



API Testing with Postman

Nagaraj Ravinuthala
Sep 2021

About the trainer



- 19 years of IT industry experience wearing multiple hats in leadership roles in Dev, QA and Delivery
- Passionate about teaching and imparting knowledge
- 3 years of training experience with over 1000 hours of programs conducted
- Areas of expertise include Automation Testing, API Testing, Performance Testing, Python, Ansible and Network Automation
- <https://www.linkedin.com/in/nagaraj-ravinuthala/>



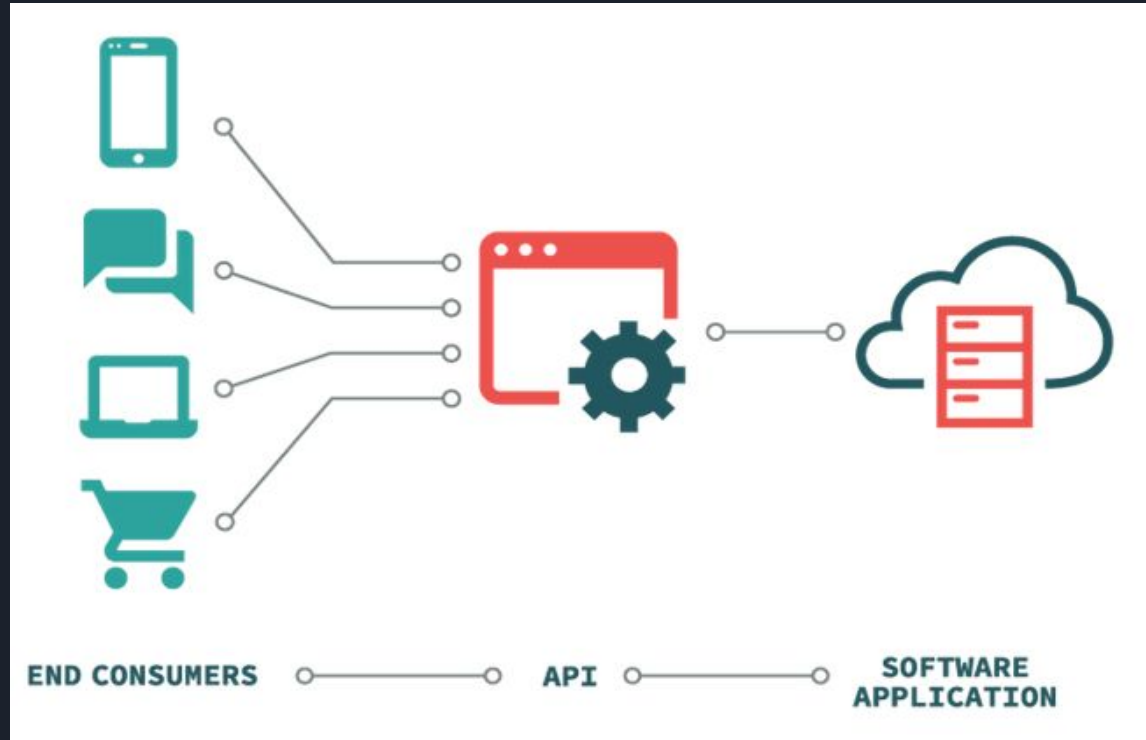
What are APIs?

- API stands for Application Programming Interface
 - Allows applications to talk to each other
- Think of the following
 - Stock tickers
 - Maps
 - Currency Conversion
 - Flight Booking
- Listed above are some of the examples of API uses

Uses of APIs



How do APIs work?





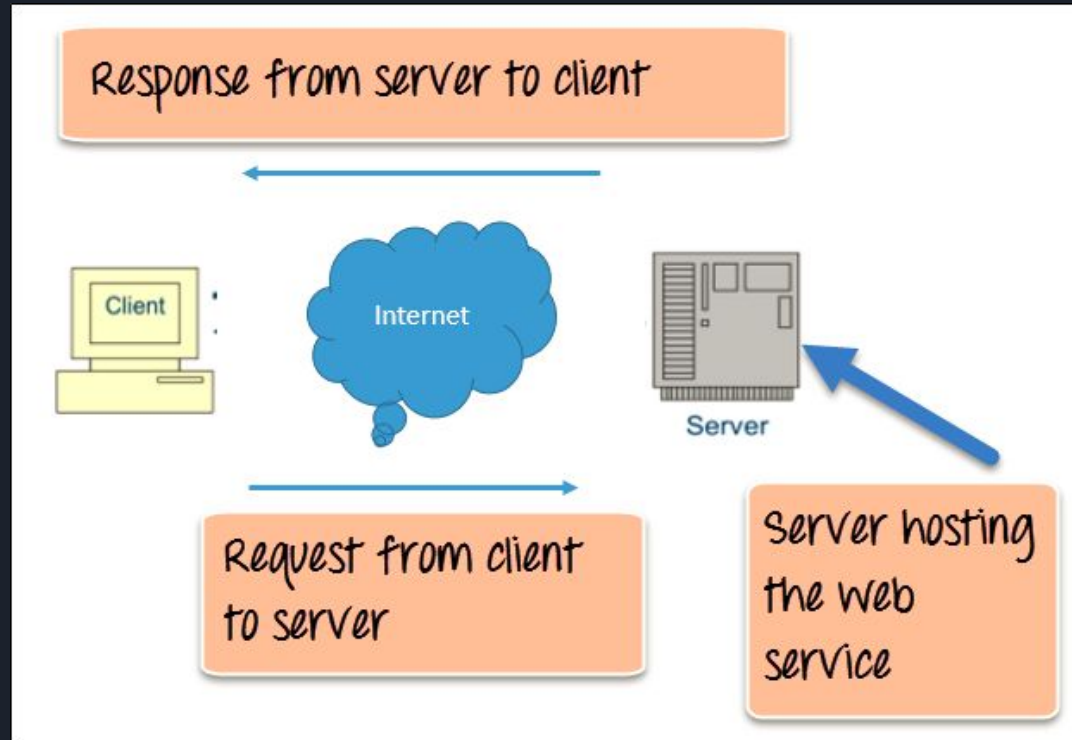
Web Services vs APIs



Web Services

- A standardized medium to propagate communication between the client and server applications on the World Wide Web (WWW)
- A software module that can be searched and invoked over internet (web) that is designed to perform a set of tasks (service)
- Simply put it is a service provided by a Provider to a Consumer over the Web
- Correlate the example we have discussed before with this
- Map Service, Stock Service, Temperature Service, Currently Conversion Service etc.

Web Services





APIs

- Well we have already seen that APIs also do pretty much the same thing as Web Services right?
- Then why two different terms?
- Well there are subtle differences
- All Web Services are APIs, but the converse is not necessarily true
- APIs need not necessarily be accessible over the Web



SOAP vs REST



Types of Web Services

- SOAP Web Services
 - Use a transport independent message exchange protocol called SOAP
 - Simple Object Access Protocol
 - Simply defines the structure of the message exchanged
 - Format used for exchange in this is mostly XML
 - Use WSDL as interface between client and server
 - Web Services Description Language



Types of Web Services (Cont.)

- RESTful Web Services
 - Use HTTP directly
 - Use JSON predominantly though they also support XML
 - Typically called as APIs
 - Usually come with documentation (e.g. Swagger) which contains all the information needed to use them



REST APIs

- REST -- **RE**presentational **State Transfer**
- Centered around Web Resources in a textual **representation**
- These resources can be read and modified via a stateless protocol like HTTP using a set of predefined operations like GET, POST, PUT, DELETE etc.
- What do we mean by Stateless?
- E.g. Phone call vs Email



REST APIs (Contd...)

- Follow Object Oriented programming paradigm of noun-verb
- There is an object or entity (noun) and we perform some actions on that (verb)
- Verbs are the actions performed on the nouns or resources
- Actions are performed by sending a request to the server
- And results of the actions are sent back to the client as response
- This request and response happens via one of the machine-readable data interchange formats we discussed earlier like XML, YAML, JSON etc.

REST APIs (Contd...)

Sample API endpoint : <https://petstore.swagger.io/v2/pet/10>

Resource

Base URL

GET

/pet/{petId} Find pet by ID

PUT

/pet Update an existing pet

DELETE

/pet/{petId} Deletes a pet

POST

/pet/{petId}/uploadImage uploads an image



HTTP Error Code Categories

Responses are grouped in five classes:

- Informational responses (100 – 199)
- Successful responses (200 – 299)
- Redirects (300 – 399)
- Client errors (400 – 499)
- Server errors (500 – 599)



XML, JSON

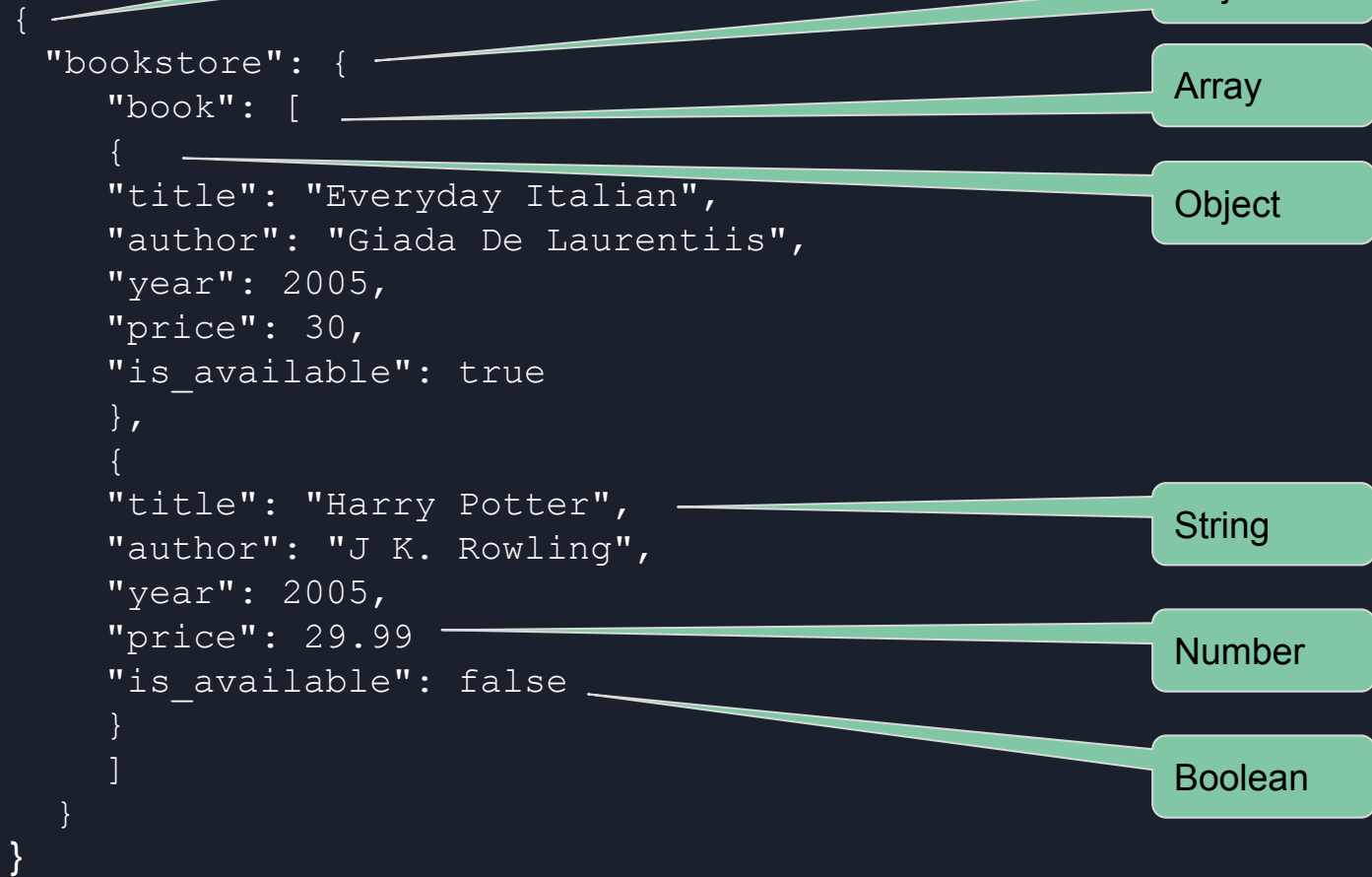
- What is the main component which is exchanged between client and server?
- Data
- This Data has to be represented in a standard way
- That is where XML or JSON come into picture
- XML - Extended Markup Language
 - A distant cousin of HTML (Hyper Text Markup Language)
- JSON - Javascript Object Notation



XML

```
<bookstore>
  <book>
    <title>Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book>
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

JSON

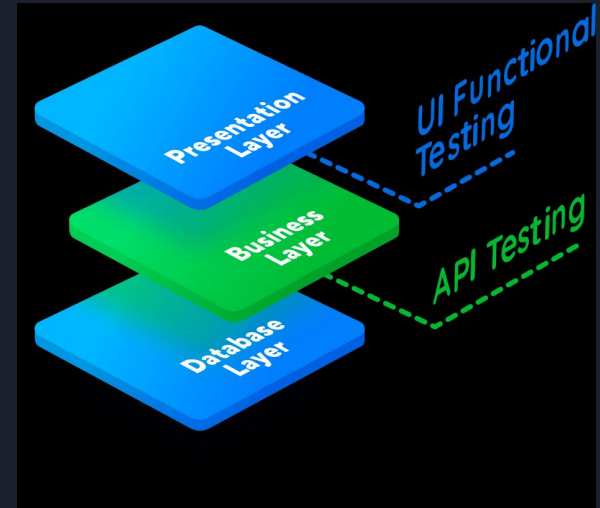




API Testing

Getting Started

- What?
 - Software testing practice that tests the APIs directly
- Why?
 - To find the defects early without waiting for the UI to be ready
- Types
 - Functionality
 - Reliability
 - Performance
 - Security





Where is it performed?

- Typical applications have 3 layers
 - UI or Presentation
 - Business Logic
 - Database
- APIs come in the Business Logic layer
- UI can keep changing based on end users likes and dislikes
- Database layer usually does not change once finalized



Importance of API Testing

- Now a days applications are built with core logic independent of backend database and frontend UI
- API layer can be thought of as backbone of the applications
- Hence it is critical that this layer:
 - Does what is it supposed to do (functionality)
 - Does not become a bottleneck (performance)
 - Does not have loopholes that can be exploited by hackers (security)
 - Available when needed (reliability)

API Testing Tools

- Popular Tools used for API Testing are
 - SoapUI
 - Postman





Postman



Installation of Postman

- Installing Postman is straight forward
- Download Postman from [Download Postman | Get Started for Free](#) for your OS
- Double click the installer
- Installation is done in a jiffy and you are ready to start using Postman



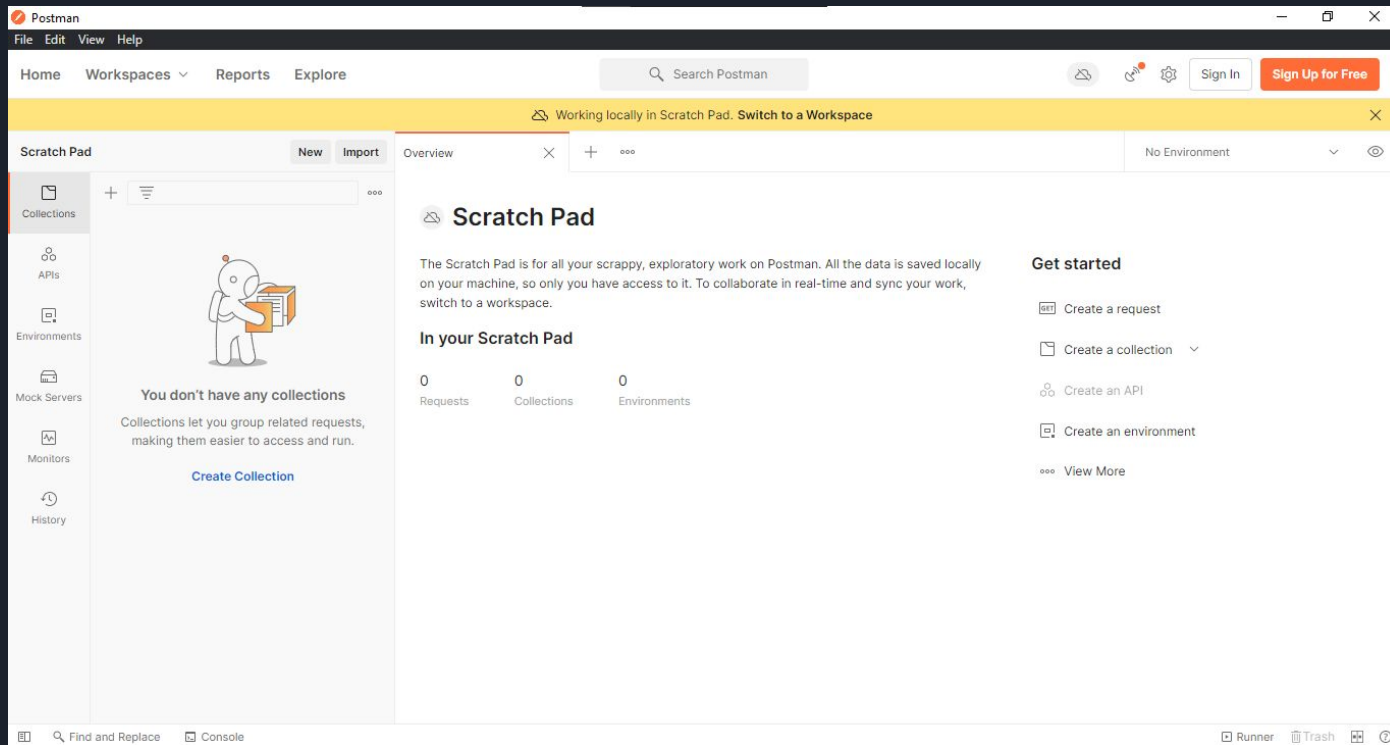
Finding your way through



Scratch Pad vs Workspace Modes

- For someone who is just getting started with Postman, the UI can be quite intimidating
- Postman allows the user to work in 2 modes
 - Scratch Pad or local mode
 - Workspace or cloud mode
- Scratch Pad mode does not require the user to sign in and all the work gets stored locally on the system where Postman is installed
- Workspace mode requires the user to register and sign in and all the work gets synchronized with the server so that it is available from any other system or device where the user might login

Scratch Pad



Workspace (Local)

grey-capsule-107831.postman.co

The screenshot displays the Postman web application interface. At the top, the navigation bar includes 'Home', 'Workspaces', 'API Network', 'Reports', and 'Explore'. A search bar is located on the right. The main content area features a greeting 'Good afternoon, Nagaraj Ravinuthala!' and a prompt to 'Pick up where you left off, catch up with your team's work.' Below this, there are several cards for getting started with Postman, including 'Start with something new', 'Import an existing file', 'Explore our public network', and 'Work smarter with Postman'. On the left sidebar, the 'Workspaces' section is expanded, showing a list of workspaces. The workspace 'grey-capsule-107831' is highlighted, with its URL 'grey-capsule-107831.postman.co' and an 'Invite' button. Below the workspace list, there is a 'Help' section with links to the 'Learning Center', 'Support Center', and 'Bootcamp'. On the right sidebar, there are sections for 'Announcements' and 'Activity Feed'. The 'Activity Feed' shows a list of recent activities, including 'Nagaraj Ravinuthala removed Private User from Team Workspace workspace' and 'Nikhil deleted the Account Service collection in AccountWorkSpace workspace'.

Postman

File Edit View Help

Home Workspaces API Network Reports Explore

Search

Upgrade

Good afternoon, Nagaraj Ravinuthala!

Pick up where you left off, catch up with your team's work.

Get started with Postman

Start with something new

Create a new request, collection, or API in a workspace

Create New →

Import an existing file

Import any API schema file from your local drive or Github

Import file →

Explore our public network

Browse featured APIs, collections, and workspaces published by the Postman community.

Explore →

Work smarter with Postman

Learn how Postman can help you at every stage of the API development.

Learn →

Workspaces

Private API Network

Integrations

Help

Learning Center

Support Center

Bootcamp

grey-capsule-107831

grey-capsule-107831.postman.co

Invite

Announcements

Activity Feed

March 8, 2021

Nagaraj Ravinuthala removed Private User from Team Workspace workspace

3:53 PM

Nagaraj Ravinuthala removed Private User from Team Workspace workspace

3:53 PM

Nagaraj Ravinuthala removed Private User from Team Workspace workspace

3:52 PM

Nikhil deleted the Account Service collection in AccountWorkSpace workspace

12:37 PM

Account Service collection archived. Your Free team reached its shared request limit. Learn more

12:37 PM

NAME LAST VIEWED MEMBERS

Workspace (On cloud)

The screenshot shows the Postman web interface for a workspace named 'grey-capsule-107831'. The browser address bar shows the URL 'https://grey-capsule-107831.postman.co'. The interface includes a navigation bar with links to Home, Workspaces, API Network, Reports, and Explore. A search bar is also present. A yellow banner at the top indicates that the team is full and suggests upgrading the plan. The main content area is divided into several sections: a sidebar on the left with links to Workspaces, Private API Network, and Integrations; a central 'Get started with Postman' section with options to start with something new, import an existing file, explore the public network, or work smarter with Postman; and a right sidebar with announcements and an activity feed. The activity feed shows recent actions by team members, such as removing private users and deleting collections.

Home Workspaces API Network Reports Explore Search Postman

Looks like your team is full. To expand, organize, manage your team effortlessly, [upgrade your plan](#).

Good afternoon, Nagaraj Ravinuthala!

Pick up where you left off, catch up with your team's work.

Get started with Postman

- Start with something new**
Create a new request, collection, or API in a workspace
[Create New](#)
- Import an existing file**
Import any API schema file from your local drive or Github
[Import file](#)
- Explore our public network**
Browse featured APIs, collections, and workspaces published by the Postman community.
[Explore](#)
- Work smarter with Postman**
Learn how Postman can help you at every stage of the API development.
[Learn](#)

grey-capsule-107831
grey-capsule-107831.postman.co

[Invite](#)

Workspaces
Private API Network
Integrations

Help
Learning Center
Support Center
Bootcamp

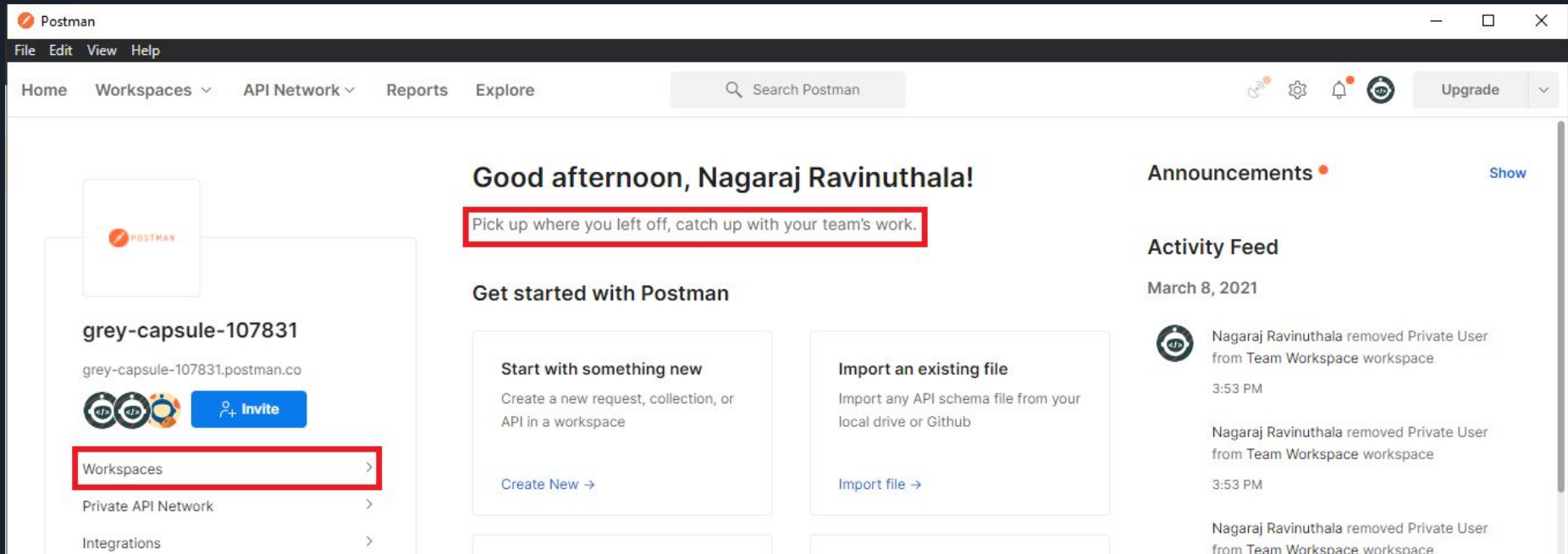
Announcements

Activity Feed
March 8, 2021

- Nagaraj Ravinuthala removed Private User from Team Workspace workspace
3:53 PM
- Nagaraj Ravinuthala removed Private User from Team Workspace workspace
3:53 PM
- Nagaraj Ravinuthala removed Private User from Team Workspace workspace
3:52 PM
- Nikhil deleted the Account Service collection in AccountWorkSpace workspace
12:37 PM
- Account Service collection archived. Your Free team reached its shared request limit. [Learn more](#)

Workspace (Cont.)

- Workspace helps you work in collaboration with your team
- Click on Workspaces to see existing or to create new workspaces



The screenshot shows the Postman web application interface. The top navigation bar includes 'Home', 'Workspaces', 'API Network', 'Reports', and 'Explore'. The 'Workspaces' menu item is highlighted with a red box. The main content area displays a greeting 'Good afternoon, Nagaraj Ravinuthala!' and a message 'Pick up where you left off, catch up with your team's work.' Below this, there are two cards: 'Start with something new' and 'Import an existing file'. The right sidebar contains 'Announcements' and 'Activity Feed' sections.

Postman

File Edit View Help

Home Workspaces API Network Reports Explore

Search Postman

Upgrade

Good afternoon, Nagaraj Ravinuthala!

Pick up where you left off, catch up with your team's work.

Get started with Postman

Start with something new

Create a new request, collection, or API in a workspace

Create New →

Import an existing file

Import any API schema file from your local drive or Github

Import file →

Announcements

Show

Activity Feed

March 8, 2021

Nagaraj Ravinuthala removed Private User from Team Workspace workspace

3:53 PM

Nagaraj Ravinuthala removed Private User from Team Workspace workspace

3:53 PM

Nagaraj Ravinuthala removed Private User from Team Workspace workspace

3:53 PM

grey-capsule-107831

grey-capsule-107831.postman.co

Workspaces

Private API Network

Integrations

Invite

Workspace (Cont.)

- By default you can see the following workspaces: My Workspace and Team Workspace
- You can create your own workspace using New Workspace button



Home / Workspaces



grey-capsule-107831's Workspaces



A directory of your team's workspaces.

New workspace

All workspaces ▾ Sort by: A to Z ▾

 AccountWorkSpace 

 **My Workspace**
This is your personal, private workspace to play around in. Only you can see the collections and APIs you create here - unless you ... 

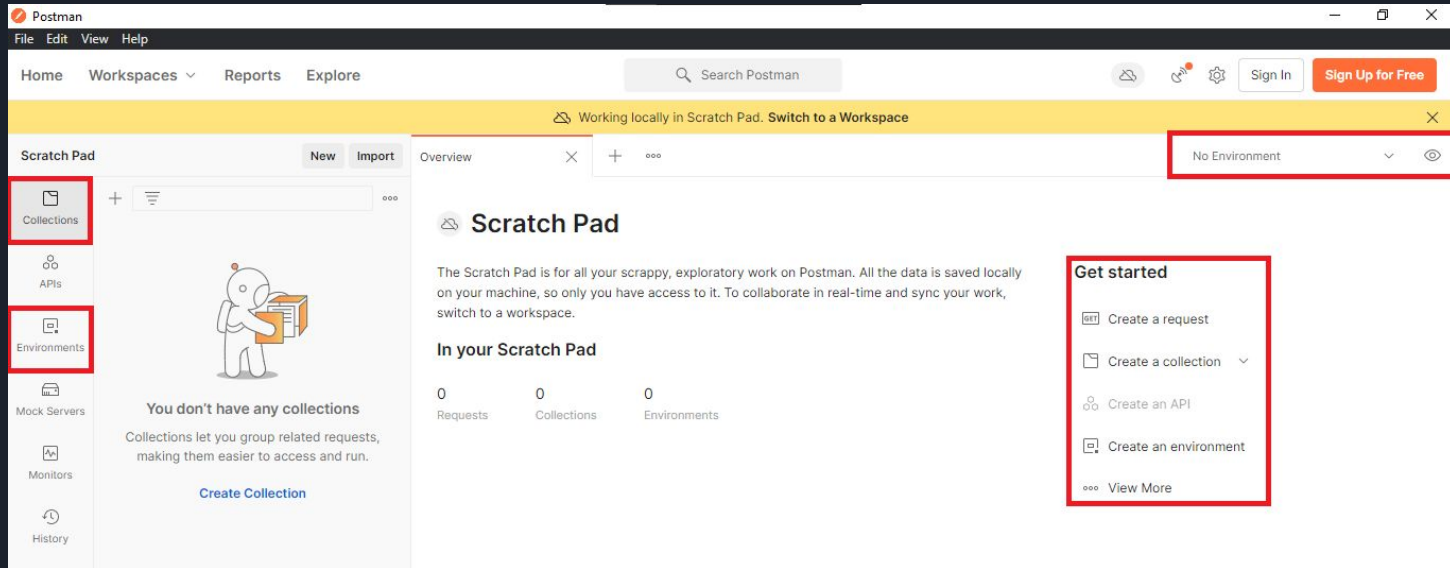
 **Team Workspace**
This is where the collaboration happens. Use this space to share and collaborate on APIs, collections, environments, monitors, and... 



Working in Scratch Pad vs Workspace

- If:
 - you are working independently
 - saving your work on the local machine is enough
 - Scratch Pad is sufficient
- If:
 - you want to work as part of a team
 - collaborate with your team
 - save your work to the cloud so that it can be accessed by other team members
 - then you can work in Workspace mode

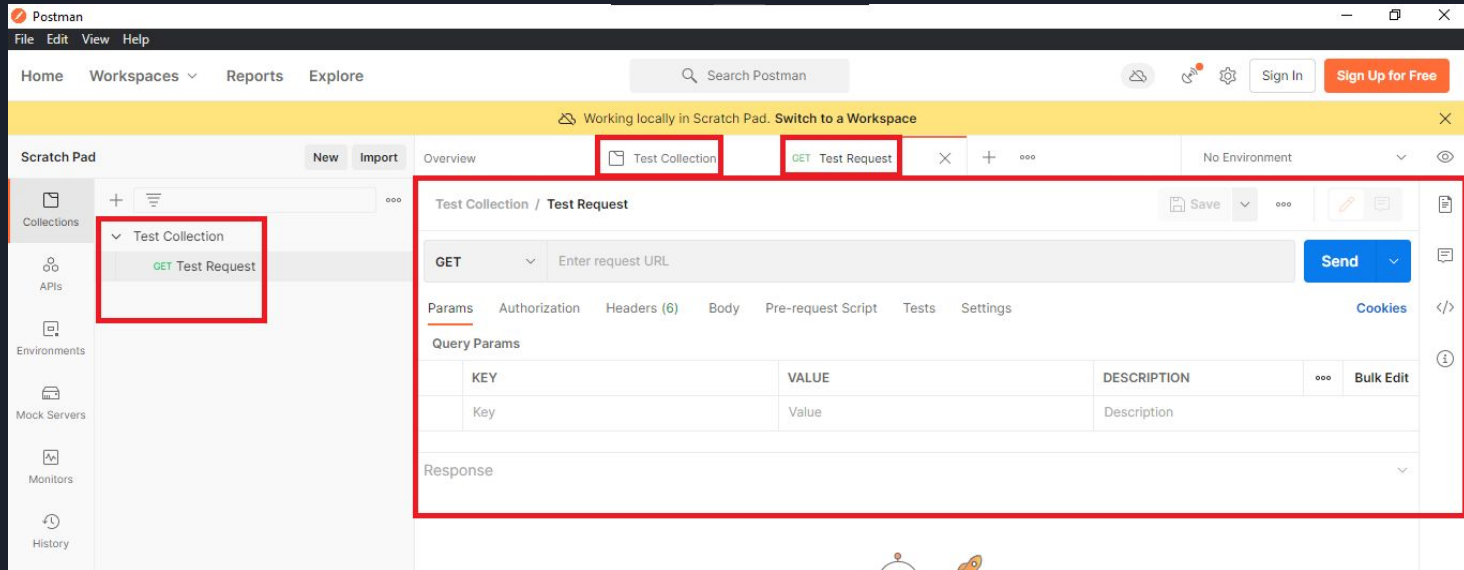
Navigating Scratch Pad Mode



Important sections which we will work with are highlighted in the figure:

- Collections
- Environments
- Get Started
- Variables

Navigating Scratch Pad Mode



Above figure shows the important areas we will be using during our work:

- Collections Explorer
- Main work area which consists of Collections, Requests etc.



Creating Collections and Requests

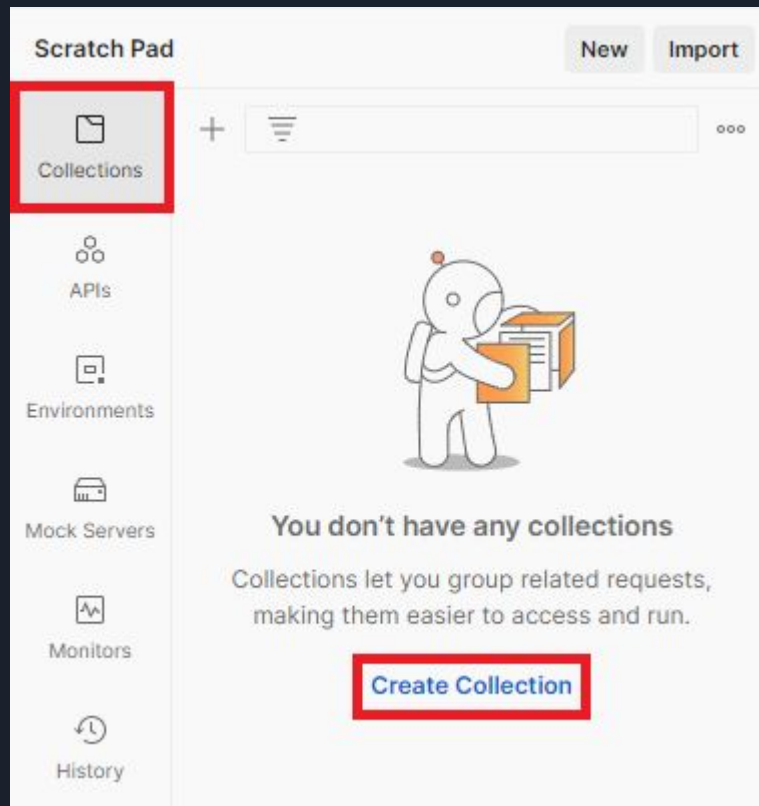


Collections

- Simply put Collections are groups of requests used to keep the workspace organized
- Typically all the requests corresponding to a module can be grouped under a collection
- Some uses of grouping Requests into Collections are:
 - To collaborate with teammates
 - To generate API Documentation
 - To organize test suites
 - To automate test runs

Collections (Cont.)

- Creating a collection is the starting point of working with API requests
- Click on Collections tab in the left navigation bar and click Create Collection link
- Remember to group related requests under collections





Collections (Cont.)

- You can also optionally specify a description
- Description will appear in the documentation when generated
- Other settings that can be configured at Collection level are:
 - Authorization
 - Pre-request script
 - Tests
 - Variables
- What are all these, will be discussed further but they are defined at Collection level so that they apply to all requests within it



Requests

- Request is nothing but hitting a remote API using any client (Postman in this case)
- Right click on the Collection and click Add Request
- Note that the default method selected is GET
- Change the method as needed
- Give it a name and fill the other required details
- One primary information to be provided is the request URL or the endpoint URL

Requests (Cont.)

The screenshot displays the Postman application interface. The top navigation bar includes 'File', 'Edit', 'View', and 'Help'. Below it, a secondary bar shows 'Home', 'Workspaces', 'Reports', and 'Explore', along with a search bar and a 'Sign In' button. A yellow banner indicates 'Working locally in Scratch Pad. Switch to a Workspace'. The main interface is divided into a left sidebar and a central workspace. The sidebar contains 'Collections', 'APIs', 'Environments', 'Mock Servers', 'Monitors', and 'History'. The central workspace is titled 'Scratch Pad' and shows a 'New Collection' with a 'New Request' button. The request is configured with a 'GET' method and the URL 'http://netbox.nexariacloud.com:8000/api/circuits/circuit-terminations/'. The 'Authorization' tab is selected, showing a dropdown menu with options like 'No Auth', 'API Key', 'Bearer Token', 'Basic Auth', 'Digest Auth', 'OAuth 1.0', 'OAuth 2.0', 'Hawk Authentication', and 'AWS Signature'. The 'Response' tab is also visible. The 'Send' button is highlighted in blue. A cartoon astronaut character is at the bottom right, with the text 'Click Send to get a response'.

Postman

File Edit View Help

Home Workspaces Reports Explore

Search Postman

Sign In Sign Up for Free

Working locally in Scratch Pad. Switch to a Workspace

Scratch Pad

New Import

Overview GET New Coll... GET New Req... GET New Req...

New Collection New Request

Save

GET http://netbox.nexariacloud.com:8000/api/circuits/circuit-terminations/ Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Type

Inherit a...

The authorization header is automatically generated from the selected authentication type.

Response

Inherit auth from parent collection

No Auth

API Key

Bearer Token

Basic Auth

Digest Auth

OAuth 1.0

OAuth 2.0

Hawk Authentication

AWS Signature

This request is using No Auth from collection [New Collection](#).

Click Send to get a response



Requests (Cont.)

- The type of requests are the same as HTTP verbs
- GET, POST, PUT, PATCH, DELETE
- These correspond to the CRUD operations from the DB world
- GET - READ
- POST - CREATE
- PUT, PATCH - UPDATE
- DELETE - DELETE

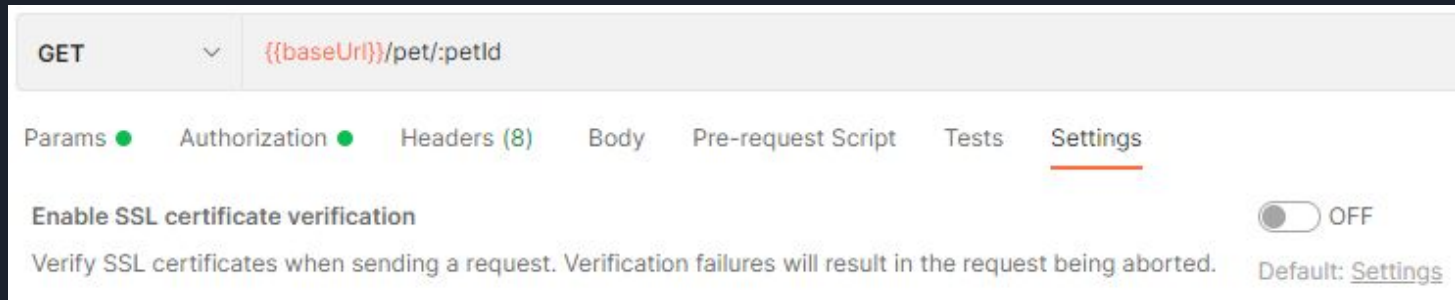


HTTPS Requests

- HTTPS is secure version of HTTP
- Refer [How To Secure Your Site with HTTPS | Google Search Central](#) to know more about https
- HTTPS involves a certificate usually issues by a certificate authority (CA) (paid) or self signed certificate (free)
- Browsers have the functionality in built to download this certificate from the server the first time and validate it for subsequent requests
- But Postman cannot do this
- So we disable SSL Verification done by Postman
- It means that we trust that the certificate is valid and move ahead

HTTPS Requests (Cont.)

- We get the error shown here is SSL verification is not disabled
- This can be done either globally under Settings or per request under the Settings tabs of the request





GET vs POST



GET Method

- Used to fetch the data about a specified resource
- It is the most common HTTP method
- The query string used to fetch the resource is sent in the URL itself
 - <https://petstore.swagger.io/v2/pet/findByStatus?status=available&status=sold>
- Can be cached
- Remain in browser history
- Can be bookmarked
- Have length restrictions on the data the can be sent in the requests



POST Method

- Used to send data to server to create/ update resource
- The Data sent to the server is sent in body and not in the URL
 - <https://petstore.swagger.io/v2/pet>
- Are never cached
- Do not remain in browser history
- Are never bookmarked
- Have no length restriction on the data the can be sent in the requests

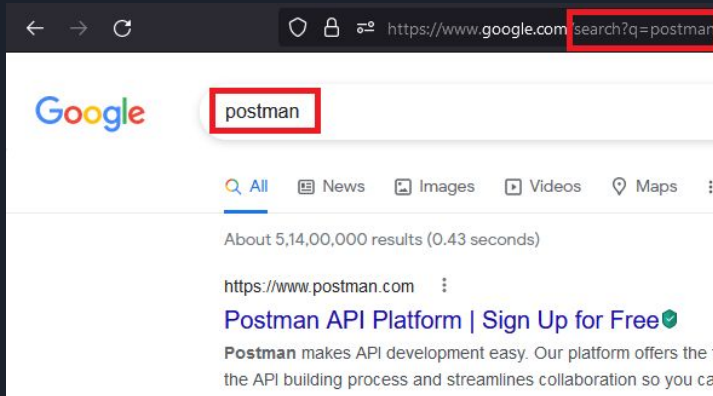


Request Parameters

- In Postman we have two types of request parameters
 - Query parameters
 - Path parameters
- Let us see them in detail

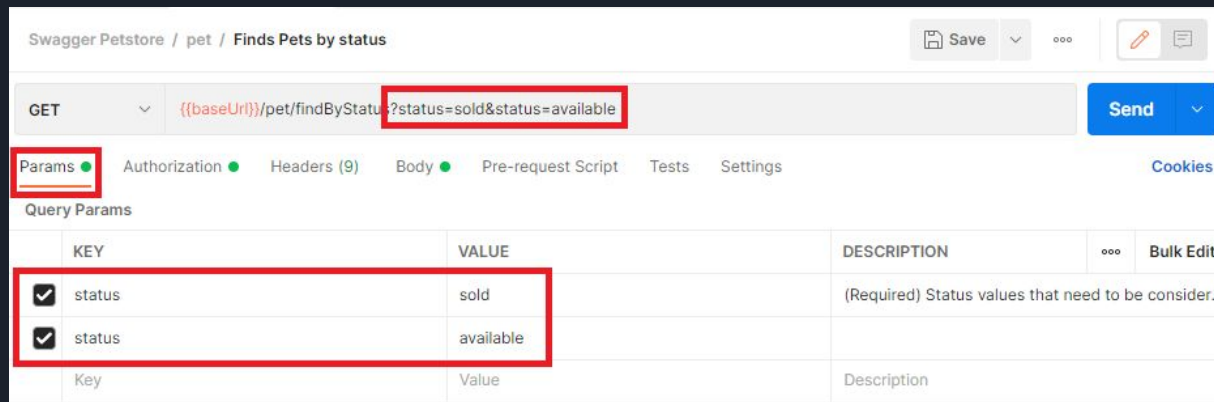
Query Parameters

- Query Parameters are options passed to the endpoint to send additional data to the server
- Are appended to the endpoint using “?”
- Passed as key, value pairs
- E.g. `/pet/findByStatus?status=sold&status=available`



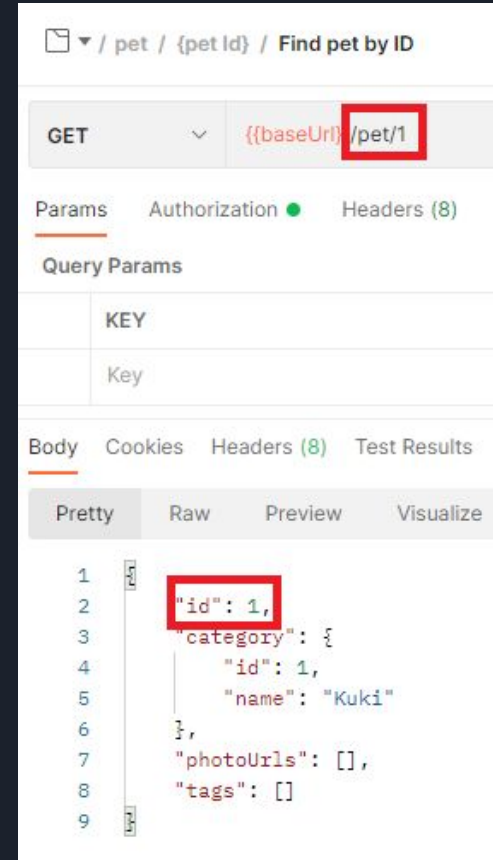
Query Parameters (Cont.)

- In Postman, query parameters can be set at the request level
- Under the request, go to the Params tab to set the query parameters
- Additional information like whether they are optional or mandatory can be provided in the description



Path Parameters

- Sometimes the additional information forms part of the endpoint url itself instead of being passed as query parameters
- GET method to fetch the pet by id requires the id to be passed in the endpoint url itself
- In such case, should we create 10 separate requests to fetch pets with 10 different ids?
- This is where the path parameter comes handy



The screenshot shows a REST client interface with the following details:

- URL Bar:** `pet / {petId} / Find pet by ID`
- Method:** `GET`
- URL:** `{{baseUrl}}/pet/1` (The `/pet/1` part is highlighted with a red box).
- Params:** Tab selected.
- Query Params:** Table with one row:

KEY
Key
- Body:** Tab selected.
- Response:** Pretty view of JSON. Line 2 shows `"id": 1,` (highlighted with a red box). The full response is:

```
1 {
2   "id": 1,
3   "category": {
4     "id": 1,
5     "name": "Kuki"
6   },
7   "photoUrls": [],
8   "tags": []
9 }
```

Path Parameters (Cont.)

- Path parameters can be specified using a placeholder name preceded by “:”
- In this case we can choose a placeholder name as “id” or “petid”
- Then the endpoint url becomes
 - <https://petstore.swagger.io/v2/pet/:id>
 - <https://petstore.swagger.io/v2/pet/:petid>
- Accordingly it is going to appear under the Params tab

The screenshot shows a REST client interface for a GET request to the endpoint `{baseUri}/pet/{petId}`. The `petId` path variable is highlighted with a red box and set to the value `1`. The response is displayed in the `Body` tab, showing a JSON object with the following structure:

```
1 {
2   "id": 1,
3   "category": {
4     "id": 1,
5     "name": "Kuki"
6   },
7   "photoUrls": [],
8   "tags": []
9 }
```

The `"id": 1,` line is highlighted with a red box.

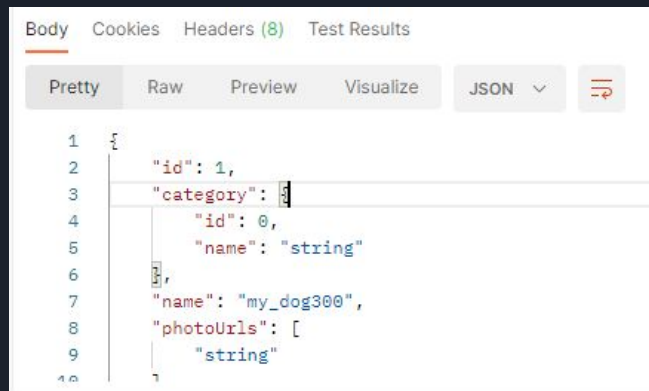


Understanding Responses

- When we send a request and receive a response, we can use the response viewer to know more about it
- Just like request, the response consists of body and headers
- Additionally response comes with a status code and message to help us understand whether it was success or failure
- As we saw earlier successful responses will have response code in 200 series
- Failed responses will have response code in either 400 or 500 series depending on whether the failure was due to client side or server side issue

Viewing Responses

- Body tab of the response section shows the main response body
- Response is shown in raw or pretty formats
- Headers tab shows response headers
- Test Results tab shows the results of test (if we added any tests)
- More on the tests later ...

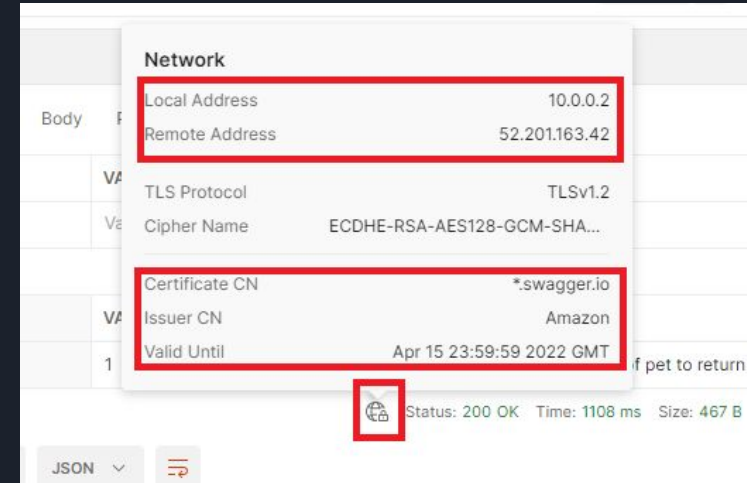


The screenshot shows a web client interface with a 'Body' tab selected. The response is displayed in a 'Pretty' format, showing a JSON object with the following structure:

```
1 {
2   "id": 1,
3   "category": 0,
4   "id": 0,
5   "name": "string"
6 },
7 "name": "my_dog300",
8 "photoUrls": [
9   "string"
10 ]
}
```

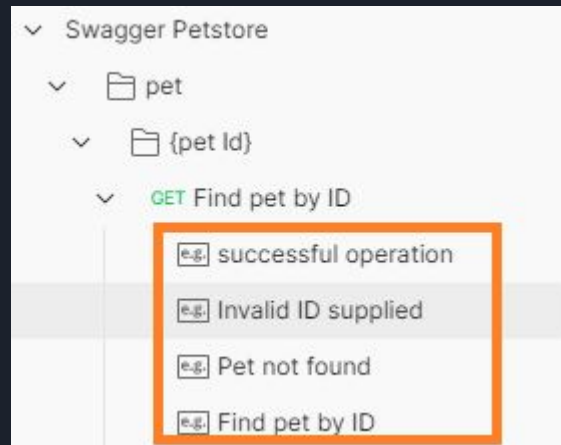
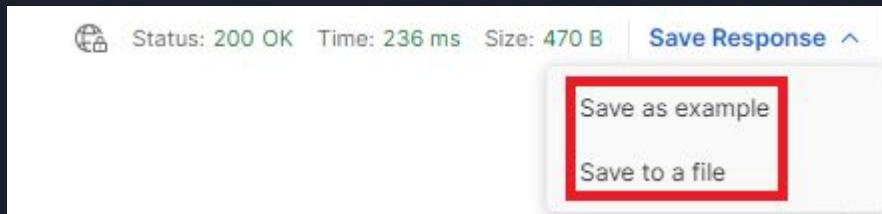
Viewing Responses (Cont.)

- The network related information like the local and remote IP, certificate information for secure requests etc. can be found by hovering the mouse over the web icon on the tabs section of the response viewer
- For https response there is a lock icon on the web icon
- It also shows issuer of the certificate, validity etc.



Saving Responses

- We can save the responses either in an external file or as examples
- Saving response as an example, helps others to understand more about it
- Saving response to a file will be useful in automation if we want to do response validation via a script





API Authentication



Authorizing requests

- Authorization ensures that client requests access the data securely
- Postman supports a wide variety of authorization mechanisms
- API Developers make use of them as per the organizational needs
- API Testers will just provide authentication/ authorization details as specified in the API documentation
- Most commonly used are:
 - Basic Auth
 - API Key
 - Bearer Token

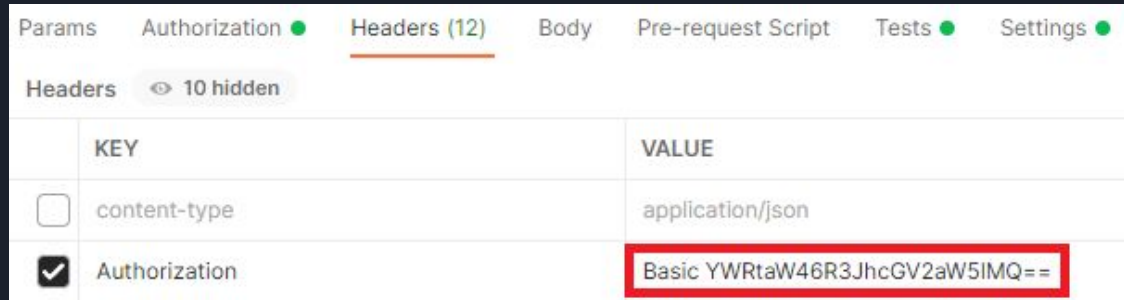


Authorizing requests (Cont.)

- Authorization can be set using the Authorization tab under the request
- If we choose No Auth Postman will not send any authorization data with the request
- If the API does not expect any credentials, it returns the data without any authorization
- We can also inherit auth details from the parent
- If the requests are grouped into collections/ folders, auth details can be set at collection/ folder level and they will be inherited by each request

Basic Auth

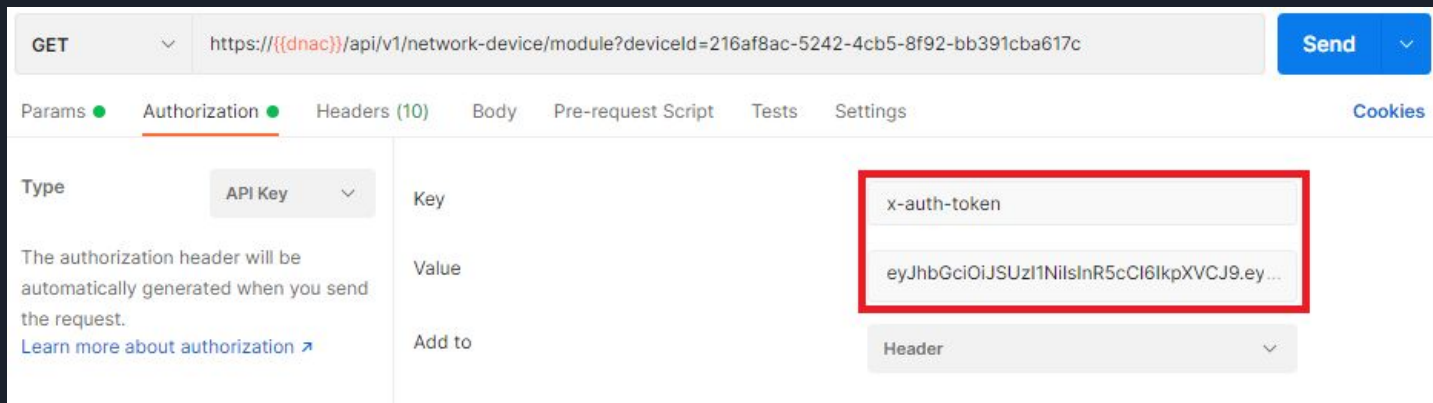
- Basic Auth involves using a verified username and password to authenticate
- Choose Basic Auth in the dropdown and enter username and password
- Base64 encoded username and password will be generated and appended with the word Basic and set as a header



Params	Authorization ●	Headers (12)	Body	Pre-request Script	Tests ●	Settings ●
Headers 10 hidden						
	KEY	VALUE				
<input type="checkbox"/>	content-type	application/json				
<input checked="" type="checkbox"/>	Authorization	Basic YWRtaW46R3JhcGV2aW5IMQ==				

API Key

- Choosing API Key, we can send a key value pair either in the header or in query parameters
- In the Authorization tab, choose API Key, enter key, value, choose Header
- Note that the API Key is set in the header



The screenshot shows the Postman interface with the Authorization tab selected. The URL bar at the top displays a GET request to `https://(dnac)/api/v1/network-device/module?deviceId=216af8ac-5242-4cb5-8f92-bb391cba617c`. Below the URL bar, the Authorization tab is active, showing a configuration for an API Key. The 'Type' is set to 'API Key'. The 'Key' field contains 'x-auth-token' and the 'Value' field contains 'eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...'. The 'Add to' dropdown is set to 'Header'. A red rectangle highlights the 'Key' and 'Value' fields.

GET `https://(dnac)/api/v1/network-device/module?deviceId=216af8ac-5242-4cb5-8f92-bb391cba617c` Send

Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies

Type: API Key

The authorization header will be automatically generated when you send the request.
[Learn more about authorization](#)

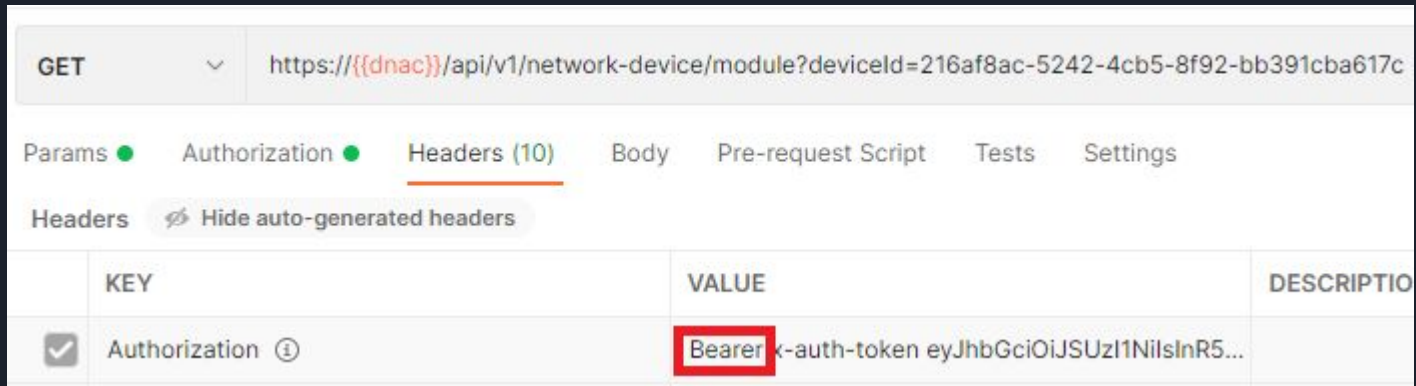
Key: x-auth-token

Value: eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...

Add to: Header

Bearer Token

- Choosing Bearer Token lets user authenticate an API using access keys like JSON Web Tokens
- When selected, Postman will append the text “Bearer” to the entered token value





Variables



Introduction to Variables

- Variables are like placeholders for data associated with a name
- We can define a variable once by giving it a name, store the data in it and use it anywhere else, any number of times using the name given
- Suppose we have added close to a hundred requests in Postman
- The developers have changed the endpoint URL
- All our requests will fail, since the endpoint URL has changed
- To make them pass, we need to edit all 100 requests and modify the endpoint URL with the new value
- Tedious and inefficient isn't it?

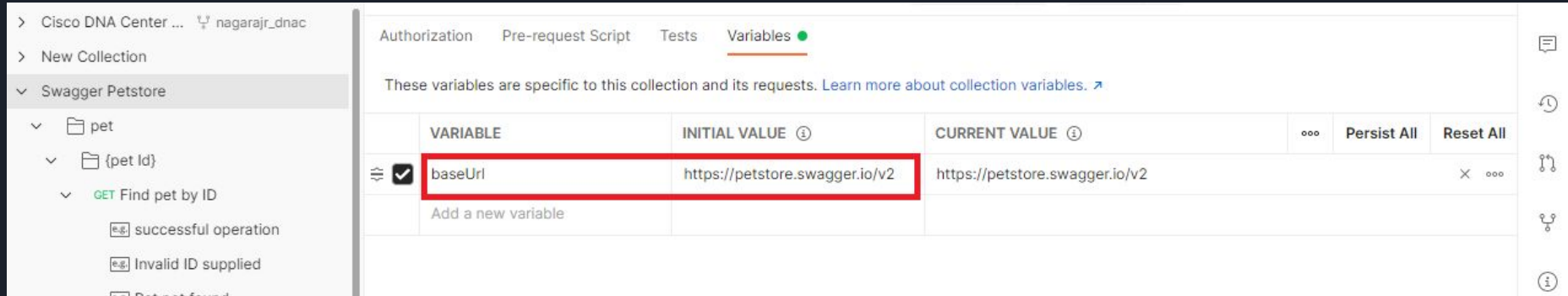


Introduction to Variables (Cont.)




- Instead, we can create a variable to store the hostname or IP Address of the server which might change
- Refer to this variable in all the 100 requests
- If the endpoint URL changes in future, we just need to modify it in Postman just once
- All our old requests continue to work fine
- Variables can be created at the Collection level and referenced for all the requests within that collection

Using Variables

- Click on a Collection and go to Variables tab
- Enter a variable name and set an initial value
- Click on Persist All



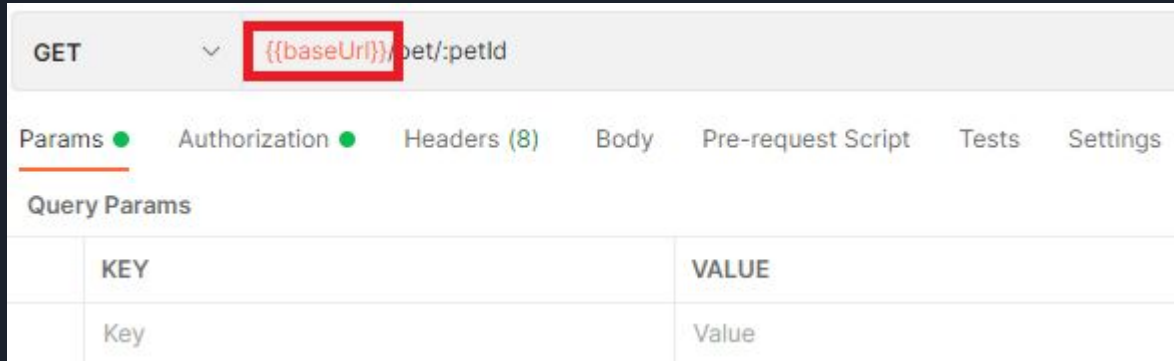
The screenshot displays the Swagger Petstore API interface. On the left, a sidebar shows the collection structure: 'Cisco DNA Center ...' (nagarajr_dnac), 'New Collection', and 'Swagger Petstore'. Under 'Swagger Petstore', there is a 'pet' collection containing a '{pet Id}' sub-collection, which includes a 'GET Find pet by ID' endpoint. The main panel shows the 'Variables' tab selected, with a red underline. Below the tab, a message states: 'These variables are specific to this collection and its requests. [Learn more about collection variables.](#)'. A table lists the variables:

	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ		Persist All	Reset All
	<input checked="" type="checkbox"/> baseUrl	https://petstore.swagger.io/v2	https://petstore.swagger.io/v2	 		
	Add a new variable					

On the right side of the interface, there are several icons: a list icon, a refresh icon, a link icon, a share icon, and an information icon.

Using Variables (Cont.)

- The variables created in the above slide can be accessed as follows
- `{{baseUrl}}` will now return the value we stored while creating the variable
- E.g. `{{baseUrl}}/pet` etc. can now be used in requests
- If the hostname or IP Address changes, we just need to go to the collection, variables section and modify once





Types of Variables

- In Postman, variables can be created at various levels as follows:
 - Locally within pre-request or test scripts
 - At the collection level
 - At the environment level
 - Variables passed from data files
 - At the Global level

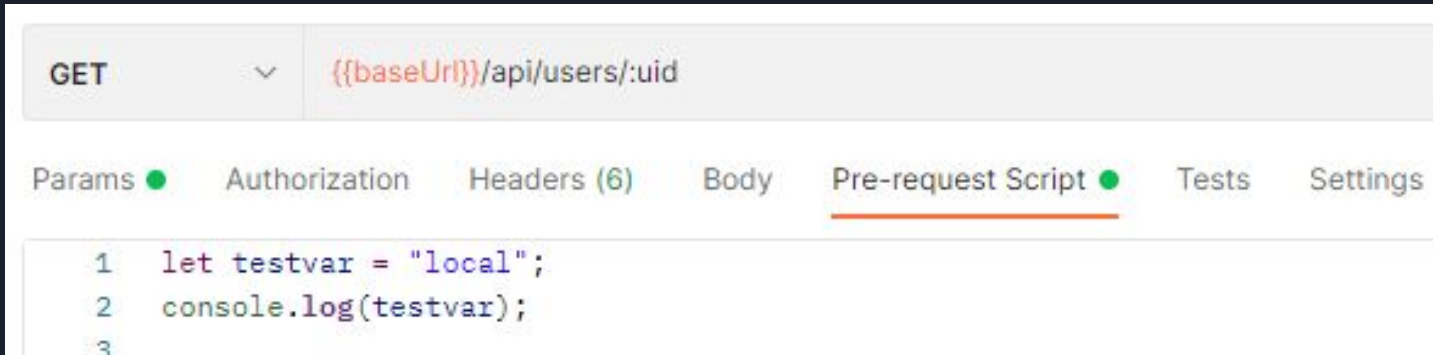


Scope of Variables

- Scope if the precedence given by Postman to the variables when they are referenced
- The preference is shown below in the descending order
 - Local (Visible only within the request)
 - Data (Variables read from data files)
 - Collection (Visible to all the requests within the collection)
 - Environment (Visible to any request using the environment)
 - Global (Visible to any request within the workspace)

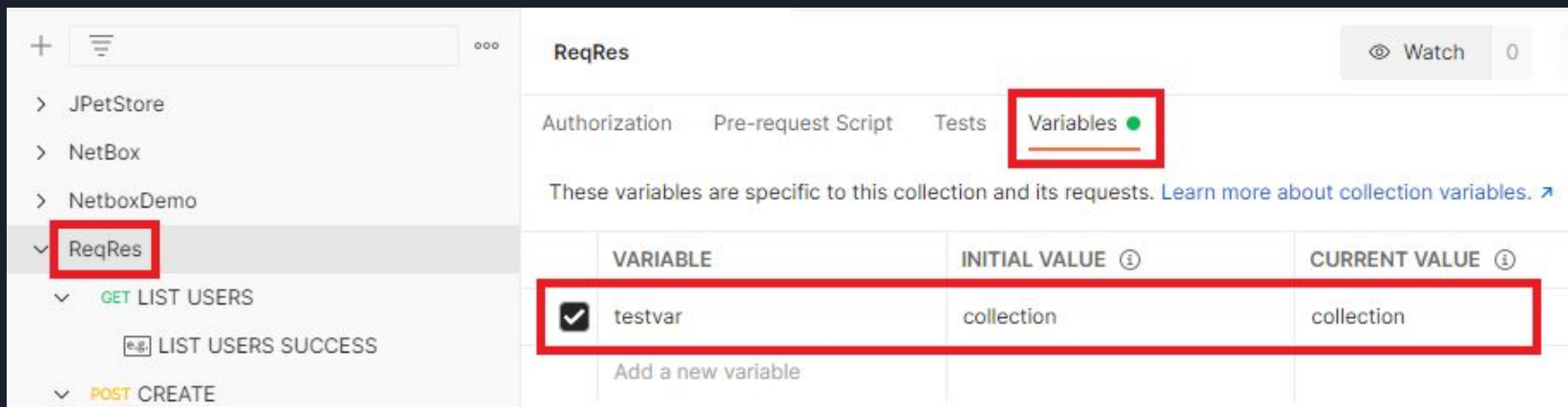
Local Variables

- Local variables have highest precedence and are visible within the request
- We can set local variables using the following command
 - `pm.variables.set("siteid", 20);`



Collection Variables

- Collection variables can be created using the Variable tab available under Collection
- Or using the following command
 - `pm.collectionVariables.set("siteid", 21);`

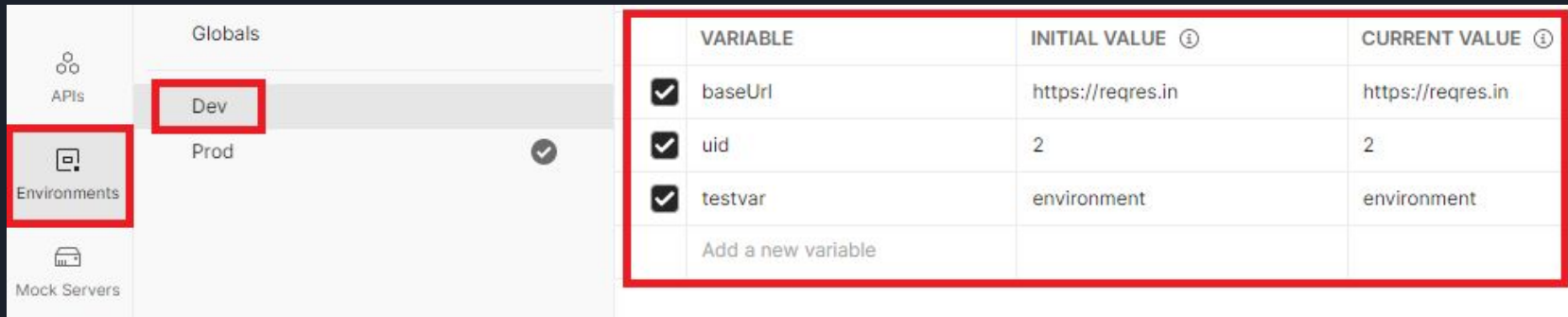


The screenshot shows the Postman interface with the 'ReqRes' collection selected in the left sidebar. The 'Variables' tab is active in the right pane, displaying a table of collection variables. The table has three columns: 'VARIABLE', 'INITIAL VALUE', and 'CURRENT VALUE'. A single variable named 'testvar' is listed, with both its initial and current values set to 'collection'. A red box highlights the 'Variables' tab, and another red box highlights the 'testvar' variable entry.

VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ
<input checked="" type="checkbox"/> testvar	collection	collection
Add a new variable		

Environment Variables

- Environment variable can be created by going to the Environments section and setting them under each environment
- Or using the following command
 - `pm.environment.set("siteid",22);`



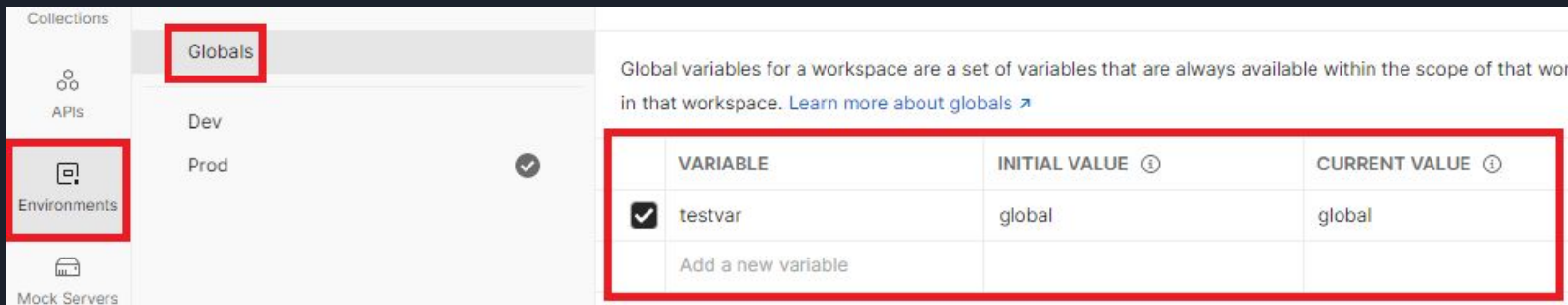
The screenshot shows the PM2 web interface. On the left sidebar, the 'Environments' icon is highlighted with a red box. In the main area, the 'Dev' environment is selected under the 'Globals' tab, also highlighted with a red box. To the right, a table displays the environment variables for the selected environment. The table has three columns: 'VARIABLE', 'INITIAL VALUE', and 'CURRENT VALUE'. It lists three variables: 'baseUrl', 'uid', and 'testvar', each with a checked checkbox in the first column. Below these is a row for 'Add a new variable'.

	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ
<input checked="" type="checkbox"/>	baseUrl	https://reqres.in	https://reqres.in
<input checked="" type="checkbox"/>	uid	2	2
<input checked="" type="checkbox"/>	testvar	environment	environment
	Add a new variable		

Global Variables

- Global variable can be created by going to the Environments section and clicking on Globals section
- Or using the following command

- `pm.globals.set("siteid",23);`



The screenshot shows the Postman interface. On the left sidebar, the 'Environments' icon is highlighted with a red box. In the main panel, the 'Globals' tab is selected and highlighted with a red box. Below the 'Globals' tab, there are two environment names: 'Dev' and 'Prod', with 'Prod' having a checkmark next to it. To the right of the 'Globals' tab, there is a text description: 'Global variables for a workspace are a set of variables that are always available within the scope of that workspace. [Learn more about globals](#)'. Below this text is a table with three columns: 'VARIABLE', 'INITIAL VALUE', and 'CURRENT VALUE'. The table contains one row with a checkmark in the first column, 'testvar' in the second, and 'global' in the third. Below the table is a link 'Add a new variable'.

	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ
<input checked="" type="checkbox"/>	testvar	global	global
	Add a new variable		

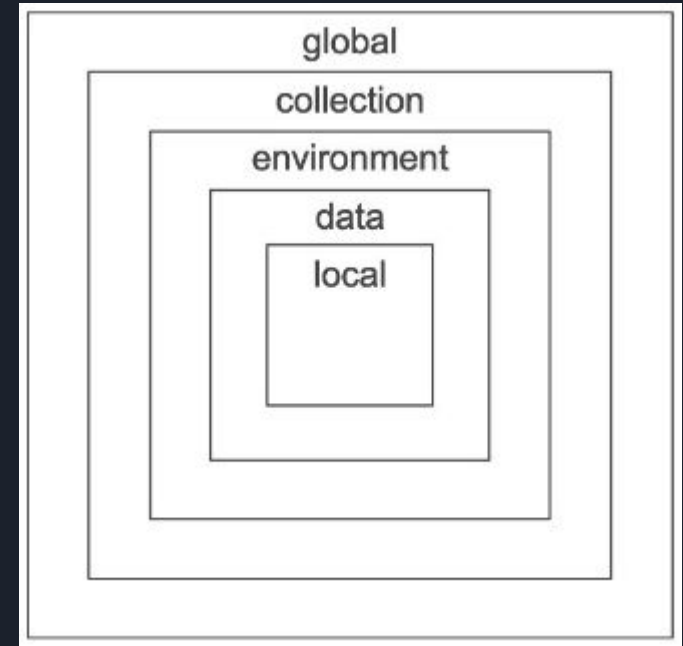


Referencing (using) Variables

- The variables which we have created above can be referenced so that we can use them either in the request URL, body or in pre-request and test scripts
- “siteid” variable can be used outside the scripts (in the url or request body) as:
 - `{{siteid}}`
- It can be used in the scripts as:
 - `pm.variables.get("siteid");`

Referencing (using) Variables (Cont.)

- If the same variable is defined at multiple places like local, collection, environment and global, when it is referenced, the value which is available at the closest scope will be returned
- So in this case the value of “siteid” is returned as 20, since that is the value present in the variable with local scope
- If “siteid” was not present in local scope, its value is returned as 21 from the collection scope and so on...





Environments



Introduction to Environments

- Simply put environment is a collection of variables that can be used in Postman requests
- Can be used to group related sets of values together
- What is the use of Environments?
- Usually the development starts on a Dev env, deployed on staging or pre prod env for testing and finally deployed to production once tested right?
- The URL, username, password, etc. are different from Dev to pre prod to prod env right?
- So in Postman we can create an environment corresponding to each of these environments

Creating Environments

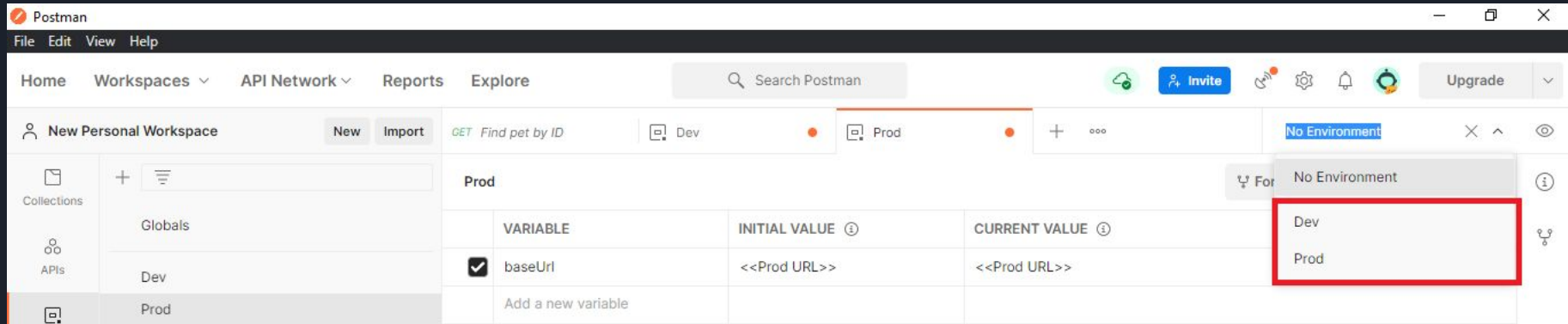
- We can create an environment by going to Environments section on the left side bar

The screenshot shows the 'New Personal Workspace' interface. On the left sidebar, the 'Environments' icon is highlighted with a red box. The 'Environments' section is expanded, showing 'Dev' and 'Prod' environments. The 'Prod' environment is selected and highlighted with a red box. The 'Prod' environment configuration table is shown below.

VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ	⋮	Persist All	Reset All
<input checked="" type="checkbox"/> baseUrl	<<Prod URL>>	<<Prod URL>>			
Add a new variable					

Using Environments

- Once environments are created with relevant variables, while running the requests we can choose an environment against which the request should be run
- This can be done using Environment dropdown at the top right corner



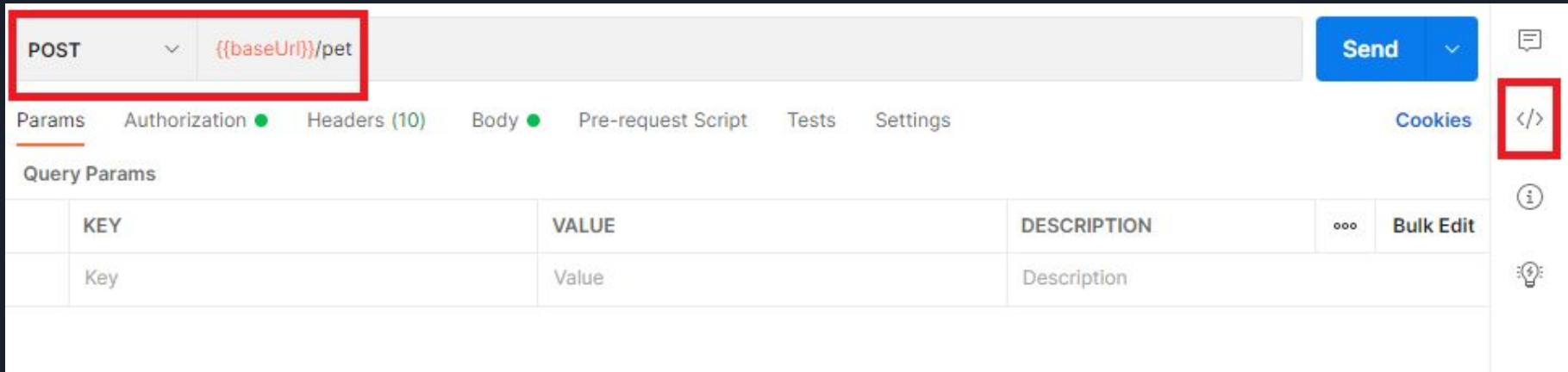


Generating Client Code

- Often times you might want to integrate your API tests as well into your Automation suite
- In such cases it is useful to have the API requests being made from the language of your choice
- While getting started, it is useful to have these pieces of code generated automatically
- Postman does just that with the “Code” feature

Generating Client Code (Cont.)

- Select any request under a collection and click on Code icon on the extreme right of the window, under the Environment quick look (eye icon)
- Code snippet sections gets expanded with a list of languages displayed in a dropdown



The screenshot displays the Postman interface for a POST request. The request method is 'POST' and the URL is '{{baseUrl}}/pet'. The 'Send' button is visible on the right. The 'Code' icon, represented by a </> symbol, is highlighted with a red box on the right sidebar. The 'Query Params' section is expanded, showing a table with columns KEY, VALUE, and DESCRIPTION.

KEY	VALUE	DESCRIPTION
Key	Value	Description

Generating Client Code (Cont.)

POST

{{baseUrl}}/pet

Send

Params

Auth

Headers (10)

Body

Pre-req.


Tests

Settings

Query Params

KEY	VALUE	DESCRIPTION	Bulk Edit
Key	Value	Description	

Response



Click Send to get a response

Type to filter

C# - RestSharp

cURL

Dart - http

Go - Native

HTTP

Java - OkHttp

Java - Unirest

JavaScript - Fetch

JavaScript - jQuery

JavaScript - XHR

C - libcurl

NodeJs - Axios

NodeJs - Native

NodeJs - Request

```
jest POST 'https://io/v2/pet' \
  : application/json'

qui",
upidatat eu
t"

7,
"

57556,
mollit"

29173,
nostrud sed minim
atat"
```

Generating Client Code (Cont.)

GET

{{baseUri}}/pet/findByStatus?status=available&

Send

Params ● Auth ● Headers (9) Body ● Pre-req. Tests Settings ⋮

Query Params

	KEY	VALUE	DESCRIPTION	⋮	Bulk Edit
<input checked="" type="checkbox"/>	status	available			
<input checked="" type="checkbox"/>	status	sold			
	Key	Value	Description		

Body ▼ 🌐 200 OK 1639 ms 106.07 KB [Save Response](#) ▼

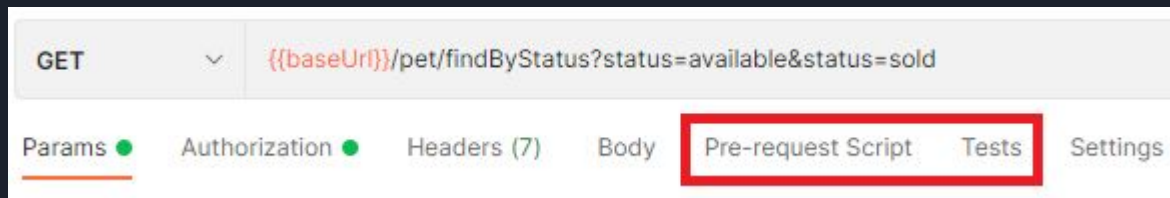
Pretty Raw Preview Visualize JSON ▼ ≡ 📄 🔍

Python - http.client ▼ ⚙️ 📄

```
1 import http.client
2 import json
3
4 conn = http.client.HTTPSConnection
5     ("petstore.swagger.io")
6 payload = json.dumps({
7     "status": "available"
8 })
9 headers = {
10     'Content-Type': 'application/json'
11 }
12 conn.request("GET", "/v2/pet/findByStatus?
13     status=available&status=sold",
14     payload, headers)
15
16 res = conn.getresponse()
17 data = res.read()
18 print(data.decode("utf-8"))
```

Scripting in Postman

- Postman supports scripting as it contains a powerful runtime based on Node.js
- This helps us to add dynamic behavior to requests and collections
- Scripts can be added to Postman before sending a request and after sending a request
- These are denoted as Pre-request Script and Tests in Postman
- Postman script editor has auto complete feature to simplify writing the script by suggesting as we type

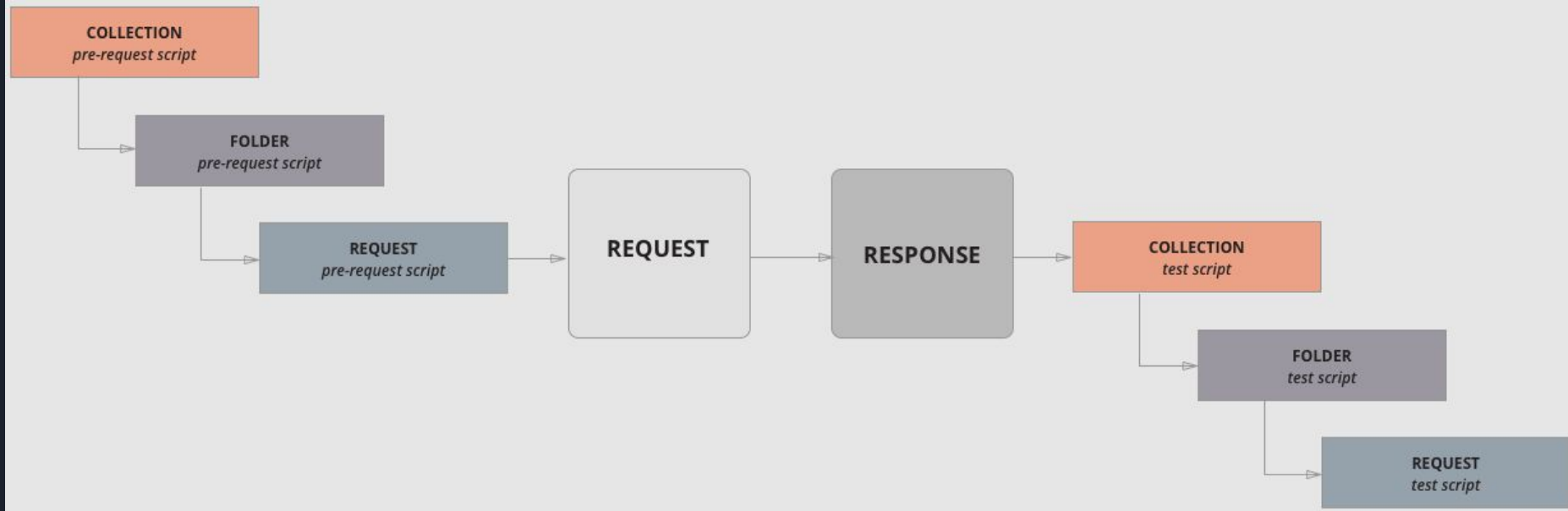




Order of execution

- Pre-request script can be associated with a Collection, a Folder and a Request
- The order of execution in such a case is as follows:
 - Script associated with a collection runs before every request in it
 - Script associated with a folder before every request in it
 - Scripts associated with Requests under a collection / folder
 - Test script associated with a collection runs after every request in it
 - Tests script associated with a folder runs after every request in it

Order of execution (Cont.)





Order of execution (Cont.)


- To verify this add a simple debugging statement using `console.log` as a pre-request script and a test script at Collection → Folder → Request levels
- E.g. In pre-request script at Collection level, we can write a statement like
 - `console.log("pre-request script at collection level");`
- And in test script we can put something like
 - `console.log("test script at collection level");`
- Same way add debug statements for folder and request



Order of execution (Cont.)

- To verify this add a simple debugging statement using `console.log` as a pre-request script and a test script at Collection → Folder → Request levels
- E.g. In pre-request script at Collection level, we can write a statement like
 - `console.log("pre-request script at collection level");`
- And in test script we can put something like
 - `console.log("test script at collection level");`
- Same way add debug statements for folder and request
- These logs can be seen in the console which can be opened using the Console tab at the bottom

Order of execution (Cont.)



```
Find and Replace Console
"pre-request script at collection level"
"pre-request script at folder level"
"pre-request script at request level"
▶ GET https://petstore.swagger.io/v2/pet/1 200 934 ms
"test script at collection level"
"test script at folder level"
"test script at request level"
```

Writing Pre-request Scripts

- Some of the things for which pre-request scripts can be used are as follows:
 - Setting variables before sending the request
 - Establishing correlation between subsequent requests in a collection by processing the data of the previous request and using it in the next request



```
1
2     .... "name": "{{name}}",
3     .... "photoUrls": [
4     ....     "adipisicing qui",
5     ....     "est fugiat cupidatat eu incididunt"
6     .... ],
7     .... "id": 39006160,
8     .... "category": {
```



Writing Tests


- Tests in Postman are similar to pre-request scripts except that they get executed right after the request
- Hence they are named as such, as they can be used to validate the response data and decide whether a request is working as expected or not
- Tests can be used to make variables dynamic, add assertions on response data, pass data in between requests and so on
- Postman gives sample code snippets that can be added and further modified if needed



Writing Tests (Cont.)

- `pm.response` object represents the response received from the server
- Test can be added using the `pm.test` function
- The function `pm.test` accepts a name which signifies the test being carried out and a function that returns a boolean value (true or false)
- This function evaluates a condition and returns true or false based on the condition
- If the function returns true that test is considered as pass, else fail


Writing Tests (Cont.)



POST ▼ {{baseUrl}}/pet

Params Authorization ● Headers (10) Body ● Pre-request Script ● Tests ● Settings


```
1 pm.test("Status test", function () {  
2   pm.response.to.have.status(200);  
3 })
```

Body Cookies Headers (8) Test Results (1/1)  Status: 200 OK

All Passed Skipped Failed

PASS Status test


Writing Tests (Cont.)



POST ▼ `{{baseUrl}}/pet2`

Params Authorization ● Headers (10) Body ● Pre-request Script ● Tests ● Settings

```
1 pm.test("Status test", function () {  
2   pm.response.to.have.status(200);  
3 })
```

Body Cookies Headers (8) Test Results (0/1)  Status: 404 Not Found

All Passed Skipped Failed

FAIL Status test | AssertionError: expected response to have status code 200 but got 404



Writing Tests (Cont.)

- <https://documenter.postman.com/view/1559645/RzZFCGFR?version=latest>
- ABove URL contains a sample workspace which can be imported into our local Postman
- It contains some sample tests which will help us understand how to write different types of tests



Writing Tests (Cont.)

- Expecting response code to have value 200
 - `pm.response.to.have.status(200);`
- Expecting env name to be something
 - `pm.expect(pm.environment.get("env")).to.equal("production");`
- Some other syntax to write tests
 - `pm.response.to.not.be.error;`
 - `pm.response.to.have.jsonBody("");`
 - `pm.response.to.not.have.jsonBody("error");`
 - `pm.response.to.be.ok;`
 - `pm.response.to.be.withBody;`
 - `pm.response.to.be.json;`

Writing Tests (Cont.)

```
pm.test("The response has all properties", () => {  
  //parse the response json and test three properties  
  
  const responseJson = pm.response.json();  
  
  pm.expect(responseJson.type).to.eql('vip');  
  
  pm.expect(responseJson.name).to.be.a('string');  
  
  pm.expect(responseJson.id).to.have.lengthOf(1);  
  
});
```

<https://www.chaijs.com/api/bdd/>



Running Collections



Using Collection Runner

- Collection Runner allows sets of requests to be run in a specified sequence
- Collections can be run against specific environments
- Can be made as data driven, accepting data from CSV and JSON files
- Can be scheduled using monitors
- Can integrate collections with CI/ CD pipeline

Starting Collection run

- Open a collection and click Run button shown at the top
- Or alternately, click Runner at the bottom of the tool

The screenshot shows the Swagger Petstore interface. At the top, there are buttons for Watch, Fork, Run, Save, Share, and a menu icon. The Run button is highlighted with a red box. Below these buttons, there are tabs for Authorization, Pre-request Script, Tests, and Variables. The Variables tab is selected and highlighted with a red box. Below the tabs, there is a text box that says "These variables are specific to this collection and its requests. [Learn more about collection variables.](#)". Below this text box is a table with columns for VARIABLE, INITIAL VALUE, and CURRENT VALUE. The table has two rows: one for baseUrl and one for name. The baseUrl row has a checkbox checked and the initial value is https://petstore.swagger.io/v2. The name row has a checkbox checked and the initial value is empty. Below the table is a button that says "Add a new variable". At the bottom of the interface, there is a footer with a Bootcamp logo, a Runner button (highlighted with a red box), a Trash button, and a help icon.

	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ	...	Persist All	Reset All
<input checked="" type="checkbox"/>	baseUrl	https://petstore.swagger.io/v2	https://petstore.swagger.io/v2			
<input checked="" type="checkbox"/>	name		ucky2			
	Add a new variable					

ⓘ Use variables to reuse values in different places. Work with the current value of a variable to prevent sharing sensitive values with your team. [Learn more about variable values](#)



Starting Collection run (Cont.)

- By default the requests will execute on the order they are shown in the runner
- We can drag them up or down to change the order of execution
- Collection can be run against any specific environment by selecting it from the dropdown
- Other options available are:
 - Running tests in multiple iterations
 - Persist variable values
 - Data driving tests
 - Cookies



Data Driven Testing in Postman

- One of the critical aspects of Automation testing, be it UI or API, is being able to run the automation suit against different sets of data
- Data driven testing solves this problem
- Data driven testing simply means supplying test data from different sources like data or file system
- Postman supports reading the data only from files, specifically 2 formats of files namely json and csv
- Data driven option is only available at the collection level
- So while segregating our tests into collections, we need to keep this also in mind



Data Driven Testing in Postman (Cont.)

- Click on Runner and drag a collection into the runner area
- Check/ uncheck the tests to be run
- Click on Select File and choose the data file from the local folder
- You can use the preview option to ensure that the data file is properly recognized by Postman and data is structured as per your need to be supplied to the test
- Click on the run button below to trigger the execution
- Ensure to add the required tests with appropriate assertions before test execution



Data Driven Testing in Postman (Cont.)

- Any variable which is being used in the API request or pre-request / test scripts can be passed from data file
- We just need to ensure that the variable names match with column names in CSV format or key names in JSON format

Data Driven Testing in Postman (Cont.)

RUN ORDER Deselect All Select All Reset

<input checked="" type="checkbox"/> POST Create Site	Iterations	1
<input checked="" type="checkbox"/> GET Get Sites	Delay	0 ms
<input checked="" type="checkbox"/> GET Get Single Site	Data	Select File
<input checked="" type="checkbox"/> PATCH Partial Update Sites	<input type="checkbox"/> Save responses ⓘ	
<input checked="" type="checkbox"/> PUT Update Sites	<input checked="" type="checkbox"/> Keep variable values ⓘ	
<input checked="" type="checkbox"/> DEL Delete Sites	<input type="checkbox"/> Run collection without using stored cookies	
	<input checked="" type="checkbox"/> Save cookies after collection run ⓘ	
	Run NetBoxDev	

Data Driven Testing in Postman (Cont.)

The screenshot displays the 'RUN ORDER' configuration panel in Postman. On the left, a list of API tests is shown, each with a checked checkbox and a color-coded label: POST (orange), GET (green), PATCH (orange), PUT (blue), and DEL (red). On the right, the 'Iterations' field is set to 2. The 'Delay' field is set to 0 ms. The 'Data' field shows a file named 'testdata.csv' selected. The 'Data File Type' is set to 'text/csv', and the 'Preview' button is highlighted. Below these fields, there are four checkboxes: 'Save responses' (unchecked), 'Keep variable values' (checked), 'Run collection without using stored cookies' (unchecked), and 'Save cookies after collection run' (checked). At the bottom right, there is a blue button labeled 'Run NetBoxDev'.

RUN ORDER Deselect All Select All Reset

- ☒ **POST** Create Site
- ☒ **GET** Get Sites
- ☒ **GET** Get Single Site
- ☒ **PATCH** Partial Update Sites
- ☒ **PUT** Update Sites
- ☒ **DEL** Delete Sites

Iterations 2

Delay 0 ms

Data Select File testdata.csv X

Data File Type text/csv Preview

☐ Save responses ⓘ

☒ Keep variable values ⓘ

☐ Run collection without using stored cookies

☒ Save cookies after collection run ⓘ

Run NetBoxDev

Data Driven Testing in Postman (Cont.)

```
testdata.csv - Notepad
File Edit Format View Help
name,slug
test site 161020211150,test-site-161020211150
test site 161020211150,test-site-161020211150
```

PREVIEW DATA

Iteration	name	slug
1	"test site 161020211150"	"test-site-161020211150"
2	"test site 161020211150"	"test-site-161020211150"

file testdata.csv X

Preview

POST `{{baseUrl}}/dcim/sites/`

Params Authorization Headers (10)

☐ none ☒ form-data ☐ x-www-form-urlencoded

```
1
2     "name": "{{name}}",
3     "slug": "{{slug}}",
4     "status": "planned"
5
```



Postman Monitors

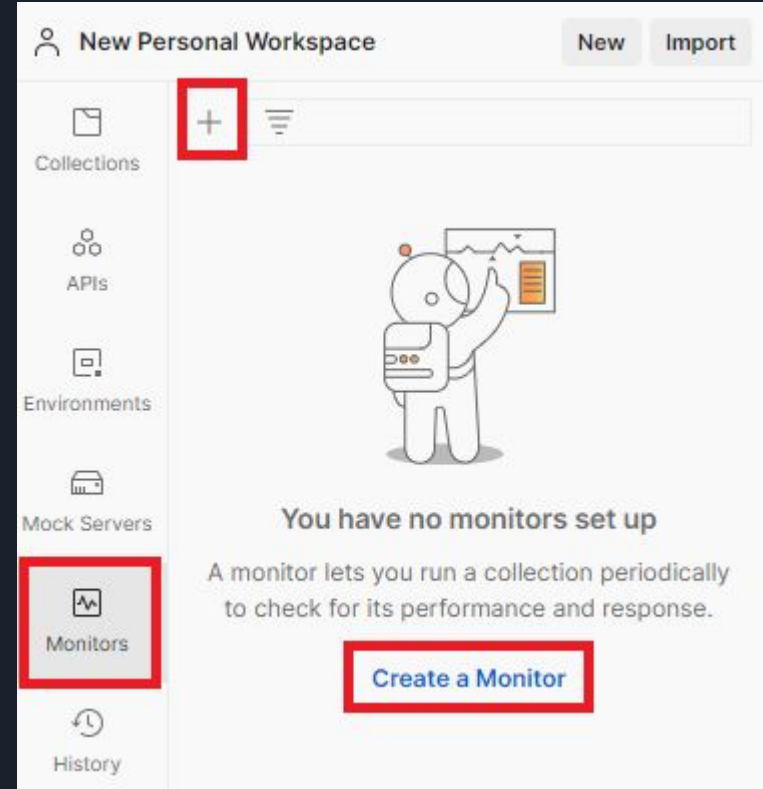


Postman Monitors

- In the previous section we have seen how to group requests into collections and run them together
- However this execution still needs to be triggered manually
- Collection run can be automated using Postman Monitors
- Monitors let us schedule automated collection runs and receive reports

Postman Monitors (Cont.)

- On the left navigation bar, click on Monitors and click the + icon or Create Monitor link to create a new monitor
- Enter a name, choose a collection to run and enter other details to create a monitor



Postman Monitors (Cont.)

Edit monitor

Monitor name

Collection

Collection tag

▼

Choose a collection tag for this monitor. If the collection is not linked to an API, a monitor can only be created on the current tag.

Environment

▼

The variable values in this environment will be used in all your monitoring calls. [Learn more](#)

Run this monitor

▼

▼

Frequent runs give you a finer view of performance but also cost more monitoring calls which might be limited by your Postman account. Check your [usage limits](#)

Regions

☒ Automatically select region

☐ Manually select region

☒ Receive email notifications for run failures and errors

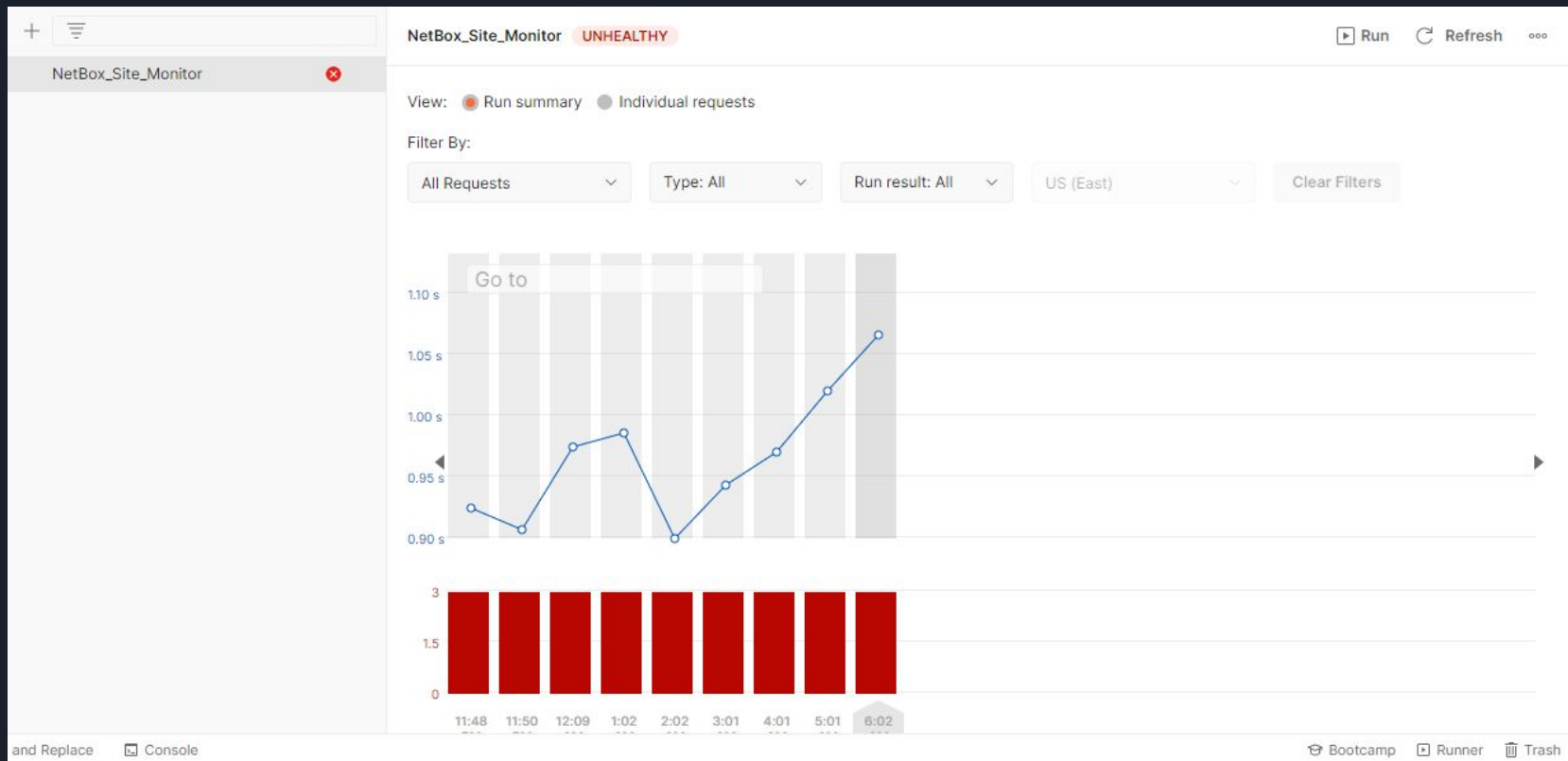
You can select one or more regions to monitor your requests from. [Learn more](#)

Bootcamp

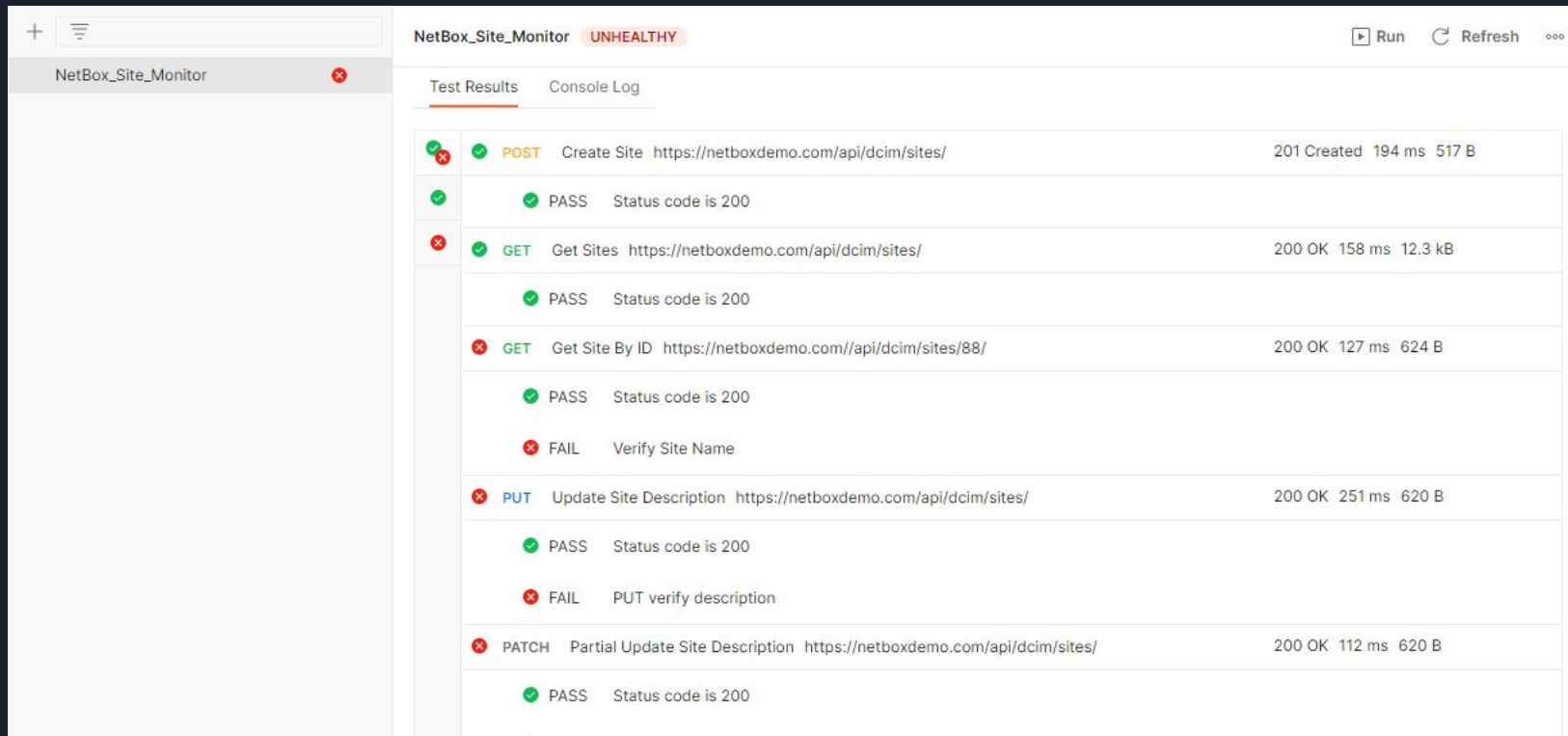
Runner

Trash

Postman Monitors (Cont.)



Postman Monitors (Cont.)



The screenshot displays the Postman Monitors interface. On the left, a sidebar shows a monitor named 'NetBox_Site_Monitor' with a red status icon. The main panel shows the monitor's details, including its name 'NetBox_Site_Monitor' and a red 'UNHEALTHY' status badge. The 'Test Results' tab is active, showing a list of test results for the monitor. The results are as follows:

Method	Test Name	URL	Status	Response	Time	Size
POST	Create Site	https://netboxdemo.com/api/dcim/sites/	201 Created	194 ms	517 B	
PASS	Status code is 200					
GET	Get Sites	https://netboxdemo.com/api/dcim/sites/	200 OK	158 ms	12.3 kB	
PASS	Status code is 200					
GET	Get Site By ID	https://netboxdemo.com/api/dcim/sites/88/	200 OK	127 ms	624 B	
PASS	Status code is 200					
FAIL	Verify Site Name					
PUT	Update Site Description	https://netboxdemo.com/api/dcim/sites/	200 OK	251 ms	620 B	
PASS	Status code is 200					
FAIL	PUT verify description					
PATCH	Partial Update Site Description	https://netboxdemo.com/api/dcim/sites/	200 OK	112 ms	620 B	
PASS	Status code is 200					

Postman Monitors (Cont.)

NetBox_Site_Monitor UNHEALTHY

Run Refresh

Test Results Console Log

- 0:01:48 Preparing run.
- 0:03:47 Preparing run.
- 0:04:42 Preparing run.
- 0:05:42 Preparing run.
- 0:06:38 Preparing run.
- 0:07:37 Preparing run.
- 0:08:33 Preparing run.
- 0:09:31 Preparing run.
- 0:09:47 Running... [Hide run logs](#)

1	0:09:47	NetBox_Site_Monitor started	
2	0:09:47	Create Site	
3	0:09:47	POST https://netboxdemo.com/api/dcim/sites/	
4	0:09:49	Get Sites	
5	0:09:49	GET https://netboxdemo.com/api/dcim/sites/	
6	0:09:50	[
7	0:09:50	1,	2,
8	0:09:50	7,	8,
9	0:09:50	37,	54,



Postman Workflows

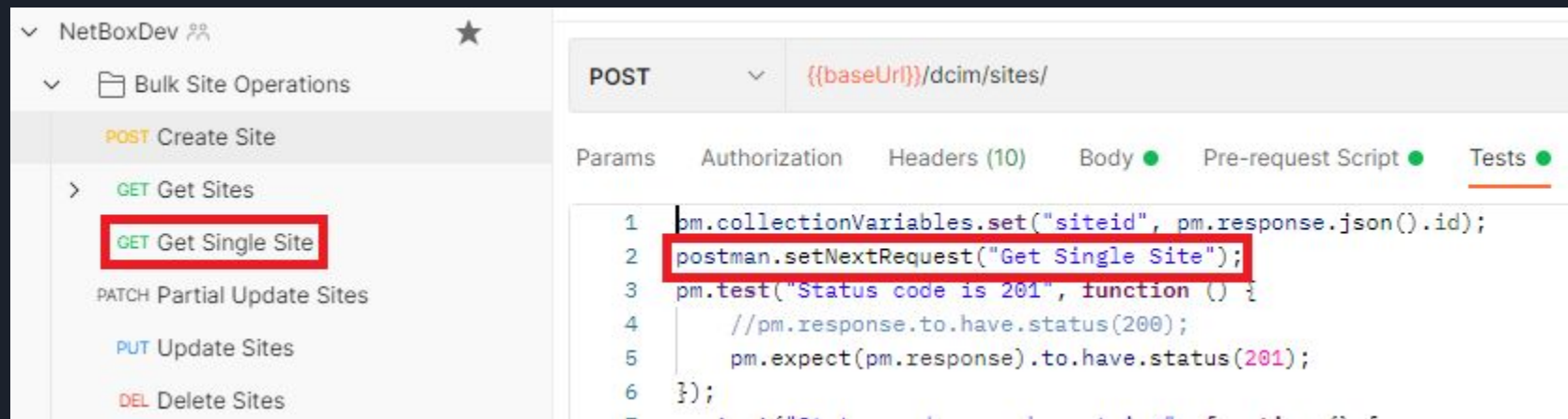


Introduction to Workflows

- Collection Runner executes the requests in the order in which they are seen
- We can drag and drop to change the order of execution, but that is not recommended for automating collection run via Monitors or command line execution
- We can change the default order of execution using the concept of Workflows

Building a request workflow

- Postman provides a function `postman.setNextRequest()`
- This can be used to control which request will be called next



The screenshot displays the Postman interface. On the left sidebar, under the 'NetBoxDev' workspace, the 'Bulk Site Operations' folder is expanded. It contains several requests: 'POST Create Site', 'GET Get Sites', 'GET Get Single Site' (highlighted with a red box), 'PATCH Partial Update Sites', 'PUT Update Sites', and 'DEL Delete Sites'. The main panel shows a 'POST' request to the endpoint `{{baseUrl}}/dcim/sites/`. The 'Tests' tab is active, showing a JavaScript test script. The first two lines of the script are highlighted with a red box:

```
1 pm.collectionVariables.set("siteid", pm.response.json().id);  
2 postman.setNextRequest("Get Single Site");  
3 pm.test("Status code is 201", function () {  
4     //pm.response.to.have.status(200);  
5     pm.expect(pm.response).to.have.status(201);  
6 });
```



Dynamic Variables



Working with dynamic variables

- There would be times when you need to test an API with large amount of data or some dummy data
- If we do not have access to such data, we can make use of Dynamic Variables in Postman
- Postman uses fake.js a Node JS library to generate fake data for different fields like names, titles, email ids, addresses etc.
- Dynamic variable names start with \$ sign
- They can be used in pre-request or test scripts
- E.g. to generate a random name we use a command
`pm.variables.replaceIn('{{randomFirstName}}')`
- Refer [Dynamic variables | Postman Learning Center](#) for further details



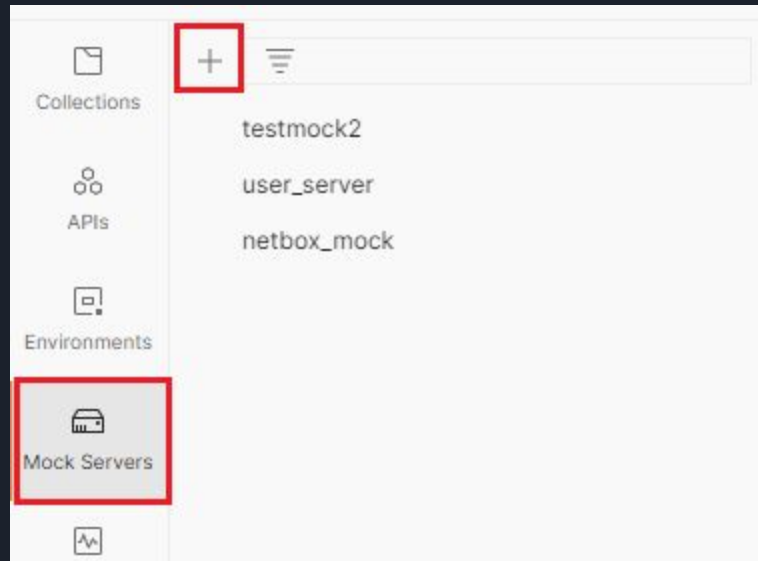
Mock Servers



Working with Mock Requests

- Service mocking is an important feature during API development or testing
- If a dependent API is not ready, we can create a mock API that simulates the behavior of the actual API
- This mock API can then be used to continue further development or test automation
- We can create a mock server and add mock requests under it with the required endpoint url, request and response formats response code etc.

Creating Mock Server (Cont.)



Creating Mock Server (Cont.)

- Choose the method, url, response code and response body that we need
- For GET requests we do not have body

Create a mock server

1. Select collection to mock 2. Configuration

Create a new collection

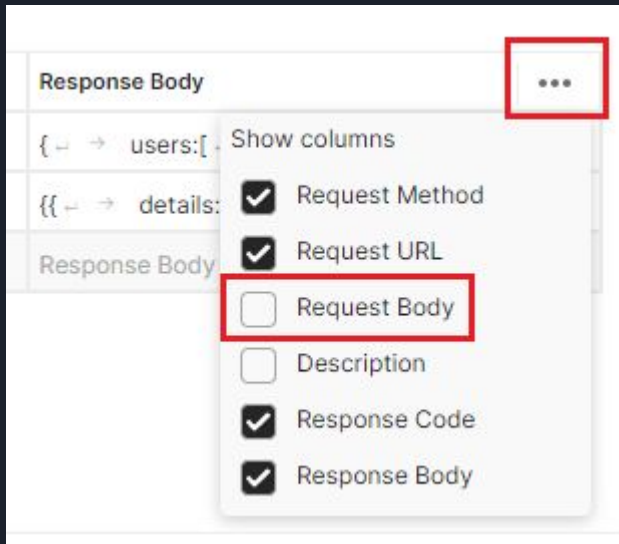
Select an existing collection

Enter the requests you want to mock. Optionally, add a request body by clicking on the (...) icon.

Request Method		Request URL	Response Code	Response Body	...
GET	▼	/users	200	{ → users:[→ → { → → → "id...	
GET	▼	/users	404	{{ → details: "user not found" → }	
GET	▼	URL	200	Response Body	

Creating Mock Server (Cont.)

- But for POST we need to send body
- By default request body is not enabled
- But we can enable it as shown below



Creating Mock Server (Cont.)

Create a mock server

1. Select collection to mock 2. Configuration

Create a new collection

Select an existing collection

Enter the requests you want to mock. Optionally, add a request body by clicking on the (...) icon.

Request Method	Request URL	Request Body	Response Code	Response Body	...
GET	/users	Request Body	200	{ → users:[→ { → → → ...	
GET	/users	Request Body	404	{ → details: "user not found" → }	
POST	/users	{ → users:[...	201	{ → users:[→ { → → → ...	
GET	URL	Request Body	200	Response Body	

Cancel

Next

Creating Mock Server (Cont.)

- Once done with method selection, click Next
- In the Configuration tab, enter name, choose an environment (optional) and click Create Mock Server

Create a mock server

☒ Select collection to mock

2. Configuration

Mock server name
my_server_mock

Environment
No Environment

☒ Save the mock server URL as an environment variable

☐ Make mock server private

☐ Simulate fixed network delay

An environment is a group of variables useful for storing and reusing values. [Learn more](#)

Note: This will create a new environment containing the URL.

Note: To call a private mock server, you'll need to add an `x-api-key` header to your requests. See how to generate a [Postman API key](#)

Back

Create Mock Server



Using Mock Server

- Go to collections to find a collection with the same name as the mock server
- Expand it to find the mock requests we created
- Note that an environment is also created with the mock server name and URL has been converted to a variable
- Send a request and verify whether the behavior is as expected

Using Mock Server (Cont.)

The screenshot shows the APITesting_Wrkspc interface with a mock server named **my_server_mock**. The endpoint is **GET {{url}}/users**. The response is a **404 Not Found** status with a JSON body: `{ details: "user not found" }`.

Left Sidebar (Collections):

- NetBoxDev
 - Bulk Site Operations
 - POST Create Site
 - GET Get Sites
 - GET Get Single Site
 - PATCH Partial Update Sites
 - PUT Update Sites
 - DEL Delete Sites
 - DNAC_Sandbox
 - Intro to writing tests - with exam...
 - my_server_mock**
 - GET /users
 - GET /users
 - Default

Main Panel (my_server_mock / /users):

- Method: **GET**
- URL: **{{url}}/users**
- Buttons: **Send**, **Save**, **Settings**
- Tabs: **Params**, **Authorization**, **Headers (7)**, **Body**, **Pre-request Script**, **Tests**, **Settings**
- Query Params** table:

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Tab:

- Status: **404 Not Found** (Time: 1613 ms, Size: 478 B)
- Save Response
- Format: **HTML**
- Response Body (Pretty):

```
1 {
2   details: "user not found"
3 }
```



CLI Execution Via Newman



Running API tests from command prompt

- While Postman Monitors help us run a collection in an automated way, this still requires Postman UI
- To be able to integration REST API testing with other automation frameworks or with CI/CD pipeline, we need to be able to trigger the tests from a command line
- This can be achieved using a Node JS library called Newman



Install Newman

- Newman is a command-line collection runner for Postman
- Newman maintains feature parity with Postman, meaning the collection run from Newman is exactly the same as it is from Postman
- Since Newman is a Node JS library, it requires Node JS for running
- Newman is available from NPM registry as well as Github
- Download and install Node JS from <https://nodejs.org/en/download/>
- Install newman using the following command
 - `npm install -g newman`



Running collection with Newman

- There are 2 ways to pass the collection to newman to run from command prompt
 - By exporting collection as json file
 - `newman run mycollection.json`
 - By sharing the collection and getting an online link
 - `newman run`
`https://www.postman.com/collections/cb208e7e64056f5294e5`
- Refer the below link for further information
 - [Running collections on the command line with Newman | Postman Learning Center](#)



Running collection with Newman (Cont.)

- If the collection uses an environment to run we need to export the environment also as a json file and pass it as follows
 - `newman run mycollection.json -e myenvironment.json`
- If the collection run is also data driven, meaning it requires data to be passed from a file (csv or json), then we need to pass the data file also from the command prompt as follows
 - `newman run mycollection.json -e myenvironment.json -d data_file.json`
- You might have to specify the complete path of the files if needed



Postman - Jenkins Integration via Newman



Introduction to Jenkins Integration

- Postman has a Node JS based fully featured testing sandbox using which we can execute JS based tests on our APIs
- Using Node JS library called Newman we can run the same tests via a CLI (Command Line Interface)
- This is our first step towards achieving integration with any of the CI/CD systems
- Once we are able to execute our tests from a command prompt or terminal, we can use the “Execute Shell” or “Execute Windows batch command” option of Jenkins to trigger our tests from Jenkins pipeline



Prerequisites

- We need the following to be installed on the build server master node or slave nodes
 - Node JS
 - NPM
 - Newman

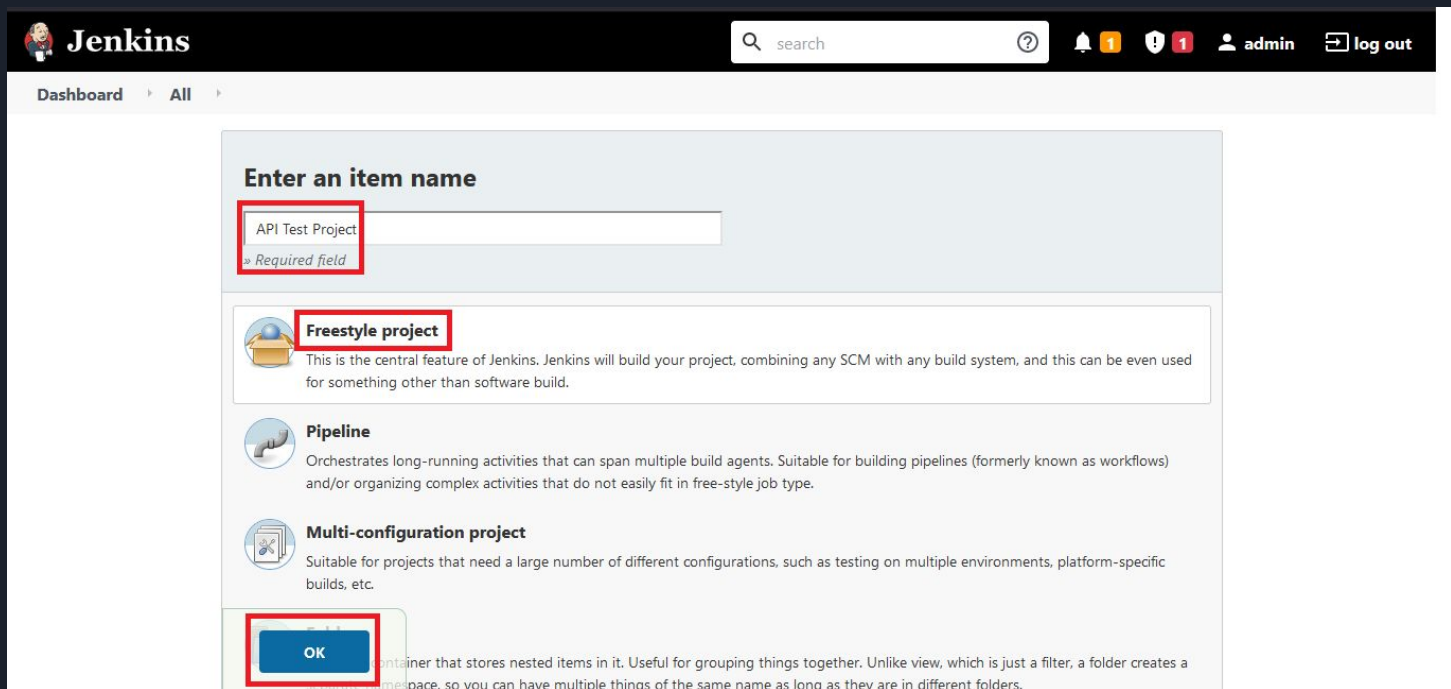
Running the collection from CLI

- Export Postman collection as a JSON link or file
- Use the following command to run the collection from command line
 - Newman run <<collection json link or json file>>
- Ensure that collection runs as expected from command line
- Newman produces the output shown below in the CLI

	executed	failed
iterations	1	0
requests	6	0
test-scripts	12	0
prerequisite-scripts	12	0
assertions	9	0
total run duration: 3.8s		
total data received: 5.33kB (approx)		
average response time: 537ms [min: 299ms, max: 1359ms, s.d.: 384ms]		

Creating project in Jenkins

- Login to Jenkins and create a freestyle project



The screenshot shows the Jenkins 'Create new item' dialog box. The 'Enter an item name' field is highlighted with a red box and contains the text 'API Test Project'. Below this, the 'Freestyle project' option is selected and highlighted with a red box. The 'OK' button at the bottom is also highlighted with a red box.

Jenkins search ? 1 1 admin log out

Dashboard ▸ All ▸

Enter an item name

API Test Project
* Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

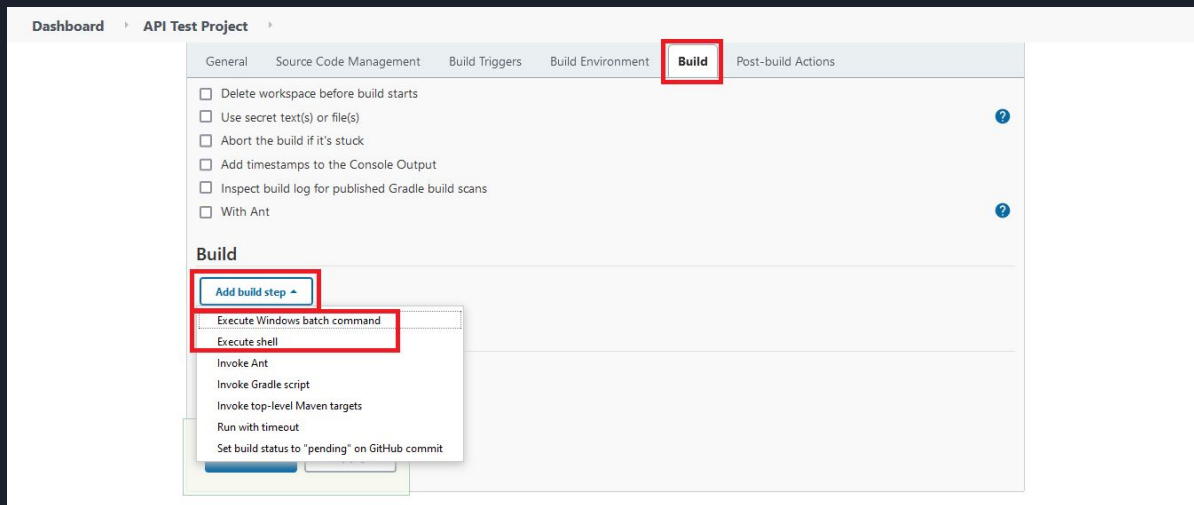
Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
A container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

OK

Creating project in Jenkins (Cont.)

- In the project go to Build tab and add a build step by choosing one of the following options based on your OS
 - Execute shell
 - Execute Windows batch command



Creating project in Jenkins (Cont.)

- Enter the command we used in the CLI, in the command window of the build step
- Ensure to add --bail option at the end to tell Newman to halt test execution on failure with status code 1
- This will be used by Jenkins to mark the build as fail

Build

Execute Windows batch command

Command

```
newman run NetBox_API_Tests.postman_collection.json -e NetBox_Private_Cloud.postman_environment.json -d netbox_data.csv --bail
```

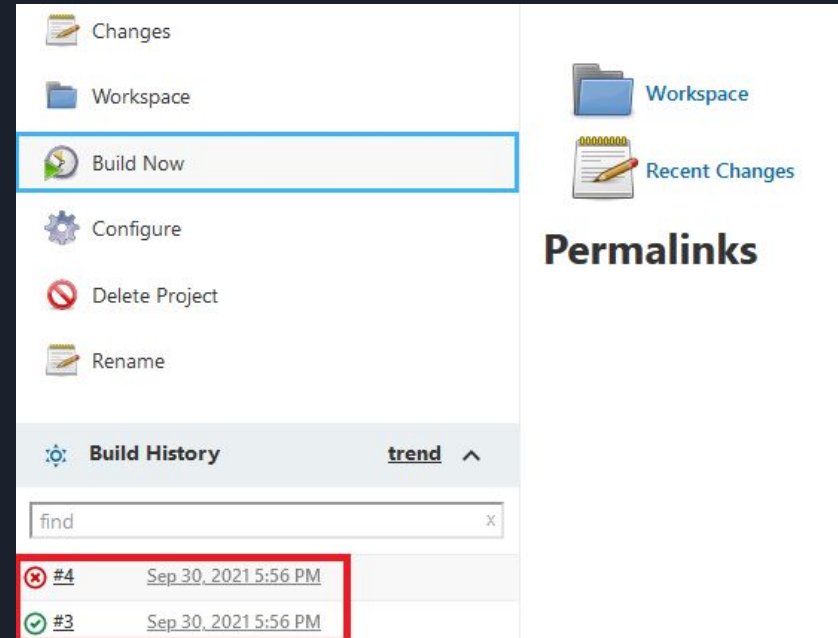
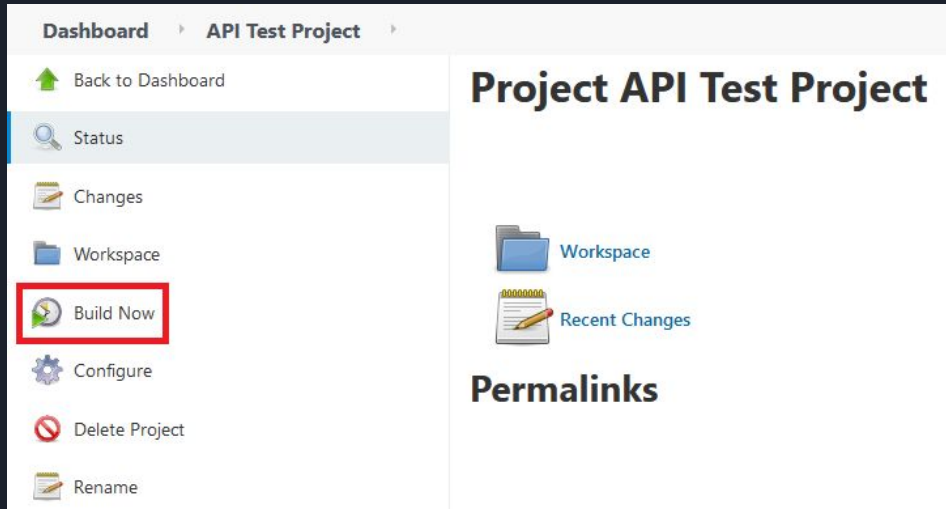
[See the list of available environment variables](#)

Advanced...

Save Apply

Test the build in Jenkins

- Trigger the build by clicking Build Now
- One successful and one failed build are shown below



Check the failed build details

- Click on any failed build and check the console output to see the reason for failure

Dashboard > API Test Project > #4

Back to Project

Status

Changes

Console Output

View as plain text

Console Output

Started by user [admin](#)
Running as SYSTEM
Building in workspace C:\Users\jenkins\AppData\Local\Jenkins\.jenkins\workspace\API Test Project
[API Test Project] \$ cmd /c call C:\Users\jenkins\AppData\Local\Temp\jenkins1107080679764670229.bat

C:\Users\jenkins\AppData\Local\Jenkins\.jenkins\workspace\API Test Project>C:\Users\nravi\AppData\Roaming\npm\npm.cmd run C:\Users\nravi\Downloads\NetBox_API_Tests.postman_collection.json -e C:\Users\nravi\Downloads\NetBox_Private_Cloud.postman_environment.json -d C:\Users\nravi\Downloads\netbox_data.csv --bail

```
# failure      detail

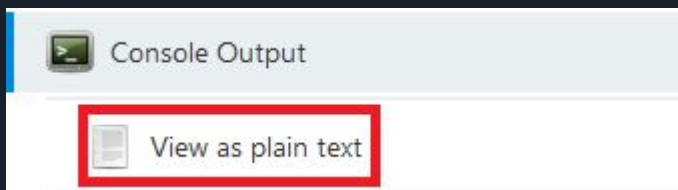
1. AssertionError  Status code is 200
                    expected 400 to be one of [ 200, 201, 204 ]
                    at assertion:0 in test-script
                    inside "Sites / Create Site"

2. AssertionFailure  Status code is 200
                    at test-script:38:17
                    inside "Sites / Create Site"

Build step 'Execute Windows batch command' marked build as failure
Finished: FAILURE
```

Check the failed build details (Cont.)

- Click “View as plain text” to see formatted output



	executed	failed
iterations	1	0
requests	2	0
test-scripts	3	1
prerequisite-scripts	4	0
assertions	2	1
total run duration: 1121ms		
total data received: 3.1kB (approx)		
average response time: 448ms [min: 270ms, max: 626ms, s.d.: 178ms]		

#	failure	detail
1.	AssertionError	Status code is 200 expected 400 to be one of [200, 201, 204] at assertion:0 in test-script inside "Sites / Create Site"
2.	AssertionFailure	Status code is 200 at test-script:38:17 inside "Sites / Create Site"

Build step 'Execute Windows batch command' marked build as failure
Finished: FAILURE



That's all folks!!!