SYSOPS TOOLS ASSIGNMENT

SUBMITTED TO: Mr. Rajat Goyal

SUBMITTED BY: Mayank (Software Trainee)

GITHUB (Web Based Platform)

- **Definition:** "GitHub" refers to a web-based platform and hosting service for version control using the Git system. It provides developers and teams with a powerful and collaborative way to manage and store their source code, track changes, and work together on software development projects.
- GitHub integrates with various Continuous Integration (CI) tools, allowing developers to automate the testing and deployment processes, ensuring code quality and efficiency.
- Once you log in to Github Portal, we will be having a hierarchy that will look like this:

Enterprises

|

Organizations

|

Teams

- Now we can select any organization there, inside each organization there will be multiple teams and in a team, there will be multiple members in which one might have any type of access like developer or administrator access etc.
- There will be multiple repositories in one organization, and we can create a new GitHub repository as well according to the requirements like we normally do.

When we will create a new repository, there will be many other sections which have different purpose, which can be given as:

1) **Projects Section:** The "Projects" section in a GitHub repository serves as a project management tool that helps organize and track the progress of tasks, issues, and features related to the repository. It provides a visual and flexible way to manage the development workflow, collaborate with team members, and keep track of project milestones. According to the requirements, one can create a github project.

2) **Packages Section:** The main purpose of using GitHub Packages is to provide a central and versioned repository for storing and sharing software packages. Developers can publish their packages to GitHub Packages such as npm packages for Node.js, RubyGems for Ruby, Maven packages for Java, Docker containers, and more. and use them in their projects or share them with others.

3) **Teams Section:**

i) A team is a group of GitHub users with specific access permissions, and it is often used to group users based on their roles or responsibilities in a project.

ii) One can create a team as well according to the requirements. While creating teams, you have to select an Identity Provider Group which will be a group created on Azure Active Directory side. So that, one can sync azure AD Groups over here on Github.

- **Enterprise Owner VS Organization Owner:** If one has any organization permission, he/she can see what is inside that particular organization. But one won't be having access to all other organization until he/she is the owner or member with some particular access whereas, enterprise owners have access to everything.

4) **Settings Section:** The settings section will be visible only to the organization owner or admin of the organization. Now settings section includes many more options, let's understand that as well:-

- **General Settings:** Like Name of organization etc.
- **Insights:** GitHub's "Insights" provides valuable data and analytics related to repositories and projects. It offers various tools and metrics that help repository owners, maintainers, and contributors gain insights into the project's performance, community engagement, and code quality. In simple words, It helps your team to make you look to keep track of the project boards and everything.
- **Billing & Plans:** The "Billing & Plans" section in GitHub provides options to manage and choose the appropriate subscription plan for your specific requirements. In the "Billing & Plans" section, you can manage your billing information, including credit card details and billing address. This ensures that your payments are processed accurately and that your account remains in good standing.
- **Repository Roles:** GitHub provides different repository roles and permissions to control access and contributions to a repository. These roles help manage collaboration and ensure that the right users have appropriate privileges within a repository. One can use predefined roles as well as can create a custom role. Predefined roles can be given as:

i) Read
ii) Write
iii) Triage
iv) Maintain
v) Admin

- **Actions:** GitHub Actions is integrated directly into GitHub, making it easy to set up and manage workflows without relying on external CI/CD services. The "Actions" section in a GitHub repository allows you to view and manage the workflows associated with that repository. From there, you can create new workflows, view workflow runs and logs, and manage the settings for your workflows.

- **Action Secrets/Variables:** In GitHub Actions, "secrets" (formerly known as "encrypted secrets") and "variables" are used to store sensitive or configurable data that is used in your workflows. These features help you keep sensitive information secure and allow you to customize your workflows without hardcoding values directly into your workflow files.
- **Third Party Access:** Under the developer settings, we will get all the options of third party access like oauth apps, PAT, and Github Apps.

i) **GitHub Apps:** GitHub Apps are a type of GitHub integration that provides a more secure and fine-grained way for third-party applications to interact with GitHub repositories and organizations. GitHub Apps have their own identity and permissions, allowing them to access specific repositories and perform actions on behalf of users or organizations.
ii) **Personal Access Token:** Personal Access Tokens (PATs) in GitHub are used to authenticate and authorize third-party applications, scripts, or command-line tools to access specific resources in a user's GitHub account. PATs act as a type of API token that grants limited access to the user's account without the need for providing their actual GitHub credentials (username and password). By default, after 30 days, the token will be expired.

- **Service User: -** "Service User" refers to a user account that is specifically created and used to represent an automated service, application, or bot. Service users are not intended for human interaction but instead act as an identity for non-human entities to perform certain actions on GitHub. By using service users, organizations and developers can ensure that automated processes and tools can perform actions on GitHub without requiring access to personal user accounts. Service users can be assigned specific

permissions, such as read-only access or more advanced privileges, depending on the tasks they need to perform.

❖ Github Commands:

- Git clone: The **git clone** command is used in Git version control system to create a copy of a remote repository on your local machine. It allows you to download all the files, commit history, and branches from the remote repository to your local computer, enabling you to work with the code locally and make changes as needed.

- Git merge: The **git merge** command is used in Git version control to combine changes from one branch into another branch. It allows you to integrate the changes made in one branch into another branch, resulting in a new commit that contains the combined changes.

- Git revert: The **git revert** command in Git is used to create a new commit that undoes the changes introduced by a specific commit. It is different from **git reset**, which removes commits from the commit history, as **git revert** maintains the commit history but adds a new commit that negates the changes made in the specified commit.

- Git log: The **git log** command in Git is used to display the commit history of a repository. When you run **git log**, it shows a list of commits in reverse chronological order, with the most recent commit appearing first.

- Git init: The **git init** command is used to initialize a new Git repository in an existing or empty directory. When you run **git init**, it sets up all the necessary files and data structures that Git needs to start version controlling the files within that directory.

- Git status: The **git status** command in Git is used to display the current status of your working directory and staging area. It shows you information about any changes you've made to your files and whether these changes have been staged (marked for the next commit) or not.

- Git add: The **git add** command in Git is used to add changes made to files in your working directory to the staging area. The staging area is an intermediate step before creating a commit in which you can review and organize the changes you want to include in the next commit.

- Git commit: The **git commit** command in Git is used to save the changes you've made to the files in your local repository. After making modifications to your files, you use this command to create a new commit, which records the changes you made, along with a commit message to describe what changes were made.

- Git push: The **git push** command in Git is used to upload the local commits you've made to a remote repository. After making changes and committing them in your local repository, you use **git push** to synchronize your local changes with the corresponding remote repository.

- Git pull: The **git pull** command in Git is used to fetch and merge changes from a remote repository into your local branch. It is a combination of two other Git commands: **git fetch** and **git merge**.

- Git checkout: The **git checkout** command in Git is a versatile command used for various purposes depending on the arguments provided. The primary use of **git checkout** is to switch between different branches in your repository, allowing you to work on a specific branch or view the content of a different branch. However, it can also be used for other tasks such as checking out specific files or reverting changes.

- Git config: The **git config** command in Git is used to configure various settings that control the behavior and appearance of Git on your local machine. These configurations are specific to the repository (local) or global (applied to all repositories on your machine).

SONARQUBE

Definition: SonarQube is an open-source platform designed to help developers and teams ensure code quality and maintain code security. It provides continuous inspection of your

codebase to identify and fix potential code quality issues, bugs, vulnerabilities, and code smells. It supports multiple programming languages, making it versatile and useful for a wide range of projects.

Below are some key features of SonarQube, along with detailed explanations:

- **Static Code Analysis:** SonarQube performs static code analysis, which means it examines the source code without actually executing it. It checks for various code quality issues, bugs, vulnerabilities, code smells, and security flaws. The analysis is based on predefined rules and coding standards, as well as custom rules that you can configure according to your project's requirements.
- **Support for Multiple Languages:** SonarQube supports a wide range of programming languages, making it suitable for diverse projects. Some of the languages it supports include Java, C/C++, C#, Python, JavaScript, TypeScript, Ruby, PHP, and many others. This multi-language support allows teams to manage code quality consistently across different parts of a project.
- **Customizable Rules and Profiles:** SonarQube allows you to customize the rules and profiles used during code analysis. You can enable/disable specific rules, set severity levels, and create custom rules to address project-specific requirements or coding standards. This flexibility allows you to tailor the analysis to match your team's coding practices.
- **Issue Tracking and Management:** After analyzing the code, SonarQube generates detailed reports that highlight the identified issues, bugs, vulnerabilities, and code smells. These issues are categorized based on their severity, and SonarQube provides various dashboards and visualizations to help you track and manage these issues effectively.
- **Code Duplication Detection:** SonarQube can detect duplicated code within a codebase. This helps developers identify areas where refactoring or code reuse can be applied to improve code maintainability and reduce redundancy.
- **Code Security Analysis:** SonarQube includes built-in security rulesets that check for potential security vulnerabilities in your code. It helps you identify common security issues, such as SQL injection, cross-site scripting (XSS), and insecure cryptographic practices, allowing you to write more secure code.
- **Integration with CI/CD Pipelines:** SonarQube seamlessly integrates with continuous integration (CI) and continuous delivery (CD) pipelines. You can configure SonarQube to automatically analyze code whenever new changes are pushed, providing quick feedback to developers about code quality and security.
- **Quality Gates:** Quality Gates are a powerful feature in SonarQube that allows you to define sets of criteria that a project must meet before it can be considered for release. Quality Gates act as a quality threshold, and if the project does not meet the defined

criteria, it cannot be promoted to production. This helps enforce code quality standards and ensures that only code meeting those standards gets released.

- **Code Coverage Analysis:** SonarQube can integrate with code coverage tools (e.g., JaCoCo for Java) to analyze the percentage of code covered by unit tests. This helps in identifying areas of the codebase that lack test coverage and need additional testing.
- **Code Hotspots and Maintainability Ratings:** SonarQube identifies code hotspots, which are areas of the codebase that have a high concentration of issues. It also provides maintainability ratings for projects, allowing teams to focus on improving code quality in the most critical areas.

In summary, SonarQube is a comprehensive code quality management tool that offers various features to analyze, track, and improve code quality, security, and maintainability in software projects. Its flexibility, support for multiple languages, and integration capabilities make it a valuable asset for development teams aiming to build robust and high-quality software applications.

SWAGGER HUB

Definition: SwaggerHub is a web-based platform that allows developers to create, document, and manage APIs (Application Programming Interfaces) in an easy and organized way. It provides tools and features that enable developers to design, test, and share their APIs with others. SwaggerHub helps streamline the API development process, making it more efficient and collaborative for teams working on building software applications or services that need to communicate with each other. After creating an API, we can sync that API with GitHub(or any CI/CD tool).

Here are some of the features of SwaggerHub:

- **API Design Editor:** SwaggerHub provides a user-friendly editor to visually create and design APIs, making it easier to define endpoints and data structures.
- **API Documentation:** It automatically generates interactive documentation for your APIs, helping developers understand how to use them effectively.
- **Collaboration:** SwaggerHub allows multiple team members to work together on API design and documentation in a coordinated manner.
- **Mock Servers:** You can create mock servers to simulate API behavior, enabling early testing and development even before the actual API is implemented.

- **API Testing:** SwaggerHub allows you to define and execute tests for your APIs to ensure they function as intended.
- **API Monitoring:** It provides real-time monitoring of API performance and usage, helping you identify and address potential issues.
- **API Publishing:** Once an API is ready, you can easily publish and share it with others, making it accessible to developers and partners.
- **Integration with Git:** SwaggerHub can be integrated with Git, making it convenient to manage API definitions using version control.
- **Security and Access Control:** It offers access control features to protect sensitive API information and ensure only authorized users can make changes.
- **API Lifecycle Management:** SwaggerHub supports the entire API lifecycle, from design to deployment and maintenance, making it a comprehensive tool for API management.

- Let's start by creating an API now by just clicking on "Create New" Button.
- After that, a template will open to configure the API. We can create the API under a user as well as inside any organization. There you just need to select a project name.

i) **Specification:** Specification is just the version of API that we want to create.
ii) **Template:** There will be some preexisting templates and we can select any one of the templates like Simple API, IOT etc.
iii) **Name:** Give any name to newly created API. Remember, the name will be in alphanumeric pattern. You can't put a space between if you put that, an error will be displayed.
iv) **Visibility:** We can set the visibility to private or public according to the requirements. For example, if you have created an API inside an organization and you have set it to private, so all the members of that particular organization will be able to see that API. But if you have provided the visibility as public, so API will be visible to everyone, irrespective of the organization.
v) At last, just click on Create API.

- Now the URL present in servers' part, you can just copy that and send it to someone so they will be able to redirect over here.
- The main thing over here is versions, what if you want to change the versions. For example, you have done some changes in your current API so after doing that, you can simply add a new version by clicking on "Add New Version" Button. One more thing to note here is if you have created 2-3 versions, so the newly created version will be the one who will be active.
- You can make your API private or public here as well, by just clicking on "Lock Icon".
- If you want to delete any version, just click on "Delete Icon".

- Under the name of created API, there will be any options like we can fork an API, we can change the owner, one can delete the API and rename the API as well.

JFROG ARTIFACTORY

Artifactory: Artifactory is a popular tool used in the software development lifecycle, particularly in the context of continuous integration, continuous delivery, and DevOps. It is a repository manager that helps manage and organize binary artifacts, such as libraries, dependencies, Docker images, and other build artifacts.

Github VS Jfrog: JFrog and GitHub are two different tools that serve distinct purposes in the software development lifecycle. While they can complement each other, they have different primary functionalities:

1**. JFrog Artifactory:**

JFrog Artifactory is a binary repository manager. It focuses on managing and organizing binary artifacts generated during the software development and build process. Artifactory supports various package formats, including Maven, Gradle, npm, Docker, NuGet, PyPI, and more. It acts as a centralized repository to store and retrieve these artifacts, enabling efficient caching, versioning, and access control.

Key features of JFrog Artifactory:

- Artifact repository management and versioning.

- Caching and proxying remote repositories to improve download speeds.

- Support for different package formats and build tools.

- Fine-grained access control and security.

- Integration with build tools and CI/CD pipelines for artifact deployment.

2. GitHub:

GitHub is a web-based hosting service primarily used for version control and collaborative software development. It is based on the distributed version control system Git and provides a platform for developers to store, manage, and collaborate on source code.

Key features of GitHub:

- Version control and code repository hosting using Git.

- Pull requests and code review workflows for collaboration.

- Issue tracking and project management capabilities.

- Continuous Integration (CI) and Continuous Deployment (CD) workflows through GitHub Actions.

- Support for documentation and wikis.

- Community engagement and social coding features.

In summary, JFrog Artifactory is focused on managing binary artifacts, while GitHub is primarily a version control and collaborative development platform. Developers often use both tools in combination, where GitHub is used for source code management, collaborative development, and CI/CD workflows, while JFrog Artifactory is used to manage and store the resulting binary artifacts generated during the development process.

Types of Repositories in Jfrog Artifactory:

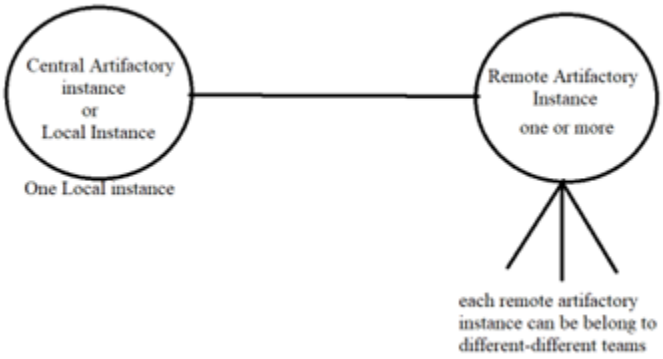
1. Federated Repo

In JFrog Artifactory, a federated repository (also known as a "Federated Repository Instance") is a feature that allows you to connect multiple Artifactory instances together in a distributed and interconnected manner. It enables you to create a network of repositories across different geographical locations or teams, providing a unified view and access to artifacts stored in these interconnected instances.

With federated repositories, you can have a central Artifactory instance (referred to as the "local instance") that is connected to one or more remote Artifactory instances (referred to as "edge nodes" or "remote instances"). Each remote Artifactory instance can belong to different teams or be in different locations, and they may have their own repositories and artifacts.

For Example: you have two big boxes of toys—one at your home and one at your friend's house. Now, wouldn't it be cool if you and your friend could easily share toys from both boxes without physically moving them back and forth?

In JFrog Artifactory, a federated repository is like a magical link between those toy boxes. It allows different Artifactory instances (think of them as the toy boxes) to connect and share their toys (artifacts) with each other.



2. Remote Repo:

Remote repository (also known as a "Remote Repository Instance") is a special type of repository that is used to proxy and cache artifacts from external sources. It acts as a bridge between Artifactory and external repositories located on the internet or other remote servers.

For Example: Imagine you have a special helper (let's call them Arti) who can go to a big store far away and bring you all the toys you want. Instead of going to the store yourself, you just ask Arti, and they bring the toys to your home.

In JFrog Artifactory, a remote repository is like Arti, your helpful assistant. It acts as a helper that goes to a faraway place on the internet (an external repository) to get special things called "artifacts." These artifacts are like toys for developers—important pieces of code, software, and libraries they need for their projects.

3. Virtual Repo:

In JFrog Artifactory, a virtual repository (also known as a "Virtual Repository Instance") is a special type of repository that acts as a unified view of multiple other repositories. It provides a way to aggregate and access artifacts from one or more local and remote repositories under a single URL.

Imagine you have several different toy boxes in your house—one for Lego toys, another for action figures, and one more for board games. It would be convenient to have a magic box that combines all these toy boxes into one, making it easy to find and play with any toy you want, without having to search through each box separately.

4. Local Repo:

In JFrog Artifactory, a local repository (also known as a "Local Repository Instance") is a type of repository that stores artifacts directly on the Artifactory server. It acts as a local storage location for the artifacts generated or used within a specific organization or development team.

Imagine you have a special box in your room where you keep all your favourite toys. This box is your personal collection, and you can easily access and organize your toys whenever you want.