

# Day 3 - Create Performance Test in Jmeter

Setelah pada materi sebelumnya teman-teman berhasil menjalankan tes JMeter pertama, berikut ini adalah beberapa tahapan yang perlu diimplementasikan pada Jmeter untuk mengembangkan skenario pengujian yang lebih kompleks dan mendetail.

Dalam materi ini, kita akan melakukan analisis kinerja pada public API <https://fakerestapi.azurewebsites.net/> untuk memastikan public API dapat menangani beban dan memberikan pengalaman user yang lancar.

## Create Test Plan and Scenario Test in Jmeter

Sebelum menguji kinerja aplikasi target, kita harus menentukan :

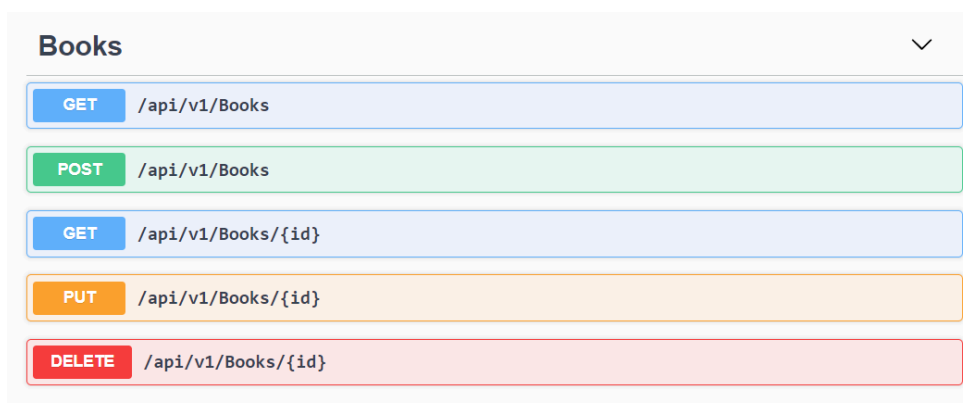
- **Normal Load:** Jumlah rata-rata users yang mengunjungi situs web ini (*misalnya 5000 users*)
- **Heavy Load:** Jumlah maksimum users yang mengunjungi situs web ini (*misalnya 10000 users*)
- Apa target kita dalam tes ini?

Untuk normal load dan heavy load, kita bisa mendapatkan datanya dari load testing awal. Jika belum pernah dilakukan pengujian kinerja aplikasi, untuk normal load dan heavy loadnya bisa kita tentukan nanti setelah kita mendapatkan data awal dari performance testing.

### 1. Menentukan Endpoint Yang Akan di Test

Sebelum memulai performance testing, kenali yang dibutuhkan dari endpoint yang akan kita test berikut ini :

<https://fakerestapi.azurewebsites.net/>



Endpoint yang akan dites :

- GET /api/v1/Books
- POST /api/v1/Books
- GET /api/v1/Books/{id}
- PUT /api/v1/Books/{id}
- DELETE /api/v1/Books/{id}

Kebutuhan masing - masing endpoint :

- Books ID

## 2. Membuat Test Plan Skenario

Skenario yang kemungkinan digunakan pada basic Jmeter ini, adalah :

- **Load testing** — di mana kita akan menetapkan jumlah user yang telah ditentukan dan periode ramp up, dan looping sebanyak  $n$  kali.
- **Stress testing** — dimana kita akan menentukan batas atas jumlah user secara bersamaan sebelum aplikasi mengalami down time.
- **Soak testing/Endurance testing** — di mana kita dapat menetapkan beban yang diharapkan dan mempertahankan beban tersebut selama periode waktu yang telah ditentukan.

Sebelum memulai performance testing, kita perlu mendefinisikan test plan kita lebih detail dengan contoh skenario seperti di bawah ini:

### 1. Test Plan Scenario - Load Testing

#### Tujuan:

Mengukur sejauh mana aplikasi dapat menangani beban user yang tinggi selama periode waktu tertentu, seperti pada periode penjualan besar atau acara promosi.

#### Langkah-langkah:

Simulasikan request dimulai dari 5 user, dengan anggapan 5 user ini adalah *normal load* saat ini. Dan setting ramp-up period diawali dengan 10 detik.

#### Hasil :

Amati waktu respons dan kapasitas sistem saat beban user meningkat.

**Specification :**

BASE\_URL : <https://fakerestartapi.azurewebsites.net/>

PATH :

- Path 1 : [GET] /api/v1/Books
- Path 2 : [POST] /api/v1/Books
- Path 3 : [GET] /api/v1/Books/\${id}
- Path 4 : [PUT] /api/v1/Books/\${id}
- Path 5 : [GET] /api/v1/Books/\${id}

THREAD GROUP PROPERTIES :

- Number of Thread (User) : 5
- Ramp-Up Period (in seconds) : 10
- Loop Count : 1

Dengan skenario diatas, maka thread akan dilakukan sebanyak 5 kali, dimana tiap thread dilakukan selama 2 (10/5) detik sebelum berganti ke thread berikutnya. Jika Loop Count diisi 2, maka testing akan diulangi sebanyak 2 kali.

## 2. Test Plan Scenario - Stress Testing

**Tujuan:**

Mengidentifikasi batasan kapasitas dan mengevaluasi kinerja aplikasi di bawah tekanan ekstrem.

**Langkah-langkah:**

Untuk mensimulasikan aplikasi dibawah tekanan, dari data load testing sebelumnya, kita bisa membuat 2 skenario. Pertama, jumlah user sama (5 users), dengan period yang sangat singkat yaitu 1 detik. Kedua, jumlah period sama (10 detik) dengan user yang bertambah beberapa kali lipat dari sebelumnya (10 sampai  $n$  users).

**Hasil :**

Perhatikan bagaimana aplikasi menanggapi, apakah ada penurunan kinerja, atau bahkan kegagalan sistem.

**Specification :**

BASE\_URL : <https://fakerestapi.azurewebsites.net/>

PATH :

- Path 1 : [GET] /api/v1/Books
- Path 2 : [POST] /api/v1/Books
- Path 3 : [GET] /api/v1/Books/\${id}
- Path 4 : [PUT] /api/v1/Books/\${id}
- Path 5 : [GET] /api/v1/Books/\${id}

THREAD GROUP PROPERTIES:

- Number of Thread (User) : 5
- Ramp-Up Period (in seconds) : 1
- Loop Count : 1

Dengan skenario diatas, maka thread akan dilakukan sebanyak 5 kali, dimana tiap thread dilakukan selama 0.2 (1/5) detik sebelum berganti ke thread berikutnya. Jika Loop Count diisi 2, maka testing akan diulangi sebanyak 2 kali.

### 3. Test Plan Scenario - Soak testing/Endurance testing

**Tujuan:**

Mengukur kinerja sistem selama periode waktu yang lama untuk menilai daya tahan dan stabilitasnya

**Langkah-langkah:**

Jalankan simulasi normal user selama periode yang panjang. Untuk menjalankan test ini, kita bisa menggunakan data load testing sebelumnya, dengan users yang sama, tapi menambahkan durasi yang lebih panjang.

**Hasil :**

Pantau apakah ada penurunan kinerja atau masalah akumulasi sumber daya seiring waktu.

**Specification :**

BASE\_URL : <https://fakerestapi.azurewebsites.net/>

PATH :

- Path 1 : [GET] /api/v1/Books
- Path 2 : [POST] /api/v1/Books
- Path 3 : [GET] /api/v1/Books/\${id}
- Path 4 : [PUT] /api/v1/Books/\${id}
- Path 5 : [GET] /api/v1/Books/\${id}

THREAD GROUP PROPERTIES :

- Number of Thread (User) : 10
- Ramp-Up Period (in seconds) : 100
- Loop Count : 1

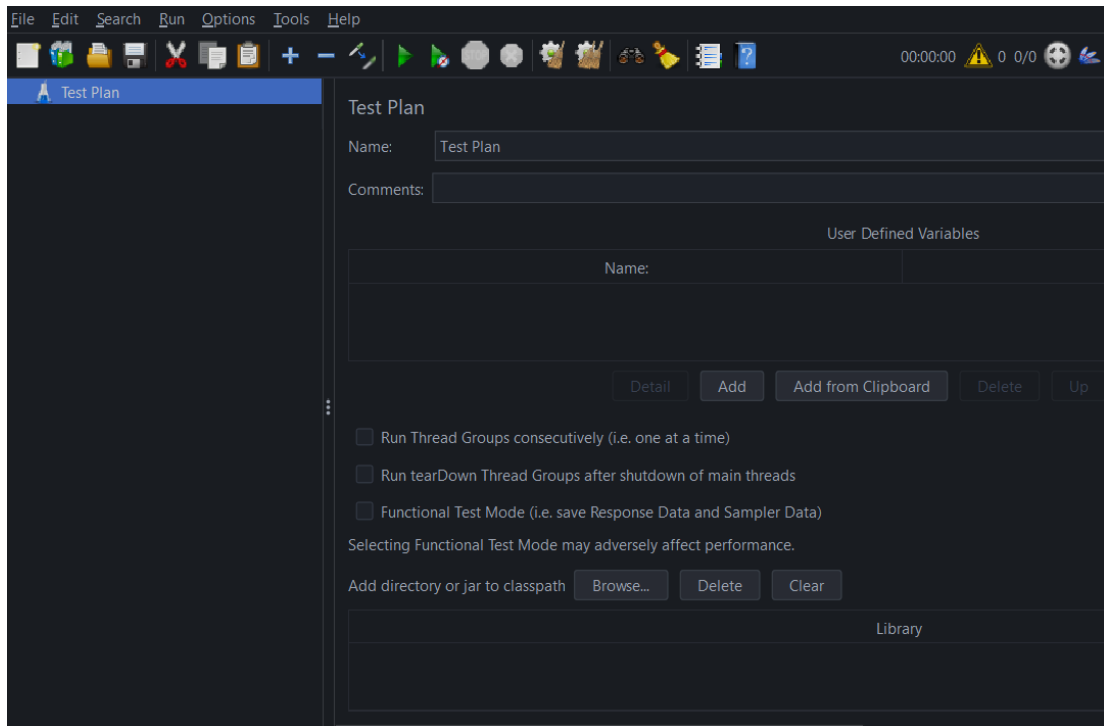
Dengan skenario diatas, maka thread akan dilakukan sebanyak 5 kali, dimana tiap thread dilakukan selama 10 (100/10) detik sebelum berganti ke thread berikutnya. Jika Loop Count diisi 2, maka testing akan diulangi sebanyak 2 kali.

### 3. Membuat Test Plan dan Skenario Test Pada Jmeter

#### Step 1 - Start Jmeter

Pertama-tama, mari kita mulai dengan launch JMeter. Untuk melakukannya, silakan buka direktori di mana JMeter diinstal dan klik dua kali pada file **jmeter.bat** yang dapat ditemukan di direktori "bin".

Bagi pengguna Linux, jalankan **\$JMETER\_FOLDER/bin/jmeter** pada terminal. Akan muncul tampilan aplikasi JMeter seperti pada gambar berikut.



Tampilan Aplikasi Desktop JMeter

## Step 2 - Create a TestPlan

Setelah berhasil membuka JMeter, kita dapat membuat *test plan* yang berisi urutan komponen-komponen uji yang berfungsi sebagai penentu bagaimana sebuah *server* akan disimulasikan.

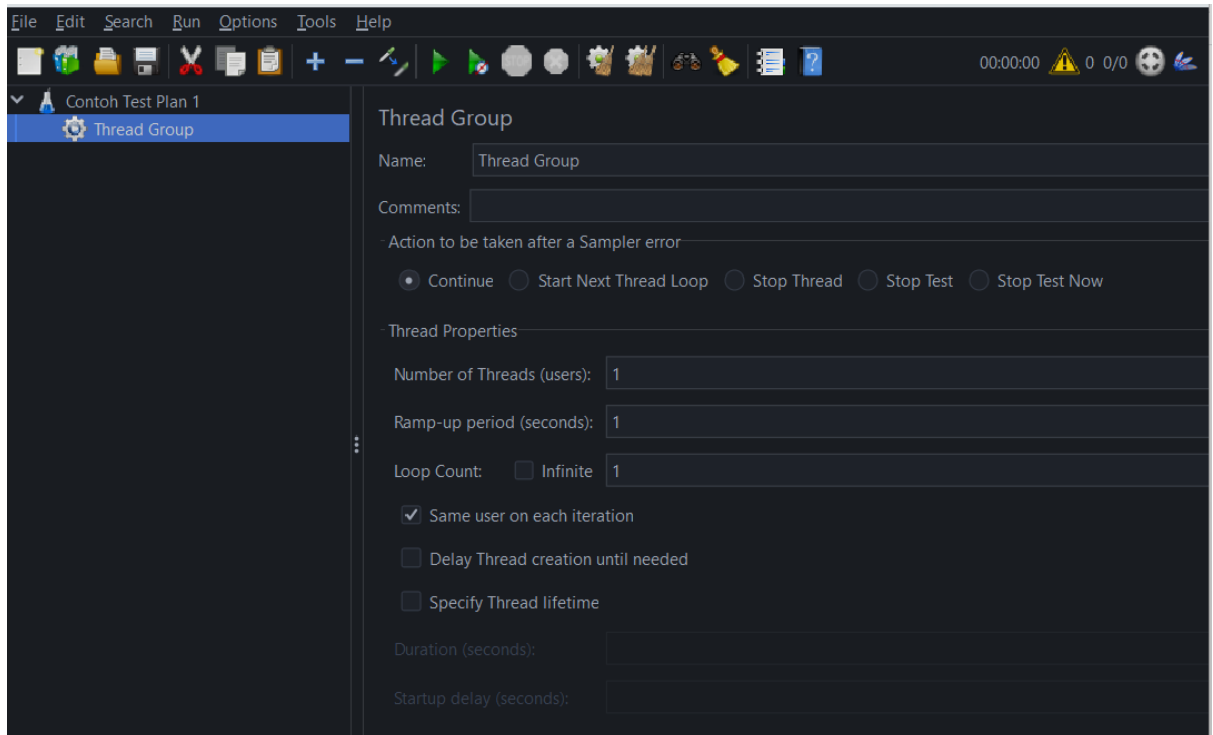
Di percobaan kita kali ini, ubah nama *test plan* menjadi “*Contoh Test Plan 1*” lalu save dengan menekan ctrl + s. Elemen Pengujian JMeter dan Test plan akan disimpan dalam format **\*.JMX**

*.JMX adalah singkatan dari Java Management Extensions.*

## Step 3 - Create a Thread Group (Users)

*Thread Group* merupakan salah satu komponen uji yang ada pada *test plan*. Cara menambah *thread group* :

1. Klik kanan pada nama *Test Plan* yang sudah kita simpan, “*Contoh Test Plan 1*”
2. Arahkan *mouse*, pilih **Add >**
3. Arahkan kembali *mouse*, pilih **Threads (Users) >**
4. Pilih **Thread Group**



*Tampilan Thread Group*

Di *thread group* terdapat beberapa *properties* yang dapat mempengaruhi skenario pengujian *performance* yang dijalankan;

- **Number of Threads (users)** : jumlah *user* virtual yang akan disimulasikan.
- **Ramp-up period (seconds)** : total durasi yang dibutuhkan seluruh skenario dijalankan dari awal sampai akhir.
- **Loop Count** : jumlah percobaan pengujian yang dijalankan. Teman-teman juga bisa membuat pengujian yang *loop*-nya tak terhingga dengan meng-klik *checkbox* “*Infinite*”.

Jadi, apabila kita memasukkan *number of threads* = 10, *ramp-up period* = 10, dan *loop count* = 1, itu artinya ada 1 *thread* yang dijalankan setiap 1 detik.

Apabila kita memasukkan *ramp-up period* = 100, itu artinya ada 1 *thread* yang dijalankan setiap 10 detik karena  $100 \text{ ramp-up period} / 10 \text{ threads} = 10$ .

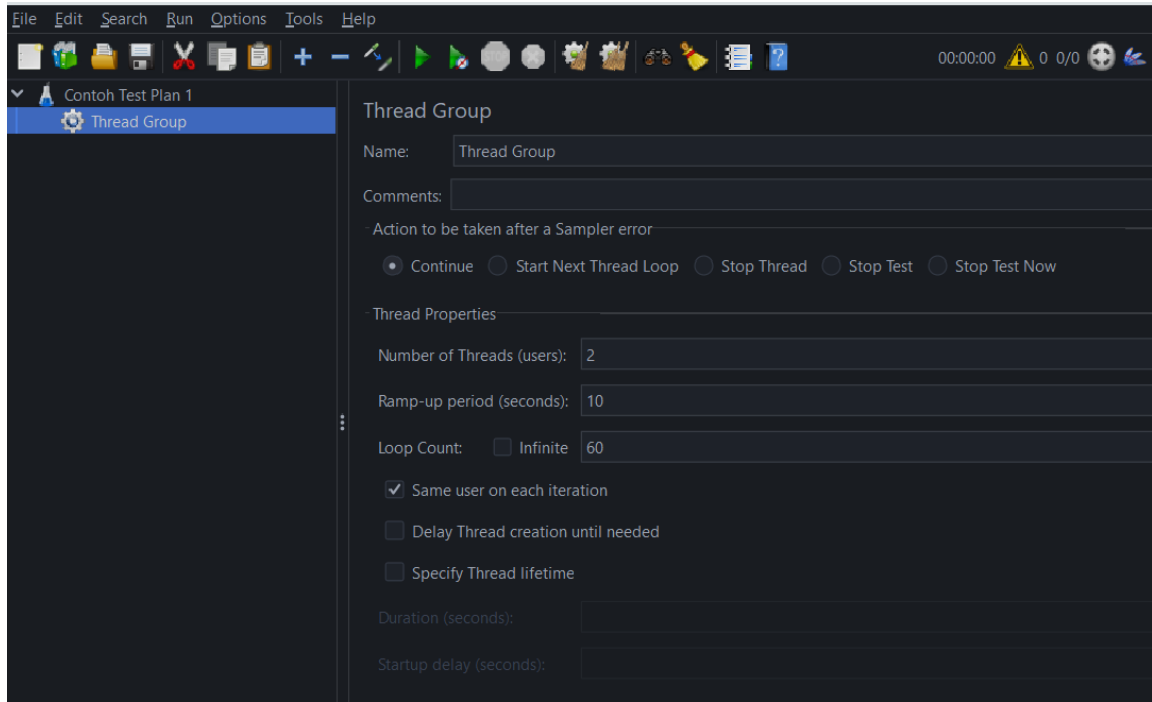
Contoh lagi, kalau *ramp-up period* = 200 dan *threads* = 10, artinya ada 1 *thread* yang dijalankan setiap 20 detik.

Catatan tambahan, kalau teman-teman ingin membuat skenario dimana di setiap *thread* ada *delay*, teman-teman bisa klik *checkbox* “*Delay Thread creation until needed*”.

Misalnya saat *ramp-up period* = 5 dan *threads* = 10, itu artinya setiap 0.5 detik 1 *thread* dijalankan.

Kalau teman-teman klik *checkbox* “*Delay Thread creation until needed*”, itu artinya JMeter akan memberikan sedikit *delay* sampai 1 *thread* selesai dijalankan, kemudian dilanjutkan dengan *thread* berikutnya.

Pilihan ini akan berguna saat kita memiliki *thread* yang sangat banyak, dan tidak mengharuskan semua *thread* aktif secara bersamaan.



Untuk contoh yang akan kita jalankan, masukkan number of threads = 2, ramp-up period = 10, dan loop = 60.

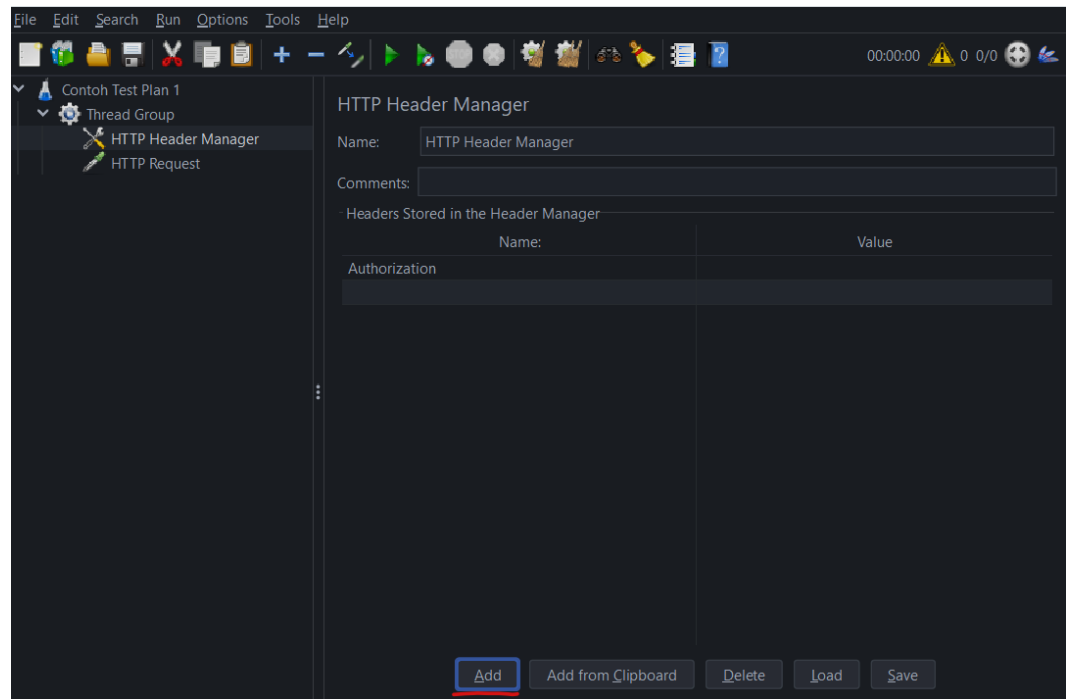
## Step 4 - Add a Config Element

### 1. Menambah HTTP Header Manager

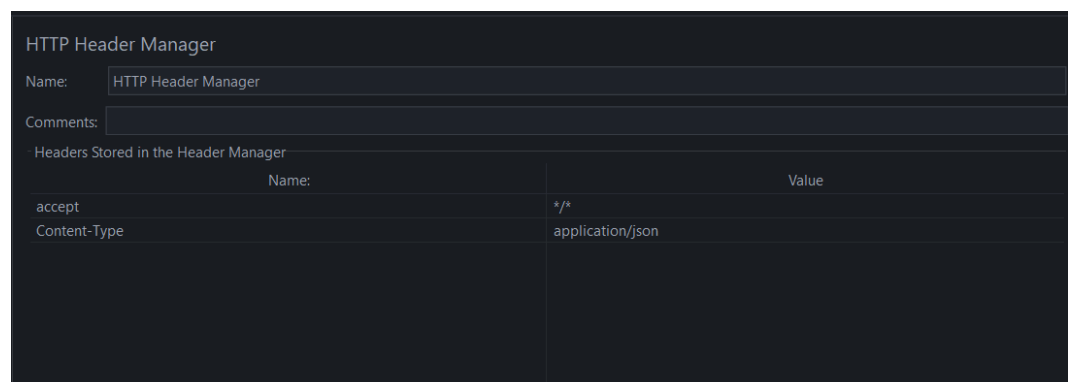
Cara menambahkan *http header manager* :

1. Pada *Thread Group*, klik kanan
2. Arahkan *mouse* pada “**Add >**”
3. Arahkan *mouse* pada “**Config Element >**”
4. Pilih “**Http Header Manager**”
5. Klik tombol “**Add**” untuk menambahkan value pada header.





Pada *header manager*, tambahkan informasi berupa :



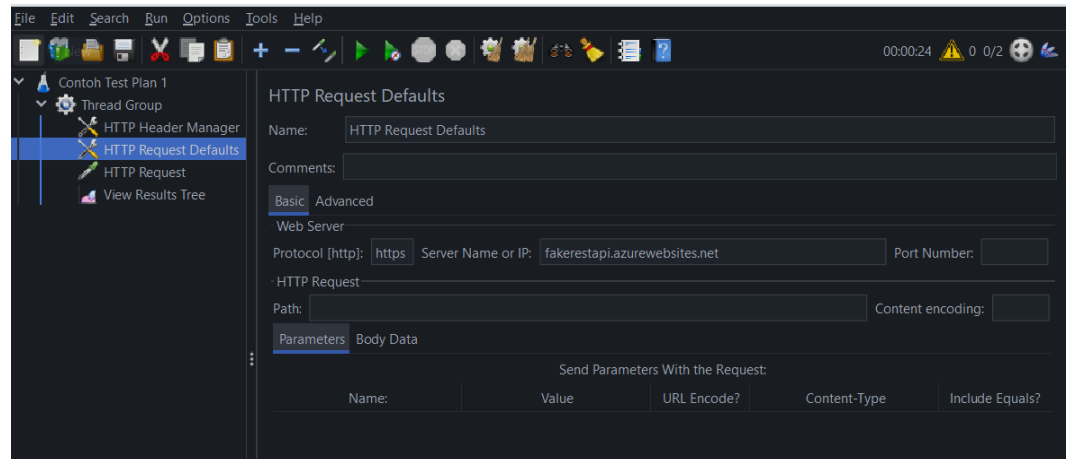
## 2. Menambah HTTP Request Default (Optional)

Setelah tahu *http request sampler*, teman-teman juga harus tahu apa itu *http request default*. *Http request default* digunakan untuk mengatur alamat *http request* secara default. *Sampler* ini akan berguna saat teman-teman mau mengirimkan banyak *request* dengan *method* yang berbeda.

Cara menambahkan *http request default* :

1. Pada *Thread Group*, klik kanan
2. Arahkan mouse pada “**Add >**”
3. Arahkan mouse pada “**Config Element >**”
4. Pilih “**Http Request Defaults**”

Di tampilan *http request default*, teman-teman bisa menambahkan informasi berupa *protocol*, *server name*, *port number*, dan *path*.

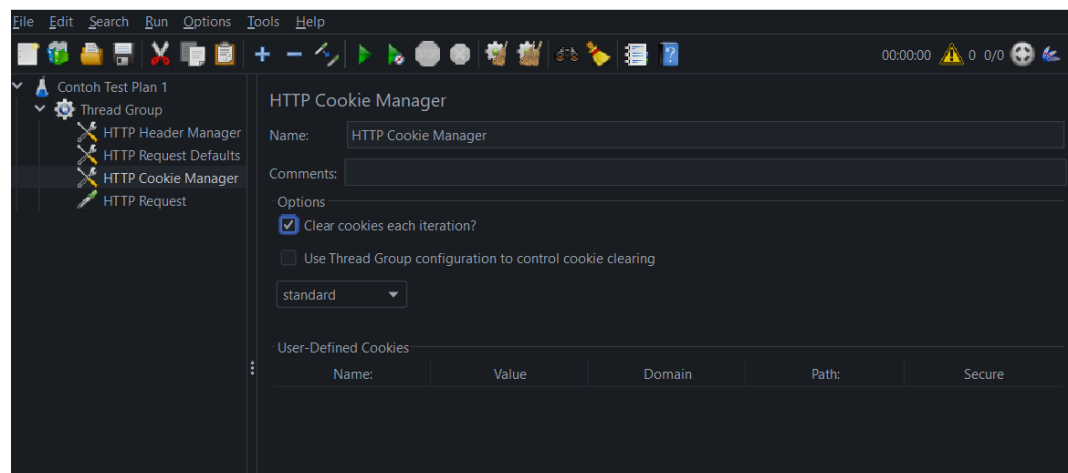


Jadi, kalau teman-teman sudah menggunakan *http request default*, pada halaman *http request*, teman-teman hanya perlu mengisi *method* dan *path* — nya saja (kalau *path* yang teman-teman jalankan lebih dari satu), karena JMeter akan otomatis menggunakan informasi yang teman-teman sudah tambahkan di *http request default*.

### 3. Menambah HTTP Cookie Manager (Optional)

*Http Cookie Manager* berguna untuk menangani *web server* yang menggunakan *cookies*. Untuk menambahkannya, dapat dilakukan dengan cara :

1. Pada *Thread Group*, klik kanan
2. Arahkan *mouse* pada “**Add >**”
3. Arahkan *mouse* pada “**Config Element >**”
4. Pilih “**Http Cookie Manager**”



#### 4. Menambah User Defined Variables

Kita akan menambahkan informasi global yang sering digunakan pada saat testing seperti informasi **baseUrl** dan **bookID** dengan User Defined Variables (UDV).

Untuk menambahkan elemen *User Defined Variables* :

1. Pada *Thread Group*, klik kanan
2. Arahkan *mouse* pada “**Add >**”
3. Arahkan *mouse* pada “**Config Element >**”
4. Pilih “**User Defined Variables**”
5. Klik tombol “**Add**” untuk menambahkan value.

Kemudian lakukan pengaturan *User Defined Variables* seperti gambar berikut:

The screenshot shows the 'User Defined Variables' configuration window. It has a 'Name' field with the value 'User Defined Variables' and an empty 'Comments' field. Below these is a table titled 'User Defined Variables' with three columns: 'Name', 'Value', and 'Description'. The table contains three rows of data:

Name	Value	Description
baseUrl	fakereapi.azurewebsites.net	
bookID	1	
protocol	https	

Pada gambar di atas kita menambahkan dua variabel yaitu **baseUrl** dan **bookID**. Nah variabel-variabel ini nantinya akan digunakan pada elemen lainnya. Kita akan menggunakan kedua variabel tersebut pada **HTTP Request Defaults** dengan memanggil variabel tersebut dengan format : `${NAMA_VARIABEL}`.

The screenshot shows the 'HTTP Request Defaults' configuration window. It has two tabs: 'Basic' and 'Advanced'. The 'Basic' tab is selected. In the 'Web Server' section, the 'Protocol' is set to 'http' and the 'Server Name or IP' is set to '\${baseUrl}'. In the 'HTTP Request' section, the 'Method' is set to 'GET' and the 'Path' is set to 'api/v1/Books/\${bookID}'. Below these are several checkboxes: 'Redirect Automatically' (unchecked), 'Follow Redirects' (checked), 'Use KeepAlive' (checked), 'Use multipart/form-data' (unchecked), and 'Browser-compatible headers' (unchecked). At the bottom, there is a 'Parameters' tab and a 'Send Parameters With the Request' section with fields for 'Name', 'Value', 'URL Encode?', 'Content-Type', and 'Include Equals?'.

## Step 5 - Add a Sampler (HTTP)

Setelah menambahkan *header manager*, kita dapat menambahkan *http request sampler*. *Http request sampler* ini merupakan tempat untuk menambahkan informasi berupa *protocol*, *IP Address*, *port number*, dan *method* serta *path* dari *web server* yang hendak kita uji.

Cara menambahkan *http request sampler* :

1. Pada *Thread Group*, klik kanan
2. Arahkan *mouse* pada “**Add >**”
3. Arahkan *mouse* pada “**Sampler >**”
4. Pilih “**HTTP Request**”

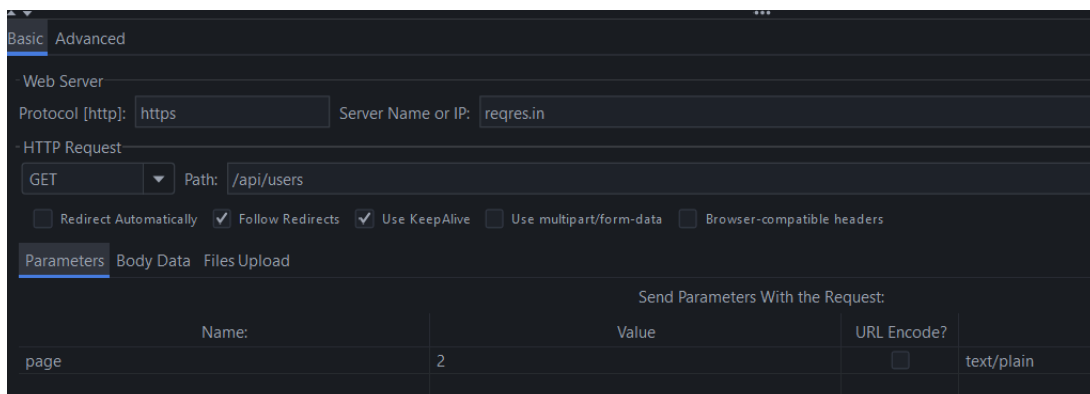
Saat tampilan *HTTP Request* terbuka, masukkan protokol *web server* yang kita gunakan (HTTP/HTTPS) kemudian masukkan *IP Address* atau *server name* lalu *port number*.

Setelah itu pilih *method* yang ingin kita gunakan (GET, POST, etc.) dan tambahkan *path* dari *API address* yang menjadi tujuan kita.

Dicontoh ini kita tambahkan :

- *Name* : *POST Books*
- *Protocol*: **kosongkan** (karena sudah di set di *HTTP Request Default*)
- *Server name*: **kosongkan** (karena sudah di set di *HTTP Request Default*)
- *Http request method*: POST
- *Path*: */api/v1/Books* (untuk *path* ini bisa dibuat *variable* juga di *UDV*)
- *Parameter*: **kosongkan**

*Ini digunakan kalau request kita perlu mengirimkan data dalam bentuk parameter, maka kita akan tambahkan parameter disini. Misalnya <https://reqres.in/api/users?page=2>. Jadi kita perlu menambahkan parameter “page” = 2, seperti contoh dibawah ini*



The screenshot shows the JMeter configuration window for an HTTP Request. The 'Basic' tab is active, displaying the following settings:

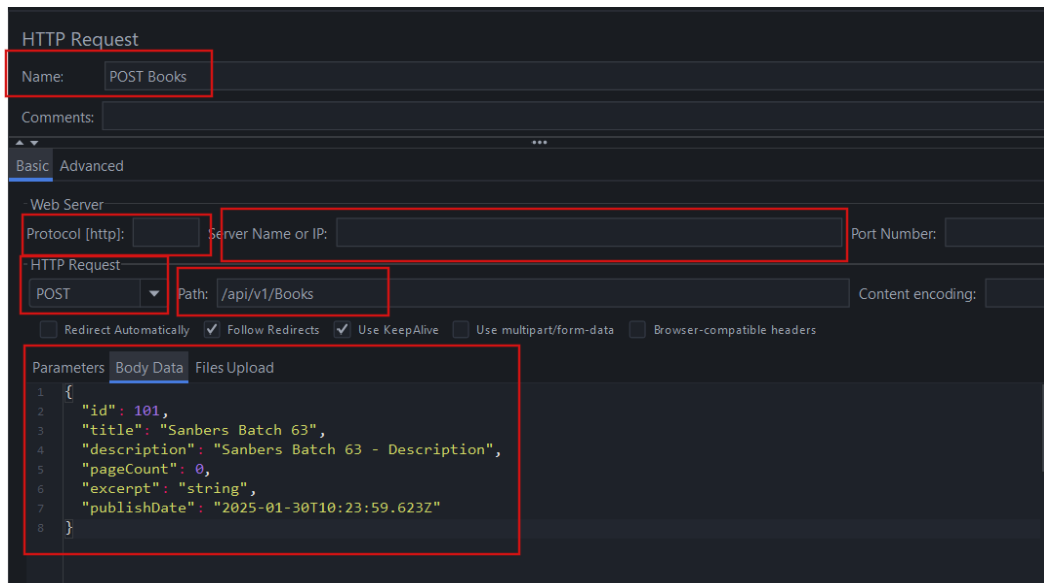
- Web Server:**
  - Protocol [http]:
  - Server Name or IP:
- HTTP Request:**
  - Method:
  - Path:
  - ☐ Redirect Automatically
  - ☒ Follow Redirects
  - ☒ Use KeepAlive
  - ☐ Use multipart/form-data
  - ☐ Browser-compatible headers
- Parameters:** (Selected tab)
  - Send Parameters With the Request: ☐
  - Table with 4 columns: Name, Value, URL Encode?, and Content Type.

Name	Value	URL Encode?	Content Type
page	2	<input type="checkbox"/>	text/plain

- **Body Data:**

Body data diisi sesuai dengan format json yang ada di swagger.

```
{
  "id": 101,
  "title": "Sanbers Batch 63",
  "description": "Sanbers Batch 63 - Description",
  "pageCount": 0,
  "excerpt": "string",
  "publishDate": "2025-01-30T10:23:59.623Z"
}
```

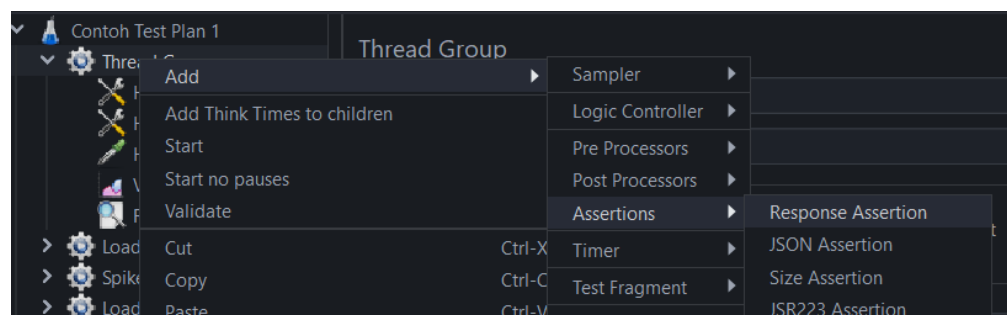


## Step 6 - Add Assertion

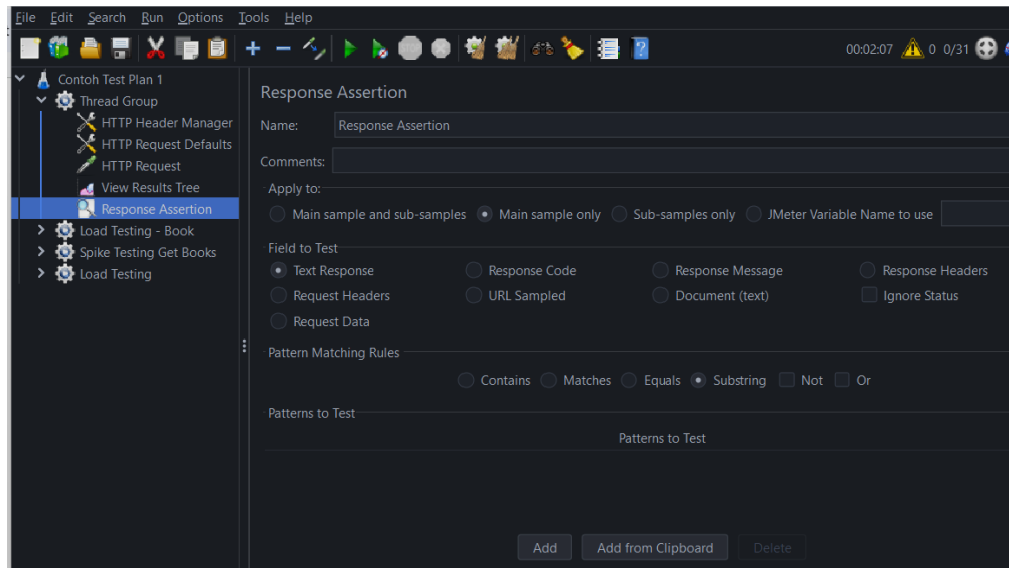
Untuk menambahkan assertion pada test plan kita, berikut adalah langkah-langkahnya:

*Step 1) Tambahkan Response Assertion*

Klik kanan **Thread Group** -> **Add** -> **Assertions** -> **Response Assertion**



Response Assertion panel akan muncul seperti gambar dibawah:



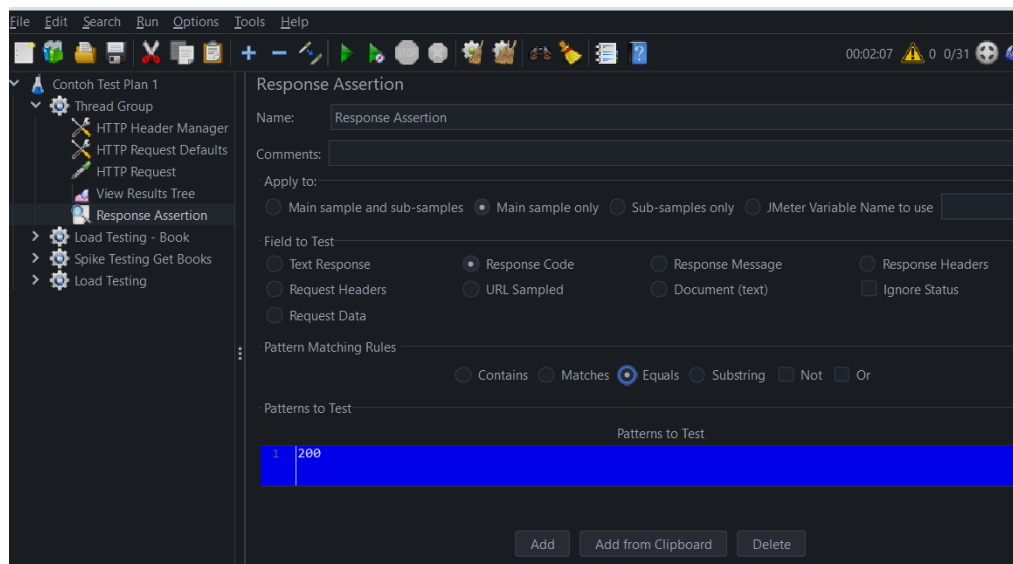
### Step 2) Tambahkan pola pada test

Saat kita mengirim permintaan ke server target, itu mungkin mengembalikan kode respons seperti di bawah ini:

- **200**: Server OK

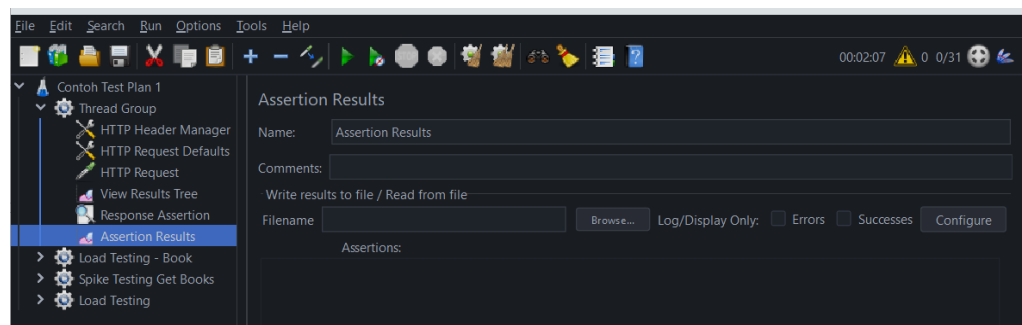
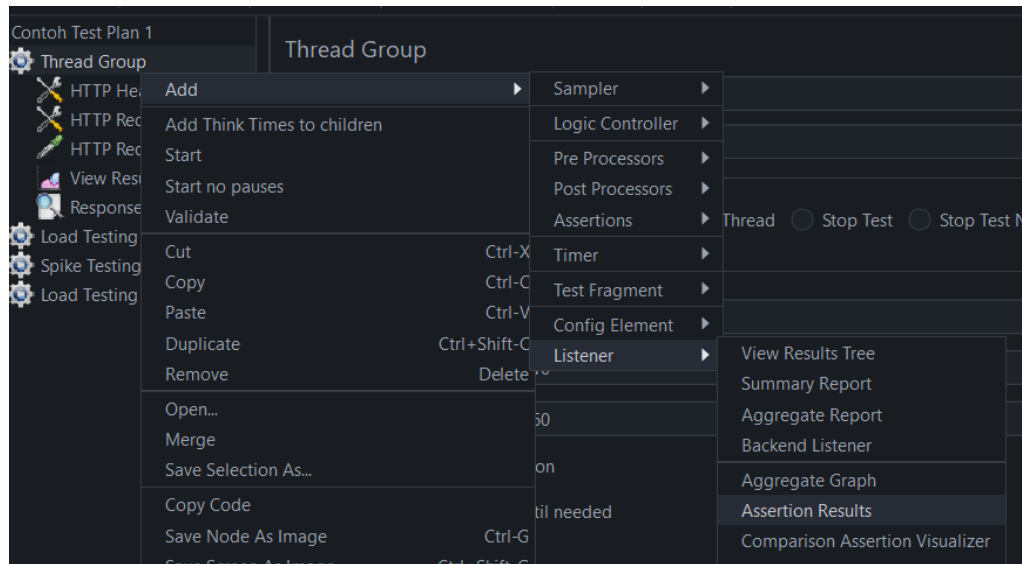
Pada **Response Field To Test**, pilih **Response Code**,

Pada Response Assertion Panel, klik **Add** -> tampilan entri kosong baru -> masukkan **200** pada Pattern to Test.



### Step 3) Tambahkan Assertion Results

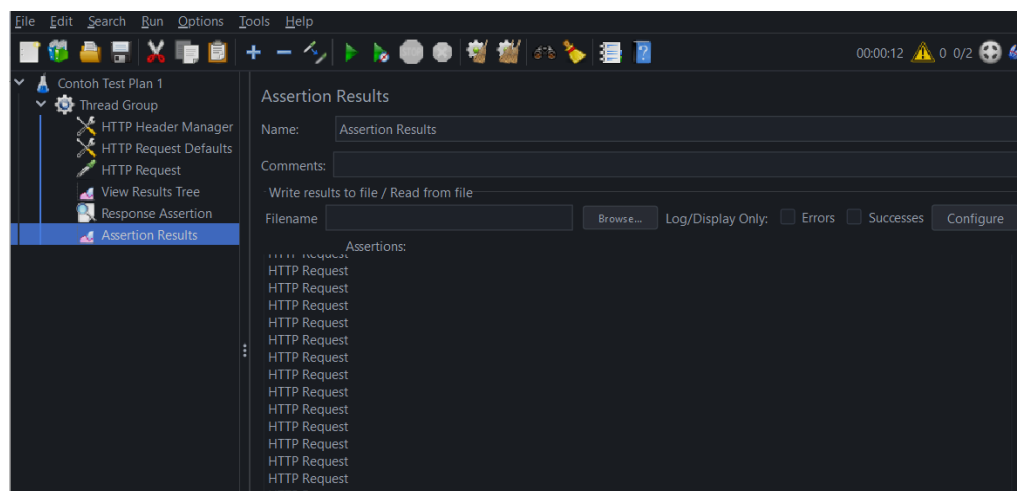
Klik kanan ada Thread Group, **Add** -> **Listener** -> **Assertion Results**

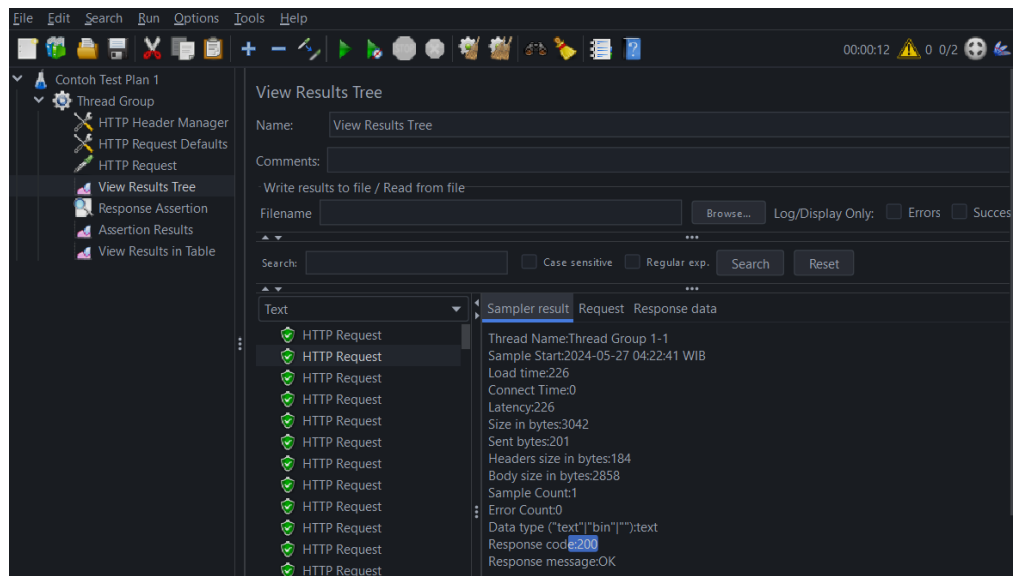
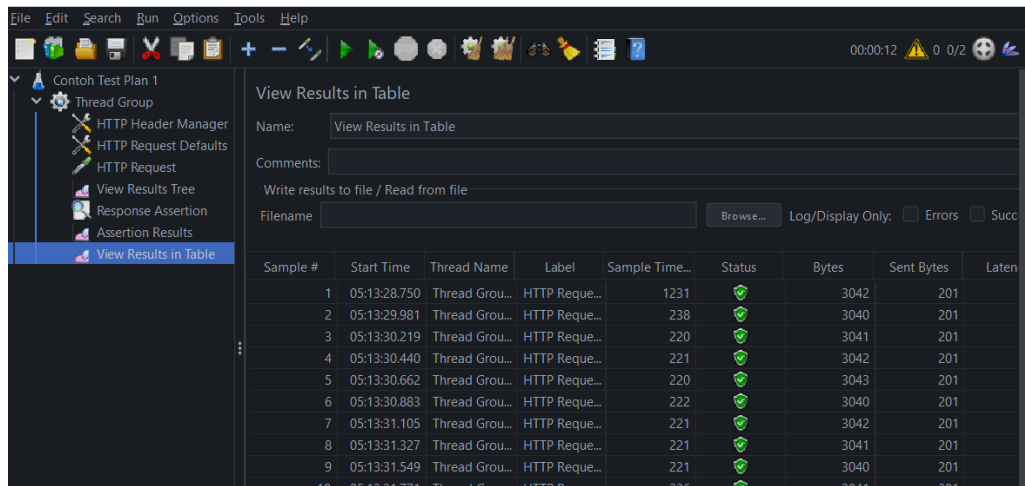


#### Step 4) Jalankan test

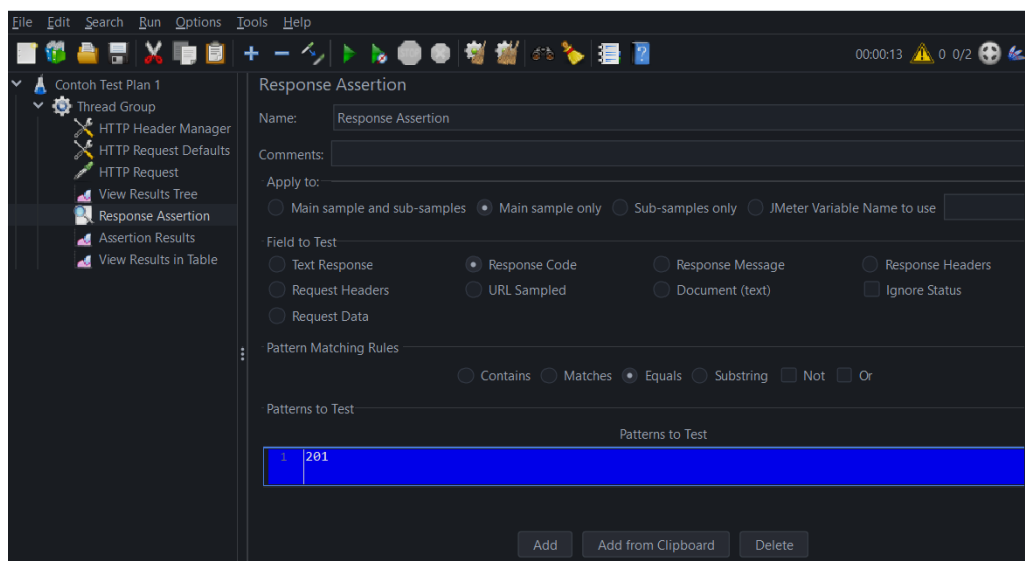
Saat kita siap menjalankan pengujian, klik tombol **Run** pada menu bar, atau shortcut **Ctrl+R**.

Hasil tes akan ditampilkan pada panel Assertion Results. Jika kode response server target berisi **200**, maka test case akan akan lolos. Kita akan melihat pesan yang ditampilkan sebagai berikut:



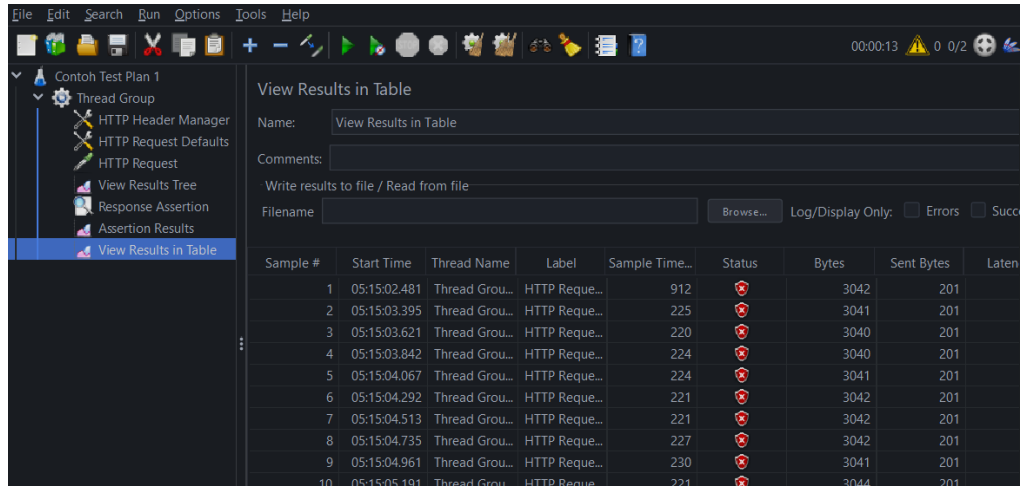


Sekarang kembali ke **Response Assertion Panel**, Kita mengubah Pola yang akan diuji dari **200** menjadi **201**.



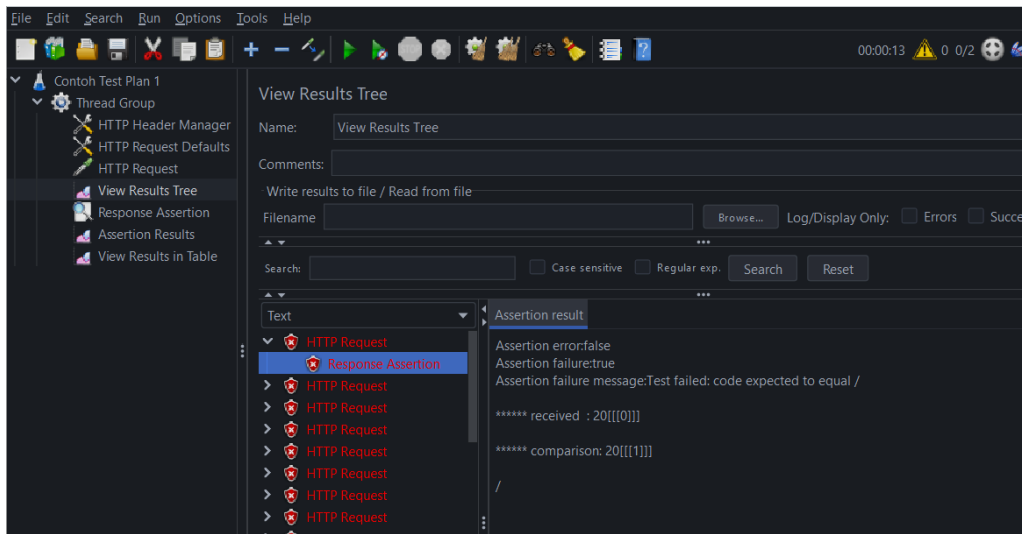
Karena kode respons server tidak berisi pola ini, kita akan melihat kasus pengujian **Failed** sebagai berikut:





The screenshot shows the 'View Results in Table' window in JMeter. The left sidebar lists the test plan structure: Contoh Test Plan 1, Thread Group, HTTP Header Manager, HTTP Request Defaults, HTTP Request, View Results Tree, Response Assertion, Assertion Results, and View Results in Table (selected). The main area displays a table of test results for the Thread Group.

Sample #	Start Time	Thread Name	Label	Sample Time...	Status	Bytes	Sent Bytes	Latency
1	05:15:02.481	Thread Grou...	HTTP Reque...	912	✖	3042	201	
2	05:15:03.395	Thread Grou...	HTTP Reque...	225	✖	3041	201	
3	05:15:03.621	Thread Grou...	HTTP Reque...	220	✖	3040	201	
4	05:15:03.842	Thread Grou...	HTTP Reque...	224	✖	3040	201	
5	05:15:04.067	Thread Grou...	HTTP Reque...	224	✖	3041	201	
6	05:15:04.292	Thread Grou...	HTTP Reque...	221	✖	3042	201	
7	05:15:04.513	Thread Grou...	HTTP Reque...	221	✖	3042	201	
8	05:15:04.735	Thread Grou...	HTTP Reque...	227	✖	3042	201	
9	05:15:04.961	Thread Grou...	HTTP Reque...	230	✖	3041	201	
10	05:15:05.191	Thread Grou...	HTTP Reque...	221	✖	3044	201	



The screenshot shows the 'View Results Tree' window in JMeter. The left sidebar lists the test plan structure: Contoh Test Plan 1, Thread Group, HTTP Header Manager, HTTP Request Defaults, HTTP Request, View Results Tree (selected), Response Assertion, Assertion Results, and View Results in Table. The main area displays a tree view of the test results.

Text	Assertion result
✖ HTTP Request	Assertion error:false
✖ Response Assertion	Assertion failure:true
> ✖ HTTP Request	Assertion failure message:Test failed: code expected to equal /
> ✖ HTTP Request	***** received : 20[[[0]]]
> ✖ HTTP Request	***** comparison: 20[[[1]]]
> ✖ HTTP Request	/

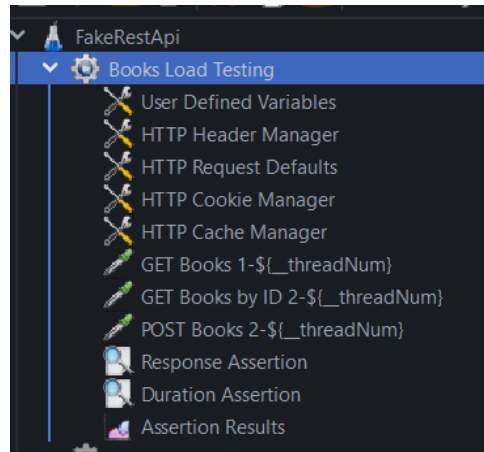
## Step 7 - Add Listeners

Setelah menyusun kerangka mulai dari *http header manager* sampai ke *http request*, kita butuh *listener*. *Listener* adalah komponen pada JMeter yang menunjukkan hasil dari skenario yang kita susun. *Listener* memiliki banyak jenis, teman-teman bisa memilih dalam bentuk apa hasil pengujian itu disajikan. Bisa dalam bentuk *tree*, *table*, *graph*, atau dalam bentuk *log*.

Cara menambahkan *listener* :

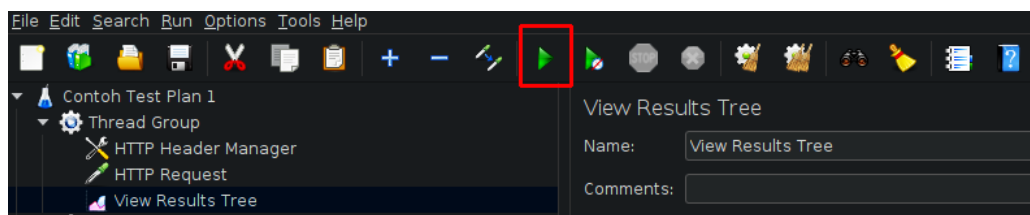
1. Pada *Thread Group*, klik kanan
2. Arahkan *mouse* pada “**Add >**”
3. Arahkan *mouse* pada “**Listener >**”
4. Teman-teman bisa memilih jenis *listener* apa saja (contoh disini menggunakan *listener* dalam bentuk *tree*)

Jadi, kerangka pengujian kita kali ini adalah seperti pada gambar berikut :

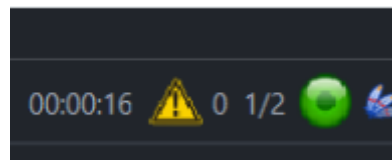


### Step 8 - To Run the Test

Setelah semua kerangka pengujian telah siap, klik tombol hijau untuk menjalankan skenario yang sudah kita susun.

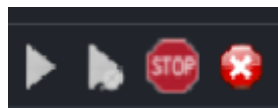


Saat JMeter dijalankan, akan menampilkan kotak hijau kecil di ujung kanan bilah menu.

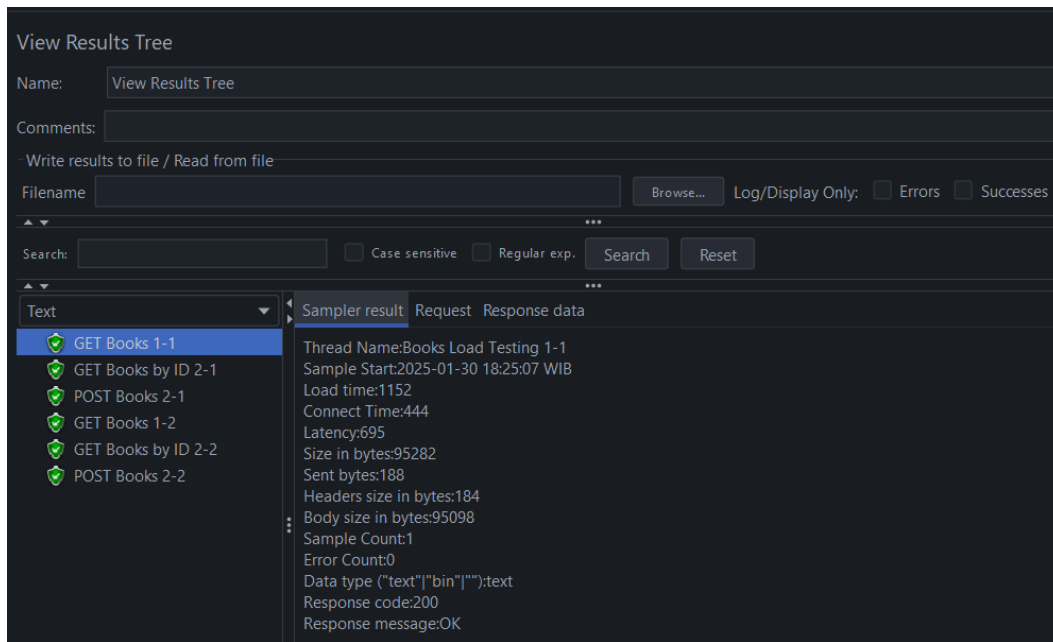


Angka disebelah kiri kotak hijau adalah jumlah thread yang aktif/jumlah total thread.

Untuk Menghentikan Tes, tekan tombol Stop atau gunakan tombol pendek Ctrl + '.



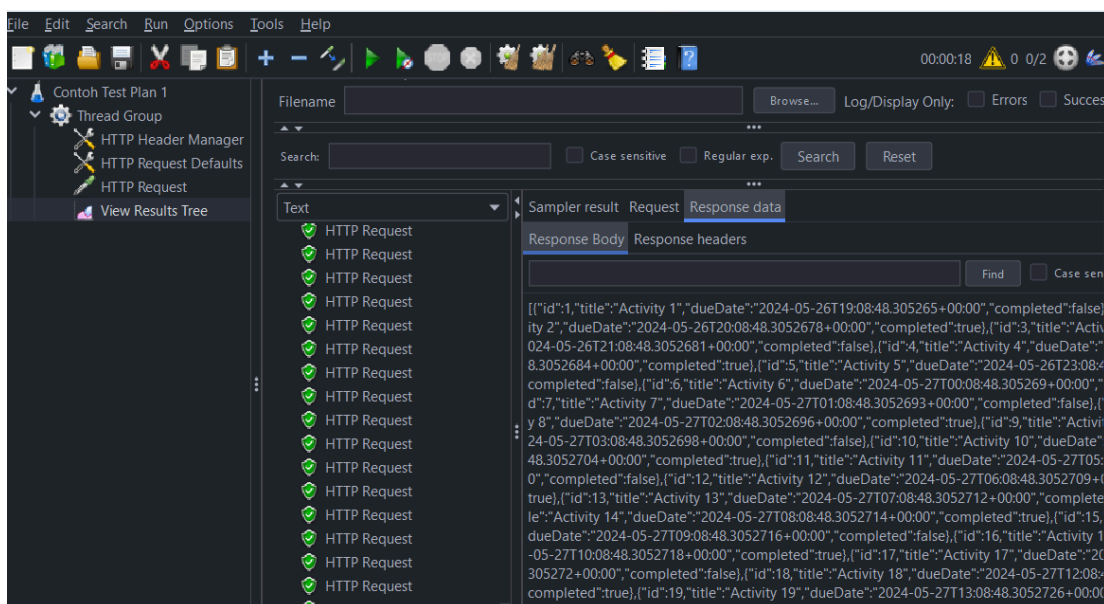
Hasil dari pengujian yang kita lakukan akan tampil di *listener* “View Results Tree” seperti pada gambar berikut.



Kalau kita perhatikan, dari *sampler result* diberikan informasi berupa *load time*, *latency*, *error count*, *response code*, *response message*, dan sebagainya.

Biasanya, melalui informasi-informasi ini, QA diharapkan mampu menarik kesimpulan terkait ketahanan sebuah *web server* saat diberikan tekanan yang bervariasi.

Untuk melihat *response body* dari API yang kita *hit* dapat dilihat dari bagian “**Response Data**” seperti pada gambar berikut.



# Action to be Taken After Sampler Error

Berikut adalah penjelasan mengenai perbedaan antara beberapa opsi kontrol alur eksekusi di JMeter, yaitu Continue, Start Next Thread Loop, Stop Thread, Stop Test, dan Stop Test Now:

## 1. Continue

**Deskripsi:** Opsi ini digunakan untuk melanjutkan eksekusi ke langkah berikutnya dalam urutan pengujian saat ini.

**Kapan Digunakan:** Kita akan menggunakan opsi ini ketika kita ingin melanjutkan eksekusi tanpa menghentikan atau mengubah alur pengujian. Misalnya, jika ada kondisi tertentu yang tidak terpenuhi, tetapi kita ingin tetap melanjutkan ke langkah berikutnya dalam thread yang sama.

## 2. Start Next Thread Loop

**Deskripsi:** Opsi ini digunakan untuk melanjutkan eksekusi ke loop berikutnya dari thread yang sedang berjalan.

**Kapan Digunakan:** Jika kita memiliki beberapa thread dan setiap thread memiliki loop (pengulangan), opsi ini akan memindahkan eksekusi ke iterasi berikutnya dari thread yang sama. Misalnya, jika kita memiliki 5 thread dan setiap thread memiliki 3 loop, memilih opsi ini akan memulai loop kedua untuk thread yang sedang berjalan.

## 3. Stop Thread

**Deskripsi:** Opsi ini menghentikan eksekusi thread yang sedang berjalan.

**Kapan Digunakan:** Kita akan menggunakan opsi ini ketika kita ingin menghentikan eksekusi thread tertentu, tetapi tetap melanjutkan eksekusi thread lainnya. Misalnya, jika ada kesalahan dalam satu thread dan kita ingin menghentikannya tanpa mempengaruhi thread lain.

## 4. Stop Test

**Deskripsi:** Opsi ini menghentikan seluruh pengujian yang sedang berjalan.

**Kapan Digunakan:** Kita akan menggunakan opsi ini ketika kita ingin menghentikan semua thread dan menghentikan pengujian secara keseluruhan. Ini berguna jika kita ingin menghentikan pengujian karena alasan tertentu, seperti kesalahan kritis atau kebutuhan untuk menghentikan pengujian.

## 5. Stop Test Now

**Deskripsi:** Opsi ini menghentikan pengujian secara langsung dan segera, tanpa menunggu thread yang sedang berjalan untuk menyelesaikan eksekusi.

**Kapan Digunakan:** Kita akan menggunakan opsi ini ketika kita perlu menghentikan pengujian dengan segera, tanpa menunggu thread untuk menyelesaikan permintaan yang sedang berlangsung. Ini berguna dalam situasi darurat atau ketika kita perlu menghentikan pengujian dengan cepat.

Perbedaan Stop Test dan Stop Test Now :

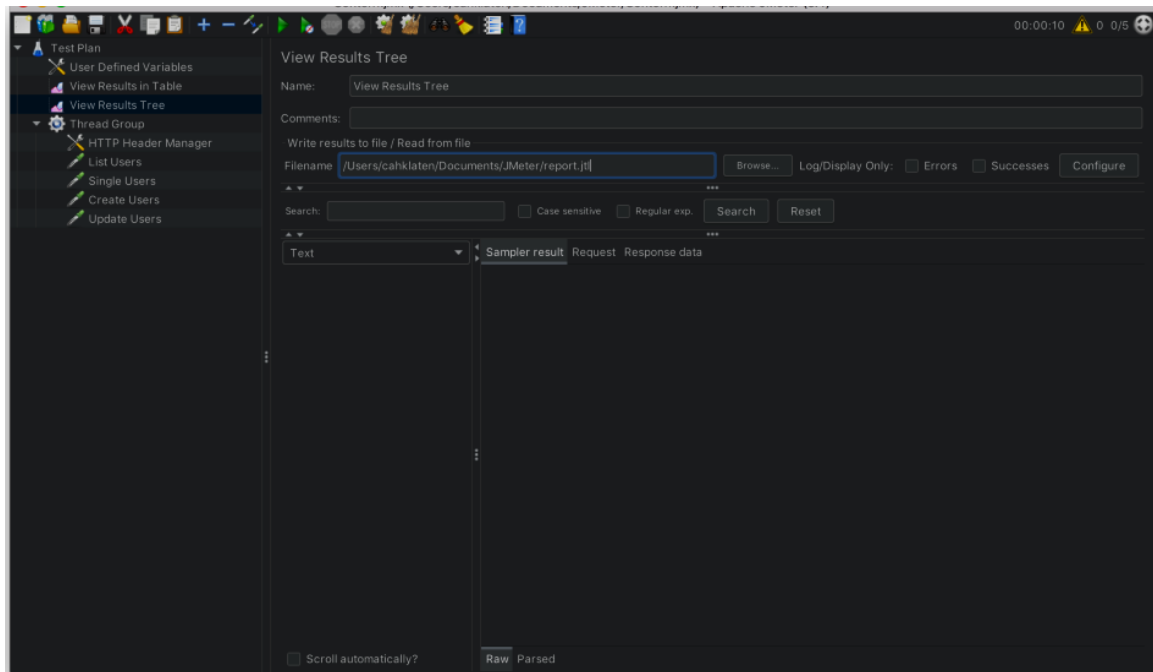
- **Stop Test:** Kita sedang melakukan pengujian beban dan ingin menghentikan pengujian setelah semua pengguna selesai melakukan permintaan mereka. Kita memilih opsi ini untuk memastikan bahwa semua data hasil pengujian tercatat dengan baik.
- **Stop Test Now:** Kita menjalankan pengujian dan tiba-tiba menemukan bahwa ada kesalahan fatal dalam konfigurasi yang dapat menyebabkan kerusakan. Kita memilih opsi ini untuk segera menghentikan semua eksekusi tanpa menunggu permintaan yang sedang berlangsung.

## HTML Reporting

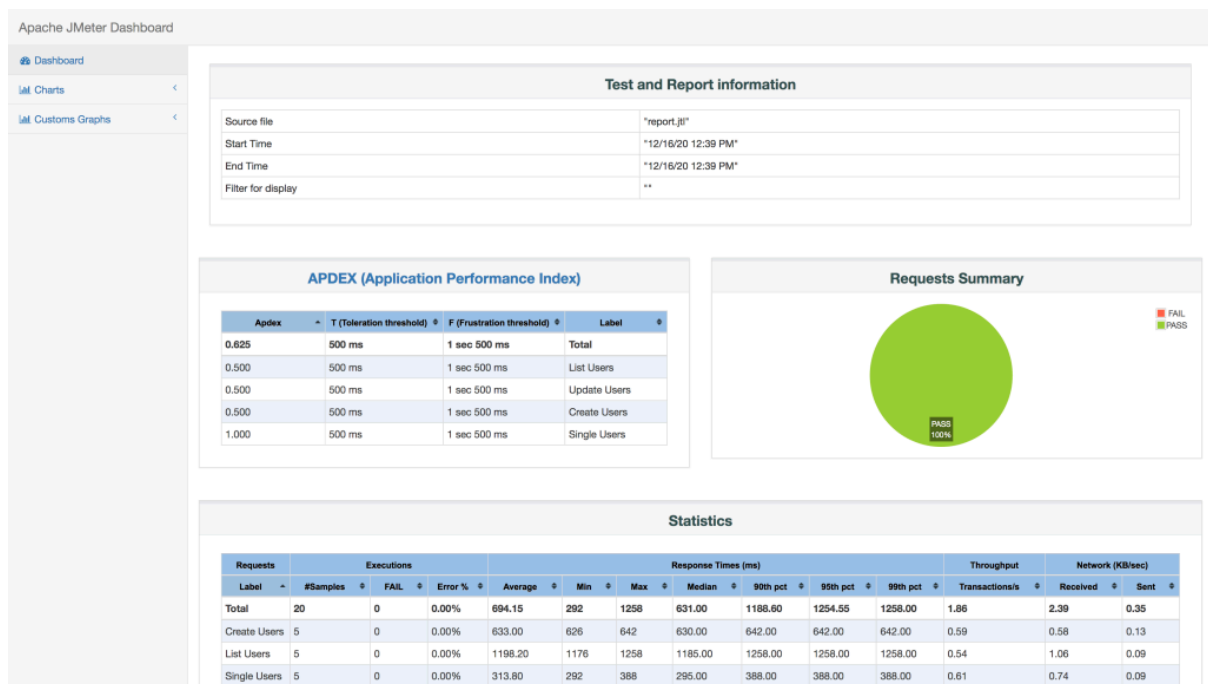
JMeter memiliki fitur untuk men-*generate* sebuah report dengan format html. Caranya cukup mudah, yaitu dengan membuat *JMeter* men-*generate* sebuah *output file* dengan ekstensi *.jtl*.

Dengan masih mengacu pada materi sebelumnya, kita dapat membuat *file .jtl* dengan cara:

1. Klik salah satu *listener* yang sudah dibuat, misalnya *View Result in Tree*.
2. Pada segment **Write result to file/Read from file**, di field **Filename** ketikkan direktori *path* beserta *output file .jtl* nya. Misal:  
`/Users/cahklaten/Documents/JMeter/report.jtl`
3. Kemudian jalankan *testing* dengan klik tombol *Play*.
4. Jika sudah selesai, buka *terminal/command prompt*.
5. Masuk ke direktori *path* pada *step 2*.
6. Kemudian ketikkan: `jmeter -g report.jtl -o report`  
**-g report.jtl** berarti *generate file report.jtl*  
**-o report** berarti *ouput generate file ada di folder report*



Pengaturan output file .jtl



Contoh report (dashboard)



Contoh report Response Time

## How to Analyze JMeter Test Result

Setelah melakukan pengujian beban pada sistem yang sedang dikembangkan, apa tindakan selanjutnya yang perlu dilakukan oleh QA?

Benar, menganalisis hasilnya lalu memberikan laporan sehingga tim *developer* dapat mengetahui apa kelemahan dari sistem yang sedang mereka kembangkan.

Kita tahu bahwa laporan yang disediakan oleh JMeter bervariasi, bisa dalam bentuk *tree*, *table*, *graph*, maupun *log*. Laporan itu bisa kita dapatkan dengan menambah komponen *Listener* pada JMeter dan dari laporan tersebut, kita sebagai QA atau *Software Tester* diharapkan dapat membantu *developer* dalam menganalisis hasil pengujian yang kita lakukan sesuai dengan skenario yang sudah dirancang sebelumnya.

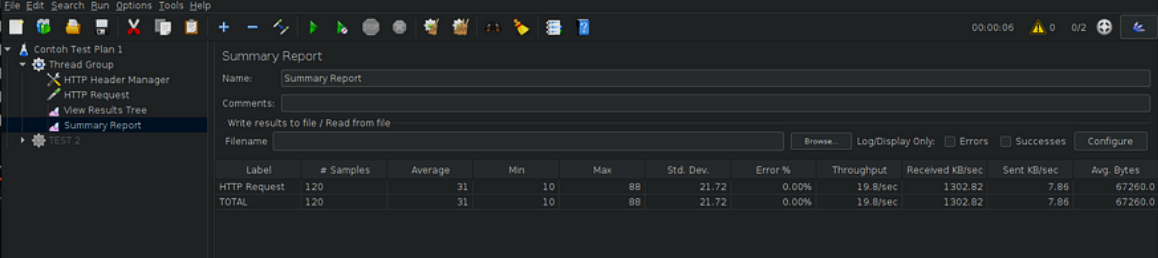
Kali ini kita akan menganalisis hasil dari pengujian *performance* dari contoh kasus yang sudah kita jalankan pada materi sebelumnya melalui salah satu jenis *report* yaitu *Summary Report*.

## Summary Report

*Summary Report* adalah salah satu jenis laporan yang disediakan oleh JMeter.

Cara menambahkan *Summary Report* adalah :

1. Klik kanan pada **Thread Group**
2. Arahkan *mouse*, pilih **Add >**
3. Arahkan kembali *mouse*, pilih **Listener >**
4. Pilih **Summary Report**



The screenshot shows the JMeter Summary Report window. The 'Name' field is set to 'Summary Report'. The 'Comments' field is empty. The 'Write results to file / Read from file' section has a 'Filename' field and a 'Browse...' button. The 'Log/Display Only' section has checkboxes for 'Errors', 'Successes', and 'Configure'. The table below shows the test results for 'TEST 2'.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	120	31	10	88	21.72	0.00%	19.8/sec	1302.82	7.86	67260.0
TOTAL	120	31	10	88	21.72	0.00%	19.8/sec	1302.82	7.86	67260.0

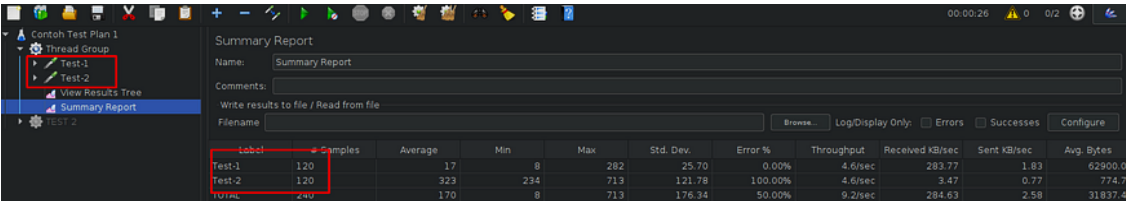
Tampilan Summary Report

Seperti pada gambar tampilan *summary report* di atas, kita dapat melihat beberapa komponen, diantaranya :

- **Name** : Berupa judul atau nama dari *summary report* kita.
- **Comments** : Berupa catatan tambahan yang mungkin dimiliki oleh QA.
- **Write results to file / Read from file** : Kita dapat membaca hasil dari pengujian yang sebelumnya sudah kita lakukan dengan *browsing file* dan hasil *summary report* dari *file* itu akan ditampilkan pada tabel.

Laporan pada *Summary Report* memberikan beberapa informasi yang disajikan dalam bentuk tabel dengan beberapa komponen, yaitu *Label*, *Samples*, *Average*, *Min*, *Max*, *Std.Dev.*, *Error %*, *Throughput*, *Received KB/sec*, *Sent KB/sec*, dan *Avg. Bytes*.

- **Label** : Merupakan nama dari *HTTP Request* yang kita jalankan. Misalnya ada dua *request* yang sedang kita jalankan secara bersamaan, dimana nama dari masing-masing *http request* tersebut adalah Test-1 dan Test-2, maka Label dari *Summary Report* tersebut ada Test-1 dan Test-2. Jelasnya dapat dilihat pada gambar berikut.



The screenshot shows the JMeter Summary Report window. The 'Name' field is set to 'Summary Report'. The 'Comments' field is empty. The 'Write results to file / Read from file' section has a 'Filename' field and a 'Browse...' button. The 'Log/Display Only' section has checkboxes for 'Errors', 'Successes', and 'Configure'. The table below shows the test results for 'TEST 2'.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Test-1	120	17	8	282	25.70	0.00%	4.6/sec	283.77	1.83	62900.0
Test-2	120	323	234	713	121.78	100.00%	4.6/sec	3.47	0.77	774.7
TOTAL	240	170	8	713	176.34	50.00%	9.2/sec	284.63	2.58	31837.4

Label HTTP Request



- **Samples** : Komponen yang mengindikasikan jumlah pengguna virtual (*virtual users*) per *request*. Jumlah *samples* ini mengacu pada *number of threads*, *ramp-up period*, dan *loop count* yang sebelumnya sudah kita *state* di *Thread Group*.

Misalnya skenario yang hendak kita jalankan adalah *number of thread* = 2, *ramp-up period* = 10, dan *loop count* = 60, maka jumlah *samples* yang kita miliki adalah 120.

- **Average** : Mengindikasikan waktu rata-rata yang dihabiskan dalam mengeksekusi masing-masing label. Dalam kasus kita, *average time* dari label *Test-1* adalah 17 milisekon dan label *Test-2* adalah 323 milisekon, sehingga total *average* yang dihasilkan adalah  $(17+323)/2 = 170$ .
- **Min** : Mengindikasikan waktu tersingkat yang dibutuhkan dalam mengeksekusi masing-masing label. Dalam kasus kita, nilai *min* untuk label *Test-1* adalah 8 milisekon dan label *Test-2* adalah 234, sehingga total *min* yang dihasilkan adalah 8 (diambil dari nilai *min* paling kecil dari masing-masing label).
- **Max** : Mengindikasikan waktu terpanjang yang dibutuhkan dalam mengeksekusi masing-masing label. Dalam kasus kita, nilai *max* untuk label *Test-1* adalah 282 milisekon dan label *Test-2* adalah 713, sehingga total *max* yang dihasilkan adalah 713 (diambil dari nilai *max* paling besar dari masing-masing label).
- **Std. Dev** : Menunjukkan penyebaran kumpulan data relatif terhadap rata-ratanya. Semakin kecil nilai dari *std.dev* menunjukkan bahwa data yang dijalankan pada masing-masing label semakin konsisten. Nilai dari *std.dev* sebaiknya lebih kecil atau sama dengan setengah dari nilai *average* dari setiap label. Dari contoh kasus yang kita jalankan dapat kita tarik kesimpulan bahwa data yang dijalankan pada label *Test-2* masih lebih konsisten dibanding data yang dijalankan pada label *Test-1* karena pada label *Test-1* nilai *std.dev* > *average*, sedangkan di label *Test-2*, nilai *std.dev* < *average*.
- **Error %** : Menunjukkan jumlah *error* dalam satuan persen yang terjadi pada setiap label. Dalam kasus kita, di label *Test-1* dilihat bahwa jumlah *error* nya 0%, sedangkan di label *Test-2* sebesar 100%.
- **Throughput** : Menunjukkan jumlah *request* yang berhasil diproses per time unit (*second*, *minute*, *hours*) oleh server. Waktu ini dikalkulasikan dari awal *sample* pertama dijalankan sampai *sample* terakhir. Berbeda dengan *std.dev*, semakin besar nilai *throughput*, semakin bagus. Dari contoh kasus yang kita jalankan, di label *Test-1* ada 4.6196489... *request* yang berhasil diproses oleh server yang kita jalankan per *second*, dan di label *Test-2* ada 4.5889101... (dibulatkan menjadi 4.6) *request* per *second*.
- **Received KB/sec** : Mengindikasikan jumlah data yang berhasil diunduh oleh server selama dilakukannya eksekusi pengujian *performance* dalam satuan *kilobyte* tiap 1 sekon.

- **Sent KB/sec** : Mengindikasikan jumlah data yang berhasil dikirim dari *server* selama dilakukannya eksekusi pengujian *performance* dalam satuan *kilobyte* tiap 1 sekon.
- **Avg Bytes** : Merupakan rata-rata *byte* yang berhasil diunduh (*download*) oleh *server*.

*Now let's do an experiment*, kalau ada contoh kasus dimana dijalankan skenario dengan informasi sebagai berikut :

Number of threads : 5000

Ramp-up period : 1

Loop-count : 1

Kemudian hasilnya :

Average : 738

Min : 155

Max : 2228

Throughput : 60.5

Dari hasil di atas kita dapat menarik kesimpulan bahwa dari semua *thread* yang dijalankan dalam waktu 1 detik (*ramp-up*), dibutuhkan waktu rata-rata eksekusi 738 *milliseconds* atau 0.7 *seconds* dengan waktu minimum yang dibutuhkan untuk mengeksekusi *thread* yaitu 155 *milliseconds* dan waktu maksimumnya 2228 *milliseconds*. Dari *throughput* kita dapat melihat *server* mampu mengeksekusi 60.5 *thread* per time unit.

## Tugas Day 3

1. Buatlah satu Test Plan pada Jmeter, silahkan gunakan server apa saja (misalnya : google.com), buatlah dengan ketentuan sebagai berikut:
    - Silahkan tentukan sendiri jumlah users, ramp-up periods, dan loop count nya
    - Pada sampler, tambahkan config element misalnya HTTP Header Manager atau HTTP Request Default
    - Tambahkan minimal 2 listeners
  2. Buat Skenario lain untuk 1000 Concurrent per minute
  3. Kemudian jalankan test plan tersebut, dan berikan hasil analisa dari pengujian performance teman-teman dalam bentuk laporan/word.
- Kirimkan tugas dalam bentuk .zip yang berisi file .jmx dan file laporan analisis teman-teman.