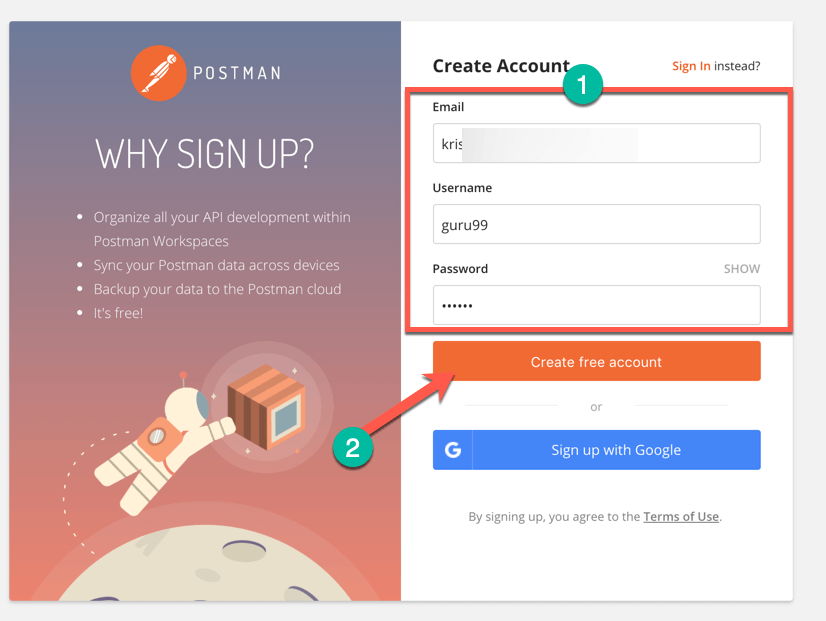
1.Demonstrate how to set up Postman.

**Step 2.1.1: Installing Postman**

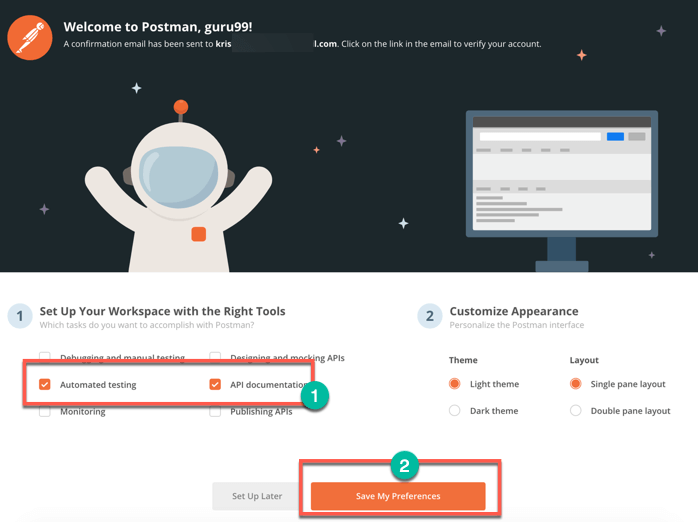
* Postman is already installed in your practice lab. Refer QA to QE lab guide -- Phase 3 for more information.

**Step 2.1.2: Setting up Postman**

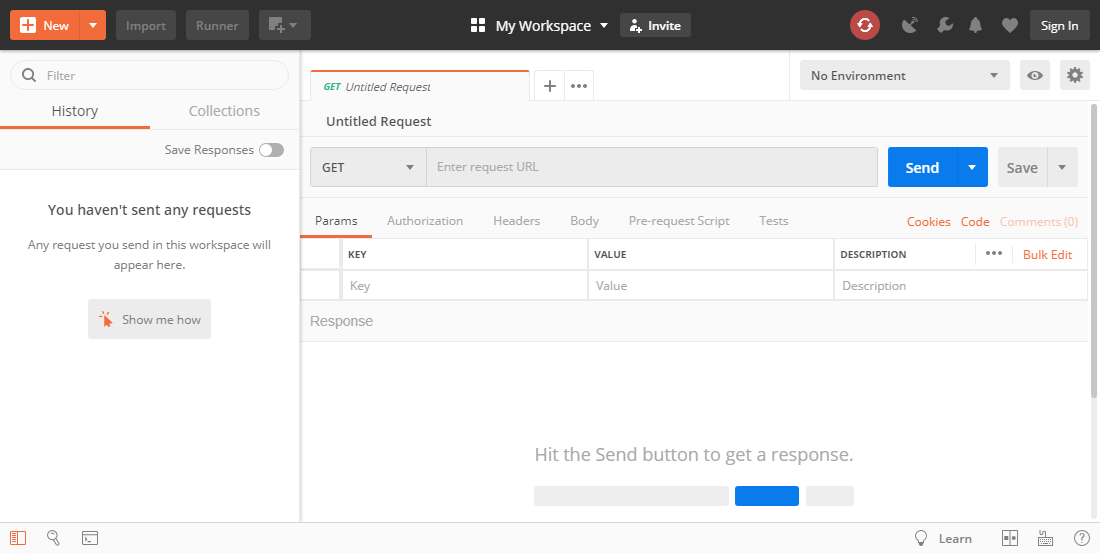
* Open Postman and click on the sign up button to create a Postman Account.



* NOTE: There are two ways to sign up for a Postman account. One is to create your own Postman account, and the other is to use a Google account. Though Postman allows users to use the tool without logging in, signing up ensures that your collection is saved and can be accessed for later use.
* Select the required workspace tools.
* Click on Save My Preferences.



* You will see this Startup Screen below:



2. Demonstrate how to create the first API request.

* Open Postman Workspace.
* On the top left corner, click on the **New option** button**.**
* In the drop-down options available, click on Request option.
* Adda name for the request.
* Scroll down and click on create a collection option and add a name for the collection.
* Click on the Save button.

Your Request is ready now

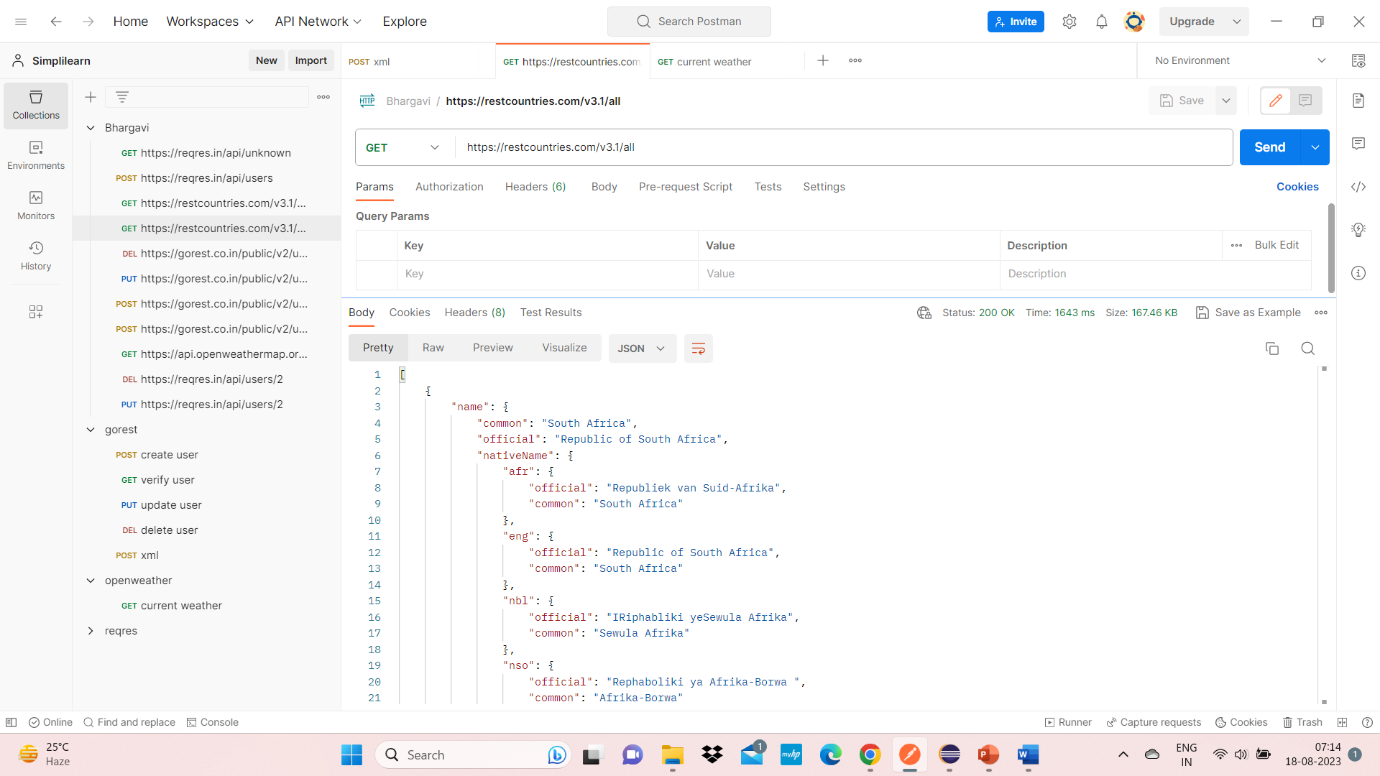
**:** Getting a response of API request on Postman

* Take one Sample URL as shown below.

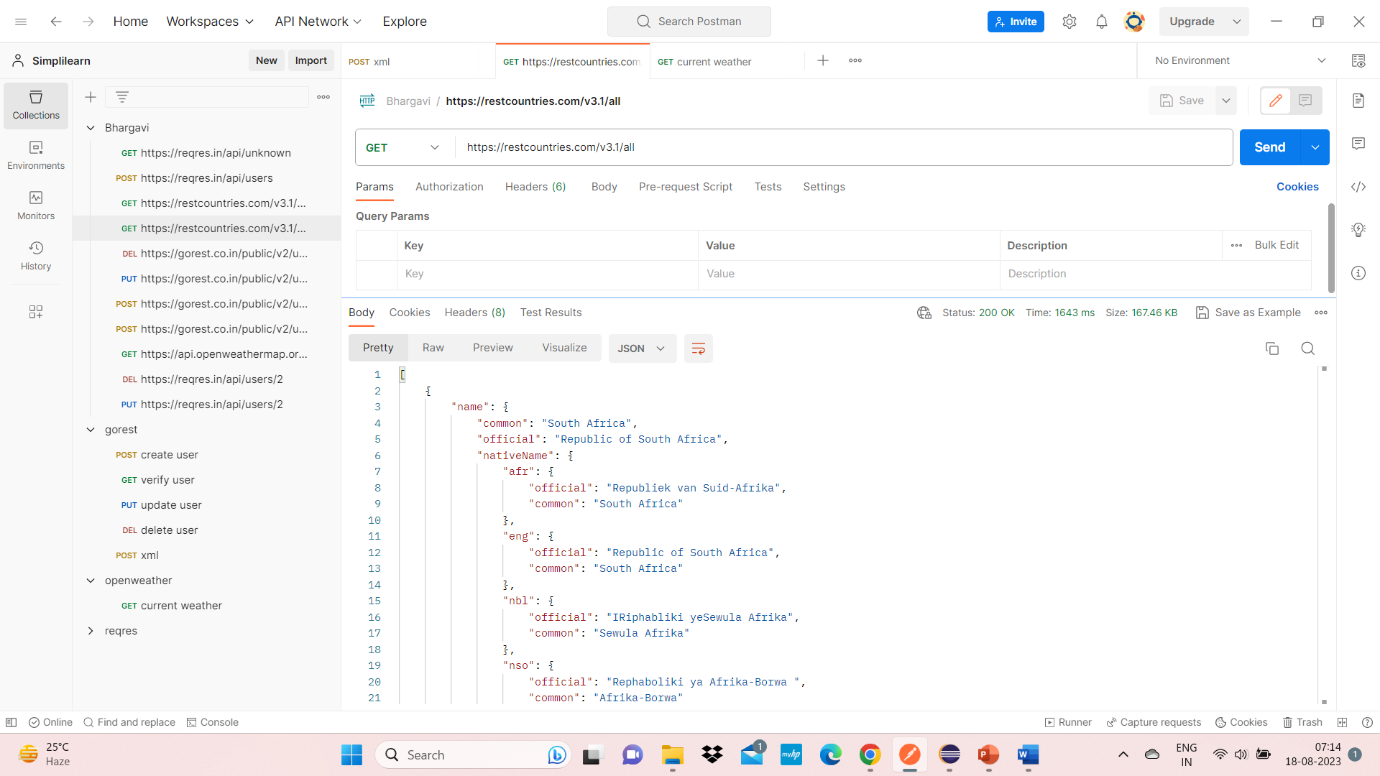
<https://restcountries.com/v3.1/all>

In the request URL field, input the above link.

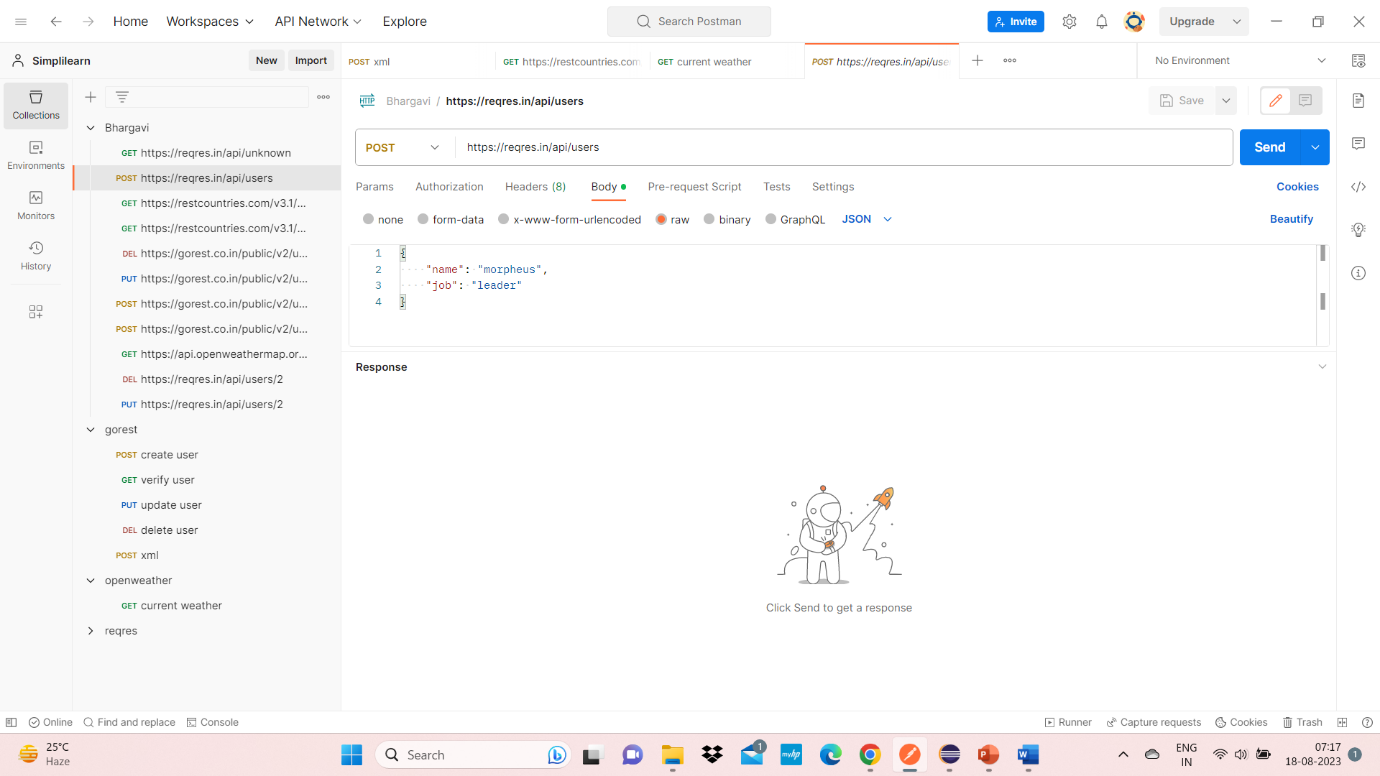
Click on the Send button



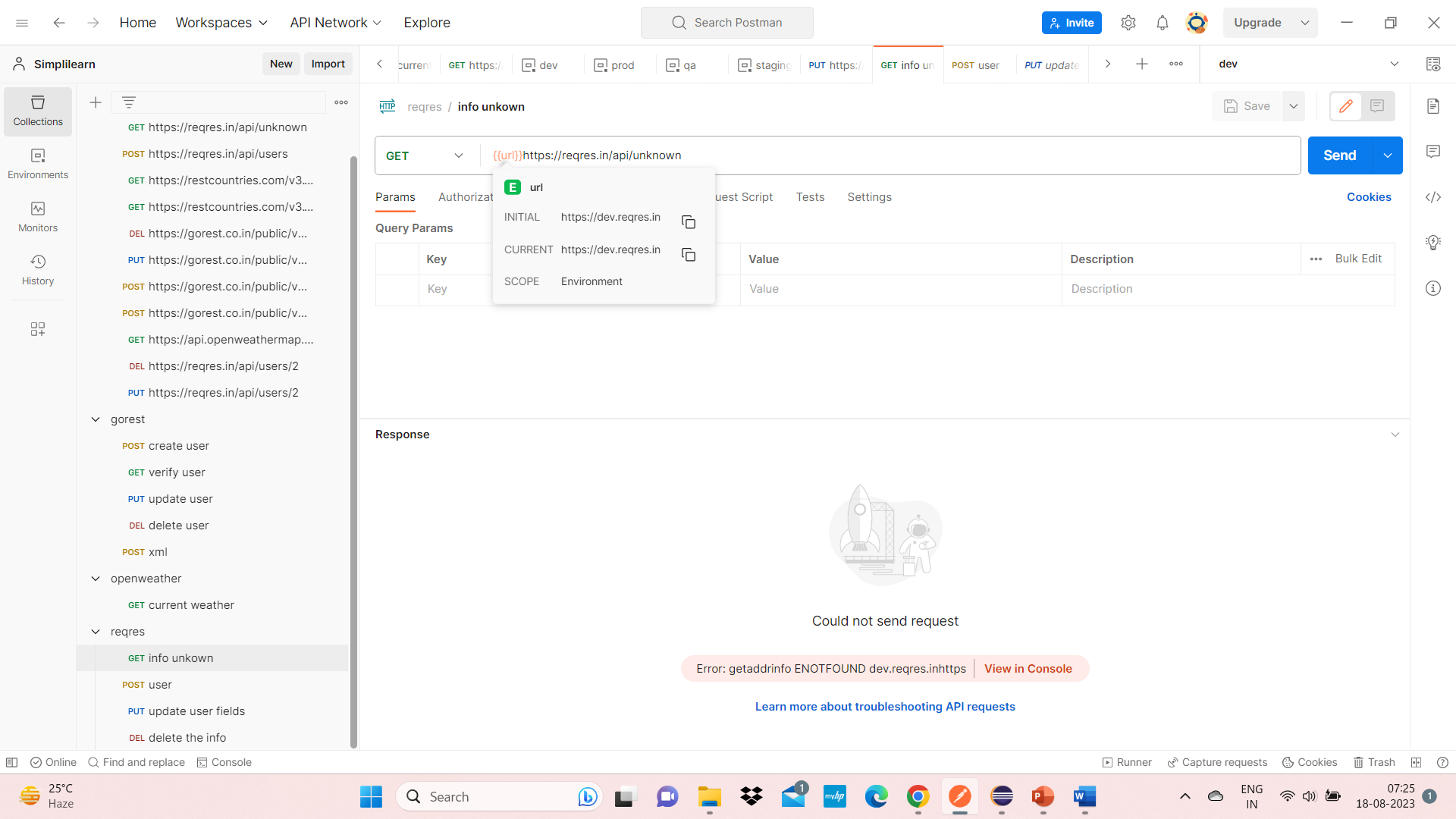
Explain how to work on GET requests, with an example.



Demonstrate how to work with POST requests.

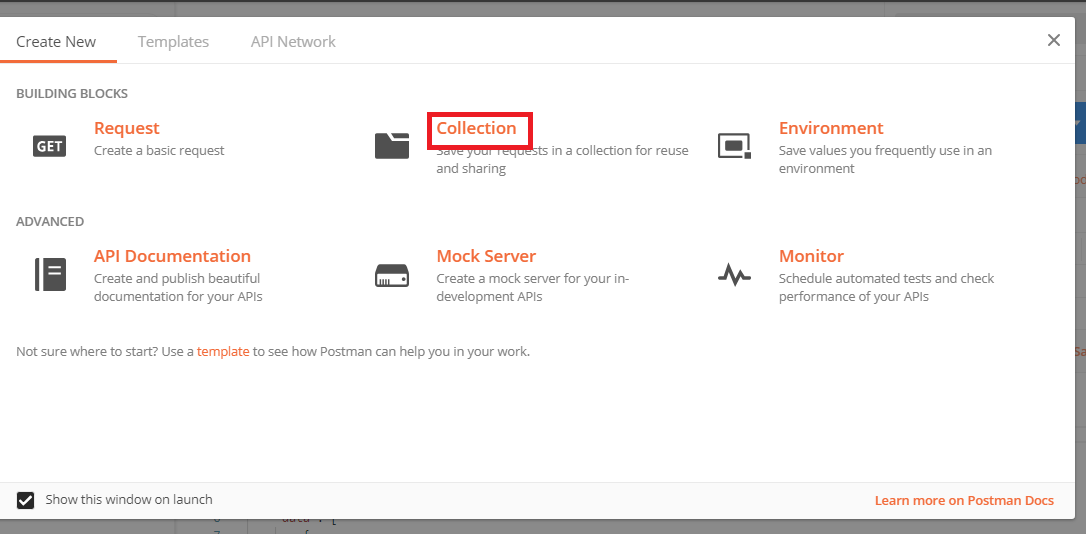


Demonstrate how to parameterize requests.



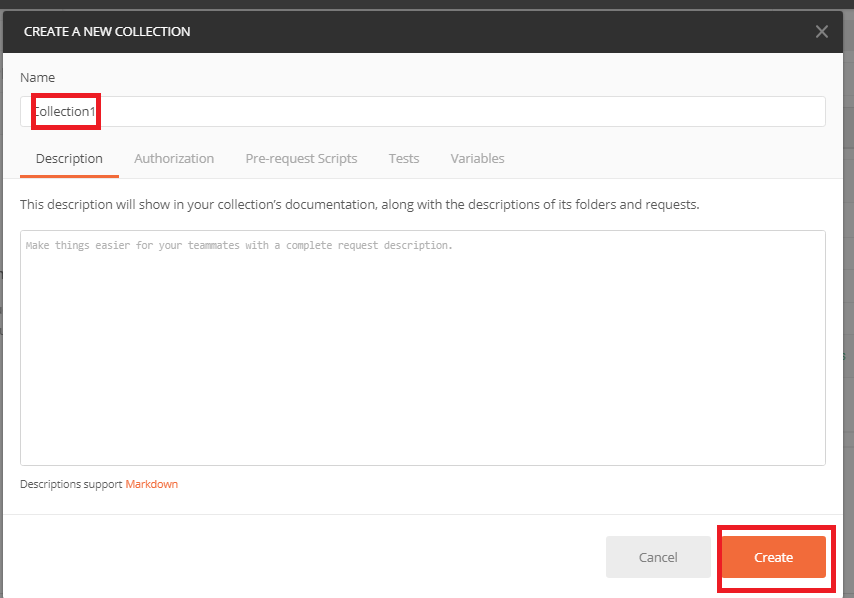
Demonstrate how to create a Collection.

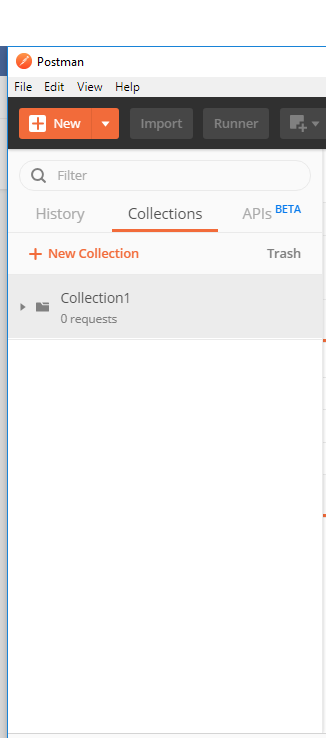
* Click on the NEW button
* Then you get a pop up like below:



**Steps 2.7.2:** Inputting the Desired name

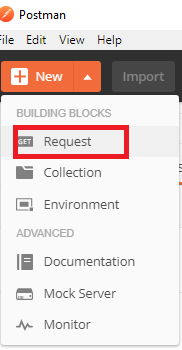
* Click on the Collection button
* Then you can pass your desire name
* Click the Create button



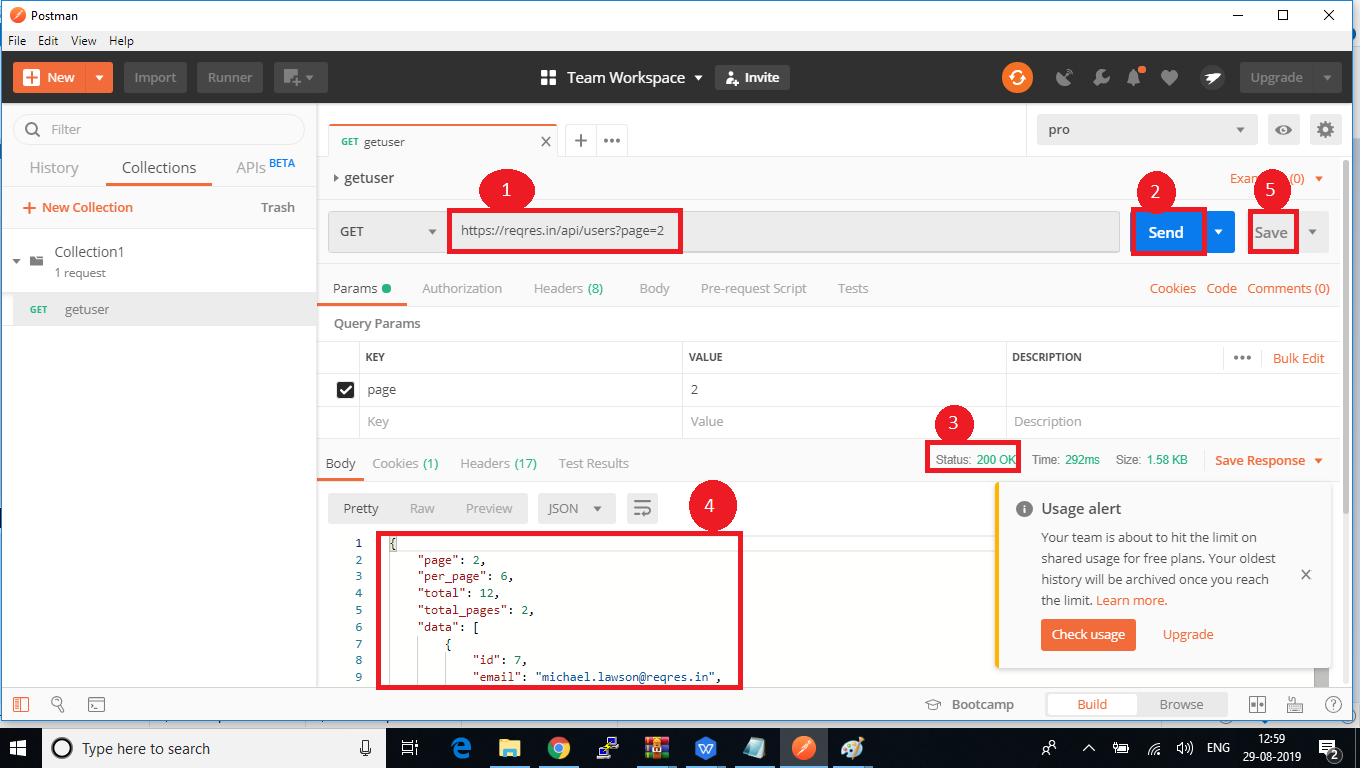


**Steps 2.7.3:** Adding a new request in Collection

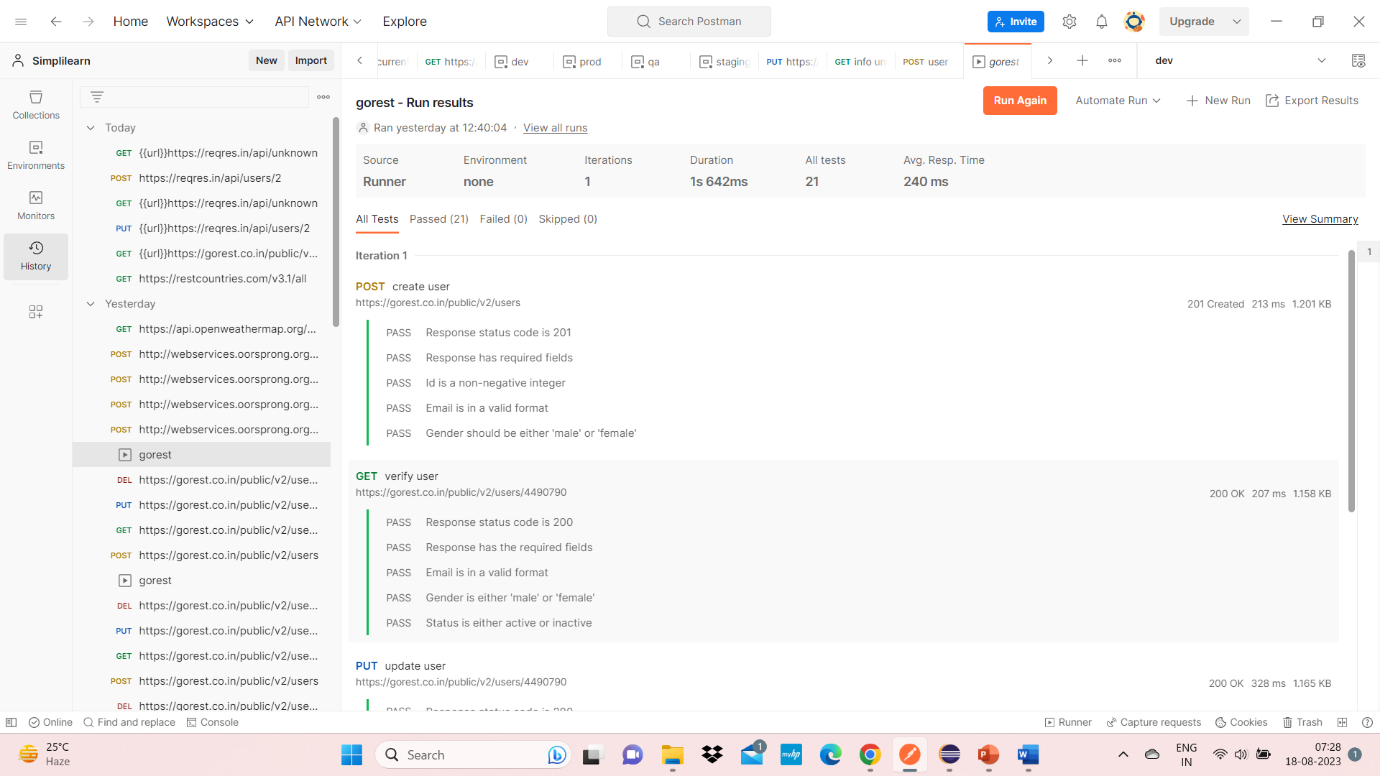
* Click on New
* Give a name

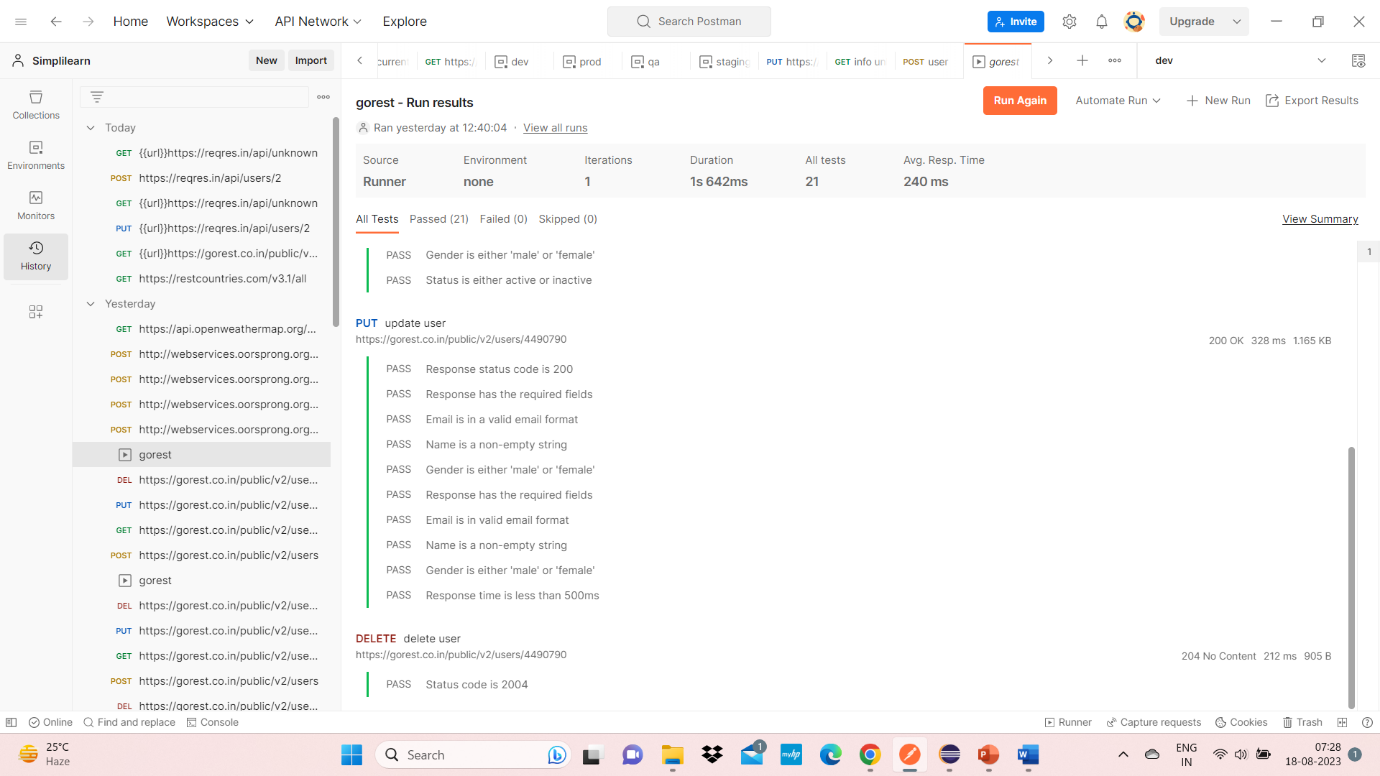


* Pass sample url (https://reqres.in/api/users?page=2)
* Save it

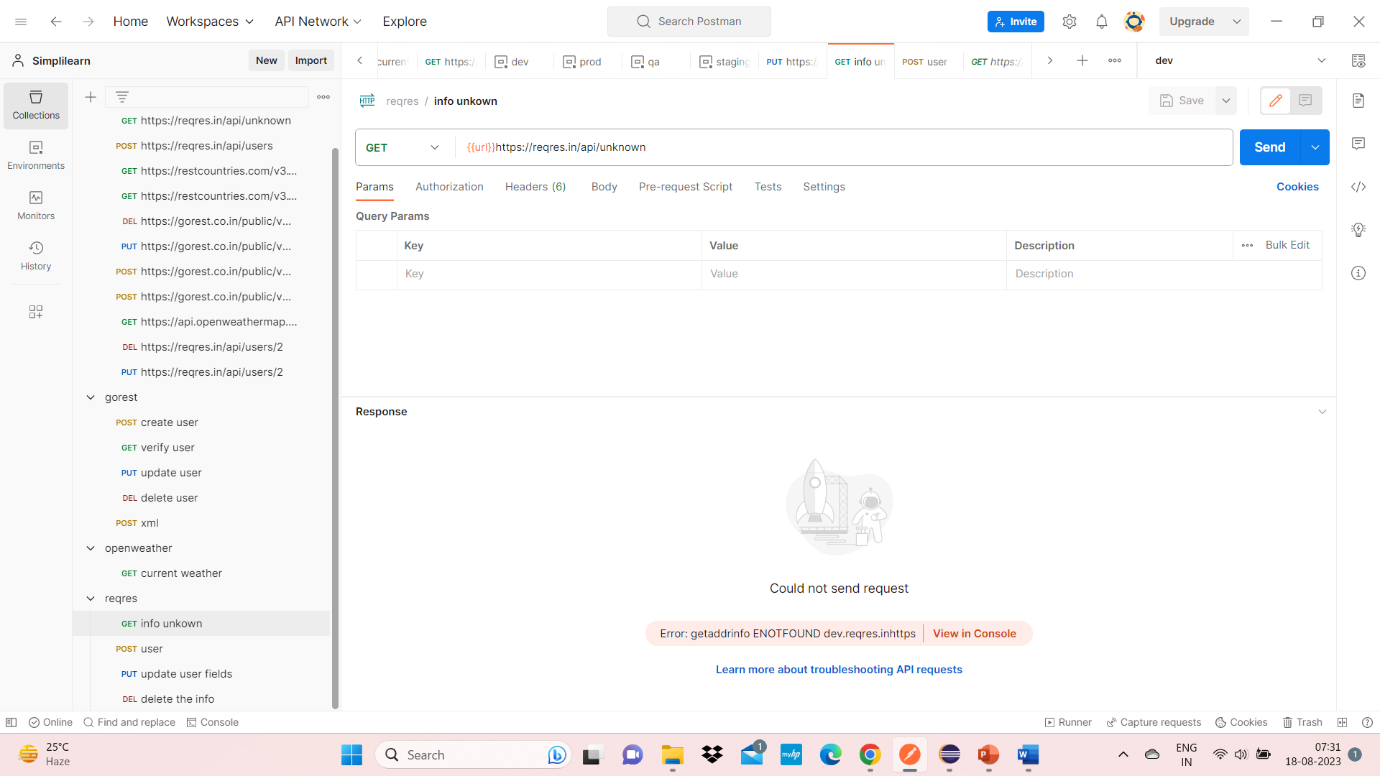


Demonstrate how to run a Collection using Collection Runner.

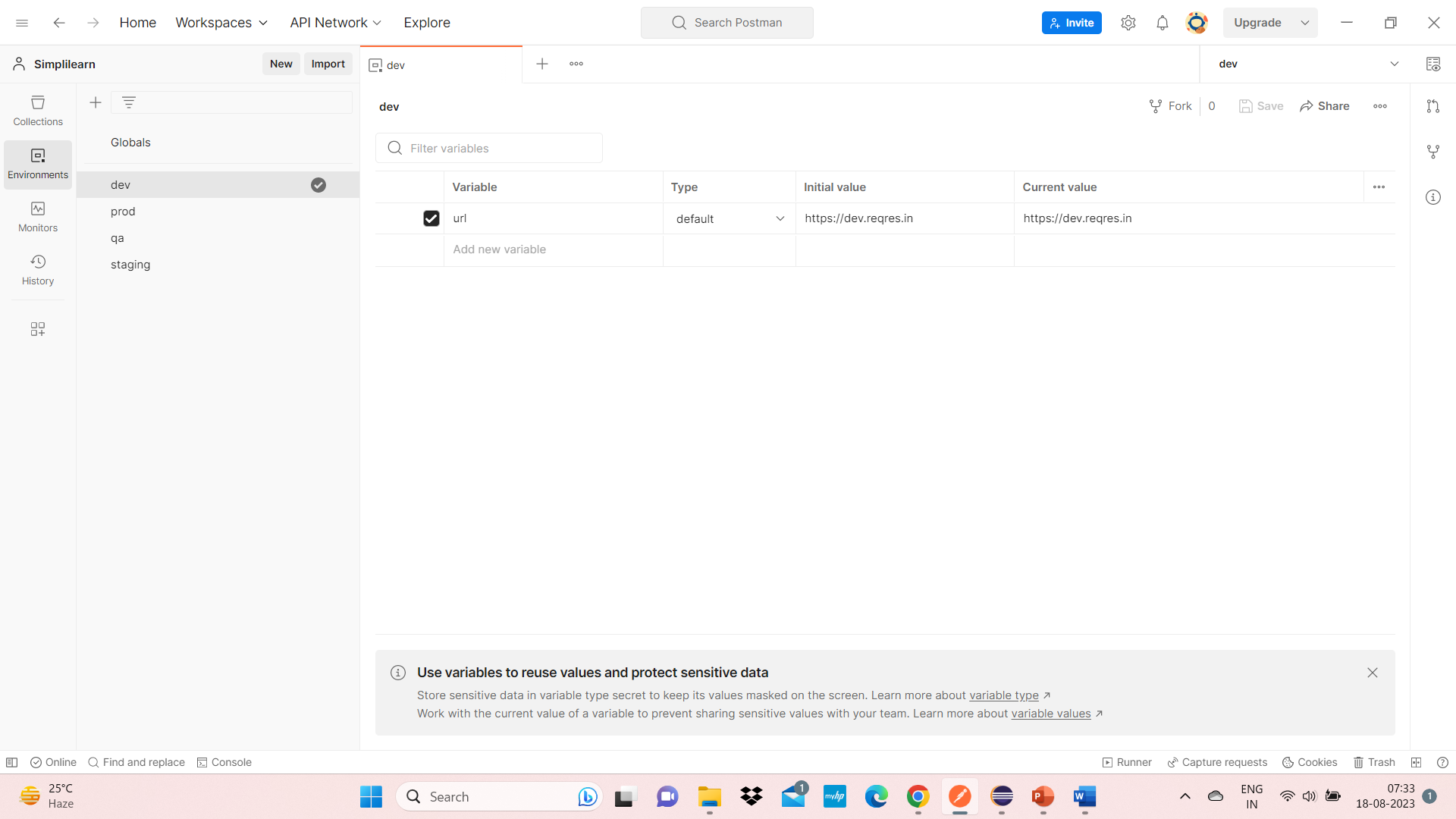
****



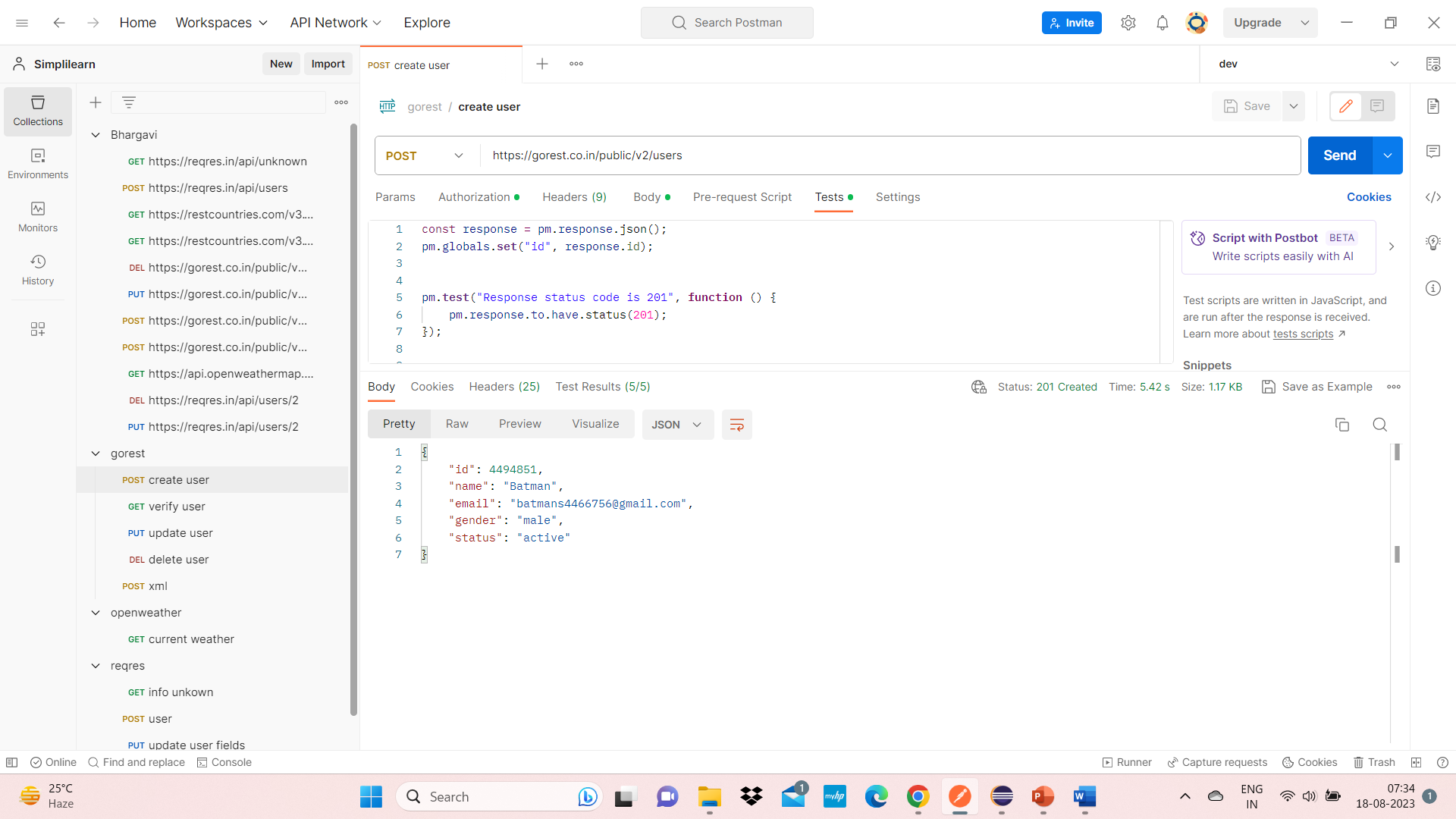
Demonstrate how to work with variables in Postman.

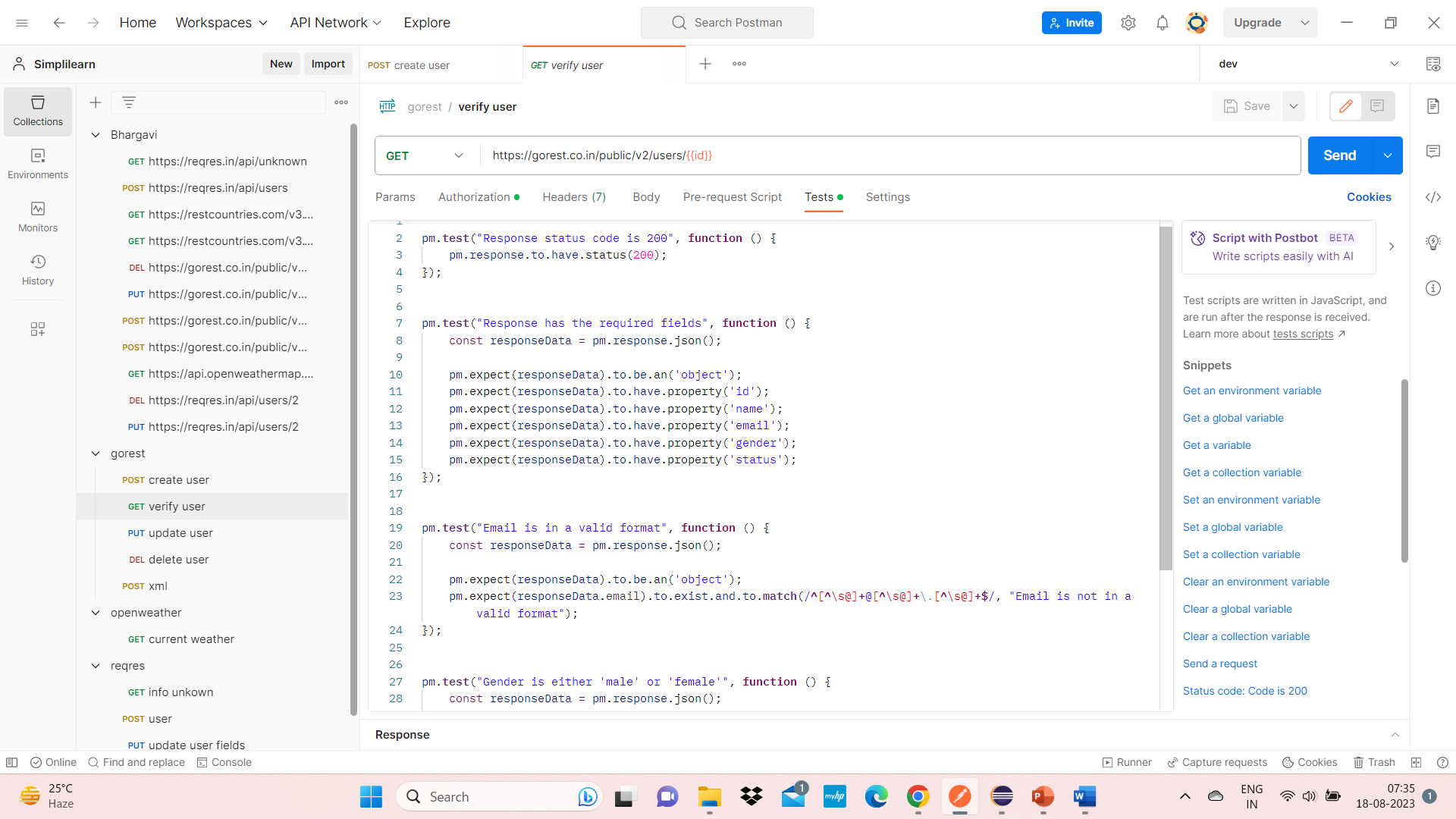


Demonstrate how environments are used in Postman.



Demonstrate how test scripts are written in Postman.





For PUT:-

pm.**test**("Response status code is 200", **function** () {

    pm.response.to.have.status(200);

});

pm.**test**("Response has the required fields", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData).to.be.an('object');

    pm.expect(responseData.email).to.exist;

    pm.expect(responseData.name).to.exist;

    pm.expect(responseData.gender).to.exist;

    pm.expect(responseData.status).to.exist;

    pm.expect(responseData.id).to.exist;

});

pm.**test**("Email is in a valid email format", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData).to.be.an('object');

    pm.expect(responseData.email).to.**match**(/**^**[\w-]**+**(\.[\w-]**+**)**\***@([\w-]**+**\.)**+**[a-zA-Z]**{2,7}$**/);

});

pm.**test**("Name is a non-empty string", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData.name).to.be.a('string').and.to.have.lengthOf.at.least(1, "Value should not be empty");

});

pm.**test**("Gender is either 'male' or 'female'", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData.gender).to.be.oneOf(['male', 'female']);

});

pm.**test**("Response has the required fields", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData).to.be.an('object');

    pm.expect(responseData.email).to.exist.and.to.be.a('string');

    pm.expect(responseData.name).to.exist.and.to.be.a('string');

    pm.expect(responseData.gender).to.exist.and.to.be.a('string');

    pm.expect(responseData.status).to.exist.and.to.be.a('string');

    pm.expect(responseData.id).to.exist.and.to.be.a('number');

});

pm.**test**("Email is in valid email format", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData.email).to.**match**(/**^**[\w-]**+**(\.[\w-]**+**)**\***@([\w-]**+**\.)**+**[a-zA-Z]**{2,7}$**/);

});

pm.**test**("Name is a non-empty string", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData.name).to.be.a('string').and.to.have.lengthOf.at.least(1, "Name should not be empty");

});

pm.**test**("Gender is either 'male' or 'female'", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData.gender).to.be.oneOf(['male', 'female']);

});

pm.**test**("Response time is less than 500ms", **function** () {

    pm.expect(pm.response.responseTime).to.be.below(500);

});

For delete:-

pm.**test**("Status code is 2004", **function** () {

    pm.response.to.have.status(204);

});

For Get:-

pm.**test**("Response status code is 200", **function** () {

    pm.response.to.have.status(200);

});

pm.**test**("Response has the required fields", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData).to.be.an('object');

    pm.expect(responseData).to.have.property('id');

    pm.expect(responseData).to.have.property('name');

    pm.expect(responseData).to.have.property('email');

    pm.expect(responseData).to.have.property('gender');

    pm.expect(responseData).to.have.property('status');

});

pm.**test**("Email is in a valid format", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData).to.be.an('object');

    pm.expect(responseData.email).to.exist.and.to.**match**(/**^**[**^**\s@]**+**@[**^**\s@]**+**\.[**^**\s@]**+$**/, "Email is not in a valid format");

});

pm.**test**("Gender is either 'male' or 'female'", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData.gender).to.exist.and.to.be.oneOf(['male', 'female']);

});

pm.**test**("Status is either active or inactive", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData.status).to.be.oneOf(["active", "inactive"]);

});

For Post:-

const response **=** pm.response.json();

pm.globals.**set**("id", response.id);

pm.**test**("Response status code is 201", **function** () {

    pm.response.to.have.status(201);

});

pm.**test**("Response has required fields", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData).to.be.an('object');

    pm.expect(responseData.id).to.exist.and.to.be.a('number');

    pm.expect(responseData.name).to.exist.and.to.be.a('string');

    pm.expect(responseData.email).to.exist.and.to.be.a('string');

    pm.expect(responseData.gender).to.exist.and.to.be.a('string');

    pm.expect(responseData.status).to.exist.and.to.be.a('string');

});

pm.**test**("Id is a non-negative integer", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData).to.be.an('object');

    pm.expect(responseData.id).to.be.a('number');

    pm.expect(responseData.id).to.be.at.least(0);

});

pm.**test**("Email is in a valid format", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData.email).to.**match**(/**^**\w**+**([\.-]**?**\w**+**)**\***@\w**+**([\.-]**?**\w**+**)**\***(\.\w**{2,3}**)**+$**/, "Email is not in a valid format");

});

pm.**test**("Gender should be either 'male' or 'female'", **function** () {

    const responseData **=** pm.response.json();

    pm.expect(responseData).to.be.an('object');

    pm.expect(responseData.gender).to.be.oneOf(['male', 'female']);

});

Demonstrate how Postman is used with Jenkins.

* Export the collection somewhere in your local machine and try running it using newman to check if everything is working fine.
* Push this collection to GitHub.
* Set up cloud machine by following the below steps –

**1) Setting up the cloud machine**

* Login into AWS Management console.
* Search for EC2 service. (Services > Compute > EC2)
* Click on Launch Instance Drop down > Select Launch Instance
* Provide a name for the new instance.
* Download key in .ppk (windows) or .pem(mac) format
* Click ‘Launch Instance’.
* Click ‘View All Instances’

**2) Connecting to the cloud machine**

For Mac users –

* Click on ‘Connect’ tab > SSH Client tab and then follow the instructions mentioned on the page. SSH Client means -> Teminal

For Window users –

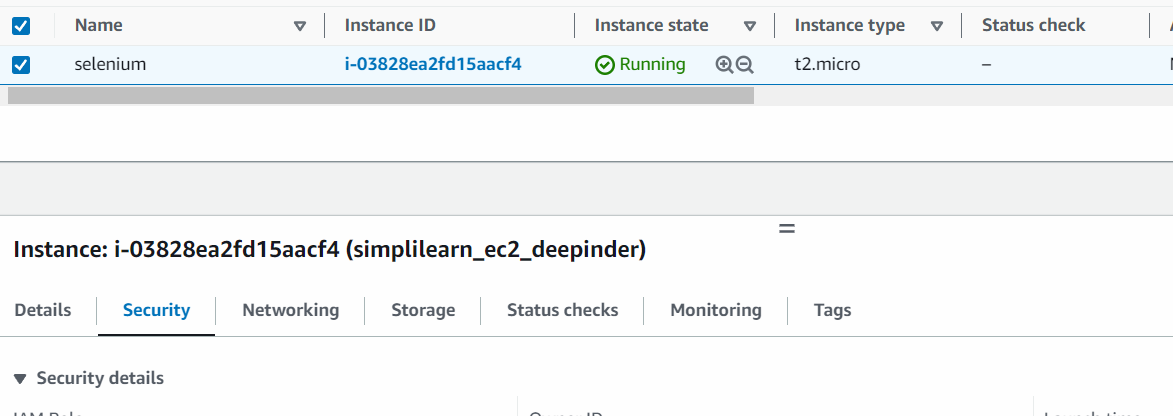
* Launch putty in your machine.
* Enter public ip address of cloud machine in putty.
* Expand SSH > Expand Auth > Click Credentials > Browse the .ppk file downloaded while creating the cloud machine.
* Click Open button.
* Accept the Security Alert, a black screen will appear.
* Type ‘ec2-user’ on the black screen and hit enter.

Run following commands using Putty on Cloud Machine

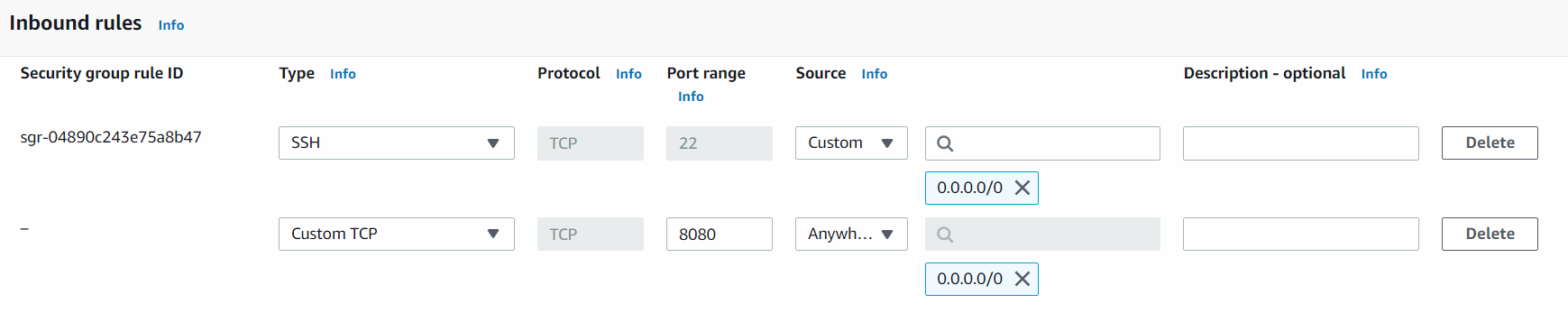
1. sudo yum update –y
2. sudo wget -O /etc/yum.repos.d/jenkins.repo \https://pkg.jenkins.io/redhat-stable/jenkins.repo
3. sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
4. sudo yum install jenkins java –y
5. sudo chkconfig --add jenkins
6. sudo systemctl start jenkins
7. sudo systemctl status jenkins

**Change Cloud machine’s security settings so that it can be accessed from any outer IP**

1. Click on ‘Security’ tab.

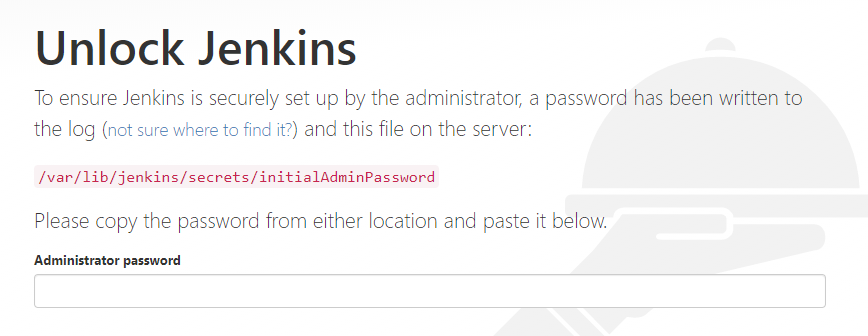


1. Click on alphanumeric string under ‘Security groups’.
2. On next page, click on ‘Edit Inbound rules’.
3. Click on ‘Add’ button on next page and select following settings –



1. Click ‘Save Rules’ button.
2. Go to ‘Details’ tab again > Copy Public IPv4 DNS.
3. Open any browser, paste the DNS from last step and type ‘:8080’ at the end and hit enter.

After executing above 7 steps, we have to configure Jenkins, for that we have to read a temp password stored at a location which is shown here –



To read the password from this location, write the commands –

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

* On the next screen, click ‘Install Suggested Plugins’.

Configure Jenkins

1. Go to Manage Jenkins > Manage Plugins > Available tab> Search for GitHub Integration and GitHub Authentication.

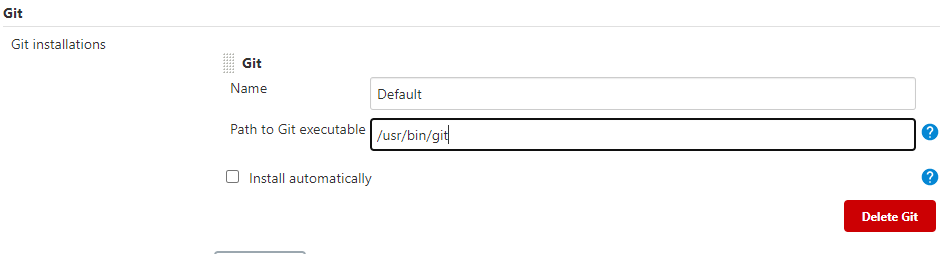
Now install git in AWS instance so that we can pull the code into it. Run the following commands on Putty –

a) sudo yum install git –y

Get the path of installation of git in AWS machine by writing in Putty

a) which git

1. Go to Manage Jenksin > Tools> Update the path as shown –

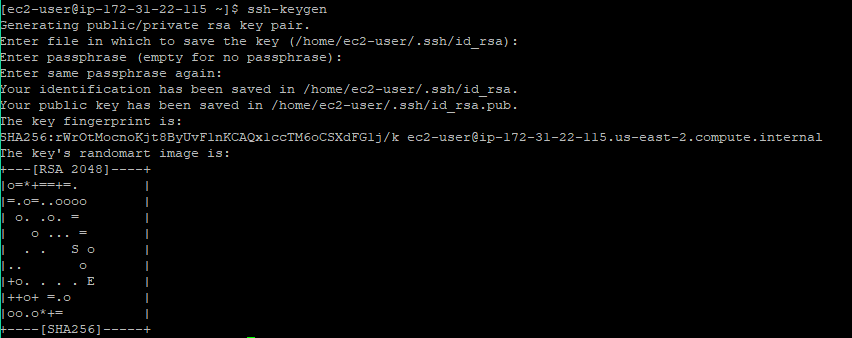


Add cloud computer as a Trusted Source in GitHub

1. Write in putty –

ssh-keygen

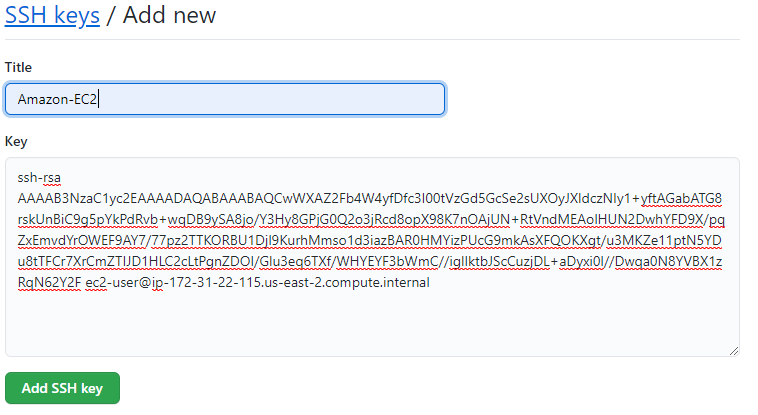
You will see following screen –



1. The key generated in the previous step is to be read from the location specified. For that write –

sudo cat /home/ec2-user/.ssh/id\_rsa.pub

1. A unique string will be printed on screen, copy that.
2. Go to GitHub and login with your credentials
3. Go to Settings > SSH and GPG Keys > New SSH Key > Paste the key copied in step 3 above> Add SSH Key



1. Go to Manage Jenkins > Tools > Under ‘JDK’ option, for ‘JDK installations’, provide any value to ‘Name’ field, check ‘install automatically’.

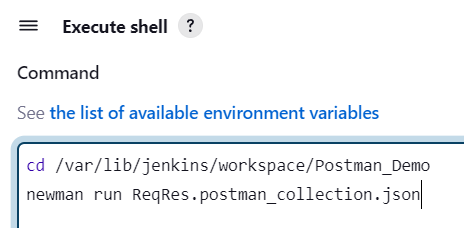
Postman test cases can be executed through Newman which is equivalent of running test cases through Command prompt. Here we will not be required to open Postman UI.

Installing Newman on Jenkins machine

1. sudo yum install -y gcc-c++ make
2. curl -sL <https://rpm.nodesource.com/setup_12.x> | sudo -E bash -
3. sudo yum install -y nodejs
4. sudo npm install -g newman

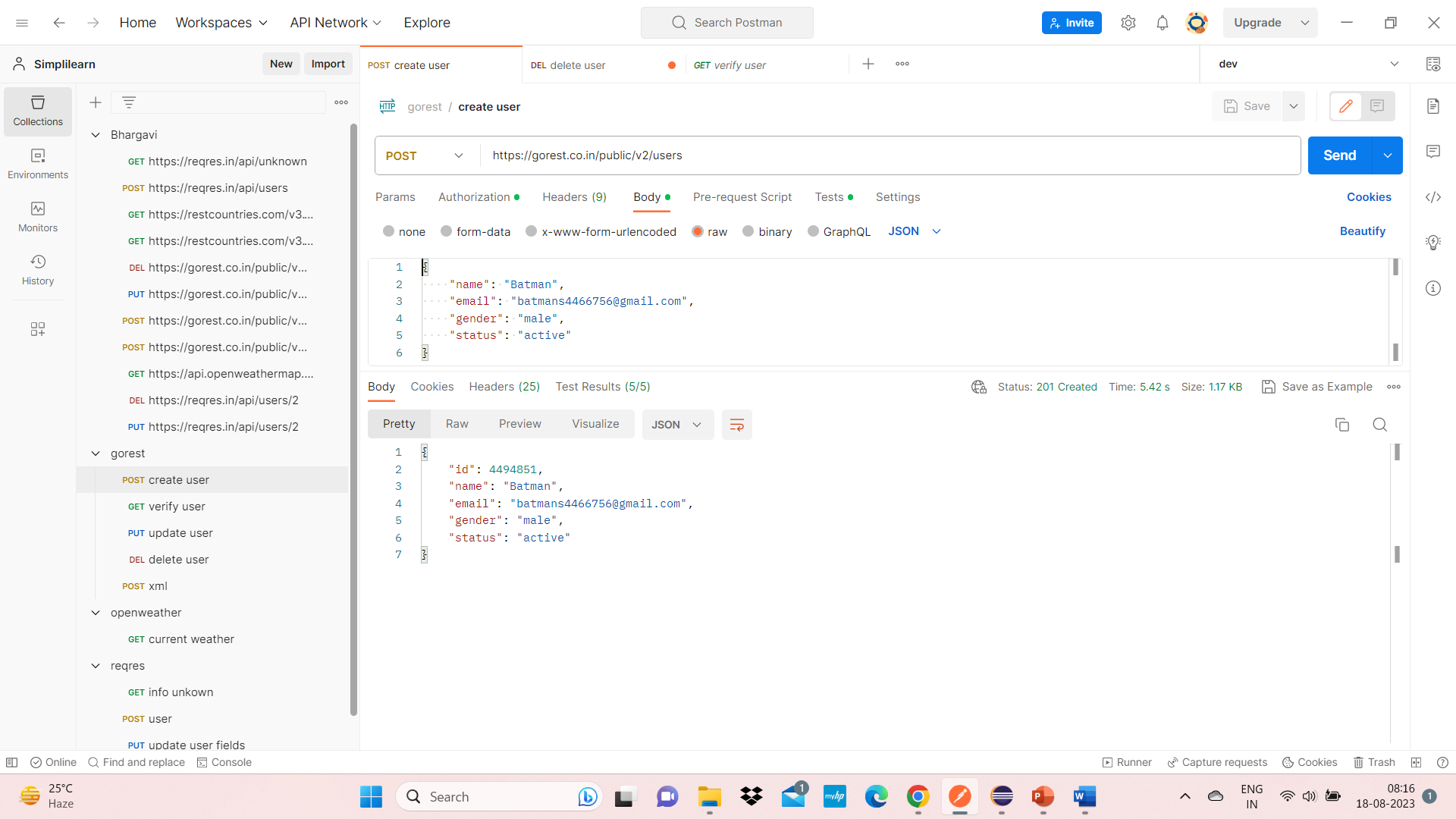
**Executing test case on Cloud Machine**

* Login into Jenkins > Go to Dashboard > New Item > Name the project > Choose FreeStyle Project > Ok
* On the next page Select GitHub project > Enter project URL
* Scroll Down to ‘Source Code Management’ > Git > Enter Repository URL
* Enter your GitHub username and Password Access Token
* Go to ‘Branches to Build’ > Update it to ‘\*/main’
* Click Save and go to the Job page.
* Click ‘Build Now’, Jenkins will pull the file from Github and the job will get executed.
* Click on Console Output of the executed job > Get the location of directory in which the JSON file from Github has been pulled (/var/lib/jenkins/workspace/Postman\_Demo).
* Go to Configue page of Job > ‘Build Steps’ > Choose ‘Execute Shell’ from drop-down > Write



* Click Save
* Click Build Now and execute the job.

Demonstrate how to get data from CSV and JSON.



Demonstrate how to run a collection remotely with URL.

