PERFORMANCE FRAMEWORK DEVELOPMENT GUIDE

TABLE OF CONTENTS

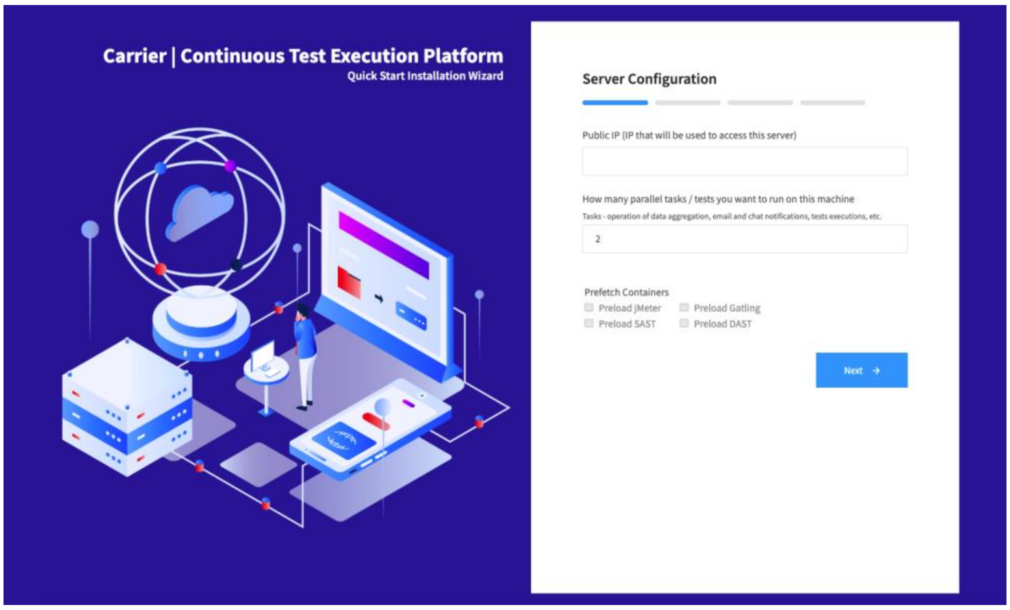
1 CREATING TEST ENVIROMENT

1.1 CARRIER DEPLOYMENT

Carrier is an open source continuous testing toolkit. It consists of tools and practices to integrate non-functional tests into delivery pipeline and organize effective feedback loop.

Carrier installer you can find here. Quick and guided installation of Carrier you can find here.

1. Provide reporting instance IP and set 2 parallel tasks


Carrier | Continuous Test Execution Platform
Quick Start Installation Wizard

Server Configuration

Public IP (IP that will be used to access this server)

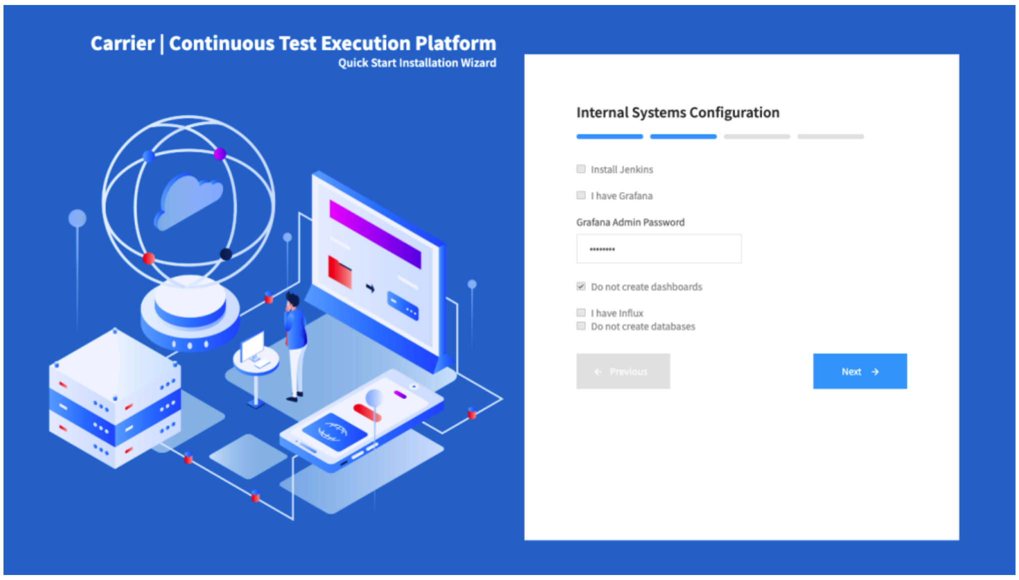
How many parallel tasks / tests you want to run on this machine

Tasks - operation of data aggregation, email and chat notifications, tests executions, etc.

2

Prefetch Containers
- ☐ Preload jMeter
- ☐ Preload Gatling
- ☐ Preload SAST
- ☐ Preload DAST

2. Set password and check “Do not create dashboards” checkbox


Carrier | Continuous Test Execution Platform
Quick Start Installation Wizard

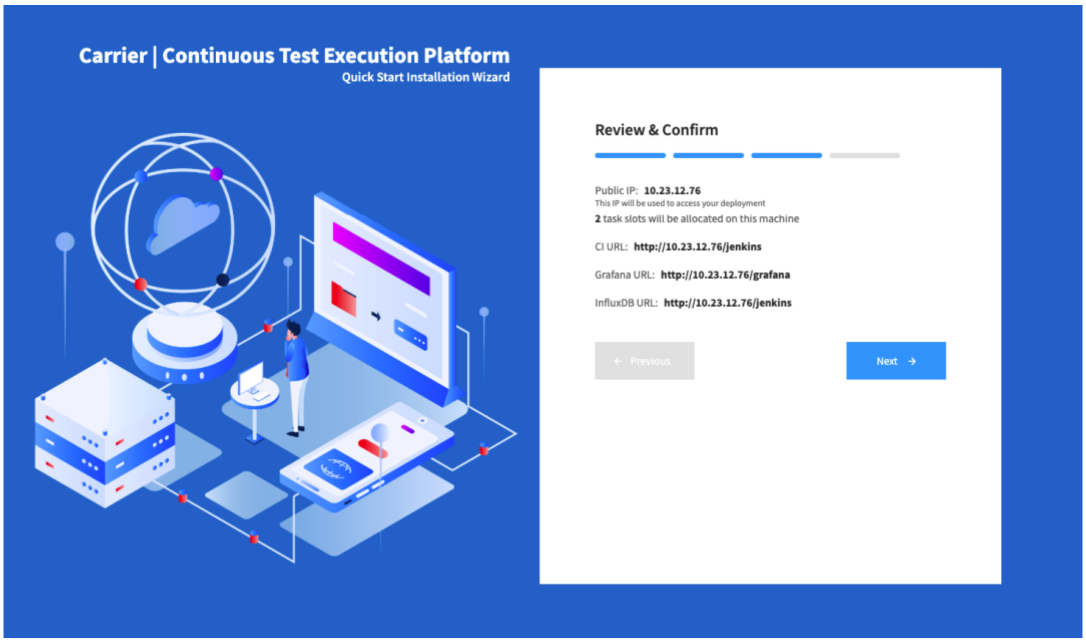
Internal Systems Configuration

- ☐ Install Jenkins
- ☐ I have Grafana

Grafana Admin Password: ••••••••

- ☑ Do not create dashboards
- ☐ I have Influx
- ☐ Do not create databases

3. Click “Next” button


Carrier | Continuous Test Execution Platform

Quick Start Installation Wizard

Review & Confirm

Public IP: **10.23.12.76**

This IP will be used to access your deployment

2 task slots will be allocated on this machine

CI URL: **http://10.23.12.76/jenkins**

Grafana URL: **http://10.23.12.76/grafana**

InfluxDB URL: **http://10.23.12.76/jenkins**

4. Once installation is done you will see `Installation complete ...` message in log trace area
5. Then you need to proceed with the data sources and dashboards creation from the 7th step of the manual installation

You can also install all components manually

1.1.1 Manual installation

To install Carrier you need docker to be installed and can use a docker-compose file. It relies on specific paths and configurations:

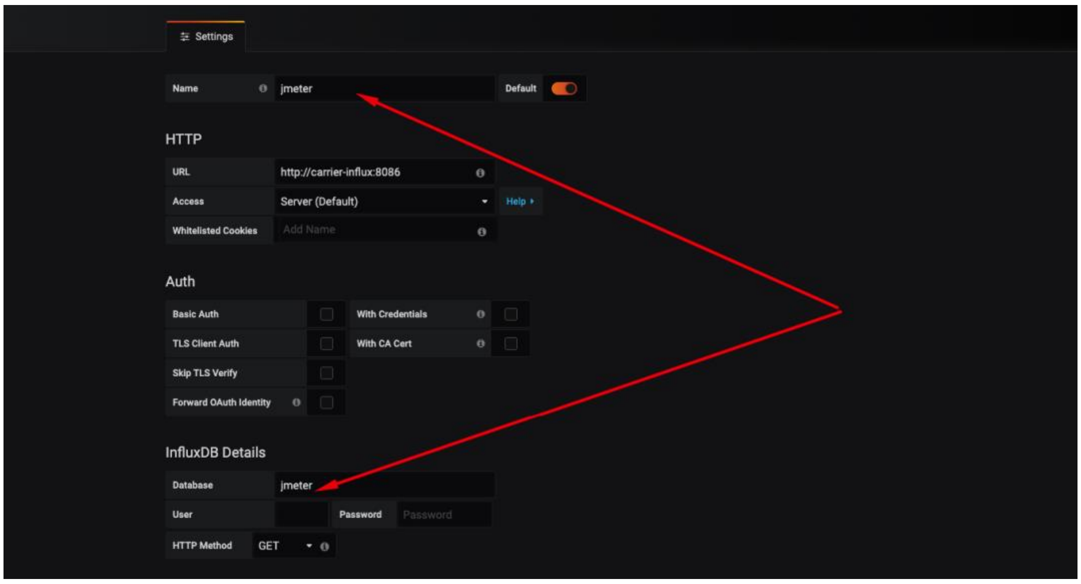
1. Replace {PUBLIC_IP} with instance public DNS
2. Create the next folders and give it write and read permissions:
   - /opt/carrier_grafana
   - /opt/carrier_influx
   - /opt/carrier_minio
   - /opt/carrier_traefik
3. Create and put traefik.toml file to /opt/carrier_traefik (file content is presented below) Content of the traefik.toml (ONLY FOR MANUAL INSTALLATION):

```
################################################################
# API and dashboard configuration
################################################################
[api]
################################################################
# Docker configuration backend
################################################################
[docker]
domain = "docker.local"
watch = true
```

4. For Mac you most likely will need to add those folders into File Sharing section of docker configuration

5. Run docker-compose up command
6. Execute command:
   `$ curl -XPOST -d bucket=tests http://localhost/artifacts/bucket`
7. Configuration of InfluxDB databases you can find below.
   InfluxDB configuration:
   1. Run command `$ docker exec -ti carrier-influx influx`
   2. Create next databases, using command `$ create database <database_name>`:
      a) jmeter
      b) comparison
      c) telegraf
      d) thresholds
      e) profiling
   3. Alter retention police for the profiling database:
      ```
      $ alter retention policy "autogen" on "profiling" duration 2w default
      ```

8. Login to Grafana on `http://<PUBLIC_IP>/grafana` and create datasources in the provided order:
   1. jmeter
   2. telegraf
   3. comparison
   4. thresholds
   5. profiling



Settings

Name: jmeter — Default: on

HTTP
- URL: http://carrier-influx:8086
- Access: Server (Default)
- Whitelisted Cookies: Add Name

Auth
- Basic Auth
- With Credentials
- TLS Client Auth
- With CA Cert
- Skip TLS Verify
- Forward OAuth Identity

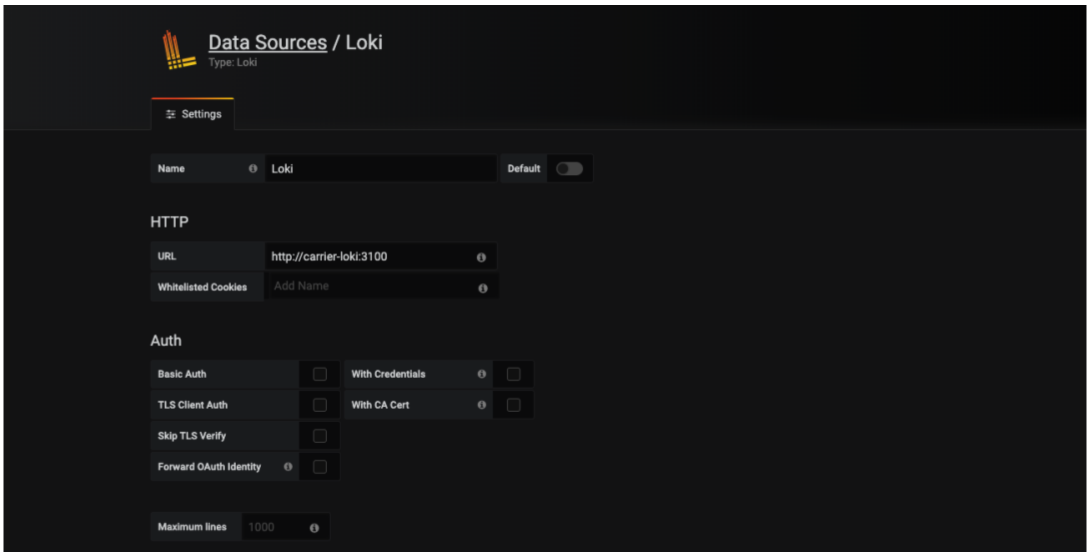
InfluxDB Details
- Database: jmeter
- User
- Password: Password
- HTTP Method: GET

You need to provide the same name for the “Name” and “Database” fields and provide “http://carrier-influx:8086” for the “URL” filed.

Important!
It is important to add datasources in the provided order.
It could fail to check for the “profiling” data source as we will create a database for it later, just ignore the alert.

9. Create datasource for Loki

1. Loki


Data Sources / Loki
Type: Loki

Settings

Name: Loki
Default

HTTP

URL: http://carrier-loki:3100
Whitelisted Cookies: Add Name

Auth

Basic Auth
With Credentials
TLS Client Auth
With CA Cert
Skip TLS Verify
Forward OAuth Identity

Maximum lines: 1000

You need to provide "Loki" name and "http://carrier-loki:3100" for the "URL" parameter

10. Configuration of Grafana dashboards you can find on the Carrier wiki page. Grafana dashboard you can download from git repo. Import all dashboards from the repository by copy-pasting the JSON.

As a result of these actions you will have Galloper UI (port 80), Redis, Loki, InfluxDB and Grafana. Grafana will be on http://<PUBLIC_IP>/grafana.

For the correct operation of the carrier components it is necessary that the following ports are opened in the instance:

- 80
- 3100
- 6379
- 8080
- 8086
- 9000
- 9999
- 8000

For AWS you can use the following command:

```
$ aws ec2 authorize-security-group-ingress --group-id <group_id> --protocol tcp --port
<port> --cidr <cidr>
```

Important!
Please note that you also have to open these port for your load generator cidr.

2 POST PROCESSING AND EMAIL NOTIFICATIONS

2.1 POST PROCESSING

After the end of the test, each load generator sends artifacts and the necessary information for post-processing to the minio bucket.

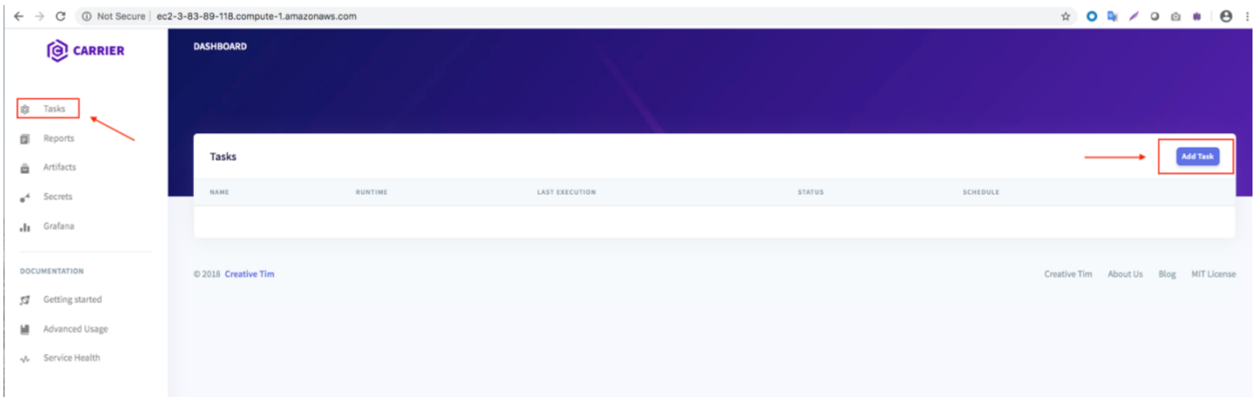
So you need to pass environment variable `galloper_url=http://{{ galloper_url }}` to control tower.

For post-processing, a bucket is created by the name of the control tower job.
Also you may specify a prefix to the files to be saved in minio. To do this, pass the variable `PREFIX` to control tower.

Post processing is used for aggregation of test results. The summary result is written in the comparison table.

To run the post processing, you need to create a task in the Galloper with post processing lambda.

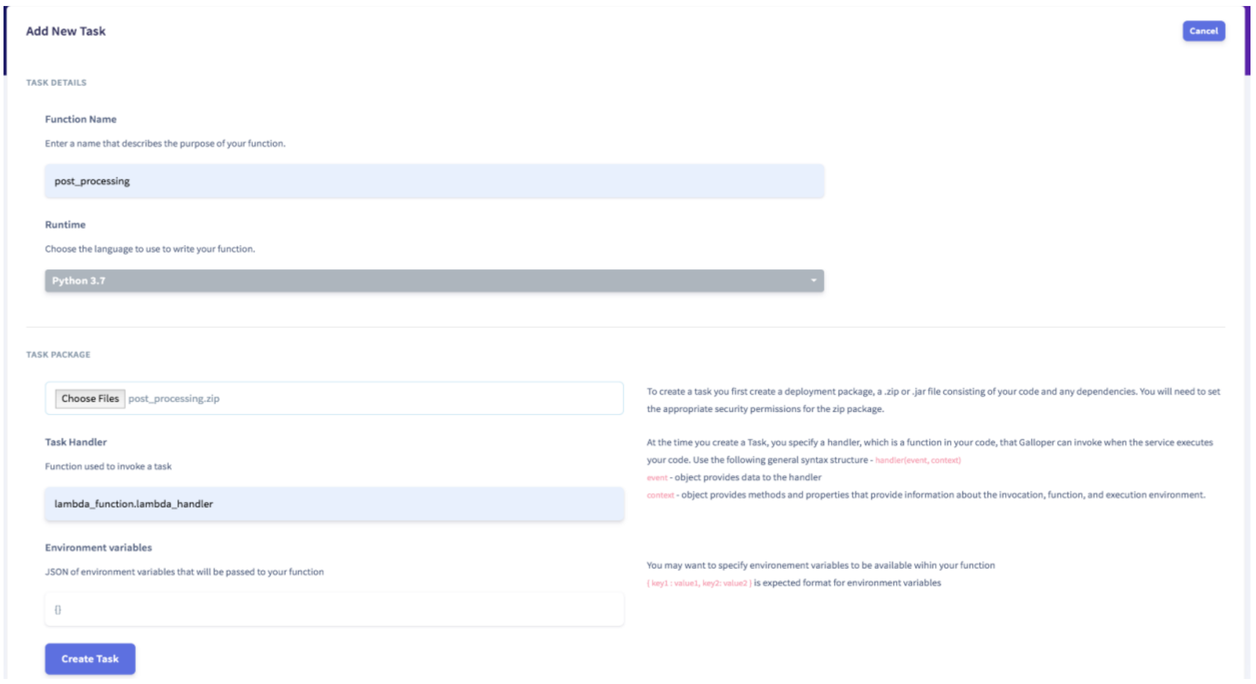
To do this, you need to go to the main page (port 80) of the instance with the carrier installed. On the left side there is a menu, you need to click on the tasks section. And then click on the "Add Task" button.



When you create a task, you need to specify a lambda handler in it – "lambda_function.lambda_handler".

The Lambda functions for notifications are written in python 3.7, so you need to specify this in the "Runtime" field when creating the task.

You also need to upload a zip file with the packed *function*.


Add New Task

Cancel

TASK DETAILS

Function Name

Enter a name that describes the purpose of your function.

post_processing

Runtime

Choose the language to use to write your function.

Python 3.7

TASK PACKAGE

Choose Files post_processing.zip

To create a task you first create a deployment package, a .zip or .jar file consisting of your code and any dependencies. You will need to set the appropriate security permissions for the zip package.

Task Handler

Function used to invoke a task

lambda_function.lambda_handler

At the time you create a Task, you specify a handler, which is a function in your code, that Galloper can invoke when the service executes your code. Use the following general syntax structure - handler(event, context)

event - object provides data to the handler

context - object provides methods and properties that provide information about the invocation, function, and execution environment.

Environment variables

JSON of environment variables that will be passed to your function

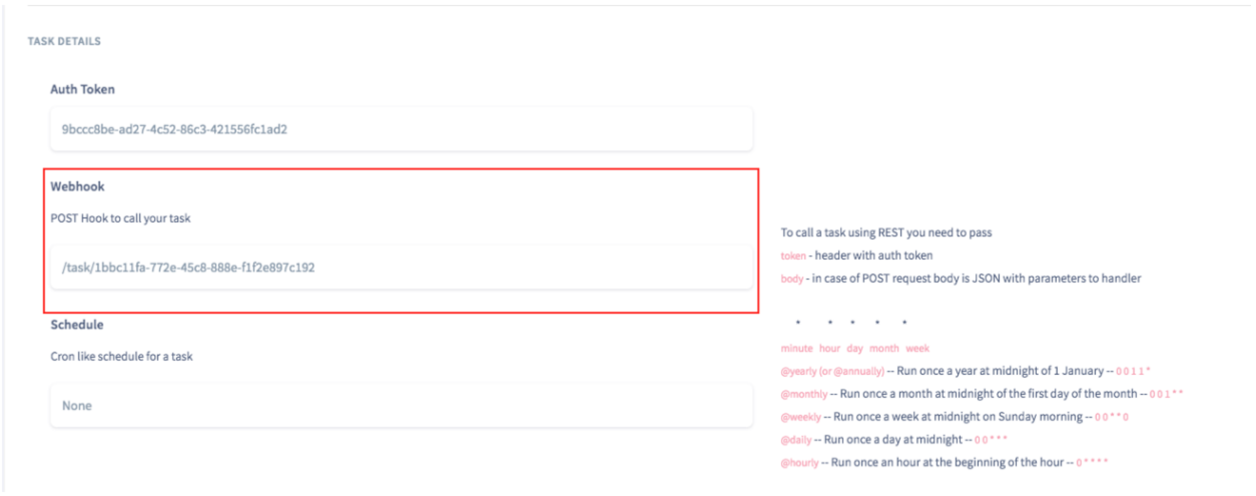
{}

You may want to specify environement variables to be available wihin your function

{ key1 : value1, key2: value2 } is expected format for environment variables

Create Task

Click “Create Task” button and you will be redirected to the page with created function details. You need to copy a webhook to the function.


TASK DETAILS

Auth Token

9bccc8be-ad27-4c52-86c3-421556fc1ad2

Webhook

POST Hook to call your task

/task/1bbc11fa-772e-45c8-888e-f1f2e897c192

To call a task using REST you need to pass

token - header with auth token

body - in case of POST request body is JSON with parameters to handler

Schedule

Cron like schedule for a task

None

* * * * *

minute hour day month week

@yearly (or @annually) -- Run once a year at midnight of 1 January -- 0 0 1 1 *

@monthly -- Run once a month at midnight of the first day of the month -- 0 0 1 * *

@weekly -- Run once a week at midnight on Sunday morning -- 0 0 * * 0

@daily -- Run once a day at midnight -- 0 0 * * *

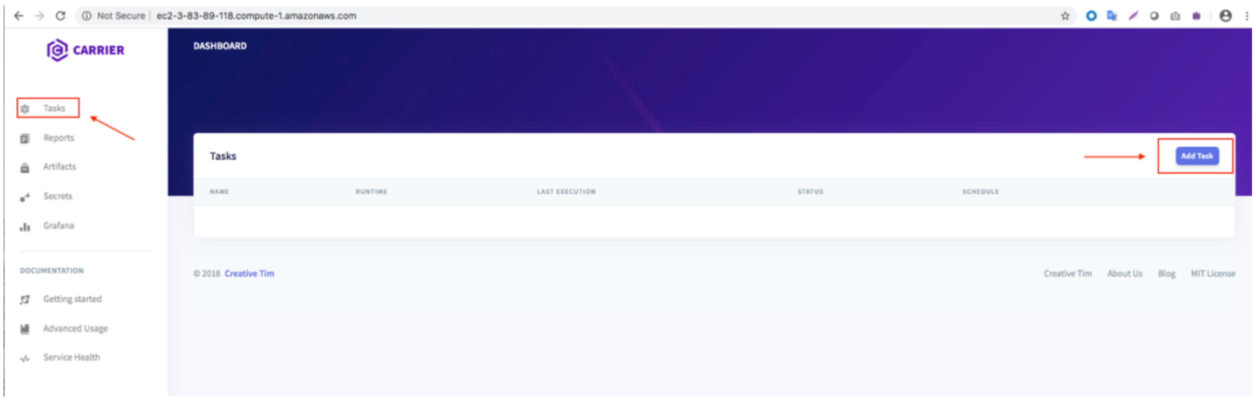
@hourly -- Run once an hour at the beginning of the hour -- 0 * * * *

Then you need to pass an environment variable `GALLOPER_WEB_HOOK=http://{{ galloper_url }}/task/{{ web_hook }}` to control tower.

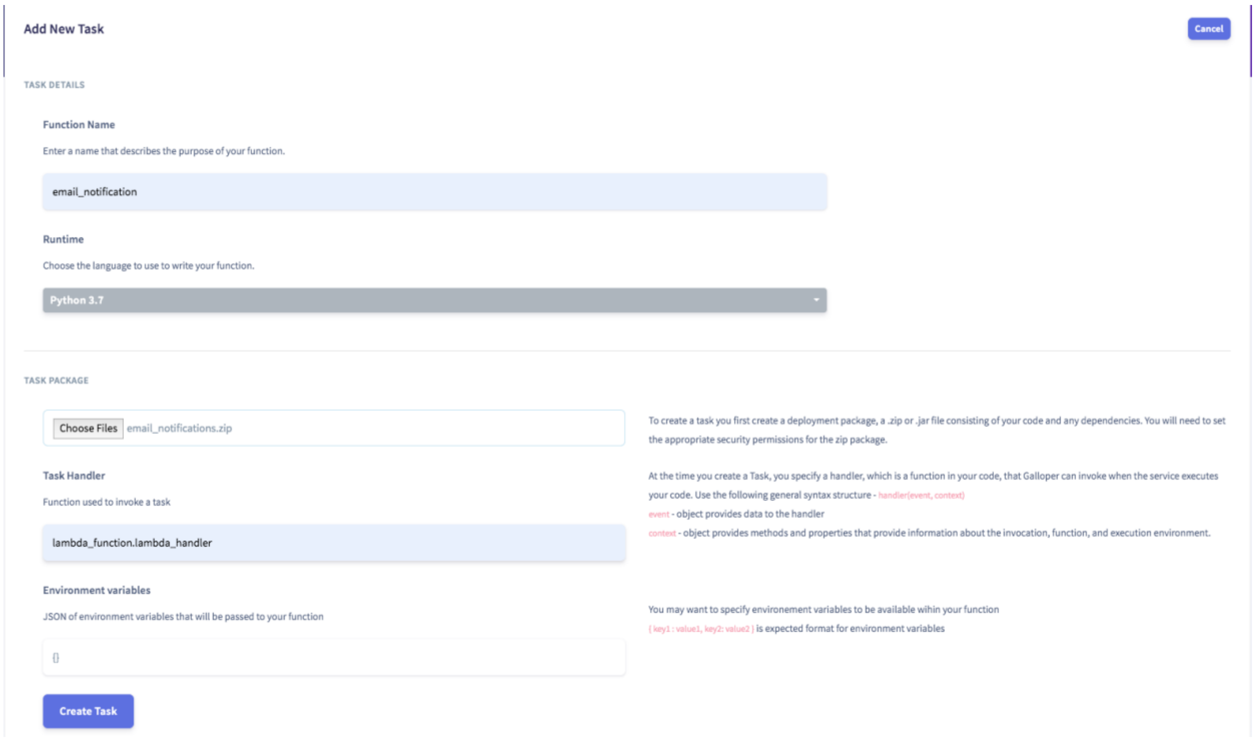
2.2 EMAIL NOTIFICATIONS

At the end of the test, you can receive an email notification with the test results. In addition to the results of the current test, the email also contains a comparison of the last five tests in the form of a table and several charts.

To use email notifications, you need to create a task in Galloper. To do this, you need to go to the main page (port 80) of the instance with the carrier installed. On the left side there is a menu, you need to click on the tasks section. And then click on the “Add Task” button.



When you create a task, you need to specify a lambda handler in it – "lambda_function.lambda_handler". The Lambda functions for notifications are written in python 3.7, so you need to specify this in the "Runtime" field when creating the task. You also need to upload a zip file with the packed function. You can find lambda function that provide email notifications here.



Function Name

Enter a name that describes the purpose of your function.

email_notification

Runtime

Choose the language to use to write your function.

Python 3.7

email_notifications.zip

To create a task you first create a deployment package, a .zip or .jar file consisting of your code and any dependencies. You will need to set the appropriate security permissions for the zip package.

Task Handler

Function used to invoke a task

lambda_function.lambda_handler

At the time you create a Task, you specify a handler, which is a function in your code, that Galloper can invoke when the service executes your code. Use the following general syntax structure - handler(event, context)

event - object provides data to the handler

context - object provides methods and properties that provide information about the invocation, function, and execution environment.

Environment variables

JSON of environment variables that will be passed to your function

{}

You may want to specify environement variables to be available wihin your function

{ key1 : value1, key2: value2 } is expected format for environment variables

You can run a task with an email notification using CURL. Example how to invoke a task using CURL:

```
curl -XPOST -H "Content-Type: application/json"
    -d '{"param1": "value1", "param2": "value2", ...}' <host>/<webhook>
```

<host> - Galloper host DNS or IP
<webhook> - POST Hook to call your task

A list of all valid parameters that can be passed to the function is provided below:

- 'test': '<test_name>' – test name, e.g. "Folio"

- 'test_type': '<test_type>' - e.g capacity, fixed_load
- 'users': '<count_of_vUsers>' - vUsers count for test execution
- 'influx_host': '<influx_host_DNS_or_IP>' -“ec2-3-83-89-118.compute-1.amazonaws.com”
- 'smpt_user': '<email>' - smpt user who will send an email
- 'smpt_password': '<password>'
- 'user_list': '<list of recipients>' - [“email.gmail.com”, “email.gmai2.com”]
- 'notification_type': 'api'

Additional params you can find on the carrier wiki page

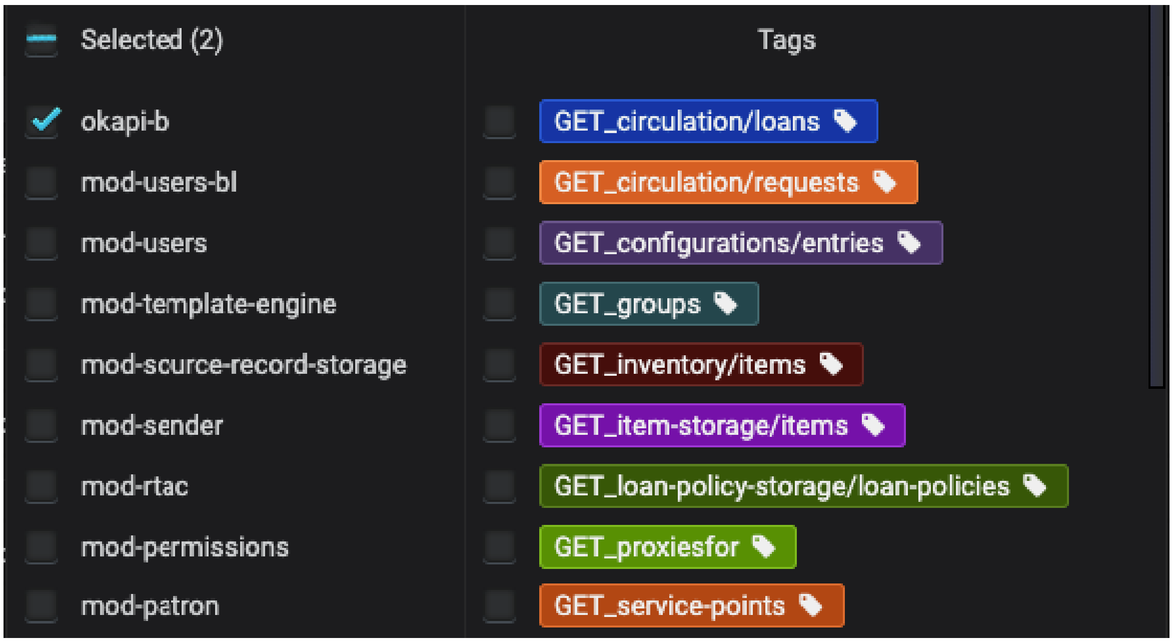
2.3 DASHBOARD CONFIGURATION

2.3.1 Java profiling dashboard

Based on the data from the profiler, a Grafana dashboard with charts and tables was developed for in-depth analysis of the results.

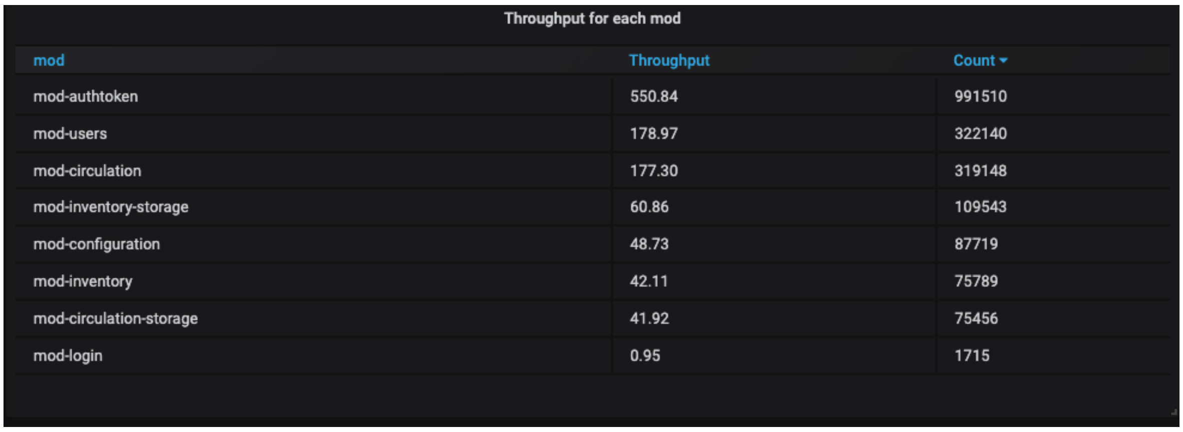
This dashboard allows us to analyze the results for each service individually. To do this, you need to configure a service filter.

The first filter allows us to select a service. You can also select one of the API request and the filter will automatically select the services that processed it.



Selected (2)	Tags
✓ okapi-b	GET_circulation/loans
mod-users-bl	GET_circulation/requests
mod-users	GET_configurations/entries
mod-template-engine	GET_groups
mod-scurce-record-storage	GET_inventory/items
mod-sender	GET_item-storage/items
mod-rtac	GET_loan-policy-storage/loan-policies
mod-permissions	GET_proxiesfor
mod-patron	GET_service-points

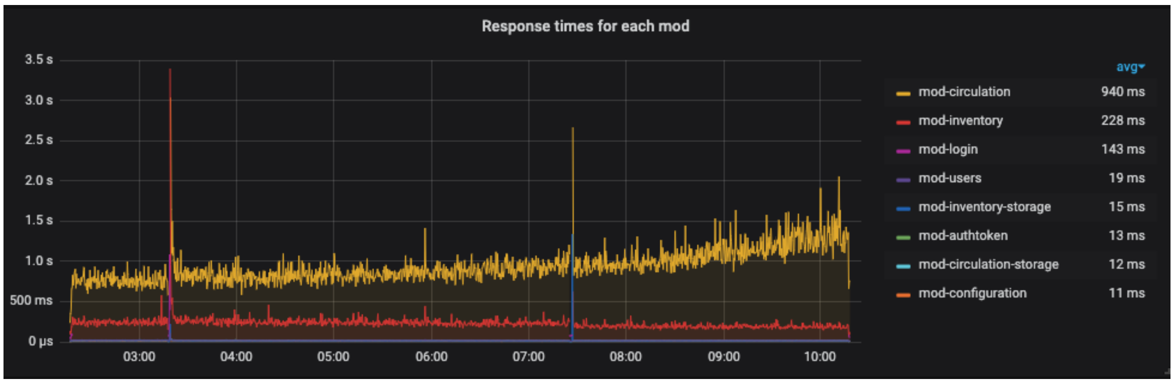
The first table and chart on the dashboard are not based on the data from the profiler, but on the data received with the x-okapi-trace response headers. This header contains information about which service processed the request and how long it took. In the table, we can see how many times a service was triggered during the test.



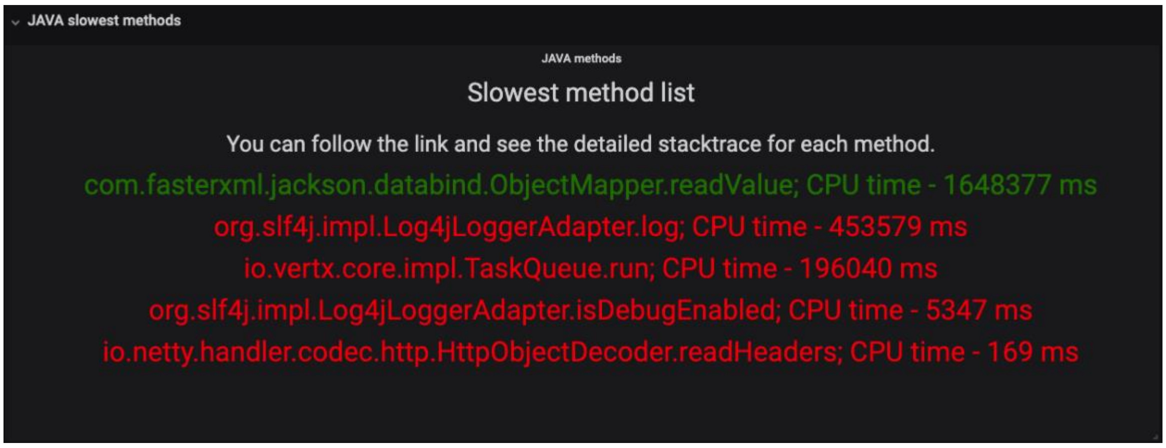
Throughput for each mod

mod	Throughput	Count
mod-authtoken	550.84	991510
mod-users	178.97	322140
mod-circulation	177.30	319148
mod-inventory-storage	60.86	109543
mod-configuration	48.73	87719
mod-inventory	42.11	75789
mod-circulation-storage	41.92	75456
mod-login	0.95	1715

On the graph we can see the response time for each service during the test. We can also filter this graph by selecting a specific service on the right side.



The following panel shows a list of the slowest methods.



JAVA slowest methods

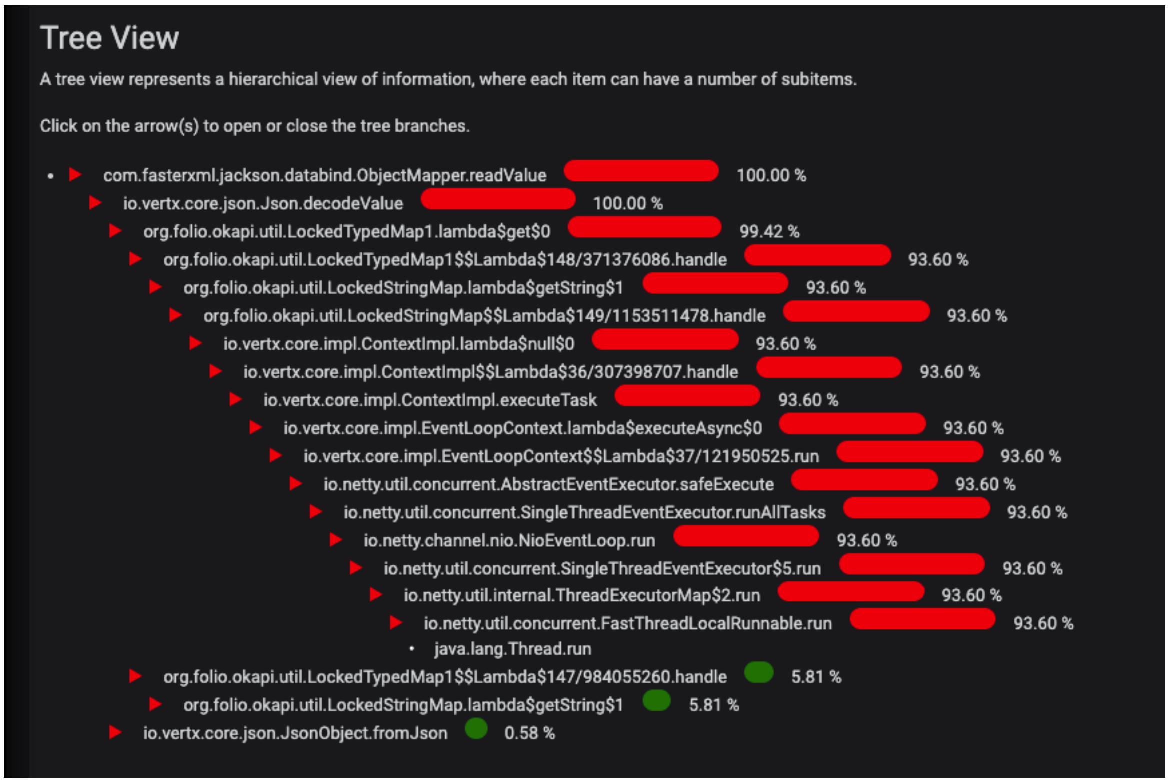
JAVA methods

Slowest method list

You can follow the link and see the detailed stacktrace for each method.

com.fasterxml.jackson.databind.ObjectMapper.readValue; CPU time - 1648377 ms
org.slf4j.impl.Log4jLoggerAdapter.log; CPU time - 453579 ms
io.vertx.core.impl.TaskQueue.run; CPU time - 196040 ms
org.slf4j.impl.Log4jLoggerAdapter.isDebugEnabled; CPU time - 5347 ms
io.netty.handler.codec.http.HttpObjectDecoder.readHeaders; CPU time - 169 ms

Each method is a link, if you click on it, a new tab will open a panel with a detailed stacktrace for this method.


Tree View

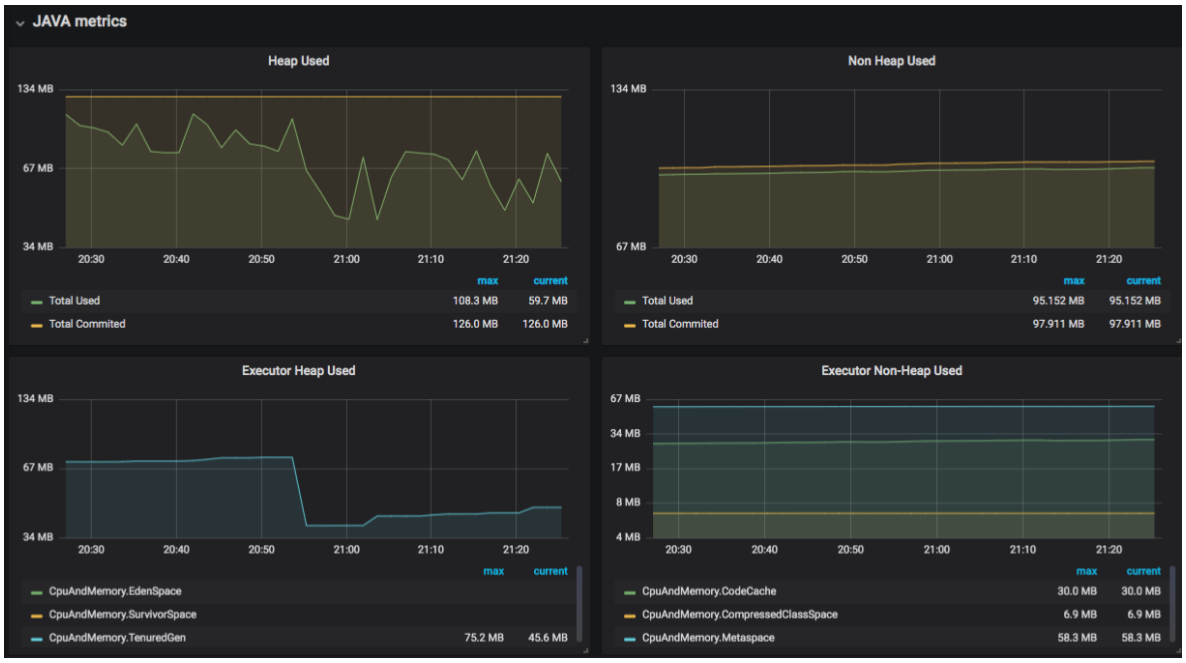
A tree view represents a hierarchical view of information, where each item can have a number of subitems.

Click on the arrow(s) to open or close the tree branches.

```
• ▶ com.fasterxml.jackson.databind.ObjectMapper.readValue  100.00 %
   ▶ io.vertx.core.json.Json.decodeValue  100.00 %
     ▶ org.folio.okapi.util.LockedTypedMap1.lambda$get$0  99.42 %
       ▶ org.folio.okapi.util.LockedTypedMap1$$Lambda$148/371376086.handle  93.60 %
         ▶ org.folio.okapi.util.LockedStringMap.lambda$getString$1  93.60 %
           ▶ org.folio.okapi.util.LockedStringMap$$Lambda$149/1153511478.handle  93.60 %
             ▶ io.vertx.core.impl.ContextImpl.lambda$null$0  93.60 %
               ▶ io.vertx.core.impl.ContextImpl$$Lambda$36/307398707.handle  93.60 %
                 ▶ io.vertx.core.impl.ContextImpl.executeTask  93.60 %
                   ▶ io.vertx.core.impl.EventLoopContext.lambda$executeAsync$0  93.60 %
                     ▶ io.vertx.core.impl.EventLoopContext$$Lambda$37/121950525.run  93.60 %
                       ▶ io.netty.util.concurrent.AbstractEventExecutor.safeExecute  93.60 %
                         ▶ io.netty.util.concurrent.SingleThreadEventExecutor.runAllTasks  93.60 %
                           ▶ io.netty.channel.nio.NioEventLoop.run  93.60 %
                             ▶ io.netty.util.concurrent.SingleThreadEventExecutor$5.run  93.60 %
                               ▶ io.netty.util.internal.ThreadExecutorMap$2.run  93.60 %
                                 ▶ io.netty.util.concurrent.FastThreadLocalRunnable.run  93.60 %
                                    • java.lang.Thread.run
       ▶ org.folio.okapi.util.LockedTypedMap1$$Lambda$147/984055260.handle  5.81 %
         ▶ org.folio.okapi.util.LockedStringMap.lambda$getString$1  5.81 %
     ▶ io.vertx.core.json.JsonObject.fromJson  0.58 %
```

This stacktrace is a set of all stacktraces that were performed during a test with this method. You can expand them by clicking on the arrow and go all the way from the selected method to the beginning of the stacktrace. The red color indicates the path that stacktrace most often follows. Also, on each part of stacktrace there is a percentage ratio of how often this method was executed. It may be useful for analysis of non-optimal program execution.

The last part of the dashboard is Java metrics such as CPU usage, heap memory, etc.



Heap Used

	max	current
Total Used	108.3 MB	59.7 MB
Total Commited	126.0 MB	126.0 MB

Non Heap Used

	max	current
Total Used	95.152 MB	95.152 MB
Total Commited	97.911 MB	97.911 MB

Executor Heap Used

	max	current
CpuAndMemory.EdenSpace		
CpuAndMemory.SurvivorSpace		
CpuAndMemory.TenuredGen	75.2 MB	45.6 MB

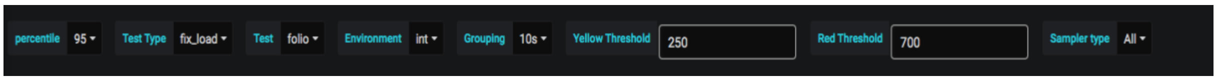
Executor Non-Heap Used

	max	current
CpuAndMemory.CodeCache	30.0 MB	30.0 MB
CpuAndMemory.CompressedClassSpace	6.9 MB	6.9 MB
CpuAndMemory.Metaspace	58.3 MB	58.3 MB

2.3.2 JMeter dashboard

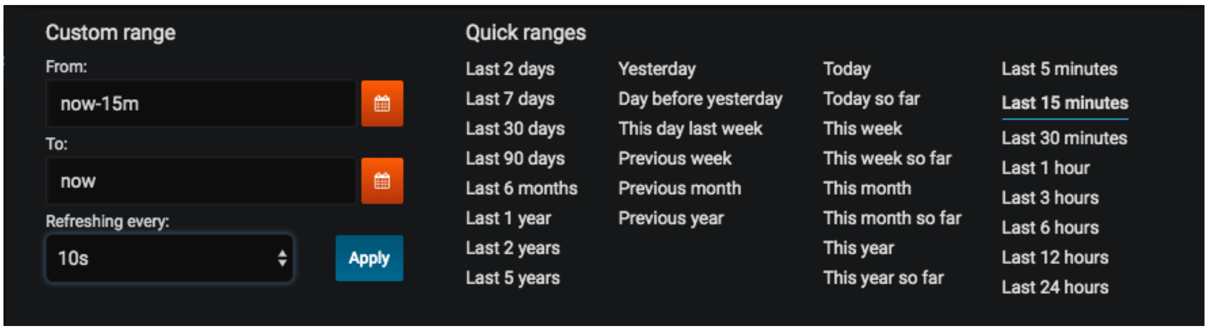
During the test execution all performance metrics are being saved to InfluxDB and displayed in Grafana.

Grafana allows us to review performance tests results using different filters which helps us to divide executed tests by parameters (e.g.: Simulation name, Test type, Environment, etc.).



percentile 95 · Test Type fix_load · Test folio · Environment int · Grouping 10s · Yellow Threshold 250 · Red Threshold 700 · Sampler type All

You can pick necessary time range in which you want to see your results. You can also set the "Refreshing every:" option. This allows you to automatically update the dashboard at specified intervals.



Custom range

From: now-15m

To: now

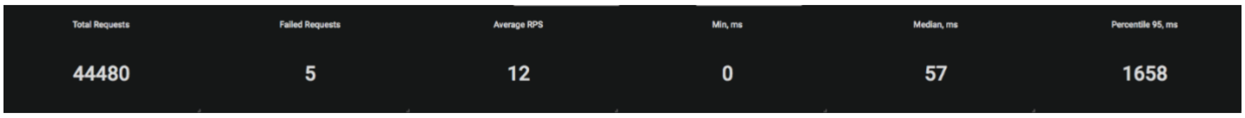
Refreshing every: 10s

Apply

Quick ranges

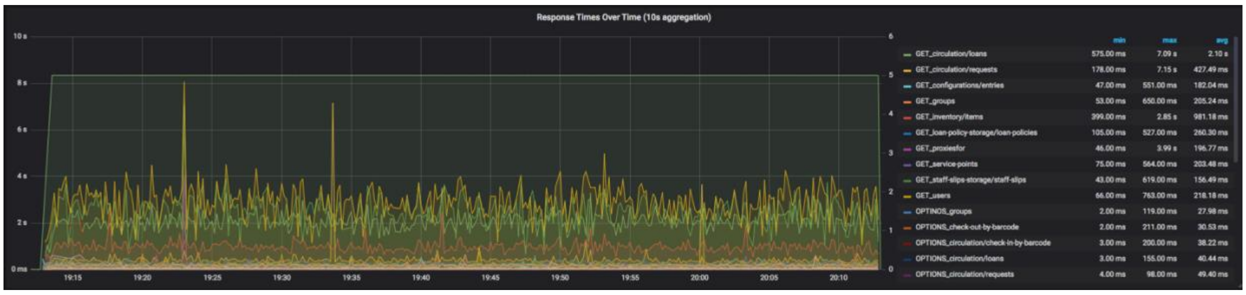
Last 2 days	Yesterday	Today	Last 5 minutes
Last 7 days	Day before yesterday	Today so far	Last 15 minutes
Last 30 days	This day last week	This week	Last 30 minutes
Last 90 days	Previous week	This week so far	Last 1 hour
Last 6 months	Previous month	This month	Last 3 hours
Last 1 year	Previous year	This month so far	Last 6 hours
Last 2 years		This year	Last 12 hours
Last 5 years		This year so far	Last 24 hours

After all filters set-up properly, you will be able to see results of test execution.
The first block consists of panels with overall information.



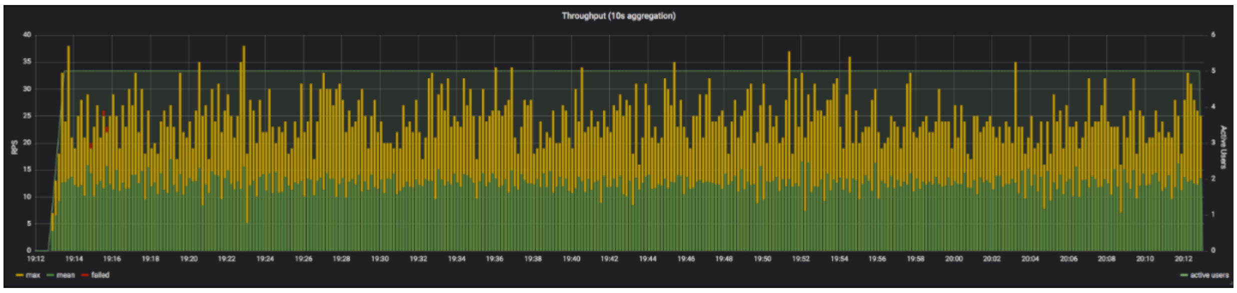
Total Requests	Failed Requests	Average RPS	Min, ms	Median, ms	Percentile 95, ms
44480	5	12	0	57	1658

The second block of the dashboard named “Response Times Over Time” contains a chart where you can see Response time for all requests or only for chosen in the right side of the block.

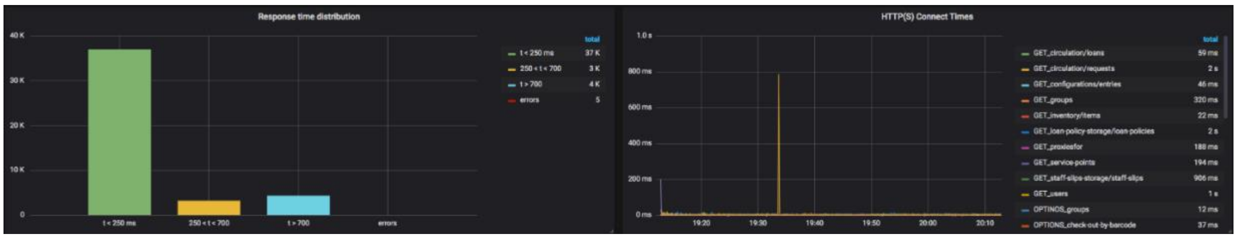


	min	max	avg
GET_circulation/loans	575.00 ms	7.09 s	2.10 s
GET_circulation/requests	178.00 ms	7.15 s	427.49 ms
GET_configurations/entries	47.00 ms	551.00 ms	182.04 ms
GET_groups	53.00 ms	650.00 ms	205.24 ms
GET_inventory/items	399.00 ms	2.85 s	981.18 ms
GET_loan-policy-storage/loan-policies	105.00 ms	527.00 ms	260.30 ms
GET_proxiesfor	46.00 ms	3.99 s	196.77 ms
GET_service-points	75.00 ms	564.00 ms	203.48 ms
GET_staff-slips-storage/staff-slips	43.00 ms	619.00 ms	156.49 ms
GET_users	66.00 ms	763.00 ms	218.18 ms
OPTINOS_groups	2.00 ms	119.00 ms	27.98 ms
OPTIONS_check-out-by-barcode	2.00 ms	211.00 ms	30.53 ms
OPTIONS_circulation/check-in-by-barcode	3.00 ms	200.00 ms	38.22 ms
OPTIONS_circulation/loans	3.00 ms	155.00 ms	40.44 ms
OPTIONS_circulation/requests	4.00 ms	98.00 ms	49.40 ms

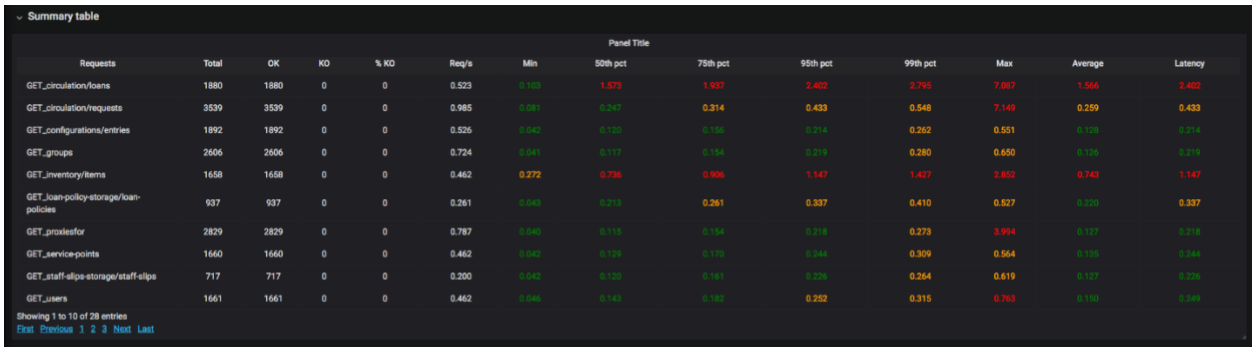
The third block of the dashboard, called "Throughput", contains a graph showing the change in throughput over time.



The following two blocks show response time distribution and HTTP Connection times during the test execution.



"Summary table" block contains a table consisting of detailed statistics for each request. In case of empty table refresh the page.



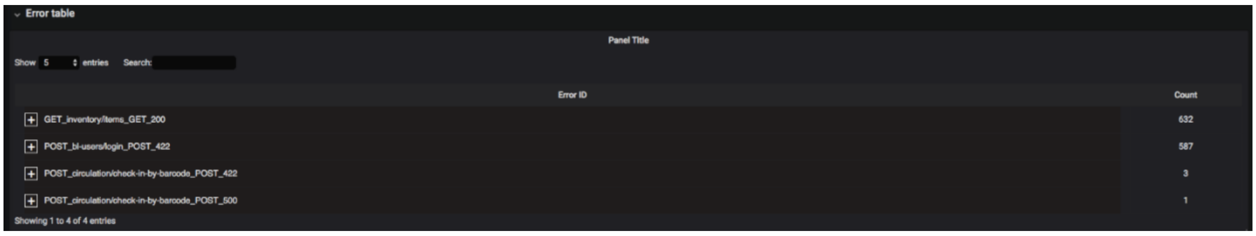
Summary table

Panel Title

Requests	Total	OK	KO	% KO	Req/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Average	Latency
GET_circulation/loans	1880	1880	0	0	0.523	0.103	1.573	1.937	2.402	2.795	7.087	1.566	2.402
GET_circulation/requests	3539	3539	0	0	0.985	0.081	0.247	0.314	0.433	0.548	7.149	0.259	0.433
GET_configurations/entries	1892	1892	0	0	0.526	0.042	0.120	0.156	0.214	0.262	0.551	0.128	0.214
GET_groups	2606	2606	0	0	0.724	0.041	0.117	0.154	0.219	0.280	0.650	0.126	0.219
GET_inventory/items	1658	1658	0	0	0.462	0.272	0.736	0.906	1.147	1.427	2.852	0.743	1.147
GET_loan-policy-storage/loan-policies	937	937	0	0	0.261	0.043	0.213	0.261	0.337	0.410	0.527	0.220	0.337
GET_proxiesfor	2829	2829	0	0	0.787	0.040	0.115	0.154	0.218	0.273	3.994	0.127	0.218
GET_service-points	1660	1660	0	0	0.462	0.042	0.129	0.170	0.244	0.309	0.564	0.135	0.244
GET_staff-slips-storage/staff-slips	717	717	0	0	0.200	0.042	0.120	0.161	0.226	0.264	0.619	0.127	0.226
GET_users	1661	1661	0	0	0.462	0.046	0.143	0.182	0.252	0.315	0.763	0.150	0.249

Showing 1 to 10 of 28 entries
First Previous 1 2 3 Next Last

The last block of the dashboard called "Error table". It contains a detailed description for each failed request during the test.



Error table

Panel Title

Show 5 entries Search:

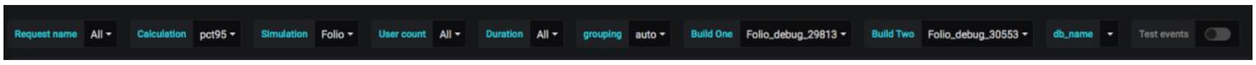
Error ID	Count
GET_inventory/items_GET_200	632
POST_bl-users/login_POST_422	587
POST_circulation/check-in-by-barcode_POST_422	3
POST_circulation/check-in-by-barcode_POST_500	1

Showing 1 to 4 of 4 entries

2.3.3 Comparison dashboard

There are some cases when you need to compare performance results between different test executions for each request.

Carrier provides Grafana dashboard for this purpose. Grafana allows us to review performance test results using different filters which helps us to divide executed tests by parameters (e.g.: Simulation name, Duration, User count, etc.).

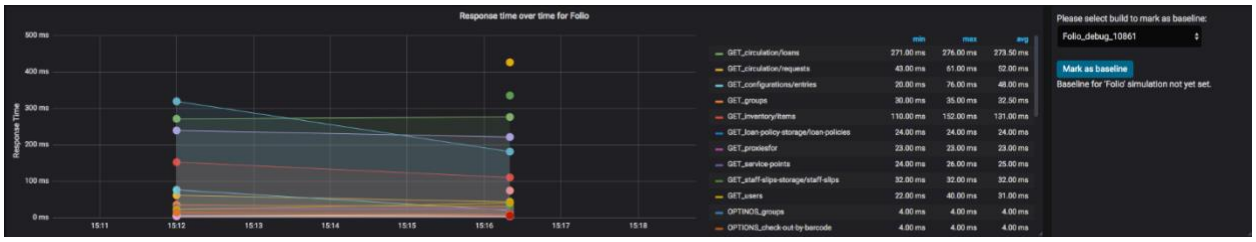


Request name: All; Calculation: pct95; Simulation: Folio; User count: All; Duration: All; grouping: auto; Build One: Folio_debug_29813; Build Two: Folio_debug_30553; db_name; Test events

Once all the filters have been configured correctly, you will be able to see a comparison of the results for the selected tests.

There are 3 main sections on the dashboard.

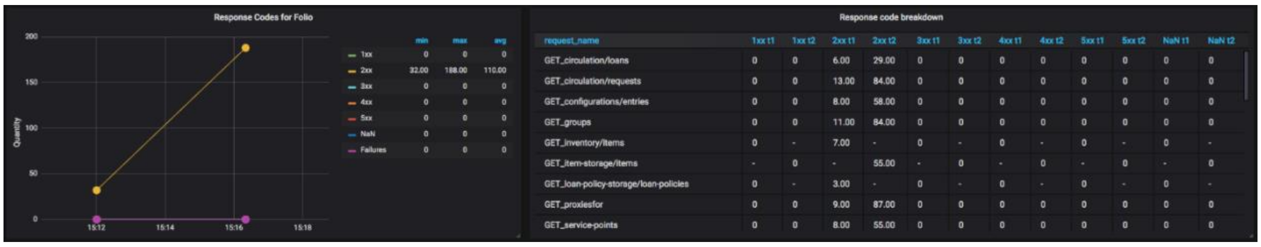
The first one shows a chart of the response time with a comparison of the selected tests for each request. To the right of this chart you can select one of the tests and mark it as baseline.



	min	max	avg
GET_circulation/loans	271.00 ms	276.00 ms	273.50 ms
GET_circulation/requests	43.00 ms	61.00 ms	52.00 ms
GET_configurations/entries	20.00 ms	76.00 ms	48.00 ms
GET_groups	30.00 ms	35.00 ms	32.50 ms
GET_inventory/items	110.00 ms	152.00 ms	131.00 ms
GET_loan-policy-storage/loan-policies	24.00 ms	24.00 ms	24.00 ms
GET_proxiesfor	23.00 ms	23.00 ms	23.00 ms
GET_service-points	24.00 ms	26.00 ms	25.00 ms
GET_staff-slips-storage/staff-slips	32.00 ms	32.00 ms	32.00 ms
GET_users	22.00 ms	40.00 ms	31.00 ms
OPTINOS_groups	4.00 ms	4.00 ms	4.00 ms
OPTIONS_check-out-by-barcode	4.00 ms	4.00 ms	4.00 ms

Please select build to mark as baseline:
Folio_debug_10861
Baseline for 'Folio' simulation not yet set.

The second section contains tables with the distribution of all requests by response code.

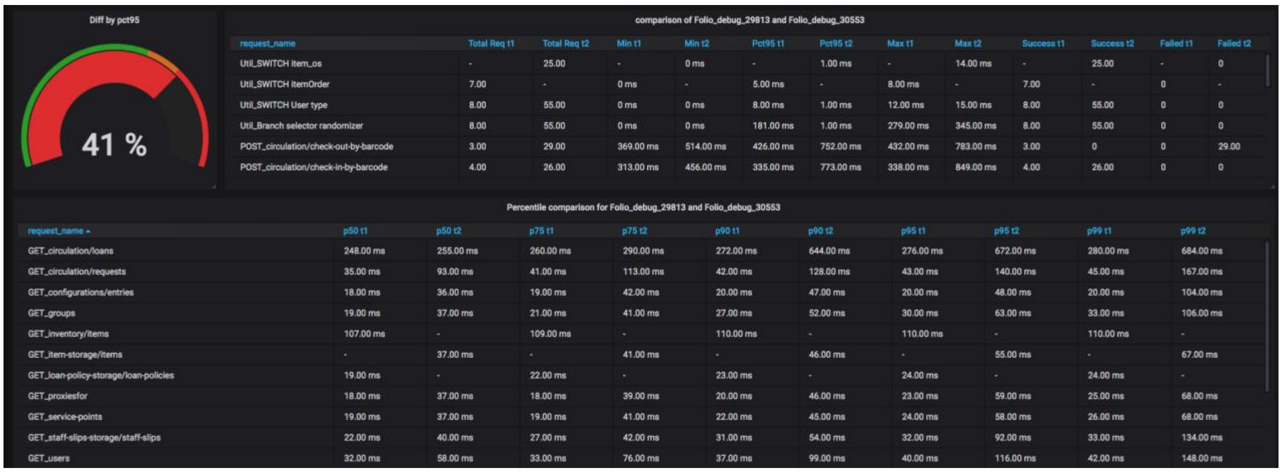


	min	max	avg
1xx	0	0	0
2xx	32.00	188.00	110.00
3xx	0	0	0
4xx	0	0	0
5xx	0	0	0
NaN	0	0	0
Failures	0	0	0

Response code breakdown

request_name	1xx t1	1xx t2	2xx t1	2xx t2	3xx t1	3xx t2	4xx t1	4xx t2	5xx t1	5xx t2	NaN t1	NaN t2
GET_circulation/loans	0	0	6.00	29.00	0	0	0	0	0	0	0	0
GET_circulation/requests	0	0	13.00	84.00	0	0	0	0	0	0	0	0
GET_configurations/entries	0	0	8.00	58.00	0	0	0	0	0	0	0	0
GET_groups	0	0	11.00	84.00	0	0	0	0	0	0	0	0
GET_inventory/items	0	-	7.00	-	0	-	0	-	0	-	0	-
GET_item-storage/items	-	0	-	55.00	-	0	-	0	-	0	-	0
GET_loan-policy-storage/loan-policies	0	-	3.00	-	0	-	0	-	0	-	0	-
GET_proxiesfor	0	0	9.00	87.00	0	0	0	0	0	0	0	0
GET_service-points	0	0	8.00	55.00	0	0	0	0	0	0	0	0

The last section contains tables comparing the response time for each request.



comparison of Folio_debug_29813 and Folio_debug_30553

request_name	Total Req t1	Total Req t2	Min t1	Min t2	Pct95 t1	Pct95 t2	Max t1	Max t2	Success t1	Success t2	Failed t1	Failed t2
Util_SWITCH item_os	-	25.00	-	0 ms	-	1.00 ms	-	14.00 ms	-	25.00	-	0
Util_SWITCH itemOrder	7.00	-	0 ms	-	5.00 ms	-	8.00 ms	-	7.00	-	0	-
Util_SWITCH User type	8.00	55.00	0 ms	0 ms	8.00 ms	1.00 ms	12.00 ms	15.00 ms	8.00	55.00	0	0
Util_Branch selector randomizer	8.00	55.00	0 ms	0 ms	181.00 ms	1.00 ms	279.00 ms	345.00 ms	8.00	55.00	0	0
POST_circulation/check-out-by-barcode	3.00	29.00	369.00 ms	514.00 ms	426.00 ms	752.00 ms	432.00 ms	783.00 ms	3.00	0	0	29.00
POST_circulation/check-in-by-barcode	4.00	26.00	313.00 ms	456.00 ms	335.00 ms	773.00 ms	338.00 ms	849.00 ms	4.00	26.00	0	0

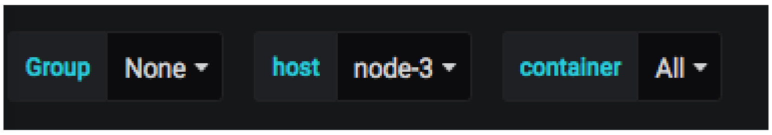
Percentile comparison for Folio_debug_29813 and Folio_debug_30553

request_name	p50 t1	p50 t2	p75 t1	p75 t2	p90 t1	p90 t2	p95 t1	p95 t2	p99 t1	p99 t2
GET_circulation/loans	248.00 ms	255.00 ms	260.00 ms	290.00 ms	272.00 ms	644.00 ms	276.00 ms	672.00 ms	280.00 ms	684.00 ms
GET_circulation/requests	35.00 ms	93.00 ms	41.00 ms	113.00 ms	42.00 ms	128.00 ms	43.00 ms	140.00 ms	45.00 ms	167.00 ms
GET_configurations/entries	18.00 ms	36.00 ms	19.00 ms	42.00 ms	20.00 ms	47.00 ms	20.00 ms	48.00 ms	20.00 ms	104.00 ms
GET_groups	19.00 ms	37.00 ms	21.00 ms	41.00 ms	27.00 ms	52.00 ms	30.00 ms	63.00 ms	33.00 ms	106.00 ms
GET_inventory/items	107.00 ms	-	109.00 ms	-	110.00 ms	-	110.00 ms	-	110.00 ms	-
GET_item-storage/items	-	37.00 ms	-	41.00 ms	-	46.00 ms	-	55.00 ms	-	67.00 ms
GET_loan-policy-storage/loan-policies	19.00 ms	-	22.00 ms	-	23.00 ms	-	24.00 ms	-	24.00 ms	-
GET_proxiesfor	18.00 ms	37.00 ms	18.00 ms	39.00 ms	20.00 ms	46.00 ms	23.00 ms	59.00 ms	25.00 ms	68.00 ms
GET_service-points	19.00 ms	37.00 ms	19.00 ms	41.00 ms	22.00 ms	45.00 ms	24.00 ms	58.00 ms	26.00 ms	68.00 ms
GET_staff-slips-storage/staff-slips	22.00 ms	40.00 ms	27.00 ms	42.00 ms	31.00 ms	54.00 ms	32.00 ms	92.00 ms	33.00 ms	134.00 ms
GET_users	32.00 ms	58.00 ms	33.00 ms	76.00 ms	37.00 ms	99.00 ms	40.00 ms	116.00 ms	42.00 ms	148.00 ms

2.3.4 Folio resource utilization dashboard

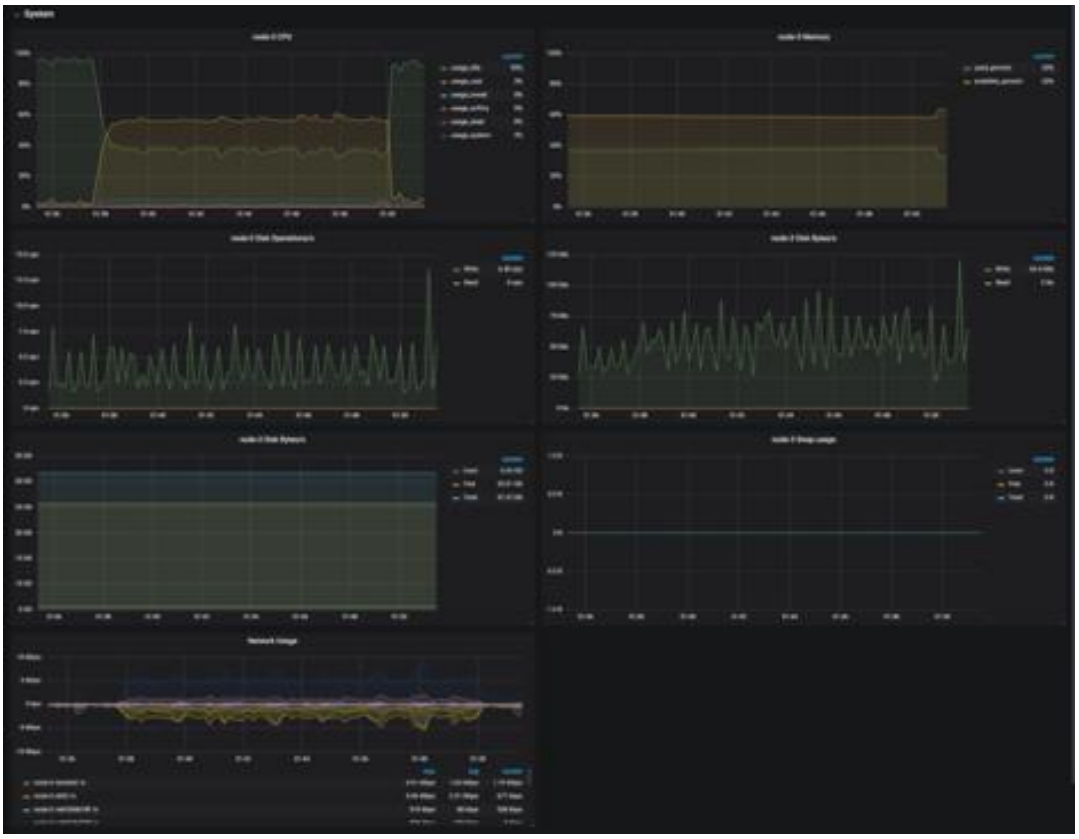
During the test, resources are monitored for each instance with Folio components. The collection of system metrics with resource utilization is done by Telegraf. Telegraf is an agent for collecting, processing, aggregating, and writing metrics. Telegraf metrics will be stored on InfluxDB, then we can visualize them on Grafana using a system dashboard.

In order to display the metrics for a particular instance, you need to configure the filters at the top of the dashboard.

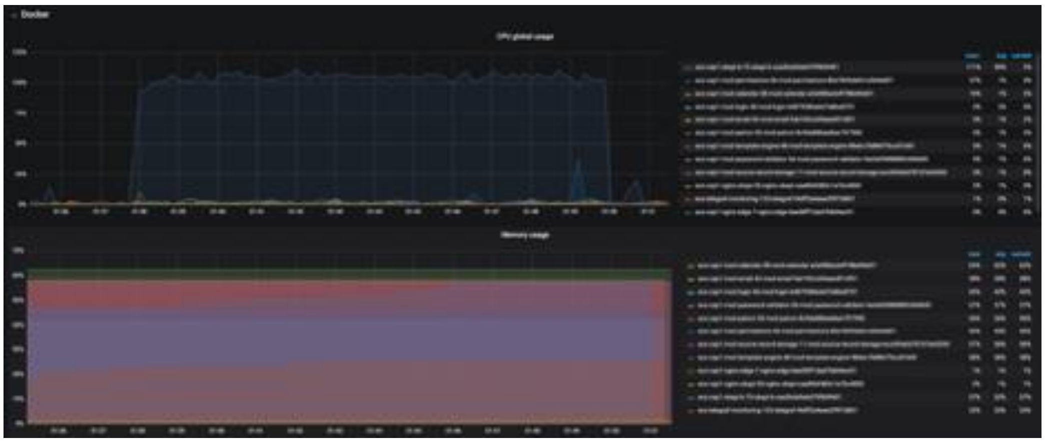


Group None
host node-3
container All

The dashboard consists of two main sections. The first is system metrics such as CPU consumption, memory, network and disk usage.



The second section is monitoring the resources inside the docker containers. It consists of two parts - CPU and memory usage. You can also filter the data in the chart for each container individually. To do this, click on the container name on the right side.



2.4 LOAD GENERATOR AWS CLOUD FORMATION TEMPLATE

Load generator AWS cloud formation template contains following parameters:

Parameter	Description
ECSCluster	Specifies the ECS Cluster Name with which the resources would be associated
IamProfile	Provides the name or the Amazon Resource Name (ARN) of the instance profile associated with the IAM role for the instance. The instance profile contains the IAM role.
SubnetIds	A list of subnet IDs for a virtual private cloud (VPC). If you specify VPCZoneIdentifier with AvailabilityZones, the subnets that you specify for this property must reside in those Availability Zones.
InstanceType	Specifies the instance type of the EC2 instance. For information about available instance types, see Available Instance Types in the Amazon EC2 User Guide for Linux Instances.
InstanceAmiId	Provides the unique ID of the Amazon Machine Image (AMI) that was assigned during registration. For more information, see Finding an AMI in the Amazon EC2 User Guide for Linux Instances.
TagName	The tags for the group.
EcsSecurityGroups	A list that contains the security groups to assign to the instances in the Auto Scaling group. The list can contain both the IDs of existing security groups and references to SecurityGroup resources created in the template.
SpotPrice	The maximum hourly price to be paid for any Spot Instance launched to fulfill the request. Spot Instances are launched when the price you specify exceeds the current Spot price. For more information, see Launching Spot Instances in your Auto Scaling Group in the Amazon EC2 Auto Scaling User Guide.
ServiceName	The name of your service. Up to 255 letters (uppercase and lowercase), numbers, and hyphens are allowed. Service names must be unique within a cluster, but you can have similarly named services in multiple clusters within a Region or across multiple Regions.
InstanceAttribute	A placement constraint for EC2 instances
LoadGeneratorsCount	The number of Amazon EC2 instances that the Auto Scaling group attempts to maintain. The number must be greater than or equal to the minimum size of the group and less than or equal to the maximum size of the group.
ReportingInstanceHost	IP address or DNS of the reporting instance (without http:// part)

LoadGeneratorMemory	Memory allocated to the perfmeter tool

Resources part consists of the following parts:

1. Cluster creation for the performance testing purpose

```
MyCluster:
 Type: AWS::ECS::Cluster
 Properties:
  ClusterName: !Ref ECSCluster
```

2. CloudWatch Logs Group creation

```
CloudwatchLogsGroup:
  Type: AWS::Logs::LogGroup
  Properties:
   LogGroupName: !Join ['-', ["ECSLogGroup", !Ref "AWS::StackName"]]
   RetentionInDays: 5
```

3. Task definition with interceptor container, more detailed info regarding to interceptor could be found here

```
taskdefinition:
  Type: AWS::ECS::TaskDefinition
  Properties:
   RequiresCompatibilities: ['EC2']
   Family: !Join ['', [!Ref "AWS::StackName", -carrier-interceptor]]
   ContainerDefinitions:
   - Name: carrier-interceptor
    Memory: 256
    Essential: 'true'
    User: root
    Image: getcarrier/interceptor:1.5
    MountPoints:
     - ContainerPath: /var/run/docker.sock
      SourceVolume: docker_sock
    Environment:
     - Name: CPU_CORES
      Value: 1
     - Name: REDIS_PASSWORD
      Value: password
     - Name: REDIS_HOST
      Value: !Ref ReportingInstanceHost
     - Name: RAM_QUOTA
      Value: !Join ['', [!Ref LoadGeneratorMemory, "g"]]
     - Name: CPU_QUOTA
      Value: 2
    LogConfiguration:
     LogDriver: awslogs
     Options:
      awslogs-group: !Ref CloudwatchLogsGroup
      awslogs-region: !Ref AWS::Region
      awslogs-stream-prefix: -carrier-interceptor
   Volumes:
    - Host:
        SourcePath: /var/run/docker.sock
      Name: docker_sock
```

4. ECSAutoScalingGroup creation with the defined parameters

```
ECSAutoScalingGroup:
  Type: AWS::AutoScaling::AutoScalingGroup
  Properties:
    VPCZoneIdentifier: !Ref SubnetIds
    LaunchConfigurationName: !Ref Instance
    MinSize: 1
    MaxSize: 10
    DesiredCapacity: !Ref LoadGeneratorsCount
    Tags:
    - Key: Name
      Value: !Ref TagName
      PropagateAtLaunch: true
  CreationPolicy:
    ResourceSignal:
      Timeout: PT15M
  UpdatePolicy:
    AutoScalingReplacingUpdate:
      WillReplace: true
```

5. Instance creation with the defined parameters. To run tests on the non-demand instance instead of spot just remove SpotPrice parameter

```
Instance:
  Type: AWS::AutoScaling::LaunchConfiguration
  Properties:
    ImageId: !Ref InstanceAmiId
    AssociatePublicIpAddress: true
    SecurityGroups: !Ref EcsSecurityGroups
    SpotPrice: !Ref SpotPrice
    InstanceType: !Ref InstanceType
    InstanceMonitoring: true
    IamInstanceProfile: !Ref IamProfile
    UserData:
      Fn::Base64: !Sub |
        #!/bin/bash -xe
        echo ECS_CLUSTER=${ECSCluster} >> /etc/ecs/ecs.config
        echo ECS_INSTANCE_ATTRIBUTES={\"instance_name\": \"${InstanceAttribute}\"} >> /etc/ecs/ecs.config
        yum install -y aws-cfn-bootstrap
        /opt/aws/bin/cfn-signal -e $? --stack ${AWS::StackName} --resource ECSAutoScalingGroup --region ${AWS::Region}
```

6. Service creation which will manage task with the interceptor and perfmeter containers

```
service:
  Type: AWS::ECS::Service
  Properties:
    Cluster: !Ref ECSCluster
    LaunchType: EC2
    DesiredCount: !Ref LoadGeneratorsCount
    TaskDefinition: !Ref taskdefinition
    ServiceName: !Ref ServiceName
    PlacementConstraints:
      - Type: memberOf
        Expression: !Join [' ', [ "attribute:instance_name ==" , !Ref InstanceAttribute]]
      - Type: distinctInstance
```

3 PROFILER INTEGRATION

3.1 JVM PROFILER

For detailed analysis of Java processes during performance testing we use an open source tool for tracing distributed JVM applications in scale - Uber JVM Profiler.

Uber JVM Profiler provides a Java Agent to collect various metrics and stack traces for JVM processes in a distributed way.

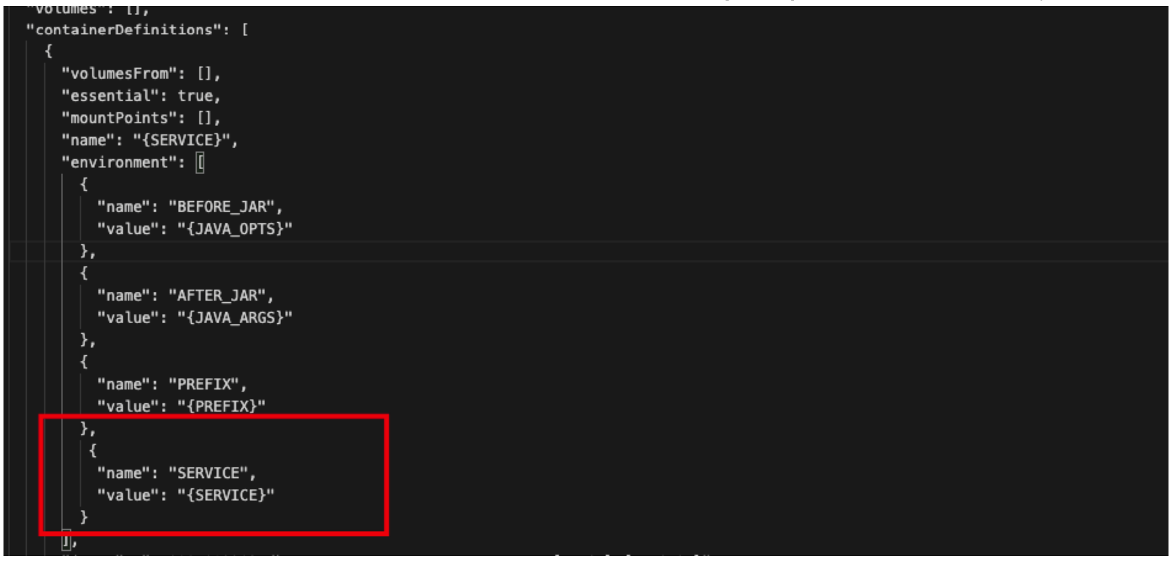
To integrate the profiler into a Java application, you need to add javaagent to the java execution line. You can download it from the github repo.

An example of starting a Java application with a profiler is presented below:

```
$ exec java -noverify -javaagent:"/usr/ms/jvm-profiler-
1.0.0.jar"=configProvider=com.uber.profiling.YamlConfigProvider,configFile="/usr/ms/config.yaml" -cp
"/usr/ms/jvm-profiler-1.0.0.jar" $BEFORE_JAR -jar $MS_JAR $AFTER_JAR
```

- "/usr/ms/jvm-profiler-1.0.0.jar" – path to the profiler .jar file;
- "/usr/ms/config.yaml" - path to the config.yaml file.

In order to be able to filter data in Grafana dashboard for each service separately, you need to pass the environment variable **SERVICE** with service name, e.g okapi-b, mod-inventory.



```
"volumes": [],
"containerDefinitions": [
  {
    "volumesFrom": [],
    "essential": true,
    "mountPoints": [],
    "name": "{SERVICE}",
    "environment": [
      {
        "name": "BEFORE_JAR",
        "value": "{JAVA_OPTS}"
      },
      {
        "name": "AFTER_JAR",
        "value": "{JAVA_ARGS}"
      },
      {
        "name": "PREFIX",
        "value": "{PREFIX}"
      },
       {
        "name": "SERVICE",
        "value": "{SERVICE}"
      }
    ],
```

You also need to create a file config.yaml and save the necessary parameters in it. An example of config.yaml file is presented below:

```
echo """reporter: com.uber.profiling.reporters.InfluxDBOutputReporter
tag: ${SERVICE}
metricInterval: 100000
durationProfiling: [com.fasterxml.*, org.*]
sampleInterval: 10000
influxdb.host: <reporting_instance_host>
influxdb.port: 8086
influxdb.database: profiling
influxdb.username:
influxdb.password:
""" > config.yaml
```

- reporter: class name for the reporter
- tag: plain text string which will be reported together with the metrics, e.g. service name
- metricInterval: how frequent to collect and report the metrics, in milliseconds.
- durationProfiling: configure to profile specific class and method, e.g. org.*, com.fasterxml.* to profile service correctly it is required to add all 3rd patry packages also.

> ***Important!***
> *Do not provide "com.*" package as the profiler has a "com.uber" package it will try to profile itself and will throw an error*

- sampleInterval: frequency (milliseconds) to do stacktrace sampling, if this value is not set or zero, the profiler will not do stacktrace sampling.
- influxdb.host: InfluxDB host DNS or IP
- influxdb.port: InfluxDB port
- influxdb.database: database name in which the metrics will be saved
- influxdb.username: InfluxDB login
- influx.password: InfluxDB password

> ***Important!***
> *It is recommended to create two revisions of tasks with and without the profiler as it uses CPU and needs to be used only for profiling purposes. Basic testing should be conducted without profiler*

3.2 POSTGRESQL SLOW QUERY PROFILER

In order for PostgreSQL to log slow queries you need to change several parameters in the file postgresql.conf and restart the service. The list of parameters is as follows:

- log_statement = 'ddl'
- log_line_prefix = '[%d] '
- log_min_duration_statement = 10 # threshold, ms (see description below)
- logging_collector = on
- log_destination = 'stderr'

You can set a threshold to query duration via log_min_duration_statement parameter. Causes the duration of each completed statement to be logged if the statement ran for at least the specified number of milliseconds. Setting this to zero prints all statement durations. Minus-one (the default) disables logging statement durations. For example, if you set it to 250ms then all SQL statements that run 250ms or longer will be logged. Enabling this parameter can be helpful in tracking down unoptimized queries in your applications.

Also, you can specify log directory via log_directory (string) parameter. When logging_collector is enabled, this parameter determines the directory in which log files will be created. It can be specified as an absolute path, or relative to the cluster data directory. This parameter can only be set in the postgresql.conf file or on the server command line. The default is pg_log.

After these changes, all slow queries to the database will be logged in a file.

In order to display the data from the file on the Grafana dashboard, you need to run the Promtail docker container on the PostgreSQL instance. Promtail is an agent which ships the contents of local logs to a private Loki instance or Grafana Cloud. It is usually deployed to every machine that has applications needed to be monitored.

Example of docker run command for the Promtail container:

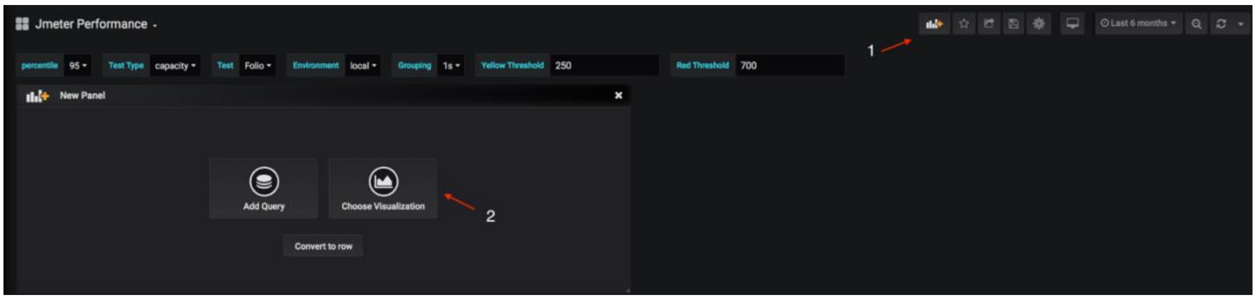
```
$ docker run -v <path_to_the_promtail-docker-config.yaml_file>:/etc/promtail/docker-config.yaml grafana/promtail:1.5 --client.url=<Loki_URL>:3100/api/prom/push -config.file=/etc/promtail/docker-config.yaml
```

You need to create a file promtail-docker-config.yaml and pass the path to it to the container run line. Content of the file:

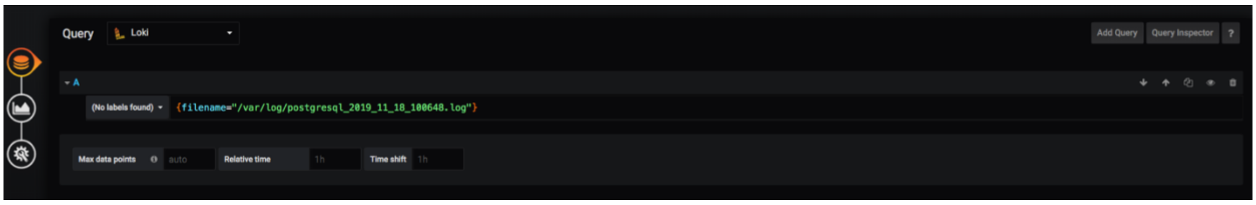
```
server:
  http_listen_port: 9080
  grpc_listen_port: 0
positions:
  filename: /tmp/positions.yaml
scrape_configs:
- job_name: system
  entry_parser: raw
  static_configs:
  - targets:
      - localhost
    labels:
      job: varlogs
      __path__: <path_to_the_postgre_logs_directory>/*log
```

After start, Promtail will send all logs from the file to Loki. Loki is a horizontally scalable, highly available, multi-tenant log aggregation system. Loki is installed together with Carrier, see p.1.1

You can visualize data from Loki in the Grafana dashboard. To do that – click "Add panel" button in the top of the Grafana dashboard and click "Choose Visualization".

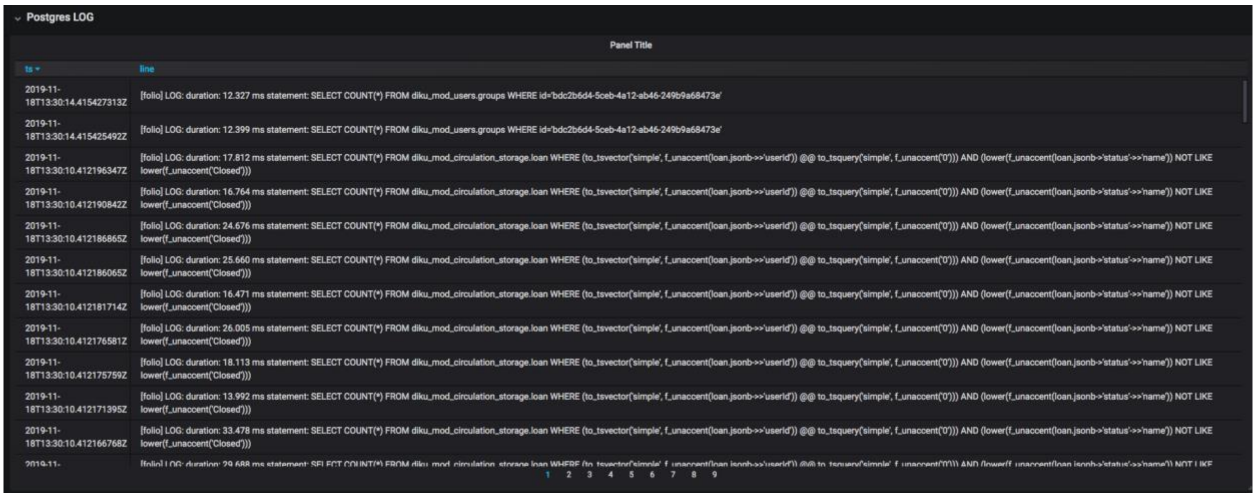


From the list of visualizations, select the table. In the "Query" field, select Loki. And then specify which file to display.



```
{filename="/var/log/postgresql_2019_11_18_100648.log"}
```

After that you will be able to see all slow PostgreSQL queries in this table.



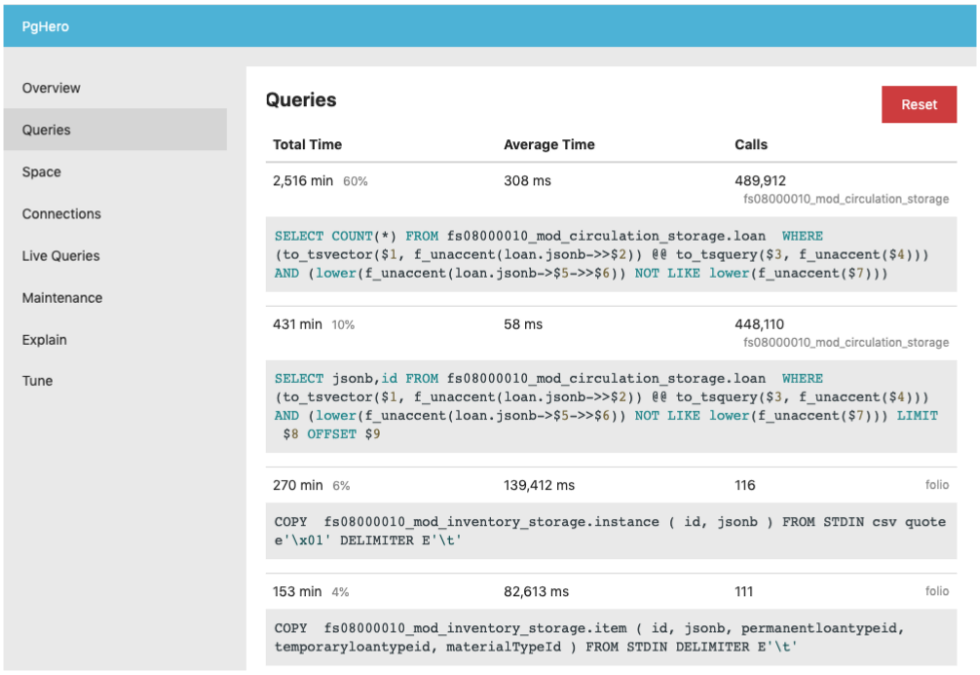
```
ts                           line
2019-11-18T13:30:14.415427313Z  [folio] LOG: duration: 12.327 ms statement: SELECT COUNT(*) FROM diku_mod_users.groups WHERE id='bdc2b6d4-5ceb-4a12-ab46-249b9a68473e'
2019-11-18T13:30:14.415425492Z  [folio] LOG: duration: 12.399 ms statement: SELECT COUNT(*) FROM diku_mod_users.groups WHERE id='bdc2b6d4-5ceb-4a12-ab46-249b9a68473e'
2019-11-18T13:30:10.412196347Z  [folio] LOG: duration: 17.812 ms statement: SELECT COUNT(*) FROM diku_mod_circulation_storage.loan WHERE (to_tsvector('simple', f_unaccent(loan.jsonb->>'userId')) @@ to_tsquery('simple', f_unaccent('0'))) AND (lower(f_unaccent(loan.jsonb->'status'->>'name')) NOT LIKE lower(f_unaccent('Closed')))
2019-11-18T13:30:10.412190842Z  [folio] LOG: duration: 16.764 ms statement: SELECT COUNT(*) FROM diku_mod_circulation_storage.loan WHERE (to_tsvector('simple', f_unaccent(loan.jsonb->>'userId')) @@ to_tsquery('simple', f_unaccent('0'))) AND (lower(f_unaccent(loan.jsonb->'status'->>'name')) NOT LIKE lower(f_unaccent('Closed')))
2019-11-18T13:30:10.412186865Z  [folio] LOG: duration: 24.676 ms statement: SELECT COUNT(*) FROM diku_mod_circulation_storage.loan WHERE (to_tsvector('simple', f_unaccent(loan.jsonb->>'userId')) @@ to_tsquery('simple', f_unaccent('0'))) AND (lower(f_unaccent(loan.jsonb->'status'->>'name')) NOT LIKE lower(f_unaccent('Closed')))
2019-11-18T13:30:10.412186065Z  [folio] LOG: duration: 25.660 ms statement: SELECT COUNT(*) FROM diku_mod_circulation_storage.loan WHERE (to_tsvector('simple', f_unaccent(loan.jsonb->>'userId')) @@ to_tsquery('simple', f_unaccent('0'))) AND (lower(f_unaccent(loan.jsonb->'status'->>'name')) NOT LIKE lower(f_unaccent('Closed')))
2019-11-18T13:30:10.412181714Z  [folio] LOG: duration: 16.471 ms statement: SELECT COUNT(*) FROM diku_mod_circulation_storage.loan WHERE (to_tsvector('simple', f_unaccent(loan.jsonb->>'userId')) @@ to_tsquery('simple', f_unaccent('0'))) AND (lower(f_unaccent(loan.jsonb->'status'->>'name')) NOT LIKE lower(f_unaccent('Closed')))
2019-11-18T13:30:10.412176581Z  [folio] LOG: duration: 26.005 ms statement: SELECT COUNT(*) FROM diku_mod_circulation_storage.loan WHERE (to_tsvector('simple', f_unaccent(loan.jsonb->>'userId')) @@ to_tsquery('simple', f_unaccent('0'))) AND (lower(f_unaccent(loan.jsonb->'status'->>'name')) NOT LIKE lower(f_unaccent('Closed')))
2019-11-18T13:30:10.412175759Z  [folio] LOG: duration: 18.113 ms statement: SELECT COUNT(*) FROM diku_mod_circulation_storage.loan WHERE (to_tsvector('simple', f_unaccent(loan.jsonb->>'userId')) @@ to_tsquery('simple', f_unaccent('0'))) AND (lower(f_unaccent(loan.jsonb->'status'->>'name')) NOT LIKE lower(f_unaccent('Closed')))
2019-11-18T13:30:10.412171395Z  [folio] LOG: duration: 13.992 ms statement: SELECT COUNT(*) FROM diku_mod_circulation_storage.loan WHERE (to_tsvector('simple', f_unaccent(loan.jsonb->>'userId')) @@ to_tsquery('simple', f_unaccent('0'))) AND (lower(f_unaccent(loan.jsonb->'status'->>'name')) NOT LIKE lower(f_unaccent('Closed')))
2019-11-18T13:30:10.412166768Z  [folio] LOG: duration: 33.478 ms statement: SELECT COUNT(*) FROM diku_mod_circulation_storage.loan WHERE (to_tsvector('simple', f_unaccent(loan.jsonb->>'userId')) @@ to_tsquery('simple', f_unaccent('0'))) AND (lower(f_unaccent(loan.jsonb->'status'->>'name')) NOT LIKE lower(f_unaccent('Closed')))
2019-11-                        [folio] LOG: duration: 29.688 ms statement: SELECT COUNT(*) FROM diku_mod_circulation_storage.loan WHERE (to_tsvector('simple', f_unaccent(loan.jsonb->>'userId')) @@ to_tsquery('simple', f_unaccent('0'))) AND (lower(f_unaccent(loan.jsonb->'status'->>'name')) NOT LIKE
```

For databases deployed in AWS RDS PgHero tool could be connected to get queries stats

1. Connect from the monitoring instance to RDS database

```
$ docker run -ti -e DATABASE_URL=postgres://<user>:<password>@<hostname>:5432/<dbname> -p 8000:8080 ankane/pghero
```

2. Open PgHero application on the http://<your_monitoring_instance_ip>:8000
3. Enable query stats in the “Overview” tab
4. Review query stats in the “Queries” tab



Queries

Total Time	Average Time	Calls
2,516 min 60%	308 ms	489,912 fs08000010_mod_circulation_storage

```
SELECT COUNT(*) FROM fs08000010_mod_circulation_storage.loan  WHERE
(to_tsvector($1, f_unaccent(loan.jsonb->>$2)) @@ to_tsquery($3, f_unaccent($4)))
AND (lower(f_unaccent(loan.jsonb->$5->>$6)) NOT LIKE lower(f_unaccent($7)))
```

Total Time	Average Time	Calls
431 min 10%	58 ms	448,110 fs08000010_mod_circulation_storage

```
SELECT jsonb,id FROM fs08000010_mod_circulation_storage.loan  WHERE
(to_tsvector($1, f_unaccent(loan.jsonb->>$2)) @@ to_tsquery($3, f_unaccent($4)))
AND (lower(f_unaccent(loan.jsonb->$5->>$6)) NOT LIKE lower(f_unaccent($7))) LIMIT
 $8 OFFSET $9
```

Total Time	Average Time	Calls
270 min 6%	139,412 ms	116 folio

```
COPY  fs08000010_mod_inventory_storage.instance ( id, jsonb ) FROM STDIN csv quote
e'\x01' DELIMITER E'\t'
```

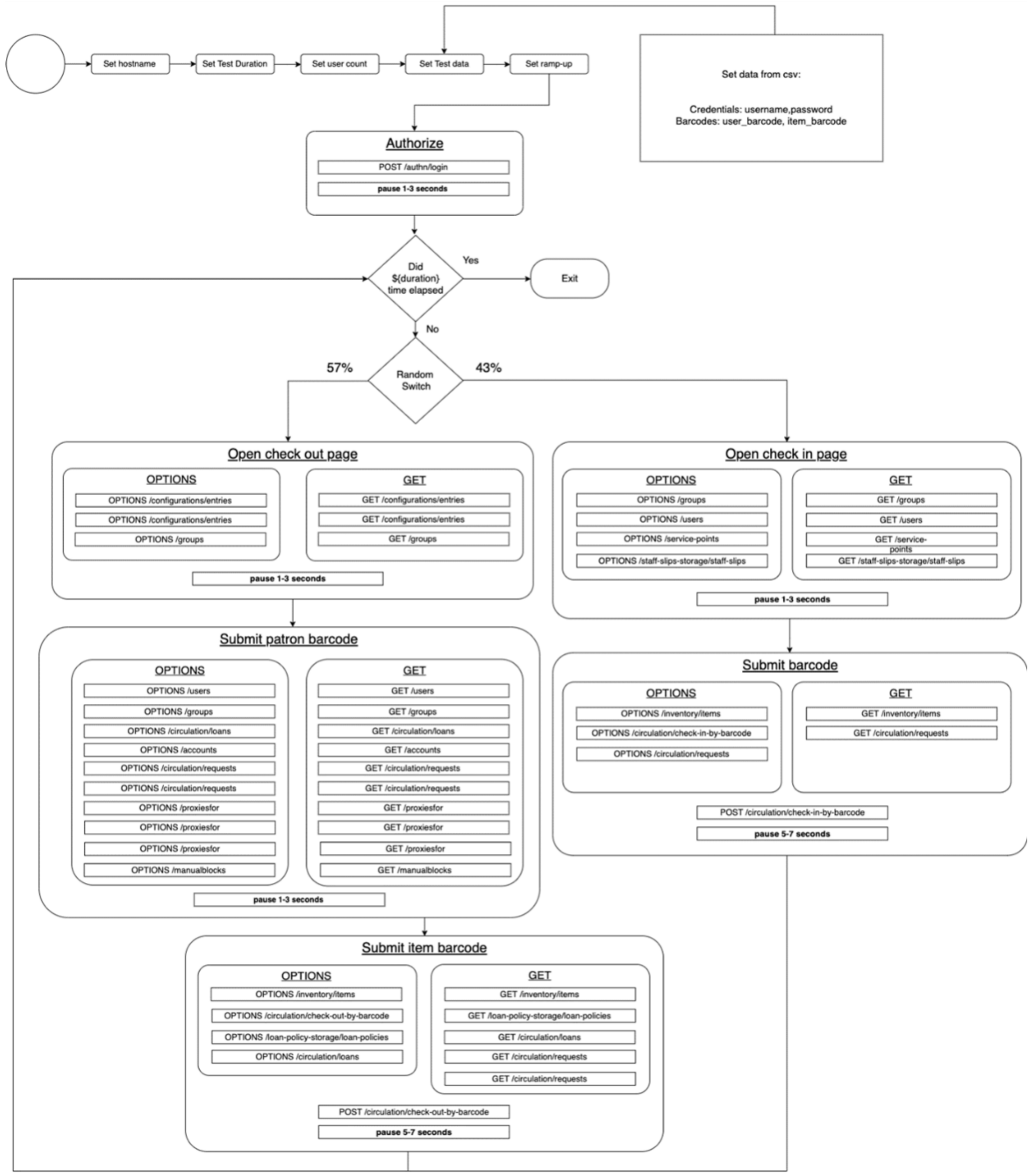
Total Time	Average Time	Calls
153 min 4%	82,613 ms	111 folio

```
COPY  fs08000010_mod_inventory_storage.item ( id, jsonb, permanentloantypeid,
temporaryloantypeid, materialTypeId ) FROM STDIN DELIMITER E'\t'
```

As alternative, RDS performance Insights could be enabled, more default information could be found here, however before enabling please read about pricing.

4 TEST STRUCTURE AND WORKFLOW

4.1 FLOW



As can be seen in the model, before starting the test, you must provide data and parameters.

4.2 FOLDER STRUCTURE AND CSV DATA

```
tests/
├── data/
│   ├── credentials.csv
│   ├── available.csv
│   ├── checked_out.csv
│   └── user_barcodes.csv
├── scripts/
│   └── BranchSelectorRandomizerScript.gvy
└── folio.jmx
```

You should put together these files and pack them into .zip archive.

You need to fill up following .csv files:

File name	Description
credentials.csv	Provides username and password provided in two columns respectively
available.csv	The list of available item barcodes, used for check-out flow
checked_out.csv	The list of checked out item barcodes, used for check-in flow
user_barcodes.csv	The list of user barcodes

Important!
Do not use white spaces in the request names as it could affect it displaying on the Grafana dashboards

If you want to scale up your JMeter test and add another flow to script, you need to follow these steps:

1. Compose your flow. Enclose each step into separate Test Fragments.
2. Go to "Define Check IN/OUT" controller and edit "Util_SWITCH User type". You need to add another "case" block to the distribution script e.g.:

```
int distribution = Integer.parseInt(vars.get("distribution"));

def result

switch (distribution){
    case 0:
        result = "checkin"
        break
    case 1:
        result = "checkout"
        break
    case 2:   //newly added "case" block
        result = "new_flow"
        break
}

vars.put("goto_check_io", result)
```

3. Add another "If Controller" with following condition:

```
"${goto_check_io}" == "new_flow"
```

Add Module Controller inside the If Contoller for each Test Fragment of your flow

4.3 CARRIER EXECUTION

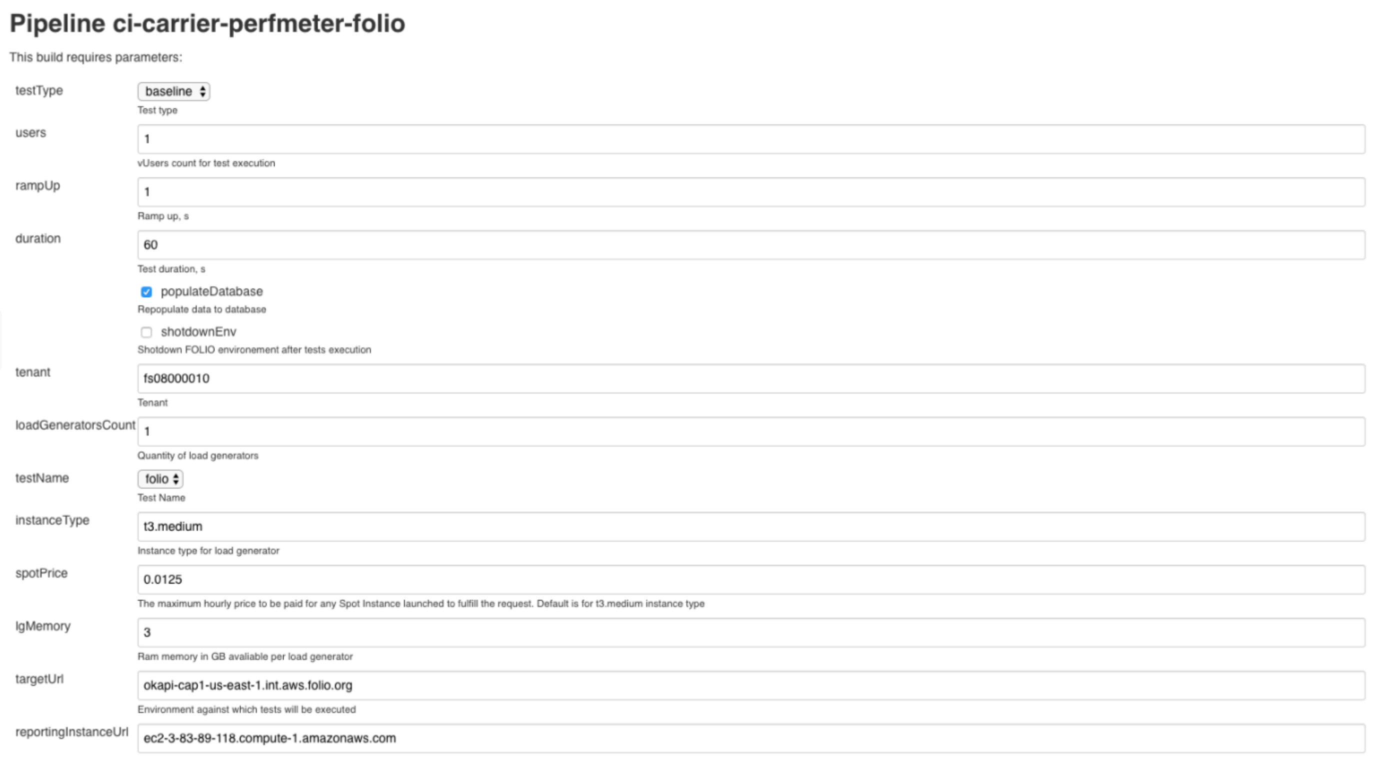
Tests are provided to carrier via artifacts.

1. Navigate to reporting instance IP
2. Go to “Artifacts” page
3. Click on the “Bucket” button on the right side of the page
4. Select “tests” bucket
5. Upload previously created .zip file via “Drop files here or click to upload.” section
6. Download perfmeter listener from the github repo.
7. Upload listener to the the bucket that you created via “Drop files here or click to upload.” section

Example syntax for test execution in docker container:

```
docker pull getcarrier/control_tower:1.5 && docker run -t --rm \
    -e REDIS_HOST=${reportingInstanceUrl} \
    -e loki_host=http://${reportingInstanceUrl} \
    -e loki_port=3100 \
    -e GALLOPER_WEB_HOOK=http://${reportingInstanceUrl}/task/${galloperTaskId} \
    -e galloper_url=http://${reportingInstanceUrl} \
    -e bucket=${bucketName}\
    -e JVM_ARGS='-Xmx${lgMemory}g' \
    -e DURATION=${duration} \
    -e additional_files='{\"tests/InfluxBackendListenerClient-1.1.jar\": \"/jmeter/apache-jmeter-5.0/lib/ext/InfluxBackendListenerClient.jar\"}' \
    -e artifact=Folio_perf_tests.zip getcarrier/control_tower:1.5 \
    -c getcarrier/perfmeter:1.5 -e '{\"cmd\": \"-n -t /mnt/jmeter/${testName}.jmx -JDISTRIBUTION=${distribution} -Jtest_name=${testName} -Jtest.type=${testType} -Jenv.type=${targetEnv} -JVUSERS=${usersCount} -JHOSTNAME=${targetUrl} -JRAMP_UP=${rampUp} -JDURATION=${duration} -Jinflux.host=${reportingInstanceUrl} \"}' -r 1 -t perfmeter -q ${loadGeneratorsCount} \
    -n supertestjob
```

4.4 PARAMETERS DESCRIPTION



Pipeline ci-carrier-perfmeter-folio

This build requires parameters:

testType	baseline Test type
users	1 vUsers count for test execution
rampUp	1 Ramp up, s
duration	60 Test duration, s
	☑ populateDatabase Repopulate data to database
	☐ shotdownEnv Shotdown FOLIO environement after tests execution
tenant	fs08000010 Tenant
loadGeneratorsCount	1 Quantity of load generators
testName	folio Test Name
instanceType	t3.medium Instance type for load generator
spotPrice	0.0125 The maximum hourly price to be paid for any Spot Instance launched to fulfill the request. Default is for t3.medium instance type
lgMemory	3 Ram memory in GB avaliable per load generator
targetUrl	okapi-cap1-us-east-1.int.aws.folio.org Environment against which tests will be executed
reportingInstanceUrl	ec2-3-83-89-118.compute-1.amazonaws.com

Parameter name	Description
testType	Optional tag, used to filter test results on Grafana
tenant	Specifies which tenant will be used for test execution and database repopulation (if enabled)
loadGeneratorsCount	Number of slave machines to run tests
testName	The test name, jmx file name (without .jmx)
instanceType	AWS instance type to use
spotPrice	Target AWS spot price, up-to-date prices for each instance type can be found here
lgMemory	Ram memory in GB available per load generator
targetUrl	Environment against which tests will be executed
reportingInstanceUrl	Galloper instance URL
galloperPostProccessingTaskId	Galloper task ID, can be obtained from task details upon task creation In Galloper. See Carrier description for details
shotdownEnv	Shutdown FOLIO environment after tests execution
populateDatabase	Defines whether to drop and repopulate the database before the test.
loki_host	Loki host or IP, used to report failed requests to Loki
artifactFile.zip	Name of .zip file with tests, see p.2.2
bucketName	Galloper bucket name, see p.2.3

loki_port	Optional, default 3100
emailsList	Recipients for email notification in format "user1@mail.com","user2@mail.com","user3@mail.com"
galloperTaskIdForNotifications	Galloper web hook for notifications, see Post processing
targetRegion	AWS region
targetEnv	Target environment
targetCluster	ECS Cluster that will be monitored

4.4.1 JMeter parameters

Parameter name	Description	Default value
HOSTNAME	Hostname to test	okapi-cap1-us-east-1.int.aws.folio.org
DISTRIBUTION	Distribution value between check-in and check-out flow. Format: {checkin}-{checkout}	43-57
VUSERS	Number of virtual users	1
RAMP_UP	The time frame (in seconds) for all users (threads) to start	14
DURATION	The time frame (in seconds) for the test execution	300

4.5 TEST EXECUTION AND ANALYSIS

In order to analyze test results, you need to set up Grafana and InfluxDB before execution.

The results of test execution can be found on

- Grafana dashboards: provides you response times, metrics, charts and resource utilization metrics
- PgHero: provides additional metrics. Here you can observe slow queries, their execution times and relations, database space usage, and some basic information about connections between mods.

Read more about InfluxBD/Grafana setup and analysis process at the Carrier Wiki page on GitHub.

4.6 CI/CD INTEGRATION

Jenkinsfile for Jenkins job can be found on ci-perf repository. It provides all essential steps for test execution pipeline, including carrier container execution, database repopulation and AWS instances start-up.

> ***Important!***
> *To execute aws and docker commands you need Jenkins with installed aws cli at least 1.16.307 version and docker v18.09+*

4.7 MANUAL TEST EXECUTION

If you want to run tests manually, you can use Carrier-customized JMeter container. It features:

1. InfluxDB Listener, which is automatically added to test for reporting
2. Failed requests automatically reported to Loki and displayed in Grafana dashboard
3. Post processing of execution metrics automatically logs aggregated states for build-to-build comparison
4. Support of masterless distributed tests execution using control-tower and interceptor
5. Post processing and Email notifications

4.8 LOCAL EXECUTION

1. Install docker
2. Zip you test files
3. Navigate to reporting instance IP
4. Go to Artifacts page
5. Click on the Bucket button on the right side of the page
6. Select “tests” bucket
7. Upload previously created .zip file via “Drop files here or click to upload.” section
8. Download perfmeter listener from the github repo.
9. Upload listener to the the bucket that you created via “Drop files here or click to upload.” section
10. Start container and pass the necessary config options to container

Example docker invocation:

```
docker run --rm -u 0:0 \
       -e galloper_url=http://${reportingInstanceUrl} \
       -e bucket=${bucketName} -e artifact=${artifactFile}.zip \
       -e loki_host=http://${reportingInstanceUrl} \
       -e JVM_ARGS='-Xms1g -Xmx2g' \
       -e additional_files='{"tests/InfluxBackendListenerClient-1.1.jar": "/jmeter/apache-jmeter-5.0/lib/ext/InfluxBackendListenerClient.jar"}' \
       getcarrier/perfmeter:1.5 \
       -n -t /mnt/jmeter/${testName}.jmx \
       -JDISTRIBUTION=${distribution} \
       -Jtest_name=${testName} \
       -Jtest.type=${testType} \
       -Jenv.type=${targetEnv} \
       -JVUSERS=${usersCount} \
       -JHOSTNAME=${targetUrl} \
       -JRAMP_UP=${rampUp} \
       -JDURATION=${duration} \
       -Jinflux.host=${reportingInstanceUrl}
```

Parameter name	Description
reportingInstanceUrl	Galloper instance URL
test_type	Optional tag, used to filter test results

env	Optional tag, used to filter test results
loki_host	Loki host or IP, used to report failed requests to Loki
loki_port	Optional, default 3100
test_name	Name of the JMeter test file that will be run
JVM_ARGS	Java heap params, like '-Xms1g -Xmx1g'
artifactFile.zip	Name of .zip file with tests, see p.2.2
bucketName	Galloper bucket name, see p.2.3

5 TEST DATA GENERATION TOOL

To perform volume tests, you may need to have a list of datasets with different sizes. A data generator has been developed for this purpose, which increases the default dataset to a specified value. The three main files for which it generates data:

- items.tsv
- instances.tsv
- recordholding.tsv

You can also use the additional parameter to specify how many items should be in the "checked out" status.

You can download test data generation tool from the git repo.

The first thing you need to do is build the docker container with generator. To do this, browse to the Dockerfile folder and run the command:

```
 $ docker build -t generator .
```

After that you can start the generator with the following command:

```
$ docker run --rm -e count=1000000 -e loans=10000 -v <path_to_the_folder>:/tmp generator
```

`<path_to_the_folder>` - folders for generated files

`-e count=1000000` - number of items, holdings and instances to generate

`-e loans=10000` - number of items with "checked out" status. Optional, default - 10% of total items count.

After that, the generator will create files in the directory you provide with <path_to_the_folder> parameter. Also, it creates 2 additional csv files with barcodes. These files are using in JMeter test. So, you need to add these files to the zip file with JMeter script.

6 TROUBLESHOOTING GUIDE

This troubleshooting Guide is intended to provide guidance to developers on the detection and correction performance infrastructure issues.

Term	Detail
Issue	Spot instance creation has stack
Cause	Your Spot request price of <price> is lower than the minimum required Spot request fulfillment price of <price>. Launching EC2 instance failed.
Solution	1. Open Cloudformation in AWS console 2. Select “performance-testing-load-generators” stack 3. Open “Events” tab and check if there any errors 4. Open “Resources” tab 5. Open ECSAutoScalingGroup Physical ID link 6. Select ECSAutoScalingGroup 7. Open Activity History at the bottom of the page 8. Check fail description 9. Check the fulfillment price and pass it in the Jenkins job parameter

Term	Detail
Issue	An error occurred (AlreadyExistsException) when calling the CreateStack operation: Stack [performance-testing-load-generators] already exists
Cause	performance-testing-load-generators was not deleted successfully
Solution	1. Open Cloudformation in AWS console 2. Check “performance-testing-load-generators” stack 3. Refresh stacks list 4. Try to delete stack manually from the AWS Console 5. Execute command using aws cli `$ aws cloudformation delete-stack --stack-name --stack-name performance-testing-load-generators`

Term	Detail
Issue	Interceptor runs random test configuration
Cause	Unresolved task remains in the Redis db and interceptor pick ups it firstly instead of currently selected task
Solution	1. ssh to the reporting instance 2. restart redis database using the following command `$ docker run -d -p 6379:6379 --name carrier-redis redis:5.0.3 redis-server --requirepass password --network=carrier_default`

Term	Detail
Issue	Reporting instance IP address was changed due to machine restarting
Cause	EC2 assings IP dynamically after instance restart

Solution	1. Delete current galloper container
	`$ docker rm -f <container_id>`
	2. restart galloper service
	`$ docker run -d -ti \`

```
-v /var/run/docker.sock:/var/run/docker.sock \
--network=carrier_default \
--restart=unless-stopped \
-e REDIS_DB=2 \
-e REDIS_HOST=carrier-redis \
-e CPU_CORES=2 \
-e APP_HOST=http://<reporting_instance_ip> \
-e MINIO_HOST=http://carrier-minio:9000 \
-e MINIO_ACCESS_KEY=admin \
-e MINIO_SECRET_KEY=password \
-e MINIO_REGION=us-east-1 \
-l 'traefik.backend=galoper' \
-l 'traefic.port=5000' \
-l 'traefik.frontend.rule=PathPrefixStrip: /' \
-l 'traefik.frontend.passHostHeader=true' \
--name carrier-galloper getcarrier/galloper:latest
```

Term	Detail
Issue	FileNotFoundError: [Errno 2] No such file or directory:
Cause	Folder structure was not respected
Solution	Plase check the folder structure in Folder structure and CSV data

Term	Detail
Issue	Controll tower throws “Trapping” or any other exeption
Cause	Interceptor was not started successfully or failed
Solution	1. Open AWS ECS console 2. Select “performance-testing” cluster 3. Open “Tasks” tab in the cluster info tab 4. Click on “carrier-interceptor” task id 5. Open “Logs” tab on the Task Details tab 6. Check logs 7. If there is an issues abour redis connection - restart redis db on the reporting instance `$ docker run -d -p 6379:6379 --name carrier-redis --network=carrier_default redis:5.0.3 redis-server --requirepass password` 8. If there is resources consumption issue - scale-up instance type for load generator and provide more RAM memory in the “lgMemory: parameter

Term	Detail
Issue	Jmeter test execution suddelny drops

Cause	There are not enough resources for load generator
Solution	1. Go to “Telegraf metrics” Grafana dashboard 2. Select load generator server 3. Check CPU, Memory and Java memory metrics 4. Increase instance type for load generator and provide more memory

Term	Detail
Issue	Jmeter metrics are absent in the InfluxDB
Cause	Jmeter listener does not have an access to the InfluxDB instance
Solution	Make sure that you opened all required porst for you load generator instance cidr

Term	Detail
Issue	Not using tags are displayed on the Grafana dashboard
Cause	
Solution	1. ssh to the reporting instance 2. execute command `$ docker exec -ti carrier-influx influx` 3. select database `$ use jemeter` 4. remove all data with desired tag `$ drop series where <tag_name>=’<tag_value>’` ***Important!*** *Be careful because if you accidentally drop important data it will be lost forever*