Module 3: Introduction to DevOps

Module Overview

- This module will introduce learners to DevOps basic concepts and understand the culture to promote development and Operation process collaboratively.

Module Objective

After completion of this session, learners will be able to:

- understand the culture to promote development and Operation process collaboratively.
- Understand the difference in DevOps and agile methodology
- Use tools used for DevOps
- Use GitHub for source code management in software development
- Understand how to work locally and remotely with GIT
- Understand the concepts of branching, merging and rebasing in GIT
- Use Jenkins tool for continuous integration in the project
- Use SonarQube is a Code Quality Assurance tool

Introduction to DevOps

What is DevOps?

- DevOps word in itself is a combination of two words one is Development and other is Operations. It is neither an application nor a tool; instead, it is just a culture to promote development and Operation process collaboratively. As a result of DevOps implementation, the speed to deliver applications and services has increased.
- DevOps enables organizations to serve their customers strongly and better in the market. In other words, we can say that DevOps is the process of alignment of IT and development operations with better and improved communication

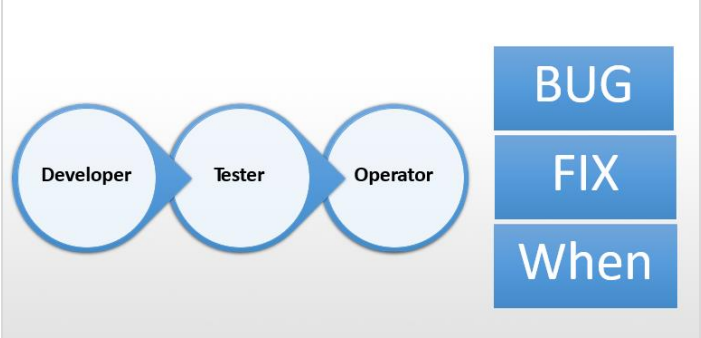
Definition:

DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production while ensuring high quality.

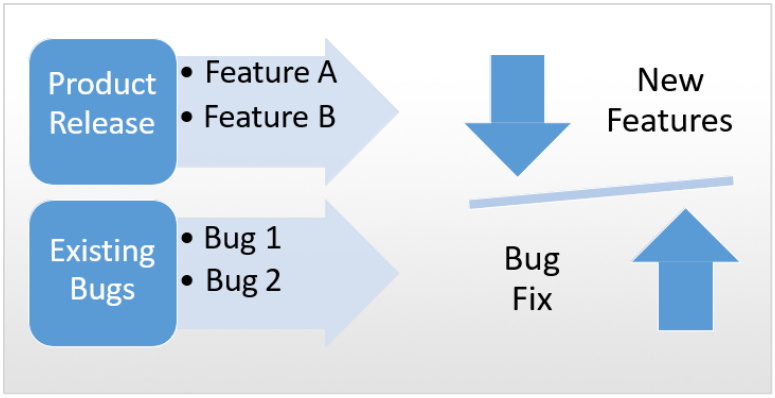
Need for DevOps

Timelines

DevOps deployments are more targeted and isolated, bugs are easier to spot and in turn, fixes are often faster and easier to implement. Your team will mostly need to check the latest code changes to be able to resolve an issue. Resolution times are inherently faster because the responsibility for troubleshooting and fixes remains contained within a single team. In fact, research shows that high-performing DevOps teams recover from failures 168 times faster than lower-performing peers.

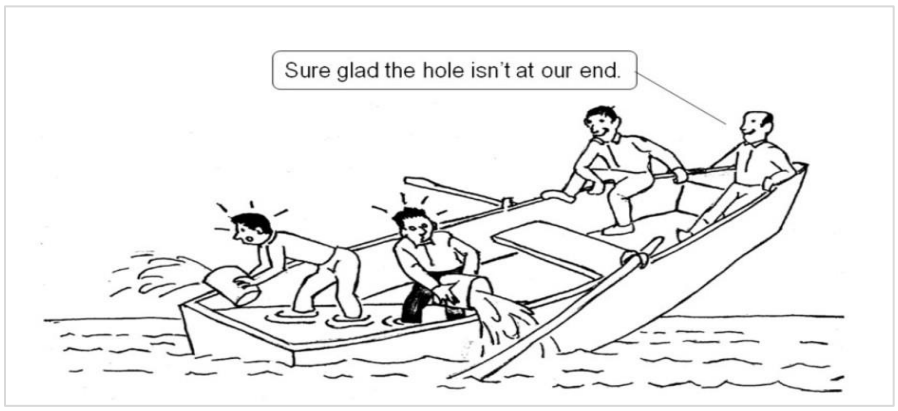


Imbalance



When you deploy a bug fix, a product enhancement, or new features into production, it uncovers other problems or might cause other bugs.

Blame Game

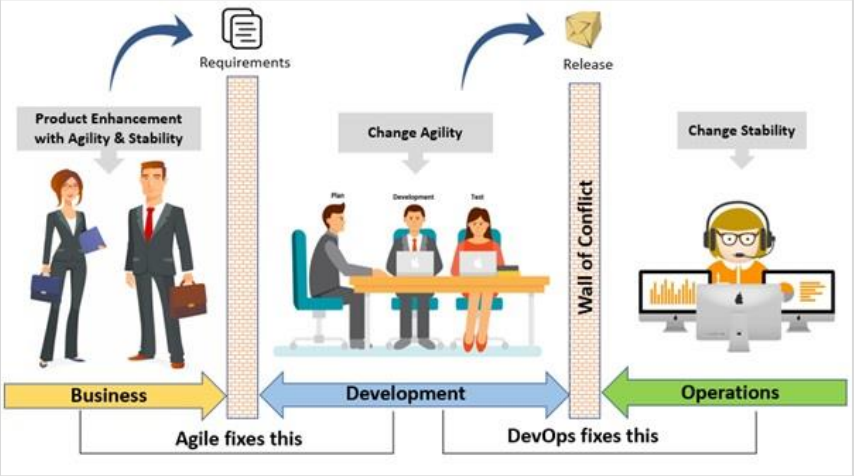


Blaming and punishing people when things go wrong impedes learning and poisons the organizational culture, leading to hiding problems until they become catastrophic. Instead of playing "the blame game" when something goes wrong, practice "Zero Blame."

Building a Zero Blame company culture requires a commitment from everyone in the organization – from executives down to individual contributors. In a Zero Blame culture, individuals and teams are rewarded for exposing problems early on when they can be resolved with less pain and for taking ownership of issues.

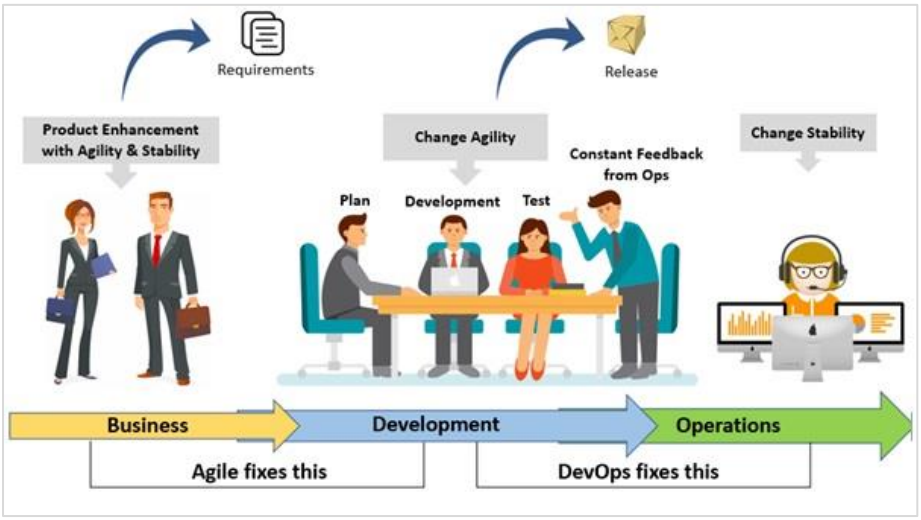
Without DevOps

The Development team works on code which is then sent to the testing team for validation against requirements. The operation team comes in toward the end of the process, where handover of release is given. DevOps aims to break these silos enabling better collaboration and performance.



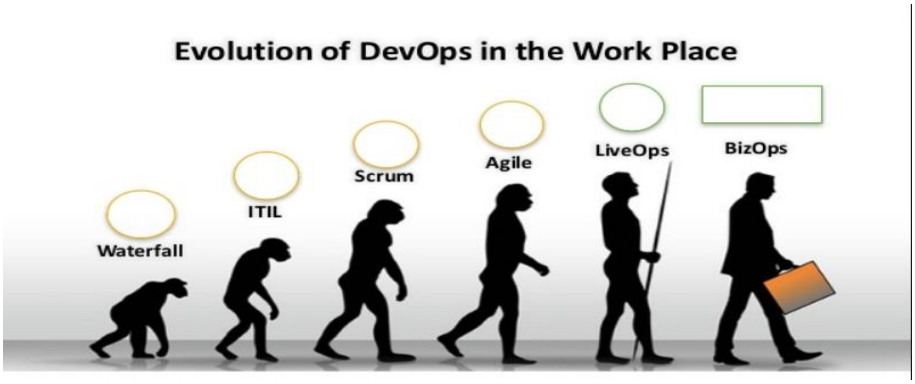
With DevOps

The Development and Operations teams work in collaboration to minimize the effort and risk involved in releasing software. But how do you ensure collaboration? This is the question faced by many organizations. Well, you can introduce collaboration by ensuring that the Operations team is giving constant feedback to the Development team about the code, analyzing the impact considering end-users, and troubleshooting any problems together to gain stability of the product.



DevOps enables a cultural change to remove the barrier between development and operations, working together for a common set of objectives.

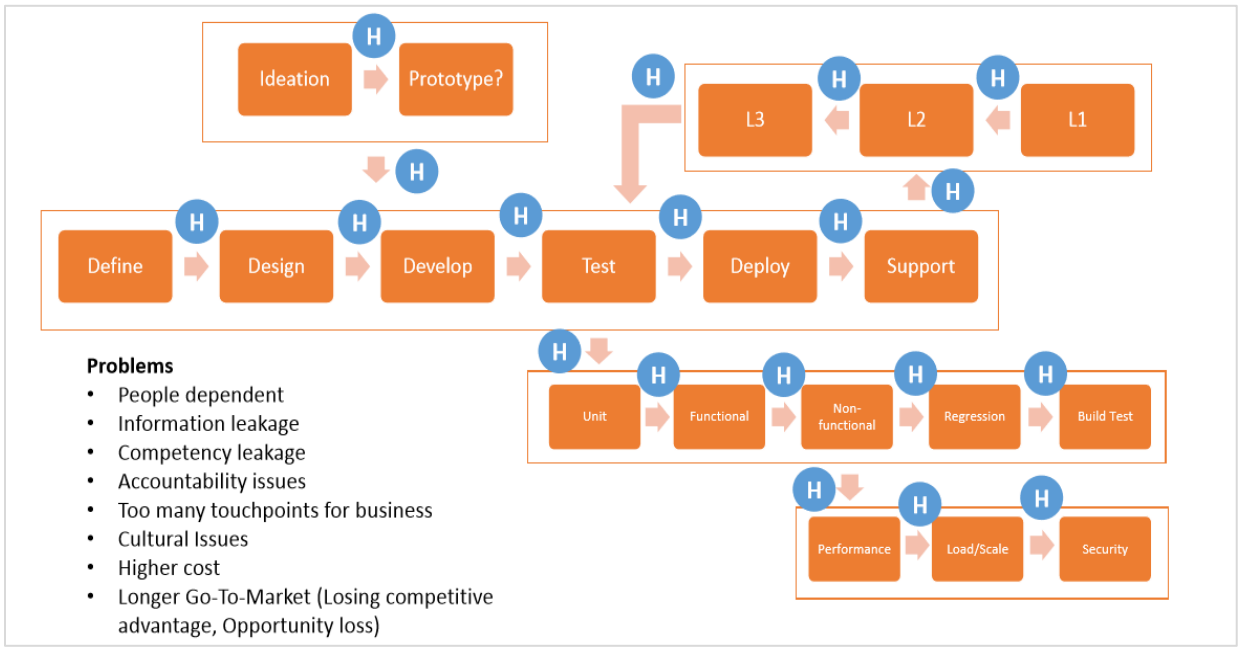
The evolution of DevOps



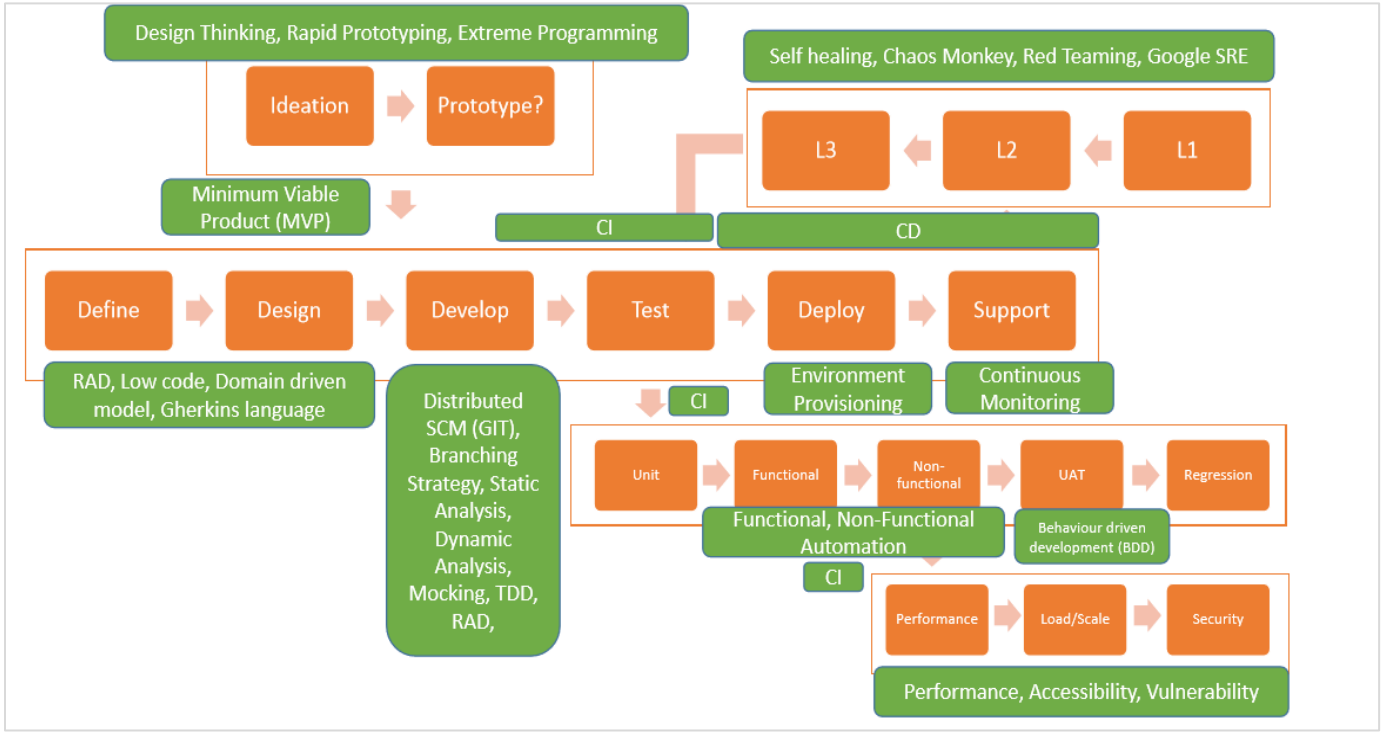
The old world before DevOps

According to the State of DevOps report by DORA, there is a direct co-relation between driving high maturity in DevOps and organizational performance, which thereby, creates better business value. The report found that enterprises that are performing well on the two key DevOps metrics of 'throughput' and 'stability'; including faster lead times from commit-to-deploy, lower change failure rates, and faster incident recovery times; have a significant edge over low performing businesses.

To drive improvements in the above metrics, the most effective way of implementation that I have come across is to adopt the strategy of waste elimination from lean management. What is waste in software development lifecycles? Handoffs. Every handoff is a waste as it reduces throughput, increases the risk of quality, and creates differences. Given below are the different hand-offs that happen in a typical software development lifecycle (SDLC) and the DevOps tools and techniques that can be adopted to eliminate these handoffs



H = Handoffs



No More Handshakes

Each of the tools and techniques to reduce handoffs can be categorized into following areas of DevOps: Continuous planning, continuous integration, continuous deployment, continuous inspection, continuous provisioning, and continuous monitoring. Some of these areas are evolving, for instance, continuous monitoring is evolving into observability as a code. If you have not commenced your DevOps journey, then the best place to start would be in an area closer to production.

Since we are talking scale here, the key point to note on DevOps is to drive it at enterprise level and not at a project or program level. I strongly believe that enterprise DevOps definition available as a platform should be centrally driven while its realization can be done by the feature teams or squads. Though the initial few months (max. 6 months) can be driven through allowing closely monitored experiments across the enterprise, the learning of these experiments should be brought in to the central enterprise DevOps platform. Apart from implementing the tools and techniques called out earlier at an enterprise level, the enterprise DevOps team would also look at improving developer experience through developer portals which can be one-stop-shops to educate on the tools and pipelines in the platform.

The Waterfall Model

The waterfall model is the earlier approach used for software development. It involves teams following a step-by-step process, only proceeding after the previous steps are completed. Each phase needs to be completed before the next phase can begin.

Let's have a look at the steps of the waterfall model.

- **Requirement Gathering and Analysis**

All the system requirements that need to be developed are collected in this phase and documented in a requirement specification document.

- **System Design**

The requirements from the previous phase are studied, and the system design is set up. The system design helps specify the hardware and the system requirements. It also helps define the system's architecture.

- **Implementation**

Based on the system design, small programs called units are developed. These units are integrated into the next phase of the process. Each of these units is developed and tested for their functionality; this process is called Unit Testing.

- **Integration and Testing**

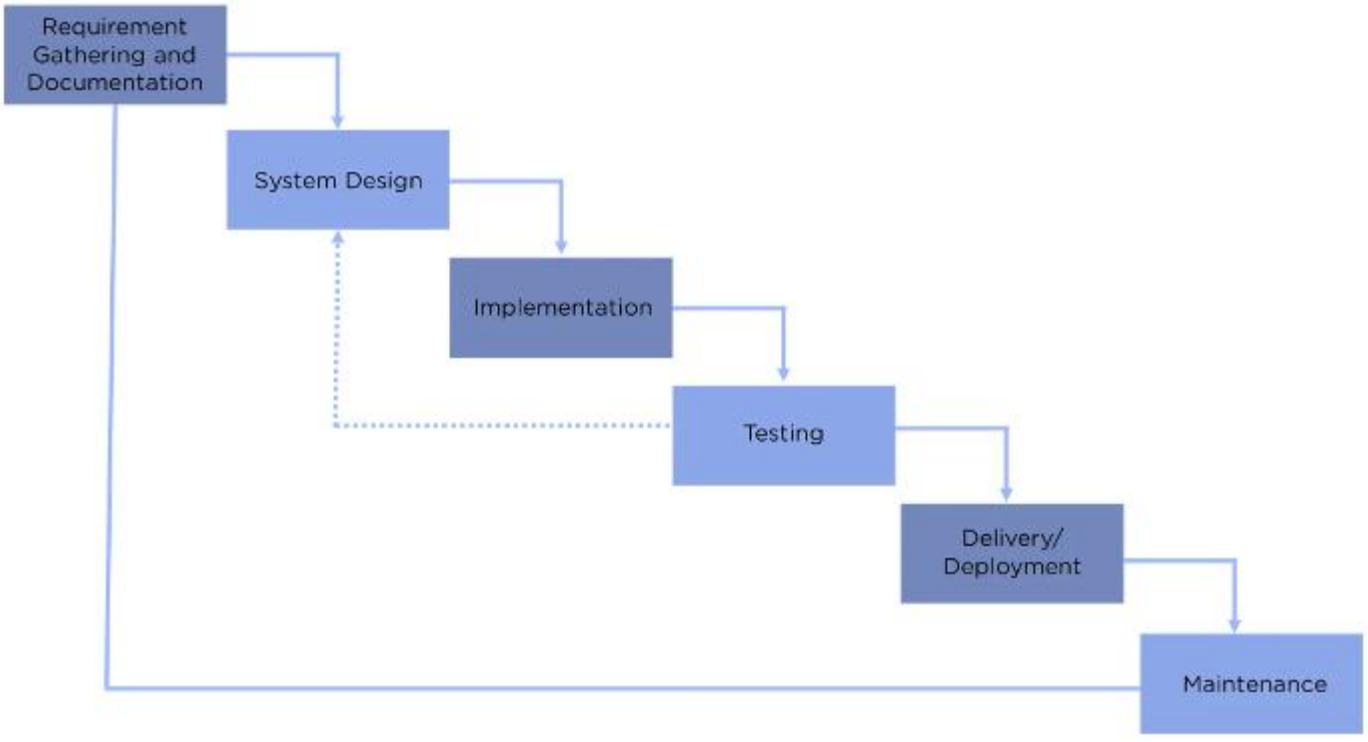
After each unit is tested, it is integrated into a system. After this, the entire system is checked for faults and failures.

- **Deployment of System**

Once functional and non-functional testing is completed, the customer environment is given access or released into the market.

- **Maintenance**

To handle issues that come up in the client environment, patches are released. Maintenance can also help to enhance the project. Maintenance can help with delivering changes to the customer environment.



The Waterfall Model's Disadvantages

- Working software isn't created until late in the project life cycle
- There's a large amount of risk and uncertainty
- Not suited for complex and object-oriented projects
- It is unsuitable for long and ongoing projects
- Measuring the progress within stages are difficult
- Changing requirements cannot be accommodated
- The end-user/client isn't focused on
- Testing is delayed until the project is completed

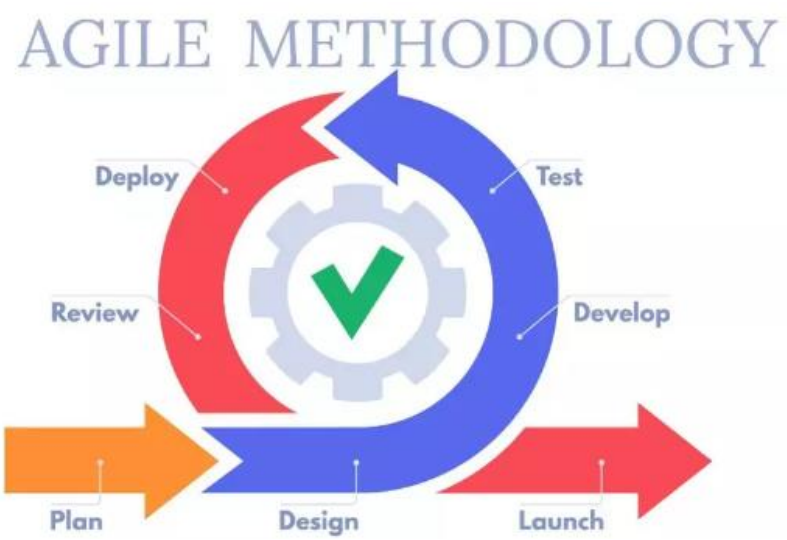
Agile Methodology

What is Agile?

Agile is a collection of principles used in software development and project management. Agile focuses on enabling teams to deliver work in small, workable increments, thus delivering value to their customers with ease. Evaluation of the requirements, plans, and results take place continuously. This helps the team in responding to changes in a quick manner.

The major principles of Agile are detailed in the Agile manifesto. Created in early 2001, the Agile manifesto details the different values and principles that embody the process. The manifesto states:

- Individuals and Interactions OVER Process and Tools
- Working Products OVER Comprehensive Documentation
- Customer Collaboration OVER Contract Negotiation
- Responding to Changes OVER Following a Plan



Agile Methodologies

1. Extreme Programming

It is a framework that enables teams to create high-quality software that helps improve their quality of life. It enables software development alongside appropriate engineering practices. It is applicable while handling changing software requirements risks caused due to new software, working with a small, extended development team, and technology that allows automated unit and functional tests.

2. Kanban

It is a method that's used to design, manage, and improve the flow of systems. Kanban enables organizations to visualize their flow of work and limit the amount of work in progress. It is used in situations where work arrives unpredictably, and where it needs to be deployed immediately without waiting for other work items.

3. Lean

It is a set of tools and principles that focuses on identifying and removing waste to speed up process development. Value is maximized, and waste is minimized. It is used in just about every industry that produces waste in some form.

4. Scrum

It is a framework used by teams to establish a hypothesis, test it, reflect on the experience, and make adjustments. It enables teams to incorporate practices from other frameworks depending on the requirements. It is used by cross-

functional teams that are working on product development, and the work is split into more than one 2-4 week iterations.

5. Crystal

It focuses on people and their interactions, rather than on tools and processes. Aimed to streamline processes and improve optimization, Crystal works on the principle that projects are unique and dynamic. It is used when the focus is on strengthening team communication, continuous integration, active user involvement, and configurable processes.

Agile Principles

To make a process Agile, the following principles need to be satisfied.

1. **Customer Satisfaction**

The customer needs to be satisfied with the quick delivery of the product.

2. **Welcome Change**

Even late in the development process, changing needs need to be addressed.

3. **Deliver Frequently**

Focus on a shorter timescale, and ensure products are delivered frequently.

4. **Work Together**

The business and development team need to work together through the course of the project.

5. **Motivated Team**

Team members must be motivated and trusted to complete the project successfully and on time.

6. **Face-to-face**

Having face-to-face interactions is one of the most effective forms of communication.

7. **Working Software**

Having working output is an indication of the progress made towards the final product.

8. **Constant Pace**

Agile promotes sustainable development.

9. **Good Design**

Improve agility by focusing on good design and technical excellence.

10. **Simplicity**

The amount of time where work isn't being done needs to be reduced.

11. **Self-Organization**

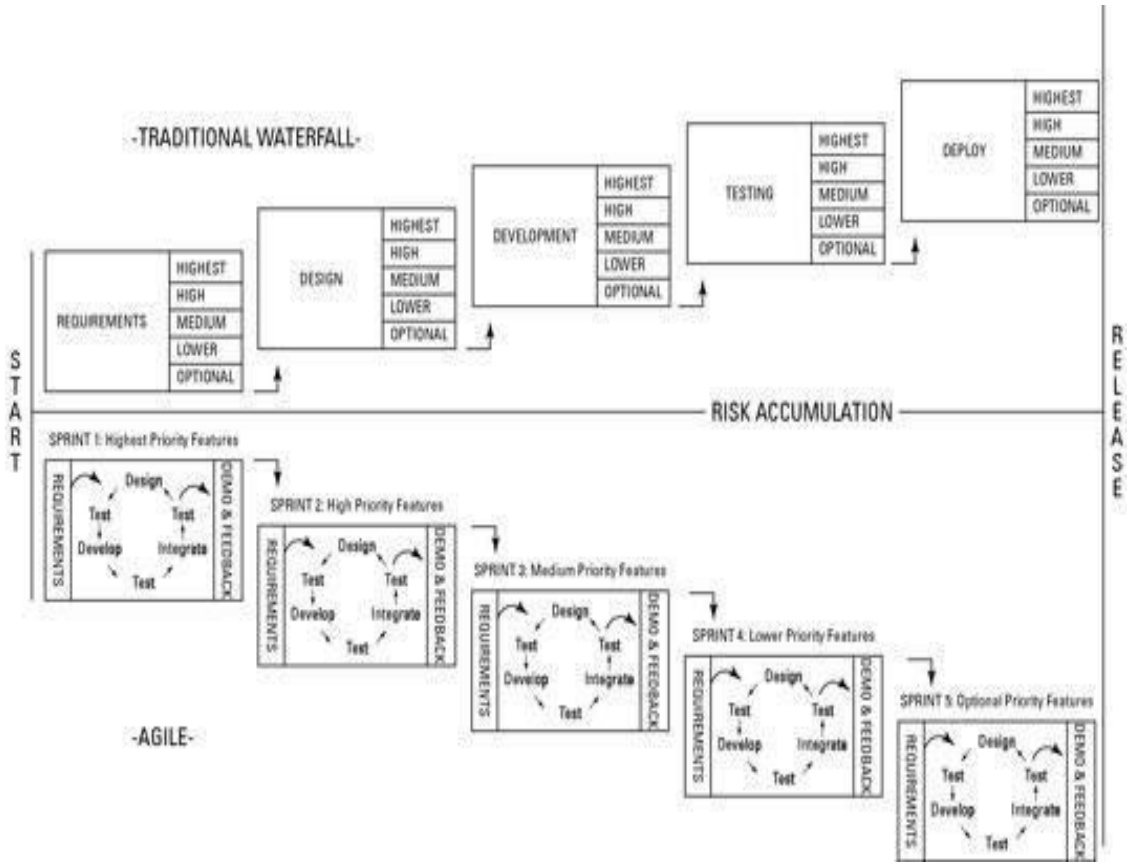
These types of teams provide the best designs, requirements, and architectures.

12. **Reflect and Adjust**

The effectiveness of the team can be improved by regularly reflecting on their work and making improvements.

Advantages of Agile

- Agile enables a large amount of collaboration and interaction between the client and the project team.
- Thanks to this, clients have improved transparency, and therefore a clearer understanding of the phases of the project is present.
- The product is delivered predictably, or sometimes earlier than expected.
- The cost of the project is predictable and follows a rigid schedule.
- Changes can refine and re-prioritize the product backlog.
- Enables the client to prioritize different features, allowing the team to ensure maximum project value.
- The project is broken down into smaller units, providing high-quality development, testing, and collaboration.



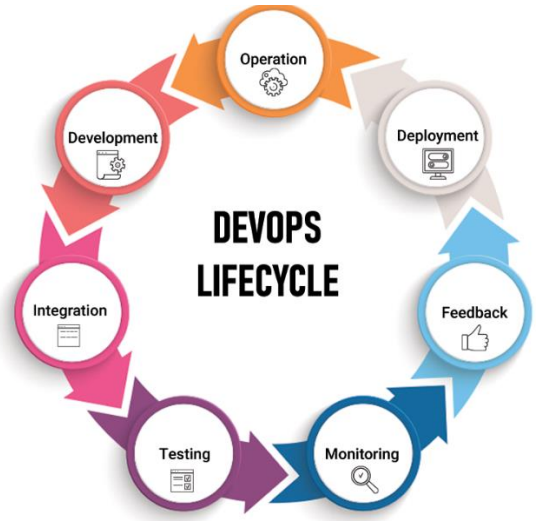
Agile vs DevOps

DevOps and Agile are the two software development methodologies with similar aims, getting the end-product as quickly and efficiently as possible. While many organizations are hoping to employ these practices, there is often some confusion between both methodologies.

Parameter	DevOps	Agile
Definition	DevOps is a practice of bringing development and operation teams together.	Agile refers to the continuous iterative approach, which focuses on collaboration, customer feedback, small, and rapid releases.
Purpose	DevOps purpose is to manage end to end engineering processes.	The agile purpose is to manage complex projects.
Task	It focuses on constant testing and delivery.	It focuses on constant changes.
Team size	It has a large team size as it involves all the stack holders.	It has a small team size. As smaller is the team, the fewer people work on it so that they can

		move faster.
Team skillset	The DevOps divides and spreads the skill set between development and the operation team.	The Agile development emphasizes training all team members to have a wide variety of similar and equal skills.
Implementation	DevOps is focused on collaboration, so it does not have any commonly accepted framework.	Agile can implement within a range of tactical frameworks such as **safe, scrum**, and **sprint**.
Duration	The ideal goal is to deliver the code to production daily or every few hours.	Agile development is managed in units of sprints. So, this time is much less than a month for each sprint.
Target areas	End to End business solution and fast delivery.	Software development.
Feedback	Feedback comes from the internal team.	In Agile, feedback is coming from the customer.
Shift left principle	It supports both variations left and right.	It supports only shift left.
Focus	DevOps focuses on operational and business readiness.	Agile focuses on functional and non-functional readiness.
Importance	In DevOps, developing, testing, and implementation all are equally important.	Developing software is inherent to Agile.
Quality	DevOps contributes to creating better quality with automation and early bug removal. Developers need to follow Coding and best Architectural practices to maintain quality standards.	The Agile produces better applications suites with the desired requirements. It can quickly adapt according to the changes made on time during the project life.
Tools	**Puppet, Chef, AWS, Ansible**, and team City OpenStack are popular DevOps tools.	**Bugzilla, Kanboard, JIRA** are some popular Agile tools.
Automation	Automation is the primary goal of DevOps. It works on the principle of maximizing efficiency when deploying software.	Agile does not emphasize on the automation.
Communication	DevOps communication involves specs and design documents. It is essential for the operational team to fully understand the software release and its network implications for the enough running the deployment process.	Scrum is the most common method of implementing Agile software development. Scrum meeting is carried out daily.
	In the DevOps, the process documentation is foremost because it will send the software to an operational team for deployment. Automation minimizes the impact of insufficient documentation. However, in the development of sophisticated software, it's difficult to transfer all the knowledge required.	The agile method gives priority to the working system over complete documentation. It is ideal when you are flexible and responsive. However, it can harm when you are trying to turn things over to another team for deploym

DevOps Lifecycle



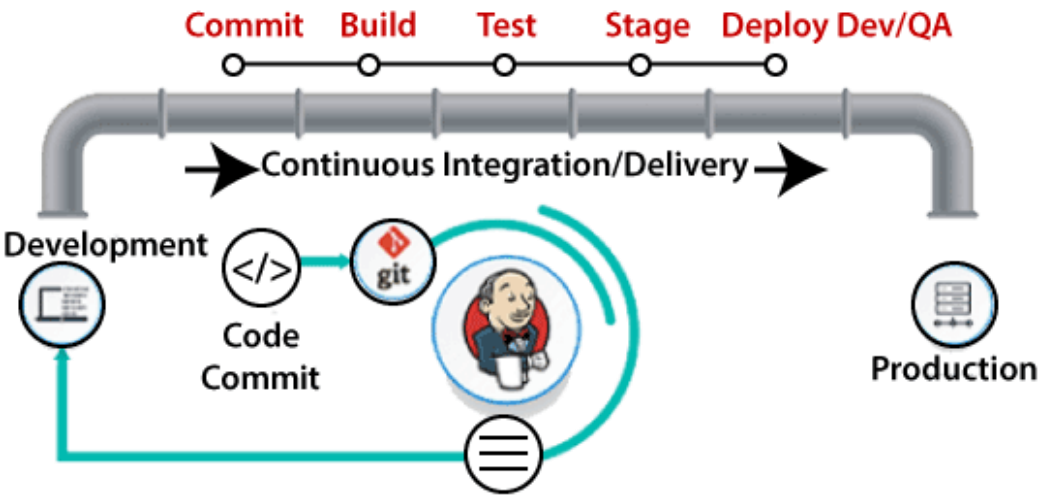
1) Continuous Development

This phase involves the planning and coding of the software. The vision of the project is decided during the planning phase. And the developers begin developing the code for the application. There are no DevOps tools that are required for planning, but there are several tools for maintaining the code.

2) Continuous Integration

This stage is the heart of the entire DevOps lifecycle. It is a software development practice in which the developers require to commit changes to the source code more frequently. This may be on a daily or weekly basis. Then every commit is built, and this allows early detection of problems if they are present. Building code is not only involved compilation, but it also includes unit testing, integration testing, code review, and packaging.

The code supporting new functionality is continuously integrated with the existing code. Therefore, there is continuous development of software. The updated code needs to be integrated continuously and smoothly with the systems to reflect changes to the end-users.



Jenkins is a popular tool used in this phase. Whenever there is a change in the Git repository, then Jenkins fetches the updated code and prepares a build of that code, which is an executable file in the form of war or jar. Then this build is forwarded to the test server or the production server.

3) Continuous Testing

This phase, where the developed software is continuously testing for bugs. For constant testing, automation testing tools such as TestNG, JUnit, Selenium, etc are used. These tools allow QAs to test multiple code-bases thoroughly in parallel to ensure that there is no flaw in the functionality. In this phase, Docker Containers can be used for simulating the test environment.

Selenium does the automation testing, and TestNG generates the reports. This entire testing phase can automate with the help of a Continuous Integration tool called Jenkins.

Automation testing saves a lot of time and effort for executing the tests instead of doing this manually. Apart from that, report generation is a big plus. The task of evaluating the test cases that failed in a test suite gets simpler. Also, we can schedule the execution of the test cases at predefined times. After testing, the code is continuously integrated with the existing code.

4) Continuous Monitoring

Monitoring is a phase that involves all the operational factors of the entire DevOps process, where important information about the use of the software is recorded and carefully processed to find out trends and identify problem areas. Usually, the monitoring is integrated within the operational capabilities of the software application.

It may occur in the form of documentation files or maybe produce large-scale data about the application parameters when it is in a continuous use position. The system errors such as server not reachable, low memory, etc are resolved in this phase. It maintains the security and availability of the service.

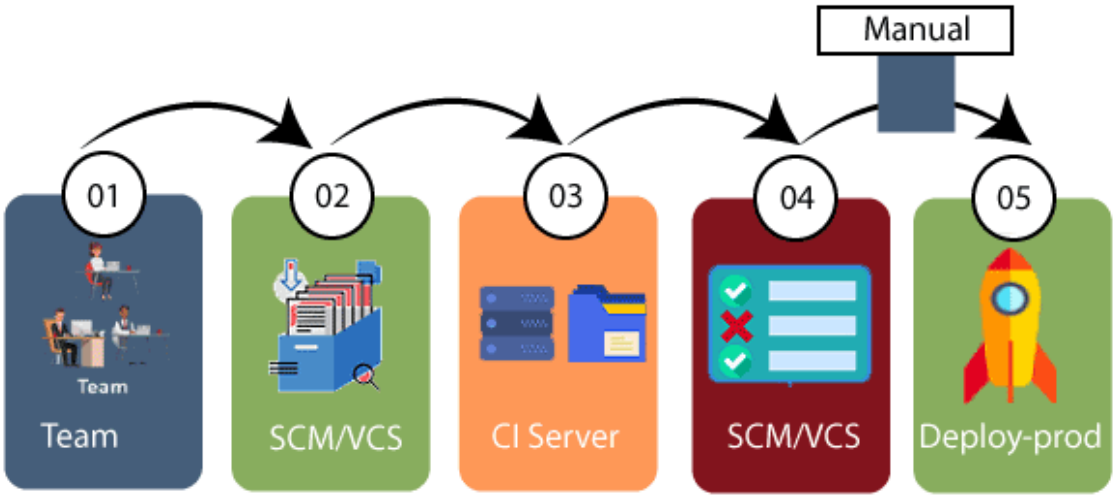
5) Continuous Feedback

The application development is consistently improved by analyzing the results from the operations of the software. This is carried out by placing the critical phase of constant feedback between the operations and the development of the next version of the current software application.

The continuity is the essential factor in the DevOps as it removes the unnecessary steps which are required to take a software application from development, using it to find out its issues and then producing a better version. It kills the efficiency that may be possible with the app and reduce the number of interested customers.

6) Continuous Deployment

In this phase, the code is deployed to the production servers. Also, it is essential to ensure that the code is correctly used on all the servers.



The new code is deployed continuously, and configuration management tools play an essential role in executing tasks frequently and quickly. Here are some popular tools which are used in this phase, such as Chef, Puppet, Ansible, and SaltStack.

Containerization tools are also playing an essential role in the deployment phase. Vagrant and Docker are popular tools that are used for this purpose. These tools help to produce consistency across development, staging, testing, and production environment. They also help in scaling up and scaling down instances softly.

Containerization tools help to maintain consistency across the environments where the application is tested, developed, and deployed. There is no chance of errors or failure in the production environment as they package and replicate the same dependencies and packages used in the testing, development, and staging environment. It makes the application easy to run on different computers.

7) Continuous Operations

All DevOps operations are based on the continuity with complete automation of the release process and allow the organization to accelerate the overall time to market continuingly.

It is clear from the discussion that continuity is the critical factor in the DevOps in removing steps that often distract the development, take it longer to detect issues and produce a better version of the product after several months. With DevOps, we can make any software product more efficient and increase the overall count of interested customers in your product.

DevOps Stages

Phase 1: Plan

Roughly equivalent to the requirements-gathering phase in traditional software development, the planning stage of a DevOps pipeline includes everything that happens before developers start writing code.

What distinguishes the DevOps way of planning is a core focus on customers' needs and experiences. Feedback from all stakeholders is integrated into the earliest stages of planning, and a collaborative approach occurs between product and project managers and developers. Collaboration tools and project management solutions are used to track the project's progress throughout all of the stages.

Phase 2: Build

In DevOps, the planning and coding phases flow into and inform one another. DevOps practitioners build in small modules, relying on automated tools to maintain version control, enforce consistent style standards and guard against security issues.

In traditional development environments, software testing is a separate phase that's performed by quality assurance (QA) teams after both individual components and the integrated application are complete. The DevOps approach instead integrates automated testing across all stages of software production. Developers run their own QA scripts early in the build process, an approach known as "shift left." This makes it possible to correct errors sooner, saving time and improving product quality.

Phase 3: Continuous Integration

Continuous integration (CI) is at the heart of the DevOps methodology. It involves frequently merging the new code written by multiple developers into a single, centralized repository. Each team member's contributions are added to the main branch of source code on a regular schedule, usually at least once a day. Automated tools are used to fetch the code, prepare the build and test it before forwarding to the appropriate staging environment or repository.

Phase 4: Deploy

What CI is to the central code repository, continuous delivery or continuous deployment (CD) is to the end user-facing production environment. The DevOps methodology calls for new features, updates or other changes to the application to be released into production on an ongoing and frequent basis.

DevOps practitioners use automated provisioning solutions to manage and monitor both test and production

environments (a concept known as infrastructure as code). This way they can be confident they're testing the software in environments configured identically to the production environment and know what works in testing will work in production. Once the code passes this automated testing, it's ready for deployment. And, DevOps practices rely on automated tools to perform this deployment.
Many leverage blue/green deployment strategies, in which new code releases are automatically applied to half the production environment. If any issues are found, it's simple to roll back the changes until they are addressed. If not, the remainder of the end-users (the other half) receive the updated version.

Phase 5: Operate
Once DevOps software has gone live in production, the operations team will rely on automated tools wherever possible for configuration management, scaling and load balancing.

Phase 6: Continuous Feedback
A DevOps organization will leverage automated solutions for soliciting and collecting customers' feedback on their experiences, and for capturing metrics describing the software's performance. This information will inform the DevOps team's future efforts to further improve the application. It serves as an essential input to the planning stage of the pipeline.
.

DevOps Practices

Five different categories of DevOps practices
We have identified five different categories of DevOps practices, which satisfy our definition. For each practice, we discuss the architectural implications.

Treat Ops as first-class citizens from the point of view of requirements: These practices fit in the high-quality aspect of the definition. Operations have a set of requirements that pertain to logging and monitoring. For example, logging messages should be understandable and useable by an operator. Involving operations in the development of requirements will ensure that these types of requirements are considered.

Adding requirements to a system from Ops may require some architectural modification. In particular, the Ops requirements are likely to be in the area of logging, monitoring, and information to support incident handling. These requirements will be like other requirements for modifications to a system: possibly requiring some minor

modifications to the architecture but, typically, not drastic modifications.

Make Dev more responsible for relevant incident handling: These practices are intended to shorten the time between the observation of an error and the repair of that error. Organizations that utilize these practices typically have a period of time in which Dev has primary responsibility for a new deployment; later on, Ops has primary responsibility.

By itself, this change is just a process change and should require no architectural modifications. However, just as with the previous practice, once Dev becomes aware of the requirements for incident handling, some architectural modifications may result.

Continuous deployment: Practices associated with continuous deployment are intended to shorten the time between a developer committing code to a repository and that code being deployed. Continuous deployment also emphasizes automated tests, to increase the quality of code making its way into production. Continuous deployment is the practice that leads to the most far-reaching architectural modifications. On the one hand, an organization can introduce continuous deployment practices with no major architectural changes. On the other hand, organizations that have adopted continuous deployment practices frequently begin moving to a microservice architecture. We cover microservice architectures and explore the reasons for adoption below.

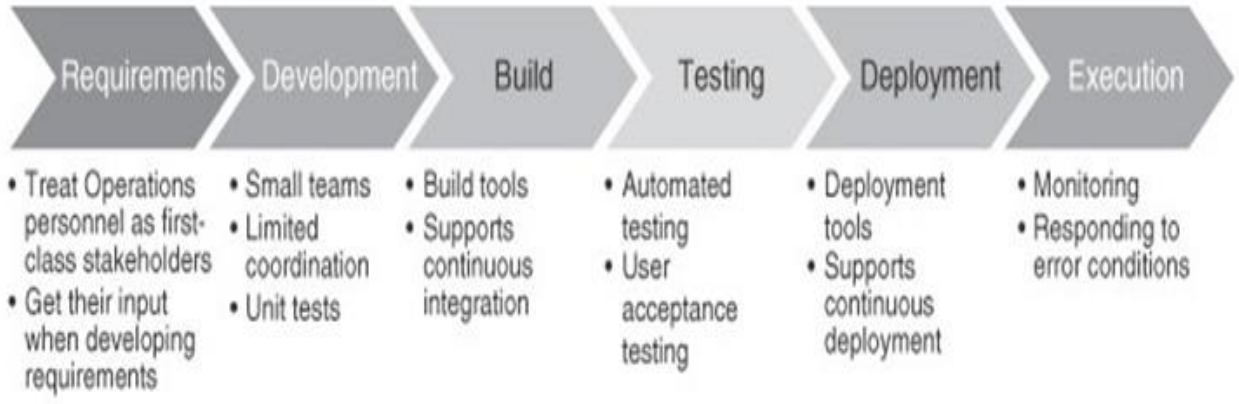
Develop infrastructure code with the same set of practices as application code: Practices that apply to the development of infrastructure code are intended both to ensure high quality in the deployed applications and to ensure that deployments proceed as planned. Errors in deployment scripts such as misconfigurations can cause errors in the application, in the environment, or in the deployment process. Applying quality control practices used in normal software development when developing operations scripts and processes will help control the quality of these specifications. These practices will not affect the application code but may affect the architecture of the infrastructure code.

Enforced deployment process used by all, including Dev and Ops personnel: These practices are intended to ensure a higher quality of deployments, e.g., by requiring the continuous deployment pipeline to be used for any change, even a small change in configuration. This avoids errors caused by ad hoc deployments and resulting misconfiguration. The practices also refer to the time that it takes to diagnose and repair an error. The normal deployment process should make it easy to trace the history of a particular virtual machine image and understand

the components that were included in that image.

In general, when a process becomes enforced, some individuals may be required to change their normal operating procedures and, possibly, the structure of the systems on which they work. One point where a deployment process could be enforced is in the initiation phase of each system. Each system, when it is initialized, verifies its pedigree. That is, it arrived at execution through a series of steps, each of which can be checked to have occurred. Furthermore, the systems on which it depends, e.g., operating systems or middleware, also have verifiable pedigrees.

DevOps lifecycle processes:



DevOps life cycle processes [Notation: Porter's Value Chain]

DevOps advocates treating operations personnel as first-class stakeholders. Preparing a release can be a very serious and onerous process. As such, operations personnel may need to be trained in the types of runtime errors that can occur in a system under development; they may have suggestions as to the type and structure of log files, and they may provide other types of input into the requirements process. At its most extreme, DevOps practices make developers responsible for monitoring the progress and errors that occur during deployment and execution, so theirs would be the voices suggesting requirements. In between are practices that cover teams, build processes, testing processes, and deployment processes.

DevOps Delivery Pipeline

A pipeline in software engineering team is a set of automated processes which allows DevOps professionals and developer to reliably and efficiently compile, build, and deploy their code to their production compute platforms.

The most common components of a pipeline in DevOps are build automation or continuous integration, test automation, and deployment automation.

A pipeline consists of a set of tools which are classified into the following categories such as:

- Source control
- Build tools
- Containerization
- Configuration management
- Monitoring
- Continuous Integration Pipeline

Continuous integration (CI) is a practice in which developers can check their code into a version-controlled repository several times per day. Automated build pipelines are triggered by these checks which allows fast and easy to locate error detection.

Some significant benefits of CI are:

- Small changes are easy to integrate into large codebases.
- More comfortable for other team members to see what you have been working.
- Fewer integration issues allowing rapid code delivery.
- Bugs are identified early, making them easier to fix, resulting in less debugging work.

Continuous Delivery Pipeline

Continuous delivery (CD) is the process that allows operation engineers and developers to deliver bug fixes, features, and configuration change into production reliably, quickly, and sustainably. Continuous delivery offers the benefits of code delivery pipelines, which are carried out that can be performed on demand.

Some significant benefits of the CD are:

Faster bug fixes and features delivery.

CD allows the team to work on features and bug fixes in small batches, which means user feedback received much quicker. It reduces the overall time and cost of the project.

DevOps Methodology

We have a demonstrated methodology that takes an approach to cloud adoption. It accounts for all the factors required for successful approval such as people, process, and technology, resulting in a focus on the following critical consideration:

- **The Teams:** Mission or project and cloud management.
- **Connectivity:** Public, on-premise, and hybrid cloud network access.
- **Automation:** Infrastructure as code, scripting the orchestration and deployment of resources.
- **On-boarding Process:** How the project gets started in the cloud.
- **Project Environment:** TEST, DEV, PROD (identical deployment, testing, and production).
- **Shared Services:** Common capabilities provided by the enterprise.
- **Naming Conventions:** Vital aspect to track resource utilization and billing.
- **Defining Standards Role across the Teams:** Permissions to access resources by job function.

CI/CD is all about DevOps

One of the biggest misconceptions about DevOps is that it's the same thing as CI/CD. The truth is that continuous integration and delivery are the key components of DevOps.

DevOps focuses on the culture and responsibility in a team. It emphasizes the need for everyone on the team to take part in each other's tasks. This improves collaboration and communication in the team.

On the other hand, CI/CD enables this culture with software and tools that emphasize automation. You can see them as a means to an end.

DevOps means NoOps

NoOps describes the concept where the cloud infrastructure is so automated, that there is no need to manage it.

NoOps is considered as the next evolution of DevOps as a development model. Just like DevOps, its goal is to improve software delivery, but by allowing developers to focus on application development instead of infrastructure and maintenance.

By using machine learning and artificial intelligence, you can automate the setup, deployment, and monitoring processes, getting closer to NoOps.

Automation eliminates all bottlenecks

Automation is one of the biggest benefits that DevOps provides. But it's not a silver bullet that will solve all your problems.

A continuous delivery process enables teams to roll out new features quickly. And, get the feedback they need really fast. This, of course, means that you have to ensure the product's quality. Moreover, you have to take care of how well it runs and its performance when scaling. You also need to ensure smooth production deployments.

Automating your CI/CD pipelines helps eliminate the bottlenecks between code commit and deploy. But this is just one stage of the software delivery process. Unless developers and testers are in a partnership, you won't be able to resolve all your problems. It's likely that you'll only move any bottlenecks to another stream.

One-size-fits-all continuous delivery pipeline

The idea that you can have one process that fits all teams and companies is impossible, contrary to popular belief. Every organization has different needs and requirements. Even projects in the same organization need different continuous delivery pipelines.

You can have projects that need only two to three environments. For example, development, test, and production environments with frequent deployments. Another project can require more environments since it has multiple stages in the software delivery cycle.

This is why the continuous delivery pipeline should represent the release process that the company is already using.

DevOps is all about tools

Conversations about DevOps have mostly focused on which tools your company is using. They then turn into philosophical battles about what are the best tools. Instead, we should be communicating about the bigger picture, the business value DevOps brings to your company.

DevOps means focusing on culture, mindset, and how individuals work together. Only after should you be choosing the right tools for your processes. Teams often look in the large ecosystem of tools trying to find the perfect solution at the beginning. They build DevOps pipelines for a very long time, which should be redone once completed.

An Atlassian research showed that the two main factors to implement DevOps successfully are the right tools and the right people.

DevOps automation tools like Microtica allow you to create pipelines and test them in hours. With these kinds of tools, you can save months of work on pipelines that might not function.

Software release is the same as in Amazon/Facebook/Google

Many world-leading companies have adopted DevOps for its benefits and flexibility. Looking at these company's success stories, we, of course, look up to their achievements. We do this without realizing their context and the steps they made to become that successful.

One thing is for sure – these organizations chose and built the tools and processes that worked best for them at the time. This doesn't necessarily mean that we need to follow these organizations. Moreover, what they did won't magically work for our business as well.

We should learn from them and find new ways to innovate and grow. Explore and find the right processes and tools that define our problem space. What will bring success to our particular business? This is what DevOps is all about.

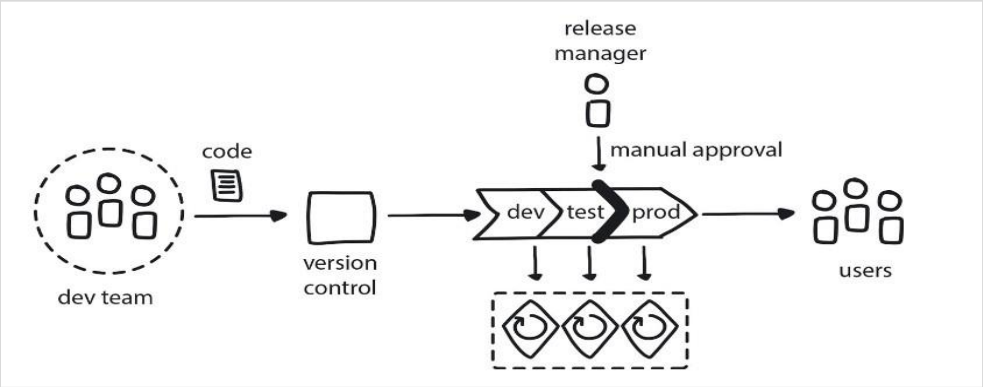
Release all the time

The idea of frequent releases has companies worried about not releasing their software continuously enough. "Ship often" has become the industry standard. However, this doesn't specify the time. It may be every two to three weeks, or it may be several times a day.

The most important thing is that you achieve team confidence that enables you to release new software when required.

CD is the ability to release code from the main branch and feel confident in it. The idea of DevOps is that your code should be releasable anytime.

So, continuous delivery doesn't mean you should release as often as you can but gives you the ability to release as often as you want. How often should be up to your company's decision.



DevOps Tools

1. Version Control Tool: Git (GitLab, GitHub, Bitbucket)

Git is perhaps the best and most widely used version control tool in a development era characterized by dynamism and collaboration. Version control provides developers with a means by which they can keep track of all the changes and updates in their codes such that in the event of a mishap, it is quite easy to return to and use the previous versions of the code and Git happens to be the best for many reasons.

Git DevOps tool is easy to implement as it is compatible with most protocols including HTTP, SSH, and FTP. It offers the best advantage for non-linear shared-repository development projects, unlike most other centralized version control tools. This makes it a good deal for mission-critical software.

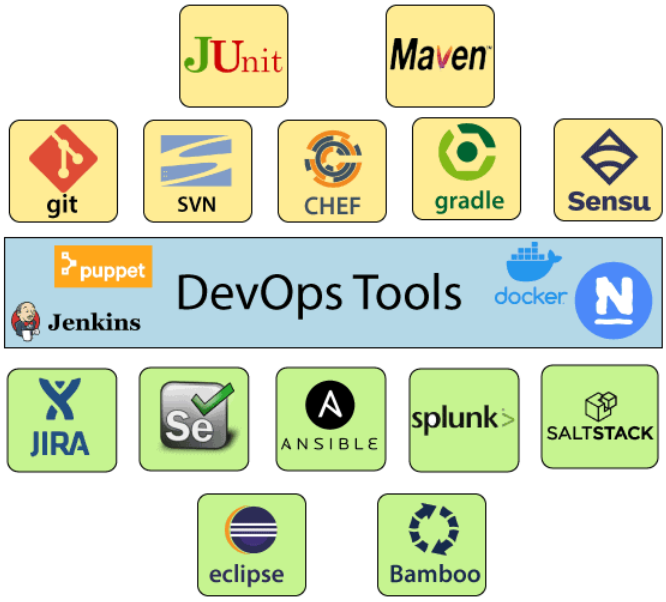
Git features three storage tools including, GitHub and GitLab cloud-hosted code repository services as well as BitBucket the source code hosting service. Of the three, GitLab and BitBucket are specifically designed for enterprise-range version control.

2. Build Tool: Maven

Maven is one of the important DevOps tools for building projects. Unlike the ANT build system, Apache Maven is more than just an automation build framework. It is also designed to manage reporting, documentation, distribution, releases, and dependencies processes. Written in Java language, Maven can build and manage projects written in Java or C#, Ruby, Scala, and other languages using project object model (POM) plugins.

Maven offers a host of benefits to its users. It eases the build and monitoring proce ss through automation and maintains a uniform build process allowing for consistency and efficiency. This tool also offers comprehensive project information through quality documentation, a valuable resource for the development of best practices hence the name Maven, translated from the Yiddish language to mean accumulator of knowledge. Finally, Maven provides a very simplified feature migration process.

It has a rich repository of plugins to enhance the build process and wide compatibility with IDEs like Eclipse, JBuilder, MyEclipse, NetBeans, IntelliJ IDEA, and others.



3. Continuous Integration Tool: Jenkins

Jenkins is an integration DevOps tool. For continuous integration (CI), Jenkins stands out as it is designed for both internal and plugin extensions. Jenkins is an open-source Java-based automation CI server that is supported by multiple operating systems including Windows, macOS, and other Unix OSs. Jenkins can also be deployed on cloud-based platforms.

Continuous Integration and Continuous Delivery are two core practices of the DevOps methodology which makes Jenkins an indispensable DevOps tool. Jenkins is compatible with most CI/CD integration tools and services thanks to the over 1,500 plugins available to provide integration points for delivering customized functionality during software development.

A valuable automation CI tool, Jenkins is pretty easy to install and configure. It is designed to support distributed workflows for accelerated and transparent builds, tests, and deployments across platforms.

4. Configuration Management Tool: Chef

Configuration management (CM) refers to the maintenance and control of the components of large complex systems in a known, consistent, and determined state throughout the DevOps life cycle. Components of an IT system may include servers, networks, storage, and applications.

For this reason, configuration management is critical to any system as it is the process by which changes in the system are tracked, properly implemented, and controlled. Further, if not automated, CM can be laborious, resource-draining, and prone to costly errors. It implements configuration tools for such repetitive administrative tasks as version management, regulatory compliance, feature releases, and processes automation, among others.

Chef, Puppet, and Ansible are handy CM automation frameworks. While Chef and Puppet are Ruby-based frameworks, Ansible is a Python-based framework.

Chef, an open-source framework, uses a master-agent model and has infrastructure as code (IAC) capabilities to automate the configuration of infrastructure. Together with its multi-platform support that includes the cloud platform, Chef remains one of the most popular DevOps tools after Puppet.

5. Configuration Management Tool: Puppet

Puppet is also open-source and uses declarative programming for system configuration, deployments, and server management DevOps tools. It is organized into reusable modules for the speedy setup of pre-configured servers and is compatible with most platforms. Like Chef, it also uses IAC, adopts a master-slave architecture, and features an intuitive user interface for ease of real-time reporting, node management, and several other tasks.

6. Configuration Management Tool: Ansible

Ansible is an open-source CM DevOps tool that is also used for deployment, automation, and orchestration. While Ansible leverages infrastructure as a code architecture, it uses SSH connection for its push nodes thus agentless. Of the three, Ansible is considered easy to learn and use as its Playbooks are written in YAML with minimal commands and are

readable by humans.

7. Container Platforms: Docker

Container platforms are application solutions that allow developers to build, test, and ship applications in resource-independent environments. Each container comprises a complete runtime environment including the specific application, its libraries, source code, configurations, and all its dependencies. Container platforms offer orchestration, automation, security, governance, and other capabilities.

DevOps heavily relies on containerization and microservices for efficient application development and deployment with Docker and Kubernetes as the most widely used container technologies.

Docker

The Docker engine is designed to automate the development, deployment, and management of containerized applications on single nodes. Docker is open-source and compatible with cloud services like AWS, GCP, and Azure Cloud. Docker also runs on Windows and Linux operating systems.

8. Container Platforms: Kubernetes

Kubernetes, on the other hand, is an automation orchestration platform that enables developers to run containerized applications across Kubernetes clusters referring to a group of nodes. Developers harness Kubernetes to automate such processes as container configuration, scaling, networking, security, and more to achieve speed and efficiency in production.

9. Communication and Collaboration: Slack

Workplace communication and collaboration technologies are as numerous and as diverse as can be imagined. And when it comes to deciding which tools best suit specific business requirements, several factors go into consideration such as integration and automation capabilities, security, user experience, as well as whether to develop, buy or rent.

One of the most popular communication and collaboration tools and for all the good reasons is Slack. First things first, Slack offers free, standard and enterprise paid versions to cater to a wide range of clients with varying needs. Slack is a standalone tool that flaunts:

- Powerful search capabilities with well-designed search modifiers to ease document tracking, management, and file sharing.
- A friendly project management architecture integrates with project management tools like Twitter, Google

Hangouts, Trello, and more.

- Powerful collaboration and communication capabilities via shared channels, direct chat, voice, and video conferencing.
- Added features like workflow builder, notification, and note-taking features.

Slack is a simplistic application with an intuitive user interface and a host of pre-built integration points that make it a brilliant solution for supplementing more than 900 other business tools. It is operable from a web browser, synchronizable with your desktop, and usable on mobile devices like tablets and smartphones. Slack also offers extensive storage space and a wide range of integrations (paid versions).

Use Cases

1. AMAZON

Problem

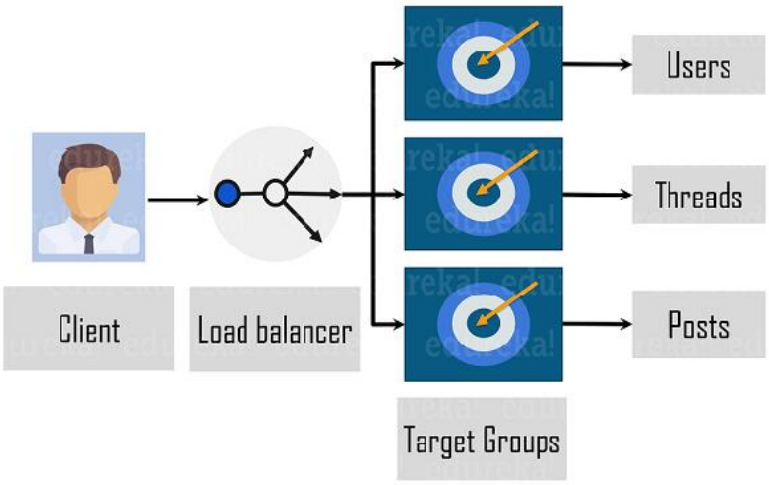
You must be aware that Amazon is one of the biggest e-commerce companies in the world. But back in 2001, their website followed a traditional monolithic architecture. Here, all processes were coupled together and run as a single service

Over time as the source files grew, it became hard to scale, maintain, and upgrade their applications on physical servers

Solution

To solve the problems of the monolithic architecture, Amazon moved from physical servers to cloud-based Amazon Web Services (AWS).

Currently, AWS follows a microservice architecture as shown in the figure below. Within this architecture, the client initially makes a request. Here a load balancer inspects the client request and assigns it to the correct microservice. In turn, the microservice has a target group that keeps track of the instances and ports. In Amazon, they are three types of microservices namely, Users, Threads, and Posts.



Features and tools

Now that we know how a microservice works, let us look at some of the tools and features Amazon implement to adopt DevOps practices.

Developers apply frequent but small changes over their code via version control tools like Git and GitHub. Practices like code deployment help fix bugs and add new features to improve the underlying software application. AWS CodeDeploy is one such service that keeps track of deployments and simplifies the software release process.

Amazon also uses Apollo, a simple one-click internal deployment tool. Apollo's job is to deploy a specified set of software across a group of hosts. It also provides versioned artifacts and test rollbacks.

On the other hand, practices like Configuration management and infrastructure-as-code help to monitor and make changes in the software. It keeps track of the system's performance and resources used by developers. Th is way, the testing team can identify problems before in hand and fix them immediately.

2. Netflix

Problem

- Just like any other organization, Netflix too adopted monolithic architecture
- To handle a huge amount of scale and traffic caused by their subscribers, Netflix was in short of commercial tools

Solution

They began to move from a monolithic to AWS cloud-based microservice architecture.

Netflix uses around 700 microservices to control each part of the full service.

The microservice architecture separates engineering teams from each other & lets them build, test, and deploy their services. This flexibility enables them to increase their pace.

Preparing for failure

Preparation is the best way to handle unexpected failures. Netflix built a tool called "Chaos Monkey" which helps in testing the stability of its application. So, Chaos Monkey enforces failures on purpose by terminating servers randomly while the developers are working on it.

It is a part of the "Netflix Simian Army" the organization's effort to find solutions for their unexpected problems.

This practice is DevOps at its best because making changes in an unfavorable environment during the development process results in the top class and highly resilient applications.

Containerization

Netflix has also developed a container management tool known as Titus. It runs existing applications without making any changes in the container, therefore, eliminating scaling issues. It handles resource sharing capacity and integrates with Amazon Web Services. Titus helps Netflix with streaming, recommendation, and content systems.

3. ADOBE

Adobe Creative cloud consists of a set of services that gives users the ability to work on different software applications. You must be aware of some of them they are Photoshop, Lightroom, Illustrator, etc.

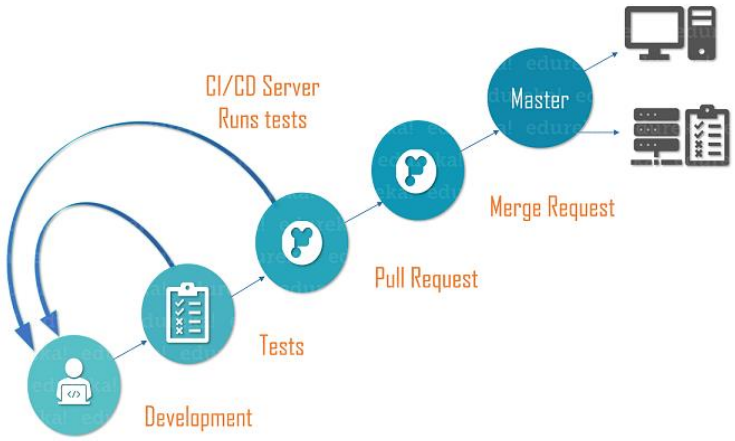
Problem

- Initially, communication between services in Adobe was point-to-point as they employed the monolithic architecture
- But as load grew rapidly on these applications, integration became an impossible task

Solution

In this DevOps transformation journey, the key tools and practices Adobe employed are Microservices, containers, and CI/CD.

The 'Adobe Experience Platform Pipeline' as shown in the figure below is a distributed, Apache Kafka based message bus for communication across Adobe solutions. Its goal was to break Adobe's internal silos and simplify communication between services by reducing the number of manual steps.



The messages received by the pipeline are replicated across 13 data centers in AWS, Azure, and Adobe data centers.

Tools

You must know by now that automation in deployment helps save time. Here I will discuss some of the automation tools used in the pipeline:

- **Git** is the version control tool to make code configurations
- **Docker** for the containerization process
- **Jenkins** is a continuous integration tool
- **Kubernetes** and **Spinnaker** for container deployment and a multi-cloud continuous delivery platform

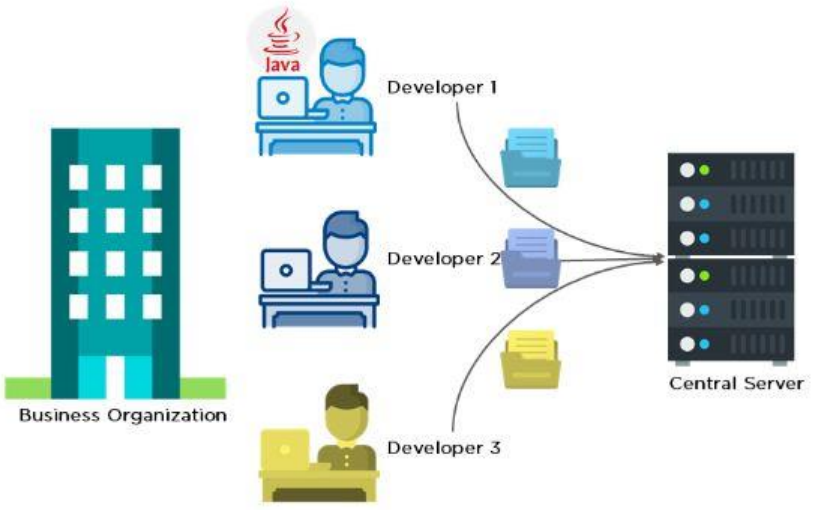
Adobe Experience Platform is the best to decrease resource costs, improve infrastructure management, and make service movements across different clouds and platforms easy.

Introduction GitHub

Git is a DevOps tool used for source code management. It is a free and open-source version control system used to handle small to very large projects efficiently. Git is used to tracking changes in the source code, enabling multiple developers to work together on non-linear development. Linus Torvalds created Git in 2005 for the development of the Linux kernel.

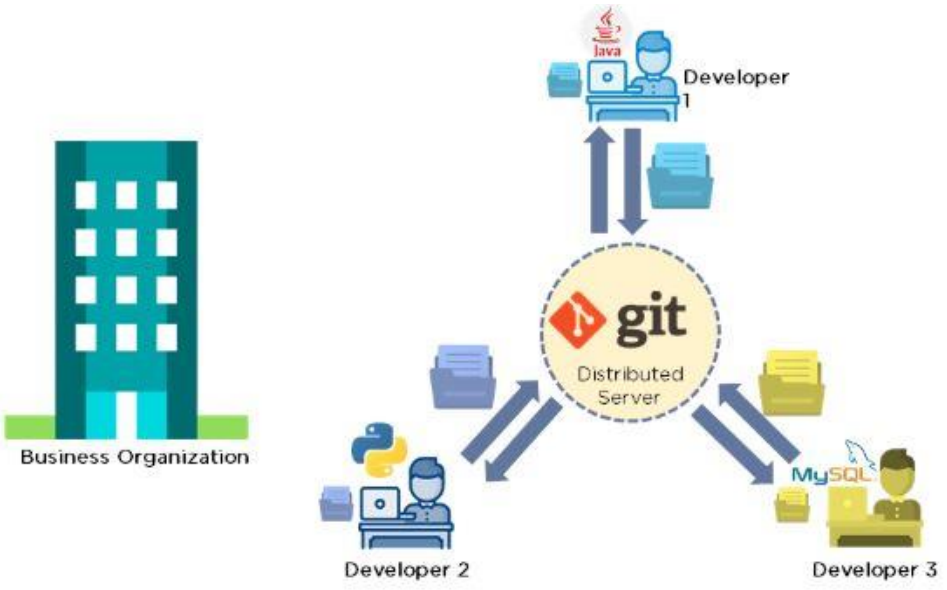
Before diving deep, let's explain a scenario before Git:

- Developers used to submit their codes to the central server without having copies of their own
- Any changes made to the source code were unknown to the other developers
- There was no communication between any of the developers



Now let's look at the scenario after Git:

- Every developer has an entire copy of the code on their local systems
- Any changes made to the source code can be tracked by others
- There is regular communication between the developers



A GitHub Use Case

Let's consider the case of Decathlon, the world's largest sporting goods retail brand. The company has over 1600 stores in 57 countries, with more than 87,000 employees.

Every company, no matter how large or small, inevitably experiences challenges and obstacles. We could best summarize Decathlon's problems as:

1. How would the company maintain workflow visibility and avoid redundancies in such a large workforce?
2. How would the company hire developers for so many diverse locations?

GitHub to the rescue! GitHub is not only an affordable resource but also features a great open-source community. Since it is a cloud-based tool, the code is conveniently visible across the entire client organization, facilitating every participant's contribution.

GitHub allows collaboration with developers from all over the world. Open-source solutions like GitHub enable potential developers to contribute and share their knowledge to benefit the global community.

The version control system, or VCS, is the element in Git that is best suited for tackling Decathlon's two problems. So, let's expand our knowledge of GitHub by taking a closer look at the Git version control system and see why it's such a game-changer.

What is a Version Control System?

The Git version control system, as the name suggests, is a system that records all the modifications made to a file or set of data so that a specific version may be called up later if needed. The system makes sure that all the team members are working on the file's latest version, and everyone can work simultaneously on the same project.

Before we dig deeper into what GitHub is, we must examine first what the 'Git' part is all about.

What is Git?

Git is a version control system used for tracking changes in computer files. It is generally used for source code management in software development.

- Git is used to tracking changes in the source code
- The distributed version control tool is used for source code management
- It allows multiple developers to work together
- It supports non-linear development through its thousands of parallel branches

What Is GitHub?

GitHub is a Git repository hosting service that provides a web-based graphical interface (GUI). It helps every team member work together on a project from anywhere, making it easy to collaborate.

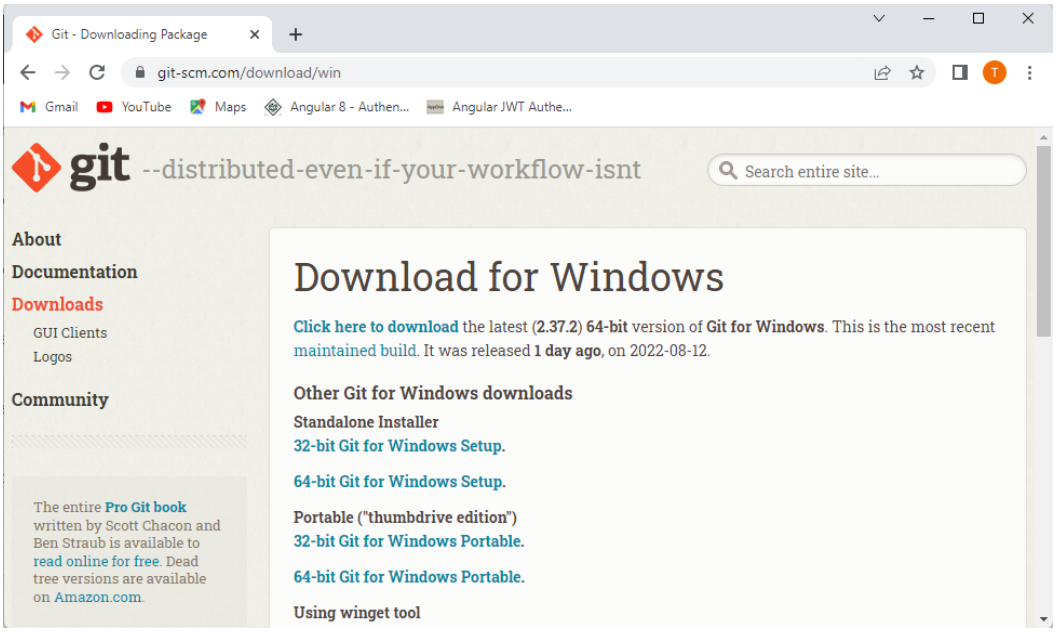
GitHub is one place where project managers and developers coordinate, track, and update their work, so projects stay transparent and on schedule. The packages can be published privately, within the team, or publicly for the open-source community. Downloading packages from GitHub enables them to be used and reused. GitHub helps all team members stay on the same page and stay organized. Moderation tools, like issue and pull request locking, helps the team focus on the code.

What does Git do?

- Manage projects with Repositories
- Clone a project to work on a local copy
- Control and track changes with Staging and Committing
- Branch and Merge to allow for work on different parts and versions of a project
- Pull the latest version of the project to a local copy
- Push local updates to the main project

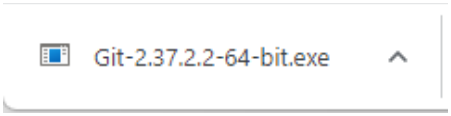
Download Git for Windows

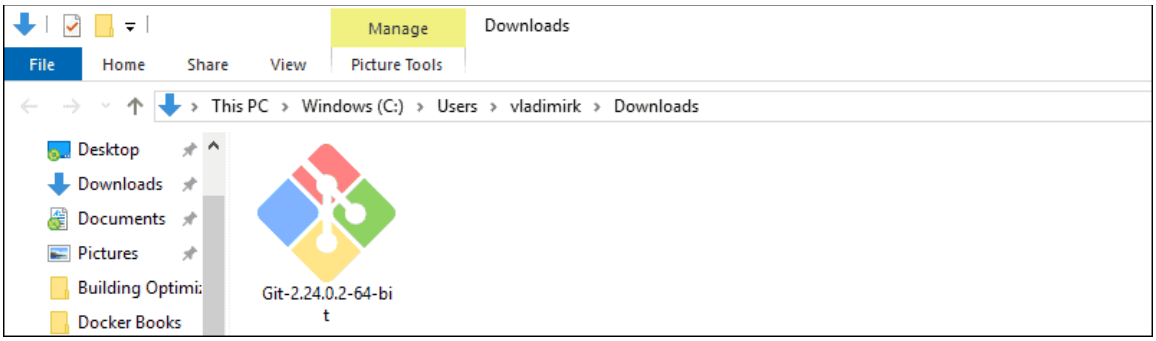
1. Browse to the official Git website: https://git-scm.com/downloads
2. Click the download link for Windows and allow the download to complete. Download the file depending on your system's type 32 or 64 bit



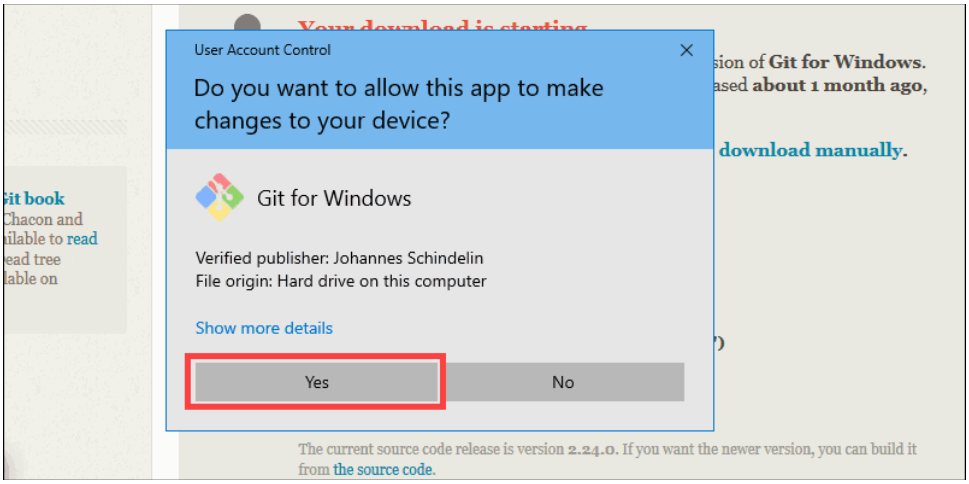
Extract and Launch Git Installer

4. Browse to the download location (or use the download shortcut in your browser). Double -click the file to extract and launch the installer.

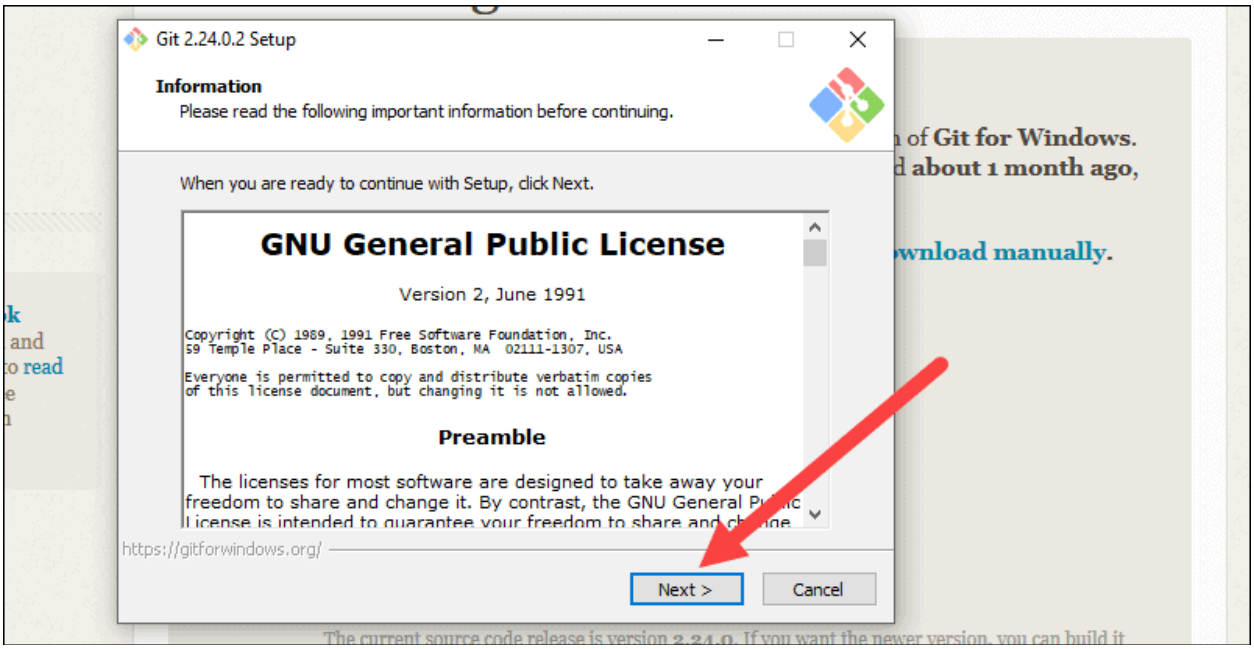




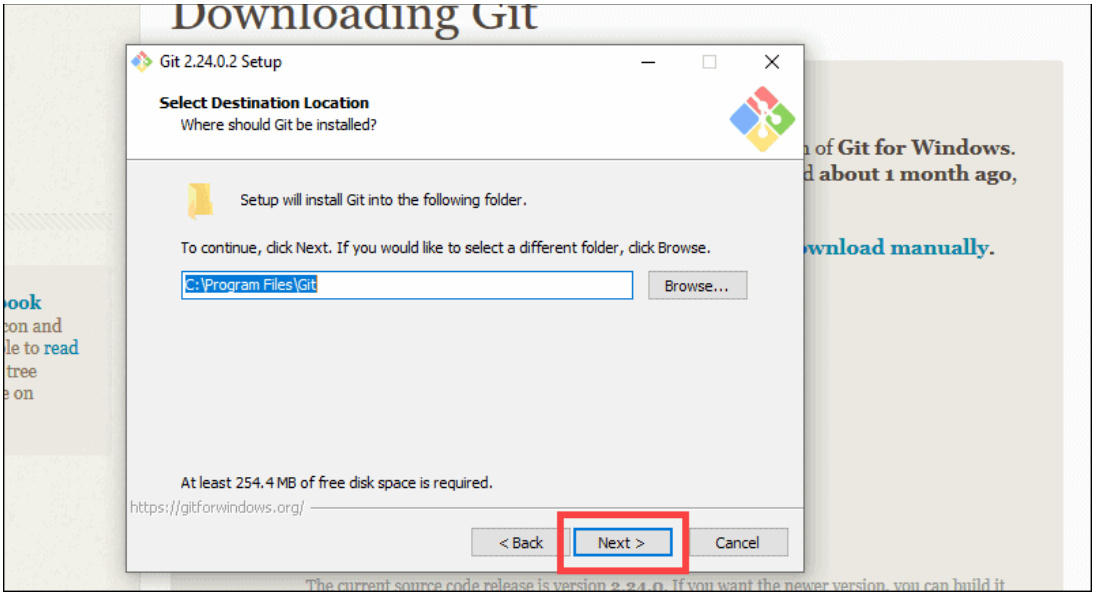
5. Allow the app to make changes to your device by clicking **Yes** on the User Account Control dialog that opens.



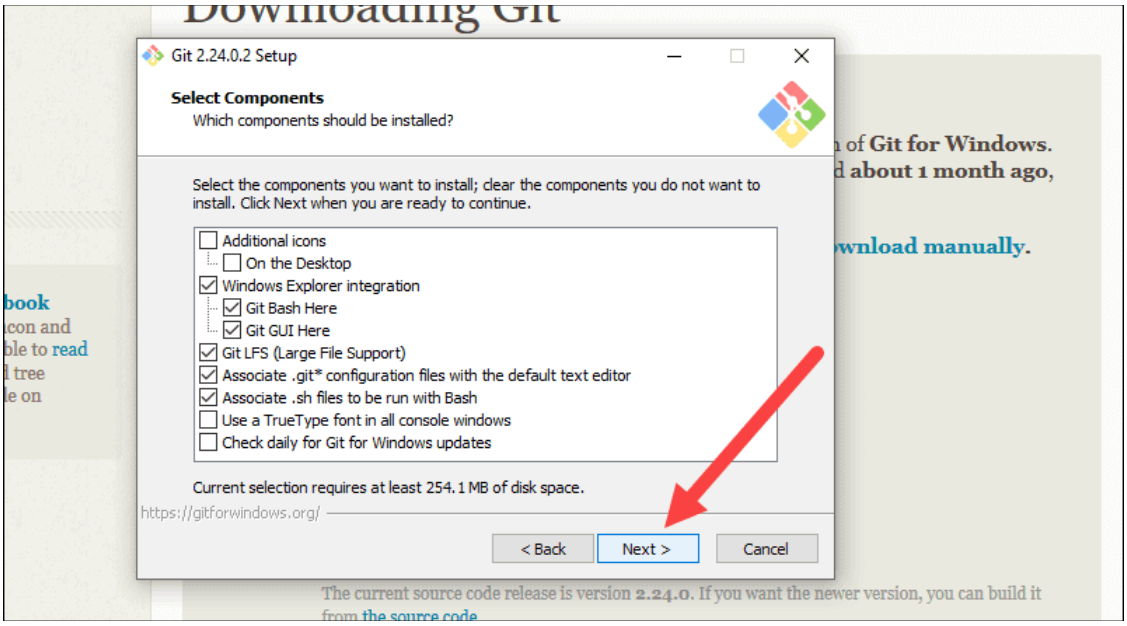
6. Review the GNU General Public License, and when you're ready to install, click **Next**.



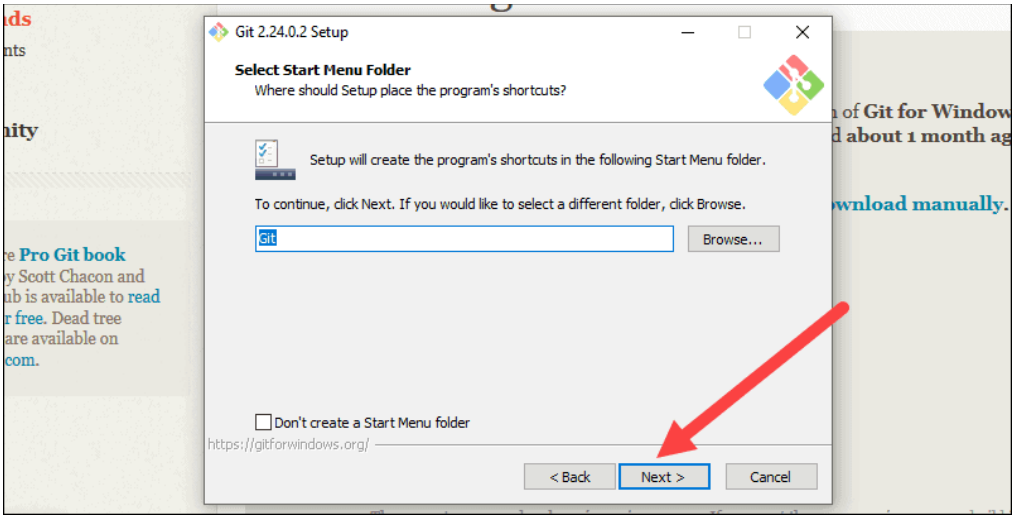
7. The installer will ask you for an installation location. Leave the default, unless you have reason to change it, and click **Next**.



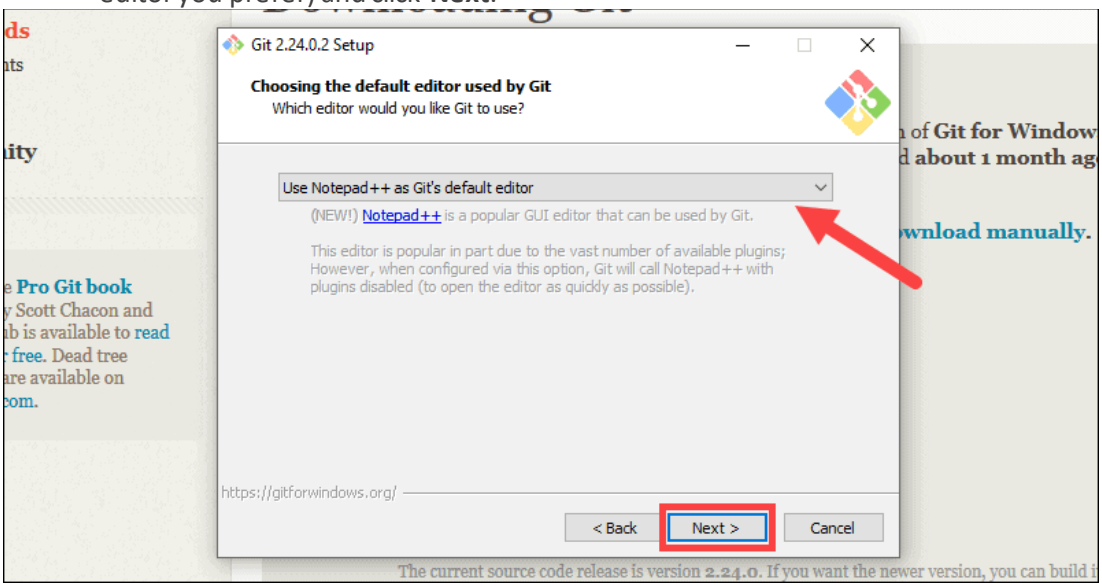
8. A component selection screen will appear. Leave the defaults unless you have a specific need to change them and click **Next**.



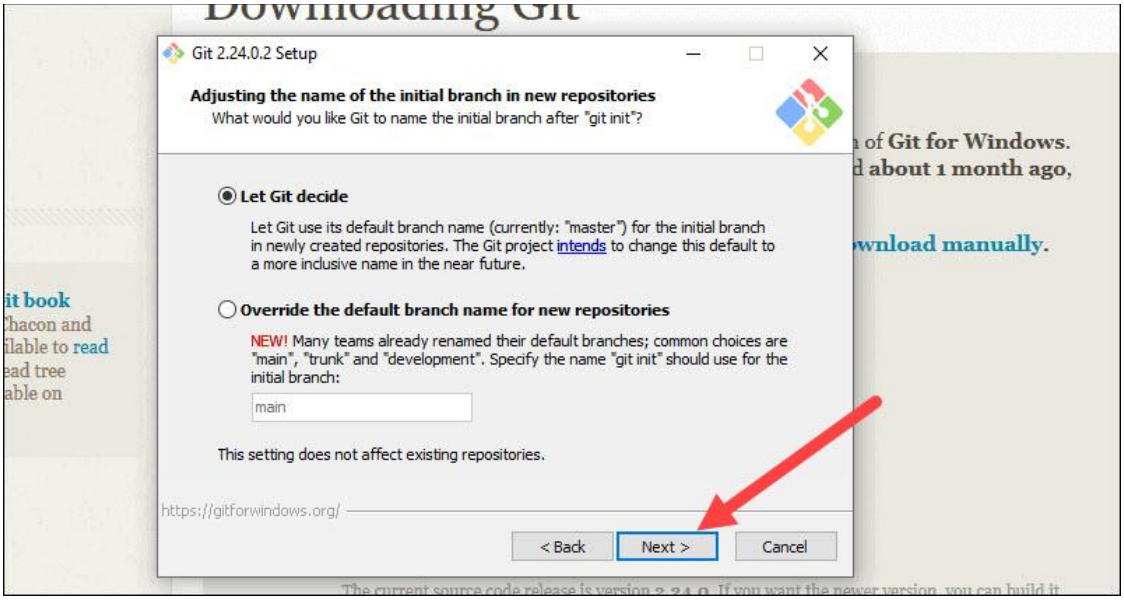
9. The installer will offer to create a start menu folder. Simply click **Next**.



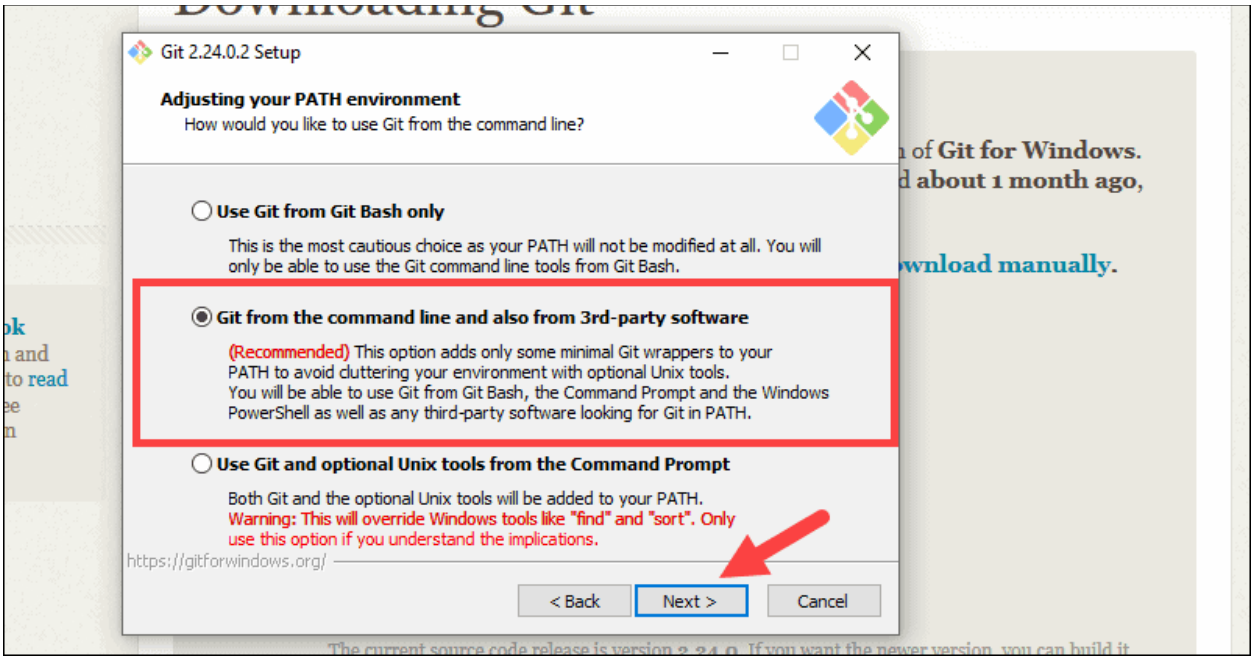
10. Select a text editor you'd like to use with Git. Use the drop-down menu to select Notepad++ (or whichever text editor you prefer) and click **Next**.



11. The next step allows you to choose a different name for your initial branch. The default is 'master.' Unless you're working in a team that requires a different name, leave the default option and click **Next.**

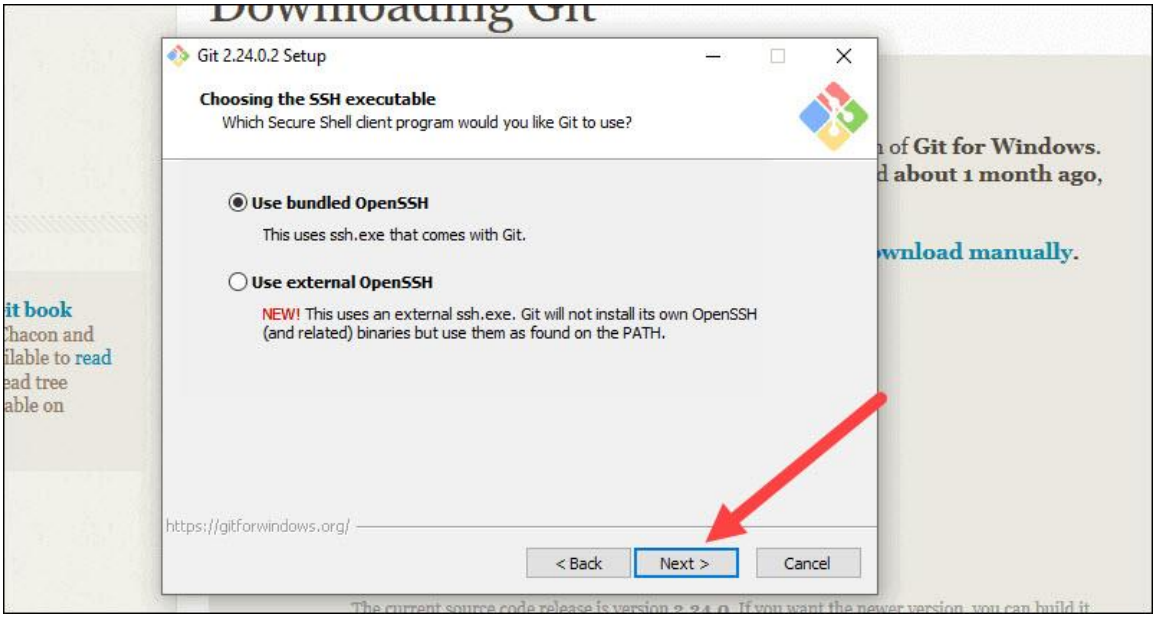


12. This installation step allows you to change the **PATH environment**. The **PATH** is the default set of directories included when you run a command from the command line. Leave this on the middle (recommended) selection and click **Next**.

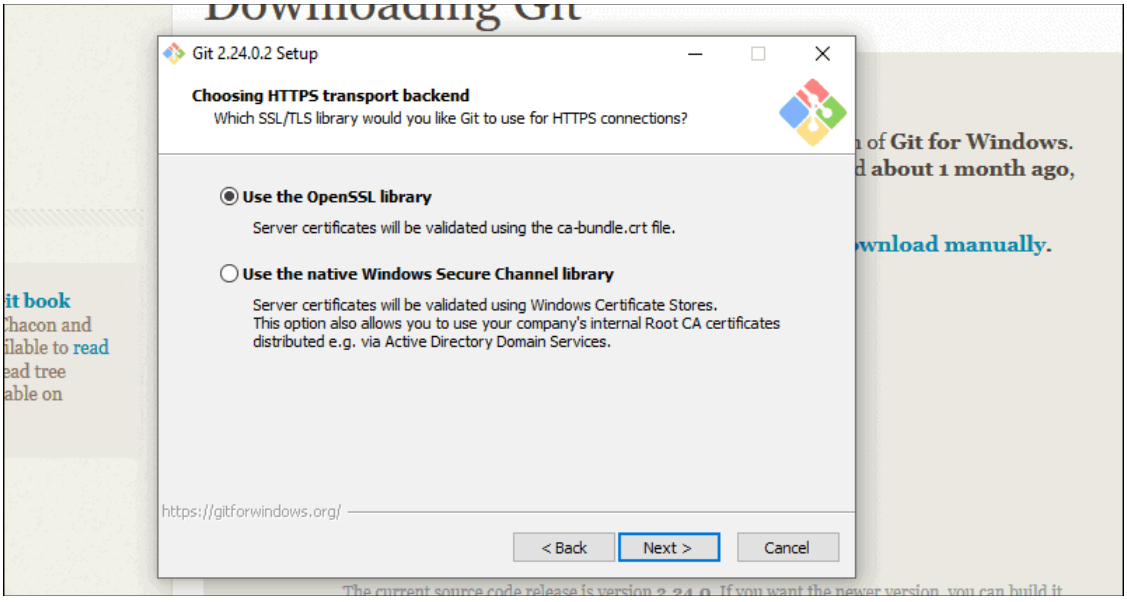


Server Certificates, Line Endings and Terminal Emulators

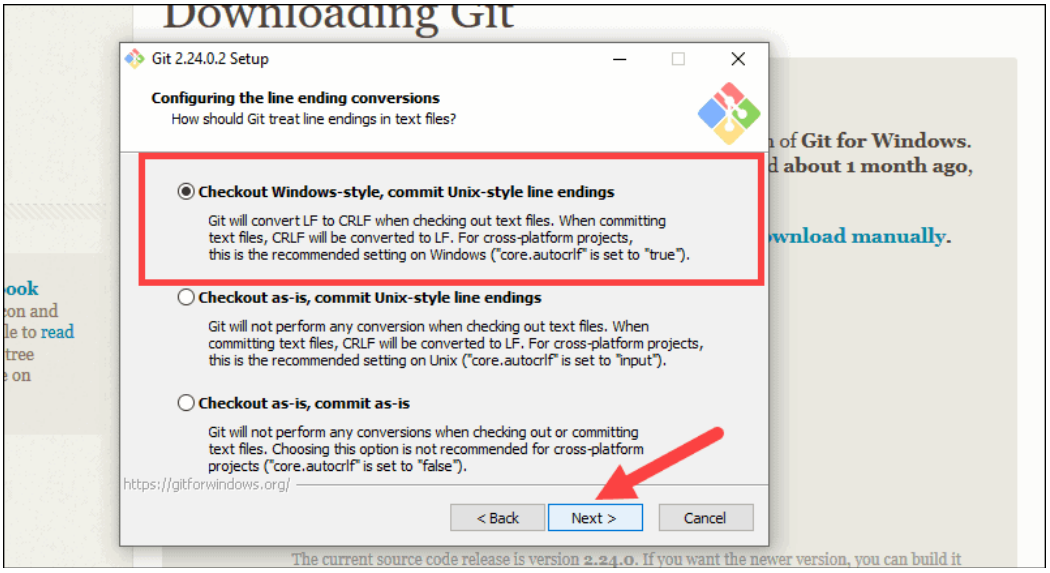
13. The installer now asks which SSH client you want Git to use. Git already comes with its own SSH client, so if you don't need a specific one, leave the default option and click **Next.**



14. The next option relates to server certificates. Most users should use the default. If you're working in an Active Directory environment, you may need to switch to Windows Store certificates. Click **Next**.



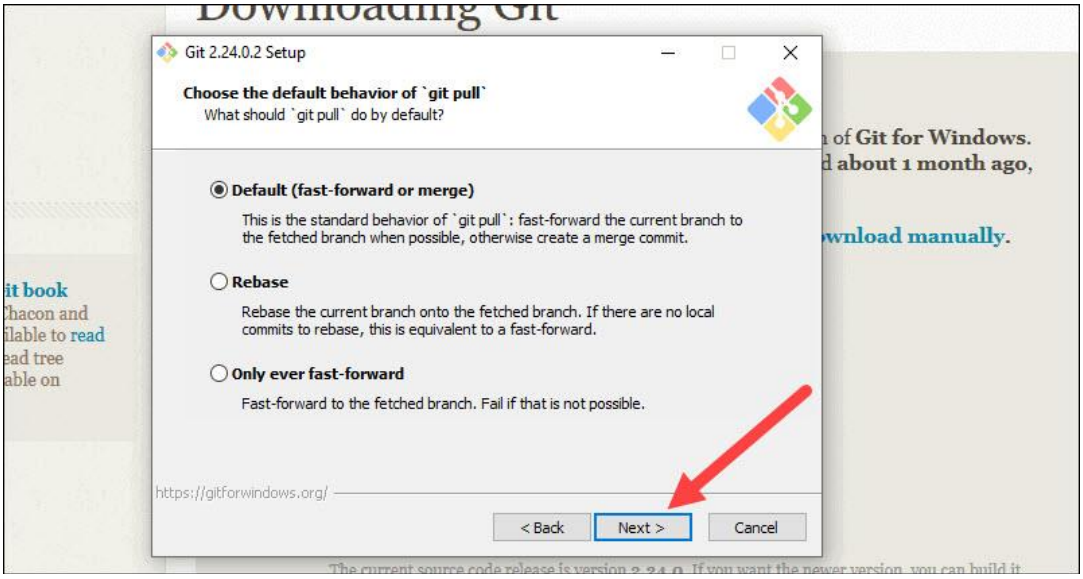
15. The next selection converts line endings. It is recommended that you leave the default selection. This relates to the way data is formatted and changing this option may cause problems. Click **Next**.



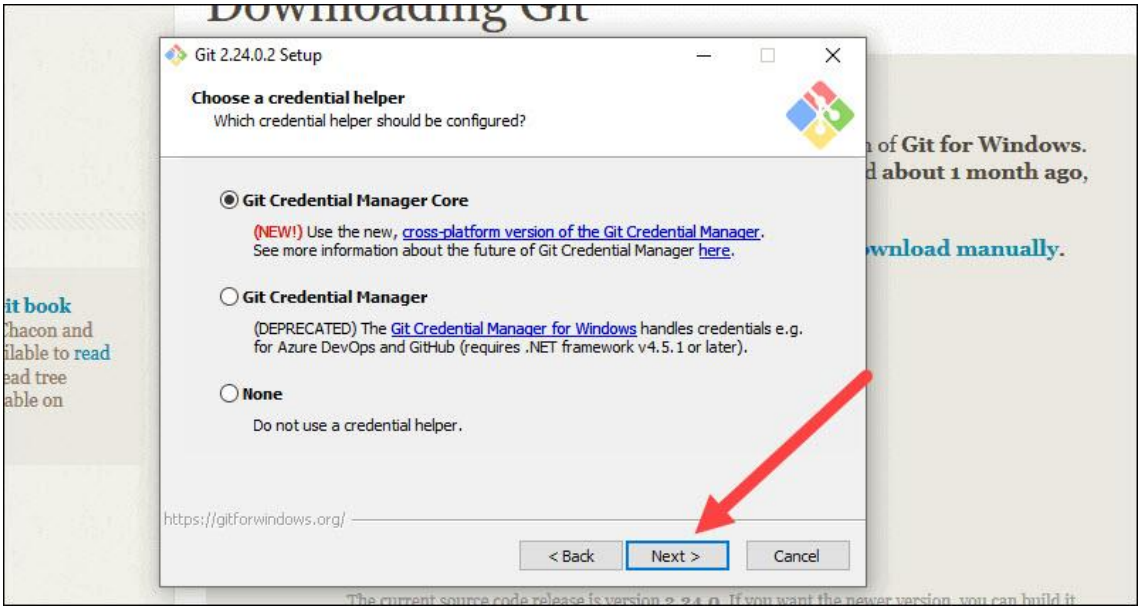
16. Choose the terminal emulator you want to use. The default MinTTY is recommended, for its features. Click **Next**.

17. The installer now asks what the **git pull** command should do. The default option is recommended unless you

specifically need to change its behavior. Click **Next** to continue with the installation.

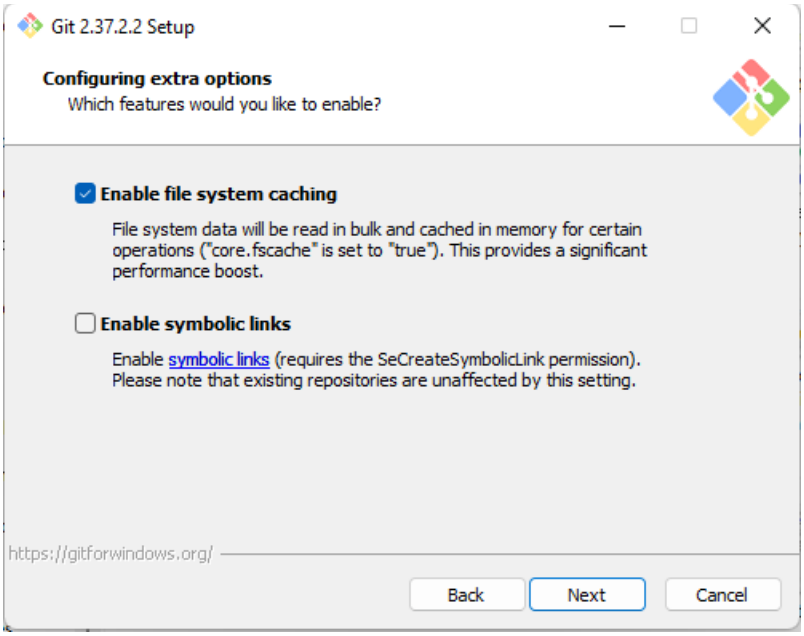


18. Next you should choose which credential helper to use. Git uses credential helpers to fetch or save credentials. Leave the default option as it is the most stable one, and click **Next**.

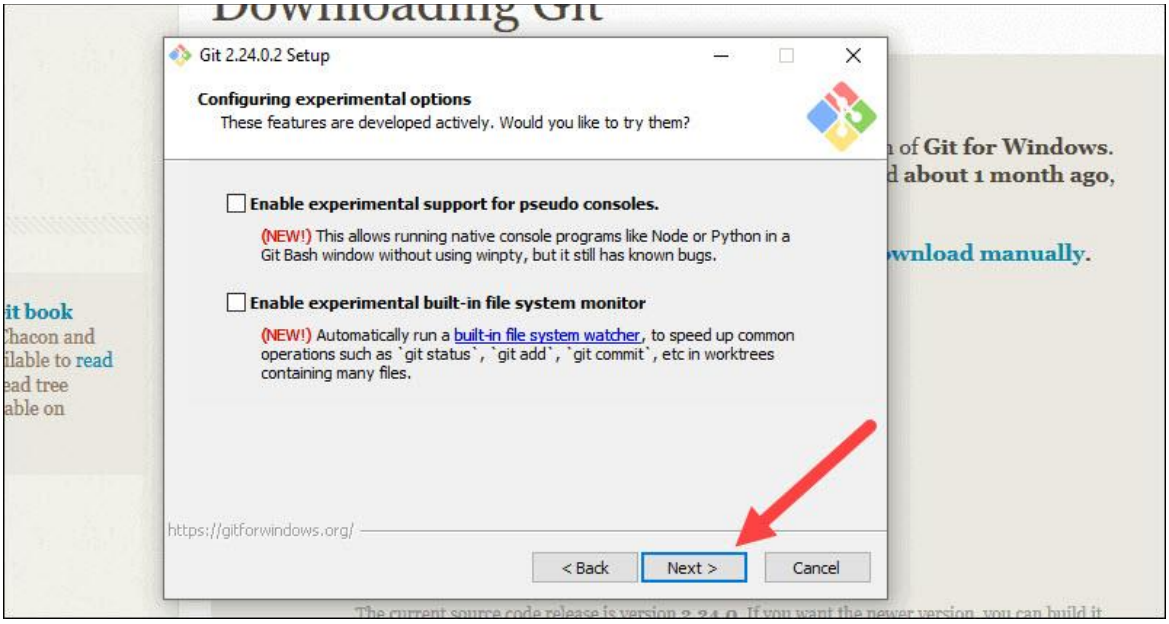


Additional Customization Options

19. The default options are recommended; however, this step allows you to decide which extra option you would like to enable. If you use symbolic links, which are like shortcuts for the command line, tick the box. Click **Next**.

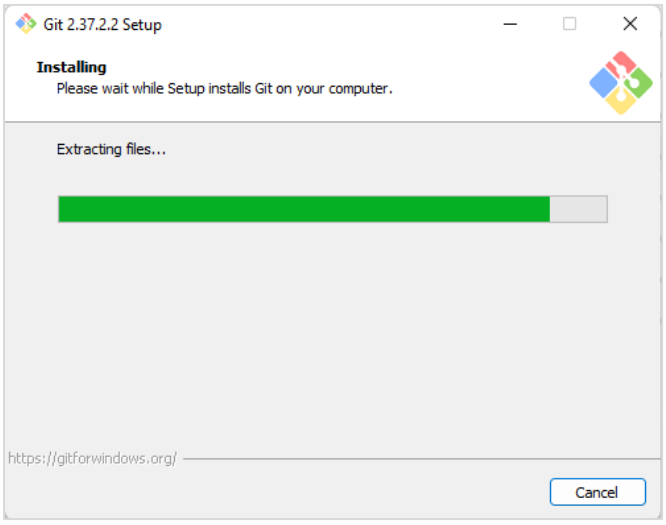


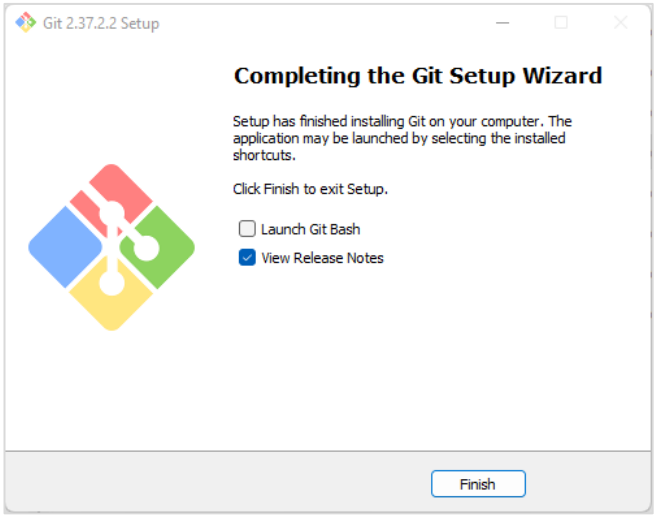
20. Depending on the version of Git you're installing; it may offer to install experimental features. At the time this article was written, the options to include support for pseudo controls and a built-in file system monitor were offered. Unless you are feeling adventurous, leave them unchecked and click **Install**.



Complete Git Installation Process

21. Once the installation is complete, tick the boxes to view the Release Notes or Launch Git Bash, then click **Finish**.



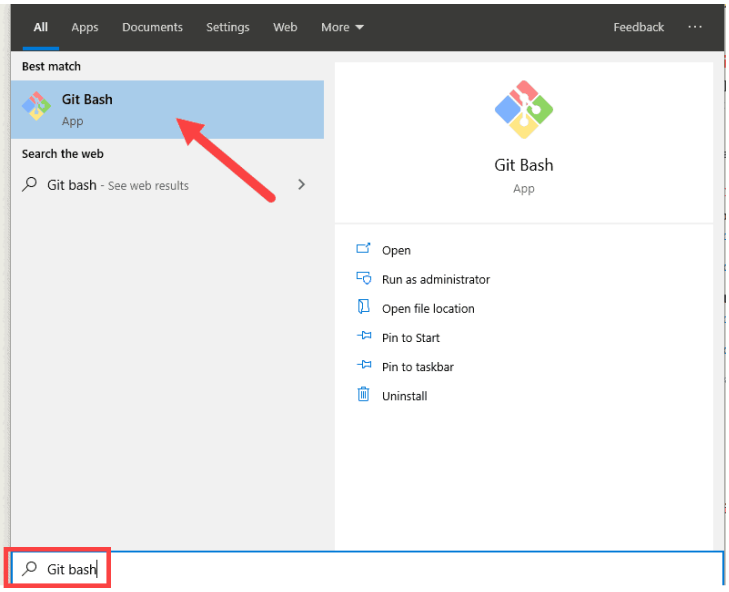


How to Launch Git in Windows

Git has two modes of use – a bash scripting shell (or command line) and a graphical user interface (GUI).

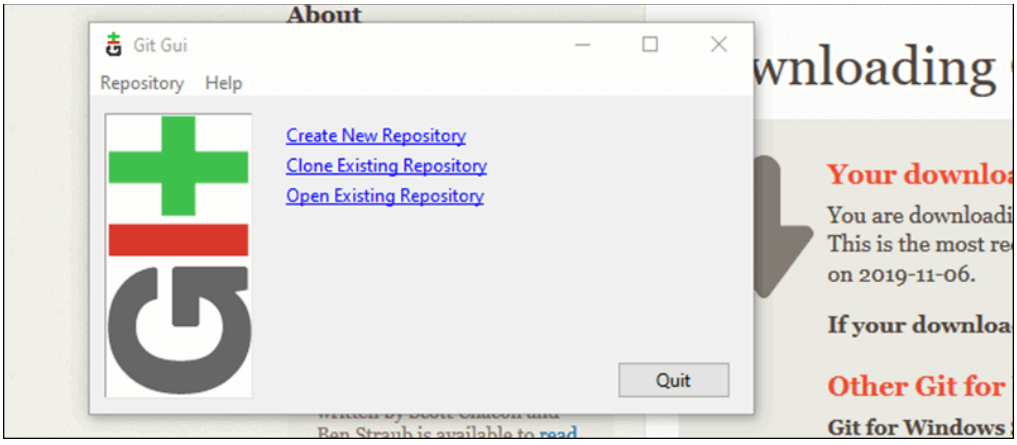
Launch Git Bash Shell

To launch Git Bash, open the Windows Start menu, type git bash and press Enter (or click the application icon).



Launch Git GUI

To launch **Git GUI** open the **Windows Start** menu, type git GUI, and press Enter (or click the application icon).



Connecting to a Remote Repository

You need a GitHub username and password for this next step.

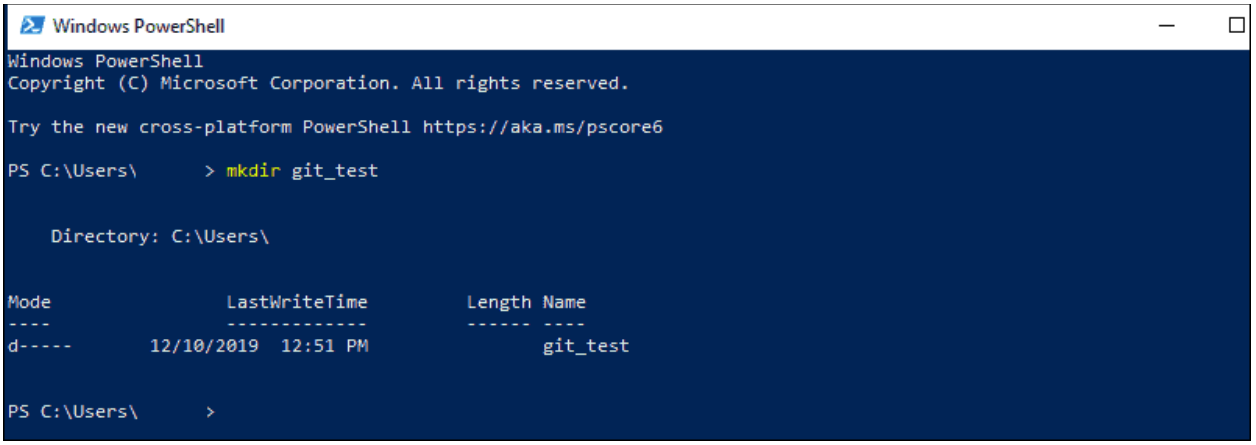
Create a Test Directory

Open a Windows PowerShell interface by pressing **Windows Key + x**, and then **I** once the menu appears.

Create a new test directory (folder) by entering the following:

```
Mkdir git test
```

An example of the PowerShell output.



Change your location to the newly created directory:

```
cd git_test
```

Note: If you already have a GitHub repository, use the name of that project instead of **git_test**.

Configure GitHub Credentials

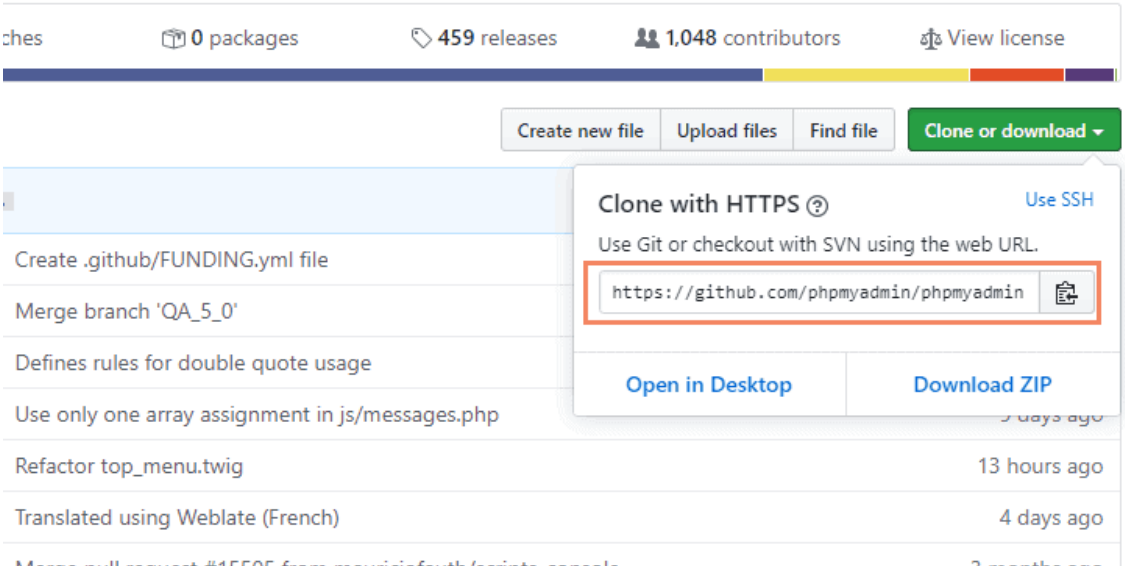
Configure your local Git installation to use your GitHub credentials by entering the following:

```
git config --global user.name "github_username"
git config --global user.email "email_address"
```

Note: Replace **github_username** and **email_address** with your GitHub credentials.

Clone a GitHub Repository

Go to your repository on GitHub. In the top right above the list of files, open the **Clone or download** drop-down menu. Copy the **URL for cloning over HTTPS**.



Switch to your PowerShell window, and enter the following:

```
git clone repository_url
```

Important: In the example above, the command will clone the repository over HTTPS. Another option is **cloning with SSH URLs**. For that option to work, you must generate an SSH key pair on your Windows workstation and assign the public key to your GitHub account.

List Remote Repositories

Your working directory should now have a copy of the repository from GitHub. It should contain a directory with the name of the project. Change to the directory:

```
cd git_project
```

Note: Replace **git_project** with the actual name of the repository you downloaded. If it's not working, you can list the contents of the current directory with the **ls command**. This is helpful if you don't know the exact name or need to check your spelling.

Once you're in the sub-directory, list the remote repositories:

```
git remote -v
```

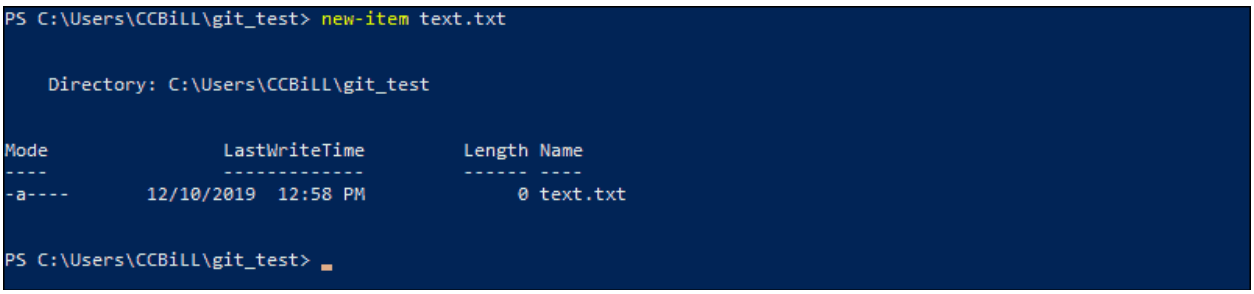
Pushing Local Files to the Remote Repository

Once you've done some work on the project, you may want to submit those changes to the remote project on GitHub.

1. For example, create a new text file by entering the following into your PowerShell window:

```
new-item text.txt
```

2. Confirmation that the new file is created.



3. Now check the status of your new Git branch and untracked files:

```
git status
```

4. Add your new file to the local project:

```
git add text.txt
```

5. Run **git status** again to make sure the text.txt file has been added. Next, commit the changes to the local project:

```
git commit -m "Sample 1"
```

6. Finally, push the changes to the remote GitHub repository:

```
git push
```

You may need to enter your username and password for GitHub.

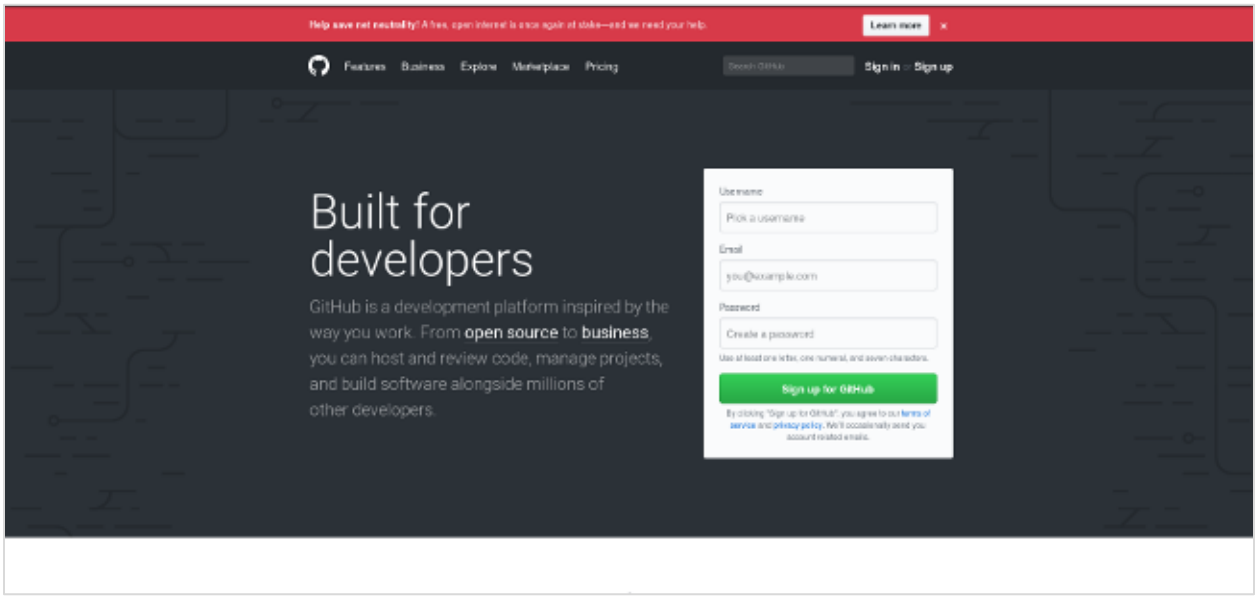
Note: You can remove a remote repository if the need for it no longer exists.

Creating first repository

Creating first repository

Step 1: Create a GitHub account

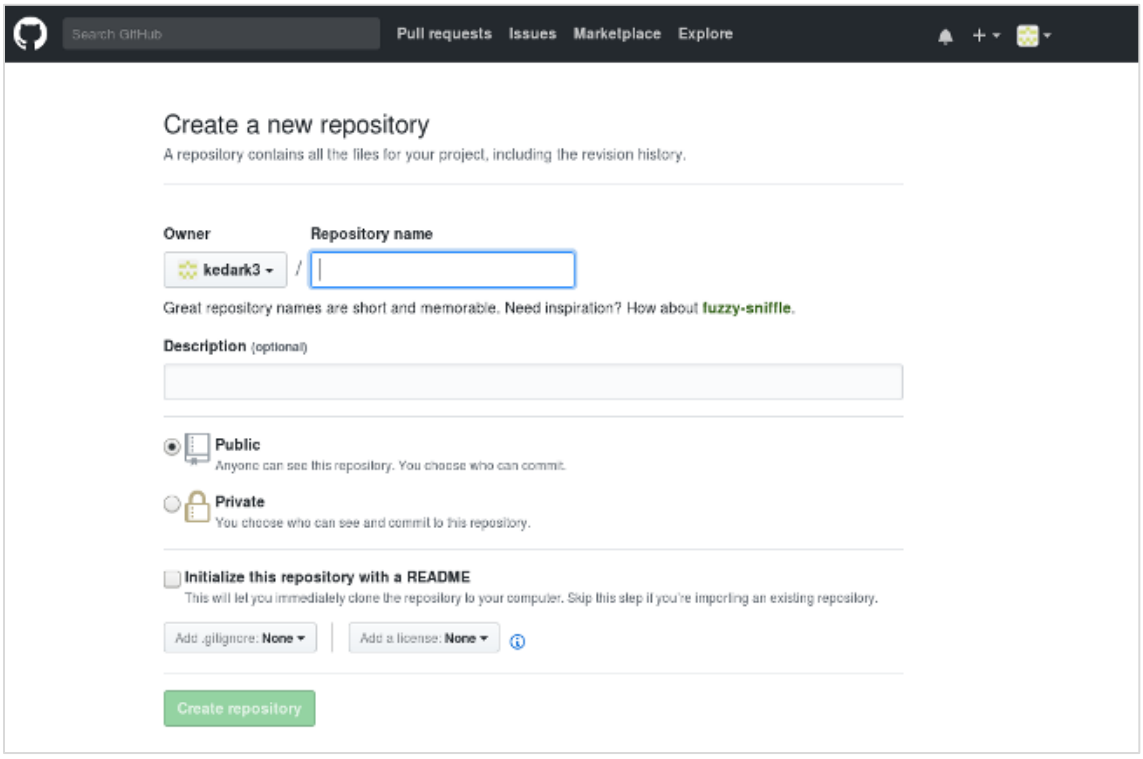
The easiest way to get started is to create an account on GitHub.com (it's free).



Pick a username (e.g., octocat123), enter your email address and a password, and click Sign up for GitHub. Once you are in, it will look something like this:

Step 2: Create a new repository

A repository is like a place or a container where something is stored; in this case we're creating a Git repository to store code. To create a new repository, select New Repository from the + sign dropdown menu (you can see I've selected it in the upper-right corner in the image above).

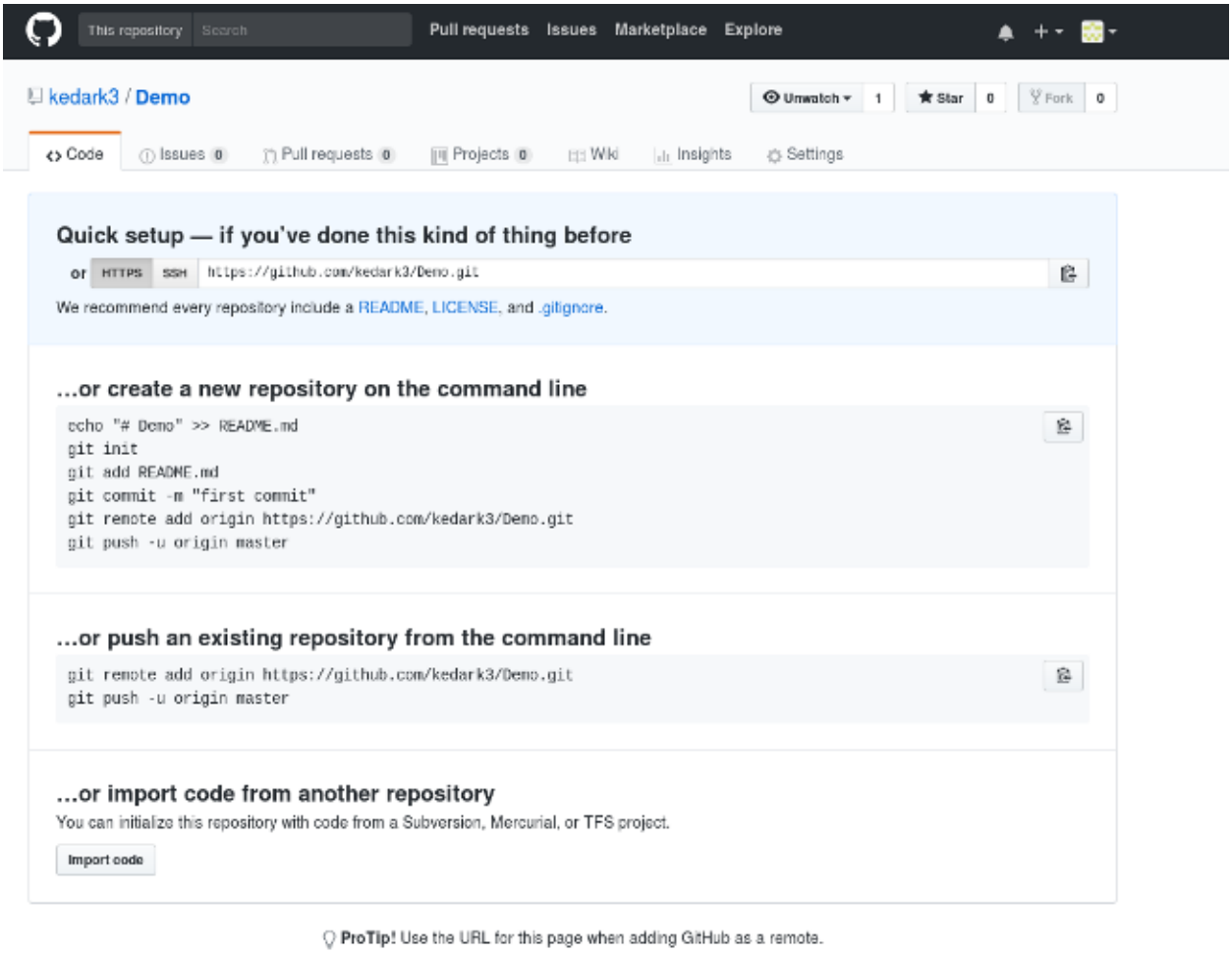


Enter a name for your repository (e.g, "Demo") and click Create Repository. Don't worry about changing any other options on this page.

Congratulations! You have set up your first repo on GitHub.com.

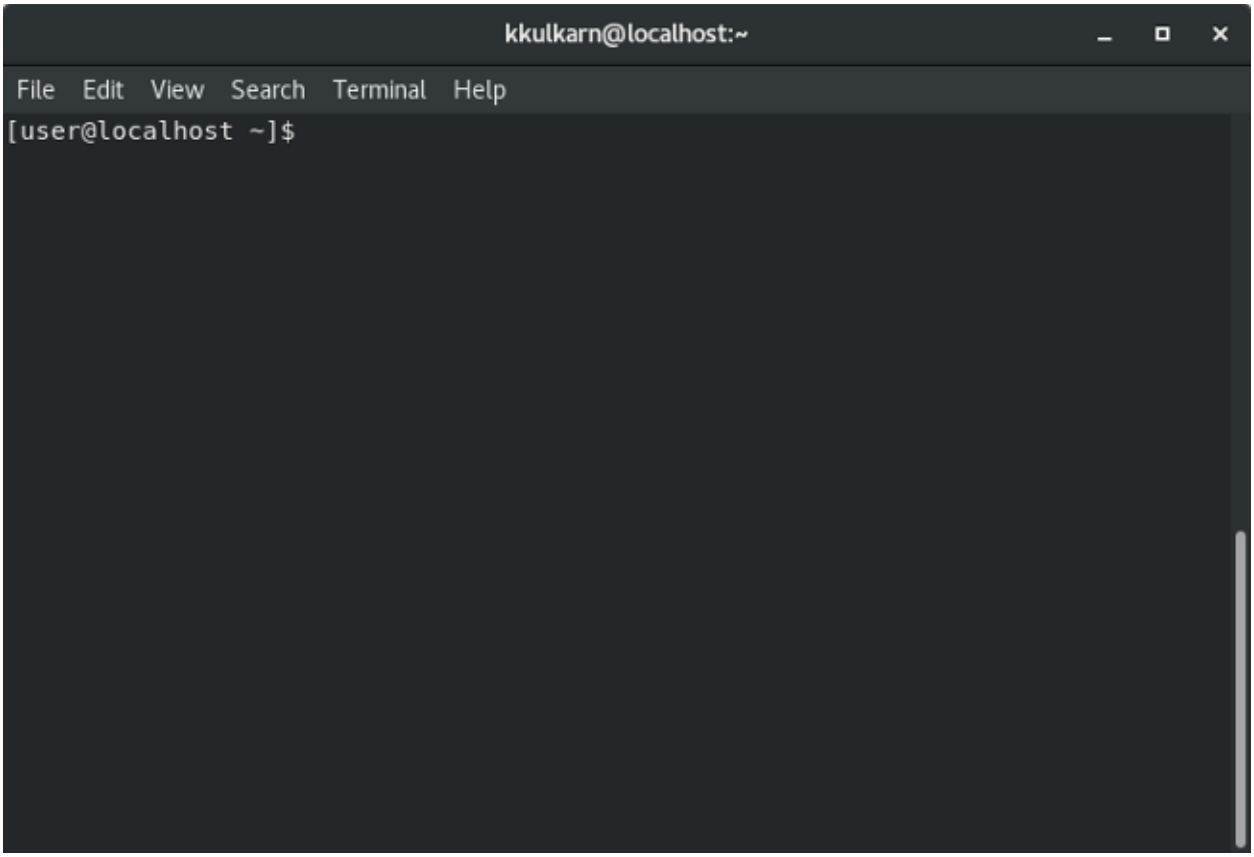
Step 3: Create a file

Once your repo is created, it will look like this:



Don't panic, it's simpler than it looks. Stay with me. Look at the section that starts "...or create a new repository on the command line," and ignore the rest for now.

Open the Terminal program on your computer.



Type git and hit Enter. If it says command bash: git: command not found, then install Git with the command for your Linux operating system or distribution. Check the installation by typing git and hitting Enter; if it's installed, you should see a bunch of information about how you can use the command.

In the terminal, type:

```
mkdir Demo
```

This command will create a directory (or folder) named Demo.

Change your terminal to the Demo directory with the command:

```
cd Demo
```

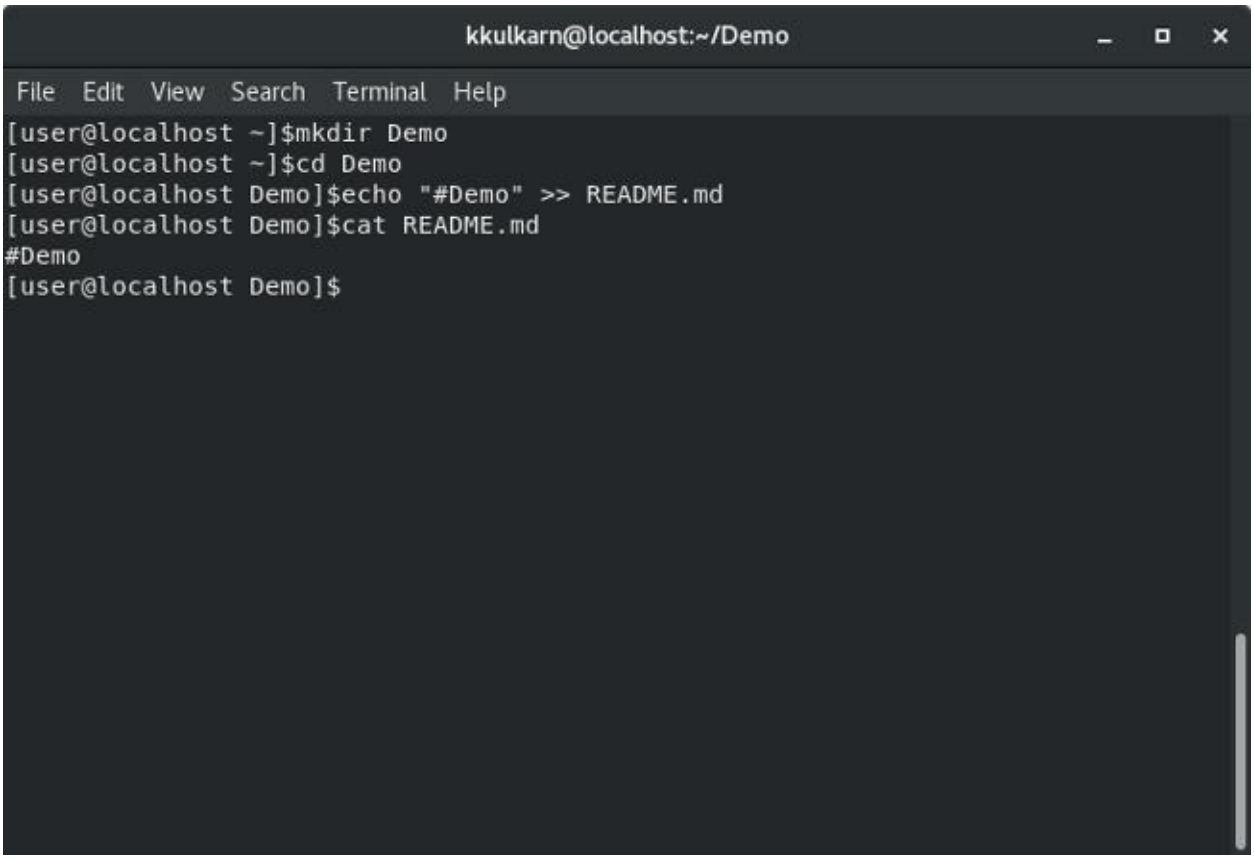
Then enter:

```
echo "#Demo" >> README.md
```

This creates a file named README.md and writes #Demo in it. To check that the file was created successfully, enter:

```
cat README.md
```

This will show you what is inside the README.md file, if the file was created correctly. Your terminal will look like this:



To tell your computer that Demo is a directory managed by the Git program, enter:

```
git init
```

Then, to tell the Git program you care about this file and want to track any changes from this point forward, enter:

```
git add README.md
```

Step 4: Make a commit

So far, you've created a file and told Git about it, and now it's time to create a commit. Commit can be thought of as a milestone. Every time you accomplish some work, you can write a Git commit to store that version of your file, so you can go back later and see what it looked like at that point in time. Whenever you make a change to your file, you create a new version of that file, different from the previous one.

To make a commit, enter:

```
git commit -m "first commit"
```

That's it! You just created a Git commit and included a message that says first commit. You must always write a message in commit; it not only helps you identify a commit, but it also enables you to understand what you did with the file at that point. So tomorrow, if you add a new piece of code in your file, you can write a commit message that says, added new code, and when you come back in a month to look at your commit history or Git log (the list of commits), you will know what you changed in the files.

Step 5: Connect your GitHub repo with your computer

Now, it's time to connect your computer to GitHub with the command:

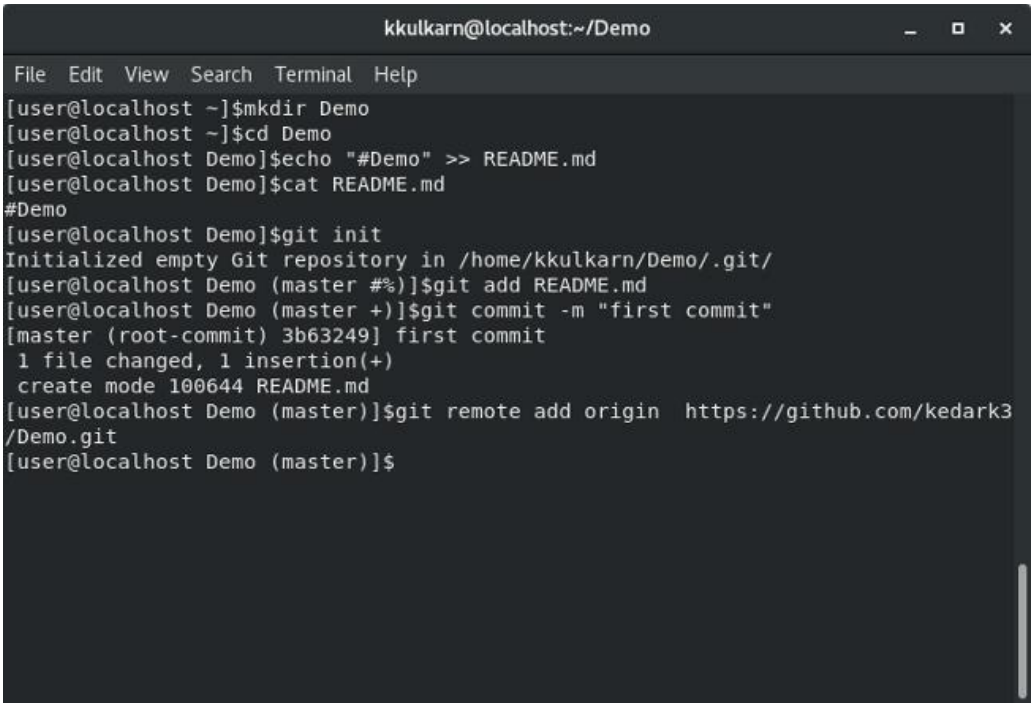
```
git remote add origin https://github.com/<your_username>/Demo.git
```

Let's look at this command step by step. We are telling Git to add a remote called origin with the address https://github.com/<your_username>/Demo.git (i.e., the URL of your Git repo on GitHub.com).

This allows you to interact with your Git repository on GitHub.com by typing origin instead of the full URL and Git will know where to send your code. Why origin? Well, you can name it anything else if you'd like.

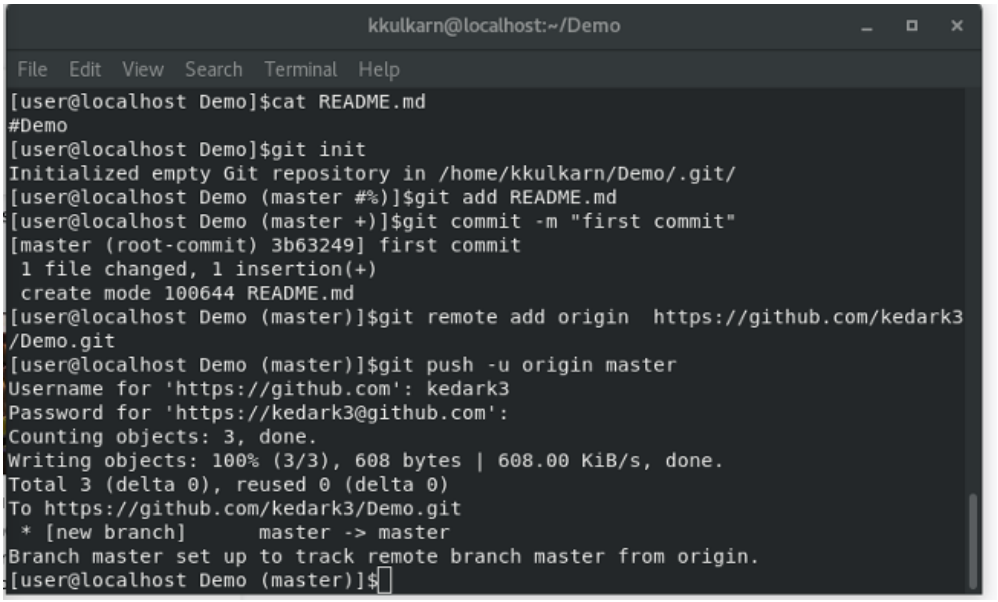
Now we have connected our local copy of the Demo repository to its remote counterpart on GitHub.com.

Your terminal looks like this:

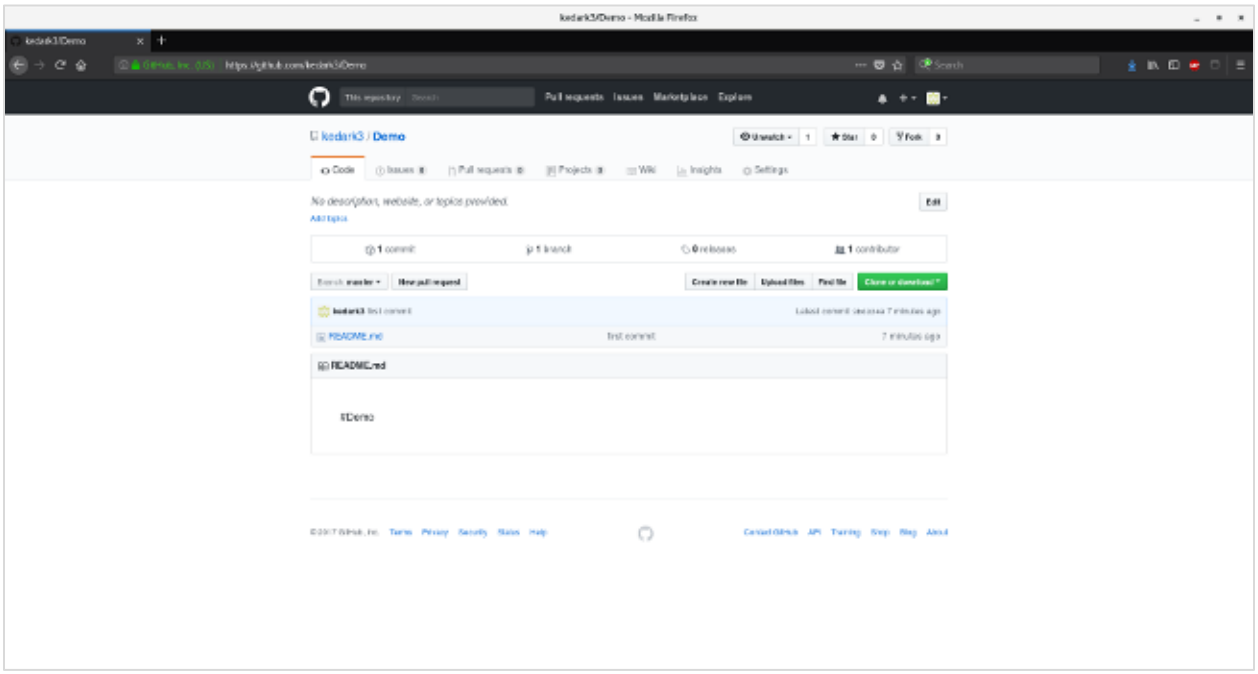


Now that we have added the remote, we can push our code (i.e., upload our README.md file) to GitHub.com.

Once you are done, your terminal will look like this:



And if you go to https://github.com/<your_username>/Demo you will see something like this:



Branching with GitHub

Working with Git Branches

In Git, a branch is a new/separate version of the main repository. Let's say you have a large project, and you need to update the design on it.

How would that work without and with Git:

Without Git:

- Make copies of all the relevant files to avoid impacting the live version
- Start working with the design and find that code depend on code in other files, that also need to be changed!
- Make copies of the dependant files as well. Making sure that every file dependency references the correct file name

- EMERGENCY! There is an unrelated error somewhere else in the project that needs to be fixed ASAP!
- Save all your files, making a note of the names of the copies you were working on
- Work on the unrelated error and update the code to fix it
- Go back to the design, and finish the work there
- Copy the code or rename the files, so the updated design is on the live version
- (2 weeks later, you realize that the unrelated error was not fixed in the new design version because you copied the files before the fix)

With Git:

- With a new branch called new-design, edit the code directly without impacting the main branch
- EMERGENCY! There is an unrelated error somewhere else in the project that needs to be fixed ASAP!
- Create a new branch from the main project called small-error-fix
- Fix the unrelated error and merge the small-error-fix branch with the main branch
- You go back to the new-design branch, and finish the work there
- Merge the new-design branch with main (getting alerted to the small error fix that you were missing)

Branches allow you to work on different parts of a project without impacting the main branch. When the work is complete, a branch can be merged with the main project.

You can even switch between branches and work on different projects without them interfering with each other .

Branching in Git is very lightweight and fast!

New Git Branch

Let add some new features to our **index.html** page.

We are working in our local repository, and we do not want to disturb or possibly wreck the main project.

So, we create a new branch:

Example:

```
[user@localhost] $ git branch hello-world-images
```

Now we created a new branch **called "hello-world-images"**

Let's confirm that we have created a new **branch**:

Example:

```
[user@localhost] $ git branch
    hello-world-images
  * master
```

We can see the new branch with the name "hello-world-images", but the * beside **master** specifies that we are currently on that **branch**.

checkout is the command used to check out a **branch**. Moving us from the current **branch**, to the one specified at the end of the command:

Example:

```
[user@localhost] $ git checkout hello-world-images
                  Switched to branch 'hello-world-images'
```

Now we have moved our current workspace from the master branch, to the new **branch**

Open your favourite editor and make some changes.

For this example, we added an image (img_hello_world.jpg) to the working folder and a line of code in the **index.html** file:

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>Hello World!</title>
<link rel="stylesheet" href="bluestyle.css">
</head>
<body>
<h1>Hello world!</h1>
<div><img src="img_hello_world.jpg" alt="Hello World from Space"
style="width:100%;max-width:960px"></div>
<p>This is the first file in my new Git Repo.</p>
<p>A new line in our file!</p>
</body>
</html>
```

We have made changes to a file and added a new file in the working directory (same directory as the **main branch**).

Now check the status of the current **branch**:

Example:

```
[user@localhost] $ git status
On branch hello-world-images
Changes not staged for commit:
  (use "git add ..." to update what will be committed)
  (use "git restore ..." to discard changes in working directory)
        modified:   index.html

Untracked files:
  (use "git add ..." to include in what will be committed)
        img_hello_world.jpg

no changes added to commit (use "git add" and/or "git commit -a")
```

So, let's go through what happens here:

- There are changes to our index.html, but the file is not staged for commit
- img_hello_world.jpg is not tracked

So, we need to add both files to the Staging Environment for this branch:

Example:

```
[user@localhost] $ git add --all
```

Using --all instead of individual filenames will **Stage** all changed (new, modified, and deleted) files.

Check the **status** of the **branch**:

Example:

```
[user@localhost] $ git status
On branch hello-world-images
Changes to be committed:
  (use "git restore --staged ..." to unstage)
    new file: img_hello_world.jpg
    modified: index.html
```

We are happy with our changes. So, we will commit them to the branch:

Example:

```
[user@localhost] $ git commit -m "Added image to Hello World"
[hello-world-images 0312c55] Added image to Hello World
2 files changed, 1 insertion(+)
create mode 100644 img_hello_world.jpg
```

Now we have a new branch, that is different from the master branch.

Git Merge

Merging is Git's way of putting a forked history back together again. The git merge command lets you take the independent lines of development created by git branch and integrate them into a single branch.

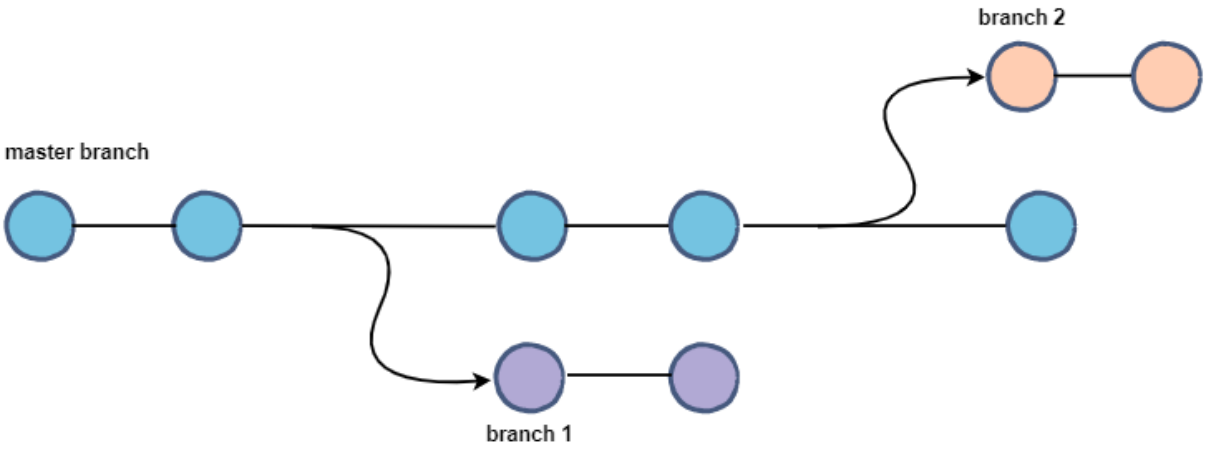
Note that all of the commands presented below merge into the current branch. The current branch will be updated to reflect the merge, but the target branch will be completely unaffected. Again, this means that **git merge** is often used in conjunction with **git checkout** for selecting the current branch and **git branch -d** for deleting the obsolete target branch.

Why do we want to merge branches in GitHub?

We merge different realities/branches in order to integrate them in the master branch. In this way, they become a single identity.

Suppose we have more than one developer working on a single project. Each developer can create and work on their branches independently.

After the final review from a tester, we merge those branches with the master branch to become a final product.



Merge Branches

We have the emergency fix ready, and so let's merge the master and emergency-fix branches.

First, we need to change to the master branch:

Example:

```
[user@localhost] $ git checkout master
                  Switched to branch 'master'
```

Now we merge the current branch (master) with emergency-fix:

Example:

```
[user@localhost] $ git merge emergency-fix
                  Updating 09f4acd..dfa79db
                  Fast-forward
                   index.html | 2 +-
                   1 file changed, 1 insertion(+), 1 deletion(-)
```

Since the emergency-fix branch came directly from master, and no other changes had been made to master while we were working, Git sees this as a continuation of master. So it can "Fast-forward", just pointing both master and emergency-fix to the same commit.

As master and emergency-fix are essentially the same now, we can delete emergency-fix, as it is no longer needed:

Example:

```
[user@localhost] $ git branch -d emergency-fix
                  Deleted branch emergency-fix (was dfa79db).
```

Merge Conflict

Now we can move over to hello-world-images and keep working. Add another image file (img_hello_git.jpg) and change index.html, so it shows it:

Example:

```
[user@localhost] $ git checkout hello-world-images
                  Switched to branch 'hello-world-images'
```

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>Hello World!</title>
<link rel="stylesheet" href="bluestyle.css">
</head>
<body>

<h1>Hello world!</h1>
<div><img src="img_hello_world.jpg" alt="Hello World from Space" style="width:100%;max-
width:960px"></div>
<p>This is the first file in my new Git Repo.</p>
<p>A new line in our file!</p>
<div><img src="img_hello_git.jpg" alt="Hello Git" style="width:100%;max-width:640px"></div>

</body>
</html>
```

Now, we are done with our work here and can stage and commit for this branch:

Example:

```
[user@localhost] $ git add --all
[user@localhost] $ git commit -m "added new image"
                  [hello-world-images 1f1584e] added new image
                   2 files changed, 1 insertion(+)
                   create mode 100644 img_hello_git.jpg
```

We see that index.html has been changed in both branches. Now we are ready to merge hello-world-images into master. But what will happen to the changes we recently made in master?

Example:

```
[user@localhost] $ git checkout master
[user@localhost] $ git merge hello-world-images
                  Auto-merging index.html
                  CONFLICT (content): Merge conflict in index.html
                  Automatic merge failed; fix conflicts and then commit the result.
```

The merge failed, as there is conflict between the versions for index.html. Let us check the status:

Example:

```
[user@localhost] $ git status
                  On branch master
                  You have unmerged paths.
                    (fix conflicts and run "git commit")
                    (use "git merge --abort" to abort the merge)

                  Changes to be committed:
                          new file:   img_hello_git.jpg
                          new file:   img_hello_world.jpg

                  Unmerged paths:
                    (use "git add ..." to mark resolution)
                          both modified:   index.html
```

This confirms there is a conflict in index.html, but the image files are ready and staged to be committed.

So, we need to fix that conflict. Open the file in our editor:

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>Hello World!</title>
<link rel="stylesheet" href="bluestyle.css">
</head>
<body>

<h1>Hello world!</h1>
<div><img src="img_hello_world.jpg" alt="Hello World from Space" style="width:100%;max-
width:960px"></div>
<p>This is the first file in my new Git Repo.</p>
<<<<<<< HEAD
<p>This line is here to show how merging works.</p>
=======
<p>A new line in our file!</p>
<div><img src="img_hello_git.jpg" alt="Hello Git" style="width:100%;max-width:640px"></div>
>>>>>>> hello-world-images

</body>
</html>
```

We can see the differences between the versions and edit it like we want:

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>Hello World!</title>
<link rel="stylesheet" href="bluestyle.css">
</head>
<body>

<h1>Hello world!</h1>
<div><img src="img_hello_world.jpg" alt="Hello World from Space" style="width:100%;max-
width:960px"></div>
<p>This is the first file in my new Git Repo.</p>
<p>This line is here to show how merging works.</p>
<div><img src="img_hello_git.jpg" alt="Hello Git" style="width:100%;max-width:640px"></div>

</body>
</html>
```

Now we can stage index.html and check the status:

Example:

```
[user@localhost] $ git add index.html
[user@localhost] $ git status
                   On branch master
                   All conflicts fixed but you are still merging.
                     (use "git commit" to conclude merge)

                   Changes to be committed:
                           new file:   img_hello_git.jpg
                           new file:   img_hello_world.jpg
                           modified:   index.html
```

The conflict has been fixed, and we can use commit to conclude the merge:

Example:

```
[user@localhost] $ git commit -m "merged with hello-world-images after fixing conflicts"
                  [master e0b6038] merged with hello-world-images after fixing conflicts
```

And delete the hello-world-images branch:

Example:

```
[user@localhost] $ git branch -d hello-world-images
                  Deleted branch hello-world-images (was 1f1584e).
```

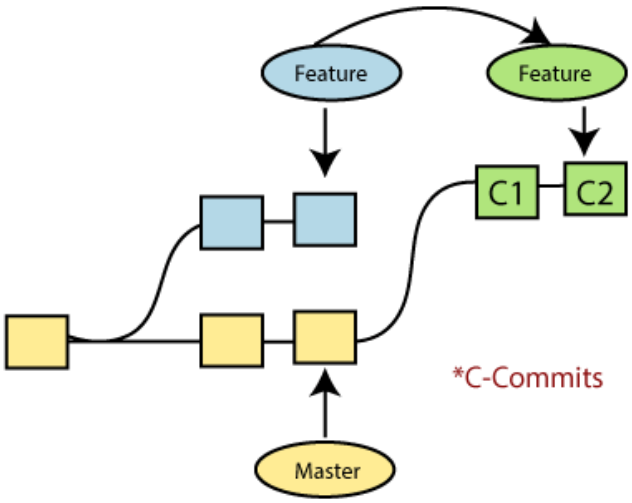
Now you have a better understanding of how branches and merging works.

Rebasing with GitHub

Rebasing is a process to reapply commits on top of another base trip. It is used to apply a sequence of commits from distinct branches into a final commit. It is an alternative of git merge command. It is a linear process of merging.

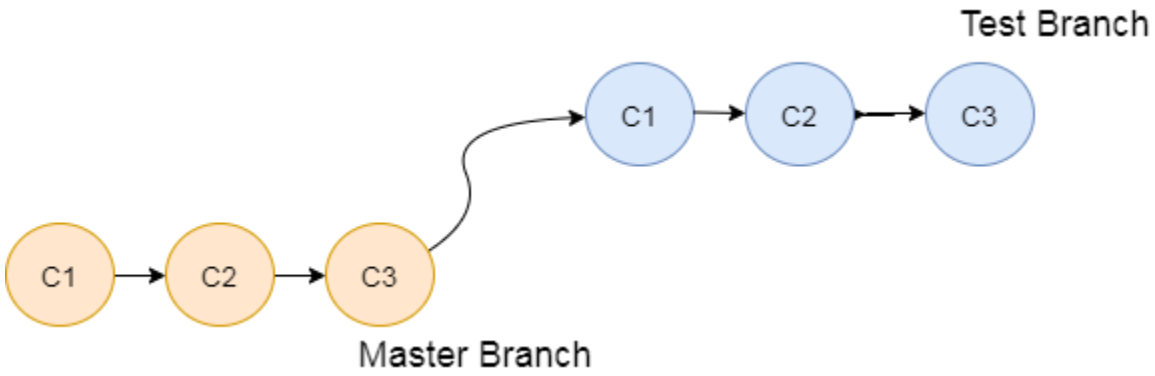
In Git, the term rebase is referred to as the process of moving or combining a sequence of commits to a new base commit. Rebasing is very beneficial and it visualized the process in the environment of a feature branching workflow.

It is good to rebase your branch before merging it.



Generally, it is an alternative of git merge command. Merge is always a forward changing record. Comparatively, rebase is a compelling history rewriting tool in git. It merges the different commits one by one.

Suppose you have made three commits in your master branch and three in your other branch named test. If you merge this, then it will merge all commits in a time. But if you rebase it, then it will be merged in a linear manner. Consider the below image:



The above image describes how git rebase works. The three commits of the master branch are merged linearly with the commits of the test branch.

Merging is the most straightforward way to integrate the branches. It performs a three-way merge between the two latest branch commits.

How to Rebase

When you made some commits on a feature branch (test branch) and some in the master branch. You can rebase any of these branches. Use the git log command to track the changes (commit history). Checkout to the desired branch you want to rebase. Now perform the rebase command as follows:

Syntax:

```
$git rebase <branch name>
```

If there are some conflicts in the branch, resolve them, and perform below commands to continue changes:

```
$ git status
```

It is used to check the status,

```
$git rebase --continue
```

The above command is used to continue with the changes you made. If you want to skip the change, you can skip as follows:

```
$ git rebase --skip
```

When the rebasing is completed. Push the repository to the origin. Consider the below example to understand the git merge command.

Suppose that you have a branch say test2 on which you are working. You are now on the test2 branch and made some changes in the project's file newfile1.txt.

Add this file to repository:

```
$ git add newfile1.txt
```

Now, commit the changes. Use the below command:

```
$ git commit -m "new commit for test2 branch."
```

The output will look like:

```
[test2 a835504] new commitfor test2 branch
 1 file changed, 1 insertion(+)
```

Switch the branch to master:

```
$ git checkout master
```

Output:

```
Switched to branch 'master.'
Your branch is up to date with 'origin/master.'
```

Now you are on the master branch. I have added the changes to my file, says newfile.txt. The below command is used to add the file in the repository.

```
$ git add newfile.txt
```

Now commit the file for changes:

```
$ git commit -m " new commit made on the master branch."
```

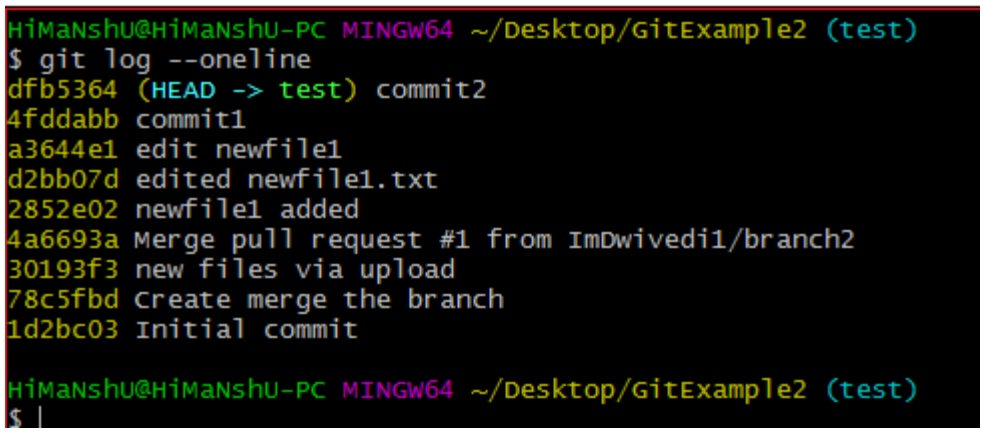
Output:

```
[master 7fe5e7a]  new commit made on master
 1 file changed, 1 insertion(+)
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
```

To check the log history, perform the below command.

```
$ git log --oneline
```

Output:



As we can see in the log history, there is a new commit in the master branch.

Git Interactive Rebase

Git facilitates with Interactive Rebase; it is a potent tool that allows various operations like edit, rewrite, reorder, and more on existing commits. Interactive Rebase can only be operated on the currently checked out branch. Therefore, set your local HEAD branch at the sidebar.

Git interactive rebase can be invoked with rebase command, just type -i along with rebase command. Here 'i' stands for interactive. Syntax of this command is given below:

Syntax:

```
$ git rebase -i
```

It will list all the available interactive options.

Output:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git rebase -i
hint: Waiting for your editor to close the file...
```

After the given output, it will open an editor with available options. Consider the below output:

Output:

git-rebase-todo

```
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented outp

# Rebase 0a1a475..0a1a475 onto 0a1a475 (1 command)
#
# Commands:
```

When we perform the git interactive rebase command, it will open your default text editor with the above output.

The options it contains are listed below:

Pick (-p):

Pick stands here that the commit is included. Order of the commits depends upon the order of the pick commands during rebase. If you do not want to add a commit, you have to delete the entire line.

Reword (-r):

The reword is quite similar to pick command. The reword option paused the rebase process and provides a chance to alter the commit message. It does not affect any changes made by the commit.

Edit (-e):

The edit option allows for amending the commit. The amending means, commits can be added or changed entirely. We can also make additional commits before rebase continue command. It allows us to split a large commit into the smaller commit; moreover, we can remove erroneous changes made in a commit.

Squash (-s):

The squash option allows you to combine two or more commits into a single commit. It also allows us to write a new commit message for describing the changes.

Fixup (-f):

It is quite similar to the squash command. It discarded the message of the commit to be merged. The older commit message is used to describe both changes.

Exec (-x):

The exec option allows you to run arbitrary shell commands against a commit.

Break (-b):

The break option stops the rebasing at just position. It will continue rebasing later with 'git rebase --continue' command.

Drop (-d):

The drop option is used to remove the commit.

Label (-l):

The label option is used to mark the current head position with a name.

Reset (-t):

The reset option is used to reset head to a label.

Exercise

1. After you add a file, it becomes

   a) Committed
   b) Modified
   c) Staged
   d) Untracked

2. What command lets you create a connection between a local and remote repository?

   a) Git remote add origin
   b) Git remote add new
   c) Git remote new origin
   d) Git remote origin

3. How do you supply a commit message to a commit?

   Git message "My first commit"
   Git add "My first commit"
   Git commit "My first commit"
   Git commit -m "I'm coding!"

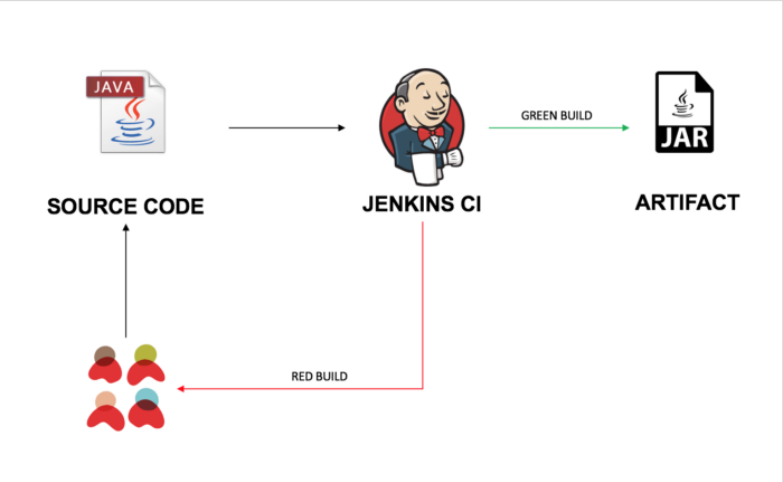
Introduction to Jenkins

What is Jenkins?

Jenkins is an open-source automation tool written in Java programming language that allows continuous integration.

Jenkins builds and tests our software projects which continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.

It also allows us to continuously deliver our software by integrating with a large number of testing and deployment technologies.



Jenkins offers a straightforward way to set up a continuous integration or continuous delivery environment for almost any combination of languages and source code repositories using pipelines, as well as automating other routine development tasks.

With the help of Jenkins, organizations can speed up the software development process through automation. Jenkins adds development life-cycle processes of all kinds, including build, document, test, package, stage, deploy static analysis and much more.

Jenkins achieves CI (Continuous Integration) with the help of plugins. Plugins is used to allow the integration of various DevOps stages. If you want to integrate a particular tool, you have to install the plugins for that tool. For example: Maven 2 Project, Git, HTML Publisher, Amazon EC2, etc.

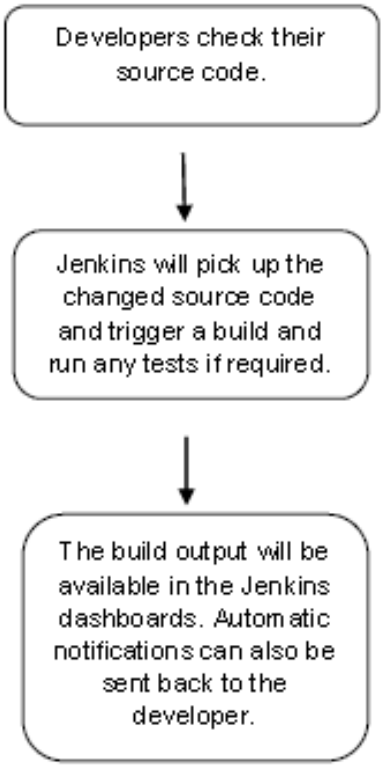
For example: If any organization is developing a project, then Jenkins will continuously test your project builds and show you the errors in early stages of your development.

Possible steps executed by Jenkins are for example:

- Perform a software build using a build system like Gradle or Maven Apache
- Execute a shell script
- Archive a build result
- Running software tests

Why Jenkins?

Jenkins is a software that allows continuous integration. Jenkins will be installed on a server where the central build will take place. The following flowchart demonstrates a very simple workflow of how Jenkins works.



What is Continuous Integration?

Continuous Integration (CI) is a development practice in which the developers are needs to commit changes to the source code in a shared repository at regular intervals. Every commit made in the repository is then built. This allows the development teams to detect the problems early.

Continuous integration requires the developers to have regular builds. The general practice is that whenever a code commit occurs, a build should be triggered.

Continuous Integration with Jenkins

Let's consider a scenario where the complete source code of the application was built and then deployed on test server for testing. It sounds like a perfect way to develop software, but this process has many problems.

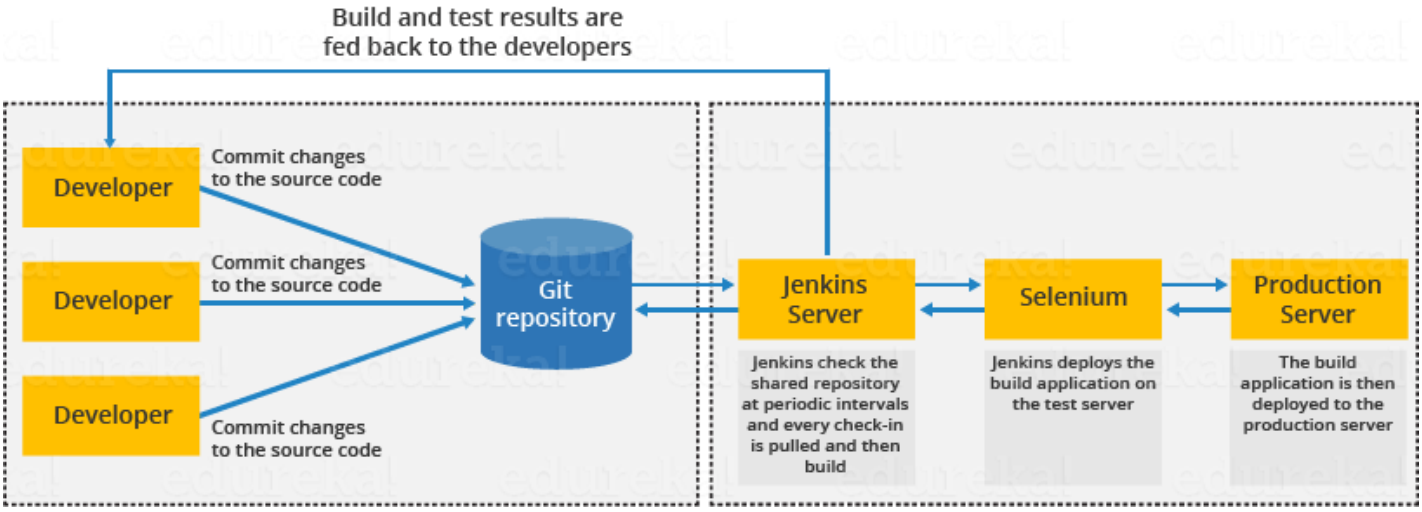
- Developer teams have to wait till the complete software is developed for the test results.
- There is a high prospect that the test results might show multiple bugs. It was tough for developers to locate those bugs because they have to check the entire source code of the application.
- It slows the software delivery process.
- Continuous feedback pertaining to things like architectural or coding issues, build failures, test status and file release uploads was missing due to which the quality of software can go down.
- The whole process was manual which increases the threat of frequent failure.

It is obvious from the above stated problems that not only the software delivery process became slow but the quality of software also went down. This leads to customer dissatisfaction.

So, to overcome such problem there was a need for a system to exist where developers can continuously trigger a build and test for every change made in the source code.

This is what Continuous Integration (CI) is all about. Jenkins is the most mature Continuous Integration tool available so let us see how Continuous Integration with Jenkins overcame the above shortcomings.

Let's see a generic flow diagram of Continuous Integration with Jenkins:



Let's see how Jenkins works. The above diagram is representing the following functions:

- First of all, a developer commits the code to the source code repository. Meanwhile, the Jenkins checks the repository at regular intervals for changes.
- Soon after a commit occurs, the Jenkins server finds the changes that have occurred in the source code repository. Jenkins will draw those changes and will start preparing a new build.
- If the build fails, then the concerned team will be notified.
- If built is successful, then Jenkins server deploys the built in the test server.
- After testing, Jenkins server generates a feedback and then notifies the developers about the build and test results.
- It will continue to verify the source code repository for changes made in the source code and the whole process keeps on repeating

Advantages of Jenkins

- It is an open source tool.
- It is free of cost.
- It does not require additional installations or components. Means it is easy to install.
- Easily configurable.
- It supports 1000 or more plugins to ease your work. If a plugin does not exist, you can write the script for it and share with community.

- It is built in java and hence it is portable.
- It is platform independent. It is available for all platforms and different operating systems. Like OS X, Windows or Linux.
- Easy support, since its open source and widely used.
- Jenkins also supports cloud-based architecture so that we can deploy Jenkins in cloud based platforms.

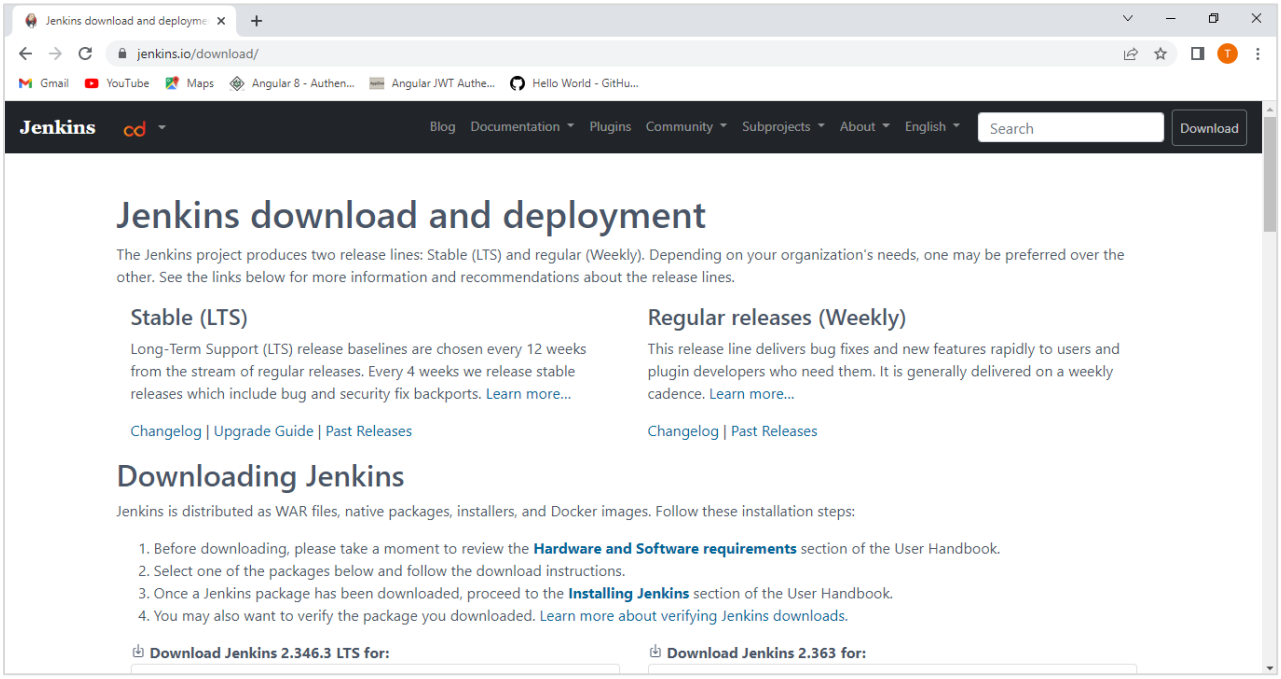
Disadvantages of Jenkins

- Its interface is out dated and not user friendly compared to current user interface trends.
- Not easy to maintain it because it runs on a server and requires some skills as server administrator to monitor its activity.
- CI regularly breaks due to some small setting changes. CI will be paused and therefore requires some developer's team attention.

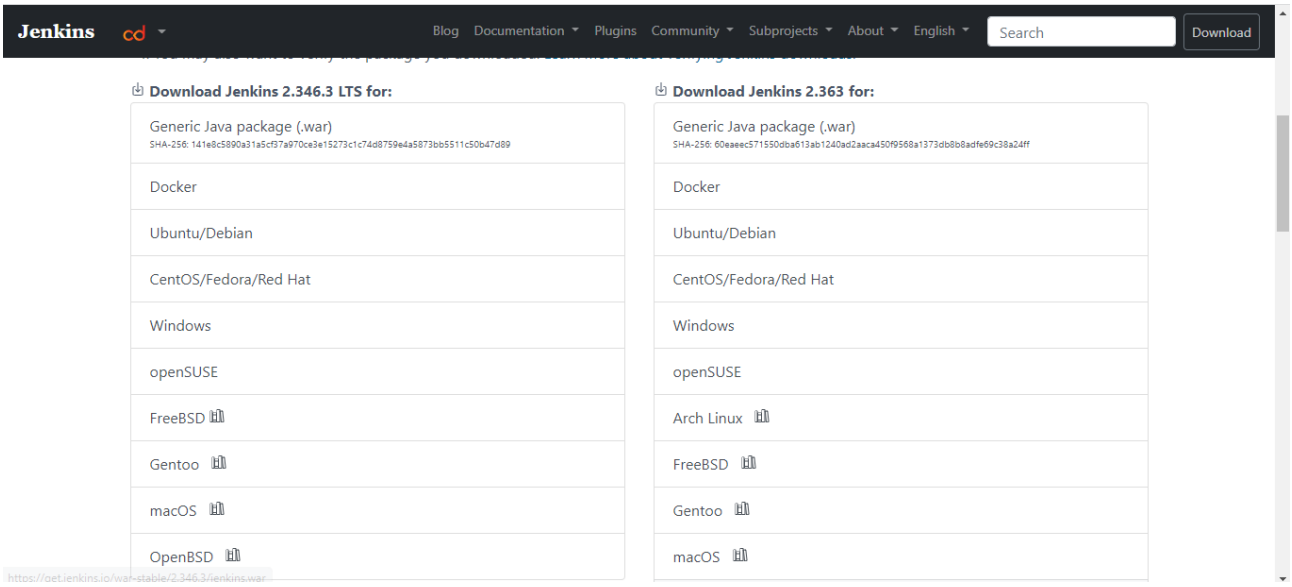
Installation

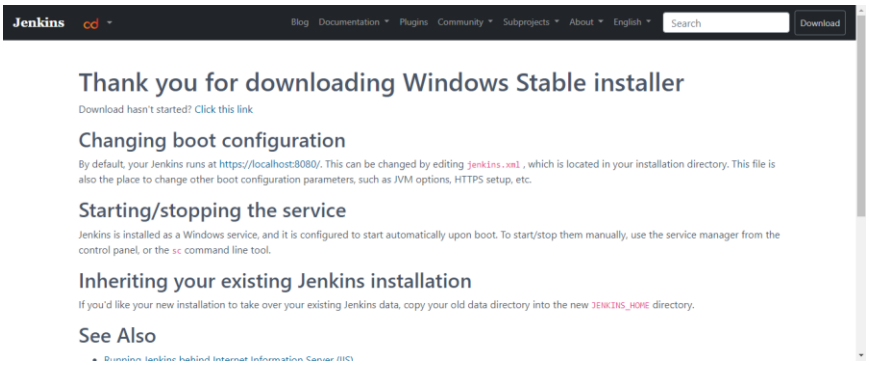
The official website for Jenkins is https://www.jenkins.io/. If you click the given link, you can get the home page of the Jenkins official website as shown below.

By default, the latest release and the Long-Term support release will be available for download. The past releases are also available for download.



Choose the operating system and the file will start downloading on your system.





Starting Jenkins

Open the command prompt. From the command prompt, browse to the directory where the jenkins.war file is present. Run the following command:

```
D:\>Java –jar Jenkins.war
```

After the command is run, various tasks will run, one of which is the extraction of the war file which is done by an embedded webserver called winstone.

```
D:\>Java –jar Jenkins.war
Running from: D:\jenkins.war
Webroot: $user.home/ .jenkins
Sep 29, 2015 4:10:46 PM winstone.Logger logInternal
INFO: Beginning extraction from war file
```

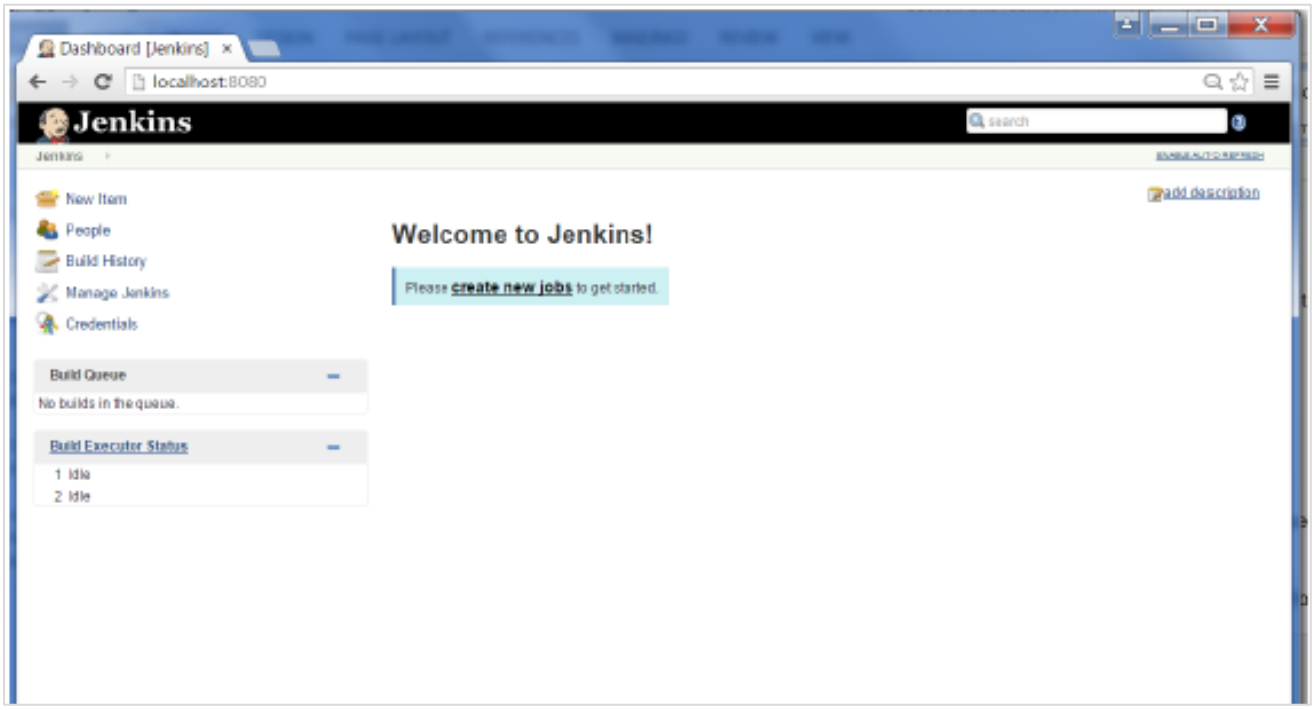
Once the processing is complete without major errors, the following line will come in the output of the command prompt.

```
INFO: Jenkins is fully up and running
```

Accessing Jenkins

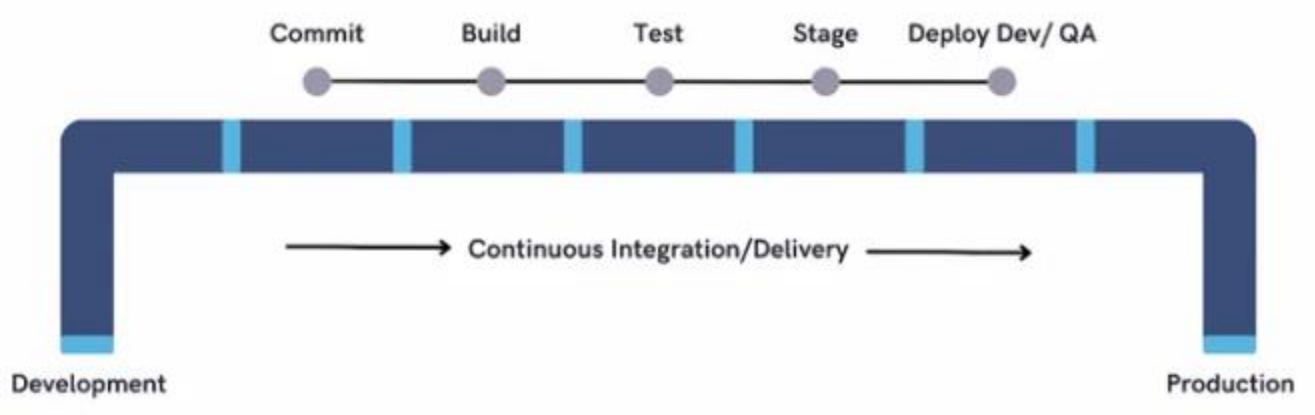
Once Jenkins is up and running, one can access Jenkins from the link – **http://localhost:8080**

This link will bring up the Jenkins dashboard.



What is the Jenkins Pipeline?

The interconnection of several tasks and events in a sequence is known as the Jenkins Pipeline. It is a pool of plugins that helps the continuous delivery pipelines with easy integration and implementation.



The primary feature of a Jenkins pipeline is that every task or job is dependent on some other task or job. However, there are different states in the case of continuous delivery pipelines: build, test, deploy, release, etc. These states are interconnected with each other.

A CD pipeline is a sequence of events in which these states work. It is an automated expression required by the processes to get version control software. Every change made to the software has to pass via multiple complex processes before the release.

This step reliably guarantees software development as it includes multiple testing and deployment stages.

How to Implement the Pipelines?

To define the Jenkins pipeline, JenkinsFile is used. It is a text file. It is used for the implementation of pipelines in code. This process is explained using DSL.

There are two syntaxes used to define the JenkinsFile.

- **Scripted pipeline syntax:** it runs on Jenkins master and uses its resources to convert pipelines into atomic commands.
- **Declarative pipeline syntax:** it is simple to create pipelines with this syntax. There are easy ways to control several aspects related to the execution of the pipelines.

Some of the advantages of JenkinsFile include:

- It is easier to review the code on the pipeline.
- You can conduct an audit on the Jenkins pipeline.
- It helps to execute full requests for the pipelines created for several branches.

With the help of continuous integration abilities, Jenkins automates the software development process.

You can have various users in the Jenkins pipeline for editing and executing several processes.

You can pause Jenkins pipeline processes till you get any user output.

Jenkins Pipeline Concepts

Pipeline: This is the user-defined block, which contains all the processes such as build, test, deploy, etc. it is a group of all the stages in a JenkinsFile. All the stages and steps are defined in this block. It is used in declarative pipeline syntax.

```
pipeline{
}
```

Node: The node is a machine on which Jenkins runs is called a node. A node block is used in scripted pipeline syntax.

```
node{
}
```

Stage: This block contains a series of steps in a pipeline. i.e., build, test, and deploy processes all come together in a stage. Generally, a stage block visualizes the Jenkins pipeline process.

Let's see an example for multiple stages, where each stage performs a specific task:

```
pipeline {
   agent any
   stages {
        stage ('Build') {
           ...
        }
        stage ('Test') {
           ...
        }
        stage ('QA') {
           ...
        }
        stage ('Deploy') {
           ...
        }
        stage ('Monitor') {
           ...
        }
   }
}
```

Step: A step is a single task that executes a specific process at a defined time. A pipeline involves a series of steps defined within a stage block.

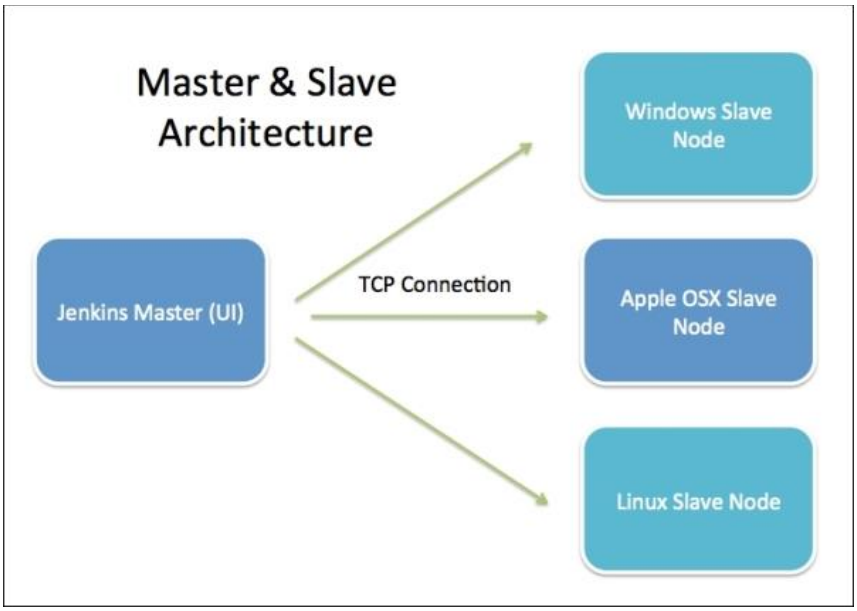
```
pipeline {
   agent any
   stages {
        stage ('Build') {
          steps {
               echo 'Running build phase...'
          }
        }
   }
}
```

Jenkins Architecture

Jenkins follows Master-Slave architecture to manage distributed builds. In this architecture, slave and master communicate through TCP/IP protocol.

Jenkins architecture has two components:

- Jenkins Master/Server
- Jenkins Slave/Node/Build Server



Jenkins Master

The main server of Jenkins is the Jenkins Master. It is a web dashboard which is nothing but powered from a war file. By default, it runs on 8080 port. With the help of Dashboard, we can configure the jobs/projects but the build takes place in Nodes/Slave. By default, one node (slave) is configured and running in Jenkins server. We can add more nodes using IP address, user name and password using the ssh, jnlp or webstart methods.

The server's job or master's job is to handle:

- Scheduling build jobs.
- Dispatching builds to the nodes/slaves for the actual execution.
- Monitor the nodes/slaves (possibly taking them online and offline as required).
- Recording and presenting the build results.
- A Master/Server instance of Jenkins can also execute build jobs directly.

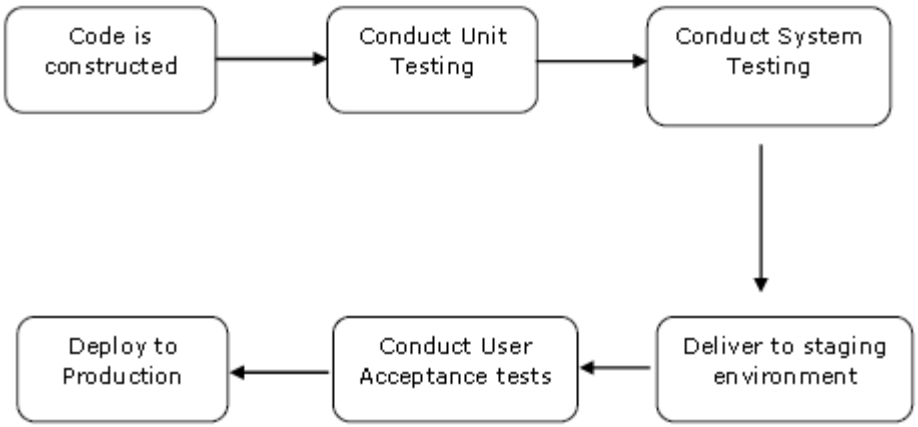
Jenkins Slave

Jenkins slave is used to execute the build jobs dispatched by the master. We can configure a project to always run on a particular slave machine, or particular type of slave machine, or simple let the Jenkins to pick the next available slave/node.

As we know Jenkins is developed using Java is platform independent thus Jenkins Master/Servers and Slave/nod es can be configured in any servers including Linux, Windows, and Mac.

Jenkins - Continuous Deployment

Jenkins is used in providing good support for continuous deployment and delivery. The flow of a software development till the deployment is shown below:

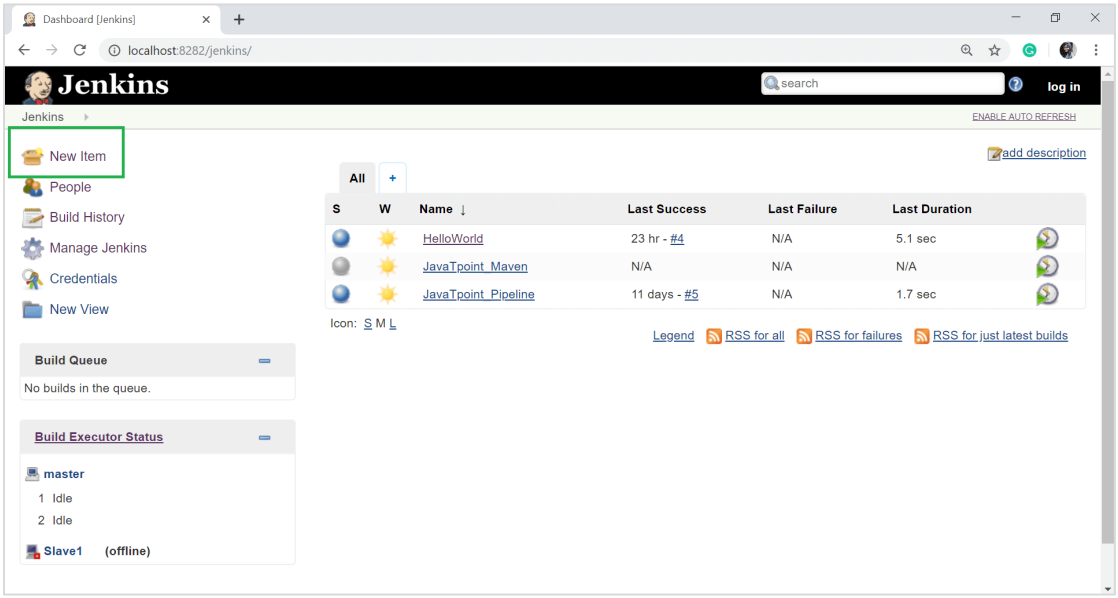


The main part of continuous deployment to make sure that the above entire process is automated. Jenkins provides various plugins for all these things. One of them is "Deploy to container" plugin, which was seen in earlier sections.

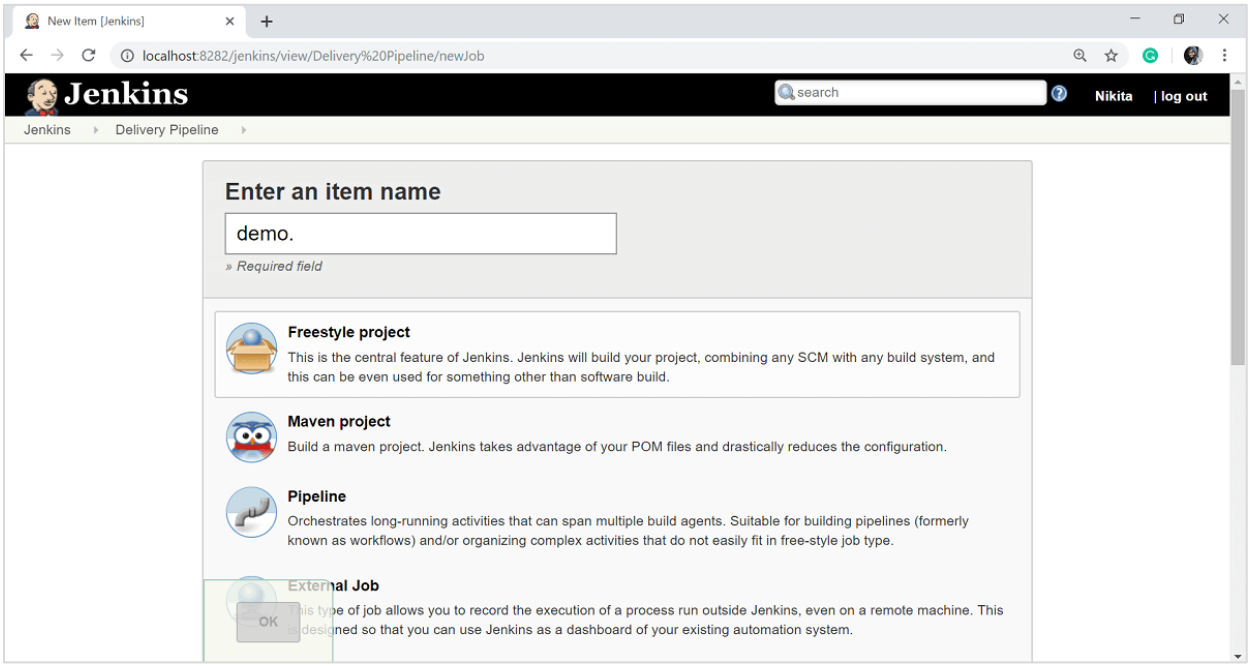
Jenkins provides various plugins which are used to give a graphical representation of the continuous deployment process.

To understand that, let's first create another project in Jenkins so that we can see how it works and which emulates the QA stage:

Step 1: Go to the Jenkins Dashboard and select New Item.

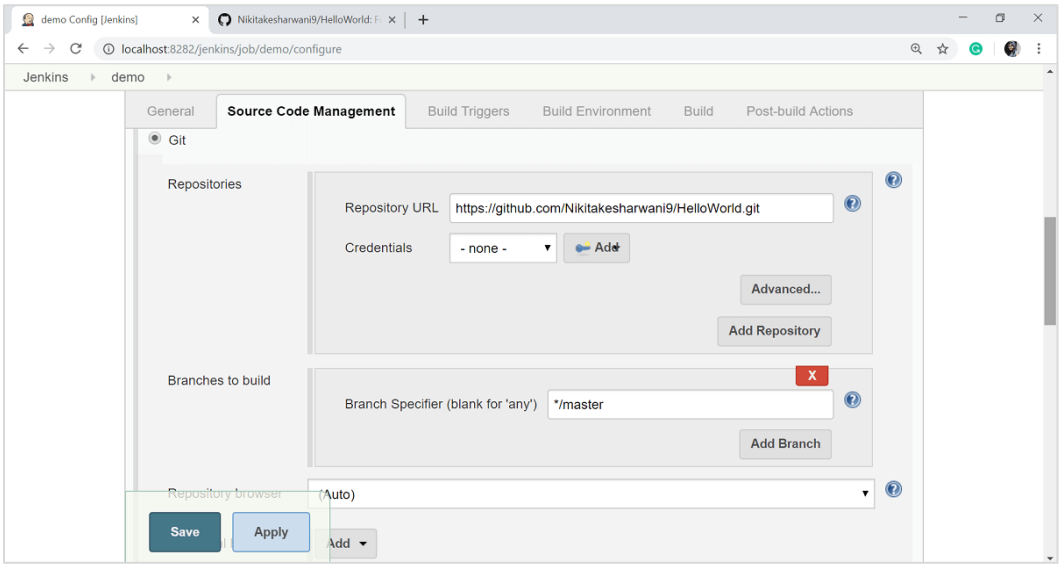


Step 2: Give the Item name and choose Freestyle project option. Here I have given the item name "demo". Click on OK button.

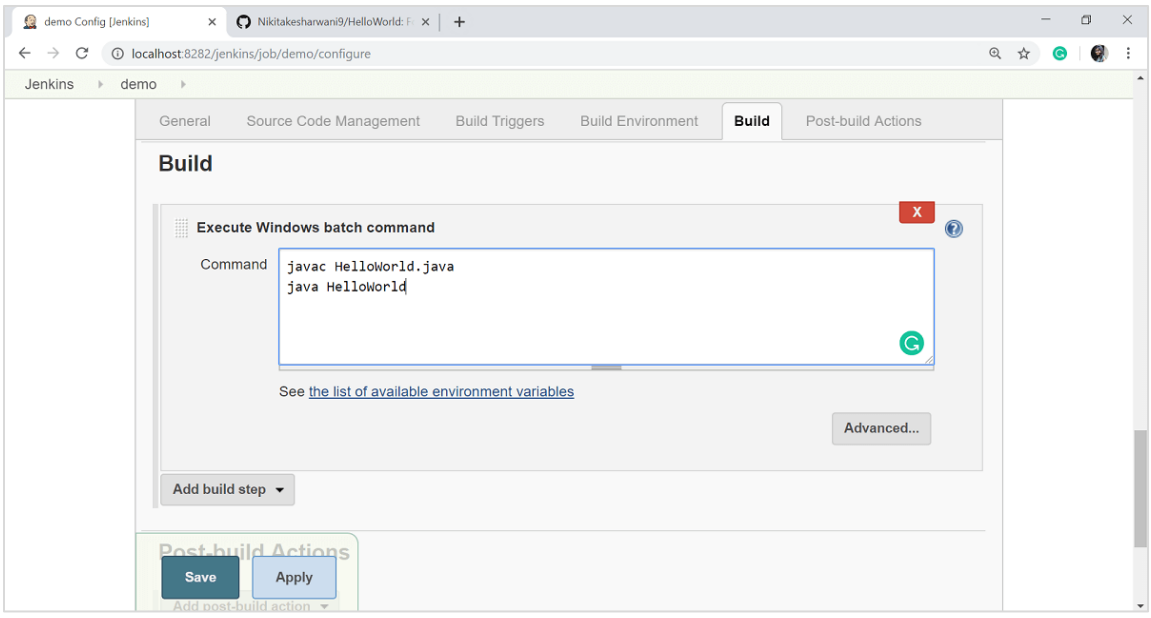


Step 3: In this example, we are keeping it simple and just using to print HelloWorld.

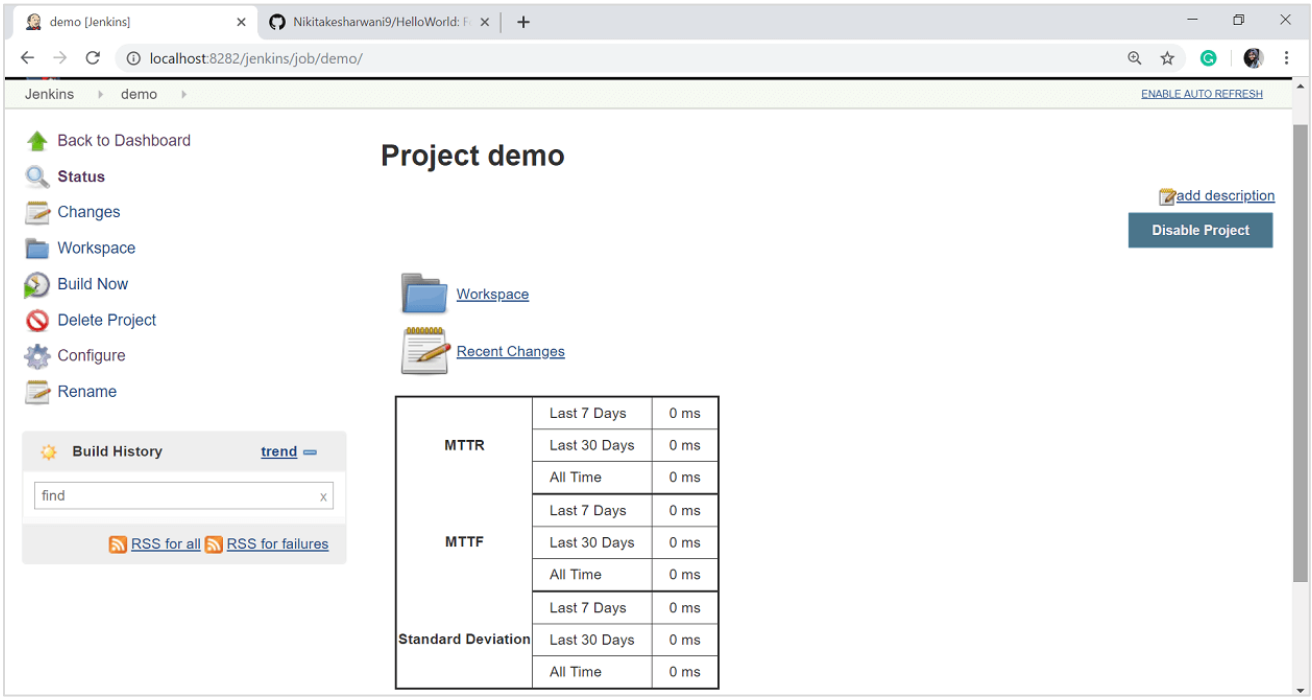
Select the Git option and enter the GitHub repository of your **HelloWorld** program in the **Repository URL** section.



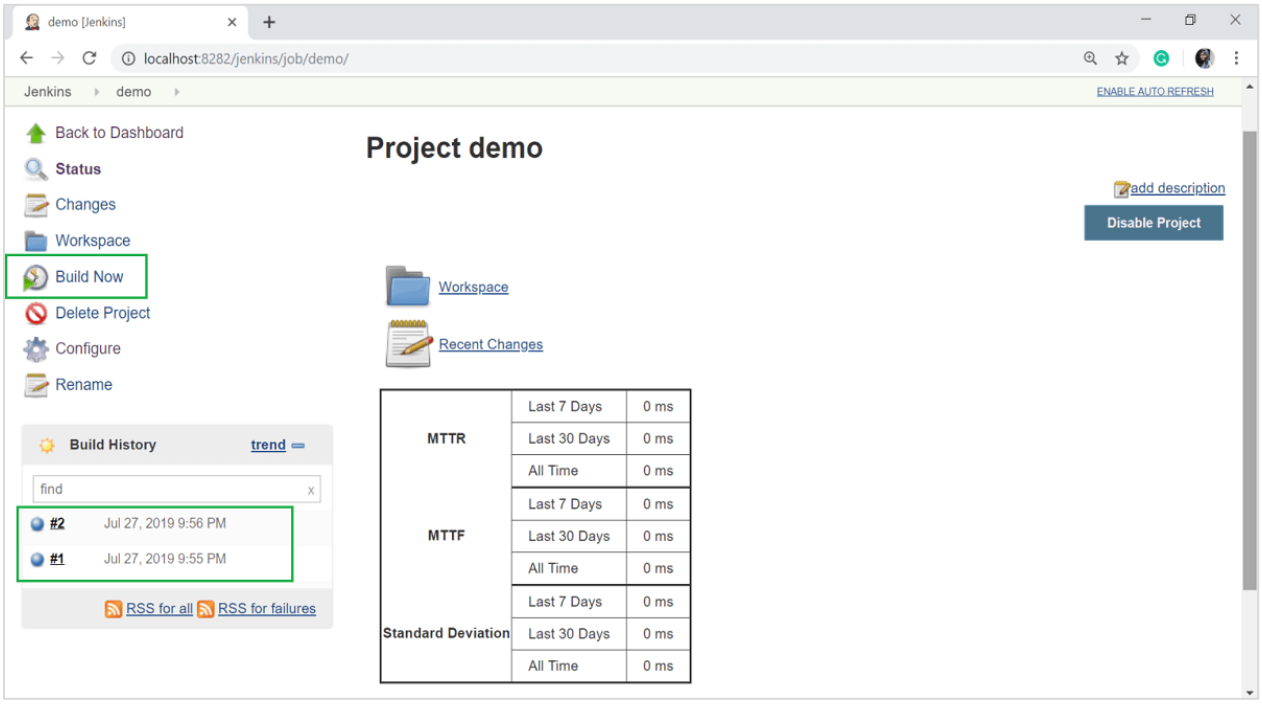
Step 4: Select the Execute Windows batch command option from the add build step button and give the command to run your java program.



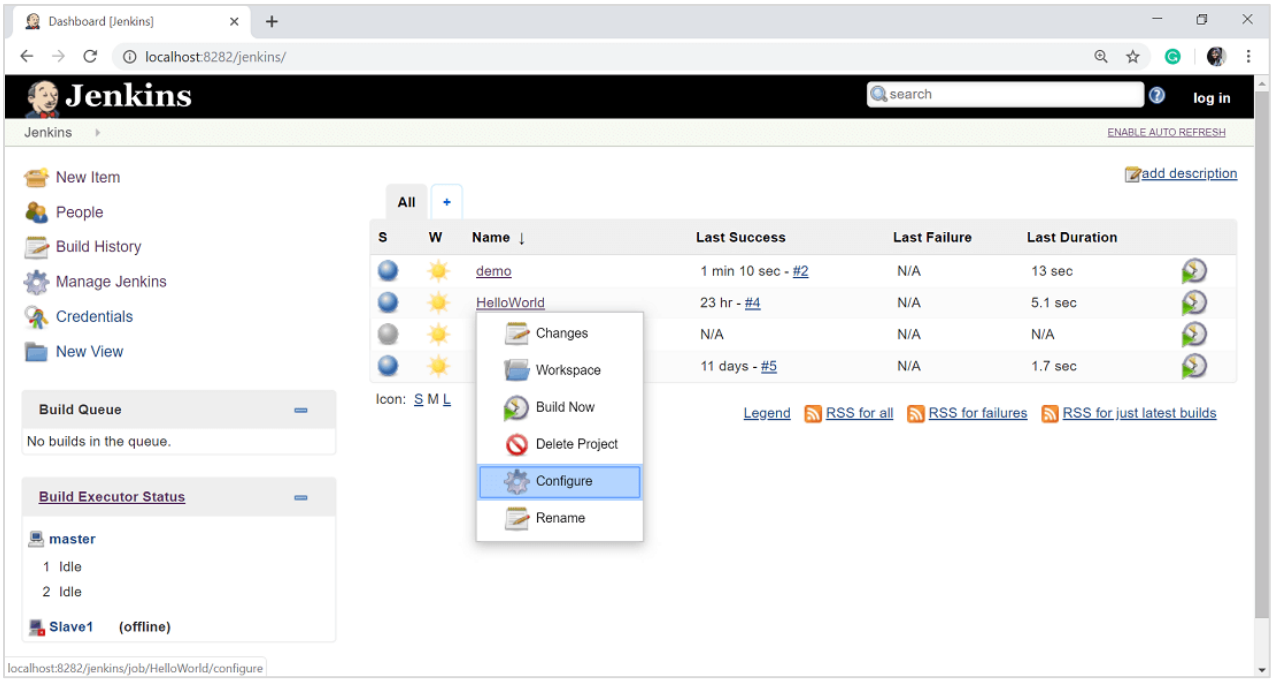
Step 5: Click on Apply then Save button.



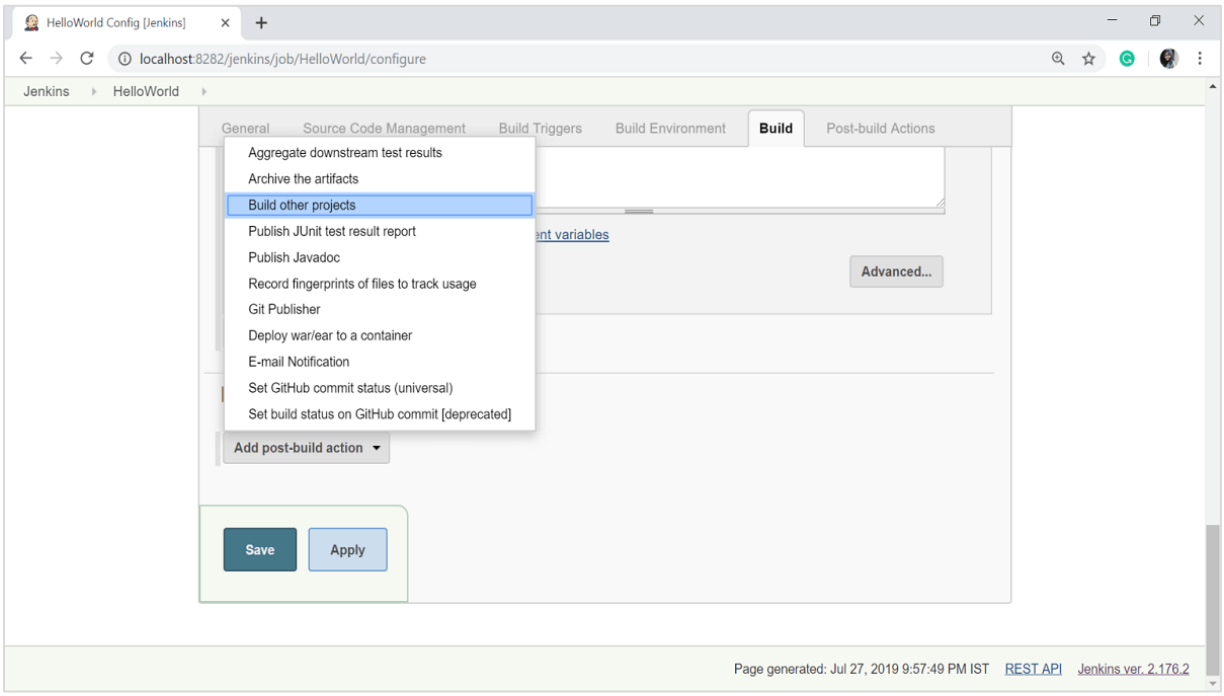
So, our project demo is now created. You can check a build to see if the build is successfully created or not. To check a build, click on the Build Now option.



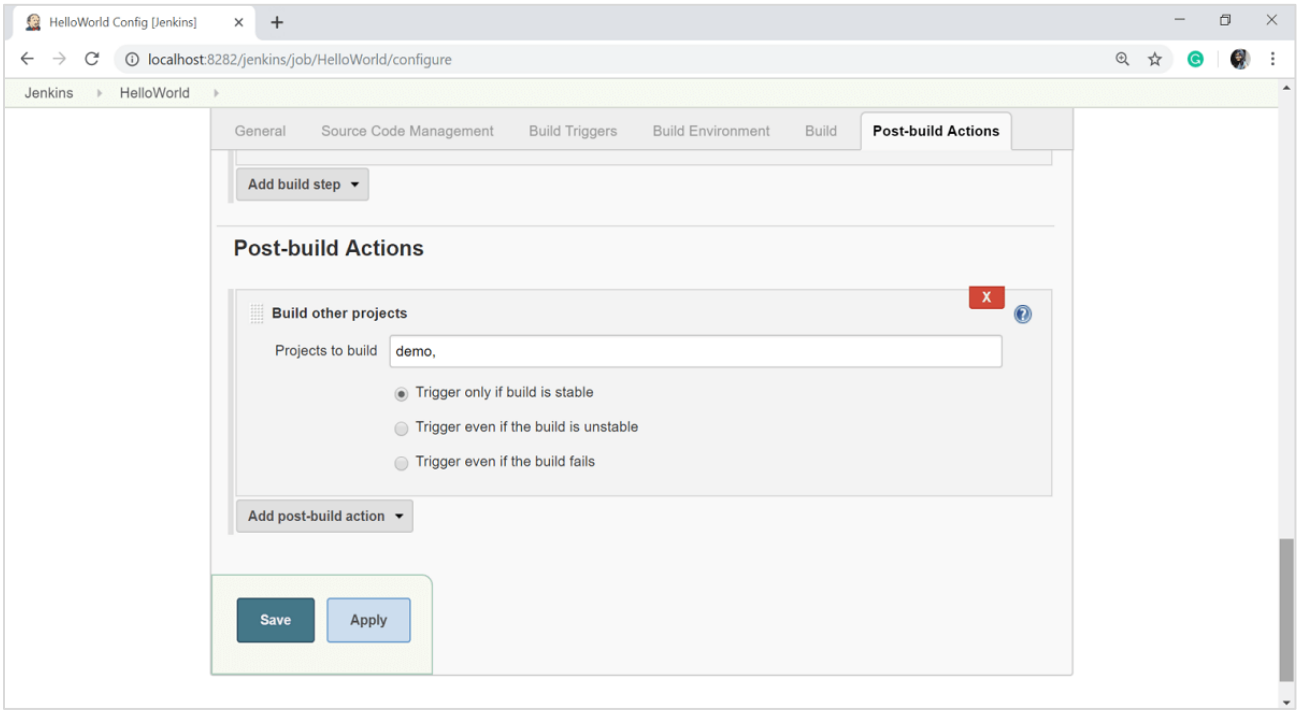
Step 6: Now, go to your previously created Helloworld project and click on the Configure option.



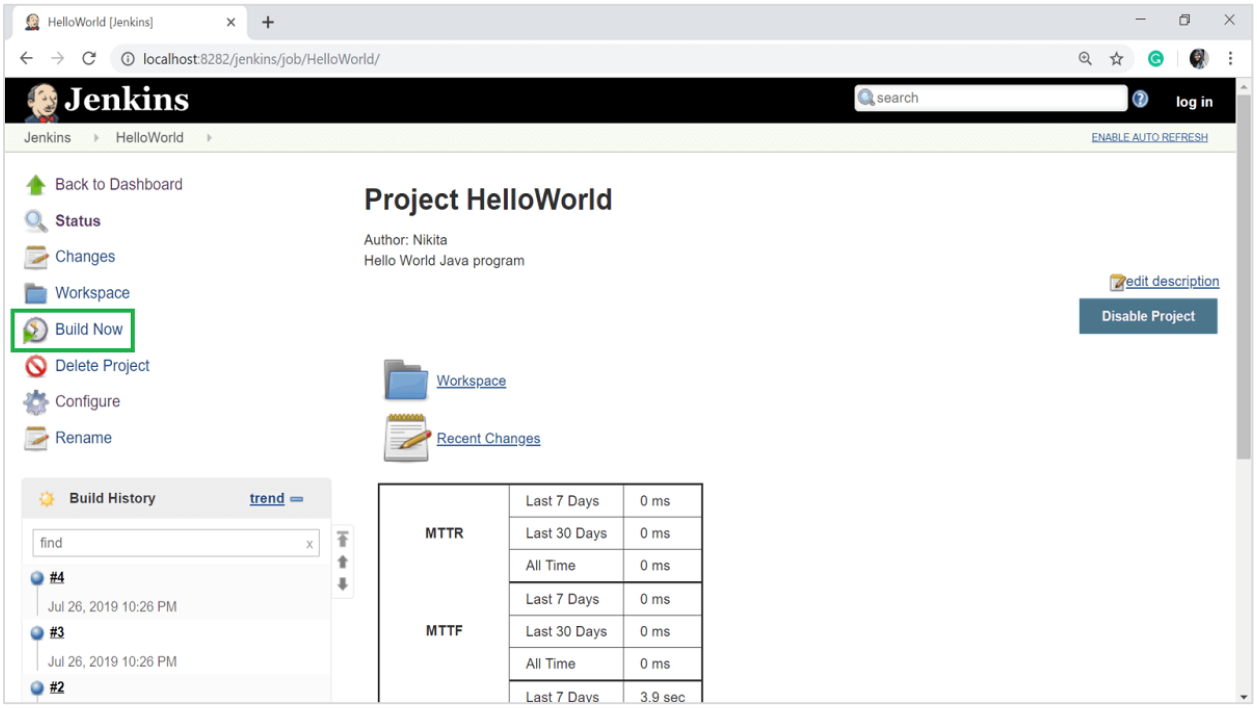
Step 7: In the Project configuration, select the Add post-build action and choose Build other projects option.



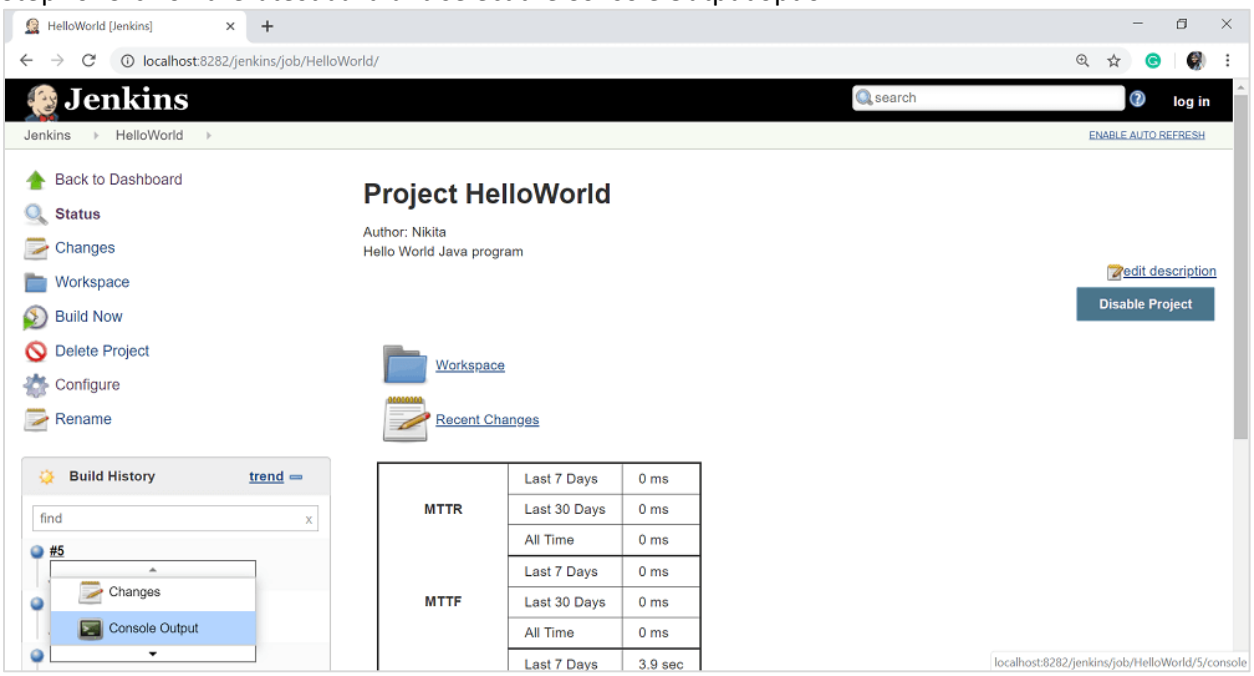
Step 8: In the Projects to build option, enter the "demo" as the project name to build. You can leave the other option as the default. Click on Apply then the Save button.



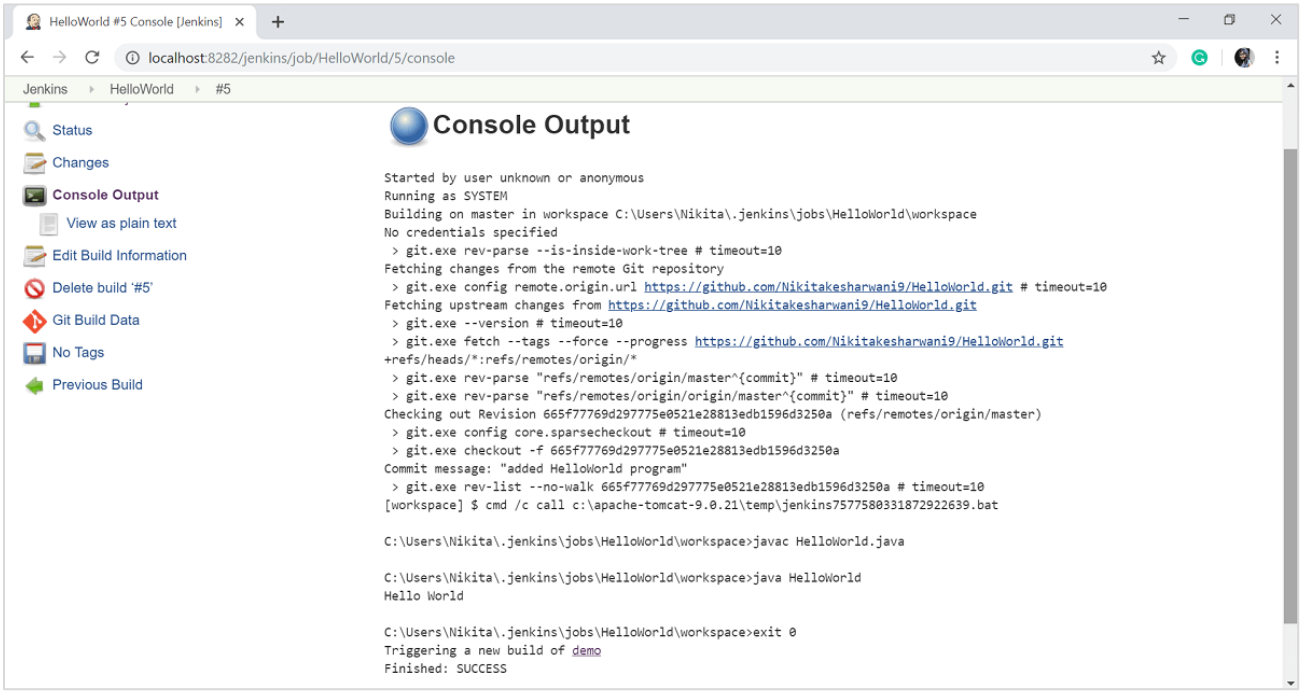
Step 9: Now, build the HelloWorld project. To do that, click on the Build Now option.



Step 10: Click on the latest build and select the Console Output option.



Now, if you see the Console output, you will also see that after the HelloWorld project is successfully built, the build of the demo project will also happen.



Jenkins - Distributed Builds

If you have larger and heavier projects which get built on a regular basis and running all of these builds on a central machine may not be the best option. In such case, you can configure other Jenkins machines to be slave machines to take the load off the master Jenkins server.

Sometimes you might also need several different environments to test your builds, in this scenario using a slave to represent each of your required environments is good idea.

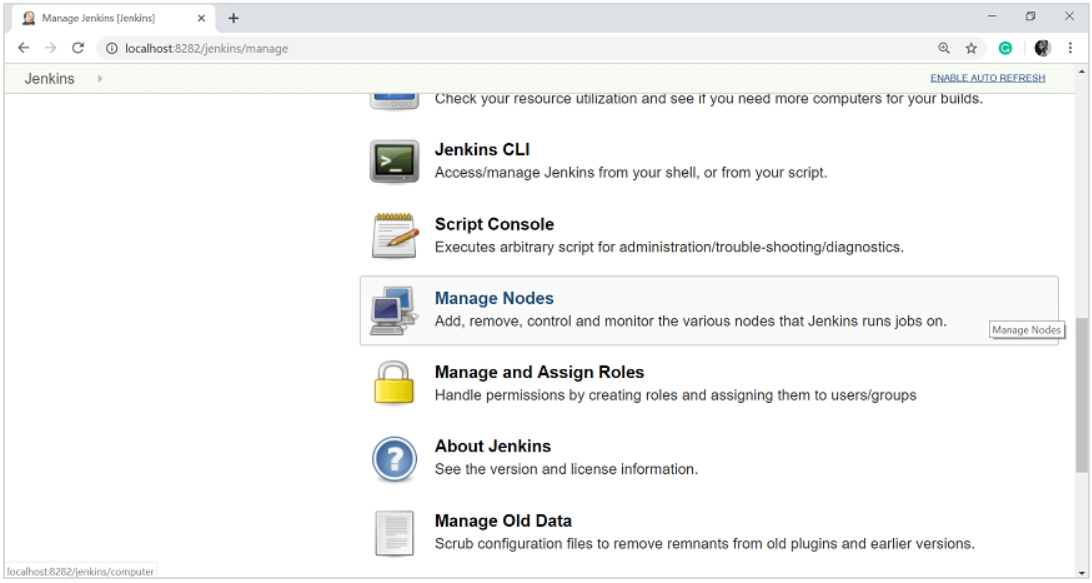
The master-slave architecture of Jenkins is used for distributed build environments, where the workload of building projects is distributed to multiple agent nodes or slaves. We can also different environments for each build.

Since each slave runs a separate program called a slave agent, there is no require to install the full Jenkins (package or compiled binaries) on a slave. There are a variety of ways to start slave agents, but at the end of the slave agent a Jenkins master needs to establish a bi-directional communication link (for example a TCP/IP socket) in order to operate.

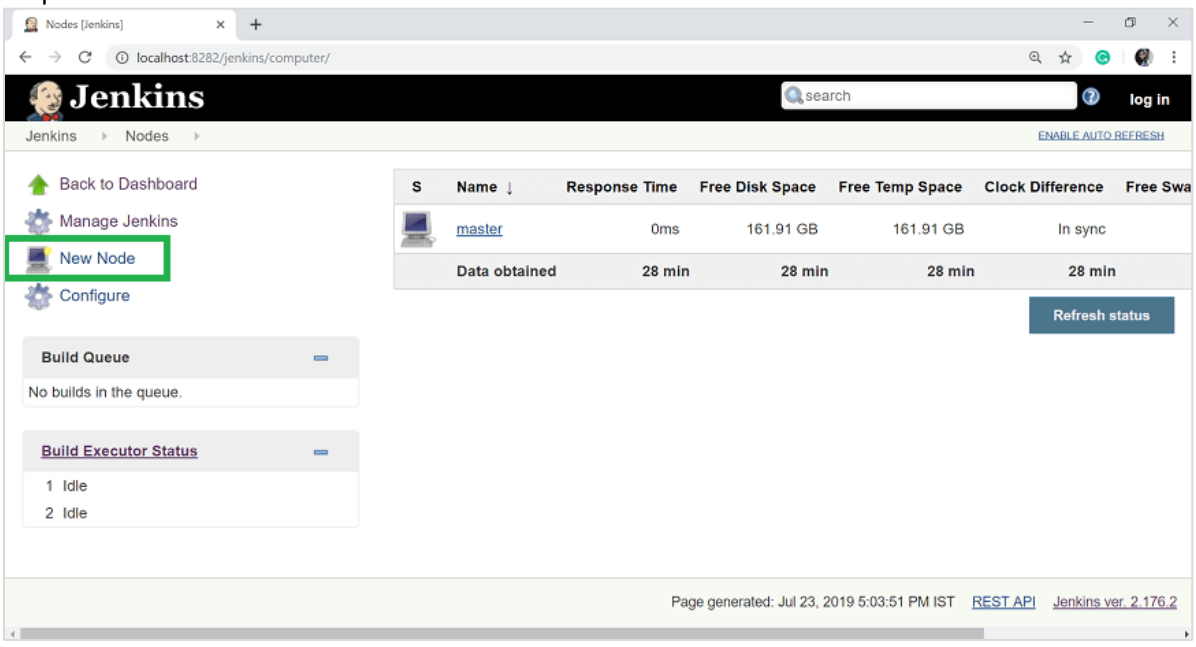
To set up the slaves/nodes in Jenkins, use the following steps:

Step 1: Go to Manage Jenkins.

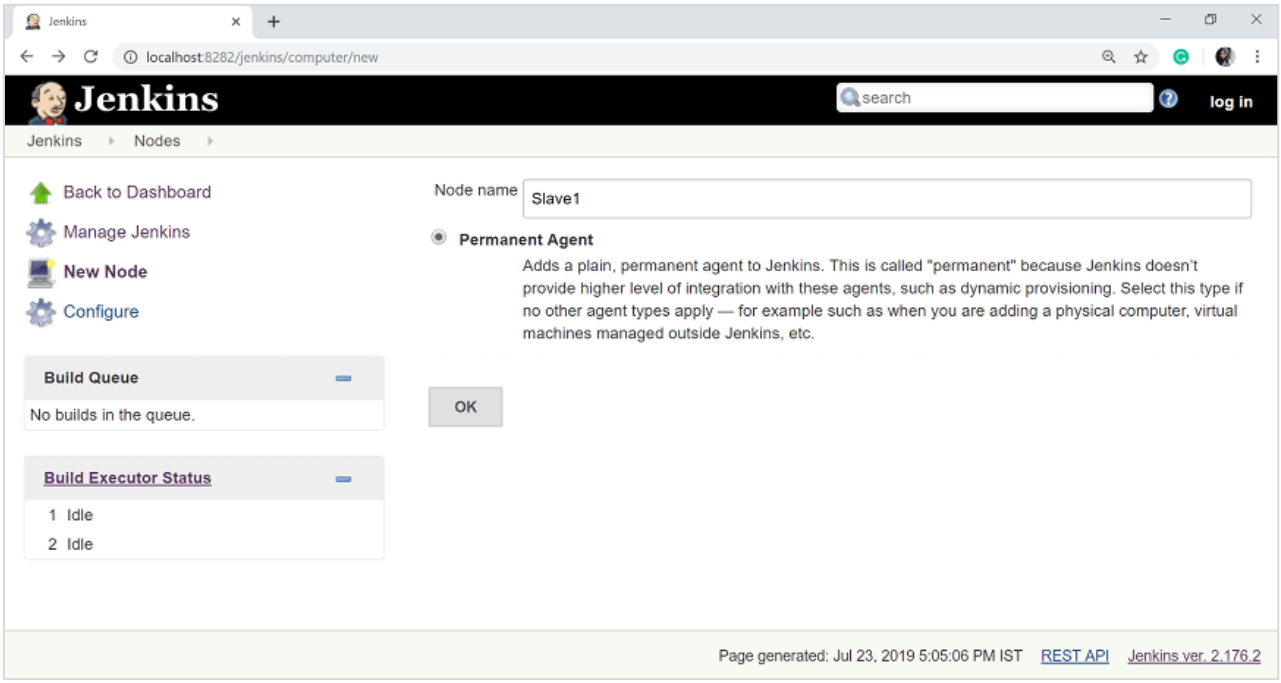
Step 2: Scroll down and select Manage Nodes.



Step 3: Click on **New Node**.

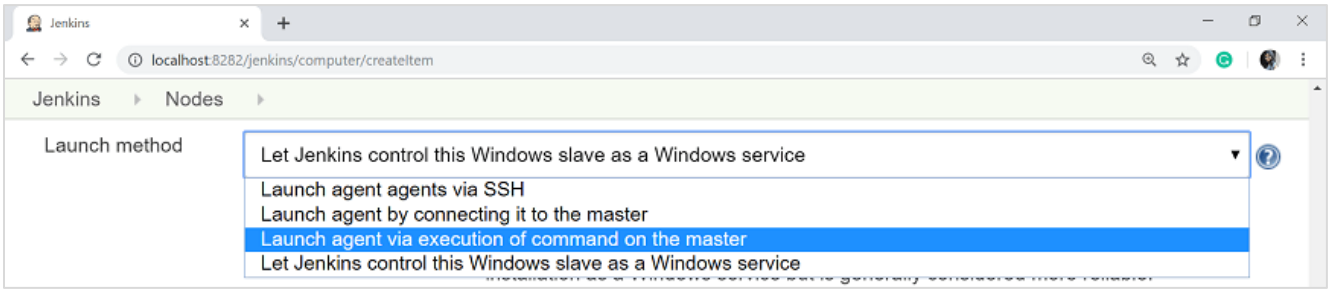


Step 4: Give a name for the new node, choose Permanent Agent option and then click OK.



Step 5: Enter the other details of the new node:

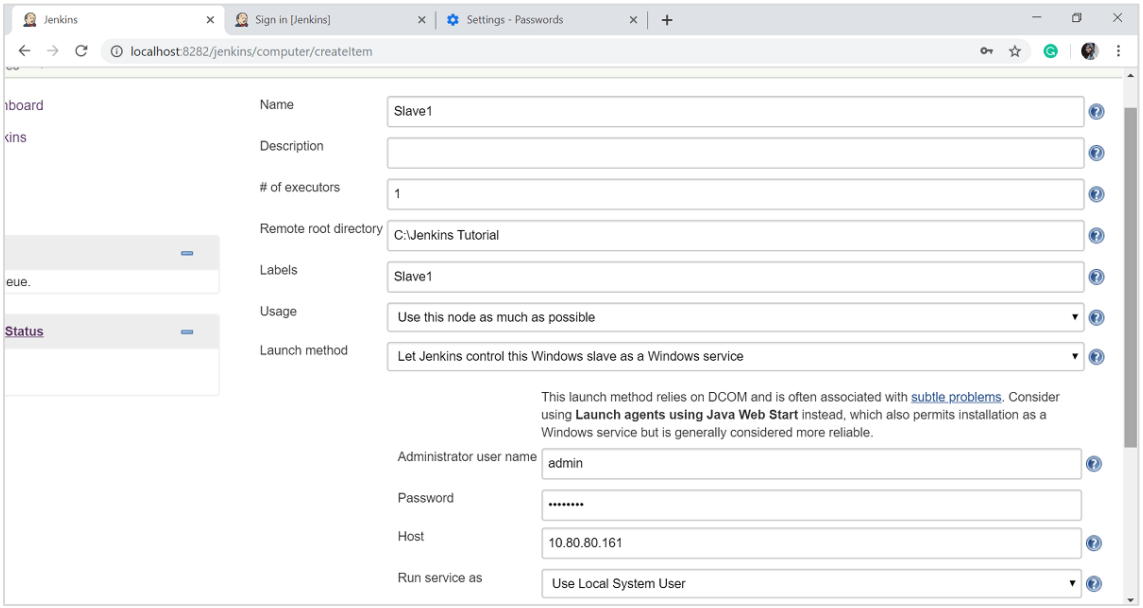
- Remote root directory: Path of the root directory
- Labels: give any label
- Usage: select use this node as much as possible
- Launch Method: There are following four methods –



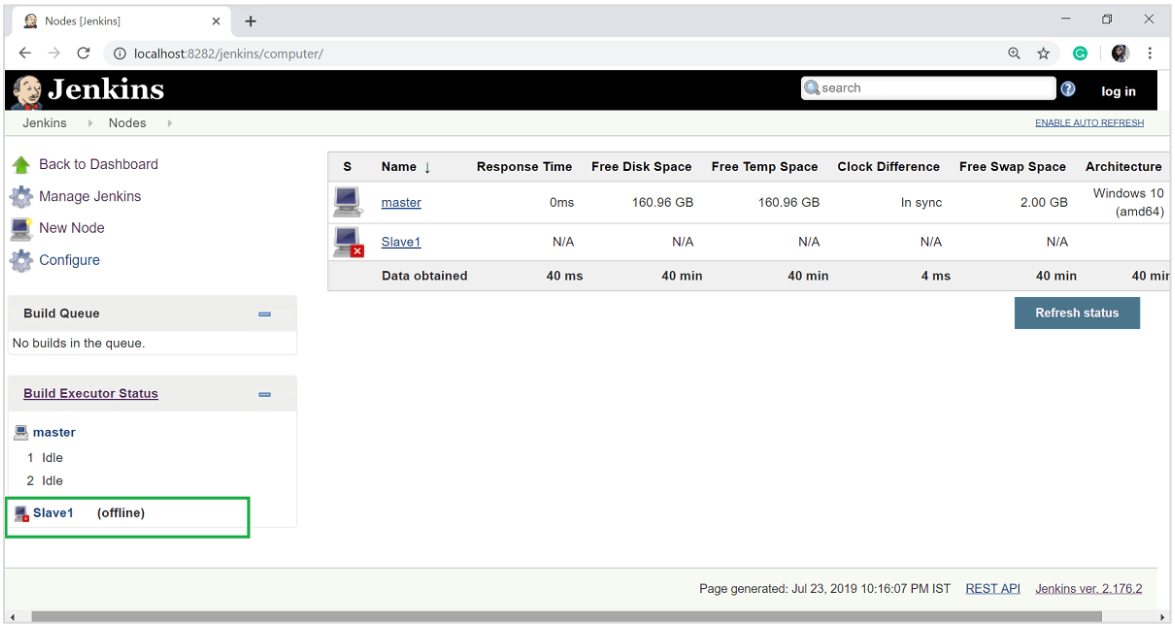
Here, I will use "Let Jenkins Control this Windows slave as a Windows service". When we select this option, you need to enter the following information:

- Administrator user name: Enter the user name of the node machine
- Password: Enter the password of the node machine

- Host: Enter the host IP
- Run service as



Step 6: Once you entered the above information, the new node machine will initially be in an offline state, but will online if all the settings in the previous screen were entered correctly.



SonarQube

An Introduction on SonarQube

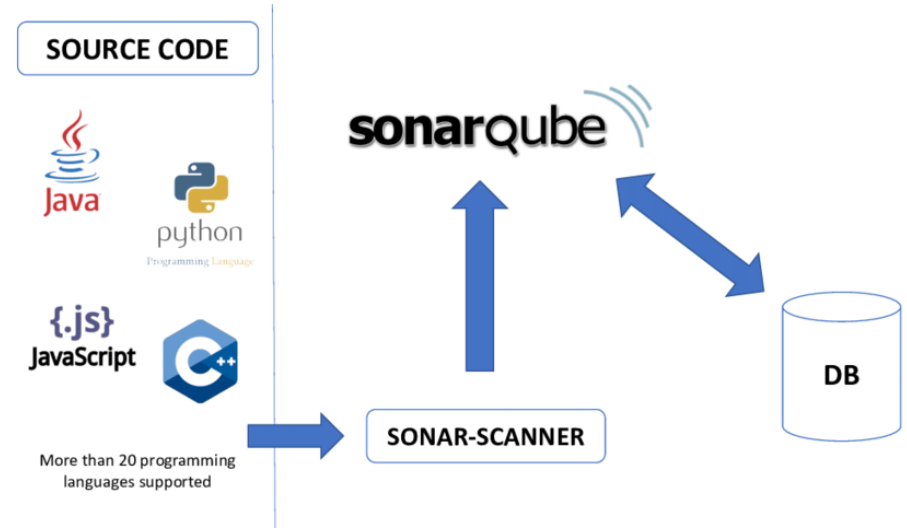
SonarQube is a Code Quality Assurance tool that collects and analyzes source code, and provides reports for the code quality of your project.

It combines static and dynamic analysis tools and enables quality to be measured continually over time. Everythin g from minor styling choices, to design errors are inspected and evaluated by SonarQube.

This provides users with a rich searchable history of the code to analyze where the code is messing up and determine whether or not it is styling issues, code defeats, code duplication, lack of test coverage, or excessively complex code.

The software will analyze source code from different aspects and drills down the code layer by layer, moving module level down to the class level, with each level producing metric values and statistics that should reveal problematic areas in the source code that needs improvement.

Sonarqube also ensures code reliability, Application security, and reducestechnical debt by making your code base clean and maintainable. Sonarqube also provides support for 27 different languages, including C, C++, Java, Javascript, PHP, GO, Python, and much more.SonarQube also provides Ci/CD integration, and gives feedback during code review with branch analysis and pull request decoration.



Why SonarQube?

- CI tools do not have a plugin which would make all of these tools work easily together
- CI tools do not have plugins to provide nice drill-down features that SonarQube has
- CI Plugins does not talk about overall compliance value
- CI plugins do not provide managerial perspective
- There is no CI plugin for Design or Architectural issues
- CI plugins do not provide a dashboard for overall project quality

Features of SonarQube are:

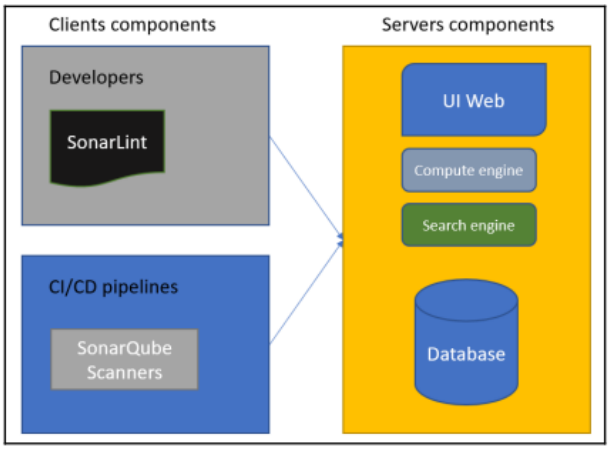
- Doesn't just show you what's wrong, but also offers quality and management tools to actively helps you correct issues
- Focuses on more than just bugs and complexity and offers more features to help the programmers write code, such as coding rules, test coverage, de-duplications, API documentation, and code complexity all within a dashboard
- Gives a moment-in-time snapshot of your code quality today, as well as trends of past and potentially future quality indicators. Also provides metrics to help you make the right decisions

Overview of the SonarQube architecture

SonarQube is a client-server tool, which means that its architecture is composed of artifacts

on the server side and also on the client side.

A simplified SonarQube architecture is shown in the following diagram:



Let's look at the components that are shown in this preceding diagram. The components

that make up SonarQube on the server side are as follows:

- A SQL Server, MySQL, Oracle, or PostgreSQL database that contains all the analysis data.
- A web application that displays dashboards.
- The compute engine, which is in charge of retrieving the analysis and processes.
- It puts them in the database.
- A search engine built with Elasticsearch.

The client-side components are as follows:

- The scanner, which scans the source code of the applications and sends the data to the compute engine.
- The scanner is usually installed on the build agents that are used to execute CI/CD pipelines.
- SonarLint is a tool that's installed on developers' workstations for real-time analysis.

How SonarQube helps to assess the quality of the code

SQALE approach is used in the SonarQube top-notch design, with specific changes. The SQALE technique, widely recognized, concentrates primarily on programming difficulty, maintenance and will not consider the program's hazards.

In essence, if a significant safety issue is found in a program, one needs to solve every stability, modifications, usability, and running strictly according to the SQALE approach and after turn again in the latest severe problem. In reality, if possible flaws in the program for such a lengthy period and gets zero consumer error complaints, it is far more critical to concentrate on remediating the latest deficiencies.

Taking it into consideration, programmers in SonarQube have changed the SQALE quality model, focusing on these key issues:

- As easy as conceivable, the system architecture must be
- Bugs and weaknesses in maintenance problems must not be missed.
- Significant development problems and privacy flaws should result in the quality gate criteria not being fulfilled.
- Code maintenance concerns are also crucial, so they shouldn't be overlooked.
- The financial plan is vital and must be conducted utilizing the SQALE analysis model.

The SonarQube Quality Gate ethics employs the feature vectors to evaluate when the program completed these controls:

- No latest bugs

- Zero latest security flaws
- New code technical debt ratio<= 5%
- Equal or more than 80 percent modified system's availability

Sonar squad must list seven terrible programmer's faults for growing technical debt:

- Bugs and possible errors
- Coding principles breached
- Redundancy of the program
- Inadequate coverage of consumers modules
- Lack of structure diffusion
- Pattern of Spaghetti
- Excessive amounts of comments

How to Use SonarQube Tool for Code Quality:

Step 1: Download and Unzip SonarQube

Prerequisites: Java (Oracle JRE11 or OpenJDK 11 minimum)

SonarQube comes in four different editions, including paid ones, however, for the purpose of this article, we will be using their free open-source community edition.

- Click the link https://www.sonarqube.org/downloads/ to download SonarQube Community Edition from their official Downloads page.
- After your download has completed, you should find a zip file waiting for you. On Windows, right click and select unzip and double click and extract on MacOS using the Archive Utility. Also, for Linux, you can unzip it using the `unzip` terminal tool.

```
unzip xyz.zip
```

Step 2: Run the SonarQube local server

At this point, you should be greeted with a little `.bat` file (if you are on windows) or a `.sh` file (if you are on Linux or Mac). The next step will be to execute this file to proceed into the console.

On Windows, navigate into the parent directory containing the `StartSonar.bat` file. This can be done by navigating to the unzipped folder ('sonarqube') and into the bin and windows-x86-xx folder as:

```
>> cd C:\sonarqube\bin\windows-x86-xx\
```

Then, run the following command to start the console:

```
>> StartSonar.bat
```

On Mac or Linux, run the following command from your shell terminal:

```
/opt/sonarqube/bin/[your_OS_name]/sonar.sh console
```

That's it! You've successfully started the SonarQube local server on your device. You can go to the URL `http://localhost:9000` (9000 is the default port, which can be changed later), and log in using the default credentials:

Step 3: Start a new SonarQube project

Now that you have successfully run your SonarQube server, you can proceed to start your first code project. Click on the '+' icon on the top right of the navigation bar and select 'Create New Project.'

In the next page, enter a unique project key and a short and suitable display Name and click on Set Up.

SonarQube uses tokens to identify you when an analysis is performed. So on the next page, generate a token using any word pair you like (like secret token or my_token).

After clicking on Generate, Sonar will provide you with a unique alphanumeric secret key. Copy it down, we will need it later for verification purposes. Then click Continue.

In the next window, select your project's main language. SonarQube supports over 27 languages including JavaScript, Python, C#, etc, so there's a high chance your project language will be included here.

Next, click on your OS type (Windows/Linux/macOS) and click on Download. It will link you to a download of SonarScanner. After the download completes, extract the file as in step 2.

Step 4: Setup Project properties and SonarScanner

This step should be pretty straightforward if you have spent any time working on code projects.

Register the

```
…\sonar-scanner-cli-<version>-<OS>\sonar-scanner-<version>-<OS>\bin
```

directory in your environment variable list. This can be done by adding an entry in the `Add Environment Variables` window for Windows.

On macOS or Linux distros, edit the required Path file to add the bin folder to the system's environment variable files.

Now we will need to set up a `properties` file for our sonar projects. This is important and we highly recommend that you don't overlook this step.

On your project folder, create a file named sonar-project.properties and copy-paste the following property variables into the file:

```
sonar.projectKey=<YOUR_PROJECT_KEY>
sonar.projectName=<PROJECT_DISPLAY_NAME>
sonar.login = <GENERATED_KEY>
sonar.scm.provider =<SCM_PROVIDER>
sonar.projectVersion=1.0
sonar.sources=src
sonar.exclusions=<EXCLUSION_DIRECTORIES(WONT_BE_SCANNED)>
sonar.ts.tslint.configPath=<CONFIG_FILE_.JSON>
sonar.typescript.lcov.report
Paths=<PATH_FOR_REPORT>
```

Here we have created our .properties file for our own project, but depending on your environment and path structure, the variables for the properties file will change. Look up the Sonar documentation for property variables related to your project language. Remember that you need to include a sonar-project.properties file on the root folder of every project that you create.

Now open a new cmd shell/terminal on your device and start up the SonarScanner to scan for code quality and security issues.

On Windows, type:

```
>> sonar-scanner.bat
```

Similarly, run the sonar-scanner shell file using the terminal if you're on macOS or Linux. If you have followed all the steps outlined in this tutorial, the scanner should start scanning without any problems.

Scanning typically takes a while, depending on the size of the project. Take a moment to relax in the meanwhile!

Step 5: View your analysis report on Sonar Dashboard

After the scan has completed through your code-base, go back to your SonarQube dashboard on http://localhost:9000 and log in using your credentials. Select your created project and you will find a code analysis report waiting for you.

The free Community Edition should include Reliability (measured in # of bugs), Security (in terms of vulnerabilities), Maintainability (depending on your code debt and smells), Coverage, and Duplications. Depending on the number and intensity of each check, the Quality Gate will either 'Pass' or 'Fail' the project. A green color grade will show areas where your code performed well while yellow and red color grades will highlight problem areas.

Furthermore, under the `Issues` tab, you can see the issues in your code, sorted according to intensity. Fixing these will help you and your team increase your quality score.

Give yourself a pat on the back, you have successfully completed a code analysis using the Sonar code quality tool! Now, you can export this report to include it in your presentations or forward it to team managers or other concerned parties.

Case Study

Once you are logged in, we're required to create a token by specifying a name – which can be our username or any other name of choice and click on the generate button.

We'll use the token later at the point of analyzing our project(s). We also need to select the primary language (Java) and the build technology of the project (Maven).

Let's define the plugin in the pom.xml:

```
<build>
  <pluginManagement>
    <plugins>
      <plugin>
        <groupId>org.sonarsource.scanner.maven</groupId>
        <artifactId>sonar-maven-plugin</artifactId>
        <version>3.4.0.905</version>
      </plugin>
    </plugins>
  </pluginManagement>
</build>
```

The latest version of the plugin is available here. Now, we need to execute this command from the root of our project directory to scan it:

```
mvn sonar:sonar -Dsonar.host.url=http://localhost:9000
 -Dsonar.login=the-generated-token
```

We need to replace the-generated-token with the token from above.

We specified the host URL of the SonarQube server and the login (generated token) as parameters for the Maven plugin.

After executing the command, the results will be available on the Projects dashboard – at http://localhost:9000.

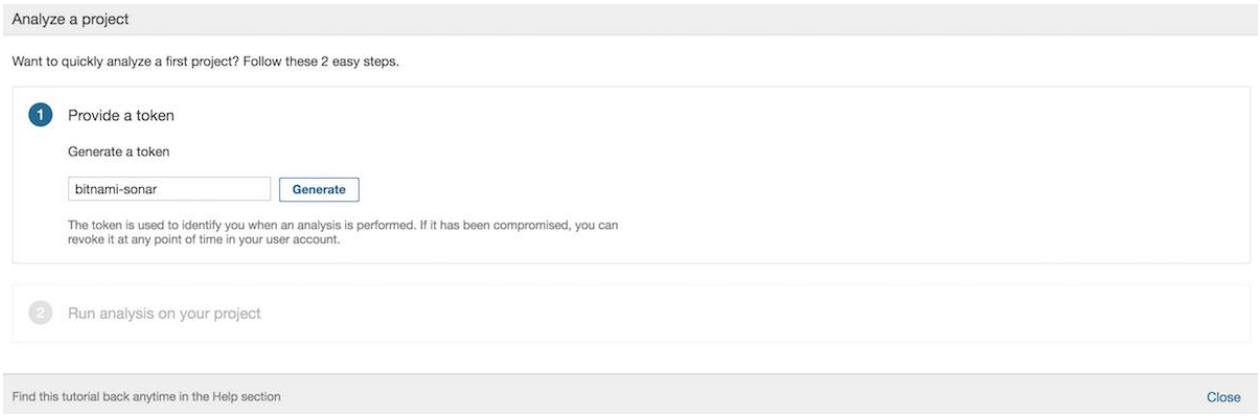
There are other parameters that we can pass to the Maven plugin or even set from the web interface; sonar.host.url, sonar.projectKey, and sonar.sources are mandatory while others are optional.

Other analysis-parameters and their default values are here. Also, note that each language-plugin has rules for analyzing compatible source code.

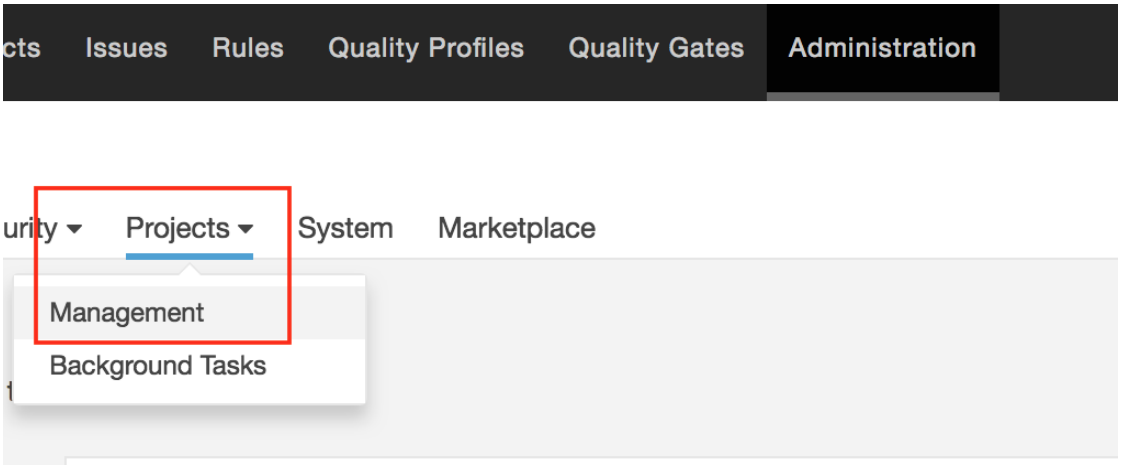
Step 1: Create A Project in SonarQube

The first step is to log in to SonarQube to create a new project and then to start analyzing your code. This tutorial provides you with a sample of code that you can use to follow the instructions below. You can try SonarQube using your own application code; just skip those steps that apply to the Bitnami sample project.

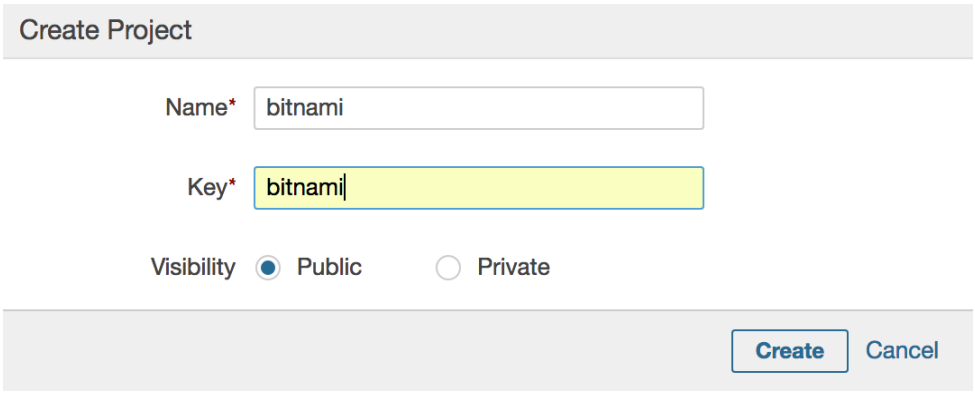
- To create a project in SonarQube:
- Log in to SonarQube.
- Generate an authentication token. Save the resulting token in a safe place. You will need it later to connect your project to SonarQube.



- Navigate to the "Administration -> Projects -> Management" section.



- In the "Projects Management" screen, click "Create Project".
- Enter a name for your project and a key. Select the visibility of the project. Click "Create" to finish the process.



Step 2: Install and Configure The SonarQube-Scanner Client

The next step consists of configuring SonarQube to start analyzing projects. To do so, you need to have git and the sonarqube-scanner client installed and configured, as well as a code repository to scan on hand.

- Log in to the server console. Learn how to connect to the server through SSH.
- | TIP: The instructions below are specific for Linux OS distributions. Follow the SonarQube official documentation to learn how to install and configure the sonar-scanner client on Windows or macOS.
- Install git. Execute the install command as root:

```
sudo su
apt-get install git
```

- Clone the sample code repository. (Skip this step if you are using your own code to test SonarQube).

```
git clone https://github.com/bitnami/sample-mean.git
```

- Download and install the latest version of the sonarqube-scanner client. Remember to replace the X.Y.Z placeholder with the corresponding version.

```
wget https://sonarsource.bintray.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-X.Y.Z-linux.zip
unzip sonar-scanner-cli-X.Y.Z-linux.zip
```

- Edit the sonar-scanner-X.Y.Z-linux/conf/sonar-scanner.properties file in order to configure project analysis. Enter the following parameters:
- sonar.host.url: IP address of your SonarQube server instance.
- sonar.projectKey: key name you gave when you created the project.
- sonar.sources: path to the project to analyze.
- sonar.login: SonarQube authentication token.

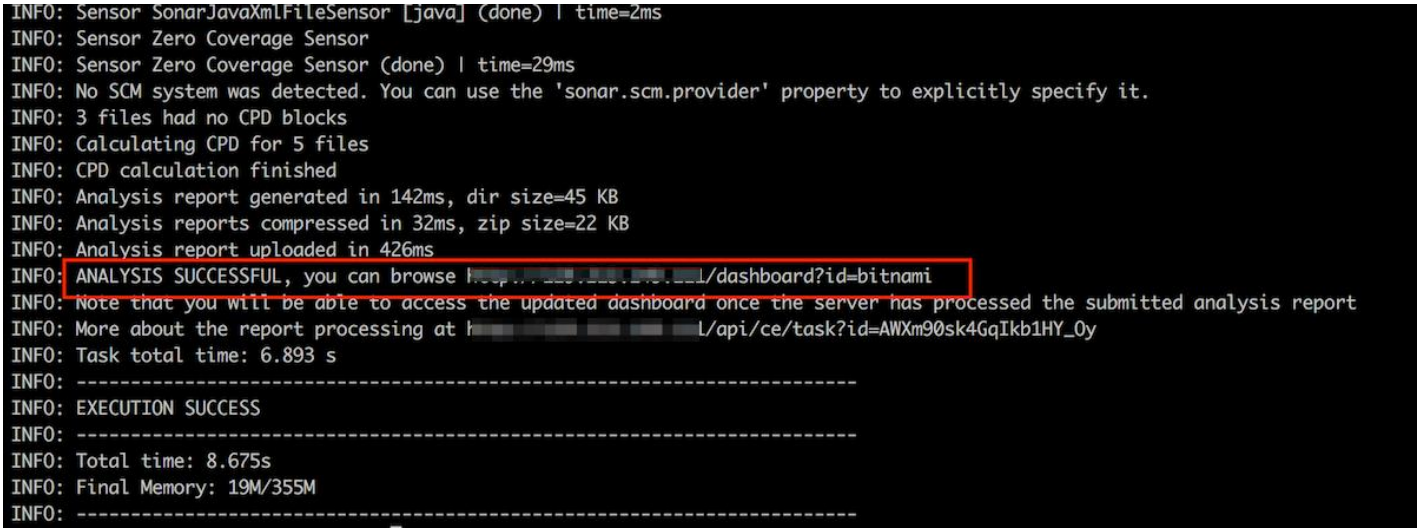
Step 3: Analyze t he Code With SonarQube And Fix Issues And Bugs

Let's see how SonarQube works by running a project test using the example provided. To do so:

- Run the sonar-scanner command to start the analysis.

```
sonar-scanner-X.Y.Z-linux/bin/sonar-scanner
```

Once it finishes, you will see in the log's information that the analysis was successfully executed. Find there the URL where the analysis results are published.



To see the results of the analysis, navigate to the SonarQube UI or enter the above-mentioned URL in your browser. The "Overview" dashboard displays the following information:

- Bugs and vulnerabilities.
- Code smells and the time you will spend fixing these errors.
- The average of your code that has been covered running the test.
- Percentage of duplications and the number of duplicated blocks were found in your code.

As seen in the image below, our sample code has one code smell that will take us two minutes to solve.