ensure that the communication between Jenkins (at ngrok server) and Github repository (via Webhook) is secure and authorised, we setup a Secret Text at the Github repository Webhook and also provide it at Jenkins job configuration.

This Secret Text will be a GitHub Personal Access Token (PAT) that has permissions to read from and write to repositories. So first create a PAT.

Configuring Webhook





Note that the ngrok Public IP changes on each run, so you will have to update Github and Jenkins configuration each time you stop and start the ngrok server :(

Remember that none of this configuration with ngrok is required if Jenkins is on a public IP like AWS.

Jenkins Pipeline

Pipeline is defined as a suite of plugins that helps you orchestrate simple or complex automation.

Jenkins Pipeline provides tools for modelling delivery pipelines “as Code” (Pipeline as Code)

CI/CD Pipeline integrates SDLC stages, steps to execute tasks in each stage, trigger jobs for a given order and show pipeline status with logs. Automation from Continuous build to Continuous Monitoring (build, test, staging, deploy and monitor)

Sequence of stages to perform can include tasks such as pulling code from the Git repository, static code analysis, building project, executing unit tests, automated tests, performance tests and deploying applications.

Types of Pipeline

Declarative

- New method
- Easy to use for beginners
- Groovy language skill is desirable

Scripted

- Traditional
- Based on Groovy Domain Specific Language (DSL)
- Multiple features - very expressive and flexible tool
- Difficult to use for beginners
- Should have working experience on Groovy language

Nowadays Declarative is the more preferred option.

```
pipeline {
        agent any
        stages {
stage('build code') {
        steps {
            /*write steps */
          }
        }
        stage ('test') {
        steps {
            /*write steps */
          }
        }
    }
}
```

Declarative Pipeline Script

```
node {
        stage ('build code' {
            /*write steps */
          }
        stage ('test') {
            /*write steps */
          }
}
```

Scripted Pipeline Script

Pipeline – contains all the script content
Agent and Node – defines the agent where the pipeline will run
Stages – contains all the stages
Steps – way to execute various jobs

Agent = Which node or virtual machine will the stage be executed on (obviously this is in a distributed system) [instead of Jenkins executing everything on the Jenkins server system itself]

So "agent any" means that whichever agent is free will be assigned that stage. In the localhost case where there is no distributed system, there is only one agent free which is the Jenkins server system, i.e. localhost itself.

Pipeline Directives

A Pipeline Directive allows you to define a list of parameters to be used in the script. Parameters should be provided once the pipeline is triggered.

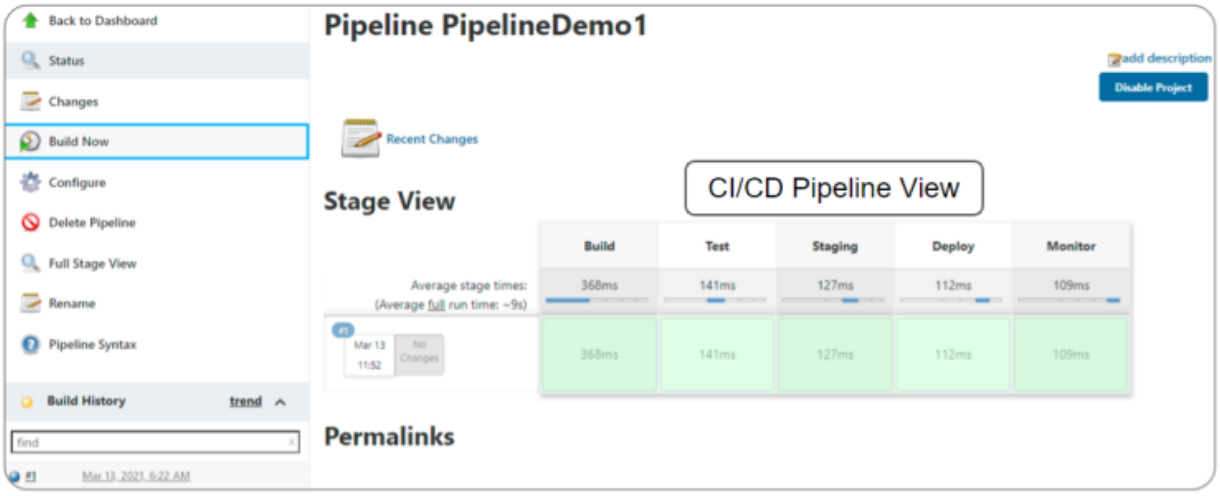
Environment – defined as environment variables
Input – prompt for input
Options – configure pipeline-specific options like retry, timeout, etc.
Parallel – list of nested stages to be run in parallel
Parameters – list of parameters to provide when triggering the Pipeline (e.g., string, password)
Post – run at the end of a Pipeline's execution (e.g., add some notification to the other end of Pipeline tasks)
Tools – defining tools or packages to auto-install and put on the PATH (e.g., Maven. JDK, Gradle)
Triggers – determines how pipelines should be triggered (e.g., cron, Poll SCM)
When – determine executing stage depending on the given condition

Jenkins Pipeline Syntax: https://www.jenkins.io/doc/book/pipeline/syntax/
VERY IMPORTANT FOR LEARNING MORE OF THE SYNTAX AND THIS WILL BE REQUIRED IN THE PROJECTS AS WELL.

Example Pipeline,

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                echo 'This is job building stage'
            }
        }
    stage('Test') {
            steps {
                echo 'This is Testing stage'
            }
        }
    stage('Staging') {
            steps {
                echo 'This is Staging environment'
            }
        }
    stage('Deploy') {
            steps {
                echo 'This is Deploying stage'
            }
        }
    stage('Monitor') {
            steps {
                echo 'This is Monitoring stage'
            }
        }
    }
}
```

You can see this in action by creating a new Jenkins project which is a Pipeline project and **not a Freestyle Project** and type this Pipeline script there.
Once the project is created, simply clicking on Build Now should give you the following stage based build status which looks cool,



Pipeline Script Demo

Write three Python scripts with the following code and upload it to a GitHub repo,

ubuntu@ip-172-31-57-56: ~

```
#! /usr/bin/python3
# This Phython program will print Hello World...
print("Hello World ...\n")
```

HelloWorld.py

ubuntu@ip-172-31-81-117: ~/git-demo

```
#!/usr/bin/python3
# Source code for summation of two numbers

def summation(data):
    return sum(data)
```

Prog1.py

ubuntu@ip-172-31-81-117: ~

```
!/usr/bin/python3
# Test case for adding two numbers
import unittest

from Prog1 import summation

class TestSum(unittest.TestCase):
    def test_list_int(self):
        """
        Test case to add two numbers
        """
        data = [23, 32]
        result = summation(data)
        self.assertEqual(result, 55)

if __name__ == '__main__':
    unittest.main()
```

Test.py

Create a Pipeline project with the following Pipeline script,

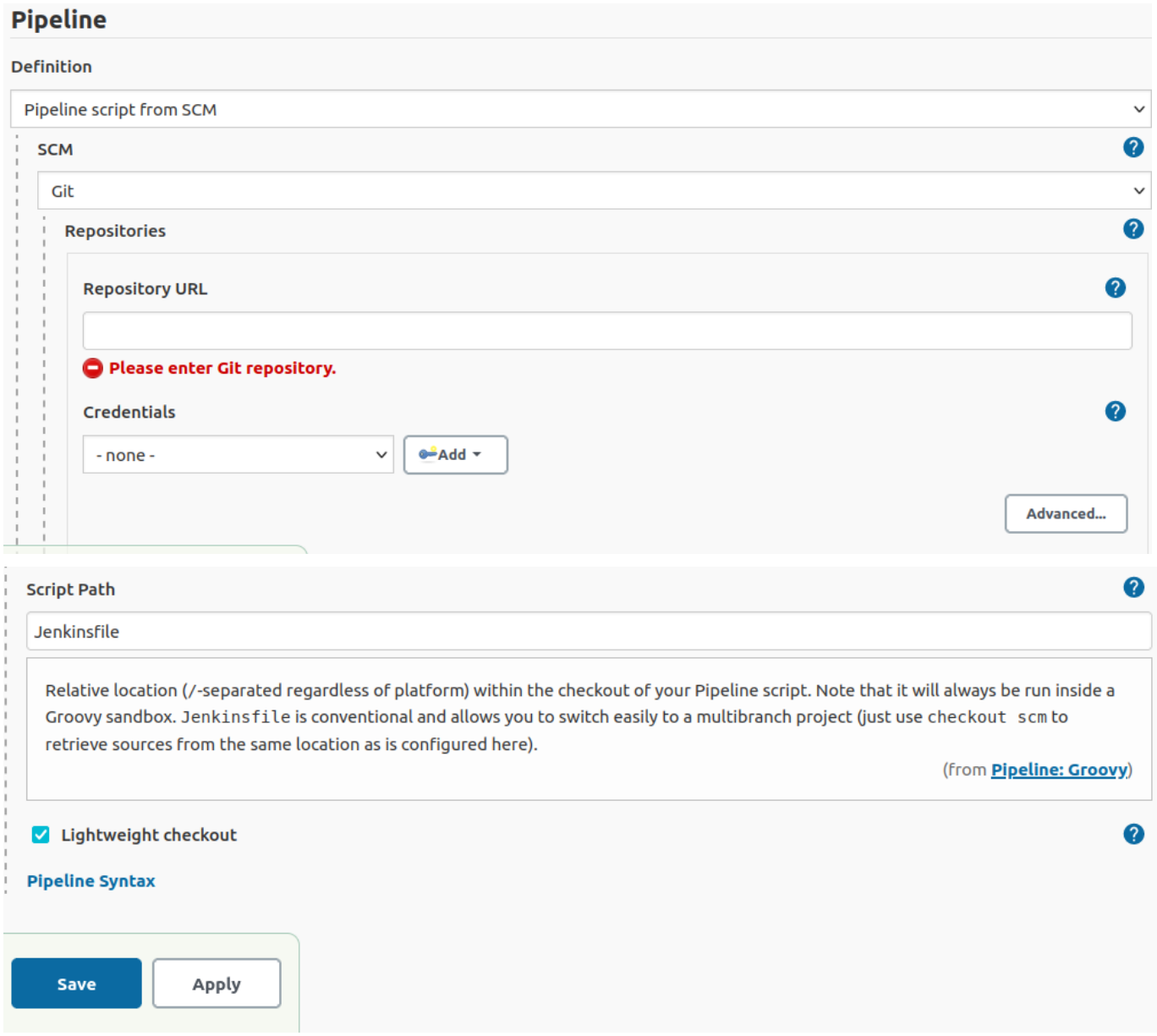
```
pipeline {
agent any
  stages {
    stage('Clone Git') {
      steps {
        git 'https://github.com/BThangaraju/Jenkins.git'
      }
    }
    stage('Build Code') {
      steps {
        sh "chmod u+x Prog1.py"
        sh "python3 Prog1.py"
      }
    }
    stage('Test Code') {
      steps {
        sh "chmod u+x Test.py"
        sh "./Test.py"
      }
    }
  }
}
```

You can then build the project and see everything in action with proper console outputs.

You can also put invalid test cases in the unittest script and you'll see that the build fails as expected with proper output.

Pipeline Script from SCM

Also, instead of configuring the Pipeline script in Jenkins, you can put even the Pipeline script in the GitHub repo. For this you have to make the following changes,



So you can place the same Pipeline Script in the specified GitHub repo with the name "Jenkinsfile"

So…

Some changes were required from the given pipeline script in order to get builds working with private GitHub repos. Following is the Pipeline script "Jenkinsfile" that finally worked for me,

```
pipeline {
agent any
  stages {
      stage('Clone Git') {
         steps {
            git credentialsId: 'github-pat',
            url: 'https://github.com/james-jasvin/Jenkins-Demo.git'
         }
      }
      stage('Build Code') {
         steps {
            sh "chmod u+x Add.py"
            sh "python3 Add.py"
         }
      }
      stage('Test Code') {
         steps {
            sh "chmod u+x Test.py"
            sh "python3 Test.py"
         }
      }
  }
}
```

“github-pat” is the ID of a Jenkins credential which contains username and password as PAT for GitHub account.

I gave it the name “github-pat” while creating the credential because it makes it easier to type in scripts like the one above but you can leave it empty and you’ll get a system-assigned id instead, but that’ll be a hassle to type in scripts as it's so long.

MODULE 3: Docker

Virtualization & Containers

Introduction to Virtualization

Software demands more and more from operating systems to applications.
– More : data, processing power, memory, network bandwidth and storage space.

Virtualization is a technology.
– Create multiple simulated environments from a single, physical hardware system.

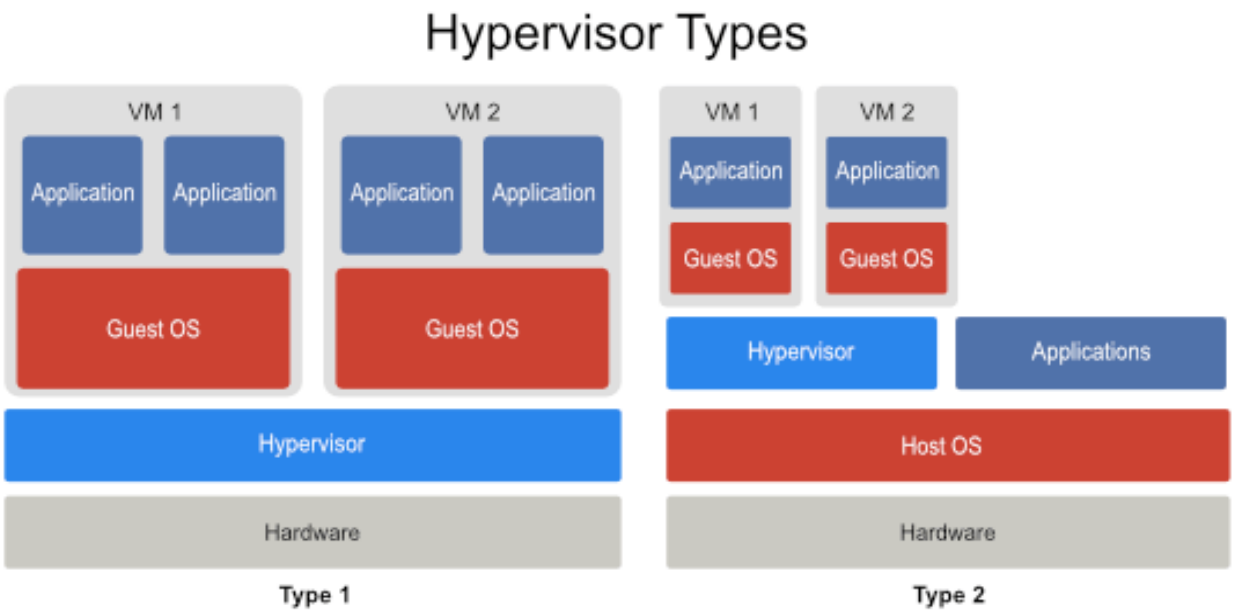
Software called a hypervisor connects directly to the hardware.
– Splits one system into separate, distinct, and secure environments known as virtual machines (VMs).

[Hypervisor = Supervisor of the Supervisor, Supervisor of the OS is the kernel and Hypervisor is Supervisor of the Kernel]

The hypervisor/VMM (Virtual Machine Manager) is the control system at the core of virtualization.
– Acts as the control and translation system between the VMs and the hardware.

Hypervisors need OS - level components to run VMs
– Memory manager, Process scheduler, I/O stack, DD, security manager and Network stack.

The physical hardware, equipped with a hypervisor is called the Host, while the many VMs that use its resources are Guests.



Type 1 Hypervisor = Bare-metal Hypervisor => There is no host OS and instead Hypervisors like KVM manages the system on top of which Guest OS and VMs can be hosted.

Type 2 Hypervisor = Hosted Hypervisor => There is a host OS on which Hypervisors like VirtualBox is installed and on top of which various VMs can be hosted.

Benefits of Virtualization

- Reduce power consumption and air conditioning needs (move to be more eco-friendly).
- Trim the building space and land requirements.
- Reduce downtime and provide high availability for critical applications (Blue-Green model).
- Simplify IT operations.
- Allow IT organisations to respond faster to changing business demands.

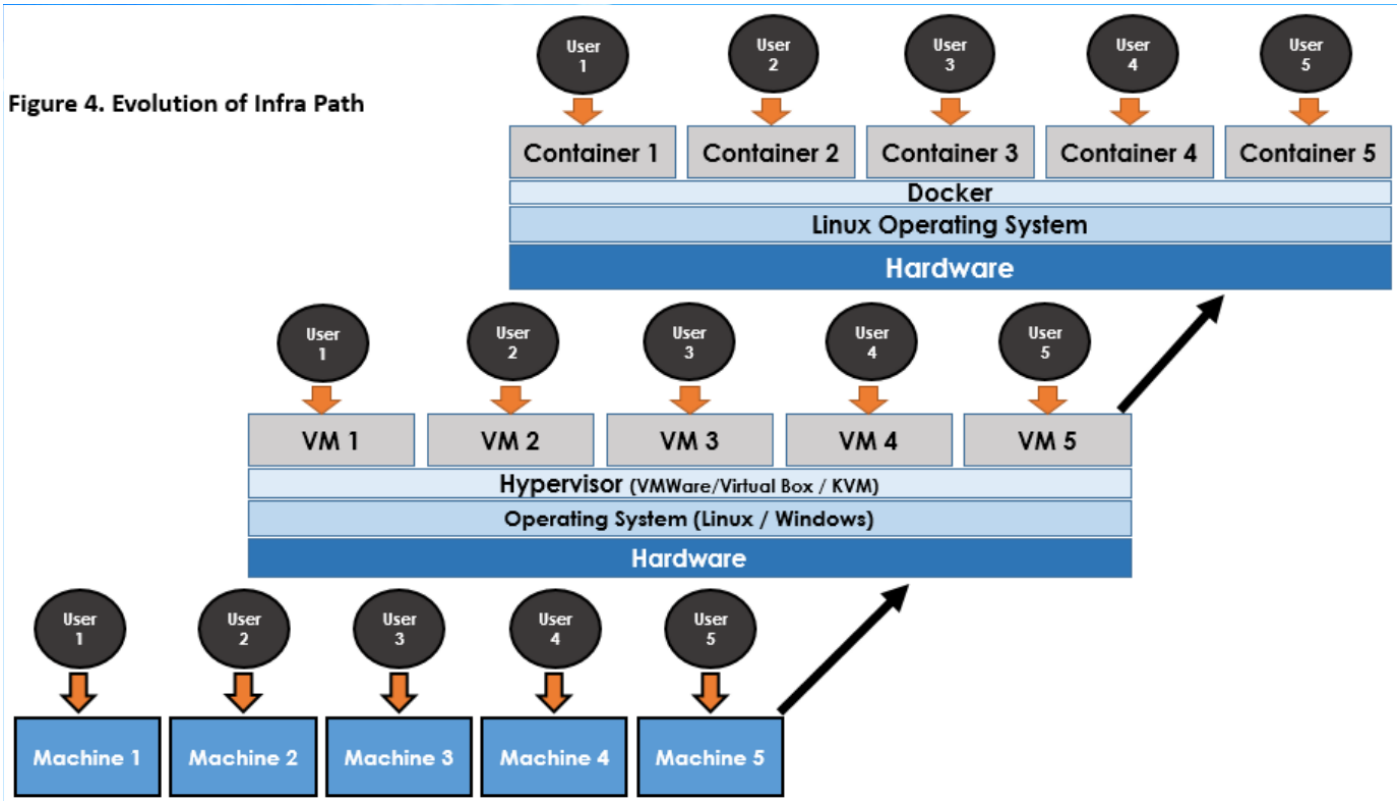
Cloud Computing

Cloud computing is a set of principles and approaches to deliver compute, network, and storage infrastructure resources, services, platforms, and applications to users on-demand across any network.

These infrastructure resources, services and applications are sourced from clouds, which are pools of virtual resources orchestrated by management and automation software.

Resources can be accessed by users on-demand through self-service portals supported by automatic scaling and dynamic resource allocation.

Containerization



Containerization is a modern virtualization method defined as a form of operating system virtualization.

Accesses a single OS kernel to power multiple distributed applications that are each developed and run in their own container.

Applications with all dependencies and libraries needed are packed as an image that can be run anywhere - desktop, traditional IT, or the cloud.

One container runs as a single process on a machine, so theoretically, you can have as many containers running on a single machine as PID_MAX (for me it was, 4194304, check using, cat /proc/sys/kernel/pid_max)

So Containerization is like Type 1 Hypervisor in some ways but without the overhead of setting up VMs and each container is a complete encapsulation of what is required to run an application => Docker Image.

Docker

Docker (Dictionary meaning) – longshoreman
somebody whose job is to load and unload cargo vessels in a port.

Docker (OSS meaning)
Automates the deployment of applications inside software containers.

Docker implements a high level API to provide lightweight containers that run processes in isolation. It is written in Go and uses Linux kernel features like namespaces and cgroups.

Isolation features
- namespaces and cgroup (control group) to allow independent containers to run avoiding the overhead of starting virtual machines.
- By using containers, resources can be isolated, services restricted and processes provisioned to have a private view of the OS with their own PID space, file system structure and network interface (and even system resources like RAM, processors, etc.).

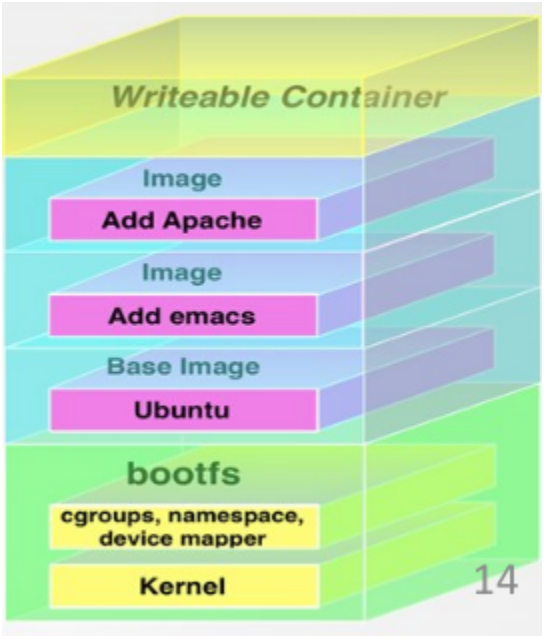
22/02/22

Docker Internals

- Reminder: Docker containers (Containerization in general) do not package up the OS. They package up the applications with everything that the application needs to run. The engine is installed on top of the OS running on a host. Containers share the OS kernel allowing a single host to run multiple containers.
- Docker makes use of Linux kernel namespaces to provide the isolated workspace called the container.
- It also uses the Copy-on-Write (COW) technique for quick provisioning.
- When you run a container, Docker creates a set of namespaces for that container. These namespaces provide a layer of isolation.
- Each aspect of a container runs in a separate namespace and its access is limited to that namespace.
- Docker Engine uses the following namespaces on Linux:
    - **– PID namespace for process isolation.**
    - **– NET namespace for managing network interfaces.**
    - **– IPC namespace for managing access to IPC resources.**
    - **– MNT namespace for managing filesystem mount points.**
    - **14**
    - **– UTS namespace for isolating kernel and version identifiers.**

Each container has its own set of IPC resources like semaphores, shared memory, message queues, etc.

Components of a Container,
Base Image determines the Linux flavour and what all commands are accessible.
Then you can add images of whatever applications you want on top of it.

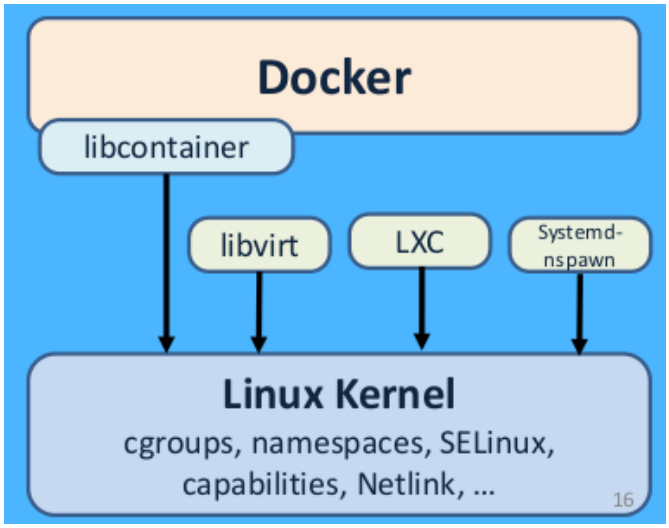


Control groups allow Docker Engine to share available hardware resources to containers and optionally enforce limits and constraints.

Docker Engine uses the following cgroups:
– **Memory cgroup** for managing accounting, limits and notifications.
– **HugeTLB cgroup** for accounting usage of huge pages by process group.
– **CPU cgroup** for managing user / system CPU time and usage.
– CPUSet cgroup for binding a group to a specific CPU. Useful for real time applications and NUMA (Non-uniform Memory Access) systems with localised memory per CPU.
– **BlkIO cgroup** for measuring & limiting the amount of blkIO by group.
– **net_cls and net_prio cgroup** for tagging the traffic control.
– **Devices cgroup** for controlling read/write access on devices.

Docker Engine

Docker Engine combines the namespaces, control groups, etc., into a wrapper called a container format.
The default container format is libcontainer. libcontainer library is a reference implementation for containers, and built on top of libvirt, LXC and systemd-nspawn.
libvirt is an open-source API, daemon and management tool for managing platform virtualization.
LXC -Combines kernel's cgroup and namespaces to provide an isolated environment.
Systemd-nspawn - used to run a command in a lightweight namespace container.

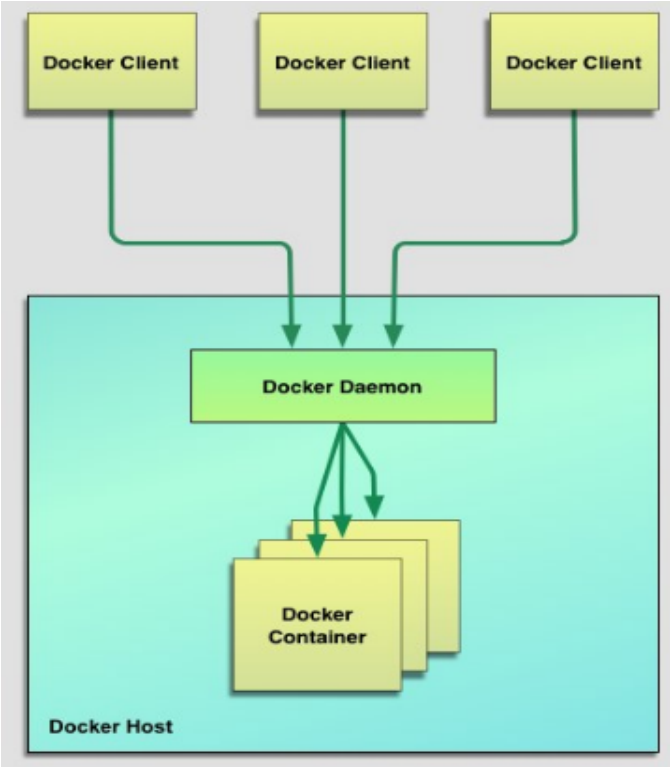


The core components that compose Docker:
– The Docker client and server
– Docker Images
– Registries
– Docker Containers

Docker Image is like an ISO file that you use to install any application or system like Ubuntu ISO file. In order to install that application you need its ISO file and similarly in order to run a Docker container, you need to have its Docker image.

These Docker Images are stored in Registries like Docker Hub.

Docker is a client-server application.
The Docker client talks to the Docker server or daemon, which, in turn, does all the work.
The Docker client and daemon can run on the same system.
The Docker client and daemon communicate using a REST API, over UNIX sockets.
An image is a read-only template with instructions for creating a Docker container.



Common Docker commands for a Docker client
[all of these queries are sent to the Docker server, i.e. Docker Daemon]

docker build => Build your own application (Docker container) and output will be a Docker image which you can now push to a registry like Docker hub.
The docker build command builds Docker images from a Dockerfile and a “context”. A build’s context is the set of files located in the specified PATH or URL. The build process can refer to any of the files in the context. For example, your build can use a *COPY* instruction to reference a file in the context.

docker pull => Specify a Docker image that needs to be pulled, it can be already present on Docker Daemon, if not, then it will be fetched from the registry (Docker Hub).
By default, *docker pull* pulls images from Docker Hub. It is also possible to manually specify the path of a registry to pull from.

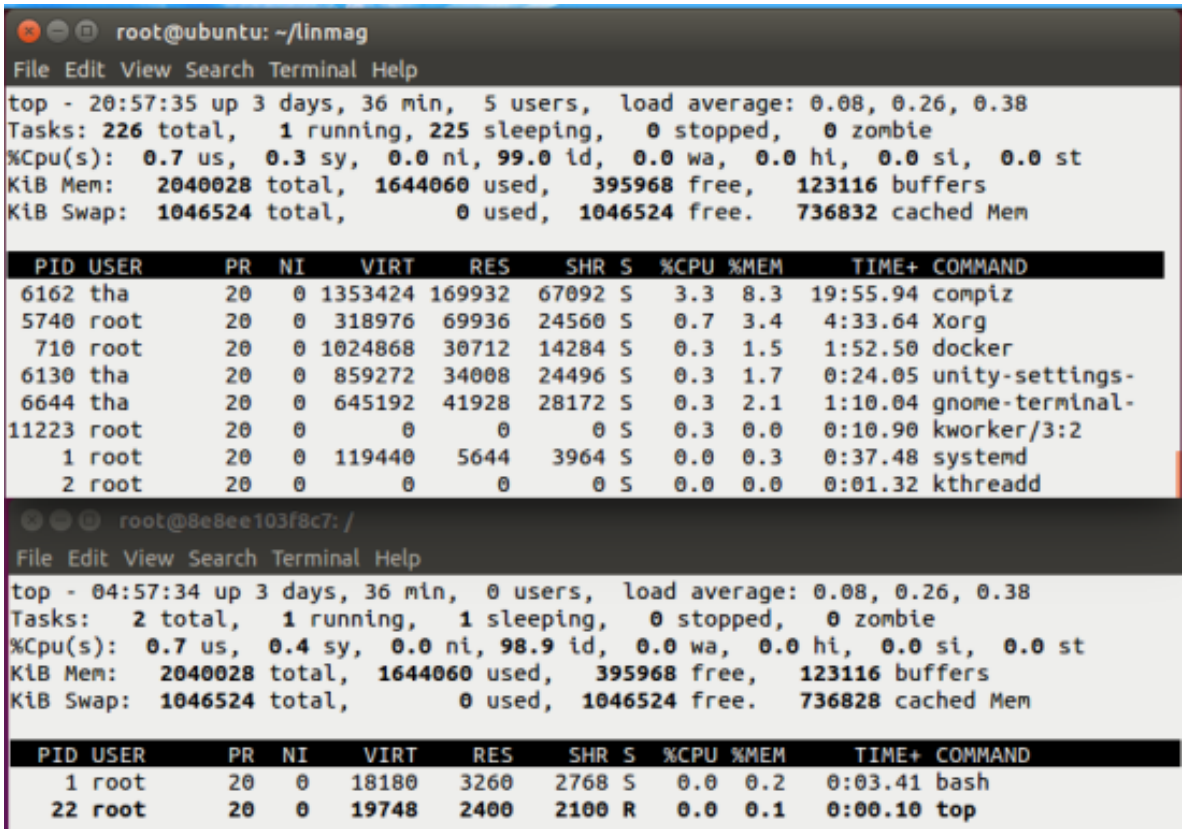
docker run => Run the specified Docker Image
The *docker run* command first creates a writeable container layer over the specified image, and then starts it using the specified command.

Virtualization vs Container

Virtualization	Container
Hardware Level Virtualization	OS Level Virtualization
Typically measured by the gigabyte	Typically measured by the megabyte
Traditional IT architectures	**Emerging IT practices**
House traditional, legacy, and monolithic workloads	cloud-native development, CI/CD, and DevOps
All the code and dependencies in one place led to oversized VMs that experienced cascading failures and downtime when pushing updates.	Workloads are broken into the smallest possible serviceable units and packaged in containers.

Containers are very very lightweight and only run the commands that are absolutely required whereas VM just ends up an entire OS along with its background processes and all of that stuff. Like not even man pages, ifconfig, vim, etc. will be there.

VM's 226 processes compared to just 2 processes running in a Container.



Benefits of Containerization

Portability - is portable and able to run uniformly and consistently across any platform or cloud.

Agility - Software developers can continue using agile or DevOps tools and processes for rapid application development and enhancement.

Speed - speeding up start-times as there is no operating system to boot.

Fault Isolation - The failure of one container does not affect the continued operation of any other containers.

Efficiency - containers are inherently smaller in capacity than a VM, this leads to higher server efficiency.

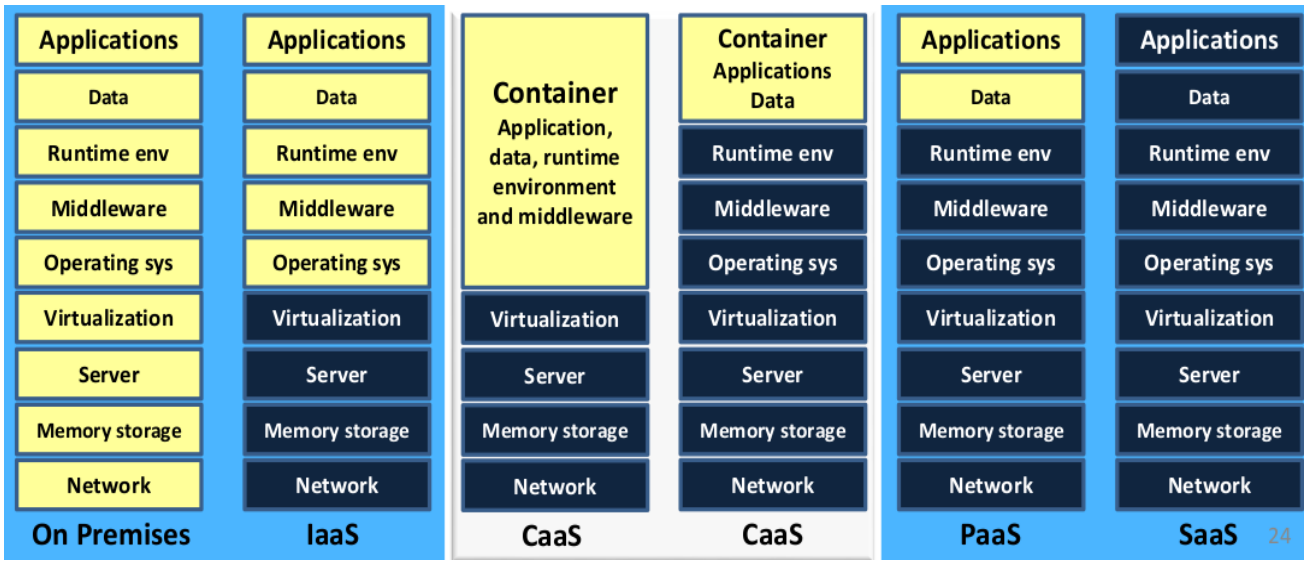
Ease of Management - A container orchestration platform automates the installation, scaling, and management of containerized workloads and services.

Security - security permissions can be defined to automatically block unwanted components from entering containers or limit communications with unnecessary resources.

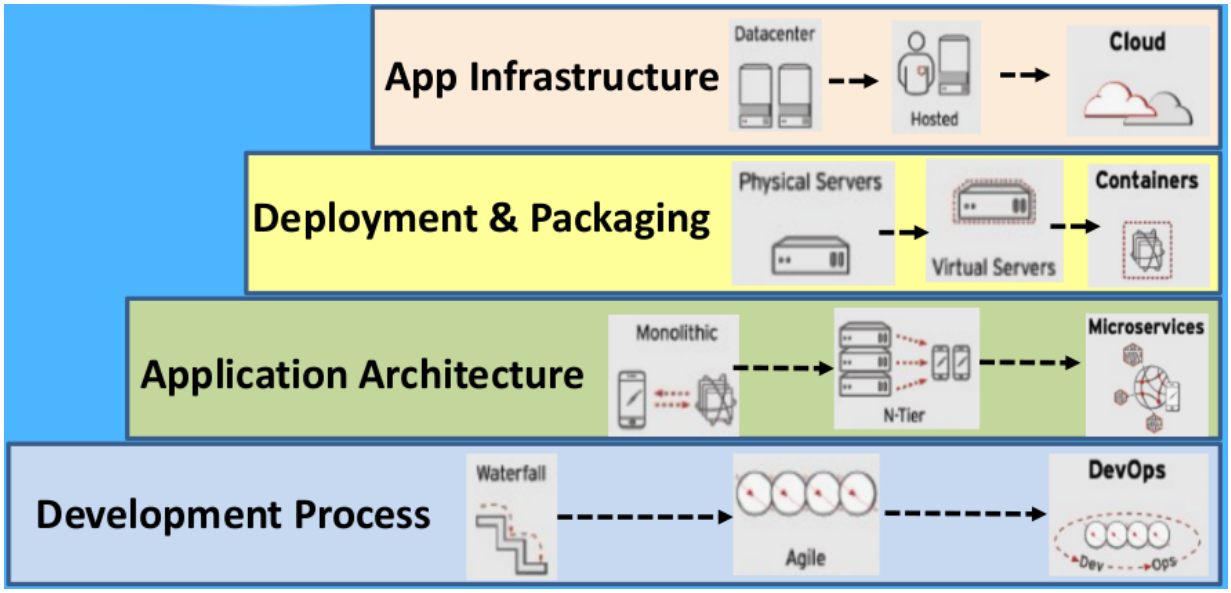
Container as a Service (CaaS)

- CaaS is a business model whereby cloud computing service providers **offer container-based virtualization as a scalable online service.**
- Container-as-a-Service is a computer cluster that is available through the cloud and is used by users to upload, create, centrally manage, and run container-based applications on the cloud platform.
- The interaction with the cloud-based container environment takes place either through the graphical user interface (GUI) or in the form of API calls.
- **CaaS typically refers to a complete container environment**, including orchestration tools, an image catalogue (docker registry), cluster management software, built-in functionality for auto scaling and a set of developer tools and APIs.

CaaS falls somewhere between Infrastructure as a Service (IaaS) and Platform as a Service (PaaS).
However, CaaS is most commonly positioned as a subset of IaaS.



Evolution of Applications



Docker Hands-On

sudo apt-get install docker.io

sudo docker pull ubuntu

Check pulled Docker Images and find Ubuntu images within them,
sudo docker images | grep ubuntu

Check available Docker containers,

sudo docker ps -a (-a = All)

Run image as a Docker container and provide an interactive terminal into the container and give name as “myContainer”
sudo docker run -it --name myContainer ubuntu /bin/bash
[ubuntu = Base Image to run, /bin/bash = Shell Script to use]

Sidenote: “sudo docker run ubuntu /bin/bash” also works but then you cannot directly interact with the container and will have to attach a bash terminal to it later.

Now you’ll be logged in as the root user for that container and now you can interact with the container via the bash shell.

And you can run “sudo docker ps -a” to see the new container.

sudo docker create for creating a container and sudo docker start <container_id> for starting a created Docker container (check man pages for more info).

You can add queries to Docker commands as well in order to programmatically execute Docker commands.

For example, remove all the created/running Docker containers,
sudo docker rm **$(sudo docker ps -a)**

rm needs to be followed by the id’s of the containers to be removed and “sudo docker ps -a” returns all the created/running Docker containers along with their id’s. $ indicates that a query string will follow and parentheses mark the start and end of a query. So all the Docker containers are removed.

Similarly we can start all the created containers using the same command.
Basically just remember the $() query string tip.

docker attach command
Use docker attach to attach your terminal’s standard input, output, and error (or any combination of the three) to a running container using the container’s ID or name. This allows you to view its ongoing output or to control it interactively, as though the commands were running directly in your terminal.

docker attach user1
Run some commands in user1 container

Then you can check the logs (stdin and stdout and commands) of user1 container outside the container, i.e. the host system (your PC), by running this command in a regular terminal,
sudo docker logs user1

sudo docker logs user1 -f => Get real-time logs from the Docker container on the host system, so memory usage, vmstat, etc. of a container can all be monitored in real-time.

Create a Docker Web Server

https://www.tutorialspoint.com/docker/building_web_server_docker_file.htm

Use this tutorial but input the Docker commands directly instead of using a Dockerfile and everything should be the same.

Docker Lab Demo

1. Docker Basic Commands

$apt-get install docker.io (Ref: https://docs.docker.com/engine/install/ubuntu/)
$docker pull ubuntu - pull docker images from dockerhub
$docker images - List all the docker images
$docker create ubuntu
$docker create --name test ubuntu
$docker create -i -t --name user1 ubuntu
$docker start user1
$docker stop user1
$docker attach user1
create containers with user.txt (user.txt contains 10 user names)
copy file from host to docker viceversa:
$docker cp user.txt user1:user.txt (host to docker)
$docker cp user1:user.txt /home/raju/user.txt (docker to host)
$docker ps -a -q -List all the existing container's id
$docker rm $(docker ps -a -q) -remove all the existing containers
$docker rmi <image name> -remove the given docker image

2. Docker access through ssh
create user account : $useradd -m user1 (-m to create /home/user1 directory)
set password for user1: $sudo passwd user1
create user1 as sudo user: $usermod -aG sudo user1
create user1 container: $sudo docker create -it -name user1 ubuntu
Run the container: $sudo docker start user1
Add the following to /home/user1/.profile file: sudo docker start -a -i user1; exit
ssh to <ip address> login as user1 and you will get a user1 container shell, command for this, "ssh user1@ip-address"

[So basically whenever you login to user1, the .profile file will execute and give you entry into the user1 Docker container, so basically user1 has no way of accessing the host system and this is how we can restrict access to users in containers via smart user management]

3. Docker Monitoring
$docker stats $(docker ps -a -q)
$docker logs user1
$docker logs -f user1 (real time)

4. Docker web server demo
$docker start –a user1 -run user1 docker container
$apt-get update
$apt-get install -y apache2
$apt-get install -y apache2-utils
$apt-get install vim
$vim /var/www/html/index.html
$change line number 228:
$service apache2 start (apt-get install systemd)
$elinks <container ip address>

5. Docker image creation and move to docker hub
$docker commit -a <Author Name> <container id> <image name> -Docker image creation
$docker tag <image id> <username/image name:latest>
$docker login (iiitb)
$docker push <username/image name>
dockerhub.com - edit and verify
Docker image stored in Host:
https://blog.thoward37.me/articles/where-are-docker-images-stored/
$cat /var/lib/docker/image/overlay2/repositories.json | python -mjson.tool

03/02/22

MODULE 2: Jenkins Continued

Jenkins Distributed Architecture

Initially, when you start working with Jenkins, you have a single server to carry out all the tasks.

The single Jenkins server is also called the Master node or the Jenkins controller.

A single Jenkins server is not enough in the following scenarios,
- When you configure more jobs

- When you orchestrate more frequent builds
- When more developers depend on one controller
- When you add incremental features in large and complex projects frequently
- When you need different environments (different OS) to test the build

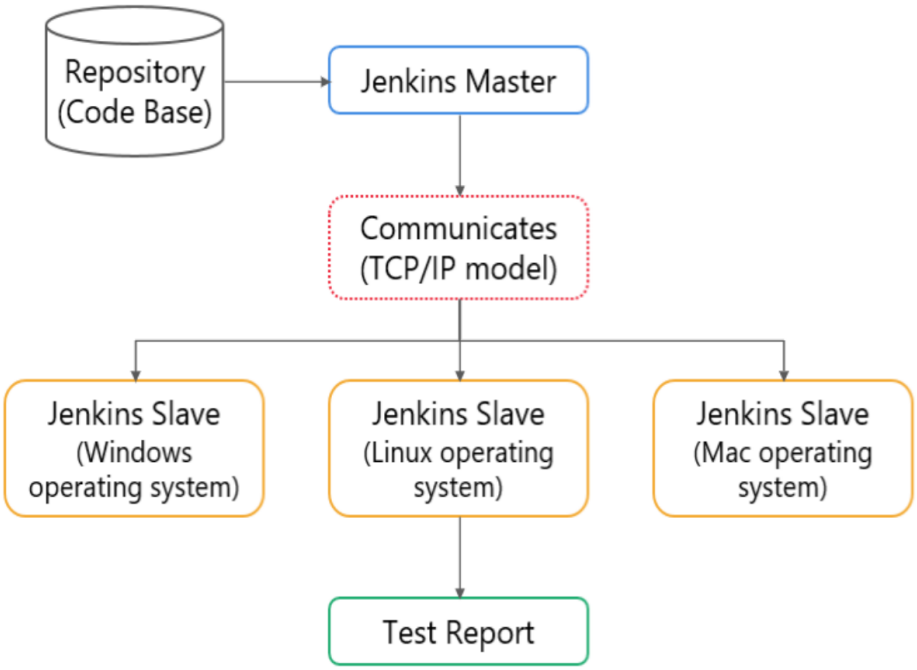
Instead of adding new team members or new projects to an existing single Jenkins controller, you can create additional Jenkins controllers to accommodate new teams or projects.

The Jenkins distributed architecture enables us to use various environments for each build project, dividing the workload across multiple agents running jobs concurrently.

Jenkins' distributed architecture is based on the idea of 'Master + Agent'. The master is responsible for coordination and providing the GUI and API endpoints, while the Agents perform the work (so Agent will not have the GUI or doesn't even need to have the Jenkins package installed, only the required build dependencies and files has to be present on the agent systems).

The Jenkins master manages the Jenkins agents and orchestrates their work by scheduling jobs on agents and monitoring them.

Agents can link to the Jenkins controller via local or cloud computers.



Note: The test reports will be generated by all Agents and they will be available for access in the Jenkins Master.

Demo

From host machine:
sudo su - jenkins
ssh-keygen

In the browser: Manage Jenkins -> Manage Credentials -> click global -> click Add Credentials
For kind - select ssh username with private key and enter the details.

ID: Master_Jenkins_Private_key
Description: Jenkins Master Private Key to Add Multiple Agents
Username: jenkins
Private Key -> Enter Directly ->copy and paste id_rsa from the server
(/var/lib/jenkins/.ssh/id_rsa)

Docker Setup:
sudo apt-get install docker.io
sudo service docker status
sudo docker pull ubuntu
sudo docker run -it --name Jenkins_Agent ubuntu /bin/bash

Inside the container execute the following command:
Running Docker Setup
adduser jenkins
usermod -aG sudo jenkins // Add jenkins as a sudo user
apt-get update
apt-get install sudo
su - jenkins

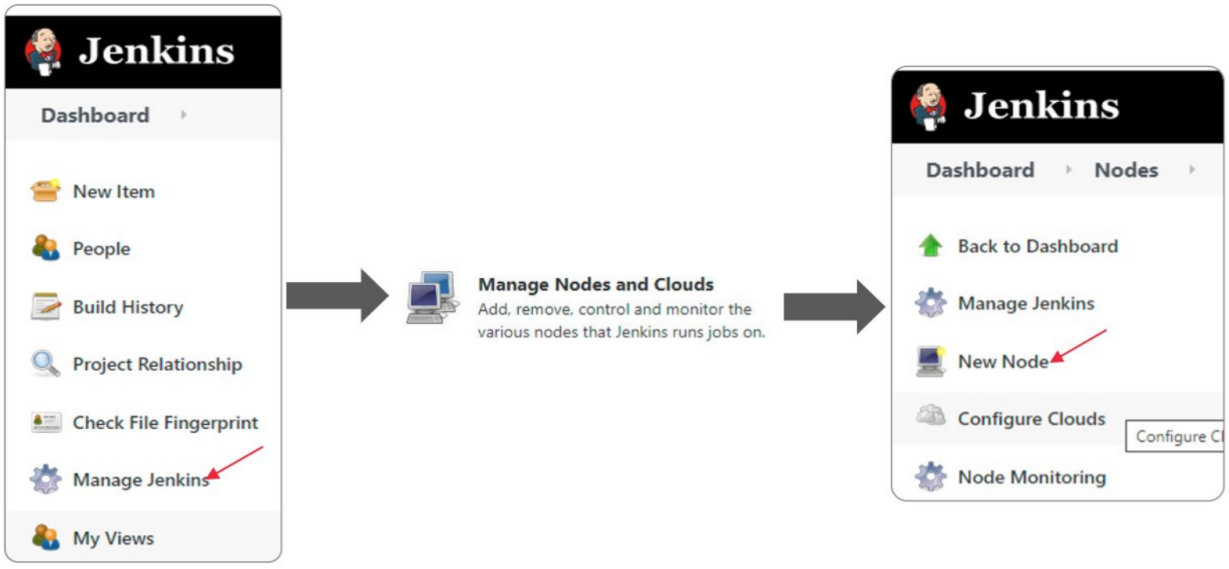
sudo apt-get install openssh-server
sudo service ssh restart
service ssh status
sudo apt install openjdk-11-jdk
java --version

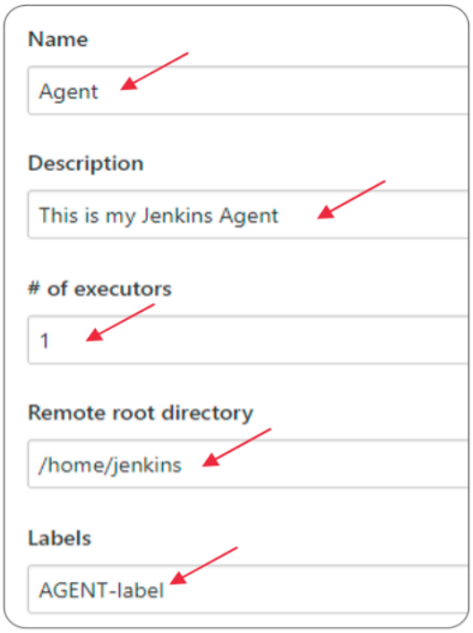
[Jenkins is written in Java, so the Agent machine must also have Java installed for master-agent communication to work properly, even though Agent does not require Jenkins itself to be installed]

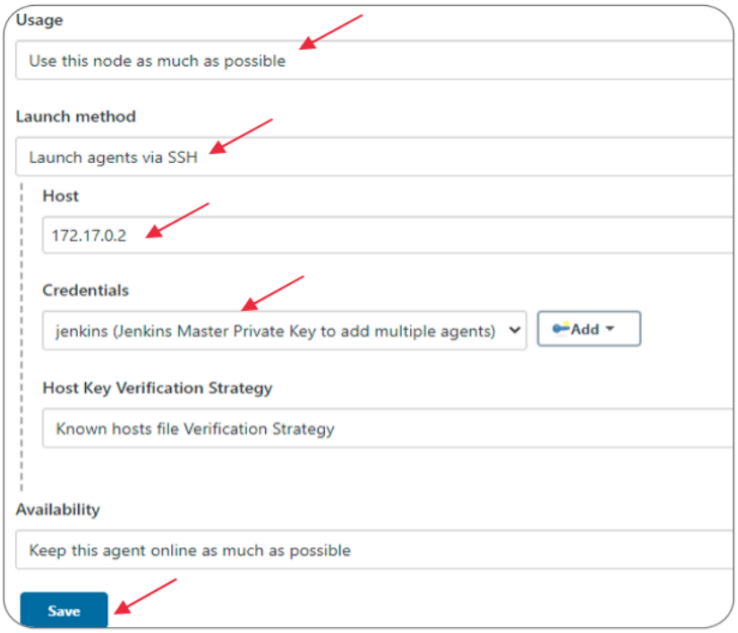
Get IP Address of container => cat /etc/hosts => 172.17.0.2
ssh-copy-id jenkins@172.17.0.2 [jenkins is user name in Agent container]

Now you can ssh into Agent container from host system by doing,
ssh jenkins@172.17.0.2 [enter Jenkins account password]

Configure Agent in Jenkins





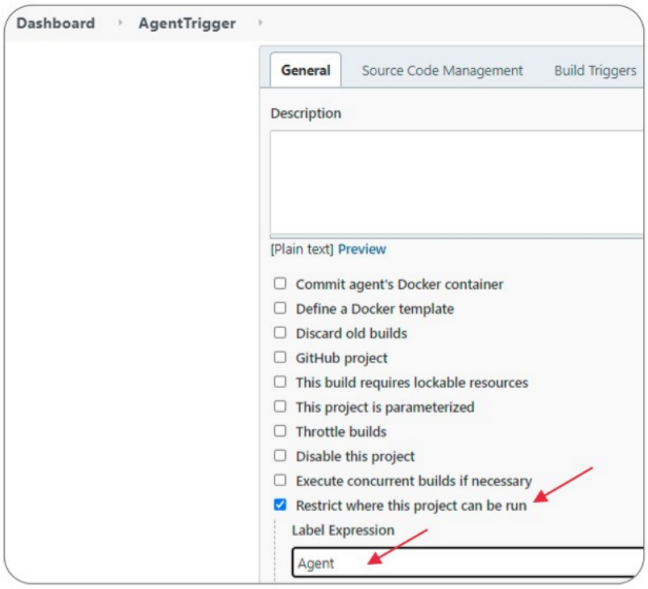


# of executors = Number of cores that will be used by the agent on the system => No. of tasks this agent can execute in parallel.

Remote root directory = Directory on the Agent container that will be used as the root directory [Run pwd on the Agent terminal to see the same path]

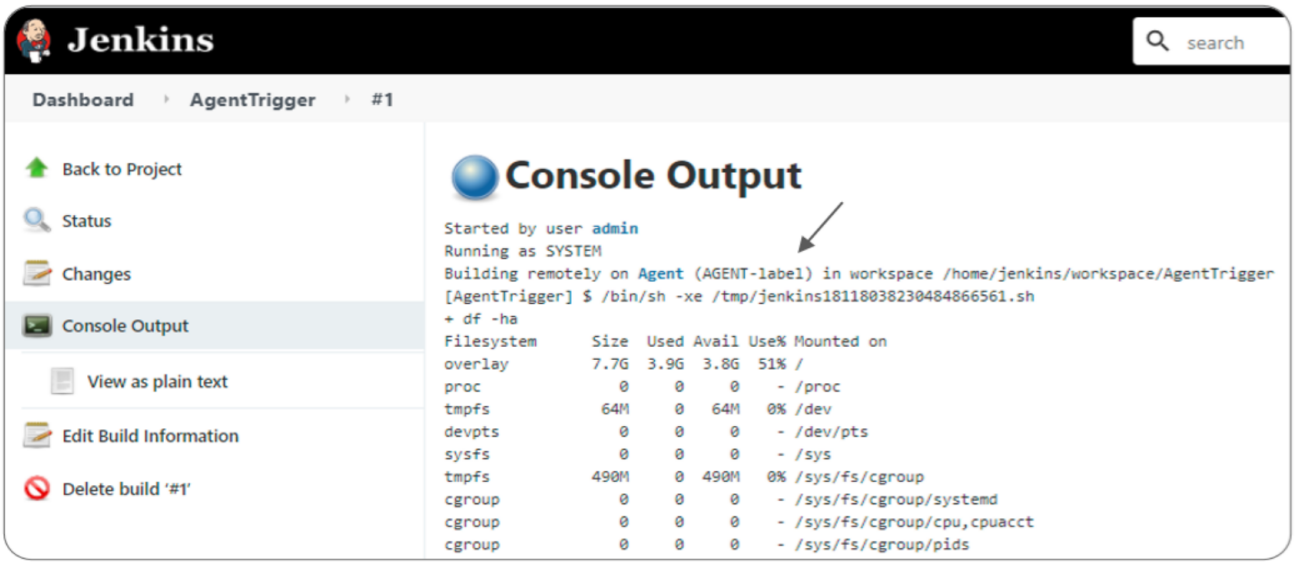
Launch Method => Host => IP Address of Agent Docker container

Create a New Project as AgentTrigger



- Add build script as: df –ha
- Save and build manually

Output:



Benefits of Jenkins Distributed Architecture

- Higher Performance
- High Availability
- Failover Mechanism
- Enhanced Security
- Rollback Mechanism from Machine Failure

SPE Midterm

7 Questions - 10 Marks - No Choice - No MCQ

Module 1 (2 questions)
- Challenges to IT Agility
- DevOps Basics
- Comparison - Traditional, Agile and DevOps [with diagram]
- DevOps barriers - IMP

Git(1 question)
- What is and types [distributed VCS more important with diagram]
- Workflow
- Difference between git clone and git fork with diagrams

CI/CD(3 questions)
- Diagrams for CI and CD [What is CI/CD]
- Benefits for CI-CD
- Jenkins Architecture
- How to build jobs (build triggers)
    - Trigger builds remotely
    - Downstream/upstream jobs => Build after other projects are built
    - Build periodically
    - Poll SCM
    - Github Hook Trigger with Polling
- User Management, Role-based authentication and security realm [need only theoretical POV]
- Pipeline types, directives [general theory on why it's useful and such, no syntax memorisation]
- Jenkins Distributed Architecture - IMP - what is it, why is it useful, architecture with diagram (can explain how we can use Docker containers/VMs as an agent to build jobs via SSH), benefits

Docker (1 question)
- Virtualization vs Containerization
- Docker Architecture and internals (cgroup + namespace)
- Benefits of Containerization + Docker