

# CCNA Day 61

## REST APIs

### 6.0 Automation and Programmability

10%



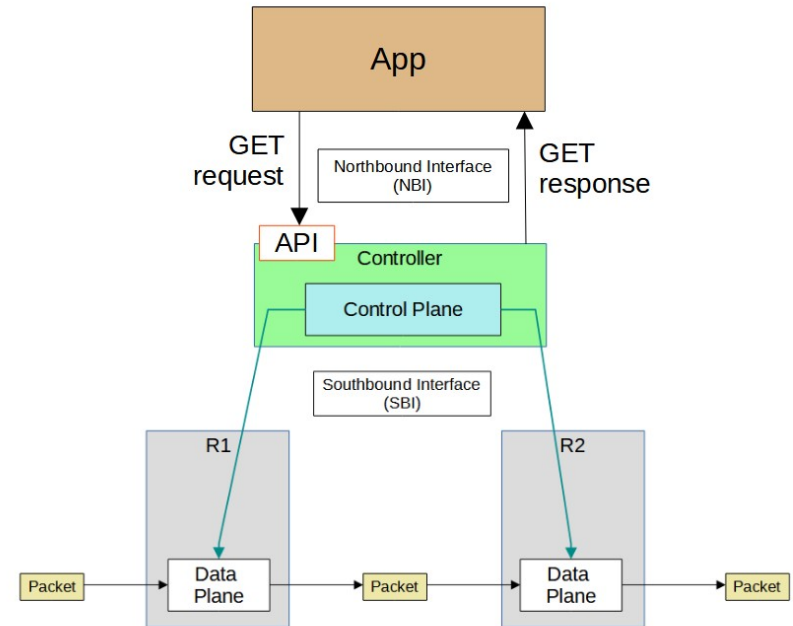
- 6.1 Explain how automation impacts network management
- 6.2 Compare traditional networks with controller-based networking
- 6.3 Describe controller-based and software defined architectures (overlay, underlay, and fabric)
  - 6.3.a Separation of control plane and data plane
  - 6.3.b North-bound and south-bound APIs
- 6.4 Compare traditional campus device management with Cisco DNA Center enabled device management
- 6.5 Describe characteristics of REST-based APIs (CRUD, HTTP verbs, and data encoding)
- 6.6 Recognize the capabilities of configuration management mechanisms Puppet, Chef, and Ansible
- 6.7 Interpret JSON encoded data



# Things we'll cover

- API Review
- CRUD operations and HTTP verbs
- REST APIs
- REST API Calls using Cisco DevNet

- An API (Application Programming Interface) is a software interface that allows two applications to communicate with each other.
- APIs are essential not just for network automation, but for all kinds of applications.
- In SDN architecture, APIs are used to communicate between apps and the SDN controller (via the NBI), and between the SDN controller and the network devices (via the SBI).
- The NBI typically uses REST APIs.
- NETCONF and RESTCONF are popular southbound APIs.



- CRUD (Create, Read, Update, Delete) refers to the operations we perform using REST APIs.
- **Create** operations are used to create new variables and set their initial values.  
→ ie. create variable “ip\_address” and set the value to “10.1.1.1”.
- **Read** operations are used to retrieve the value of a variable.  
→ ie. what is the value of variable “ip\_address”?
- **Update** operations are used to change the value of a variable.  
→ ie. change the value of variable “ip\_address” to “10.2.3.4”.
- **Delete** operations are used to delete variables.  
→ ie. delete variable “ip\_address”.
- HTTP uses *verbs* (aka. *methods*) that map to these CRUD operations.
- REST APIs typically use HTTP.

# HTTP Verbs

Purpose	CRUD Operation	HTTP Verb
Create new variable	<b>Create</b>	<b>POST</b>
Retrieve value of variable	<b>Read</b>	<b>GET</b>
Change the value of variable	<b>Update</b>	<b>PUT, PATCH</b>
Delete variable	<b>Delete</b>	<b>DELETE</b>

You can see the different HTTP verbs here: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>

# HTTP Request

- When an HTTP client sends a request to an HTTP server, the HTTP header includes information like this:
  - An HTTP Verb (ie. GET)
  - A URI (Uniform Resource Identifier), indicating the resource it is trying to access.



- Here's an example of a URI (which we will use in the demonstration later):

<https://sandboxdnac.cisco.com/dna/intent/api/v1/network-device>

Below the URI, the components are labeled: "scheme" (under https), "authority" (under //sandboxdnac.cisco.com), and "path" (under /dna/intent/api/v1/network-device).

# HTTP Request

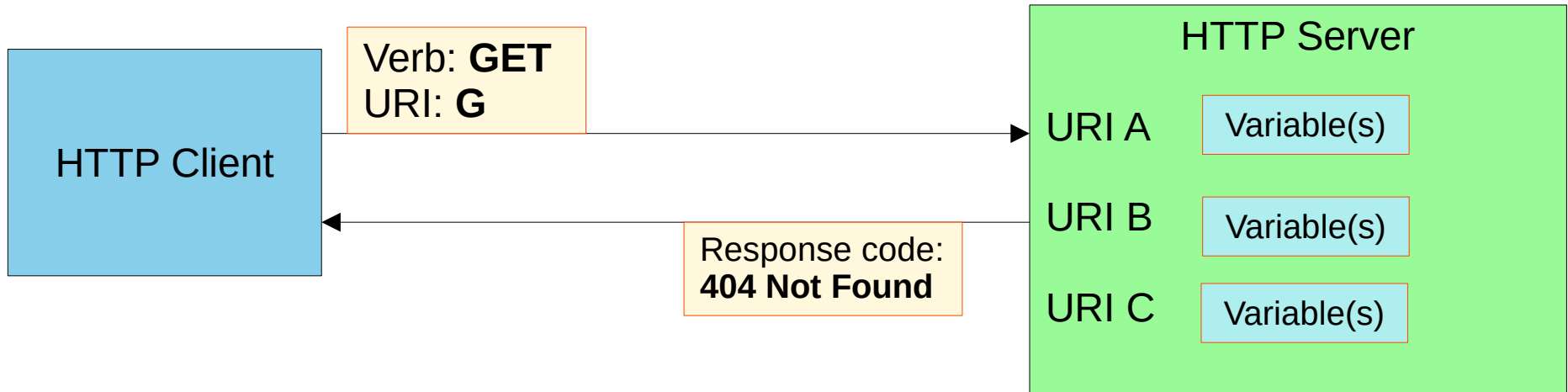
- The HTTP request can include additional headers which pass additional information to the server.  
→ Check the list at <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>

IP Header	TCP Header	Verb	URI	Additional Headers	Data
--------------	---------------	------	-----	--------------------	------

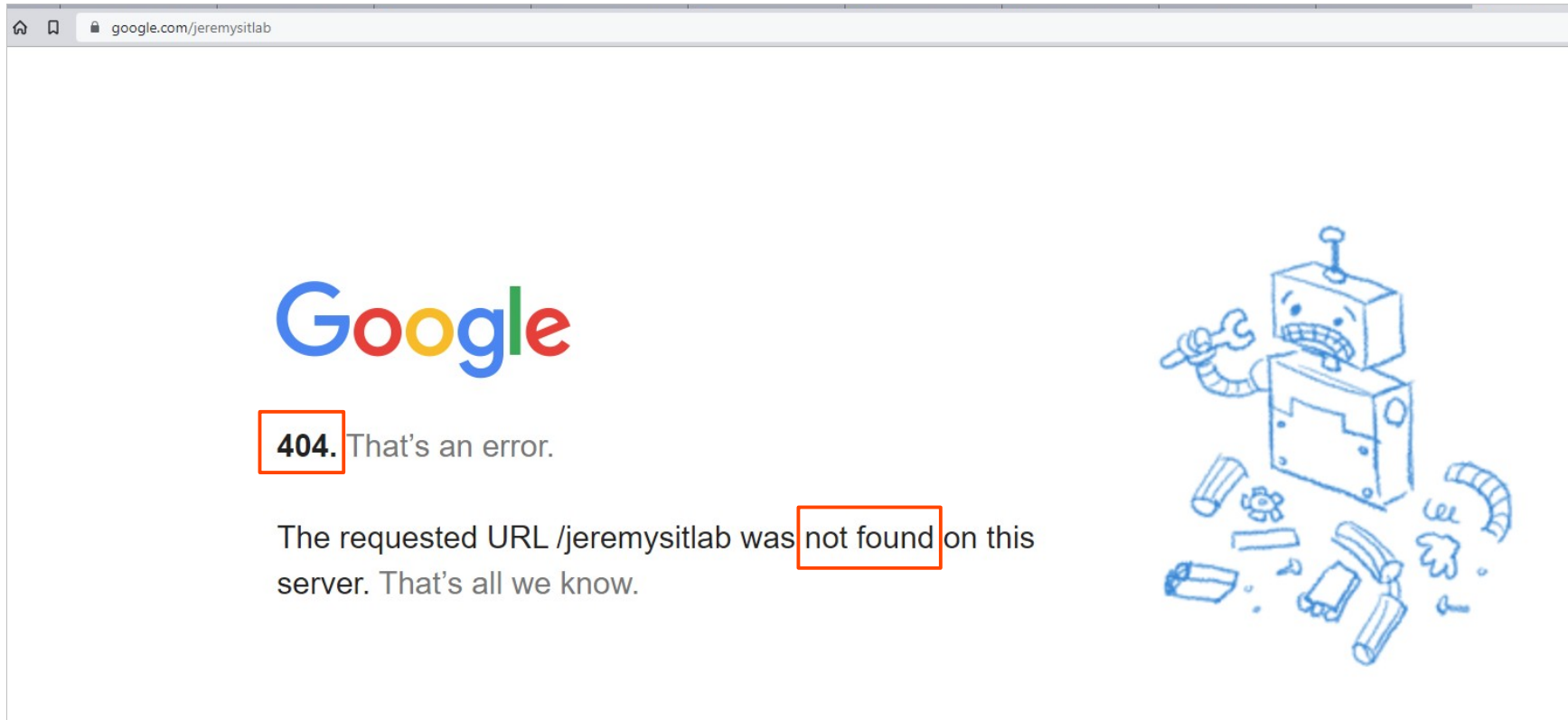
- An example would be an **Accept** header, which informs the server about the type(s) of data that can be sent back to the client.  
→ ie. **Accept: application/json** or **Accept: application/xml**
- You can also view standard HTTP header fields with some examples at [https://en.wikipedia.org/wiki/List\\_of\\_HTTP\\_header\\_fields](https://en.wikipedia.org/wiki/List_of_HTTP_header_fields)
- When a REST client makes an API call (request) to a REST server, it will send an HTTP request like the one above.  
\*REST APIs don't *have* to use HTTP for communication, although HTTP is the most common choice.

# HTTP Response

- The server's response will include a status code indicating if the request succeeded or failed, as well as other details.
- The first digit indicates the class of the response:
  - **1xx** *informational* – the request was received, continuing process
  - **2xx** *successful* – the request was successfully received, understood, and accepted
  - **3xx** *redirection* – further action needs to be taken in order to complete the request
  - **4xx** *client error* – the request contains bad syntax or cannot be fulfilled

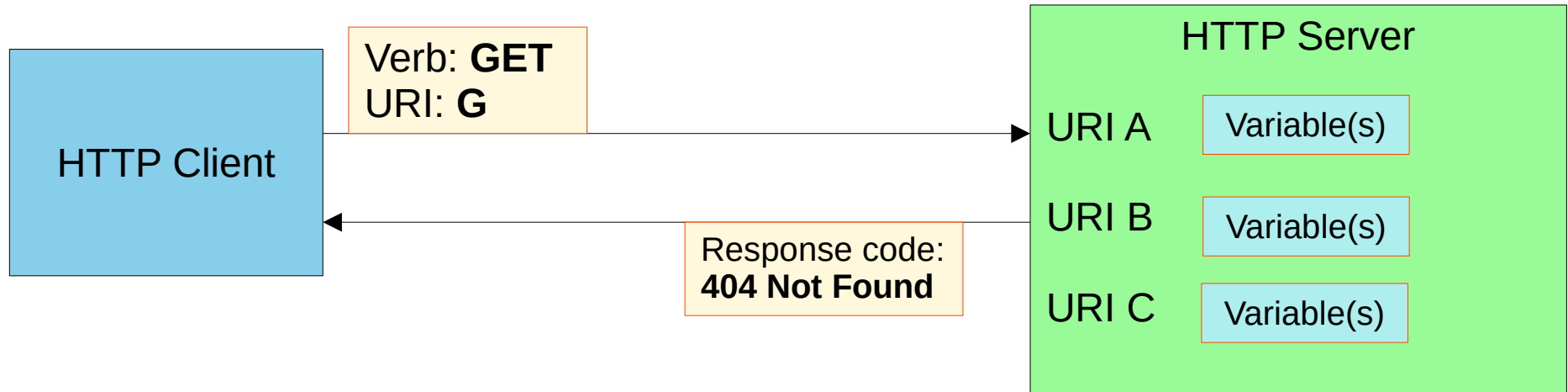






# HTTP Response

- The server's response will include a status code indicating if the request succeeded or failed, as well as other details.
- The first digit indicates the class of the response:
  - **1xx** *informational* – the request was received, continuing process
  - **2xx** *successful* – the request was successfully received, understood, and accepted
  - **3xx** *redirection* – further action needs to be taken in order to complete the request
  - **4xx** *client error* – the request contains bad syntax or cannot be fulfilled
  - **5xx** *server error* – the server failed to fulfill an apparently valid request



# HTTP Response

Here are some examples of each HTTP Response class:

- **1xx Informational**
  - **102 Processing** indicates that the server has received the request and is processing it, but the response is not yet available.
- **2xx Successful**
  - **200 OK** indicates that the request succeeded.
  - **201 Created** indicates that the request succeeded and a new resource was created (ie. in response to POST)
- **3xx Redirection**
  - **301 Moved Permanently** indicates that the requested resource has been moved, and the server indicates its new location.
- **4xx Client Error**
  - **401 Unauthorized** means the client must authenticate to get a response.
  - **404 Not Found** means the requested resource was not found.
- **5xx Server Error**
  - **500 Internal Server Error** means the server encountered something unexpected that it doesn't know how to handle.

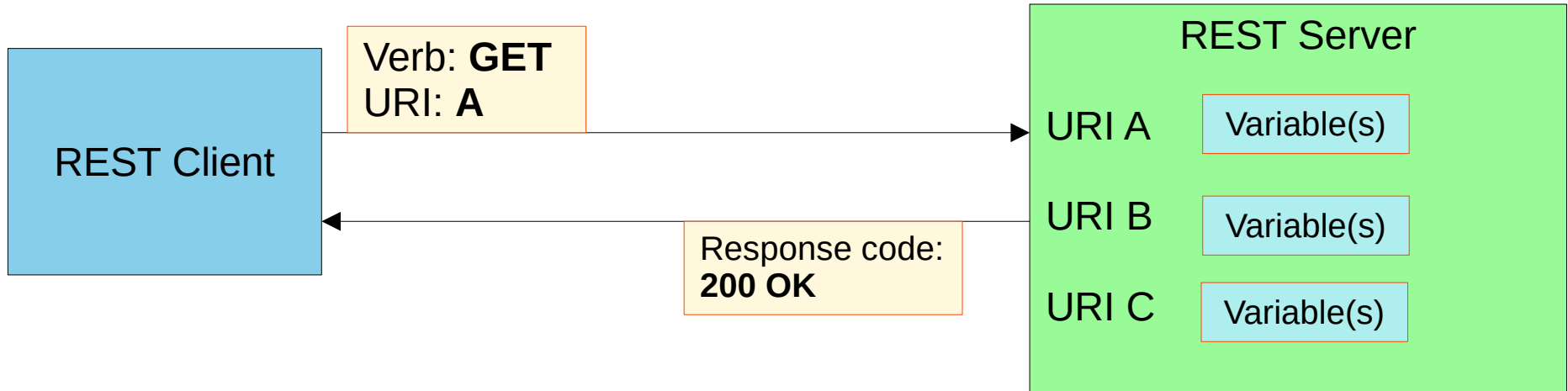
For more HTTP Response codes: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

For information about which Response codes to expect in response to each HTTP verb: <https://www.restapitutorial.com/lessons/httpmethods.html>

- REST stands for Representational State Transfer.
- **REST APIs** are also known as **REST-based APIs** or **RESTful APIs**.
  - REST isn't a specific API. Instead, it describes a set of rules about how the API should work.
- The six constraints of RESTful architecture are:
  - Uniform Interface
  - Client-server
  - Stateless
  - Cacheable or non-cacheable
  - Layered system
  - Code-on-demand (optional)
- For more details, check out: <https://restfulapi.net/rest-architectural-constraints/>

# REST: Client-server

- **REST APIs** use a client-server architecture.
- The client uses API calls (HTTP requests) to access the resources on the server.
- The separation between the client and server means they can both change and evolve independently of each other.  
→ When the client application changes or the server application changes, the interface between them must not break.



# REST: Stateless

- **REST APIs** exchanges are stateless.
- This means that each API exchange is a separate event, independent of all past exchanges between the client and server.
  - The server does not store information about previous requests from the client to determine how it should respond to new requests.
- If authentication is required, this means that the client must authenticate with the server for each request it makes.
- TCP is an example of a stateful protocol.
- UDP is an example of a stateless protocol.

\*Although REST APIs use HTTP, which uses TCP (stateful) as its Layer 4 protocol, HTTP and REST APIs themselves aren't stateful. The functions of each layer are separate!

# REST: Cacheable or Non-Cacheable

- **REST APIs** must support caching of data.
- *Caching* refers to storing data for future use.
  - For example, your computer might cache many elements of a web page so that it doesn't have to retrieve the entire page every time you visit it.
  - This improves performance for the client and reduces the load on the server.
- Not all resources have to be cacheable, but cacheable resources **MUST** be declared as cacheable.

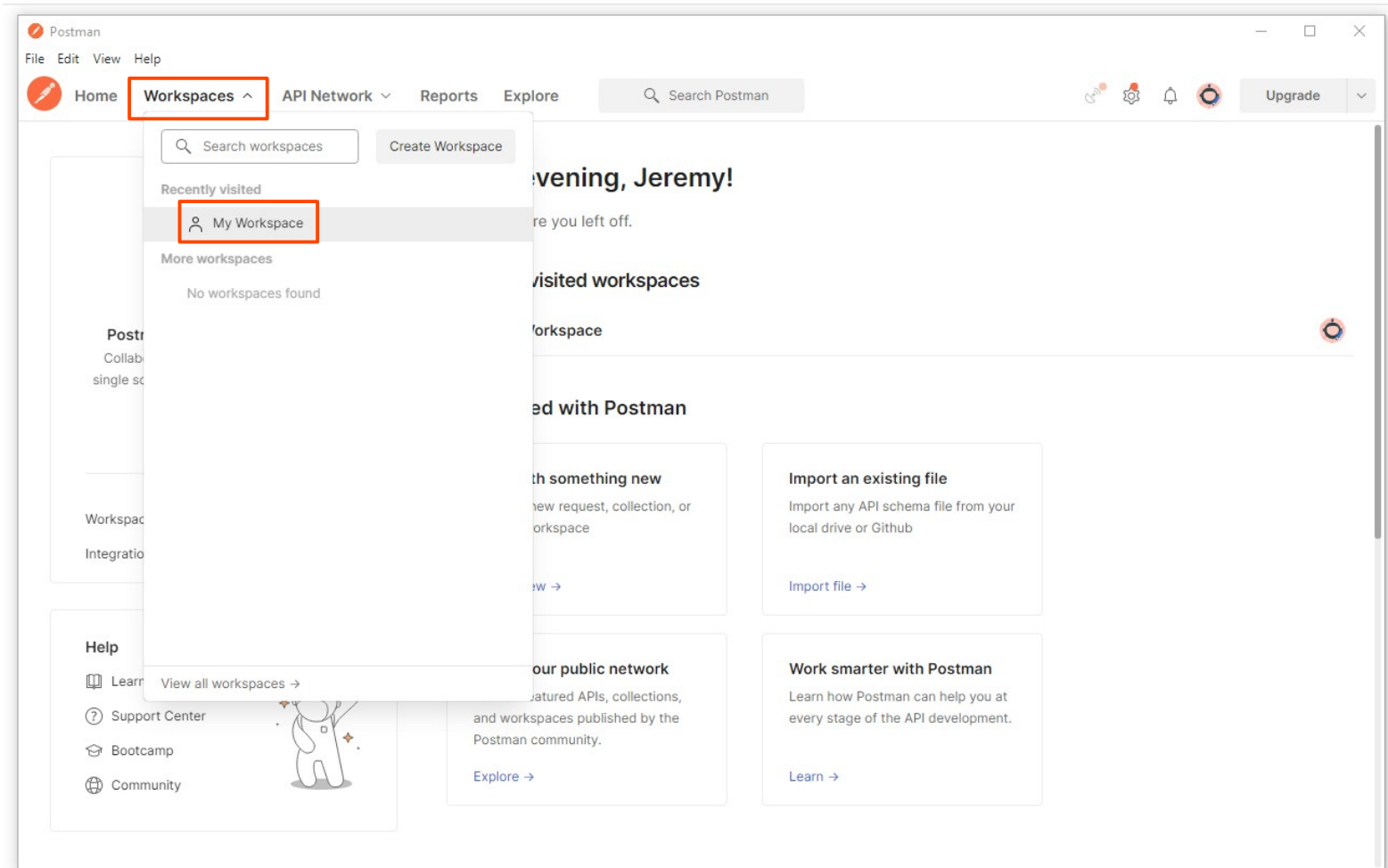
- **REST** (Representational State Transfer) **APIs** (aka **REST-based APIs** or **RESTful APIs**) are APIs that follow the REST architecture.
  - REST isn't a specific API. Instead, it describes a set of rules about how the API should work.
- The six constraints of RESTful architecture are:
  - Uniform Interface
  - Client-server
  - Stateless
  - Cacheable or non-cacheable
  - Layered system
  - Code-on-demand (optional)
- For more details, check out: <https://restfulapi.net/rest-architectural-constraints/>
- For applications to communicate over a network, networking protocols must be used to facilitate those communications.
  - For REST APIs, HTTP(S) is the most common choice.

Remember the CRUD actions, HTTP client request verbs, HTTP server response codes, and the basic characteristics of REST APIs.

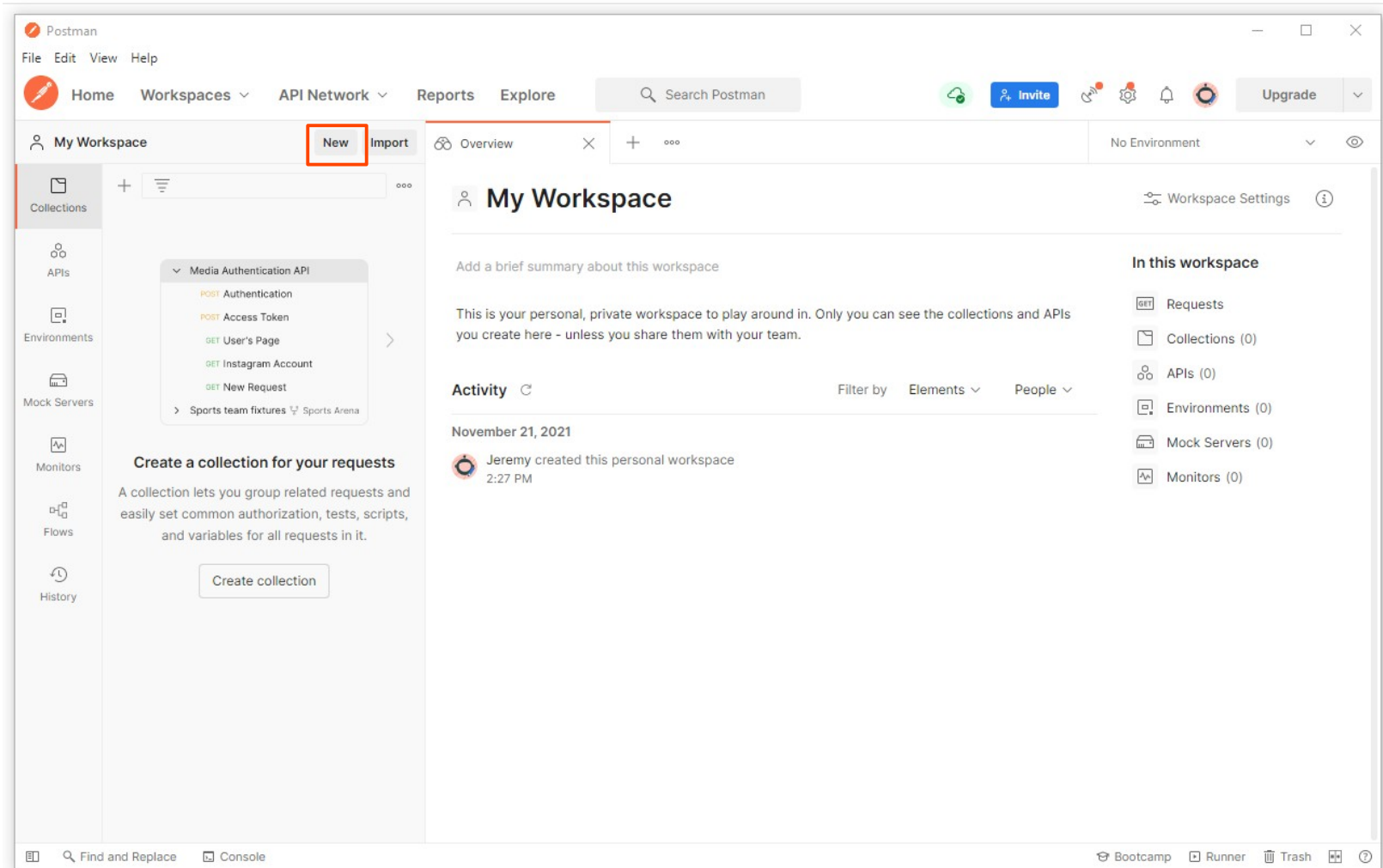


- “Cisco DevNet is Cisco's developer program to help developers and IT professionals who want to write applications and develop integrations with Cisco products, platforms, and APIs.”
  - DevNet offers lots of free resources such as courses, tutorials, labs, sandboxes, documentation, etc. to learn about automation and develop your skills.
  - There is also a DevNet certification track that you can pursue if you're interested in automation.
  - We will use their Cisco **DNA Center** Sandbox to send a REST API call using **Postman**.
    - DNA Center is one of Cisco's SDN controllers. We will cover it in more detail in the next video!
    - Postman is a platform for building and using APIs.
  - To start:
    - Make an account on [developer.cisco.com](https://developer.cisco.com)
    - Make an account on [postman.com](https://www.postman.com) + download the desktop app (<https://www.postman.com/downloads/>).
  - I will be following this tutorial on DevNet: <https://developer.cisco.com/docs/dna-center/#!/getting-started>
- \*you don't have to follow along, but I recommend you do!*

# API Calls to DNA Center



# API Calls to DNA Center



Postman

File Edit View Help

Home Workspaces API Network Reports Explore Search Postman

My Workspace **New** Import

Collections +

APIs

Environments

Mock Servers

Monitors

Flows

History

Media Authentication API

- POST Authentication
- POST Access Token
- GET User's Page
- GET Instagram Account
- GET New Request
- Sports team fixtures Sports Arena

Create a collection for your requests

A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.

Create collection

Overview

My Workspace

Add a brief summary about this workspace

This is your personal, private workspace to play around in. Only you can see the collections and APIs you create here - unless you share them with your team.

Activity

Filter by Elements People

November 21, 2021

Jeremy created this personal workspace 2:27 PM

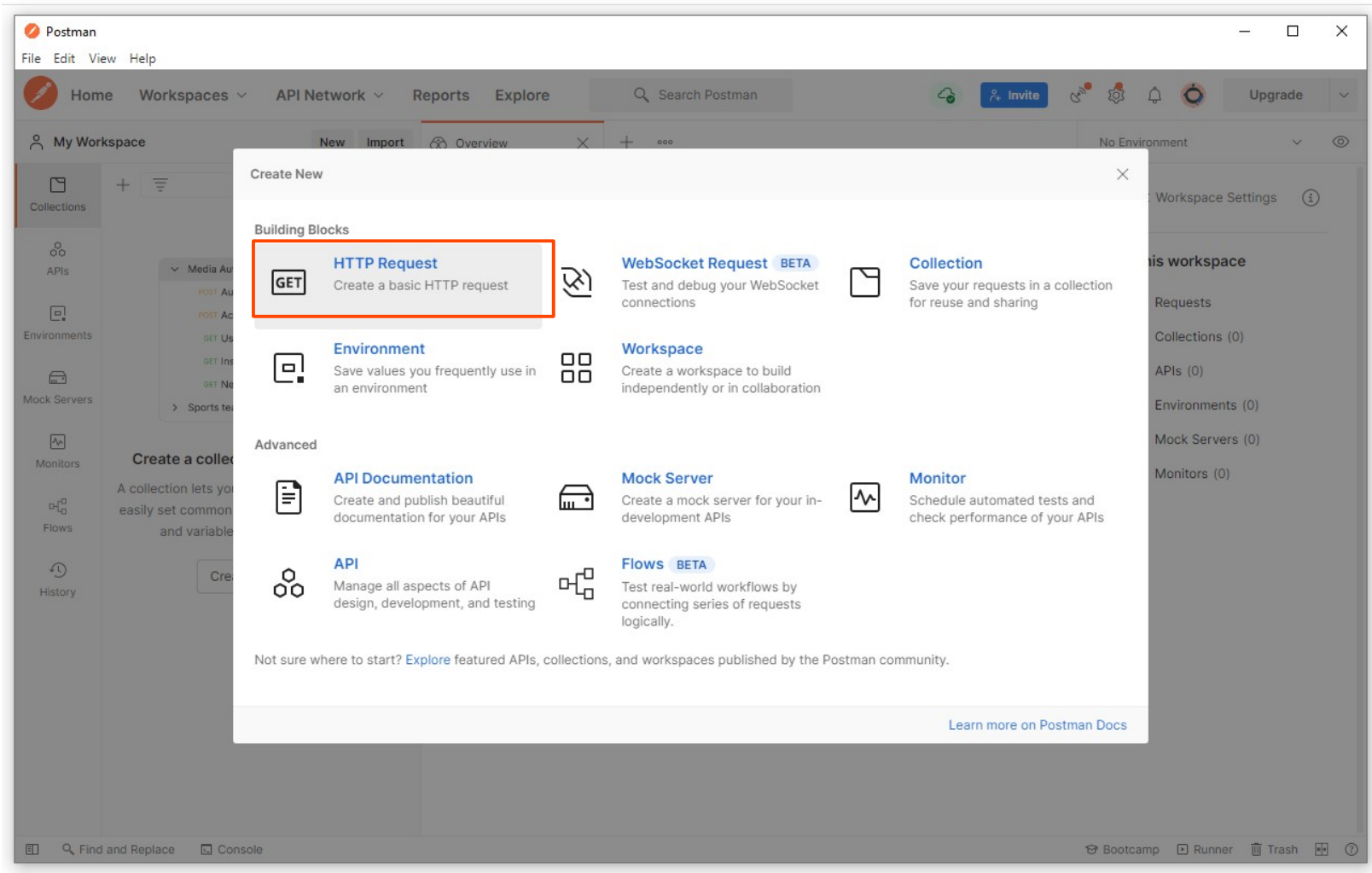
In this workspace

- Requests
- Collections (0)
- APIs (0)
- Environments (0)
- Mock Servers (0)
- Monitors (0)

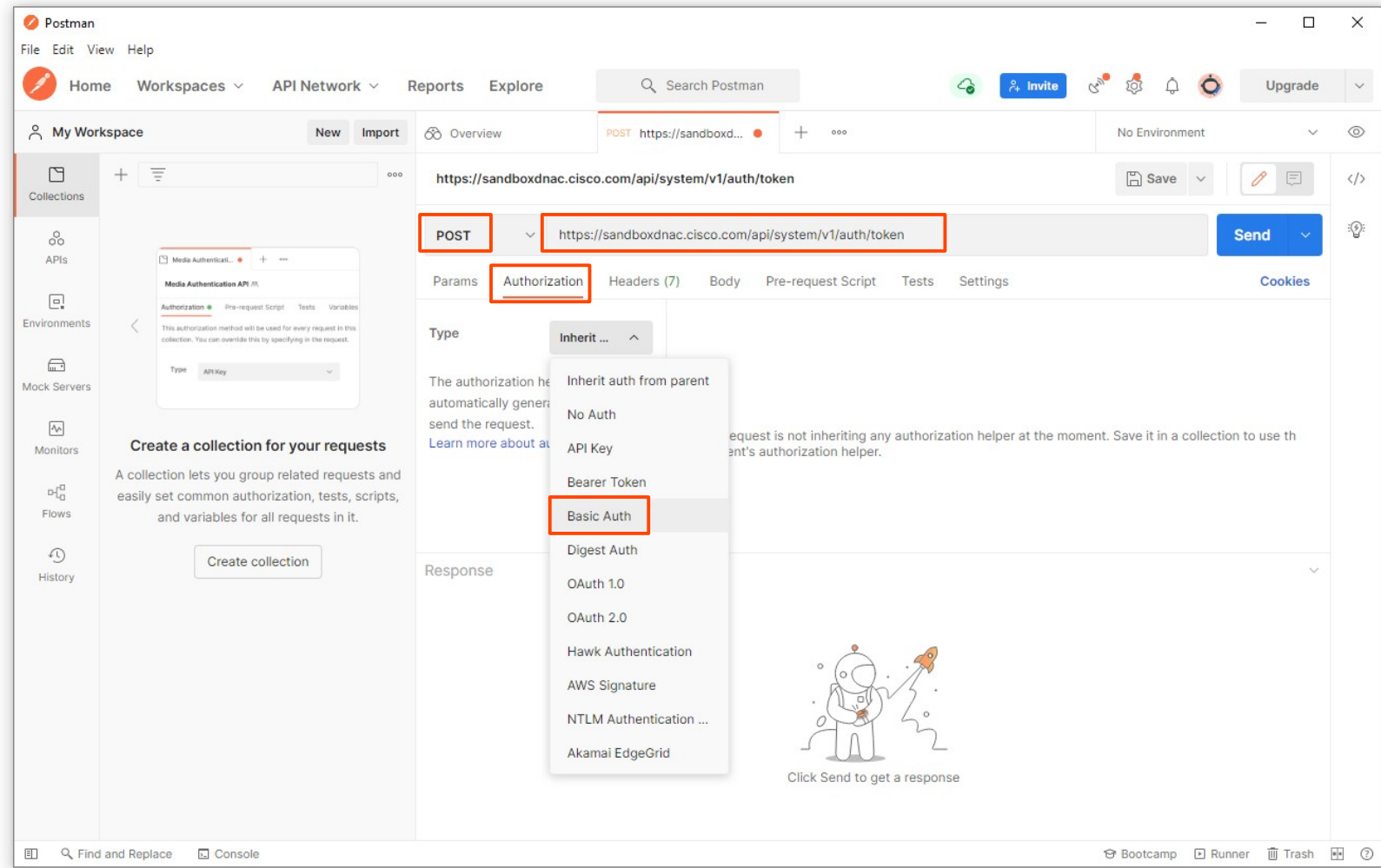
Find and Replace Console

Bootcamp Runner Trash

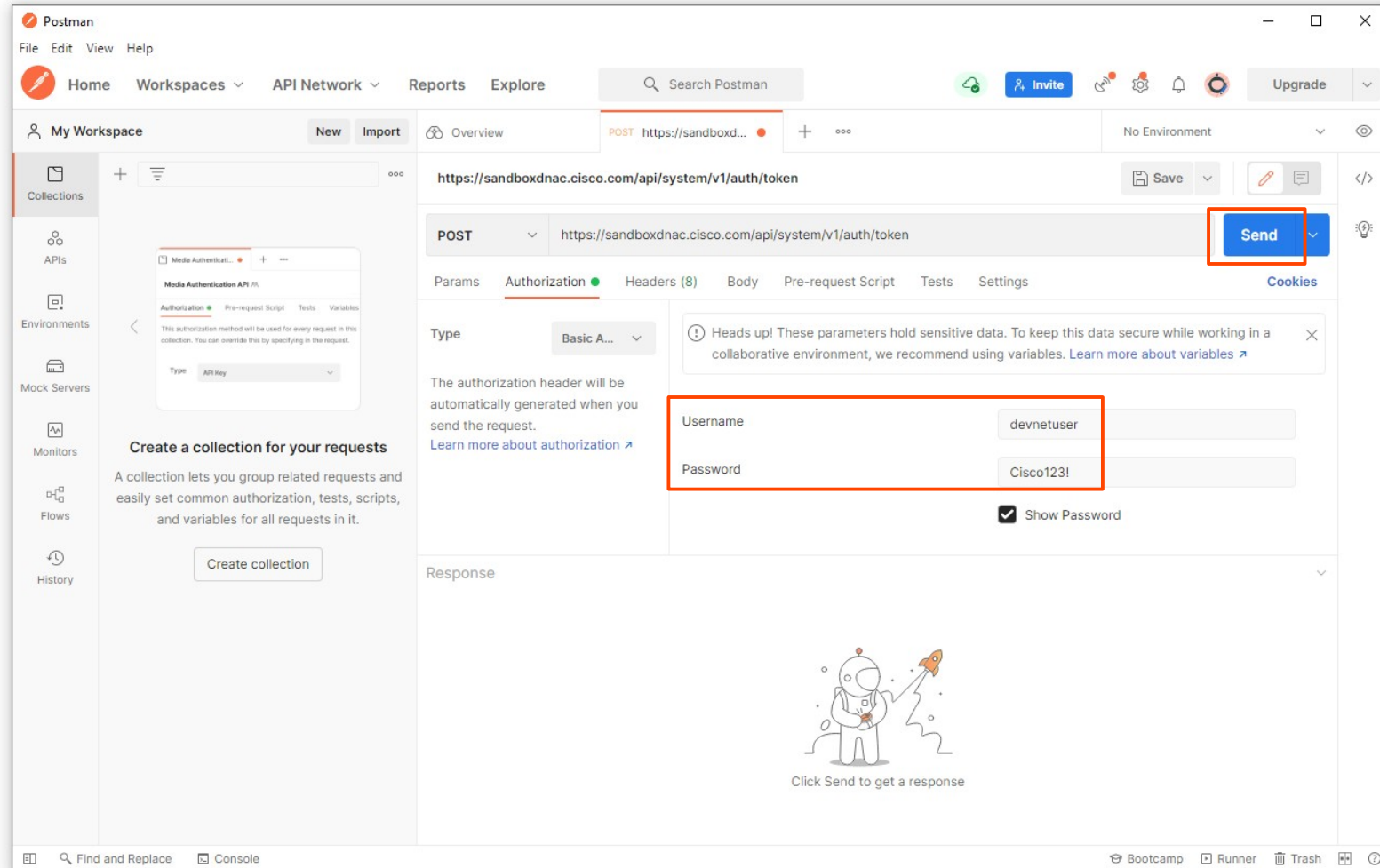
# API Calls to DNA Center



# API Calls to DNA Center



# API Calls to DNA Center

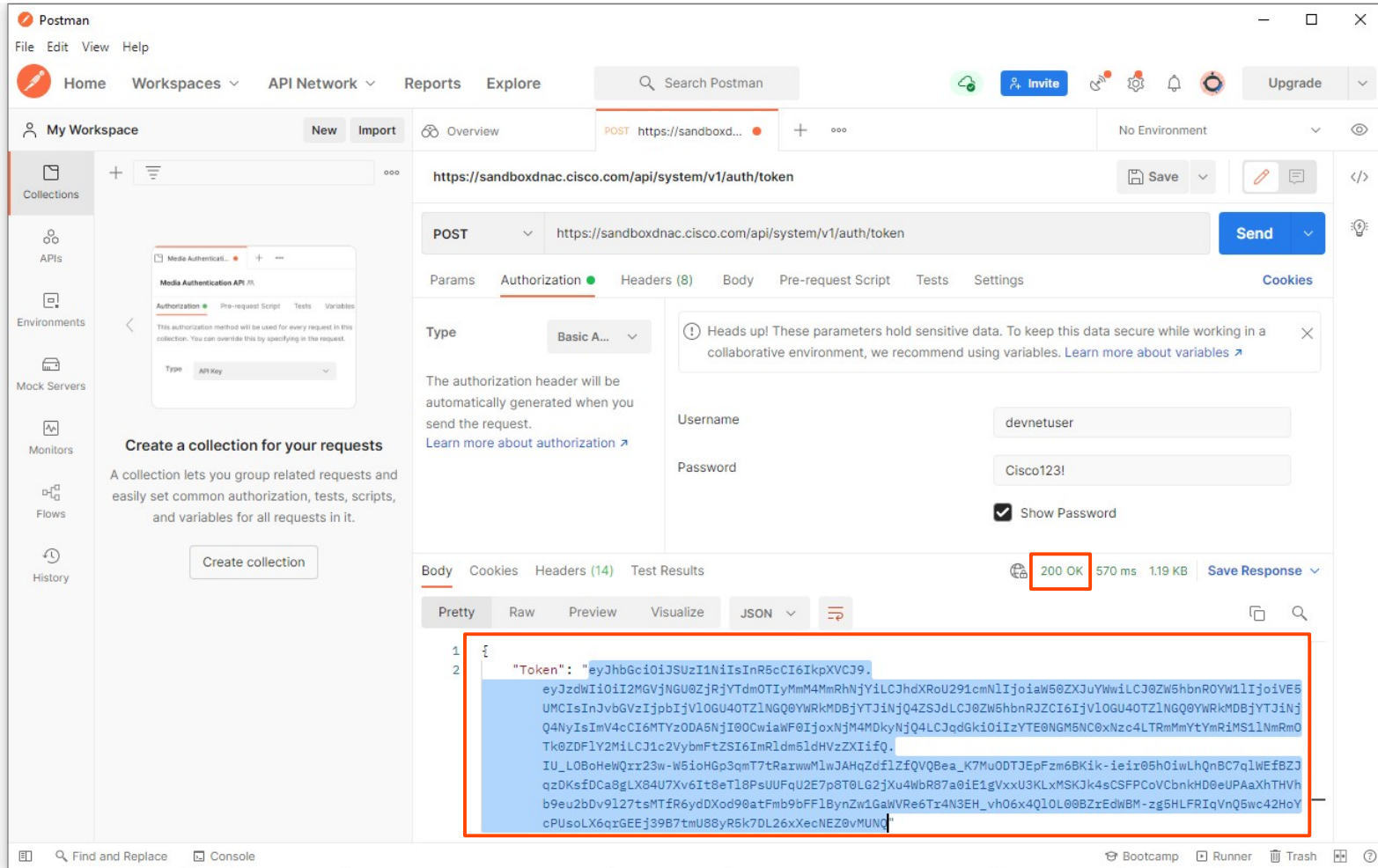


The screenshot shows the Postman application interface. The main workspace displays a POST request to the endpoint `https://sandboxdnac.cisco.com/api/system/v1/auth/token`. The request is configured with Basic Authentication, where the Username is `devnetuser` and the Password is `Cisco123!`. The Password field is highlighted with a red box. A warning message states: "Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables". The interface also shows a "Send" button, which is also highlighted with a red box. The left sidebar contains navigation options like Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The bottom status bar includes a search bar, console, and various tool icons.

**Postman Interface Details:**

- Request Method:** POST
- URL:** `https://sandboxdnac.cisco.com/api/system/v1/auth/token`
- Authorization:** Basic Authentication
  - Username:** `devnetuser`
  - Password:** `Cisco123!`
  - Show Password:** ☒
- Warning:** Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)
- Action:** Send

# API Calls to DNA Center



The screenshot shows the Postman application interface. The URL bar displays `https://sandboxdnac.cisco.com/api/system/v1/auth/token`. The request method is set to **POST**. The **Authorization** tab is active, showing a **Basic Auth** configuration with the username `devnetuser` and password `Cisco123!`. The **Body** tab is selected, showing a JSON response with a **200 OK** status. The response body contains a **Token** field with a long alphanumeric string.

**Request Details:**

- Method: **POST**
- URL: `https://sandboxdnac.cisco.com/api/system/v1/auth/token`
- Authorization: **Basic Auth**
- Username: `devnetuser`
- Password: `Cisco123!`
- Show Password: ☒

**Response Details:**

- Status: **200 OK**
- Time: `570 ms`
- Size: `1.19 KB`
- Save Response: ☐

**Response Body (JSON):**

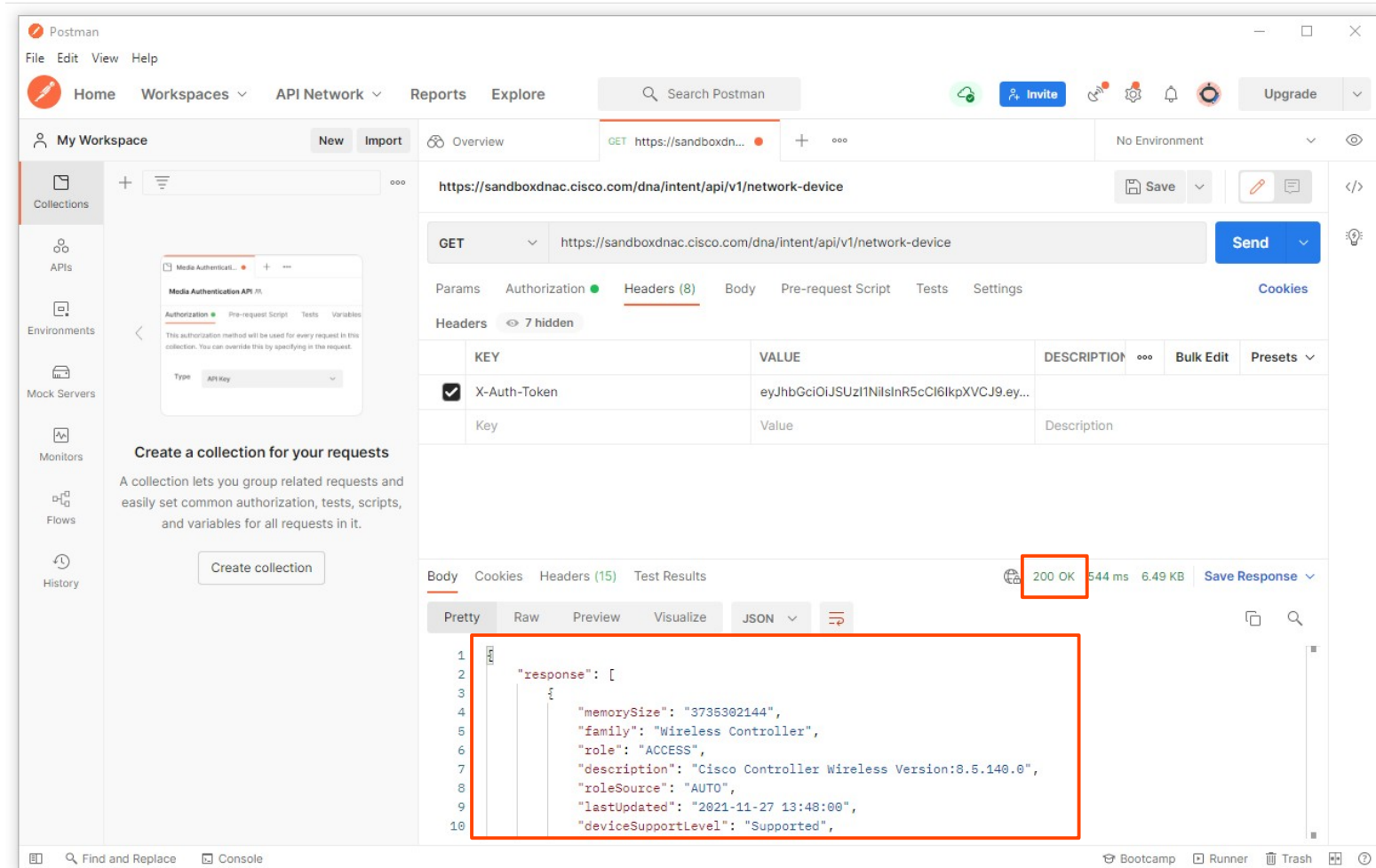
```
{
  "Token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI2MGVjNGU0ZjRjYTdmOTIyMmM4MmRhNjYiLCJhdXRoU291cmNlIjoiaW50ZXJyYmwiLCJ0ZW5hbnR5YWR1IjoiVE5UMCIsIn3vbnVzIjpbIjVlOGU0OTZlNGQ0YWRkM0BjYjYjInjQ4ZSjdlC30ZW5hbnRjZCI6IjVlOGU0OTZlNGQ0YWRkM0BjYjYjInjQ4NyIsInMv4cCI6MTYzODAsInR5cCI6Ij00CwIiwiaWF0IjoxNjM4MDkyNjQ4LCJqdGkiOiIzYTE0NGM5NC0xNzc4LTRmMmYtYmRiMS1lNmRmOtk0ZDFlY2MiLCJ1c2VybmFtZSI6ImRldm5ldHVzZXIiOiQ.eyJ0b2R5bG90ZjRjYWRkM0BjYjYjInjQ4ZSjdlC30ZW5hbnRjZCI6IjVlOGU0OTZlNGQ0YWRkM0BjYjYjInjQ4NyIsInMv4cCI6MTYzODAsInR5cCI6Ij00CwIiwiaWF0IjoxNjM4MDkyNjQ4LCJqdGkiOiIzYTE0NGM5NC0xNzc4LTRmMmYtYmRiMS1lNmRmOtk0ZDFlY2MiLCJ1c2VybmFtZSI6ImRldm5ldHVzZXIiOiQ."
}
```







# API Calls to DNA Center



Postman interface showing an API call to DNA Center.

**URL:** `https://sandboxdnac.cisco.com/dna/intent/api/v1/network-device`

**Method:** GET

**Headers (8):**

KEY	VALUE	DESCRIPTION
X-Auth-Token	eyJhbGciOiJIUzUzIiwiaW50IjoiR5cCI6IkpXVCJ9.eyJ...	
Key	Value	Description

**Response:** 200 OK, 544 ms, 6.49 KB

**JSON Body:**

```

1 {
2   "response": [
3     {
4       "memorySize": "3735302144",
5       "family": "Wireless Controller",
6       "role": "ACCESS",
7       "description": "Cisco Controller Wireless Version:8.5.140.0",
8       "roleSource": "AUTO",
9       "lastUpdated": "2021-11-27 13:48:00",
10      "deviceSupportLevel": "Supported",
    }
  ]
}

```

# API Calls to DNA Center

```
{
  "memorySize": "NA",
  "family": "Switches and Hubs",
  "role": "ACCESS",
  "roleSource": "AUTO",
  "lastUpdated": "2021-11-27 13:50:20",
  "deviceSupportLevel": "Supported",
  "softwareType": "IOS-XE",
  "softwareVersion": "17.3.3",
  "macAddress": "84:8a:8d:05:76:00",
  "collectionInterval": "Global Default",
  "inventoryStatusDetail": "<status><general
code=\\"SUCCESS\\"/></status>",
  "serialNumber": "FCW2220G09V",
  "lastUpdateTime": 1638021020343,
  "hostname": "leaf1.abc.inc",
  "tagCount": "0",
  "tunnelUdpPort": null,
  "uptimeSeconds": 2648950,
  "waasDeviceMode": null,
  "apManagerInterfaceIp": "",
  "bootDateTime": "2021-10-28 18:10:20",
  "collectionStatus": "Managed",
  "locationName": null,
  "managementIpAddress": "10.10.20.81",
  "platformId": "C9300-24U",
  "reachabilityFailureReason": "",
  "reachabilityStatus": "Reachable",
  "series": "Cisco Catalyst 9300 Series Switches",
  "snmpContact": "",
  "snmpLocation": "",
```

```
    "upTime": "29 days, 19:40:48.91",
    "apEthernetMacAddress": null,
    "associatedWlcIp": "",
    "errorCode": null,
    "errorDescription": null,
    "interfaceCount": "0",
    "lineCardCount": "0",
    "lineCardId": "",
    "managedAtleastOnce": true,
    "location": null,
    "type": "Cisco Catalyst 9300 Switch",
    "managementState": "Managed",
    "instanceUuid": "aa0a5258-3e6f-422f-9c4e-9c196db115ae",
    "instanceTenantId": "5e8e896e4d4add00ca2b6487",
    "id": "aa0a5258-3e6f-422f-9c4e-9c196db115ae"
  }
}
```



# Things we covered

- API Review
- CRUD operations and HTTP verbs
- REST APIs
- REST API Calls using Cisco DevNet

Remember the CRUD actions, HTTP client request verbs, HTTP server response codes, and the basic characteristics of REST APIs.

What HTTP response code would you expect to receive if you tried to GET a resource that doesn't exist on the server?

a) 403

b) 404

c) 500

d) 504

Which of the following is NOT a constraint of RESTful architecture?

- a) Client-Server
- b) Cacheable
- c) Stateful
- d) Layered System
- e) Uniform Interface

Which category of HTTP response would you expect in response to a successful request?

- a) 1xx
- b) 2xx
- c) 3xx
- d) 4xx
- e) 5xx

HTTP verbs PUT and PATCH are equivalent to what CRUD operation?

a) C

b) R

c) U

d) D

Which of the following would you expect to find as the *scheme* of a URI?

- a) HTTPS
- b) sandboxdnac.cisco.com
- c) /dna/intent/api/v1/network-device
- d) accept:application/xml