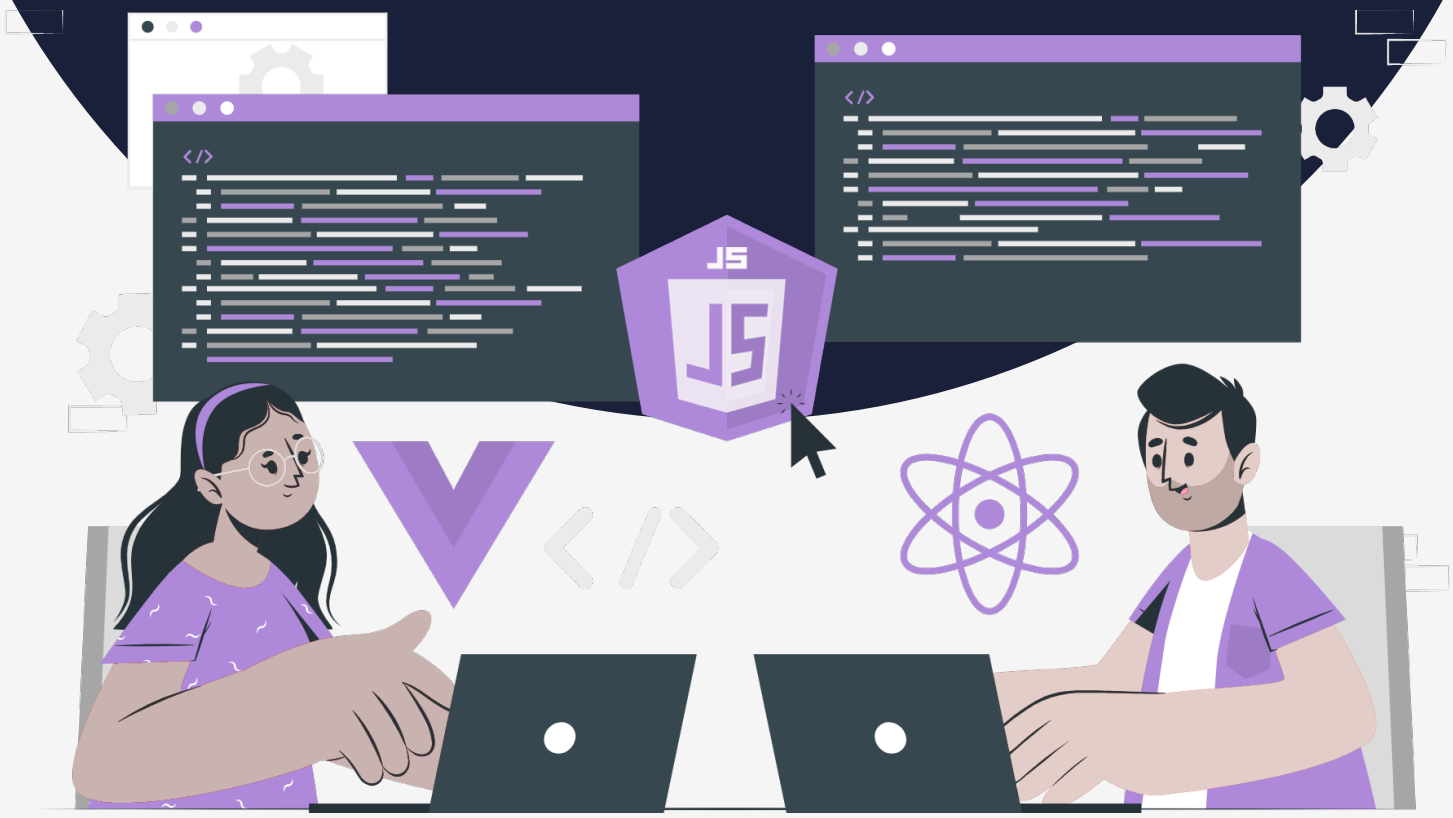
Lesson:

NodeJS Architecture



PW SKILLS

List of content:

Architecture

Node.js is a programming language that utilizes both **JavaScript and C/C++** to create its core components. Node.js is a platform that utilizes JavaScript and is primarily employed for developing web applications that are highly focused on **input/output operations**, including but not limited to chat applications and multimedia streaming websites. The platform is constructed using Google Chrome's V8 JavaScript engine. A web application is a type of software that executes on a server and is displayed by a client's browser that obtains all the application's resources over the internet.

Web application consists of the following components:

Client: User who interacts with the server by sending out requests.
Server: The server is in charge of receiving client requests, performing tasks, and returning results. It serves as a bridge between the front-end and the stored data, allowing clients to perform operations on the data.
Database: where a web application's data is stored. Depending on the client's request, the data can be created, modified, and deleted.

To manage several concurrent clients, Node.js employs a **"Single Threaded Event Loop"** design. The JavaScript **event-based model and the JavaScript callback mechanism** are employed in the Node.js Processing Model. It employs two fundamental concepts:

- **Asynchronous model**
- **Non-blocking of I/O operations**

Lets Understand it, with the help of some scenario...

Consider an example of Restaurant who is trying hard to impress their customers with premium services...

1. Hotel with the capacity of 50 Tables decides to hire 50 Waiters to Cater to each of the table individually. Whenever a new customer enters the hotel, a individual Waiter is assigned to each table.
2. The customer enters the restaurant and takes time to **look for the menu** and decide what to order, meanwhile the waiter keeps on waiting for customer to complete a time consuming task of deciding the order to be placed. The waiter in this case is in waiting mode, doing nothing.
3. Once the order is decided, the waiter takes the **order to the chef for preparation**. The Chef will consume another 15 minutes for processing the order, meanwhile the Waiter is idle again, waiting for the food preparation
4. Once the food is prepared, the Waiter **brings the food to the customer** for Consumption, the customer starts eating the food. Meanwhile the Waiter is waiting for the customer to finish eating the order.
5. Once the customer is **done eating**, the waiter takes off the **cutlery back to Kitchen** and **process the Bill**, the bill is presented to the customer. Once the customer is gone, this waiter is now available to serve some other table.

In the entire above process, there is inefficiency involved, the waiter just like the main thread keeps on waiting for any time-consuming task to complete. During this wait time the productivity of this resource (Waiter) is absolute Zero. We have limited number of resources available, if these resources are not utilized in optimized manner, the overall productivity and effectiveness will reduce for the entire system.

Therefore we require some alternate methods to make our available resources (Thread / Waiter) more productive and reduce the waiting time that it spends of these time consuming tasks.

To understand, with this analogy ,how node works, lets consider the same example of Restaurant who want to work in more optimized manner this time:

1. Hotel with the capacity of 50 Tables decides to hire 1 waiter to Cater to the request. This time we are not assigning a separate waiter to each available table.
2. Waiter is **available to address the customer**, while the customer **takes time to look for the menu** and decide what to order, the waiter is available to address some new customers and cater to some existing customers.
3. Once the order is decided, the waiter takes the order to the chef for preparation. While the Chef prepares for the food, the waiter is available to cater other customers as well rather than waiting for food to be cooked.
4. Once the food is prepared, the Waiter brings the food to the customer for Consumption, the customer starts eating the food. Rather than waiting for customer to complete, the chef will move to other customers and cater them simultaneously
5. Once the customer is done eating, the waiter takes off the cutlery back to Kitchen and process the Bill, the bill is presented to the customer. And this waiter is again available to any other customer.

This way the capacity of Waiter is fully utilized and the resources available to the Hotel are fully utilized. Hence the performance of the system increases.

Now if we see the waiter as a thread, node functionality works in exactly the same way.

Let us first look at the components before jumping to the actual working of nodeJS.

Components

Node.js architecture consists of several components that work together to handle incoming client requests and process responses. These components are:

1. Requests: Node.js receives two types of requests, incoming and outgoing. Incoming requests can be either complex (blocking) or simple (non-blocking), depending on the task that needs to be performed by the web application or software.

2. Node.js Server: It is a server-side platform that receives client requests, processes them, and sends the corresponding responses back to the clients.

3. Event Queue: Incoming client requests are stored in the event queue and then passed on to the event loop one by one.

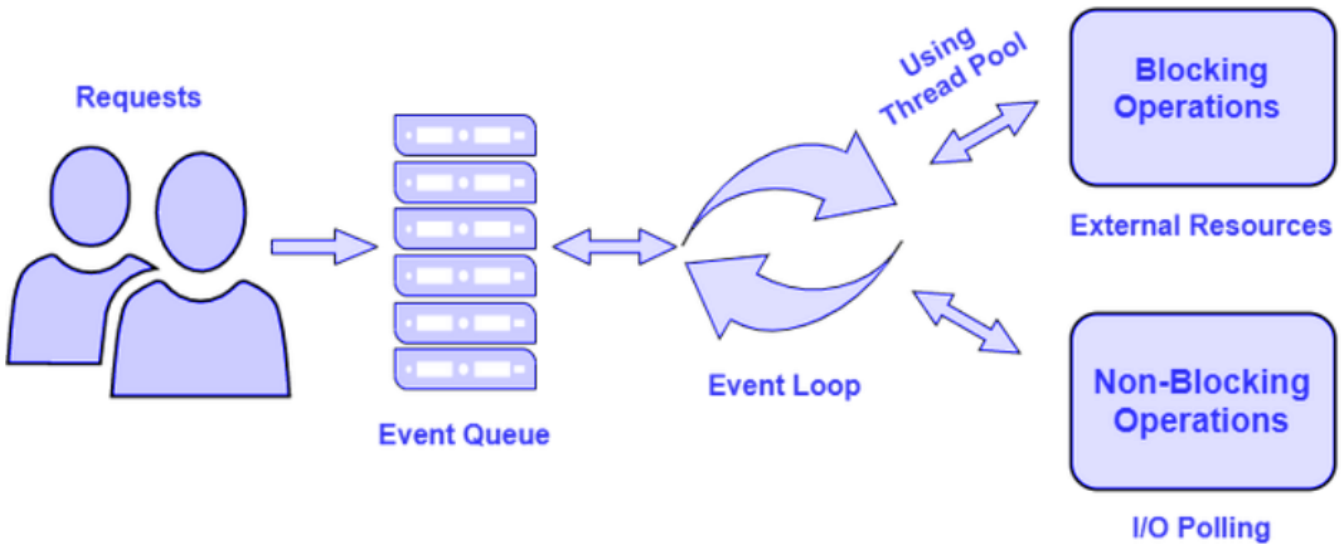
4. Event Loop: It is an indefinite loop where requests are received, processed, and responses are returned to the clients. The event loop is a critical component of Node.js architecture, as it allows for non-blocking I/O operations.

5. Thread Pool: Node.js has a pool of threads available for carrying out tasks required to fulfill client requests. This ensures that multiple requests can be processed simultaneously, improving the performance of Node.js applications.

6. External Resources: These are resources used by Node.js for blocking client requests, computation, data storage, etc. Examples of external resources include databases, file systems, and network resources.

Overall, these components work together to create a scalable, efficient, and non-blocking architecture that is ideal for building web applications and software.

Working



- Users send requests (blocking or non-blocking) to the server for performing operations.
- The requests enter the Event Queue first at the server-side.
- The Event queue passes the requests sequentially to the event loop. The event loop checks the nature of the request (blocking or non-blocking).
- Event Loop processes the non-blocking requests which do not require external resources and returns the responses to the corresponding clients
- For blocking requests, a single thread is assigned to the process for completing the task by using external resources.
- After the completion of the operation, the request is redirected to the Event Loop which delivers the response back to the client.

Node.js is a JavaScript runtime that is built on Chrome's V8 JavaScript engine and is single-threaded. The event loop is the primary component that allows Node.js to run I/O operations from blocking to non-blocking manner, and it manages all asynchronous tasks, such as code in the callback function. Once a task is completed, it is sent back to the execution queue.

Advantages of NodeJS:

Compared to other server-side languages, Node js has distinct advantages. Its asynchronous model and non-blocking I/O operation improve the scalability and performance of web applications built on other frameworks. Node js can easily handle multiple client requests without requiring multiple threads, consuming less memory and resources. Additionally, it is highly scalable and provides high performance. Node js is also flexible with multiple frameworks and makes the development process easier.