Jenkins Automation

(using python-jenkins library)

By Surisetti Lochani Vilehya

INDEX:

#1: Jenkins Set-up

- **Prerequisites**
  - A system running Ubuntu or Debian
  - sudo privileges
  - Java 17 or 21 installed
- To install jenkins the java version must be 17 or 21

```
$  sudo apt update

$  sudo apt install openjdk-17-jdk

Or

$  sudo apt update

$  sudo apt install openjdk-21-jdk
```

- If multiple versions of Java exist, configure the default version: Select the appropriate Java version and press Enter.

```
$ sudo update-alternatives --config java
```

- Add Jenkins Repository and Key :

```
$ sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
$ echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/" | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
```

- Install Jenkins :

```
$ sudo apt-get update

$ sudo apt-get install -y jenkins
```

- Start and Enable Jenkins Service and check the Jenkins service status:

```
$ sudo systemctl start jenkins

$ sudo systemctl enable jenkins

$ sudo systemctl status jenkins
```

- Access Jenkins :
  - Jenkins runs on port 8080 by default. Open a web browser and go to `http://<your-server-ip>:8080`
- Retrieve Initial Admin Password

```
$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

#1.1: python-jenkins library

- We use python-jenkins library to interact with the jenkins-server
- python-jenkins is a Python module that allows users to interact with a Jenkins server programmatically using its REST API.
- It helps automate Jenkins tasks such as creating jobs, triggering builds, managing plugins, and retrieving build results.

#1.1.1: Installation of python-jenkins

- Before using python-jenkins, you need to install it. You can do this using pip:

```
$ python -m venv myenv
$ source myenv/bin/activate  # On macOS/Linux
$ myenv\Scripts\activate      # On Windows
$ pip install python-jenkins
```

#1.1.2: Using Jenkins Class from python-jenkins

- **Class definition:**

```
class jenkins.Jenkins(url, username=None, password=None,
timeout=<object object>)
```

- **Parameters :**
  - url (str): The URL of the Jenkins server.
  - username (str, optional): The username for authentication.

    - password (str, optional): The API token or password for authentication.
    - timeout (int, optional): Server connection timeout in seconds.

- Usage Example:

```
import jenkins

# Connect to Jenkins server
server = jenkins.Jenkins('http://your-jenkins-url:8080',
username='admin', password='your-api-token')

# Get Jenkins server version
version = server.get_version()
print(f'Connected to Jenkins, version: {version}')
```

#2: Jenkins Jobs

- A Jenkins job is a task or a unit of work that Jenkins performs.
- Jobs are the core components of Jenkins' continuous integration and continuous delivery pipeline. They define the steps to be executed in an automated way, such as building, testing, deploying, or any other task that can be automated.
- A Jenkins job typically consists of the following:
    - **Source Code:** A job is often triggered by a change in the source code repository (e.g., Git, SVN).
    - **Build Triggers:** A job can be triggered manually, by a specific event (such as a commit or pull request), or on a regular schedule (like a cron job).
    - **Build Steps:** The actual actions that the job performs, like running scripts, building code, testing, or packaging artifacts.
    - **Post-Build Actions:** These are actions that occur after the build, like sending notifications, archiving build artifacts, or deploying code to a staging or production environment.
    - **Configuration:** Each job has its own configuration, typically defined in XML format, which specifies all the above aspects.

- Types of Jenkins Jobs:
  - **Freestyle Project:** The simplest type of job, allowing you to run basic scripts and commands.
  - **Pipeline**: More complex and flexible jobs that allow you to define the sequence of tasks in a Jenkinsfile (a script that defines the pipeline).
  - **Multibranch Pipeline**: A type of pipeline job that automatically discovers branches in a repository and builds them.
  - **Build Flow**: Allows the configuration of multiple jobs with more complex flow logic.
  - **Matrix** Project: Used to run a set of tests with different configurations or environment

#2.1: Jenkins Basic information API Methods

S.No.	API Method	Description	Parameters	Returns
1	**get_info(item='', query=None)**	Retrieves information about the Jenkins master or a specific item.	**item** (str, optional): The item to get info about. **query** (str, optional): XPath to extract specific info.	Dictionary with details about Jenkins master or item.
2	**get_whoami(depth=0)**	Retrieves information about the authenticated user.	**depth** (int, optional): JSON depth (default: 0).	Dictionary with user details (fullName, id, etc.).
3	**get_version()**	Retrieves the version number of the Jenkins master.	No parameters.	Jenkins version number (str).
4	**get_jobs(folder_depth=0, folder_depth_per_request=10, view_name=None)**	Get list of jobs. Each job is a dictionary with 'name', 'url', 'color' and 'fullname' keys.	**folder_depth –** Number of levels to search, int. By default 0, which will limit search to toplevel. None disables the limit. **folder_depth_per_request –** Number of levels to fetch at once, int. See get_all_jobs(). **view_name –** Name of a Jenkins view for which to retrieve jobs,	list of jobs,

			str. By default, the job list is not limited to a specific view.	
5	**get_plugins_info(depth=2)**	Retrieves information about all installed plugins (deprecated).	**depth** (int, optional): JSON depth (default: 2).	List of plugin details in dictionary format.
6	**get_plugins(depth=2)**	Retrieves plugin information with version comparison helper.	**depth** (int, optional): JSON depth (default: 2).	Dictionary of installed plugins with short/long names.

#2.1: Jenkins Build API Methods

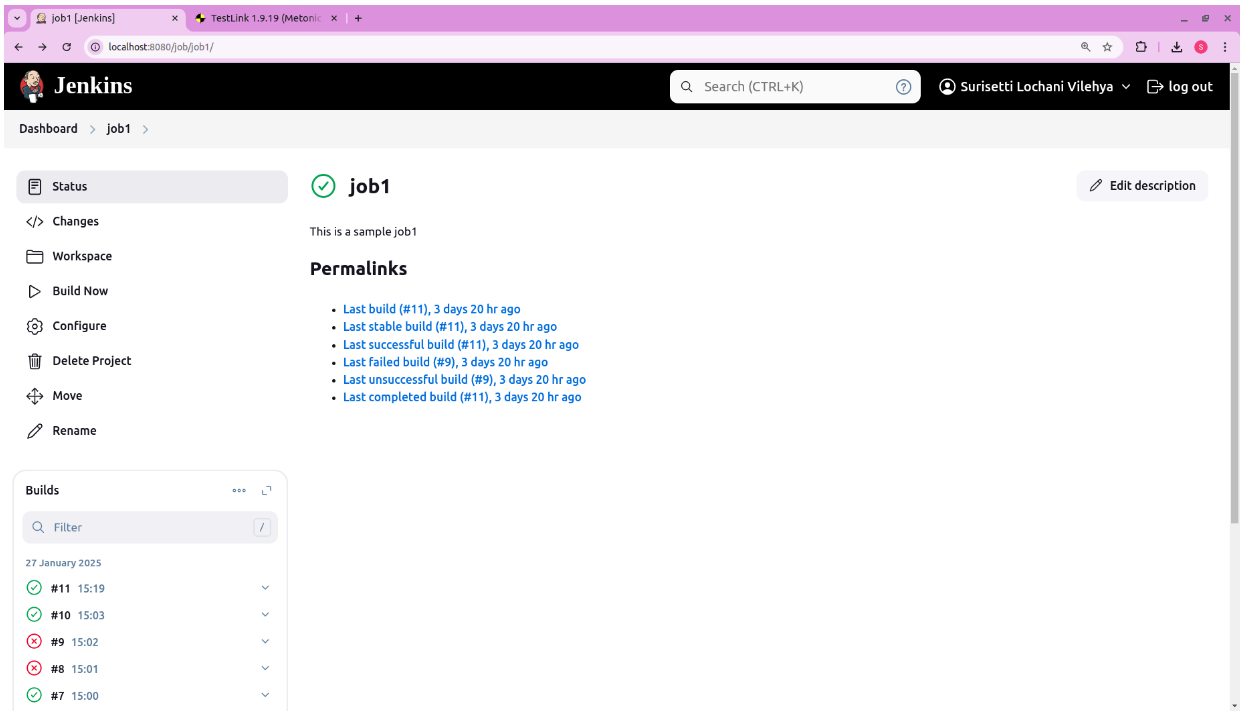
S.No.	API Method	Description	Parameters	Returns
1	**create_job(name, config_xml)**	Create a new Jenkins job.	name (str) – Job name config_xml (str) – Job configuration in XML	None (Job is created on the Jenkins server).
2	**create_job(name, jenkins.EMPTY_CONFIG_XML)**	Create a new Jenkins job with empty configuration.	name {str} - Jenkins job name. jenkins.EMPTY_CONFIG_XML - a predefined constant or variable in the Jenkins API that holds an empty job configuration in XML format	None (Job is created on the Jenkins server).
3	**get_job_config(name)**	Get the configuration of an existing job in XML format.	name (str) – Job name	Job configuration in XML format.
4	**reconfig_job(name, config_xml)**	Change the configuration of an existing job.	name (str) – Job name config_xml (str) – New job configuration in XML	None (Job is reconfigured on the Jenkins server).
5	**job_exists(name)**	Check whether a Jenkins job exists.	name (str) – Job name	True if the job exists, False otherwise.
6	**jobs_count()**	Get the total number of jobs on the Jenkins server.	No parameters	Total number of jobs (int).
7	**copy_job(from_name, to_name)**	Copy an existing Jenkins job. The source and destination must be in the same folder.	from_name (str) – Source job name to_name (str) – Destination job name	None (Job is copied to the new name).
8	**rename_job(from_name, to_name)**	Rename an existing Jenkins job. The source and destination must be in the same folder.	from_name (str) – Current job name to_name (str) – New job name	None (Job is renamed on the Jenkins server).
9	**delete_job(name)**	Delete a Jenkins job permanently.	name (str) – Job name	None (Job is deleted from the Jenkins server).
10	**enable_job(name)**	Enable a disabled Jenkins job.	name (str) – Job name	None (Job is enabled on the Jenkins server).
11	**disable_job(name)**	Disable a Jenkins job.	name (str) – Job	None (Job is disabled on the Jenkins server)

12	**get_job_info(name, depth=0, fetch_all_builds=False)**	Retrieves detailed information for a specific job	name (str): Job name. depth (int): JSON depth level. fetch_all_builds (bool): Fetch all builds (True) or the most recent 100 builds (False).	Dictionary containing detailed job information.
13	**get_job_info_regex(pattern, depth=0, folder_depth=0, folder_depth_per_request=10)**	Retrieves information about jobs whose names match the specified regex pattern.	pattern (str): Regex pattern to match job names. depth (int): JSON depth level. folder_depth (int): Folder depth level to search through. folder_depth_per_request (int): Number of levels to fetch at once.	List of dictionaries containing job information that match the regex
14	**get_job_name(name)**	Returns the name of a Jenkins job, used for identity verification.	name (str): Job name.	Job name (str) or None if the job does not exist.
15	**wipeout_job_workspace(name)**	Wipes out the workspace for a given Jenkins job.	name (str): Job name.	None (Job workspace is wiped out on the Jenkins server).

#2.2: config_xml of job

- config_xml in Jenkins refers to the configuration file of a Jenkins job, written in XML format. This XML file defines the job’s properties, including:
    - **Job name:** The name of the Jenkins job.
    - **Build steps:** Commands or actions that the job performs when triggered (e.g., shell commands, Maven builds).
    - **Triggers:** How and when the job is triggered (e.g., on code commit, scheduled time).
    - **Post-build actions**: Actions that are performed after the build completes (e.g., sending notifications, archiving artifacts).
    - **Parameters**: If the job has parameters, they are defined here.

- Jenkins provides a REST API that allows you to interact with Jenkins jobs programmatically, including retrieving the config_xml.
- If your Jenkins server is running at http://your-jenkins-server and you want to get the configuration of a job named my_job, the API URL would be:
  - [`http://your-jenkins-server/job/my_job/config.xml`](http://your-jenkins-server/job/my_job/config.xml)
- To fetch the config_xml, you can make a simple HTTP GET request. Here's how you can do this:
  - `curl http://your-jenkins-server/job/my_job/config.xml`
- If authentication is required (e.g., if Jenkins is protected by username and API token), you can pass the credentials as follows:
  - `curl -u your_username:your_api_token http://your-jenkins-server/job/my_job/config.xml`
- Using Browser You can also open the configure section of the Job and directly slide down in your web browser, and you would see the REST API option, click on it



job1

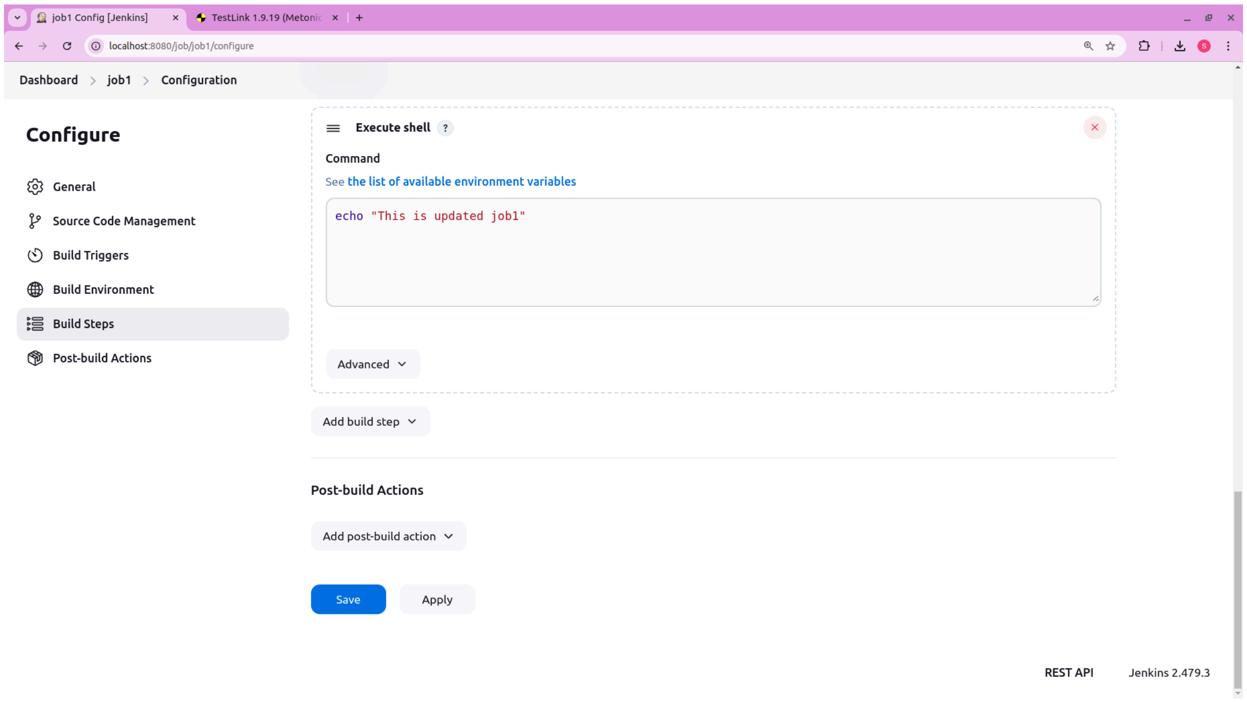
This is a sample job1

Permalinks

- Last build (#11), 3 days 20 hr ago
- Last stable build (#11), 3 days 20 hr ago
- Last successful build (#11), 3 days 20 hr ago
- Last failed build (#9), 3 days 20 hr ago
- Last unsuccessful build (#9), 3 days 20 hr ago
- Last completed build (#11), 3 days 20 hr ago

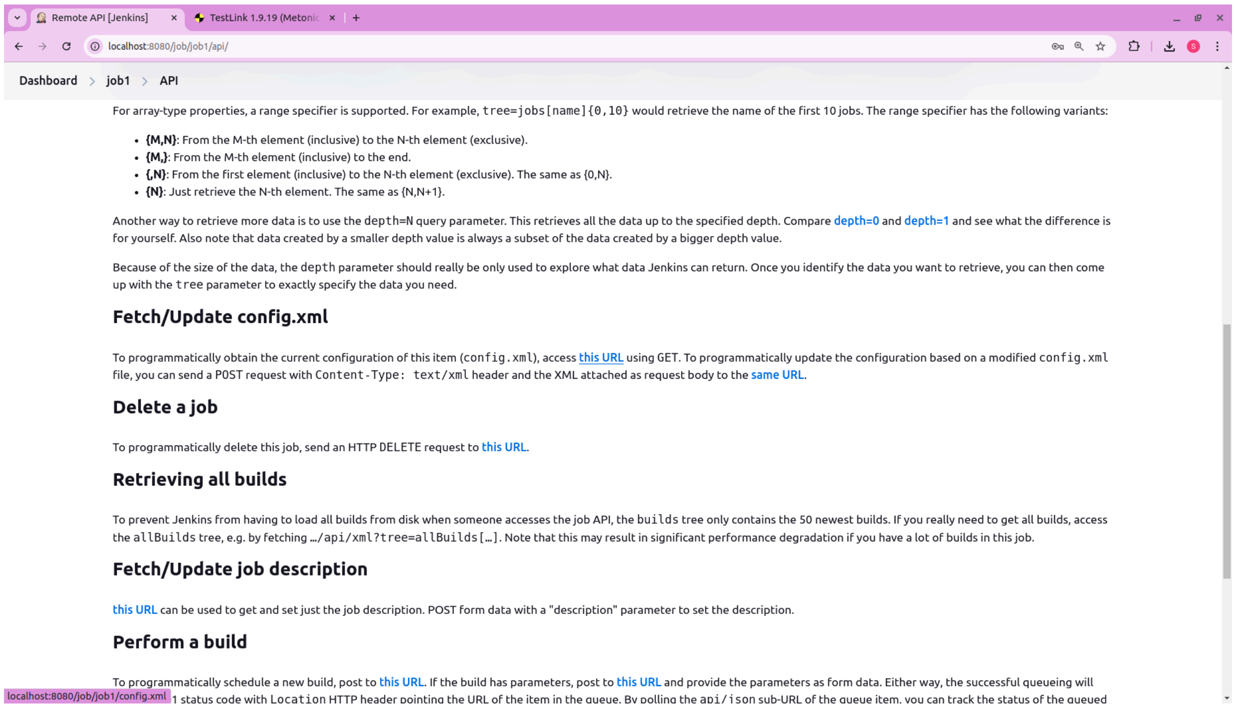
27 January 2025

#11	15:19
#10	15:03
#9	15:02
#8	15:01
#7	15:00



```
echo "This is updated job1"
```

- Click on the “this URL” link in the “Fetch/Update config.xml” section , you will get the xml configuration of the job



For array-type properties, a range specifier is supported. For example, `tree=jobs[name]{0,10}` would retrieve the name of the first 10 jobs. The range specifier has the following variants:

- **{M,N}**: From the M-th element (inclusive) to the N-th element (exclusive).
- **{M,}**: From the M-th element (inclusive) to the end.
- **{,N}**: From the first element (inclusive) to the N-th element (exclusive). The same as {0,N}.
- **{N}**: Just retrieve the N-th element. The same as {N,N+1}.

Another way to retrieve more data is to use the `depth=N` query parameter. This retrieves all the data up to the specified depth. Compare depth=0 and depth=1 and see what the difference is for yourself. Also note that data created by a smaller depth value is always a subset of the data created by a bigger depth value.

Because of the size of the data, the `depth` parameter should really be only used to explore what data Jenkins can return. Once you identify the data you want to retrieve, you can then come up with the `tree` parameter to exactly specify the data you need.

Fetch/Update config.xml

To programmatically obtain the current configuration of this item (`config.xml`), access this URL using GET. To programmatically update the configuration based on a modified `config.xml` file, you can send a POST request with `Content-Type: text/xml` header and the XML attached as request body to the same URL.

Delete a job

To programmatically delete this job, send an HTTP DELETE request to this URL.

Retrieving all builds

To prevent Jenkins from having to load all builds from disk when someone accesses the job API, the `builds` tree only contains the 50 newest builds. If you really need to get all builds, access the `allBuilds` tree, e.g. by fetching …`/api/xml?tree=allBuilds[…]`. Note that this may result in significant performance degradation if you have a lot of builds in this job.

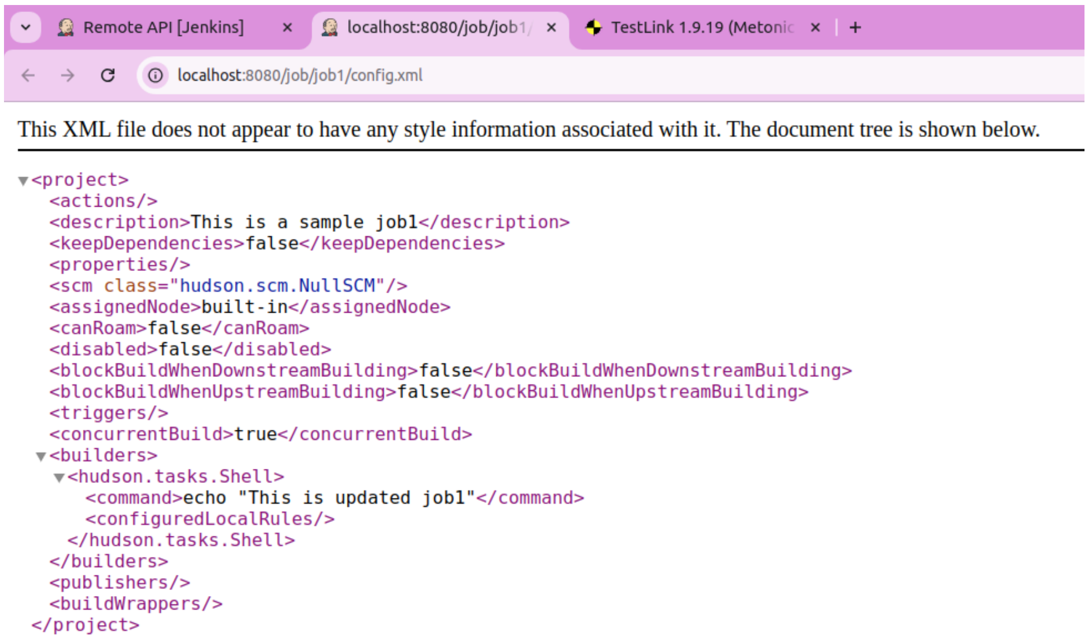
Fetch/Update job description

this URL can be used to get and set just the job description. POST form data with a "description" parameter to set the description.

Perform a build

To programmatically schedule a new build, post to this URL. If the build has parameters, post to this URL and provide the parameters as form data. Either way, the successful queueing will

1 status code with `Location` HTTP header pointing the URL of the item in the queue. By polling the `api/json` sub-URL of the queue item, you can track the status of the queued



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<project>
   <actions/>
   <description>This is a sample job1</description>
   <keepDependencies>false</keepDependencies>
   <properties/>
   <scm class="hudson.scm.NullSCM"/>
   <assignedNode>built-in</assignedNode>
   <canRoam>false</canRoam>
   <disabled>false</disabled>
   <blockBuildWhenDownstreamBuilding>false</blockBuildWhenDownstreamBuilding>
   <blockBuildWhenUpstreamBuilding>false</blockBuildWhenUpstreamBuilding>
   <triggers/>
   <concurrentBuild>true</concurrentBuild>
 ▼<builders>
   ▼<hudson.tasks.Shell>
      <command>echo "This is updated job1"</command>
      <configuredLocalRules/>
    </hudson.tasks.Shell>
   </builders>
   <publishers/>
   <buildWrappers/>
 </project>
```

#2.2.1: sample config_xml files

1. Job_config.xml

```
<project>
   <actions/>
   <description>GPIO tests</description>
   <keepDependencies>false</keepDependencies>
   <properties/>
   <scm class="hudson.scm.NullSCM"/>
   <canRoam>true</canRoam>
   <disabled>false</disabled>
   <blockBuildWhenDownstreamBuilding>false</blockBuildWhenDownstreamBuilding>
   <blockBuildWhenUpstreamBuilding>true</blockBuildWhenUpstreamBuilding>
   <triggers>
       <hudson.triggers.TimerTrigger>
           <spec>50 01 22 01 03</spec>
       </hudson.triggers.TimerTrigger>
   </triggers>
   <concurrentBuild>false</concurrentBuild>
   <builders>
       <hudson.tasks.Shell>
           <command>echo $WORKSPACE</command>
           <configuredLocalRules/>
       </hudson.tasks.Shell>
   </builders>
   <publishers/>
   <buildWrappers/>
</project>
```

2. **Job_config.xml to trigger when upstream job pass**
   Here the threshold values we set in xml are fixed, They are:
   - To trigger job when upstream job pass : name-SUCCESS, ordinal-0, color-BLUE
   - To trigger job when upstream job pass : name-UNSTABLE, ordinal-1, color-YELLOW
   - To trigger job when upstream job pass : name-FAILURE, ordinal-2, color-RED

```
<project>
   <actions/>
   <description>This is a sample job1</description>
   <keepDependencies>false</keepDependencies>
   <properties/>
   <scm class="hudson.scm.NullSCM"/>
   <canRoam>true</canRoam>
   <disabled>false</disabled>
   <blockBuildWhenDownstreamBuilding>false</blockBuildWhenDownstreamBuilding>
   <blockBuildWhenUpstreamBuilding>false</blockBuildWhenUpstreamBuilding>
   <triggers>
       <jenkins.triggers.ReverseBuildTrigger>
           <spec/>
           <upstreamProjects>job2</upstreamProjects>
           <threshold>
               <name>SUCCESS</name>        <!-- use "UNSTABLE" to trigger even
upstream job build has warning  -->
               <ordinal>0</ordinal>        <!-- use '1' for unstable -->
               <color>BLUE</color>         <!-- use 'YELLOW' for unstable -->
               <completeBuild>true</completeBuild>
           </threshold>
       </jenkins.triggers.ReverseBuildTrigger>
   </triggers>
   <concurrentBuild>true</concurrentBuild>        <!-- enable it for concurrent
building -->
   <builders>
       <hudson.tasks.Shell>
           <command>echo "This is updated job1"</command>
           <configuredLocalRules/>
       </hudson.tasks.Shell>
   </builders>
   <publishers/>
   <buildWrappers/>
</project>
```

3. **Job_config.xml to trigger when upstream job fail**

```
<project>
   <actions/>
   <description>This is a sample job1</description>
   <keepDependencies>false</keepDependencies>
   <properties/>
   <scm class="hudson.scm.NullSCM"/>
   <canRoam>true</canRoam>
   <disabled>false</disabled>
   <blockBuildWhenDownstreamBuilding>false</blockBuildWhenDownstreamBuilding>
   <blockBuildWhenUpstreamBuilding>false</blockBuildWhenUpstreamBuilding>
   <triggers>
       <jenkins.triggers.ReverseBuildTrigger>
           <spec/>
           <upstreamProjects>job2</upstreamProjects>
           <threshold>
               <name>FAILURE</name>
               <ordinal>2</ordinal>
               <color>RED</color>
               <completeBuild>false</completeBuild>
           </threshold>
       </jenkins.triggers.ReverseBuildTrigger>
   </triggers>
   <concurrentBuild>false</concurrentBuild>
   <builders>
       <hudson.tasks.Shell>
           <command>echo "This is updated job1"</command>
           <configuredLocalRules/>
       </hudson.tasks.Shell>
   </builders>
   <publishers/>
   <buildWrappers/>
</project>
```

4. **View_config.xml**

```
<?xml version="1.1" encoding="UTF-8"?>
<hudson.model.ListView>
 <name>My new Job List</name>
 <filterExecutors>false</filterExecutors>
 <filterQueue>false</filterQueue>
 <properties class="hudson.model.View$PropertyList"/>
 <jobNames>
   <comparator class="java.lang.String$CaseInsensitiveComparator"/>
   <string>job1</string>
   <string>job2</string>
   <string>job3</string>
 </jobNames>
 <jobFilters/>
 <columns>
   <hudson.views.StatusColumn/>
   <hudson.views.WeatherColumn/>
   <hudson.views.JobColumn/>
   <hudson.views.LastSuccessColumn/>
   <hudson.views.LastFailureColumn/>
   <hudson.views.LastDurationColumn/>
   <hudson.views.BuildButtonColumn/>
 </columns>
 <recurse>false</recurse>
</hudson.model.ListView>
```

5. **Pipeline_comfig.xml**

```
<flow-definition plugin="workflow-job@1498.v33a_0c6f3a_4b_4">
<actions>

<org.jenkinsci.plugins.pipeline.modeldefinition.actions.DeclarativeJobAction
plugin="pipeline-model-definition@2.2218.v56d0cda_37c72"/>

<org.jenkinsci.plugins.pipeline.modeldefinition.actions.DeclarativeJobPropertyT
rackerAction plugin="pipeline-model-definition@2.2218.v56d0cda_37c72">
           <jobProperties/>
           <triggers/>
           <parameters/>
           <options/>

</org.jenkinsci.plugins.pipeline.modeldefinition.actions.DeclarativeJobProperty
TrackerAction>
   </actions>
   <description/>
   <keepDependencies>false</keepDependencies>
   <properties/>
   <definition class="org.jenkinsci.plugins.workflow.cps.CpsFlowDefinition"
plugin="workflow-cps@4009.v0089238351a_9">
       <script>
           pipeline {
               agent any

               stages {
                   stage('Checkout') {
                       steps {
                           checkout scmGit(
                               branches: [[name: '*/main']],
                               extensions: [],
                               userRemoteConfigs: [[credentialsId:
'jenkins-pipeline-python', url:
'https://github.com/lochu-55/Jenkins_Automation.git']]
                           )
                       }
                   }

                   stage('Build') {
                       steps {
                           git branch: 'main', credentialsId:
'jenkins-pipeline-python', url:
'https://github.com/lochu-55/Jenkins_Automation.git'
                           sh 'python3 sample_list_operations.py'
                       }
                   }

                   stage('Test') {
                       steps {
                           echo 'this job has been tested'
                       }
                   }
               }
           }
       </script>
       <sandbox>true</sandbox>
   </definition>
   <triggers/>
   <disabled>false</disabled>
</flow-definition>
```

6. Node_config.xml

```
<slave>
   <name>jyo_gpio</name>
   <description>jyothsna Node's description</description>
   <remoteFS>/home/vlab/jenkins</remoteFS>
   <numExecutors>1</numExecutors>
   <mode>NORMAL</mode>
   <retentionStrategy class="hudson.slaves.RetentionStrategy$Always" />
   <launcher class="hudson.plugins.sshslaves.SSHLauncher"
plugin="ssh-slaves@3.1021.va_cc11b_de26a_e">
       <host>172.16.203.40</host>
       <port>22</port>
       <credentialsId>jyo</credentialsId>
       <launchTimeoutSeconds>60</launchTimeoutSeconds>
       <maxNumRetries>10</maxNumRetries>
       <retryWaitTime>15</retryWaitTime>
       <sshHostKeyVerificationStrategy
class="hudson.plugins.sshslaves.verifiers.NonVerifyingKeyVerificationStrategy"
/>
       <tcpNoDelay>true</tcpNoDelay>
   </launcher>
   <label>jyo</label>
   <nodeProperties />
</slave>
```

#3: Jenkins job builds

In Jenkins, a build refers to the process where Jenkins runs a job to execute a sequence of steps defined by the job's configuration. This could include compiling code, running tests, deploying applications, and other tasks required in a Continuous Integration/Continuous Deployment (CI/CD) pipeline.

#3.1: How Builds Work

- When a job is triggered (either manually, via a webhook, or according to a schedule), Jenkins creates a **build**.
- The **build number** identifies the specific instance of that job's execution. Each time the job is triggered, the build number increments.
- A build can be in various states: **building**, **success**, **failure**, **unstable**, etc.
- Jenkins also provides options to **stop**, **delete**, and **monitor** builds, which are useful in cases where builds take too long, are stuck, or need to be cleaned up.

#3.2: Important Aspects of Jenkins Builds

#3.2.1: Build Queue

- Jenkins has a queue that stores jobs waiting to be built. When a job is triggered, Jenkins first places it in the queue, and once an executor is available, the build is executed.

#3.2.2: Build Triggers

- **Builds can be triggered in various ways such as:**
    - **Manually:** Triggering the build via the Jenkins UI or REST API.
    - **Webhooks:** Automatically triggered when code changes are pushed to a repository (e.g., via GitHub webhooks).
    - **Scheduled Builds:** Triggered at predefined times.
    - **Pipeline Jobs:** Where builds are part of a larger pipeline.

#3.2.3: Build Results

- After a build finishes, Jenkins classifies the result as:
    - **Success:** The job completed successfully.
    - **Failure:** The job failed (usually due to test failures or compilation issues).
      Unstable: The job finished but with some issues, such as failed tests.

    - **Aborted**: The build was manually stopped before completion.

#3.2.4: Build Information:

- Jenkins provides detailed information about each build, such as logs, build artifacts, and test results. You can access this information via the Jenkins web interface or REST API.

#3.2.5: Stopping and Deleting Builds

- You can stop a running build if it's taking too long or if you want to cancel it.
- Once a build completes, you can delete it if it is no longer needed to free up storage or for cleanup purposes.

#3.3: Jenkins Build API Methods

S.No.	API Method	Description	Parameters	Returns
1	**build_job(name, parameters=None, token=None)**	Triggers a build for the specified Jenkins job.	name (str): Job name. parameters (dict, optional): Job parameters. token (str, optional): API token.	Queue item number (str) to track the build.
2	**stop_build(name, number)**	Stops a running Jenkins build.	name (str): Job name. number (int): Build number.	None (Build will be stopped).
3	**delete_build(name, number)** ****Its not working , need to do some changes in code of API to make it work**	Deletes a specific Jenkins build permanently.	name (str): Job name. number (int): Build number.	None (Build will be deleted).

4	**get_running_builds()**	Retrieves a list of all currently running builds.	No parameters.	List of running builds (list of dicts with keys: name, number, url, node, executor).
5	**get_build_info(name, number)**	Retrieves detailed information about a specific build.	name (str): Job name. number (int): Build number.	Build information (dict) including logs, results, etc.
6	**get_build_console_output(name, number)**	Retrieves the console output (logs) of a specific build.	name (str): Job name. number (str or int): Build number.	Console output text (logs).
7	**get_build_test_report(name, number, depth=0)**	Retrieves the test results report for a specific build.	name (str): Job name. number (str or int): Build number. depth (int, optional): JSON depth (default: 0).	Test report (dict) with details of passed/failed tests.
8	**get_build_env_vars(name, number, depth=0)**	Get build environment variables.	name(str): Job name, str number(int): – Build number, str (also accepts int) depth: JSON depth, int	dictionary of build env vars, dict or None for workflow jobs, or if InjectEnvVars plugin not installed

#3.3.1: delete_build API issue

- changes to be done in delete_build API code are :
    - Go to .venv/lib/site-packages/jenkins/__init__.py
    - Edit the delete_build() function and wipeout_job_workspace()

```python
def delete_build(self, name, number):
    """Delete a Jenkins build.

    :param name: Name of Jenkins job, ``str``
    :param number: Jenkins build number for the job, ``int``
    """
    folder_url, short_name = self._get_job_folder(name)
    self.jenkins_open(requests.Request('POST',
                      self._build_url(DELETE_BUILD, locals()),b''))

def wipeout_job_workspace(self, name):
    """Wipe out workspace for given Jenkins job.

    :param name: Name of Jenkins job, ``str``
    """
    folder_url, short_name = self._get_job_folder(name)
    self.jenkins_open(requests.Request('POST',
                      self._build_url(WIPEOUT_JOB_WORKSPACE,
                                      locals()),b''))
```

    - Remove the b'' beside the local() in _build_url function

```python
def delete_build(self, name, number):
    """Delete a Jenkins build.

    :param name: Name of Jenkins job, ``str``
    :param number: Jenkins build number for the job, ``int``
    """
    folder_url, short_name = self._get_job_folder(name)
    self.jenkins_open(requests.Request('POST',
                      self._build_url(DELETE_BUILD, locals())))

def wipeout_job_workspace(self, name):
    """Wipe out workspace for given Jenkins job.
    :param name: Name of Jenkins job, ``str``
    """
    folder_url, short_name = self._get_job_folder(name)
    self.jenkins_open(requests.Request('POST',
                          self._build_url(WIPEOUT_JOB_WORKSPACE,
                                          locals())))
```

- Edit the maybe_add_crumb() function

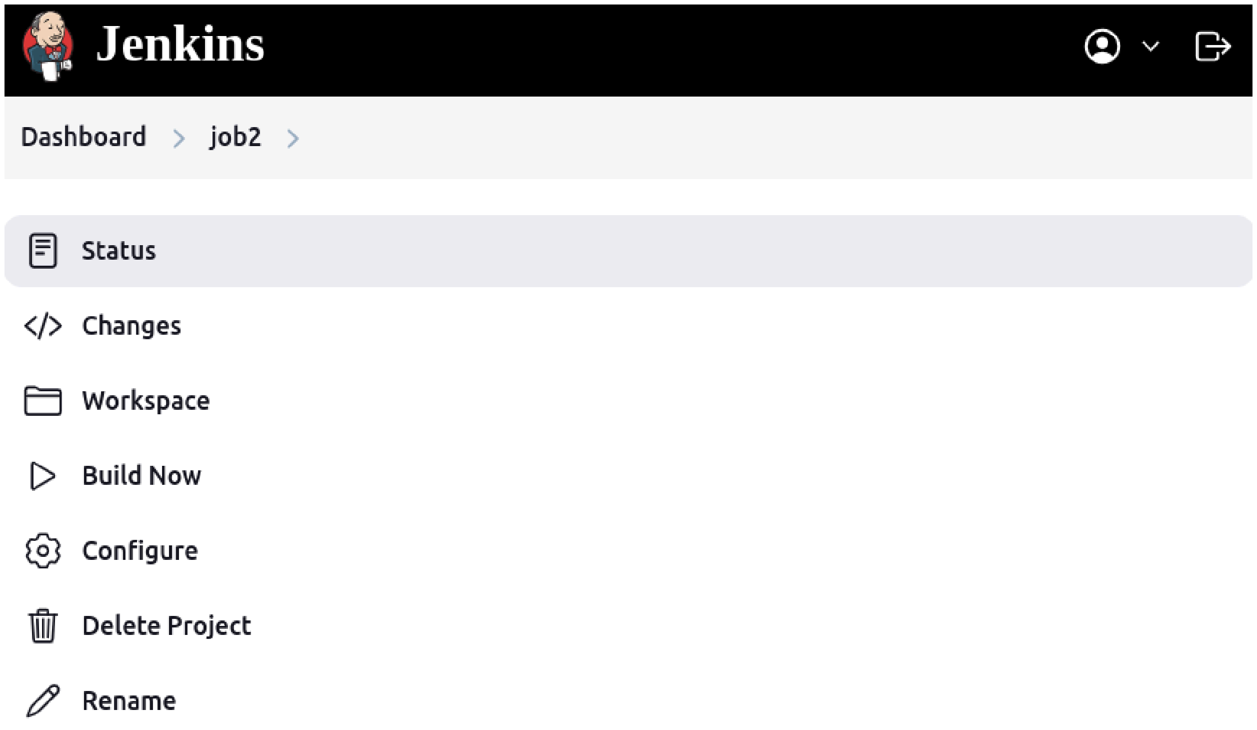
```
def maybe_add_crumb(self, req):
    # We don't know yet whether we need a crumb
    if self.crumb is None:
        try:
            response = self.jenkins_open(requests.Request(
                'GET', self._build_url(CRUMB_URL)), add_crumb=False)
            if not response:
                raise EmptyResponseException("Empty response for
crumb")
        except (NotFoundException, EmptyResponseException):
            self.crumb = False
        else:
            self.crumb = json.loads(response)
    if self.crumb:
        req.headers[self.crumb['crumbRequestField']] =
self.crumb['crumb']
```

- Replace “if self.crumb:” with “if self.crumb and isinstance(req.headers, dict):”

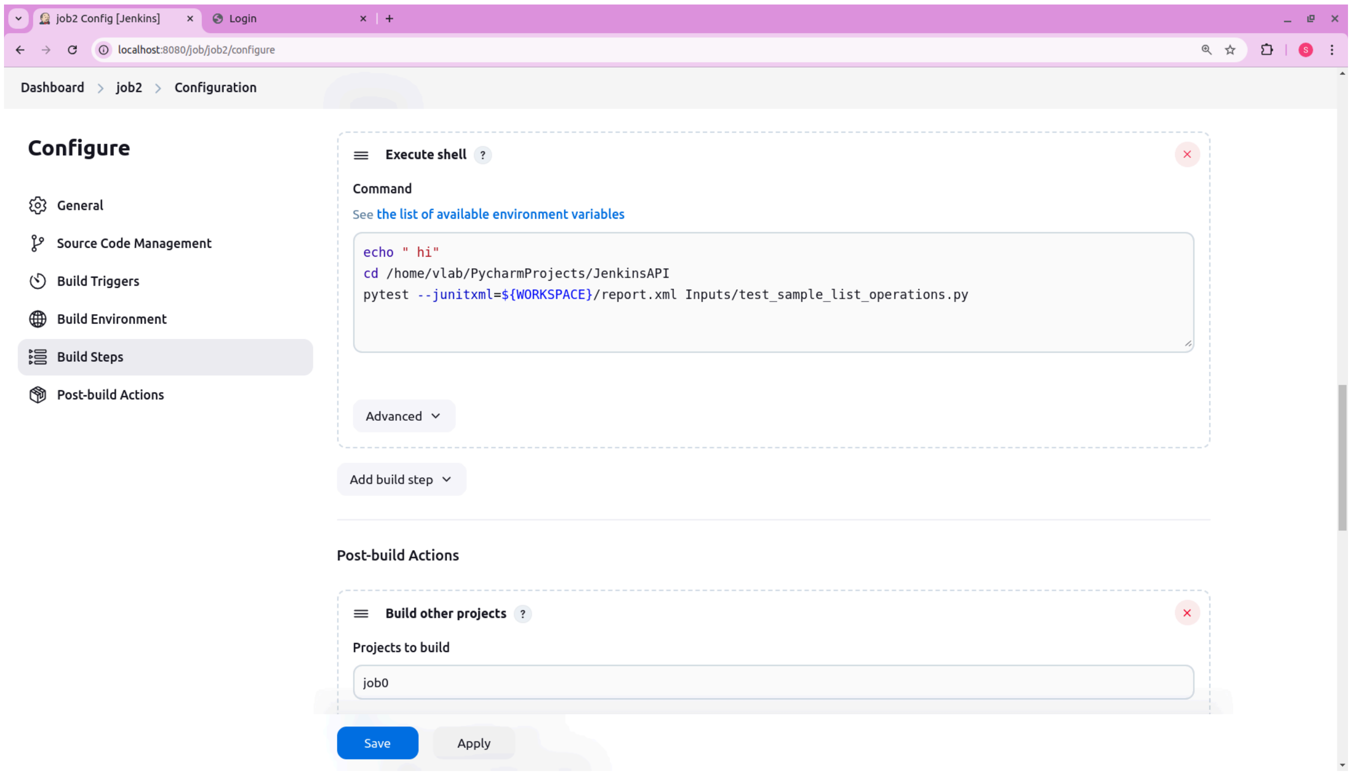
```
def maybe_add_crumb(self, req):
    # We don't know yet whether we need a crumb
    if self.crumb is None:
        try:
            response = self.jenkins_open(requests.Request(
                'GET', self._build_url(CRUMB_URL)), add_crumb=False)
            if not response:
                raise EmptyResponseException("Empty response for
crumb")
        except (NotFoundException, EmptyResponseException):
            self.crumb = False
        else:
            self.crumb = json.loads(response)
    if self.crumb and isinstance(req.headers, dict):
        req.headers[self.crumb['crumbRequestField']] =
self.crumb['crumb']
```

#3.3.2: Creating test reports

- Go to configure option of the job in jenkins



- You must use any testing library in build steps to generate the .xml reports like pytest
    - `pytest --junitxml=${WORKSPACE}/report.xml Inputs/test_sample_list_operations.py`
    - `${WORKSPACE} = /var/lib/jenkins/workspace/job`


Dashboard > job2 > Configuration

Configure

General
Source Code Management
Build Triggers
Build Environment
Build Steps
Post-build Actions

Execute shell

Command

See the list of available environment variables

```
echo " hi"
cd /home/vlab/PycharmProjects/JenkinsAPI
pytest --junitxml=${WORKSPACE}/report.xml Inputs/test_sample_list_operations.py
```

Advanced

Add build step

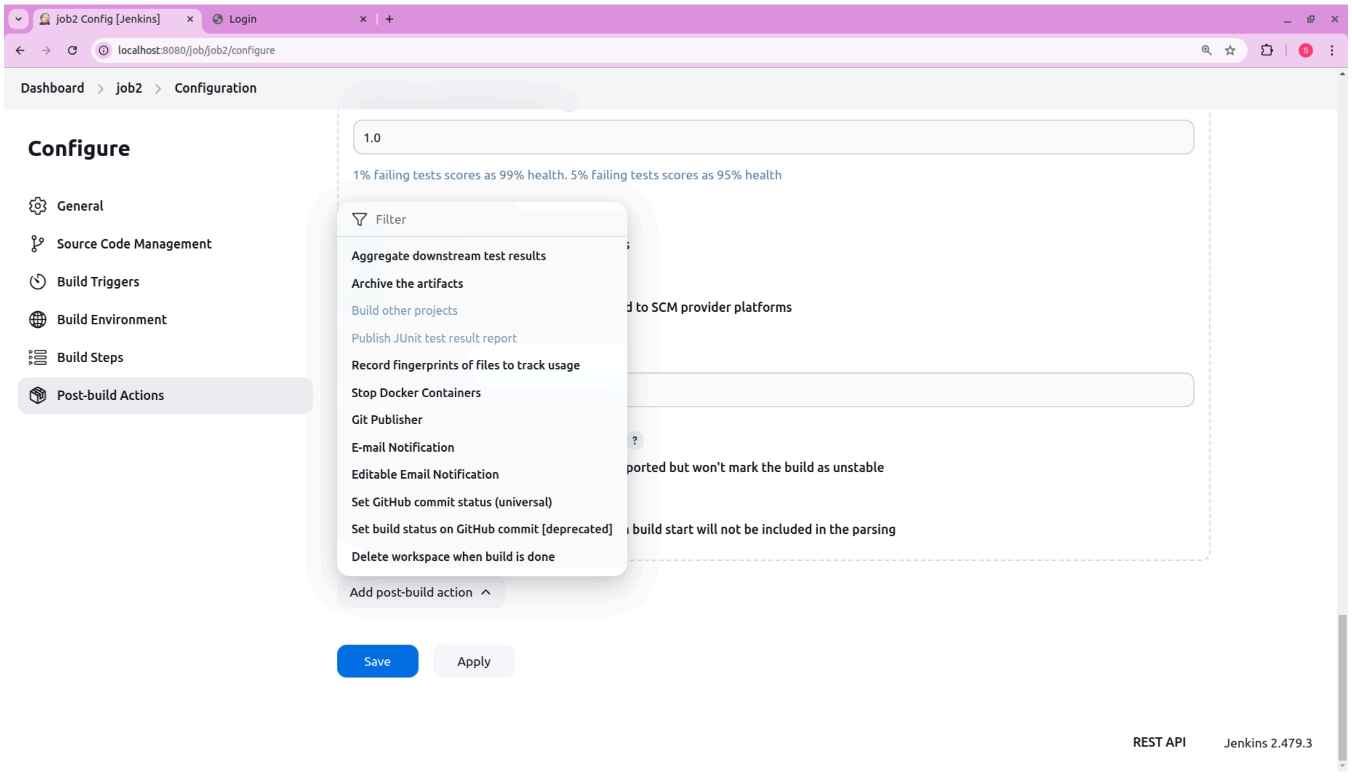
Post-build Actions

Build other projects

Projects to build: job0

Save Apply

- Now build the job to get the .xml report
- In next build to generate a test , select the “Publish JUnit test REsult report” option in Post-build actions


Dashboard > job2 > Configuration

Configure

General
Source Code Management
Build Triggers
Build Environment
Build Steps
Post-build Actions

1.0

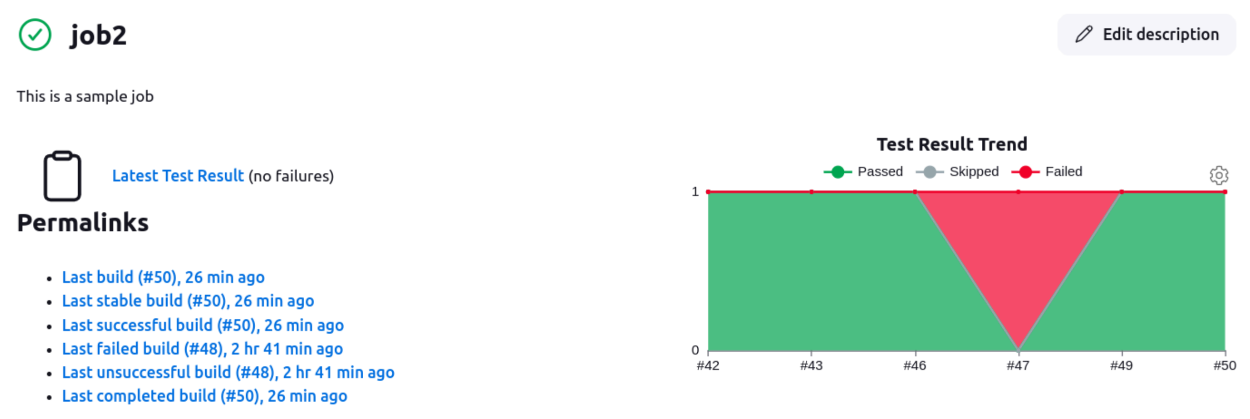
1% failing tests scores as 99% health. 5% failing tests scores as 95% health

Filter
Aggregate downstream test results
Archive the artifacts
Build other projects
Publish JUnit test result report
Record fingerprints of files to track usage
Stop Docker Containers
Git Publisher
E-mail Notification
Editable Email Notification
Set GitHub commit status (universal)
Set build status on GitHub commit [deprecated]
Delete workspace when build is done

Add post-build action

Save Apply

REST API Jenkins 2.479.3



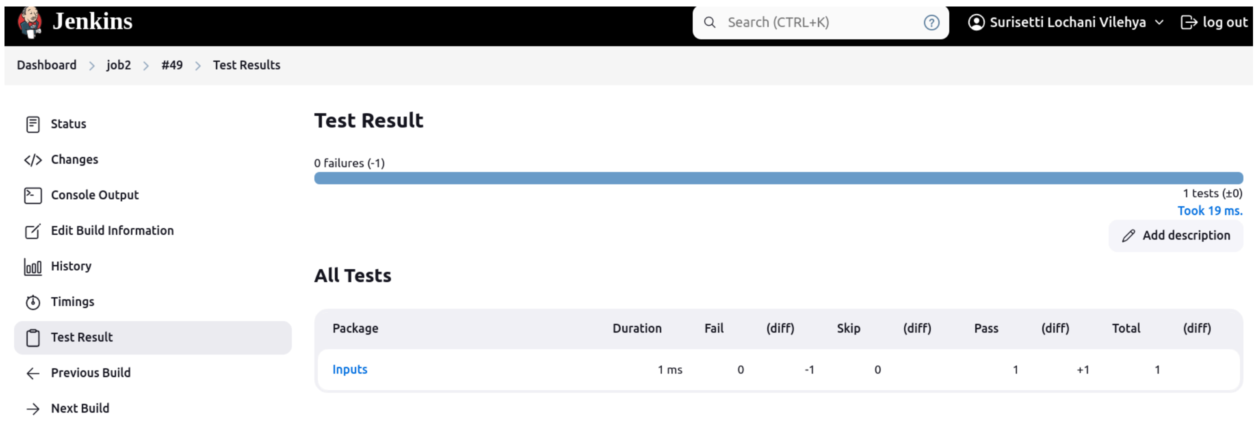
job2

This is a sample job

Latest Test Result (no failures)

Permalinks

- Last build (#50), 26 min ago
- Last stable build (#50), 26 min ago
- Last successful build (#50), 26 min ago
- Last failed build (#48), 2 hr 41 min ago
- Last unsuccessful build (#48), 2 hr 41 min ago
- Last completed build (#50), 26 min ago



Test Result

0 failures (-1)

1 tests (±0)
Took 19 ms.

All Tests

Package	Duration	Fail	(diff)	Skip	(diff)	Pass	(diff)	Total	(diff)
Inputs	1 ms	0	-1	0		1	+1	1	

#3.3.3 get_build_artifact API issue

- Get artifacts from job
- Parameters:
  name – Job name, str
  number – Build number, str (also accepts int)
  artifact – Artifact relative path, str
- Returns: artifact to download, dict

- But This API is not working , may be some issue in the development of the API side
- Alternatively we may use other library named **JenkinsAPI** for retrieving the artifacts of job build info

#4: Artifacts in Jenkins

#4.1: What Are Artifacts in Jenkins?

- Artifacts in Jenkins are files that are generated as part of a job's build process and need to be saved for future use. These files can include:
    - Executable binaries (.jar, .war, .zip)
    - Logs (.log, .txt)
    - Reports (.xml, .json, .html)
    - Test results (.html, JUnit reports)
    - Compiled code or deployment packages

#4.2: How to Create Artifacts in Jenkins?

To store and retrieve artifacts, follow these steps:

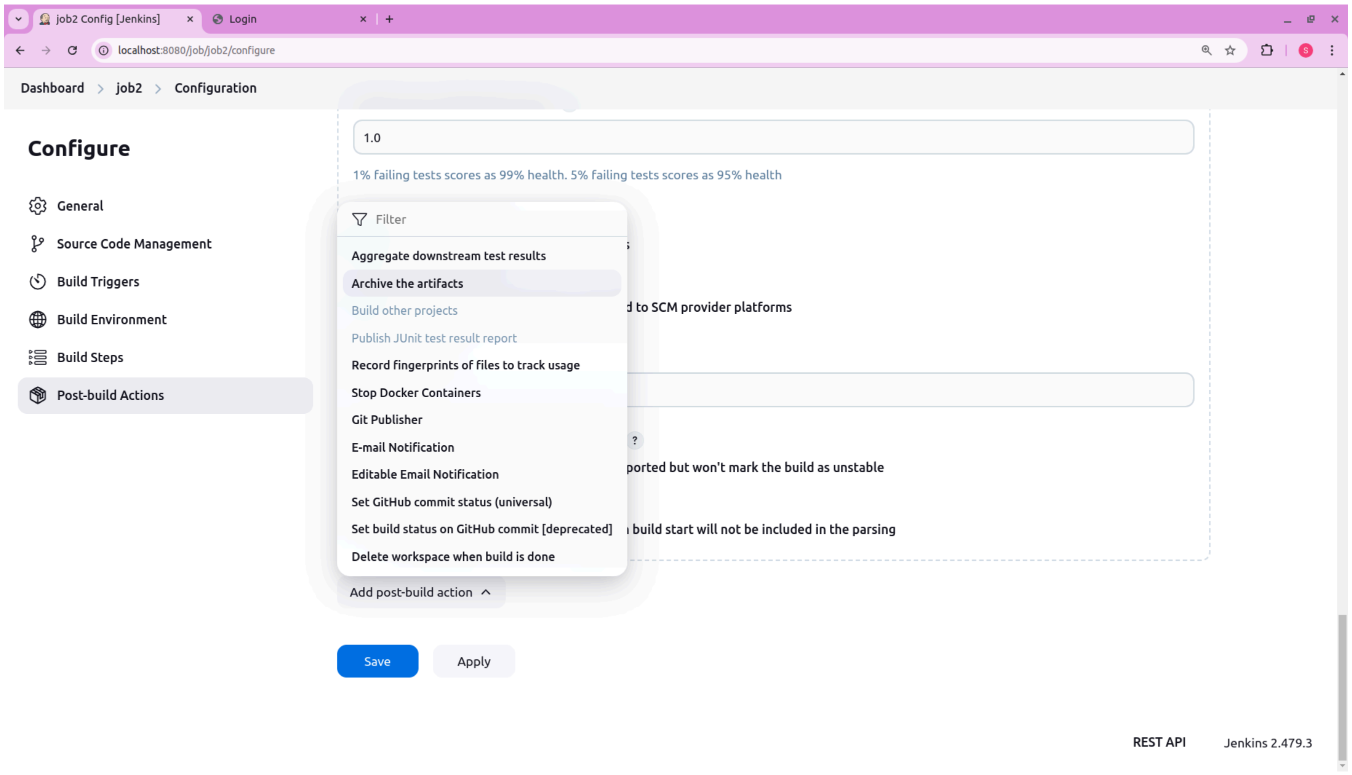
Step 1: Generate the Output Files

- Before archiving, ensure that your build process produces output files (artifacts). These can be created using scripts such as:

```
$ echo "This is my build output" > output.txt —> produces the output test file
$ pytest --junitxml=${WORKSPACE}/report.xml
Inputs/test_sample_list_operations.py —> produces the .xml file as output
```

Step 2: Configure Artifact Archiving post build action in Jenkins

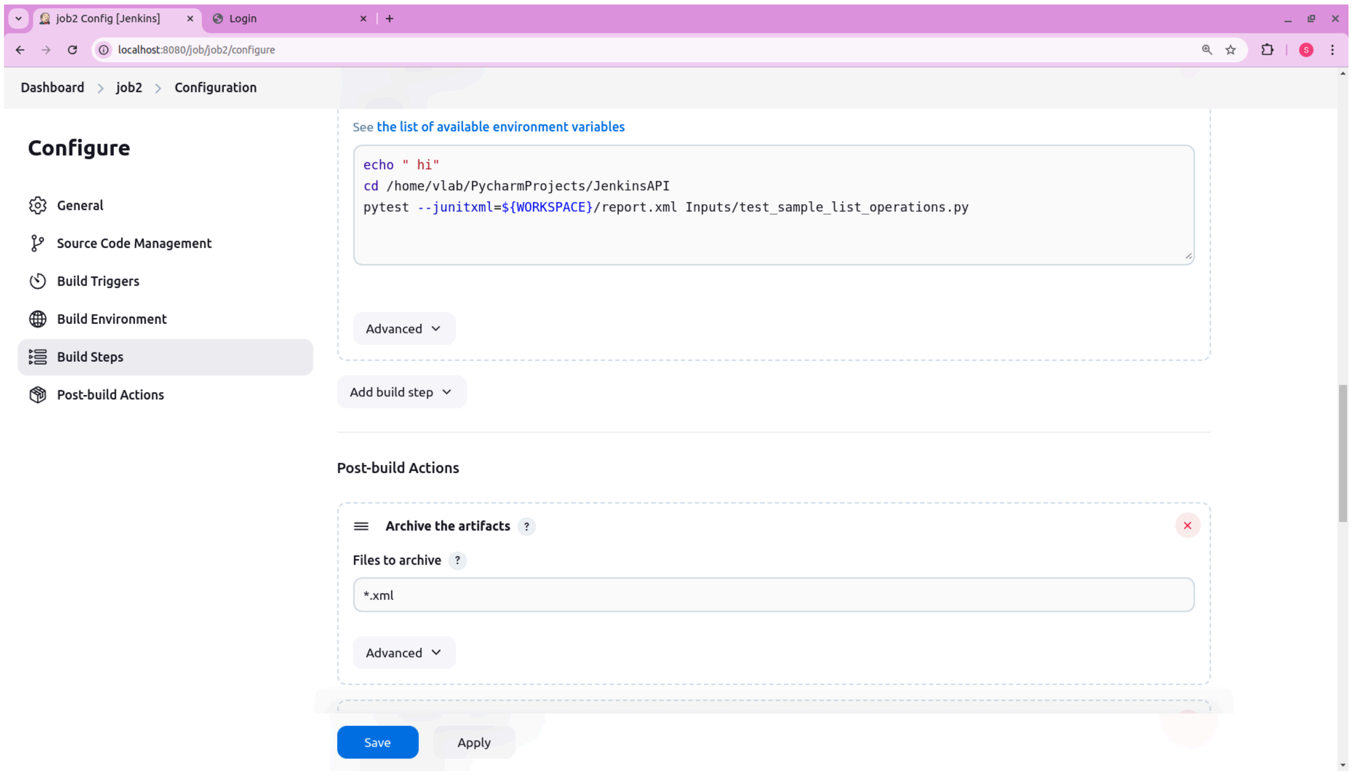
- Go to Jenkins Dashboard → Select your Job.
- Click on "Configure".
- Scroll down to "Post-build Actions".
- Select "Archive the artifacts".


job2 Config [Jenkins]
Login
localhost:8080/job/job2/configure
Dashboard > job2 > Configuration
Configure
General
Source Code Management
Build Triggers
Build Environment
Build Steps
Post-build Actions
Filter
Aggregate downstream test results
Archive the artifacts
Build other projects
Publish JUnit test result report
Record fingerprints of files to track usage
Stop Docker Containers
Git Publisher
E-mail Notification
Editable Email Notification
Set GitHub commit status (universal)
Set build status on GitHub commit [deprecated]
Delete workspace when build is done
Add post-build action
Save
Apply
REST API
Jenkins 2.479.3

1.0

1% failing tests scores as 99% health. 5% failing tests scores as 95% health

- In the "Files to archive" field, enter the path of the file(s) to be archived.
- Example:
  - *.log (Archives all .log files)
  - output.txt (Archives a specific file)
  - reports/*.xml (Archives all XML files in reports/ folder)
- Click "Save".


Dashboard > job2 > Configuration

Configure

General
Source Code Management
Build Triggers
Build Environment
Build Steps
Post-build Actions

See the list of available environment variables

```
echo " hi"
cd /home/vlab/PycharmProjects/JenkinsAPI
pytest --junitxml=${WORKSPACE}/report.xml Inputs/test_sample_list_operations.py
```

Advanced

Add build step

Post-build Actions

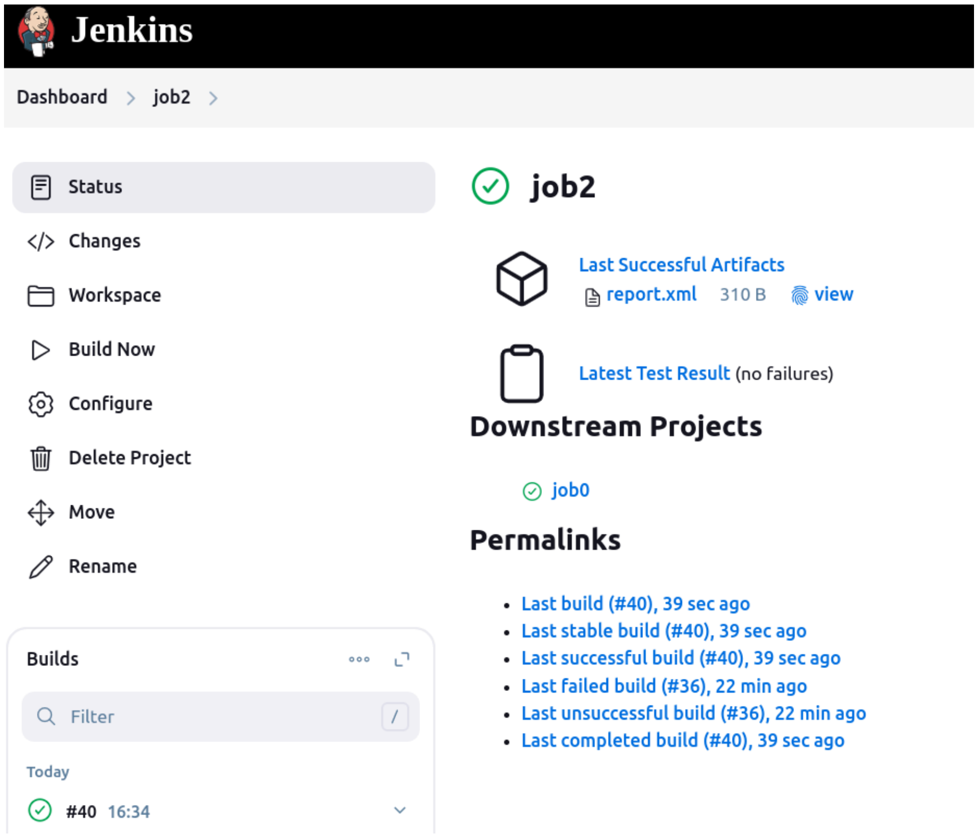
Archive the artifacts

Files to archive

*.xml

Advanced

Save Apply


Jenkins

Dashboard > job2 >

Status
Changes
Workspace
Build Now
Configure
Delete Project
Move
Rename

job2

Last Successful Artifacts
report.xml 310 B view

Latest Test Result (no failures)

Downstream Projects

job0

Permalinks

- Last build (#40), 39 sec ago
- Last stable build (#40), 39 sec ago
- Last successful build (#40), 39 sec ago
- Last failed build (#36), 22 min ago
- Last unsuccessful build (#36), 22 min ago
- Last completed build (#40), 39 sec ago

Builds

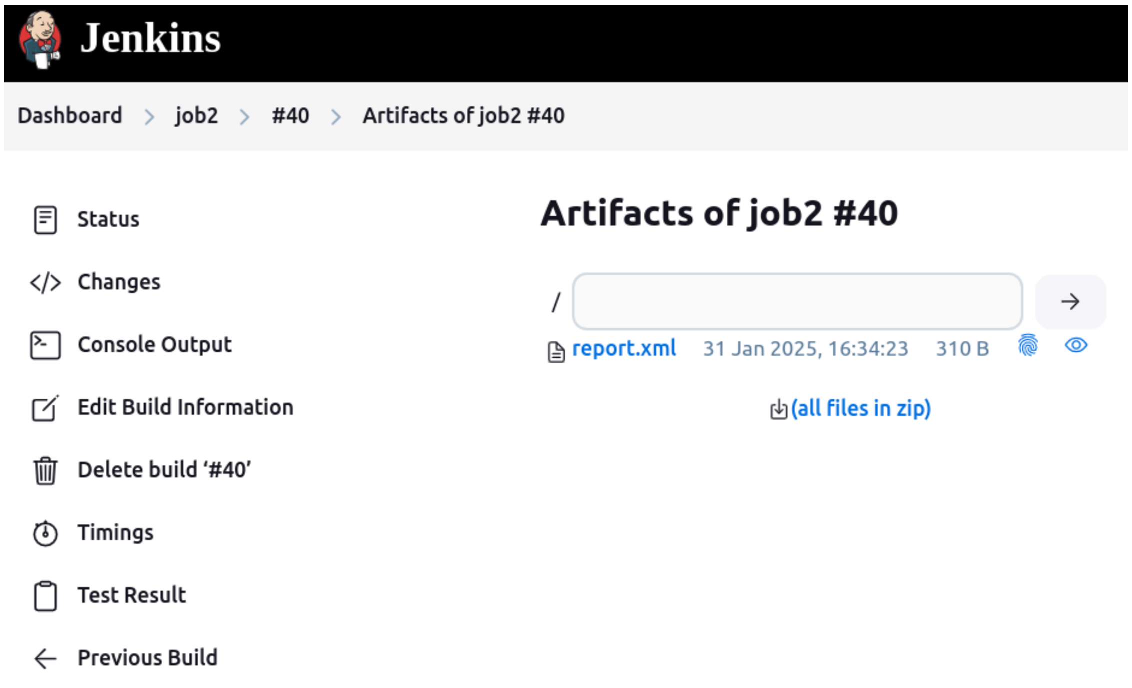
Filter

Today

#40 16:34

#4.3: How to Access Artifacts?

- Once the build completes, artifacts are available in the "Build Artifacts" section of the job.
    - Navigate to Your Job
    - Click on the Latest Build → Artifacts
    - Download the stored files



Artifacts of job2 #40

/

report.xml 31 Jan 2025, 16:34:23 310 B

(all files in zip)

#4.4: Using Jenkins REST API to Retrieve Artifacts

- You can fetch artifacts programmatically using Jenkins API.

```
from jenkinsapi.jenkins import Jenkins

# Connect to Jenkins
server = Jenkins("http://your-jenkins-url:8080", username="admin",
password="your-api-token")

# Get artifacts from last build
job = server.get_job("My_Job")
build = job.get_last_build()
for artifact in build.get_artifacts():
    artifact.save('downloads/')
```

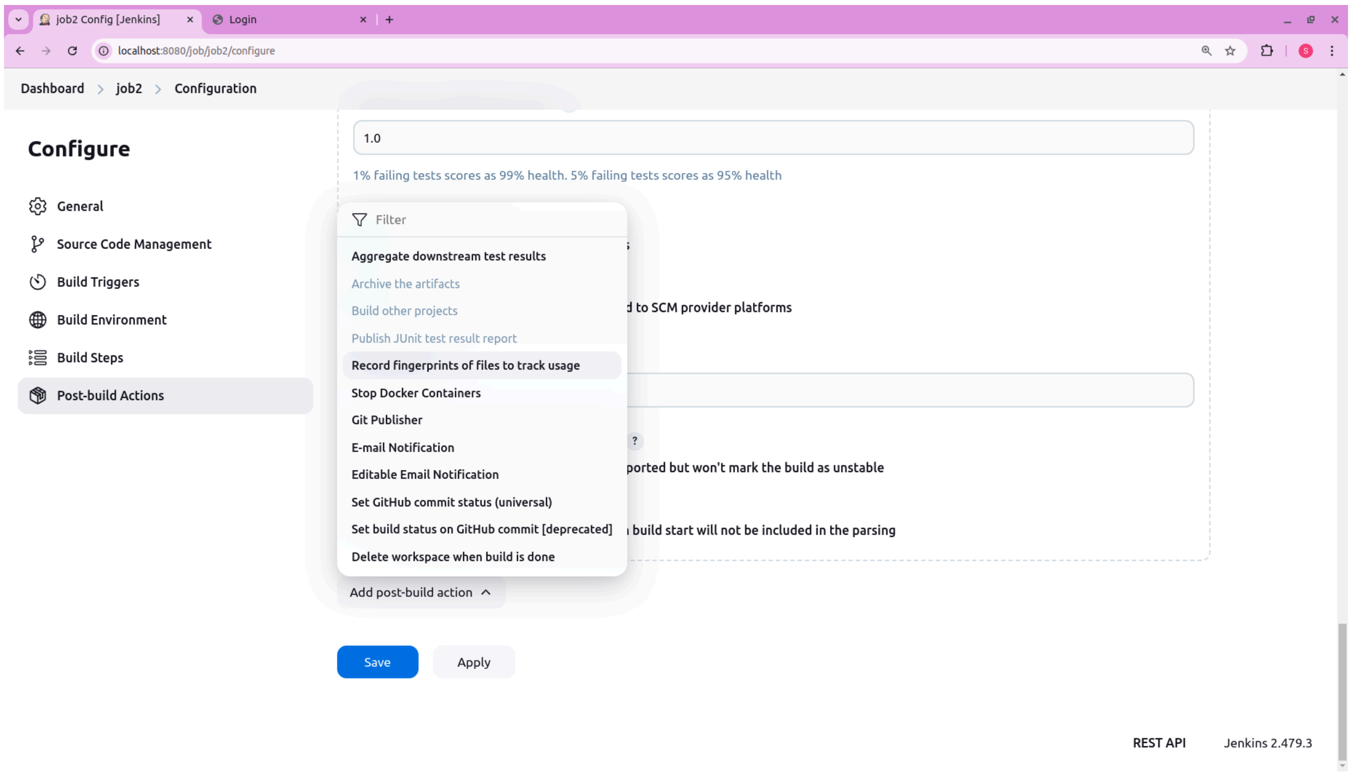
- While running this above code you may getting a warning or an error like :

```
MD5 cannot be checked if fingerprints are not enabled
```

- So , we must enable the fingerprints option in the configuration of the job before the build

#4.5: Enabling the fingerprints for Artifacts

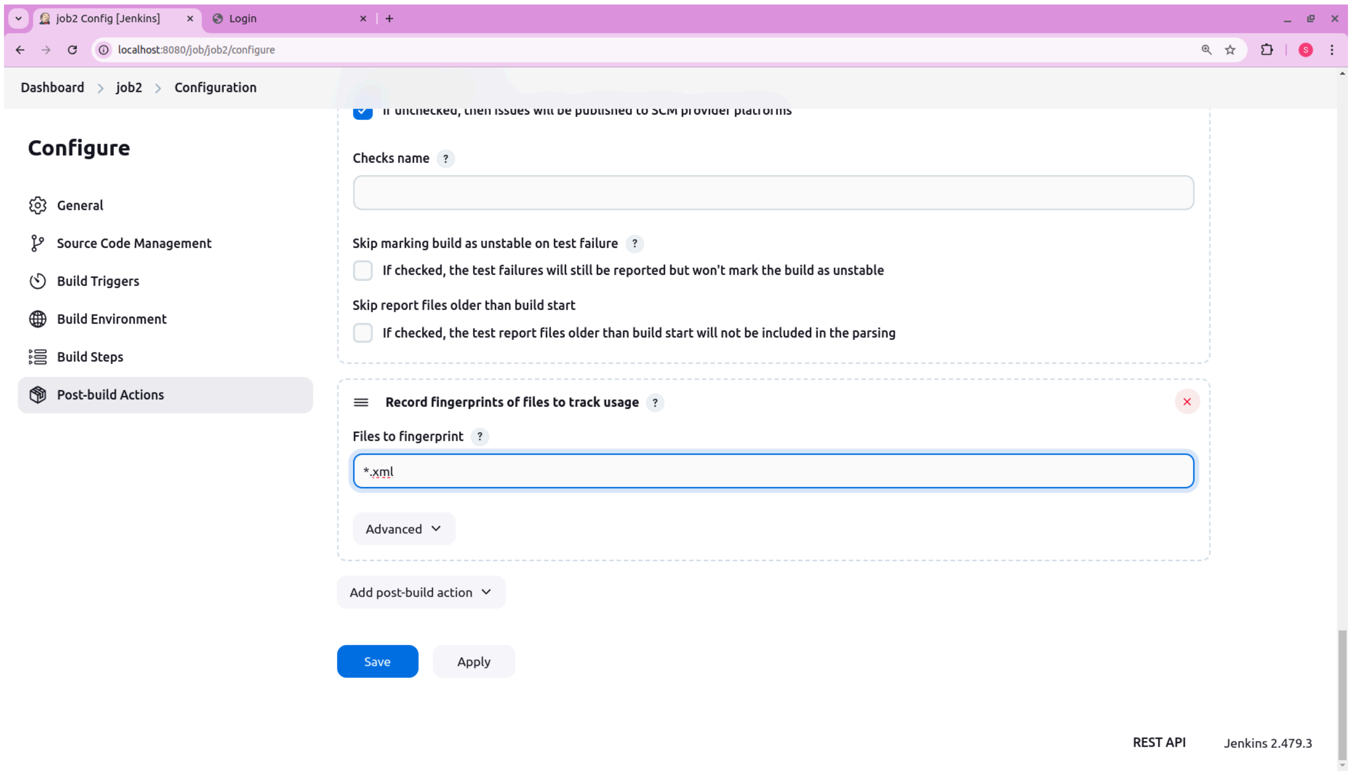
- Navigate to the job where you want to enable fingerprints (e.g., My_Job).
- On the left sidebar, click Configure.
- Scroll down to the Post-build Actions section and click Add post-build action
- There you will see the option called "Record fingerprints of files to track usage"



1.0

1% failing tests scores as 99% health. 5% failing tests scores as 95% health

- Enter the path of the file(s) to which you want to add fingerprint.


Dashboard > job2 > Configuration

Configure

- General
- Source Code Management
- Build Triggers
- Build Environment
- Build Steps
- Post-build Actions

[✓] If unchecked, then issues will be published to SCM provider platforms

Checks name

Skip marking build as unstable on test failure

[ ] If checked, the test failures will still be reported but won't mark the build as unstable

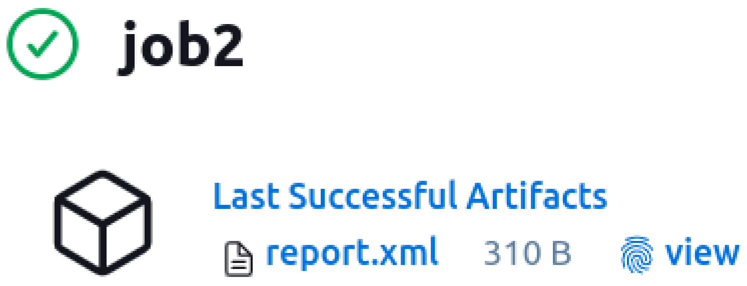
Skip report files older than build start

[ ] If checked, the test report files older than build start will not be included in the parsing

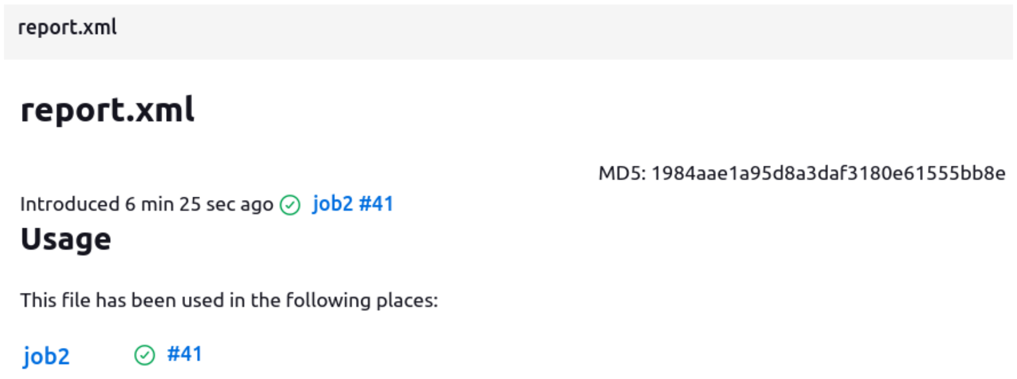
Record fingerprints of files to track usage

Files to fingerprint: *.xml

- After configuring the job, scroll down and click the **Save** button to save the job configuration.
- Run a build on the job.
- After the build completes, go to the **Build** section of the job. Where you will find the "Last successful artifacts section" with the files saved



- Click on the fingerprint symbol beside the file to view MD5 value assigned to it


report.xml

report.xml

MD5: 1984aae1a95d8a3daf3180e61555bb8e

Introduced 6 min 25 sec ago job2 #41

Usage

This file has been used in the following places:

job2 #41

#4.6: Jenkins Artifacts API methods and execution

1. Since , the **"get_build_artifact"** method in the **python-jenkins** package is not working as expected for getting artifacts, and we can use the **jenkinsapi** package instead
2. Install jenkinsapi package
   ```
   $ pip install jenkinsapi
   ```
3. First connect to the jenkins server using Jenkins class from the jenkinsapi package
   ```
   jenkins = Jenkins(jenkins_url, username=username, password=password)
   ```
4. Then, create a job object using "get_job(name)" method
   ```
   job = jenkins.get_job(job_name)
   ```
5. Create a Build Object in Jenkins using get_build() method of job object
   ```
   build = job.get_build(build_number)
   ```
6. Now using the methods of build objects we can retrieve the artifacts info and save them to a local machine. They are:

S.No	Method Name	Description	Parameters	Returns
1	**get_artifact_dict()**	Retrieves all artifacts of a specific Jenkins build as a dictionary, where keys are artifact names and values are Artifact objects.	None	Dictionary {artifact_name: ArtifactObj}

2	**save(fspath, strict_validation=False)**	Saves a specific artifact to a given file path, including the filename. The directory must exist before calling this method.	- fspath (str) → Full file path with filename (e.g., /home/user/artifacts/output.xml). - strict_validation (bool, default=False) → Whether to validate.	File path where artifact is saved (str).
3	**save_to_dir(dirpath, strict_validation=False)**	Saves an artifact into a directory, using its default filename from Jenkins. The directory must exist.	- dirpath (str) → Directory path where the artifact should be saved. - strict_validation (bool, default=False) → Whether to validate.	File path where artifact is saved (str).

#5: Views in Jenkins

#5.1: What Are Jenkins Views?

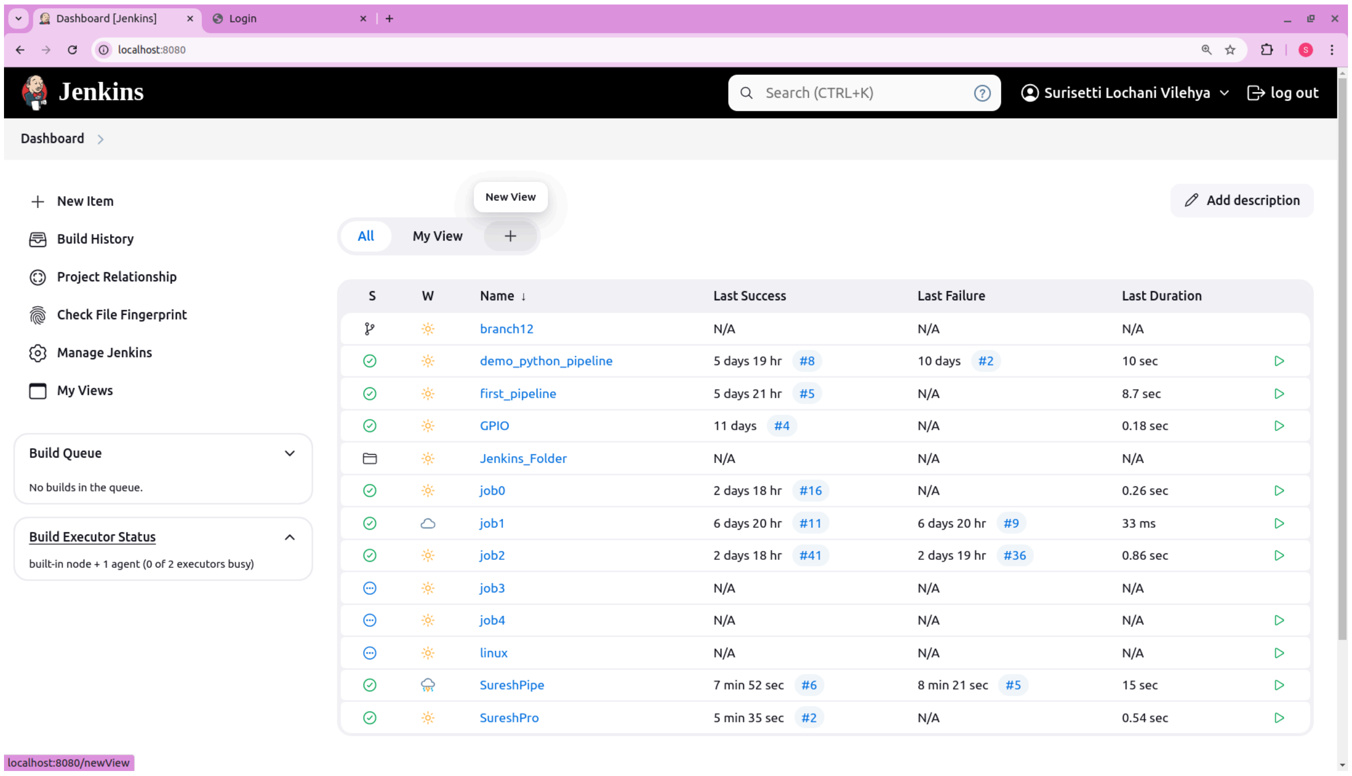
Jenkins **Views** help organize and categorize jobs within the Jenkins dashboard. They allow users to group jobs based on projects, teams, or any logical categorization, improving job visibility and manageability.

#5.2: Types of Views in Jenkins

- **List View**
  Shows items in a simple list format. You can choose which jobs are to be displayed in which view.
- **My View**
  This view automatically displays all the jobs that the current user has access to.

#5.3: How to Create a View in Jenkins

1. Navigate to the Jenkins Dashboard
2. Click on **"New View"** (the '+' symbol beside all) in the left sidebar.


Dashboard [Jenkins] | Login

localhost:8080

Jenkins | Search (CTRL+K) | Surisetti Lochani Vilehya | log out

Dashboard >

- New Item
- Build History
- Project Relationship
- Check File Fingerprint
- Manage Jenkins
- My Views

Build Queue

No builds in the queue.

Build Executor Status

built-in node + 1 agent (0 of 2 executors busy)

New View

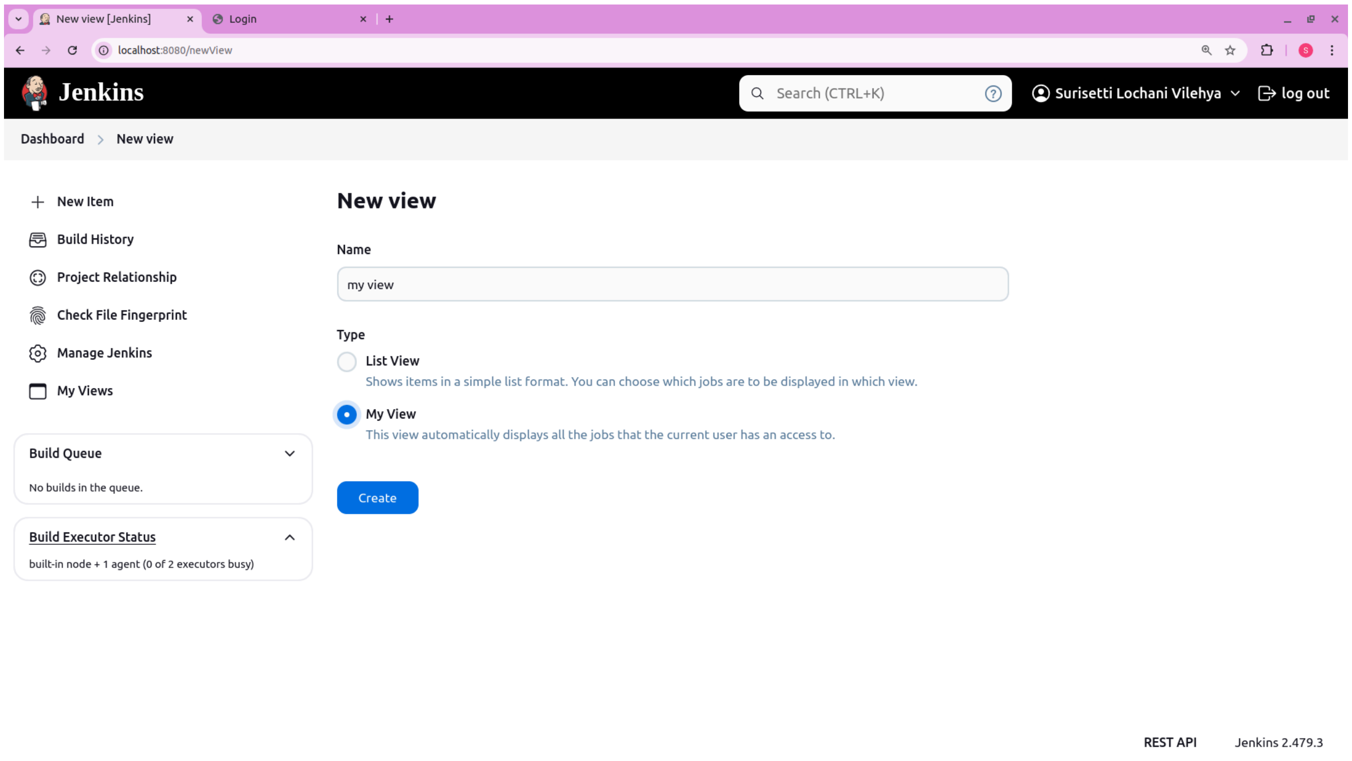
All | My View | +

Add description

S	W	Name ↓	Last Success	Last Failure	Last Duration
	☀	branch12	N/A	N/A	N/A
✓	☀	demo_python_pipeline	5 days 19 hr #8	10 days #2	10 sec
✓	☀	first_pipeline	5 days 21 hr #5	N/A	8.7 sec
✓	☀	GPIO	11 days #4	N/A	0.18 sec
	☀	Jenkins_Folder	N/A	N/A	N/A
✓	☀	job0	2 days 18 hr #16	N/A	0.26 sec
✓	☁	job1	6 days 20 hr #11	6 days 20 hr #9	33 ms
✓	☀	job2	2 days 18 hr #41	2 days 19 hr #36	0.86 sec
⋯	☀	job3	N/A	N/A	N/A
⋯	☀	job4	N/A	N/A	N/A
⋯	☀	linux	N/A	N/A	N/A
✓	☁	SureshPipe	7 min 52 sec #6	8 min 21 sec #5	15 sec
✓	☀	SureshPro	5 min 35 sec #2	N/A	0.54 sec

localhost:8080/newView

3. Enter a **View Name** (e.g., "Team Projects").
4. Select a view type:
    - **List View** (default)
    - **My View** (for personal job tracking)


New view [Jenkins] | Login

localhost:8080/newView

Jenkins | Search (CTRL+K) | Surisetti Lochani Vilehya | log out

Dashboard > New view

- New Item
- Build History
- Project Relationship
- Check File Fingerprint
- Manage Jenkins
- My Views

Build Queue

No builds in the queue.

Build Executor Status

built-in node + 1 agent (0 of 2 executors busy)

New view

Name: my view

Type

List View [✓ My View]

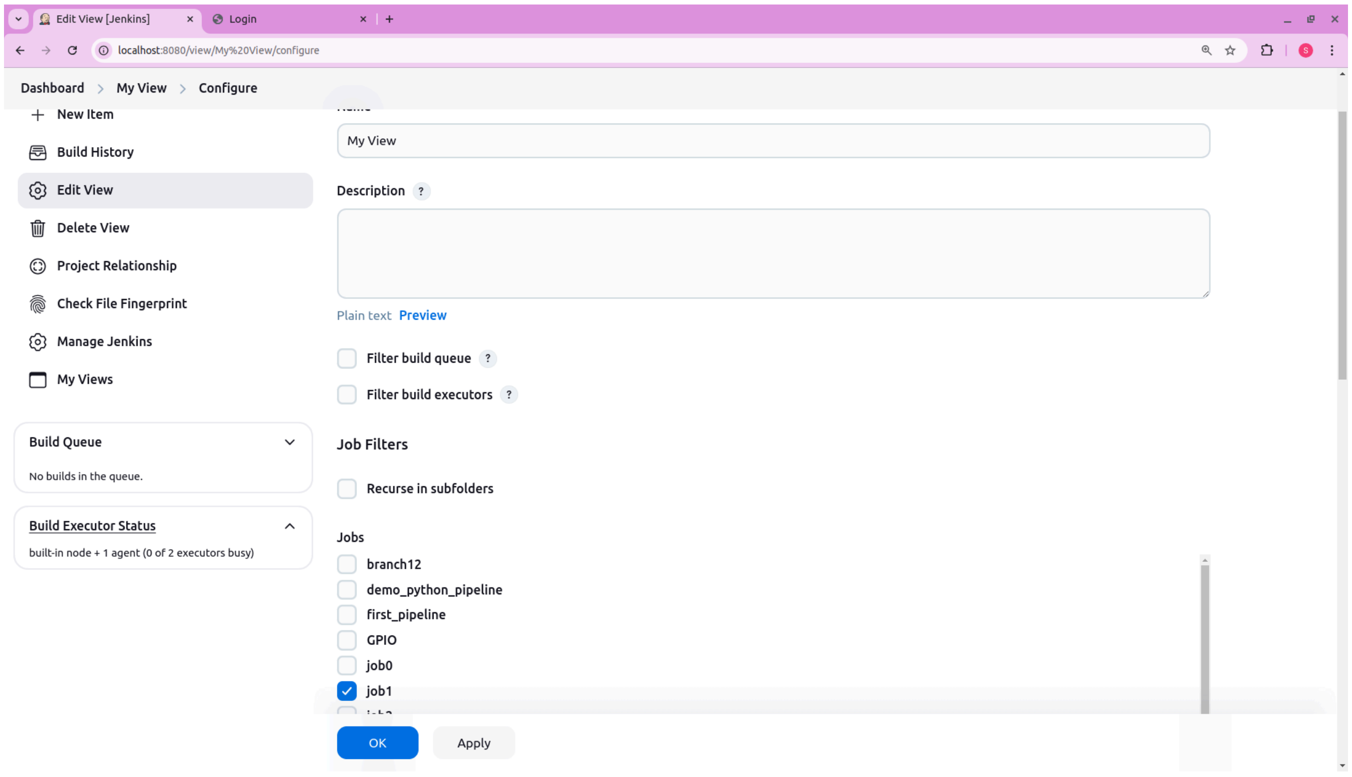
List View: Shows items in a simple list format. You can choose which jobs are to be displayed in which view.

My View: This view automatically displays all the jobs that the current user has an access to.

Create

REST API | Jenkins 2.479.3

5. If you choose list type , mention the job filters to select the jobs to be included in the newly created view
6. Optionally configure **columns**, **sorting**, and **grouping** (for the appearance of the jobs in dashboard)


Edit View [Jenkins] | Login

localhost:8080/view/My%20View/configure

Dashboard > My View > Configure

Name: My View

Description

Plain text Preview

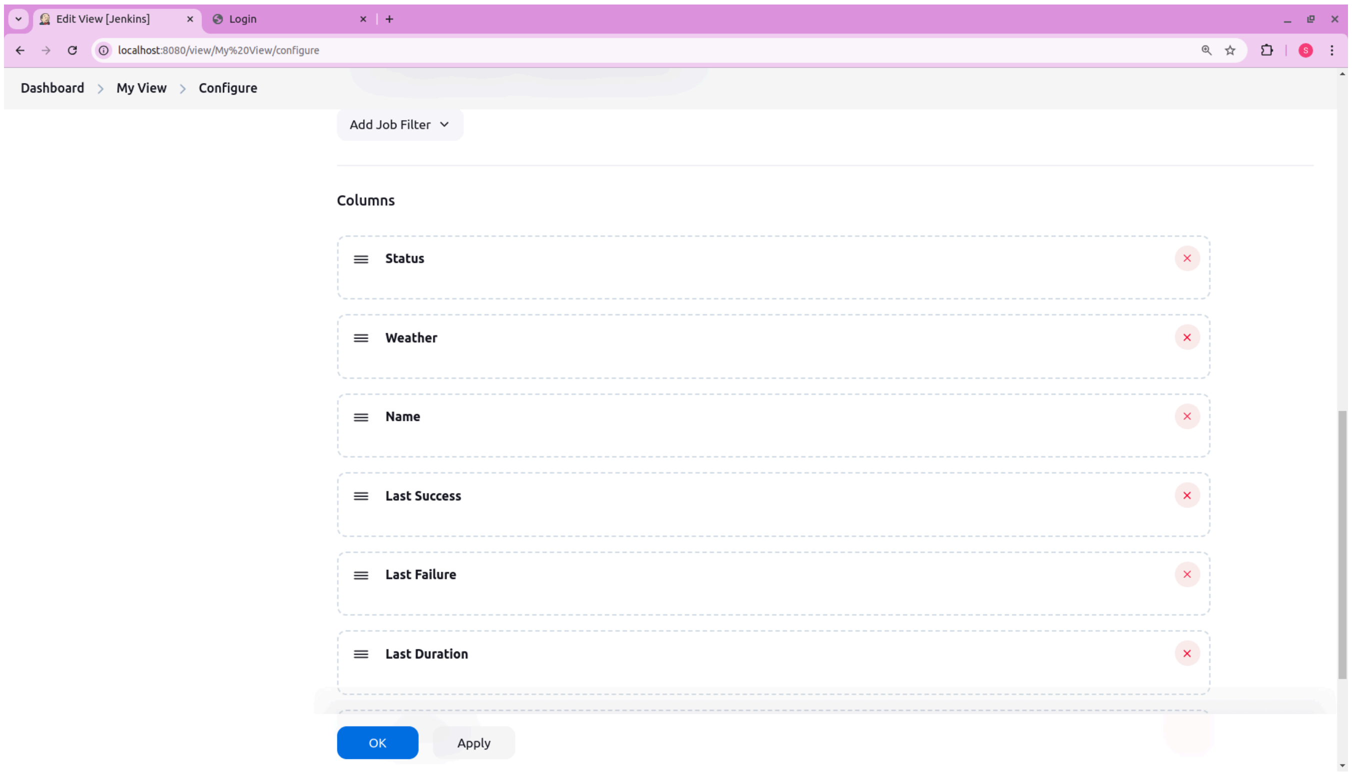
- [ ] Filter build queue
- [ ] Filter build executors

Job Filters

- [ ] Recurse in subfolders

Jobs

- [ ] branch12
- [ ] demo_python_pipeline
- [ ] first_pipeline
- [ ] GPIO
- [ ] job0
- [x] job1


Edit View [Jenkins] | Login

localhost:8080/view/My%20View/configure

Dashboard > My View > Configure

Columns

- Status
- Weather
- Name
- Last Success
- Last Failure
- Last Duration

#5.4: Jenkins View API methods

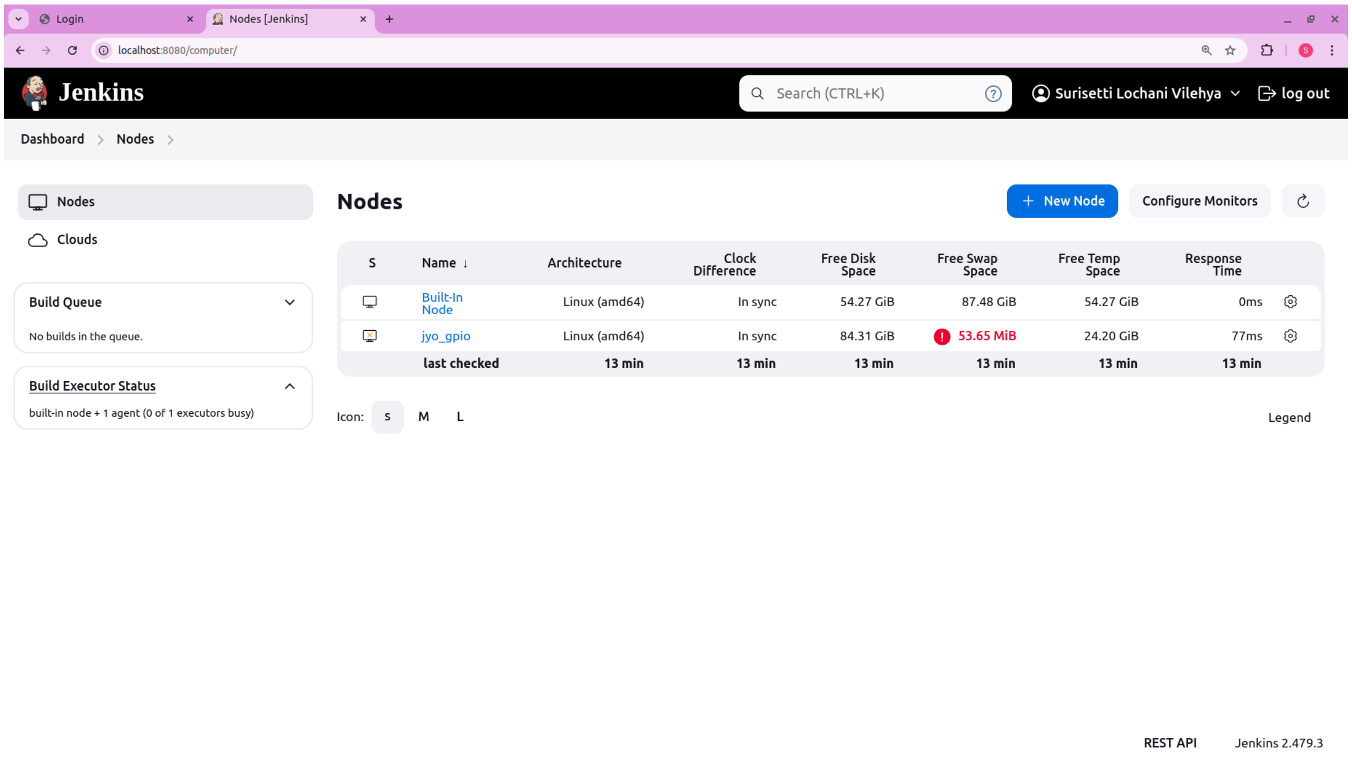
S.No	API Method	Description	Parameters	Returns
1	**get_view_name(name)**	Returns the name of a view using the API. Used to verify if a view exists.	name (str): View name	str: Name of view or None
2	**assert_view_exists(name, exception_message='view[%s] does not exist')**	Raises an exception if a view does not exist.	name (str): View name, exception_message (str): Exception message format	Throws JenkinsException if view does not exist
4	**get_views()**	Retrieves a list of all views running in Jenkins.	None	list[dict]: List of views with name and url keys
5	**delete_view(name)**	Deletes a Jenkins view permanently.	name (str): View name	None
6	**create_view(name, config_xml)**	Creates a new Jenkins view.	name (str): View name, config_xml (str): View configuration XML	None
7	**reconfig_view(name, config_xml)**	Modifies the configuration of an existing view.	name (str): View name, config_xml (str): New XML configuration	None
8	**get_view_config(name)**	Retrieves the configuration of an existing Jenkins view in XML format.	name (str): View name	str: XML configuration of the view

#6: Jenkins Nodes

#6.1: What Are Jenkins Nodes?

In Jenkins, nodes refer to machines (physical or virtual) that Jenkins uses to execute jobs. These nodes can be categorized into:

- **Controller (formerly Master):** The main Jenkins server that manages jobs and configurations.
- **Agents (formerly Slaves):** Additional machines that execute jobs based on the controller's instructions.



Build Queue

No builds in the queue.

Build Executor Status

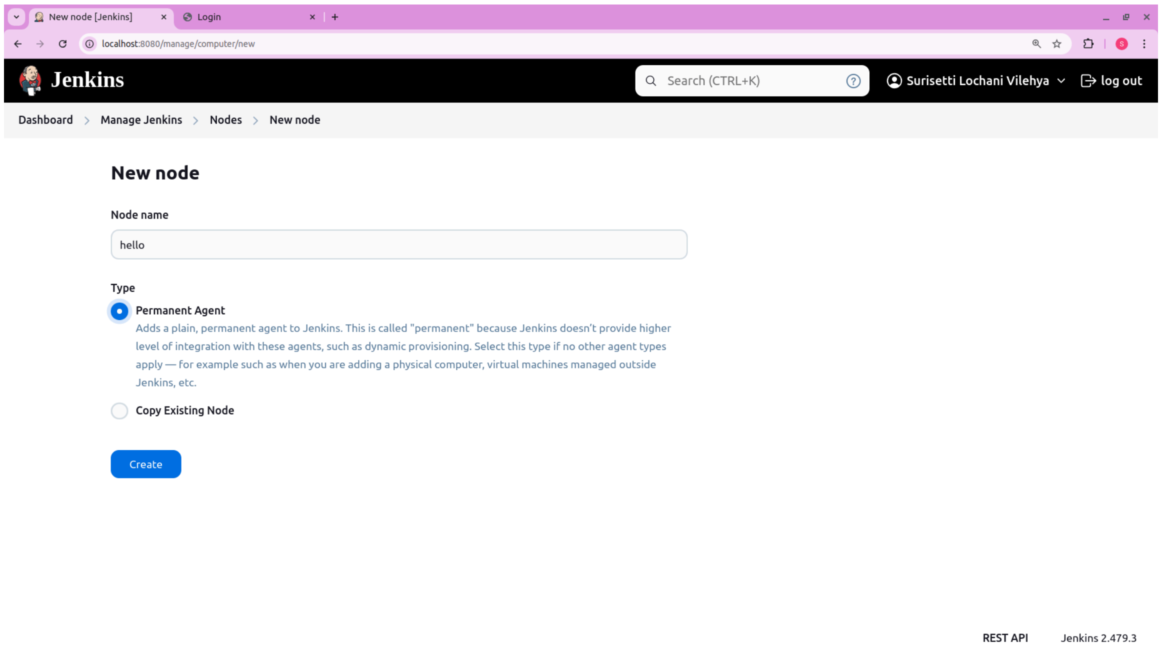
built-in node + 1 agent (0 of 1 executors busy)

Nodes

S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Built-In Node	Linux (amd64)	In sync	54.27 GiB	87.48 GiB	54.27 GiB	0ms
	jyo_gpio	Linux (amd64)	In sync	84.31 GiB	53.65 MiB	24.20 GiB	77ms
	last checked	13 min	13 min	13 min	13 min	13 min	13 min

#6.2: How to Configure Jenkins Nodes

1. Go to Manage Jenkins → Nodes.
2. Click on **New Node** and enter a name for the agent.
3. Select **Permanent Agent** and click **OK**.


New node [Jenkins] | Login

localhost:8080/manage/computer/new

Jenkins | Search (CTRL+K) | Surisetti Lochani Vilehya | log out

Dashboard > Manage Jenkins > Nodes > New node

New node

Node name

hello

Type

● Permanent Agent

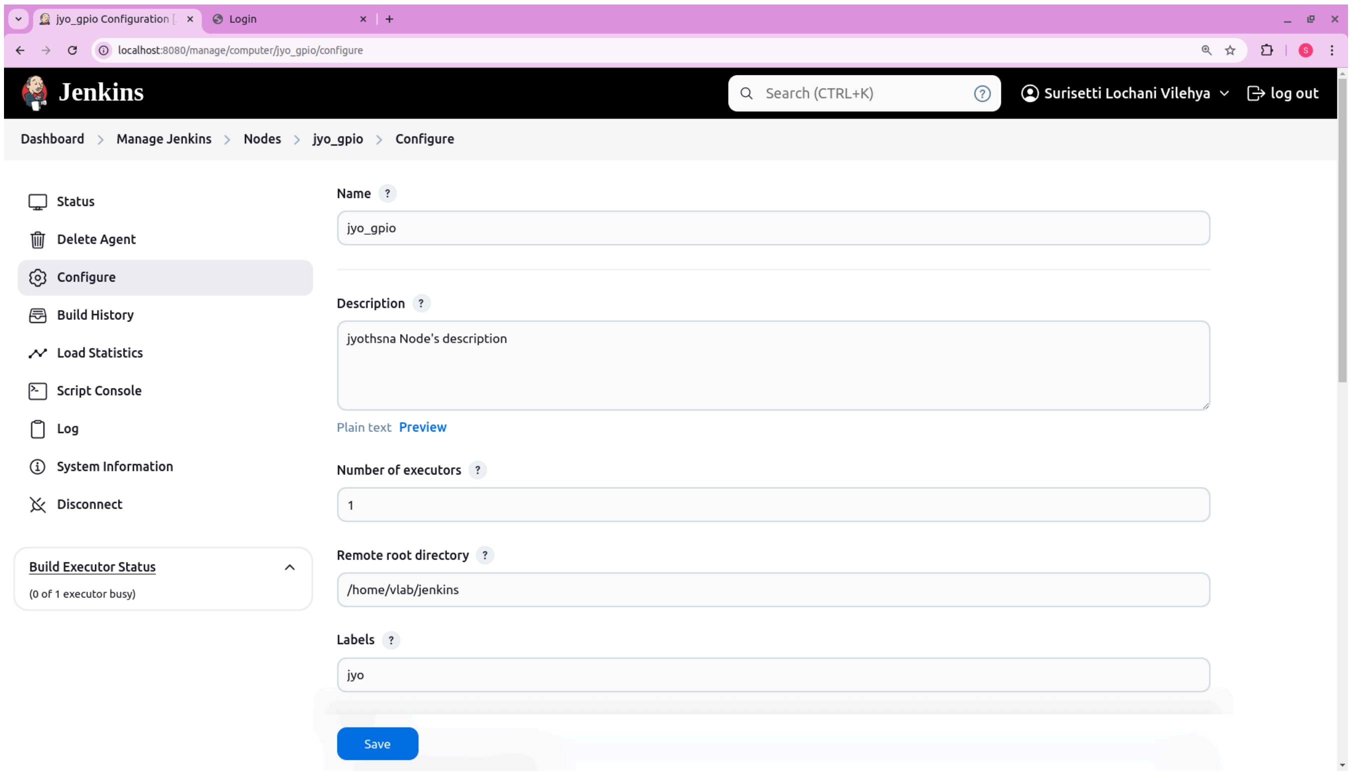
Adds a plain, permanent agent to Jenkins. This is called "permanent" because Jenkins doesn't provide higher level of integration with these agents, such as dynamic provisioning. Select this type if no other agent types apply — for example such as when you are adding a physical computer, virtual machines managed outside Jenkins, etc.

○ Copy Existing Node

Create

REST API Jenkins 2.479.3

4. In the Node Configuration Page, provide:
    a. **Remote root directory:** Path where Jenkins agent files will be stored (e.g., /var/jenkins).
    b. **Number of executors:** Number of concurrent jobs this node can run.
    c. **Labels:** Tags to group similar nodes (e.g., linux-node, windows-node).


jyo_gpio Configuration | Login

localhost:8080/manage/computer/jyo_gpio/configure

Jenkins | Search (CTRL+K) | Surisetti Lochani Vilehya | log out

Dashboard > Manage Jenkins > Nodes > jyo_gpio > Configure

Status · Delete Agent · Configure · Build History · Load Statistics · Script Console · Log · System Information · Disconnect

Build Executor Status

(0 of 1 executor busy)

Name: jyo_gpio

Description: jyothsna Node's description

Plain text Preview

Number of executors: 1

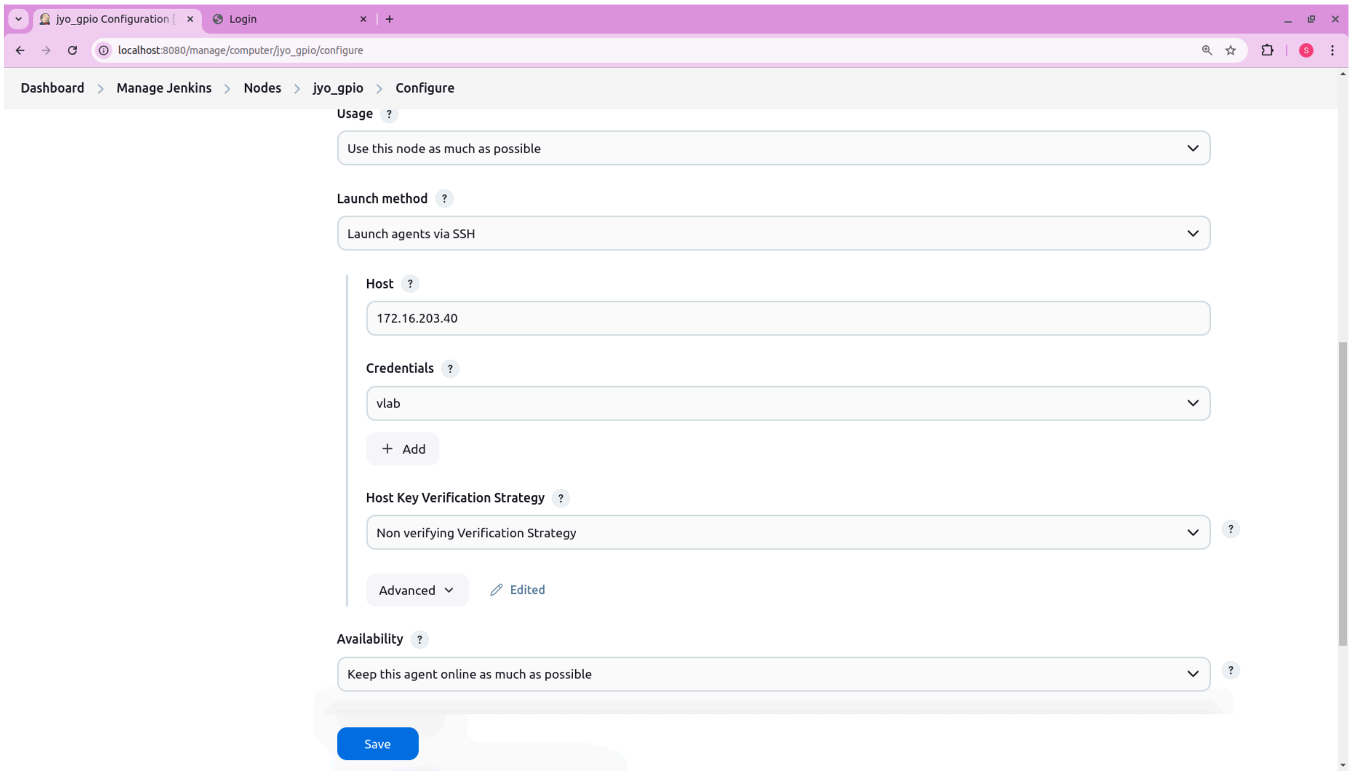
Remote root directory: /home/vlab/jenkins

Labels: jyo

Save

    d. **Usage:**

        i. "Use this node as much as possible" (default).
        ii. "Only build jobs with assigned labels" (for specific workloads).
    e. **Launch method:**
        i. SSH (Recommended for Linux/Mac). : Here provide the host number and credentials
        ii. Launch agent by connecting to controller .
        iii. Docker (for cloud-based agents).
5. Click Save.


jyo_gpio Configuration
Login
localhost:8080/manage/computer/jyo_gpio/configure
Dashboard > Manage Jenkins > Nodes > jyo_gpio > Configure
+ Add
Advanced
Edited
Save

Usage: Use this node as much as possible

Launch method: Launch agents via SSH

Host: 172.16.203.40

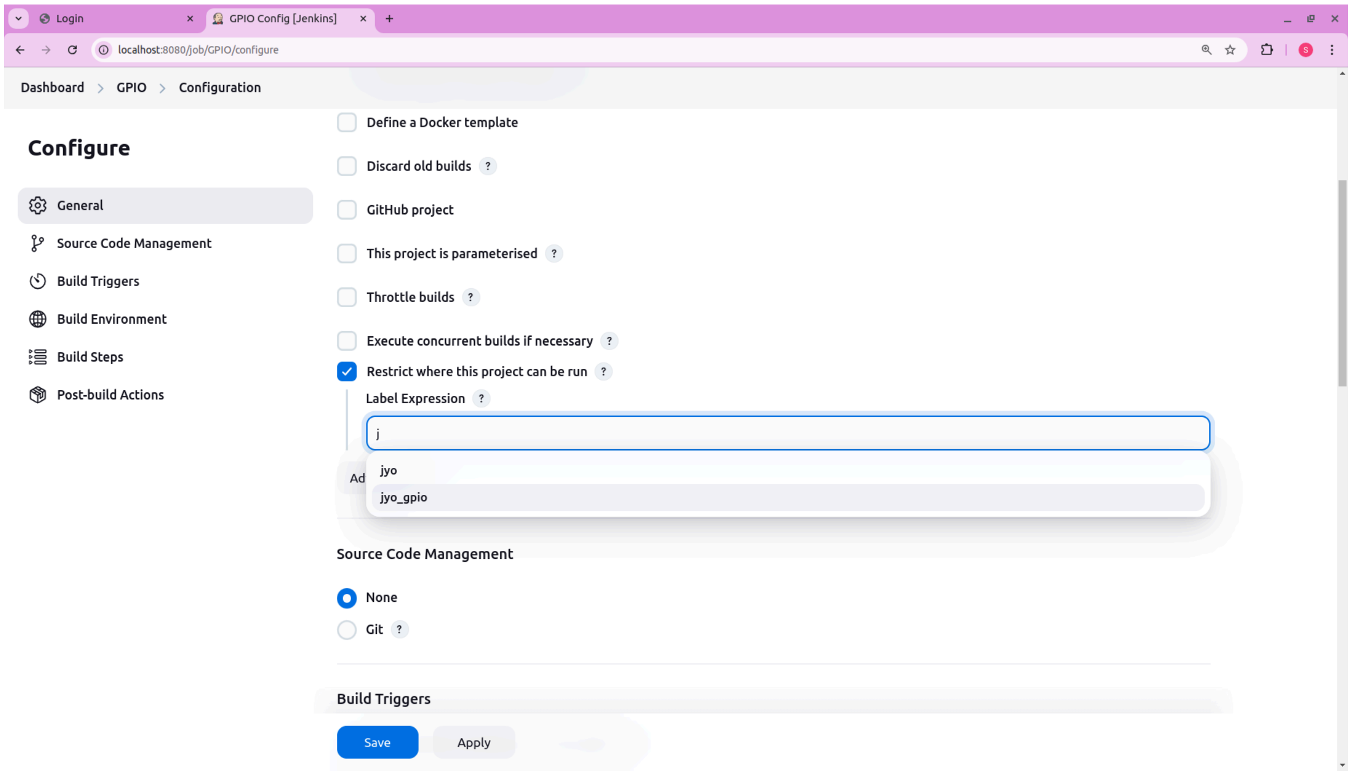
Credentials: vlab

Host Key Verification Strategy: Non verifying Verification Strategy

Availability: Keep this agent online as much as possible

#6.3: Assign Jobs to a Specific Node

1. Open the Jenkins job configuration.
2. Enable "Restrict where this project can be run".
3. Enter the node label (e.g., linux-node).
4. Click Save.



[ ] Define a Docker template
[ ] Discard old builds
[ ] GitHub project
[ ] This project is parameterised
[ ] Throttle builds
[ ] Execute concurrent builds if necessary
[✓] Restrict where this project can be run

Label Expression: j

jyo
jyo_gpio

Source Code Management

[✓ None] Git

Build Triggers

#6.4 Jenkins Nodes API methods

S.No	Method Name	Description	Parameters	Returns
1	**get_nodes(depth=0)**	Get a list of nodes connected to the Master. Each node is a dictionary with name and offline status.	depth (int): JSON depth level (default=0)	List of nodes [{ str: str, str: bool}]
2	**get_node_info(name, depth=0)**	Get detailed information about a specific node.	name (str): Node name depth (int): JSON depth level (default=0)	Dictionary containing node details
3	**node_exists(name)**	Check whether a node exists in Jenkins.	name (str): Node name	True if node exists, else False
4	**assert_node_exists(name, exception_message='node[%s] does not exist')**	Raise an exception if a node does not exist.	name (str): Node name exception_message (str): Exception message format	Throws JenkinsException if the node does not exist

5	**delete_node(name)**	Permanently delete a Jenkins node.	name (str): Node name	None
6	**disable_node(name, msg='')**	Disable a Jenkins node (take it offline).	name (str): Node name msg (str): Offline message (optional)	None
7	**enable_node(name)**	Enable a previously disabled node.	name (str): Node name	None
8	**create_node(name, numExecutors=2, nodeDescription=None, remoteFS='/var/lib/jenkins', labels=None, exclusive=False, launcher=jenkins.LAUNCHER_SSH, launcher_params={})**	Create a new Jenkins node with the specified parameters.	name (str): Node name numExecutors (int): Number of executors (default=2) nodeDescription (str): Description (optional) remoteFS (str): Remote filesystem path (default='/var/lib/jenkins') labels (str): Labels associated with the node (optional) exclusive (bool): Use only for tied jobs (default=False) launcher (str): Launch method launcher_params (dict): Additional launcher parameters	None
9	**get_node_config(name)**	Get the XML configuration of a Jenkins node.	name (str): Node name	Node configuration in XML format
10	**reconfig_node(name, config_xml)**	Modify the existing configuration of a Jenkins node.	name (str): Node name config_xml (str): New XML configuration	None