

Laporan
Pengujian dan penjaminan Kualitas
Perangkat Lunak

**“Automated Testing : Pengujian non Fungsional – Performance Testing
dengan k6”**



Disusun oleh
Akmal Rendiansyah
Nim.2211083040

Prodi D4 Teknologi Rekayasa Perangkat Lunak
Jurusan Teknologi Informasi
Politeknik Negeri Padang

2024

A. Landasan Teori

Automated testing atau pengujian otomatis adalah metode pengujian perangkat lunak menggunakan alat otomatis untuk menjalankan tes yang telah direncanakan, membandingkan hasil aktual dengan hasil yang diharapkan, dan melaporkan hasilnya. Ini berlawanan dengan pengujian manual, di mana penguji manusia harus melakukan semua langkah pengujian secara manual. Automated testing membantu mempercepat proses pengujian, meningkatkan cakupan pengujian, dan mengurangi kesalahan manusia.

Seiring dengan kemajuan teknologi perangkat lunak, kompleksitas aplikasi dan kebutuhan akan pengujian yang lebih cepat dan akurat meningkat. Dalam dekade terakhir, perkembangan dalam metode pengembangan perangkat lunak seperti Agile dan DevOps telah mendorong perlunya integrasi pengujian otomatis untuk mendukung siklus pengembangan yang lebih pendek dan pengiriman berkelanjutan (continuous delivery).

Performance testing difokuskan pada beberapa aspek kunci:

1. Kecepatan (Speed): Mengukur seberapa cepat sistem merespon permintaan pengguna.
2. Skalabilitas (Scalability): Menentukan jumlah maksimum pengguna atau beban yang dapat ditangani sistem secara efektif.
3. Stabilitas (Stability): Menganalisis stabilitas sistem di bawah beban yang terus meningkat atau skenario beban yang kompleks.

Jenis – jenis Performance Testing

- Load Testing: Pengujian ini mengukur kapasitas sistem dengan meningkatkan beban pengguna secara bertahap untuk menentukan kemampuan maksimum sistem dalam kondisi normal. Contoh aplikasi adalah menguji ketahanan sistem penjualan tiket yang menghadapi peningkatan traffic selama acara-acara tertentu.
- Stress Testing: Dilakukan untuk menentukan batas kemampuan sistem dengan memberikan beban berlebih hingga sistem mengalami kegagalan. Ini membantu memahami titik jenuh dan kegagalan sistem pada skenario beban ekstrim, seperti lonjakan trafik mendadak pada aplikasi e-commerce selama promosi besar-besaran.

- **Endurance Testing:** Bertujuan untuk menguji kinerja sistem di bawah beban normal untuk jangka waktu yang panjang untuk mendeteksi potensi kebocoran memori atau masalah degradasi performa lainnya yang mungkin muncul selama operasi terusmenerus.
- **Spike Testing:** Menguji respon sistem terhadap lonjakan beban secara tiba-tiba dan besar untuk memastikan sistem dapat menangani lonjakan pengguna yang signifikan tanpa mengalami penurunan kinerja. Contoh aplikasi adalah pengujian akses website hasil ujian yang diakses banyak pengguna dalam waktu yang bersamaan.

Dalam melaksanakan performance testing, alat otomatisasi seperti Apache JMeter, WebLoad, dan SmartMeter.io sering digunakan. Alat-alat ini memungkinkan pengujian beban dan tekanan, serta mendukung berbagai protokol, yang menjadikannya pilihan populer di kalangan software tester dan QA. Pemilihan alat tergantung pada kebutuhan spesifik dan skenario pengujian yang akan dilakukan

B. Langkah – langkah pengujian non fungsional denga K6

1. Pertama kunjungi situs Chocholatey dan copy bagian Installation seperti dibawah ini.

Install Using Windows Cmd Shell

First, we need ensure that we are using an administrative shell.

Next, copy the following command to the *cmd.exe* shell.

```
@"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile -InputFormat None -ExecutionPolicy Bypass -Command "[System.Net.ServicePointManager]::SecurityProtocol = 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))" && SET "PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin"
```

And then press enter.

2. Lakukan Install Chocolatey di Command Prompt dan jalankan sebagai Administrator.

```

C:\Windows\System32>@%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile -InputFormat None -ExecutionPolicy Bypass -Command "[System.Net.ServicePointManager]::SecurityProtocol = 3872; iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))" && SET "PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin"
Forcing web requests to allow TLS v1.2 (Required for requests to Chocolatey.org)
Getting latest version of the Chocolatey package for download.
Not using proxy.
Getting Chocolatey from https://community.chocolatey.org/api/v2/package/chocolatey/2.2.2.
Downloading https://community.chocolatey.org/api/v2/package/chocolatey/2.2.2 to C:\Users\CINDYS-1\AppData\Local\Temp\chocolatey\chocoInstall\chocolatey.zip
Not using proxy.
Extracting C:\Users\CINDYS-1\AppData\Local\Temp\chocolatey\chocoInstall\chocolatey.zip to C:\Users\CINDYS-1\AppData\Local\Temp\chocolatey\chocoInstall
Installing Chocolatey on the local machine
Creating ChocolateyInstall as an environment variable (targeting 'Machine')
Setting ChocolateyInstall to 'C:\ProgramData\chocolatey'
WARNING: It's very likely you will need to close and reopen your shell
before you can use choco.
Restricting write permissions to Administrators
We are setting up the Chocolatey package repository.
The packages themselves go to 'C:\ProgramData\chocolatey\lib'
(i.e. C:\ProgramData\chocolatey\lib\yourPackageName).
A shim file for the command line goes to 'C:\ProgramData\chocolatey\bin'
and points to an executable in 'C:\ProgramData\chocolatey\lib\yourPackageName'.
Creating Chocolatey folders if they do not already exist.

chocolatey.nupkg file not installed in lib.
Attempting to locate it from bootstrapper.
WARNING: Not setting tab completion: Profile file does not exist at 'C:\Users\CINDY
STEFANI\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1'.
Chocolatey (choco.exe) is now ready.
You can call choco from anywhere, command line or powershell by typing choco.
Run choco ?? for a list of functions.
You may need to shut down and restart powershell and/or consoles
first prior to using choco.
Ensuring Chocolatey commands are on the path
Ensuring chocolatey.nupkg is in the lib folder

```

3. Selanjutnya lakukan Install k6

```

C:\Windows\System32>choco install k6
Chocolatey v2.2.2
Installing the following packages:
k6
By installing, you accept licenses for the packages.
Progress: Downloading k6 0.51.0... 100%

k6 v0.51.0 [Approved]
k6 package files install completed. Performing other installation steps.
The package k6 wants to run 'chocolateyInstall.ps1'.
Note: If you don't run this script, the installation will fail.
Note: To confirm automatically next time, use '-y' or consider:
choco feature enable -n allowGlobalConfirmation
Do you want to run the script?([Y]es/[A]ll - yes to all/[N]o/[P]rint): y

Extracting 64-bit C:\ProgramData\chocolatey\lib\k6\tools\k6-v0.51.0-windows-amd64.zip to C:\ProgramData\chocolatey\lib\k6\tools..
C:\ProgramData\chocolatey\lib\k6\tools
ShimGen has successfully created a shim for k6.exe
The install of k6 was successful.
Software installed to 'C:\ProgramData\chocolatey\lib\k6\tools'

Chocolatey installed 1/1 packages.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

```

- Setelah install k6 berhasil maka ketikkan k6 di CMD agar memastikan bahwa k6 nya sudah berhasil terinstall. Jika sudah berhasil maka tampilannya akan tampil seperti gambar dibawah ini.

```
C:\Windows\System32>k6
```



Usage:

```
k6.exe [command]
```

Available Commands:

archive	Create an archive
cloud	Run a test on the cloud
completion	Generate the autocompletion script for the specified shell
help	Help about any command
inspect	Inspect a script or archive
login	Authenticate with a service
new	Create and initialize a new k6 script
pause	Pause a running test
resume	Resume a paused test
run	Start a test
scale	Scale a running test
stats	Show test metrics
status	Show test status
version	Show application version

- Langkah selanjutnya, buat folder untuk menyimpan file testing yang akan dilakukan nanti.

```
C:\Windows\System32>mkdir k6-test
```

```
C:\Windows\System32>cd k6-test
```

- Selanjutnya buka folder yang sudah dibuat di Visual Studio Code dengan mengetikkan perintah **code .** di CMD , maka folder akan terbuka otomatis di Visual Studio Code.

```
C:\Windows\System32\k6-test>code .
```

```
C:\Windows\System32\k6-test>
```

- Selanjutnya buat file js dengan nama 01-sample.js , lalu ketikan kode berikut di file yang sudah dibuat.

```
JS 01-sample.js X
JS 01-sample.js > ...
1  import http from 'k6/http';
2
3  export default function () {
4      http.get('http://test.k6.io');
5  }
6
```

8. Langkah selanjutnya, jalankan file yang sudah dibuat di CMD dengan mengetikkan **k6 run 01-sample.js** jika berhasil maka akan tampil seperti gambar dibawah ini.

```
C:\Windows\System32\k6-test>k6 run 01-sample.js

      /\_/\
     /  _  \
    /__  \__\
   /  _  \  \
  /__  \__\  \
 /  _  \  \  \
/_  _  \__\  \

execution: local
script: 01-sample.js
output: -

scenarios: (100.00%) 1 scenario, 1 max VUs, 10m30s max duration (incl. graceful stop):
 * default: 1 iterations for each of 1 VUs (maxDuration: 10m0s, gracefulStop: 30s)

data_received.....: 17 kB 4.8 kB/s
data_sent.....: 546 B 152 B/s
http_req_blocked.....: avg=606.51ms min=350.79ms med=606.51ms max=862.24ms p(90)=811.09ms p(95)=836.67ms
http_req_connecting.....: avg=247.57ms min=240.53ms med=247.57ms max=254.62ms p(90)=253.21ms p(95)=253.92ms
http_req_duration.....: avg=1.18s min=244.87ms med=1.18s max=2.12s p(90)=1.94s p(95)=2.03s
{ expected_response:true }...: avg=1.18s min=244.87ms med=1.18s max=2.12s p(90)=1.94s p(95)=2.03s
http_req_failed.....: 0.00% / 0 * 2
http_req_receiving.....: avg=497.85µs min=0s med=497.85µs max=995.7µs p(90)=896.13µs p(95)=945.91µs
http_req_sending.....: avg=0s min=0s med=0s max=0s p(90)=0s p(95)=0s
http_req_tls_handshaking.....: avg=310.85ms min=0s med=310.85ms max=621.71ms p(90)=559.53ms p(95)=590.62ms
http_req_waiting.....: avg=1.18s min=244.87ms med=1.18s max=2.12s p(90)=1.93s p(95)=2.03s
http_reqs.....: 2 0.55588/s
iteration_duration.....: avg=3.59s min=3.59s med=3.59s max=3.59s p(90)=3.59s p(95)=3.59s
iterations.....: 1 0.27794/s
vus.....: 1 min=1 max=1
vus_max.....: 1 min=1 max=1

running (00m03.6s), 0/1 VUs, 1 complete and 0 interrupted iterations
default / [=====] 1 VUs 00m03.6s/10m0s 1/1 iters, 1 per VU
```

9. Selanjutnya lakukan semua jenis testing non functional dengan kodingan yang ada di k6 documentation. Pertama yaitu **load testing** , buat kan file dan kodingan seperti dibawah ini , kodingan nya bisa langsung diambil di website k6.

```

load-testing.js X
load-testing.js > default
1  import http from 'k6/http';
2  import { sleep } from 'k6';
3
4  export const options = {
5    // Key configurations for avg load test in this section
6    stages: [
7      { duration: '5m', target: 100 }, // traffic ramp-up from 1 to 100 users over 5 minutes.
8      { duration: '30m', target: 100 }, // stay at 100 users for 30 minutes
9      { duration: '5m', target: 0 }, // ramp-down to 0 users
10   ],
11 };
12
13 export default () => {
14   const urlRes = http.get('https://test-api.k6.io');
15   sleep(30);
16   // MORE STEPS
17   // Here you can have more steps or complex script
18   // Step1
19   // Step2
20   // etc.
21 };

```

10. Selanjutnya jalan kan file **load_testing** untuk memastikan kodingan tidak ada yang error.

```

C:\Windows\System32\k6-test>k6 run load_testing.js

      A R K 6
     .io

execution: local
script: load_testing.js
output: -

scenarios: (100.00%) 1 scenario, 100 max VUs, 40m30s max duration (incl. graceful stop):
* default: Up to 100 looping VUs for 40m0s over 3 stages (gracefulRampDown: 30s, gracefulStop: 30s)

running (02m31.2s), 050/100 VUs, 2254 complete and 0 interrupted iterations
default  [=>-----] 050/100 VUs  02m31.2s/40m00.0s

```

11. Selanjutnya jenis testing yaitu **stress_testing** , ketikan kodingan seperti gambar di bawah ini.


```
stress-testing.js X
stress-testing.js > [?] options > [?] stages > [?] target
1  import http from 'k6/http';
2  import { sleep } from 'k6';
3
4  export const options = {
5    // Key configurations for avg load test in this section
6    executor: 'ramping-arrival-rate',
7    stages: [
8      { duration: '2h', target: 2000000 }, // traffic ramp-up from 1 to 100 users over 5 minutes.
9    ],
10 };
11
12 export default () => {
13   const urlRes = http.get('https://test-api.k6.io');
14   sleep(30);
15   // MORE STEPS
16   // Here you can have more steps or complex script
17   // Step1
18   // Step2
19   // etc.
20 };
```

12. Langkah selanjutnya , jalankan file **stress_testing** di cmd untuk memeriksa apakah kodingannya sudah benar atau belum.

```
C:\Windows\System32\k6-test>k6 run stress_testing.js

  M K6 .io

WARN[0000] There were unknown fields in the options exported in the script  error="json: unknown field \"executor\""
  execution: local
    script: stress_testing.js
    output: -

  scenarios: (100.00%) 1 scenario, 2000000 max VUs, 3h16m51.7s max duration (incl. graceful stop):
    * default: Up to 2000000 looping VUs for 2h0m0s over 1 stages (gracefulRampDown: 30s, gracefulStop: 30s)

C:\Windows\System32\k6-test>-----] 0281273/2000000 VUs initialized
default [-----]
```

13. Selanjutnya jenis testing yaitu **soak_testing** , buat kan kodingan seperti gambar dibawah ini.


```

1  import http from 'k6/http';
2  import { sleep } from 'k6';
3
4  export const options = {
5    // Key configurations for avg load test in this section
6    stages: [
7      { duration: '5m', target: 100 }, // traffic ramp-up from 1 to 100 users over 5 minutes.
8      { duration: '10h', target: 100 }, // stay at 100 users for 30 minutes
9      { duration: '5m', target: 0 }, // ramp-down to 0 users
10   ],
11 };
12
13 export default () => {
14   const urlRes = http.get('https://test-api.k6.io');
15   sleep(30);
16   // MORE STEPS
17   // Here you can have more steps or complex script
18   // Step1
19   // Step2
20   // etc.
21 };

```

14. Jalankan file **soak_testing** di cmd untuk memeriksa apakah kodingannya sudah benar atau belum.

```

C:\Windows\System32\k6-test>k6 run soak_testing.js

      /\_/\
     /  _  \
    /_____\ \
   /         \
  /           \
 /             \
/               \
\               /
 \             /
  \           /
   \         /
    \_____/

execution: local
script: soak_testing.js
output: -

scenarios: (100.00%) 1 scenario, 100 max VUs, 10h10m30s max duration (incl. graceful stop):
 * default: Up to 100 looping VUs for 10h10m0s over 3 stages (gracefulRampDown: 30s, gracefulStop: 30s)

running (00h00m06.1s), 003/100 VUs, 0 complete and 0 interrupted iterations
default  [-----] 003/100 VUs  00h00m06.1s/10h10m00.0s

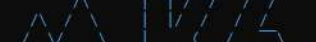
```

15. Jenis testing terakhir adalah **spike_testing** , ketikkan kodingan deperti gambar berikut ini di filenya.

```
1  import http from 'k6/http';
2  import { sleep } from 'k6';
3
4  export const options = {
5    // Key configurations for avg load test in this section
6    stages: [
7      { duration: '1m', target: 200 }, // traffic ramp-up from 1 to 100 users over 5 minutes.
8      { duration: '5', target: 0 }, // ramp-down to 0 users
9    ],
10 };
11
12 export default () => {
13   const urlRes = http.get('https://test-api.k6.io');
14   sleep(30);
15   // MORE STEPS
16   // Here you can have more steps or complex script
17   // Step1
18   // Step2
19   // etc.
20 };
```

16. Jalankan file **spike_testing** di cmd untuk memeriksa apakah kodingannya sudah benar atau belum.

```
C:\Windows\System32\k6-test>k6 run spike_testing.js
```



```
.io

execution: local
script: spike_testing.js
output: -

scenarios: (100.00%) 1 scenario, 200 max VUs, 1m30s max duration (incl. graceful stop):
    * default: Up to 200 looping VUs for 1m0.005s over 2 stages (gracefulRampDown: 30s, gracefulStop: 30s)

running (0m06.3s), 021/200 VUs, 0 complete and 0 interrupted iterations
default [==>-----] 021/200 VUs  0m06.3s/1m00.0s
```

C. Kesimpulan

Dengan mengimplementasikan performance testing secara efektif, organisasi dapat memastikan bahwa aplikasi mereka mampu menangani beban kerja yang diharapkan dan memberikan pengalaman pengguna yang optimal tanpa penurunan kinerja yang signifikan. Penggunaan alat seperti **Apache JMeter** dan **K6** mempermudah pengujian dan membantu dalam mendeteksi potensi masalah sebelum terjadi pada lingkungan produksi.