**JMeter**

**Настройка, простые тесты, переменные, load testing, spike testing, stability testing**

Выполнил: Герман Буйницкий

*Перед работой установим дополнительно плагины, которые понадобятся в работе.*

Заходим на сайт <https://jmeter-plugins.org/install/Install/>

Сохраняем файл plugins-manager.jar

Закидываем скачанный файл в JMeter по адресу:

C:\apache-jmeter-5.4.1\apache-jmeter-5.4.1\lib\ext

И перезапускаем JMeter, если он запущен.

Заходим в Options – Plugins Manager.

В открывшемся окне переключаемся на Available Plugins и качаем нужные плагины, после этого жмем на кнопку в нижней части окна «Apply Changes and Restart JMeter»

Устанавливал Custom Thread Groups, 3 Basic Graphs, 5 Additional Graphs, KPI vs KPI Graphs.

**Создание нового набора потоков**

Test plan – правой клавишей мыши (ПКМ) – Add – Thread (Users) – Thread Group.

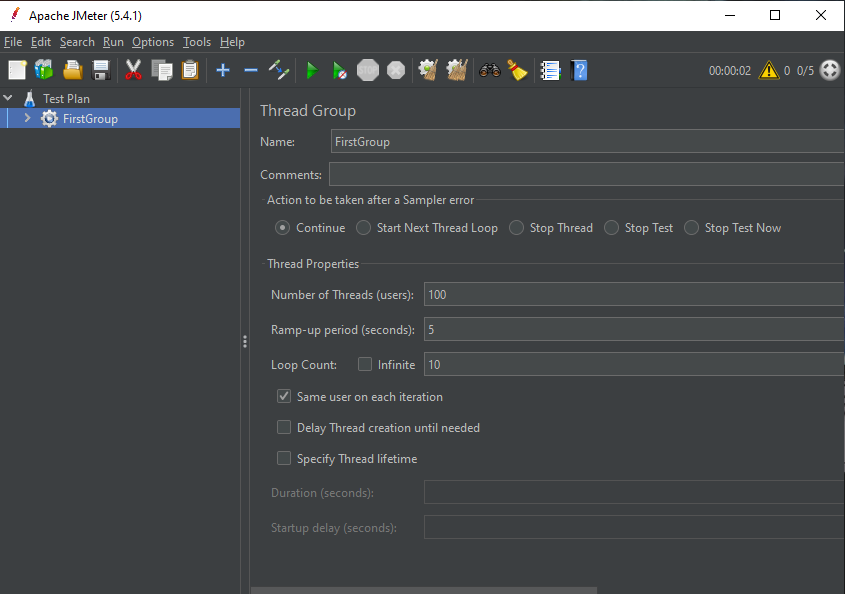
Вводим название для группы - FirstGroup

Выбираем чекбокс «Continue»

Количество юзеров в прогоне: 100

Количество секунд Ramp-up: 5

Loop Count – сколько раз пользователи выполнят свои сценарии: 10



Добавляем запросы.

Жмем ПКМ на созданную “FirstGroup” – Add – Sampler – HTTP request.

Name: GetRequest

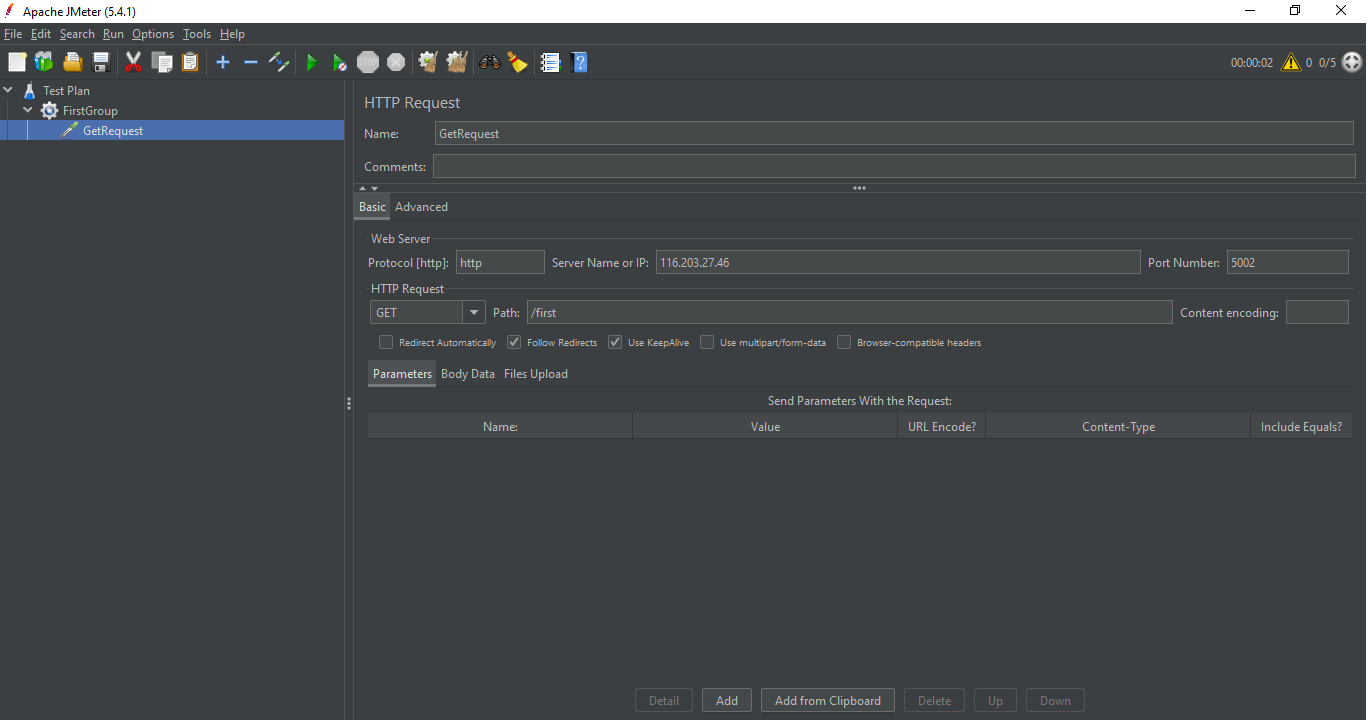
Protocol: http

Server Name or IP: 116.203.27.46

Port Number: 5002

Метод: GET

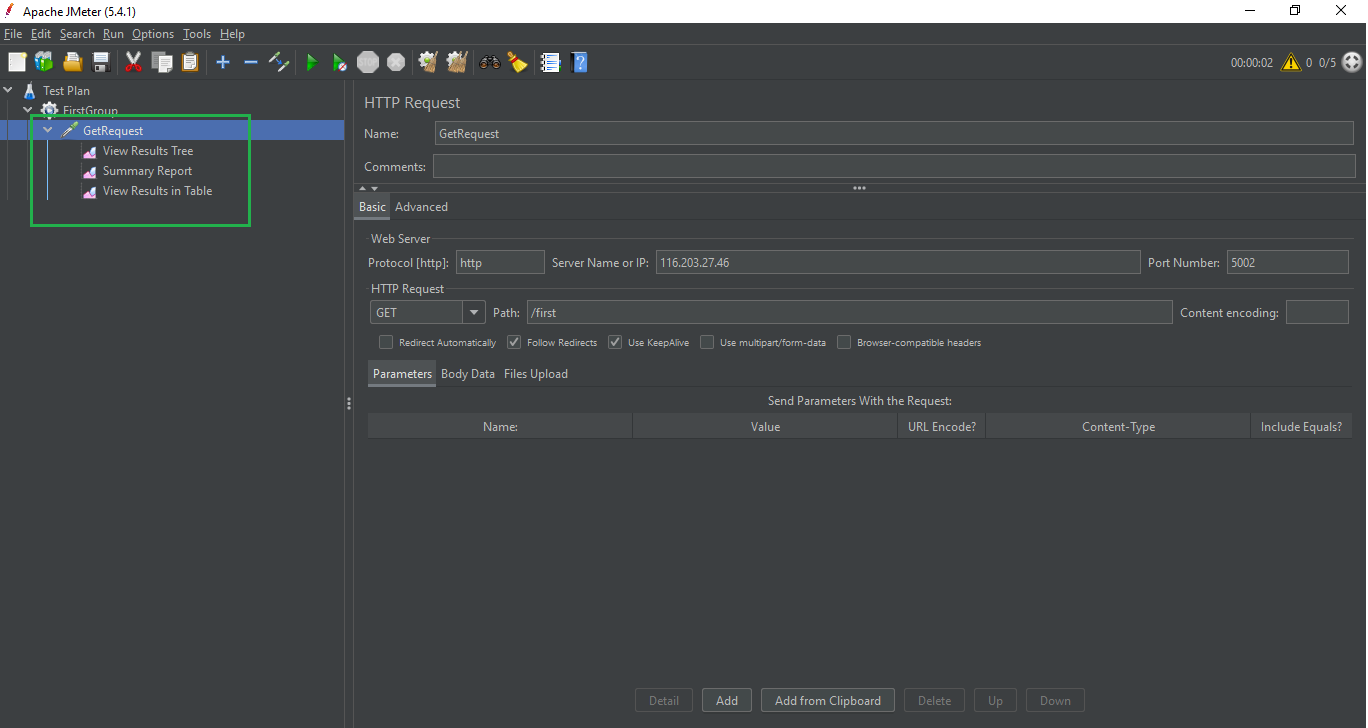
Путь (endpoint): /first



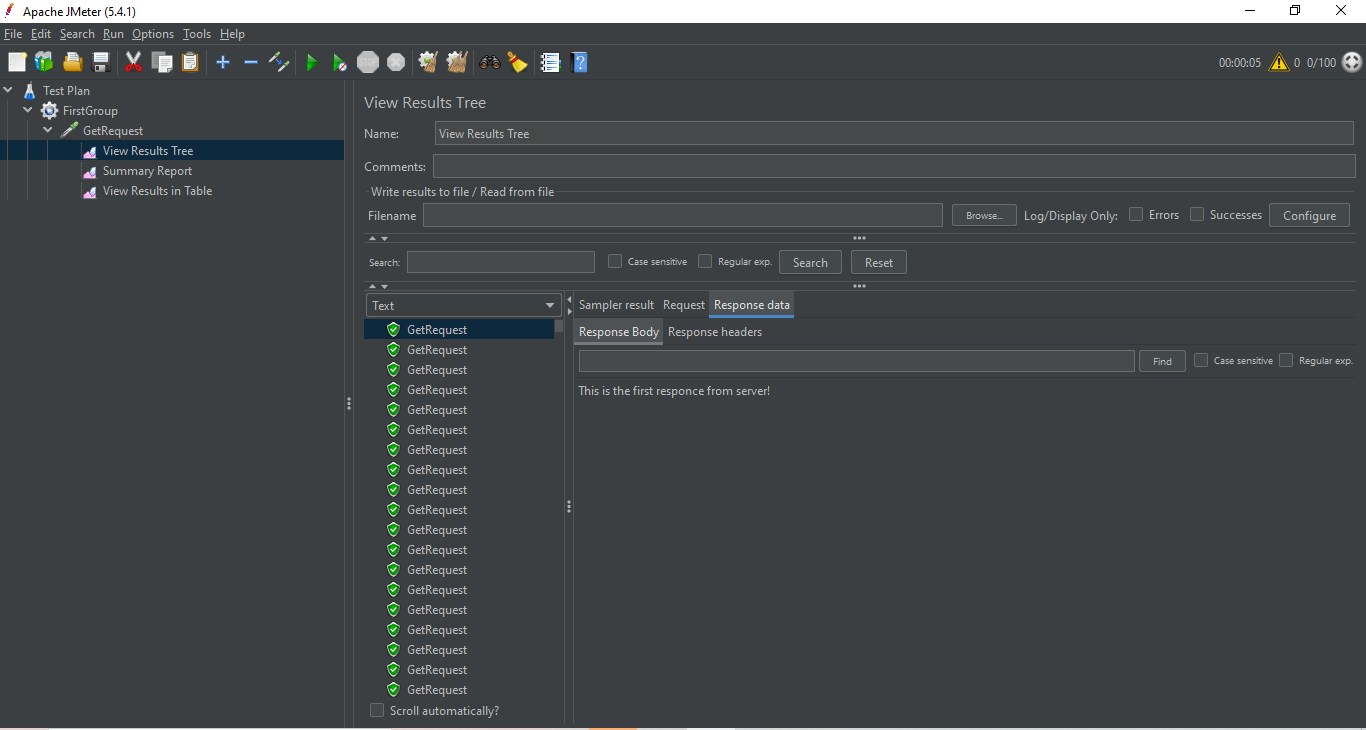
Добавим “слушателей”

Жмем ПКМ “FirstGroup” – Add – Listener

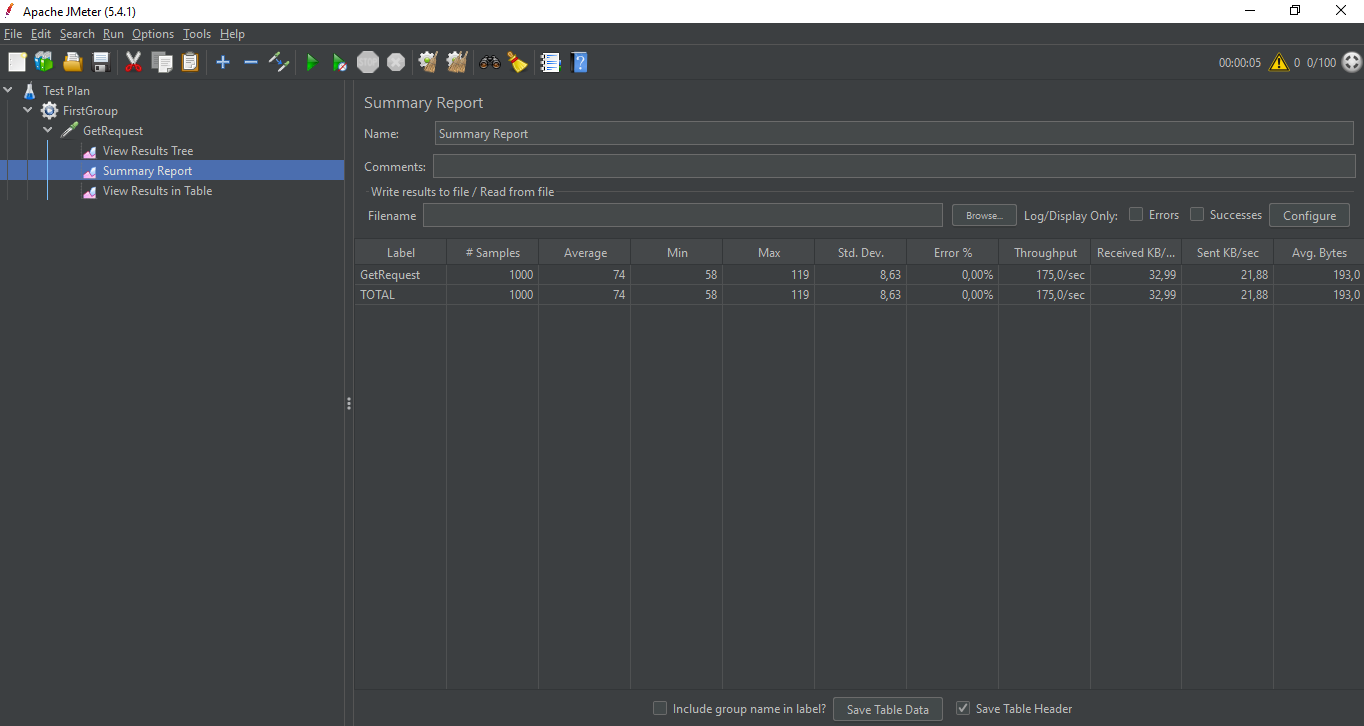
И выбираем: View Result Tree, Summary Report, View Result in Table



Нажимаем кнопку Start и смотрим результат.



*View Result Tree*



*Summary Report*

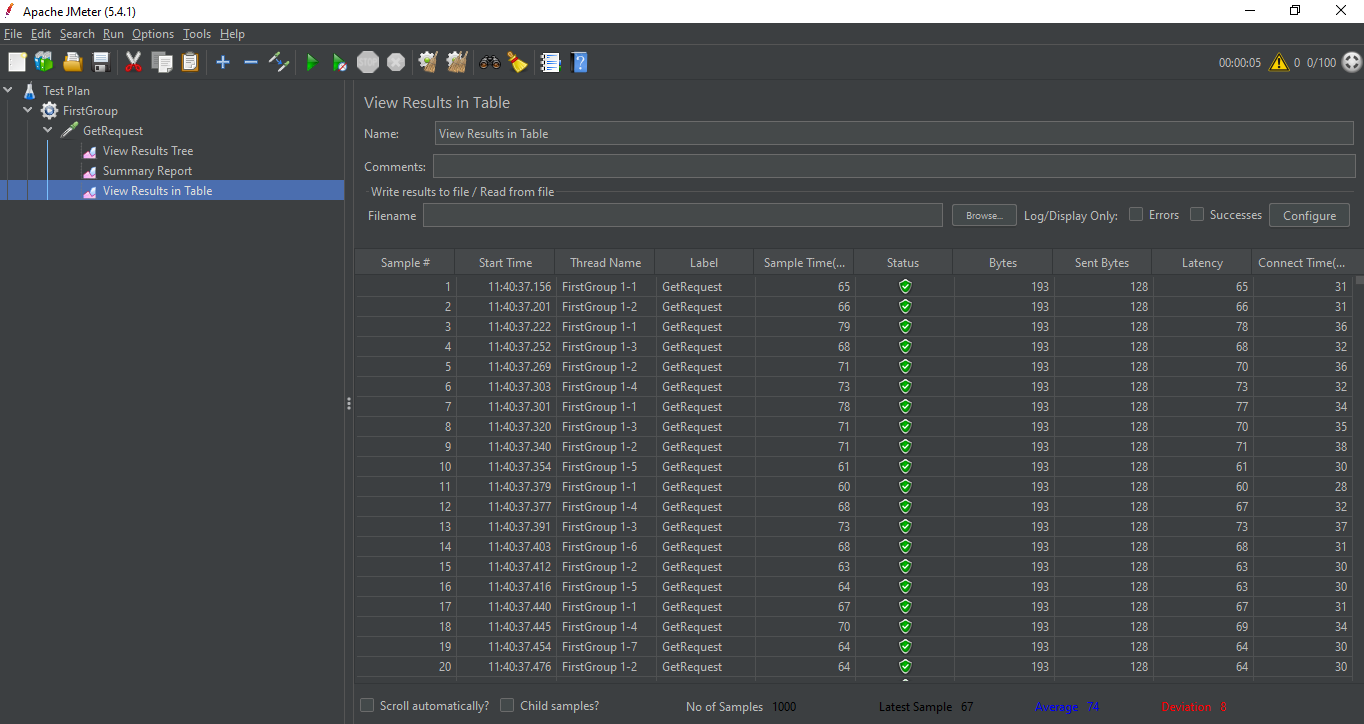
Samples: сколько всего было отправлено запросов

Average: среднее время ожидание ответа от сервера каждого запроса (в миллисекундах).

Std. Dev: стандартное отклонение

Error %: процент ошибок

Throughput: пропускная способность – количество запросов в секунду



*View Result in Table*

Временно отключим FirstGroup (ПКМ - Disable).

Создадим новый набор потоков, аналогичный первому - SecondGroup.

Добавим дефолтное значение для запросов:

ПКМ SecondGroup – Add – Config Element – HTTP Request Defaults.

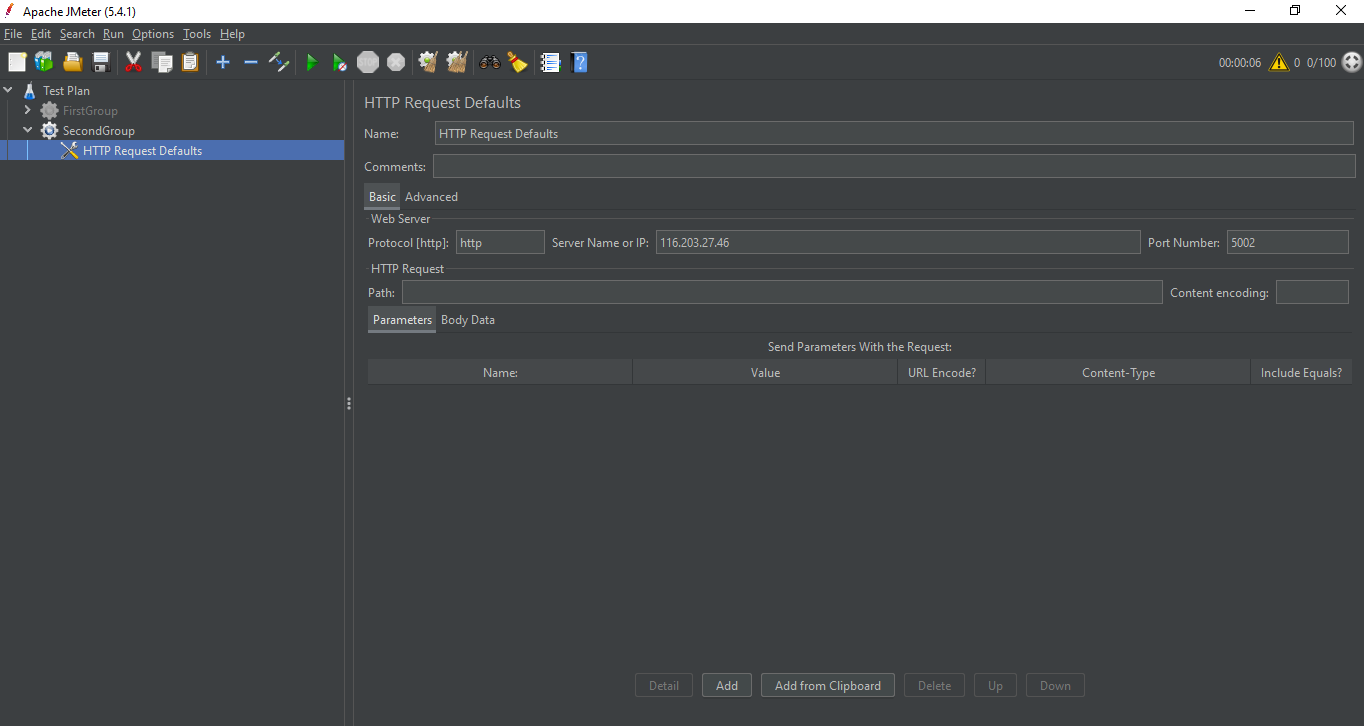
И задаем параметры

Protocol: http

Server Name or IP: 116.203.27.46

Port Number: 5002

Теперь все запросы в этой группе будут идти с этими параметрами



Добавим новый запрос:

Name: Login

Метод: POST

Path: /login

Также добавим параметры (Кнопка Add внизу окна):

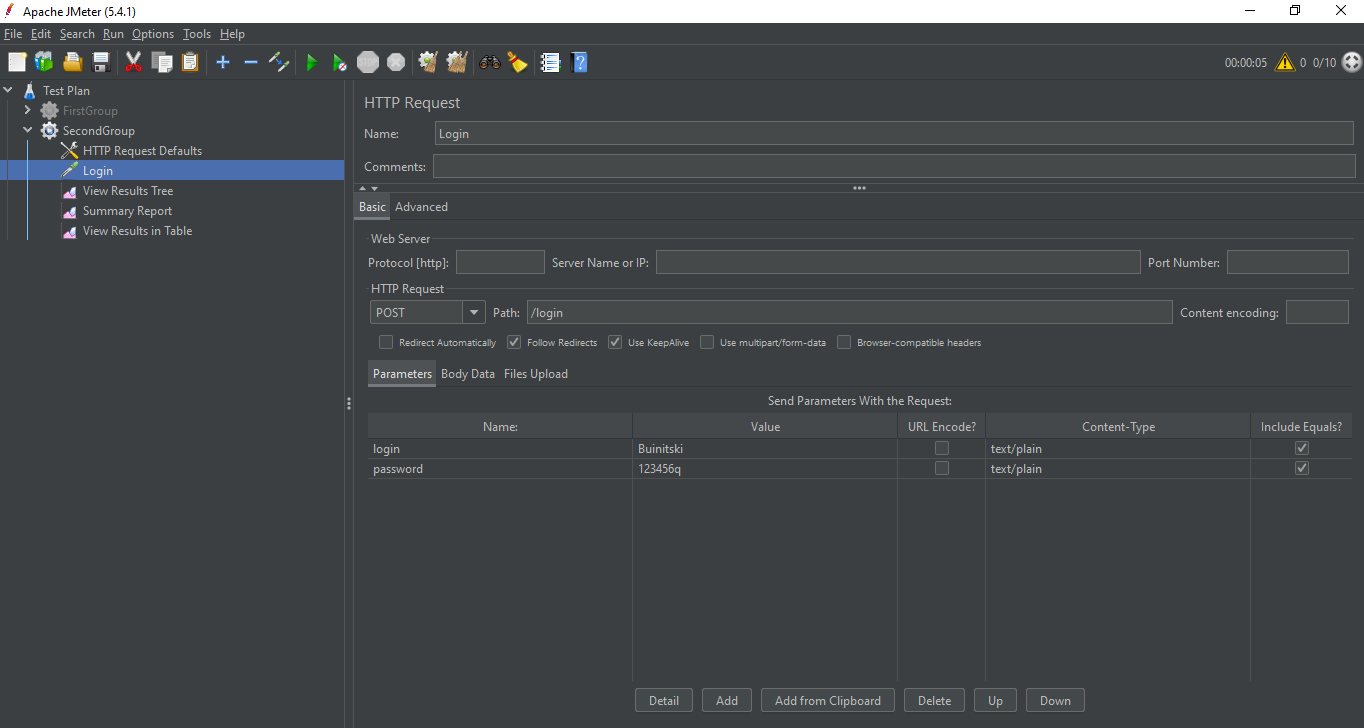
login: Buinitski

password: 123456q

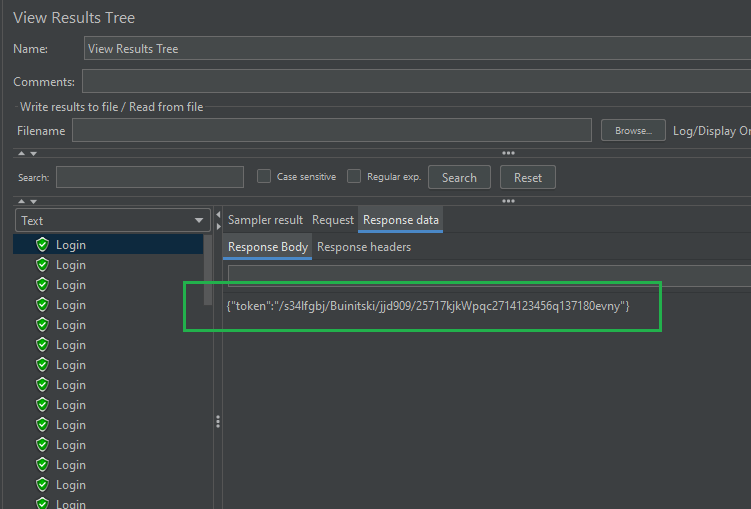
Добавим “слушателей”

Жмем ПКМ “SecondGroup” – Add – Listener

И выбираем: View Result Tree, Summary Report, View Result in Table



Нажимаем Start и в View Result Tree получаем список, в котором в Response data можно увидеть полученные в ответ токены.



*(!) Тестовый сервер настроен таким образом, что логинит с одним и тем же логином и паролем :)*

**Работа с окружением**

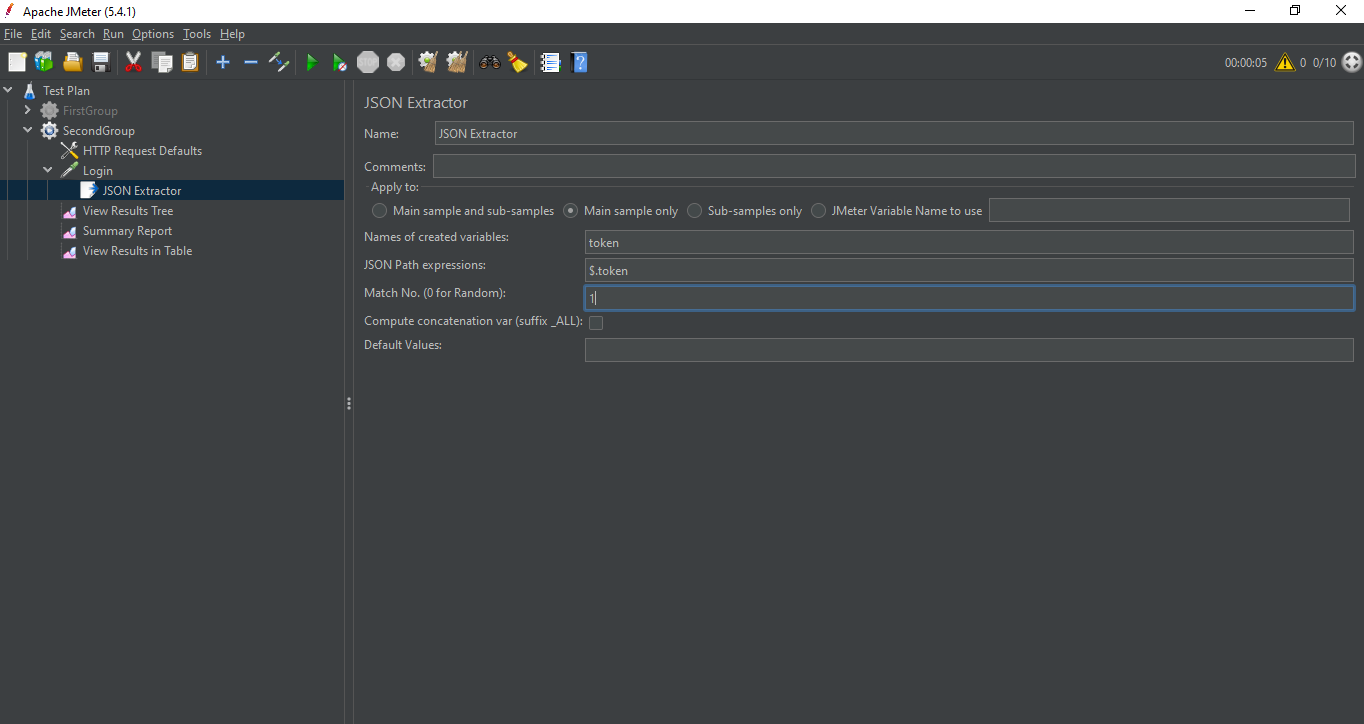
Чтобы извлечь данные для переменной:

ПКМ Login – Add – Post Processors – JSON Extractor

Names of created variable (имя переменной): token

JSON Path expressions (куда зайти за токеном): $.token

Match No (должно соответствовать 1 к 1): 1



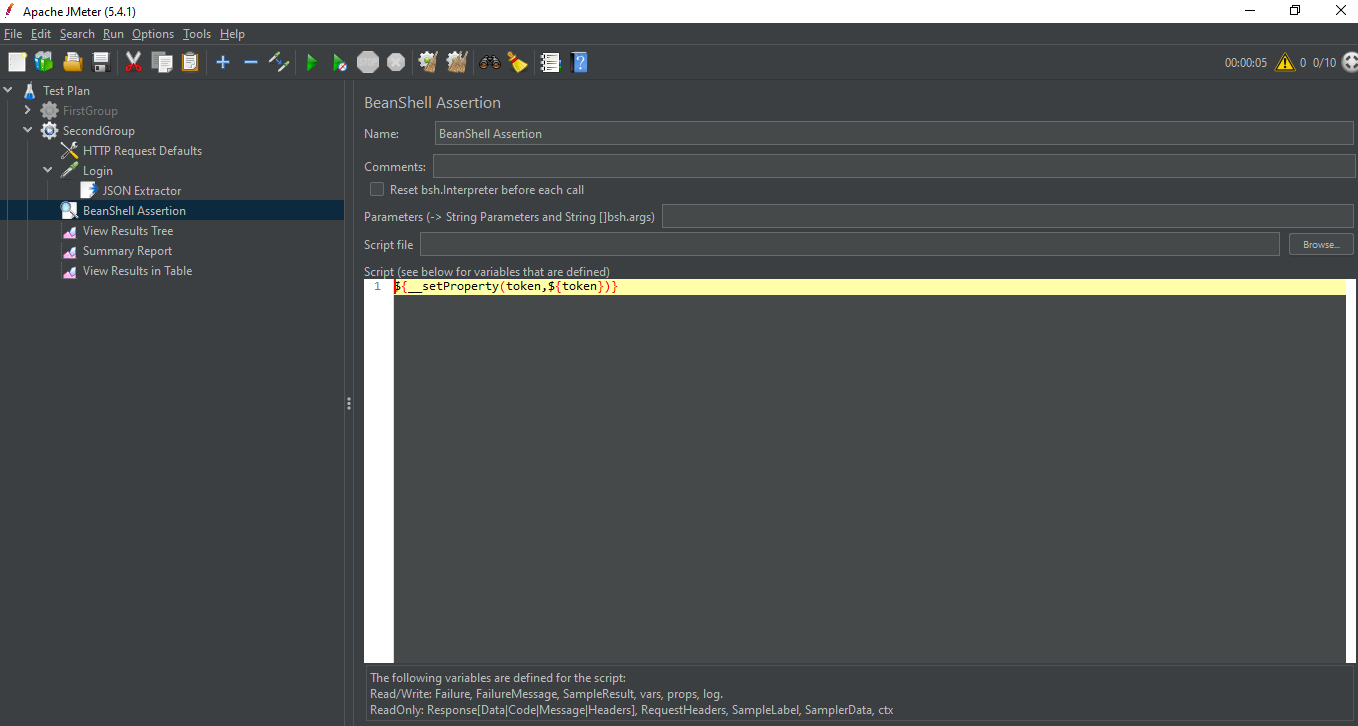
Чтобы положить в переменную окружения используем BeanShell Assertion.

ПКМ SecondGroup – Add – Assertions – BeanShell Assertion

Передвинуть из конца списка под Login, так как это влияет на порядок выполнения.

И вводим:

${\_\_setProperty(token,${token})}



Создаем новую группу потоков, только не как обычно “Thread Group”, а используем предварительно установленный плагин Stepping Thread Group.

ПКМ Test Plan – Add – Thread (Users) – jp@gc Stepping Thread Group

This group will start: 100

First, wait for (сначала подождать N сек): 5

Then start (количество потоков): 10

Next, add (добавлять N потоков): 10

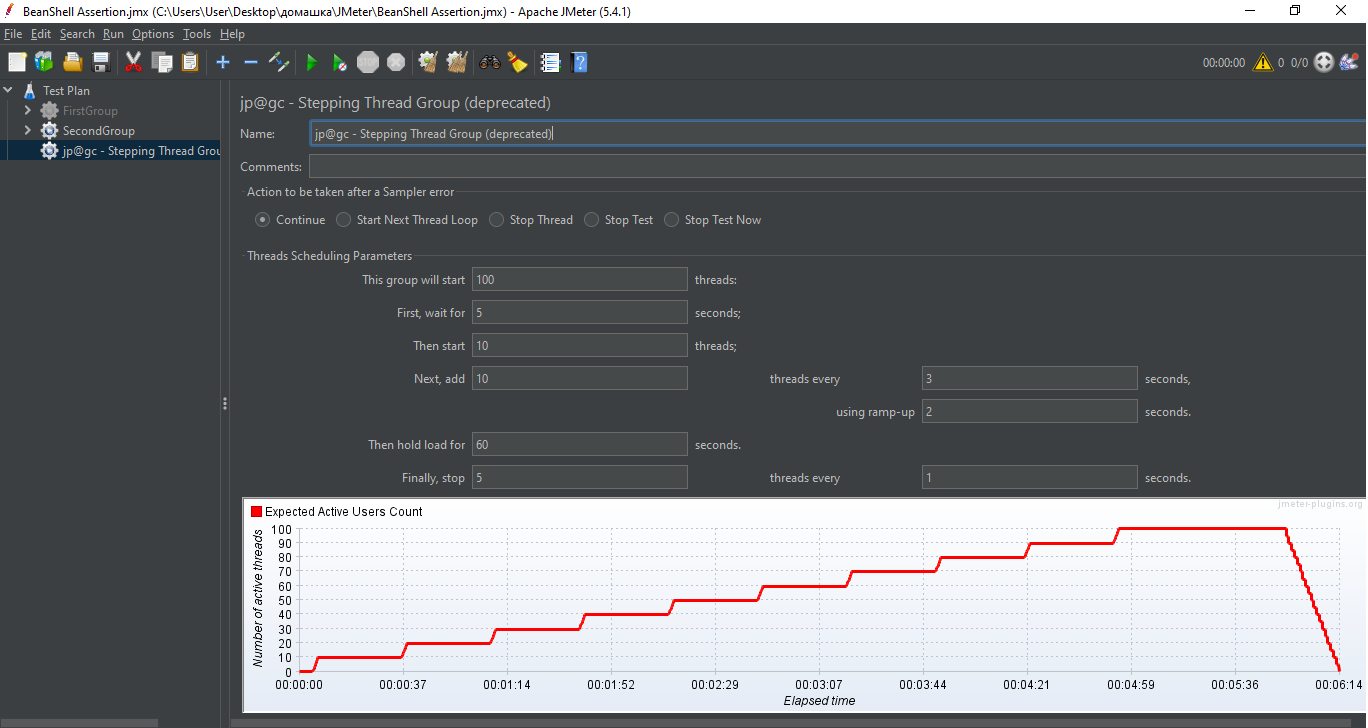
Threads every (каждые N секунд): 3

Using ramp-up (секунды, в течение которых будут добавлены новые потоки): 2

Then hold load for (держать нагрузку N секунд): 60

Finaly stop (останавливать по N потоков): 5

Threads every (каждые N секунд): 1



Также добавим дефолтное значение для запросов.

ПКМ jp@gc Stepping Thread Group – Add – Config Element – HTTP Request Defaults.

И задаем параметры

Protocol: http

Server Name or IP: 116.203.27.46

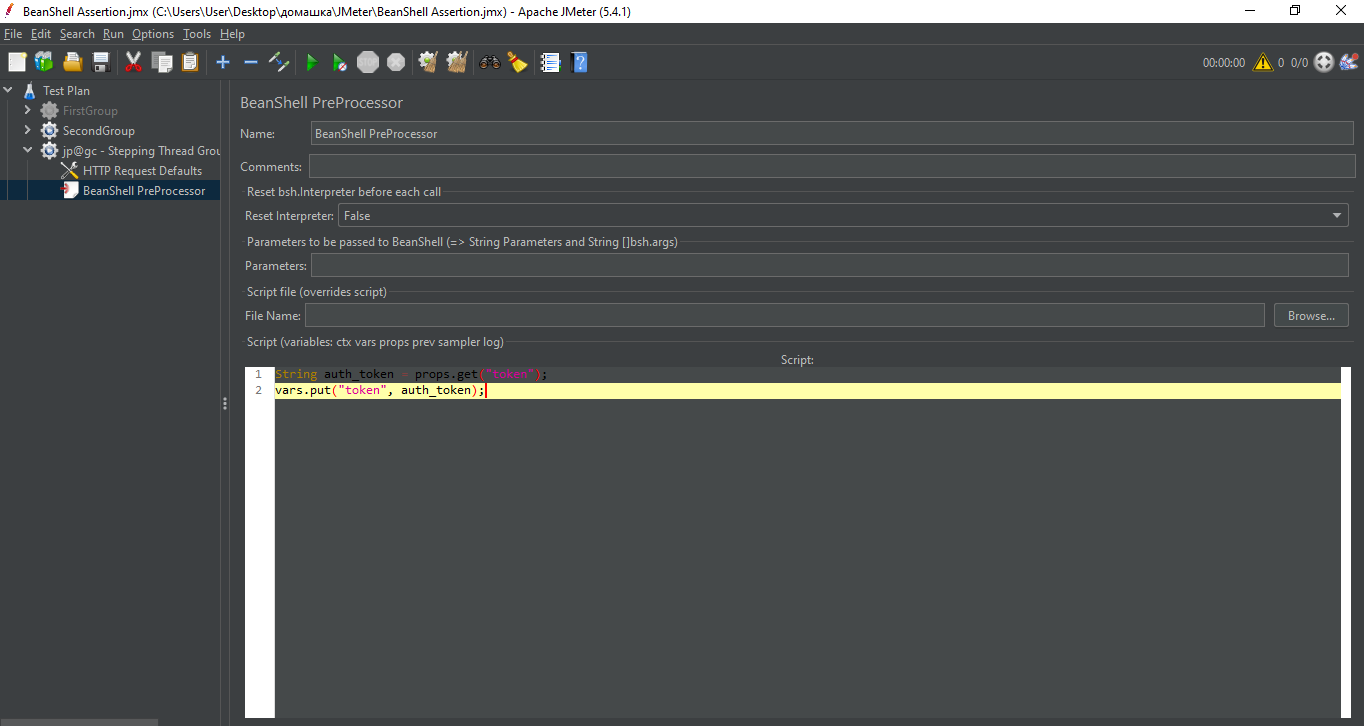
Port Number: 5002

ПКМ jp@gc Stepping Thread Group – Add – Pre Processors – BeanShell PreProcessor

И вставляем в него код:

String auth\_token = props.get("token");

vars.put("token", auth\_token);



Cоздадим запрос:

Name: new\_data2

Метод: POST

Path: /new\_data

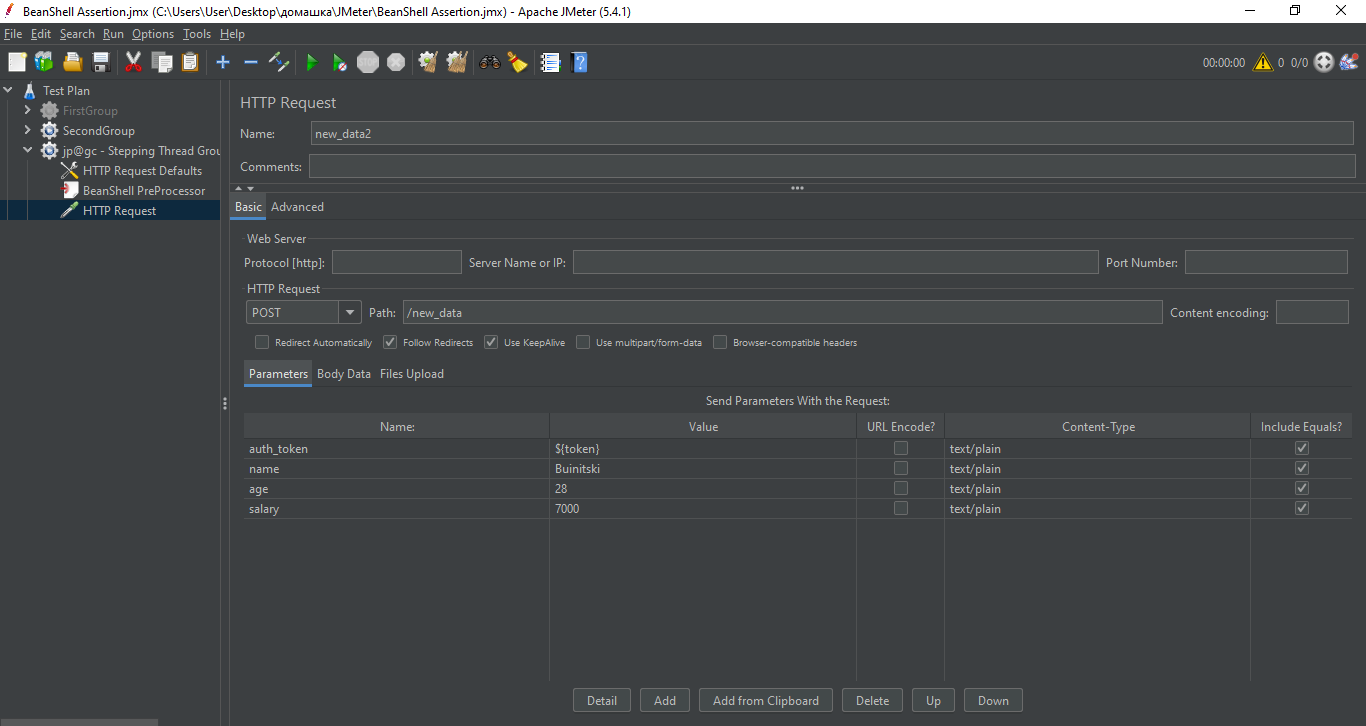
Также добавим параметры (Кнопка Add внизу окна):

auth\_token: ${token}

name: Buinitski

age: 28

salary: 7000



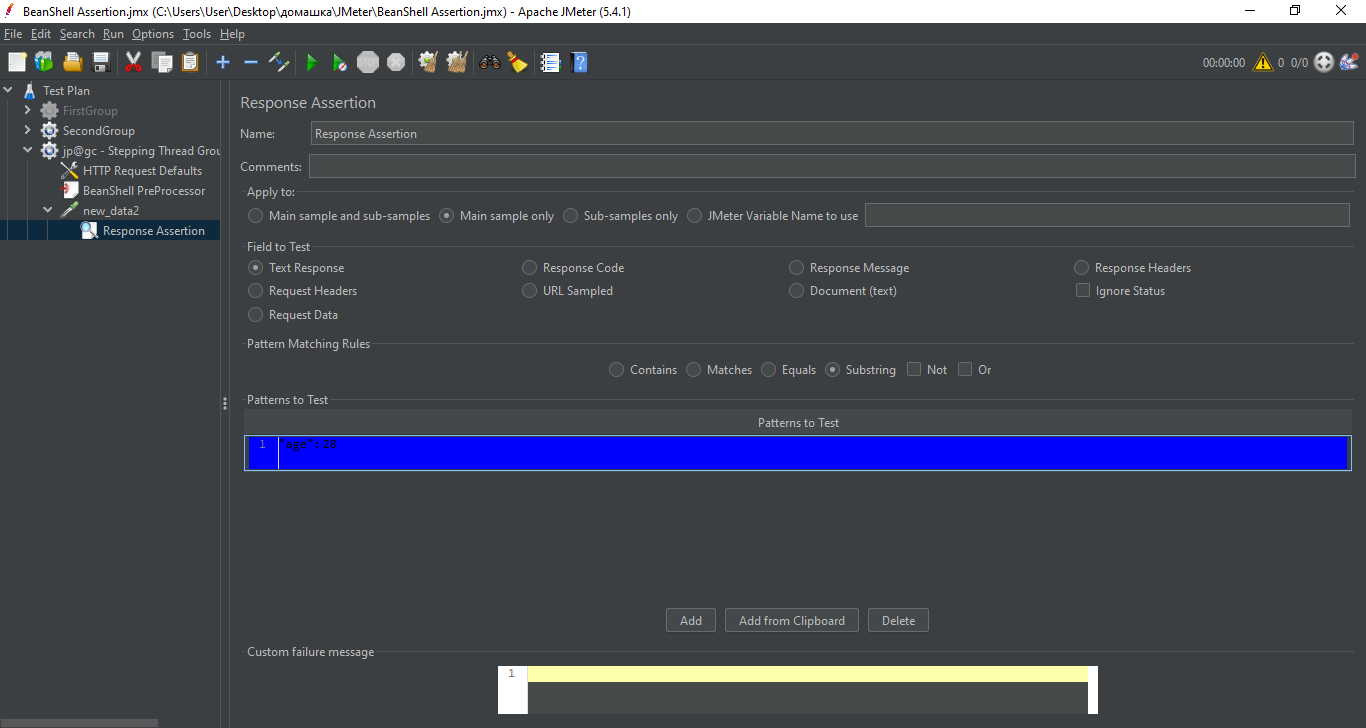
Добавим Response Assertion (Правила по которым JMeter будет понимать, правильно ли сервер нам ответил).

ПКМ new\_data2 – Add – Assertions - Response Assertion

В разделе Patterns to Test жмем Add и вводим:

“age”:28

(так как вводимый мной ранее возраст 28).

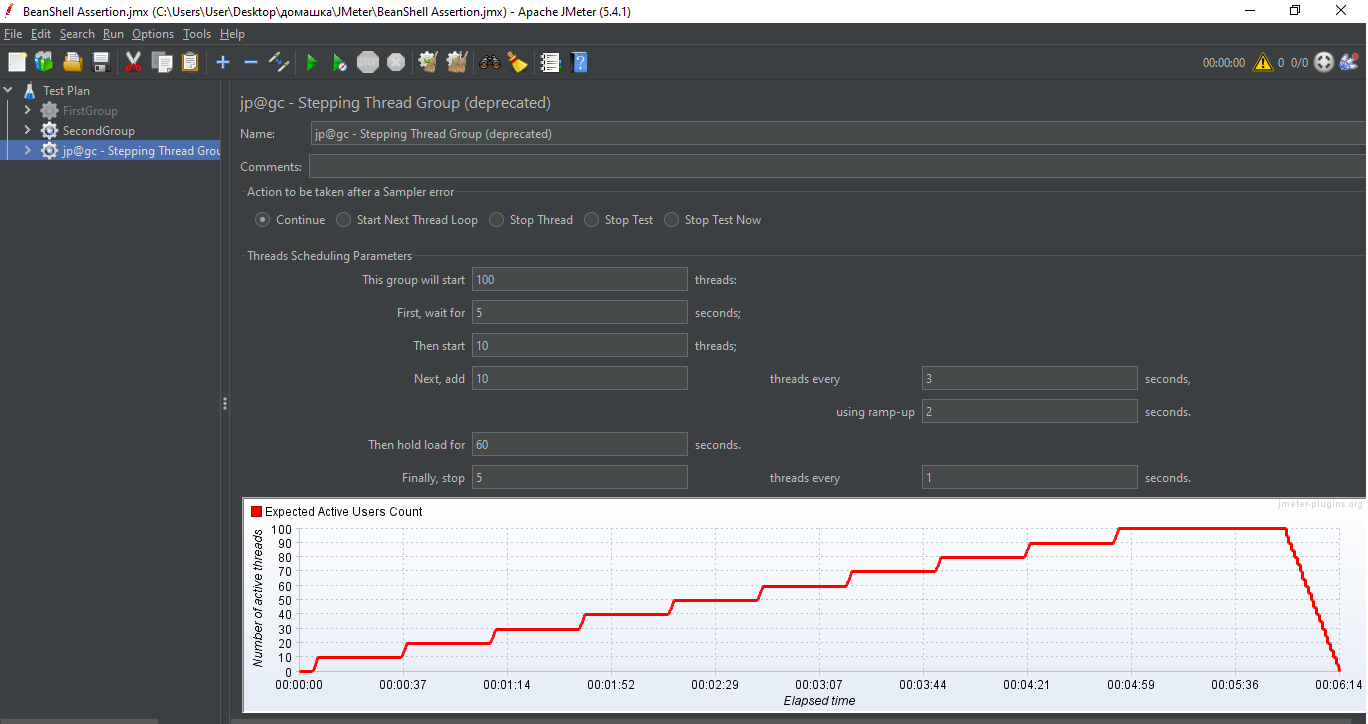


Добавим “слушателей”

Жмем ПКМ “ jp@gc Stepping Thread Group ” – Add – Listener

И выбираем: View Result Tree, Summary Report, View Result in Table, Transaction Throughput vs Threads (пропускная работа сервера при определенном количестве активных пользователей).

По итогу получаем два активных потока:

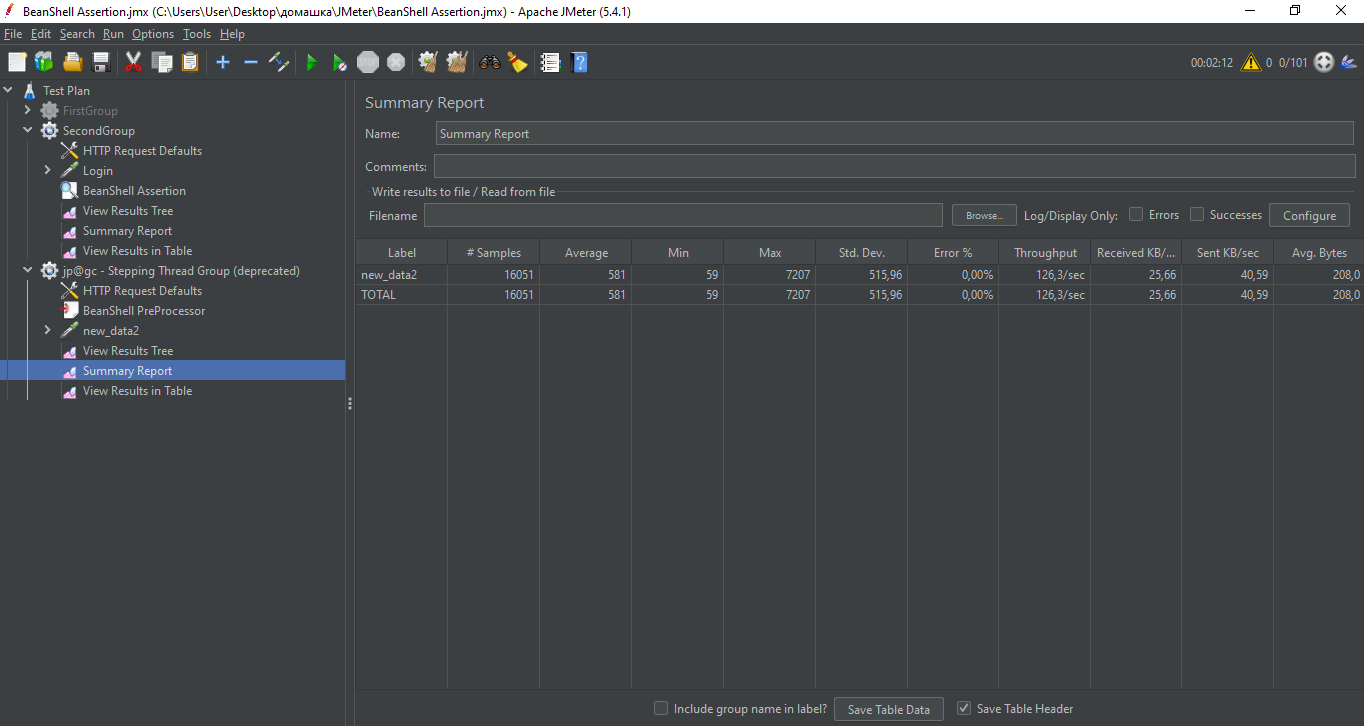


В SecondGroup меняем значения на 1 (Количество юзеров в прогоне: 1, Количество секунд Ramp-up: 1, Loop Count – сколько раз пользователи выполнят свои сценарии: 1). Количество юзеров в прогоне: 100,

Количество секунд Ramp-up: 5,

Loop Count – сколько раз пользователи выполнят свои сценарии: 10

Идем в jp@gc Stepping Thread Group - Summary Report и жмем Start



*Summary Report. Итоговый результат*

**Нагрузочное тестирование**

Проведем нагрузочный тест.

Сделаем дубликат jp@gc Stepping Thread Group и назовем его LoadTest

Отключим все активные группы, кроме LoadTest и SecondGroup (там мы логинимся)

Добавим еще два дополнительных запроса с методом GET

Name: GetRequest1

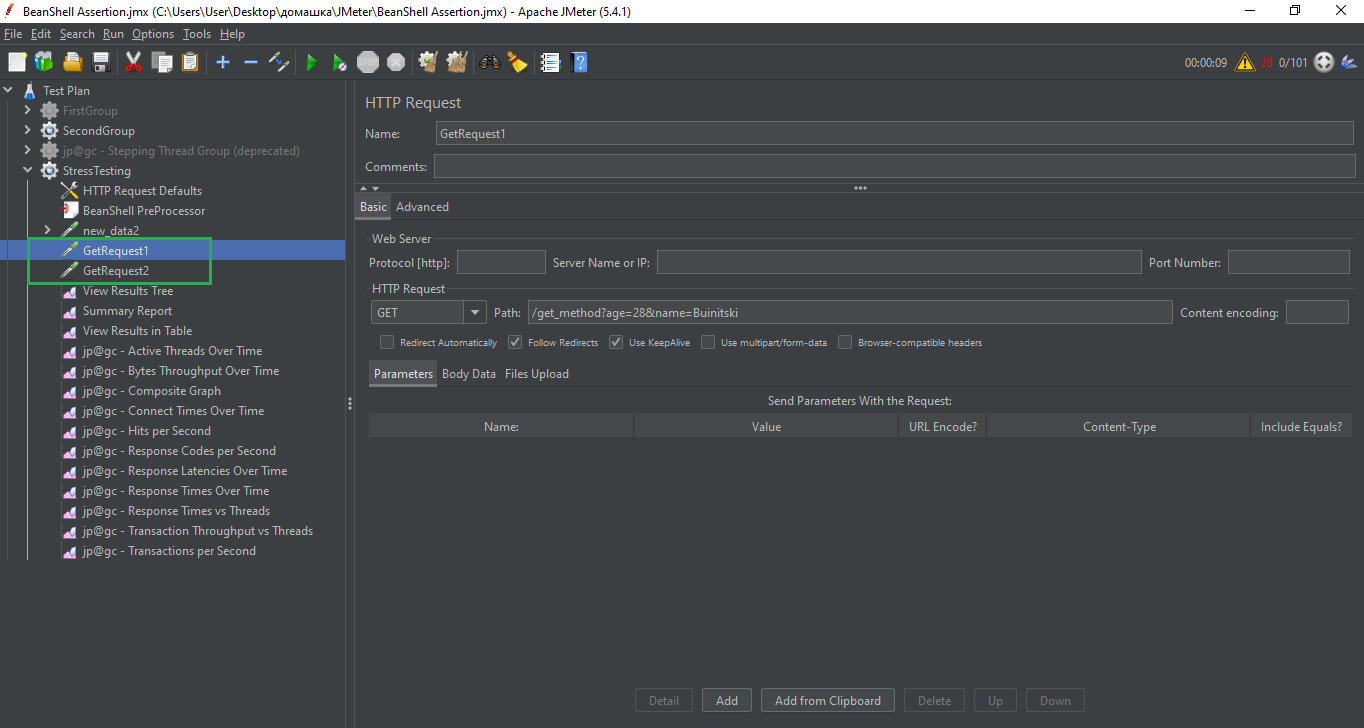
Method: GET

Path: /get\_method?age=28&name=Buinitski

Name: GetRequest2

Method: GET

Path: /get\_method?age=27&name=Jane



**Задача:** необходимо определить три класса в заданном диапазоне нагрузки:

1. Сервер работает без ошибок
2. Ошибок не более 5 %
3. Ошибок больше 5 %

Выставляем параметры:

This group will start: 1000

First, wait for (сначала подождать N сек): 5

Then start (количество потоков): 10

Next, add (добавлять N потоков): 50

Threads every (каждые N секунд): 5

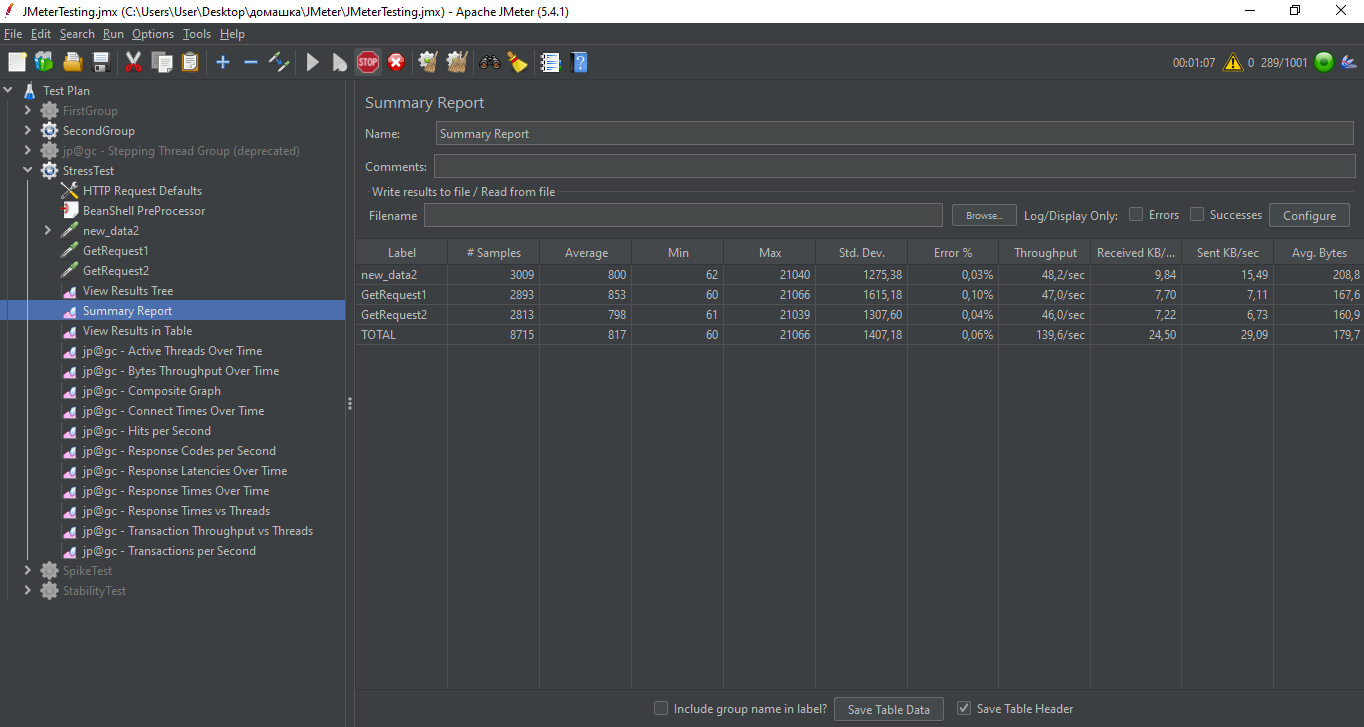
Using ramp-up (секунды, в течение которых будут добавлены новые потоки): 5

Then hold load for (держать нагрузку N секунд): 600

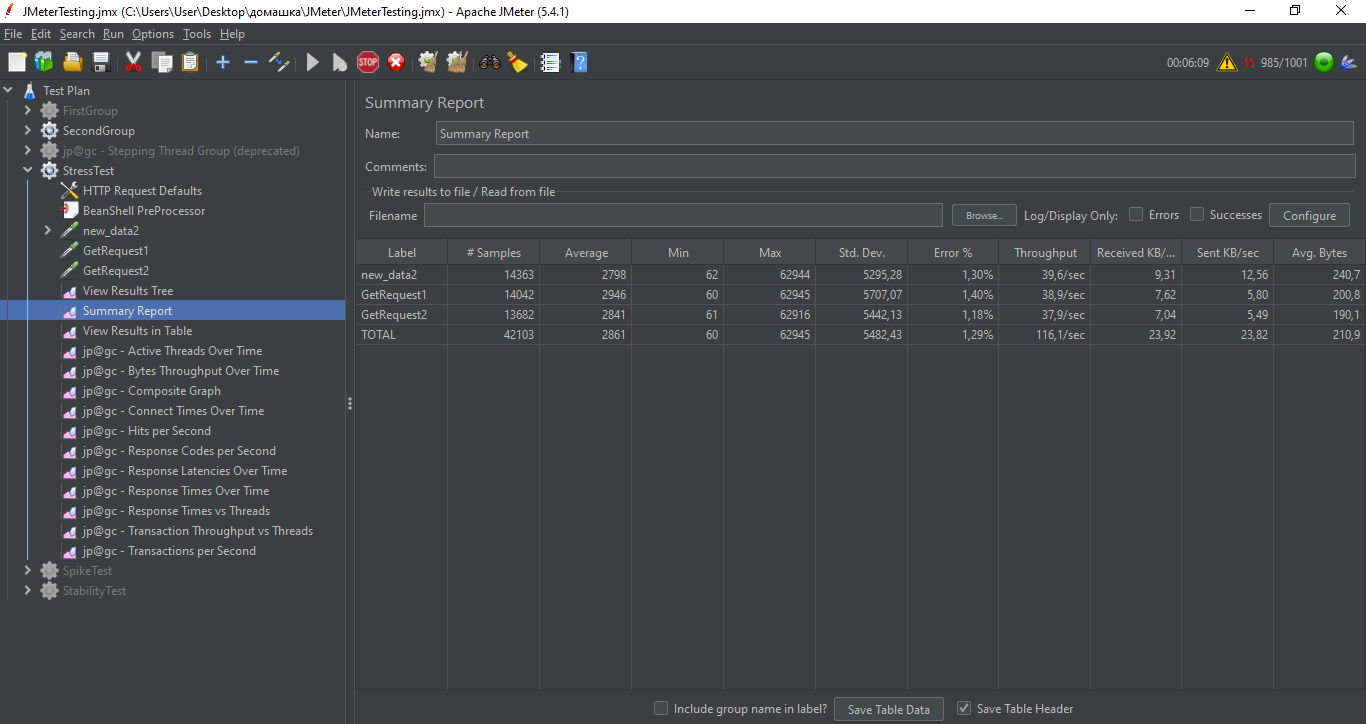
Finaly stop (останавливать по N потоков): 5

Threads every (каждые N секунд): 1

Результат: Сервер работает без ошибок до 280 пользователей, после 280 до 1000 количество ошибок не превышает 5%. При дополнительном тесте больше 1000 ошибки превысили 5%.



*Сервер работает без ошибок до 280*



*После 280 количество ошибок не превышает 5%*

**Спайк тест**

Проверим результат предыдущего теста (280 пользователей) с помощью спайк теста и убедимся, что сервер работает без ошибок с количеством 280 при резко увеличенной нагрузке за заданный период.

Сделаем дубликат StressTest и назовем его SpikeTest

Отключим все активные группы, кроме SpikeTest и SecondGroup (там мы логинимся)

Выставляем параметры:

This group will start: 280

First, wait for (сначала подождать N сек): 5

Then start (количество потоков):

Next, add (добавлять N потоков):

Threads every (каждые N секунд):

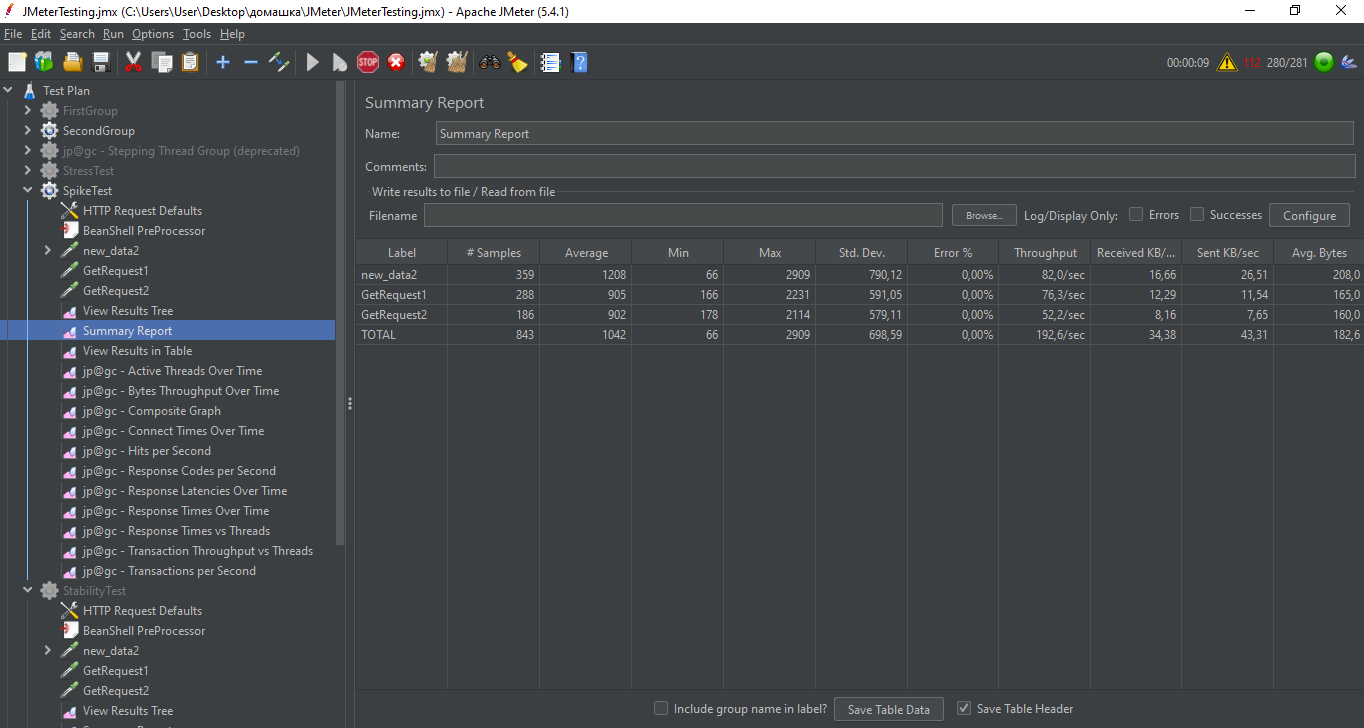
Using ramp-up (секунды, в течение которых будут добавлены новые потоки):

Then hold load for (держать нагрузку N секунд): 20

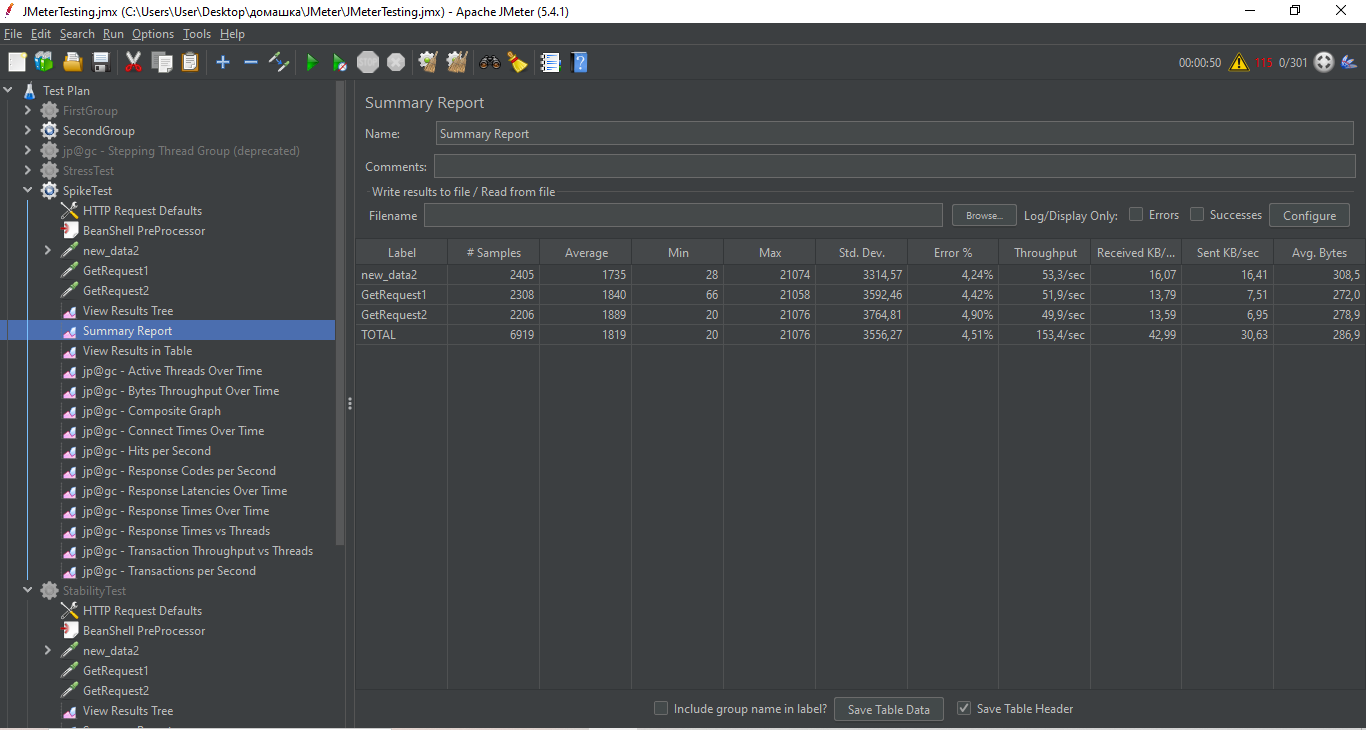
Finaly stop (останавливать по N потоков): 10

Threads every (каждые N секунд): 2

Результат: Сервер работает без ошибок при 280, последующие тесты с увеличением нагрузки до 290 и 300 приводят к ошибкам.



*При нагрузке 280 ошибки отсутствуют*



*При нагрузке 300 ошибки есть*

**Тестирование стабильности**

Сделаем дубликат StressTest и назовем его StabilityTest

Отключим все активные группы, кроме StabilityTest и SecondGroup (там мы логинимся)

Выставляем параметры:

This group will start: 140

First, wait for (сначала подождать N сек): 5

Then start (количество потоков): 10

Next, add (добавлять N потоков): 10

Threads every (каждые N секунд): 3

Using ramp-up (секунды, в течение которых будут добавлены новые потоки): 3

Then hold load for (держать нагрузку N секунд): 6000

Finaly stop (останавливать по N потоков): 10

Threads every (каждые N секунд): 2

Результат: Стабильная работа сервера.

