Single-threaded, non-blocking performance in Node.js works great for a single process. But eventually, one process in one CPU is not going to be enough to handle the increasing workload of your application.

Using multiple processes is the best way to scale a Node application. Node.js is designed for building distributed applications with many nodes. This is why it’s named Node. Scalability is baked into the platform and it’s not something you start thinking about later in the lifetime of an application.

The Child Processes Module

* We can easily spin a child process using Node’s child\_process module and those child processes can easily communicate with each other with a messaging system.
* The child\_process module enables us to access Operating System functionalities by running any system command inside a, well, child process.

A "child process" typically refers to a separate and independent program or subprocess that is created and run by another program, often called the "parent process." Child processes are commonly used in computer programming for various purposes, including:

* Concurrency: Child processes allow a program to perform multiple tasks simultaneously. Each child process runs independently of the parent process, enabling concurrent execution of tasks, which can improve overall program performance.
* Isolation: Child processes are isolated from the parent process and other child processes. This isolation means that issues or failures in one child process typically do not affect the stability of the parent process or other child processes. This is important for maintaining the robustness and reliability of a program.
* Parallelism: In multi-core or multi-CPU systems, child processes can be used to take advantage of parallel processing capabilities. Each child process can run on a separate CPU core, potentially speeding up the overall execution of tasks.
* Running External Programs: Child processes can be used to execute external programs or scripts from within a parent program. For example, a parent program written in one programming language might spawn a child process to run a program written in another language.
* Modularity: Child processes can be used to break down complex tasks into smaller, more manageable components. Each child process can handle a specific part of a larger task, making the code more modular and easier to maintain.

In the context of Node.js or other programming environments, child processes are often used to run separate instances of Node.js scripts or external commands. They provide a way to execute tasks in the background or in parallel, ensuring that the main program remains responsive and stable.

Overall, a child process is a way for a program to create and manage smaller, independent programs or tasks to achieve specific goals efficiently and without compromising the overall system's reliability.

Child processes in Node.js allow you to

1. Run external programs or scripts.
2. Perform tasks concurrently without blocking the main program.
3. Keep your main program stable even if a child process encounters an issue.

In summary, child processes and the **spawn** method in Node.js provide a powerful way to run external programs concurrently, handle their output, and ensure the stability of your application. They are crucial for building efficient, responsive, and reliable Node.js applications.

Child Process in Node.js:

A child process in Node.js is an independent program or subprocess created and run by another program, called the parent process.

Child processes are used to perform tasks concurrently, run external programs or scripts, and keep the main program stable even if a child process encounters an issue.

They are essential for achieving concurrency, scalability, and robustness in Node.js applications.

Child processes are created and managed using the child\_process module in Node.js.

spawn Method in Node.js:

spawn is a method in the child\_process module used to create child processes in Node.js.

It is used for running external programs or scripts concurrently with the main Node.js program.

spawn is non-blocking and asynchronous, allowing the main program to continue executing without waiting for the child process to finish.

It enables streaming of data from the child process's standard output and standard error streams.

The spawn method is versatile, allowing you to pass arguments, set environment variables, and control the child process's behavior.

It is suitable for various use cases, including running external commands, executing scripts, and handling data processing tasks.

//exec

The exec method is part of the child\_process module in Node.js.

It is used for executing shell commands or running external programs asynchronously.

Unlike spawn, which allows you to run a command with arguments in an array, exec accepts a single string that represents the entire shell command you want to execute.

The exec method runs the command in a shell, which means you can use shell-specific syntax and features, such as pipes (|), redirection (>, <), and command chaining.

It provides a callback function that is called when the child process completes. The callback function receives three parameters: error, stdout (standard output), and stderr (standard error).