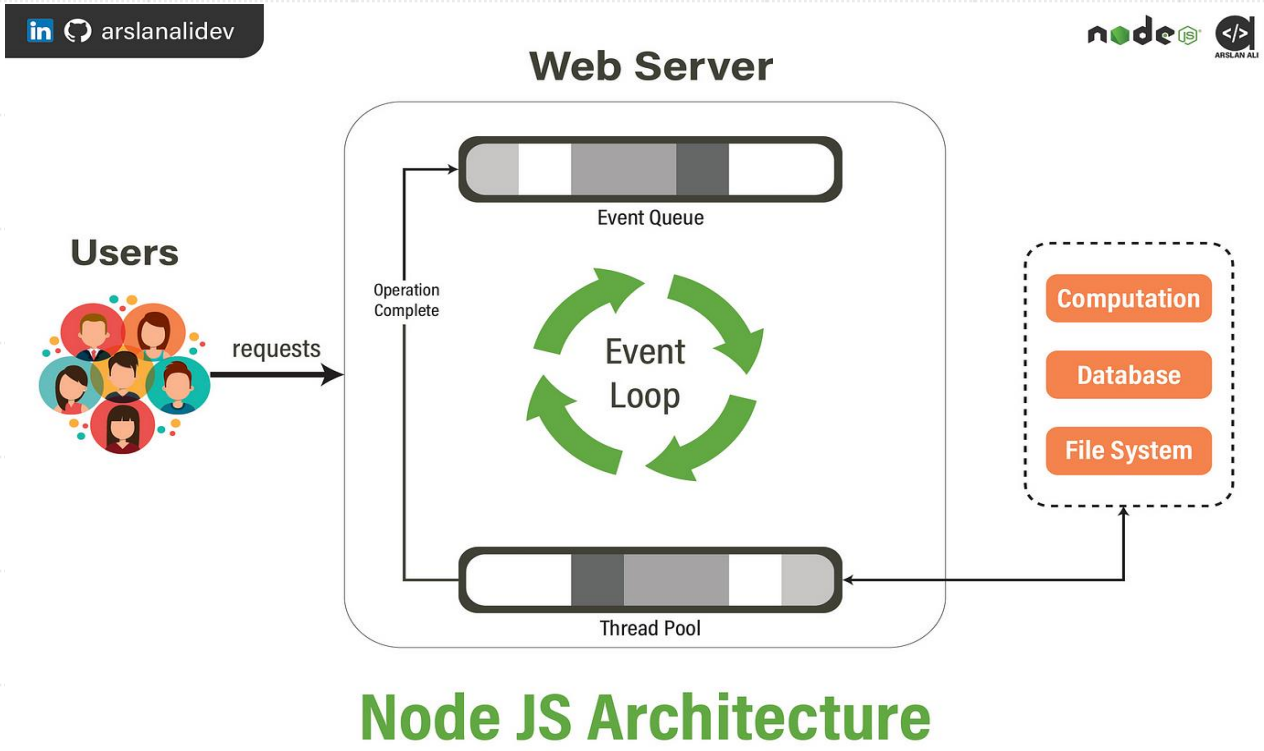
Server-side Web Development 2: Node.js

Backend Web Development

by: Salahuddin ElKazak

Introduction to Node.js for PHP Beginners

- **Asynchronous & Event-Driven:** Node.js handles multiple client-server interactions efficiently.
- **Created by Ryan Dahl in 2009** to improve concurrency.
- Powered by **Google's V8 Engine** (same as Chrome) for fast JavaScript execution and efficient garbage collection.



Src: https://medium.com/nerd-for-tech/nodejs-web-server-architecture-a21d02a33bad

Quick Comparison: Node.js vs. PHP

Node.js

- generates HTML in response to HTTP requests
- requires custom coding for tasks like sending HTML and setting HTTP headers
- provides more control
- may feel more complex to PHP beginners
- Developed in 2009

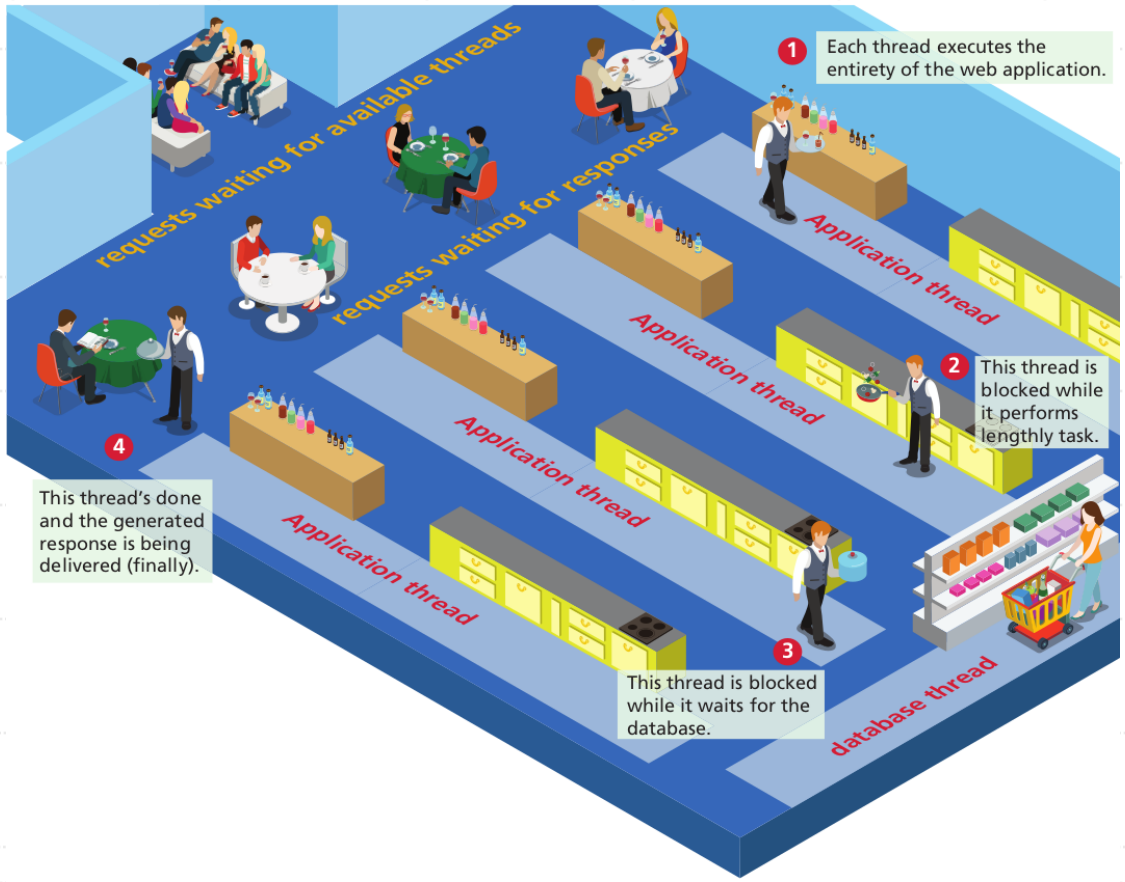
PHP

- generates HTML in response to HTTP requests
- simplifies HTML integration, easily embedded into HTML
- provides less flexibility
- may feel simpler in general
- Developed in 1995

Key Advantages of Node.js

- **JavaScript Everywhere**:
  - Single language (JavaScript) for both client and server.
  - Increases developer productivity.
  - Eases hiring by focusing on a single, popular language.
- **Push Architectures**:
  - Ideal for push-based applications (e.g., real-time chat).
  - Avoids inefficient polling seen in PHP or Ruby, making Node perfect for real-time apps.
- **Nonblocking, Asynchronous Architecture**:
  - Single-threaded event loop enables handling many simultaneous requests efficiently.
  - Reduces the time spent on context switching compared to traditional multi-threaded servers.
  - Well-suited for high-concurrency applications like Walmart's Black Friday traffic.

Blocking vs. NonBlocking



PHP Blocking vs. Node.js NonBlocking

PHP

```
if ( $result = $db->fetchFromDataBase($sql) ) {
   // do something with results
   . . .
}
if ( $data = $service->retrieveFromService($url, $querystring)
   ) {
   // do something with data
   . . .
}
// doesn't need $result or $data, yet still blocked by them
doSomethingElseReallyImportant();
```

Node.js

```
fetchFromDataBase(sql, function(results) {
   // do something with results
   . . .
});
retrieveFromService(url, querystring, function(data) {
   // do something with data
   . . .
});
// this isn't blocked by two previous lines
doSomethingElseReallyImportant();
```

Node.js Ecosystem & Adoption

- **Rich Ecosystem of Tools and Code**:
  - Access to npm, the Node Package Manager, with a massive library of prebuilt code.
  - Supports modern web development approaches like microservices and serverless computing.
- **Broad Adoption**:
  - Used by major companies like Netflix, eBay, LinkedIn, and PayPal.
  - Integral to emerging tech trends like the ***Internet of Things*** and ***cloud services***.

NETFLIX

ebay

PayPal

Node.js Disadvantages

- **Not Ideal for Traditional Web Apps**
  - Complex for accessing relational databases (e.g., MySQL, PostgreSQL).
  - Best suited for real-time, data-intensive applications using NoSQL databases.
- **Single-Thread Limitation**
  - Unsuitable for heavy computational tasks (e.g., video processing, scientific computing).
  - IO-heavy tasks benefit from nonblocking architecture, but long-running tasks monopolize the thread.
- **Complexity for Developers**
  - Asynchronous programming can be difficult, even for experienced JavaScript developers.
  - Callback queues and promises/async-await add to development complexity.

Rapid Lab Exercise: Setup and Run Node!

1. Search in google for "nodejs prebuilt installer windows" or go to
2. Follow the steps and show me that you can run ***node -v*** and ***npm -v***
3. Run this code:

```
// make use of the http module
const http = require('http');
// configure HTTP server to respond with simple message to all requests
const server = http.createServer(function (request, response) {
    response.writeHead(200, {"Content-Type": "text/plain"});
    response.write("Hello this is our first node.js application");
    response.end();
});
// Listen on port 8080 on localhost
const port = 8080;
server.listen(port);
// display a message on the terminal
console.log("Server running at port=" + port);
```

Too much code for a simple file server?

```
const http = require("http");
const fs = require("fs");
const path = require("path");

const server = http.createServer((req, res) => {
  const filePath = path.join(process.cwd(), req.url);

  fs.access(filePath, fs.constants.F_OK, (err) => {
    if (err) {
      res.writeHead(404);
      res.end('404 Not Found');
      return;
    }

fs.readFile(filePath, (err, data) => {
      if (err) {
        res.writeHead(500);
        res.end('500 Internal Server Error');
        return;
      }
      res.writeHead(200);
      res.end(data);
    });
  });
});

server.listen(7000, "localhost");
console.log("Server running at http://127.0.0.1:7000/");
```

With express!

```
const path = require("path");
const express = require("express");
const app = express();
const options = {
   // maps root requests (e.g. "/") to subfolder named "public"
   root: path.join(__dirname, "public")
};
// With express, you define handlers for routes.
app.get("/:filename", (req, resp) => {
   resp.sendFile(req.params.filename, options, (err) => {
      if (err) {
         console.log(err);
         resp.status(404).send("File Not Found");
      }
      else {
         console.log("Sent:", req.params.filename);
      }
   });
});
app.listen(8080, () => {
   console.log("Example express file server listening on port 8080");
});
```