

Plugins & Versioning

In Jenkins, plugins and versioning are crucial for extending functionality and managing updates.

Here's how they work:

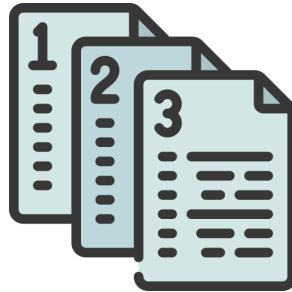
Plugins in Jenkins



Jenkins is highly modular, and its core functionalities can be extended via plugins. Plugins allow Jenkins to integrate with a wide variety of tools and platforms like Git, Docker, Kubernetes, and more.

- **Plugin Manager:** Jenkins comes with a Plugin Manager that allows you to search, install, update, and remove plugins.
- **Popular Plugins:**
 - **Git Plugin:** Integrates Jenkins with Git repositories.
 - **Pipeline Plugin:** Defines complex build pipelines using a scripted or declarative DSL.
 - **Docker Plugin:** Builds, runs, and manages Docker containers.
 - **Blue Ocean:** Provides a modern, user-friendly UI for Jenkins pipelines.
- **Installation:** Plugins can be installed via the Jenkins UI or uploaded manually. It's important to keep them updated to the latest versions for security and stability reasons.

Versioning in Jenkins



Versioning in Jenkins primarily refers to tracking changes in jobs, configurations, and code.

- **Job DSL Plugin:** Allows you to version control Jenkins jobs as code. You define jobs and pipelines in a DSL, store it in your version control system, and Jenkins can recreate or update jobs based on this DSL.
- **Pipeline-as-Code (Jenkinsfile):** Jenkins supports versioning of pipeline definitions through Jenkinsfile. This file is placed in your repository and defines the build steps for Jenkins to execute. It's version-controlled alongside your codebase.
- **Jenkins Core Versioning:** Jenkins itself has a core version, and updating Jenkins ensures you get the latest security patches, new features, and performance improvements. You can configure Jenkins to notify you of updates or update manually.

Workspace in Jenkins

The workspace in Jenkins is the directory on the Jenkins server where Jenkins stores the files related to a job.

- **Purpose of Workspace:** When Jenkins triggers a job, it clones the code repository (e.g., Git or SVN) and places it in the workspace. All subsequent build steps operate on this workspace directory.
- **Location:** Workspaces are located under the `$JENKINS_HOME/workspace/` directory on the Jenkins server, with each job having its own subdirectory.
- **Workspace Output:**
 - **Compiled Artifacts:** In build jobs (e.g., Java Maven builds), the workspace contains compiled files like `.jar` or `.war` files.
 - **Logs:** Build logs are generated in the workspace and include information about each step in the build.
 - **Reports:** Test reports (JUnit, coverage, etc.) are usually generated in the workspace.

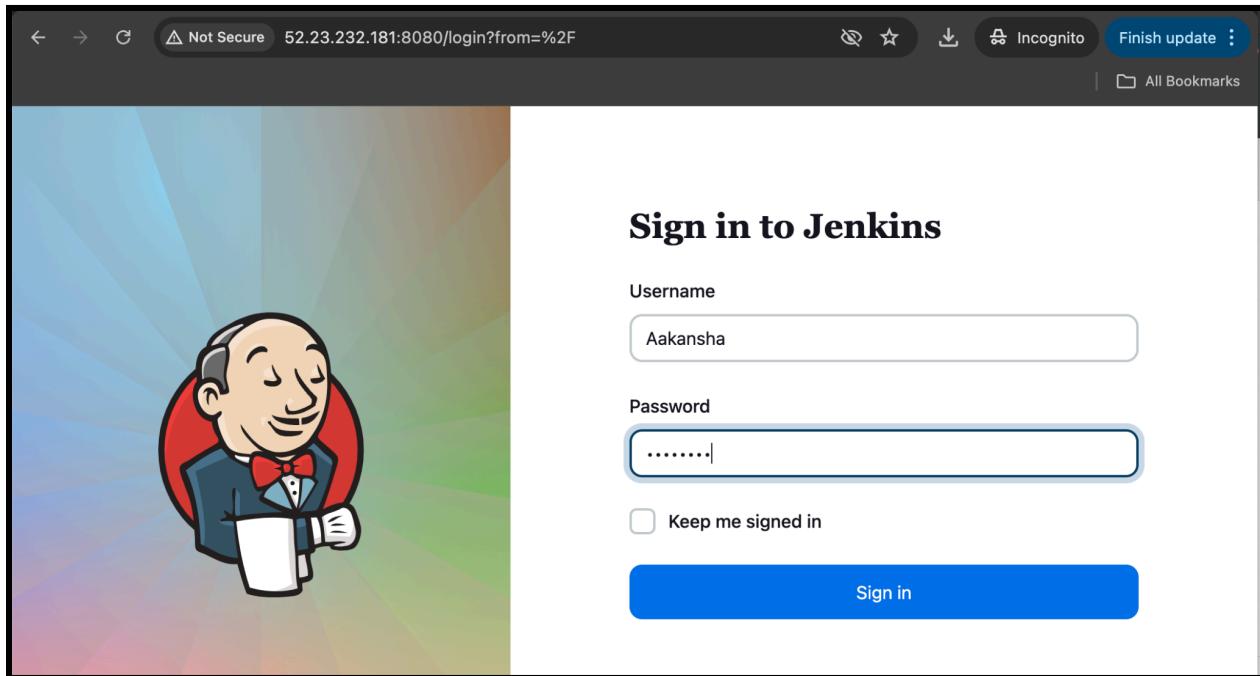
Managing Workspace

- **Clean Workspace:** Often, after a build, the workspace needs to be cleaned to avoid leftover files from previous builds. Plugins like Workspace Cleanup can help automate this process.
- **Disk Space:** Large workspaces can consume a lot of disk space. It's important to monitor and clean them periodically.





Practice Time



Let's talk about the workspace and the output being generated in the workspace.

We can see our artifact in the target directory at the workspace.

The screenshot shows the Jenkins interface for a build workspace. On the left, there's a sidebar with links like Status, Changes, Workspace, Build Now, Configure, Delete Project, and Rename. Below that is a 'Build History' section showing two builds: #4 (Sep 26, 2024, 1:33 PM) and #3 (Sep 25, 2024, 12:47 PM). The main area is titled 'Workspace of Build on Built-In Node' and shows a tree view of the workspace contents under 'Build / target /'. The contents include: classes, generated-sources/annotations, generated-test-sources/test-annotations, maven-archiver, maven-status/maven-compiler-plugin, site/jacoco, surefire-reports, test-classes/com/visualpathit/account, vprofile-v2, jacoco.exec, and vprofile-v2.war. There are download links for each file.

- Now everytime we will be running the job this artifact will be replaced by the newer artifact and we will be running the job if there will be any code change.
- Let's say we deploy this artifact and it works fine. Then there is a new code change and we run the build one more time.
- So now the new artifact will replace the old artifact and we will deploy the new artifact to the server and things break 😞
- Now we want to rollback to the previous artifact or we want the previous artifact.
- But we can't get it now as it is overwritten.
- So to preserve the artifact we can do versioning.
- Let's start with some simple versioning by executing shell commands and install required plugins for versioning.
- The target directory is volatile, if someone runs mvn clean everything will be gone from there.

Now let's create another job for versioning.

Dashboard > All > New Item

New Item

Enter an item name

Select an item type

-  **Freestyle project**
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.
-  **Pipeline**
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
-  **Multi-configuration project**
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.
-  **Folder**

OK

Below this copy everything from the previous job build and click ok.

Dashboard > All > New Item

-  Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.
-  **Folder**
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.
-  **Multibranch Pipeline**
Creates a set of Pipeline projects according to detected branches in one SCM repository.
-  **Organization Folder**
Creates a set of multibranch project subfolders by scanning for repositories.

If you want to create a new item from other existing, you can use this option:

Copy from

OK

Go down and remove the archive of the artifact because it will start storing this artifact into different- different directories with different- different build numbers which will consume the disk space.

The screenshot shows the Jenkins 'Configuration' screen for a job named 'Versioning-Builds'. On the left, there's a sidebar with links: General, Source Code Management, Build Triggers, Build Environment, Build Steps, and Post-build Actions. 'Post-build Actions' is currently selected and highlighted with a blue background. In the main area, under 'Post-build Actions', there is one entry: 'Archive the artifacts'. This entry has a 'Delete' button with a red 'X' icon. Below it, there's a 'Files to archive' field containing the pattern '**/*.war'. There are also 'Advanced' and 'Add post-build action' buttons. At the bottom of the screen are 'Save' and 'Apply' buttons.

Save and run this job

Now once the job runs successfully we must be having the artifact in our workspace.

The screenshot shows the Jenkins 'Workspace' screen for the 'Versioning-Builds' job. On the left, there's a sidebar with links: Status, Changes, Workspace, Configure, Delete Project, and Rename. 'Workspace' is currently selected and highlighted with a blue background. In the main area, there's a title 'Workspace of Versioning-Builds on Built-In Node'. Below it, there's a 'Versioning-Builds /' breadcrumb and a search bar. The workspace directory structure is shown: .git, ansible, src, target, vagrant, Jenkinsfile, pom.xml, and README.md. Each file has its last modified date, size, and an 'eye' icon. At the bottom right, there's a link '(all files in zip)'. At the bottom left, there's a 'Build History' section with a 'trend' dropdown, a 'Filter...' input, and a list of builds: '#1 (Sep 26, 2024, 1:46 PM)'. At the very bottom, there are 'Atom feed for all' and 'Atom feed for failures' links.

Configure the job and come down to the build section. Add build step and select execute shell.

The screenshot shows the Jenkins 'Configuration' page for a job named 'MAVEN3'. The left sidebar lists 'General', 'Source Code Management', 'Build Triggers', 'Build Environment', 'Build Steps' (which is selected and highlighted in grey), and 'Post-build Actions'. The main area shows 'Goals' set to 'install'. A 'Execute shell' step is expanded, showing the command:

```
mkdir -p versions  
cp target/vprofile-v2.war versions/vprofile-$BUILD_ID
```

Below the command, there is a link to 'See the list of available environment variables'. At the bottom of the step, there is an 'Advanced' dropdown menu. At the very bottom of the configuration page are 'Save' and 'Apply' buttons.

→ These commands create a directory to store different versions of the build artifact and copy the .war file into it, with a name that includes the build ID for easy identification.

Commands Explanation

1. **mkdir -p versions:**
 - This command creates a directory named `versions`.
 - The `-p` option ensures that the directory is created only if it doesn't already exist. It also creates any necessary parent directories.
2. **cp target/vprofile-v2.war versions/vprofile-\$BUILD_ID:**
 - This copies the file `vprofile-v2.war` from the `target/` directory (usually where the build artifacts are generated, like after a Maven build) into the `versions/` directory.
 - The destination file is renamed to `vprofile-$BUILD_ID`, where `$BUILD_ID` is an environment variable in Jenkins that holds the unique identifier of the current build. This ensures each build artifact is stored with a version number tied to the specific build, allowing for versioning and easier tracking of different build outputs.

❖ Check the link for Jenkins Set Environment Variables

<https://wiki.jenkins.io/display/JENKINS/Building+a+software+project>

Now run the build a few times.

Dashboard > Versioning-Builds >

 Status

 Changes

 Workspace

 Build Now

 Configure

 Delete Project

 Rename

 **Versioning-Builds**

Build artifact from vprofile source code

Permalinks

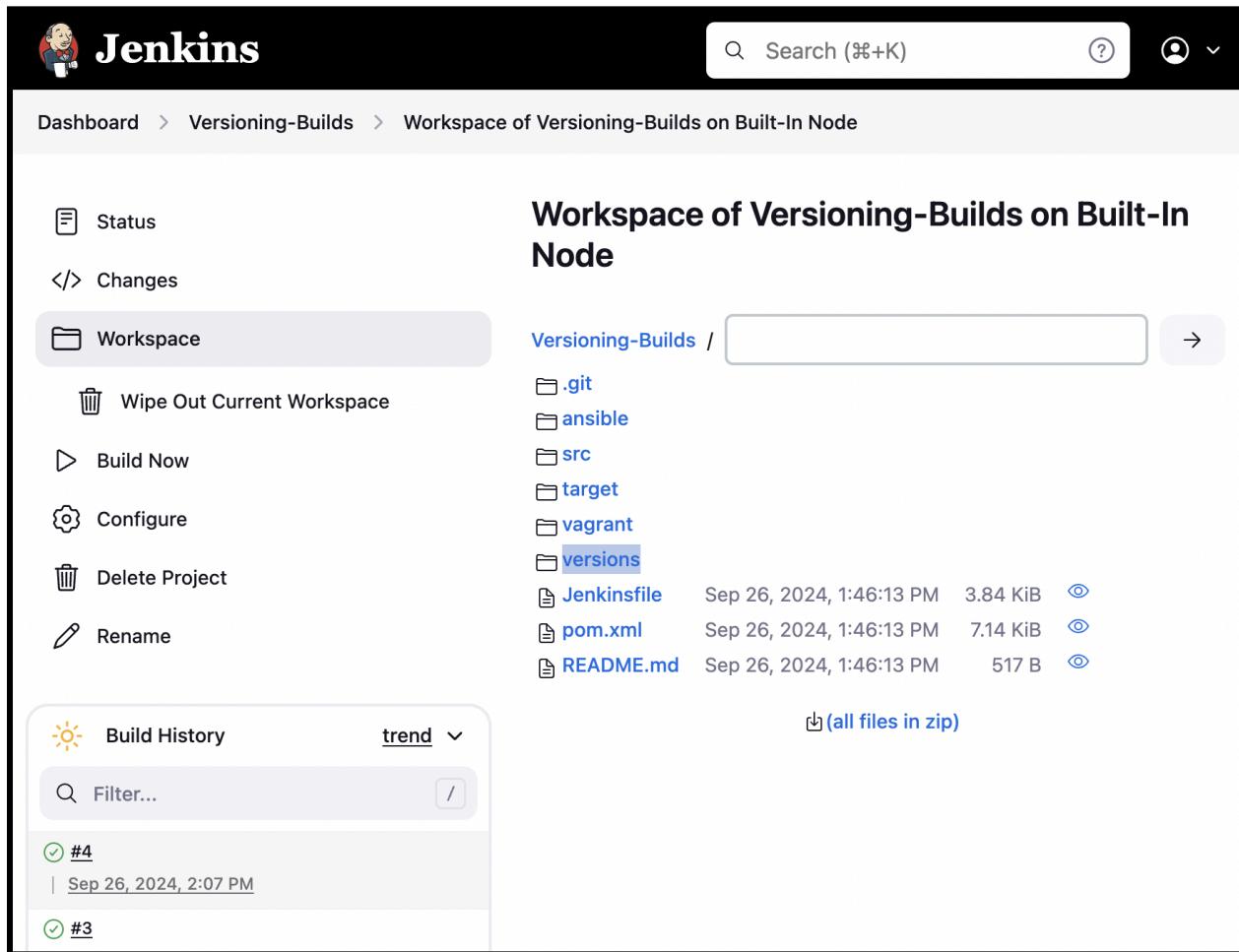
- [Last build \(#1\), 13 min ago](#)
- [Last stable build \(#1\), 13 min ago](#)
- [Last successful build \(#1\), 13 min ago](#)
- [Last completed build \(#1\), 13 min ago](#)

 **Build History**

 Filter... /

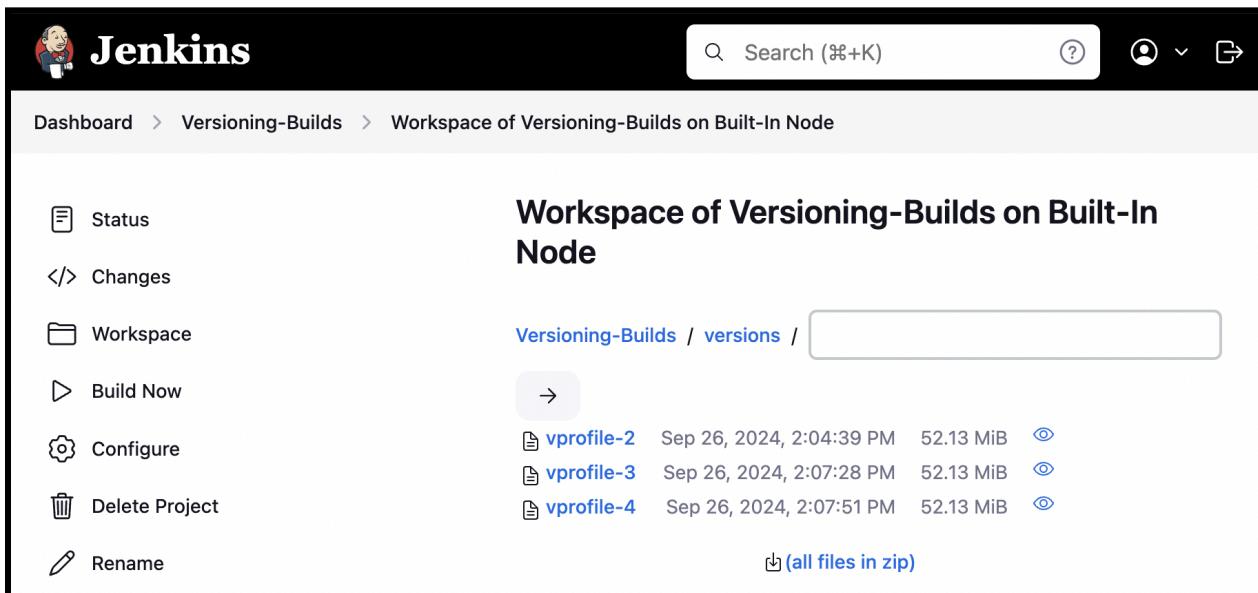
#	Date
#4	Sep 26, 2024, 2:07 PM
#3	Sep 26, 2024, 2:07 PM
#2	Sep 26, 2024, 2:04 PM
#1	Sep 26, 2024, 1:46 PM

Now let's check the workspace. We can see the versions directory.



The screenshot shows the Jenkins interface for a project named "Versioning-Builds". The left sidebar has options like Status, Changes, Workspace (which is selected), Build Now, Configure, Delete Project, and Rename. The main content area is titled "Workspace of Versioning-Builds on Built-In Node". It shows a file tree under "Versioning-Builds /": .git, ansible, src, target, vagrant, and versions. The "versions" directory is highlighted. Inside "versions", there are three files: Jenkinsfile, pom.xml, and README.md, all updated on Sep 26, 2024, at 1:46:13 PM. A "Build History" section shows two builds: #4 (Sep 26, 2024, 2:07 PM) and #3. A "trend" dropdown is set to "trend". A "Filter..." input field is present. A "ZIP" button is available to download all files.

We can see the different versions of the artifact.



This screenshot shows the same Jenkins workspace as above, but the "versions" directory is expanded. The file tree under "Versioning-Builds / versions /" shows four files: vprofile-2, vprofile-3, vprofile-4, and a ".git" folder. All four files were last modified on Sep 26, 2024, at different times between 2:04:39 PM and 2:07:51 PM, and each is 52.13 MiB in size. A "Build History" section is also visible on the left.

Can we use some other versioning format?

What if I want to specify my own, like a user argument.

❖ Let's Try!

Go to configure.. Jenkins job has one option here in the general: This project is parameterised.

The screenshot shows the Jenkins General configuration page. At the top, there is a breadcrumb navigation: Dashboard > Versioning-Builds > Configuration. On the left, a sidebar titled "Configure" lists several sections: General (selected), Source Code Management, Build Triggers, Build Environment, Build Steps, and Post-build Actions. The "General" section contains a "Description" field with the text "Build artifact from vprofile source code". Below the description is a "Plain text" link and a "Preview" link. Under the "General" tab, there are several checkboxes: "Discard old builds", "GitHub project", "This project is parameterized" (which is checked), "Throttle builds", and "Execute concurrent builds if necessary". At the bottom of the page are "Save" and "Apply" buttons.

→ This will be taking the user input.

Add parameters. Let's take the string parameter.

The screenshot shows the Jenkins 'Configure' screen for a GitHub project. On the left, there is a sidebar with options like General, Source Code Management, Build Triggers, Build Environment, Build Steps, and Post-build Actions. The 'General' tab is selected. On the right, there is a section titled 'This project is parameterized' with a checked checkbox. Below it is a dropdown menu titled 'Add Parameter' with a filter field. The menu lists several parameter types: Boolean Parameter, Choice Parameter, Credentials Parameter, File Parameter, Multi-line String Parameter, Password Parameter, Run Parameter, and String Parameter. The 'String Parameter' option is highlighted with a light blue background.

Now when the user enters the value it's going to store it in this variable and we can use it in our job.

The screenshot shows the same Jenkins 'Configure' screen as before, but now with a new 'String Parameter' added. The 'String Parameter' configuration card is open, showing fields for 'Name' (set to 'VERSION'), 'Default Value' (empty), 'Description' (empty), and 'Plain text Preview' (empty). There is also a checkbox for 'Trim the string'. At the bottom of the card is a 'Save' button.

Let's use our variable and save it.

The screenshot shows a Jenkins job configuration step titled "Execute shell". The command entered is:

```
mkdir -p versions
# cp target/vprofile-v2.war versions/vprofile-$BUILD_ID
cp target/vprofile-v2.war versions/vprofile-$VERSION
```

Below the command, there is an "Advanced" dropdown button.

Now let's test it.

The screenshot shows the Jenkins Project Versioning-Builds page. The "Build with Parameters" button is highlighted. A parameter named "VERSION" is set to "2.0". The "Build" button is green and labeled "Build". The "Cancel" button is grey. On the left, there is a sidebar with options like Status, Changes, Workspace, Build with Parameters (which is selected), Configure, Delete Project, and Rename. Below the sidebar is a "Build History" section showing builds #4, #3, and #2, each with a timestamp of "Sep 26, 2024, 2:07 PM".

Now let's see the output.

The screenshot shows the Jenkins interface for a project named "Versioning-Builds". The left sidebar contains links for Status, Changes, Workspace, Build with Parameters, Configure, Delete Project, and Rename. The main area is titled "Workspace of Versioning-Builds on Built-In Node". It displays a list of build artifacts: vprofile-2, vprofile-2.0, vprofile-3, and vprofile-4. Each artifact has a timestamp (Sep 26, 2024), size (52.13 MiB), and a "View" link. Below the artifacts is a link "(all files in zip)". To the left, there is a "Build History" section with a "trend" dropdown, a "Filter..." search bar, and a list of builds: #5 (Sep 26, 2024, 2:20 PM).

Here it is 🎉

Question Time

Why is it important to avoid making Jenkins jobs interactive, and how does it impact the automation process in a CI/CD pipeline?

Avoiding interactive jobs in Jenkins is generally recommended for several key reasons:

1. Automation and Unattended Execution

- Jenkins jobs are meant to run in an automated, continuous integration/continuous delivery (CI/CD) environment without manual intervention. Interactive jobs that require user input (like prompting for confirmation or asking for credentials) break the automation flow, requiring someone to monitor and respond, which defeats the purpose of CI/CD.

2. Consistency and Reliability

- Non-interactive jobs ensure consistent, repeatable builds and deployments. If user interaction is needed, different responses or mistakes during interaction can cause inconsistent results. Removing human input ensures that jobs run the same way every time.

3. Scalability

- CI/CD environments often need to scale, running many jobs in parallel across multiple nodes or in distributed systems. Interactive jobs can block the execution of other jobs, limiting scalability, especially when jobs need to be manually handled.

4. Pipeline Automation

- Jenkins pipelines are designed to be triggered automatically by events (like commits to a repository) or scheduled to run at specific times. Interactive jobs stop the pipeline and delay the entire process, making it inefficient for continuous integration.

5. Error Handling and Monitoring

- Non-interactive jobs are easier to monitor, log, and manage errors. Interactive prompts make it harder to track automated failures and troubleshoot issues, especially when running jobs remotely or at scale.

6. Integration with Other Tools

- Interactive jobs hinder integration with external tools and systems, such as version control, artifact repositories, or deployment services, which expect automated, non-blocking job execution.

By keeping jobs non-interactive, Jenkins can operate continuously and autonomously, leading to more reliable, scalable, and efficient pipelines.

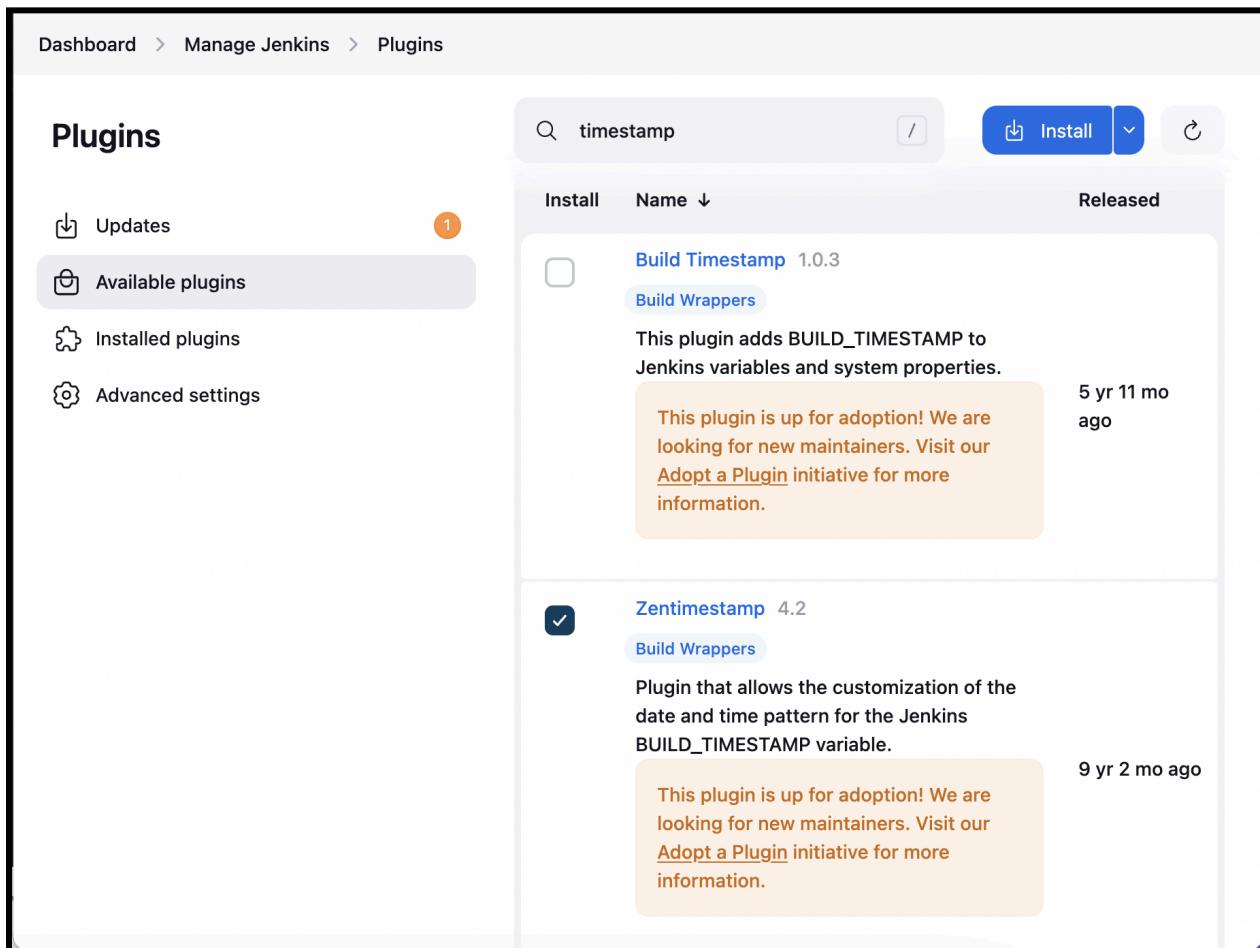
Application of plugins

Now let's move to a better way of versioning using plugins.

Go to Dashboard > Manage Jenkins > Plugins

The screenshot shows the Jenkins Manage Jenkins interface. On the left, there's a sidebar with links like 'New Item', 'Build History', 'Manage Jenkins' (which is highlighted), and 'My Views'. Below that are sections for 'Build Queue' (empty) and 'Build Executor Status' (1 Idle, 2 Idle). The main area is titled 'Manage Jenkins' and contains a 'System Configuration' section with icons for 'System', 'Tools', 'Nodes', 'Clouds', and 'Appearance'. A prominent 'Plugins' section is shown, featuring a gear icon with a red notification dot containing the number '1'. It describes how to add, remove, disable or enable plugins to extend Jenkins' functionality. The URL '52.23.232.181:8080/manage/pluginManager' is visible at the bottom of the browser window.

Let's look for a plugin that gives us a timestamp.



The screenshot shows the Jenkins 'Manage Jenkins > Plugins' interface. A search bar at the top contains the text 'timestamp'. Below it, a table lists two plugins:

Install	Name ↓	Released
<input type="checkbox"/>	Build Timestamp 1.0.3 Build Wrappers This plugin adds BUILD_TIMESTAMP to Jenkins variables and system properties. This plugin is up for adoption! We are looking for new maintainers. Visit our Adopt a Plugin initiative for more information.	5 yr 11 mo ago
<input checked="" type="checkbox"/>	Zentimestamp 4.2 Build Wrappers Plugin that allows the customization of the date and time pattern for the Jenkins BUILD_TIMESTAMP variable. This plugin is up for adoption! We are looking for new maintainers. Visit our Adopt a Plugin initiative for more information.	9 yr 2 mo ago

Click on install and wait for some time.



Please wait while Jenkins is restarting ...

Your browser will reload automatically when Jenkins is ready.

Safe Restart

Builds on agents can usually continue.

The download is in progress.

This screenshot shows the Jenkins 'Plugins' page under 'Manage Jenkins'. The left sidebar has links for 'Updates', 'Available plugins', 'Installed plugins' (which is selected), and 'Advanced settings'. The right panel is titled 'Download progress' and contains two sections: 'Go back to the top page' (with a note about starting use) and 'Restart Jenkins when installation is complete and no jobs are running' (with a checkbox). A large orange progress bar at the bottom indicates the download is in progress.

Check the installed plugins.

This screenshot shows the Jenkins 'Plugins' page under 'Manage Jenkins'. The left sidebar has links for 'Updates', 'Available plugins', 'Installed plugins' (selected), and 'Advanced settings'. The right panel lists installed plugins with a search bar at the top. It shows the 'Timestamper' plugin (version 1.27) is enabled, and the 'Zentimestamp plugin' (version 4.2) is up for adoption. Both have checkboxes and red 'x' icons.

Name	Enabled
Timestamper 1.27	<input checked="" type="checkbox"/> x
Zentimestamp plugin 4.2	<input checked="" type="checkbox"/> x

Go back to the versioning job and click configure.

The screenshot shows the Jenkins configuration interface for a 'Versioning-Builds' job. The left sidebar lists configuration sections: General (selected), Source Code Management, Build Triggers, Build Environment, Build Steps, and Post-build Actions. The 'General' section contains a 'Description' field with the text 'Build artifact from vprofile source code'. A 'Date and Time Pattern' field is set to 'yy-MM-dd_HHMM'. A note below it says: 'You must specify a `java.text.SimpleDateFormat` pattern. For example give the following value: yyyyMMddHHmmss.' (from Jenkins Zentimestamp plugin). There are also three unchecked checkboxes: 'Discard old builds', 'GitHub project', and 'This project is parameterized'. At the bottom are 'Save' and 'Apply' buttons.

Add the variable \$BUILD_TIMESTAMP. Save and build now.

The screenshot shows the Jenkins build steps page. It displays an 'Execute shell' step with the command:

```
mkdir -p versions
# cp target/vprofile-v2.war versions/vprofile-$BUILD_ID
# cp target/vprofile-v2.war versions/vprofile-$VERSION
cp target/vprofile-v2.war versions/vprofile-$BUILD_TIMESTAMP
```

. Below the command is an 'Advanced' dropdown and an 'Add build step' button.

Let's check the workspace now.

The screenshot shows the Jenkins interface for the 'Versioning-Builds' project. The left sidebar contains links for Status, Changes, Workspace, Build Now, Configure, Delete Project, and Rename. The main content area is titled 'Workspace of Versioning-Builds on Built-In Node'. It shows a list of build artifacts: vprofile-2, vprofile-2.0, vprofile-24-09-26_1442 (selected), vprofile-3, and vprofile-4. Below the list is a link '(all files in zip)'. On the left, there is a 'Build History' section showing builds #6 and #5. Build #6 was run on Sep 26, 2024, at 2:42 PM. Build #5 was run on Sep 26, 2024, at 2:20 PM.

Artifact	Created	Size	Action
vprofile-2	Sep 26, 2024, 2:04:39 PM	52.13 MiB	🔗
vprofile-2.0	Sep 26, 2024, 2:20:52 PM	52.13 MiB	🔗
vprofile-24-09-26_1442	Sep 26, 2024, 2:42:45 PM	52.13 MiB	🔗
vprofile-3	Sep 26, 2024, 2:07:28 PM	52.13 MiB	🔗
vprofile-4	Sep 26, 2024, 2:07:51 PM	52.13 MiB	🔗

So it's looking better now. We can give more types of format.

As Jenkins is primarily a continuous integration server and not meant for long-term storage, it's important to transfer these artifacts to a more suitable location to avoid consuming excessive disk space. Congratulations on reaching this stage, and keep up the great work as you move forward!

