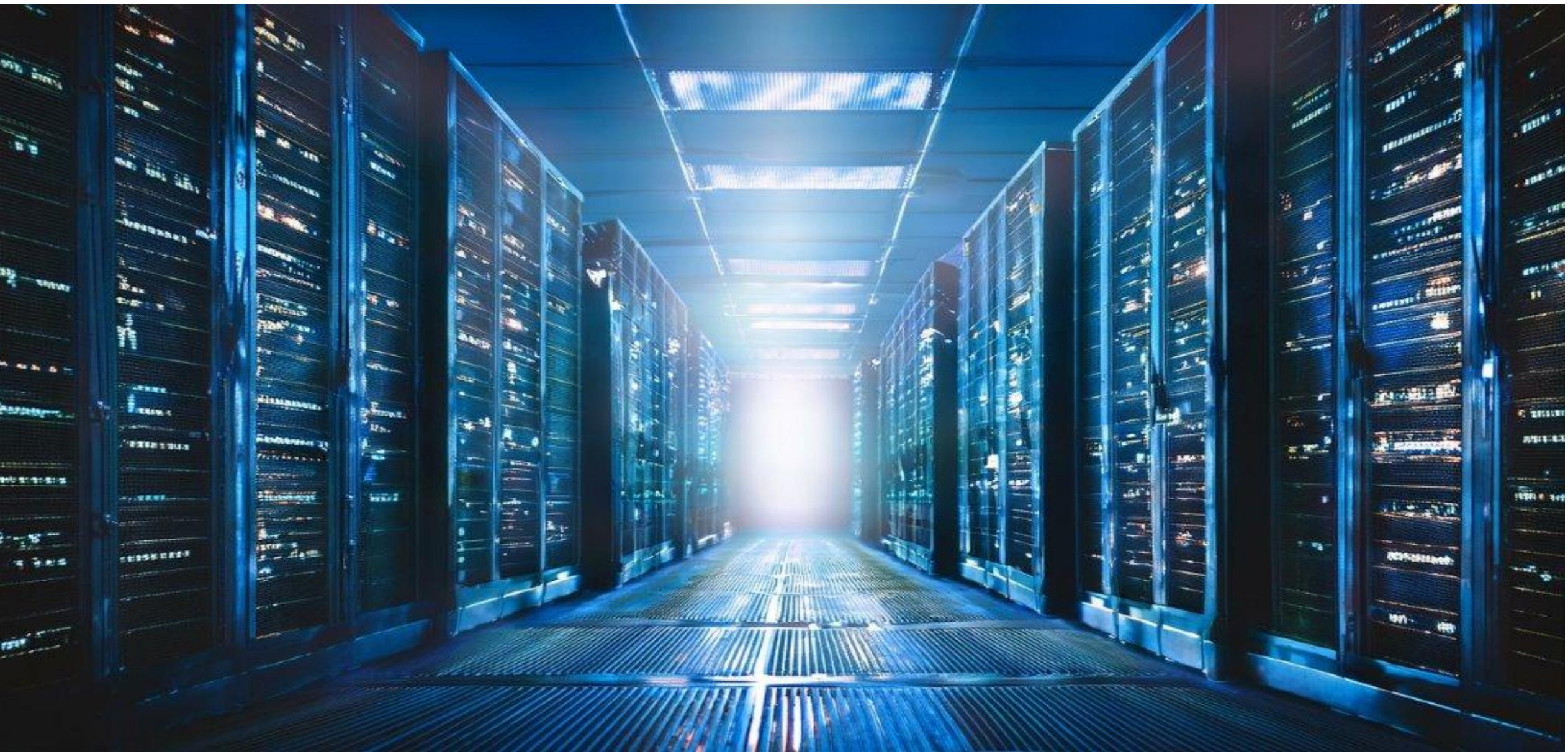


Infrastructure Automation Tools





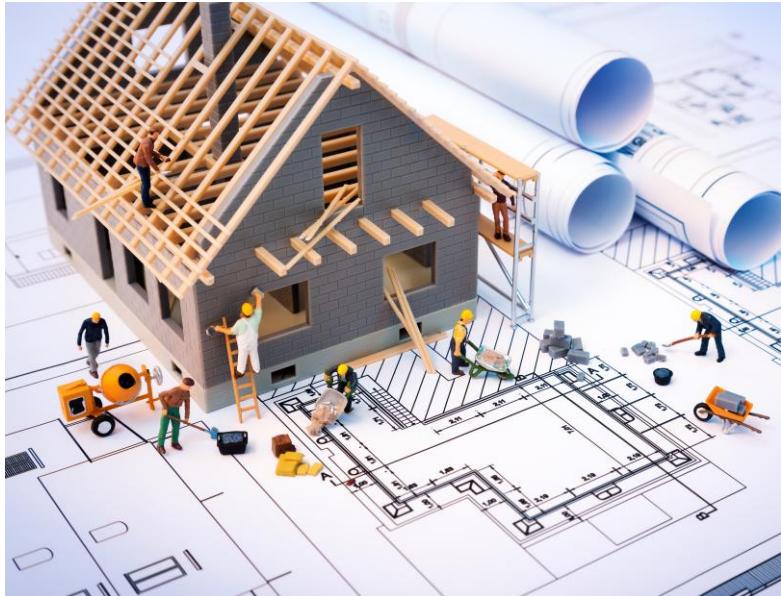
WORKFORCE DEVELOPMENT



CI/CD and Infra Automation



Learning Objectives



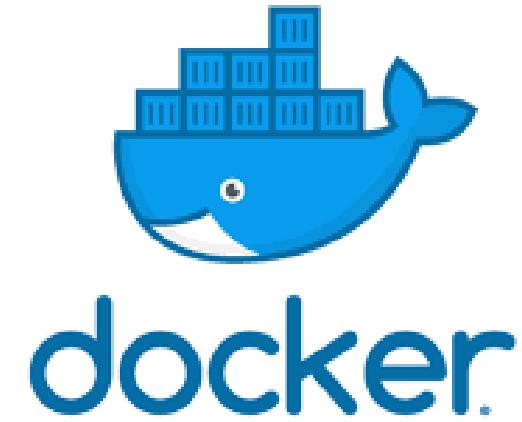
Day 2 Objectives

- Build and push Docker images using Jenkins pipelines
- Manually deploy a containerized application on a virtual machine
- Automate deployment using Ansible with an inventory file and playbook

Jenkins



Automation tools



Jenkins



Jenkins is an open-source automation server used to build, test, and deploy software. It runs pipelines that allow teams to automate repetitive tasks and enforce consistent workflows.

Key points

- One of the most widely used CI/CD tools
- Uses jobs and pipelines to automate builds and deployments
- Highly extensible through plugins
- Can run on your own servers for full control

Jenkins Controller and Agents



Jenkins uses a controller–agent architecture to run pipelines efficiently.

- **Controller** manages Jenkins: schedules jobs, provides the UI, stores configurations
- **Agents** are worker machines that execute the actual build and deployment steps
- Agents can be Linux, Windows, or containers depending on your needs
- This model allows Jenkins to scale and run multiple jobs in parallel

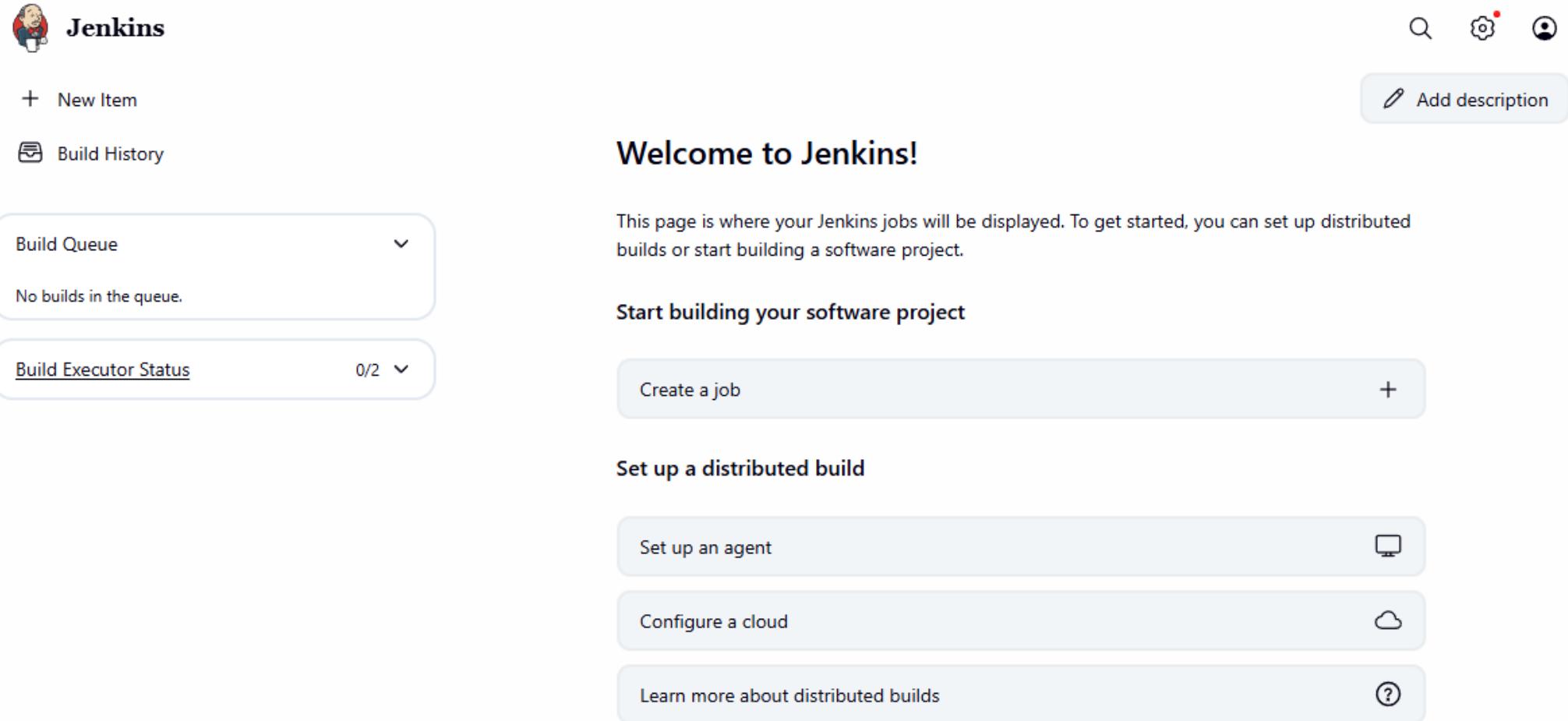
Jenkins Controller and Agents



The Jenkins controller is a persistent server that stores your pipelines, settings, and job history. It provides the user interface and decides when jobs should run, giving you a stable place to manage and track all automation.

When a pipeline runs, the controller spins up an agent to execute the steps. Agents are ephemeral and disappear once the job is finished, but all logs, results, and pipeline definitions remain on the controller. This keeps builds clean while preserving everything you need.

Jenkins Welcome Page



The screenshot shows the Jenkins welcome page. At the top left is the Jenkins logo and the word "Jenkins". To the right are search, settings, and user icons. Below the header, there are two main sections: "Welcome to Jenkins!" and "Start building your software project". The "Welcome to Jenkins!" section contains a message about the purpose of the page and a "Create a job" button. The "Start building your software project" section contains three buttons: "Set up a distributed build", "Set up an agent", and "Configure a cloud". On the left side, there are two collapsed panels: "Build Queue" (showing "No builds in the queue.") and "Build Executor Status" (showing "0/2"). At the bottom left is a copyright notice, and at the bottom right is the number "10".

Welcome to Jenkins!

This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.

Start building your software project

Create a job +

Set up a distributed build

Set up an agent

Configure a cloud

Learn more about distributed builds ?

Build Queue
No builds in the queue.

Build Executor Status
0/2

Jenkins

New Item

Build History

Add description

© 2025 by Innovation In Software Corporation

10

Jenkins Create Pipeline

New Item

Enter an item name

First-Pipeline

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** (Selected)
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**
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.

OK

Jenkins supports many item types, such as Freestyle jobs, Multi-branch pipelines, and traditional projects.

For CI/CD, the most common and recommended option is **Pipeline**, which lets you define your automation using code.

Another popular choice is **Multibranch Pipeline**, which automatically creates pipelines for each branch in a repository.

Jenkins SCM for Pipeline

The screenshot shows the Jenkins Pipeline configuration page for a pipeline named 'First-Pipeline'. The left sidebar under 'Configure' has 'Pipeline' selected. The main area is titled 'Pipeline' with the sub-instruction 'Define your Pipeline using Groovy directly or pull it from source control.' Below this is the 'Definition' section set to 'Pipeline script from SCM'. The 'SCM' dropdown is set to 'Git'. Under 'Repositories', there is one entry with 'Repository URL' as 'https://github.com/d-blanco/jenkins-demo-app', 'Credentials' as '- none -', and 'Advanced' settings. There is also a '+ Add Repository' button. Under 'Branches to build', there is one entry with 'Branch Specifier (blank for \'any\')' as '*/*main'. At the bottom are 'Save' and 'Apply' buttons.

A pipeline has many configuration options, including scheduling, webhook triggers, and build behavior.

The most important section here is **SCM (Source Control Management)**, which lets Jenkins pull your pipeline code directly from GitHub. Storing pipeline code in version control keeps it backed up, reviewable, and versioned instead of relying on Jenkins' internal storage.

Jenkins SCM for Pipeline

The screenshot shows the Jenkins 'Manage Jenkins' interface. At the top, there's a navigation bar with a Jenkins logo, the text 'Jenkins / Manage Jenkins', and a search bar labeled 'Search settings'. Below this, the page is divided into several sections:

- System Configuration**:
 - System**: Configure global settings and paths.
 - Plugins**: Add, remove, disable or enable plugins that can extend the functionality of Jenkins.
 - Clouds**: Add, remove, and configure cloud instances to provision agents on-demand.
- Tools**: Configure tools, their locations and automatic installers.
- Nodes**: Add, remove, control and monitor the various nodes that Jenkins runs jobs on.
- Appearance**: Configure the look and feel of Jenkins.
- Security**:
 - Security**: Secure Jenkins; define who is allowed to access/use the system.
 - Credential Providers**: Configure the credential providers and types.
- Credentials**: Configure credentials.
- Users**: Create/delete/modify users that can log in to this Jenkins.
- Status Information**:
 - System Information**: Displays various environmental information to assist trouble-shooting.
- System Log**: System log captures output from `java.util.logging` related to Jenkins.

Jenkins can be extended through **plugins**, which add new features, integrations, and tools to the platform. The general settings page allows you to install and manage these plugins. Common plugins include Git integration, Docker pipeline steps, and credentials management.

This page also provides access to **secrets and secure credentials**, which Jenkins stores safely and allows pipelines to reference without exposing them in logs or scripts.

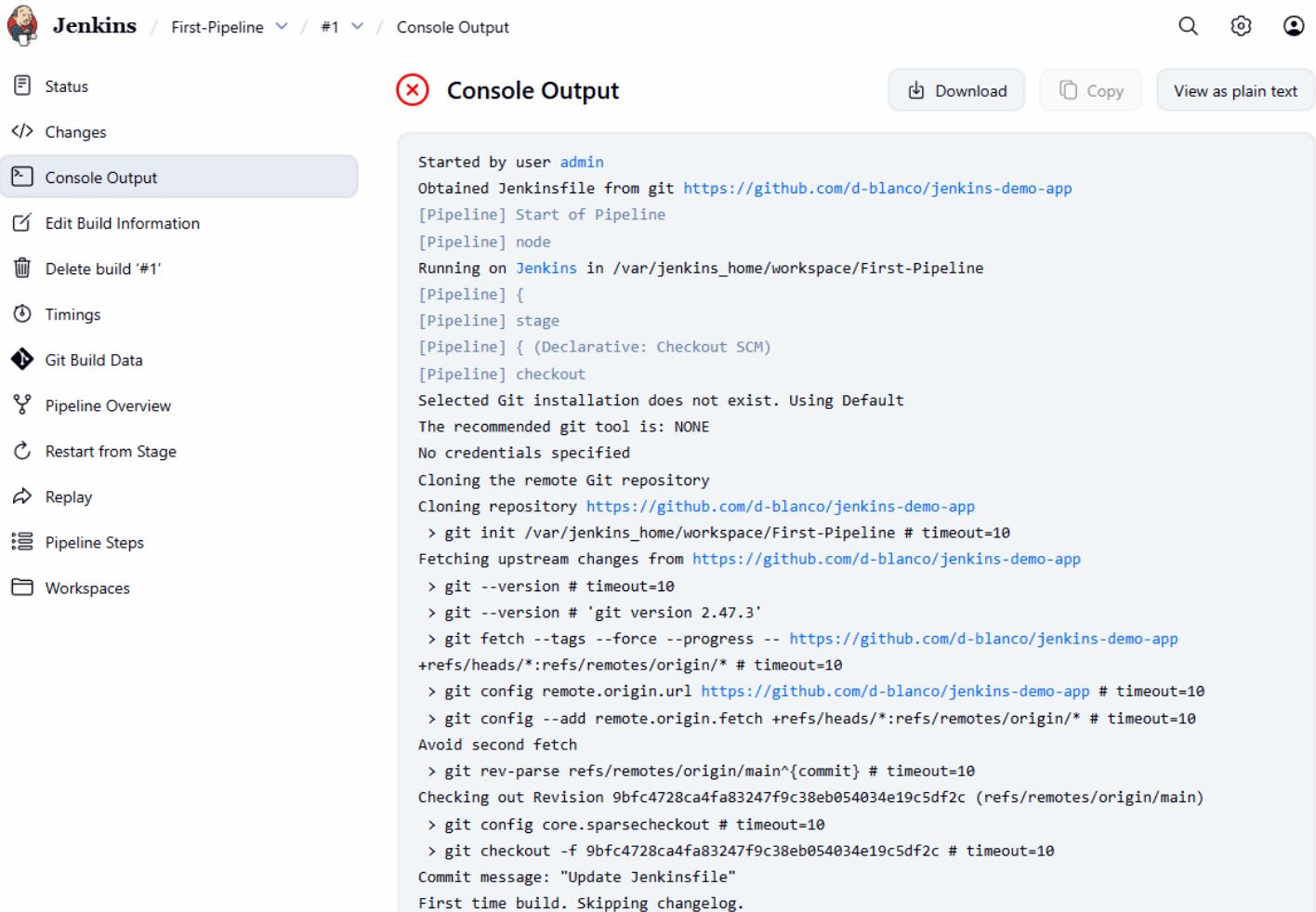
Jenkins SCM for Pipeline

The home page shows all pipelines and their current or last build status. From here, you can quickly view results, trigger new runs, or open a pipeline's details.

The screenshot shows the Jenkins home page. At the top left is the Jenkins logo and the word "Jenkins". To the right are search, settings, and user icons. Below the header are buttons for "New Item", "Build History", and "Add description". On the left, there are two dropdown menus: "Build Queue" (showing "No builds in the queue.") and "Build Executor Status" (showing "0/2"). In the center, a table displays pipeline information. The table has columns: Status (S), Warning (W), Name (sorted by name), Last Success, Last Failure, and Last Duration. One row is visible for "First-Pipeline", which has a red "X" icon, a yellow cloud icon, and a green play icon. At the bottom, there is a legend for "Icon: S M L" and three dots for more options.

S	W	Name ↓	Last Success	Last Failure	Last Duration
✖	⚡	First-Pipeline	N/A	9 min 54 sec #1	8.8 sec

Jenkins SCM for Pipeline



The screenshot shows the Jenkins interface for a pipeline named 'First-Pipeline' with build '#1'. The 'Console Output' tab is selected in the left sidebar. The main content area displays the log output for the pipeline's execution. The logs show the process of obtaining the Jenkinsfile from GitHub, starting the pipeline, and performing a git checkout. The log ends with a note about it being the first time build and skipping the changelog.

```
Started by user admin
Obtained Jenkinsfile from git https://github.com/d-blanco/jenkins-demo-app
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/First-Pipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/d-blanco/jenkins-demo-app
> git init /var/jenkins_home/workspace/First-Pipeline # timeout=10
Fetching upstream changes from https://github.com/d-blanco/jenkins-demo-app
> git --version # timeout=10
> git --version # 'git version 2.47.3'
> git fetch --tags --progress -- https://github.com/d-blanco/jenkins-demo-app
+refs/heads/*:refs/remotes/origin/* # timeout=10
> git config remote.origin.url https://github.com/d-blanco/jenkins-demo-app # timeout=10
> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
> git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision 9bfc4728ca4fa83247f9c38eb054034e19c5df2c (refs/remotes/origin/main)
> git config core.sparsecheckout # timeout=10
> git checkout -f 9bfc4728ca4fa83247f9c38eb054034e19c5df2c # timeout=10
Commit message: "Update Jenkinsfile"
First time build. Skipping changelog.
```

The console output page shows the logs for a **specific run** of a pipeline, letting you see exactly what happened during that execution. It is the main place to troubleshoot any errors.

These logs are generated on the **agent** that runs the job but are saved in Jenkins so they remain available even after the agent disappears. From this view, you can also access additional run details and settings related to that build.

Jenkins SCM for Pipeline

The screenshot shows the Jenkins Pipeline Overview for a build labeled '#3'. The pipeline graph displays five stages: Start, Checkout SCM, Build Docker Image, Push to Docker Hub, and Post Actions, all of which have succeeded. Below the graph, the 'Push to Docker Hub' stage is expanded to show its detailed logs. The logs for this stage include several docker commands and their execution times: 0.27s, 9.5s, 19ms, 0.27s, 17ms, and 2.1s. The final log entry shows the push command: '+ docker push registry.hub.docker.com/danblanco/jenkins-demo-app:latest'.

```
+ docker push registry.hub.docker.com/danblanco/jenkins-demo-app:latest
The push refers to repository [registry.hub.docker.com/danblanco/jenkins-demo-app]
808fa6baa7bd: Preparing
```

The build overview page shows the results of a specific pipeline run but organizes the output by **stages** instead of a single continuous log. This makes it easier to see which stage succeeded, failed, or took the longest.

Each stage expands to show its own logs and details, giving you a clearer picture of where problems occurred without scrolling through the full console output.

Jenkins SCM for Pipeline

```
pipeline {  
    agent any  
  
    stages {  
        stage('Hello') {  
            steps {  
                echo 'Hello World!'  
            }  
        }  
  
        stage('Hello with Shell') {  
            steps {  
                sh 'echo "Hello from shell!"'  
            }  
        }  
    }  
}
```

In Jenkins, a **stage** represents a major section of the pipeline, similar to a step in GitHub Actions. It defines the high-level flow of the build, such as “Build,” “Test,” or “Deploy.”

Inside each stage, Jenkins uses **steps**, which are the individual actions that run. Steps can execute shell commands or use the Jenkins Pipeline **DSL (Domain Specific Language)**—a structured, readable syntax for defining pipeline behavior.

This example shows a simple pipeline with two stages and one step in each, demonstrating how Jenkins organizes and executes pipeline logic.

Jenkins SCM for Pipeline

```
pipeline {  
    agent any  
  
    stages {  
        stage('Hello') {  
            steps {  
                echo 'Hello World!'  
            }  
        }  
  
        stage('Hello with Shell') {  
            steps {  
                sh 'echo "Hello from shell!"'  
            }  
        }  
    }  
}
```

This example is a **Declarative Pipeline**, written in **Groovy**, which provides a structured and opinionated way to define stages, steps, and the overall pipeline flow. Declarative pipelines are the recommended modern approach because they are easier to read, validate, and maintain.

Jenkins also supports **Scripted Pipelines**, which are fully Groovy-based and give complete programmatic control. Scripted pipelines are more flexible but harder to write and understand, so they are used mostly for advanced automation.

Although Jenkins does not use YAML for pipelines like GitHub Actions, it does allow pipelines to be defined in multiple ways:

Jenkins



Jenkins



Tools like GitHub Actions, Jenkins, Docker, and others are often combined to move software from development to production. These tools can overlap or even replace each other, but teams often use several of them together to handle different responsibilities.

Common reasons pipelines become complex

- One tool handles builds, another handles deployments
- Security or approval workflows add extra layers
- Infrastructure teams and application teams use different systems
- Organizations evolve over time instead of starting from scratch

Real pipelines often look like a chain of interconnected tools, each performing a specific part of the release process.

POP QUIZ:

What is the primary role of the Jenkins controller?

- A. To execute all pipeline steps directly
- B. To store pipeline configurations and manage job scheduling
- C. To automatically deploy applications to production
- D. To run containers as part of CI/CD



POP QUIZ:

What is the primary role of the Jenkins controller?

- A. To execute all pipeline steps directly
- B. **To store pipeline configurations and manage job scheduling**
- C. To automatically deploy applications to production
- D. To run containers as part of CI/CD



POP QUIZ:

What is a Jenkins agent?

- A. A temporary worker that runs pipeline steps and disappears afterward
- B. A permanent machine that stores build logs
- C. A Docker container that runs the Jenkins UI
- D. A backup server for the controller



POP QUIZ:

What is a Jenkins agent?

- A. **A temporary worker that runs pipeline steps and disappears afterward**
- B. A permanent machine that stores build logs
- C. A Docker container that runs the Jenkins UI
- D. A backup server for the controller



POP QUIZ:

In Jenkins, what is a *stage* used for?

- A. A single low-level command executed by the agent
- B. Storing environment variables for the controller
- C. Installing plugins during runtime
- D. Grouping major sections of the pipeline such as Build, Test, or Deploy



POP QUIZ:

In Jenkins, what is a *stage* used for?

- A. A single low-level command executed by the agent
- B. Storing environment variables for the controller
- C. Installing plugins during runtime
- D. **Grouping major sections of the pipeline such as Build, Test, or Deploy**



POP QUIZ:

What does the Pipeline DSL (Domain Specific Language) do?

- A. Automatically translates Jenkinsfiles into YAML
- B. Provides structured syntax for defining stages and steps in a pipeline
- C. Allows Jenkins to replace Dockerfiles
- D. Encrypts Jenkins credentials



POP QUIZ:

What does the Pipeline DSL (Domain Specific Language) do?

- A. Automatically translates Jenkinsfiles into YAML
- B. **Provides structured syntax for defining stages and steps in a pipeline**
- C. Allows Jenkins to replace Dockerfiles
- D. Encrypts Jenkins credentials



POP QUIZ:

Where are Jenkins pipeline logs stored after a run completes?

- A. On the ephemeral agent permanently
- B. They are deleted immediately after the agent shuts down
- C. Stored on the Jenkins controller and accessible in the Console Output
- D. Saved only in Docker Hub



POP QUIZ:

Where are Jenkins pipeline logs stored after a run completes?

- A. On the ephemeral agent permanently
- B. They are deleted immediately after the agent shuts down
- C. **Stored on the Jenkins controller and accessible in the Console Output**
- D. Saved only in Docker Hub



POP QUIZ:

What is the main advantage of storing a Jenkinsfile in SCM (like GitHub)?

- A. It allows Jenkins to run without a controller
- B. Pipelines become versioned, backed up, and updated through code changes
- C. Jenkins can automatically create EC2 instances
- D. It removes the need for agents



POP QUIZ:

What is the main advantage of storing a Jenkinsfile in SCM (like GitHub)?

- A. It allows Jenkins to run without a controller
- B. **Pipelines become versioned, backed up, and updated through code changes**
- C. Jenkins can automatically create EC2 instances
- D. It removes the need for agents



Lab: Jenkins Docker



Manual Deployment



Manual Deployment



Manual Deployment



Manual to Automated



Before you automate a deployment, you must understand how to do it manually. Automation simply repeats the steps you would perform yourself, so if the manual process is unclear or incorrect, the automated version will fail as well.

Knowing the manual steps gives you insight into what must be installed, configured, started, and verified on the server. This ensures your automation tools can reproduce the deployment reliably and consistently.

Deployment Via Containers



Since Docker Hub is our delivery source for packaged images, we need a way to run those images on a server. Images run inside **containers**, and there are many platforms that can manage them, such as Kubernetes or AWS ECS.

For this simple example, we will use the **Docker Runtime** directly on a virtual machine. This allows us to pull the image, run it as a container, and clearly understand the manual steps before we automate the process.

Deployment Via Docker



Deployment Requirements

- A virtual machine we can SSH into
- A public IP address to access the application
- Firewall or security group rules open for the app's ports
- Docker installed and the Docker service running

Manual Deployment Steps

- Pull the image from Docker Hub
- Run the container with correct port bindings
- Provide any required environment variables or config values
- Optionally run the container in detached (daemon) mode

Automating Deployments



Before automating deployment, you should be able to manage the container manually. This includes viewing logs, stopping and starting the container, and testing the application directly on the VM.

Understanding how the container behaves in a manual workflow helps ensure that your automation will be accurate, reliable, and easier to troubleshoot.

Containers



- A container stops when its **main process** exits. Docker does not keep containers alive on its own.
- Containers read **environment variables** at runtime, allowing the same image to run with different configurations in dev, test, or production.
- Images are **immutable**, but containers are not. You configure them through runtime settings, not by changing the image.
- Logs and output come directly from the main process, which makes them easy to capture in a CI/CD pipeline.
- Since images package dependencies, builds are consistent across environments, which is ideal for automated testing and delivery.

Docker Run

The screenshot shows a navigation bar with links: dockerdocs, Get started, Guides, Manuals, and Reference (which is underlined). A sidebar on the left lists various Docker commands, with 'docker container run' highlighted by a light gray box. The main content area shows the URL 'Home / Reference / CLI reference / docker / docker container / docker container run'. The title 'docker container run' is displayed. Below it, there are three sections: 'Description' (Create and run a new container from an image), 'Usage' (docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]), and 'Aliases' (docker run). The 'Description' section includes a paragraph about running a command in a new container and pulling the image if needed. The 'Aliases' section notes that 'docker run' is an alias for this command.

Home / Reference / CLI reference / docker / docker container / docker container run

docker container run

Description Create and run a new container from an image

Usage `docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]`

Aliases [? docker run](#)

Description

The `docker run` command runs a command in a new container, pulling the image if needed and starting the container.

You can restart a stopped container with all its previous changes intact using `docker start`. Use `docker ps -a` to view a list of all containers, including those that are stopped.

Docker Logs

The screenshot shows a navigation bar with links: dockerdocs, Get started, Guides, Manuals, and Reference (which is underlined). A sidebar on the left lists various Docker commands, with 'docker container logs' highlighted by a light gray box. The main content area shows the URL 'Home / Reference / CLI reference / docker / docker container / docker container logs'. The title 'docker container logs' is displayed. Below it, there are three sections: 'Description' (Fetch the logs of a container), 'Usage' (code block: `docker container logs [OPTIONS] CONTAINER`), and 'Aliases' (code block: `docker logs`). The 'Description' section includes text about the command's purpose and links to logging drivers and follow functionality.

Home / Reference / CLI reference / docker / docker container / docker container logs

docker container logs

Description Fetch the logs of a container

Usage `docker container logs [OPTIONS] CONTAINER`

Aliases `docker logs`

Description

The `docker logs` command batch-retrieves logs present at the time of execution.

For more information about selecting and configuring logging drivers, refer to [Configure logging drivers](#).

The `docker logs --follow` command will continue streaming the new output from the container's `STDOUT` and `STDERR`.

Passing a negative number or a non-integer to `--tail` is invalid and the value is set to `all` in that case.

POP QUIZ:

Why do we perform a manual deployment before automating it?

- A. Because automation repeats the steps you already understand manually
- B. Because automation tools cannot run without manual deployment
- C. Because Docker requires it for licensing
- D. Because VMs only allow manual deployment



POP QUIZ:

Why do we perform a manual deployment before automating it?

- A. **Because automation repeats the steps you already understand manually**
- B. Because automation tools cannot run without manual deployment
- C. Because Docker requires it for licensing
- D. Because VMs only allow manual deployment



POP QUIZ:

What is required before you can manually deploy a container to a VM?

- A. A Kubernetes manifest
- B. An SSH key and Docker installed on the VM
- C. A load balancer already configured
- D. A Jenkins agent running on the VM



POP QUIZ:

What is required before you can manually deploy a container to a VM?

- A. A Kubernetes manifest
- B. An SSH key and Docker installed on the VM**
- C. A load balancer already configured
- D. A Jenkins agent running on the VM



POP QUIZ:

What does the `-p 80:8000` flag do when running a container?

- A. It installs Docker
- B. It scales containers across multiple VMs
- C. It maps a VM port to a container port
- D. It validates the Dockerfile before running



POP QUIZ:

What does the `-p 80:8000` flag do when running a container?

- A. It installs Docker
- B. It scales containers across multiple VMs
- C. **It maps a VM port to a container port**
- D. It validates the Dockerfile before running



POP QUIZ:

What should you do AFTER manually deploying the container?

- A. Push the VM image to Docker Hub
- B. Test the running application
- C. Delete the Dockerfile
- D. Disable all inbound traffic



POP QUIZ:

What should you do AFTER manually deploying the container?

- A. Push the VM image to Docker Hub
- B. **Test the running application**
- C. Delete the Dockerfile
- D. Disable all inbound traffic



POP QUIZ:

What happens if a container's main process exits?

- A. Docker automatically rebuilds the image
- B. Docker restarts the VM
- C. The container stops completely
- D. The container switches to a backup process



POP QUIZ:

What happens if a container's main process exits?

- A. Docker automatically rebuilds the image
- B. Docker restarts the VM
- C. **The container stops completely**
- D. The container switches to a backup process



Lab: Manual Deployment



Ansible Deployment



Ansible



Ansible is an automation tool that configures systems, installs software, and runs tasks across servers using simple YAML playbooks. It connects over SSH and does not require agents on the machines it manages.

It is commonly used for repeatable deployments, server configuration, and automating manual steps — making it a perfect fit for automating the container deployment you just performed manually.

Ansible Features



Playbook

The main file that defines *what* you want to do. A playbook contains one or more plays, and each play runs tasks on a group of hosts.

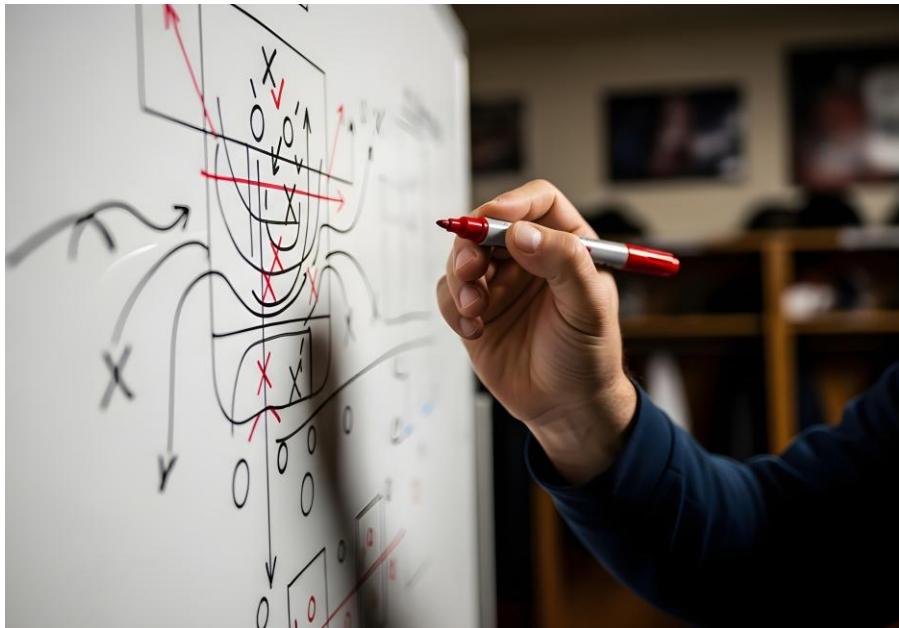
Tasks

The individual actions Ansible performs, such as installing a package, copying a file, or starting a service. Tasks are listed inside a playbook or can be organized into roles for reuse.

Inventory

A list of servers Ansible will manage. Inventories can be simple files with IP addresses or dynamic sources that fetch servers based on tags or cloud metadata.

Ansible Features



Roles

A way to organize playbooks into reusable components. Roles bundle tasks, variables, templates, and handlers into a structured format to keep projects clean and maintainable.

Variables

Values that can change between environments or hosts. Variables let you customize deployments without modifying the playbook itself.

Handlers

Special tasks that run only when notified, often used for actions like restarting a service after a configuration change.

Idempotency

A core Ansible principle ensuring that running the same playbook multiple times produces the same result. Ansible only makes changes when something is actually different.

Ansible

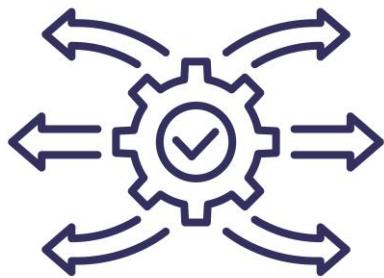


Ansible lets us turn the manual deployment process into repeatable, reliable automation. Instead of logging into servers and typing commands, Ansible enforces configuration the same way every time.

Key points

- Eliminates human error
- Makes deployments repeatable across many servers
- Ensures systems converge to the same state
- Integrates naturally with CI/CD pipelines
- Deployment becomes “infrastructure as code”

Continuous Deployment



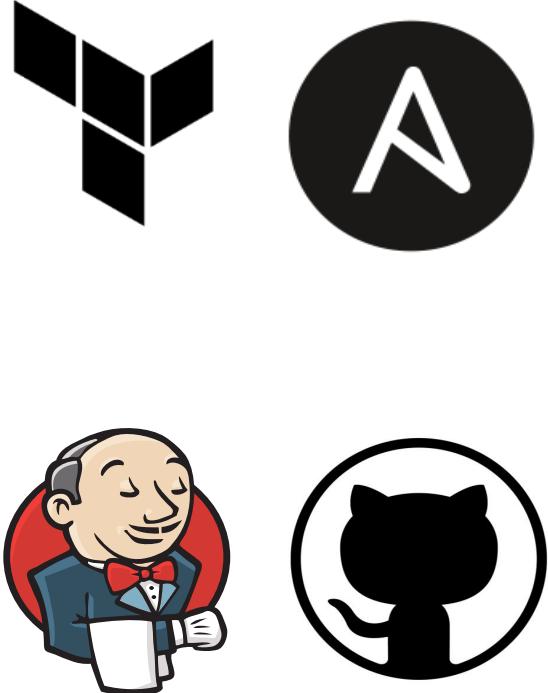
Ansible is typically used during the **CD (Continuous Deployment)** phase to configure servers and deploy application versions.

Common patterns

- CI (GitHub Actions / Jenkins) builds & tests the app
- CI pushes the image to Docker Hub
- CD uses Ansible to deploy the image to servers
- Repeat for blue/green or rolling updates

Build → Deliver → Deploy

Tools for CICD



Tool	Purpose	Example
Terraform	Create infrastructure	EC2, VPC, SGs
Ansible	Configure / deploy software	Install Docker, run container
Jenkins / GitHub Actions	Orchestrate automation	Trigger builds/deploy

These tools are complementary of each other

Ansible Playbook



A deployment playbook usually includes:

- Installing or configuring dependencies (Docker, packages, services)
- Pulling the correct application version
- Starting the application
- Verifying the service is running
- Ensuring idempotency for repeated runs

Idempotency



Idempotency means the playbook produces the same result every time you run it.

Why it matters for deployment

- Safe to re-run after failures
- Ensures consistent environments
- Detects real changes (new version, config updates)
- Allows hands-off automation

POP QUIZ:

What is the purpose of an Ansible playbook?

- A. To define the desired state and steps needed to configure or deploy systems
- B. To store SSH keys for remote hosts
- C. To open firewall ports automatically
- D. To build Docker images for CI workflows



POP QUIZ:

What is the purpose of an Ansible playbook?

- A. **To define the desired state and steps needed to configure or deploy systems**
- B. To store SSH keys for remote hosts
- C. To open firewall ports automatically
- D. To build Docker images for CI workflows



POP QUIZ:

Why is Ansible useful for deployment across multiple servers?

- A. It ignores server differences and assumes all hosts are identical
- B. It can run commands in parallel and enforce consistent configuration
- C. It only runs on servers with Docker pre-installed
- D. It requires no SSH permissions



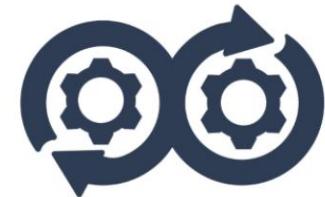
POP QUIZ:

Why is Ansible useful for deployment across multiple servers?

- A. It ignores server differences and assumes all hosts are identical
- B. **It can run commands in parallel and enforce consistent configuration**
- C. It only runs on servers with Docker pre-installed
- D. It requires no SSH permissions



Lab: Ansible Deployment



Congratulations



You accomplished a full end-to-end deployment workflow today.

Today you learned

- How Jenkins runs pipelines using the controller/agent model
- How to build and push Docker images through Jenkins
- How to manually deploy and test containers on a VM
- How to automate deployments with Ansible using playbooks, tasks, and roles

You now have both the manual and automated foundations needed for real-world deployment workflows.