Python with Jenkins - Yatri Cloud

Creator: Yatharth Chauhan

Integrating Jenkins with Python allows you to automate various aspects of your software development lifecycle, including continuous integration and continuous delivery (CI/CD) pipelines. Jenkins is a widely used open-source automation server that supports building, deploying, and automating software projects. Below, we will explore how to set up Jenkins with Python, create Jenkins jobs, and manage Jenkins through Python scripts.

1. **Job**: A specific task that Jenkins performs, such as building a project or running tests.
2. **Pipeline**: A series of steps that define how software is built, tested, and deployed.
3. **Node**: A machine that Jenkins uses to run jobs, which can be either the master node or any agent nodes.
4. **Plugins**: Extensions that add functionalities to Jenkins, such as integrating with various tools and platforms.

Setting Up Jenkins

1. **Installation**: You can install Jenkins on your local machine or server. Follow the official installation guide for your operating system.

2. **Access Jenkins**: Once installed, you can access Jenkins by navigating to `http://localhost:8080` in your web browser.

3. **Configure Jenkins**: Complete the initial setup by following the on-screen instructions, including installing recommended plugins.

Integrating Python with Jenkins

You can integrate Python with Jenkins in several ways, including:

1. **Using Jenkins Pipelines**: Define your build process using Jenkins' built-in pipeline capabilities.
2. **Using Jenkins REST API**: Interact with Jenkins programmatically through its REST API using Python.

Example 1: Creating a Jenkins Job with Python

You can use the Jenkins Job DSL plugin to define jobs in a more manageable way. However, we will use the `python-jenkins` library to interact with Jenkins.

Step 1: Install the `python-jenkins` Library

You can install the library using pip:

```
pip install python-jenkins
```

Step 2: Create a Jenkins Job

Here's how to create a Jenkins job using Python:

```python
import jenkins

# Connect to Jenkins server
jenkins_url = 'http://localhost:8080'  # Your Jenkins URL
username = 'yatricloud_username'
password = 'yatricloud_api_token'  # Use an API token instead of a password for
better security

server = jenkins.Jenkins(jenkins_url, username=username, password=password)

# Define the job configuration
job_name = 'yatricloud-job'
job_config = '''<project>
  <builders>
    <hudson.tasks.Shell>
      <command>echo "Hello, Jenkins!"</command>
    </hudson.tasks.Shell>
  </builders>
</project>'''

# Create a new job
server.create_job(job_name, job_config)
print(f"Job '{job_name}' created successfully.")
```

Explanation:

- **jenkins.Jenkins()**: Connects to the Jenkins server using the provided URL, username, and API token.
- **create_job()**: Creates a new job with the specified name and configuration.

Example 2: Triggering a Jenkins Job

You can trigger a job that you have created from Python as follows:

```python
import jenkins
import time

# Connect to Jenkins server
jenkins_url = 'http://localhost:8080'
username = 'yatricloud_username'
password = 'yatricloud_api_token'

server = jenkins.Jenkins(jenkins_url, username=username, password=password)

# Trigger the job
job_name = 'yatricloud-job'
server.build_job(job_name)

print(f"Job '{job_name}' triggered successfully.")
```

```
# Wait for the job to finish
while True:
    last_build_number = server.get_job_info(job_name)['lastBuild']['number']
    build_info = server.get_build_info(job_name, last_build_number)
    if build_info['building']:
        print("Job is still running...")
        time.sleep(10)  # Wait before checking again
    else:
        print(f"Job finished with result: {build_info['result']}")
        break
```

Explanation:

- **build_job()**: Triggers the specified job in Jenkins.
- **get_job_info()**: Retrieves information about the job, including the status of the last build.
- The script waits until the job completes and then prints the result.

Example 3: Using Jenkins Pipelines

You can also define a Jenkins pipeline that runs Python scripts or jobs. Here's a basic example of how to create a Jenkins pipeline job using Python.

```
import jenkins

# Connect to Jenkins server
jenkins_url = 'http://localhost:8080'
username = 'yatricloud_username'
password = 'yatricloud_api_token'

server = jenkins.Jenkins(jenkins_url, username=username, password=password)

# Define the pipeline job configuration
pipeline_job_name = 'yatricloud-pipeline-job'
pipeline_script = '''pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                echo 'Building...'
                sh 'python -c "print(\'Hello from Python!\')"'
            }
        }
        stage('Test') {
            steps {
                echo 'Testing...'
                sh 'pytest tests/'  // Assuming you have tests in a "tests"
directory
            }
        }
    }
}'''

# Create a new pipeline job
server.create_job(pipeline_job_name, jenkins.Pipeline(pipeline_script))
print(f"Pipeline job '{pipeline_job_name}' created successfully.")
```

Explanation:

- **Pipeline Syntax**: The example uses Jenkins pipeline syntax to define a simple CI/CD process with build and test stages.
- **sh command**: Runs shell commands, which can include invoking Python scripts or commands.

Example 4: Using Jenkins REST API with Python

You can also interact with Jenkins using its REST API. Here's a simple example of how to get the status of a Jenkins job using Python's `requests` library.

```
import requests
from requests.auth import HTTPBasicAuth

# Jenkins server details
jenkins_url = 'http://localhost:8080'
job_name = 'yatricloud-job'
username = 'yatricloud_username'
password = 'yatricloud_api_token'

# Get job information
response = requests.get(f"{jenkins_url}/job/{job_name}/lastBuild/api/json",
auth=HTTPBasicAuth(username, password))

if response.status_code == 200:
    job_info = response.json()
    print(f"Job '{job_name}' last build status: {job_info['result']}")
else:
    print(f"Failed to get job info: {response.status_code}")
```

Explanation:

- **requests.get()**: Sends a GET request to retrieve the last build information of the specified job.
- **HTTPBasicAuth**: Handles basic authentication for the request.

Best Practices for Using Python with Jenkins

1. **API Tokens**: Use API tokens instead of passwords for authentication to enhance security.
2. **Environment Variables**: Store sensitive information (e.g., credentials) in environment variables instead of hardcoding them.
3. **Error Handling**: Implement error handling to manage failures gracefully when interacting with Jenkins.

4. **Version Control**: Keep your Jenkins job configurations and scripts in version control for better collaboration and tracking.
5. **Modularization**: Break down complex jobs into smaller, modular scripts or functions for better maintainability.

Integrating Jenkins with Python enables you to automate CI/CD processes effectively. By using libraries like `python-jenkins` and the Jenkins REST API, you can programmatically manage Jenkins jobs, trigger builds, and retrieve job statuses. This integration can significantly enhance your development workflows, making them more efficient and reliable.
