**POSTMAN API AUTOMATION BUILD** RUNBOOK

|  |  |
| --- | --- |
| Date submitted | 06/04/2020 |
| Document version: | V1.0.0 |

Document Instructions

<This document template contains directions for use or sample entries. These directions are enclosed in brackets (< >) and are italicized. They are included to help you fill out the form. As you complete the form, delete these instructions.

Please follow this document naming convention to facilitate document search and retrieval:

<project code (if appropriate)> <project name (abbreviated)> <document name (abbreviated)> <version (if appropriate)>

All documents should be posted to the appropriate project folder in SharePoint. >

Document History

<The document history is a log of changes that are made to the document, who made the changes, and when. For example, the initial creation of the document may contain the following: Version 0.1, Date 1/1/2004, Author Charlie Brown, Status Initial creation. Subsequent updates to the document will be Version 0.2, 0.3, etc. The first published version of the document should be Version 1.0.>

Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author(s)** | **Revision Notes** |
| 1.0 | 06/04/2020 | IBM | Initial draft |

Approvals

<In lieu of this Approval section, which requires multiple signatures on one document, you may elect to use an Approval Memo. Distribute both the Detailed Design Document and the Approval Memo, requesting that only the memo be printed, signed and returned to indicate approval.>

Your signature below indicates that this document meets its objectives and is acceptable.

|  |  |  |  |
| --- | --- | --- | --- |
| **Project Role** | **Name** | **Signature** | **Date** |
| IT Project Manager |  |  |  |
| IT Project Manager |  |  |  |
| Application Architect |  |  |  |
| Development Director |  |  |  |
| Functional Director |  |  |  |
| Enterprise Architect |  |  |  |

Table of Contents

[1 Introduction 4](file:///C:\Users\AjayaRout\Desktop\Poatman%20API%20Automation%20Build%20Runbook.docx#_Toc44686518)

[1.1 Purpose 4](file:///C:\Users\AjayaRout\Desktop\Poatman%20API%20Automation%20Build%20Runbook.docx#_Toc44686519)

[1.2 Background 4](file:///C:\Users\AjayaRout\Desktop\Poatman%20API%20Automation%20Build%20Runbook.docx#_Toc44686520)

[1.3 Scope 4](file:///C:\Users\AjayaRout\Desktop\Poatman%20API%20Automation%20Build%20Runbook.docx#_Toc44686521)

[1.4 Reference 4](file:///C:\Users\AjayaRout\Desktop\Poatman%20API%20Automation%20Build%20Runbook.docx#_Toc44686522)

[2 AUTOMATION API TEST 5](file:///C:\Users\AjayaRout\Desktop\Poatman%20API%20Automation%20Build%20Runbook.docx#_Toc44686523)

[2.1 WHY AUTOMATION IN POSTMAN 5](file:///C:\Users\AjayaRout\Desktop\Poatman%20API%20Automation%20Build%20Runbook.docx#_Toc44686524)

[2.2 STEPS FOR POSTMAN CONFIGURATION 5](file:///C:\Users\AjayaRout\Desktop\Poatman%20API%20Automation%20Build%20Runbook.docx#_Toc44686525)

[3 RUN THE TESTS AND DEBUG 11](file:///C:\Users\AjayaRout\Desktop\Poatman%20API%20Automation%20Build%20Runbook.docx#_Toc44686526)

[Introduction to Newman 12](file:///C:\Users\AjayaRout\Desktop\Poatman%20API%20Automation%20Build%20Runbook.docx#_Toc44686527)

[4 INTEGRATING TESTING INTO CI/CD PIPELINE 15](file:///C:\Users\AjayaRout\Desktop\Poatman%20API%20Automation%20Build%20Runbook.docx#_Toc44686528)

1. Introduction

## Purpose

This document describes the automation build steps of postman through runner and some continuous build tool (i.e. Jenkins).

## Background

This document describes the build steps through postman as well as configuration of command line tool to build the postman collection.

## Scope

This document provides Automation testing of the api through postman.

## Reference

|  |  |  |
| --- | --- | --- |
| **Reference** | **Document Name** | **Comment** |
| Functional and Non-functional requirements |  |  |
| Global Ecom Wireframes |  |  |

1. AUTOMATION API TEST

## WHY AUTOMATION IN POSTMAN

Automation is the new norm across sectors. Automation, using postman in testing can improve and increase the depth and scope of tests for a better software quality.

Postman Automation is truly important while testing products with a large number of integrations and/or frequent releases.

API testing is also known as Integration testing. Integration testing focuses on verifying that the interactions of many small components can integrate together without issue. Since API tests bypass the user interface, they tend to be quicker and much more maintainable than GUI tests. Therefore, a good QA team will make fairly accurate projections based on the backlog at hand and the general information about the project and its architecture and use automation for regression test.

## STEPS FOR POSTMAN CONFIGURATION

**2.1.1 Tools and Dependency for Automation testing**

1. Postman  
2. Node  
3. Newman

**2.1.2 Installing Postman in system**

**1. Visit**<https://chrome.google.com/webstore/category/extensions>  
**2. Enter “Postman Chrome” in “Search the store” section**  
**3. In App section postman will be listed**  
**4. Click on “ADD TO CHROME”**

**2.1.3 Creating Postman Collection**

In postman there is an option to create the collection of API requests in a single JSON file which will take part in the JSON. The collection contains all the request, Headers, Body and testcases. The all things are consolidated in a single JSON file. The collection will take part in the Automation testing through postman runner or any other third party tool.

**Sample postman collection: -**



**2.1.3 Create Test Suita**

For every request there is an option to add test cases in the postman. There is various inbuilt template for test cases present in the postman. The test cases are run when a request executed single or from a collection.

Write custom test suites in JavaScript in the Postman Sandbox for each request. Use [Postman's PM API](https://learning.postman.com/docs/postman/scripts/postman-sandbox-api-reference/) to write even more powerful tests that allow you to assert a set of predefined rules for testing to enable better and cleaner tests.

Below are some testcases we can use in collection or in single request

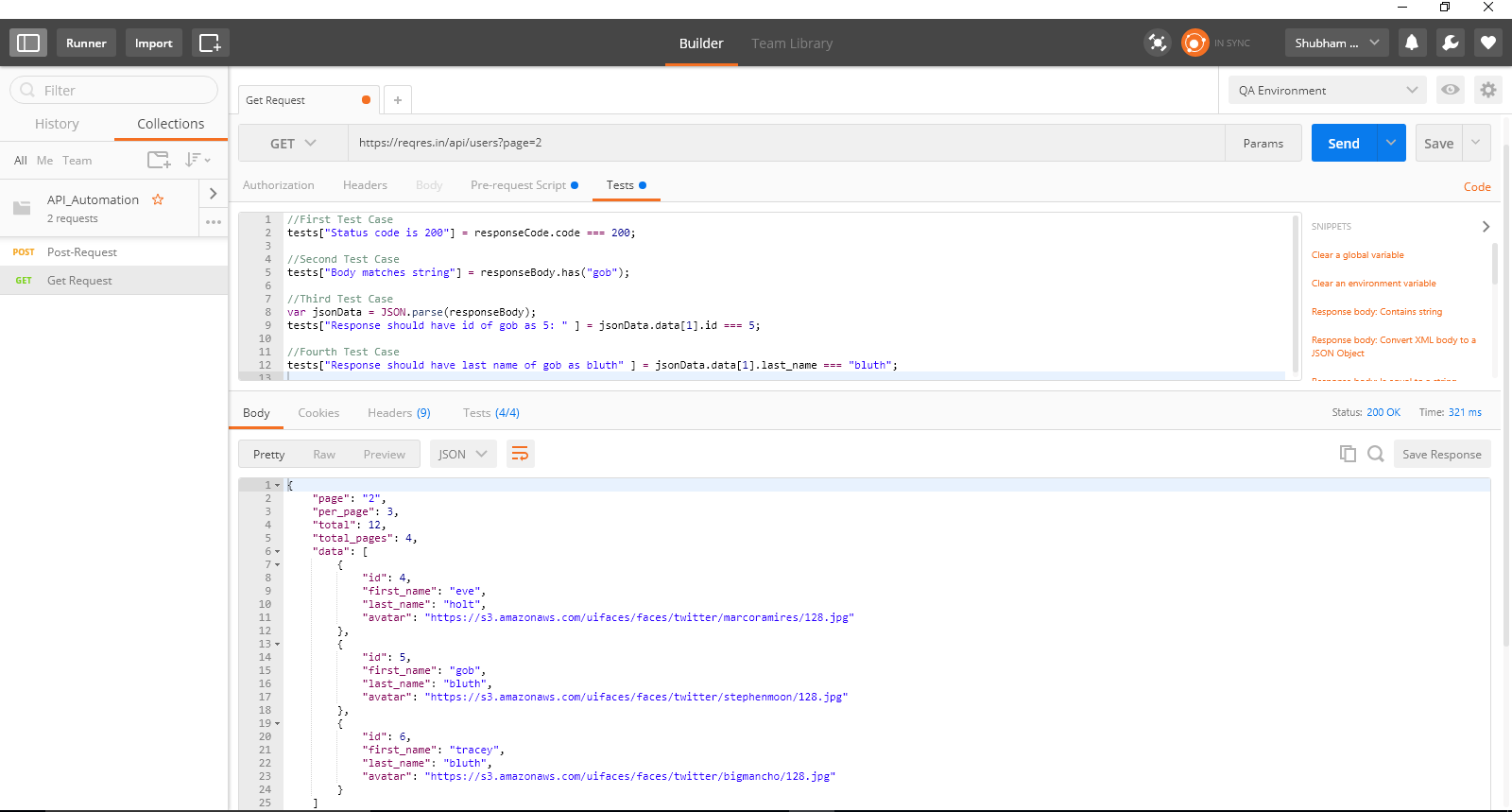
**//**First Test Case  
tests["Status code is 200"] = responseCode.code === 200;

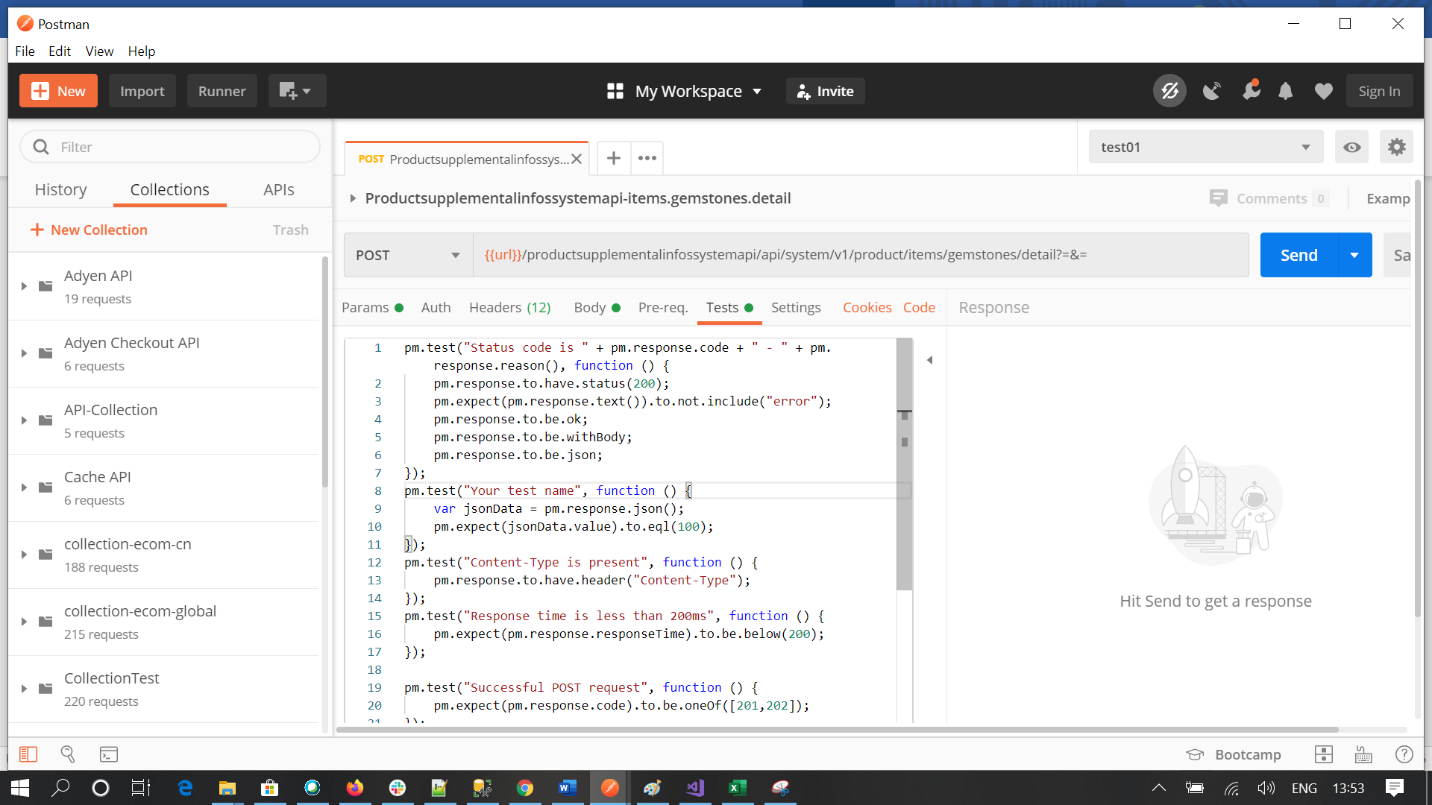
//Second Test Case  
tests["Body matches string"] = responseBody.has("gob");

//Third Test Case  
var jsonData = JSON.parse(responseBody);  
tests["Response should have id of gob as 5: " ] = jsonData.data[1].id === 5;

//Fourth Test Case  
tests["Response should have last name of gob as bluth" ] = jsonData.data[1].last\_name === "bluth";

Above example includes 4 test cases.  
• The first test case is checking for the response code of API. The Pass condition for the test case should be 200 (response code).  
• The second test case is validating for any key or value as :- gob.  
• The third test case is validating for a value of key id as 5 from JSON response.  
• The fourth test case checking for the value of key “bluth” as “Newman” from JSON response.



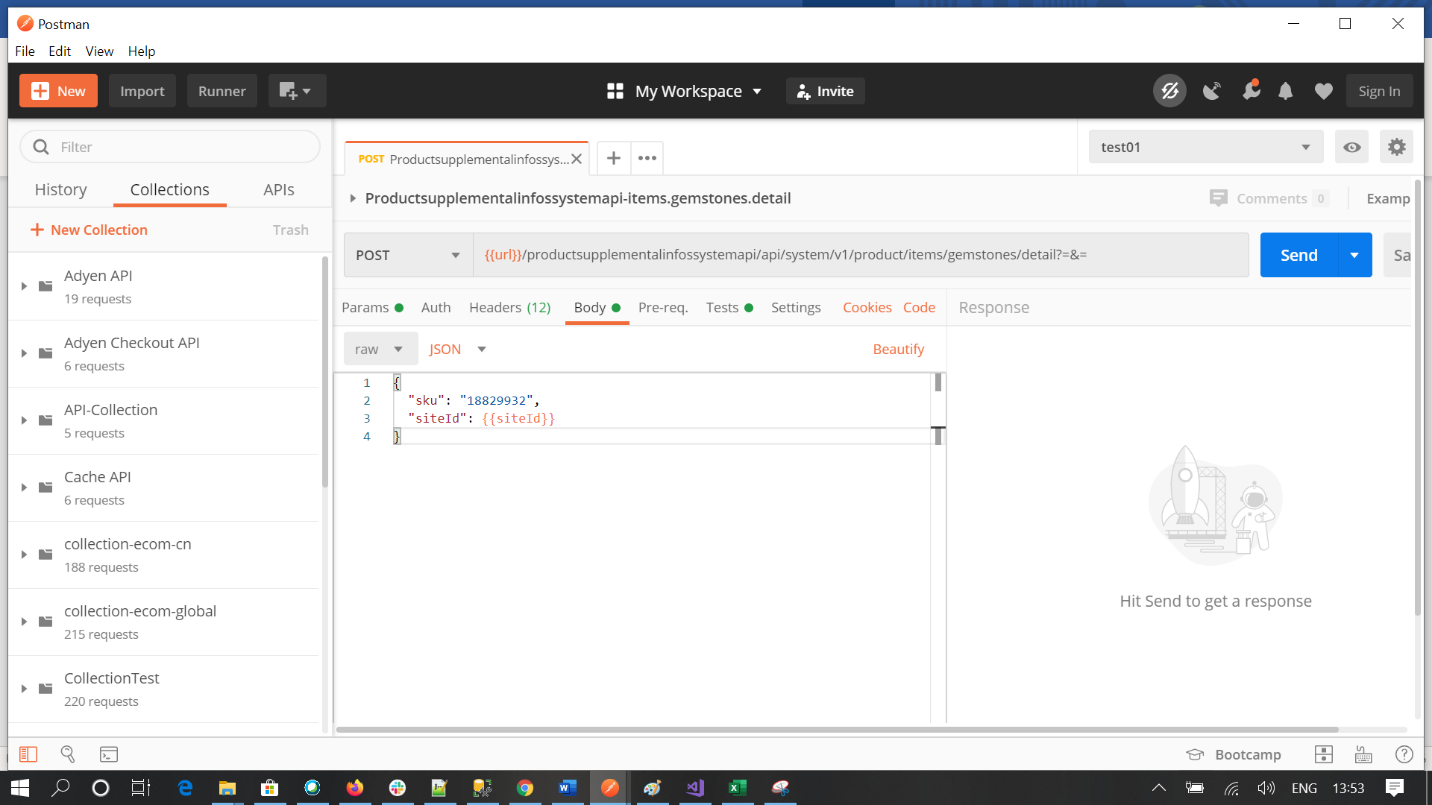


**Parameterize Requests**

Avoid running the same test by testing multiple instances of the same request with parameterization. Insert variables in any part of the request and Postman will replace them with values from a data file or environment variable.

**Variables: -**

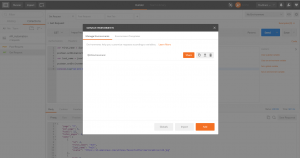
There are two types of variables – global and environment. Global variables are for all collections whereas the environment variables are defined for a specific set of collections as per the environment which can be selected from a drop-down or no environment can be selected. Once defined variables can be used in request with format surrounded by curly brackets: {{VARIABLE\_NAME}}. This can be seen in the image below.



The global variables are variables which can we consumed by any collection. So for any data that is accessible for all collections, we define them as global variables.  
The data defined for a specific environment is called as Environmental variable. We can select any one environment at a time for a collection or no environment as per choice.

• In Postman, at the top right section, click on (\*) icon.  
• Now click on MANAGE ENVIRONMENTS (refer the image below)

• All the existing environments will be listed under Manage Environments.  
• To create an environment, click on Add button at the bottom of the pop-up and enter a new Environment.  
• To add a global variable, click on the Globals button listed at the bottom of the pop-up.  
• For setting a new variable(global and environment), we need to define the New key and Value.

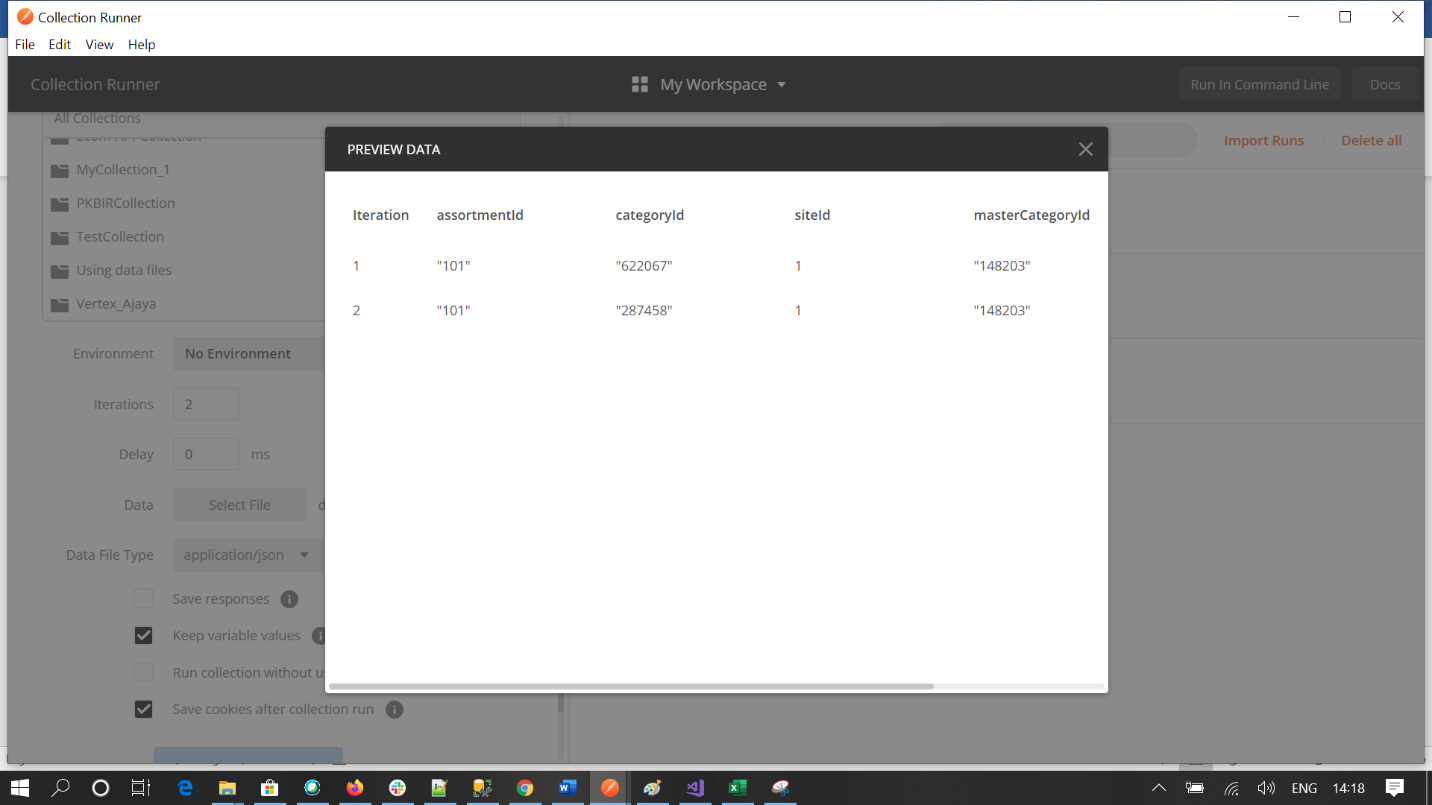


**Reading through Data File**

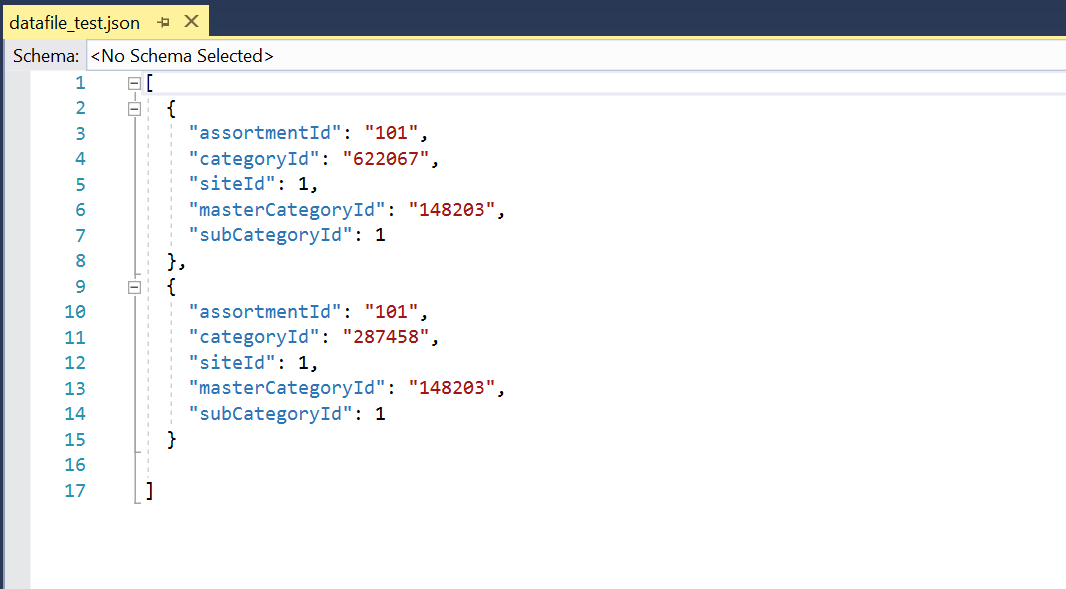
To run the test cases the postman runner and as well as Newman has a option to apply the external data from external file like CSV file as well as JSON file.

**Import Data file into Runner**

* Select Data file in the postman runner.
* Select the file which u want to take part in the test case runner. Follow the below format for setup a data file.
* Click preview to view the data respect to variable u have used.
* Click Run to check the test cases.
* The data should be added as per iteration u want in the test run.



**Sample Data file**



**Import Data file into Newman through Command Line**

The external data file also import in command line using newman. Same JSON or CSV file can be used as external data file.

Below command can be used as to select a data file.

**newman run collection.json -e dev01.json -d datafile\_test.json -k**

**-d = Denotes the external data file import to test run.**

**Save the Output File**

After running the test case we have a option to save the output file to a external file. This can be applying in both postman runner as well as in Newman command line.

Through Newman

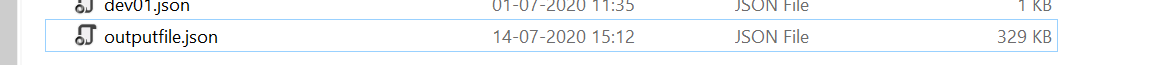
1. To save the output test case file through Newman we need to install a package in the system to run through.

**npm install newman-reporter-csv --save-dev**

2. The above command will help to install the third-party package to save the output json file.

3. After successfully installation we need to run the below command to save the out put file.

**newman run collection.json -e dev01.json -d datafile\_test.json --reporters cli,json --reporter-json-export outputfile.json -k**



1. RUN THE TESTS AND DEBUG

After all setup through postman it’s time to run our test cases through various tool.

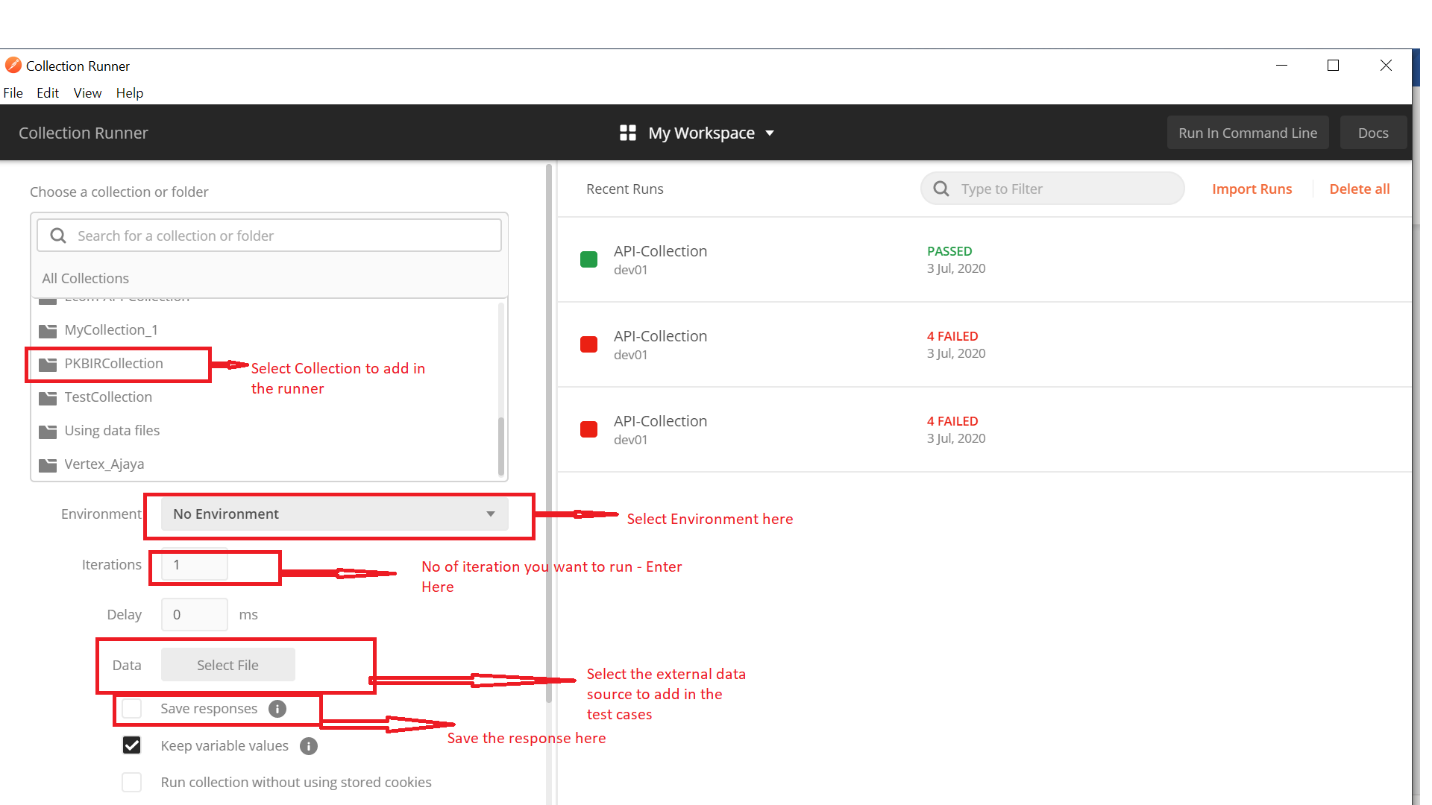
Use the collection runner in the Postman app to test a collection and view the results in real time. Or use [Newman](https://learning.postman.com/docs/postman/collection-runs/command-line-integration-with-newman/) to view test results in the terminal. For any issues, debug in Postman's Console Log. Built specifically for API development, it records the raw request and response, certificates, and error logs from scripts.

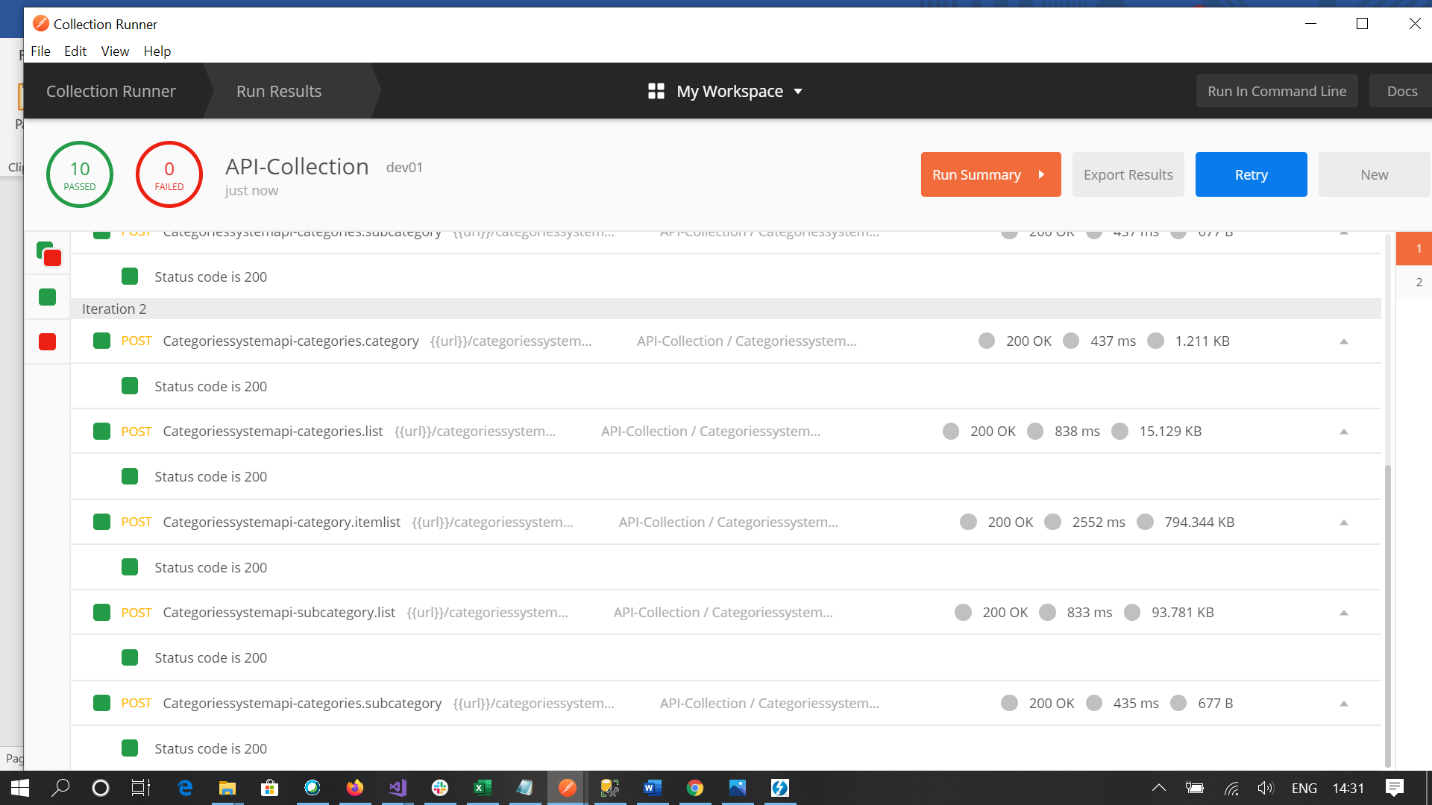
**USING COLLECTION RUNNER**

Postman collection runner is a postman tool to run the collection in one time with all test cases.

**Below step should follow**

* Open the runner.
* Select the collection you want to run through this runner.
* Select environment if u have any in the collection dependency.
* Select External data file if u want to pass data from external data file.
* Enter iteration what ever u want
* Click the Run collection to see the result.





**USING COMMAND LINE TOOL**

To run the test cases through a command line interface we use a node module called Newman.

### Introduction to Newman

Newman is Postman's collection runner engine that sends API requests, receives the response and then runs your test against the response.It is like postman's command line companion.It is extensible and so can be integrated into continuous integration servers and build systems.

**Benefits of using Newman**

* Makes it easy to run a collection of tests using command line
* Gives ability to run a collection of tests written in postman right from within most build tools
* It allows to generate and store report directly from command line  
  Installing Newman

**Installing Newman**

* Install NodeJS – <https://nodejs.org/download/>. Click on the 32-bit or 64-bit Windows Installer package, depending on your machine configuration
* Add the Node executable to your system path. Go to the Control Panel > System and Security > System > Advanced System Settings > Environment variables. Append this to the end of the PATH variable: ;C:\Program Files\Nodejs  
  If you installed Node in a different location, you’ll need to set the PATH accordingly.
* Type “npm install -g newman”. It should take a few minutes to install.

**Run Through Newman**

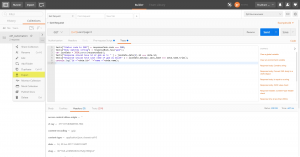
* Newman is a companion tool for postman that runs Collections from the command line.
* Newman maintains feature parity with Postman and allows user to run collections in a similar way they are executed inside the collection runner in the Postman app.

Example: -

newman run postman\_collection.json -e environment.postman\_environment.json -g globals.postman\_globals.json --reporters cli,html

In above command -e represents environment variable, -g global variable, -d csv data file name, --reporters cli,html represents output as console and  
html reports.

**Steps to export JSON for newman :-**1. In postman, under Collections section, select the required collection.  
2. Click on Export as shown in the image below.  
3. Select “Collection v2”  
4. Click on Export  
5. Save this JSON



**Steps to export JSON of global and environment variables for newman :-**

1. In postman, click on the (\*) located at the top right section.  
2. Click on Manage Environment  
3. Now click on download icon placed against the environment name  
4. Save the JSON on the same location where collection was saved  
5. Now to download global variables,click on the “Globals” button placed at the bottom of the pop-up.  
6. Click on “Download as JSON” button.  
7. Save the JSON on the same location as others

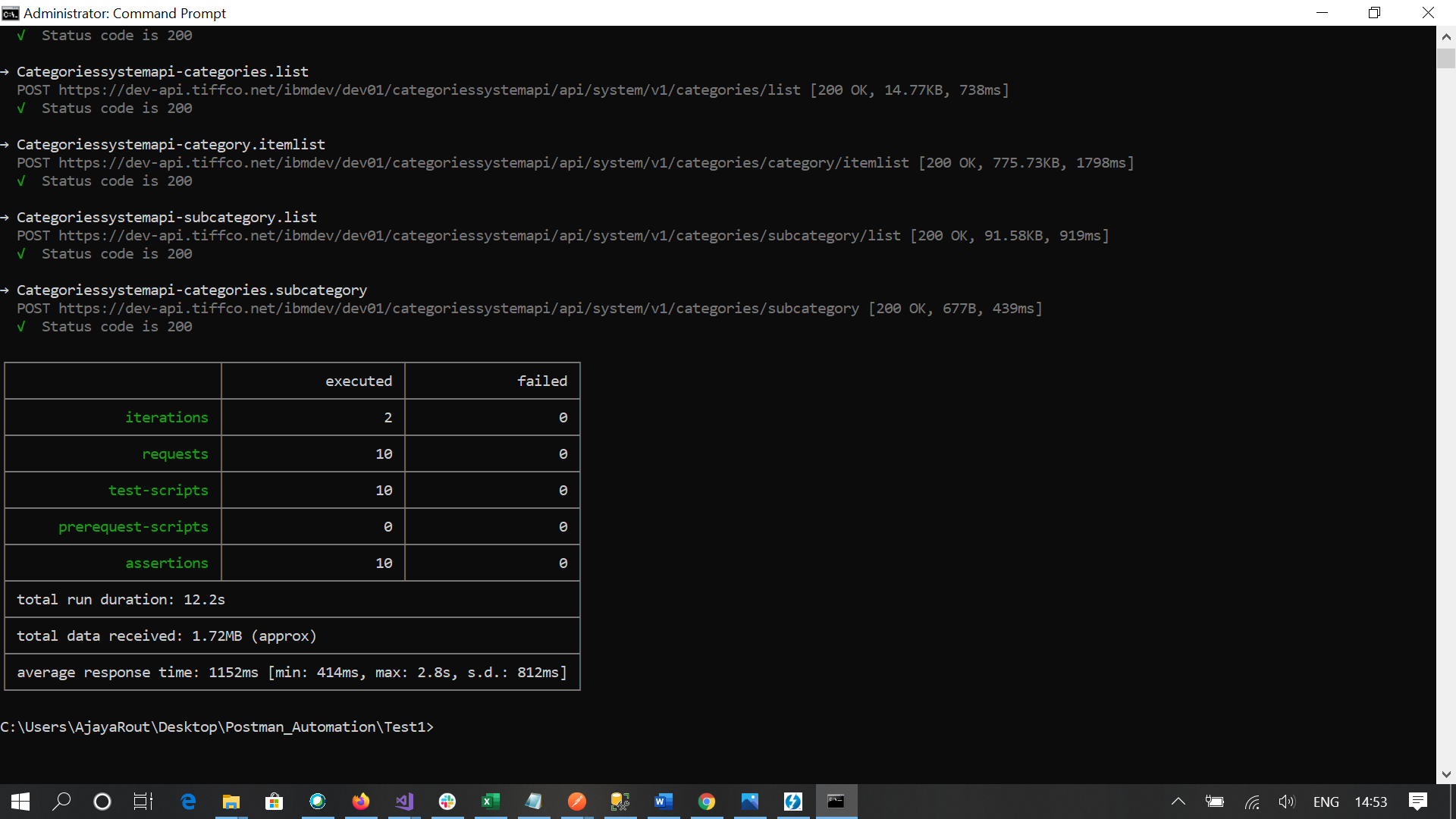
Now, we have all the JSON files required to run our test cases in Newman. The collection JSON - contains all our test scripts, URL, Headers, Prerequisite Script etc contains most of the data except variables. The other two JSON files contain our key and values respectively.

Now it’s time to run our test cases from command line

**Steps to run test cases from command line :-**

• Open CMD  
• Now reach to the folder where your JSON are saved  
• Now hit command as below

**newman run collection.json -e dev01.json -d datafile\_test.json -k**



As soon as the execution is completed, Newman will create an execution report in same folder if you mention a output file name in the command.

1. INTEGRATING TESTING INTO CI/CD PIPELINE

Reuse your test suites to create a CI/CD pipeline that tests at every push. You can seamlessly integrate your Postman instance with [**Jenkins**](https://learning.postman.com/docs/postman/collection-runs/integration-with-jenkins/) to create your own pipeline or add it to your existing pipeline as a build step.

**Prerequisites**

**1**. Install Node Js (<https://nodejs.org/en/>) (version – 12.18.0/latest)

2. Install npm (Version – 6.12.0/latest) (Note : npm will install with node js)

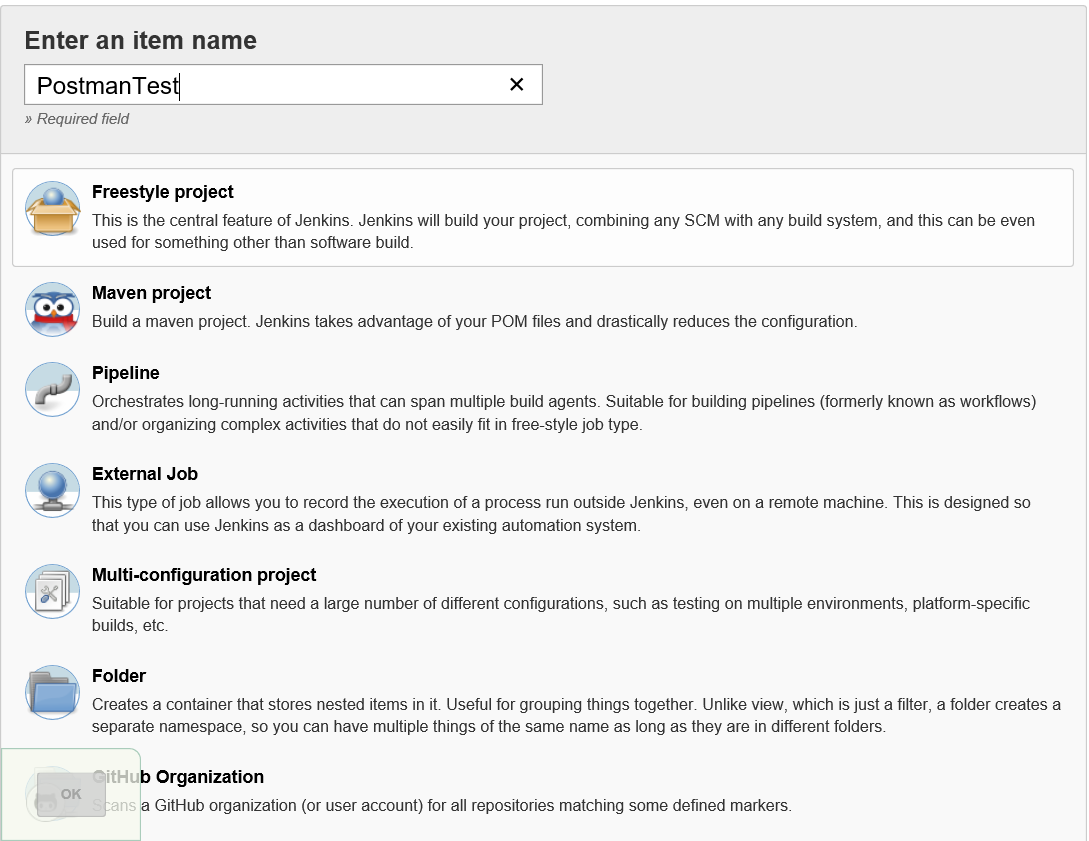
1. Install Newman (npm install -g newman) (version-latest) (https://www.npmjs.com/package/newman)

2. Install Newman Report (npm install newman-reporter-csv --save-dev ) (Version - latest)

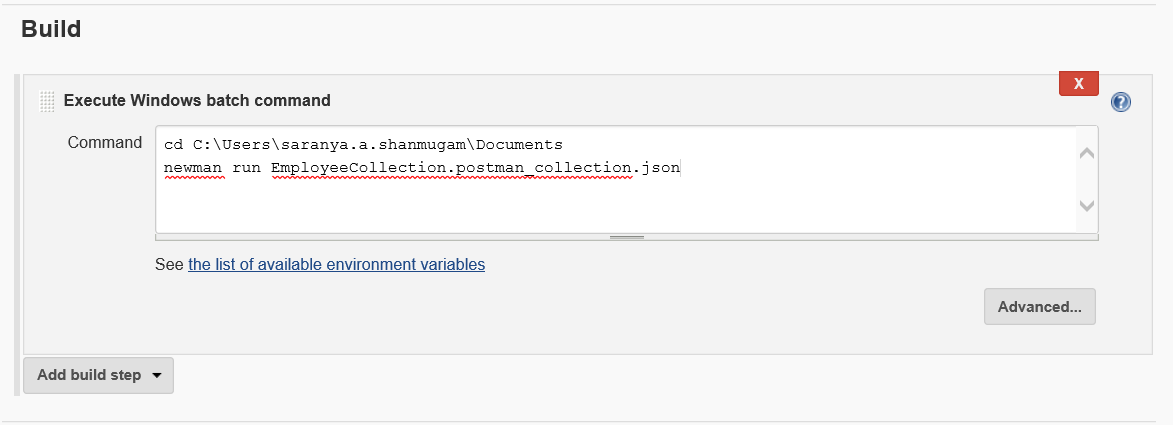
( <https://www.npmjs.com/package/newman-reporter-csv>)

**STEPS TO SETUP IN JENKINS**

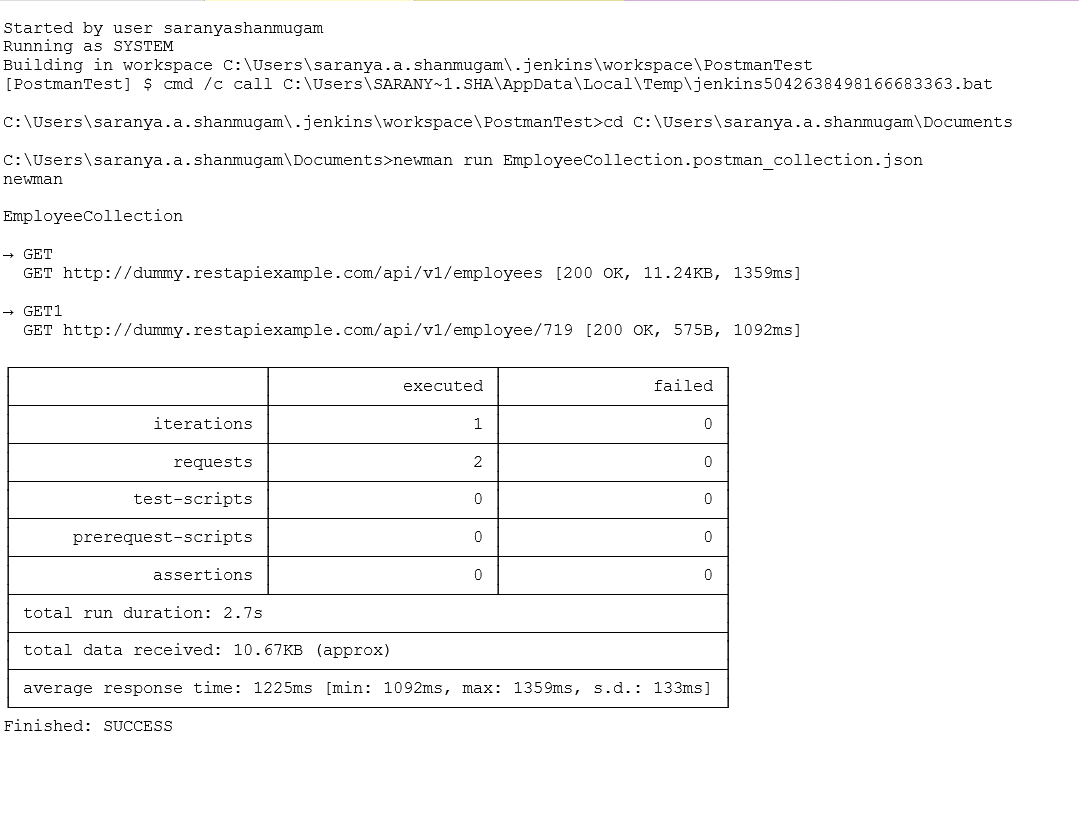
**1.**Create a Freestyle project in Jenkins



2.Configure the build section with below command to execute Postman collections test:



3. Click the "Build Now" option from Jenkins to execute every command from the build section one by one and you can see your collection result from Jenkins.



4.Configure Jenkins with the Build Triggers option to trigger the build every minute automatically so that Jenkins will check the latest change from the location where you have stored the Postman collection and it triggers the build automatically.

