# Terraform

Terraform is HashiCorp's infrastructure as code tool. It lets you define resources and infrastructure in human-readable, declarative configuration files, and manages your infrastructure's lifecycle. Using Terraform has several advantages over manually managing your infrastructure:

* Terraform can manage infrastructure on multiple cloud platforms.
* The human-readable configuration language helps you write infrastructure code quickly.
* Terraform's state allows you to track resource changes throughout your deployments.
* You can commit your configurations to version control to safely collaborate on infrastructure.

Terraform plugins called providers let Terraform interact with cloud platforms and other services via their application programming interfaces (APIs). HashiCorp and the Terraform community have written over 1,000 providers to manage resources on Amazon Web Services (AWS), Azure, Google Cloud Platform (GCP), Kubernetes, Helm, GitHub, Splunk, and DataDog, just to name a few. Find providers for many of the platforms and services you already use in the Terraform Registry. If you don't find the provider you're looking for, you can write your own.

To deploy infrastructure with Terraform:

* Scope - Identify the infrastructure for your project.
* Author - Write the configuration for your infrastructure.
* Initialize - Install the plugins Terraform needs to manage the infrastructure.
* Plan - Preview the changes Terraform will make to match your configuration.
* Apply - Make the planned changes.

Terraform allows you to collaborate on your infrastructure with its remote state backends. When you use HCP Terraform (free for up to five users), you can securely share your state with your teammates, provide a stable environment for Terraform to run in, and prevent race conditions when multiple people make configuration changes at once.

You can also connect HCP Terraform to version control systems (VCSs) like GitHub, GitLab, and others, allowing it to automatically propose infrastructure changes when you commit configuration changes to VCS. This lets you manage changes to your infrastructure through version control, as you would with application code.

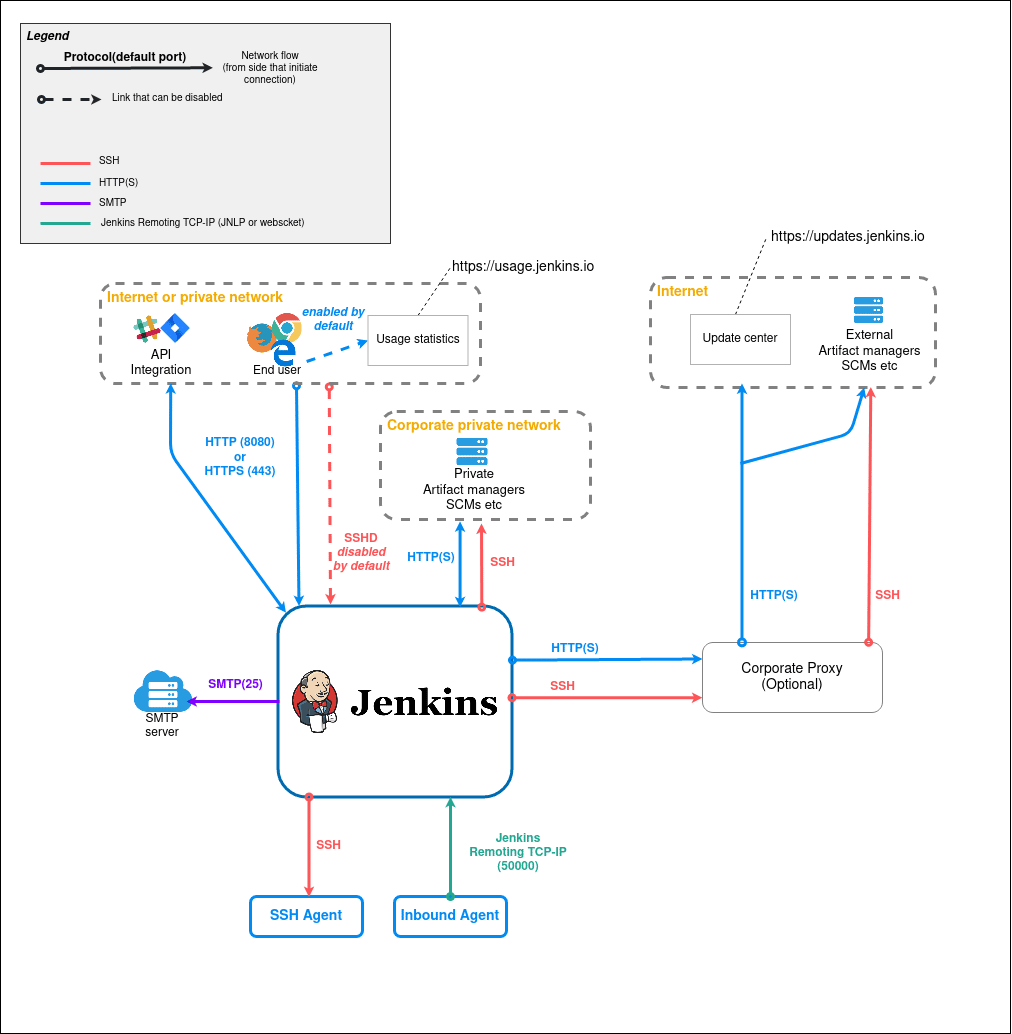
# Jenkins

Jenkins is an open source continuous integration (CI) server. It manages and controls several stages of the software delivery process, including build, documentation, automated testing, packaging, and static code analysis. Jenkins is a highly popular DevOps tool used by thousands of development teams.

Jenkins automation is commonly triggered by code changes in repositories like GitHub, Bitbucket, and GitLab, and integrates with build tools like Maven and Gradle. Jenkins supports the use of container technologies like Docker and Kubernetes for testing and packaging of software releases, but it is neither a Kubernetes native solution nor a container-native CI solution.

Jenkins elements interact and work together in the following way:

* Developers modify the source code, committing changes to the repository, and Jenkins creates a new build in order to handle the new Git commit.
* Jenkins can work in “push” or “pull” mode. The Jenkins CI server is either triggered by an event such as a code commit, or it can regularly check the repository for changes.
* The build server builds the code and generates an artifact. If the build fails, the developer receives an alert.
* Jenkins deploys the built application/executable to the test server, which can execute continuous, automated tests. Developers receive alerts if their changes impact functionality.
* Jenkins optionally deploys the changes to the production server if the code has no issues.



## Jenkins Advantages and Disadvantages

* Here are some of the key advantages of Jenkins:
* Highly extensible with a huge variety of existing plugins. Plugins contribute to Jenkins’ flexibility and rich scripting and declarative language which supports advanced, custom pipelines.
* Robust and reliable at almost any scale.
* Mature and battle-tested.
* Supports hybrid and multi-cloud environments.
* Offers an extensive knowledge base, documentation, and community resources.
* Based on Java, an enterprise development language with a broad ecosystem, making it suitable for legacy enterprise environments.
* Here are some disadvantages of Jenkins:
* **Single server architecture**—uses a single server architecture, which limits resources to resources on a single computer, virtual machine, or container. Jenkins doesn’t allow server-to-server federation, which can cause performance issues in large-scale environments.
* **Jenkins sprawl—**this is a common problem which also stems from lack of federation. Multiple teams using Jenkins can create a large number of standalone Jenkins servers that are difficult to manage.
* **Relies on dated Java architectures and technologies**—specifically Servlet and Maven. In general, Jenkins uses a monolithic architecture and is not designed for newer Java technologies such as Spring Boot or GraalVM.
* **Not container native**—Jenkins was designed in an era before containers and Kubernetes gained popularity, and while it supports container technology, it does not have nuanced support for container and orchestration mechanisms.
* **Difficult to implement in production environments—**developing continuous delivery pipelines with Jenkinsfiles requires coding in a declarative or scripting language, and complex pipelines can be difficult to code, debug, and maintain.
* **Offers no functionality for real production deployments**—“deploying with Jenkins” means running a fully customized set of scripts to handle the deployment.
* **Jenkins itself requires deployment**—this can be difficult to automate. Organizations that need to combine Jenkins with a continuous delivery solution have traditionally used configuration management to do this, but this adds another layer of complexity and is error-prone.
* **Complicated plugin management**—Jenkins has nearly 2,000 plugins, which can be overwhelming to sort through until you find a useful plugin. Many plugins also have dependencies that increase the management burden, while some plugins may conflict with each other. There is no guarantee a plugin you use will continue to be maintained.
* **Groovy expertise requirements**—Jenkins has programmatic pipelines implemented in Groovy, a language that is currently not in wide use and can make scripts difficult to work with. Jenkins supports scripted and declarative Groovy modes.