# Assignment No 1

**Aim:**

To implement continuous deployment (CD) using **Ansible** — automated, repeatable, and idempotent deployment of application releases from source control to target servers, integrated into a CI pipeline so every accepted change can be deployed automatically and safely.

**Theory:**

Continuous Deployment is the practice of automatically deploying every change that passes automated tests to production (or a release environment). Ansible is an agentless configuration-management and orchestration tool that runs over SSH, expresses automation as readable YAML playbooks, and emphasizes idempotence (running the same playbook repeatedly produces the same result).

Why use Ansible for CD?

- **Agentless & simple** — connect over SSH, no agents to install.
- **Idempotent** — playbooks are designed to leave systems in the desired state.
- **Human-readable** — YAML playbooks and roles are easy for teams to audit.
- **Extensible** — roles, modules, and integrations (with Jenkins, GitHub Actions, GitLab CI, etc.).
- **Safe operations** — you can add checks, dry-runs (--check), handlers, and rollback steps.

Key CD concepts with Ansible:

- **Control node**: machine where Ansible runs (CI runner or dedicated server).
- **Managed nodes**: target servers (app servers, load balancers).
- **Inventory**: list/grouping of managed nodes.
- **Playbooks / Roles**: declarative tasks describing desired state.
- **Handlers**: special tasks triggered when a change occurs (e.g., restart service).
- **Vault**: securely manage secrets (API keys, SSH keys).
- **CI Integration**: CI tool triggers Ansible on merge to main/release branch.
- **Rollback**: defined process or playbook to revert to previous release (e.g., previous artifact, git tag, or snapshot).

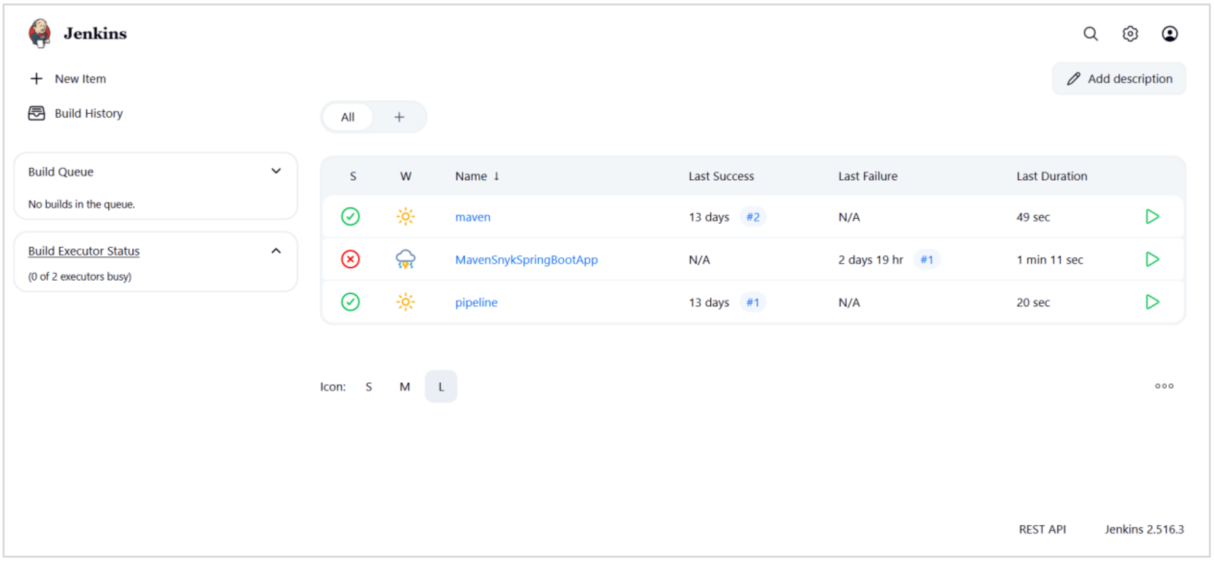
**Tools & prerequisites**

- Ansible (control node)
- SSH access from control node to managed nodes (key-based auth)
- Git repo with application and/or build artifacts (or artifact registry)
- CI system (GitHub Actions, GitLab CI, Jenkins, etc.)
- Optional: Ansible Vault for secrets, load balancer, monitoring & alerting
- Target environment with appropriate runtime (Node, Python, Docker, etc.)

**Procedure / Steps:**

**Step 1: Setup Jenkins**

1. Install Jenkins on the control server.
2. Start the Jenkins service and open the Jenkins dashboard in a browser (http://localhost:8080).
3. Install the **recommended plugins** during the setup wizard.


Jenkins

+ New Item

Build History

Add description

All +

Build Queue

No builds in the queue.

Build Executor Status

(0 of 2 executors busy)

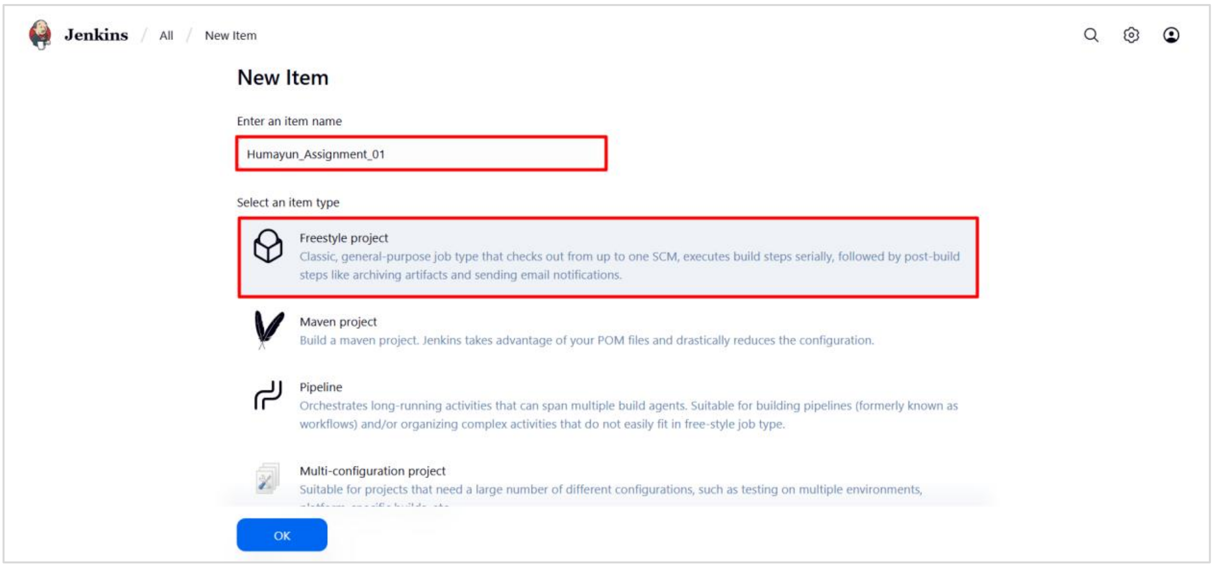
| S | W | Name ↓ | Last Success | Last Failure | Last Duration |
|---|---|---|---|---|---|
| ✓ | ☀ | maven | 13 days #2 | N/A | 49 sec |
| ✗ | ⛈ | MavenSnykSpringBootApp | N/A | 2 days 19 hr #1 | 1 min 11 sec |
| ✓ | ☀ | pipeline | 13 days #1 | N/A | 20 sec |

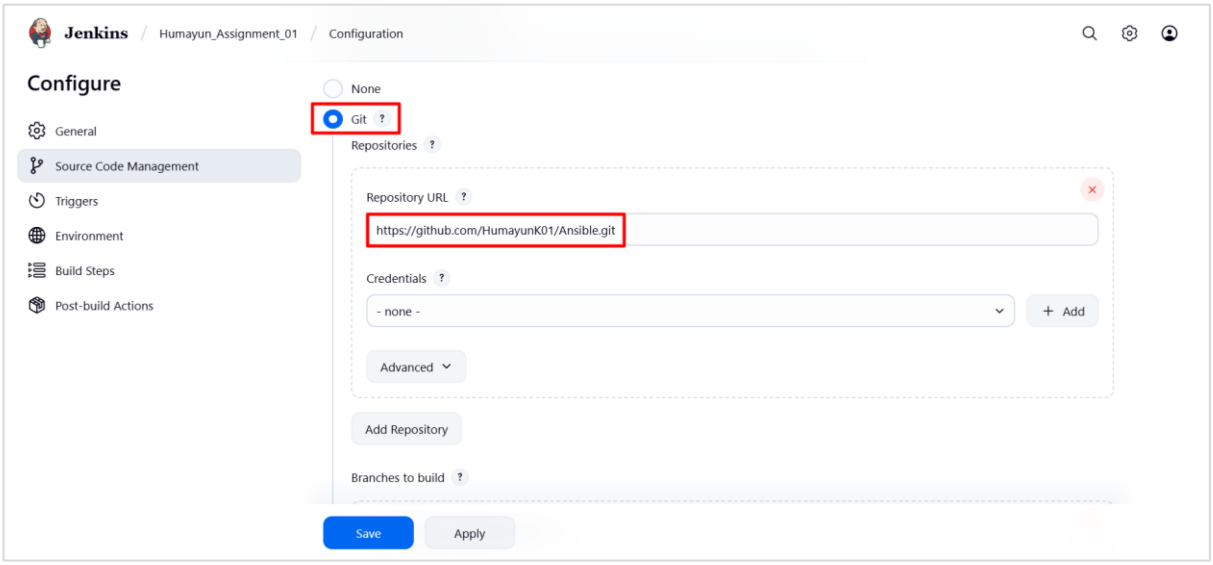
Icon: S M L

REST API Jenkins 2.516.3

**Step 2: Create a New Jenkins Project**

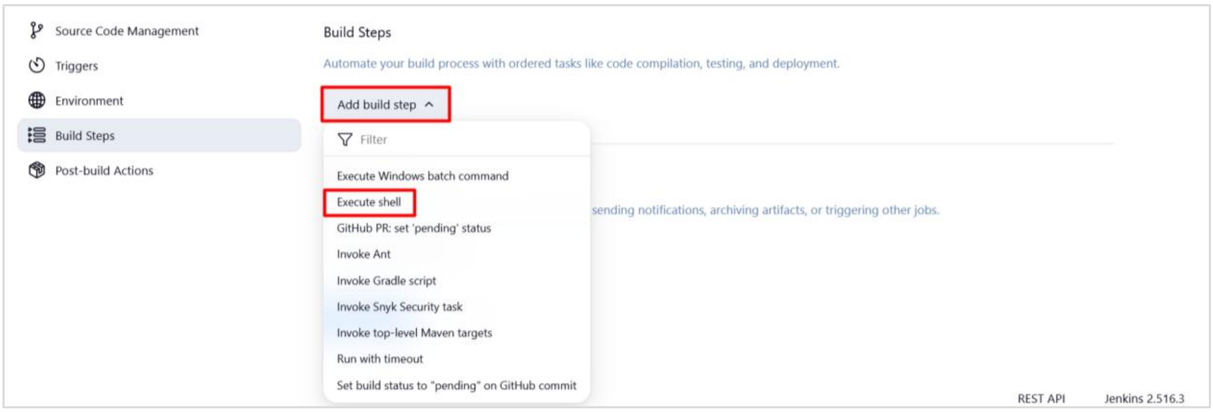
1. Click on **"New Item"** → Enter name (Any Name) → Choose **Freestyle Project** → Click **OK**.
2. In the project configuration page, scroll down to the **Source Code Management** section.
3. Select **Git** and provide the repository URL where the Ansible playbooks are stored (e.g., https://github.com/HumayunK01/Ansible.git).


Jenkins / All / New Item
New Item
Enter an item name
Humayun_Assignment_01
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.
Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.
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,
OK



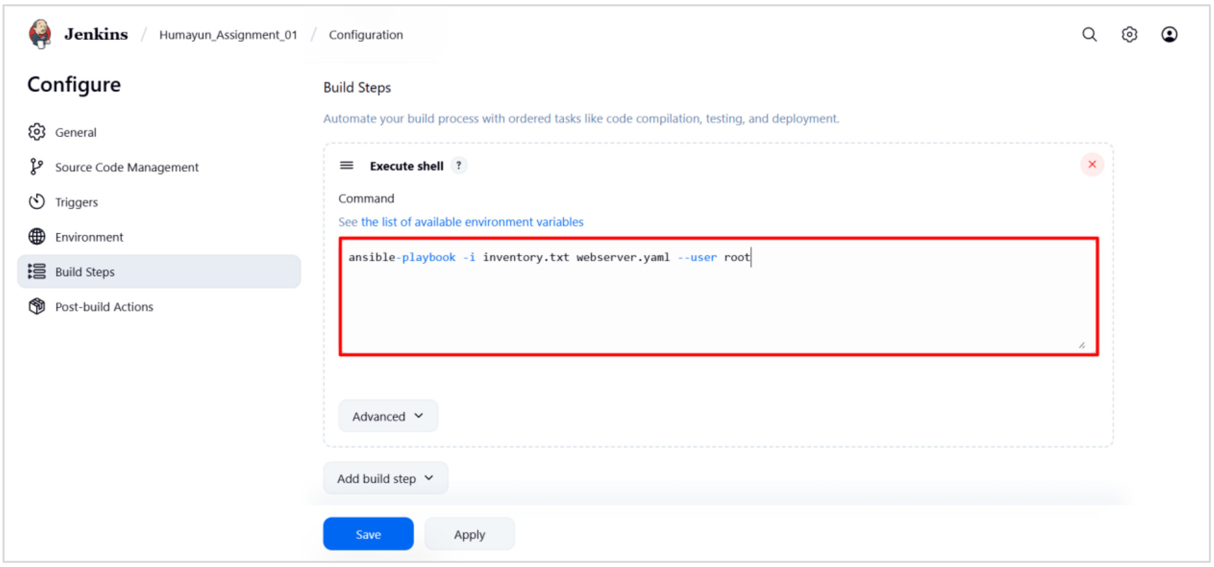
## Step 3: Configure Jenkins Build Step

1. In Jenkins project configuration, go to **Build Steps** → Click **Add build step** → Select **Execute Shell**.


Source Code Management
Triggers
Environment
Build Steps
Post-build Actions
Build Steps
Automate your build process with ordered tasks like code compilation, testing, and deployment.
Add build step
Filter
Execute Windows batch command
Execute shell
GitHub PR: set 'pending' status
Invoke Ant
Invoke Gradle script
Invoke Snyk Security task
Invoke top-level Maven targets
Run with timeout
Set build status to "pending" on GitHub commit
sending notifications, archiving artifacts, or triggering other jobs.
REST API
Jenkins 2.516.3

2. Enter the command to run the playbook:

```
>> ansible-playbook -i inventory.txt webserver.yaml
```


Jenkins / Humayun_Assignment_01 / Configuration
Configure
General
Source Code Management
Triggers
Environment
Build Steps
Post-build Actions
Build Steps
Automate your build process with ordered tasks like code compilation, testing, and deployment.
Execute shell
Command
See the list of available environment variables
Advanced
Add build step
Save
Apply

```
ansible-playbook -i inventory.txt webserver.yaml --user root
```

3. Click **Save**

**Step 4: Configure SSH Authentication**

1. On Jenkins server, switch to the Jenkins user:

   ```
   >> sudo su - jenkins
   ```

2. Generate SSH key pair:

   ```
   >> ssh-keygen
   ```

3. Copy SSH public key to target host:

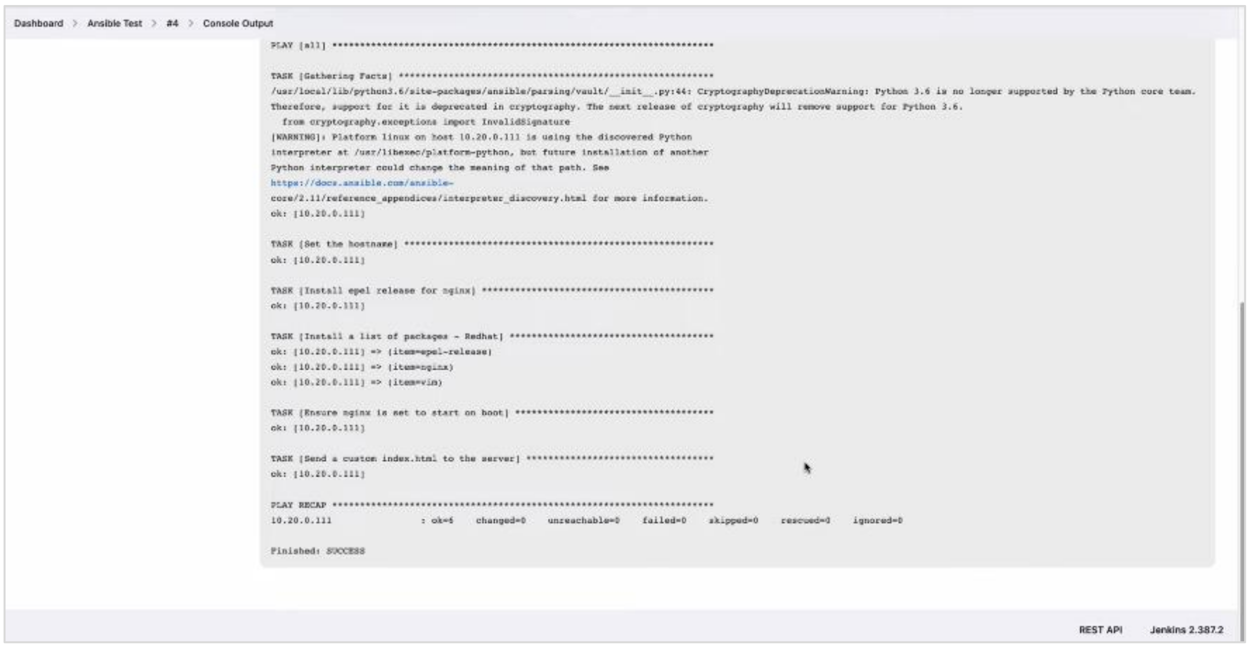
   ```
   >> ssh-copy-id root@10.20.0.111
   ```

4. Test SSH connection:

   ```
   >> ssh root@10.20.0.111
   ```

**Step 5: Build and Verify**

1. Return to Jenkins dashboard.
2. Click **Build Now**.
3. Open **Console Output** to view the live execution logs.
4. Observe:
   - Ansible connecting to the remote host.
   - Tasks being executed (installing Nginx, configuring web server, etc.).
   - Successful completion message.



```
PLAY [all] *********************************************************************

TASK [Gathering Facts] *********************************************************
/usr/local/lib/python3.6/site-packages/ansible/parsing/vault/__init__.py:44: CryptographyDeprecationWarning: Python 3.6 is no longer supported by the Python core team.
Therefore, support for it is deprecated in cryptography. The next release of cryptography will remove support for Python 3.6.
  from cryptography.exceptions import InvalidSignature
[WARNING]: Platform linux on host 10.20.0.111 is using the discovered Python
interpreter at /usr/libexec/platform-python, but future installation of another
Python interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-
core/2.11/reference_appendices/interpreter_discovery.html for more information.
ok: [10.20.0.111]

TASK [Set the hostname] ********************************************************
ok: [10.20.0.111]

TASK [Install epel release for nginx] ******************************************
ok: [10.20.0.111]

TASK [Install a list of packages - Redhat] *************************************
ok: [10.20.0.111] => (item=epel-release)
ok: [10.20.0.111] => (item=nginx)
ok: [10.20.0.111] => (item=vim)

TASK [Ensure nginx is set to start on boot] ************************************
ok: [10.20.0.111]

TASK [Send a custom index.html to the server] **********************************
ok: [10.20.0.111]

PLAY RECAP *********************************************************************
10.20.0.111                : ok=6    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

Finished: SUCCESS
```

**Conclusion:**

Continuous deployment using **Ansible and Jenkins** was successfully implemented. Jenkins automated the execution of Ansible playbooks across the target servers, ensuring consistent and repeatable deployments. This integration demonstrates how Jenkins can act as an orchestrator, scheduling and executing infrastructure tasks while Ansible performs configuration management and deployment, leading to a robust CI/CD pipeline.