Fundamentals of Software Development Lifecycle/ Application Lifecycle Management

2022 - 2023

Overview

Software Development Life Cycle (SDLC) is a process used by the software industry to

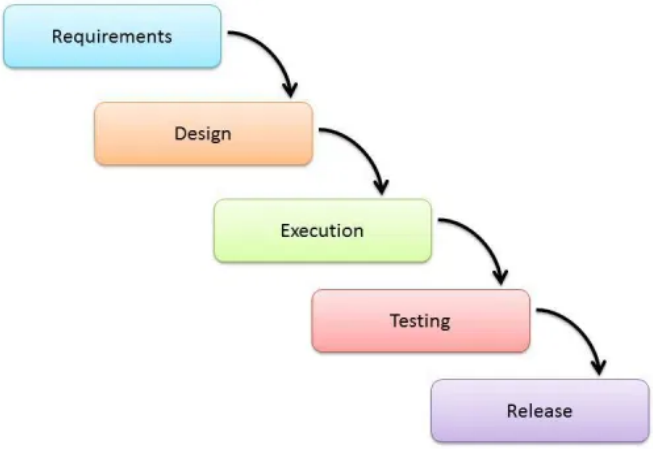
design, develop and test high quality softwares. The SDLC aims to produce high-quality software that meets or exceeds customer expectations, reaches completion within time and cost estimates.

Goals

1. The goal of the SDLC is to produce superior software that meets and exceeds all customer expectations and demands.
2. The SDLC defines and outlines a detailed plan with stages, or phases, that each encompass their own process and deliverables.

Modules

1. Fundamentals of Web Applications
2. Introduction to Software Development Life Cycle (Application Lifecycle Management)
3. Essential Technologies of SDLC
4. Execution of Software Development Life Cycle (Application Lifecycle Management)



Module -1

Fundamentals of Web Applications

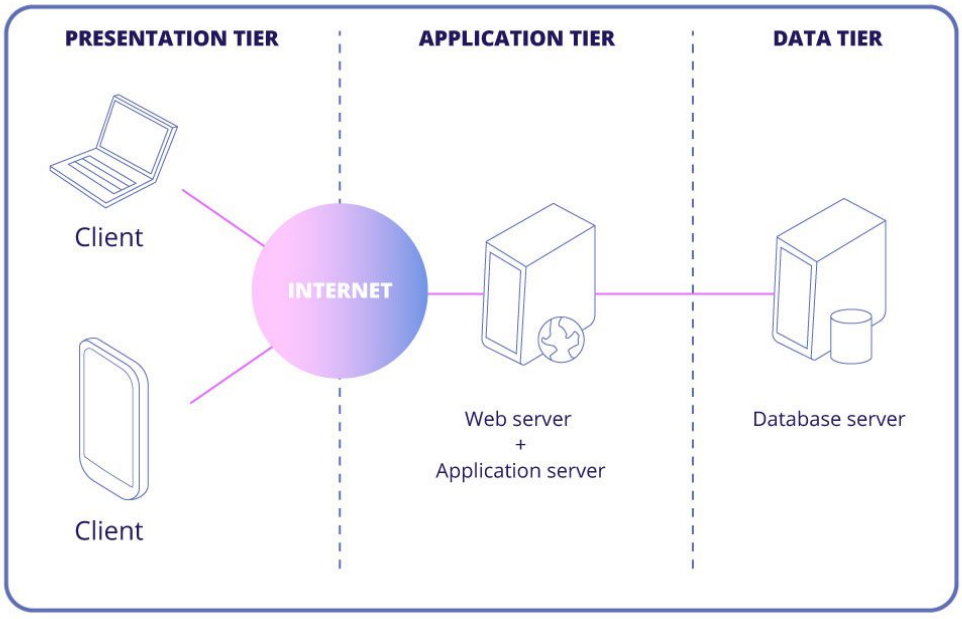
1. Web Application Architecture
2. Web Technologies
3. Web Technologies used in Projects

Web Application Architecture:

A web app architecture presents a layout with all the software components (such as databases, applications and middleware) and how they interact with each other. The modern day applications contain three-tiered architecture. Three-tier architecture, which separates applications into three logical computing tiers, is the predominant software architecture for traditional client-server applications. For decades three-tier architecture was the prevailing architecture for client-server applications.

The three tiers of applications in this architecture are:

- the presentation tier, or user interface;
- the application tier, where data is processed;
- the data tier, where the data associated with the application is stored and managed.



Web Technologies

Presentation tier:

The presentation tier is the user interface and communication layer of the application, where the end user interacts with the application.

- Its main purpose is to display information to and collect information from the user. This top-level tier can run on a web browser, as a desktop application, or a graphical user interface (GUI), for example.
- Web presentation tiers are usually developed using Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS) and JavaScript.
- HTML tells a browser how to display the content of web pages, while CSS styles that content.

Application tier:

The application tier, also known as the logic tier or middle tier, is the heart of the application. In this tier, information collected in the presentation tier is processed - sometimes against other information in the data tier - using business logic, a specific set of business rules. The application tier can also add, delete or modify data in the data tier. The application tier is typically developed using React JS, Angular JS, Python, Java, (Salesforce), Express JS, Node JS and communicates with the data tier using API calls.

Data tier:

The data tier, sometimes called database tier, data access tier or back-end, is where the information processed by the application is stored and managed. This can be a relational database management system such as PostgreSQL, MySQL, MariaDB, Oracle, DB2, Informix or Microsoft SQL Server, or in a NoSQL Database server such as MongoDB, Cassandra, Couch DB.

In a three-tier application, all communication goes through the application tier. The presentation tier and the data tier cannot communicate directly with one another.

Benefits:

- Faster development
- Improved scalability
- Improved reliability
- Improved security

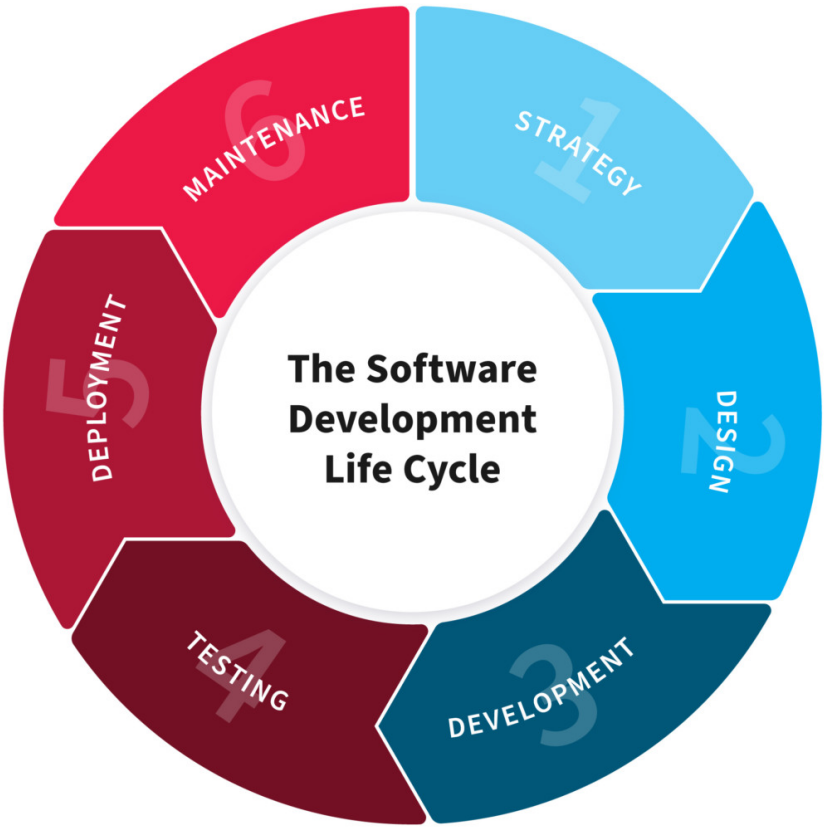
Module - 2

Introduction to Software Development Life Cycle (Application Lifecycle Management)

- What is SDLC?
- SDLC Methodologies
- Waterfall Methodology
- Agile Methodology
- Roles involved in the process of Software Development Life Cycle/ Application Lifecycle Management
- Scrum Framework

What is the Software Development Life Cycle?

A software development life cycle (SDLC) is a methodology followed to create high-quality software. By adhering to a standard set of tools, processes, and duties, a software development team can build, design, and develop products that meet or exceed their clients' expectations.



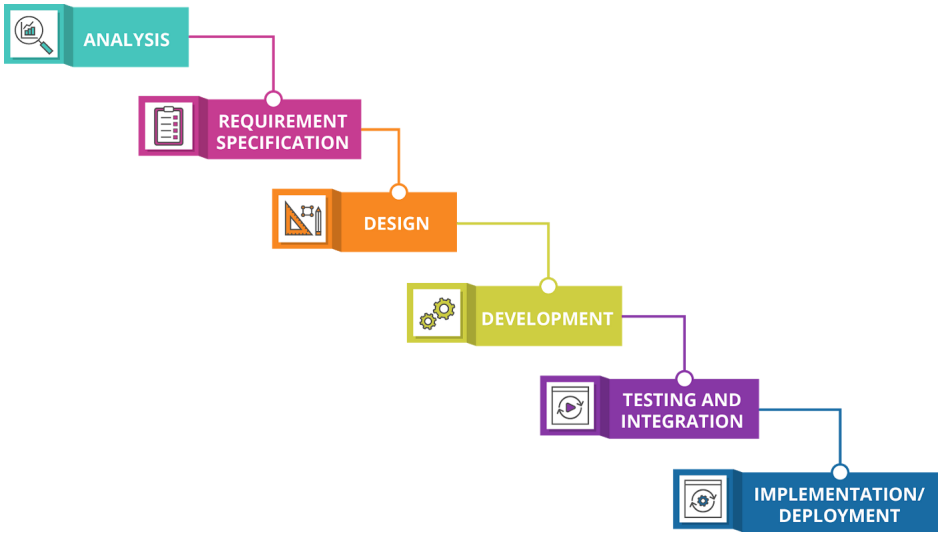
SDLC Methodologies:

- Waterfall
- Lean
- Iterative
- Spiral
- Agile

Each of these approaches varies in some ways from the others, but all have a common purpose: to help teams deliver high-quality software as quickly and cost-effectively as possible. Out of all these, the two most common methodologies are Waterfall and Agile.

Waterfall Methodology:

The Waterfall methodology—is a sequential development process that flows like a waterfall through all phases of a project (analysis, design, development, and testing, for example), with each phase completely wrapping up before the next phase begins.



Disadvantages of Waterfall Methodology:

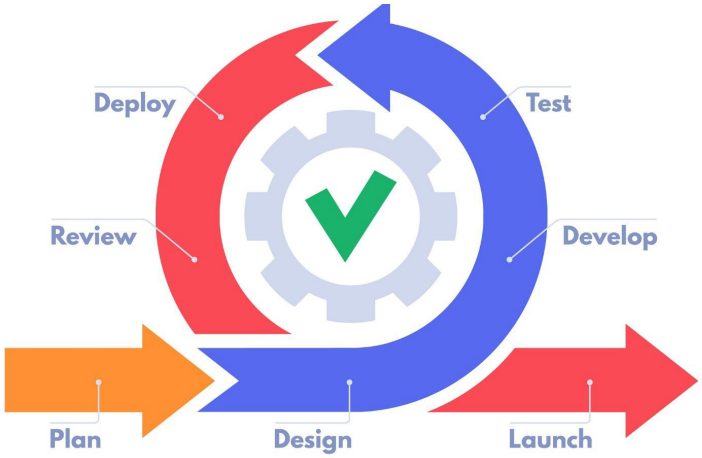
The Waterfall methodology's insistence on upfront project planning and commitment to a certain defined progress means that it is less flexible, other reasons the Waterfall methodology may not work include:

- Projects can take longer to deliver.
- Clients are not involved in the design and implementation stages.
- Deadline creep—when one phase in the process is delayed, all the other phases are delayed.

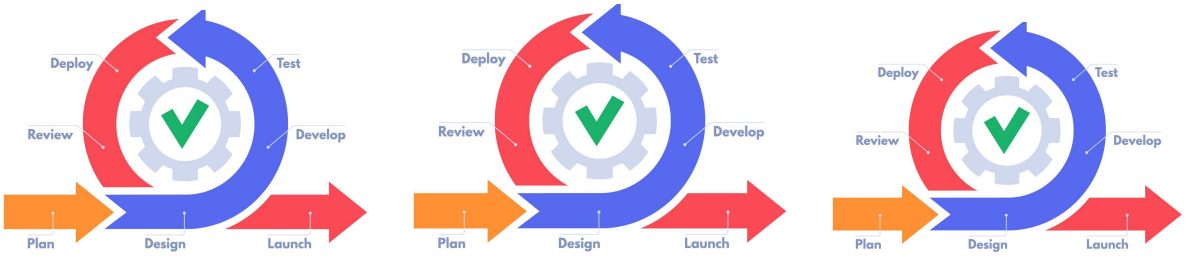
Agile Methodology:

Agile is an iterative approach to project management and software development that helps teams deliver value to their customers faster and with fewer headaches. Instead of betting everything on a "big bang" launch, an agile team delivers work in small, but consumable, increments. Requirements, plans, and results are evaluated continuously so teams have a natural mechanism for responding to change quickly.

Agile Single Iteration:



Agile Development Cycle:



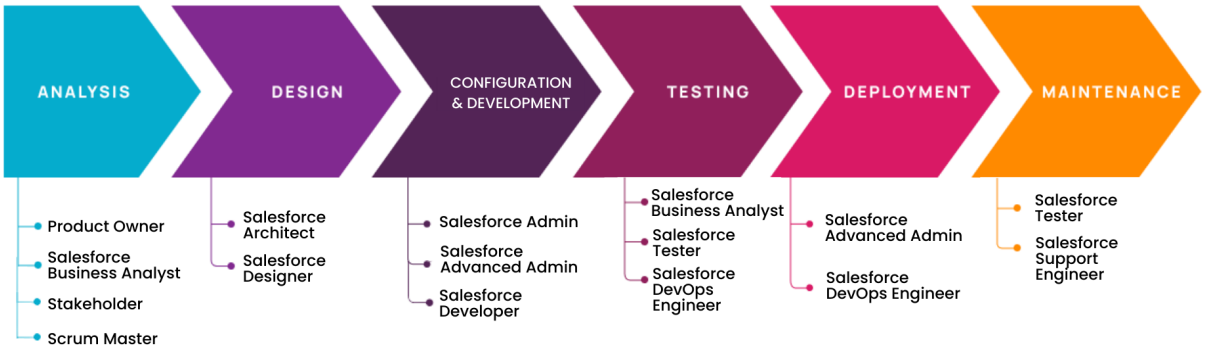
Iteration 1 **Iteration 2** **Iteration 3**

Advantages of Agile Methodology:

1. Satisfy the client via early, continuous delivery of valuable software.
2. Adapt to changing requirements, even late in development.
3. Use a shorter timescale (between a couple of weeks to a couple of months) for delivering working software.
4. Developers and business people should work together on a daily basis.
5. Regularly reflect on how your team can be more effective and adjust accordingly.
6. Motivate individuals by giving them an ideal environment, support, and trust.
7. Convey information via face-to-face communication.

8. Use working software as the main measurement of progress.
9. Maintain a constant pace and promote sustainable development.
10. Pay continuous attention to good design and technical excellence.
11. Keep things as simple as possible and allow the teams to self-organize.

The Process of Software Development Life Cycle/ Application Lifecycle Management (Roles):



Product Owner: The Product Owner (PO) is a member of the Agile Team responsible for defining Stories and prioritizing the Team Backlog to streamline the execution of program priorities while maintaining the conceptual and technical integrity of the Features or components for the team.

Salesforce Business Analyst: A Salesforce business analyst is a person who processes, interprets and documents business processes, products, services and software through analysis of data. The role of a business analyst is to ensure business efficiency increases through their knowledge of both IT and business function.

Stakeholder: Stakeholder(s) are people and organization units who frequently interface with the product owner, scrum master and scrum team to provide them with inputs and facilitate creation of the project's products and services, influencing the project throughout the project's development.

Scrum Master: A professional who ensures the scrum framework is followed. Scrum has a clearly defined set of roles and rituals that should be followed and the scrum master works with each member of the scrum team to guide and coach the team through the scrum framework.

Salesforce Architect: A Salesforce Architect helps design and deliver solutions for enterprise-grade customers using Salesforce products. The primary responsibility is to recommend the best solution for a given set of requirements and articulate the trade-offs involved in choosing one solution over another.
Salesforce Designer: A Salesforce Designer is passionate about (and certified in) creating human-centered design (HCD) experiences on the Salesforce Platform. To do this, they use

Relationship Design, a creative practice that drives social and business value by building strong relationships. People are at the heart of everything they do.

Salesforce Admin: Salesforce Administrators work with stakeholders to define system requirements and customize the platform. To put it simply, they enable users to get the most out of Salesforce technology. A Salesforce Admin best understands how to make the platform work for their company's goals.

Salesforce Advanced Admin: Salesforce Advanced Admins access breadth of applications, the features and functions available to an end user, and the advanced configuration, management, and application extending options available to an Administrator across the Sales Cloud, Service Cloud, and Salesforce Chatter applications

Salesforce Developer: A Salesforce developer is a special type of computer programmer who writes software on the Salesforce CRM platform or another piece of Salesforce cloud technology. This ranges from creating websites to developing apps. A Salesforce developer works with computer code like JavaScript and HTML on a daily basis.

Salesforce Tester: The major function of a Salesforce Tester includes services like Functional Testing, Unit Testing, Manual Testing, Automated Testing, Load Testing, Deployment Testing, and Security Testing. Testers having experience with AssureClick, Selenium, QTP etc. play a vital role in Salesforce Automated Testing

Salesforce DevOps Engineer: Salesforce DevOps Engineer responsible for implementing scalable solutions including continuous delivery, optimization, monitoring, release management and support the end to end release process.

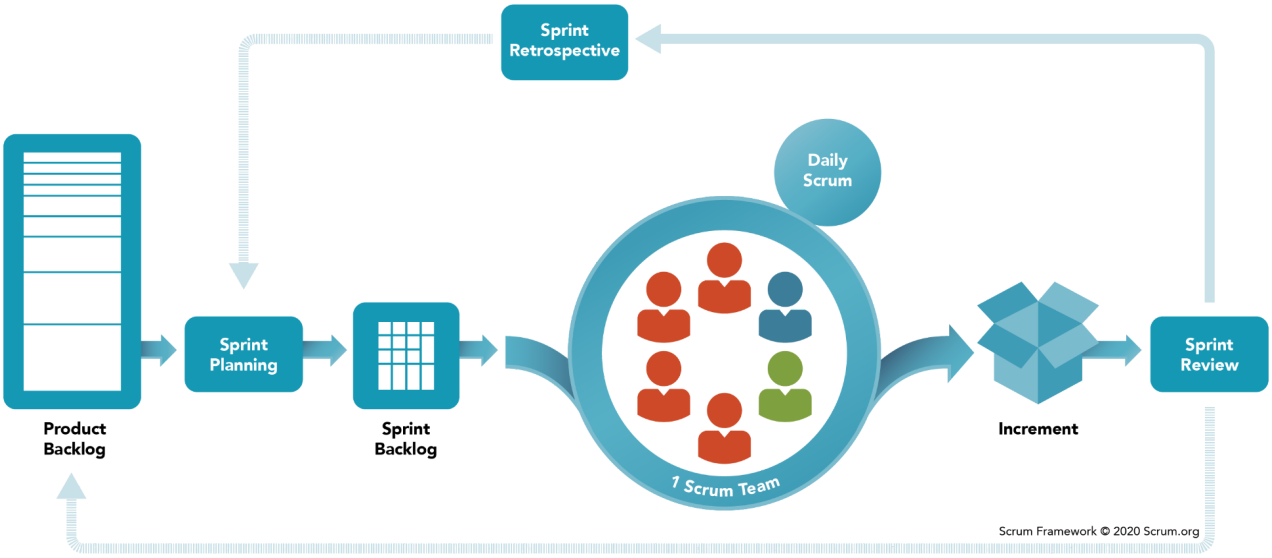
Salesforce Support Engineer: Assist third-party developers to troubleshoot their integration with salesforce.com APIs, Apex, Visualforce and implementation of other salesforce.com developer products. This will involve debugging, troubleshooting, and taking responsibility to see that the issue is fully resolved.

Scrum Framework:

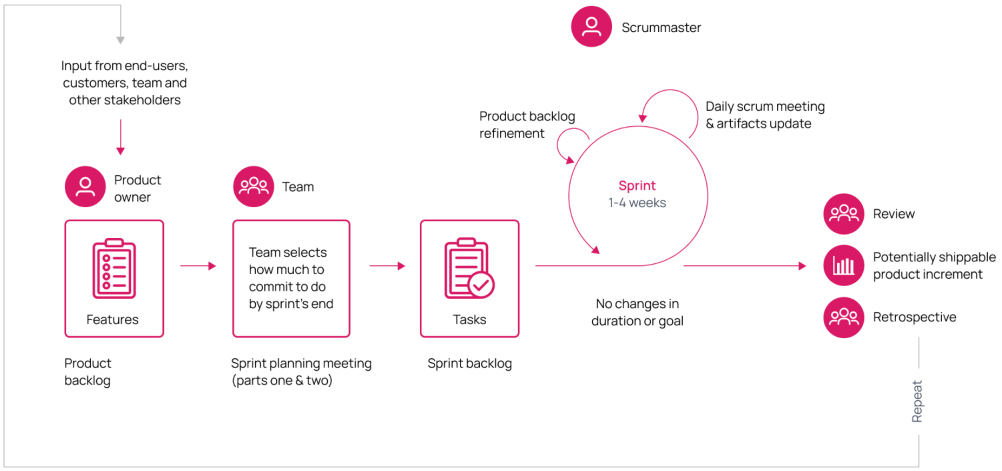
Scrum is a framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible. The Scrum Team consists of one Scrum Master, one Product Owner, and Developers.

In a nutshell, Scrum requires a Scrum Master to foster an environment where:

1. Product Owner orders the work for a complex problem into a Product Backlog.
2. The Scrum Team turns a selection of the work into an Increment of value during a Sprint.
3. The Scrum Team and its stakeholders inspect the results and adjust for the next Sprint.
4. Repeat



A Scrum Framework usually consists of 5 different phases:



Scrum Terminology:

Scrum Team:

A Scrum Team is expected to adapt the moment it learns anything new through Inspection. The Scrum Team commits to achieving its goals and to supporting each other. Scrum Team members respect each other to be capable, independent people, and are respected as such by the people with whom they work.

Scrum Master

Scrum Masters are true leaders who serve the Scrum Team and the larger

organization. The Scrum Master serves the Scrum Team in several ways, including coaching the team members in self-management and cross-functionality. The Scrum Master serves the Scrum Team in several ways, including helping the Scrum Team focus on creating high-value Increments that meet the Definition of the planning.

Sprint Planning:

The Scrum Master serves the Scrum Team in several ways, including ensuring that all Scrum events take place and are positive, productive, and kept within the timebox. This resulting plan is created by the collaborative work of the entire Scrum Team. The whole Scrum Team then collaborates to define a Sprint Goal that communicates why the Sprint is valuable to stakeholders.

Sprints:

The Sprint is a container for all other events. Sprints are the heartbeat of Scrum, where ideas are turned into value. [Sprints] are fixed length events of one month or less to create consistency. A new Sprint starts immediately after the conclusion of the previous Sprint. All the work necessary to achieve the Product Goal, including Sprint Planning, Daily Scrums, Sprint Review, and Sprint Retrospective, happen within Sprints.

Sprint Review:

The Sprint Review is a working session and the Scrum Team should avoid limiting it to a presentation. The Scrum Team discusses what went well during the Sprint, what problems it encountered, and how those problems were (or were not) solved.

Product Backlog:

The Product Backlog is the single source of work undertaken by the Scrum Team. 10 Product Backlog items that can be Done by the Scrum Team within one Sprint are deemed ready for selection in a Sprint Planning event.

Product Goal and Sprint Goal:

The Product Goal is the long-term objective for the Scrum Team. They must fulfill (or abandon) one objective before taking on the next. The Sprint Goal also creates coherence and focus, encouraging the Scrum Team to work together rather than on separate initiatives.

Daily Scrum:

The purpose of the Daily Scrum is to inspect progress toward the Sprint Goal and adapt the Sprint Backlog as necessary, adjusting the upcoming planned work. The Daily Scrum is a 15-minute event for the Developers of the Scrum Team. To reduce complexity, [the Daily Scrum] is held at the same time and place every working day of the Sprint.

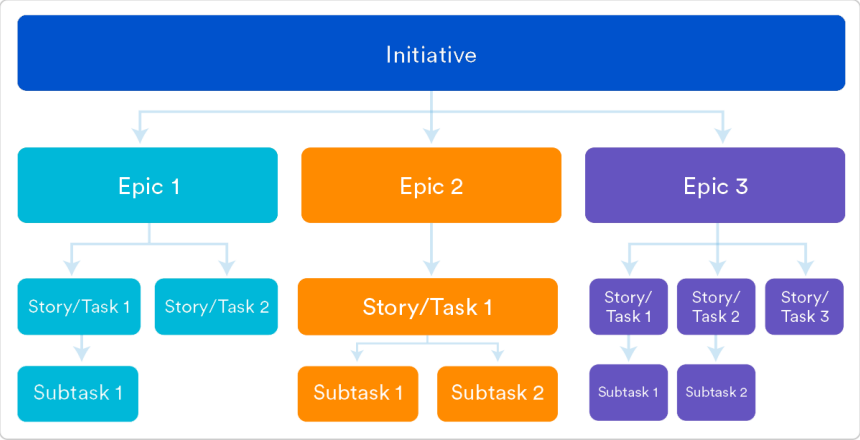
Spring Retrospective:

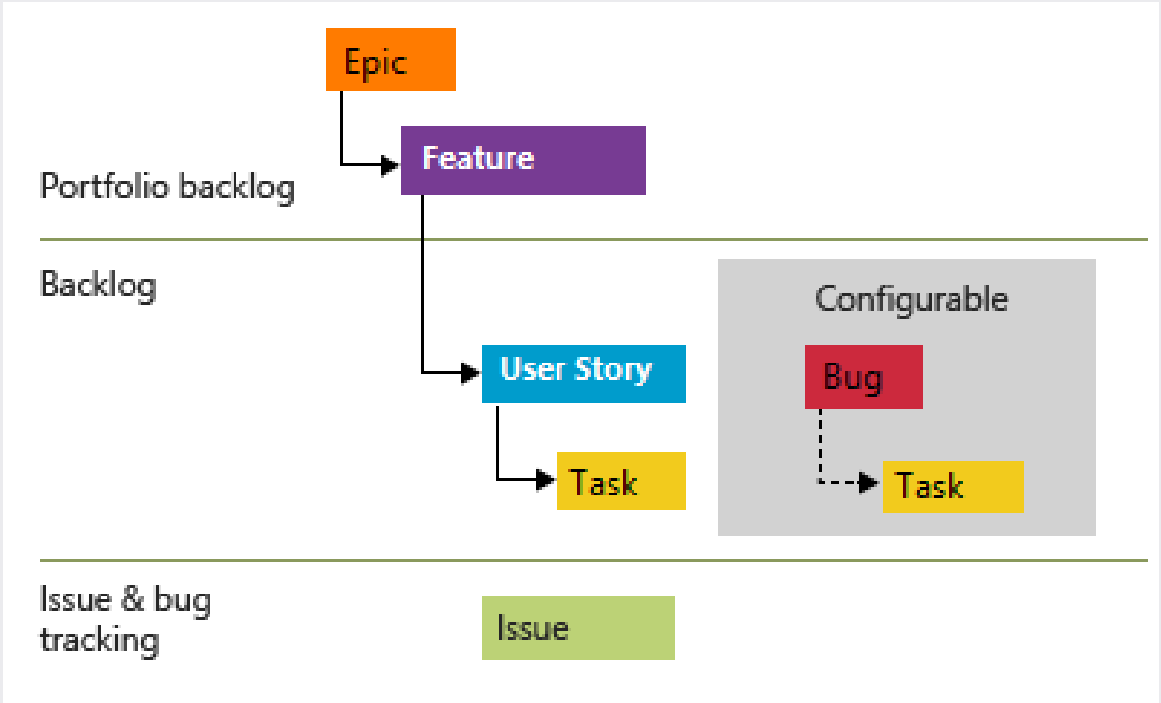
The purpose of the Sprint Retrospective is to plan ways to increase quality and effectiveness. The Scrum Team inspects how the last Sprint went with regards to individuals, interactions, processes, tools, and their Definition of Done.

Steps of a Scrum Workflow:

Various items are included in the backlog, like features, bugs and defects, information attainment, and technical work. Large items are transformed into "user stories" and "epics".

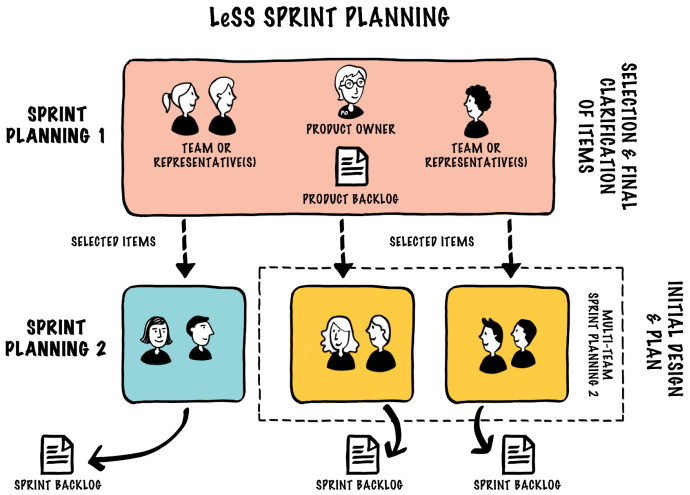
Epics – Large chunks of work that can be divided into stories





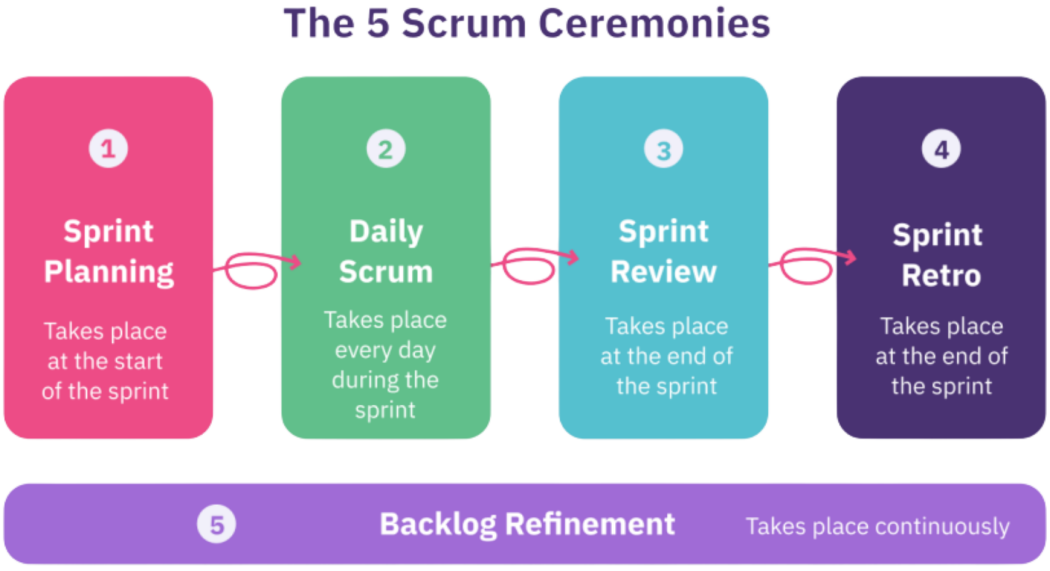
User Stories – Short requirements written from an end user's perspective

1. Epics and user stories can both be put into the product backlog, but only user stories are included in the sprint backlog.
2. Next comes sprint planning and creating the sprint backlog. The Scrum team selects the most important user stories and breaks them up into smaller tasks. User stories need to be made as small as possible, as the average Sprint only lasts 2 weeks.
3. After the Sprint is planned, it's time to get to work. Throughout the Sprint, daily Scrum meetings are held. These last for about 15 minutes and aim to gather the status of each team member.
4. Full life cycle testing is carried out, as each task is a working product. After testing, each Sprint is demonstrated to the customer.
5. Retrospective is next, in which the team discusses what went well, what can be improved, and the lessons learned during the Sprint. After that, the next Sprint is planned, and the cycle begins again.



Roles:

1. StakeHolder - CEO (Kona)
2. Product Owner - Mani
3. Business Analyst - Bala
4. Scrum Master - Sai
5. Salesforce Admin -2
6. Salesforce Advanced Admin - 1
7. Salesforce Developer - 3
8. Salesforce Tester - 1
9. Salesforce DevOps Engineer -1



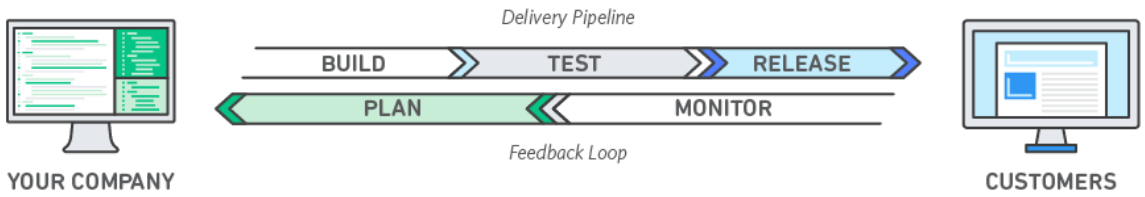
Module - 3

Essential Technologies of SDLC

- What is Salesforce DevOps?
- What is Cloud, SAAS, IAAS, PAAS?
- What is Salesforce Testing?

What is DevOps?

DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market.



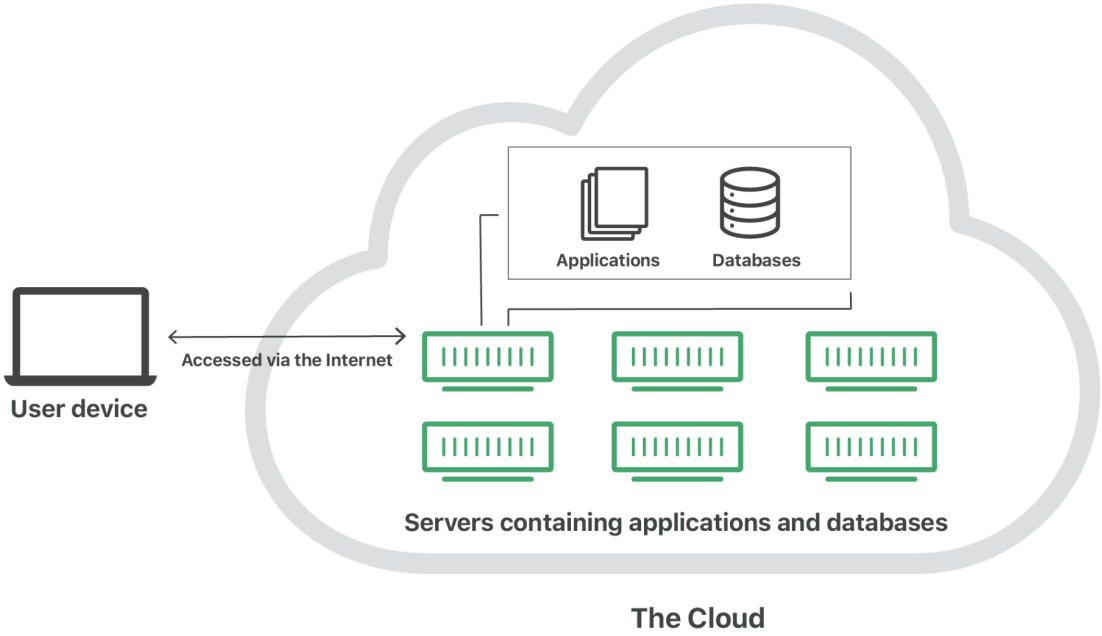
What does a DevOps Engineer do?

A DevOps engineer introduces processes, tools, and methodologies to balance needs throughout the software development life cycle, from coding and deployment, to maintenance and updates.

(Include specific content about Salesforce DevOps)

What is Cloud - SAAS, IAAS, PAAS?

"The cloud" refers to servers that are accessed over the Internet, and the software and databases that run on those servers. Cloud servers are located in data centers all over the world. By using cloud computing, users and companies do not have to manage physical servers themselves or run software applications on their own machines.



Main models of Cloud Computing:

Software-as-a-Service (SaaS): SaaS applications are hosted on cloud servers, and users access them over the Internet. SaaS is like renting a house: the landlord maintains the house, but the tenant mostly gets to use it as if they owned it.

Examples: Salesforce, MailChimp, and Slack.

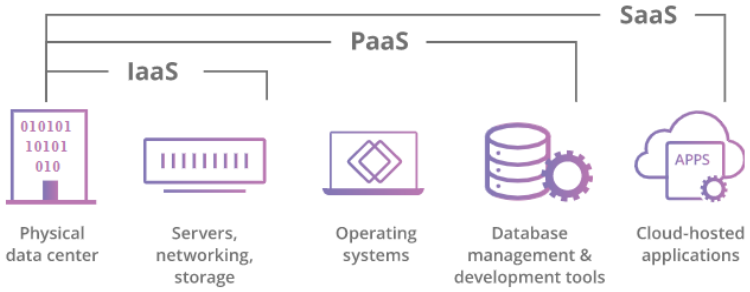
Platform-as-a-Service (PaaS): In this model, companies don't pay for hosted applications; instead they pay for the things they need to build their own applications. PaaS vendors offer everything necessary for building an application, including development tools, infrastructure, and operating systems, over the Internet. PaaS can be compared to renting all the tools and equipment necessary for building a house, instead of renting the house itself.

Examples: Heroku and Microsoft Azure.

Infrastructure-as-a-Service (IaaS): In this model, a company rents the servers and storage they need from a cloud provider. They then use that cloud infrastructure to build

their applications. IaaS is like a company leasing a plot of land on which they can build whatever they want — but they need to provide their own building equipment and materials.

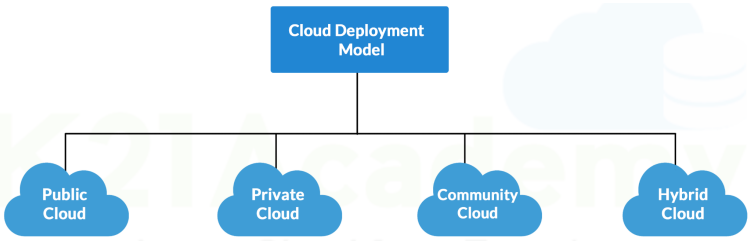
Examples: DigitalOcean, Google Compute Engine, and OpenStack.



The most common cloud deployments are:

- **Private cloud**: A private cloud is a server, data center, or distributed network wholly dedicated to one organization.
- **Public cloud:** A public cloud is a service run by an external vendor that may include servers in one or multiple data centers. Unlike a private cloud, public clouds are shared by multiple organizations. Using virtual machines, individual servers may be shared by different companies, a situation that is called "multi tenancy" because multiple tenants are renting server space within the same server.
- **Hybrid cloud:** Hybrid cloud deployments combine public and private clouds, and may even include on-premises legacy servers. An organization may use their private cloud for some services and their public cloud for others, or they may use the public cloud as backup for their private cloud.

- **Multi-cloud:** Multi-cloud is a type of cloud deployment that involves using multiple public clouds. In other words, an organization with a multi-cloud deployment rents virtual servers and services from several external vendors — to continue the analogy used above, this is like leasing several adjacent plots of land from different landlords.



What is Testing?

Software testing is the process of evaluating and verifying that a software product or application does what it is supposed to do. The benefits of testing include preventing bugs, reducing development costs and improving performance.

Three types of Testing include:

- Manual testing
- Automation testing
- Automation with Selenium testing

Manual testing:

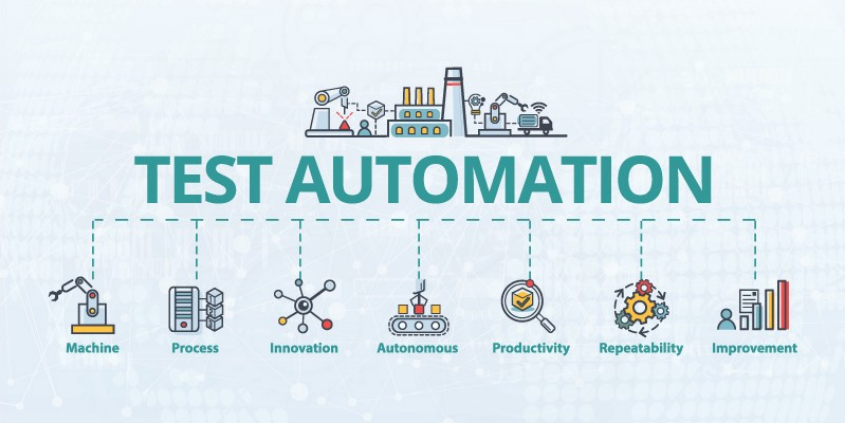
Manual testing is a software testing process in which test cases are executed manually without using any automated tool. All test cases executed by the tester manually according to the end user's perspective. It ensures whether the application is working, as mentioned in the requirement document or not. Test cases are planned and implemented to complete almost 100 percent of the software application. Test case reports are also generated manually.

Automation testing:

Automation Testing is a software testing technique that performs using special automated testing software tools to execute a test case suite. On the contrary, Manual Testing is performed by a human sitting in front of a computer carefully executing the test steps.

Automation with Selenium testing:

Selenium is a free (open-source) automated testing framework used to validate web applications across different browsers and platforms. You can use multiple programming languages like Java, C#, Python etc to create Selenium Test Scripts. Testing done using the Selenium testing tool is usually referred to as Selenium Testing.



Module - 4

Execution of Software Development Life Cycle (Application Lifecycle Management)

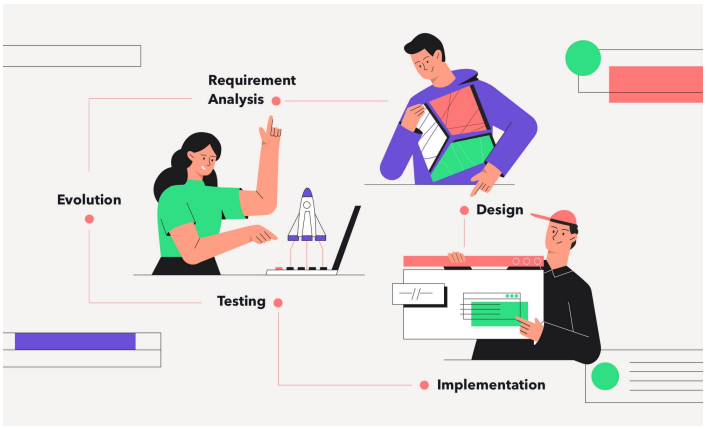
- Analysis: Azure Boards, Jira.
- Introduction to Azure DevOps
- Code Management: Azure Repos, Git and GitHub.
- Testing Release: Test Plans
- Salesforce Deployment & Maintenance: CI/CD Pipelines

#1 Planning and Analysis:

Requirements gathering is the first most crucial phase of the software development life cycle. It decides how your software will look and perform at the end. To build quality software, it is imperative to spend ample time in this phase. During this step, the project managers or business analysts meet with the customer and ask for the software specifications. Once you are done with the requirements gathering, you need to draw the project timeline and scope.

Steps of Planning and Analysis:

1. Defining business problem and scope
2. Producing detailed project schedule
3. Confirming project feasibility
4. Staffing the project with accurate resources
5. Planning the scrum based on the requirements
6. Launching the project



Tools required for Planning and Analysis:

The two major tools that are used for Planning and Analysis process in SDLC are

1. Azure Boards
2. Jira

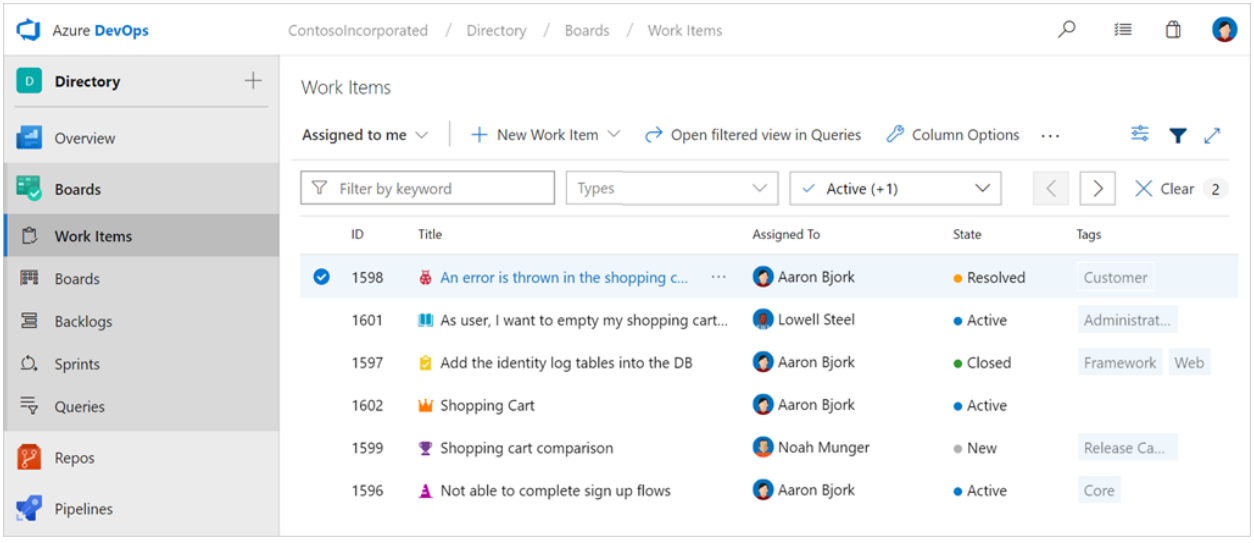
Azure Boards:

Azure Boards is a service for managing the work for your software projects. Teams need tools that flex and grow. Azure Boards does just that, bringing you a rich set of capabilities including native support for Scrum and Kanban, customizable dashboards, and integrated reporting.

Work items

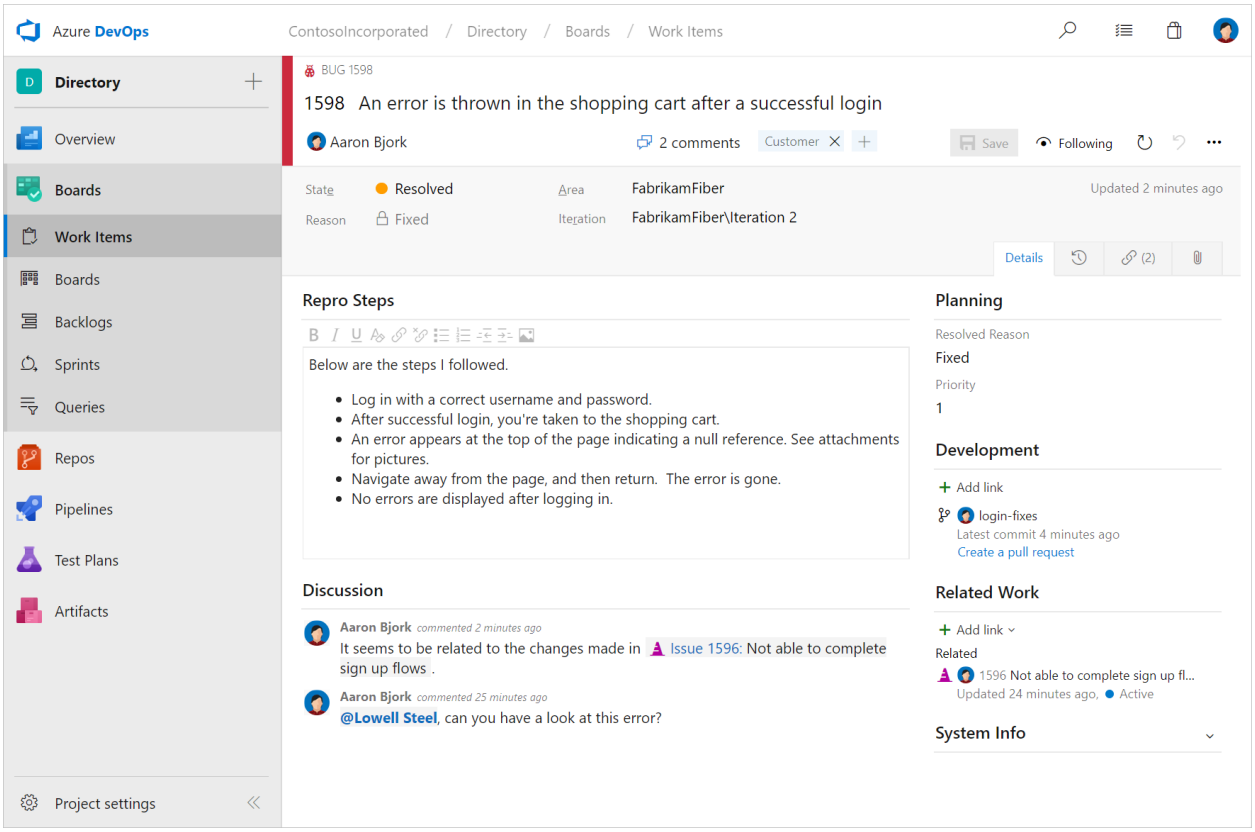
All work in Azure Boards is tracked through an artifact called a work item. Work items are where you and your team describe the details of what's needed. Each work item uses a state model to track and communicate progress.

For example, a common state model might be: New > Active > Closed. As work progresses, items are updated accordingly, allowing everyone who works on the project to have a complete picture of where things are at. Below is a picture of the work items hub in Azure Boards. This page is the home for all work items and provides quick filters to allow you to find the items you need.



Opening a work item brings you to a much richer view, including the history of all changes, any related discussion, and links to development artifacts including branches, pull requests,

commits, and builds. Work items are customizable, supporting the ability to add new fields, create rules, and modify aspects of the layout.

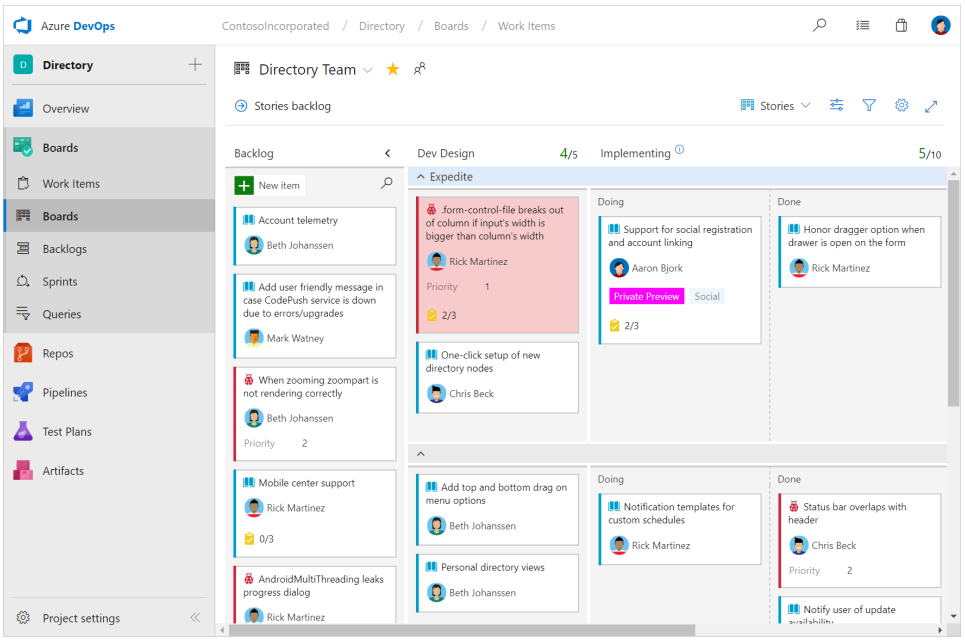


Boards, Backlogs, and Sprints

Azure Boards provides a variety of choices for planning and managing work. Let's look at a few of the core experiences.

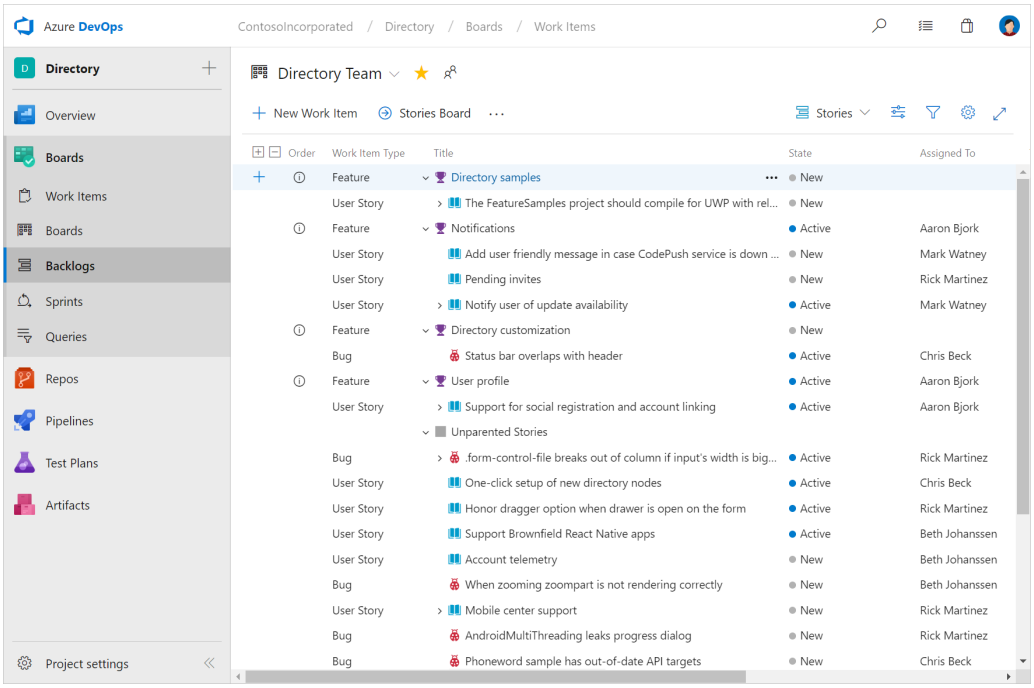
Boards

Each project comes with a pre-configured Kanban board perfect for managing the flow of your work. Boards are highly customizable allowing you to add the columns you need for each team and project. Boards support swim lanes, card customization, conditional formatting, filtering, and even WIP limits.



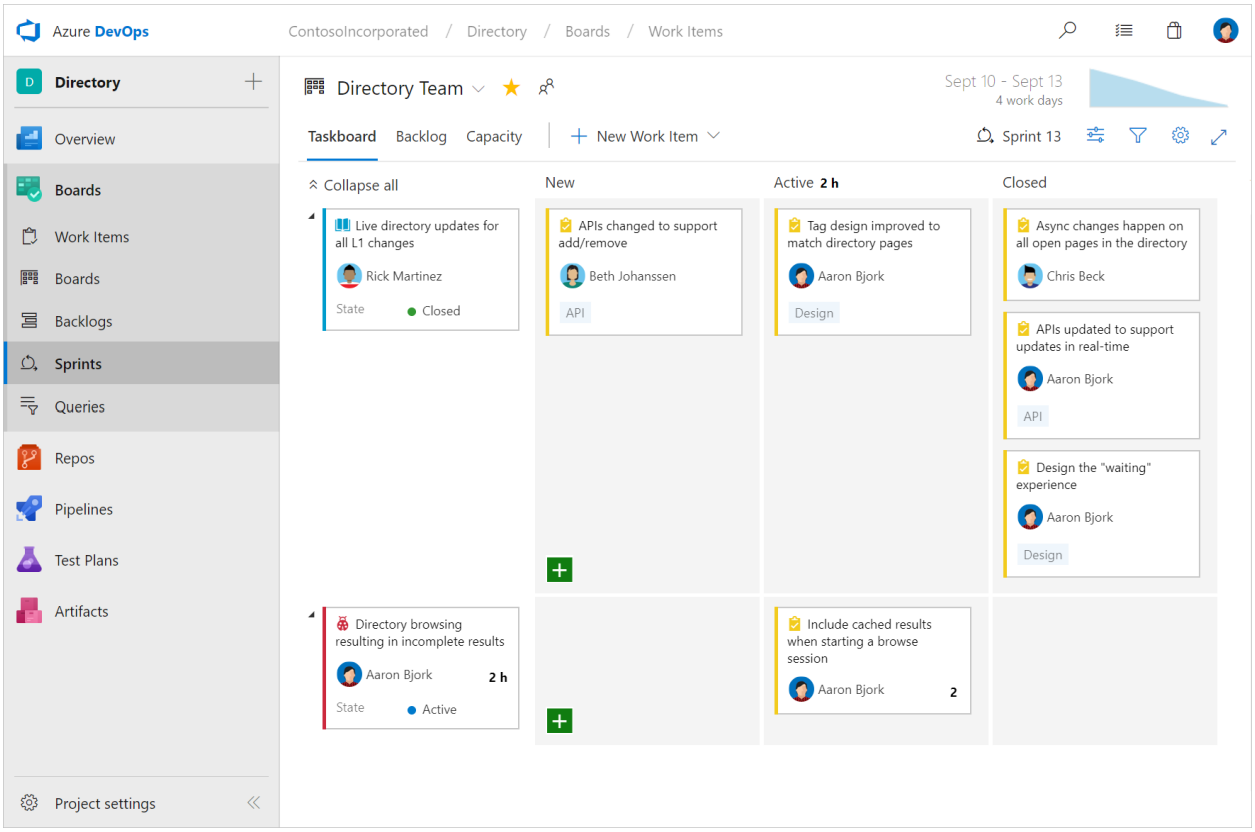
Backlogs

Backlogs help you keep things in order of priority, and to understand the relationships between your work. Drag and drop items to adjust the order, or quickly assign work to an upcoming sprint.



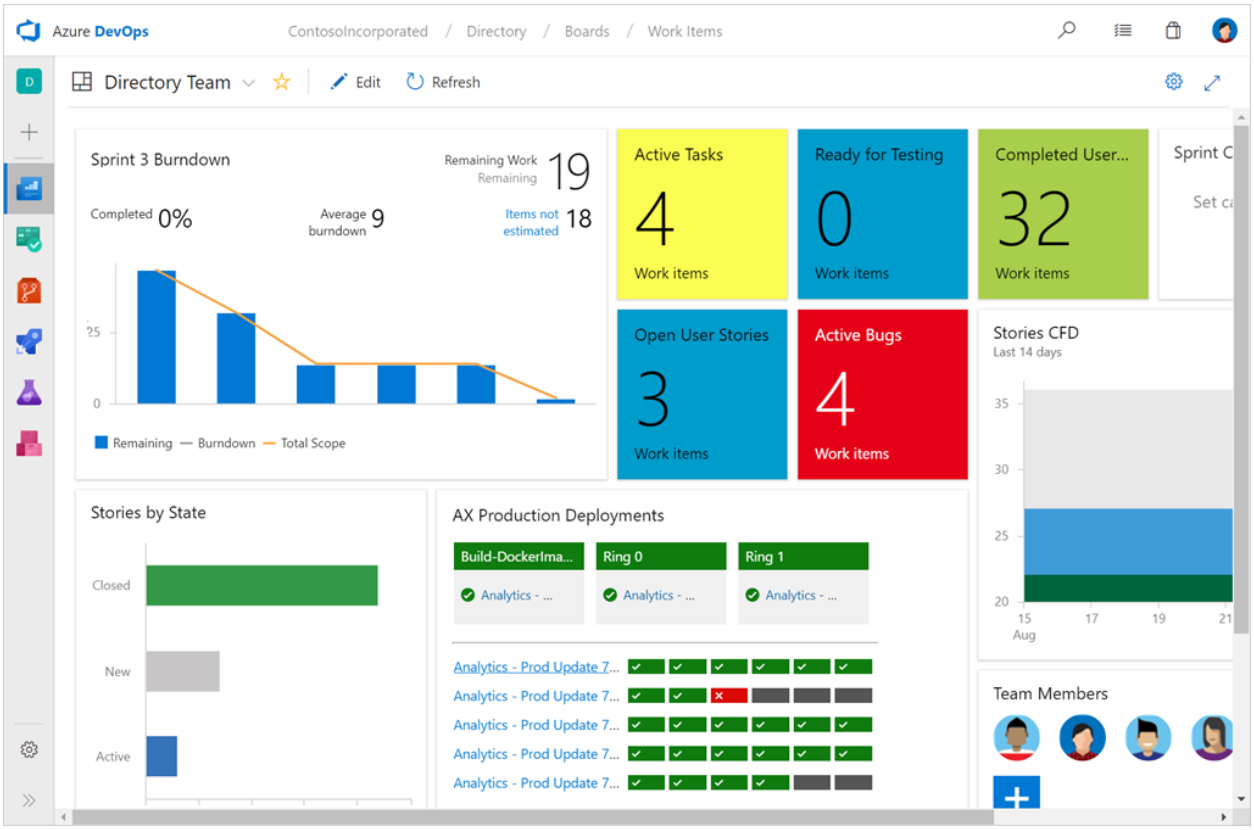
Sprints

Finally, sprints give you the ability to create increments of work for your team to accomplish together. Each sprint comes equipped with a backlog, taskboard, burndown chart, and capacity planning view to help you and your team deliver your work on time.



Dashboards

In any project, it's critical that you have a clear view of what's happening. Azure Boards comes complete with a rich canvas for creating dashboards. Add widgets as needed to track progress and direction.



Queries

And finally, one of the most powerful features in Azure Boards is the query engine. Queries let you tailor exactly what you're tracking, creating easy to monitor KPIs. It's simple to create new queries and pin them to dashboards for quick monitoring and status.

Jira:

Getting started with Jira

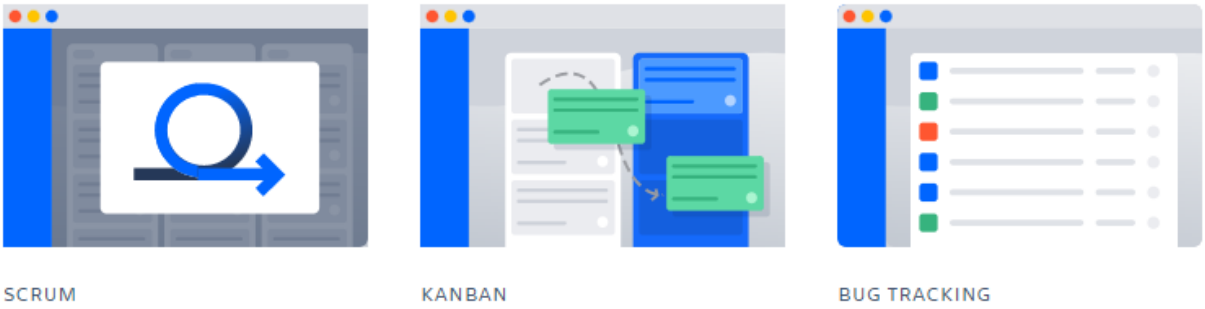
Step 1 - Create a project

1. In the top-left corner, click the Jira home icon .
2. In the top-right corner, select Create project.

Step 2 - Pick a template

The Jira template library houses dozens of templates across a variety of different categories, and is designed to get your team started quickly and successfully. You can

choose a template from all the Jira products you own (Jira Software, Jira Service Management, and Jira Work Management). Today, Jira Software offers three templates:



1. **Scrum:** For agile teams that work from a backlog, plan and estimate their work in sprints, and deliver work on a regular schedule.
2. **Kanban**: For agile teams that monitor work in a continuous flow (rather than in sprints), with a focus on managing in-progress work. (Includes the option of a kanban backlog.)
3. **Bug tracking**: For teams that don't need boards and prefer to manage development tasks and bugs in a list view

Project Types

For the scrum and kanban templates only, you will also be prompted to choose a project type.

- Team-managed projects are suited for independent teams who want to control their own working processes and practices in a self-contained space.
- Company-managed projects are set up and maintained by Jira admins. This project type is designed for teams who want to standardize a way of working across many teams, such as sharing a workflow.

The fundamental difference between the two project types is how they are administered, and whether that occurs at the team level or at a company/Jira admin level.

Step 3 - Set up your columns

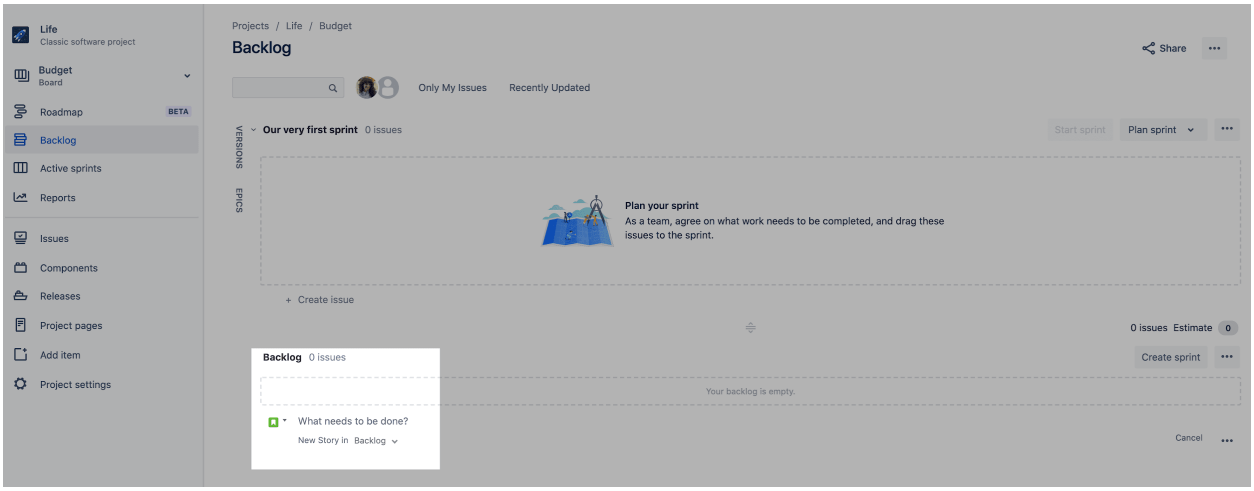
Setting up columns can be done in two ways:

- Company managed
- Team managed

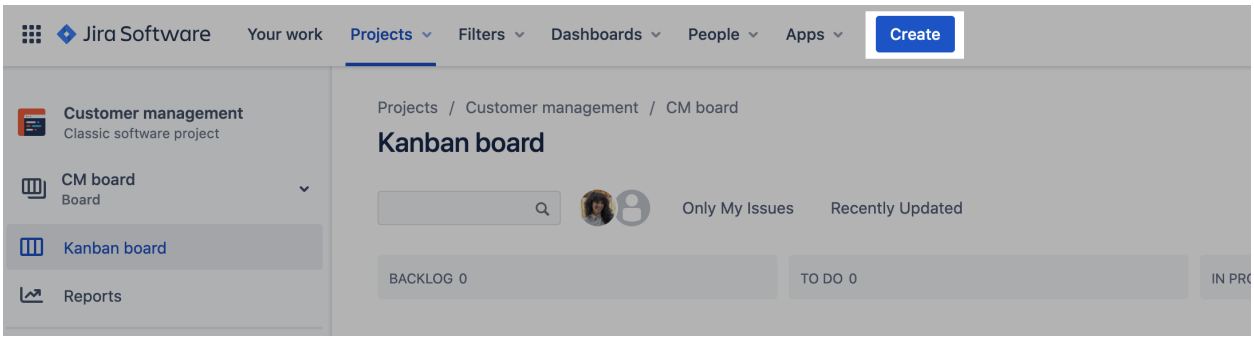
Step 4 - Create an issue

Scrum and kanban are two frameworks for agile project management. In Jira Software, scrum vs kanban projects have different features to help teams that are using either framework.

For Scrum teams: Select Backlog in the project menu on the left and then hit + Create issue to start adding work to your team's backlog.



For Kanban teams: Navigate to your Kanban board and select Create in the global menu. Your issue will appear in the Backlog column.



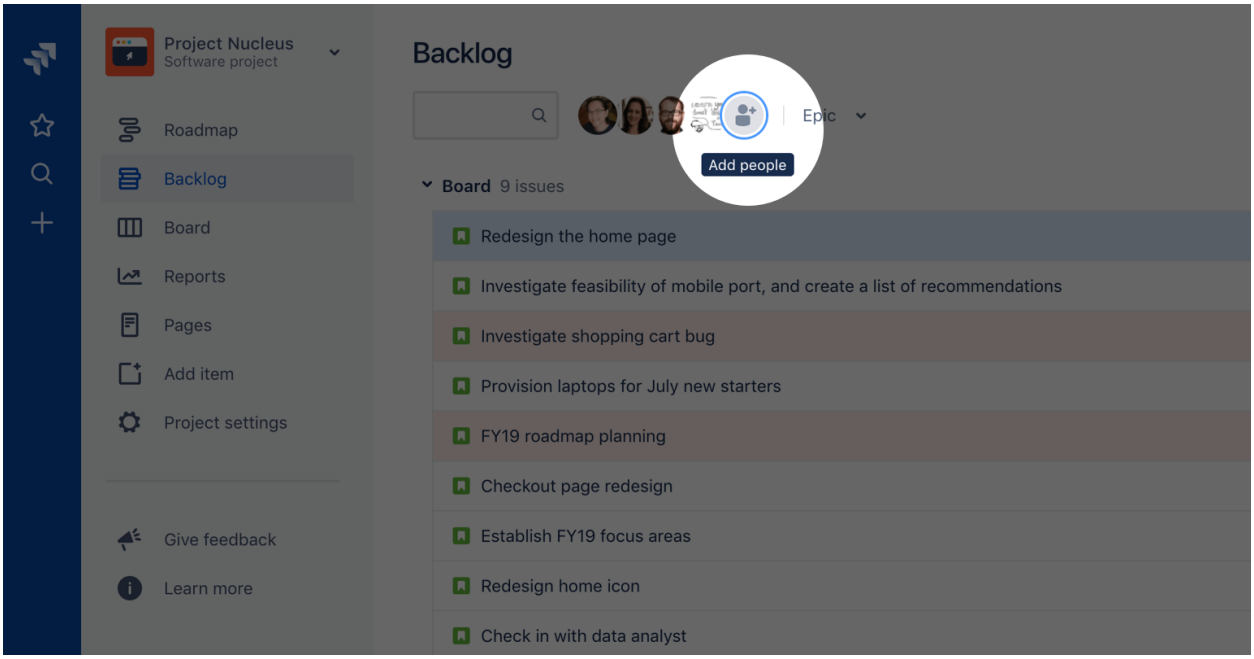
Step 5 - Invite your team

It's time to get the party started! Once you have enough work represented on your board, start inviting team members.

In the project menu on the left, select Project settings.

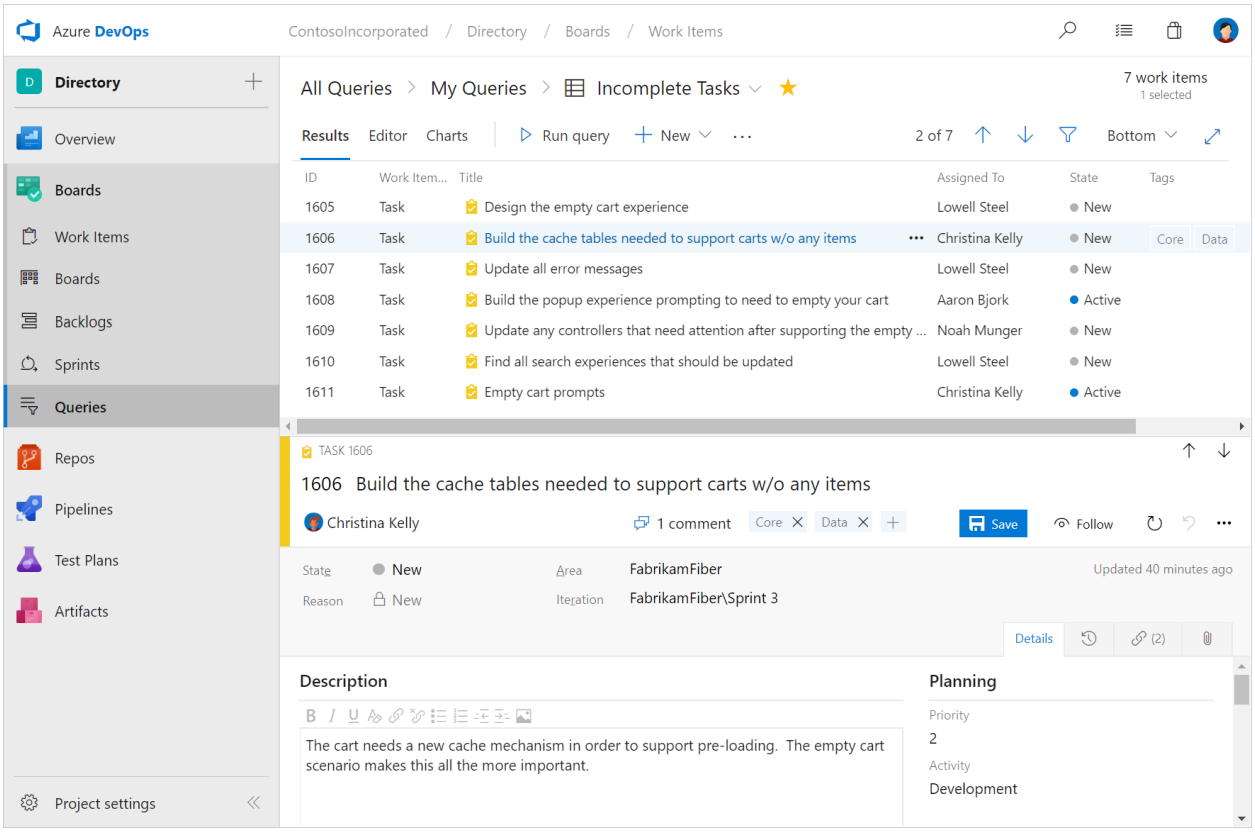
1. Select People.

2. In the top-right corner, select Add people.
3. Search for your team member's email address, and select Add.

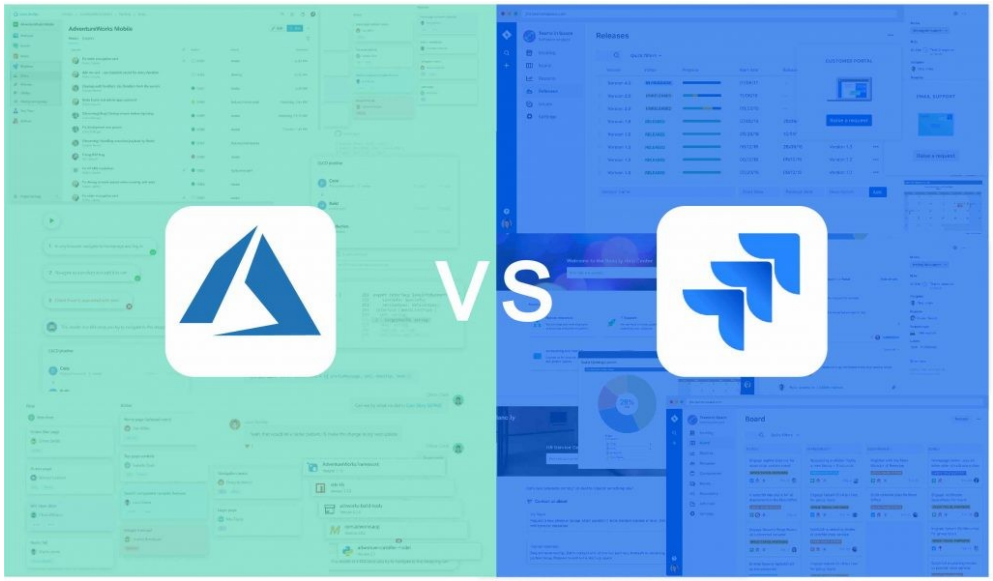


Step 6 - Move work forward

Now that your team is in your Jira site, you're ready to collaborate and track work together. If you're in a scrum project, you'll need to create and start a sprint to begin tracking work. If you're in a kanban project, you can start tracking work on the board. To track work items, move an issue from one column to another as it progresses through your team's workflow.

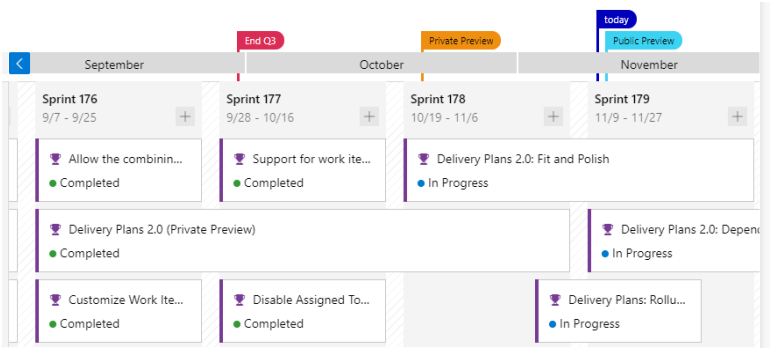


Azure Boards vs Jira



Jira vs. Azure DevOps – Aligning across the Enterprise

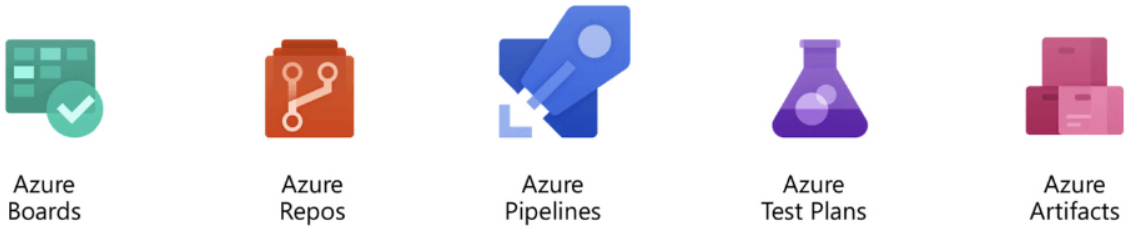
1. ***Spend less, get more:*** Clients have preferences and generally those preferences are due to familiarity. The familiarity factor is important when establishing a new partnership. The partner and client need to respect one another's tools when planning a project, so all parties can communicate and start work immediately. Jira has been in the IT and development community for many years and is very familiar with the IT teams we encounter. It is easy to migrate from Jira to DevOps – let us look at some of the reasons this is critical in an Enterprise project.
2. ***Agile:*** Both support the ability to set up and manage a project via basic, agile, scrum, and CMMI with the ability to switch between.
3. ***Enterprise Customization***: Both systems allow for customizations at the project level, Azure Boards allows the customization of a process that can be used across many projects.
4. ***Kanban:*** Both systems allow configurable Kanban boards enabling visualization of work items and current status.
5. ***Backlogs:*** Jira supports 3 backlog levels while Azure Boards support 5. Jira does not support hierarchical drill down and drill up views.
6. ***Sprints***: Supported by both Jira and Azure.
7. ***Roadmaps and Portfolio Planning:*** Jira has a separate product, Jira Align, available at a separate cost. Azure Boards offers Delivery Plans and Portfolio Plans extensions at no cost on the marketplace.
8. ***Search:*** Supported by both Jira and Azure.
9. ***Integration with Testing:*** Azure Boards supports critical integrations between work items and test plans.
10. ***Multiple Project queries:*** Azure Boards support queries across the enterprise by enabling cross-project queries which can be displayed in Dashboards.
11. ***Reporting and Analytics:*** Azure Boards support complex reporting and analytics with dashboard widgets, and Power BI reporting capabilities. Jira does not support reporting and analytics without costly add-ons from its marketplace.
12. ***GitHub Integration:*** Supported by both Jira and Azure.
13. ***Dashboard Support:*** Azure DevOps has a breadth of life cycle widgets for build, release and monitoring. Jira does not.
14. ***Enterprise Scale:*** Jira Cloud maxes out at 2000 users.



Introduction to Azure DevOps

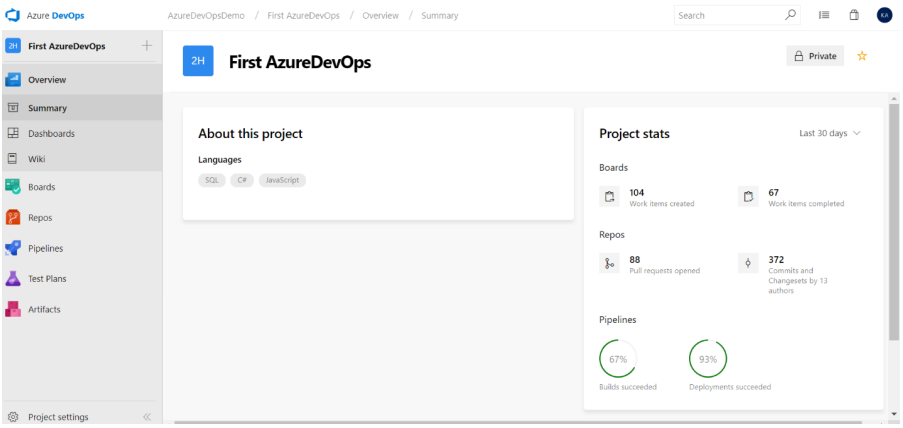
Azure DevOps, a modern DevOps tool of planning, developing, testing and deploying modern apps with optimized release cycle for quality delivery of applications. Azure DevOps provides a tool which can help you to track software building progress and help you to make decisions to deliver great software to end users. Azure DevOps services are not dependent on cloud or platform. Azure DevOps includes the following services :

Azure DevOps includes Git repositories as source control, build and release management tools, work planning and tracking tools, testing tools and support services like Slack, Trello and Azure services.



Azure Pipelines (Build and Release)

Azure Pipeline is a cloud-hosted pipeline for fast CI/CD that works with any language, platform, and cloud. By connecting to any source control like GitHub, this service can release changes continuously to any cloud. YAML files are very useful in writing build and release definitions. Azure Pipelines has components like build, release, library, task groups, deployment groups. Azure Pipelines has advanced workflows with native container support and features which allow monitoring CI/CD stages.



Azure Boards (Work)

Azure Boards helps to plan, track, and discuss work across the team. Azure Boards is a powerful agile tool for managing Kanban board, reporting, and product backlog.

1. Azure boards have components like work items, backlogs, Boards, queries, sprints details.
2. Work items can be bug, epic, issue, task or features. This service is sprint ready and built for insights to improve productivity.
3. We can manage user authentication and authorization, team, project, and organization-level settings.
4. Azure Boards helps you to write queries to retrieve specific work items from the system.

Azure Artifacts (Packages)

Azure Artifact service manages the dependencies used in source code. Azure Artifacts can host and share packages (like NPM, Nuget, Maven) feeds from public and private sources.

1. These artifacts simplify the job building process.
2. These stored artifacts are easy to integrate with Azure Pipelines.
3. Azure Artifacts are managed packages hosted on cloud and indexed.

Azure Repos

Azure Reops service includes unlimited cloud-hosted private Git repository for your project. This is standard Git service and works as distributed source controls.

1. Azure Repos supports all Git clients and all IDEs, all editors.
2. You may do an effective Git code review, and can raise pull requests.
3. Azure Repos supports branching strategy, so that you can merge the code after successful build and passing all the test cases to maintain high code quality.
4. Access to the repositories are managed by Azure AD, hence source code access management is fast and easy.

Azure Test Plans

Azure test plan service helps to do automated and manual testing. Testing of an app is an integral part of CI/CD and agile processes. Simple XML files can be used for load testing as well.

1. Azure Test Plans provides manual and exploratory testing tools. Hence, executing multiple scenario based scripted tests gives end to end traceability.
2. Test results are beneficial to record software bugs and defects.
3. Automated tests will typically execute in a Pipeline.
4. Stakeholders feedback can be captured in work items.

Code Management: Azure Repos, Git and GitHub

Source code management (SCM) is the process of tracking modifications and managing changes to source code. SCM allows developers and other stakeholders to see a complete history of all changes made to a shared codebase. This ensures developers are working with up-to-date code and that there are no conflicting code changes.

Version Control

Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control systems are software tools that help software teams manage changes to source code over time.

Version Control, also known as revision control or source control, is the management of changes to documents, computer programs, large web sites, and other collections of information.

- You can think of a version control system ("VCS") as a kind of "database".
- In VCS we call it as “REPOSITORY”

What does a Version control system do?

1. Version control allows for the ability to revert a document to a previous version.
2. If a mistake is made, developers can turn back the clock and get back the previous working application avoiding disruption, Version control system is critical for allowing editors(developers) to track each other's edits and correct mistakes.

3. Changes are usually identified by a number or letter code, termed as the "revision number" or “commit id”.

Why use a Version Control System?

Have you ever -

- Made a change to code, realized it was a mistake and wanted to revert back?
- Lost code and didn’t have a backup of that code?
- Had to maintain multiple versions of a product?
- Wanted to see the difference between two (or more) versions of your code?
- Wanted to prove that a particular change in code broke an application or fixed an application?
- Wanted to see how much work is being done, and where, when and by whom?
- Wanted to experiment with a new feature without interfering with working code?

Advantages of Version Control System

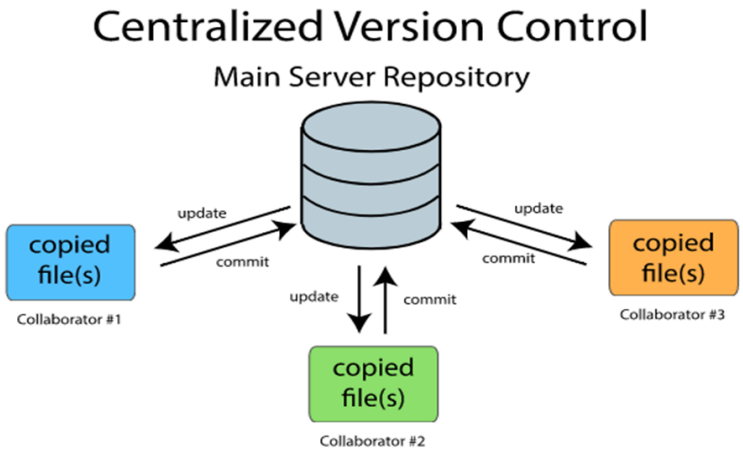
1. Backup
2. Collaboration
3. Storing Versions

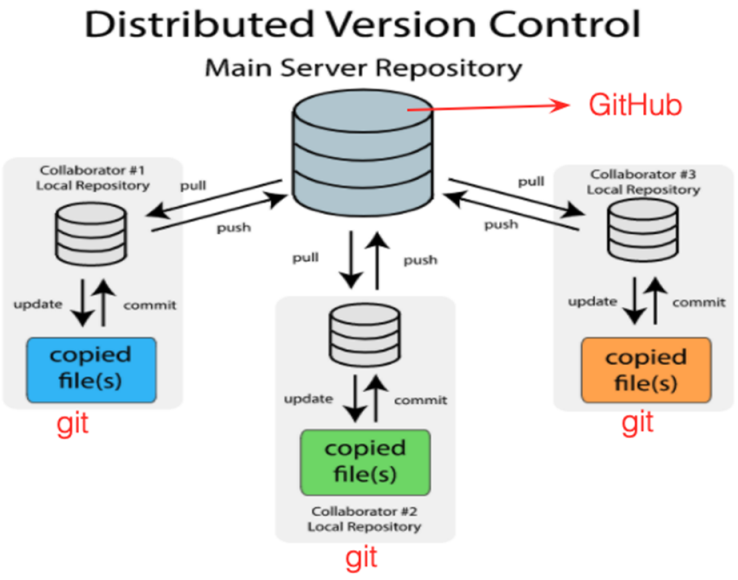
4. Restoring Previous Versions
5. Understanding What Happened

Centralized Version Control System

Uses a central server to store all files and enables team collaboration. But the major drawback of CVCS is its single point of failure, i.e., failure of the central server.

Unfortunately, if the central server goes down for an hour, then during that hour, no one can collaborate at all.





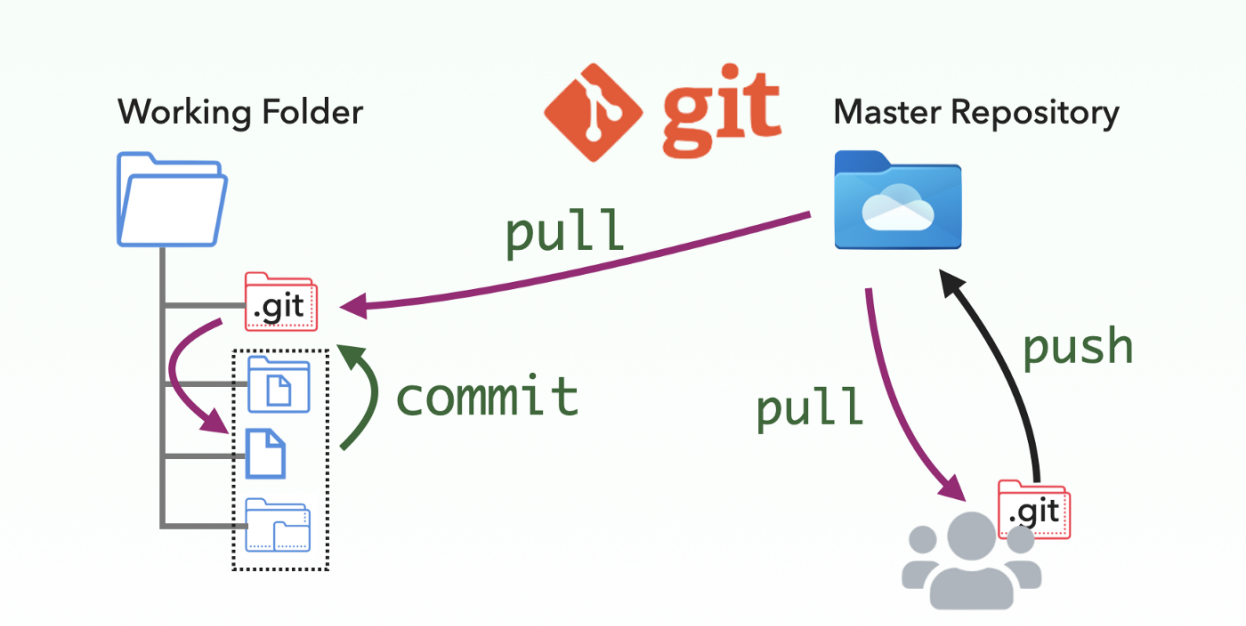
Types of Centralized Version Control Systems:

The two predominant Centralized Version Control Systems in the contemporary world, apart from Azure Repos are:

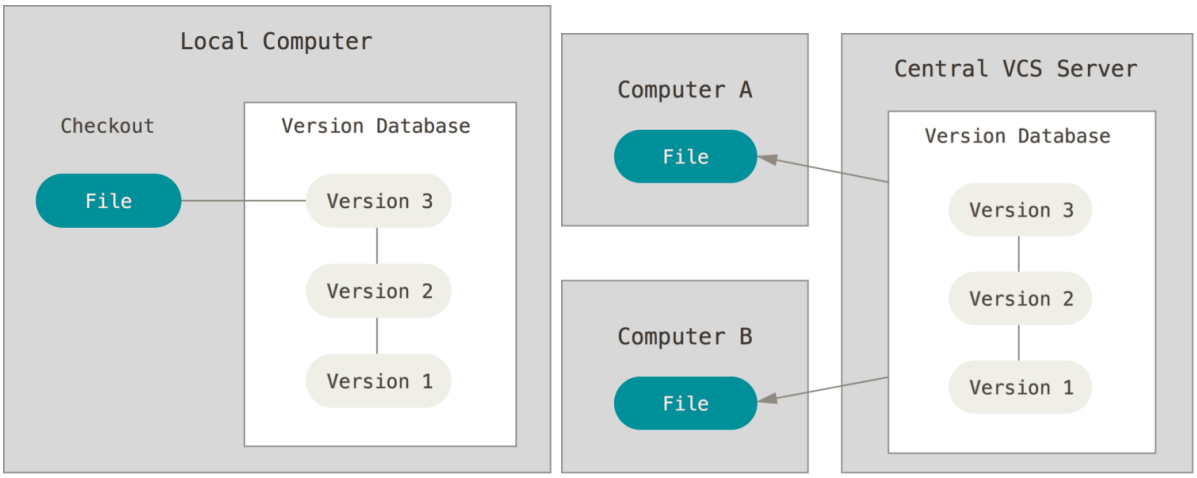
1. Git
2. Github

What is Git?

Git is a Version Control System for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for source code management in software development, but it can be used to keep track of changes in any set of files. As a distributed revision control system it is aimed at speed, data integrity and support for distributed workflows.

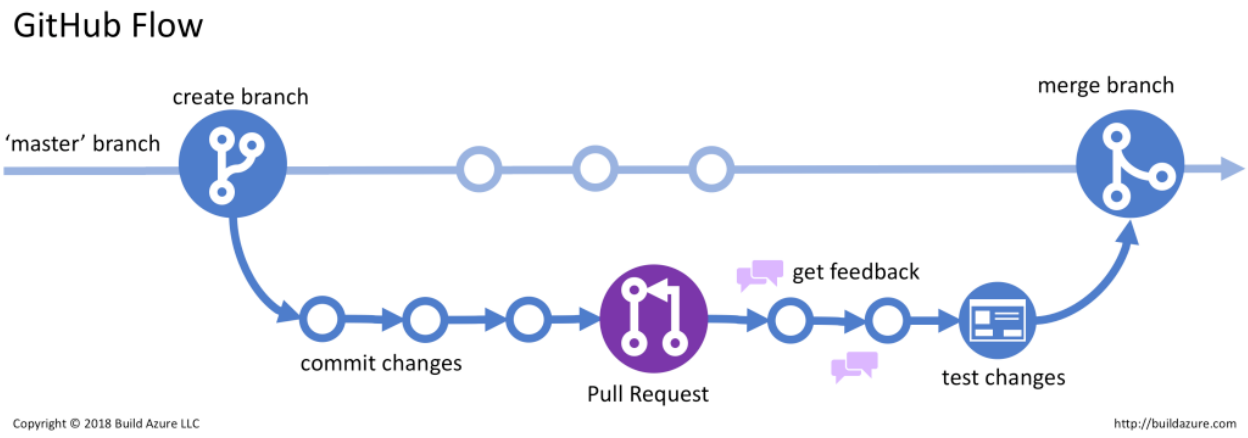


Git is a free and open source Distributed Version Control System designed to handle everything from small to very large projects with speed and efficiency. Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS etc



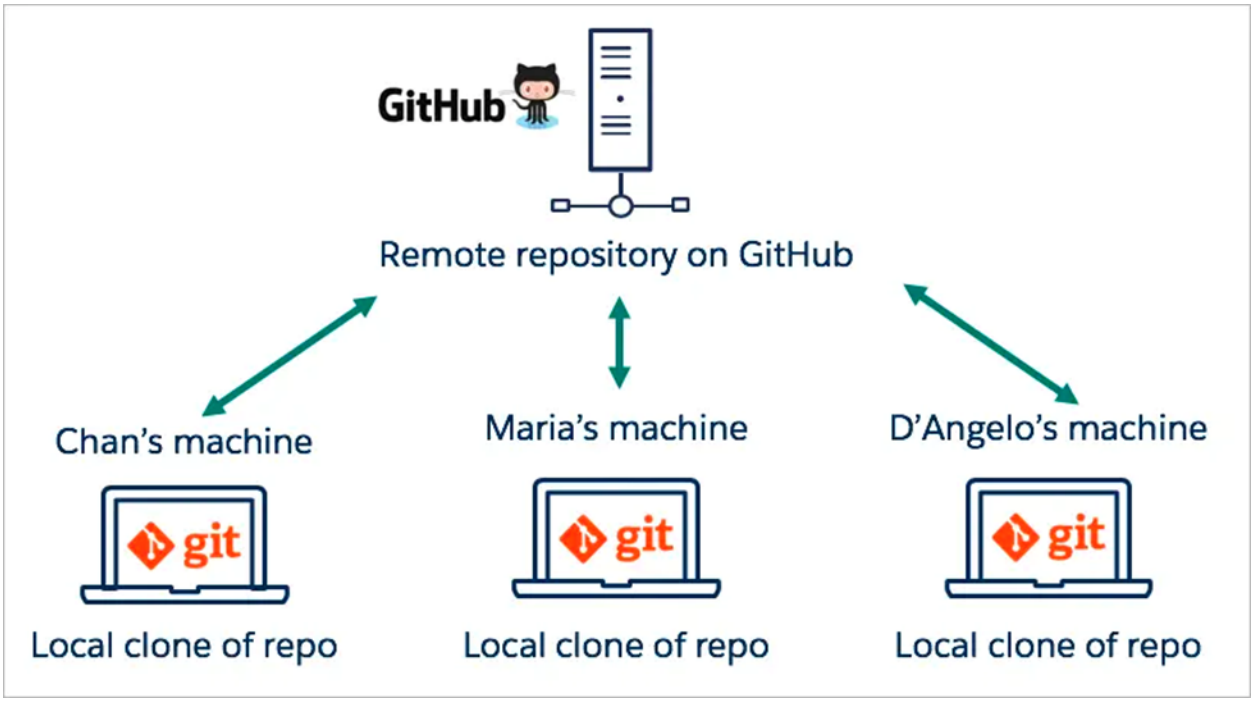
What is Github?

GitHub, Inc. is a provider of Internet hosting for software development and version control using Git. It offers the distributed version control and source code management functionality of Git, plus its own features.



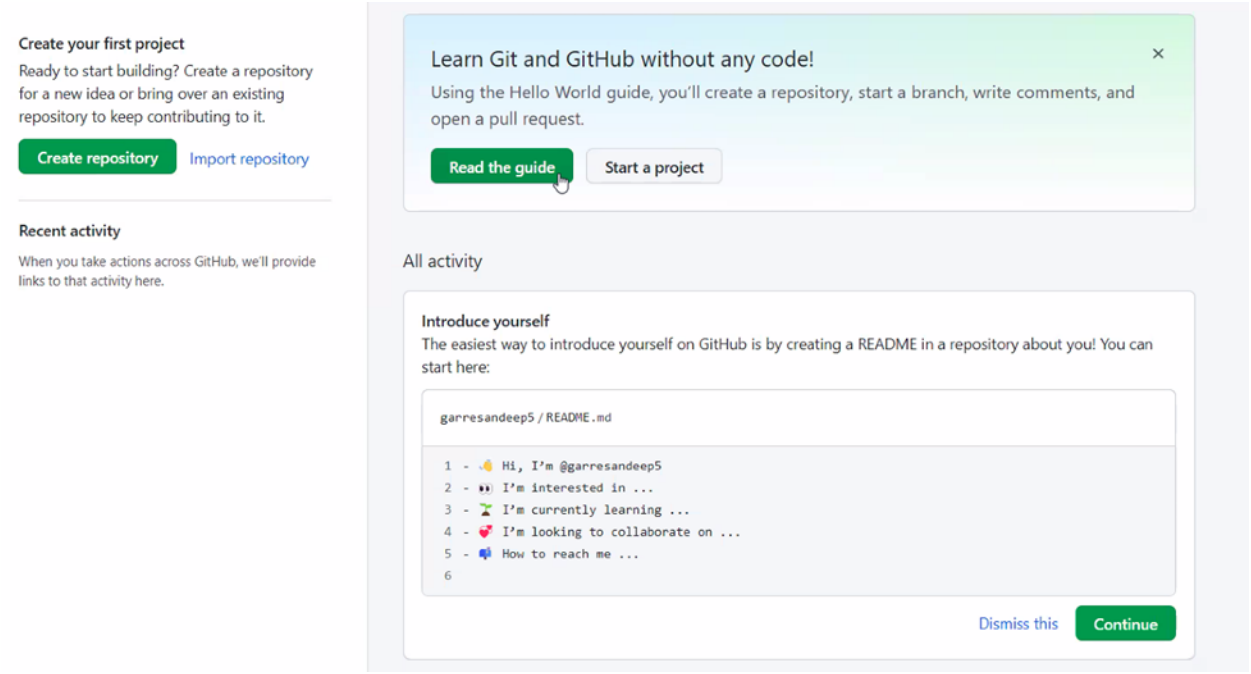
Creating a Repository in Git/Github:

A repository contains all of your project's files and each file's revision history. You can discuss and manage your project's work within the repository. You can own repositories individually, or you can share ownership of repositories with other people in an organization.



Create GitHub Account - https://github.com/

1. A repository contains all of your project's files and each file's revision history. You can discuss and manage your project's work within the repository.
2. To put your project up on GitHub, you'll need to create a repository for it to live in.
3. You can use repositories to collaborate with others and track your work.
4. In the upper-right corner of any page, use the drop-down menu, and select New repository.
5. Type a short, memorable name for your repository. For example, "frontend-app".
6. Optionally, add a description of your repository. For example, "My first Application on GitHub."
7. Choose a repository visibility.
8. Select Initialize this repository with a README.
9. Click Create repository.
10. A commit is like a snapshot of all the files in your project at a particular point in time.



Installing Git:

1. To check if git is available or not use:
2. The git status command displays the state of the working directory and the staging area. It lets you see which changes have been staged, which haven't, and which files aren't being tracked by Git.
3. When you clone a repository, you don't get one file, you get the entire repository - all files, all branches, and all commits
4. Cloning a repository is typically only done once, at the beginning of your interaction with a project.

Setting the Configuration

git config --global user.name "ravi2krishna"

git config --global user.email "email@gmail.com"

git config --global color.ui true

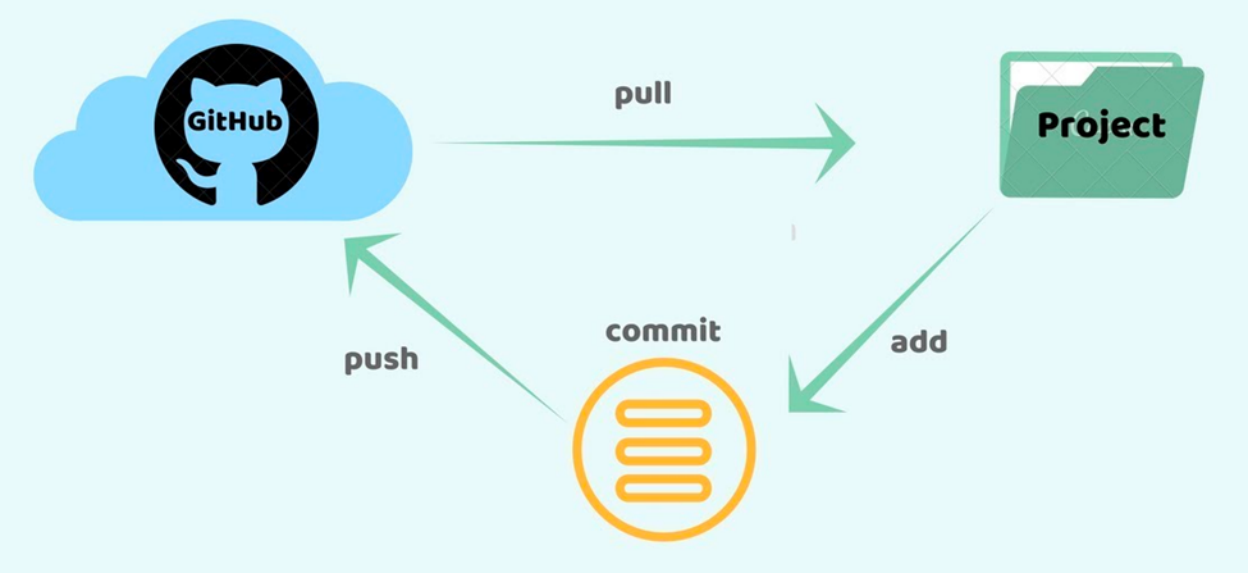
git config --global user.email "you@example.com"

Different Commands in Git/Github:

- The git add command - adds a change in the working directory to the staging area. It tells Git that you want to include updates to a particular file in the next commit.
- The git commit command - creates a commit, which is like a snapshot of your repository. These commits are snapshots of your entire repository at specific times. You should make new commits often, based around logical units of change. Over time, commits should tell a story of the history of your repository and how it came to be the way that it currently is. Commits include lots of metadata in addition to the contents and message, like the author, timestamp, and more.

git commit -m "descriptive commit message"

- The git push command - The git push command is used to upload local repository content to a remote repository. Pushing is how you transfer commits from your local repository to a remote repo.
- Changes are usually identified by a number or letter code, termed as the "revision number" or “commit id”.



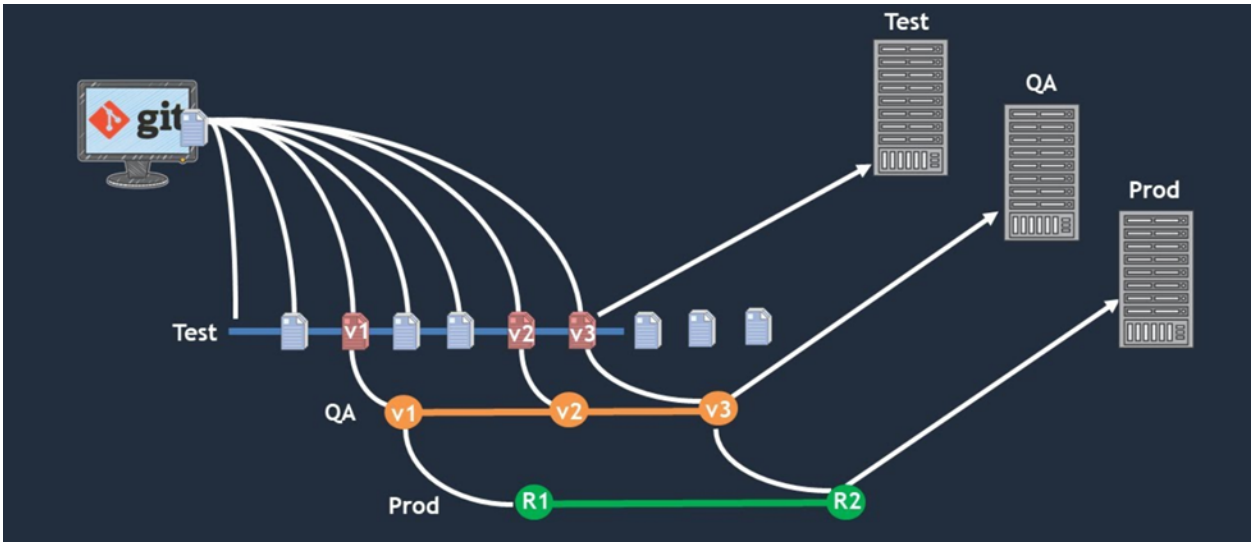
Branching in Git:

Branching means you diverge from the main line(master - working copy of application) of development and continue to do work without messing with that main line.

In a collaborative environment, it is common for several developers to share and work on the same source code. Some developers will be fixing bugs while others would be implementing new features. Therefore, there has got to be a manageable way to maintain different versions of the same code base.

This is where the branch function comes to the rescue. Branch allows each developer to branch out from the original code base and isolate their work from others. Another good thing about branch is that it helps Git to easily merge the versions later on.

It is a common practice to create a new branch for each task (eg. bug fixing, new features etc.)



Creating and deleting branches within your repository:

You can create or delete branches directly on GitHub.

Creating a branch

1. On GitHub.com, navigate to the main page of the repository.
2. Optionally, if you want to create your new branch from a branch other than the default branch for the repository, click NUMBER branches then choose another branch:
3. Click the branch selector menu.
4. Type a unique name for your new branch, then select Create branch

Deleting a branch

1. If the branch you want to delete is the repository's default branch, you must choose a new default branch before deleting the branch.
2. On GitHub.com, navigate to the main page of the repository.
3. Above the list of files, click NUMBER branches.
4. Scroll to the branch that you want to delete, then click .

Creating Branch CLI

When you create a new branch, git creates a new pointer for you to move around to view the branches.

Switching Branches

To switch between branches, you can use git checkout command

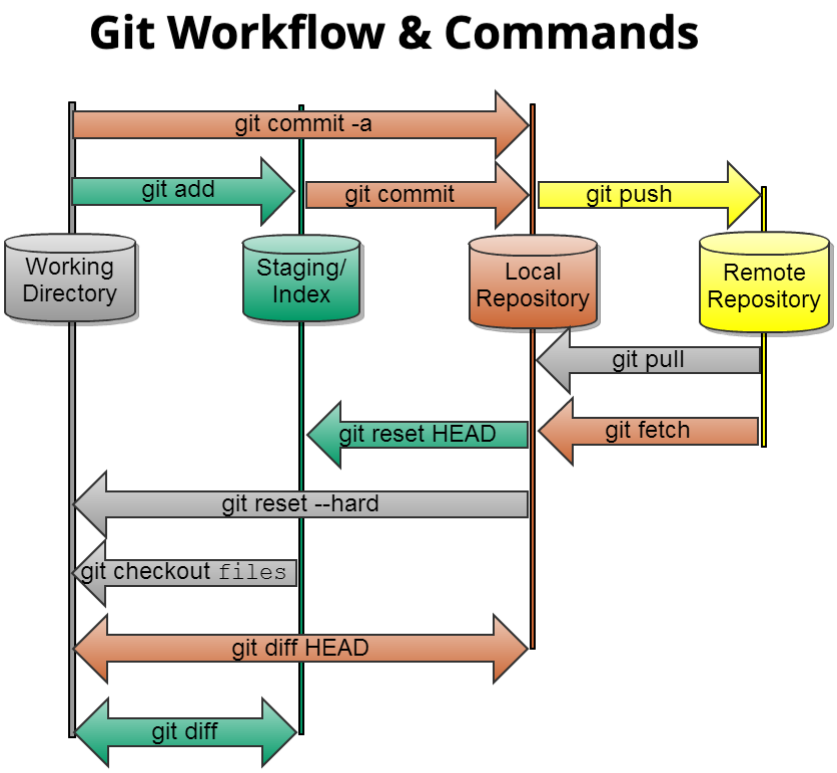
Merging Branches

The git merge command lets you take the independent lines of development created by the git branch and integrate them into a single branch.

Git merge will combine multiple sequences of commits into one unified history. In the most frequent use cases, git merge is used to combine two branches.

Deleting Branch CLI

Delete a branch on your local filesystem



Branching lets you have different versions of a repository at one time.

By default, your repository has one branch named main that is considered to be the definitive branch. You can use branches to experiment and make edits before committing them to main.

When you create a branch off the main branch, you're making a copy, or snapshot, of main as it was at that point in time. If someone else made changes to the main branch while you were working on your branch, you could pull in those updates.

This diagram shows:

1. The main branch
2. A new branch called feature
3. The journey that feature takes before merged into main

Have you ever saved different versions of a file? Something like:

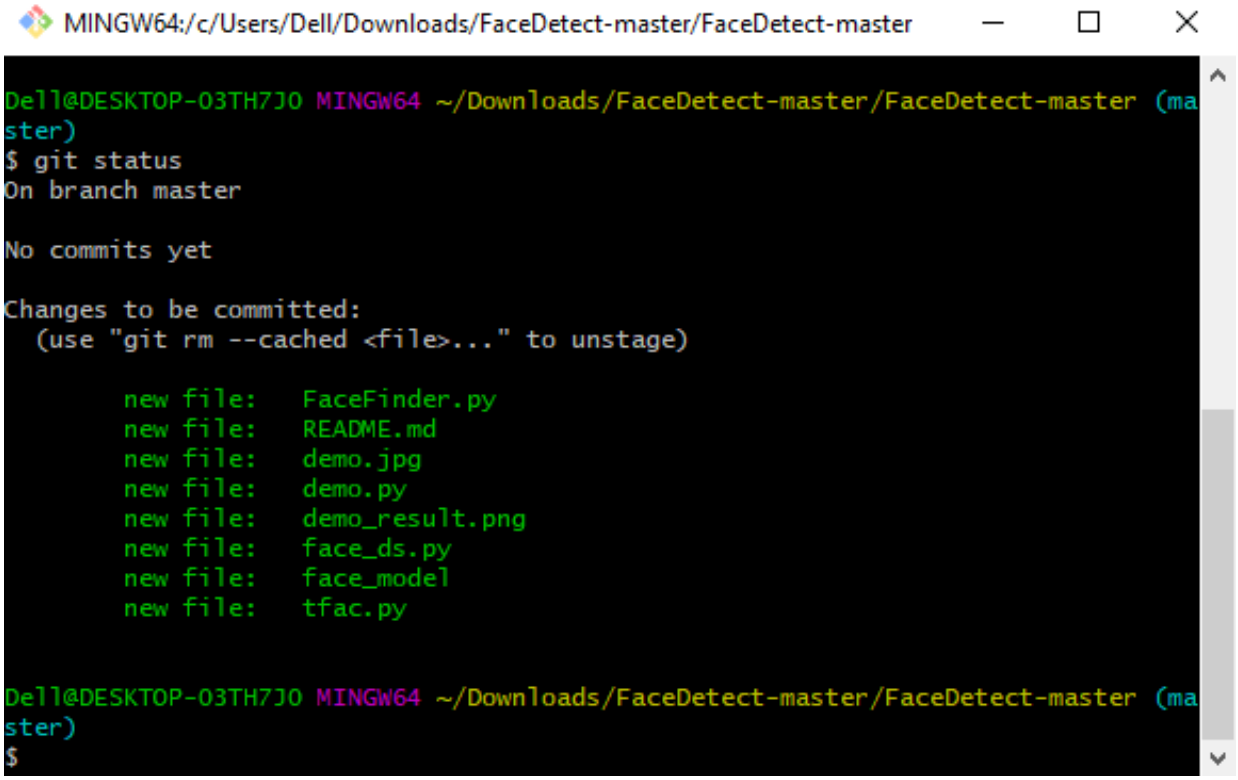
- story.txt
- story-joe-edit.txt
- story-joe-edit-reviewed.txt

Branches accomplish similar goals in GitHub repositories.

Here at GitHub, our developers, writers, and designers use branches for keeping bug fixes and feature work separate from our main (production) branch. When a change is ready, they merge their branch into main.

Pushing Changes:

- Use git push to push commits made on your local repository to a remote repository.
- The git push command is used to upload local repository content to a remote repository. Pushing is how you transfer commits from your local repository to a remote repo.
- When you created your new repository, you initialized it with a README file. README files are a great place to describe your project in more detail, or add some documentation such as how to install or use your project. The contents of your README file are automatically shown on the front page of your repository.



Pull Request:

1. Pull requests let you tell others about changes you've pushed to a branch in a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch.
2. After initializing a pull request, you'll see a review page that shows a high-level overview of the changes between your branch (the compare branch) and the repository's base branch.
3. After you're happy with the proposed changes, you can merge the pull request. If you're working in a shared repository model, you create a pull request and you, or someone else, will merge your changes from your feature branch into the base branch you specify in your pull request.

Making and committing changes

When you created a new branch in the previous step, GitHub brought you to the code page for your new readme-edits branch, which is a copy of main.

You can make and save changes to the files in your repository. On GitHub, saved changes are called commits. Each commit has an associated commit message, which is a description explaining why a particular change was made. Commit messages capture the history of your changes so that other contributors can understand what you've done and why.

1. Click the README.md file.
2. Click to edit the file.
3. In the editor, write a bit about yourself.
4. In the Commit changes box, write a commit message that describes your changes.
5. Click Commit changes.

These changes will be made only to the README file on your readme-edits branch, so now this branch contains content that's different from main.

Opening a pull request

Now that you have changes in a branch off of main, you can open a pull request.

Pull requests are the heart of collaboration on GitHub. When you open a pull request, you're proposing your changes and requesting that someone review and pull in your contribution and merge them into their branch. Pull requests show diffs, or differences, of the content from both branches. The changes, additions, and subtractions are shown in different colors.

As soon as you make a commit, you can open a pull request and start a discussion, even before the code is finished.

By using GitHub's @mention feature in your pull request message, you can ask for feedback from specific people or teams, whether they're down the hall or 10 time zones away.

You can even open pull requests in your own repository and merge them yourself. It's a great way to learn the GitHub flow before working on larger projects.

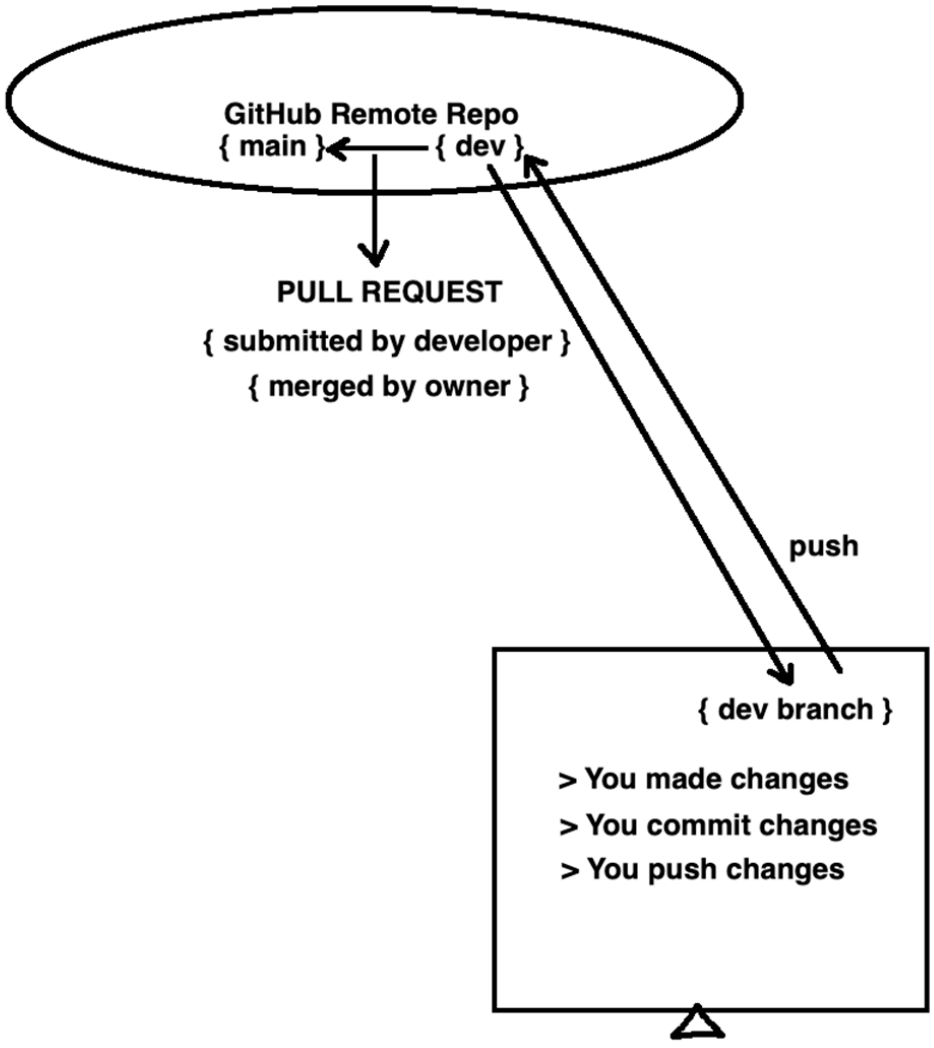
Click the Pull requests tab of your hello-world repository.

- Click New pull request
- In the Example Comparisons box, select the branch you made, readme-edits, to compare with main (the original).
- Look over your changes in the diffs on the Compare page, make sure they're what you want to submit.

- Click Create pull request.
- Give your pull request a title and write a brief description of your changes. You can include emojis and drag and drop images and gifs.
- Click Create pull request.
- Your collaborators can now review your edits and make suggestions.

Merging your pull request

- In this final step, you will merge your readme-edits branch into the main branch.
- Click Merge pull request to merge the changes into main.
- Click Confirm merge.
- Go ahead and delete the branch, since its changes have been incorporated, by clicking Delete branch.



Merge Pull Request

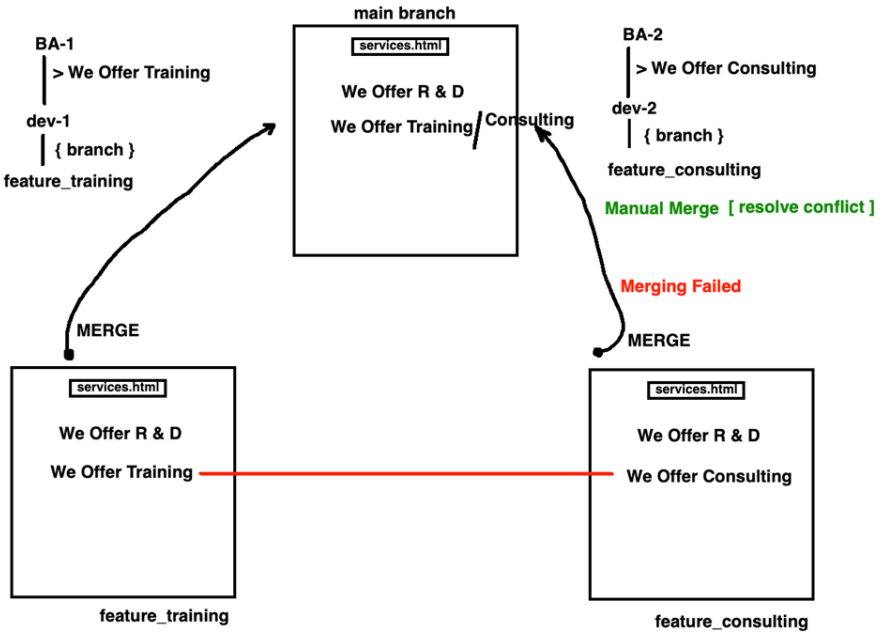
Merge a pull request into the upstream branch when work is completed. Anyone with push access to the repository can complete the merge.

In a pull request, you propose that changes you've made on a head branch should be merged into a base branch. By default, any pull request can be merged at any time, unless the head branch is in conflict with the base branch. However, there may be restrictions on when you can merge a pull request into a specific branch. For example, you may only be able to merge a pull request into the default branch if required status checks are passing. For more information, see "protected branches."

- Under your repository name, click Pull requests
- In the "Pull Requests" list, click the pull request you'd like to merge.

Depending on the merge options enabled for your repository, you can:

Merge all of the commits into the base branch by clicking Merge pull request. If the Merge pull request option is not shown, then click the merge drop down menu and select Create a merge commit.



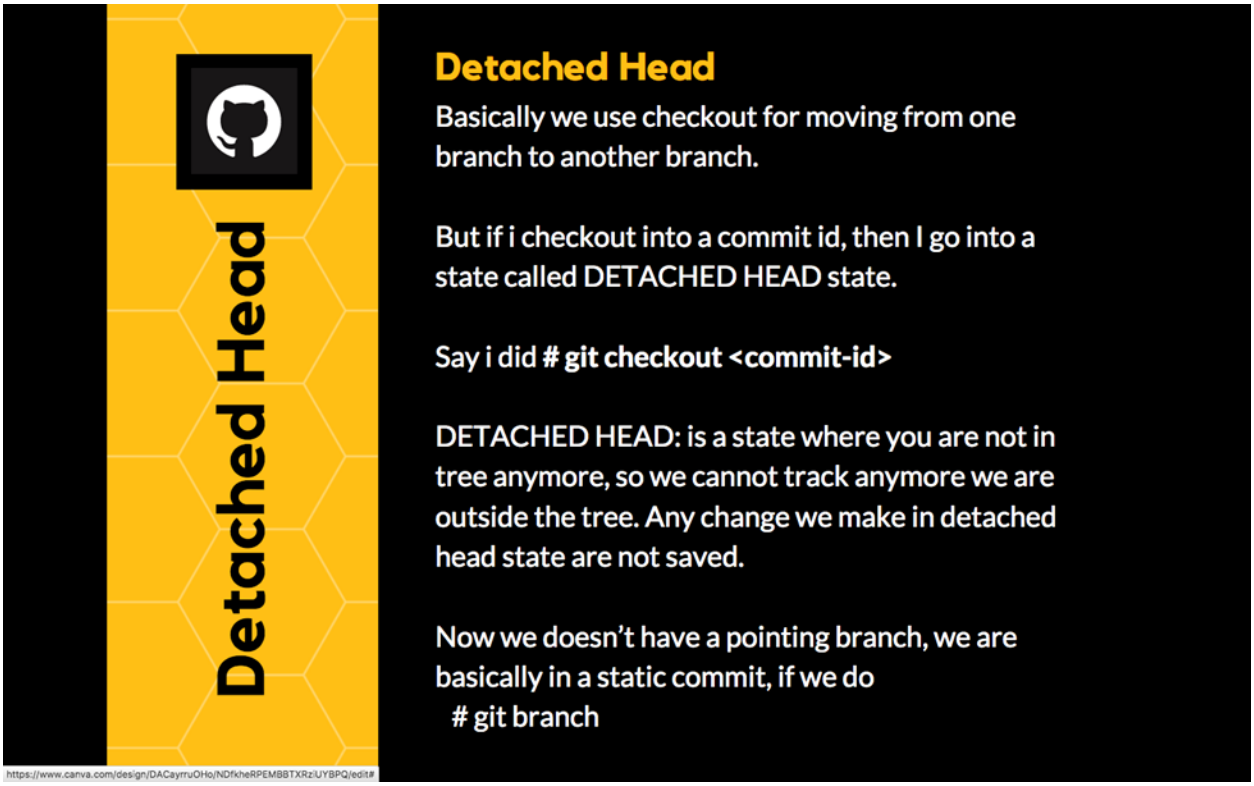
Merge Conflicts

- A Merge Conflict happens when two branches have been modified with the changes and are subsequently merged.

- Git doesn't know which changes to keep, and thus needs human intervention to resolve the conflict.

Undoing Changes

- Made changes but didn't stage them and commit, use -- checkout
- Manual undo
- git checkout -- <file>
- Made changes but staged the changes, use reset
- git reset HEAD <file>
- Made changes, staged the changes and commited changes
- git revert <commit>

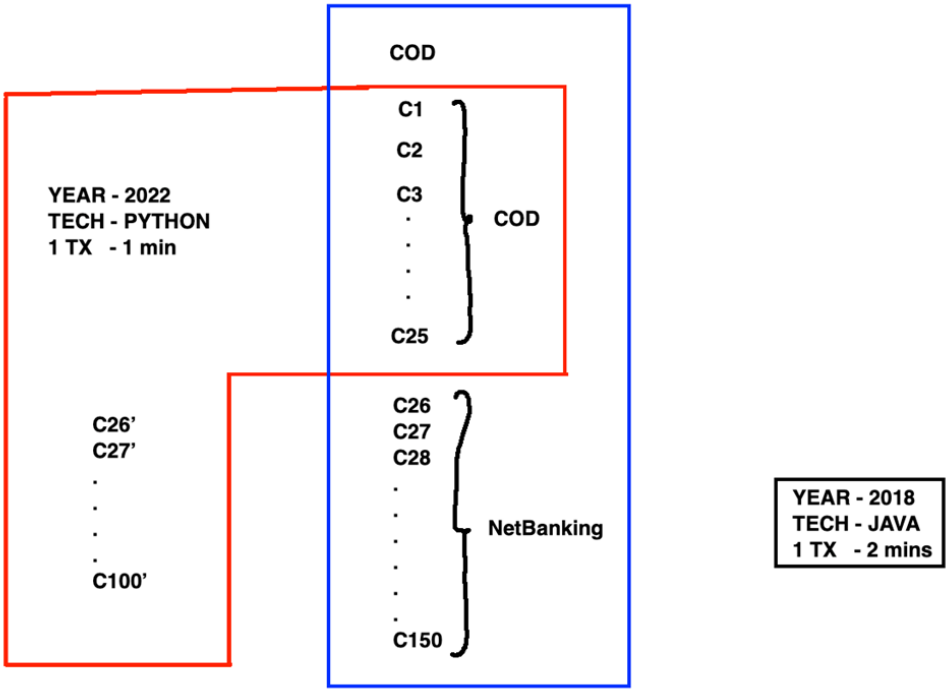


Reason for checking out into a commit id

To know what happened at that particular commit.

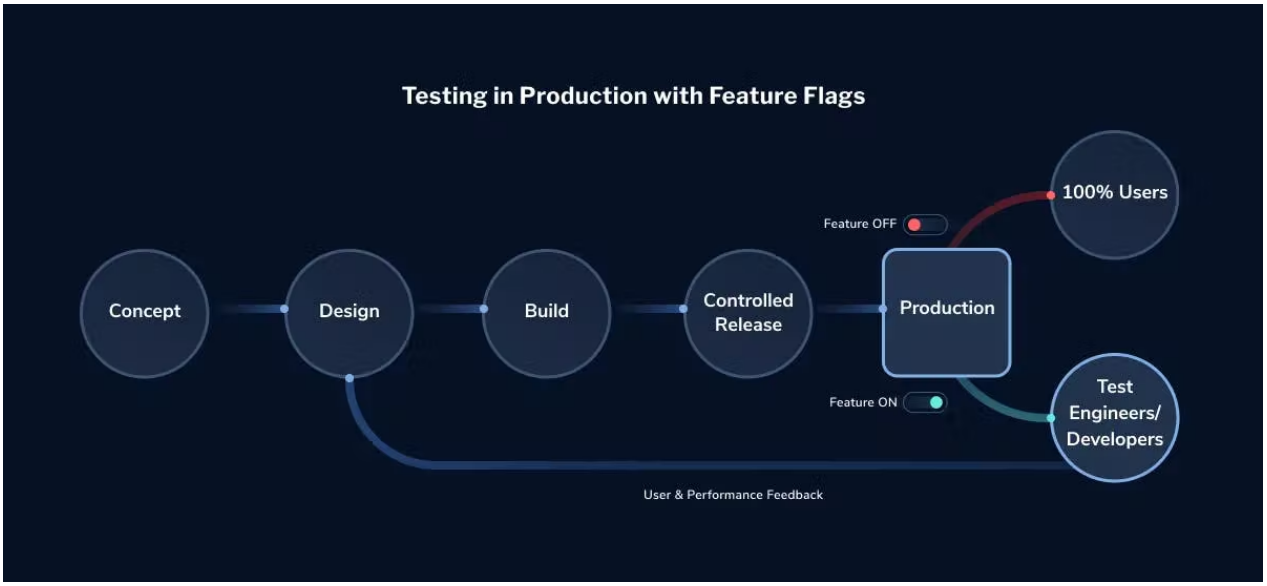
Let's say you have created a file and put some phone no in there, and now person1 changed the file and committed with a message "ph no changed", then again person2 changed the file and committed with same message "ph no changed" later again person3 changed the file and committed with same message "ph no changed".

Now here we can't rely on commit message itself, this is were detached head is needed.



Testing Release:

Release testing refers to coding practices and test strategies that give teams confidence that a software release candidate is ready for users. Release testing aims to find and eliminate errors and bugs from a software release so that it can be released to users. Let's dive in and explore several methods used to perform release testing.



Approach to test release:

Release testing is a testing approach rather than a single testing method. It spans several common software tests like unit tests, smoke tests, regression tests, and so on. Modern testing practices have also emerged that align with CI/CD and DevOps. More and more teams, for example, have begun testing small batches of functionality in production. And they're using LaunchDarkly feature flags to do this in a safe, controlled manner.

An ideal testing approach should optimize for speed, stability, and user experience. That is, you want to ship code as fast as possible while maintaining high application performance and delivering a topflight experience to end-users. Running tests in production with feature flags can help development teams strike that perfect balance.

Test Plans:

A Test Plan refers to a detailed document that catalogs the test strategy, objectives, schedule, estimations, deadlines, and the resources required for completing that particular project. Think of it as a blueprint for running the tests needed to ensure the software is working properly – controlled by test managers.

Components of a Test Plan

1. ***Scope*:** Details the objectives of the particular project. Also, it details user scenarios to be used in tests. If necessary, the scope can specify what scenarios or issues the project will not cover.
2. ***Schedule***: Details start dates and deadlines for testers to deliver results.
3. ***Resource Allocation***: Details which tester will work on which test.
4. ***Environment***: Details the nature, configuration, and availability of the test environment.
5. ***Tools***: Details what tools are to be used for testing, bug reporting, and other relevant activities.
6. ***Defect Management***: Details how bugs will be reported, to whom and what each bug report needs to be accompanied by. For example, should bugs be reported with screenshots, text logs, or videos of their occurrence in the code?
7. ***Risk Management***: Details what risks may occur during software testing, and what risks the software itself may suffer if released without sufficient testing.
8. ***Exit Parameters:*** Details when testing activities must stop. This part describes the results that are expected from the QA operations, giving testers a benchmark to compare actual results to.

How to create a Test Plan?

Creating a Test Plan involves the following steps:

1. Product Analysis
2. Designing Test Strategy
3. Defining Objectives
4. Establish Test Criteria
5. Planning Resource Allocation
6. Planning Setup of Test Environment
7. Determine test schedule and estimation
8. Establish Test Deliverables

Deliverables required before testing
Documentation on

- Test Plan
- Test Design

Deliverables required during testing
Documentation on

- Test Scripts
- Simulators or Emulators (in early stages)
- Test Data

- Error and execution logs

Deliverables required after testing
Documentation on

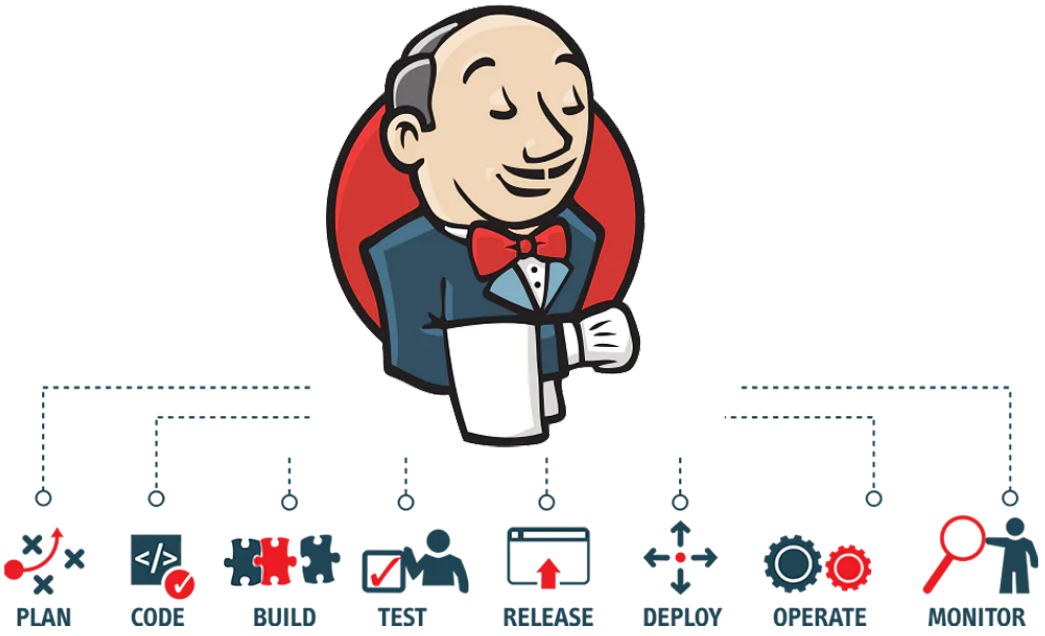
- Test Results
- Defect Reports
- Release Notes

Maintenance: CI/CD Pipelines

What is a CI/CD Pipeline?

The term “CI/CD pipeline” refers to the Continuous Integration/Continuous Delivery pipeline. Before we dive deep into this segment, let's first understand what is meant by the term ‘pipeline.’

A pipeline is a concept that introduces a series of events or tasks that are connected in a sequence to make quick software releases. For example, there is a task, that task has got five different stages, and each stage has got several steps. All the steps in phase one must be completed, to mark the latter stage to be complete.



What is Continuous Integration (CI)?

Continuous Integration is a practice that integrates code into a shared repository. It uses automated verifications for the early detection of problems. Continuous Integration doesn't eliminate bugs but helps in finding and removing them quickly.

What is Continuous Delivery (CD)?

Continuous Delivery is the phase where the changes are made in the code before deploying. The team in this phase decides what is to be deployed to the customers and when. The final goal of the pipeline is to make deployments.

When both these practices come together, all the steps are considered automated, resulting in the process we know as CI/CD. Implementation of CI/CD enables the team to deploy codes quickly and efficiently. The process makes the team more agile, productive, and confident.

Jenkins for CI/CD Pipeline

Jenkins is the DevOps tool that is most used for CI/CD pipelines. Therefore, we must have a look at the basics of Jenkins and understand why it is the most sought-after tool to build this pipeline.

Best Practices for CI/CD Pipeline Security

- DevOps comes with its own set of security risks, so it’s important to have a set of best practices in place. These include:
- Employ one-time passwords for your more sensitive systems and tools
- Use password managers and rotate your passwords after every use
- Take out all hard coded secrets from CI/CD config files and Jenkins files
- Make sure secrets aren’t accidentally passed along during builds for pull requests occurring in CI/CD pipeline
- Follow the practice of least privilege. This means giving access only to the requisite secrets, nothing beyond that. This practice applies to applications, employee access, systems, and connected devices that require permissions or privileges to perform certain tasks.
- Use authentication measures to confirm machine identity
- Keep track of who has access to what resources. Create a clear repository of access management, regardless of whether it’s task-based, time-based, or role-based.
- Distribute secrets among Jenkinsfiles to reduce the risk of attacks on the files
- Scan scripted builds and regularly monitor source code for vulnerabilities before deploying an app into production