

## DevOps?

The word “DevOps” was coined in 2009 by Patrick Debois, who became one of its gurus. The term was formed by combining “development” and “operations,” which provides a starting point for understanding exactly what people typically mean when they say “DevOps.” Notably, DevOps isn’t a process or a technology or a standard. Many devotees refer to DevOps as a “culture”—a viewpoint that New Relic favors. We also use the term “DevOps movement” when talking about topics such as adoption rates and trends for the future, and “DevOps environment” to refer to an IT organization that has adopted a DevOps culture.

*“DevOps represents a change in IT culture, focusing on rapid IT service delivery through the adoption of agile, lean practices in the context of a system-oriented approach. DevOps emphasizes people (and culture), and seeks to improve collaboration between operations and development teams. DevOps implementations utilize technology— especially automation tools that can leverage an increasingly programmable and dynamic infrastructure from a life cycle perspective.”*

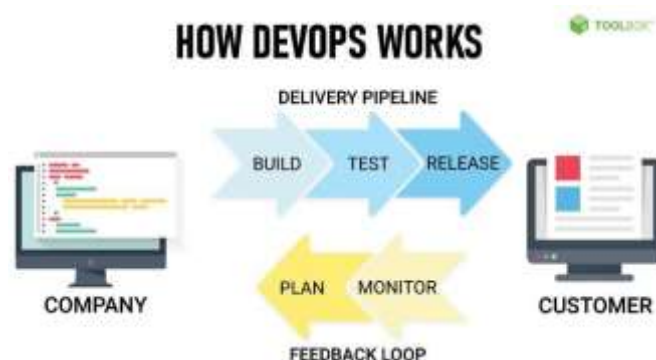
DevOps is defined as a combination of processes and tools created to facilitate organizations in delivering services and applications much faster than they can through conventional software development processes. It helps increase customers’ confidence in the applications that an organization offers, thereby allowing the company to flourish and achieve its business goals faster.



Under the DevOps model, development and operations teams work in constant cohesion throughout the entire project lifecycle, starting right from development to deployment. When security is the main focus, the quality assurance team is tightly knitted with the DevOps team throughout the app lifecycle. In this situation, some DevOps teams are also referred to as **DevSecOps**. Close coordination with the QA team ensures that no loopholes are left unchecked in the provided service/app.

## How DevOps Works?

A DevOps process can be summarized as an infinite loop that comprises the following stages — build, test, and release through the delivery pipeline and plan and monitor through feedback, which resets the loop again. With such an amazing combination, teams use tech stack and tooling that assists them in reliably developing apps. Moreover, going away from the norm, teams use automated processes here. DevOps tools also allow engineers to complete different tasks independently. Be it provisioning infrastructure or deploying code, they can accomplish these tasks without being dependent on one another. As such, the DevOps model accelerates the overall application development process.



## **Key Goals and Benefits of DevOps**

### **Goals of DevOps**

The fast-paced growth of the IT industry and continuous advancements in technology make it critical to set DevOps goals that are experimental and challenging for companies to compete and thrive in the market. Here are the key goals and principles that every successful DevOps program has in common.

1. Ensures effective collaboration between teams: Effective collaboration in any process relies on shared ownership. During the development process, all those involved should embrace the fact that everyone is equally responsible for the entire development process. Whether it is development, testing, or deployment, each team member should be involved. They should understand that they have an equal stake in the final outcome. In the DevOps paradigm, passing of work from one team to another is completely defined and broken down. This accelerates the entire process of development since collaboration between all the teams involved is streamlined.
2. Creates scalable infrastructure platforms: The primary focus of DevOps is to create a sustainable infrastructure for applications that make them highly scalable. According to the demands of the modern-day business world, scalable apps have become an absolute necessity. In an ideal situation, the process of scaling should be reliable and fully automated. As a result, the app will have the ability to adapt to any situation when a marketing effort goes viral. With the app being scalable, it can adjust itself to large traffic volumes and provide an immaculate user experience.
3. Builds on-demand release capabilities: Companies must focus on keeping their software in a 'releasable' state. Continuous delivery will allow the software to add new features and go live at any stage. DevOps aims to automate the process of release management because it has a plethora of advantages. Automated release management is predictable, fast, and very consistent. Moreover, through automation, companies can release new versions as per their requirements. Automated release management also has complete and thorough audit trails, as these are essential for compliance purposes.
4. Provides faster feedback: Automating monotonous tasks such as testing and reporting will accelerate the process of rapid feedback. Since the development team will know what has to change, it can roll out the updated version faster. In addition, the team can better understand the impact of the changes that it has done in the software lifecycle. A concrete understanding of changes will assist team members in working efficiently in tandem. With rapid feedback, the operations team and developers can make better decisions collectively and enhance the app's performance.

### **Benefits of DevOps**

DevOps helps organizations deliver added value to their customers. Here are some compelling benefits of DevOps.

1. Smarter work and faster release: With DevOps, your development team can release the required deliverables quickly. Faster release of deliverables will keep you miles ahead of your competitors, which is very important in today's cut-throat business realm. Businesses should understand that if their review cycle is not automated, it will slow down the release process. Moreover, the inclusion of disparate tools will lead to context switching and higher costs. Thus, DevOps can help rectify this worrisome business situation.
2. Quick resolution of issues: In a business world where speed and accuracy are paramount, a fast feedback loop will help you thrive. With DevOps, the communication process becomes seamless, and, as such, it minimizes the time required to solve issues. Without open communication, key issues can slip out of mind, which will have serious repercussions in the long run. DevOps fosters open

communication that helps resolve issues, thus unblocking the release pipeline faster.

3. Better collaboration between teams: DevOps paves the way for more dynamic and round-the-clock communication between teams. It renders an environment for mutual collaboration and integration among teams that are distributed globally. Eliminating the traditional departmental barriers between teams forms a new sense of ownership, wherein each team member feels equally responsible for meeting delivery timelines. This collaboration contributes to happier and more engaged employees.
4. Fostering innovative mindsets: With DevOps, deployment phases of the application are more relaxed as compared to traditional methods. This is because it streamlines the entire process, ensures that there are no lapses in quality, and allows on-time and efficient release. Thus, as everything is in order, the development team is more at peace. This allows it to think out of the box and provide additional value to the user. Having a development team with an innovative mindset is a boon for any business organization. An innovative approach, in itself, has immense scope and leads to better quality and resolution of issues at hand. Thus, through DevOps, the process of expanding the horizon of an app becomes much easier.
5. Faster threat detection: Automated and continuous testing of the code will make the process of threat detection faster. As developers can locate problem areas at an early stage, they can then resolve them faster. Thus, DevOps is a vital cog in maintaining and enhancing the quality and performance of an app. As the overall build of the app is in capable hands, teams working together are empowered to share feedback as and when necessary.
6. Increased customer satisfaction: Customer satisfaction is paramount in any day and age, irrespective of the business one is involved in. DevOps is known for enhancing customer experience, which ultimately increases the level of customer satisfaction. Dissatisfied customers are never a good sign for any business. Feedback loops are an important component of DevOps. These loops empower end users to track the progress of app development at various stages.
7. In addition, they can suggest changes (if any) or give their inputs to make the app more customer-centric. Due to their dynamic nature, feedback loops help developers and customers remain on the same page. Moreover, DevOps accelerates the process of app development, which eventually lessens the delivery timer. This has a positive impact on the customer satisfaction ratio.
8. Providing the much-needed edge: Along with staying true to their development process, companies need to ensure that they sustain themselves in the cut-throat competition. Implementing DevOps can be your trump card to provide your organization with that much-needed edge. Competitive advantage is necessary, as it can become the deciding factor in the popularity of an application in many cases. Some factors set expert businesses apart from mediocre ones:
  - Top-quality features
  - Quicker and timely software releases
  - Maximizing return on investments
  - Listening to constructive feedback

### **Difference between Agile and DevOps**

**Agile:** Agile program advancement comprises different approaches to computer program improvement beneath which prerequisites and arrangements advance through the collaborative exertion of self-organizing and cross-functional groups and their

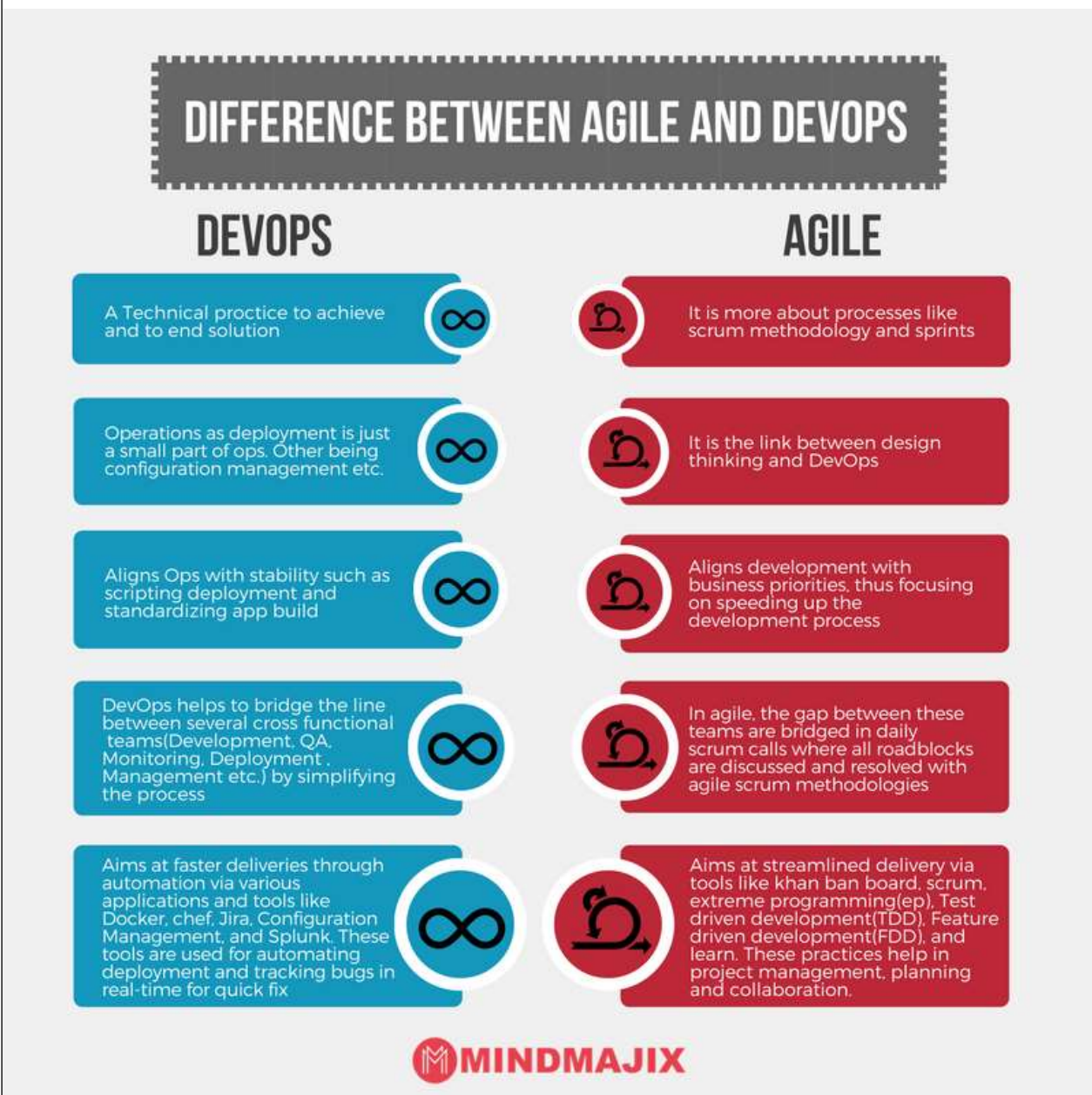
customer/end client.

**DevOps:** DevOps could be a set of hones that combines program improvement and information-technology operations which points to abbreviating the framework's advancement life cycle and giving nonstop conveyance with tall program quality.

**Example:** Facebook's mobile app which is updated every two weeks effectively tells users you can have what you want and you can have it. Now ever wondered how Facebook was able to do social smoothing? It's the DevOps philosophy that helps facebook and sure that apps aren't outdated and that users get the best experience on Facebook. Facebook accomplishes this true code ownership model that makes its developers responsible that includes testing and supporting through production and delivery for each kernel of code.

They write and update their true policies like this but Facebook has developed a DevOps culture and has successfully accelerated its development lifecycle.

### Difference between Agile and DevOps:





# DevOps Tools

## 1. Git (GitLab, GitHub, Bitbucket)

Git remains indispensable in software development and DevOps due to its pivotal role in version control, collaborative coding, and efficient project management. As technology has accelerated, the need for streamlined and organized code management has never been greater.

Git empowers developers to collaborate on codebases, effortlessly creating and merging branches for new features and bug fixes. Its distributed nature ensures developers can work seamlessly offline, an increasingly valuable feature in today's remote and distributed work environments.

Additionally, Git facilitates the tracking of code modifications, making it easier to identify when and why specific changes were made, a critical aspect of maintaining code quality and security. Software development is essential in driving innovation and advancing progress, and Git maintains its prominent position as the bedrock of efficient, cooperative, and secure coding methodologies.

## 2. Maven

Due to its enduring significance in managing project dependencies, building, and project lifecycle management, Maven remains a pivotal tool in SD and DevOps. As a robust build automation and project management tool, Maven simplifies the complexities of Java-based project development by streamlining the compilation, testing, packaging, and distribution processes. It ensures consistent and reproducible builds, making it easier for development teams to collaborate efficiently and deliver high-quality software.

Maven's role in managing dependencies and facilitating continuous integration and deployment remains crucial. Its ability to handle complex build scenarios and integrate seamlessly with modern DevOps practices makes it indispensable for ensuring software projects' reliability, maintainability, and scalability in 2024 and beyond.

## 3. Jenkins

Its importance lies in its role as a powerful automation server that enables continuous integration and continuous delivery (CI/CD) pipelines. Jenkins streamlines software development by automating tasks such as building, testing, and deploying code changes, ensuring that software is delivered quickly and highly. With the growing complexity of modern applications, the need for efficient CI/CD processes has become even more paramount.

Jenkins provides flexibility, extensibility, and a vast library of plugins that cater to a wide range of technologies and tools, making it adaptable to diverse development environments. As organizations prioritize speed, reliability, and collaboration in their software development practices, Jenkins stands as a cornerstone tool, enabling teams to achieve seamless automation and efficient delivery of software solutions.

## 4. Chef

Chef, a powerful automation platform, is crucial in managing infrastructure as code. Chef empowers organizations to achieve scalability, reliability, and speed seamlessly. By allowing the automation of server provisioning, configuration, and maintenance, Chef enhances efficiency and consistency across the entire infrastructure, reducing manual errors and ensuring that infrastructure remains desired.

Moreover, Chef integrates smoothly with various cloud providers, containerization technologies, and other DevOps tools, making it adaptable to the ever-evolving tech landscape. As organizations prioritize agility and scalability, Chef remains a vital tool

in automating complex infrastructure tasks and enabling DevOps teams to focus on innovation and delivery.

## **5. Puppet**

Puppet is essential because it simplifies the management and orchestration of complex IT infrastructures by allowing administrators to define infrastructure as code. It ensures consistency and repeatability in configuration across servers, cloud instances, and containers. Businesses increasingly rely on diverse, dynamic, and hybrid infrastructures.

Puppet's importance lies in its ability to streamline provisioning, configuration, and continuous compliance, thus reducing operational complexity, minimizing errors, and accelerating software delivery. Puppet continues to empower organizations to efficiently manage and scale their infrastructure while maintaining high levels of security and compliance, making it a crucial tool for DevOps teams.

## **6. Ansible**

Ansible is a powerful and widely adopted automation and configuration management tool important in 2024 for several reasons. This tool stands out for its simplicity and versatility. It empowers organizations to automate repetitive tasks, provisioning of infrastructure, and configuration management across diverse environments, making it an invaluable asset for DevOps and IT teams.

Furthermore, Ansible's agentless architecture, declarative language, and a vast library of pre-built modules make it accessible to both beginners and seasoned professionals. As organizations prioritize efficiency, scalability, and the rapid deployment of applications and services, Ansible remains an indispensable DevOps toolkit, helping teams streamline operations, enhance security, and maintain infrastructure at scale, all while reducing manual errors and increasing agility in a fast-paced technological landscape.

## **7. Docker**

Docker is crucial in modern software development and DevOps practices. It can simplify and streamline the management of applications across various environments. Docker containers encapsulate an app and its dependencies, ensuring consistent and reproducible deployments from development to production.

This technology enhances portability and scalability, accelerates development cycles, and reduces the "it works on my machine" problem. In a rapidly evolving software landscape, Docker's containerization approach remains crucial for achieving efficient, isolated, and highly flexible application deployment, making it an essential component of DevOps and continuous delivery pipelines.

## **8. Kubernetes**

Kubernetes, often abbreviated as K8s, play a central role in modern software development and operations. Its importance lies in its ability to orchestrate, manage, and automate containerized applications at scale. As organizations increasingly embrace microservices architectures and containerization for their applications, Kubernetes provides the essential infrastructure for deploying, scaling, and maintaining these containers efficiently.

The tool's resilience, self-healing capabilities, and support for hybrid and multi-cloud environments make it vital for achieving agility, reliability, and cost-effectiveness in application deployment. It serves as the backbone of cloud-native ecosystems, enabling organizations to accelerate software delivery, improve resource utilization, and respond effectively to the evolving demands of the digital landscape.

## 9. Slack

Slack is a crucial tool for businesses and organizations worldwide. Its significance lies in facilitating seamless communication and collaboration among teams, whether working in the same office or remotely. Slack's real-time messaging, file sharing, and integration capabilities streamline workflow, enhance productivity and keep teams connected across different time zones and locations.

As the work landscape evolves, with more companies embracing hybrid and remote work models, Slack is a vital hub for quick decision-making, project coordination, and knowledge sharing. With an ever-expanding ecosystem of integrations and features, Slack remains at the forefront of modern workplace communication, making it essential for businesses to stay agile, efficient, and competitive.

## 10. AWS Cloud Computing and Storage in DevOps

AWS (Amazon Web Services) Cloud Computing and Storage are crucial in DevOps because they provide scalable, flexible, and cost-effective infrastructure for DevOps practices. AWS offers many services, including compute resources, databases, container orchestration, and serverless computing, which align perfectly with modern software development and deployment demands.

Organizations adopt DevOps to accelerate software delivery. AWS provides the foundation for rapidly deploying and scaling applications, supporting continuous integration and continuous delivery (CI/CD) pipelines, and automating infrastructure provisioning through tools like AWS CloudFormation.

Furthermore, AWS's storage solutions enable efficient data management, backup, and recovery, ensuring the resilience and reliability required for DevOps operations. As cloud technology evolves, AWS remains at the forefront, enabling DevOps teams to focus on innovation and efficiency.

## 11. Azure Cloud Computing and Storage in DevOps

Azure Cloud Computing and Storage will be pivotal in DevOps practices in 2024 and beyond. Azure offers a comprehensive cloud ecosystem that enables organizations to scale their infrastructure, deploy applications, and store data efficiently. Azure provides essential services for continuous integration and continuous deployment (CI/CD), automation, monitoring, and security. Its cloud computing capabilities facilitate the provisioning of resources on demand, ensuring that development and testing environments are readily available.

Azure's storage solutions, including Azure Blob Storage, Azure Files, and Azure SQL Database, enable secure data storage and retrieval, supporting the data-driven aspects of DevOps. Besides, Azure's integration with DevOps tools like Azure DevOps Services streamlines the software development lifecycle, enhancing collaboration and automation.

## 12. GCP Cloud Computing and Storage in DevOps

Google Cloud Platform (GCP) offers robust cloud computing and storage solutions. GCP provides a scalable, reliable, and highly available infrastructure essential for modern DevOps practices. With its comprehensive set of services, including Google Compute Engine, Google Kubernetes Engine, Cloud Storage, and BigQuery, GCP empowers DevOps teams to build, deploy, and manage applications easily. Its emphasis on automation, infrastructure as code, and container orchestration aligns seamlessly with DevOps principles.

Moreover, GCP's cutting-edge technologies, such as AI and machine learning capabilities, provide DevOps practitioners with advanced tools for monitoring, analytics, and automation, making it a powerful choice for organizations seeking to optimize their software development

and delivery processes.

### 13. Monitoring, Alerting, and Incident Response Tools: SignalFx

Monitoring, alerting, and incident response tools like SignalFx are pivotal in DevOps and software development. As software systems become complex and distributed, the need for real-time visibility into performance and the ability to respond swiftly to incidents is significant. SignalFx excels in this regard by providing advanced monitoring and observability solutions that enable organizations to detect anomalies, trace issues across microservices proactively, and set up intelligent alerts.

As applications scale, cloud-native architectures become the norm, and user expectations for reliability grow, SignalFx's capabilities are crucial. It empowers DevOps teams to ensure high availability, optimize resource utilization, and maintain a seamless user experience by identifying and addressing performance issues before they impact end-users. It is one of the most essential tools for modern software operations.

### 14. Appdynamics

AppDynamics, a leading application performance management and monitoring platform, remains critically important as it ensures the optimal performance of modern digital businesses. As organizations rely on complex and distributed software systems, proactively monitoring, troubleshooting, and optimizing these applications becomes essential.

AppDynamics provides real-time visibility into application performance, allowing businesses to swiftly identify bottlenecks, latency issues, and errors.

With the ever-growing complexity of applications, the importance of AppDynamics lies in its ability to empower organizations to deliver exceptional user experiences, maintain application reliability, and swiftly respond to performance issues, thereby ensuring the continued success and competitiveness of digital businesses.

### 15. Raygun

It is a crucial tool in software development and DevOps because it ensures application reliability and performance. Raygun is an application monitoring and error-tracking platform that empowers development teams to identify, diagnose, and resolve real-time issues.

With software systems growing in complexity and the increased demand for seamless user experiences, Raygun's importance lies in providing actionable insights into application errors and performance bottlenecks. It enables organizations to proactively address issues, reduce downtime, and enhance user satisfaction, leading to higher software quality and improved customer experiences.

Software is central to businesses across industries. Raygun's role in maintaining application health and facilitating rapid issue resolution makes it a fundamental tool for DevOps professionals and software developers.

### 16. Splunk Cloud

Splunk Cloud helps organizations gain critical insights from the ever-expanding volume of data generated in today's digital landscape. As businesses increasingly rely on data-driven decision-making, Splunk Cloud stands out as a robust and scalable platform for monitoring, searching, analyzing, and visualizing machine-generated data. Its importance lies in providing real-time visibility into the health and performance of complex systems, applications, and infrastructures, enabling rapid incident detection and response.

As cybersecurity threats evolve, Splunk Cloud's advanced security analytics and threat detection capabilities remain indispensable for safeguarding against cyberattacks and ensuring data integrity. In a world where data is a strategic asset, Splunk Cloud's role in



harnessing the power of data for operational excellence and security cannot be overstated.

### 17. Selenium

It remains a vital tool in software testing and automation due to its enduring relevance in ensuring the quality of web applications. As technology evolves, web applications become increasingly complex, requiring thorough testing across various browsers and platforms.

With its robust automation capabilities and extensive browser compatibility, Selenium allows developers and QA teams to automate repetitive testing tasks efficiently, conduct cross-browser testing, and ensure that web applications function flawlessly across diverse environments.

Its open-source nature, active community support, and integration with other DevOps tools make Selenium a go-to choice for organizations striving for continuous delivery and the rapid deployment of high-quality software, a cornerstone of modern software development practices.

### 18. Gremlin

Gremlin is an essential tool in chaos engineering, which has become increasingly critical for ensuring the resilience and reliability of modern software systems. As technology advances and complex distributed systems become the norm, the potential for unexpected failures and outages also rises.

Gremlin allows organizations to proactively identify weaknesses and vulnerabilities in their infrastructure and applications by simulating controlled failures, such as network disruptions, service outages, and resource constraints.

By intentionally inducing chaos and monitoring the system's response, teams can uncover weaknesses before they lead to costly downtime or security breaches. Gremlin facilitates organizations to build more robust, fault-tolerant systems that can withstand real-world challenges and deliver uninterrupted services to users.

### 19. ServiceNow

ServiceNow is a vital platform for organizations seeking to streamline their IT service management and beyond. Its significance lies in its ability to provide a unified, cloud-based solution for automating and optimizing various business processes, including ITSM, ITOM, HR, customer service, and more.

Due to the rapid digitization of services, remote work, and the growing complexity of technology infrastructures, ServiceNow offers a comprehensive approach to managing workflows, resolving issues, and delivering services efficiently. Its intelligent automation capabilities, analytics, and AI-driven insights empower organizations to enhance productivity, agility, and customer satisfaction while reducing operational costs.

ServiceNow's role in orchestrating and integrating diverse systems and processes makes it an indispensable tool for driving digital transformation and ensuring smooth operations in the ever-evolving business landscape of 2024.

### 20. Status Service Updates: The Status Page

"Status Service Updates: The Status Page" is a critical tool for organizations and businesses of all sizes. In today's world, where online services and applications are integral to operations, ensuring their availability and reliability is essential. It provides real-time information to users and stakeholders about the operational status of services, applications, and infrastructure. The Status Page plays a crucial role in transparency, trust-building, and customer satisfaction by promptly communicating service disruptions, planned maintenance, and incident resolutions.

Downtime can often lead to significant financial losses and damage to a company's reputation, so having a practical Status Page becomes not just a convenience but a necessity. It allows organizations to showcase their commitment to transparency and responsiveness in addressing service-related issues, ultimately fostering stronger customer relationships and trust.

## **21. ELK (Elasticsearch, Logstash and Kibana)**

ELK, which stands for Elasticsearch, Logstash, and Kibana, continues to shine in DevOps and IT operations. This powerful trio of tools remains essential for organizations seeking effective log management, monitoring, and data visualization. Elasticsearch is a highly scalable and fast search engine that enables real-time data indexing and search.

Logstash facilitates the collection, processing, and transformation of log data from various sources, making it compatible with Elasticsearch. Kibana, on the other hand, provides a user-friendly interface for visualizing and analyzing data, offering customizable dashboards and powerful data exploration capabilities.

ELK's significance in 2024 lies in its ability to empower organizations with comprehensive insights into their systems, applications, and infrastructure. It ultimately facilitates quick problem resolution, proactive monitoring, and data-driven decision-making in an increasingly complex and fast-paced technological landscape.

## **22. GitLab CI/CD**

GitLab CI/CD's significance lies in its ability to automate the complete software delivery pipeline, from code changes to deployment, in a single integrated environment. GitLab CI/CD ensures rapid and reliable delivery of software updates. It enables continuous integration (CI) by automatically building and testing code changes, allowing teams to catch issues early in the development cycle.

Furthermore, the continuous deployment (CD) aspect automates the release and deployment process, reducing the risk of human errors and enabling organizations to deliver features and updates to users swiftly and confidently. GitLab CI/CD's importance is further accentuated as businesses seek to accelerate digital transformation efforts, respond rapidly to changing market demands, and maintain a competitive edge through efficient and automated software delivery practices.

## **23. Scripting**

Scripting remains vital due to its pivotal role in automating and streamlining various aspects of software development, system administration, and DevOps practices. Scripting languages like Python, Bash, and PowerShell empower tech professionals to write code that can execute repetitive tasks, manipulate data, and orchestrate complex processes efficiently.

Scripting facilitates rapid prototyping, configuration management, and the creation of automated deployment pipelines. It enhances productivity, ensures consistency and reduces human error in tasks ranging from software testing and deployment to infrastructure provisioning and monitoring. As organizations increasingly embrace DevOps and cloud-native technologies, scripting stays competitive and adaptive in the tech landscape.

## **24. Terraform**

Terraform plays a crucial role in modern infrastructure provisioning and management. It allows organizations to define and deploy infrastructure as code, enabling the automated creation and configuration of cloud resources, containers, and other infrastructure components. Cloud computing, microservices, and containerization have become the norm in

2024. Terraform provides the agility and scalability required to keep up with the dynamic demands of modern applications.

Terraform's importance lies in its ability to bring consistency, version control, and automation to infrastructure operations, thereby reducing manual errors, streamlining DevOps workflows, and facilitating applications' rapid and reliable deployment in an increasingly complex and cloud-centric environment. As organizations adopt cloud-native technologies, Terraform remains essential to ensure efficient and consistent infrastructure management.

## **25. Phantom**

Phantom enhances security automation and incident response capabilities. In today's rapidly evolving threat landscape, organizations face a constant barrage of cybersecurity incidents, and the ability to respond swiftly and effectively is necessary. It provides a platform for automating security workflows, from detecting and investigating potential threats to orchestrating responses and mitigating risks.

Phantom's importance lies in its capacity to reduce response times, increase consistency in incident handling, and free up manual resources from repetitive tasks. With the growing complexity of cyber threats, Phantom empowers security teams to defend against attacks and safeguard critical assets proactively.

## **26. Nagios**

Nagios, an open-source monitoring and alerting system, remains vital due to its enduring significance in maintaining the reliability and performance of IT infrastructure and applications. Organizations increasingly rely on complex systems and services. Nagios plays a crucial role by providing real-time monitoring and alerting capabilities, allowing IT teams to detect and address issues before they impact users or cause system outages.

Its versatility, extensibility, and support for both on-premises and cloud environments make Nagios a valuable tool for ensuring critical systems' availability, stability, and security, aligning perfectly with the demands of modern IT operations and DevOps practices.

## **27. Vagrant**

Vagrant continues to play a crucial role in software development and DevOps. It is a tool that simplifies creating and managing reproducible development environments. Its importance lies in its ability to provide developers and DevOps teams with a consistent and isolated environment for software development, testing, and deployment.

With the ever-evolving complexity of software stacks, dependencies, and infrastructure configurations, Vagrant remains essential in ensuring these environments are easily shareable, scalable, and maintainable. It allows developers to work seamlessly across various operating systems and provides a standardized setup that minimizes compatibility issues.

## **28. Sentry**

Sentry plays a critical role in modern software development and DevOps practices. With software applications' increasing complexity and scale, identifying and addressing errors and issues has become crucial.

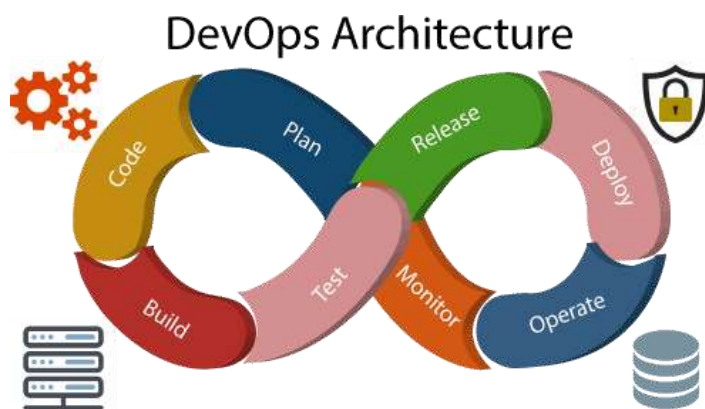
Sentry is vital because it provides real-time error tracking and monitoring, allowing development teams to proactively detect and diagnose issues, whether they occur in production or during development. Its importance is minimizing downtime, improving user experience, and maintaining software systems' overall health and reliability.

## 29. Gradle

Gradle continues to be a vital tool in software development and DevOps. Gradle is an advanced build automation system that plays a crucial role in managing dependencies, building projects, and orchestrating complex workflows efficiently. Its importance lies in its versatility and scalability, as it caters to various project sizes and types.

Gradle's ability to easily handle multi-language, multi-project builds and its support for plugin-based customization make it indispensable in modern software development. As organizations increasingly adopt microservices architectures and cloud-native technologies, Gradle's capabilities are instrumental in managing the complexity of building, testing, and deploying applications across diverse environments.

## DevOps Architecture



Development and operations both play essential roles in order to deliver applications. The deployment comprises analyzing the **requirements, designing, developing, and testing** of the software components or frameworks.

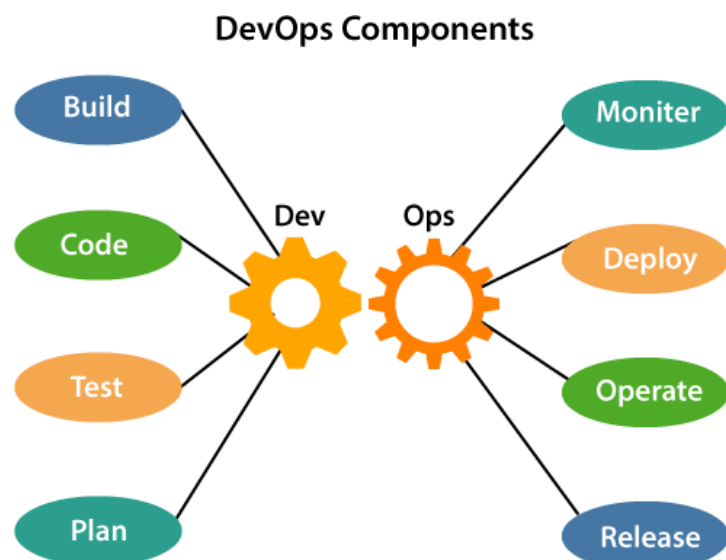
The operation consists of the administrative processes, services, and support for the software. When both the development and operations

are combined with collaborating, then the DevOps architecture is the solution to fix the gap between deployment and operation terms; therefore, delivery can be faster.

DevOps architecture is used for the applications hosted on the cloud platform and large distributed applications. Agile Development is used in the DevOps architecture so that integration and delivery can be contiguous. When the development and operations team works separately from each other, then it is time-consuming to **design, test, and deploy**. And if the terms are not in sync with each other, then it may cause a delay in the delivery. So DevOps enables the teams to change their shortcomings and increases productivity.

Below are the various components that are used in the DevOps architecture:

1) **Build:** Without DevOps, the cost of the consumption of the resources was evaluated based on the pre-defined individual usage with fixed hardware allocation. And with DevOps, the usage of cloud, sharing of resources comes into the picture, and the build is dependent upon the user's need,





which is a mechanism to control the usage of resources or capacity.

**2) Code:** Many good practices such as Git enables the code to be used, which ensures writing the code for business, helps to track changes, getting notified about the reason behind the difference in the actual and the expected output, and if necessary reverting to the original code developed. The code can be appropriately arranged in **files, folders**, etc. And they can be reused.

**3) Test:** The application will be ready for production after testing. In the case of manual testing, it consumes more time in testing and moving the code to the output. The testing can be automated, which decreases the time for testing so that the time to deploy the code to production can be reduced as automating the running of the scripts will remove many manual steps.

**4) Plan:** DevOps use Agile methodology to plan the development. With the operations and development team in sync, it helps in organizing the work to plan accordingly to increase productivity.

**5) Monitor:** Continuous monitoring is used to identify any risk of failure. Also, it helps in tracking the system accurately so that the health of the application can be checked. The monitoring becomes more comfortable with services where the log data may get monitored through many third-party tools such as **Splunk**.

**6) Deploy:** Many systems can support the scheduler for automated deployment. The cloud management platform enables users to capture accurate insights and view the optimization scenario, analytics on trends by the deployment of dashboards.

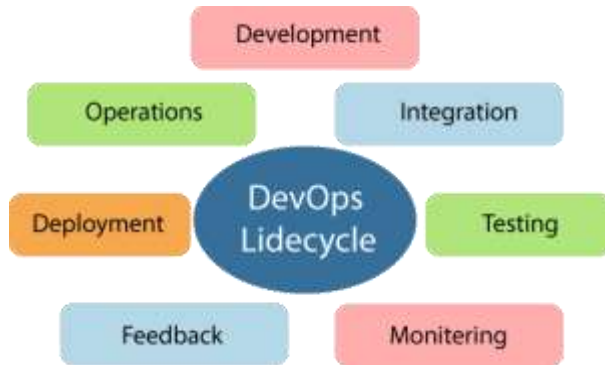
**7) Operate:** DevOps changes the way traditional approach of developing and testing separately. The teams operate in a collaborative way where both the teams actively participate throughout the service lifecycle. The operation team interacts with developers, and they come up with a monitoring plan which serves the IT and business requirements.

**8) Release:** Deployment to an environment can be done by automation. But when the deployment is made to the production environment, it is done by manual triggering. Many processes involved in release management commonly used to do the deployment in the production environment manually to lessen the impact on the customers.

## **DevOps Lifecycle**

DevOps defines an agile relationship between operations and Development. It is a process that is practiced by the development team and operational engineers. Learning DevOps is not complete without understanding the DevOps lifecycle phases. The DevOps lifecycle includes seven phases as given below:

together from beginning to the final stage of the product.



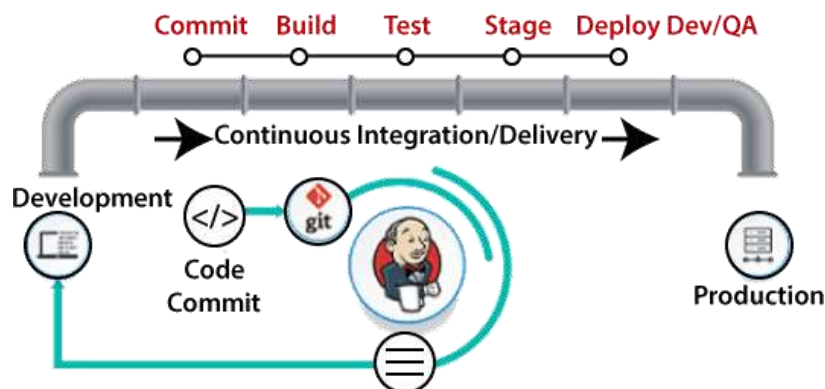
### 1) Continuous Development

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.

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.

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.

#### **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 continually. 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.

## **AWS**

AWS stands for Amazon Web Services, It is an expanded cloud computing platform provided by Amazon Company. AWS provides a wide range of services with a pay-as-per-use pricing model over the Internet such as Storage, Computing power, Databases, Machine Learning services, and much more. AWS facilitates for both businesses and individual users with effectively hosting the applications, storing the data securely, and making use of a wide variety of tools and services improving management flexibility for IT resources.

### **Advantages & Features of AWS:**

1. **Cost savings:** One of the biggest benefits of AWS is that it can help businesses save money. As mentioned previously, **businesses can avoid the high upfront costs of traditional infrastructure with AWS** and pay only for the resources they use. Traditionally, businesses had to invest in hardware and software upfront, which often led to overspending.

Let's look at this for example – if a business needs to run a website that gets 1000 visitors per day, they would need to purchase and maintain enough servers to support this traffic. With AWS, the business only pays for the compute resources they use when someone visits their website. This can result in significant cost savings.

2. **Flexibility:** Another key benefit of AWS is its flexibility. Businesses are able to customize their virtual environment – whether the operating system, database, programming language, or something else – to meet their specific needs. Especially in today's climate, the **migration process to the cloud** should be as frictionless as possible – and AWS makes that possible. Regardless of your use case or industry, **AWS can be tailored to fit your needs**, whether you're looking for a single cloud-hosted application or an entire suite of integrated solutions.
3. **Reliability:** AWS is known for being reliable, with an **uptime of 99.9%**. This makes it a **great platform for mission-critical applications that need to be available 24/7**. AWS also offers the ability to deploy resources across multiple availability zones for even greater reliability. The cloud platform also has a number of features that make it easier to ensure reliability, such as autoscaling and auto-healing. Autoscaling allows businesses to automatically scale their resources up or down based on demand, while auto-healing enables them to quickly identify and replace any faulty components.
4. **Security:** Businesses can take advantage of advanced security features, such as identity and access management, to help protect their data. Their tough infrastructure with an end-to-end approach is designed to withstand attacks and AWS provides customers with tools to help them monitor and respond to threats. When it comes to storage, Amazon S3 provides customers with a secure and reliable way to store and access data. The service is designed to be highly scalable and resilient, with built-in redundancy. Fine-grain identity and access controls can be applied to S3 buckets and objects, giving customers control over who has access to their data. **Security tasks can be automated with AWS CloudFormation**, making it easier for businesses to manage their security policies. And, you can rest easy knowing that AWS takes privacy seriously, with comprehensive customer data protection and compliance measures.



5. **Compliance:** By compliance, we mean that certain businesses are required to follow specific regulations. Financial services companies in the United States, for example, must comply with the **Sarbanes-Oxley Act**, while healthcare, education, and energy companies must comply with HIPAA and other regulations. AWS provides a number of compliance-related features and services, such as data encryption and identity and access management, to help businesses meet these requirements.
6. **High-Performance:** Interested in delivering your applications quickly and efficiently? Taking advantage of AWS features such as **auto-scaling and load balancing** will help ensure your applications are always available and running optimally. AWS can help businesses improve their performance by offering a variety of cloud-based services, including **Amazon Elastic Compute Cloud (EC2)**, which provides high-performance computing resources, and **Amazon CloudFront**, which delivers content quickly and securely to users around the world. Others include machine learning (ML) and analytics services, such as Amazon SageMaker and Amazon Athena. These services provide the tools businesses need to quickly and easily analyze their data for insights. Fast networking in the cloud is also possible with AWS, thanks to its Elastic Load Balancing (ELB) and Amazon Virtual Private Cloud (VPC). With ELB, businesses can balance their workloads across multiple instances for increased performance, while VPC allows businesses to create isolated private networks in the cloud.
7. **Developer Tools:** Developer tools are designed to make it easier for developers to create, deploy, and manage applications – and AWS provides developers with what they need to build applications quickly and easily. By leveraging developer tools, **developers can save time and money by automating tedious tasks**. They also benefit from access to **AWS's extensive library of pre-built applications** that can help them get their projects off the ground quickly. Services such as Amazon Elastic Beanstalk and Amazon CloudFormation can help them automate the process of creating and deploying applications. Other ways developers can improve productivity with AWS include using AWS CodeCommit to store and manage source code.
8. **Integration:** Thanks to its many integrations with other Amazon services, as well as third-party services, AWS makes it easy for businesses to get started with cloud computing. AWS provides a wide range of services that can be easily integrated into existing business infrastructure. This allows businesses to add new features and capabilities without having to make major changes or invest in new hardware or software. For instance, if a business wants to add mobile capabilities to its website, **it can take advantage of Amazon's Mobile SDK and Web Services**. These tools allow businesses to quickly develop and deploy mobile apps that connect directly with their existing infrastructure.
9. **Management Console:** The AWS management console is a web-based interface that provides users with a simple way to interact with and manage their AWS resources – essentially a place where you can access and manage everything on the cloud. It provides a graphical view of all the resources associated with an account, as well as tools for creating and configuring new resources. Compared to traditional command-line interfaces, the **AWS management console saves time and makes it easier for users to get the most out of their AWS services**. Not only that, but *your business gets access to 350+ free digital training courses through the **AWS Academy***, covering topics such as cloud fundamentals, DevOps, security, and big data. This means you can train your employees on how to use AWS, and in turn, help them become more efficient at their jobs.

- 10. Scalability:** With an on-demand service, businesses can quickly spin up new servers as needed with just a few clicks. This makes it much easier to scale resources up or down as demand changes, allowing businesses to save costs and maintain performance even during peak periods. For example, if a business is expecting a sudden surge in traffic due to an advertising campaign or seasonal event, they can easily add more capacity to their server infrastructure to handle the increased load. **Bru Textiles**, a specialty textile company in Belgium, was able to quickly scale its infrastructure by leveraging AWS. Bru Textiles went digital to grow and offer new services. Embracing technology, they brought in digital twin technology to give their customers an idea of the texture and essence of their physical fabrics.

## **AWS Applications**

- **Storage and Backup:** Storage and backup are important for any Cloud Computing service. AWS provides you with reliable storage services like Amazon Simple Storage Service to store large-scale data and backup services like AWS Backup to take backups of this data, which is stored in other AWS services. AWS stores the data in three different availability zones so that if one fails, you can still access your data. This makes AWS storage reliable and easily accessible. Therefore, companies with huge application data to store and backup securely can use AWS.
- **Big Data:** One of the biggest challenges faced by companies these days is Big Data. The companies are struggling to store their large amounts of data using traditional methods. With AWS Big Data storage services, they can manage to store their data even if the data limit increases unexpectedly as AWS provides virtually unlimited data storage with scale-in and scale-out options. AWS offers easy access and faster data retrieval as well. For data processing, it offers services like EMR, with which the companies can easily set up, operate, and scale their big data. Therefore, efficiently storing and managing Big Data is among the top AWS applications.
- **Enterprise IT:** AWS is a one-stop solution for any IT business. Many features of it such as secure storage, scalability, flexibility, and elasticity support companies to innovate faster than ever before. Using AWS for IT enterprises makes them profitable in terms of both money and time. As AWS maintains its cloud architecture, it need not waste time and money on professionals to do the same.
- **Social Networking:** Social networking is essential for businesses in the present-day scenario where Digital Marketing is key, and it is easier with AWS. Companies can connect with customers and stakeholders and communicate through social networking sites and develop their business. Services like AWS social networking engine, which is powered by TurnKey GNU/Linux (HVM) AMI stack, are used for performance and scalability to help companies build a suitable social networking site and gain profits.
- **Mobile Apps:** Mobile applications are embedded with day-to-day life. With AWS, you have the facility to create an app in your desired programming language. You can also keep up the applications that are consistently accessible and solid with high compute, storage, database, and application services. You can take advantage of AWS auto-scaling and managed relational database service for the better performance of your apps.
- **Websites:** AWS offers a wide range of website hosting options to create the best website for customers. Its services like Amazon Lightsail have everything, such as a virtual machine, SSD-based storage, data transfer, DNS management, and a static IP, to launch a website in such a way that the user can manage the website easily. Amazon EC2, AWS Lambda, Elastic Load Balancing, AWS Amplify, Amazon S3, etc. also help users build reliable and scalable websites.

- **Gaming:** AWS has been serving many gaming studios. Combining Amazon EC2 and S3 services with CloudFront enables gaming websites to deliver high-quality gaming experiences to their customers regardless of location.

## Use Cases of AWS

- **Netflix**

Netflix is an entertainment platform that started in the United States, but eventually, it expanded to many countries and soon became popular. However, once Netflix confronted the scalability problem because of the sudden increase in viewers. That made Netflix choose AWS services. Netflix reports that when it started using AWS services like DynamoDB and Cassandra for its distributed databases, it could handle the data easily. So, scalability is a great advantage of AWS. Netflix has adapted around 100,000 server instances from AWS for computing and storage databases, analytics, recommendation engines, and video transcoding as well.

- **McDonald's**

McDonald's is the world's largest fast-food company that serves around 64 million people per day. The growth of this company has gone to another level when it started home deliveries. By utilizing AWS services, McDonald's created a platform that integrates local restaurants with delivery partners such as Uber Eats. Scalability is also a reason for the company to choose AWS services. Moreover, with AWS Microservices Architecture, McDonald's platform can scale 20,000 orders per second and integrate with the global partners easily.

- **Airbnb**

Airbnb is an international online marketplace for rental homes. This platform connects people who are looking for rental accommodation with those who want to rent out their houses. Quite soon, Airbnb became unable to handle the constant streaming of data on the website from its customers. That is when it started using Amazon EC2 service and Elastic Load Balancing, which distributes incoming traffic to multiple Amazon EC2 instances. In this way, Airbnb could avoid traffic, and customers could use the online platform without any disruption.

- **Novartis**

Novartis is the best example for AWS use cases in healthcare. Novartis is one of the world's largest healthcare companies that provides solutions for patients' well-being. It adapted Amazon EC2 services and built a platform using other services such as Amazon Simple Storage Service, Amazon Elastic Block Store, and four availability zones. Data Analysts of Novartis are taking advantage of the AWS services and still implementing new solutions for the patients.

- **Expedia**

Expedia is a worldwide online travel agency that has always focused on the constant development and innovation of its platform to offer an extraordinary user experience for its clients. Since 2010, Expedia has been using AWS services to build a standard deployment model for better infrastructure as AWS offers the best data security through different availability zones.

- **Samsung**

If you are using Samsung mobile phones, then you may know about the Samsung app store. For setting up the apps stacked in its store, the company started using AWS services. Using AWS app development services, Samsung wanted to provide its customers with the facility to download the apps anywhere without any network traffic.

- **NASA**

NASA (National Aeronautics and Space Administration) has always wondered about creating a library to present people with all its achievements through pictures and videos of space. Later on, it created such platforms, but because it had 10 different NASA centers, it couldn't provide the best experience for viewers. So, all it wanted was to create an easy-access platform for

people to search for and view images and videos. Then, NASA started adopting many services from AWS to solve this problem, which included Amazon Elastic Compute Cloud, Elastic Load Balancing, Amazon Simple Storage Service, Amazon Simple Queue Service, etc. Among these, Amazon S3 helped the company store all the incoming data such as photos, videos, and audio files without any hassle.

- **Facebook**

Facebook, without a doubt, is a widespread social media platform. To build a scalable application, Facebook used services such as Amazon Elastic Compute Cloud, Amazon Simple Storage Service, Amazon Relational Database Service, Amazon SimpleDB, Amazon CloudFront, Amazon Simple Queue Service, etc. Amazon RDS helps the platform to make it easy to set up, operate, and scale the database in the cloud.

### **Various Services offered by AWS**

- Amazon EC2 (Elastic Cloud computing)
- Amazon RDS (Relational Database Services)
- Bonus Service: Amazon Connect
- Amazon S3 (Simple Storage Service)
- Amazon Lambda
- Amazon Cognito
- Amazon Glacier
- Amazon SNS (Simple Notification Service)
- Bonus Service: Amazon Lex
- Amazon Lightsail
- Amazon VPC (Virtual Private Cloud)
- Amazon Kinesis
- Amazon Inspector
- Amazon Auto-scaling
- Amazon IAM (Identity and Access Management)
- Dynamo DB
- Amazon SQS (Simple Queue Service)
- Amazon ElastiCache
- Amazon Chime
- AWS Athena
- CodeCatalyst
- Web Application Firewall
- AWS Amplify
- AWS Rekognition
- AWS QuickSight
- AWS Cloudformation
- AWS Management Console

The Important Cloud Services according to various categories that are provided by AWS are given below :

### **1. Compute**

- **Amazon EC2:** Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It allows organizations to obtain and configure virtual compute capacity in the cloud. You can select from a variety of operating systems and resource configurations like memory, CPU, and storage that are required for your



application. Amazon EC2 enables you to increase or decrease capacity within minutes. You can use one or hundreds or even thousands of server instances simultaneously. Because this is all controlled with web service APIs, your application can automatically scale itself up and down depending on its needs. Amazon EC2 is integrated with most AWS services, such as Amazon Simple Storage Service (Amazon S3), Amazon Relational Database Service (Amazon RDS), and Amazon Virtual Private Cloud (Amazon VPC) to provide a complete, secure solution for computing applications. Amazon EC2 is an example of Infrastructure as a Service(IaaS). EC2 delivers secure, reliable, cost-effective compute and high-performance compute infrastructure so as to meet the needs of demanding businesses. Amazon EC2 is one of the easiest ways of providing servers on AWS Cloud and also the access to Operating system.

- **AWS Lambda:** AWS Lambda is a serverless, event-driven compute service that allows you to run code without managing servers. You pay only for the compute time you consume and there is no charge when your code is not running. With AWS Lambda, you can run code for any type of application with zero administration. Just upload your code, and Lambda takes care of everything required to run and scale your code with high availability. You can set up your code to automatically trigger from other AWS services, or you can call it directly from any web or mobile app. But triggering Lambda is possible with over 200 AWS services. You can only pay for what you have used. The compute time that you consume, you are needed to pay for it. You just only need to upload your code and everything required to run will take care of by Lambda and it automatically scales your code with high availability.
- **AWS Elastic Beanstalk:** AWS Elastic Beanstalk is a Platform as a Service that facilitates quick deployment of your applications by providing all the application services that you need for your application. Beanstalk is a plug-and-play platform that allows working with multiple programming languages and environments. Elastic Beanstalk supports a large range of platforms like Node js, Java, PHP, Python, and Ruby. So, you can develop your application to meet your requirements and simply deploy it on Elastic Beanstalk. The main aim to use AWS Elastic Beanstalk is to allow you to focus on the deployment and management of your applications. You can simply upload your code, and AWS Elastic Beanstalk automatically handles the deployment, from capacity provisioning, load balancing, and auto-scaling to application health monitoring. At the same time, you retain full control over the AWS resources powering your application and can access the underlying resources at any time.

## 2. Networking

- **Amazon VPC:** Amazon VPC is your network environment in the cloud. It allows you to create a private network within the AWS cloud that uses many of the same concepts and constructs as an on-premises network. Amazon VPC also gives you complete control of the network configuration. Customers can define normal networking configuration items such as IP address ranges, subnet creation, route table creation, network gateways, and security settings. Amazon VPC is an AWS foundational service and integrates with numerous AWS services. For instance, Amazon EC2 instances are deployed into your

Amazon VPC. Similarly, Amazon Relational Database Service (Amazon RDS) database instances deploy into your Amazon VPC, where the database is protected by the structure of the network just like your on-premises network. You can easily launch AWS resources into a virtual network by Amazon Virtual Private Cloud. An isolated virtual network environment in the AWS cloud is created by Amazon VPC.

- **Amazon Route 53:** Amazon Route 53 is a highly available and scalable cloud Domain Name System (DNS) web service. It is designed to give developers and businesses an extremely reliable and cost-effective way to route end users to Internet applications by translating human-readable names, such as [www.geeksforgeeks.com](http://www.geeksforgeeks.com), into the numeric IP addresses that computers use to connect to each other. Amazon Route 53 is fully compliant with IPv6 as well.

### 3. Storage

- **Amazon S3 (Simple Storage Service):** Amazon Simple Storage Service (Amazon S3) is object storage with a simple web service interface to store and retrieve any amount of data from anywhere on the web. It is designed to provide an infinite amount of storage and it is delivered with 99.999999999% durability. You can use Amazon S3 as primary storage for cloud-native applications as a target for backup and recovery and disaster recovery. It offers industry-leading scalability, data availability, security, and performance. It's simple to move large volumes of data into or out of Amazon S3 with Amazon's cloud data migration options. Once data is stored in Amazon S3, it can be automatically tiered into lower cost, longer-term cloud storage classes like Amazon S3 Standard – Infrequent Access and Amazon Glacier for archiving.
- **Amazon Glacier:** Amazon Glacier is a secure, durable, and extremely low-cost storage service for data archiving and long-term backup. Data stored in Amazon Glacier takes several hours to retrieve, which is why it's ideal for archiving. The fastest access to your archive data is via Amazon Glacier.

### 4. Databases

- **Amazon RDS (Relational Database Service):** Amazon Relational Database Service (Amazon RDS) makes it easy to set up, operate, and scale a relational database in the cloud. It provides cost-efficient and resizable capacity while managing time-consuming database administration tasks, freeing you up to focus on your applications and business. You can find Amazon RDS is also available on several database instance types – optimized for memory, performance, or I/O. Amazon RDS provides you with six familiar database engines to choose from, including Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle, and Microsoft SQL Server.
- **Amazon DynamoDB (Non-Relational Database):** Amazon DynamoDB is a fast and flexible NoSQL database service for all applications that need consistent, single-digit millisecond latency at any scale. It is a fully managed database and supports both document and key-value data models. When you create a database table that can store and retrieve any amount of data you can simply use Amazon DynamoDB that will serve any level of requested traffic. Its flexible data model and reliable performance make it a great fit for mobile, web, gaming, Internet of Things (IoT), and many other applications. DynamoDB provides many features like
  - built-in security

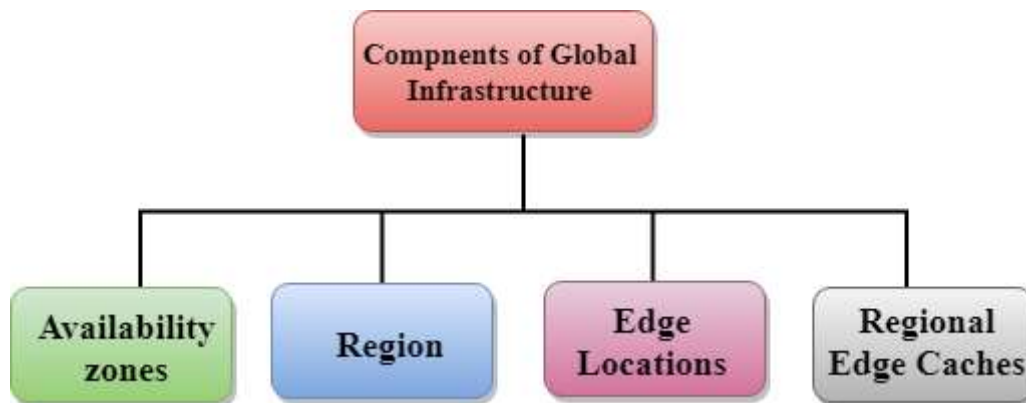
- backups
- automated multi-region replication
- in-memory caching
- data export tools.

## **Global Infrastructure of AWS**

- AWS is a cloud computing platform which is globally available.
- Global infrastructure is a region around the world in which AWS is based. Global infrastructure is a bunch of high-level IT services which is shown below:
- AWS is available in 19 regions, and 57 availability zones in December 2018 and 5 more regions 15 more availability zones for 2019.

The following are the components that make up the AWS infrastructure:

- Availability Zones
- Region
- Edge locations
- Regional Edge Caches



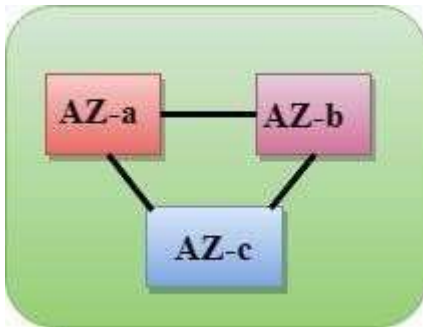
### **Availability zone as a Data Center**

- An availability zone is a facility that can be somewhere in a country or in a city. Inside this facility, i.e., Data Centre, we can have multiple servers, switches, load balancing, firewalls. The things which interact with the cloud sits inside the data centers.
- An availability zone can be a several data centers, but if they are close together, they are counted as 1 availability zone.

### **Region**

- A region is a geographical area. Each region consists of 2 more availability zones.

- A region is a collection of data centers which are completely isolated from other regions.
- A region consists of more than two availability zones connected to each other through links.



- Availability zones are connected through redundant and isolated metro fibers.

### Edge Locations

- Edge locations are the endpoints for AWS used for caching content.
- Edge locations consist of CloudFront, Amazon's Content Delivery Network (CDN).
- Edge locations are more than regions. Currently, there are over 150 edge locations.
- Edge location is not a region but a small location that AWS have. It is used for caching the content.
- Edge locations are mainly located in most of the major cities to distribute the content to end users with reduced latency.
- For example, some user accesses your website from Singapore; then this request would be redirected to the edge location closest to Singapore where cached data can be read.

### Regional Edge Cache

- AWS announced a new type of edge location in November 2016, known as a Regional Edge Cache.
- Regional Edge cache lies between CloudFront Origin servers and the edge locations.
- A regional edge cache has a large cache than an individual edge location.
- Data is removed from the cache at the edge location while the data is retained at the Regional Edge Caches.
- When the user requests the data, then data is no longer available at the edge location. Therefore, the edge location retrieves the cached data from the Regional edge cache instead of the Origin servers that have high latency.



## What Is Cloud Computing?

Cloud computing is the use of hardware and software components in an off-premises location to deliver a service to a network. Users can access files and applications from any device that can access the internet.

Some features and capabilities include:

- Cloud providers can pull the computing resources to provide services to multiple customers with the help of a multi-tenant model
  - Cloud computing provides an on-demand self-service, which helps administrators monitor performance
  - Servers are maintained easily and there is nearly zero downtime
  - Users can access cloud data and upload it on the cloud from any device with a solid internet connection
  - Cloud environments can be modified according to the user's requirements and is easily accessible
  - Clouds are highly secure, making data breaches more unlikely
  - Migrating to the cloud eliminates the need to buy on-premises infrastructure
  - It offers pay-as-you-go pricing, meaning you only pay for the resources you use
1. **Infrastructure as a Service:** IaaS delivers virtualized computing resources over the Internet. Users can rent virtual machines, storage, and networking infrastructure, allowing for easy scalability without investing in physical hardware. Examples include AWS EC2 and Azure **Virtual Machines**.
  2. **Platform as a Service:** PaaS offers a robust platform for developers to build, deploy, and manage apps without worrying about the underlying infrastructure. It simplifies application development and deployment, with services like Google App Engine and Heroku leading the way.
  3. **Software as a Service:** SaaS offers software applications on a subscription basis, accessible via a web browser. Users don't need to install or maintain software locally, making it ideal for collaboration tools (e.g., **Microsoft 365**, Google Workspace) and CRM systems (e.g., Salesforce).
  4. **Function as a Service:** FaaS allows developers to execute code responding to events without managing servers. It's highly scalable and cost-efficient, exemplified by **AWS Lambda** and Azure Functions. FaaS is also known as serverless computing.
  5. **Container as a Service:** CaaS enables the deployment and management of containerized applications using orchestration tools like Kubernetes. It provides portability and scalability for applications across different cloud environments.

**Difference between main cloud computing services**

Terms	IAAS	PAAS	SAAS
<b>Stands for</b>	Infrastructure as a service.	Platform as a service.	Software as a service.
<b>Uses</b>	IAAS is used by network architects.	PAAS is used by developers.	SAAS is used by the end user.
<b>Access</b>	IAAS gives access to the resources like virtual machines and virtual storage.	PAAS gives access to run time environment to deployment and development tools for application.	SAAS gives access to the end user.
<b>Model</b>	It is a service model that provides virtualized computing resources over the internet.	It is a cloud computing model that delivers tools that are used for the development of applications.	It is a service model in cloud computing that hosts software to make it available to clients.
<b>Technical understanding.</b>	It requires technical knowledge.	Some knowledge is required for the basic setup.	There is no requirement about technicalities company handles everything.
<b>Popularity</b>	It is popular among developers and researchers.	It is popular among developers who focus on the development of apps and scripts.	It is popular among consumers and companies, such as file sharing, email, and networking.
<b>Percentage rise</b>	It has around a 12% increment.	It has around 32% increment.	It has about a 27 % rise in the cloud computing model.
<b>Usage</b>	Used by the skilled developer to develop unique applications.	Used by mid-level developers to build applications.	Used among the users of entertainment.
<b>Cloud services.</b>	Amazon Web Services, sun, vCloud Express.	Facebook, and Google search engine.	MS Office web, Facebook and Google Apps.
<b>Enterprise services.</b>	AWS virtual private cloud.	Microsoft Azure.	IBM cloud analysis.
<b>Outsourced cloud services.</b>	Salesforce	Force.com, Gigaspaces.	AWS, Terremark
<b>User Controls</b>	Operating System, Runtime, Middleware, and Application data	Data of the application	Nothing

**Google cloud services**

Google offers a seven wide range of Services:

- **Compute**
- **Networking**
- **Storage and Databases**
- **Big Data**
- **Machine Learning**
- **Identity & Security**
- **Management and Developer Tools**

1. **Compute:** GCP provides a scalable range of computing options you can tailor to match your needs. It provides highly customizable virtual machines. and the option to deploy your code directly or via containers.

- Google Compute Engine
- Google App Engine
- Google Kubernetes Engine
- Google Cloud Container Registry
- Cloud Functions

2. **Networking:** The Storage domain includes services related to **networking**, it includes the following services

- Google Virtual Private Cloud (VPC)
- Google Cloud Load Balancing
- Content Delivery Network
- What is Google Cloud Connect
- Google Cloud DNS
- What is Google Cloud Web Hosting

3. **Storage and Databases:** The Storage domain includes services related to data **storage**, it includes the following services

- Google Cloud Storage
- Cloud SQL
- Cloud Bigtable
- Google Cloud Datastore
- Persistent Disk

4. **Big Data:** The Storage domain includes services related to **big data**, it includes the following services

- Google BigQuery
- Google Cloud Dataproc
- Google Cloud Datalab
- Google Cloud Pub/Sub

5. **Cloud AI:** The Storage domain includes services related to **machine learning**, it includes the following services

- Cloud Machine Learning
- Vision API
- Speech API

- Natural Language API
- Translation API
- Jobs API

**6. Identity & Security:** The Storage domain includes services related to **security**, it includes the following services

- Cloud Resource Manager
- Cloud IAM
- Cloud Security Scanner
- Cloud Platform Security

**7. Management Tools:** The Storage domain includes services related to **monitoring and management**, it includes the following services

- Stackdriver
- Monitoring
- Logging
- Error Reporting
- Trace
- Cloud Console

**8. Developer Tools:** The Storage domain includes services related to **development**, it includes the following services

- Cloud SDK
- Deployment Manager
- Cloud Source Repositories
- Cloud Test Lab

## **AZURE**

Azure is Microsoft's cloud platform, just like Google has its Google Cloud and Amazon has its Amazon Web Service or AWS.000. Generally, it is a platform through which we can use Microsoft's resources. For example, to set up a huge server, we will require huge investment, effort, physical space, and so on. In such situations, Microsoft Azure comes to our rescue. It will provide us with virtual machines, fast processing of data, analytical and monitoring tools, and so on to make our work simpler. The pricing of Azure is also simpler and cost-effective. Popularly termed as "*Pay As You Go*", which means how much you use, pay only for that.

*Microsoft Azure Used for*

- **Deployment Of applications:** You can develop and deploy the application in the azure cloud by using the service called Azure App Service and Azure Functions after deploying the applications end users can access it.
- **Identity and Access Managment:** The application and data which is deployed and stored in the Microsoft Azure can be secured with the help of Identity and Access Managment. It's commonly used for single sign-on, multi-factor authentication, and identity governance.
- **Data Storage and Databases:** You can store the data in Microsoft azure in service like blob storage for unstructured data, table storage for NoSQL



data, file storage, and Azure SQL Database for relational databases. The service can be scaled depending on the amount of data we are getting.

- **DevOps and Continuous Integration/Continuous Deployment (CI/CD):** Azure DevOps will provide some tools like including version control, build automation, release management, and application monitoring

*Following are some of the services Microsoft Azure offers:*

1. **Compute:** Includes Virtual Machines, Virtual Machine Scale Sets, Functions for serverless computing, Batch for containerized batch workloads, Service Fabric for microservices and container orchestration, and Cloud Services for building cloud-based apps and APIs.
2. **Networking:** With Azure, you can use a variety of networking tools, like the Virtual Network, which can connect to on-premise data centers; Load Balancer; Application Gateway; VPN Gateway; Azure DNS for domain hosting, Content Delivery Network, Traffic Manager, ExpressRoute dedicated private network fiber connections; and Network Watcher monitoring and diagnostics
3. **Storage:** Includes Blob, Queue, File, and Disk Storage, as well as a Data Lake Store, Backup, and Site Recovery, among others.
4. **Web + Mobile:** Creating Web + Mobile applications is very easy as it includes several services for building and deploying applications.
5. **Containers:** Azure has a property that includes Container Service, which supports Kubernetes, DC/OS or Docker Swarm, and Container Registry, as well as tools for microservices.
6. **Databases:** Azure also included several SQL-based databases and related tools.
7. **Data + Analytics:** Azure has some big data tools like HDInsight for Hadoop Spark, R Server, HBase, and Storm clusters
8. **AI + Cognitive Services:** With Azure developing applications with artificial intelligence capabilities, like the Computer Vision API, Face API, Bing Web Search, Video Indexer, and Language Understanding Intelligent.
9. **Internet of Things:** Includes IoT Hub and IoT Edge services that can be combined with a variety of machine learning, analytics, and communications services.
10. **Security + Identity:** Includes Security Center, Azure Active Directory, Key Vault, and Multi-Factor Authentication Services.
11. **Developer Tools:** Includes cloud development services like Visual Studio Team Services, Azure DevTest Labs, HockeyApp mobile app deployment and monitoring, Xamarin cross-platform mobile development, and more.

**Difference between AWS (Amazon Web Services), Google Cloud, and Azure**

	AWS	Google Cloud	Azure
<b>Technology</b>	<u>EC2 (Elastic Compute Cloud)</u>	<u>Google Compute Engine(GCE)</u>	VHD (Virtual Hard Disk)
<b>Databases Supported</b>	AWS fully supports relational and NoSQL databases and Big Data.	Technologies pioneered by Google, like Big Query, Big Table, and Hadoop, are databases, and Big Data,naturally fully supported.	Azure supports both relational and NoSQL through Windows AzureTable and HDInsight.
<b>Pricing</b>	Per hour — rounded up.	Per minute — rounded up	Per minute — rounded up.
<b>Models</b>	On demand, reserved spot.	On demand — sustained use.	Per minute- rounded up commitments(Pre-paid or monthly)
<b>Difficulties</b>	Many enterprises find it difficult to understand the company cost structure.	Fewer features and services.	Less “Enterprise-ready.
<b>Storage Services</b>	<u>Simple Storage Service(S3)</u> <u>Elastic Block Storage.</u> <u>Elastic File storage.</u>	<u>Blob Storage</u> <u>Queue Storage.</u> <u>File Storage</u> <u>Disk Storage.</u> <u>Data Lake Store</u>	Cloud storage. Persistent Disk Transfer appliance.
<b>Machine Learning</b>	<u>Sage maker.</u> Lex. polly.And many more	Machine learning Azure Bot service Cognitive service	Cloud speech AI Cloud Video Intelligence. Cloud Machine learning engine

**GIT**

Git is a distributed version control system (DVCS) that helps manage and track changes in source code during software development. It was created by Linus Torvalds in 2005 and has become one of the most widely used version control systems in the software development industry.

*Some key concepts and features of Git:*

- **Version Control:** Git allows developers to keep track of changes made to their code over time. This includes modifications, additions, and deletions of files.
- **Distributed System:** Git is a distributed version control system, meaning that each developer has a complete copy of the entire repository, including its full history. This allows developers to work independently and merge their changes when necessary.
- **Branching:** Git enables developers to create branches, which are essentially separate lines of development. This allows for the parallel development of features or bug fixes without affecting the main codebase.

- **Merging:** Git provides tools for merging changes from one branch into another. This is essential when multiple developers are working on different branches and need to bring their changes together.
- **History Tracking:** Git maintains a complete history of changes made to the codebase. Developers can view, revert, or analyze changes made over time.
- **Remote Repositories:** Git supports remote repositories, allowing developers to collaborate with others by pushing and pulling changes to and from a shared repository. Platforms like GitHub, GitLab, and Bitbucket provide hosting services for Git repositories.
- **Staging Area:** Git uses a staging area (also known as the index) to prepare and review changes before committing them to the repository. This allows developers to selectively include or exclude specific changes.
- **Open Source:** Git is an open-source tool, and its source code is freely available for modification and distribution.

### Various GIT Components:

Git is composed of several key components that work together to enable version control and collaborative development. Here are the main components of Git:

- **Repository (Repo):** A repository is a directory or storage space where your project and its version history are stored. It contains all the files and directories associated with your project, along with the metadata and configuration information.
- **Working Directory:** The working directory is the directory on your local machine where you manipulate files and make changes to your project. It is essentially your local copy of the repository.
- **Index (Staging Area):** The index, also known as the staging area, is a middle ground where changes are prepared before being committed to the repository. It allows you to selectively stage changes, which means you can choose which modifications to include in the next commit.
- **Commit:** A commit is a snapshot of the changes made to the files in the repository. It represents a specific point in the project's history and is accompanied by a commit message that describes the changes.
- **Branch:** A branch is a parallel line of development within a repository. It allows developers to work on different features or bug fixes simultaneously without affecting the main codebase. Branches can be merged to incorporate changes into other branches.
- **Head:** HEAD is a reference to the latest commit in the currently checked-out branch. It essentially points to the tip of the branch you are currently on.
- **Remote:** A remote is a version of the repository stored on a different server. Git supports collaboration by allowing developers to push and pull changes between their local repository and remote repositories. Platforms like GitHub, GitLab, and Bitbucket are examples of remote repositories.
- **Clone:** Cloning is the process of creating a copy of a remote repository on your local machine. This allows you to start working on your own copy of a project.
- **Fetch:** The fetch operation retrieves changes from a remote repository but does not automatically merge them into your working directory. It is useful for reviewing changes before deciding to merge.
- **Pull:** Pull is a combination of fetch and merge. It retrieves changes from a remote repository and automatically merges them into your working directory.

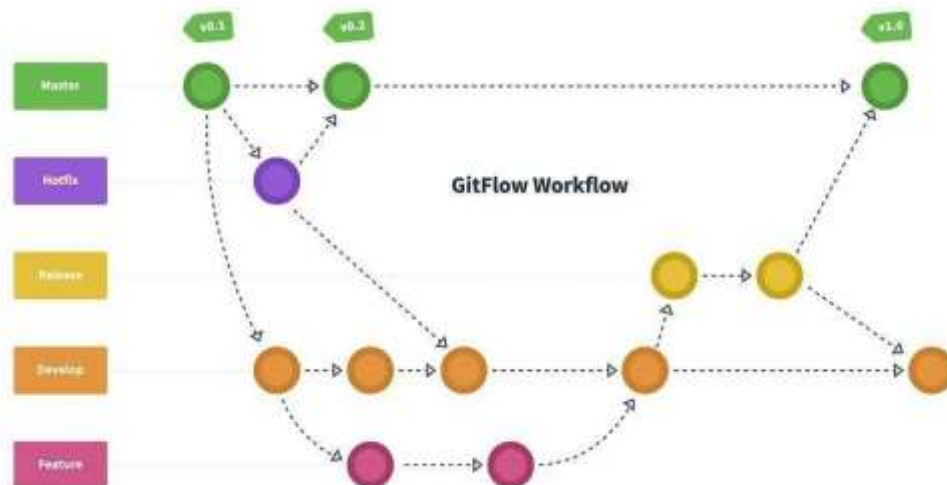
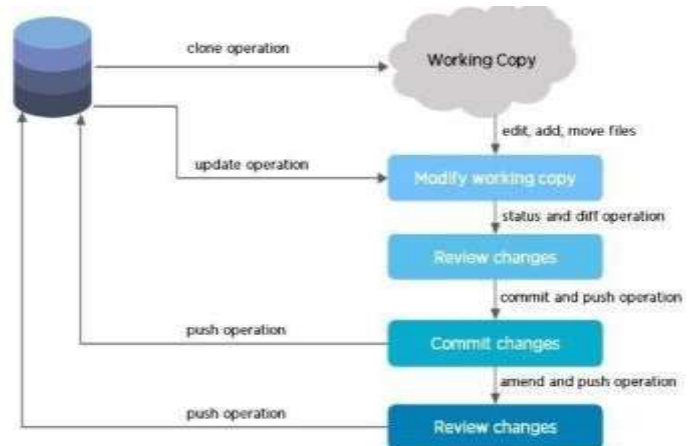
- || **Push:** Push is the operation that sends your committed changes to a remote repository, making them accessible to others.

## Git workflow

The Git workflow is divided into three states:

1. **Working directory** - Modify files in your working directory
2. **Staging area (Index)** - Stage the files and add snapshots of them to your staging area
3. **Git directory (Repository)** -

Perform a commit that stores the snapshots permanently to your Git directory. Checkout any existing version, make changes, stage them and commit.



Git Flow is a structured branching model designed for projects with well-defined release cycles and a need for strict quality control.

## **Branches:**

The branching model described is commonly known as the Gitflow Workflow. It's a branching strategy that defines a strict branching model

designed to facilitate collaboration and streamline the release process. Let's go into detail about each branch:

1. **Master Branch:** The `master` branch represents the main codebase and contains production-ready code. This branch is typically stable and should only include thoroughly tested and approved changes. Each commit on the `master` branch represents a new version or release of the software.
2. **Develop Branch:** The `develop` branch is an integration branch where various feature branches are merged. It serves as a staging area for testing new features and ensuring they work well together before merging into the `master` branch. This branch may have ongoing development work and is not necessarily always in a production-ready state.
3. **Feature Branches:** Feature branches are created for developing new features or implementing changes. These branches are typically based on the `develop` branch. Once a feature is complete, the branch is merged back into the `develop` branch. Feature branches allow developers to work on specific tasks without affecting the main codebase.
4. **Release Branch:** The `release` branch is created when the `develop` branch reaches a point where it is ready for a production release. This branch is used for final testing, bug fixes, and preparing the code for deployment. No new features should be added to the release branch. Once the release is deemed stable, it is merged into both the `master` branch and the `develop` branch.
5. **Hotfix Branch:** The `hotfix` branch is used to quickly address critical issues or bugs in the production code. It is created directly from the `master` branch. Hotfixes are intended to be small and focused on resolving the specific issue at hand. Once the hotfix is



complete, it is merged into both the `master` branch and the `develop` branch to ensure that the fix is applied to future releases.

### Here is the typical flow:

- Developers work on feature branches based on the `develop` branch.
- Completed features are merged into the `develop` branch.
- When ready for a release, a `release` branch is created from `develop`.
- The release branch undergoes testing and bug fixes.
- The release branch is merged into both `master` and `develop` once it's stable.
- If a critical issue arises in production, a `hotfix` branch is created from `master`.
- The hotfix is merged into both `master` and `develop` to keep both branches in sync.
- This Gitflow Workflow helps maintain a structured development process, ensuring that features are developed, tested, and released in a controlled manner.

### Example Scenario:

- Imagine you're working on a large software project with a team of 10 developers. You have a major release planned for every six months.
- You create a *"feature/Ticket-Id"* branch to develop a new login system.
- Once the feature is complete, it's merged into the *"develop"* branch for integration and testing.
- As the release date approaches, you create a *"release/v1.0"* branch to freeze code for the upcoming release.
- Any critical issues discovered in the production environment are fixed in *"hotfix"* branch and merged into *"master"* and *"develop."*

## GIT INSTALLATION

### Git for Windows stand-alone installer

- Download the latest Git for Windows installer.
- When you've successfully started the installer, you should see the **Git Setup** wizard screen. Follow the **Next** and **Finish** prompts to complete the installation. The default options are pretty sensible for most users.
- Open a Command Prompt (or Git Bash if during installation you elected not to use Git from the Windows Command Prompt).

- Run the following commands to configure your Git username and email using the following commands, replacing Emma's name with your own. These details will be associated with any commits that you create:

```
$ git config --global user.name "CIT_CHENNAI"
$ git config --global user.email CITCHENNAI@atlassian.com
```

- **Optional:** *Install the Git credential helper on Windows*

Bitbucket supports pushing and pulling over HTTP to your remote Git repositories on Bitbucket. Every time you interact with the remote repository, you must supply a username/password combination. You can store these credentials, instead of supplying the combination every time, with the Git Credential Manager for Windows.

## **BASIC COMMANDS OF GIT:**

Some basic Git commands along with their syntax and examples:

- **Initialize a Repository:**

Syntax: ``git init``

Example: ``git init``

- **Clone a Repository:**

Syntax: ``git clone <repository_url>``

Example: ``git clone https://github.com/example/repository.git``

- **Check Repository Status:**

Syntax: ``git status``

Example: ``git status``

- **Add Changes to Staging Area:**

Syntax: ``git add <file(s)>``

Example: ``git add file.txt``

- **Commit Changes:**

Syntax: ``git commit -m "Commit message"``

Example: ``git commit -m "Add new feature"``

- **Create a New Branch:**

Syntax: ``git branch <branch_name>``

Example: ``git branch feature-branch``

- **Switch to a Branch:**

Syntax: ``git checkout <branch_name>``

Example: ``git checkout feature-branch``

**OR**

Syntax: ``git switch <branch_name>`` (Git version 2.23 and later)

Example: ``git switch feature-branch``

- **Create and Switch to a New Branch:**

Syntax: ``git checkout -b <new_branch_name>``

Example: ``git checkout -b new-feature``

**OR**

Syntax: ``git switch -c <new_branch_name>`` (Git version 2.23 and later)

Example: ``git switch -c new-feature``

- **Merge Changes from One Branch to Another:**

Syntax: ``git merge <branch_name>``

Example: ``git merge feature-branch``

- **View the Commit History:**

Syntax: ``git log``

Example: ``git log``

- **Push Changes to a Remote Repository:**

Syntax: ``git push <remote_name> <branch_name>``

Example: ``git push origin master``

- **Pull Changes from a Remote Repository:**

Syntax: ``git pull <remote_name> <branch_name>``

Example: ``git pull origin master``

- **Show the Differences Between Working Directory and Staging Area:**

Syntax: ``git diff``

Example: ``git diff``

- **Show the Differences Between Staging Area and Last Commit:**

Syntax: ``git diff --cached``

Example: ``git diff --cached``

- **Show the Differences Between Working Directory and Last Commit:**

Syntax: ``git diff HEAD``

Example: ``git diff HEAD``

## **GITHUB**

GitHub is an increasingly popular programming resource used for code sharing. It's a social networking site for programmers that many companies and organizations use to facilitate project management and collaboration. According to statistics collected in October 2020, it is the most prominent source code host, with over 60 million new repositories created in 2020 and boasting over 56 million total developers.

GitHub is a Git repository hosting service that provides a web-based graphical interface. It is the world's largest coding community. Putting a code or a project into GitHub brings it increased, widespread exposure. Programmers can find source codes in many different languages and use the command-line interface, Git, to make and keep track of any changes. GitHub helps every team member work together on a project from any location while facilitating collaboration. You can also review previous versions created at an earlier point in time.

### **GitHub's Features?**

1. **Easy Project Management:** GitHub is a place where project managers and developers come together to coordinate, track, and update their work so that projects are transparent and stay on schedule.
2. **Increased Safety With Packages** Packages can be published privately, within the team, or publicly to the open-source community. The packages can be used or reused by downloading them from GitHub.
3. **Effective Team Management** GitHub helps all the team members stay on the same page and organized. Moderation tools like Issue and Pull Request Locking help the team to focus on the code.
4. **Improved Code Writing** Pull requests help the organizations to review, develop, and propose new code. Team members can discuss any implementations and proposals through these before changing the source code.
5. **Increased Code Safety** GitHub uses dedicated tools to identify and analyze vulnerabilities to the code that other tools tend to miss. Development teams everywhere work together to secure the software supply chain, from start to finish.
6. **Easy Code Hosting** All the code and documentation are in one place. There are millions of repositories on GitHub, and each repository has its own tools to help you host and release code.

## **HOSTING SERVICE FOR GIT REPOSITORY**

When it comes to hosting Git repositories, various platforms provide a robust infrastructure for collaborative development, version control, and project management. Choosing the right hosting service depends on factors like ease of use, collaboration features, and integration capabilities. Here are some popular Git hosting services widely utilized in the software development community:

1. **GitHub:** GitHub stands out as one of the most prevalent Git hosting platforms, offering a user-friendly interface, powerful collaboration features, and seamless integration with various tools. It serves as an ideal choice for open-source projects, private repositories, and team collaboration.
2. **GitLab:** GitLab is a comprehensive web-based Git repository manager that not only provides source code management but also includes features like continuous integration. It caters to both cloud-based and self-hosted solutions, giving users flexibility in hosting their repositories.
3. **Bitbucket:** Owned by Atlassian, Bitbucket is another popular Git repository hosting service. Supporting both Git and Mercurial repositories, it offers features like code collaboration, issue tracking, and continuous integration. Bitbucket is often preferred by teams using other Atlassian tools such as Jira and Confluence.
4. **GitKraken Glo Boards:** GitKraken Glo Boards is an integrated task and issue tracking service linked with GitKraken, a Git client. This platform allows teams to manage tasks directly associated with their Git repositories and provides a visual approach to monitoring project progress.
5. **SourceForge:** SourceForge, with a long history, hosts open-source software projects and offers version control, bug tracking, and project management tools. While not as prominent as some other options, it remains a viable choice for numerous projects.
6. **AWS CodeCommit:** As part of Amazon Web Services (AWS), AWS CodeCommit is a fully managed source control service. It seamlessly integrates with other AWS services and provides a secure and scalable environment for hosting Git repositories.

*Selecting the most suitable Git hosting service depends on your team's requirements, project size, and preferences for cloud-based or self-hosted solutions. Each platform has its strengths, catering to specific use cases within the software development landscape.*

## **Difference between GIT and GITHUB**

Git and GitHub are related concepts but serve different purposes in the context of version control and collaborative software development.

**Git:** Git is a distributed version control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals include speed, data integrity, and support for distributed, non-linear workflows.

**GitHub:** GitHub is a web-based Git repository hosting service, which offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features.



<b>S.No.</b>	<b>Git</b>	<b>GitHub</b>
1	Git is a software.	GitHub is a service.
2	Git is a command-line tool	GitHub is a graphical user interface
3	Git is installed locally on the system	GitHub is hosted on the web
4	Git is maintained by linux.	GitHub is maintained by Microsoft.
5	Git is focused on version control and code sharing.	GitHub is focused on centralized source code hosting.
6	Git is a version control system to manage source code history.	GitHub is a hosting service for Git repositories.
7	Git was first released in 2005.	GitHub was launched in 2008.
8	Git has no user management feature.	GitHub has a built-in user management feature.
9	Git is open-source licensed.	GitHub includes a free-tier and pay-for-use tier.
10	Git has minimal external tool configuration.	GitHub has an active marketplace for tool integration.
11	Git provides a Desktop interface named Git Gui.	GitHub provides a Desktop interface named GitHub Desktop.
12	Git competes with CVS, Azure DevOps Server, Subversion, Mercurial, etc.	GitHub competes with GitLab, Bit Bucket, AWS Code Commit, etc.

### Introduction of Maven:

Apache Maven is an automation tool. The tool is written in Java. It was initially released on 13 July 2004. It is developed by the Apache software foundation. It is part of the Jakarta Project. It is working on two aspects: how software is built, and its dependencies. It was created by Jason van Zyl. It is built by using a plugin-based architecture that allows it to make the use of any application controllable by standard input. It dynamically downloads Java libraries.

Understanding the problem without Maven

There are many problems that we face during the project development. They are discussed below:

- 1) Adding set of Jars in each project:** In case of struts, spring, hibernate frameworks, we need to add set of jar files in each project. It must include all the dependencies of jars also.
- 2) Creating the right project structure:** We must create the right project structure in servlet, struts etc, otherwise it will not be executed.
- 3) Building and Deploying the project:** We must have to build and deploy the project so that it may work.

Maven simplifies the above-mentioned problems. It does mainly following tasks.

1. It makes a project easy to build
2. It provides uniform build process (maven project can be shared by all the maven projects)
3. It provides project information (log document, cross referenced sources, mailing list, dependency list, unit test reports etc.)
4. It is easy to migrate for new features of Maven

Apache Maven helps to manage

- Builds
- Documentation
- Reporting
- SCMs
- Releases
- Distribution

What is Build Tool?

A build tool is used for building a process. It does following:

- Generates source code (if auto-generated code is used)
- Generates documentation from source code
- Compiles source code
- Packages compiled code into JAR or ZIP file
- Installs the packaged code in local repository, server repository, or central repository

### Installation of Maven:

To install maven on windows, you need to perform following steps:

1. Download maven and extract it
2. Add JAVA\_HOME and MAVEN\_HOME in environment variable
3. Add maven path in environment variable
4. Verify Maven

1) Download Maven

To install maven on windows, you need to download apache maven first.

Download Maven latest Maven software from Download latest version of Maven  
For example: **apache-maven-3.1.1-bin.zip**

2) Add MAVEN\_HOME in environment variable **Right click on MyComputer -> properties -> Advanced System Settings -> Environment variables -> click new button**

Now **add MAVEN\_HOME** in variable name and path of maven in variable value. It must be the home directory of maven i.e. outer directory of bin. For example: **E:\apache-maven-3.1.1**

3) Add Maven Path in environment variable

Click on new tab if path is not set, then set the path of maven. If it is set, edit the path and append the path of maven. Here, we have installed JDK and its path is set by default, so we are going to append the path of maven.

The path of maven should be **%maven home%/bin**. For example, **E:\apache-maven-3.1.1\bin**.

4) Verify maven **To verify whether maven is installed or not, open the command prompt and write: mvn -version**

Now it will display the version of maven and jdk including the maven home and java home.

**POM FILES:** POM is an acronym for Project Object Model. The pom.xml file contains information of project and configuration information for the maven to build the project such as dependencies, build directory, source directory, test source directory, plugin, goals etc.

Maven reads the pom.xml file, then executes the goal.

Before maven 2, it was named as project.xml file. But, since maven 2 (also in maven 3), it is renamed as pom.xml.

Elements of maven pom.xml file

For creating the simple pom.xml file, you need to have following elements:

Element	Description
project	It is the root element of pom.xml file.
modelVersion	It is the sub element of project. It specifies the modelVersion. It should be set to
groupId	It is the sub element of project. It specifies the id for the project group.
artifactId	It is the sub element of project. It specifies the id for the artifact (project). An artifact is something that is either produced or used by a project. Examples of artifacts produced by Maven for a project include: JARs, source and binary distributions, and WARs.
version	It is the sub element of project. It specifies the version of the artifact under group.

## Maven pom.xml file with additional elements

Here, we are going to add other elements in pom.xml file such as:

Element	Description
packaging	defines packaging type such as jar, war etc.
name	defines name of the maven project.
url	defines url of the project.
dependencies	defines dependencies for this project.
dependency	defines a dependency. It is used inside dependencies.
scope	defines scope for this maven project. It can be compile, provided, runtime and system.

*Example for pom.xml file*

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.javatpoint.application1</groupId>
<artifactId>my-application1</artifactId>
<version>1.0</version>
<packaging>jar</packaging>
<name>Maven Quick Start Archetype</name>
<url>http://maven.apache.org</url>
<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.8.2</version>
<scope>test</scope>
</dependency>
</dependencies>
</project>
```

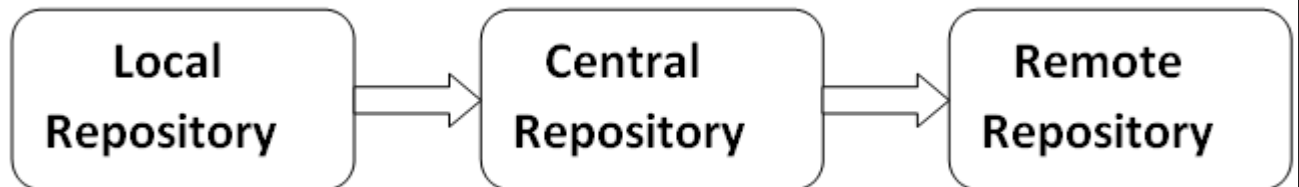
## Maven Repository

A maven repository is a directory of packaged JAR file with pom.xml file. Maven searches for dependencies in the repositories. There are 3 types of maven repository:

1. Local Repository
2. Central Repository
3. Remote Repository

Maven searches for the dependencies in the following order:

Local repository then Central repository then Remote repository.



If dependency is not found in these repositories, maven stops processing and throws an error.

#### 1) Maven Local Repository

Maven local repository is located in your local system. It is created by the maven when you run any maven command.

By default, maven local repository is `%USER_HOME%\.m2` directory. For example: `C:\Users\SSS IT\.m2`.

#### Update location of Local Repository

We can change the location of maven local repository by changing the `settings.xml` file. It is located in `MAVEN_HOME/conf/settings.xml`, for example: `E:\apache-maven-3.1.1\conf\settings.xml`.

#### 2) Maven Central Repository

Maven central repository is located on the web. It has been created by the apache maven community itself.

The path of central repository is: <http://repo1.maven.org/maven2/>.

The central repository contains a lot of common libraries that can be viewed by this url <http://search.maven.org/#browse>.

#### 3) Maven Remote Repository

Maven remote repository is located on the web. Most of libraries can be missing from the central repository such as JBoss library etc, so we need to define remote repository in `pom.xml` file.

Let's see the code to add the junit library in `pom.xml` file.



*pom.xml*

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.javatpoint.application1</groupId>
  <artifactId>my-application1</artifactId>
  <version>1.0</version>
  <packaging>jar</packaging>
  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.8.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Any repository of maven is available in **mvnrepository.com**

### **Maven Plugins:**

The **maven plugins** are central part of maven framework, it is used to perform specific goal.

According to Apache Maven, there are 2 types of maven plugins.

1. Build Plugins
2. Reporting Plugins

#### **Build Plugins**

These plugins are executed at the time of build. These plugins should be declared inside the **<build>** element.

#### **Reporting Plugins**

These plugins are executed at the time of site generation. These plugins should be declared inside the **<reporting>** element.

#### **Maven Core Plugins**

A list of maven core plugins are given below:

Plugin	Description
clean	clean up after build.
compiler	compiles java source code.
deploy	deploys the artifact to the remote repository.
failsafe	runs the JUnit integration tests in an isolated classloader.
install	installs the built artifact into the local repository.
resources	copies the resources to the output directory for including in the JAR.
site	generates a site for the current project.
surefire	runs the JUnit unit tests in an isolated classloader.
verifier	verifies the existence of certain conditions. It is useful for integration tests.

Example for maven plugin:

**The *compiler* plugin is used to compile the source code of a Maven project.** This plugin has two goals, which are already bound to specific phases of the default lifecycle:

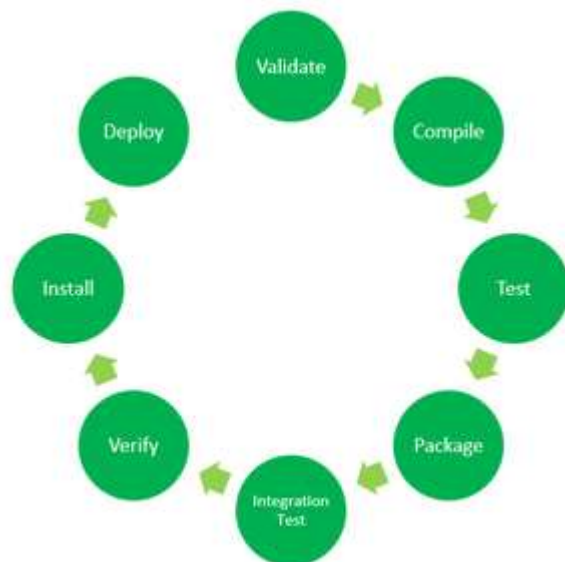
- *compile* – compile main source files
- *testCompile* – compile test source files

Here's the *compiler* plugin in the POM:

```
<plugin>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.12.1</version>
  <configuration>
    <source>1.8</source>
    <target>1.8</target>
  </configuration>
</plugin>
```

## Maven Build Lifecycle:

A Build Lifecycle is a well-defined sequence of phases, which define the order in which the goals are to be executed. Here phase represents a stage in life cycle. As an example, a typical **Maven Build Lifecycle** consists of the following sequence of phases.



- **Validate:** This step validates if the project structure is correct. For example – It checks if all the dependencies have been downloaded and are available in the local repository.
- **Compile:** It compiles the source code, converts the .java files to .class, and stores the classes in the target/classes folder.
- **Test:** It runs unit tests for the project.
- **Package:** This step packages the compiled code in a distributable format like JAR or WAR.
- **Integration test:** It runs the integration tests for the project.
- **Verify:** This step runs checks to verify that the project is valid and meets the quality standards.
- **Install:** This step installs the packaged code to the local Maven repository.
- **Deploy:** It copies the packaged code to the remote repository for sharing it with other developers.

There are always **pre** and **post** phases to register **goals**, which must run prior to, or after a particular phase.

When Maven starts building a project, it steps through a defined sequence of phases and executes goals, which are registered with each phase.

Maven has the following three standard lifecycles –

- **default:** This is the main lifecycle, as it's responsible for project deployment.
- **clean:** Handles project cleaning, ensuring that all artifacts generated by previous builds are removed.
- **site:** Manages the creation of the project's site documentation.

### Default Lifecycle (default)

Most Maven users will be familiar with the default lifecycle. It is a general model of a build process for a software application. The first phase is validate and the last phase is deploy.

- **Validate:** This step validates if the project structure is correct. For example – It checks if all the dependencies have been downloaded and are available in the local repository.
- **Compile:** It compiles the source code, converts the .java files to .class, and stores the classes in the target/classes folder.
- **Test:** It runs unit tests for the project.
- **Package:** This step packages the compiled code in a distributable format like JAR or WAR.
- **Integration test:** It runs the integration tests for the project.
- **Verify:** This step runs checks to verify that the project is valid and meets the quality standards.
- **Install:** This step installs the packaged code to the local Maven repository.
- **Deploy:** It copies the packaged code to the remote repository for sharing it with other developers.

### Clean Lifecycle (clean)

The first lifecycle in Maven. Running mvn clean invokes the clean lifecycle which consists of three lifecycle phases:

- pre-clean : execute processes needed prior to the actual project cleaning
- clean : remove all files generated by the previous build
- post-clean : execute processes needed to finalize the project cleaning

### Site Lifecycle (site)

Maven does more than build software artifacts from project, it can also generate project documentation and reports about the project, or a collection of projects. Project documentation and site generation have a dedicated lifecycle which contains four phases:

1. pre-site : execute processes needed prior to the actual project site generation
2. site : generate the project's site documentation
3. post-site : execute processes needed to finalize the site generation, and to prepare for site deployment
4. site-deploy: deploy the generated site documentation to the specified web server

### Maven Profiles:

A Build profile is a set of configuration values, which can be used to set or override default values of Maven build. Using a build profile, you can customize build for different environments such as Production v/s Development environments.

### **Types of Build Profile**

Build profiles are majorly of three types.

#### **Type Where it is defined**

Per Project Defined in the project POM file, pom.xml

User Defined in Maven settings xml file (%USER\_HOME%/.m2/settings.xml)

Global Defined in Maven global settings xml file (%M2\_HOME%/conf/settings.xml)

### **Profile Activation**

A Maven Build Profile can be activated in various ways.

- Explicitly using command console input.
- Through maven settings.
- Based on environment variables (User/System variables).
- OS Settings (for example, Windows family).
- Present/missing files.

Now, under **src/main/resources**, there are three environment specific files –

#### **File Name & Description**

Sl.No

##### **env.properties**

1

default configuration used if no profile is mentioned.

##### **env.test.properties**

2

test configuration when test profile is used.

##### **env.prod.properties**

3

production configuration when prod profile is used.

### **Explicit Profile Activation**

In the following example, we will attach maven-antrun-plugin:run goal to test the phase. This will allow us to echo text messages for different profiles. We will be using pom.xml to define different profiles and will activate profile at command console using maven command.



Assume, we've created the following pom.xml in C:\MVN\project folder.

```
<project xmlns = "http://maven.apache.org/POM/4.0.0"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation = "http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.companyname.projectgroup</groupId>
  <artifactId>project</artifactId>
  <version>1.0</version>
  <profiles>
    <profile>
      <id>test</id>
      <build>
        <plugins>
          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-antrun-plugin</artifactId>
            <version>1.1</version>
            <executions>
              <execution>
                <phase>test</phase>
                <goals>
                  <goal>run</goal>
                </goals>
              </execution>
            </executions>
            <configuration>
```

```

        <tasks>

            <echo>Using env.test.properties</echo>

            <copy file="src/main/resources/env.test.properties"

                tofile="${project.build.outputDirectory}/env.properties"/>

        </tasks>

    </configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</profile>

</profiles>

</project>

```

Now open the command console, go to the folder containing pom.xml and execute the following **mvn** command. Pass the profile name as argument using -P option.

```
C:\MVN\project>mvn test -Ptest
```

Maven will start processing and displaying the result of test build profile.

### Profile Activation via Maven Settings

Open Maven **settings.xml** file available in %USER\_HOME%/.m2 directory where **%USER\_HOME%** represents the user home directory. If settings.xml file is not there, then create a new one.

Add test profile as an active profile using active Profiles node as shown below in example.

```

<settings xmlns = "http://maven.apache.org/POM/4.0.0"

    xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

```

```
http://maven.apache.org/xsd/settings-1.0.0.xsd">

<mirrors>

  <mirror>

    <id>maven.dev.snaponglobal.com</id>

    <name>Internal Artifactory Maven repository</name>

    <url>http://repo1.maven.org/maven2/</url>

    <mirrorOf>*</mirrorOf>

  </mirror>

</mirrors>

<activeProfiles>

  <activeProfile>test</activeProfile>

</activeProfiles>

</settings>
```

Now open command console, go to the folder containing pom.xml and execute the following **mvn** command. Do not pass the profile name using -P option. Maven will display result of test profile being an active profile.

```
C:\MVN\project>mvn test
```

### **Profile Activation via Environment Variables**

Now remove active profile from maven settings.xml and update the test profile mentioned in pom.xml. Add activation element to profile element as shown below.

The test profile will trigger when the system property "env" is specified with the value "test". Create an environment variable "env" and set its value as "test".

```
<profile>

  <id>test</id>

  <activation>

    <property>

      <name>env</name>
```

```
<value>test</value>
```

```
</property>
```

```
</activation>
```

```
</profile>
```

Let's open command console, go to the folder containing pom.xml and execute the following **mvn** command.

```
C:\MVN\project>mvn test
```

### Profile Activation via Operating System

Activation element to include os detail as shown below. This test profile will trigger when the system is windows XP.

```
<profile>
```

```
<id>test</id>
```

```
<activation>
```

```
<os>
```

```
<name>Windows XP</name>
```

```
<family>Windows</family>
```

```
<arch>x86</arch>
```

```
<version>5.1.2600</version>
```

```
</os>
```

```
</activation>
```

```
</profile>
```

Now open command console, go to the folder containing pom.xml and execute the following **mvn** commands. Do not pass the profile name using -P option. Maven will display result of test profile being an active profile.

```
C:\MVN\project>mvn test
```

### Profile Activation via Present/Missing File

Now activation element to include OS details as shown below. The test profile will trigger when **target/generated-sources/axistools/wsdl2java/com/companyname/group** is missing.

```
<profile>

  <id>test</id>

  <activation>

    <file>

      <missing>target/generated-sources/axistools/wsdl2java/

        com/companyname/group</missing>

      </file>

    </activation>

  </profile>
```

Now open the command console, go to the folder containing pom.xml and execute the following **mvn** commands. Do not pass the profile name using -P option. Maven will display result of test profile being an active profile. C:\MVN\project>mvn test

### **Maven create and build artifacts:**

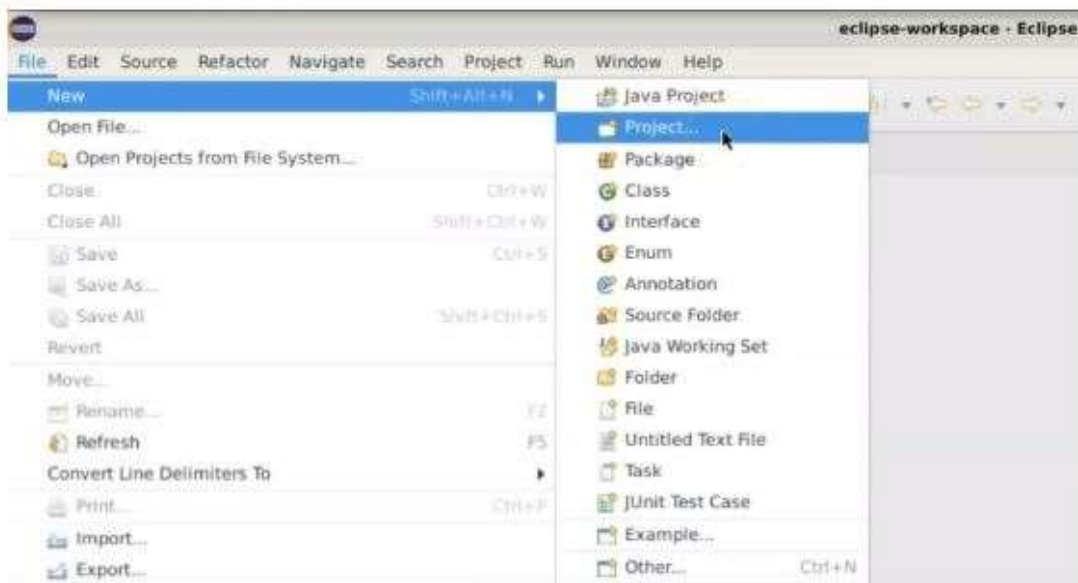
The first step is to open Eclipse, which comes with the integrated [Maven environment](#).

The Eclipse window opens on the screen. Complete the following steps:

- Go to the File option
- In the drop-down menu, select New
- Select the Project option

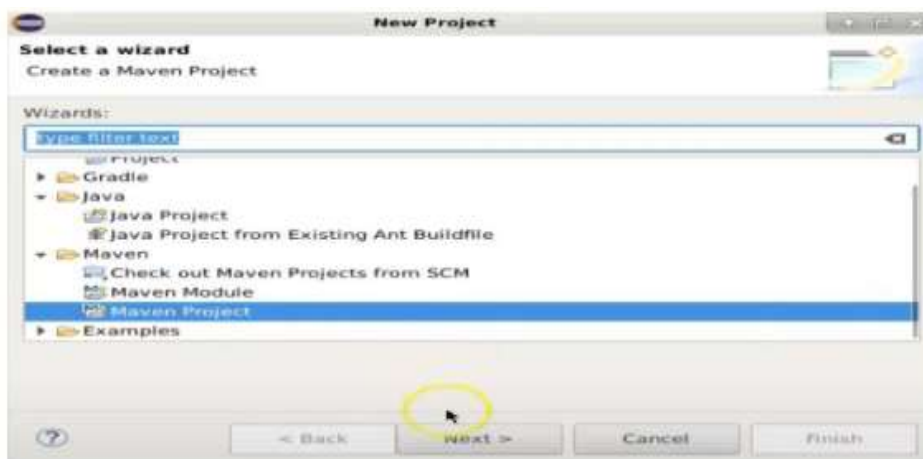
If you want to create a Java project, you can select the “Java Project” option. Since we are not creating a Java project specifically, we have chosen the “Project” option.





The dialog box that appears on the screen will display different types of projects.

- Select the Maven Project option
- Click on Next

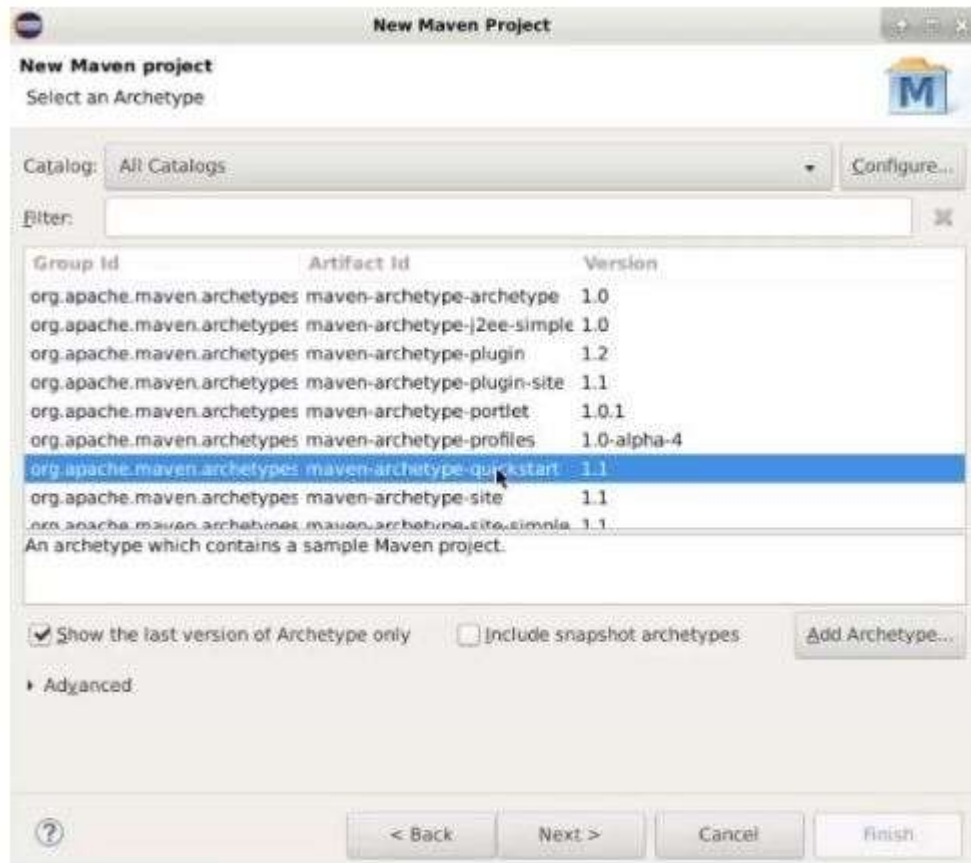


A dialog box will appear. Select the default workspace.

- Click on “Next”

Several Group IDs, Artifact IDs, and Versions will then appear.

- Select a plugin there and click on “Next”

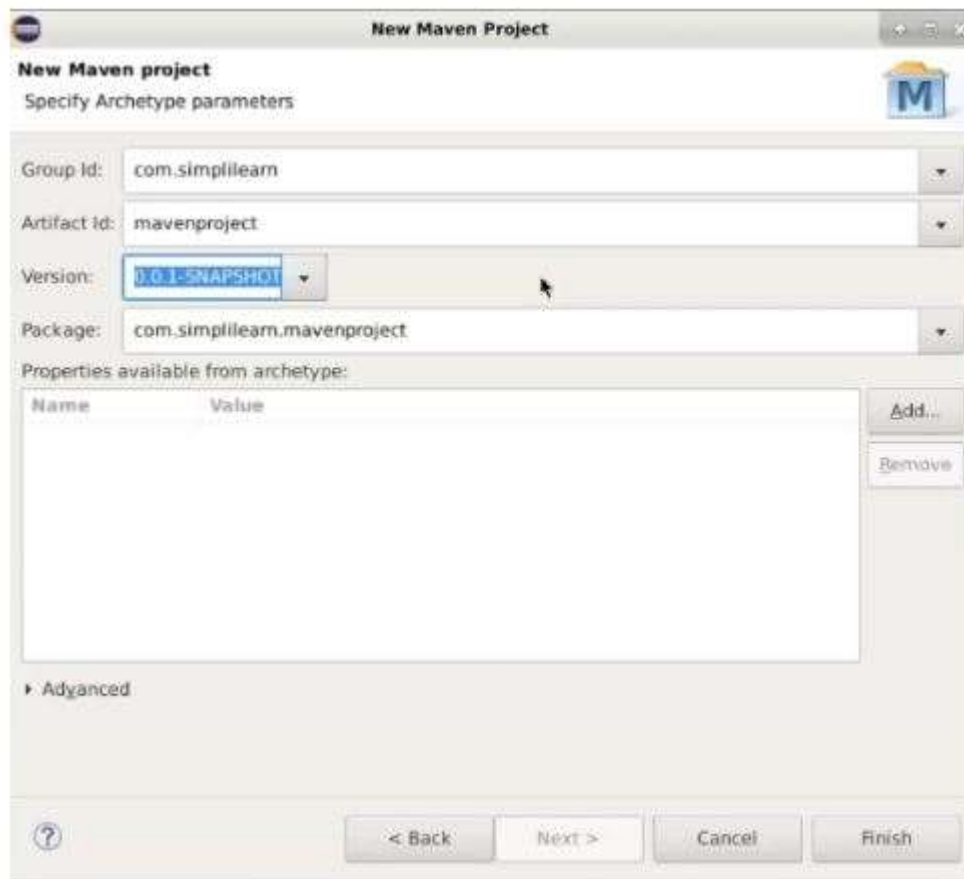


In the next dialog box that appears, you'll complete the following steps:

- Enter the Group ID  
"com.xyz"
- Enter the Artifact ID  
"mavenproject"
- The version will appear on the screen

These items can all be modified at a later time if needed.

- Click on "Finish"



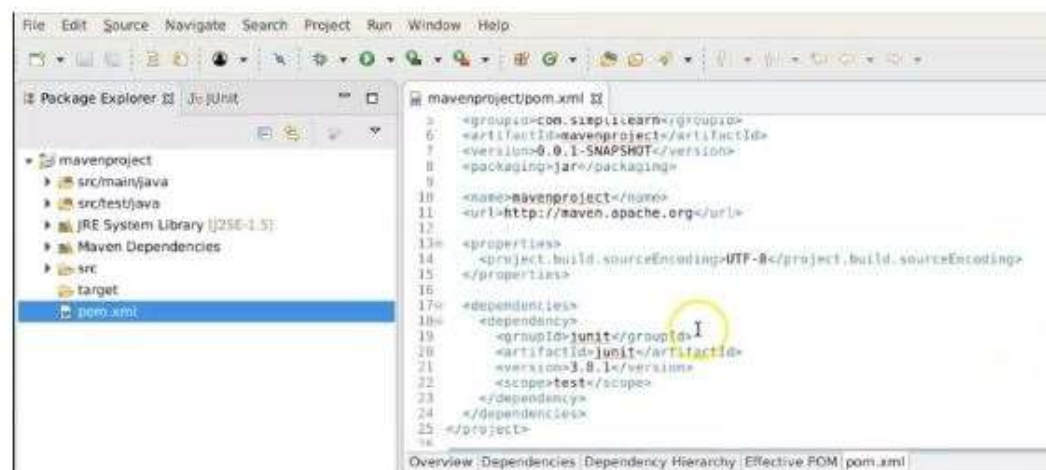
The project is now created.

- Open the pom.xml file

You can see all the basic information that you have entered on the screen, such as the Artifact ID, Group ID, etc.

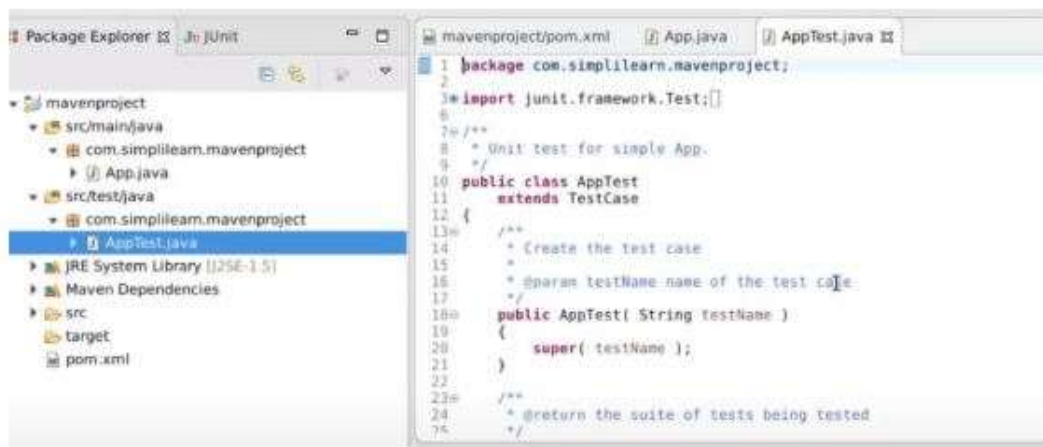
You can see the junit dependencies have been added.

This process takes place by default in Eclipse. There will also be some by default test cases.



There you can find AppTest.java to be a default test case.

When you click on that, you can see the test cases written in JUnit on your Eclipse screen.

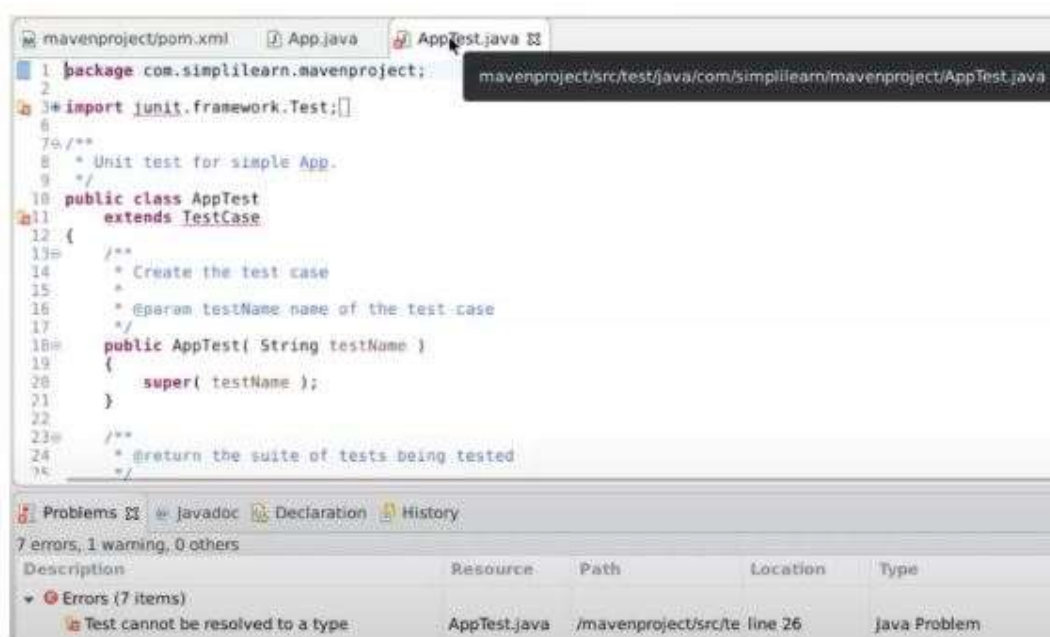


When it comes to adding more test cases, it will depend on the user, but these test cases and commands can easily be added in the workspace.

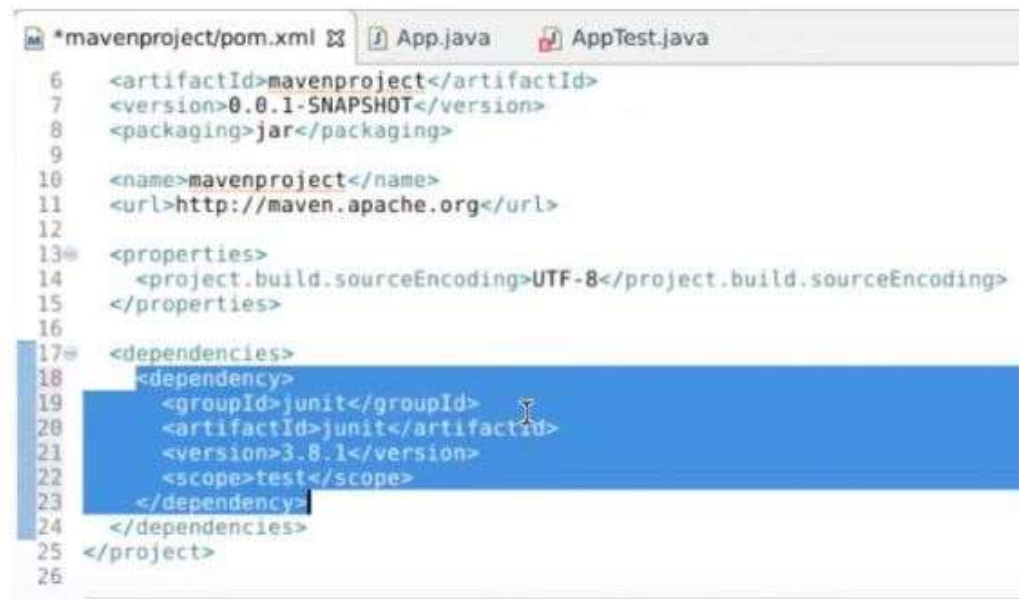
If we try to remove certain dependencies from our file, we will receive error messages. To troubleshoot this, complete the following steps:

- Go to another tab: mavenproject/pom.xml
- Delete any dependencies
- Save the file

Immediately, there will be several error messages in the AppTest.java.



Return to the previous screen and undo the deletion. The errors that occurred will disappear.



```

6 <artifactId>mavenproject</artifactId>
7 <version>0.0.1-SNAPSHOT</version>
8 <packaging>jar</packaging>
9
10 <name>mavenproject</name>
11 <url>http://maven.apache.org</url>
12
13 <properties>
14   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15 </properties>
16
17 <dependencies>
18   <dependency>
19     <groupId>junit</groupId>
20     <artifactId>junit</artifactId>
21     <version>3.8.1</version>
22     <scope>test</scope>
23   </dependency>
24 </dependencies>
25 </project>
26

```

### Dependency Management:

The *dependencyManagement* and *dependencies* are especially useful for multi-module projects.

### Dependency Management :

This tag consists of a *dependencies* tag which itself might contain multiple *dependency* tags. Each *dependency* is supposed to have at least three main tags: *groupId*, *artifactId*, and *version*. For example:

```
<dependencyManagement>
```

```
<dependencies>
```

```
<dependency>
```

```
<groupId>org.apache.commons</groupId>
```

```
<artifactId>commons-lang3</artifactId>
```

```
<version>3.14.0</version>
```

```
</dependency>
```

```
</dependencies>
```

```
</dependencyManagement>
```



The above code just declares the new artifact *commons-lang3*, but it doesn't really add it to the project dependency resource list.

### ***Dependencies:***

This tag contains a list of *dependency* tags. Each *dependency* is supposed to have at least two main tags, which are *groupId* and *artifactId*.

For example:

```
<dependencies>

  <dependency>

    <groupId>org.apache.commons</groupId>

    <artifactId>commons-lang3</artifactId>

    <version>3.14.0</version>

  </dependency>

</dependencies>
```

The *version* and *scope* tags can be inherited implicitly if we have used the *dependencyManagement* tag before in the POM file. *DependencyManagement* is just a declaration, and it does not really add a dependency. *Dependencies* tag adds the actual dependency to the project.

An example for adding the JUnit library dependency:

```
<dependencyManagement>

  <dependencies>

    <dependency>

      <groupId>junit</groupId>

      <artifactId>junit</artifactId>

      <version>4.13.2</version>

      <scope>test</scope>

    </dependency>

  </dependencies>

</dependencyManagement>
```

**Introduction of Gradle:**

The list of features that Gradle provides,

- Gradle is available with separate Domain Specific Language (DSL) based on Groovy language.
- It provides the declarative language elements. Those elements also provide build-by-convention support for Java, Groovy, OSGI, Web and Scala.

**Language for dependency based programming**

The declarative language lies on a top of a general purpose task graph, which can be fully supported in the build.

**Structure your build**

Gradle allows you to apply common design principles to your build. It will give you a perfect structure for build, so that, you can design well-structured and easily maintained, comprehensible build.

**Deep API**

By using this API, you can monitor and customise its configuration and execution behavior to the core.

**Gradle scales**

Gradle can easily increase the productivity, from simple and single project builds to huge enterprise multi-project builds.

**Multi-project builds**

Gradle supports the multi-project builds and partial builds. If you build a subproject, Gradle takes care of building all the subprojects, that the subproject depends on.

**Different ways to manage your builds**

Gradle supports different strategies to manage your dependencies.

Gradle is the first build integration tool

Gradle is fully supported for your ANT tasks, Maven and Ivy repository infrastructure for publishing and retrieving dependencies. It also provides a converter for turning a Maven pom.xml to Gradle script.

**Ease of migration**

Gradle can easily adapt to any structure. Therefore, you can always develop your Gradle build in the same branch, where you can build live script.

## Gradle Wrapper

Gradle Wrapper allows you to execute the Gradle builds on machines, where Gradle is not installed. This is useful for continuous integration of servers.

### Free open source

Gradle is an open source project, and licensed under the Apache Software License (ASL).

### Groovy

Gradle's build script are written in Groovy programming language. The whole design of Gradle is oriented towards being used as a language and not as a rigid framework. Groovy allows you to write your own script with some abstractions. The whole Gradle API is fully designed in Groovy language.

### Installation of Gradle:

#### Prerequisites to install Gradle

JDK and Groovy are the prerequisites for Gradle installation.

Gradle requires JDK version 6 or later to be installed in the system. It uses the JDK libraries which are installed, and sets to the JAVA\_HOME environmental variable.

Gradle carries its own Groovy library, therefore, we need not install Groovy explicitly. If it is installed, that is ignored by Gradle.

**The steps to install Gradle in your system are explained below.**

#### Step 1 – Verify JAVA Installation

First of all, you need to have Java Software Development Kit (SDK) installed on your system. To verify this, execute Java –version command in any of the platform you are working on.

#### In Windows

Execute the following command to verify Java installation. I have installed JDK 1.8 in my system.

```
C:\> java - version
```

#### Output

The output is as follows –

```
java version "1.8.0_66"
```

```
Java(TM) SE Runtime Environment (build 1.8.0_66-b18)
```

Java HotSpot(TM) 64-Bit Server VM (build 25.66-b18, mixed mode)

## Step 2 – Download Gradle Build File

Download the latest version of Gradle from the link available at <https://gradle.org/install/>.

## Step 3 – Set Up Environment for Gradle

Setting up environment means, we have to extract the distribution file and copy the library files into proper location. Set up **GRADLE\_HOME** and **PATH** environmental variables. This step is platform dependent.

### In Windows

Extract the downloaded zip file named **gradle-2.11-all.zip** and copy the distribution files from **Downloads\gradle-2.11\** to **C:\gradle\** location.

After that, add the **C:\gradle** and **C:\gradle\bin** directories to the **GRADLE\_HOME** and **PATH** system variables.

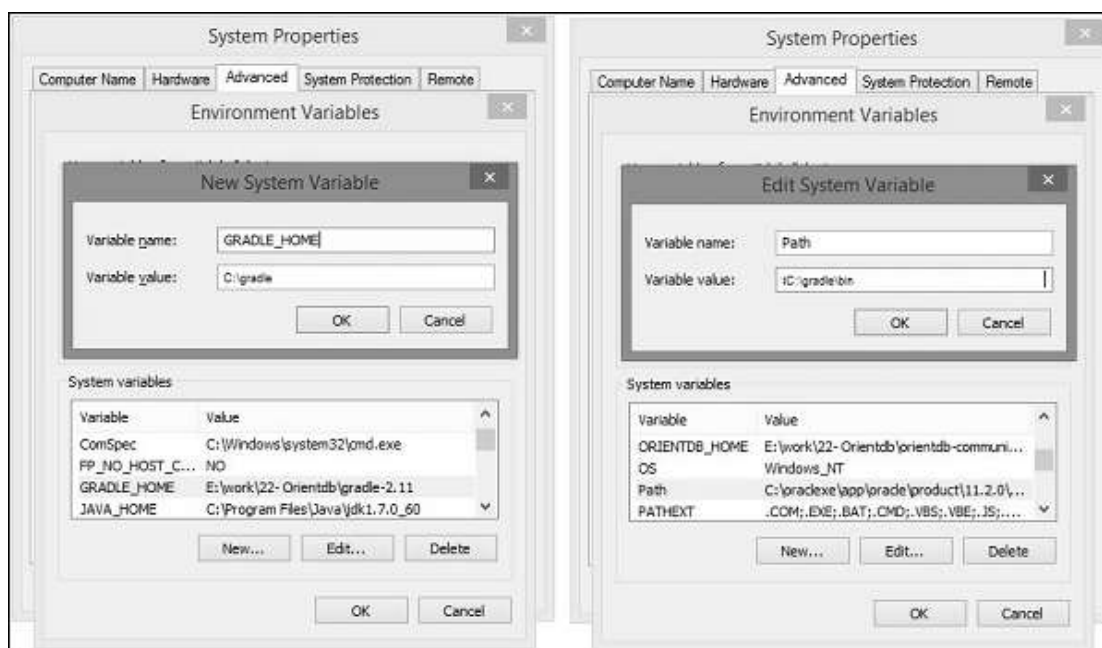
Follow the given instructions – **Right Click On My Computers -> Click On Properties -> Advanced System Settings -> Click On Environmental Variables.**

There you will find a dialog box for creating and editing system variables.

Click on new button for creating **GRADLE\_HOME** variable (follow the left side screenshot).

Click on Edit for editing the existing Path system variable (follow the right side screenshot).

Follow the below given screenshots.



## Step 4 – Verify the Gradle installation

### In windows

You can execute the following command in command prompt.

```
C:\> gradle -v
```

### Output

Here you will find the Gradle version.

```
-----  
Gradle 2.11  
-----
```

```
Build time: 2016-02-08 07:59:16 UTC
```

```
Build number: none
```

```
Revision: 584db1c7c90bdd1de1d1c4c51271c665bfcba978
```

```
Groovy: 2.4.4
```

```
Ant: Apache Ant(TM) version 1.9.3 compiled on December 23 2013
```

```
JVM: 1.7.0_60 (Oracle Corporation 24.60-b09)
```

```
OS: Windows 8.1 6.3 amd64
```

### Understanding build using Gradle:

The Gradle build is a process of creating a Gradle project. When we run a gradle command, it will look for a file called build.gradle in the current directory. This file is also called the Gradle build script. The build configuration, tasks, and plugins are described in this file. The build script describes a project and its tasks.

Let's create a small Gradle project, run some of the basic Gradle commands, and understand how Gradle manages the project.

Follow the steps below to create and test a Gradle project.

### Step1: Open the command line and create a directory



First, Open the command line and create a directory for the project and change directory to it.

Let's create a demo directory.

```
C:\Users\HiMaNshU>mkdir demo
C:\Users\HiMaNshU>cd demo
```

## Step2: Initialize a Gradle project

To generate a Gradle project, run the gradle init command. It will generate a simple project. With this project, we will explore and understand everything that is generated.

When we run the gradle init command, it will ask for some basic requirements. First, it will ask the type of project that we want to create. It will give four options:

1. 1: basic
2. 2: application
3. 3: library
4. 4: Gradle plugin

Select our requirements. Hence, we are just making a demo project so that we will select the basic option. To select basic option, press 1 and Enter key. Consider the below output:

```
C:\Users\HiMaNshU\demo>gradle init
Starting a Gradle Daemon, 1 incompatible and 1 stopped Daemons could
ed, use --status for details

Select type of project to generate:
 1: basic
 2: application
 3: library
 4: Gradle plugin
Enter selection (default: basic) [1..4] 1
```

Next, it will ask for **DSL**. There are two options that are available for DSL:

1. 1: Groovy
2. 2: Kotlin

Groovy is the default DSL provided by Gradle. Select **build script DSL**.

```
Select build script DSL:
 1: Groovy
 2: Kotlin
Enter selection (default: Groovy) [1..2] 1
```

Next, it will ask for the **project name**. Type the project name and press Enter key. It will take a while to build a project. After the successful execution of the project, we will get a message **BUILD SUCCESSFUL**.

```

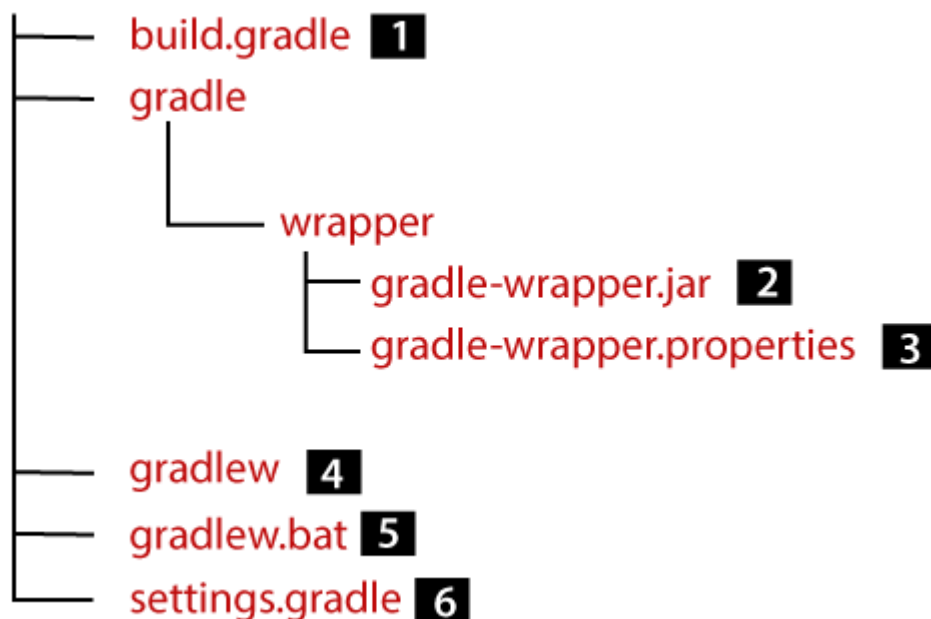
Project name (default: demo): First_Gradle

> Task :init
Get more help with your project: https://guides.gradle.org/creating-
uils

BUILD SUCCESSFUL in 2m 56s
2 actionable tasks: 2 executed
C:\Users\HiMaNshU\demo>mkdir src

```

Now we have successfully created a Gradle project. Now, what will happen to our specified directory? Consider the below structure of the Gradle project.



It is the default structure of a Gradle project. Gradle will generate the following things for us:

1. The **gradle** file is build script for configuring the current project.
2. An **executable JAR** file is used as a Gradle wrapper.
3. **Configuration properties** for Gradle Wrapper.
4. The **gradlew** is a Gradle wrapper script for UNIX based OS.
5. The **bat** is the Gradle Wrapper script for Windows.
6. The **settings script** for configuring the Gradle build.

### Step3: Create a task

Gradle supports APIs for creating and managing tasks through a Groovy-based DSL or Kotlin-based DSL. Every project contains a collection of tasks for some basic operation.

Gradle supports a library of tasks that configure the project. For example, there is a Copy task, which copies files from one location to another. The Copy task is one of the most used tasks In Gradle.

To use the Copy task in build script, follow the below process.

**Step1:** Create a directory called src

```
C:\Users\HiMaNshU\demo>mkdir src  
C:\Users\HiMaNshU\demo>cd src
```

**Step2:** Add a file called **myfile.txt** in the src directory. Add the single line "Hello, World!" to it, also, we can leave it empty.

```
C:\Users\HiMaNshU\demo\src>echo myfile.txt  
myfile.txt
```

Define a task called Copy in **build.gradle** file. It will copy the src directory to a new directory called dest. We don't have to create the dest directory; the Copy task will do it for us.

1. task copy(type: Copy, group: "Custom", description: "The sources are copied to dest d  
irectory") {
2. from "src"
3. into "dest"
4. }

We can provide anything in a group and description. Also, we can omit them, but doing so will also be omitted from the report of the task used later.

Now execute our new copy task:

```
C:\Users\HiMaNshU\demo>gradle copy  
  
BUILD SUCCESSFUL in 2s  
1 actionable task: 1 executed  
C:\Users\HiMaNshU\demo>
```

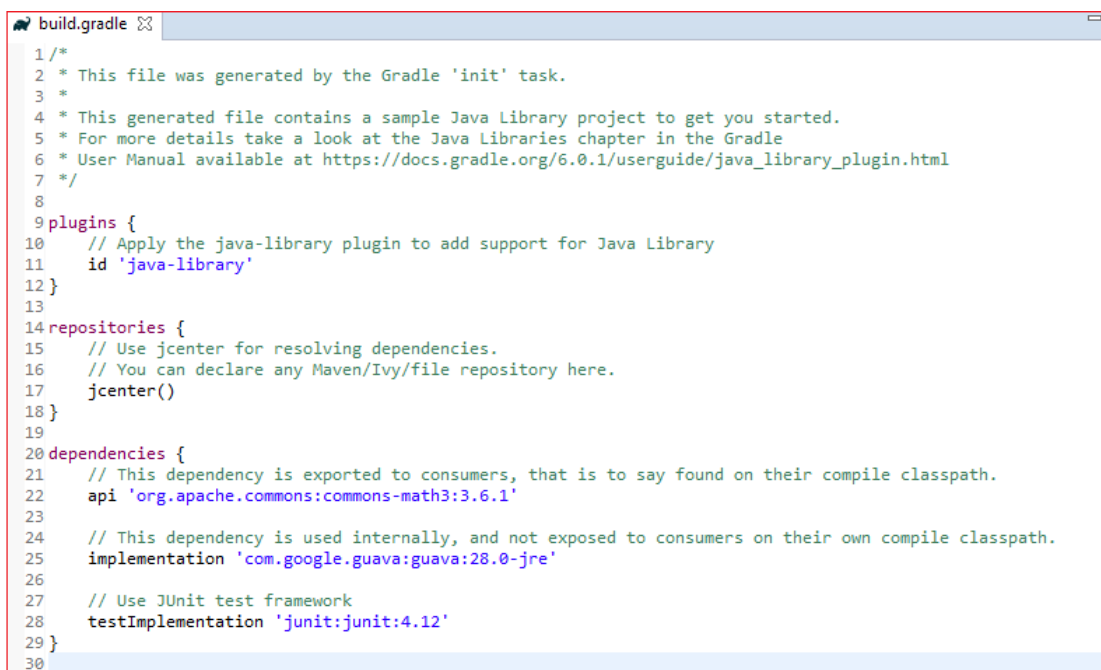
After the successful execution of the task, we will get **BUILD SUCCESSFUL** message.

The build.gradle file

The build.gradle file is build script of a Gradle project. All the tasks and plugins are defined in this file.

When we run a gradle command, it looks for a file called build.gradle in the current directory. Although we have called it a build script, strictly, it is a build configuration script. The build script defines a project and its tasks.

The default **build.gradle** file looks like as follows:



```

1 /*
2  * This file was generated by the Gradle 'init' task.
3  *
4  * This generated file contains a sample Java Library project to get you started.
5  * For more details take a look at the Java Libraries chapter in the Gradle
6  * User Manual available at https://docs.gradle.org/6.0.1/userguide/java_library_plugin.html
7  */
8
9 plugins {
10     // Apply the java-library plugin to add support for Java Library
11     id 'java-library'
12 }
13
14 repositories {
15     // Use jcenter for resolving dependencies.
16     // You can declare any Maven/Ivy/file repository here.
17     jcenter()
18 }
19
20 dependencies {
21     // This dependency is exported to consumers, that is to say found on their compile classpath.
22     api 'org.apache.commons:commons-math3:3.6.1'
23
24     // This dependency is used internally, and not exposed to consumers on their own compile classpath.
25     implementation 'com.google.guava:guava:28.0-jre'
26
27     // Use JUnit test framework
28     testImplementation 'junit:junit:4.12'
29 }
30

```

The **build.gradle** file contains three default sections. They are as follows:

- **plugins:** In this section, we can apply the java-library plugin to add support for java library.
- **Repositories:** In this section, we can declare internal and external repository for resolving dependencies. We can declare the different types of repository supported by Gradle like Maven, Ant, and Ivy.
- **Dependencies:** In this section, we can declare dependencies that are necessary for a particular subject.

Additionally, we can declare other project-related modules like a task in this file.

Display the Information of the Gradle project

To understand the structure, dependencies and debugging problems of a build, Gradle provides many built-in features that display information on a project.

Following are some basic commands to display the information of the project:

### Listing projects

In Gradle, all the sub-projects of a project in the workspace can be listed in a hierarchy. To do so, run the below command from the root directory of the project.

1. `gradle -q projects`

```

To see all tasks and more detail, run gradle tasks --all
To see more detail about a task, run gradle help --task <task>
C:\Users\HiMaNshU\eclipse-workspace>gradle -q projects

-----
Root project
-----

Root project 'eclipse-workspace'
No sub-projects

To see a list of the tasks of a project, run gradle <project-path>:tasks
For example, try running gradle :tasks
C:\Users\HiMaNshU\eclipse-workspace>

```

## Listing Tasks

Gradle allows us to list all the essential tasks of the project. To list the task, run the below command:

1. `gradle -q tasks`

### Output:

```
C:\Users\HiMaNshU\eclipse-workspace>gradle -q tasks
```

```
-----
```

Tasks runnable from root project

```
-----
```

Build Setup tasks

```
-----
```

init - Initializes a new Gradle build.

wrapper - Generates Gradle wrapper files.

Help tasks

```
-----
```

buildEnvironment - Displays all buildscript dependencies declared in root project

'eclipse-workspace'.

components - Displays the components produced by root project 'eclipse-workspace'

'. [incubating]

dependencies - Displays all dependencies declared in root project 'eclipse-works'

pace'.

dependencyInsight - Displays the insight into a specific dependency in root project 'eclipse-workspace.'

dependentComponents : It displays the dependent components of components in the root project 'eclipse-workspace.' [incubating]

help - Displays a help message.

model - Displays the configuration model of root project 'eclipse-workspace.' [incubating]

projects - Displays the sub-projects of root project 'eclipse-workspace.'

properties - Displays the properties of root project 'eclipse-workspace.'

tasks - Displays the tasks runnable from root project 'eclipse-workspace.'

To see all tasks and more detail, run gradle tasks --all

To see more detail about a task, run gradle help --task <task>

To list all the tasks of the project, run the below command:

1. gradle tasks -all

To display more details about a task, run the below command:

1. gradle help --task

### Listing Dependencies

In Gradle, we can list the dependencies which are broken down by the configuration. To list the dependencies, run the below command:

1. gradle -q dependencies

### Output:

```
C:\Users\HiMaNshU\eclipse-workspace>gradle -q dependencies

-----
Root project
-----

No configurations

A web-based, searchable dependency report is available by adding the --scan option.
```



Install & Configure Jenkins, Jenkins Architecture Overview, creating a Jenkins Job, configuring a Jenkins job, Introduction to Plugins, Adding Plugins to Jenkins, commonly used plugins (Git Plugin, Parameter Plugin, HTML Publisher, Copy Artifact and Extended choice parameters). Configuring Jenkins to work with java, Git and Maven, creating a Jenkins Build and Jenkins workspace.

## Installing and Configuring Jenkins

### Overview

Jenkins is a widely-used open-source automation server that helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery (CI/CD). Below are detailed steps and considerations for installing and configuring Jenkins.

### Installation

#### 1. System Requirements:

- **Operating System:** Jenkins can run on any major operating system.
- **Java:** Jenkins requires Java Runtime Environment (JRE) 8 or later.

#### 2. Download Jenkins:

- Download the latest version of Jenkins from the official Jenkins website.

#### 3. Installation Steps:

- **Windows:**
  - Run the `.msi` installer and follow the installation wizard.
  - After installation, Jenkins will start automatically as a Windows service.
- **Linux:**
  - Add Jenkins repository and import the GPG key.
  - Use package manager to install Jenkins (e.g., `apt-get install jenkins`).
  - Start Jenkins using `systemctl` (`systemctl start jenkins`).

#### 4. Initial Setup:

- After installation, Jenkins can be accessed through a web browser at `http://<your_server_ip_or_domain>:8080`.
- Unlock Jenkins by entering the initial admin password, which can be found in the `jenkins_home` directory.

#### 5. Customize Jenkins:

- Install suggested plugins or select the required plugins manually.
- Create the first admin user.

### Configuration

**1. Global Configuration:**

- Navigate to **Manage Jenkins > Configure System**.
- Set up global environment variables, email notifications, and other system-wide settings.

**2. Security Configuration:**

- Configure security settings under **Manage Jenkins > Configure Global Security**.
- Use the built-in user database or integrate with external authentication mechanisms like LDAP.

**3. Node Configuration:**

- Jenkins can distribute build loads to multiple nodes.
- Configure nodes by navigating to **Manage Jenkins > Manage Nodes and Clouds**.

**4. Job Configuration:**

- Create new jobs by clicking **New Item** on the Jenkins dashboard.
- Configure job-specific settings such as source code management, build triggers, and post-build actions.

**5. Pipeline Configuration:**

- Jenkins supports pipeline as code using **Jenkinsfile**.
- Define your pipeline stages in a **Jenkinsfile** placed in the root of your project repository.

**6. Plugins:**

- Enhance Jenkins functionality by installing plugins from the **Manage Jenkins > Manage Plugins** section.

**7. Backup and Restore:**

- Regularly back up the **jenkins\_home** directory, which contains Jenkins configurations, job configurations, and build history.
- Use plugins like **ThinBackup** for automated backups.

Ref: [https://youtube.com/playlist?list=PL6fErFppaj35spJjPy41-lruDjw2kRV-&si=Vzd9AAMv53xwWB\\_S](https://youtube.com/playlist?list=PL6fErFppaj35spJjPy41-lruDjw2kRV-&si=Vzd9AAMv53xwWB_S)

Ref: [https://bit.ly/youtube\\_jenkins](https://bit.ly/youtube_jenkins) ([click here](#))

## Jenkins Architecture Overview

Jenkins is a robust automation server widely used for continuous integration and continuous delivery (CI/CD) processes. Here's a detailed breakdown of the Jenkins architecture and its core components.

### Core Components of Jenkins Architecture

#### 1. Jenkins Master:

- **Role:** The Jenkins master is the central control unit.
- **Functions:**
  - **Scheduling Jobs:** Assigns build jobs to the appropriate agents.
  - **Monitoring Agents:** Keeps track of the state and health of agents.
  - **Job Execution:** Executes build jobs directly if no agents are available.
  - **User Interface:** Provides a web-based interface for users to configure jobs, view build results, and manage the Jenkins environment.

#### 2. Build Agents (Nodes):

- **Role:** Agents are responsible for executing the build jobs assigned by the master.
- **Types:** Agents can run on different operating systems and hardware configurations.
- **Configuration:**
  - **Static Agents:** Permanently assigned to the Jenkins master.
  - **Dynamic Agents:** Created and destroyed as needed, often used in cloud environments.
- **Communication:** Agents communicate with the master using the Jenkins Remoting protocol.

#### 3. Job Configuration:

- **Types of Jobs:**
  - **Freestyle Projects:** Basic job type with simple configurations.
  - **Pipeline Jobs:** Scripted or declarative pipelines that define the entire build process as code.
  - **Multi-Configuration (Matrix) Projects:** Allows testing across different environments and configurations.
- **Components:**
  - **Source Code Management:** Integration with version control systems like Git, SVN, etc.
  - **Build Triggers:** Conditions that start the job, such as code commits, scheduled times, or manual triggers.
  - **Build Steps:** Actions performed during the build, such as compiling code, running tests, and packaging artifacts.
  - **Post-Build Actions:** Steps executed after the build, like deploying artifacts, sending notifications, or archiving results.

#### 4. Plugins:

- **Role:** Extend Jenkins functionality without modifying the core software.
- **Types:**
  - **Source Control Plugins:** Git, SVN, Mercurial.
  - **Build Tools Plugins:** Maven, Gradle, Ant.
  - **Notification Plugins:** Email, Slack, HipChat.
  - **Reporting Plugins:** JUnit, Cobertura, Checkstyle.
- **Management:** Plugins can be installed, updated, and configured via the Jenkins web interface.

#### 5. Master-Slave Architecture:

- **Concept:** The Jenkins master distributes build tasks to multiple agents, enabling distributed builds.
- **Benefits:**
  - **Scalability:** Handle more builds simultaneously.
  - **Resource Management:** Assign builds to agents with the necessary resources.
  - **Isolation:** Run builds in isolated environments to avoid conflicts.

### Workflow and Data Flow

#### 1. Job Trigger:

- Jobs can be triggered by various events:
  - **Manual Trigger:** Initiated by a user through the Jenkins interface.
  - **Scheduled Trigger:** Using CRON-like syntax to schedule jobs.
  - **SCM Trigger:** Automatically triggered by changes in the source code repository.
  - **Upstream/Downstream Trigger:** Triggered by the completion of other jobs.

#### 2. Build Execution:

- **Job Assignment:** The master assigns the job to an available agent based on labels, availability, and resource requirements.
- **Build Environment Setup:** The agent sets up the environment, including checking out the code, installing dependencies, and configuring the workspace.
- **Execution:** The agent runs the build steps as defined in the job configuration.

#### 3. Build Results:

- **Logs:** Captures console output and logs from the build process.
- **Artifacts:** Stores build artifacts like binaries, packages, and reports.
- **Test Results:** Collects and displays test results and code coverage reports.
- **Build History:** Maintains a history of all builds, including status, duration, and changes.

#### 4. Notification and Reporting:

- **Notifications:** Jenkins can send notifications through various channels (email, chat, etc.) upon build completion or failure.

- **Dashboards:** Provides dashboards for visualizing build status, trends, and metrics.
- **Reports:** Generates and displays reports on test results, code quality, and other metrics.

### Summary

Jenkins' architecture is designed to be flexible and scalable. The master-agent model allows for efficient distribution of build tasks, while plugins provide extensive customization options. By leveraging Jenkins' comprehensive job configuration and robust notification and reporting capabilities, teams can streamline their CI/CD pipelines and improve software quality and delivery speed.

<image>

<image>

<image>

<image>

### Creating a Jenkins Job

Creating a Jenkins job involves setting up a new project within Jenkins to automate various tasks such as building code, running tests, and deploying applications. Here is a step-by-step guide to creating a Jenkins job.

#### Prerequisites

- Jenkins installed and running.
- Basic understanding of the Jenkins interface.
- Access to the Jenkins dashboard.

#### Step-by-Step Guide

##### 1. Access Jenkins Dashboard:

- Open your web browser and navigate to your Jenkins instance URL (e.g., <http://localhost:8080>).

##### 2. Create a New Job:

- On the Jenkins dashboard, click on the "New Item" link on the left-hand side menu.

##### 3. Enter Job Name:

- Enter a name for your new job in the "Enter an item name" field.
  - Choose the type of job you want to create. For most uses, "Freestyle project" is a good starting point.
  - Click "OK".
4. **Configure the Job:**
- After clicking "OK", you will be directed to the job configuration page.
5. **General Configuration:**
- **Description:** Provide a brief description of the job.
  - **Discard Old Builds:** Optionally set this to limit the number of old builds Jenkins keeps.
6. **Source Code Management:**
- **Select Version Control System:** Choose the version control system (e.g., Git, Subversion).
  - **Repository URL:** Enter the repository URL.
  - **Credentials:** Add credentials if required.
  - **Branch Specifier:** Specify the branch (e.g., `*/main`).
7. **Build Triggers:**
- **Build Periodically:** Use CRON syntax to schedule builds.
  - **Poll SCM:** Jenkins will check the repository for changes at specified intervals.
  - **Other Triggers:** Configure other triggers such as GitHub hooks, upstream projects, etc.
8. **Build Environment:**
- Configure the build environment settings like setting environment variables, running scripts before the build, etc.
9. **Build Steps:**
- Click on "Add Build Step" and choose the appropriate build step (e.g., "Execute Shell", "Invoke Ant", "Invoke Gradle script").
  - **Example:** For executing a shell script, add the necessary shell commands.
10. **Post-build Actions:**
- Click on "Add post-build action" and choose the appropriate action (e.g., "Archive the artifacts", "Publish JUnit test result report").
  - Configure the settings for each post-build action.
11. **Save the Configuration:**
- Once you have configured all the necessary settings, click "Save" at the bottom of the page.
12. **Run the Job:**
- On the job's main page, click "Build Now" to run the job immediately.



### 13. Monitor the Job:

- Click on the build number in the "Build History" to view the detailed output and logs of the build process.
- Check the console output for any errors or warnings.

### Practical Example

#### Setting up a Freestyle Project for a Maven Build:

##### 1. Create New Item:

- Name: `MyMavenProject`
- Type: `Freestyle project`

##### 2. Source Code Management:

- **Git:**
  - Repository URL: `https://github.com/example/my-maven-project.git`
  - Branch Specifier: `*/main`

##### 3. Build Triggers:

- **Poll SCM:** `H/15 * * * *` (poll every 15 minutes)

##### 4. Build Steps:

- **Invoke Top-Level Maven Targets:**
  - Goals: `clean install`

##### 5. Post-build Actions:

- **Publish JUnit test result report:**
  - Test report XMLs: `**/target/surefire-reports/*.xml`

##### 6. Save and Build:

- Click "Save".
- Click "Build Now" on the job's main page.

By following these steps, you can set up a Jenkins job to automate your build process for various types of projects. Jenkins provides extensive customization options, allowing you to tailor the job to meet your specific requirements.

## Configuring a Jenkins Job

## 1. Install Jenkins

First, make sure Jenkins is installed. You can download it from the [official Jenkins website](https://jenkins.io/).

## 2. Access the Jenkins Dashboard

Open your web browser and go to <http://your-server-ip:8080> to access the Jenkins dashboard.

## 3. Create a New Job

### 1. Start a New Job:

- Click on "**New Item**" on the left-hand menu.

### 2. Name the Job:

- Enter a name for your job in the "Enter an item name" field.

### 3. Select Job Type:

- Choose the type of job (e.g., Freestyle project, Pipeline).
- Click "**OK**" to proceed.

## 4. Configure the Job

### General Settings

#### 1. Project Description:

- Enter a description for your job.

#### 2. Discard Old Builds:

- Set up a policy to manage the number of builds Jenkins keeps.

### Source Code Management (SCM)

#### 1. Select SCM:

- Choose your source code management system (e.g., Git).

#### 2. Configure Repository:

- Enter the repository URL and provide credentials if needed.

#### 3. Branch Specification:

- Specify the branch to build (e.g., [main](#), [develop](#)).

### Build Triggers

#### 1. Choose Build Triggers:

- **Poll SCM:** Check for changes at intervals.
- **Build Periodically:** Schedule builds.
- **GitHub hook trigger:** Trigger builds on GitHub push.
- **Build after other projects:** Trigger builds based on other jobs.

## Build Environment

### 1. Configure Build Environment:

- **Clean workspace:** Delete workspace before build starts.
- **Manage secrets:** Use secret texts or files.
- **Node selection:** Specify where to run the job.

## Build Steps

### 1. Add Build Steps:

- Click "**Add build step**".
- Choose the build step type (e.g., **Execute shell**, **Invoke Gradle script**).
- Enter the commands or scripts needed for the build.

## Post-build Actions

### 1. Add Post-build Actions:

- Click "**Add post-build action**".
- Common actions include:
  - **Archive artifacts:** Store build artifacts.
  - **Email notification:** Send build status emails.
  - **Publish test results:** Aggregate test results.
  - **Trigger other jobs:** Trigger other jobs based on results.

## 5. Save and Build

### 1. Save the Configuration:

- Click "**Save**".

### 2. Build the Job:

- Click "**Build Now**" to manually trigger a build.
- View progress and logs by clicking on the build number.

## 6. Monitor and Manage Builds

### 1. Check Build History:

- View past builds and their statuses.

### 2. Console Output:

- Click on a build number to see detailed logs.

### 3. Build Artifacts:

- Access any artifacts produced by the build.

## 7. Advanced Configuration (Optional)

### 1. Pipeline as Code:

- For complex workflows, use Jenkins Pipelines. Create a **Jenkinsfile** in your repository.

### 2. Plugins:

- Extend Jenkins functionality with plugins from the Jenkins Plugin Manager.

---

By following these steps, you can set up a Jenkins job to automate builds, integrate with your source code management system, and perform various actions based on build results.

## Introduction to Jenkins Plugins

### What Are Jenkins Plugins?

Jenkins plugins are extensions that add extra functionality to Jenkins. They allow you to customize and extend Jenkins to better fit your specific needs. Plugins can provide integrations with other tools, add new features, and improve existing functionalities.

### Why Use Plugins?

- **Extend Functionality:** Add features not available in the core Jenkins.
- **Integrate Tools:** Seamlessly integrate with other tools and platforms (e.g., GitHub, Docker).
- **Improve Productivity:** Automate more tasks and streamline your workflow.
- **Customize UI:** Tailor the Jenkins interface to better suit your preferences and needs.

### How to Manage Plugins

#### 1. Accessing the Plugin Manager

##### 1. Go to Jenkins Dashboard:

- Open your Jenkins dashboard.

## 2. Navigate to Plugin Manager:

- Click on "**Manage Jenkins**".
- Select "**Manage Plugins**".

## 2. Installing Plugins

### 1. Available Plugins:

- Go to the "**Available**" tab to see a list of plugins that can be installed.

### 2. Search for Plugins:

- Use the search bar to find specific plugins.

### 3. Select Plugins:

- Check the box next to the plugins you want to install.

### 4. Install Plugins:

- Click "**Install without restart**" or "**Download now and install after restart**".

## 3. Updating Plugins

### 1. Go to Updates:

- Navigate to the "**Updates**" tab to see plugins with available updates.

### 2. Select Plugins:

- Check the box next to the plugins you want to update.

### 3. Update Plugins:

- Click "**Download now and install after restart**".

## 4. Managing Installed Plugins

### 1. Installed Plugins:

- Go to the "**Installed**" tab to see all installed plugins.

### 2. Uninstall Plugins:

- Click the "**Uninstall**" button next to the plugin you want to remove.

### 3. Check Plugin Versions:

- View the current version of each installed plugin.

## Popular Jenkins Plugins

### 1. Git Plugin

- **Description:** Integrates Jenkins with Git, allowing Jenkins to pull from and push to Git repositories.
- **Use Case:** Automate builds, deployments, and tests with your Git repositories.

## 2. Pipeline Plugin

- **Description:** Enables Jenkins to define complex build processes using code (Jenkinsfile).
- **Use Case:** Create multi-step and multi-branch pipelines for CI/CD.

## 3. Docker Plugin

- **Description:** Integrates Jenkins with Docker, allowing Jenkins to build and run Docker images.
- **Use Case:** Automate Docker container builds and deployments.

## 4. Email Extension Plugin

- **Description:** Provides advanced email notifications based on build results.
- **Use Case:** Customize email notifications for build successes, failures, and other statuses.

## 5. Blue Ocean Plugin

- **Description:** Offers a modern, user-friendly UI for Jenkins.
- **Use Case:** Improve the user experience with a more intuitive and visually appealing interface.

## Creating Custom Plugins

If the available plugins don't meet your needs, you can create custom plugins. Jenkins provides a comprehensive [Plugin Development Guide](#) to help you get started.

## Conclusion

Jenkins plugins are essential for extending the functionality and improving the efficiency of your Jenkins setup. By managing and utilizing plugins effectively, you can tailor Jenkins to better meet your specific requirements and streamline your CI/CD pipeline.

---

By following this guide, you can understand and manage Jenkins plugins to enhance your Jenkins environment.

## Adding Plugins to Jenkins

Plugins are a crucial part of Jenkins, allowing you to extend its functionality to suit your project's specific needs. Whether you're integrating Jenkins with other tools, customizing



your job configuration, or adding new build steps, plugins provide the flexibility you need. Here's a detailed guide on how to add plugins to Jenkins.

## 1. Why Plugins Are Important

- Extending Jenkins Functionality: Plugins allow Jenkins to integrate with various tools and technologies, such as Git, Maven, Docker, and more.
- Customizing Jobs: With plugins, you can add specific steps, triggers, and post-build actions to your jobs, making Jenkins adaptable to any workflow.
- Automation: Plugins help automate processes like code quality checks, notifications, and deployments.

## 2. Accessing the Plugin Manager

1. Navigate to Jenkins Dashboard: Open your Jenkins dashboard in a web browser (e.g., <http://localhost:8080>).
2. Go to Plugin Manager: From the dashboard, click on 'Manage Jenkins' on the left-hand side menu. On the Manage Jenkins page, click on 'Manage Plugins'. This will take you to the Plugin Manager, where you can view, install, and update plugins.

## 3. Exploring Available Plugins

1. Available Tab: The 'Available' tab lists all plugins that are available for installation. This includes thousands of plugins categorized by functionality (e.g., SCM, build tools, user interface enhancements).
2. Search for Plugins: Use the search box to quickly find the plugin you need. For example, if you're looking to integrate Jenkins with Git, type 'Git' in the search box.
3. Popular Plugins: Jenkins highlights popular plugins at the top of the Available tab. These are commonly used plugins that most users find essential.

## 4. Installing Plugins

1. Selecting Plugins to Install: Check the box next to each plugin you want to install. You can select multiple plugins at once.
2. Install Without Restart: After selecting your plugins, scroll down and click 'Install without restart'. Jenkins will install the plugins immediately, and you can continue using Jenkins during the installation.

3. Install After Restart: If you prefer, you can choose to 'Download now and install after restart'. This will install the plugins after Jenkins is restarted, ensuring that all changes are applied without interference.

4. Installation Progress: The installation process will show a progress bar for each plugin. Once a plugin is installed, it will move to the 'Installed' tab.

## 5. Configuring Installed Plugins

1. Plugin Configuration: After installation, some plugins may require additional configuration. Navigate to 'Manage Jenkins' > 'Configure System' or 'Configure Global Security' to adjust settings for your new plugins.

2. Global Tool Configuration: For build tools like Maven or JDKs installed via plugins, you can configure them under 'Global Tool Configuration'. Here, you define the paths to the tools or specify installation options.

## 6. Updating and Managing Plugins

1. Updating Plugins: Regularly update your plugins to ensure compatibility with the latest version of Jenkins and to benefit from new features or security patches. Go to the 'Updates' tab in the Plugin Manager. Jenkins will show you all plugins with available updates. Click 'Update' to install the latest versions.

2. Uninstalling Plugins: If a plugin is no longer needed, you can uninstall it from the 'Installed' tab. Select the plugin and click 'Uninstall'. Uninstallation typically requires a Jenkins restart to complete.

## 7. Commonly Used Plugins

Here are a few essential plugins that you might consider adding to your Jenkins setup:

1. Git Plugin: Integrates Jenkins with Git, allowing you to pull code from GitHub, Bitbucket, or other Git repositories.

2. Pipeline Plugin: Enables the creation and management of Jenkins pipelines, a powerful way to define complex build, test, and deployment processes.

3. Maven Integration Plugin: Adds support for Maven projects, allowing you to build and manage Maven-based projects directly within Jenkins.

4. Email Extension Plugin: Provides advanced email notification options for build statuses, including customizable email templates.

5. Blue Ocean: An alternative user interface for Jenkins that simplifies pipeline creation and offers a modern, user-friendly design.

## 8. Troubleshooting Plugin Issues

1. **Compatibility Issues:** Sometimes, plugins might conflict with each other or with the Jenkins version. In such cases, refer to the plugin's documentation or Jenkins logs to diagnose the issue.
2. **Rollback:** If a plugin update causes problems, you can rollback to a previous version from the 'Installed' tab by selecting the desired version.
3. **Restart Jenkins:** Some plugin changes require a Jenkins restart. Ensure all jobs are completed before restarting to avoid any interruptions.

## Commonly Used Jenkins Plugins

### 1. Git Plugin

**Overview:** The Git Plugin integrates Jenkins with Git repositories, allowing Jenkins to clone, pull, and manage Git-based source code repositories.

#### Key Features:

- **Source Code Management:** Configures Jenkins to use Git repositories as the source code for jobs.
- **Polling:** Supports polling the Git repository for changes to trigger builds automatically.
- **Branch and Tag Support:** Allows you to specify branches or tags to build from.
- **Credentials:** Manages authentication for private repositories using various methods (e.g., SSH keys, username/password).

#### Configuration:

1. **Install the Plugin:**
  - Go to **Manage Jenkins > Manage Plugins > Available** tab, search for "Git Plugin," and install it.
2. **Configure Job:**
  - In the job configuration, select **Git** under **Source Code Management**.
  - Enter the repository URL and configure credentials if needed.
  - Define the branch to build from (e.g., **main**).

#### Usage:

- Typically used in continuous integration pipelines to fetch code from Git repositories before running build steps.

## 2. Parameter Plugin

**Overview:** The Parameter Plugin allows Jenkins jobs to accept parameters at build time, enabling dynamic and flexible builds.

### Key Features:

- **Parameter Types:** Supports various parameter types such as string, choice, boolean, and more.
- **Default Values:** Provides default values for parameters.
- **Prompt for Parameters:** Users are prompted to enter values for parameters when starting a build.

### Configuration:

1. **Install the Plugin:**
  - Go to **Manage Jenkins > Manage Plugins > Available** tab, search for "Parameterized Builds," and install it.
2. **Configure Job:**
  - In the job configuration, check **This project is parameterized.**
  - Add parameters of different types (e.g., string, choice) and configure their options and default values.

### Usage:

- Useful for creating jobs that require user input or need to be customized for different build scenarios.

---

## 3. HTML Publisher

**Overview:** The HTML Publisher Plugin allows Jenkins to publish HTML reports and artifacts generated during the build process.

### Key Features:

- **Report Publishing:** Publishes HTML reports, dashboards, or other HTML artifacts.
- **Report Directory:** Allows specifying the directory containing HTML reports.
- **Index Page:** Sets a default HTML file to be displayed when accessing the report.

### Configuration:

1. **Install the Plugin:**
  - Go to **Manage Jenkins > Manage Plugins > Available** tab, search for "HTML Publisher Plugin," and install it.
2. **Configure Job:**
  - In the job configuration, go to the **Post-build Actions** section.
  - Select **Publish HTML reports.**

- Specify the directory containing HTML reports and the index page.

**Usage:**

- Ideal for displaying build reports such as test results or code coverage metrics in a user-friendly HTML format.

---

## 4. Copy Artifact

**Overview:** The Copy Artifact Plugin allows Jenkins jobs to copy build artifacts from other jobs, facilitating artifact reuse across different jobs.

**Key Features:**

- **Artifact Copying:** Copies files from one build to another, which can be from a specific build or the latest successful build.
- **Triggering Builds:** Can be used in downstream jobs to fetch artifacts from upstream jobs.

**Configuration:**

1. **Install the Plugin:**
  - Go to **Manage Jenkins > Manage Plugins > Available** tab, search for "Copy Artifact Plugin," and install it.
2. **Configure Job:**
  - In the job configuration, go to the **Build** section.
  - Add a **Copy artifacts from another project** build step.
  - Specify the project name, build number or criteria, and the target directory for copied artifacts.

**Usage:**

- Commonly used in multi-job pipelines where artifacts need to be shared between different jobs or stages.

---

## 5. Extended Choice Parameter

**Overview:** The Extended Choice Parameter Plugin provides advanced parameter types for Jenkins jobs, such as multi-select lists, checkboxes, and more complex formats.

**Key Features:**

- **Parameter Types:** Includes multi-select, checkboxes, and other advanced input options.

- **Dynamic Choices:** Allows dynamic generation of choices from scripts or external sources.

#### Configuration:

1. **Install the Plugin:**
  - Go to **Manage Jenkins > Manage Plugins > Available** tab, search for "Extended Choice Parameter Plugin," and install it.
2. **Configure Job:**
  - In the job configuration, check **This project is parameterized**.
  - Add an **Extended Choice Parameter**.
  - Configure the parameter type, choices, and other settings.

#### Usage:

- Useful for scenarios where complex user input is needed or where multiple selection options are required.

## Configuring Jenkins to Work with Java, Git, and Maven

### 1. Configuring Jenkins to Work with Java

**Overview:** Jenkins requires Java to run. Configuring Java in Jenkins involves specifying the Java Development Kit (JDK) installations Jenkins should use.

#### Steps:

1. **Install Java Development Kit (JDK):**
  - Ensure that JDK is installed on your system. You can download it from the [Oracle website](#) or use OpenJDK.
2. **Configure JDK in Jenkins:**
  - Open Jenkins and go to **Manage Jenkins > Global Tool Configuration**.
  - Scroll down to **JDK** section and click **Add JDK**.
  - Enter a name for the JDK installation (e.g., **JDK 11**).
  - Check **Install automatically** to let Jenkins download and install the JDK, or specify the path to an existing JDK installation.
  - If specifying the path manually, provide the **JAVA\_HOME** directory.
3. **Example Configuration:**
  - **Name:** JDK 11
  - **JAVA\_HOME:** `/usr/lib/jvm/java-11-openjdk`
4. **Verify JDK Configuration:**

You can verify the JDK configuration by creating a simple Jenkins job and adding a build step that prints the Java version using:



```
bash
Copy code
java -version
```

---

## 2. Configuring Jenkins to Work with Git

**Overview:** The Git Plugin integrates Jenkins with Git repositories, enabling Jenkins to clone, pull, and manage Git-based source code repositories.

### Steps:

- 1. Install the Git Plugin:**
    - Go to **Manage Jenkins > Manage Plugins**.
    - Under the **Available** tab, search for **Git Plugin** and install it.
  - 2. Configure Git in Jenkins:**
    - Navigate to **Manage Jenkins > Global Tool Configuration**.
    - Scroll down to the **Git** section and click **Add Git**.
    - Specify the path to the Git executable or use the default path.
    - Optionally, configure additional settings such as Git installation locations.
  - 3. Example Configuration:**
    - Name:** Git
    - Path to Git executable:** `/usr/bin/git` (or use the default if Git is installed in a standard location).
  - 4. Configure Job to Use Git:**
    - Create or edit a Jenkins job and go to **Source Code Management**.
    - Select **Git**.
    - Enter the repository URL (e.g., `https://github.com/user/repository.git`).
    - Configure credentials if the repository is private.
    - Specify the branch to build (e.g., `main`).
  - 5. Example Repository URL:**
    - Repository URL:** `https://github.com/example/repo.git`
    - Branch Specifier:** `main`
- 

## 3. Configuring Jenkins to Work with Maven

**Overview:** The Maven Plugin integrates Jenkins with Apache Maven, allowing Jenkins to use Maven to build projects.

### Steps:

- 1. Install Maven:**

- Ensure that Apache Maven is installed on your system. You can download it from the [Apache Maven website](#).
- 2. **Configure Maven in Jenkins:**
  - Go to **Manage Jenkins > Global Tool Configuration**.
  - Scroll down to the **Maven** section and click **Add Maven**.
  - Enter a name for the Maven installation (e.g., **Maven 3.8.6**).
  - Check **Install automatically** to let Jenkins download Maven, or specify the path to an existing Maven installation.
- 3. **Example Configuration:**
  - **Name:** Maven 3.8.6
  - **MAVEN\_HOME:** /usr/share/maven
- 4. **Configure Maven in a Jenkins Job:**
  - Create or edit a Jenkins job and go to **Build** section.
  - Add a build step and select **Invoke top-level Maven targets**.
  - Choose the Maven version configured earlier.
  - Specify the goals to run (e.g., **clean install**).
- 5. **Example Maven Goals:**
  - **Goals:** **clean install**
- 6. **Configure Build Environment:**
  - Ensure that the job's build environment is set up to use Maven. This might involve setting environment variables or configuring build scripts.
- 7. **Example Environment Variables:**
  - **MAVEN\_OPTS:** **-Xms512m -Xmx2048m** (if you need to set specific JVM options for Maven).

---

By following these steps, Jenkins will be properly configured to work with Java, Git, and Maven, allowing you to set up and manage continuous integration and delivery pipelines effectively. If you need further customization or run into issues, consulting the documentation for each tool or plugin can provide additional guidance.

## Creating a Jenkins Build and Understanding Jenkins Workspace

### 1. Creating a Jenkins Build

**Overview:** A Jenkins build is a process where Jenkins executes a series of steps defined in a job configuration. This typically involves compiling code, running tests, and generating artifacts.

#### Steps to Create a Jenkins Build:

1. **Access Jenkins Dashboard:**

- Open your Jenkins dashboard, usually found at <http://localhost:8080>.
- 2. Create a New Job:**
  - Click on **New Item** on the left sidebar.
  - Enter a name for your job.
  - Choose a job type. Common types include:
    - **Freestyle project**: For simple build jobs with basic configuration.
    - **Pipeline**: For more complex build processes using a Jenkinsfile.
  - Click **OK** to proceed.
- 3. Configure Job Details:**
  - **General:**
    - Enter a description for your job.
    - Configure options such as discarding old builds if needed.
  - **Source Code Management:**
    - Choose **Git**, **Subversion**, or another source control system.
    - Enter the repository URL and credentials if the repository is private.
    - Specify the branch or tag to build from.
  - **Build Triggers:**
    - Set up triggers to start the build. Common options include:
      - **Poll SCM**: Jenkins will periodically check for changes in the source code repository.
      - **Build periodically**: Schedule builds at specific intervals.
      - **GitHub hook trigger for GITScm polling**: Trigger builds based on GitHub webhooks.
  - **Build Environment:**
    - Configure the build environment, such as setting up environment variables or cleaning up before the build starts.
  - **Build Steps:**
    - Add build steps by clicking **Add build step**.
    - Common build steps include:
      - **Execute shell**: Run shell commands or scripts.
      - **Invoke Gradle script**: Use Gradle to build the project.
      - **Invoke Ant**: Use Apache Ant for building.
  - **Post-build Actions:**
    - Configure actions to perform after the build completes, such as:
      - **Archive the artifacts**: Save build artifacts for later use.
      - **Publish JUnit test result report**: Display test results in Jenkins.
      - **Send build notifications**: Notify stakeholders of build status.
- 4. Save and Build:**
  - Click **Save** to store the job configuration.
  - To start a build, click **Build Now** in the job dashboard.

**Example Build Configuration:**

- **Job Name:** MyApp-Build
  - **Source Code Management:** Git
    - **Repository URL:** <https://github.com/user/myapp.git>
    - **Branch:** `main`
  - **Build Step:** Execute shell
    - **Command:** `mvn clean install`
  - **Post-build Action:** Archive artifacts
    - **Files to archive:** `target/*.jar`
- 

## 2. Understanding Jenkins Workspace

**Overview:** The Jenkins workspace is a directory where Jenkins stores files and artifacts related to a particular build. Each job has its own workspace, which is used to perform build operations.

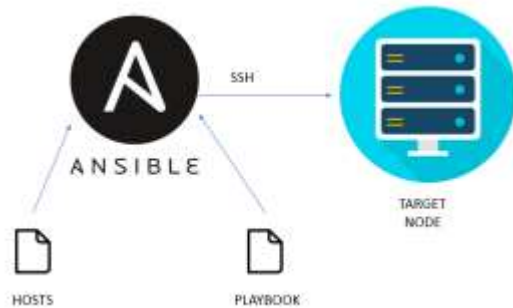
### Key Aspects of Jenkins Workspace:

1. **Workspace Location:**
  - By default, the workspace is located in the Jenkins home directory, typically at `/var/lib/jenkins/workspace/` on Linux systems or `C:\Program Files (x86)\Jenkins\workspace\` on Windows.
2. **Workspace Structure:**
  - Each job gets its own subdirectory within the workspace. For example, a job named `MyApp-Build` will have its workspace at `/var/lib/jenkins/workspace/MyApp-Build/`.
  - The workspace contains:
    - **Source Code:** The code pulled from the repository.
    - **Build Artifacts:** Files generated during the build process.
    - **Logs:** Logs related to the build process.
3. **Workspace Usage:**
  - **Building:** During the build process, Jenkins checks out the code into the workspace, executes build steps, and generates artifacts.
  - **Archiving Artifacts:** After the build, files specified in the post-build actions are archived from the workspace.
  - **Cleaning Up:** Jenkins may clean up workspaces based on job configurations or policies to save disk space.
4. **Customizing Workspace:**
  - **Configure Custom Workspace Location:**
    - In the job configuration, under the **Advanced Project Options**, you can specify a custom workspace directory.
  - **Workspace Cleanup:**
    - Use plugins like the **Workspace Cleanup Plugin** to manage and clean up workspaces automatically.

**Example of Workspace Structure:**

- **Workspace Directory:** `/var/lib/jenkins/workspace/MyApp-Build/`
    - **Source Code:** `/var/lib/jenkins/workspace/MyApp-Build/src/`
    - **Build Artifacts:** `/var/lib/jenkins/workspace/MyApp-Build/target/`
    - **Logs:** `/var/lib/jenkins/workspace/MyApp-Build/logs/`
- 

This guide should provide a comprehensive understanding of creating a Jenkins build and managing Jenkins workspaces.

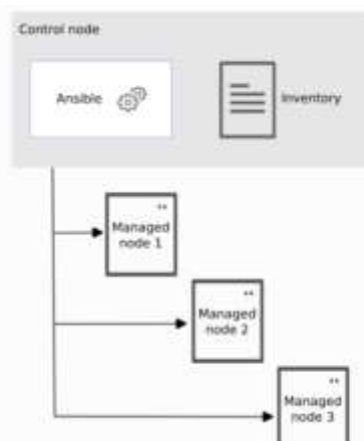


Ansible is a powerful DevOps tool designed for automating tasks on remote servers or nodes. In simple terms, it allows you to automate commands and functions on multiple remote machines from a central 'master' node. To illustrate its usefulness, consider a scenario where you need to reboot dozens or even hundreds of remote hosts. You could manually SSH into each one and initiate the reboot, or you can use Ansible to streamline the process, making it efficient and offering a wide range of additional functionalities.

In essence, Ansible operates much like the second method mentioned, using SSH (Secure Shell), a secure communication protocol, to control remote nodes in a secure and optimized manner.

## Ansible Components

### Ansible Core Components



#### 1. Control Node –

- The central or main node where Ansible is installed.
- Used to trigger commands like ansible and ansible-inventory on other nodes.
- Acts as the orchestrator for Ansible operations.

#### 2. Manage Node–

- A remote or slave node where tasks are executed or controlled by Ansible.
- These are the servers or devices you want to manage or automate.

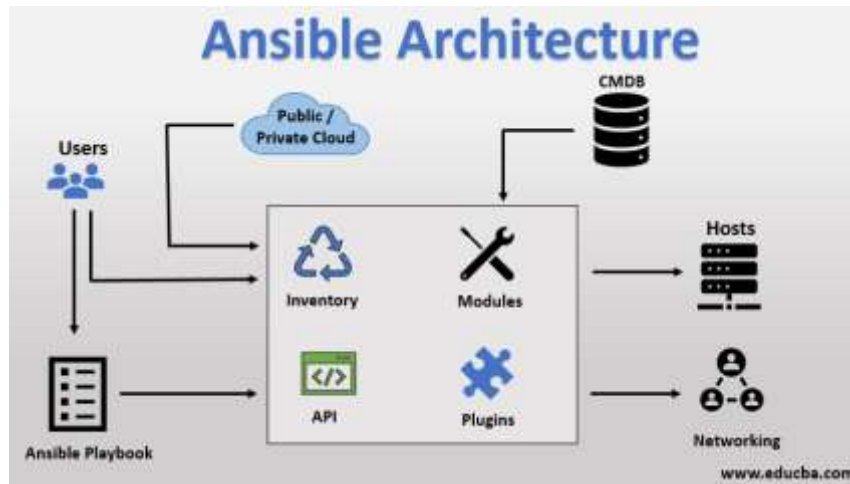
#### 3. Inventory –

- A list of managed node IPs and configurations.



- Logically organized, typically using file formats like YAML or INI.
- Created on the control node to describe the deployment of hosts to Ansible.

### Ansible Additional Components



#### 1. Ad-Hoc Commands –

- These are one-off commands that you can execute using the ansible command.
- Useful for quick tasks or tests on remote nodes.

#### 2. Plugins –

- Plugins are pieces of code that extend Ansible's core functionality.
- Ansible uses a plugin architecture for flexibility and expandability.
- Examples include connectivity plugins for establishing connections and cache plugins.
- [Learn more about Ansible plugins.](#)

#### 3. Module–

- Modules are built-in functions that can be used to perform various tasks.
- They eliminate the need to write custom code for common operations.
- Examples include modules for package management (apt, yum), service management, and more.
- [Explore Ansible modules.](#)

#### 4. Playbook –

- Playbooks are a sequence of plays that define the order of tasks executed by Ansible.
- A play consists of a list of tasks that target managed nodes in an inventory.
- Tasks, in turn, are composed of one or more modules that specify operations.
- Playbooks provide a structured way to define and automate complex workflows.

**5. Roles –**

- Roles provide an organized environment for managing complex tasks.
- They include templates, playbooks, inventories, error handlers, vars, and meta information.
- Ideal for handling larger, multi-step automation processes.

**6. Collections –**

- Collections are distribution formats for Ansible content.
- They encompass playbooks, roles, modules, and plugins.
- [Find Ansible collections.](#)

**7. Galaxy –**

- Ansible Galaxy is a platform for sharing and downloading collections.
- It allows the Ansible community to collaborate and exchange automation content.

**Setting Up the Ansible Control Node****Prerequisite:**

Before you begin setting up Ansible, it's essential to ensure you meet the following prerequisites:

- **SSH:** Make sure SSH is installed on your Linux system. SSH is crucial for secure communication between the control node and managed nodes.

**Installation Steps:**

1. Open your terminal.
2. Run the following command to install Ansible:

```
1python3 -m pip install --user ansible
```

3. SSH Key:
  - For secure and direct connections to managed nodes, you'll need to provide a public SSH key. You can generate one using the ssh-keygen command.
  - Follow the on-screen instructions to create your SSH key pair. This key pair is a crucial element of Ansible's secure communications.
4. For specific installation instructions tailored to your Linux distribution, [click here](#).
5. Try to connect ssh using once your manage node is set up.

```
1ssh <user>@<ip>
```

**Note:** If you're using a different operating system, find the appropriate installation instructions [here](#).

**Setting Up the Ansible Manage Node**

**Steps:****1. Install OpenSSH-Server:**

- To enable remote management of the managed node, you'll need to install the OpenSSH server. Use the following command (assuming you're using a Debian-based Linux distribution like Ubuntu):

```
1sudo apt install openssh-server
```

- This command installs the OpenSSH server, allowing secure remote access.

**2. Create a User:**

- For convenience and consistency, it's helpful to create a user on the managed node with the same name as the user on the control node. This makes it easier to manage SSH keys and ensures a smoother experience.

**3. Configure SSH Key:**

- To establish secure and direct connections to the managed node, you need to paste the public SSH key from the control node into the `authorized_keys` file located in the `.ssh` directory of the user's home folder on the managed node.
- The path is typically `/home/<user>/.ssh/authorized_keys`, where `<user>` is the username you created or are using on the managed node. You can use the `ssh-copy-id` command to automate the process of copying your public key to the managed node. For example:

```
1ssh-copy-id <user>@<managed_node_ip>
```

- This command securely copies your public key to the `authorized_keys` file, allowing passwordless SSH authentication.

**Managing Ansible Inventories**

In the world of Ansible, inventories are like the backbone of your automation infrastructure. They serve as a vital component that lists and organizes the managed nodes, making automation tasks seamless and organized. Inventories use a parent-child concept that allows you to create groupings, which come in handy during specific tasks or scenarios.

**Why Are Inventories Important?**

- Inventories provide a comprehensive list of managed nodes, making it easy to interact with and manage them.
- Groupings in inventories enable you to organize your infrastructure, allowing you to target specific subsets of nodes based on your needs. For instance, when you need to perform tasks in a particular location, groupings become essential.

**Creating Inventories**

There are two primary methods to create an inventory on the control node: using a YAML file or an INI file.

**1. INI Inventory Example (inventory.ini):**

```

1[virtualmachines]
2host1 ansible_host:192.168.0.1
3192.168.0.2

```

2. In this INI file, it captures the IP addresses of managed nodes. The default username used for SSH connections is typically the working user of your control node.
3. **ansible\_host**: This parameter is used to specify the IP address, although you can write the IP directly.

#### 4. Inventory.yaml

```

1---
2virtualmachines: # Define a class name to represent a particular group of devices.
3  hosts: # Define the hosts.
4      vm01: # Define a parent name to indicate a class of parent type.
5          ansible_host: 192.168.0.1 # Specify the IP address.
6          http_port: 80 # (optional) Define the HTTP port.
7          ansible_user: auriga # (optional) Provide the username if it's different from the
            control node's user.

```

- **ansible\_user**: Use this to specify a different user if the managed node's user is not the same as the control node's user.
  - **http\_port**: Specify a port if necessary.
- In real-world scenarios, inventory files can be more complex, reflecting the diverse and extensive infrastructure of organizations. You can explore more configurations here.

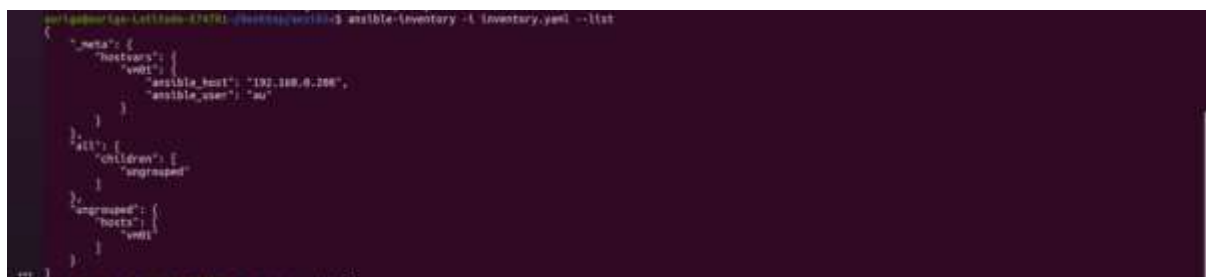
### Verifying Your Inventory

After creating your inventory, it's a good practice to verify its correctness. This step ensures that the inventory is correctly structured and accessible to Ansible. To do this, use the following command:

```
1ansible-inventory -i inventory.yaml --list
```

- **-i**: Indicates the inventory file you want to use.
- **--list**: Requests the listing of the inventory content.

**output:**



```

{
  "meta": {
    "hostvars": {
      "vm01": {
        "ansible_host": "192.168.0.1",
        "ansible_user": "auriga"
      }
    }
  },
  "all": {
    "children": [
      "ungrouped"
    ]
  },
  "ungrouped": {
    "hosts": [
      "vm01"
    ]
  }
}

```

By following these steps, you have effectively set up and verified your inventory, ensuring a solid foundation for your Ansible automation.

### Using Ansible Ad-Hoc Commands

#### Introduction:

In Ansible, ad-hoc commands are your go-to solution for executing quick, one-off tasks on remote nodes. They provide a straightforward and efficient way to interact with managed nodes without the need for creating full-fledged playbooks. Ad-hoc commands are particularly useful when you need immediate results without the overhead of playbook development.

#### When to Use Ad-Hoc Commands:

Ad-hoc commands are best suited for scenarios where the task at hand is simple and doesn't require the complexity of a playbook. They are perfect for tasks like system health checks, package installation, service management, or any other single-operation job.

#### Ad-Hoc Command Syntax:

Ad-hoc commands follow a specific syntax that comprises various components, each serving a unique role:

- **Target Group:** This is the group of hosts you intend to target with the ad-hoc command.
- **Module (-m):** Specifies the module to execute. Modules are Ansible's building blocks for performing tasks, and they can range from basic operations like "ping" to more advanced tasks such as package management.
- **Inventory File (-i):** Indicates the location of your inventory file, which defines the list of target devices.

#### Example: Ping All Inventory Devices

To illustrate the use of ad-hoc commands, let's consider a simple task: pinging all devices listed in your inventory. Here's the command:

```
1 ansible virtualmachines -m ping -i inventory.yaml
```

virtualmachines is the target group. In this case, it could be any group, or you can use all to target all devices in your inventory.

- **-m ping** specifies the "ping" module, a basic module that checks the reachability of the managed nodes.
- **-i inventory.yaml** points to the inventory file containing the list of devices.
- The "ping" module sends a test command to the target devices and reports their status, confirming whether they are responsive.

#### Output and Result:



```

verig@verig:~$ ansible all -i inventory.yaml -m ping
ansible: [WARNING]: Using the password 'verig' to authenticate to the host 192.168.1.101.
ansible: [WARNING]: Using the password 'verig' to authenticate to the host 192.168.1.102.
192.168.1.101: UNREACHABLE!
192.168.1.102: pong
  
```

In the above example the ssh host is not reachable.



In the above output image it is working fine

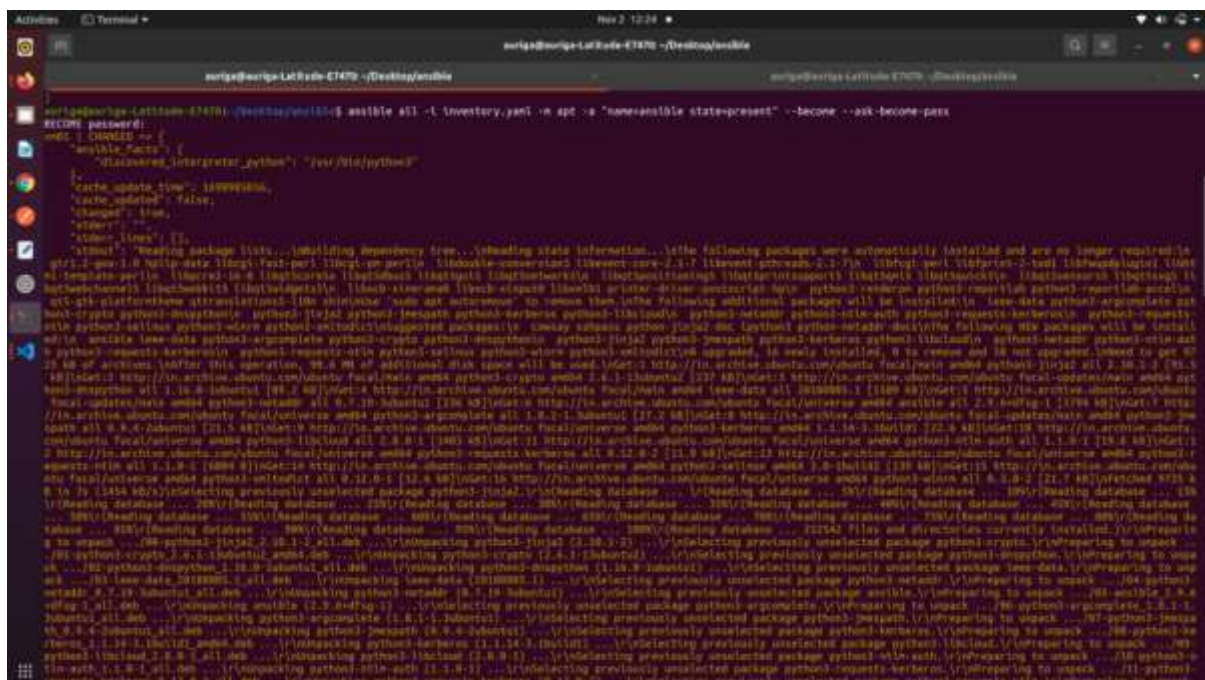
### Example: Installing Nginx Using Apt Module

For a more practical example, let's use the "apt" module to install Nginx on your managed nodes. Here's the command:

1 ansible all -i inventory.yml -m apt -a "name=nginx state=present" --become --ask-become-pass

- -m apt specifies the "apt" module, which is responsible for package management.
- -a "name=nginx state=present" includes variables for the module to act upon, such as specifying that Nginx should be installed (state=present).
- --become signifies that the command should run with elevated privileges (sudo).
- --ask-become-pass prompts for the sudo password of the managed node to ensure the installation proceeds smoothly.

### Output and Result:



By using these ad-hoc commands, you can perform quick, task-specific operations on your managed nodes, saving time and effort in your automation tasks.

### Common

Ansible offers an extensive library of modules for various tasks, from system administration to application deployment. You can explore the full list of Ansible modules in the official documentation to find the most suitable [module](#) for your specific requirements.

### Modules:



**Best****Practices:**

When working with ad-hoc commands, consider using SSH key-based authentication for secure, passwordless access. It streamlines the authentication process and enhances the security of your automation tasks.

**Using Ansible Playbooks****Introduction:**

In the world of Ansible, playbooks are the cornerstone of structured and complex automation. While ad-hoc commands are perfect for quick and isolated tasks, playbooks step in when you need a well-orchestrated sequence of operations. They allow you to tackle multi-step automation scenarios where tasks depend on each other, and conditional actions are required.

**Let's delve deeper into the importance of playbooks:****The Role of Playbooks:**

Playbooks serve as the framework for orchestrating automation tasks that involve a sequence of operations. Whether it's configuring servers, deploying applications, or managing infrastructure, playbooks provide a structured way to define how these tasks are executed.

**Ad-Hoc Limitations:**

While ad-hoc commands are ideal for single, immediate tasks, they are not designed for complex, multi-step automation. Playbooks come to the rescue when tasks have interdependencies and need to be executed in a coordinated manner.

**Readability and Reusability:**

Playbooks are authored in YAML, a human-readable and straightforward format. This not only makes them easy to write but also facilitates sharing and collaboration. You can reuse playbooks across various scenarios, saving time and effort.

**Conditional and Looping Logic:**

Playbooks offer advanced features, including conditional statements and looping, that allow you to adapt automation to different situations. This flexibility makes playbooks versatile and capable of handling a wide range of automation needs.

In essence, playbooks are your tool of choice when automation tasks become multi-faceted and require a structured and logical approach. They provide the power to streamline and automate complex workflows with precision and efficiency.

**Example: Ping All Inventory Devices**

To illustrate the use of a playbook, let's consider a simple task: pinging all devices listed in your inventory. Here's the command:

**playbook.yaml**

```
1---
2 -name : My First Play #Name of play
3 hosts: virtualmachines #defining host can be all
```

4 tasks:

5 -name: Ping My Hosts #Name of task

6 ansible.builtin.ping: # can also write ping

To run a playbook, use the following command:

1 ansible-playbook -i inventory.yaml playbook.yaml

### Output and Result:

```

ansible@swrtpa-Latitude-E7470: ~/Desktop/ansible
ansible-playbook -i inventory.yaml playbook-eg1.yaml

PLAY [My First Play] *********************************************************************
TASK [Gathering Facts] *********************************************************************
ok: [swrtpa-Latitude-E7470]

TASK [Ping Devices] *********************************************************************
ok: [swrtpa-Latitude-E7470]

PLAY RECAP *********************************************************************
swrtpa-Latitude-E7470: 1 ok, 1 changed=0, unreachable=0, failed=0, skipped=0, rescued=0, ignored=0
  
```

As you can see there is gathering facts task which we didn't create is running it is the default task which ping the connection.

### Example: Installing Nginx Using Apt Module

For a more practical example, let's use the "apt" module to install Nginx on your managed nodes. Here's the command:

**playbook.yaml**

1---

2- name: Install Nginx

3 hosts: all

4 become: yes

5 tasks:

6 - name: Install Nginx using apt

7 apt:

8 name: nginx

9 state: present

To run a playbook, use the following command:

```
1 ansible-playbook -i inventory.yaml playbook.yaml --ask-become-pass
```

This command executes the specified playbook while prompting for the necessary privilege escalation password.

### Output and Result:

```

suriqa@suriqa-Latitude-E7470 ~/Desktop/ansible
suriqa@suriqa-Latitude-E7470 ~/Desktop/ansible$ ansible-playbook -i inventory.yaml playbook-eg2.yaml
PLAY [Install Nginx] *****
TASK [Gathering Facts] *****
suriqa@suriqa-Latitude-E7470 ~/Desktop/ansible$ ansible-playbook -i inventory.yaml playbook-eg2.yaml --ask-become-pass
BECOME password:
PLAY [Install Nginx] *****
TASK [Gathering Facts] *****
TASK [Install Nginx using apt] *****
PLAY RECAP *****
  ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
suriqa@suriqa-Latitude-E7470 ~/Desktop/ansible$

```

As shown in the example image above, the first command encountered an error due to the absence of the `--ask-become-pass` flag. This flag is essential when your task requires elevated privileges, as it prompts Ansible to request the sudo password for authentication.

The corrected command includes the `--ask-become-pass` flag, ensuring that the necessary privileges are obtained before executing the task. This is particularly important when working with tasks that require administrative access, such as package installations or system configurations.

### Note:

In this blog, we primarily focus on the core and fundamental components of Ansible, providing an introductory overview and understanding of its key concepts. While Ansible offers advanced features such as “Collections,” “Ansible Galaxy,” “Roles,” “Dynamic Inventory,” and “Custom Modules and Plugins,” we recognize that these topics are extensive and may require separate dedicated discussions. For the purpose of this blog, we aim to establish a strong foundation of Ansible’s core components. If you’re interested in diving deeper into these advanced topics, we recommend exploring Ansible’s official documentation and additional resources dedicated to each subject.

VIRTUAL MACHINE: <https://www.youtube.com/watch?v=7DLfOGt8YvA&list=PLhW3qG5bs-L9S272lwi9encQOL9nMOnRa&index=2>

VAGRANT: <https://www.youtube.com/watch?v=7DLfOGt8YvA&list=PLhW3qG5bs-L9S272lwi9encQOL9nMOnRa&index=2>

How to install and setup Vagrant (windows and mac os)

How to install and VirtualBox

How to create and run virtual machines using Vagrant

Go inside the virtual machine and run commands

Vagrant commands to manage virtual machines

Install VM

Start VM

Connect to VM

Stop VM

Destroy VM

Check status

Vagrant - Getting Started | Install > Setup > Use

Step 1 - Install Vagrant <https://www.vagrantup.com/downloads> Check vagrant is installed vagrant --version

Step 2 - Select a VM Provider. Vagrant has direct support for VirtualBox, Hyper-V, Docker

Install VirtualBox <https://www.virtualbox.org/wiki/Downl...>

Step 3 - Create a new folder for the Vagrant project

Step 4 - On the terminal or command line navigate to the folder and initiate the vagrant project  
vagrant init

This will create a new Vagrantfile in the folder

Vagrantfile is a configuration file that defines the settings for your virtual machine

Step 5 - Choose a box to use <https://app.vagrantup.com/boxes/search>

A box is a pre-configured virtual machine image that you can use as a starting point for your virtual machine

Step 6 - Add configuration of the box in vagrantfile

For example, you could use the "ubuntu/bionic64" box by adding the following line to your Vagrantfile:

```
config.vm.box = "ubuntu/bionic64"
```

We can also directly add configuration for the virtual machine using the following commands

`vagrant init centos/7` (if vagrantfile does not already exists)

`vagrant box add centos/7` (will add box to vagrant, but will not create Vagrant file)

Step 7 - Start virtual machine using command `vagrant up`

This will create a new virtual machine using the box you selected and start it. The first time, Vagrant will download the box from the internet

Step 8 - SSH into the virtual machine `vagrant ssh`

Vagrant Box - 7 Commands

`vagrant box add`

Adds a box to your local box repository

`vagrant box add ubuntu/focal64`

**`vagrant box list`**

Lists all boxes in your local box repository

`vagrant box list`

`vagrant box outdated`

Checks if any boxes in your local box repository are outdated

`vagrant box outdated`

`vagrant box update`

Updates a box to a new version

`vagrant box update ubuntu/focal64`

`vagrant box repackage`

Repackages a box with a new name and metadata

`vagrant box repackage ubuntu/focal64 --name my-new-box`

`vagrant box prune`

Removes outdated boxes from your local box repository

`vagrant box prune`

vagrant box remove

Removes a box from your local box repository

vagrant box remove ubuntu/focal64

Location of VM boxes

Mac OS X and Linux: ~/.vagrant.d/boxes

Windows: C:/Users/USERNAME/.vagrant.d/boxes

Vagrant Commands

vagrant init

Initializes a new Vagrant environment by creating a Vagrantfile

vagrant init centos/7

vagrant up

Creates and configures the guest machine

vagrant ssh

Logs in to the guest machine via SSH

vagrant ssh-config

Outputs OpenSSH valid configuration to connect to the VMs via SSH

vagrant halt

Stops the guest machine

vagrant suspend

Suspends the guest machine

vagrant resume

Resumes a suspended guest machine

**vagrant reload**

Reloads the guest machine by restarting it

**vagrant destroy**

Stops and deletes all traces of the guest machine

**vagrant status**

Shows the status of the current Vagrant environment

**vagrant package**

Packages a running virtual environment into a reusable box

`vagrant package --output mybox.box`

**vagrant provision**

Runs any configured provisioners against the running VM.

**vagrant plugin install**

Installs a Vagrant plugin

`vagrant plugin install myplugin`

**vagrant plugin list**

Lists all installed Vagrant plugins

**vagrant plugin uninstall**

Uninstalls a Vagrant plugin

`vagrant plugin uninstall myplugin`

**Useful TIPS**

`--help` To get help for any Vagrant command e.g. `vagrant --help` or `vagrant init --help`

`vboxmanage list vms` If using Virtualbox

`vboxmanage list runningvms` If using Virtualbox



**References:**

Vagrant - <https://developer.hashicorp.com/vagra...>

VirtualBox - <https://www.virtualbox.org/wiki/Downl...>

Vagrant Boxes Search - <https://app.vagrantup.com/boxes/search>

---

ANSIBLE INSTALLATION: [https://www.youtube.com/watch?v=Ops94dNqZ-0&list=PLhW3qG5bs-L\\_Mjj22rz9e44LI-CUnN2vu&index=3](https://www.youtube.com/watch?v=Ops94dNqZ-0&list=PLhW3qG5bs-L_Mjj22rz9e44LI-CUnN2vu&index=3)

**Part A****Ansible Controller Machine Setup**

Step 1 - Install VirtualBox and Vagrant on your local machine.

Step 2 - Open a terminal and navigate to the directory where you want to set up your Ansible project.

Step 3 - Create a new directory for your Ansible controller VM by running the command `mkdir ansible-controller`

Step 4 - Navigate to the directory and create a new file called Vagrantfile by running the command `vagrant init centos/7`

Step 5 - Edit the Vagrantfile and add the lines to the end of the file to provision Ansible on the VM

Vagrantfile for creating VM for Ansible Controller

```
Vagrant.configure("2") do |config|
  config.vm.define "ansible-controller" do |controller|
    controller.vm.hostname = "controller"
  end
  config.vm.box = "centos/7"
  config.vm.provision "shell", inline: <<-SHELL
    sudo yum install epel-release -y
    sudo yum install ansible -y
  SHELL
end
```

Step 6 - Save & check its a valid vagrantfile `vagrant validate` Then run command `vagrant up` to start the VM

Step 7 - Once the VM is up and running, connect to it using SSH by running the command `vagrant ssh`

Check ansible is installed - `ansible --version`

Step 8 - Create a new directory for your Ansible project on the controller VM by running the command `mkdir ansible-project`

Step 9 - Navigate to the ansible-project directory and create a new file called hosts by running the command `touch hosts`

Step 10 - Create a new file called `playbook.yml`. This file will contain the tasks you want to perform on your managed hosts

As of now the hosts and the playbook file are empty

We will now create some host machines that will be controlled by the Ansible controller

## Part B

### Ansible Host Machines Setup

Step 1 - On terminal navigate to your Ansible Project folder

Step 2 - Create a new directory for your host machines by running the command `mkdir host-machines`

Step 3 - Navigate to host-machines directory and create a new Vagrantfile by running the command `vagrant init centos/7`

Step 4 - Edit the Vagrantfile and modify the following lines to set up two Vagrant machines:

Vagrantfile for creating VMs for Ansible Host

```
Vagrant.configure("2") do |config|
```

```
  config.vm.box = "centos/7"
```

```
  config.vm.define "web" do |web|
```

```
    web.vm.hostname = "web"
```

```
    web.vm.network "private_network", ip: "192.168.33.10"
```

```
  end
```

```
  config.vm.define "db" do |db|
```

```
db.vm.hostname = "db"

db.vm.network "private_network", ip: "192.168.33.11"

end

config.vm.network "forwarded_port", guest: 80, host: 8080, auto_correct: true

config.vm.usable_port_range = (8000..9000)

end
```

Step 6 - Save & check its a valid vagrantfile `vagrant validate` Then run command `vagrant up` to start the VM

Step 7 - Check the status of machines `vagrant status`

Once the VMs are up, connect to them using SSH `vagrant ssh <machine-name>` e.g  
`vagrant ssh web`

This completes the process of setting up host machines

### Part C

Making connection between controller and host machines

Step 1 - Make sure all machines are up and running

Step 2 - Run command `ip addr` on each machine and check they have IP addresses in the same range (e.g. 192.168.33.x).

Step 3 - On Controller machine run the command `ssh-keygen` to generate an SSH key pair

Step 4 - Goto `~/.ssh` folder and check the public and private keys generated

Step 5 - Copy the public key to the host machines by running the command `ssh-copy-id <user>@<host>`

For example, to copy the public key to the web machine, run the command `ssh-copy-id vagrant@192.168.33.10`

Can do this manually by copying the contents of the `.pub` file generated by `ssh-keygen` and pasting it into the `~/.ssh/authorized_keys` file on the host machines

Step 6 - Test the SSH connection by running the `ssh` command with the IP address of the host machines-

For example: `ssh vagrant@192.168.33.10`

`ssh vagrant@192.168.33.11`

## Ansible Hands-On | Step by Step for Beginners

Based on this video - <https://youtu.be/Ops94dNqZ-0>

- ✓ How to setup Ansible Controller machine
- ✓ How to setup Ansible Host machines
- ✓ Making connection between Controller and Hosts
- ✓ Adding host and playbook file on Controller
- ✓ Run Playbook to configure Host Machines

We will need Linux machines for this DEMO

You can use any Linux machines or setup using any cloud platforms like AWS

In this Demo I am going to use Vagrant to create Linux Virtual Machines

In any case the process and steps will remain same

### Reference:

Create Free Linux on AWS and connect from Windows and Mac OS - <https://youtu.be/6bb-Oqbl8-E>

Vagrant Beginner Playlist - <https://www.youtube.com/playlist?list=PLhW3qG5bs-L9S272lwi9encQOL9nMOnRa>

Vagrant VM Boxes - <https://app.vagrantup.com/centos/boxes/7>

---

## Part A - Ansible Controller Machine Setup

Step 1 - Install VirtualBox and Vagrant on your local machine.

Step 2 - Open a terminal and navigate to the directory where you want to set up your Ansible project.

Step 3 - Create a new directory for your Ansible controller VM by running the command **mkdir ansible-controller**

Step 4 - Navigate to the directory and create a new file called Vagrantfile by running the command **vagrant init centos/7**

Step 5 - Edit the Vagrantfile and add the following lines to the end of the file to provision Ansible on the VM:

```
config.vm.provision "shell", inline: <<-SHELL
  sudo yum install epel-release -y
  sudo yum install ansible -y
SHELL
```

## Vagrantfile for creating VM for Ansible Controller

```
Vagrant.configure("2") do |config|
  config.vm.define "ansible-controller" do |controller|
    controller.vm.hostname = "controller"
  end
  config.vm.box = "centos/7"
  config.vm.provision "shell", inline: <<-SHELL
    sudo yum install epel-release -y
    sudo yum install ansible -y
  SHELL
end
```

Step 6 - Save & check its a valid vagrantfile **vagrant validate** Then run command **vagrant up** to start the VM

Step 7 - Once the VM is up and running, connect to it using SSH by running the command **vagrant ssh**

Check ansible is installed - **ansible --version**

Step 8 - Create a new directory for your Ansible project on the controller VM by running the command **mkdir ansible-project**

Step 9 - Navigate to the ansible-project directory and create a new file called hosts by running the command **touch hosts**

Step 10 - Create a new file called **playbook.yml**. This file will contain the tasks you want to perform on your managed hosts

As of now the hosts and the playbook file are empty

We will now create some host machines that will be controlled by the Ansible controller

---

## Part B - Ansible Host Machines Setup

Step 1 - On terminal navigate to your Ansible Project folder

Step 2 - Create a new directory for your host machines by running the command

Step 3 - Navigate to host-machines directory and create a new Vagrantfile by running the command **vagrant init centos/7**

Step 4 - Edit the Vagrantfile and modify the following lines to set up two Vagrant machines:

Vagrantfile for creating VMs for Ansible Host

```
Vagrant.configure("2") do |config|
  config.vm.box = "centos/7"

  config.vm.define "web" do |web|
    web.vm.hostname = "web"
    web.vm.network "private_network", ip: "192.168.33.10"
  end

  config.vm.define "db" do |db|
    db.vm.hostname = "db"
    db.vm.network "private_network", ip: "192.168.33.11"
  end

  config.vm.network "forwarded_port", guest: 80, host: 8080, auto_correct: true
  config.vm.usable_port_range = (8000..9000)

end
```

Step 6 - Save & check its a valid vagrantfile **vagrant validate** Then run command **vagrant up** to start the VM

Step 7 - Check the status of machines **vagrant status**

Once the VMs are up, connect to them using SSH **vagrant ssh <machine-name>** e.g **vagrant ssh web**

This completes the process of setting up host machines

---

## Part C - Making connection between controller and host machines

Step 1 - Make sure all machines are up and running

Step 2 - Run command **ip addr** on each machine and check they have IP addresses in the same range (e.g. 192.168.33.x).

Step 3 - On Controller machine run the command **ssh-keygen** to generate an SSH key pair

Step 4 - Goto **~/.ssh** folder and check the public and private keys generated

Step 5 - Copy the public key to the host machines by running the command

**ssh-copy-id <user>@<host>**

For example, to copy the public key to the web machine, run the command

**ssh-copy-id vagrant@192.168.33.10**

Can do this manually by copying the contents of the .pub file generated by ssh-keygen and pasting it into the ~/.ssh/authorized\_keys file on the host machines

Step 6 - Test the SSH connection by running the ssh command with the IP address of the host machines-

For example: **ssh vagrant@192.168.33.10**  
**ssh vagrant@192.168.33.11**

---

## Part D - Adding host and playbook file on Controller and Run Playbook

Step 1 - Connect to the ansible controller machine using ssh **vagrant ssh <machine name>**

Step 2 - Edit the hosts file (**vi hosts**) to add the IP addresses or hostnames of the machines you want to control. For example:

### Ansible hosts file

```
[webservers]  
192.168.33.10
```

```
[dbservers]  
192.168.33.11
```

Step 3 - Run the command **ansible all -m ping -i hosts** to test the connection to the host machines

Step 4 - Now edit the playbook.yml file and add instructions for host machines

For e.g. to install the Apache web server on your webservers, you can use the following playbook

### Ansible Playbook to install and start Apache web server

```
---  
- name: Install Apache web server  
  hosts: dbservers  
  become: true
```



tasks:

- name: Install Apache

  - yum:

    - name: httpd

    - state: latest

- name: Start Apache

  - service:

    - name: httpd

    - state: started

    - enabled: true

Step 5 - You can now run the playbook on the managed hosts by running the command **ansible-playbook -i hosts playbook.yml**

Step 6 - Once the playbook has run, you can access the web servers by opening a web browser and navigating to the IP addresses of your web servers

---

## Part E - Adding Ansible within Vagrantfile as Provisioner

If we install Ansible and Vagrant on the same machine, we can use Ansible as a Provisioner in Vagrantfile

```
config.vm.provision "ansible" do |ansible|  
  ansible.playbook = "playbook.yml"
```

## UNIT V - BUILDING DEVOPS PIPELINES USING AZURE

**Create Github Account, Create Repository, Create Azure Organization, Create a new pipeline, Build a sample code, Modify azure-pipelines.yaml file**

### Github Account Creation

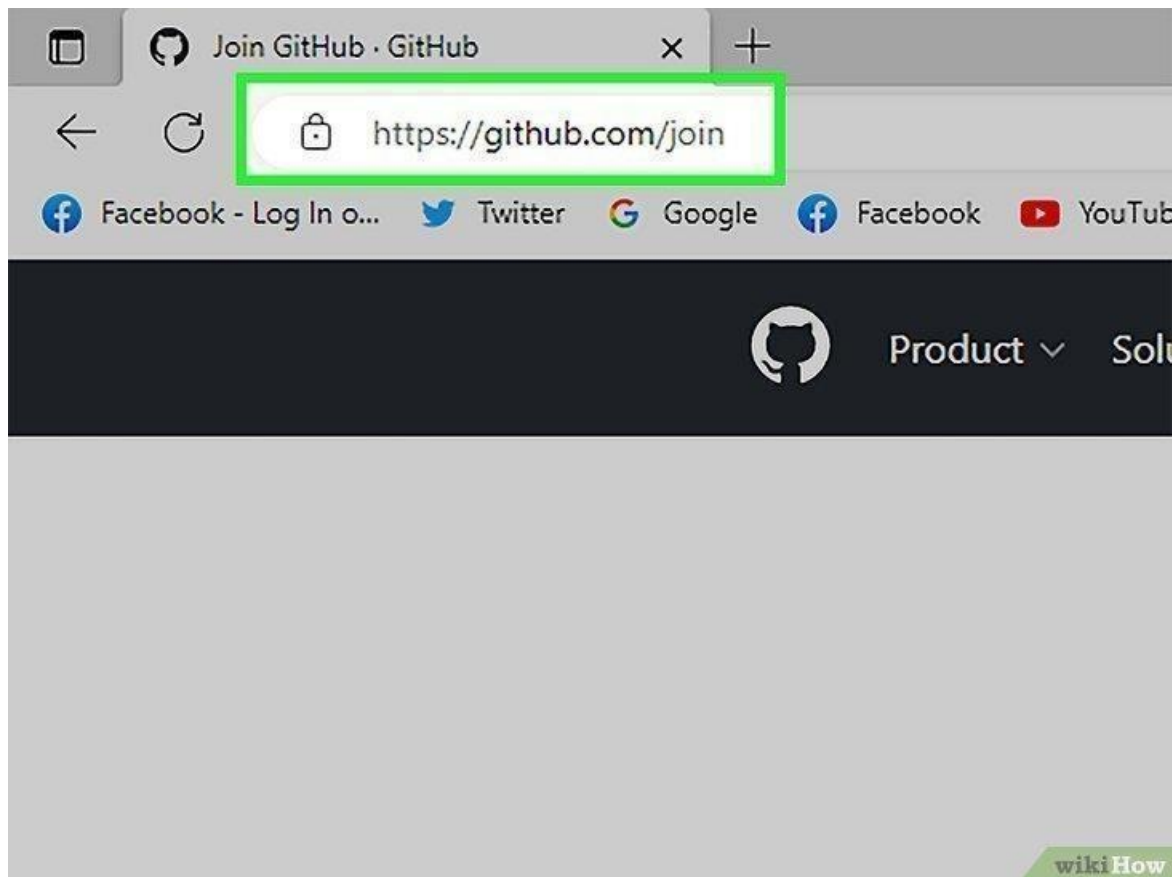
#### What is Github?

GitHub is a code hosting platform for collaboration and version control.

GitHub lets you and others work together on projects from anywhere.

GitHub is owned by Microsoft, provides access to public(free) and private(paid) repositories.

#### Steps to create Github Account:



Step 1: Go to <https://github.com/join> in a web browser.

**Step 2: Enter your personal details.** In addition to creating a username and entering an email address, you'll also have to create a password. Your password must be at least 15 characters in length or at least 8 characters with at least one number and lowercase letter.

Join GitHub

## First, let's create your user account

**Username \***

 ✓

**Email address \***

**Password \***

 ✓

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter.  
[Learn more.](#)

**Email preferences**

☒ Send me occasional product updates, announcements, and offers.

**Verify your account**

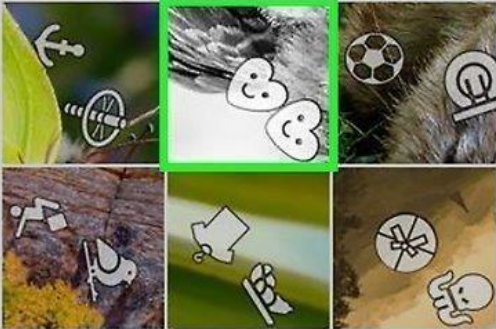
**Step 3: Click Verify to start the verification puzzle.** The instructions vary by puzzle, so just follow the on-screen instructions to confirm that you are a human. A green checkmark will appear after completing the puzzle.



Email preferences

☒ Send me occasional product updates, announcements, and offers.

Verify your account

Pick one square that shows two identical objects.




wikiHow

**Step 4: Click the green Create account button.** It's below the form, at the bottom of the page. This will take you to an email verification page.

Verify your account

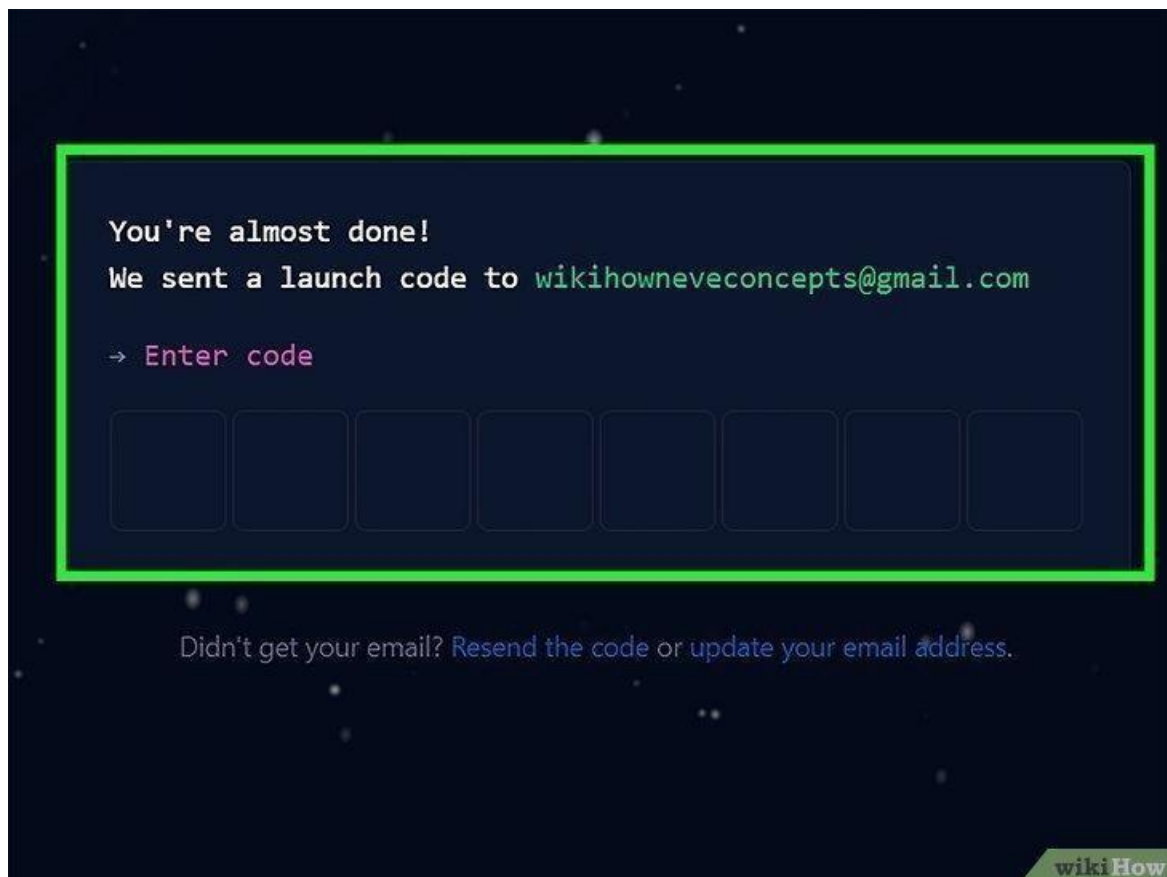


Create account

By creating an account, you agree to the [Terms of Service](#). For more information about GitHub's privacy practices, see the [GitHub Privacy Statement](#). We'll occasionally send you account-related

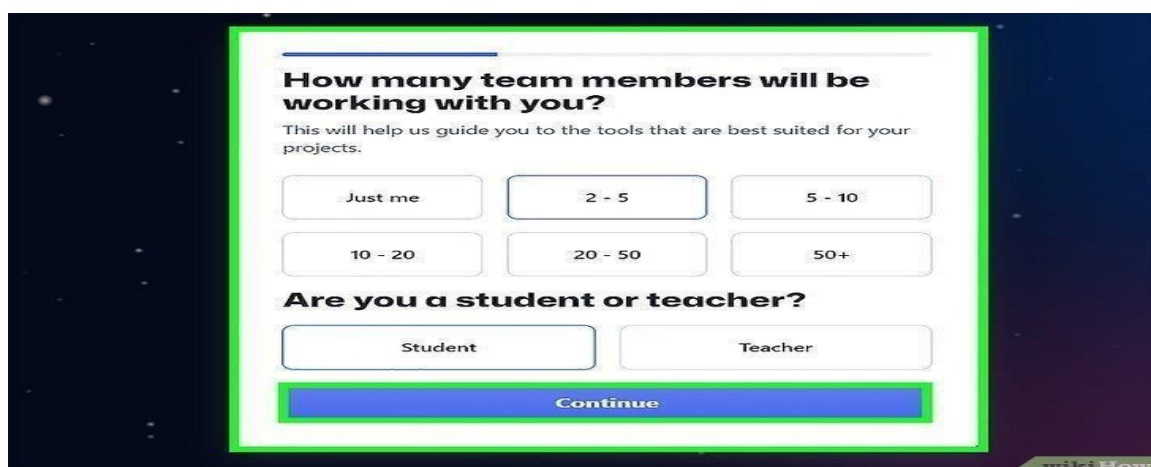
wikiHow

Carefully review the Terms of Service at <https://help.github.com/en/articles/github-terms-of-service> and the Privacy Statement at <https://help.github.com/en/articles/github-privacy-statement> before you continue.



**Step 5: Verify your email by entering the code.** After clicking Create account, you'll receive an email with a code. Enter this code on the verification page. Entering the code will automatically take you to the welcome page.

**Step 6:** Select your preferences and click Continue. GitHub displays a quick survey that can help you tailor your experience to match what you're looking for. You'll be sent to the plan selection page after completing the survey.



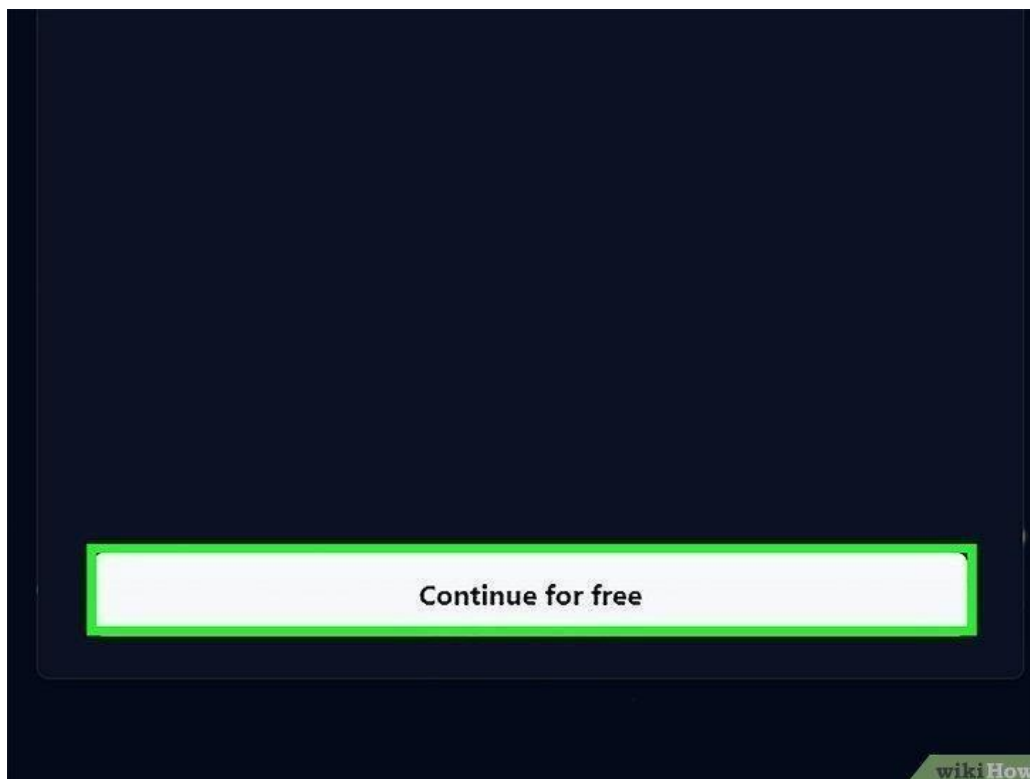
**Step 7:** Note the types of plans offered by GitHub. There are a few different plans to choose from, varying in the amount of features provided.



The screenshot shows the GitHub Student Developer Pack plan selection page. At the top, it says "Learn to ship software like a pro." and "GitHub gives students free access to the best developer tools so they can learn by doing." Below this, there are two columns of features. The left column is titled "Free" and lists: "Unlimited public/private repositories", "2,000 CI/CD minutes/month (Free for public repositories)", "500MB of Packages storage (Free for public repositories)", "120 core-hours of Codespaces compute", "15GB of Codespaces storage", and "Community support". The right column is titled "Get additional student benefits" and lists "GitHub Pro" benefits: "Protect your branches (Ensure that collaborators on your repository cannot make irrevocable changes to branches)", "Draft pull requests", "Pages and Wikis", "3,000 CI/CD minutes/month (Free for public repositories)", "2GB of Packages storage (Free for public repositories)", "180 core-hours of Codespaces compute", "20GB of Codespaces storage", and "Web-based support". A "wikiHow" logo is visible in the bottom right corner.

Free	Get additional student benefits
Unlimited public/private repositories	GitHub Pro
2,000 CI/CD minutes/month <small>Free for public repositories</small>	Protect your branches <small>Ensure that collaborators on your repository cannot make irrevocable changes to branches.</small>
500MB of Packages storage <small>Free for public repositories</small>	Draft pull requests
120 core-hours of Codespaces compute	Pages and Wikis
15GB of Codespaces storage	3,000 CI/CD minutes/month <small>Free for public repositories</small>
Community support	2GB of Packages storage <small>Free for public repositories</small>
	180 core-hours of Codespaces compute
	20GB of Codespaces storage
	Web-based support

**Step 8:** Select the free plan. On the plan selection page, scroll down to click the button for choosing a free plan. This will immediately take you to your GitHub dashboard.



The screenshot shows a dark blue background with a large, white button with a green border that says "Continue for free". A "wikiHow" logo is visible in the bottom right corner.

- If you choose a paid plan, you'll have to enter your payment information as requested before you can continue.
- If you want to upgrade your Github account in the future, click the menu at the top-right corner, select Settings, and choose Billing and plans to view your options.

### GitHub essentials are:

- Repositories
- Branches
- Commits
- Pull Requests
- Git (the version control software GitHub is built on)

### Repository:

A GitHub repository can be used to store a development project.

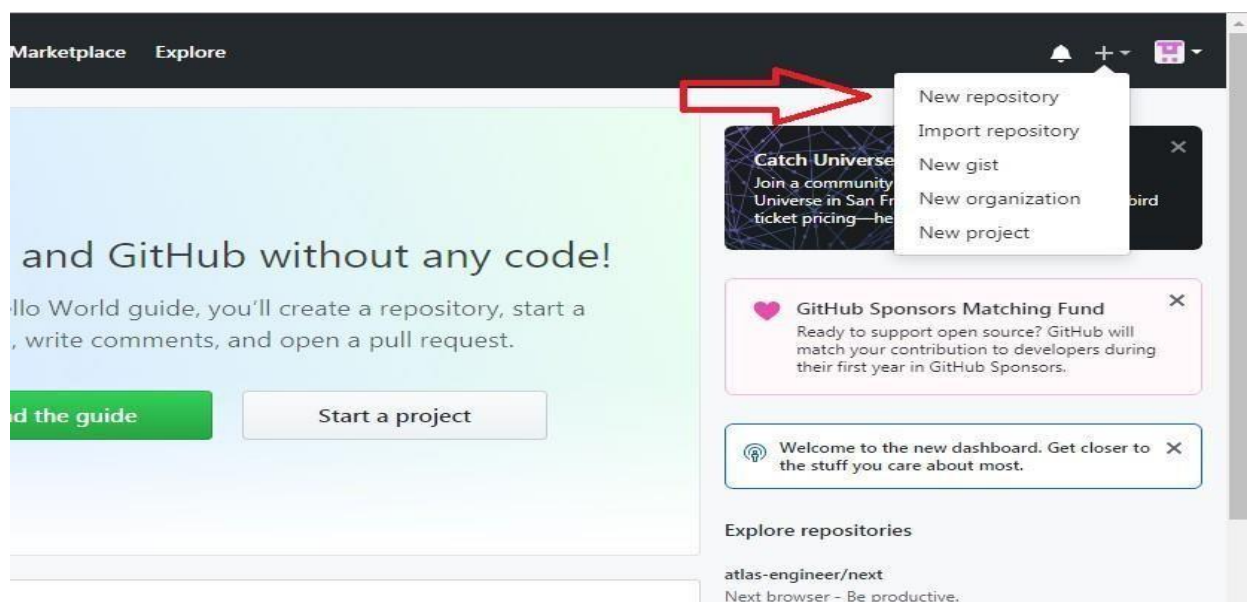
It can contain folders and any type of files (HTML, CSS, JavaScript, Documents, Data, Images).

A GitHub repository should also include a licence file and a README file about the project.

A GitHub repository can also be used to store ideas, or any resources that you want to share.

### Github Repository Creation

**Step 1:** Click on the **new repository** option






**Step 2:** After clicking **new repository option**, we will have to initialize some things like, naming our **project**, choosing the **visibility** etc. After performing these steps click **Create Repository button**.

### Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?  
[Import a repository.](#)

Owner: Namanbhatia7 / Repository name: Resume  This is going to be name of our project

Great repository names are short and memorable. Need inspiration? How about [animated-memory](#)?


Description (optional):

☒ **Public** Keep this as public selected  
Anyone can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository. We can add a project description if we want.

Skip this step if you're importing an existing repository.

☒ **Initialize this repository with a README** Tick the README option  
This will let you immediately clone the repository to your computer.

Add .gitignore: None | Add a license: None 

**Create repository** After performing above steps, Click this button

**Step 3:** After clicking the button, we will be directed to below page. Right now the only file we have is a readme file.

Namanbhatia7 / Resume Unwatch 1 Star 0 Fork 0

[Code](#) [Issues 0](#) [Pull requests 0](#) [Projects 0](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

No description, website, or topics provided. [Edit](#)

[Manage topics](#)

1 commit 1 branch 0 releases 1 contributor

Branch: master [New pull request](#) [Create new file](#) [Upload files](#) [Find File](#) [Clone or download](#)

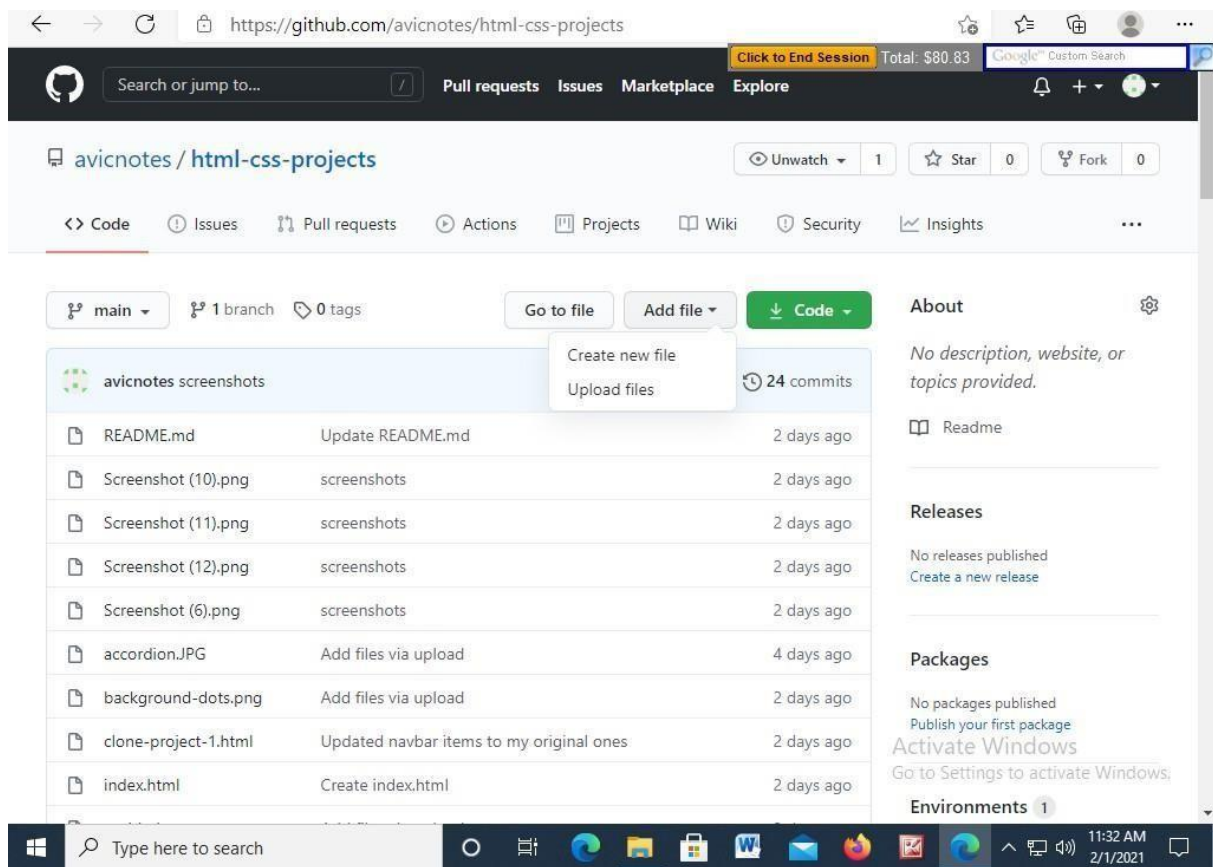
Namanbhatia7 Initial commit Latest commit 676ac98 now

README.md Initial commit now

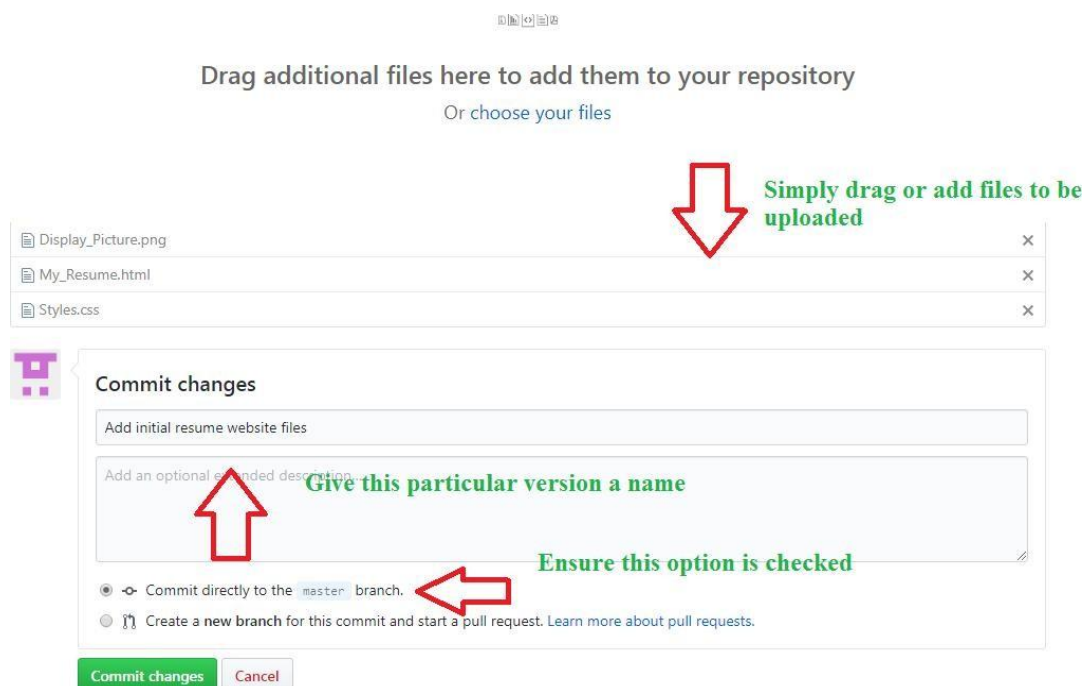
README.md [Edit](#)

Resume

**Step 4:** Now click on the “Upload files” button.



**Step 5:** Follow the steps mentioned and click “commit changes”.



**Step 6:** Now you will see that all of our files uploaded in our github.

Namanbhatia7 Add initial resume website files		Latest commit f47be5b 1 minute ago
Display_Picture.png	Add initial resume website files	1 minute ago
My_Resume.html	Add initial resume website files	1 minute ago
README.md	Initial commit	15 minutes ago
Styles.css	Add initial resume website files	1 minute ago

## **Branch:**

- A GitHub branch is used to work with different **versions** of a repository at the same time.
- By default a repository has a **master** branch (a production branch).
- Any other branch is a **copy** of the master branch (as it was at a point in time).
- New Branches are for bug fixes and feature work separate from the master branch. When changes are ready, they can be merged into the master branch. If you make changes to the master branch while working on a new branch, these updates can be pulled in.

## **Commits:**

At GitHub, changes are called commits.

Each commit (change) has a description explaining why a change was made.

## **Pull Requests :**

- Pull Requests are the heart of GitHub **collaboration**.
- With a pull request you are proposing that your changes should be **merged** (pulled in) with the master.
- Pull requests show content **differences**, changes, additions, and subtractions in colors (green and red).
- As soon as you have a commit, you can open a pull request and start a discussion, even before the code is finished.

## **Git:**

- Git was created by Linus Torvalds in 2005 to develop Linux Kernel
- Git is an **open-source distributed version control system**. It is designed to handle minor to major projects with high speed and efficiency.
- It is developed to co-ordinate the work among the developers. The version control allows us to track and work together with our team members at the same workspace.

## CREATE AZURE ORGANIZATION

### AZURE:

**Microsoft Azure**, often referred to as **Azure**

**cloud computing** platform run by **Microsoft**. It offers access, management, and the development of applications and services through global **data centers**

It also provides a range of capabilities, including **software as a service (SaaS)**, **platform as a service**, and **infrastructure as a service (IaaS)**.

It was officially launched as Windows Azure in February 2010 and later renamed Microsoft Azure on March 25, 2014

Microsoft Azure supports many **programming languages**, tools, and frameworks, including Microsoft-specific and third-party software and **systems**.

### Organization:

All organizations must be manually created via the web portal. We don't support automated creation of organizations. We do support automated organization configuration, project creation, and resource provisioning via **REST API**.

### Prerequisites:

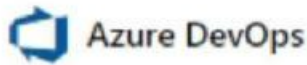
To **plan your organizational structure**.

Microsoft accounts or authenticate users with

Microsoft Entra ID. For more information, see **Choosing your organization administrator account type**.

### Create an organization:

1. Sign in to **Azure DevOps**.
2. Select **New organization**
3. Confirm information, and then select **Continue**



Taking you to your Azure  
DevOps organization...



**Congratulations, you're an organization owner!**

Sign in to your organization at any  
time, <https://dev.azure.com/{yourorganization}>.

**With your organization, the following aspects are included in the free tier:**

**First five users free (Basic license):**

**Azure Pipelines:**

- One **Microsoft-hosted CI/CD** (one concurrent job, upto 30 hours per month)
- One self-hosted CI/CD concurrent job

**Azure Boards:**

Work item tracking and Kanban boards

**Azure Repos:**

Unlimited private Git repos

**Azure Artifacts:**

Two GB free per organization

**Build applications with Azure:**

Azure DevOps enables you to build, test, and deploy any application to any cloud or on premises

To configure build pipelines that continuously build, test, and verify your applications.

**Part 1: Get started with Azure DevOps**

**Part 2: Build applications with Azure DevOps**

**Part 3: Deploy applications with Azure DevOps**

**Create a build pipeline with Azure Pipelines:**

**Prerequisites:**

Familiarity with forking and cloning a GitHub repo

**Account requirements:**

An Azure DevOps organization

- To use Microsoft-hosted agents, your Azure DevOps organization must have access to **Microsoft- hosted** parallel jobs. **Check your parallel jobs and request a free grant.**
- You can use GitHub Codespaces to complete the module, even if your Azure DevOps organization doesn't have any parallel jobs.

A **GitHub** account

**Software requirements:**

If using GitHub Codespaces to complete the module, there are no software requirements as all software is included in the Codespace

If using a local development environment with Microsoft-hosted agents,

**you must have the following software installed:**

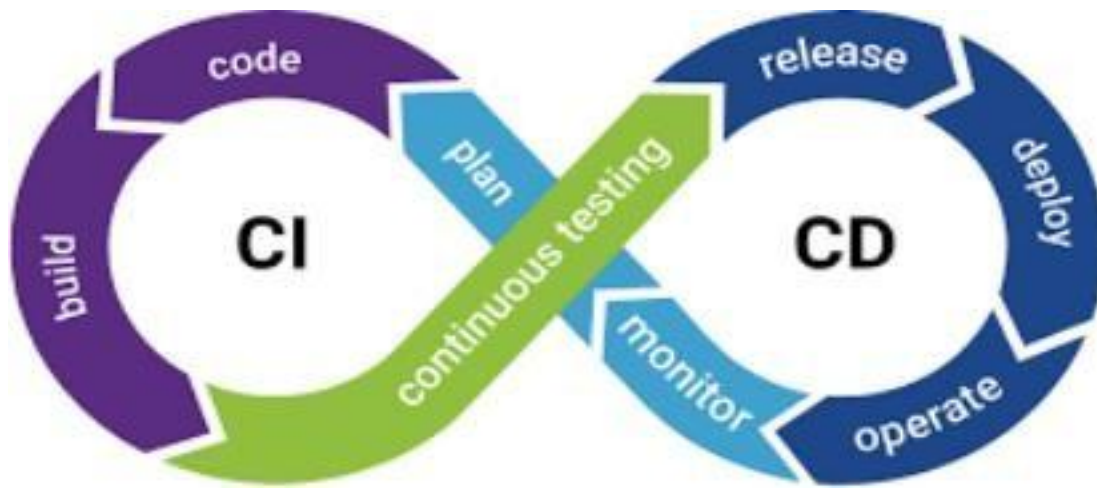
- **Visual Studio Code**
- **.NET 6.0 SDK**
- **Git**

## Create a pipeline in azure

What is pipeline

Azure Pipelines supports continuous integration (CI) and continuous delivery (CD) to continuously test, build, and deploy your code. You accomplish this by defining a pipeline.

The latest way to build pipelines is with the YAML pipeline editor. You can also use Classic pipelines with the Classic editor.



## Create your first pipeline

This is a step-by-step guide to using Azure Pipelines to build a sample application from a Git repository. This guide uses YAML pipelines configured with the YAML pipeline editor.

If you'd like to use Classic pipelines instead, see [Define your Classic pipeline](#). For guidance on using TFVC, see [Build TFVC repositories](#).

## Prerequisites - Azure DevOps

A GitHub account where you can create a repository.

An Azure DevOps organization. Create one for free. If your team already has one, then make sure you're an administrator of the Azure DevOps.



## Get the Java sample code

To get started, fork the following repository into your GitHub account.

<https://github.com/MicrosoftDocs/pipelines-java>

## Create your first Java pipeline

1. Sign-in to your Azure DevOps organization and go to your project.
2. Go to Pipelines, and then select New pipeline.
3. Do the steps of the wizard by first selecting GitHub as the location of your source code.
4. You might be redirected to GitHub to sign in. If so, enter your GitHub credentials.
5. When you see the list of repositories, select your repository.
6. You might be redirected to GitHub to install the Azure Pipelines app. If so, select Approve & install.
7. Azure Pipelines will analyze your repository and recommend the Maven pipeline template.
8. When your new pipeline appears, take a look at the YAML to see what it does. When you're ready, select Save and run.
9. You're prompted to commit a new `azure-pipelines.yml` file to your repository. After you're happy with the message, select Save and run again.

If you want to watch your pipeline in action, select the build job.

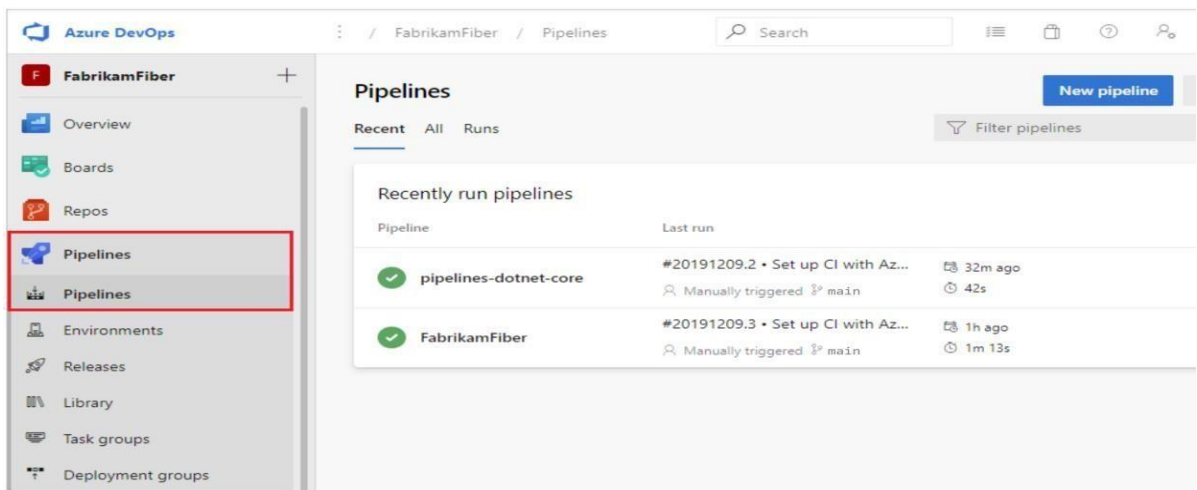
You just created and ran a pipeline that we automatically created for you, because your code appeared to be a good match for the Maven template.

You now have a working YAML pipeline (`azure-pipelines.yml`) in your repository that's ready for you to customize!

10. When you're ready to make changes to your pipeline, select it in the Pipelines page, and then Edit the azure-pipelines.yml file.

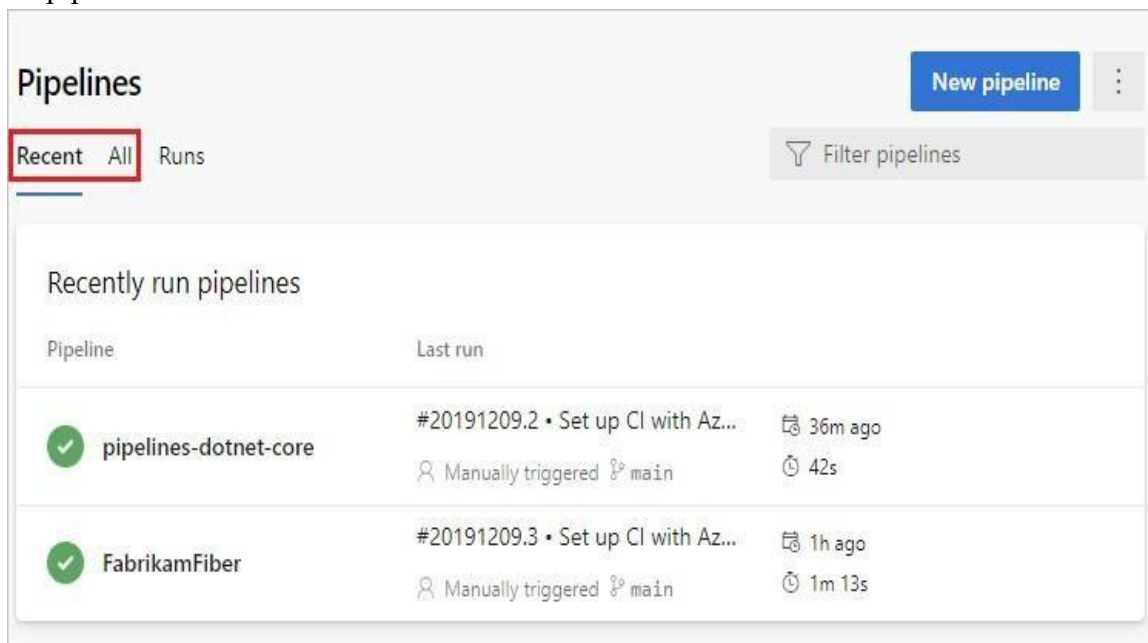
## View and manage your pipelines

You can view and manage your pipelines by choosing Pipelines from the left-hand menu to go to the pipelines landing page.

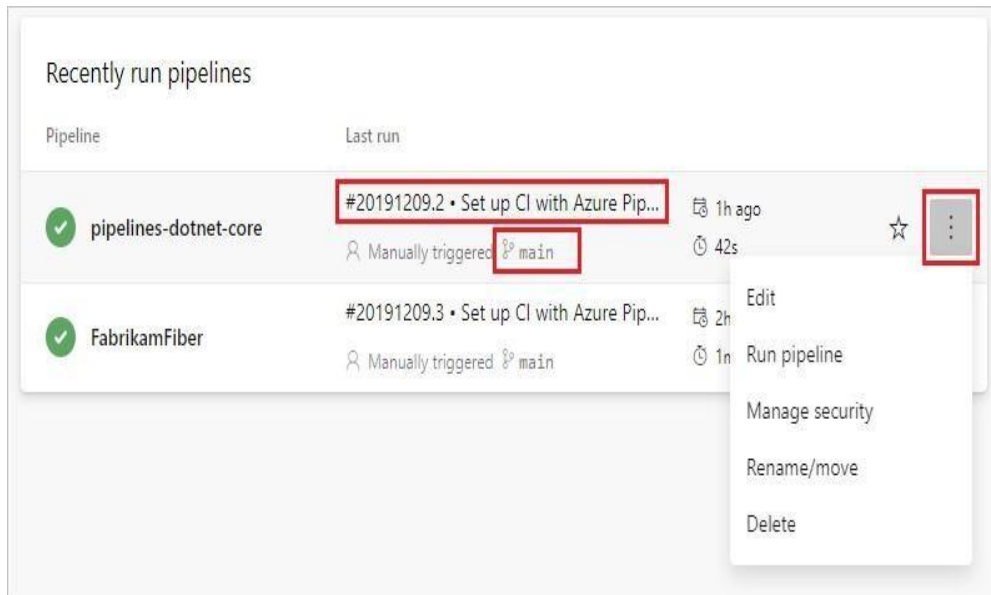


From the pipelines landing page you can view pipelines and pipeline runs, create and import pipelines, manage security, and drill down into pipeline and run details.

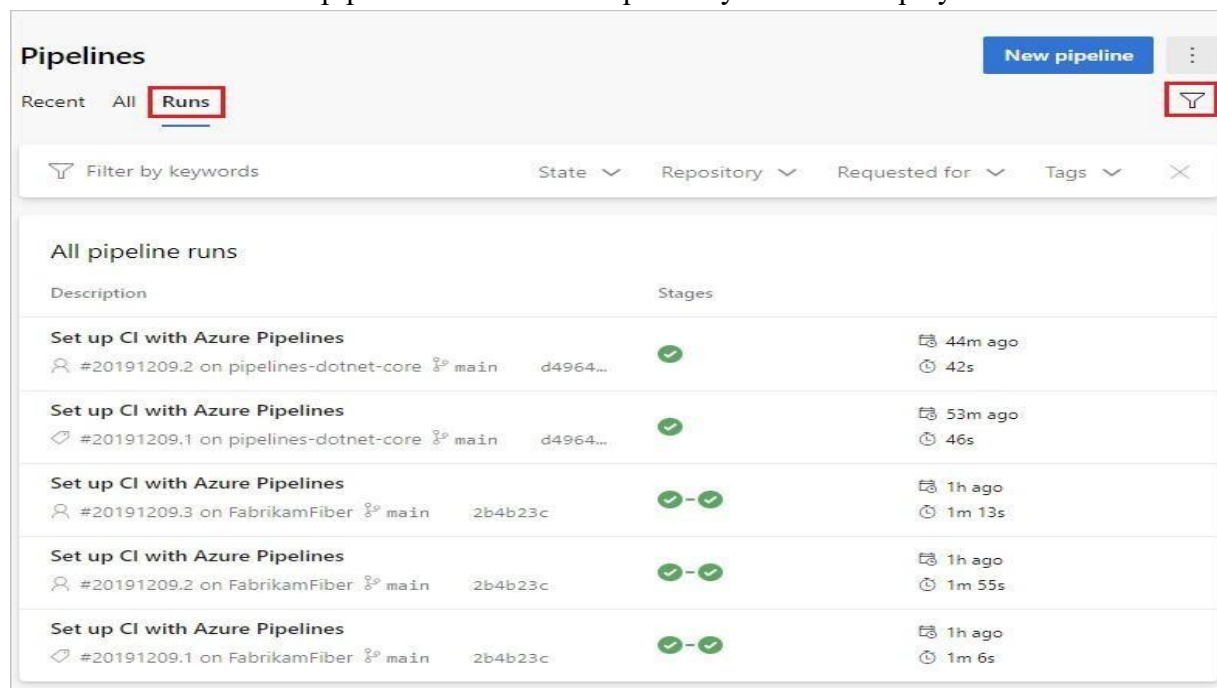
Choose Recent to view recently run pipelines (the default view), or choose All to view all pipelines.



Select a pipeline to manage that pipeline and view the runs. Select the build number for the last run to view the results of that build, select the branch name to view the branch for that run, or select the context menu to run the pipeline and perform other management actions.

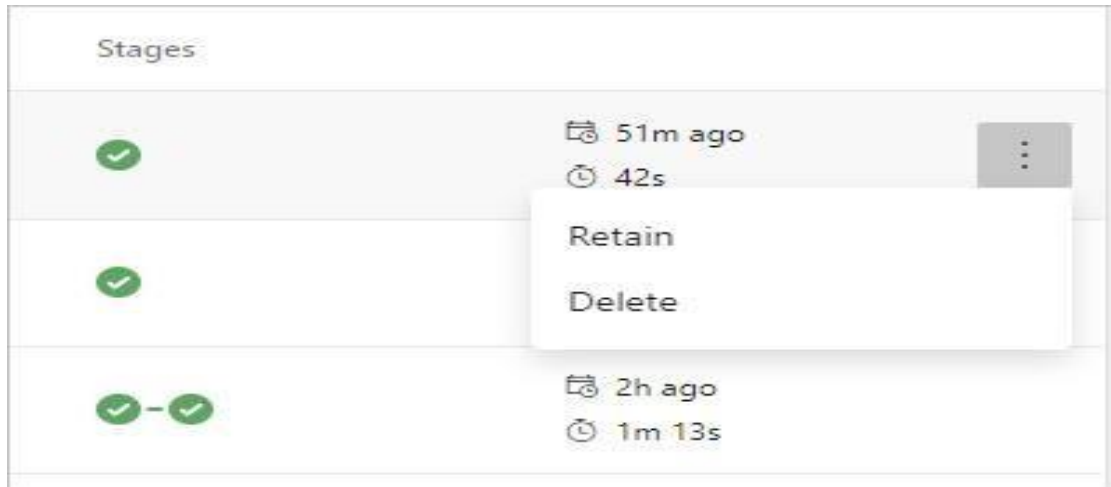


Select Runs to view all pipeline runs. You can optionally filter the displayed runs.



Select a pipeline run to view information about that run.

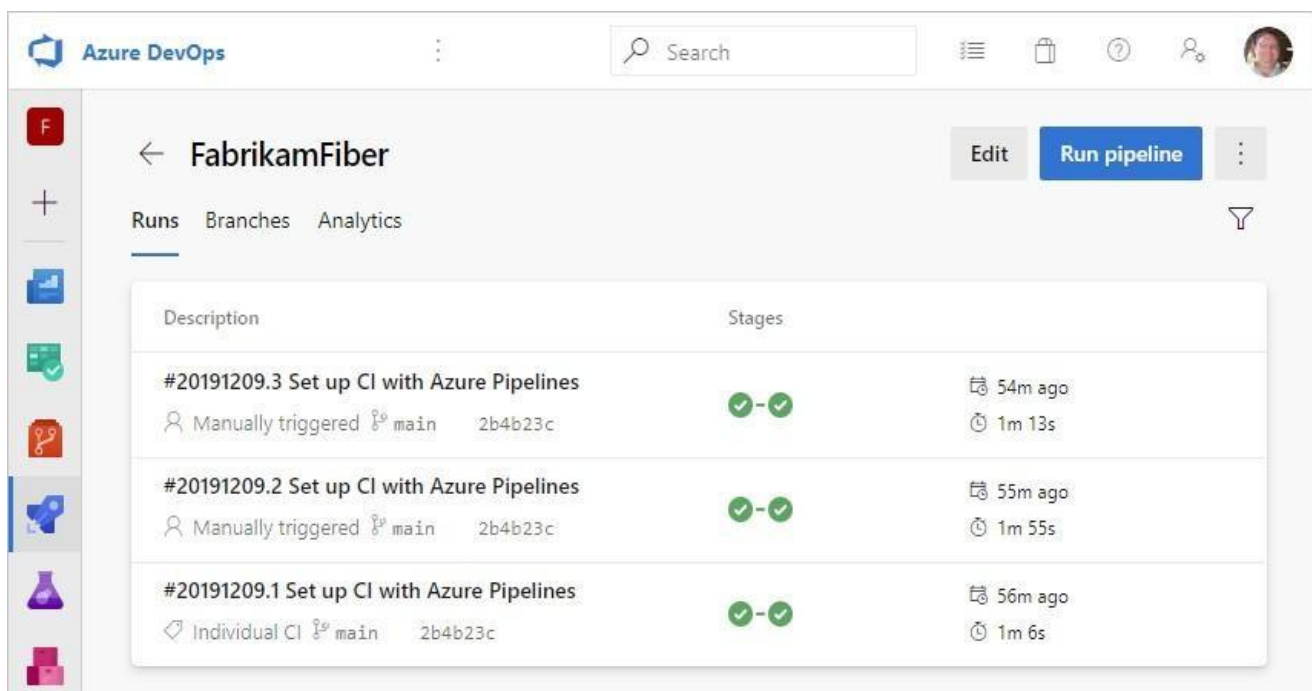
You can choose to Retain or Delete a run from the context menu. For more information on run retention, see [Build and release retention policies](#).



## View pipeline details

The details page for a pipeline allows you to view and manage that pipeline.

Choose Edit to edit your pipeline. For more information, see [YAML pipeline editor](#). You can also edit your pipeline by modifying the `azure-pipelines.yml` file directly in the repository that hosts the pipeline.



Choose Edit to edit your pipeline. For more information, see [YAML pipeline editor](#). You can also edit your pipeline by modifying the `azure-pipelines.yml` file directly in the repository that hosts the pipeline.

## View pipeline run details

From the pipeline run summary you can view the status of your run, both while it is running and when it is complete.

The screenshot shows the Azure Pipelines interface for a specific run. At the top, the run is identified by a green checkmark, the name '#20191210.2 Update azure-pipelines.yml for Azure Pipe...', and the location 'on FabrikamFiber'. A 'Run new' button and a menu icon are to the right. Below this, the 'Summary' tab is selected, showing details about the run: it was triggered by Steve Danielson, on the main branch of FabrikamFiber, at 12:56 PM today. The duration is 1m 9s, and there are 2 commits, 1 linked work item, and 1 published artifact. Below the summary, the 'Stages' tab is selected, showing a sequence of two stages: 'Build' (41s, 1 job completed) and 'Deploy' (13s, 1 job completed, 1 artifact). The stages are connected by a line, indicating a sequential flow.

From the summary pane you can view job and stage details, download artifacts, and navigate to linked commits, test results, and work items. From the summary pane you can view job and stage details, download artifacts, and navigate to linked commits, test results, and work items.

## Jobs and stages

The jobs pane displays an overview of the status of your stages and jobs. This pane may have multiple tabs depending on whether your pipeline has stages and jobs, or just jobs. In this example, the pipeline has two stages named Build and Deploy. You can drill down into the pipeline steps by choosing the job from either the Stages or Jobs pane.

Choose a job to see the steps for that job

From the

The screenshot shows the Azure Pipelines interface. On the left, a sidebar titled 'Jobs in run #20191...' lists the steps of a job. The 'Build' step is selected and expanded, showing its sub-steps: 'Initialize job', 'Checkout', 'CmdLine', 'Component Detect', 'Post-job: Checkout', and 'Finalize Job'. The 'Deploy' section shows 'DeployWeb', and the 'Finalize build' section shows 'Report build status'. On the right, the detailed view of the 'Build' job is shown, including a green checkmark icon, the job name 'Build', and a list of job details: 'Pool: Azure Pipelines', 'Image: Ubuntu-16.04', 'Agent: Hosted Agent', 'Started: Today at 1:13 PM', and 'Duration: 40s'. A search icon and a menu icon (three dots) are visible in the top right corner of the job details panel.

Step	Duration
Build	40s
Initialize job	1s
Checkout	3s
CmdLine	2s
Component Detect	32s
Post-job: Checkout	<1s
Finalize Job	<1s
DeployWeb	10s
Report build status	<1s

steps view, you can review the status and details of eachstep. From the Moreactions you can toggle timestamps or view a raw log of all steps in the pipeline.

The screenshot shows the detailed view of the 'Build' job. The job name 'Build' is displayed with a green checkmark icon. Below the job name, the job details are listed: 'Pool: Azure Pipelines', 'Image: Ubuntu-22.04', 'Agent: Hosted Agent', 'Started: Today at 1:13 PM', and 'Duration: 40s'. A search icon and a menu icon (three dots) are visible in the top right corner of the job details panel. The menu is open, showing two options: 'View job raw log' and 'Toggle timestamps'.

Build

1 Pool: Azure Pipelines

2 Image: Ubuntu-22.04

3 Agent: Hosted Agent

4 Started: Today at 1:13 PM

5 Duration: 40s

6

7 ► Job preparation parameters

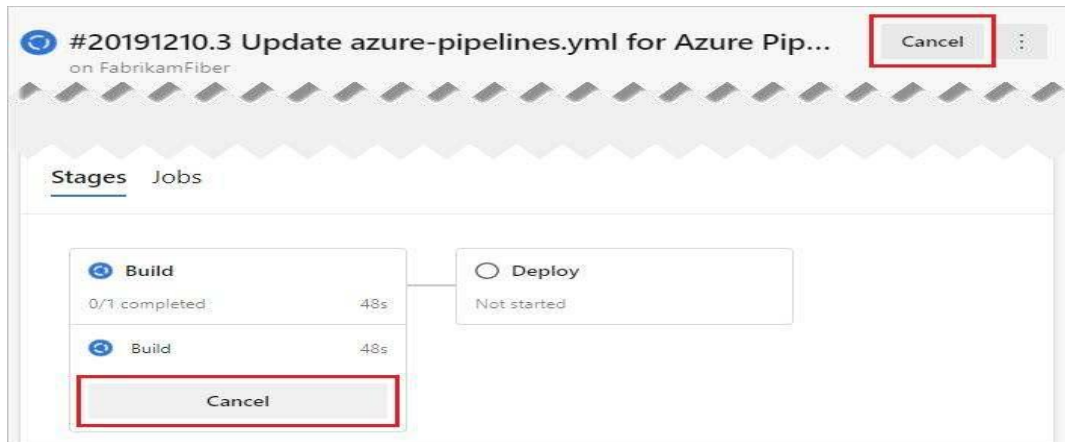
View job raw log

Toggle timestamps

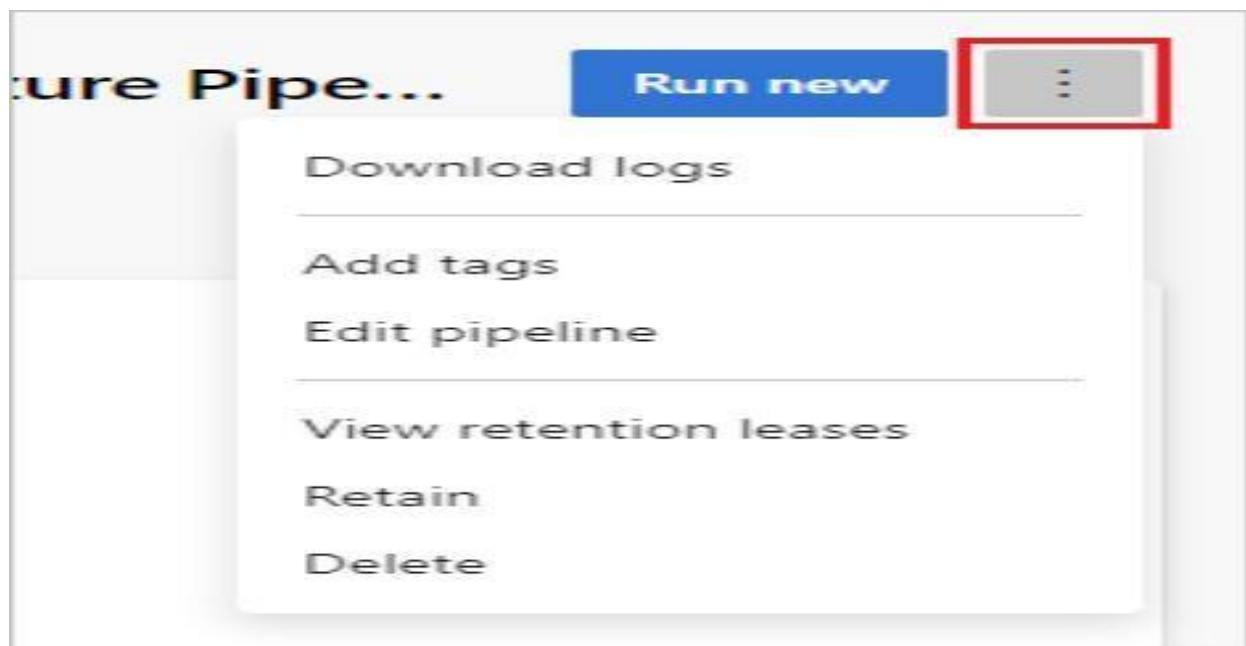
## Cancel and re-run a pipeline

If the pipeline is running, you can cancel it by choosing Cancel. If the run has completed, you can re-run the pipeline by choosing Run new.

Pipeline run more actions menu:



From the More actions menu you can download logs, add tags, edit the pipeline, delete the run, and configure retention for the run.






### **Add a status badge to your repository**

Many developers like to show that they're keeping their code quality high by displaying a status badge in their repo.



#### **To copy the status badge to your clipboard:**

1. In Azure Pipelines, go to the Pipelines page to view the list of pipelines. Select the pipeline you created in the previous section.
2. Select , and then select Status badge.
3. Select Status badge.
4. Copy the sample Markdown from the Sample markdown section.

#### **Now with the badge Markdown in your clipboard, take the following steps in GitHub:**

1. Go to the list of files and select Readme.md. Select the pencil icon to edit.
2. Paste the status badge Markdown at the beginning of the file.
3. Commit the change to the main branch.
4. Notice that the status badge appears in the description of your repository.

#### **To configure anonymous access to badges for private projects:**

1. Navigate to Project Settings in the bottom left corner of the page
2. Open the Settings tab under Pipelines
3. Toggle the Disable anonymous access to badges slider under General

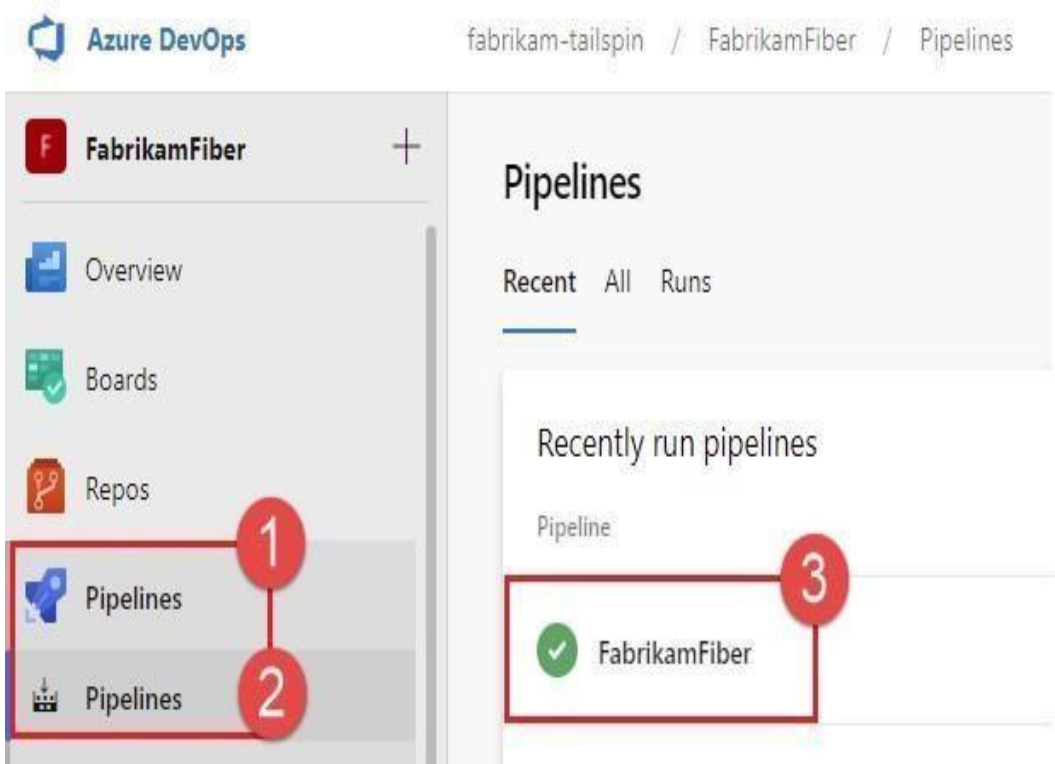
## Azure-Pipeline

- Azure Pipelines provides a YAML pipeline editor that you can use to author and edit your pipelines.
- The YAML editor is based on the [Monaco Editor](#).
- The editor provides tools like Intellisense support and a task assistant to provide guidance while you edit a pipeline.
- You can also edit pipelines by modifying the **azurepipelines.yml** file directly in your pipeline's repository using a text editor of your choice

### Edit a YAML pipeline

To access the YAML pipeline editor, do the following steps.

- Sign in to your organization
- (<https://dev.azure.com/{yourorganization}>).
- Select your project, choose **Pipelines**, and then select the pipeline you want to edit. You can browse pipelines by **Recent**, **All**, and **Runs**.
- Choose **Edit**.
- Make edits to your pipeline using [Intellisense](#) and the [task assistant](#) for guidance.



← FabrikamFiber

Edit

Run pipeline

⋮

Runs

Branches

Analytics

⌵

Description	Stages	
<div>#20210609.3 Update azure-pipelines.yml for Azure Pipelines</div> <div> <div>Manually triggered for</div> <div> <div></div> <div>main</div> </div> <div> <div>61bba8c</div> <div></div> </div> </div>	<div>✓</div>	<div> <div>Wednesday</div> <div>21s</div> </div>
<div>#20210609.2 Update azure-pipelines.yml for Azure Pipelines</div> <div> <div>Manually triggered for</div> <div> <div></div> <div>main</div> </div> <div> <div>bc6c105</div> <div></div> </div> </div>	<div>✓</div>	<div> <div>Wednesday</div> <div>30s</div> </div>
<div>#20210609.1 Update azure-pipelines.yml for Azure Pipelines</div> <div> <div>Manually triggered for</div> <div> <div></div> <div>main</div> </div> <div> <div>53eb92b</div> <div></div> </div> </div>	<div>✓</div>	<div> <div>Wednesday</div> <div>45s</div> </div>
<div>#20210520.1 Updated README.md</div> <div> <div>Individual CI for</div> <div> <div></div> <div>new-branch</div> </div> <div> <div>1ada387</div> <div></div> </div> </div>	<div>✓</div>	<div> <div>May 20</div> <div>13s</div> </div>

← azure-cli-example

Variables

Run

⋮

main

azure-cli-example / azure-pipelines.yml

```

30 pool: default
31
32 steps:
33   # Specify python version and install if needed
34   - task: UsePythonVersion@0
35     condition: false
36     inputs:
37       versionSpec: '3.x'
38       architecture: 'x64'
39
40   # Update pip to latest
41   - bash: python -m pip install --upgrade pip
42     condition: false
43     displayName: 'Upgrade pip'
44
45   container
46   continueOnError
47   name
48   parameters
49   pr
50   resources
51   schedules
52   services
53   strategy
54   variables
55   workspace

```

Tasks

Search tasks

dotnet

.NET Core

Build, test, package, or publish a dotnet applicatio.

android

Android signing

Sign and align Android APK files

ant

Ant

Build with Apache Ant

appcenter

App Center distribute

Distribute app builds to testers and users via Visu.

appcenter

App Center test

Test app packages with Visual Studio App Center

archive

Archive files

Compress files into .7z, .tar.gz, or .zip

arm

ARM template deployment

Deploy an Azure Resource Manager (ARM) templ.

azure

Azure App Service deploy

Deploy to Azure App Service a web, mobile, or AP.

- The YAML pipeline editor provides several keyboardshortcuts, which we show in the following examples.
- Choose **Ctrl+Space** for Intellisense support while you'reediting the YAML pipeline.

- The task assistant provides a method for adding tasks to your YAML pipeline.
- To display the task assistant, edit your YAML pipeline and choose **Show assistant**.

### Understand the azure-pipelines.yml file

- A pipeline is defined using a YAML file in your repo. Usually, this file is named azure-pipelines.yml and is located at the root of your repo.
- Navigate to the **Pipelines** page in Azure Pipelines, select the pipeline you created, and choose **Edit** in the context menu of the pipeline to open the YAML editor for the pipeline.
- This pipeline runs whenever your team pushes a change to the main branch of your repo or creates a pull request. It runs on a Microsoft-hosted Linux machine.
- The pipeline process has a single step, which is to run the Maven task.

### YAML:Code

```
trigger:
- main

strategy:
  matrix:
    jdk10_linux:
      imageName: "ubuntu-latest"
      jdkVersion: "1.10"
    jdk11_windows:
      imageName: "windows-latest"
      jdkVersion: "1.11"
  maxParallel: 2

pool:
  vmImage: $(imageName)

steps:
- task: Maven@4
  inputs:
    mavenPomFile: "pom.xml"
    mavenOptions: "-Xmx3072m"

    javaHomeOption: "JDKVersion"
    jdkVersionOption: $(jdkVersion)
    jdkArchitectureOption: "x64"
    publishJUnitResults: true
    testResultsFiles: "**/TEST-*.xml"
  goals:
    "package"
```

### Change the platform to build

- Navigate to the editor for your pipeline by selecting **Edit pipeline** action on the build, or by selecting **Edit** from the pipeline's main page.
- To choose a different platform like Windows or Mac, change the vmImage value:
- pool:
- vmImage: "windows-latest" **Add steps**
- You can add more **scripts** or **tasks** as steps to your pipeline. A task is a pre-packaged script. You can use tasks for building, testing, publishing, or deploying your app. For Java, the

Maven task we used handles testing and publishing results, however, you can use a task to publish code coverage results too.

## Customize CI triggers

Pipeline triggers cause a pipeline to run. You can use `trigger:` to cause a pipeline to run whenever you push an update to a branch. YAML pipelines are configured by default with a CI trigger on your default branch (which is usually `main`). You can set up triggers for specific branches or for pull request validation. For a pull request validation trigger, just replace the `trigger:` step with `pr:` as shown in the two examples below. By default, the pipeline runs for each pull request change.

- If you'd like to set up triggers, add either of the following snippets at the beginning of your `azure-pipelines.yml` file.

```
YAML Copy  
  
trigger:  
  - main  
  - releases/*
```

```
YAML Copy  
  
pr:  
  - main  
  - releases/*
```

You can specify the full name of the branch (for example, `main`) or a prefix-matching

- Open the YAML editor for your pipeline.
- Add the following snippet to the end of your YAML file.

### Build using multiple versions

## Add steps

You can add more `scripts` or `tasks` as steps to your pipeline. A task is a pre-packaged script. You can use tasks for building, testing, publishing, or deploying your app. For Java, the Maven task we used handles testing and publishing results, however, you can use a task to publish code coverage results too.

- Open the YAML editor for your pipeline.
- Add the following snippet to the end of your YAML file.

```
YAML Copy  
  
- task: PublishCodeCoverageResults@1  
  inputs:  
    codeCoverageTool: "JaCoCo"  
    summaryFileLocation: "$(System.DefaultWorkingDirectory)**/site/jacoco/  
reportDirectory: "$(System.DefaultWorkingDirectory)**/site/jacoco"  
failIfCoverageEmpty: true
```

- Select **Save** and then confirm the changes.
- You can view your test and code coverage results by selecting your build and going to the **Test** and **Coverage** tabs.

Build using multiple versions  
Rename/move pipeline

Name

ScheduledTriggerTest 1234

Select folder

\

...

Cancel

Save

Pipeline settings

✕

Processing of new run requests

- ☒ Enabled
- ☐ Paused
- ☐ Disabled

YAML file path

azure-pipelines.yml

▼

☐ Automatically link work items included in this run

Cancel

Save

YAML

```
trigger:
- main

strategy:
  matrix:
    jdk10_linux:
      imageName: "ubuntu-latest"
      jdkVersion: "1.10"
    jdk11_windows:
      imageName: "windows-latest"
      jdkVersion: "1.11"
  maxParallel: 2

pool:
  vmImage: $(imageName)

steps:
- task: Maven@4
  inputs:
    mavenPomFile: "pom.xml"
    mavenOptions: "-Xmx3072m"
    javaHomeOption: "JDKVersion"
    jdkVersionOption: $(jdkVersion)
    jdkArchitectureOption: "x64"
    publishJUnitResults: true
    testResultsFiles: "**/TEST-*.xml"
    goals: "package"
```