What is Automated Testing?

Tests are automated by creating test suites that can run again and again. Postman can be used to automate many types of tests including unit tests, functional tests, integration tests, end-to-end tests, regression tests, mock tests, etc. Automated testing prevents human error and streamlines testing.

Postman’s Collections let you organize requests you make in Postman. They also form the basis of other useful features like mocks, monitors and documentation.

Diagram

Description automatically generated

### What is a test in Postman?

With Postman, you can add scripts to your request to use [dynamic variables](https://learning.postman.com/docs/postman/scripts/postman-sandbox-api-reference/#dynamic-variables), [pass data between requests](https://www.postman.com/docs/postman/scripts/postman_sandbox#environment-and-global-variables), and [write tests](https://learning.postman.com/docs/postman/scripts/test-scripts/). Code added under the **Pre-request Script** tab will execute before your request is sent, and code added under the **Tests** tab will execute after your response is received.

Diagram

Description automatically generated

Tests are scripts written in JavaScript that are executed after a response is received. Tests can be run as part of a single request or run with a collection of requests.

In the Postman app, the request builder at the top contains the **Tests** tab where you write your tests. The response viewer at the bottom contains a corresponding **Test Results** tab where you can view the results of your tests.

A screenshot of a computer

Description automatically generated with medium confidence

### Testing automation

You can automate your tests by integrating Postman’s command line tool Newman with your favorite Continuous Integration. Once you have built a collection, Collection runs allow you to automate your API testing, and you can schedule runs using monitors. You can integrate collection runs to your CI/CD pipeline using Postman's CLI Newman.

**API testing pipeline needs 2 key stages:**

1. **Well-defined tests for your APIs.**
2. **Ability to run your tests on-demand and on a schedule.**

Writing good tests

When it comes to testing APIs, you need to assert on the responses sent by the application. You can test for the response data-structure, presence (or absence) of specific parameters in the response, response timing, response headers, cookies and response status. This is where Postman Collections come in. A collection is a group of API requests that can be executed at a go. You can write tests for each request and for group of requests. Postman tells you how many of these tests pass or fail when you run the collection. You would have collections for each of your test suites.

Running tests

Integrate newman, Postman’s command-line collection runner, in your CI systems and run your collections on-demand as part of your CI pipeline. These run on your setup. On the other hand, Monitors in Postman let you schedule collection runs on pre-defined intervals.

# Integrate API Test Suite in AWS CodePipeline

NEWMAN CLI Options

|  |
| --- |
| $newman run -h |
|  | Options - Additional args: |
|  | Utility: |
|  | -h, --help output usage information |
|  | -v, --version output the version number |
|  | Basic setup: |
|  | --folder [folderName] Specify a single folder to run from a collection. |
|  | -e, --environment [file|URL] Specify a Postman environment as a JSON [file] |
|  | -d, --data [file] Specify a data file to use either json or csv |
|  | -g, --global [file] Specify a Postman globals file as JSON [file] |
|  | -n, --iteration-count [number] Define the number of iterations to run |
|  | Request options: |
|  | --delay-request [number] Specify a delay (in ms) between requests [number] --timeout-request [number] Specify a request timeout (in ms) for a request |
|  | Misc.: |
|  | --bail Stops the runner when a test case fails |
|  | --silent Disable terminal output --no-color Disable colored output |
|  | -k, --insecure Disable strict ssl |
|  | -x, --suppress-exit-code Continue running tests even after a failure, but exit with code=0 |
|  | --ignore-redirects Disable automatic following of 3XX responses |

After successful collection export, We need to create a buildspec file to tell AWS CodeBuild how we want to run the collection. Before running the collection, we need to install The Newman NPM package in the pre\_build phase and then call the Newman CLI with the collection we want to run in the build phase. We can also specify report options to generate an HTML file at the end. We will upload this file to S3 after CodeBuild executes the collection:

|  |
| --- |
| version:0.2 |
|  |  |
|  | phases: |
|  | pre\_build: |
|  | commands: |
|  | - npm install newman --global |
|  | build: |
|  | commands: |
|  | - newman run -r html,cli sample\_service.postman\_collection --reporter-html-export report.html |
|  |  |
|  | artifacts: |
|  | files: |
|  | - report.html |

Make sure you run the newman command in your local to ensure all the scripts are getting executed. Here is a sample output of the newman script.

Graphical user interface

Description automatically generated

Diagram

Description automatically generated

Provide the yml file information that will enable to trigger the scripts via newman. Specify where the artifacts should be placed after the tests are executed

Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, application

Description automatically generated

**Creating an S3 Bucket to Save the Report**

From AWS Console > Storage > S3, create a new bucket named postmanreport (you will need to use a unique name). You can enable version control on the S3 bucket to see historical reports.

Graphical user interface, application

Description automatically generated

**Create a CodePipeline**

1. Sign in to the AWS Management Console and open the AWS CodePipeline console at http://console.aws.amazon.com/codepipeline. Choose create a pipeline and provide a name.
2. In Repository name, choose the name of the AWS CodeCommit repository you created before.
3. In build step select the build provider (AWS Codebuild).
4. AWS CodePipeline uses Amazon CloudWatch Events to detect changes in your AWS CodeCommit source repository and branch
5. Once the tests are run reports would be available in S3.

Graphical user interface, text, application, email

Description automatically generated

Once the Pipeline is executed, you will see both the **Source** and**Build** stages in Green (unless there are any errors)

Graphical user interface, application

Description automatically generated

<https://aws.amazon.com/blogs/devops/automating-your-api-testing-with-aws-codebuild-aws-codepipeline-and-postman/>

<https://blog.postman.com/how-to-auto-sync-postman-collections-with-aws-codecommit-repositories/>