BlazeMeter & GitLab Integration

Version 1.1

Table of Contents

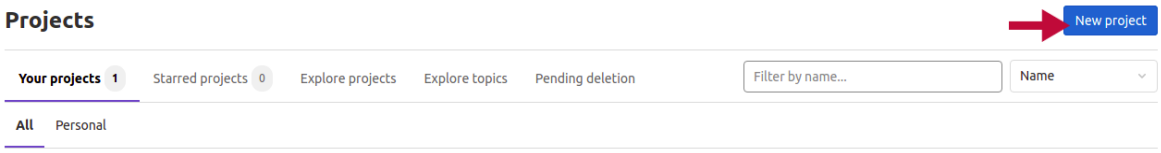
Introduction

This document is aimed at those customers that would like to create and run BlazeMeter tests from their GitLab CI/CD. By design, GitLab CI/CD relies on runners on which jobs get executed inside docker containers. A BlazeMeter docker image was developed for this integration that performs various functions, including but not limited to, creating a test, running an existing test, updating the test files of an existing test etc. The full list of functions performed by the docker image are detailed out in the later sections of this document.

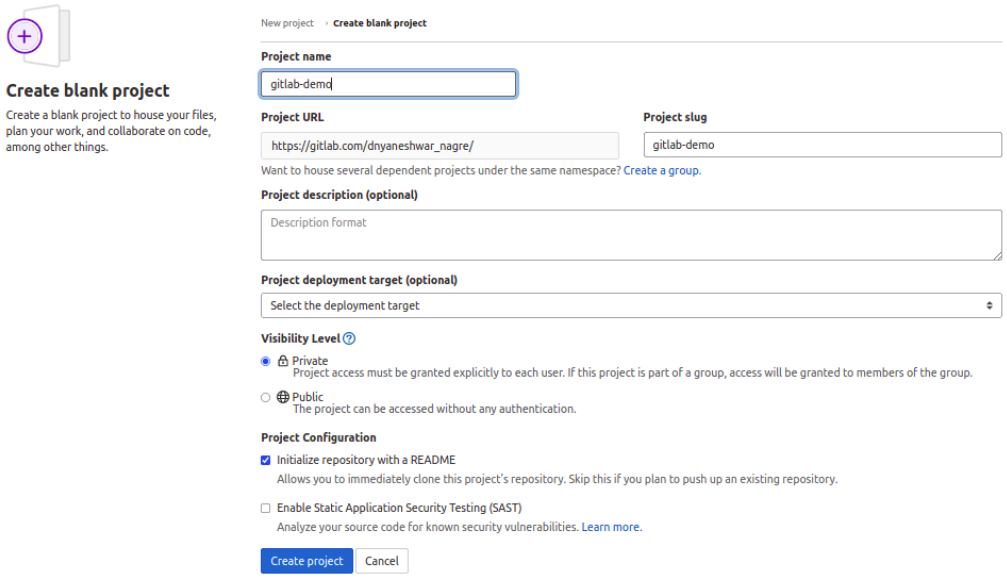
Getting Started

This section details the steps involved in creation of a simple job in GitLab CI/CD.

- Login to your GitLab CI/CD account
- Once you are successfully logged in, click on New Project on the top right corner.

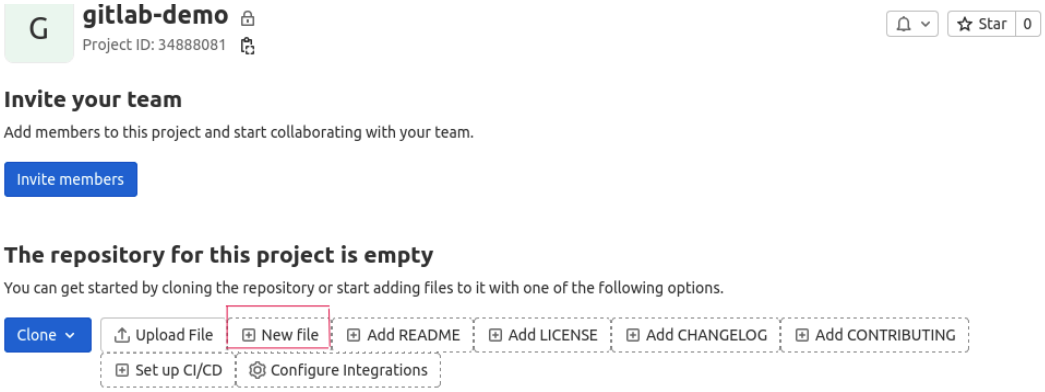


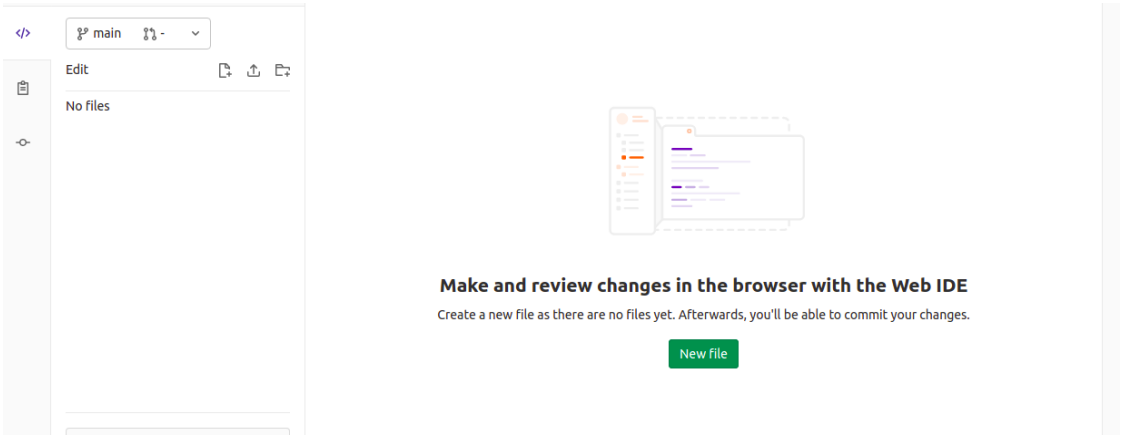
- Create a project as shown below

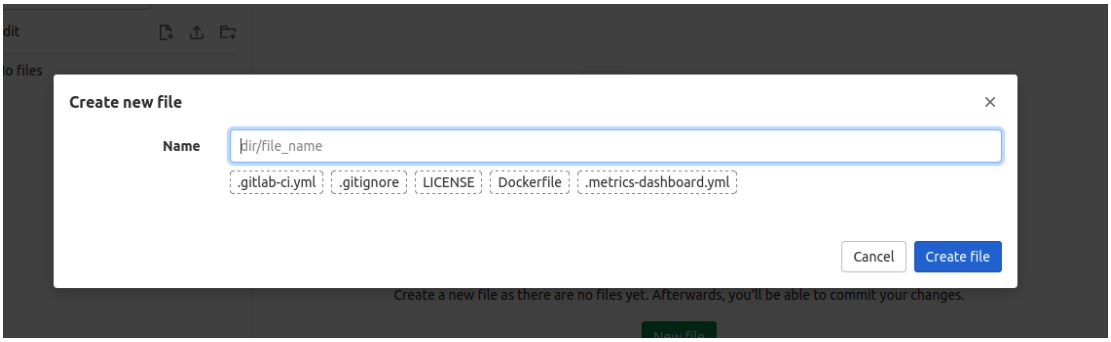


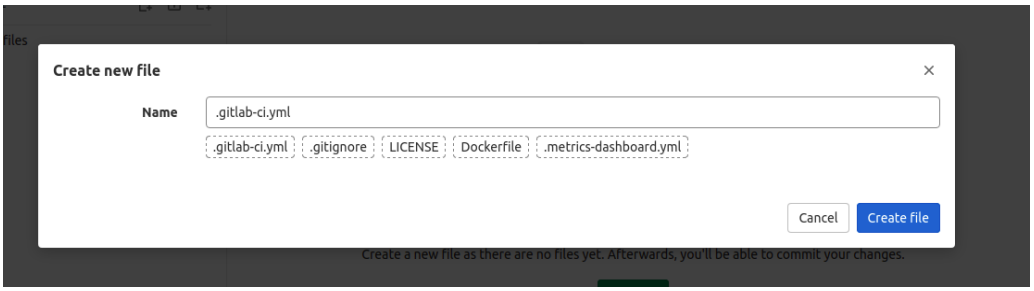
- GitLab CI/CD relies on a .gitlab-ci.yml file where you define the following:
    a. The structure and order of jobs that the runner should execute.
    b. The decisions the runner should make when specific conditions are encountered.

- Follow the screenshots below to create your .gitlab-ci.yml file.

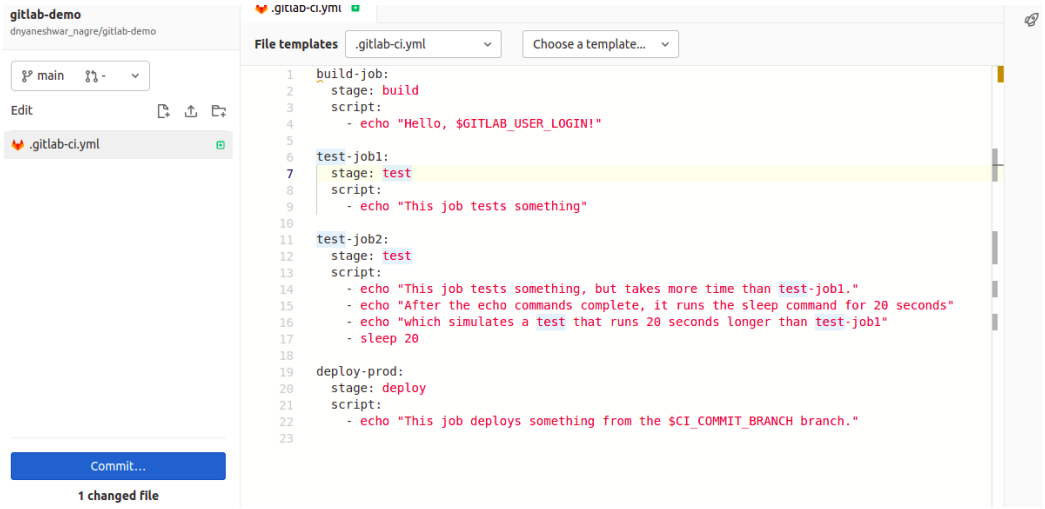




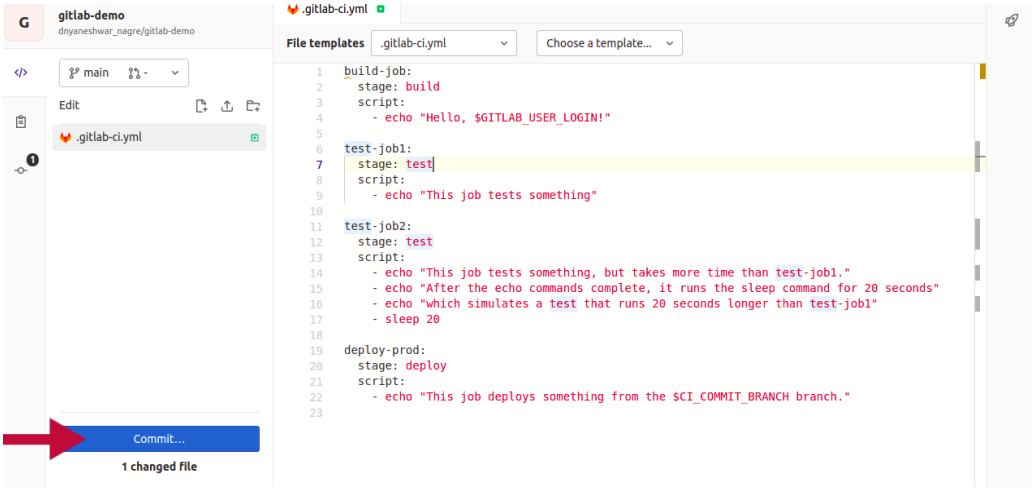




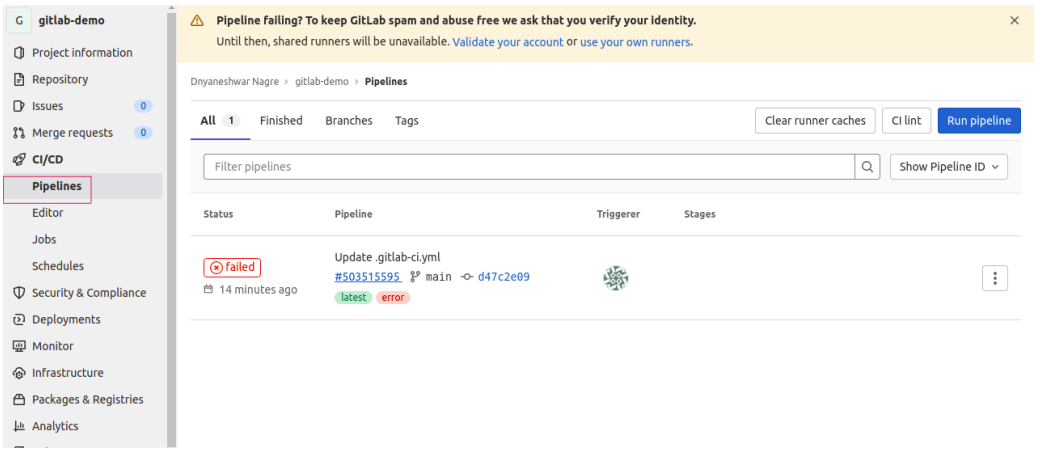
- Below is a screenshot of a sample .gitlab-ci.yml file that prints a bunch of statements to the console.



- Click ***Commit*** to save your changes.



- You may now run the jobs in your pipeline by navigating to CI/CD >> Pipelines and clicking on the Run Pipeline button.



BlazeMeter Docker Image & its Functions

The docker image lets you perform several functions. This section details each function along with syntax that you need to use in the .gitlab-ci.yml file.

Create New Test

Follow the configuration below to create and execute a new test in BlazeMeter.

```
image: blazerunner/blazemeter:gitlab

variables:
  apiKey: "xxxx"
  apiSecret: "xxxx"
  createTest: "true"
  inputStartFile: "xxxx"
  testName: "xxxx"
  projectId: "xxxx"
  continuePipeline: "false"

job 0:
  stage: deploy
  script:
    - "pwsh /Blazemeter-run.ps1 -apiKey $apiKey -apiSecret $apiSecret
-createTest $createTest -testName $testName -projectId $projectId -
inputStartFile $inputStartFile -continuePipeline $continuePipeline"
```

Run Existing Test

Follow the configuration below to run an existing BlazeMeter test.

```
image: blazerunner/blazemeter:gitlab

variables:
  apiKey: "xxxx"
  apiSecret: "xxxx"
  testIdInput: "xxxx"
  continuePipeline: "false"

job 0:
  stage: deploy
  script:
    - "pwsh /Blazemeter-run.ps1 -apiKey $apiKey -apiSecret $apiSecret
-testIdInput $testIdInput -continuePipeline $continuePipeline"
```

Add or Replace Files Existing Test

Replace or Add the dependency files before running the test

If an existing BlazeMeter test of yours requires you to have dependent files, follow the configuration below to upload those files to an existing test and run it.

If your test requires multiple files to be uploaded, you must put all those files under a folder and reference the folder path inside the ***inputAllFiles*** variable.

```
image: blazerunner/blazemeter:gitlab

variables:
  apiKey: "xxxx"
  apiSecret: "xxxx"
  testIdInput: "xxxx"
  inputAllFiles: "xxxx"
  uploadFileCheck: "true"
  continuePipeline: "false"

job 0:
  stage: deploy
  script:
    - "pwsh /Blazemeter-run.ps1 -apiKey $apiKey -apiSecret $apiSecret
-testIdInput   $testIdInput   -uploadFileCheck   $uploadFileCheck   -
inputAllFiles $inputAllFiles -continuePipeline $continuePipeline"
```

Replace or Add the configuration files (main test file) before running the test

Follow the configuration below to update the main test file of an existing test and run it.

```
image: blazerunner/blazemeter:gitlab

variables:
  apiKey: "xxxx"
  apiSecret: "xxxx"
  testIdInput: "xxxx"
  uploadFileCheck: "true"
  inputStartFile: "xxxx"
  continuePipeline: "false"

job 0:
  stage: deploy
  script:
    - "pwsh /Blazemeter-run.ps1 -apiKey $apiKey -apiSecret $apiSecret
-testIdInput   $testIdInput   -uploadFileCheck   $uploadFileCheck   -
inputStartFile $inputStartFile -continuePipeline $continuePipeline"
```

Update variables in Taurus Files

If your test is defined as a Taurus .yaml file and it contains variables that you would like to be updated at run-time, then the ***envVariable*** feature comes handy.

Let's assume you have a variable called username in your Taurus .yaml file and you want the value of that variable to be set to jdoe. You may do so by passing the key-value pair as username:jdoe, which then replaces the variable username in your .yaml file with the value jdoe.

This works only in combination with the ***inputStartFile*** variable, which means that the file gets updated locally on the runner and the updated file gets uploaded to your BlazeMeter test before execution.

```
image: blazerunner/blazemeter:gitlab

variables:
  apiKey: "xxxx"
  apiSecret: "xxxx"
  createTest: "true"
  inputStartFile: "xxxx"
  testName: "xxxx"
  projectId: "xxxx"
  envVariable: '{"key":"value"}'
  continuePipeline: "false"

job 0:
  stage: deploy
  script:
    - "pwsh /Blazemeter-run.ps1 -apiKey $apiKey -apiSecret $apiSecret
-createTest $createTest -testName $testName -projectId $projectId -
inputStartFile    $inputStartFile    -envVariable    $envVariable    -
continuePipeline $continuePipeline"
```

Run Existing Test and Download Artifacts log file

Follow the configuration below to run an existing test and download the artifacts to the CI Project directory.

```
image: blazerunner/blazemeter:gitlab

variables:
  apiKey: "xxxx"
  apiSecret: "xxxx"
  testIdInput: "xxxx"
  continuePipeline: "false"

job 0:
  stage: deploy
  script:
    - "pwsh /Blazemeter-run.ps1 -apiKey $apiKey -apiSecret $apiSecret
-testIdInput $testIdInput -continuePipeline $continuePipeline"
    - ls -a ./tmp/artifacts
  artifacts:
    paths:
      - $CI_PROJECT_DIR/tmp/artifacts
```

Add JMeter Properties

If your test is JMeter based and you would like to add/update a JMeter property, you may do it by using the ***jmeterProperties*** variable as shown in the example below.

```
image: blazerunner/blazemeter:gitlab

variables:
  apiKey: "xxxx"
  apiSecret: "xxxx"
  testIdInput: "xxxx"
  continuePipeline: "false"
  jmeterProperties: "key=value"

job 0:
  stage: deploy
  script:
    - "pwsh /Blazemeter-run.ps1 -apiKey $apiKey -apiSecret $apiSecret
-testIdInput  $testIdInput  -continuePipeline  $continuePipeline  -
jmeterProperties $jmeterProperties"
    - ls -a ./tmp/artifacts
  artifacts:
    paths:
      - $CI_PROJECT_DIR/tmp/artifacts
```

Customize the Report Name

Use the ***reportName*** parameter to give a name to your test report.

```
image: blazerunner/blazemeter:gitlab

variables:
  apiKey: "xxxx"
  apiSecret: "xxxx"
  testIdInput: "xxxx"
  reportName: "xxxx"
  continuePipeline: "false"

job 0:
  stage: deploy
  script:
    - "pwsh /Blazemeter-run.ps1 -apiKey $apiKey -apiSecret $apiSecret
-testIdInput $testIdInput -reportName $reportName -continuePipeline
$continuePipeline"
```

Add notes to your Test Report

Use the ***note*** parameter to add notes to your test report.

```
image: blazerunner/blazemeter:gitlab

variables:
  apiKey: "xxxx"
  apiSecret: "xxxx"
  testIdInput: "xxxx"
  note: "xxxx"
  continuePipeline: "false"

job 0:
  stage: deploy
  script:
    - "pwsh /Blazemeter-run.ps1 -apiKey $apiKey -apiSecret $apiSecret
-testIdInput    $testIdInput    -note    $note    -continuePipeline
$continuePipeline"
```

Override iterations configuration in existing JMeter tests

If your JMeter test is iteration based and not duration based, then you may override the iterations value in the JMX through the use of the variables ***iterationsConfig*** and ***iterations*** as shown in the example below.

```
image: blazerunner/blazemeter:gitlab

variables:
  apiKey: "xxxx"
  apiSecret: "xxxx"
  testIdInput: "xxxx"
  iterationsConfig: "true"
  iterations: "xxxx"
  continuePipeline: "false"

job 0:
  stage: deploy
  script:
    - "pwsh /Blazemeter-run.ps1 -apiKey $apiKey -apiSecret $apiSecret
-testIdInput  $testIdInput  -iterationsConfig  $iterationsConfig  -
iterations $iterations -continuePipeline $continuePipeline"
```

Override iterations configuration in JMX and create a new test

This function is similar to the previous one except for the fact that it also creates a new test on the fly.

```
image: blazerunner/blazemeter:gitlab

variables:
  apiKey: "xxxx"
  apiSecret: "xxxx"
  createTest: "true"
  inputStartFile: "xxxx"
  testName: "xxxx"
  projectId: "xxxx"
  iterationsConfig: "true"
  iterations: "xxxx"
  continuePipeline: "false"

job 0:
  stage: deploy
  script:
    - "pwsh /Blazemeter-run.ps1 -apiKey $apiKey -apiSecret $apiSecret -createTest $createTest -inputStartFile $inputStartFile -testName $testName -projectId $projectId -iterationsConfig $iterationsConfig -iterations $iterations -continuePipeline $continuePipeline"
```

Override the load configuration parameters - Concurrency, Ramp Up and Duration

Follow the configuration below to override the load configuration parameters.

```
image: blazerunner/blazemeter:gitlab

variables:
  apiKey: "xxxx"
  apiSecret: "xxxx"
  createTest: "true"
  inputStartFile: "xxxx"
  testName: "xxxx"
  projectId: "xxxx"
  totalUsers: "xxxx"
  duration: "xxxx"
  rampUp: "xxxx"
  continuePipeline: "false"

job 0:
  stage: deploy
  script:
    - "pwsh /Blazemeter-run.ps1 -apiKey $apiKey -apiSecret $apiSecret
-createTest $createTest -inputStartFile $inputStartFile -testName
$testName -projectId $projectId -totalUsers $totalUsers -duration
$duration -rampUp $rampUp -continuePipeline $continuePipeline"
```

Update Test Data

This function is specific to Scriptless Functional Tests that leverage the Test Data feature. When you use the Test Data module in your tests, you are basically referencing variables in your test that are managed by the Test Data module. These variable values can be overridden from your

pipeline by use of the ***modelData*** parameter. It requires the use of key-value pairs as shown below.

```
image: blazerunner/blazemeter:gitlab

variables:
  apiKey: "xxxx"
  apiSecret: "xxxx"
  testIdInput: "xxxx"
  modelData: '{"key":"value"}'
  continuePipeline: "false"

job 0:
  stage: deploy
  script:
    - "pwsh /Blazemeter-run.ps1 -apiKey $apiKey -apiSecret $apiSecret
-testIdInput  $testIdInput  -modelData  $modelData  -continuePipeline
$continuePipeline"
```

Run an existing test by passing the Test Name instead of Test ID

You may execute tests by passing the Test Name as opposed to a Test ID as shown below. If the Test Name passed does not exist on the BlazeMeter side, the job fails with an error message.

```
image: blazerunner/blazemeter:gitlab

variables:
  apiKey: "xxxx"
  apiSecret: "xxxx"
  continuePipeline: "false"
  showTailLog: "false"
  testName: "xxxx"
  projectId: "xxxx"
  testRunByTestName: "true"

job 0:
  stage: deploy
  script:
    - pwsh /Blazemeter-run.ps1 -apiKey $apiKey -apiSecret $apiSecret -
continuePipeline $continuePipeline -showTailLog $showTailLog -testName
```

```
$testName -projectId $projectId -testRunByTestName $testRunByTestName
    - ls -a ./tmp/artifacts
  artifacts:
    paths:
      - $CI_PROJECT_DIR/tmp/artifacts
```

Appendix: Docker Image Variables Explained

There are several variables available for use with the BlazeMeter docker image. This section describes the purpose of each of these variables.

- apiKey
    - User's BlazeMeter API Key
    - DataType: String
    - Format
        - - apiKey <apiKey>
- apiSecret
    - User's BlazeMeter API Secret
    - DataType: String
    - Format
        - -apiSecret <apiSecret>
- testIdInput
    - BlazeMeter Test ID
    - DataType: String
    - Format
        - -testIdInput <testIdInput>
- showTailLog
    - If enabled, shows a running log in real-time in the Gitlab console
    - DataType: Boolean (default is true)
    - Format
        - -showTailLog <showTailLog>
- createTest
    - If enabled, creates a new test in BlazeMeter
    - DataType: Boolean (default is false)
    - Format
        - -createTest <createTest>
- testName
    - Name of the BlazeMeter Test
    - DataType: String
    - Format
        - -testName <testName>
- inputAllFiles
    - Used when uploading multiple files while creating or updating the test
    - DataType: String
    - This needs ***uploadFileCheck*** flag to be set to true
    - Format
        - -inputAllFiles <inputAllFiles>
- inputStartFile
    - Used when uploading single start file while test creation
    - DataType: String
    - Format
        - -inputStartFile <inputStartFile>

- uploadFileCheck
    - When Used in conjunction with inputAllFiles, this must be set to true
    - DataType: Boolean (default is false)
    - Format
        - -uploadFileCheck <uploadFileCheck>
- totalUsers
    - Number of target concurrent virtual users
    - DataType: Integer (default is 20)
    - Format
        - -totalUsers <totalUsers>
- duration
    - Time to hold target concurrency (in minutes)
    - DataType: Integer (default is 20)
    - Format
        - -duration <duration>
- projectId
    - BlazeMeter Project ID
    - DataType: String
    - Format
        - -projectId <projectId>
- rampUp
    - Ramp-up time to reach target concurrency (in minutes)
    - DataType: String (default is 1 minute)
    - Format
        - -rampUp <rampUp>
- continuePipeline
    - When set to false, the pipeline waits until the test execution completes (holding up any other subsequent tasks
    - When set to true, the job just kicks off the BlazeMeter test, but continues the execution of subsequent tasks in the pipeline, if any
    - Artifacts can be downloaded to Gitlab CI Directory only if this is set to false
    - DataType: Boolean (default is true)
    - Format
        - -continuePipeline <continuePipeline>
- multiTests
    - If the test being kicked off is a multi-test, this value must be set to true
    - DataType: Boolean (default is false)
    - Format
        - -multiTests <multiTests>
- functionalTest
    - If a functional test suite is being kicked off, this value must be set to true
    - DataType: Boolean (default is false)

    - Format
        - -functionalTest <functionalTest>
- modelData
    - Used in cases where the variables inside the Test Data Model need to be updated
    - DataType: String (Only Key Value Pairs are accepted in the format shown below)
    - Format
        - -modelData <'{"email": "\"$EMAIL\"", "password": "\"$PASSWORD\""}'>
- envVariable
    - This is similar to the modelData parameter. The only difference in this case is that instead of updating the test data, this parameter is used to update any variables in an existing Taurus .yaml file
        - For instance, if a Taurus .yaml file has two parameters ***username*** and ***password.*** To update those values, you may use the format below.
    - Format
        - -envVariable <"{'username':'$USERNAME','password':'$PASSWORD'}">
- jmeterProperties
    - Used to add Jmeter Properties
    - DataType: String (Key-Value Pairs as shown below)
    - Format
        - -jmeterProperties <”key=value”>
- reportName
    - Report Name in BlazeMeter
    - DataType: String
    - Format
        - -reportName <reportName>
- note
    - Notes section of a given report in BlazeMeter
    - DataType: String
    - Format
        - -note <note>
- iterationsConfig
    - Run a test based on iterations and not duration
    - DataType: Boolean (default is false)
        - -iterationsConfig <iterationsConfig>
- iterations
    - In case of iterations based test, set the number of iterations
    - DataType: Integer (default is 1)
    - Format
        - -iterations <iterations>
- testRunByTestName
    - Run a test by its name as opposed to its ID

    - DataType: Boolean (default is false)
    - Format
        - - testRunByTestName <testRunByTestName>
- ignoreSLA
    - When set to true, the job always returns a ***Success***
    - DataType: Boolean (default is false)
    - Format
        - - ignoreSLA <ignoreSLA>