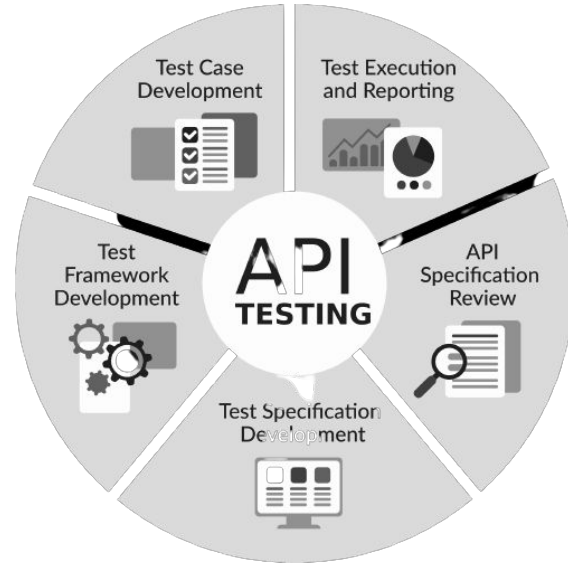


# Rest Assured Basics



Pramod Dutta  
Lead SDET.



## Jenkins + Rest Assured

# Agenda

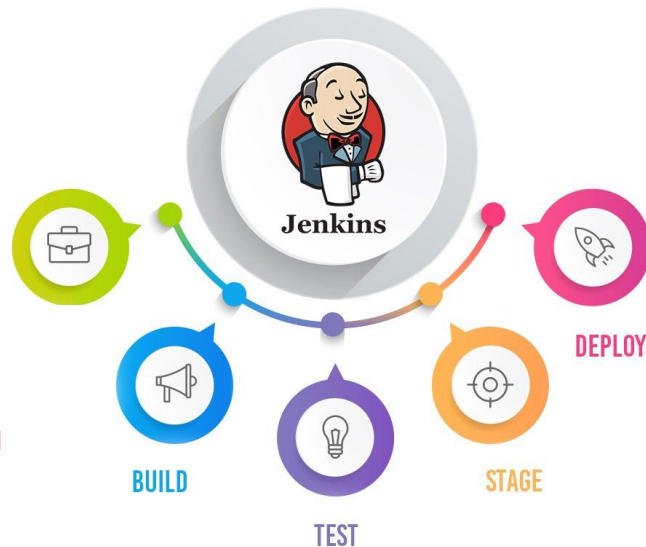
- Jenkins Overview
- Running POSTMAN project in Jenkins.
- Rest Assured Hello World
- Rest Assured Overview

# Install Jenkins

<https://www.jenkins.io/download/>

Sample commands:

- Install the latest LTS version: `brew install jenkins-lts`
- Install a specific LTS version: `brew install jenkins-lts@YOUR_VERSION`
- Start the Jenkins service: `brew services start jenkins-lts`
- Restart the Jenkins service: `brew services restart jenkins-lts`
- Update the Jenkins version: `brew upgrade jenkins-lts`



# What is FreeStyle Job in Jenkins

Freestyle means improvised or unrestricted.

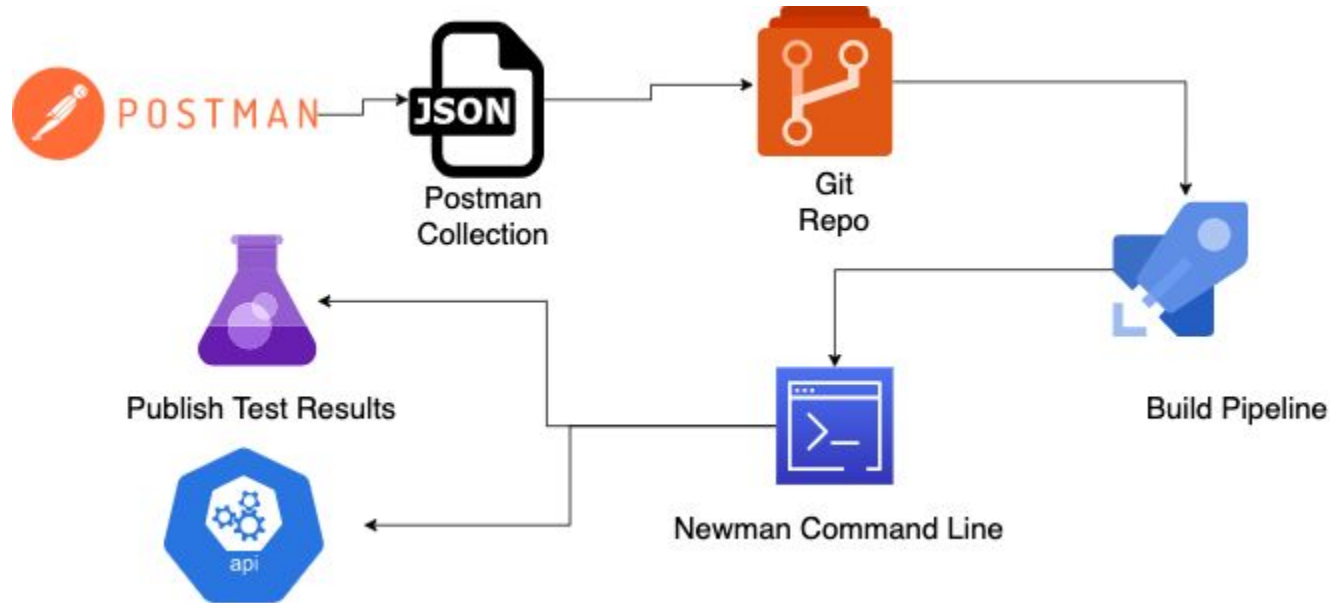
According to the official Jenkins wiki, a freestyle project is a typical build job or task. This could be as simple as running tests, building or packaging an application, sending a report, or even running some commands. Before any tests are run, data is collated.

Jenkins Freestyle Project is a **repeatable build job, script, or pipeline that contains steps and post-build actions**



The image shows a screenshot of the Jenkins Freestyle Project configuration page. The page has a light gray header with several tabs: 'General', 'Source Code Management', 'Build Triggers', 'Build Environment', 'Build', and 'Post-build Actions'. The 'General' tab is currently selected and highlighted. Below the tabs, there is a section labeled 'Description' with a large text area for input.

# Run Postman Collection on Jenkins





# Why We need Parameters in Jenkins?

- Make our Project more generic
- You can change branch, env and other param while running
- You can use logic based on env which command to run
  - E.g if Windows run this
  - Else run this
-



# Executing Bash script in Jenkins

- `START "" notepad.exe D:\Project\notepad.bat`



# Extended Choice Parameter

Adds extended functionality to Choice parameter.

## Extended Choice Parameter

[Documentation](#)

[Releases](#)

[Issues](#)

[Dependencies](#)

Adds extended functionality to Choice parameter.

For info on how to use groovy script feature use this [Link](#)

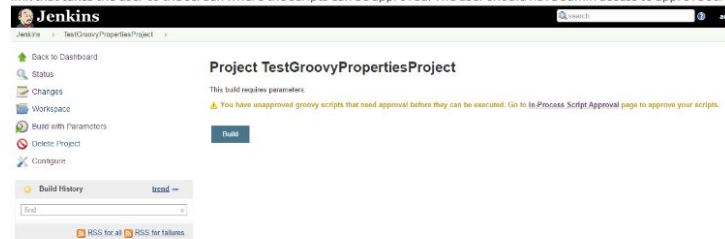
### Change Log

Version 0.64 (Apr 17, 2016)

Show user friendly error message for script approval issues.

Script security has been incorporated into this [plugin](#) for security reasons. You may see warning messages related to security with groovy-script execution from this version on-wards.

A warning message is displayed on the screen if the user tries to execute groovy scripts that have not been approved. The message has a link that takes the user to the screen where the scripts can be approved. The user should have admin access to approve scripts.







# Rest Assured Basics



# What is REST Assured?

- REST Assured is a Java Domain Specific Language (DSL) API used for writing automated tests for RESTful APIs.
- It is used to invoke REST web services for POST, GET, PUT, DELETE, OPTIONS, PATCH, and HEAD requests and validate the response.



# Benefits of REST Assured.

- Contains tons of helper methods and abstraction layers.
- Remove the need for writing a lot of boilerplate code for connections, sending a request, and parsing a response.
- Easy to Understand Domain specific Language.
- Seamless integration with Java, TestNG.

# Install Rest Assured in Maven Project



- <https://mvnrepository.com/artifact/io.rest-assured/rest-assured>

The screenshot shows the Maven Repository page for the artifact `io.rest-assured:rest-assured:4.3.1`. The page includes a line graph showing the artifact's popularity over time, a table of metadata, and a code snippet for adding the dependency to a Maven project.

**REST Assured » 4.3.1**  
Java DSL for easy testing of REST services

License	Apache 2.0
HomePage	<a href="http://code.google.com/p/rest-assured">http://code.google.com/p/rest-assured</a>
Date	(Jul 03, 2020)
Files	bundle (682 KB) <a href="#">View All</a>
Repositories	Central
Used By	833 artifacts

Build tools: [Maven](#) [Gradle](#) [SBT](#) [Ivy](#) [Grape](#) [Leiningen](#) [Buildr](#)

```
<!-- https://mvnrepository.com/artifact/io.rest-assured/rest-assured -->
<dependency>
  <groupId>io.rest-assured</groupId>
  <artifactId>rest-assured</artifactId>
  <version>4.3.1</version>
  <scope>test</scope>
</dependency>
```

☒ Include comment with link to declaration



# Given/When/Then

## given()

- given part describes the state of the world before you begin the behavior you're specifying in this scenario.
- Building the DSL expression request.
- You can add Headers, params, message body, authentication.
- Its is pre request method(before making actual GET,POST, PUT. DELETE).



# Given/When/Then

when()

- When section is that behavior that you're specifying.
- After when we mention the HTTP Method
- Body



# Given/When/Then

## Then()

- Finally the then section describes the changes you expect due to the specified behavior.
- After Request is made this will give you body of the response
- You can assert the response now.
- Validation happens here.



# Given/When/Then

<https://martinfowler.com/bliki/GivenWhenThen.html>





# Given/When/Then

```
1 package com.thetestingacademy.ra;
2
3 import io.restassured.RestAssured;
4 import org.testng.annotations.Test;
5
6 import static org.hamcrest.Matchers.equalTo;
7
8 public class GivenWhenThenDemo {
9
10     @Test
11     public void getData(){
12         //https://reqres.in/api/users?page=3
13         RestAssured.baseURI = "https://reqres.in";
14         RestAssured.basePath = "api/users/";
15
16         RestAssured.given().param( s: "page", ...objects: "3") RequestSpecificat
17             .when()
18             .get() Response
19             .then() ValidatableResponse
20             .body( s: "page", equalTo( operand: 3));
21     }
22 }
23
```



# Download Pom.xml

<https://codeshare.io/2B0gbK>



# Best Practices For Writing Automation Test Code

- Follow Programming Language Guidelines( method name-getPayload(), class name- APIEndpoints, package name-com.testingsumo.backend.tests, variable name-authToken)
- Follow Oops Concepts Wherever Possible- Abstraction(base classes), Inheritance(multiple implementation of same things/multiple inheritance), Polymorphism(many forms with something different), Data Hiding(hide unnecessary/sensitive info), Encapsulation(Bind small entities into a single large entity)
- Reduce code duplicacy (think before writing new code, can i use/make change in existing code?)
- Increase code reusability
- Make your code generic wherever possible
- Leave no hardcoded data in source code
- Keep your static data outside the source code
- Keep your dynamic data dynamic in testcode (fetch it from db queries or scripts)
- Test your code properly, use IDE options such as call heirarcy or show usage to test your changes end 2 end



# Best Practices For Writing Automated Testing Code Cont...

- Use Extensive logging- everything which is part of source code should be analyzed from logs without looking at the source code
- Generate and save failure proofs outside the src code- videos/data/screenshots/logs
- Focus on making your code scalable and faster without compromising the code quality
- Your code should be platform and system independent
- Use as many assertions as possible focus on automated testing rather than automation
- Leave no hardcoded data in source code
- Always think for the future, separate out tech dependencies so that migration to new tech is easy in case it is needed
- Keep your tests independent for better results in multithreading unless they are related (example publisher subscriber related tests)
- Use Proper Documentation
- Create code which is can be easily read and modified by others

**Thanks, for attending Class**

**I hope you liked it.**

**Fin.**