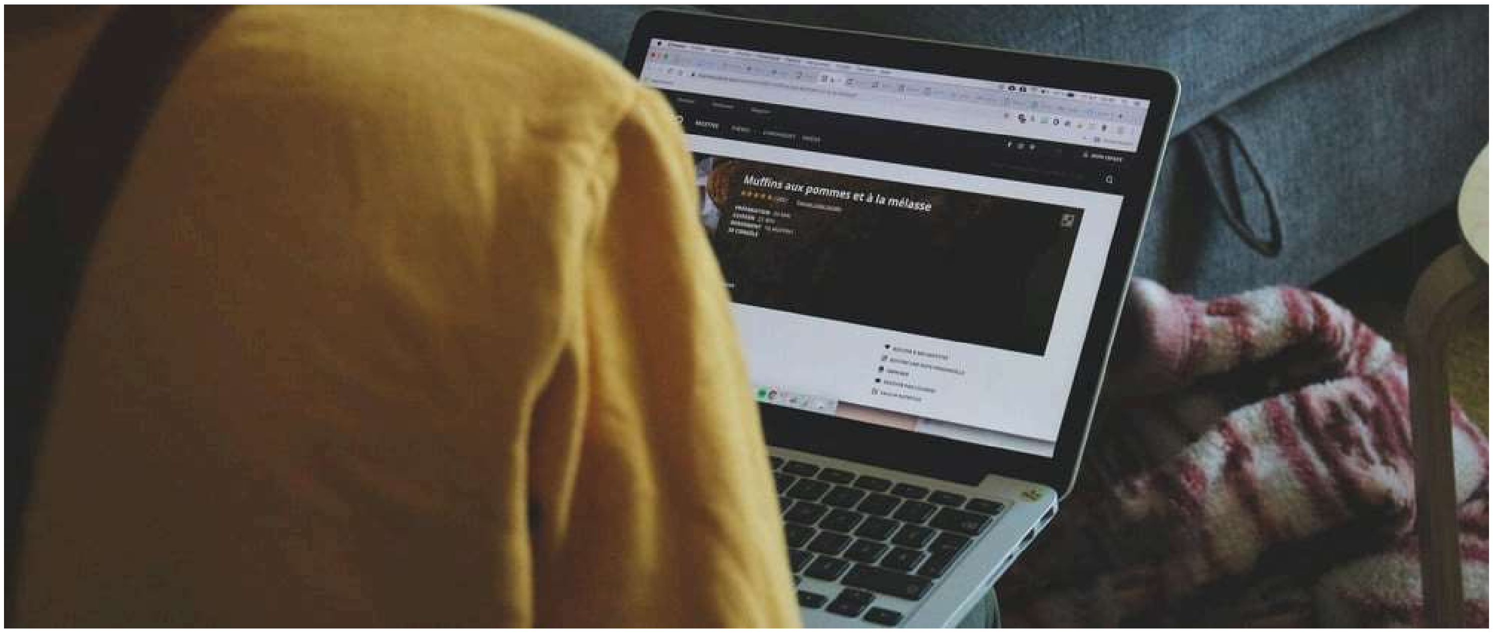


Raphael Jambalos
Posted on 15 abr 2021

18 1

Simulate Real-World User Behavior in Apache JMeter Load Tests

#devops #jmeter #tutorial #aws

Load test your website with Apache JMeter (3 Part Series)

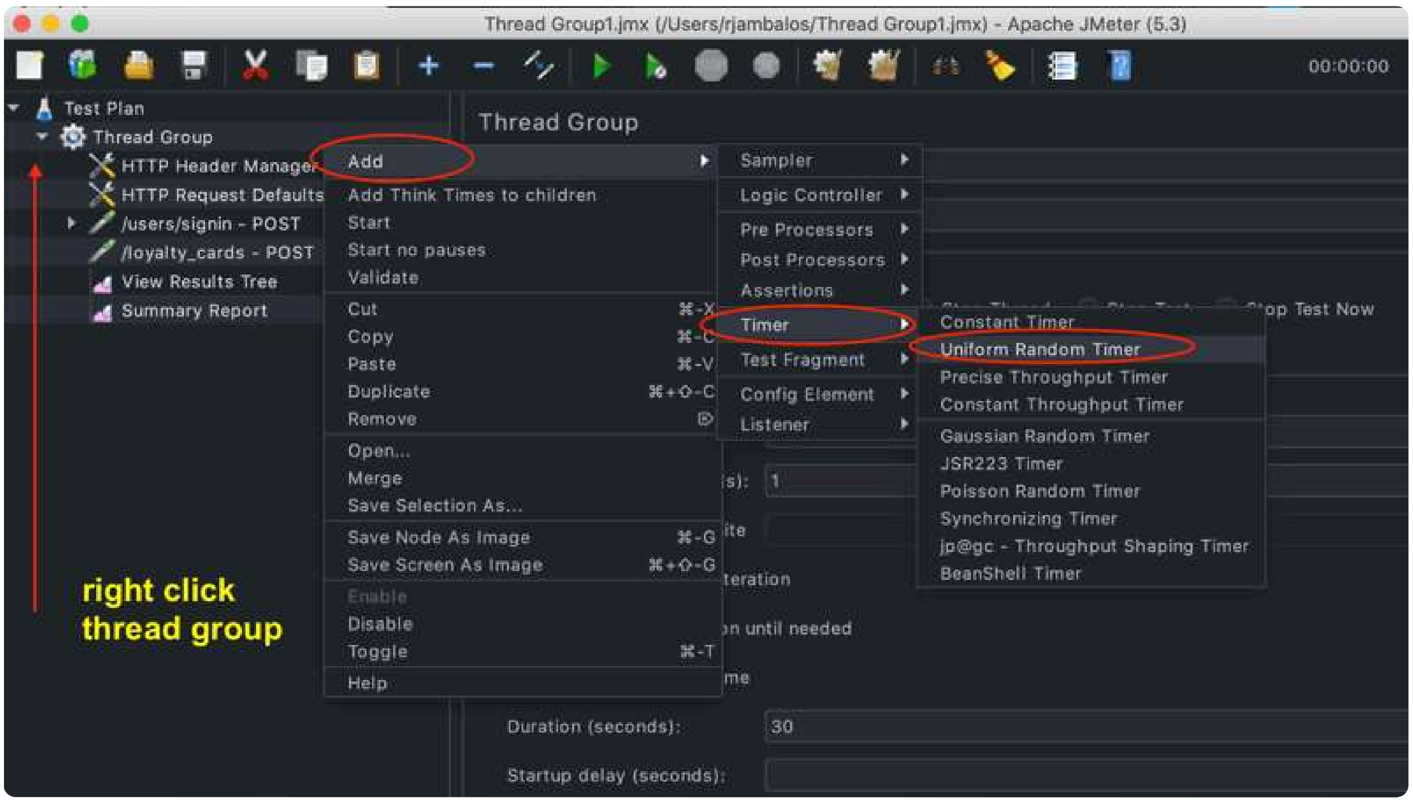
A realistic load test helps you prepare your application and infrastructure adequately for the surge of people that will browsing and using your site. By configuring your test plan to mimic real-world user behavior, the results you get from the load test can help inform design and capacity decisions you will make to keep your site robust.

In the previous post, we created a basic Jmeter Test Plan for a 3-endpoint API written in Flask and Python. That's a good start. But to create a Test Plan that closely resembles real user behavior, we have to add more elements to our test plan.

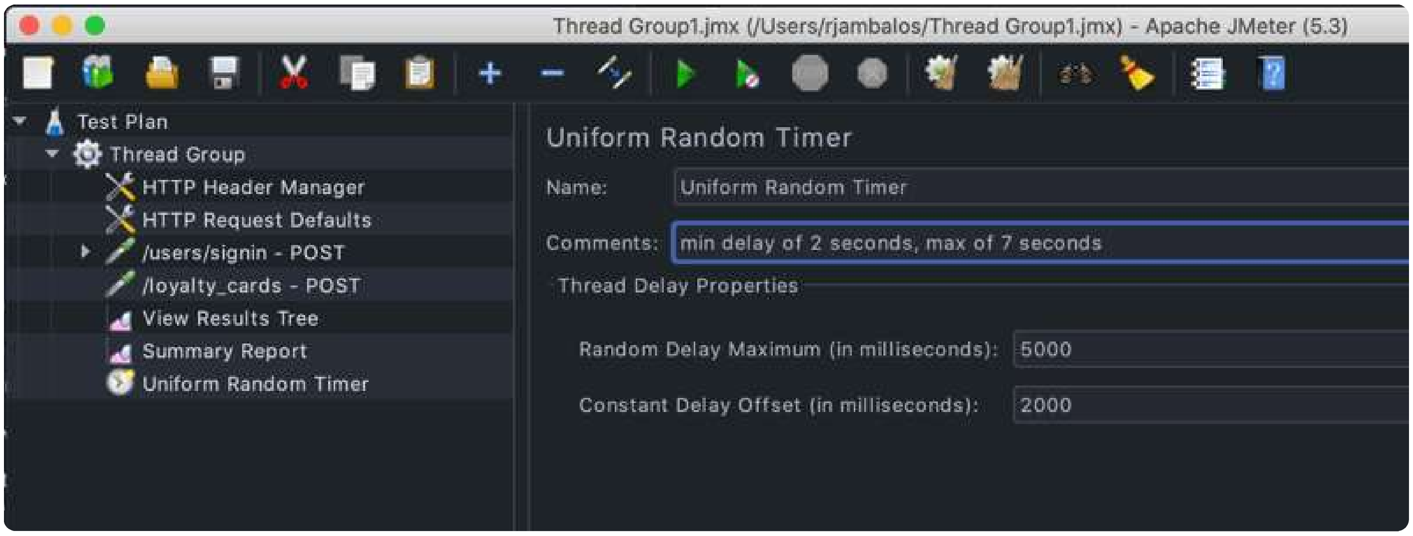
If you want to follow along without going through the previous post, you can pick up right where we left off from the previous blog post by downloading this JMeter test plan. Also, you will have to set up the Flask application, so you will have the same application we are load testing against. Setup instructions are found in the README.me file of this repository.

[1] Add Thinking Time with the "Uniform Random Timer"

When users browse your website, they look around the page first for a few seconds before moving on to the next page. They click the "View Product" page, take at least a few seconds to look at the product, and then add it to the cart. To simulate this thinking time, we add the Uniform Random Timer to our load tests:



Once created, let's configure a minimum delay of 2secs and a maximum delay of 7 seconds. JMeter chooses a random value between 2s-7s for the delay. You may want to adjust this value to reflect better how users actually use your site.


Thread Group1.jmx (/Users/rjambalos/Thread Group1.jmx) - Apache JMeter (5.3)

- Test Plan
  - Thread Group
    - HTTP Header Manager
    - HTTP Request Defaults
    - /users/signin - POST
    - /loyalty_cards - POST
    - View Results Tree
    - Summary Report
    - Uniform Random Timer

Uniform Random Timer

Name: Uniform Random Timer

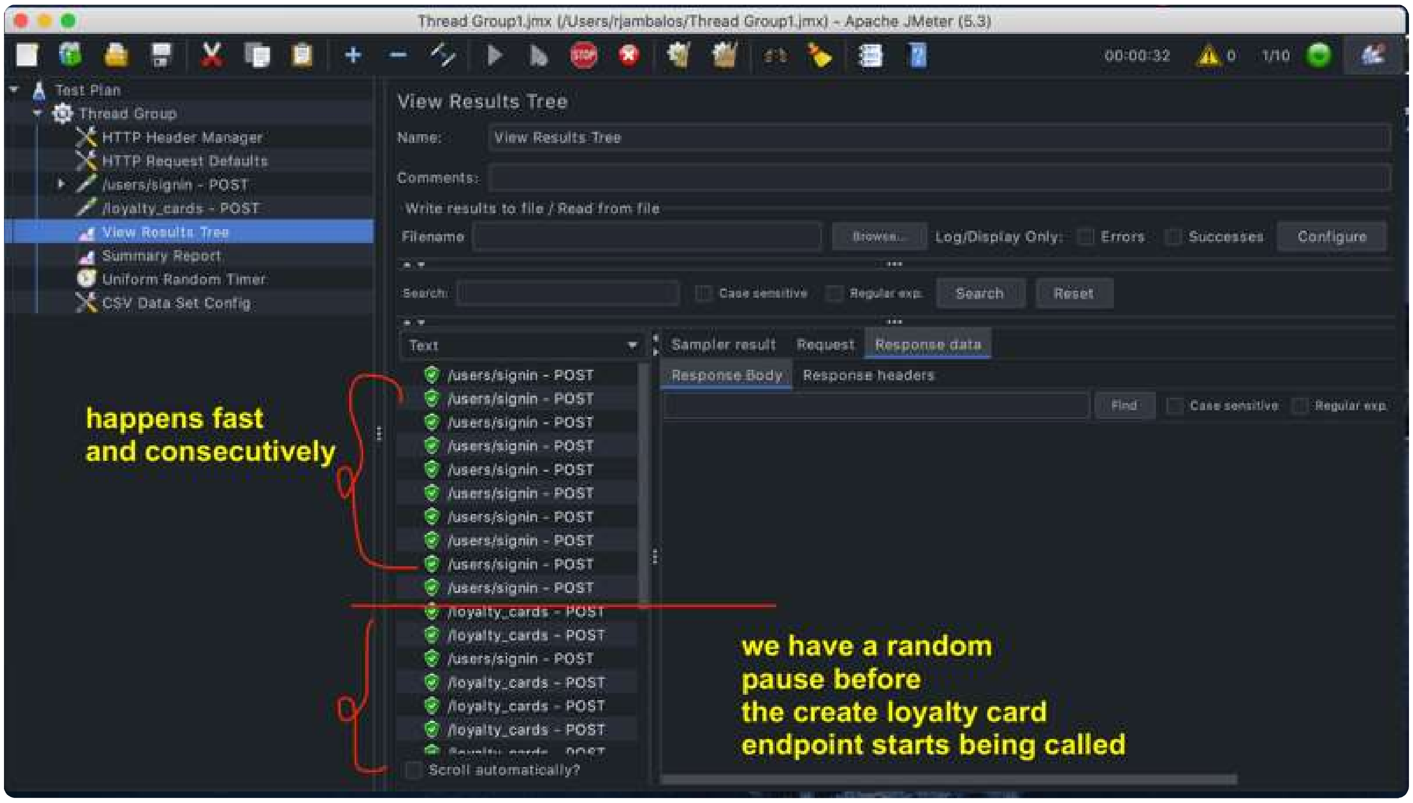
Comments: min delay of 2 seconds, max of 7 seconds

Thread Delay Properties

Random Delay Maximum (in milliseconds): 5000

Constant Delay Offset (in milliseconds): 2000

Now, let's run the load test. You'd see it takes a bit longer than before because of the delays introduced. This means we have successfully emulated user thinking time.


Thread Group1.jmx (/Users/rjambalos/Thread Group1.jmx) - Apache JMeter (5.3)

00:00:32 0 1/10

- Test Plan
  - Thread Group
    - HTTP Header Manager
    - HTTP Request Defaults
    - /users/signin - POST
    - /loyalty_cards - POST
    - View Results Tree
    - Summary Report
    - Uniform Random Timer
    - CSV Data Set Config

View Results Tree

Name: View Results Tree

Comments:

Write results to file / Read from file

Filename Browse... Log/Display Only: Errors Successes Configure

Search: Case sensitive Regular exp. Search Reset

Text

/users/signin - POST
/users/signin - POST
/users/signin - POST
/users/signin - POST
/users/signin - POST
/users/signin - POST
/users/signin - POST
/users/signin - POST
/users/signin - POST
/users/signin - POST
/loyalty_cards - POST
/loyalty_cards - POST
/users/signin - POST
/loyalty_cards - POST
/loyalty_cards - POST
/loyalty_cards - POST

Scroll automatically?

Sampler result Request Response data

Response Body Response headers

Find Case sensitive Regular exp.

happens fast and consecutively
we have a random pause before the create loyalty card endpoint starts being called

[2] Vary user input with the "CSV Data Input"

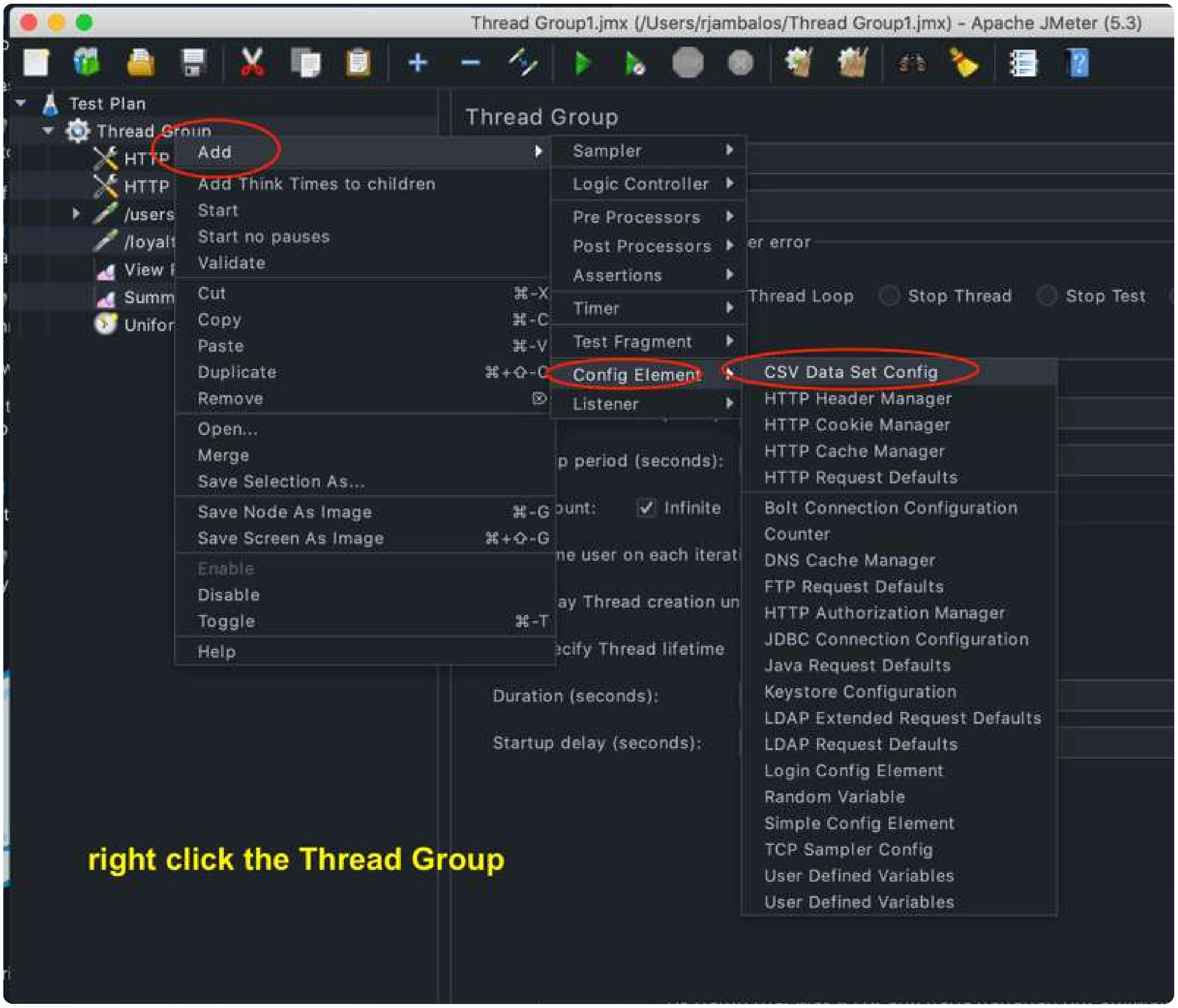
Users type in different sign-in credentials when they browse your site. They also buy different products. This puts a strain on your system because the response is more difficult to cache.

If we have a test plan with 20 virtual users (VUs) logging in using the same email and password and buying the same product, our performance test results become too optimistic. Our database or cache might have been optimized already to keep returning the same results.

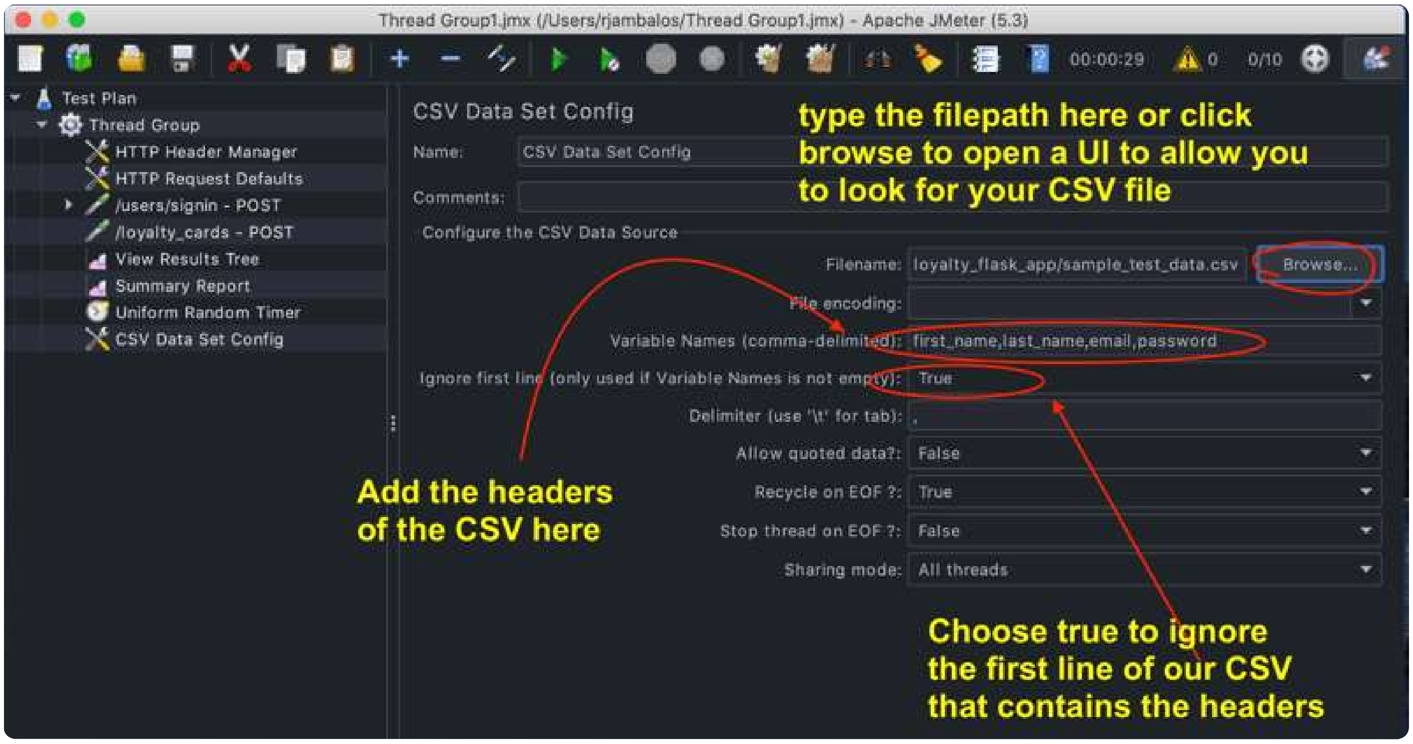
To fix this, we introduce the "CSV Data Input" element into our test plan. First, let's define the CSV file from which we will get the values. Save this CSV file with a filename of "sample_data.csv". Place it in the same directory as we saved the JMeter load test we are working with.

```
first_name,last_name,email,password
raphael,jambalos,raphael.jambalos@gmail.com,jambyiscool
test,jambalos,test.driver@gmail.com,driveiscool
ultra,jambalos,ultra.player@gmail.com,ultraplayeriscool
super,jambalos,super.user@gmail.com,superuseriscool
code,jambalos,code.maestro@gmail.com,codemaestroiscool
```

Then, let's use this CSV file in the load test by creating the "CSV Data Set Config" element.



In the filename part, find the CSV file you just created. You could type `sample_data.csv` if you followed my earlier instruction of just placing the CSV file in the same directory as the JMeter test plan. If the file is stored elsewhere, click Browse to find the file.


Thread Group1.jmx (/Users/rjambalos/Thread Group1.jmx) - Apache JMeter (5.3)
Test Plan
Thread Group
HTTP Header Manager
HTTP Request Defaults
/users/signin - POST
/loyalty_cards - POST
View Results Tree
Summary Report
Uniform Random Timer
CSV Data Set Config
type the filepath here or click browse to open a UI to allow you to look for your CSV file
Add the headers of the CSV here
Choose true to ignore the first line of our CSV that contains the headers

CSV Data Set Config

Name: CSV Data Set Config

Comments:

Configure the CSV Data Source

Filename: loyalty_flask_app/sample_test_data.csv Browse...

File encoding:

Variable Names (comma-delimited): first_name,last_name,email,password

Ignore first line (only used if Variable Names is not empty): True

Delimiter (use '\t' for tab): ,

Allow quoted data?: False

Recycle on EOF ?: True

Stop thread on EOF ?: False

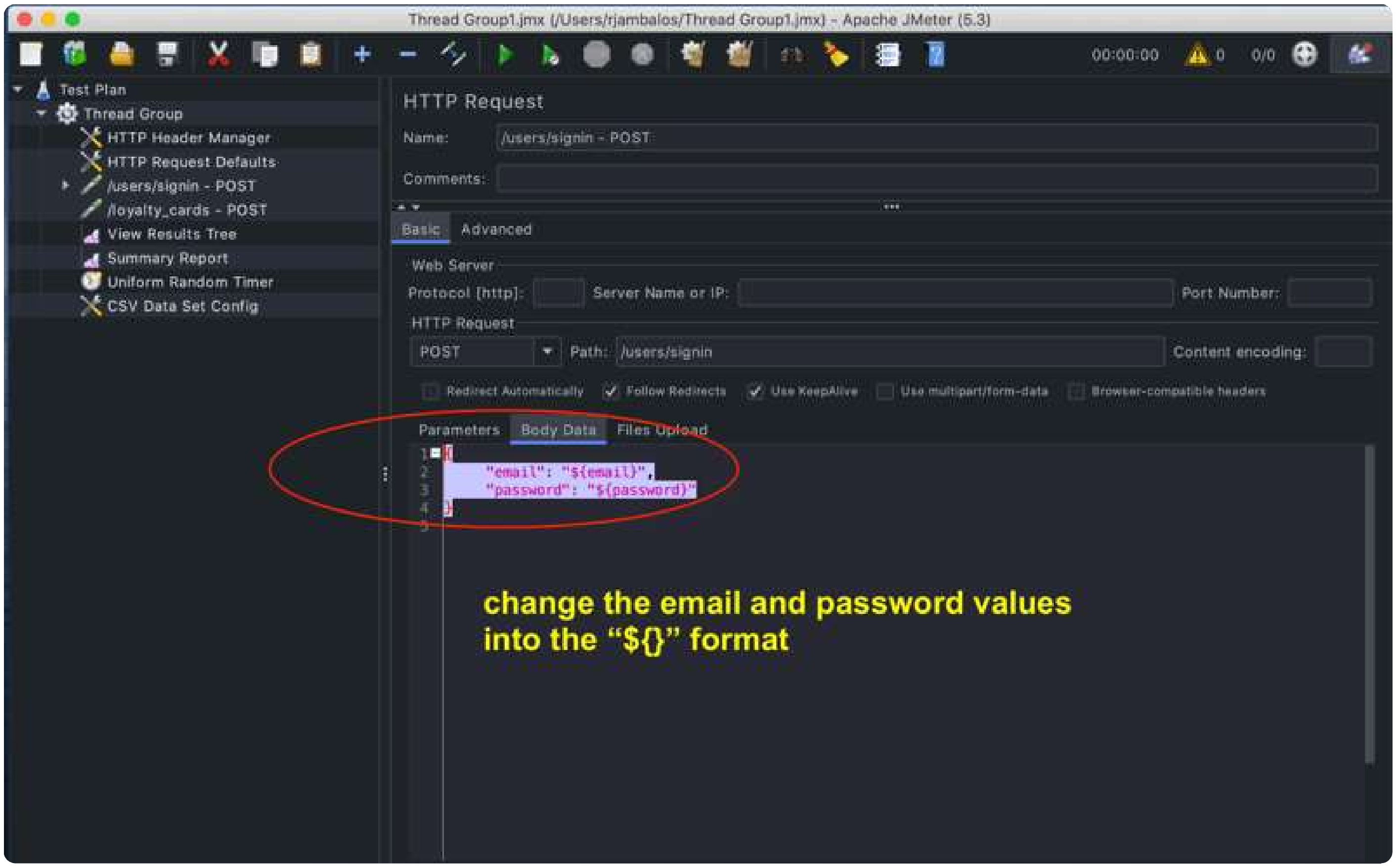
Sharing mode: All threads

For each virtual user, the CSV Data Set Config gets one row from the CSV and loads it as values to the variable names you specified in the element. For example, VU#1 will have the row "ultra,jambalos,ultra.player@gmail.com,ultraplayeriscool" and the variable names available to that user will be:

- first_name: ultra
- last_name: jambalos
- email: ultra.player@gmail.com
- password: ultraplayeriscool

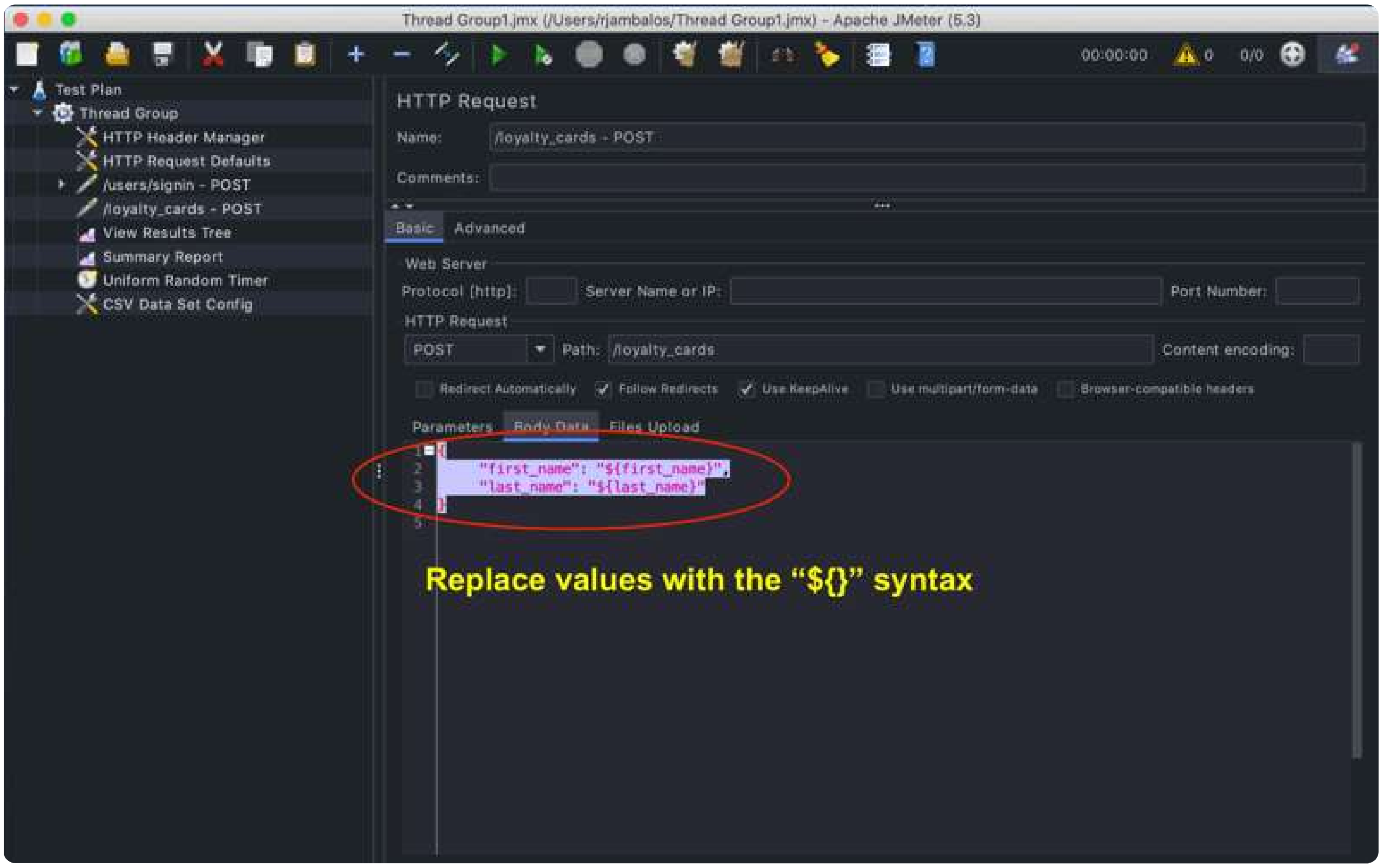
VU#2 will have a different row and hence have a different set of values for the four variables specified above. In order to have different values for each VU, we need to review the 2 endpoints we created in the previous post and replace some value fields with variable names instead of hardcoded data.

For the sign-in route, replace the value of the email and password keys with these values: "${email}" and "${password}".



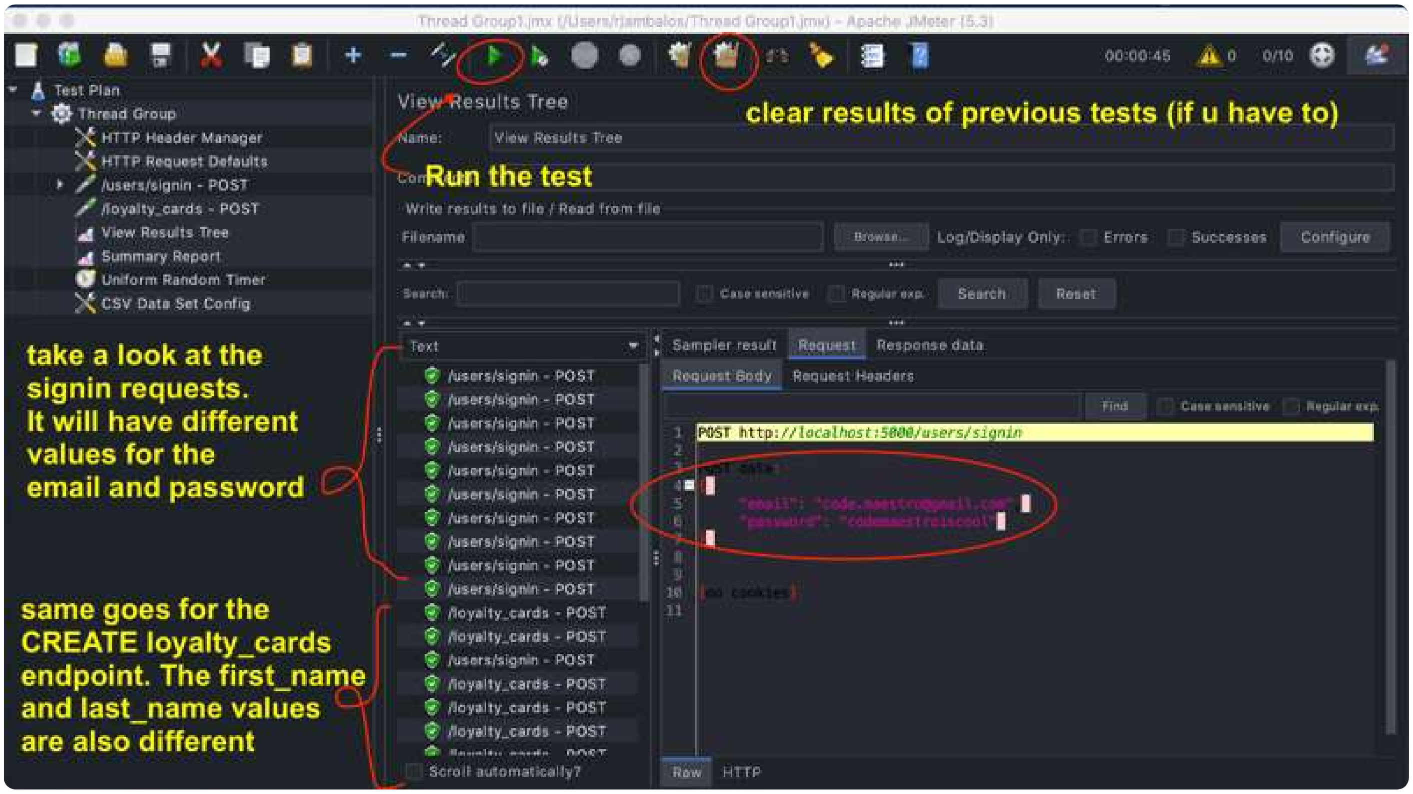
```
{
    "email": "${email}",
    "password": "${password}"
}
```

For the route loyalty cards create, do the same for first name and last name:



```
{
    "first_name": "${first_name}",
    "last_name": "${last_name}"
}
```

Now, running the load test, we see different values used for email, password, first_name, and last_name



clear results of previous tests (if u have to)

Run the test

take a look at the signin requests. It will have different values for the email and password

same goes for the CREATE loyalty_cards endpoint. The first_name and last_name values are also different

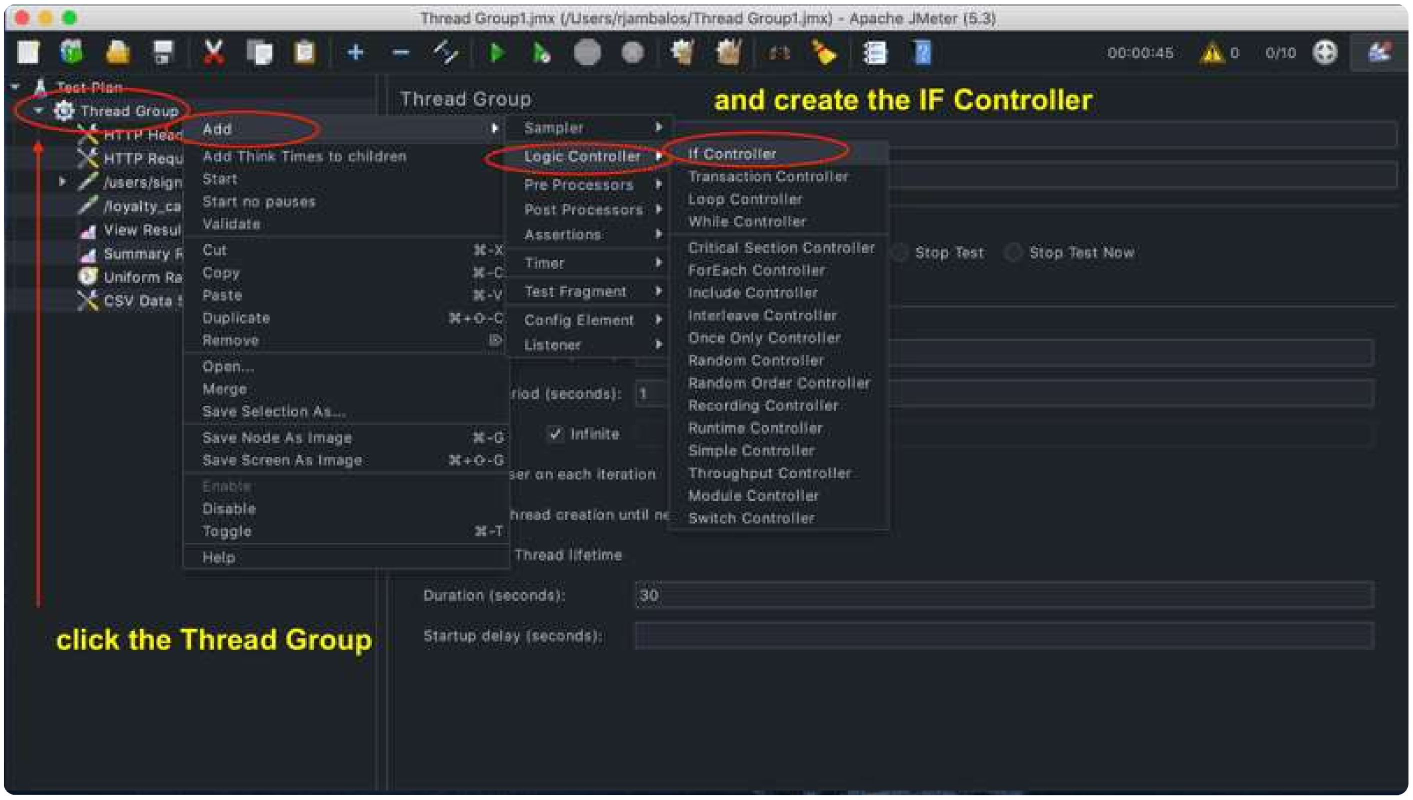
```
/users/signin - POST
/users/signin - POST
/users/signin - POST
/users/signin - POST
/users/signin - POST
/users/signin - POST
/users/signin - POST
/users/signin - POST
/users/signin - POST
/users/signin - POST
/loyalty_cards - POST
/loyalty_cards - POST
/users/signin - POST
/loyalty_cards - POST
/loyalty_cards - POST
/loyalty_cards - POST
```

```
POST http://localhost:5000/users/signin

POST data:
{
    "email": "code.maestro@gmail.com",
    "password": "codemaestroiscool"
}


[no cookies]
```

[3] Add conditions for gateway actions

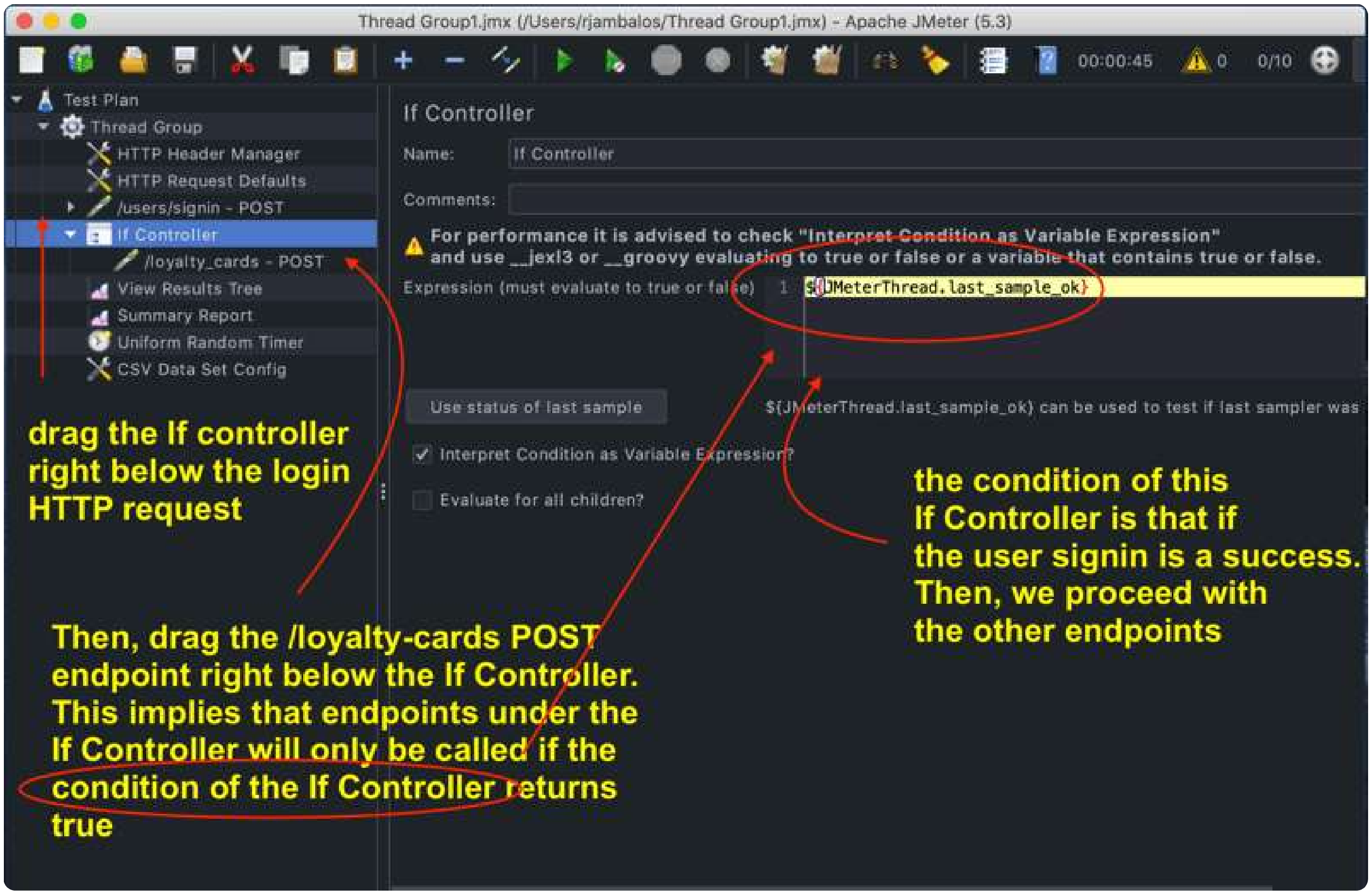
When the call to the login endpoint fails, the actions after it also fails. If we have 10 endpoints to call after login, a failed login overstates an error 10 times over. Also, real-world users retry the login rather than trying pages that are inaccessible to them.

The solution to this problem is to ensure that each VU only tries the 10 endpoints *if the login succeeded*. For this, we create the If Controller:



Now, drag the If Controller right below the sign-in endpoint and make sure to "drag in" the /loyalty-cards POST endpoint. This makes sure that the endpoint will only execute if the login was a success. Make sure to set the condition of the if config element to:

```
${JMeterThread.last_sample_ok}
```



For performance it is advised to check "Interpret Condition as Variable Expression" and use __jexl3 or __groovy evaluating to true or false or a variable that contains true or false.

Expression (must evaluate to true or false) `${JMeterThread.last_sample_ok}`

${JMeterThread.last_sample_ok} can be used to test if last sampler was

drag the If controller right below the login HTTP request

Then, drag the /loyalty-cards POST endpoint right below the If Controller. This implies that endpoints under the If Controller will only be called if the condition of the If Controller returns true

the condition of this If Controller is that if the user signin is a success. Then, we proceed with the other endpoints

[4] Set up a test environment similar to the production

Up to this point, our load test is configured to allow us to test quickly so we can get feedback on what to improve. This is why our load test only runs for 30secs and has only 10 VUs. At this volume, we are sure that even if you host your application locally, the load will be adequately handled.

But load tests are not of much value if we test at low volumes. After all, load tests answer the question: "how many users making requests concurrently can my website handle?". To have a load test that truly adds value to our team, we need to test at higher volumes (think 100-500 VU or more). And to test at higher volumes, we need to set up a test environment that can accommodate that load.

In setting up the test environment, we need to make it similar to our production setup:

- The database and the application servers are of the same size and count.
- The application and web servers have the same configuration as the production setup
- The data in the database is roughly the same size as in production.

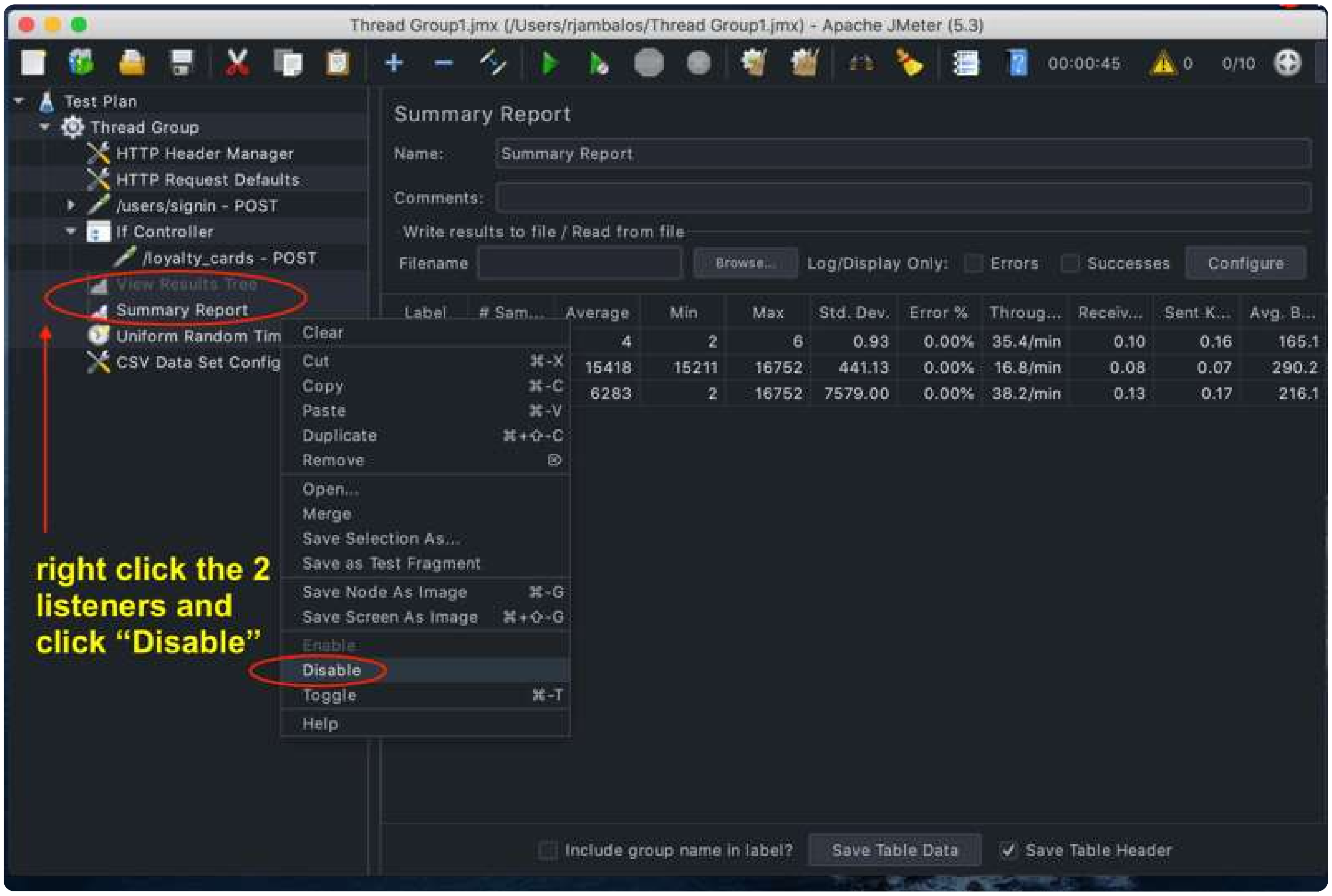
For some teams, replicating the database and server fleet (i.e., 10 r5.large EC2 instances and 1 24.xlarge database) won't be possible. For this, what we usually do is a scaled-down version of the app (i.e., think 2 r5.large EC2 instance and 4xlarge database). Then, we multiply the capacity by the scaling factor to get the estimated capacity of production.

For our example, the scaling factor is 5. Our test environment is 1/5 the size of our production setup. If our test environment can accommodate 500 users, we assume the production setup can accommodate 2500 users.

[5] Disable Listeners during high-volume load tests

Thus far, we had the "View Results Tree" and "Summary Report" listeners to help us build and improve our load test. But when testing on high volumes, these 2 listeners suck up a lot of resources (particularly memory) that we could have used for the load test operation.

When you are ready for your high-volume test, disable the two listeners. Once you do, it should be grayed out in the GUI:



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename [Browse...] Log/Display Only: ☐ Errors ☐ Successes [Configure]

Label	# Sam...	Average	Min	Max	Std. Dev.	Error %	Throug...	Receiv...	Sent K...	Avg. B...
		4	2	6	0.93	0.00%	35.4/min	0.10	0.16	165.1
		15418	15211	16752	441.13	0.00%	16.8/min	0.08	0.07	290.2
		6283	2	16752	7579.00	0.00%	38.2/min	0.13	0.17	216.1

☐ Include group name in label? [Save Table Data] ✓ Save Table Header

[6] Run Jmeter in CLI mode

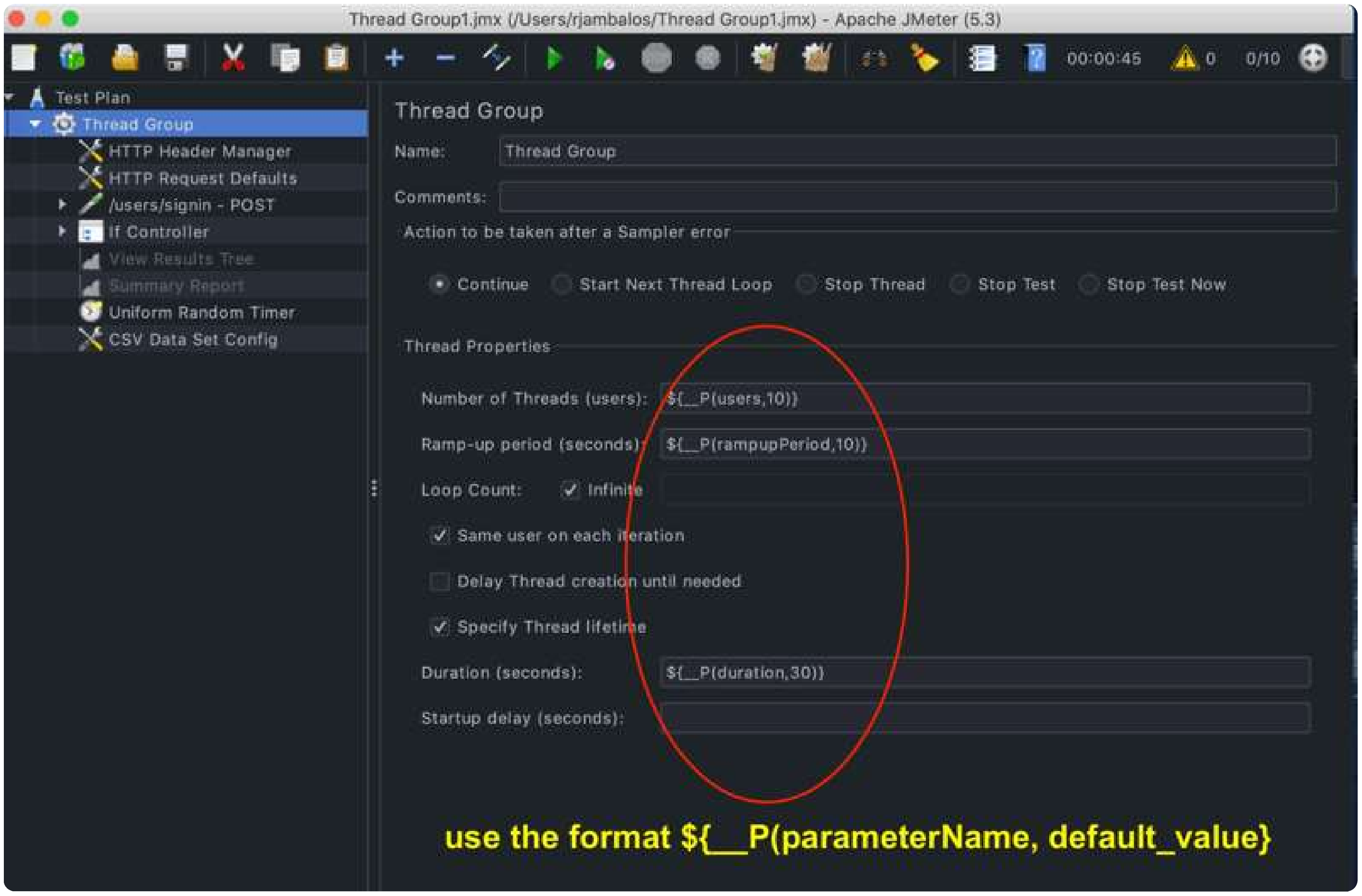
When load testing, we usually change the parameters of the load test. I usually start at 100 VU. If the app can handle it, I push it up to 200 VU, then 300 VU, etc.

At the current state of our load test, if we use the CLI, we would have to modify the test plan locally with the GUI, then re-upload it to get from 100VU to 200VU. There has to be a better way!

For this, we introduce CLI parameters. Click on the thread group and replace the values of the fields with the following values species below. Notice that the 1st parameter sets the variable name and the second parameter sets the value of this variable in case there is none provided in the CLI command.

- Number of Threads: `${__P(users,10)}`
- Ramp-up period: `${__P(rampupPeriod,10)}`
- Duration: `${__P(duration,30)}`

Your Thread Group should look like this:



Thread Group

Name: Thread Group

Comments:

Action to be taken after a Sampler error

[✓ Continue] Start Next Thread Loop Stop Thread Stop Test Stop Test Now

Thread Properties

Number of Threads (users): ${__P(users,10)}

Ramp-up period (seconds): ${__P(rampupPeriod,10)}

Loop Count: ✓ Infinite

✓ Same user on each iteration

Delay Thread creation until needed

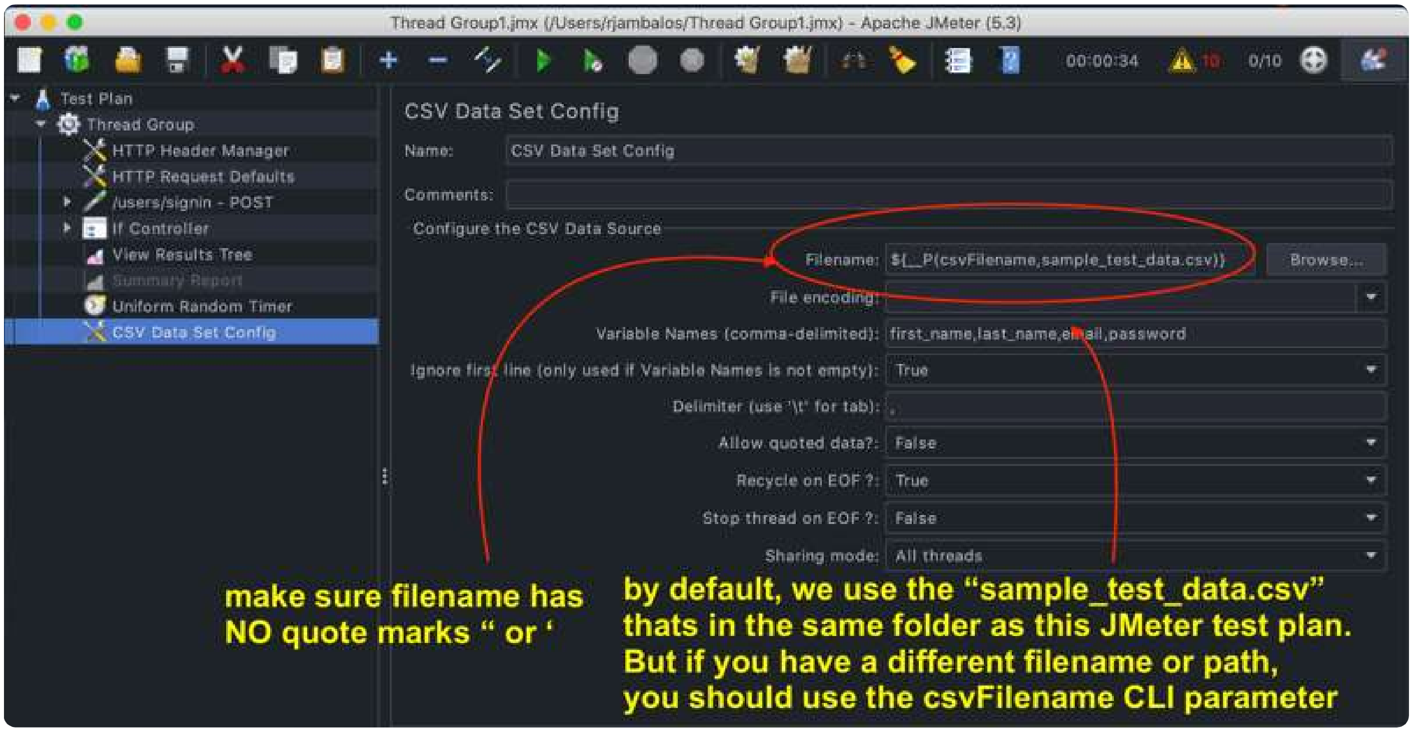
✓ Specify Thread lifetime

Duration (seconds): ${__P(duration,30)}

Startup delay (seconds):

use the format ${__P(parameterName, default_value}

We also modify our CSV Data Set Config to parameterize the CSV filename using `${__P(csvFilename,'sample_test_data.csv)}`.


Thread Group1.jmx (/Users/rjambalos/Thread Group1.jmx) - Apache JMeter (5.3)

- Test Plan
  - Thread Group
    - HTTP Header Manager
    - HTTP Request Defaults
    - /users/signin - POST
    - If Controller
    - View Results Tree
    - Summary Report
    - Uniform Random Timer
    - CSV Data Set Config

CSV Data Set Config

Name: CSV Data Set Config

Comments:

Configure the CSV Data Source

Filename: ${__P(csvFilename,sample_test_data.csv)}

File encoding:

Variable Names (comma-delimited): first_name,last_name,email,password

Ignore first line (only used if Variable Names is not empty): True

Delimiter (use '\t' for tab): ,

Allow quoted data?: False

Recycle on EOF ?: True

Stop thread on EOF ?: False

Sharing mode: All threads

make sure filename has NO quote marks " or '

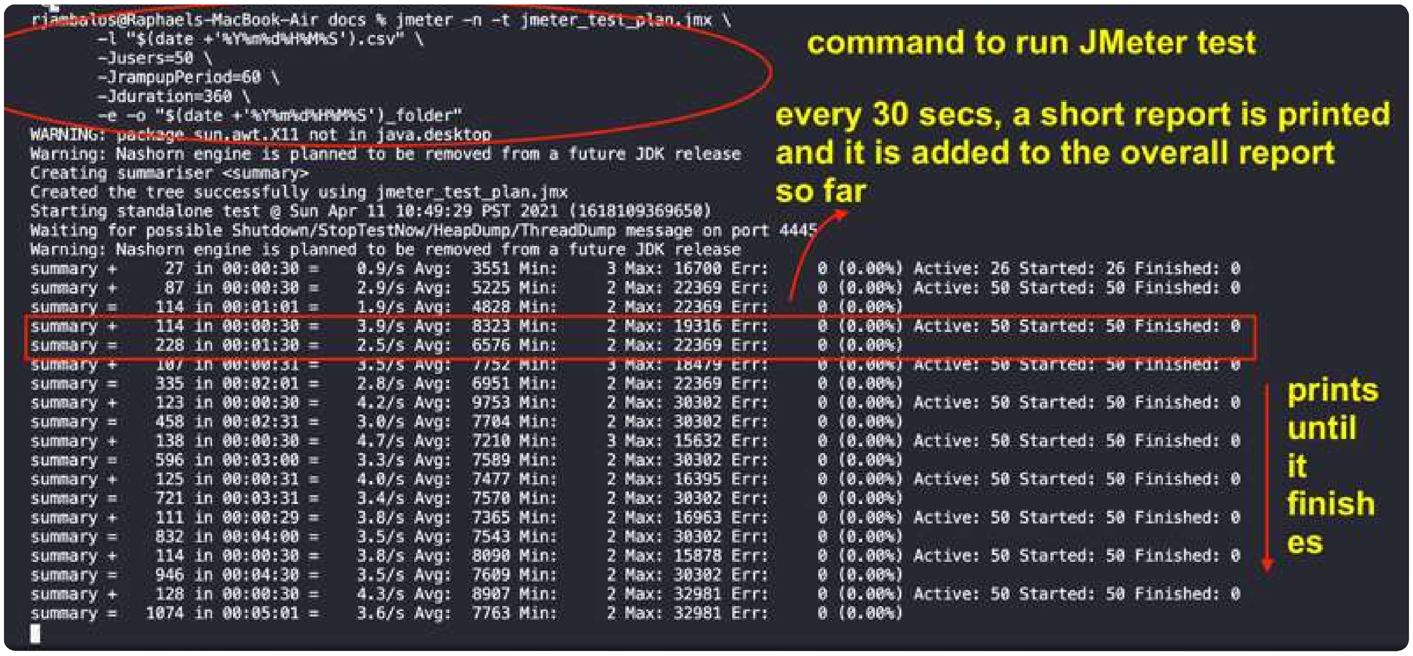
by default, we use the "sample_test_data.csv" thats in the same folder as this JMeter test plan. But if you have a different filename or path, you should use the csvFilename CLI parameter

Now, it's time to run our load test via CLI. Notice that we intentionally did not include the `csvFilename` parameter and instead relied on its default value.

```
jmeter -n -t jmeter_test_plan.jmx \
        -l "$(date +'%Y%m%d%H%M%S').csv" \
        -Jusers=50 \
        -JrampupPeriod=60 \
        -Jduration=360 \
        -e -o "$(date +'%Y%m%d%H%M%S')_folder"
```

We specified a load test using 50 Virtual Users (VUs), a duration of 6 minutes (360s), and a ramp-up period of 1 minute. This is definitely more high volume than our 10VU / 30s load testing we have done prior. We also specified the JMeter test plan file that we have been working on. We also specified the paths on where to save the output CSV file and the HTML report (you will see both of this later).

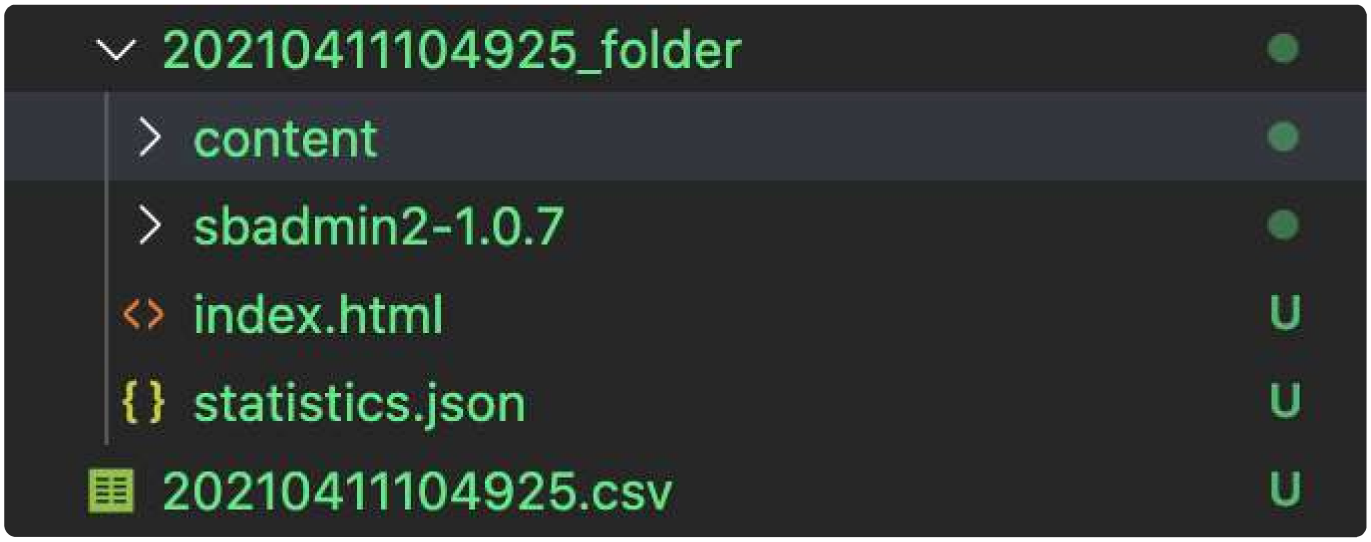
Once you run the test, your CLI should look like this.



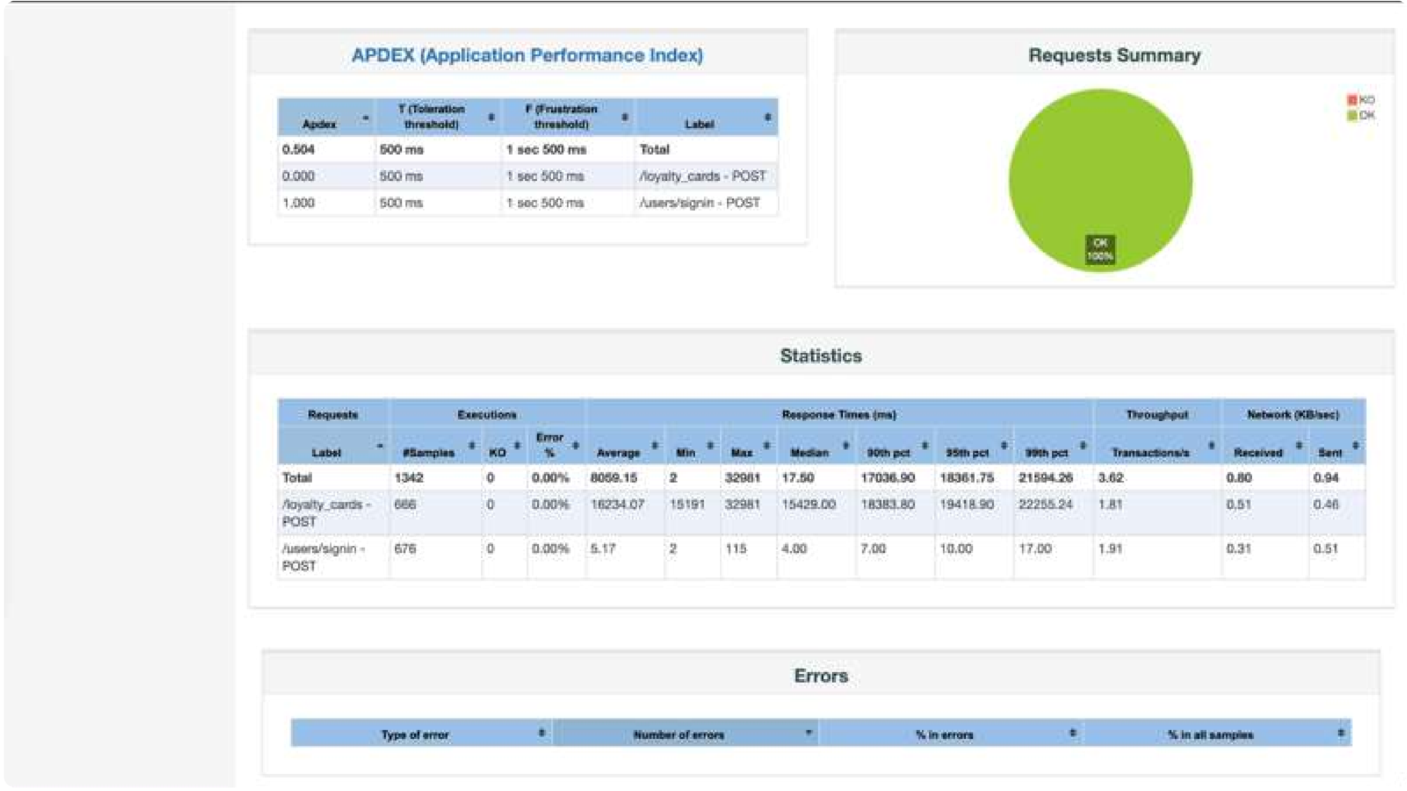
```
rjambalos@Raphaels-MacBook-Air docs % jmeter -n -t jmeter_test_plan.jmx \
        -l "$(date +'%Y%m%d%H%M%S').csv" \
        -Jusers=50 \
        -JrampupPeriod=60 \
        -Jduration=360 \
        -e -o "$(date +'%Y%m%d%H%M%S')_folder"
WARNING: package sun.awt.X11 not in java.desktop
Warning: Nashorn engine is planned to be removed from a future JDK release
Creating summariser <summary>
Created the tree successfully using jmeter_test_plan.jmx
Starting standalone test @ Sun Apr 11 10:49:29 PST 2021 (1618109369650)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
Warning: Nashorn engine is planned to be removed from a future JDK release
summary +     27 in 00:00:30 =    0.9/s Avg:  3551 Min:     3 Max: 16700 Err:     0 (0.00%) Active: 26 Started: 26 Finished: 0
summary +     87 in 00:00:30 =    2.9/s Avg:  5225 Min:     2 Max: 22369 Err:     0 (0.00%) Active: 50 Started: 50 Finished: 0
summary =    114 in 00:01:01 =    1.9/s Avg:  4828 Min:     2 Max: 22369 Err:     0 (0.00%)
summary +    114 in 00:00:30 =    3.9/s Avg:  8323 Min:     2 Max: 19316 Err:     0 (0.00%) Active: 50 Started: 50 Finished: 0
summary =    228 in 00:01:30 =    2.5/s Avg:  6576 Min:     2 Max: 22369 Err:     0 (0.00%)
summary +    107 in 00:00:31 =    3.5/s Avg:  7752 Min:     3 Max: 18479 Err:     0 (0.00%) Active: 50 Started: 50 Finished: 0
summary =    335 in 00:02:01 =    2.8/s Avg:  6951 Min:     2 Max: 22369 Err:     0 (0.00%)
summary +    123 in 00:00:30 =    4.2/s Avg:  9753 Min:     2 Max: 30302 Err:     0 (0.00%) Active: 50 Started: 50 Finished: 0
summary =    458 in 00:02:31 =    3.0/s Avg:  7704 Min:     2 Max: 30302 Err:     0 (0.00%)
summary +    138 in 00:00:30 =    4.7/s Avg:  7210 Min:     3 Max: 15632 Err:     0 (0.00%) Active: 50 Started: 50 Finished: 0
summary =    596 in 00:03:00 =    3.3/s Avg:  7589 Min:     2 Max: 30302 Err:     0 (0.00%)
summary +    125 in 00:00:31 =    4.0/s Avg:  7477 Min:     2 Max: 16395 Err:     0 (0.00%) Active: 50 Started: 50 Finished: 0
summary =    721 in 00:03:31 =    3.4/s Avg:  7570 Min:     2 Max: 30302 Err:     0 (0.00%)
summary +    111 in 00:00:29 =    3.8/s Avg:  7365 Min:     2 Max: 16963 Err:     0 (0.00%) Active: 50 Started: 50 Finished: 0
summary =    832 in 00:04:00 =    3.5/s Avg:  7543 Min:     2 Max: 30302 Err:     0 (0.00%)
summary +    114 in 00:00:30 =    3.8/s Avg:  8090 Min:     2 Max: 15878 Err:     0 (0.00%) Active: 50 Started: 50 Finished: 0
summary =    946 in 00:04:30 =    3.5/s Avg:  7609 Min:     2 Max: 30302 Err:     0 (0.00%)
summary +    128 in 00:00:30 =    4.3/s Avg:  8907 Min:     2 Max: 32981 Err:     0 (0.00%) Active: 50 Started: 50 Finished: 0
summary =   1074 in 00:05:01 =    3.6/s Avg:  7763 Min:     2 Max: 32981 Err:     0 (0.00%)
```

command to run JMeter test
every 30 secs, a short report is printed and it is added to the overall report so far
prints until it finish es

After the test, notice that JMeter has created a folder and a CSV file in the directory we specified in the command. The CSV contains a report of every HTTP request made to our test environment.



The folder contains an HTML report with graphs and tables showing how the test went. Open the file `index.html` to see this report:



APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.504	500 ms	1 sec 500 ms	Total
0.000	500 ms	1 sec 500 ms	/loyalty_cards - POST
1.000	500 ms	1 sec 500 ms	/users/signin - POST

Requests Summary

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	1342	0	0.00%	8059.15	2	32981	17.50	17036.90	18361.75	21594.26	3.62	0.80	0.94
/loyalty_cards - POST	666	0	0.00%	16234.07	15191	32981	15429.00	18383.80	19418.90	22255.24	1.81	0.51	0.46
/users/signin - POST	676	0	0.00%	5.17	2	115	4.00	7.00	10.00	17.00	1.91	0.31	0.51

Errors

Type of error	Number of errors	% in errors	% in all samples

You can use this report to examine how the load test went. There are also advanced options to configure the contents of this report.

What's next?

Now, we have a realistic load test plan that can be configured to run hundreds of virtual users. So far, we have run the test plan on our local machine. But our local machine has a limit. It starts to fail once we do a load test with 1000 VUs or more.

In the next post, we detail options on how to run load tests to accommodate a high number of virtual users.

Special Thanks

Special thanks to Allen for making my posts more coherent.

The authoritative JMeter book of Rodrigues, Mouawad, and Milamber entitled Master Apache JMeter From load testing to DevOps really helped me understand JMeter in more clarity. I highly suggest you purchase this book if you want to learn more about JMeter.

Photo by Sigmund on Unsplash

Load test your website with Apache JMeter (3 Part Series)

Top comments (1)

RituAgrawal1384 • 15 mar 23

Hi Raphael,

I am looking for next post on how to run load tests to accommodate a high number of virtual users. Can you please help me with the link if we have that.

Create up to 10 Postgres Databases on Neon's free plan.

If you're starting a new project, Neon has got your databases covered. No credit cards. No trials. No getting in your way.

Try Neon for Free →

Raphael Jambalos

Avid Reader, Curious Learner. AWS Certified ☁

LOCATION
Manila

EDUCATION
University of the Philippines

WORK
Senior Cloud Developer at eCloudvalley

JOINED

20 may 2018

More from Raphael Jambalos

AWS Network Challenge 2: Deploy File Upload App on EC2, RDS, DocumentDB

#webdev #aws #beginners

Full Stack Serverless Challenge #1: AWS Amplify Gen 2

#aws #webdev #typescript #fullstack

AWS Network Challenge 1: Deploy Web App to EC2 / Two-Tier VPC Architecture

#aws #webdev #network

Neon PROMOTED

Create up to 10 Postgres Databases on Neon's free plan.

If you're starting a new project, Neon has got your databases covered. No credit cards. No trials. No getting in your way.

Try Neon →