



# NodeJS Dasar

Eko Kurniawan Khannedy

# Eko Kurniawan Khannedy

- Technical architect at one of the biggest ecommerce company in Indonesia
- 12+ years experiences
- [www.programmerzamannow.com](http://www.programmerzamannow.com)
- [youtube.com/c/ProgrammerZamanNow](https://youtube.com/c/ProgrammerZamanNow)





# Eko Kurniawan Khannedy

- Telegram : [@khannedy](https://t.me/khannedy)
- Facebook : [fb.com/ProgrammerZamanNow](https://fb.com/ProgrammerZamanNow)
- Instagram : [instagram.com/programmerzamannow](https://instagram.com/programmerzamannow)
- Youtube : [youtube.com/c/ProgrammerZamanNow](https://youtube.com/c/ProgrammerZamanNow)
- Telegram Channel : [t.me/ProgrammerZamanNow](https://t.me/ProgrammerZamanNow)
- Tiktok : <https://tiktok.com/@programmerzamannow>
- Email : [echo.khannedy@gmail.com](mailto:echo.khannedy@gmail.com)



## Sebelum Belajar

- Sudah menyelesaikan Kelas Roadmap JavaScript dari Programmer Zaman Now



# Agenda

- Pengenalan NodeJS
- Pengenalan Concurrency
- NodeJS Architecture
- Menginstall NodeJS
- NodeJS REPL
- Standard Library
- Dan lain-lain

---

# Pengenalan NodeJS



# Pengenalan NodeJS

- NodeJS diperkenalkan pertama kali oleh Ryan Dahl pada tahun 2009
- NodeJS merupakan teknologi yang bisa digunakan untuk menjalankan kode JavaScript diluar Web Browser
- NodeJS dibuat dari V8 Engine, yaitu Engine untuk Google Chrome
- NodeJS merupakan project yang Free dan OpenSource
- <https://nodejs.org/>



# Kenapa Belajar NodeJS

- NodeJS mempopulerkan paradigma JavaScript Everywhere, dimana dengan menggunakan NodeJS, kita bisa membuat aplikasi berbasis server side dengan bahasa pemrograman JavaScript
- Hal ini membuat kita hanya butuh belajar bahasa pemrograman JavaScript untuk membuat aplikasi web misalnya, sehingga tidak butuh belajar bahasa pemrograman lain seperti PHP atau Java untuk server side web nya
- Saat ini NodeJS sangat populer dan banyak sekali digunakan di perusahaan teknologi, terutama untuk membantu pengembangan Web Frontend





## Yang Tidak Bisa Dilakukan di NodeJS

- Pada kelas JavaScript, kita sudah membahas banyak sekali fitur JavaScript yang berjalan di Browser
- Karena NodeJS tidak berjalan di Browser, jadi tidak semua fitur JavaScript bisa dilakukan di NodeJS
- Fitur seperti Document Object Model dan banyak Web API tidak bisa dilakukan di NodeJS, hal ini karena DOM dan beberapa Web API berjalan membutuhkan Browser



# Text Editor

- NodeJS menggunakan bahasa pemrograman JavaScript, oleh karena itu kita bisa menggunakan Text Editor apapun untuk membuat aplikasi menggunakan NodeJS, misal :
- Visual Studio Code
- JetBrains WebStorm
- Sublime
- Atom
- NodePad++
- Dan lain-lain

---

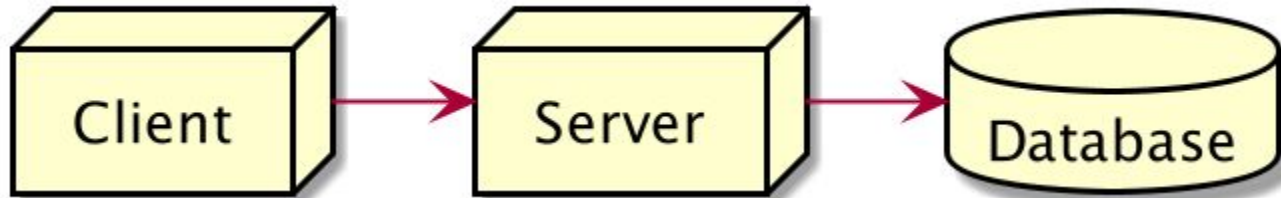
# Web Application



# Web Application

- Web Application adalah aplikasi yang berjalan di Server dan ditampilkan di Browser Client
- Saat kita membuat Web Application, biasanya akan dibagi menjadi 3 bagian, Client, Server dan Database

## Diagram Web Application





# Client

- Client merupakan user interface atau bagian frontend dari web application, yang digunakan oleh pengguna web application
- Client digunakan untuk berinteraksi dengan Server, baik itu mengirim data atau menerima data
- Frontend biasanya dibuat menggunakan HTML, CSS dan JavaScript



# Server

- Server bertanggung jawab untuk menerima request dari Client, mengerjakan request yang dikirim dan membalas request berupa response ke Client
- Server bertugas sebagai backend untuk web application, dimana semua logic aplikasi akan dilakukan di Server
- Biasanya Server dibuat menggunakan PHP, Python, Java, .NET dan banyak bahasa pemrograman lainnya
- Dengan adanya NodeJS, sekarang kita bisa membuat Server menggunakan JavaScript



# Database

- Database adalah tempat untuk menyimpan data web application
- Data disimpan dan diambil oleh Server.
- Client tidak bisa langsung mengambil atau menyimpan data ke Database secara langsung, oleh karena itu perlu penengah untuk melakukannya, yaitu Server
- Database biasanya menggunakan aplikasi sistem basis data seperti misalnya MySQL, PostgreSQL, MongoDB dan lain-lain



---

# Concurrency dan Parallel



# Sejarah

- Dahulu, komputer hanya menjalankan sebuah program pada satu waktu
- Karena hanya bisa menjalankan satu program pada satu waktu, hal ini tidak efisien dan memakan waktu lama karena hanya bisa mengerjakan satu tugas pada satu waktu
- Semakin kesini, sistem operasi untuk komputer semakin berkembang, sekarang sistem operasi bisa menjalankan program secara bersamaan dalam proses yang berbeda-beda, terisolasi dan saling independen antar program



# Thread

- Program biasanya berjalan dalam sebuah proses, dan proses akan memiliki resource yang independen dengan proses lain
- Sekarang, sistem operasi tidak hanya bisa menjalankan multiple proses, namun dalam proses kita bisa menjalankan banyak pekerjaan sekaligus, atau bisa dibilang proses ringan atau lebih dikenal dengan nama Thread
- Thread membuat proses aplikasi bisa berjalan tidak harus selalu sequential, kita bisa membuat proses aplikasi berjalan menjadi asynchronous atau parallel

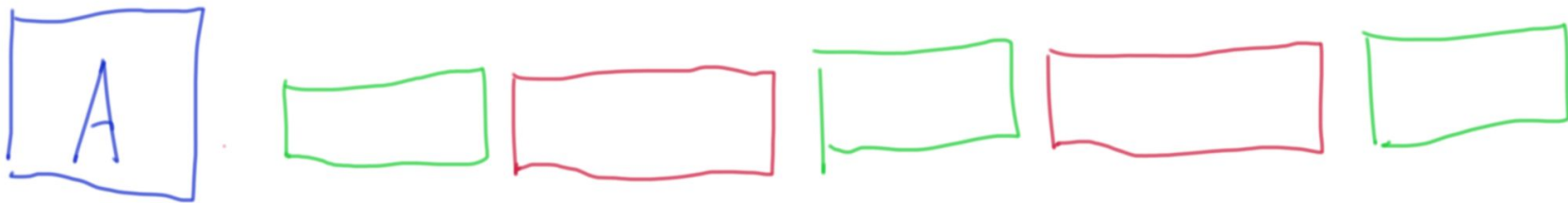


# Concurrency vs Parallel

- Kadang banyak yang bingung dengan concurrency dan parallel, sebenarnya kita tidak perlu terlalu memusingkan hal ini
- Karena saat ini, kita pasti akan menggunakan keduanya ketika membuat aplikasi
- Concurrency artinya mengerjakan beberapa pekerjaan satu persatu
- Parallel artinya mengerjakan beberapa pekerjaan sekaligus pada satu waktu



## Diagram Concurrency





## Diagram Parallel





## Contoh Concurrency dan Parallel

- Browser adalah aplikasi yang concurrent dan parallel
- Browser akan melakukan proses concurrent ketika membuka web, browser akan melakukan http request, lalu mendownload semua file web (html, css, js) lalu merender dalam bentuk tampilan web
- Browser akan melakukan proses parallel, ketika kita membuka beberapa tab web, dan juga sambil download beberapa file, dan menonton video dari web streaming

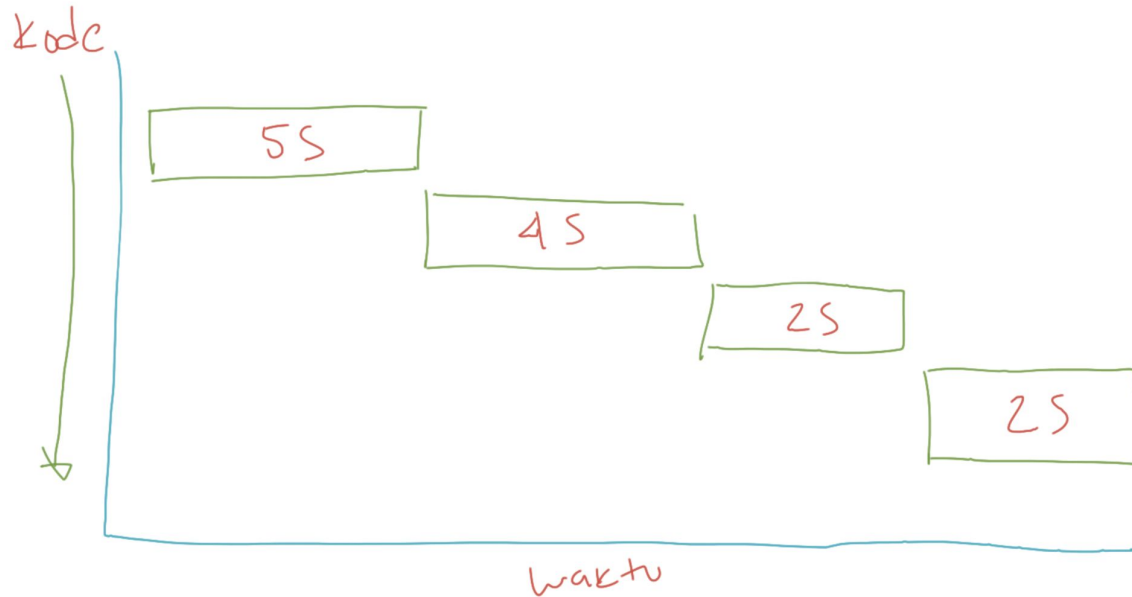


# Synchronous vs Asynchronous

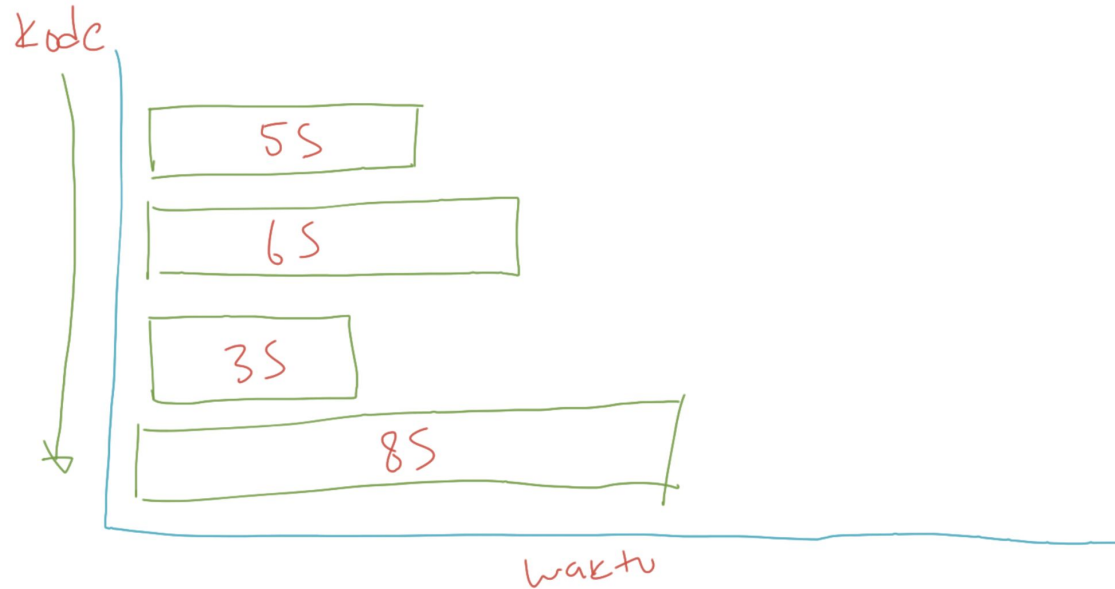
- Saat membuat aplikasi yang concurrent atau parallel, kadang kita sering menemui istilah synchronous dan asynchronous
- Tidak perlu bingung dengan istilah tersebut, secara sederhana
- Synchronous adalah ketika kode program kita berjalan secara sequential, dan semua tahapan ditunggu sampai prosesnya selesai baru akan dieksekusi ke tahapan selanjutnya
- Sedangkan, Asynchronous artinya ketika kode program kita berjalan dan kita tidak perlu menunggu eksekusi kode tersebut selesai, kita bisa lanjutkan ke tahapan kode program selanjutnya



## Diagram Synchronous



## Diagram Asynchronous



---

# Threadpool Web Model

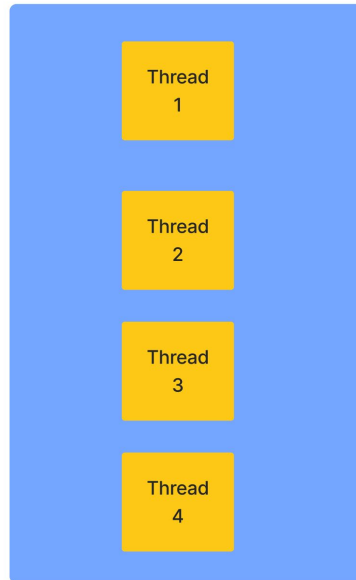


# Threadpool

- Pada materi sebelumnya sudah dijelaskan bahwa thread adalah proses ringan yang biasa dibuat saat membuat aplikasi
- Walaupun bisa dibilang ringan, namun jika terlalu banyak membuat thread, maka tetap akan memberatkan sistem operasi kita
- Oleh karena itu, biasanya kita akan menggunakan threadpool untuk melakukan management thread
- Threadpool merupakan tempat dimana kita menyimpan thread, ketika kita butuh kita akan ambil dari threadpool, ketika sudah selesai, kita akan kembalikan thread nya ke threadpool
- Dengan threadpool, kita bisa memanfaatkan thread yang sama berkali-kali, tanpa harus membuat thread baru terus menerus



# Diagram Threadpool

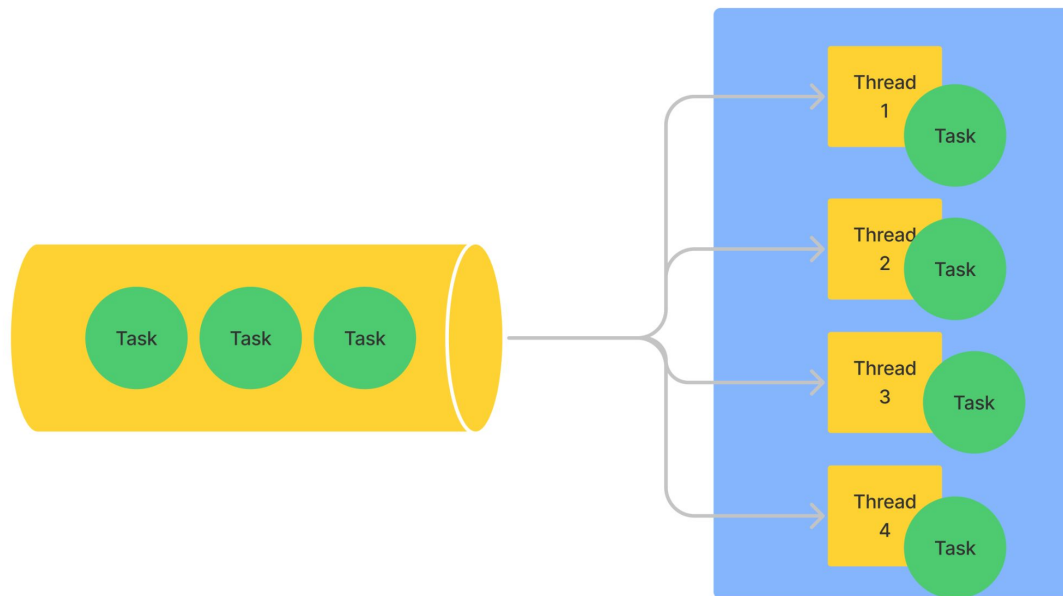




# Threadpool Queue

- Apa yang terjadi ketika semua thread sedang bekerja? Bagaimana jika kita ingin meminta thread ke threadpool untuk mengerjakan sesuatu?
- Jika semua thread penuh, kita tidak bisa meminta lagi thread ke threadpool. Kita harus menunggu sampai ada thread yang tidak sibuk
- Dimana kita harus menunggu sampai ada thread tersedia untuk digunakan?
- Biasanya threadpool memiliki tempat untuk menyimpan tugas yang belum dikerjakan oleh thread di tempat bernama queue (antrian)
- Ketika kita mengirim perintah ke threadpool, perintah tersebut akan dikirim ke queue, lalu perintah-perintah itu akan dieksekusi satu per satu oleh thread yang tersedia di threadpool

# Diagram Threadpool Queue





# Threadpool Web Model

- Dahulu pembuatan web application sangat populer menggunakan threadpool model
- Setiap request yang masuk ke web server akan diproses oleh satu buah thread
- Dengan demikian ketika banyak request masuk, semua bisa diproses secara paralel karena akan ditangani oleh thread masing-masing
- Namun threadpool model ini memiliki kekurangan, ketika thread sedang sibuk semua, secara otomatis request selanjutnya harus menunggu sampai ada thread yang selesai melakukan tugas sebelumnya
- Contoh web server yang menggunakan threadpool model, seperti Apache HTTPD, Apache Tomcat, dan lain-lain



---

# Blocking dan Non-Blocking



# Blocking

- Saat kita membuat kode program, secara default kode program akan berjalan secara blocking atau synchronous
- Artinya kita harus menunggu sebuah kode selesai sebelum kode selanjutnya dieksekusi
- Contoh ketika kita membuat kode program untuk membaca file, jika kode kita blocking, maka kita harus menunggu program selesai membaca file, baru kita bisa melanjutkan kode program selanjutnya



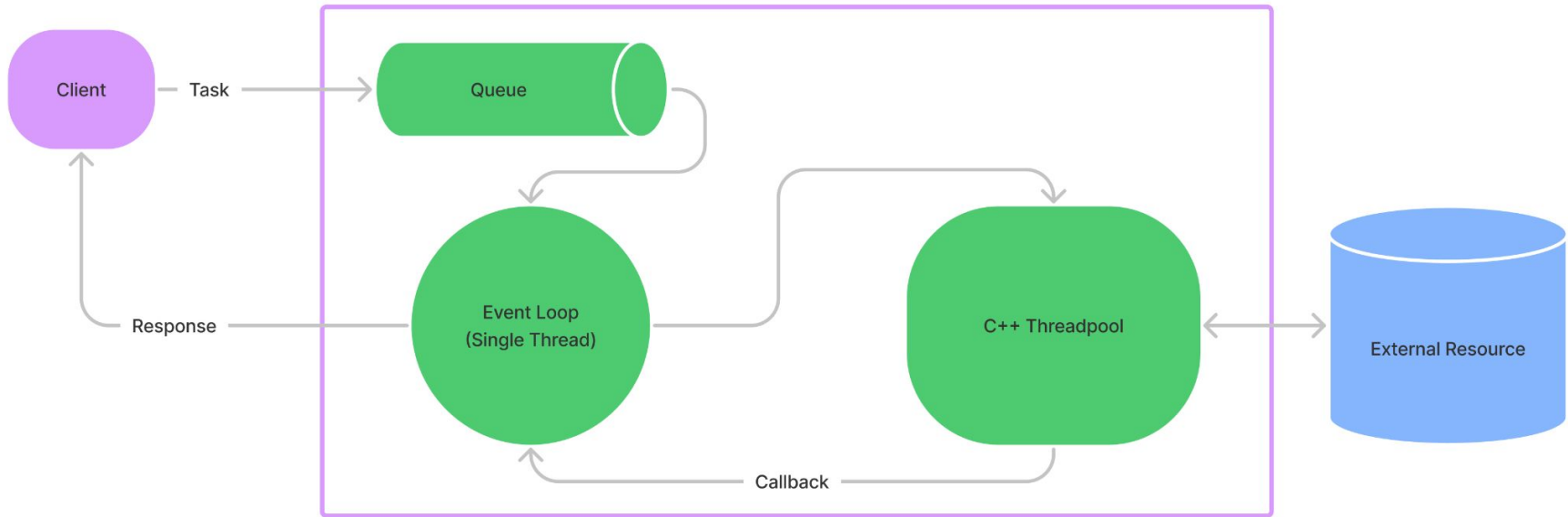
# Non-Blocking

- Non-Blocking berbeda dengan Blocking, kode program Non-Blocking akan dieksekusi tanpa harus menunggu kode program tersebut selesai
- Non-Blocking akan dijalankan secara asynchronous
- Ketika memanggil kode Non-Blocking, biasanya kita perlu mengirimkan callback untuk dipanggil oleh kode Non-Blocking tersebut ketika kodenya sudah selesai
- Contoh-contoh Non-Blocking sudah kita bahas di kelas JavaScript Async, seperti AJAX, Fetch API, dan lain-lain
- Di NodeJS, hampir semua fiturnya mendukung kode Non-Blocking

---

# NodeJS Architecture

# NodeJS Architecture





# Event-Loop

- Event-Loop merupakan single thread proses yang digunakan untuk mengeksekusi kode Non-Blocking
- Karena Event-Loop hanya menggunakan single thread, maka kita harus berhati-hati ketika membuat blocking code, karena bisa memperlambat proses eksekusi kode kita
- Event-Loop sendiri sebenarnya tugasnya hanya menerima dan mengirim eksekusi kode ke C++ Threadpool, oleh karena itu selalu usahakan menggunakan kode nonblocking agar proses blocking-nya dikerjakan di C++ threadpool
- Event-Loop akan menerima response dari C++ threadpool yang di kirim via callback



# C++ Threadpool

- NodeJS Menggunakan C++ Threadpool untuk workernya, yaitu threadpool untuk melakukan pekerjaan
- Libuv adalah library yang digunakan di NodeJS, dimana secara default libuv menggunakan 4 thread di dalam threadpool nya, hal ini menjadikan kita bisa melakukan 4 pekerjaan blocking sekaligus dalam satu waktu.
- Jika terlalu banyak pekerjaan blocking, kita bisa mengubah jumlah thread di libuv dengan pengaturan environment variable `UV_THREADPOOL_SIZE`
- <http://docs.libuv.org/en/v1.x/threadpool.html>

---

# Menginstall NodeJS





# Menginstall NodeJS Manual

- Download versi NodeJS LTS (Long Term Support)
- <https://nodejs.org/en/download/>



# Menginstall NodeJS dengan Package Manager

- <https://github.com/nvm-sh/nvm>
- <https://community.chocolatey.org/packages/nodejs>
- <https://formulae.brew.sh/formula/node>



## Setting PATH NodeJS

- Setelah menginstall NodeJS, disarankan melakukan setting PATH NodeJS pada sistem operasi kita
- Hal ini agar mudah ketika kita mengakses program NodeJS menggunakan terminal / command prompt



## Kode : Mengecek NodeJS

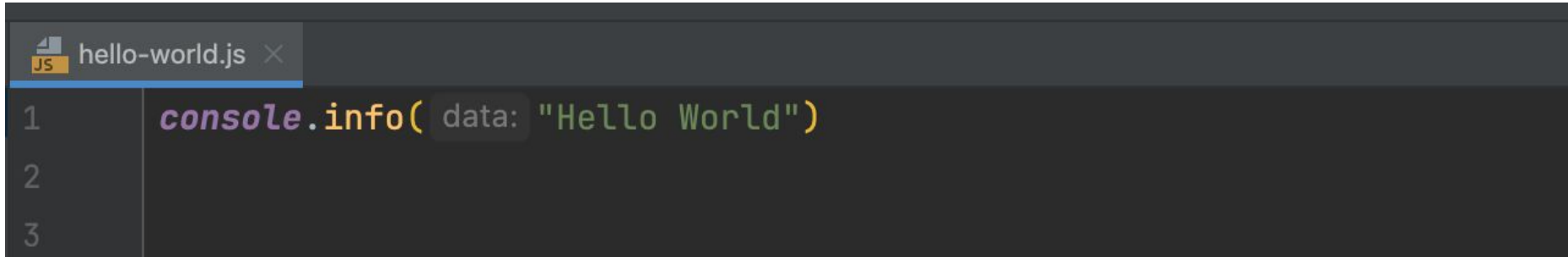
```
→ ~ node --version  
v16.13.0  
→ ~ █
```

---

# Hello World



## Kode : Hello World



```
hello-world.js x
1 console.info(data: "Hello World")
2
3
```



# Menjalankan Kode JavaScript

- Karena NodeJS tidak memerlukan Web Browser, jadi kita bisa langsung menjalankan program JavaScript kita menggunakan aplikasi NodeJS lewat terminal / command prompt, dengan perintah :
- `node namafile.js`



## Kode : Menjalankan Hello World

```
→ belajar-nodejs-dasar node hello-world.js
```

```
Hello World
```

```
→ belajar-nodejs-dasar
```



---

# NodeJS REPL



## REPL (Read Eval Print Loop)

- REPL singkatan dari Read Eval Print Loop
- Yaitu mekanisme dimana program bisa membaca langsung kode program yang diketikkan, lalu mengeksekusinya, menampilkan hasilnya, lalu mengulangi dari awal lagi
- NodeJS mendukung REPL, sehingga lebih mudah ketika belajar
- Namun tetap, saya menyarankan menyimpan kode program di file JavaScript, agar lebih mudah diubah ketika terjadi masalah
- Untuk menggunakan NodeJS REPL, cukup jalankan aplikasi node saja



## Kode : REPL

```
→ ~ node
```

```
Welcome to Node.js v16.13.0.
```

```
Type ".help" for more information.
```

```
> .help
```

```
.break    Sometimes you get stuck, this gets you out
```

```
.clear    Alias for .break
```

```
.editor   Enter editor mode
```

```
.exit     Exit the REPL
```

```
.help     Print this help message
```

```
.load     Load JS from a file into the REPL session
```

```
.save     Save all evaluated commands in this REPL session to a file
```

```
Press Ctrl+C to abort current expression, Ctrl+D to exit the REPL
```

```
> █
```

---

# NodeJS Standard Library



# NodeJS Standard Library

- Saat kita belajar JavaScript, di Web Browser, terdapat fitur-fitur yang bernama Web API
- <https://developer.mozilla.org/en-US/docs/Web/API>
- Kebanyakan fitur Web API hanya berjalan di Web Browser, sehingga tidak bisa jalan di NodeJS
- NodeJS sendiri hanya menggunakan bahasa pemrograman JavaScript nya, namun tidak mengadopsi fitur Web API nya, karena itu hanya berjalan di Web Browser
- NodeJS sendiri memiliki standard library yang bisa kita gunakan untuk mempermudah pembuatan aplikasi
- <https://nodejs.org/dist/latest-v16.x/docs/api/>



# Modules



# Modules

- Standard Library yang terdapat di NodeJS bisa kita gunakan seperti layaknya JavaScript Modules
- Jika belum mengerti tentang JavaScript Modules, silahkan pelajari kelas saya tentang JavaScript Modules
- Karena NodeJS menggunakan Modules, jika kita ingin menggunakan Modules, kita juga perlu memberi tahu bahwa file JavaScript kita menggunakan Modules, caranya dengan mengubah nama file dari .js menjadi .mjs



## Kode : Contoh Standard Library

```
standard-library.mjs x
1 import os from "os"
2
3 console.info(os.platform());
4 console.table(os.cpus());
5
```



---

# Require Function



# Require Function

- Awal ketika NodeJS rilis, fitur JavaScript Modules belum rilis, namun sekarang JavaScript sudah banyak menggunakan JavaScript Modules
- NodeJS pun awalnya tidak menggunakan JavaScript Modules, namun sekarang NodeJS sudah bisa menggunakan JavaScript Modules, dan sangat direkomendasikan menggunakannya
- Namun awal sebelum Modules, NodeJS menggunakan function `require()` untuk melakukan import file
- Di materi ini saya sengaja bahas, agar tidak bingung ketika kita melihat tutorial yang masih menggunakan function `require`



## Kode : Function Require

```
require-function.js ×  
1  const os = require("os");  
2  
3  console.info(os.platform());  
4  console.table(os.cpus());  
5
```

---

# Global Async di Module



# Global Async

- Saat kita belajar JavaScript, untuk menggunakan Async Await, biasanya kita perlu membuat terlebih dahulu function yang kita tandai sebagai async
- Saat kita menggunakan Module, secara default, global code adalah Async, oleh karena itu kita bisa menggunakan Async Await
- Kecuali jika kita membuat function, maka function tersebut harus kita tandai sebagai Async jika ingin menggunakan Async Await



## Kode : JavaScript

```
async.js x
1 function samplePromise() {
2   return Promise.resolve( value: "Eko");
3 }
4
5 const data = await samplePromise(); // error
6 console.info(data);
7
```



## Kode : JavaScript Module

```
async.mjs x
1 function samplePromise() {
2     return Promise.resolve( value: "Eko");
3 }
4
5 const data = await samplePromise();
6 console.info(data);
7
```

---

OS





# OS

- OS merupakan standard library yang bisa digunakan untuk mendapatkan informasi tentang sistem operasi yang digunakan
- <https://nodejs.org/dist/latest-v16.x/docs/api/os.html>



## Kode : OS

```
JS os.mjs x
1  import os from "os";
2
3  console.info(os.platform());
4  console.info(os.arch());
5  console.table(os.cpus());
6  console.info(os.uptime());
7  console.info(os.totalmem());
8  console.info(os.freemem());
9  console.table(os.networkInterfaces());
10
```

—

Path



# Path

- Path merupakan standard library yang bisa kita gunakan untuk bekerja dengan lokasi file dan directory / folder
- <https://nodejs.org/dist/latest-v16.x/docs/api/path.html>

## Kode : Path

```
path.mjs x
1  import path from "path";
2
3  const file = "/Users/khannedy/contoh.html";
4
5  console.info(path.sep);
6  console.info(path.dirname(file));
7  console.info(path.basename(file));
8  console.info(path.extname(file));
9  console.info(path.parse(file));
10
```



# File System



# File System

- File System merupakan standard library yang bisa digunakan untuk memanipulasi file system
- Dalam File System, terdapat 3 jenis library
- Pertama library yang bersifat blocking atau synchronous
- Kedua library yang bersifat non-blocking atau asynchronous menggunakan callback
- Ketika library yang bersifat non-blocking atau asynchronous tapi menggunakan promise
- <https://nodejs.org/dist/latest-v16.x/docs/api/fs.html>



## Kode : File System

```
file-system.mjs x
1  import fs from "fs";
2
3  const buffer = fs.readFileSync(path: "file-system.mjs");
4
5  console.info(buffer.toString());
6
7  fs.writeFileSync(file: "temp.txt", data: "Hello World");
8
```



---

# Debugger



# Debugger

- NodeJS memiliki fitur debugger, dimana kita bisa mengikuti tahapan eksekusi program di NodeJS
- Hal ini sangat cocok ketika kita melakukan proses debugging, mencari sebab masalah yang terjadi di aplikasi kita
- <https://nodejs.org/dist/latest-v16.x/docs/api/debugger.html>



# Breakpoint

- Dalam debugging, terdapat istilah breakpoint, yaitu lokasi dimana kita ingin menghentikan sementara eksekusi kode program
- Biasanya ini dilakukan untuk mengawasi data-data di sekitar lokasi berhentinya tersebut
- Untuk menambahkan breakpoint, kita bisa menggunakan kata kunci: debugger



## Menjalankan Mode Debug

- Jika kita menjalankan file JavaScript hanya dengan menggunakan perintah `node namafile.js`, maka secara default dia tidak akan jalan dalam mode debug
- Agar jalan dalam mode debug, kita harus menambahkan perintah `inspect` :  
`node inspect namafile.js`



# Perintah Debugger

Saat masuk ke mode debug, ada beberapa perintah yang bisa kita gunakan dalam melakukan debugging

- cont, c: Continue execution
- next, n: Step next
- step, s: Step in
- out, o: Step out
- pause: Pause running code



## Kode : Debugger



```
debugger.mjs x
1  function sayHello(name) {
2      debugger;
3      return `Hello ${name}`;
4  }
5
6  const firstName = "Eko";
7
8  console.info(sayHello(firstName));
9
```

---

**DNS**



# DNS

- DNS merupakan standard library yang bisa digunakan untuk bekerja dengan DNS (domain name server)
- <https://nodejs.org/dist/latest-v16.x/docs/api/dns.html>



## Kode : DNS

```
dns.mjs x
1  import dns from "dns";
2
3  function callback(error, ip) {
4      console.info(ip);
5  }
6
7  dns.lookup( hostname: "www.programmerzamannow.com", callback)
8
```

## Kode : DNS Promise

```
dns.mjs x
1 import dns from "dns/promises";
2
3 const lookup = await dns.lookup( hostname: "www.programmerzamannow.com");
4
5 console.info(lookup.family);
6 console.info(lookup.address);
7
```



# Events



# Events

- Events adalah standard library di NodeJS yang bisa digunakan sebagai implementasi Event Listener
- Di dalam Events, terdapat sebuah class bernama EventEmitter yang bisa digunakan untuk menampung data listener per jenis event.
- Lalu kita bisa melakukan emit untuk mentrigger jenis event dan mengirim data ke event tersebut
- <https://nodejs.org/dist/latest-v16.x/docs/api/events.html>

## Kode : Events

```
events.mjs x
1  import {EventEmitter} from "events"
2
3  const emitter = new EventEmitter();
4  emitter.addListener( eventName: "hello", listener: (name) => {
5      console.info( data: `Hello ${name}` );
6  });
7  emitter.addListener( eventName: "hello", listener: (name) => {
8      console.info( data: `HaLo ${name}` );
9  });
10
11 emitter.emit( eventName: "hello", args: "Eko");
12
```

---

# Globals



# Globals

- Di dalam NodeJS, terdapat library berupa variable atau function yang secara global bisa diakses dimana saja, tanpa harus melakukan import
- Kita bisa melihat detail apa saja fitur yang terdapat secara global di halaman dokumentasinya
- <https://nodejs.org/dist/latest-v16.x/docs/api/globals.html>



## Kode : Globals

```
globals.mjs x
1 setTimeout( handler: function () {
2     console.info( data: "Hello World");
3 }, timeout: 3000);
4 |
```





# Process



# Process

- Process merupakan standard library yang digunakan untuk mendapatkan informasi proses NodeJS yang sedang berjalan
- Process juga merupakan instance dari EventEmitter, sehingga kita bisa menambahkan listener kedalam Process
- <https://nodejs.org/dist/latest-v16.x/docs/api/process.html>



# Kode : Process

```
process.mjs x
1  import process from "process";
2
3  process.addListener( event: "exit", listener: function (exitCode : number ){
4      console.info( data: `NodeJS exit with code ${exitCode}`);
5  });
6
7  console.info(process.version);
8  console.table(process.argv);
9  console.table(process.report);
10 console.table(process.env)
11
12 process.exit( code: 1);
13
14 console.info( data: "Not Printed because already exit");
15
```



# Readline



# Readline

- Readline merupakan standard library yang digunakan untuk membaca input
- Namun pada saat dibuat video ini, Readline hanya mendukung versi callback di versi NodeJS LTS 16.
- Di NodeJS 17 sudah mendukung Promise sehingga lebih mudah digunakan, namun itupun masih dalam tahap experimental
- <https://nodejs.org/dist/latest-v16.x/docs/api/readline.html>



## Kode : Readline

```
readline.mjs x
1 import process from "process";
2 import readline from "readline";
3
4 const input = readline.createInterface( options: {
5     input: process.stdin,
6     output: process.stdout
7 });
8
9 input.question( query: "Siapa nama anda? : ", callback: function (nama : string ) {
10     console.info( data: `Hello ${nama}`);
11     input.close();
12 });
13
```

---

# Report



# Report

- Report merupakan fitur yang terdapat di NodeJS untuk membuat laporan secara otomatis dalam file ketika sesuatu terjadi pada aplikasi NodeJS kita
- <https://nodejs.org/dist/latest-v16.x/docs/api/report.html>



## Kode : Error pada Aplikasi NodeJS

```
report.mjs x
1  import process from "process";
2
3  process.report.reportOnFatalError = true;
4  process.report.reportOnUncaughtException = true;
5  process.report.reportOnSignal = true;
6  process.report.filename = "report.json";
7
8  function sampleError() {
9      throw new Error("Ups");
10 }
11
12 sampleError();
13
```

---

# Buffer



# Buffer

- Buffer merupakan object yang berisikan urutan byte dengan panjang tetap.
- Buffer merupakan turunan dari tipe data Uint8Array
- <https://nodejs.org/dist/latest-v16.x/docs/api/buffer.html>



## Kode : Buffer

JS buffer.mjs

```
1
2  const buffer = Buffer.from( arrayBuffer: "Eko");
3  console.info(buffer.toString());
4
5  buffer.reverse();
6  console.info(buffer.toString());
7
8
```



# Buffer Encoding

- Buffer juga bisa digunakan untuk melakukan encoding dari satu encoding ke encoding yang lain
- Ada banyak encoding yang didukung oleh Buffer, misal utf8, ascii, hex, base64, base64url dan lain-lain



## Kode : Buffer Encoding

buffer-encoding.mjs

```
1
2  const buffer = Buffer.from( str: "Eko Kurniawan Khannedy", encoding: "utf8");
3
4  console.info(buffer.toString( encoding: "base64"));
5  console.info(buffer.toString( encoding: "hex"));
6
7  const buffer2 = Buffer.from( str: "RWtvIEt1cm5pYXdhbmlBLaGFubmVkeQ==", encoding: "base64url");
8
9  console.info(buffer2.toString( encoding: "utf8"));
10
```

---

# Stream



# Stream

- Stream adalah standard library untuk kontrak aliran data di NodeJS
- Ada banyak sekali Stream object di NodeJS
- Stream bisa jadi object yang bisa dibaca, atau bisa di tulis, dan Stream adalah turunan dari EventEmitter
- <https://nodejs.org/dist/latest-v16.x/docs/api/stream.html>





## Kode : Stream

```
stream.mjs x
1  import fs from "fs";
2
3  const writer = fs.createWriteStream( path: "target.log");
4  writer.write( chunk: "Eko");
5  writer.write( chunk: "Kurniawan");
6  writer.write( chunk: "Khannedy");
7  writer.close();
8
9  const reader = fs.createReadStream( path: "target.log");
10 reader.on( event: "data", listener: function (data : Buffer | string ) {
11     console.info(data.toString());
12     reader.close();
13 })
14
15
```

—

Timer



# Timer

- Timer merupakan standard library untuk melakukan scheduling
- Function di Timer terdapat di globals, sehingga kita bisa menggunakannya tanpa melakukan import, namun semua function Timer menggunakan Callback
- Jika kita ingin menggunakan Timer versi Promise, kita bisa meng-import dari module timer/promise
- <https://nodejs.org/dist/latest-v16.x/docs/api/timers.html>



## Kode : Timer

```
timer.mjs x
1  setInterval( handler: () => {
2      console.info( data: `Timer at ${new Date()}` );
3  }, timeout: 1000);
4
5
6
```



## Kode : Timer Promise

```
timer-promise.mjs x
1 import timers from "timers/promises"
2
3 for await (const startTime of timers.setInterval( delay: 1000, new Date())) {
4     console.info( data: `Start Timer at ${startTime}`);
5 }
6
```

—  
Net



# Net

- Net merupakan standard library yang bisa digunakan untuk membuat network client dan server berbasis TCP
- Net Server dan Client merupakan object Stream, sehingga kita bisa baca datanya, tulis datanya dan juga menambahkan listener
- <https://nodejs.org/dist/latest-v16.x/docs/api/net.html>

## Kode : Net Server

```
net-server.mjs x
1  import net from "net";
2
3  const server = net.createServer( connectionListener: function (client : Socket ) {
4      console.info( data: "Client connected");
5      client.on( event: "data", listener: function (data : Buffer ) {
6          console.info( data: `Receive data from client : ${data.toString()}`);
7          client.write( buffer: `Hello ${data.toString()}\r\n`);
8      })
9  });
10
11  server.listen( port: 3000, hostname: "localhost");
12
```



## Kode : Net Client

```
net-client.mjs x
1  import net from "net";
2
3  const connection = net.createConnection( options: {port: 3000, host: "localhost"});
4
5  setInterval( handler: function () {
6      connection.write( buffer: "Eko\r\n");
7  }, timeout: 2000);
8
9  connection.addListener( event: "data", listener: function (data : Buffer ) {
10      console.info( data: `Receive data from server : ${data.toString()}`);
11  })
12
```

---

URL



# URL

- URL merupakan standard library untuk bekerja dengan URL
- <https://nodejs.org/dist/latest-v16.x/docs/api/url.html>



## Kode : URL

```
url.mjs
1 import {URL} from "url";
2
3 const pzn = new URL( input: "https://www.programmerzamannow.com/belajar?kelas=nodejs");
4
5 console.info(pzn.toString());
6 console.info(pzn.protocol);
7 console.info(pzn.host);
8 console.info(pzn.pathname);
9 console.info(pzn.searchParams);
10
```

## Kode : Mengubah URL

```
url.mjs x
1 import {URL} from "url";
2
3 const pzn = new URL(input: "https://www.programmerzamannow.com/belajar?kelas=nodejs");
4
5 pzn.host = "www.khannedy.com";
6 const searchParam = pzn.searchParams;
7 searchParam.append(name: "status", value: "premium");
8
9 console.info(pzn.toString());
10 |
```

---

Util



# Util

- Util adalah standard library yang berisikan utility-utility yang bisa kita gunakan untuk mempermudah pembuatan kode program di NodeJS
- <https://nodejs.org/dist/latest-v16.x/docs/api/util.html>



## Kode : Util

```
util.mjs x
1  import util from "util";
2
3  console.info(util.format( format: "Nama : %s", param: "Eko"));
4
5  const person = {firstName: "Eko", lastName: "Khannedy"};
6  console.info(util.format( format: "Person : %j", person));
7
```



---

**Zlib**



# Zlib

- Zlib adalah standard library yang digunakan untuk melakukan kompresi menggunakan Gzip
- <https://nodejs.org/dist/latest-v16.x/docs/api/zlib.html>



## Kode : Zlib Compress

```
zlib.mjs x
1  import zlib from "zlib";
2  import fs from "fs";
3
4  const source = fs.readFileSync( path: "zlib.mjs");
5  const result = zlib.gzipSync(source);
6
7  fs.writeFileSync( file: "zlib.mjs.txt", result);
8
```



## Kode : Zlib Decompress

```
zlib-decompres.mjs x
JS
1  import zlib from "zlib";
2  import fs from "fs";
3
4  const source = fs.readFileSync( path: "zlib.mjs.txt");
5  const result = zlib.unzipSync(source);
6  console.info(result.toString());
7
8
```

---

# Console



# Console

- Console adalah standard library yang sudah sering kita gunakan
- Secara global, object console bisa kita gunakan tanpa harus melakukan import module, dan console melakukan print text nya ke stdout
- Namun jika kita juga bisa membuat object Console sendiri jika kita mau
- <https://nodejs.org/dist/latest-v16.x/docs/api/console.html>



## Kode : Console

```
console.mjs x
1 import {Console} from "console";
2 import fs from "fs";
3
4 const logFile = fs.createWriteStream( path: "application.log");
5
6 const log = new Console({
7   stdout: logFile,
8   stderr: logFile
9 });
10
11 log.info("Hello world")
12 log.error("Ups");
13
```

---

# Worker Threads





# Worker Threads

- Worker Threads adalah standard library yang bisa kita gunakan untuk menggunakan thread ketika mengeksekusi JavaScript secara paralel
- Worker Threads sangat cocok ketika kita membuat kode program yang butuh jalan secara paralel, dan biasanya kasusnya adalah ketika kode program kita membutuhkan proses yang CPU intensive, seperti misalnya enkripsi atau kompresi
- Cara kerja Worker Threads mirip dengan Web Worker di JavaScript Web API
- [https://nodejs.org/dist/latest-v16.x/docs/api/worker\\_threads.html](https://nodejs.org/dist/latest-v16.x/docs/api/worker_threads.html)



## Kode : Main Thread

```
worker-main.mjs x
1  import {threadId, Worker} from "worker_threads";
2
3  const worker1 = new Worker( filename: "./worker.mjs");
4  const worker2 = new Worker( filename: "./worker.mjs");
5
6  worker1.addListener( event: "message", listener: function (message) {
7      console.info( data: `thread-${threadId} receive message : ${message}`);
8  });
9  worker2.addListener( event: "message", listener: function (message) {
10     console.info( data: `thread-${threadId} receive message : ${message}`);
11 });
12
13 worker1.postMessage( value: 10);
14 worker2.postMessage( value: 10);
15
```



## Kode : Worker Thread

```
worker.mjs x
1  import {parentPort, threadId} from "worker_threads";
2
3  parentPort.addListener( event: "message", listener: function (message) {
4    for (let i = 0; i < message; i++) {
5      console.info( data: `thread-${threadId} send message ${i}`);
6      parentPort.postMessage(i);
7    }
8    parentPort.close();
9  })
10
```

---

# HTTP Client



# HTTP Client

- NodeJS juga memiliki standard library untuk HTTP
- Salah satu fitur di module HTTP adalah HTTP Client, dimana kita bisa melakukan simulasi HTTP Request menggunakan NodeJS
- Terdapat 2 jenis module HTTP di NodeJS, HTTP dan HTTPS
- <https://nodejs.org/dist/latest-v16.x/docs/api/http.html>
- <https://nodejs.org/dist/latest-v16.x/docs/api/https.html>

## Kode : HTTP Client

```
http-client.mjs
1 import https from "https";
2
3 const url = "https://hookb.in/1gmqywqrLLfd6N0061k8";
4 const request = https.request(url, options: {
5   method: "POST",
6   headers: {
7     "Content-Type": "application/json",
8     "Accept": "application/json",
9   }
10 }, callback: function (response : IncomingMessage ) {
11   response.addListener( event: "data", listener: function (data) {
12     console.info( data: `Receive : ${data.toString()}`)
13   })
14 });
```

```
16 const body = JSON.stringify( value: {
17   firstName: "Eko",
18   lastName: "Khannedy",
19 });
20
21 request.write(body);
22 request.end();
23
```

---

# HTTP Server



# HTTP Server

- Standard Library HTTP juga tidak hanya bisa digunakan untuk membuat HTTP Client, tapi juga bisa digunakan untuk membuat HTTP Server
- Untuk kasus sederhana, cocok sekali jika ingin membuat HTTP Server menggunakan standard library NodeJS, namun untuk kasus yang lebih kompleks, direkomendasikan menggunakan library atau framework yang lebih mudah penggunaannya
- <https://nodejs.org/dist/latest-v16.x/docs/api/http.html>





## Kode : Simple HTTP Server

```
http-server.mjs x
1 import http from "http";
2
3 const server = http.createServer( requestListener: (request : IncomingMessage , response : ServerResponse ) => {
4     response.write( chunk: "Hello World");
5     response.end();
6 });
7
8 server.listen( port: 3000);
9
```

# Kode : Request Response HTTP Server

```
1 import http from "http";
2
3 const server = http.createServer( requestListener: (request : IncomingMessage , response : ServerResponse ) => {
4   if (request.method === "POST") {
5     request.addListener( event: "data", listener: function (data) {
6       response.setHeader( name: "Content-Type", value: "application/json");
7       response.write(data);
8       response.end();
9     })
10   } else {
11     response.write( chunk: "Hello World");
12     response.end();
13   }
14 });
15
16 server.listen( port: 3000);
17
```

---

# Cluster



# Cluster

- Seperti yang dijelaskan di awal, bahwa NodeJS itu secara default dia berjalan single thread, kecuali jika kita membuat thread manual menggunakan worker thread, tapi tetap dalam satu process
- NodeJS memiliki standard library bernama Cluster, dimana kita bisa menjalankan beberapa process NodeJS secara sekaligus
- Ini sangat cocok ketika kita menggunakan CPU yang multicore, sehingga semua core bisa kita utilisasi dengan baik, misal kita jalankan process NodeJS sejumlah CPU core
- <https://nodejs.org/dist/latest-v16.x/docs/api/cluster.html>



## Cluster Primary dan Worker

- Di dalam Cluster, terdapat 2 jenis aplikasi, Primary dan Worker
- Primary biasanya digunakan sebagai koordinator atau manajer untuk para Worker
- Sedangkan Worker sendiri adalah aplikasi yang menjalankan tugas nya

## Kode : Cluster Primary

```
cluster.mjs x
1 import cluster from "cluster";
2 import http from "http";
3 import os from "os";
4 import process from "process";
5
6 if (cluster.isPrimary) {
7   for (let i = 0; i < os.cpus().length; i++) {
8     cluster.fork();
9   }
10  cluster.addListener( event: "exit", listener: function (worker : Worker ) {
11    console.info( data: `Worker ${worker.id} is exited`);
12  })
13 }
14
```



## Kode : Cluster Worker

```
14
15 if (cluster.isWorker) {
16   const server = http.createServer( requestListener: (request : IncomingMessage , response : ServerResponse ) => {
17     response.write( chunk: `Response from process ${process.pid}`);
18     response.end();
19     process.exit();
20   })
21   server.listen( port: 3000);
22   console.info( data: `Start cluster worker ${process.pid}`);
23 }
24
25 |
```

---

# Materi Selanjutnya





## Materi Selanjutnya

- NPM (Node Package Manager)
- NodeJS Unit Test
- ExpressJS
- NodeJS Database
- Dan lain-lain