**Giới thiệu về NodeJS**

Bên cạnh các nền tảng (platform) gạo cội để phát triển các ứng dụng phần mềm như JVM (Java Virtual Machine) hay .NET, Node JS là đứa em được ra đời sau và đã mang lại làn sóng mới trong giới lập trình. Ngày nay bạn có thể thấy các ứng dụng xây dựng dựa trên nền tảng Node ở bất cứ đâu, từ các ứng dụng web trên phía Server, cho đến các ứng dụng desktop phía người dùng cuối, hay trong các ứng dụng về kết nối vạn vật (internet of things)… Bài viết này sẽ giới thiệu với các bạn nền tảng Node và những thứ làm cho nền tảng này trở nên thật đặc biệt, dù sinh sau đẻ muộn nhưng Node đã trở thành một trong những nền tảng lập trình rất phổ biến những ngày hôm nay, thậm chí không hề kém cạnh những nền tảng đi trước như JVM hay .NET.

I. Sự hình thành của Node

Chắc trong chúng ta chẳng ai còn xa lạ gì với trình duyệt web Google Chrome và có một điều nữa cũng chẳng có gì mới mẻ là trình duyệt này có khả năng để mở ra các trang web và chạy các đoạn mã Javascript trên trang web đó. Việc thực thi các đoạn mã Javascript như trên là nhờ vào một Javascript engine nằm bên trong trình duyệt Chrome có tên là V8. Trong thực tế V8 đã chứng tỏ nó là một engine mạnh mẽ, được viết bằng C++, V8 có khả năng để thông dịch và thực hiện các đoạn mã Javascript rất là nhanh. Và tất cả những điều kể trên chỉ là câu chuyện về việc lập trình ở phía người dùng cuối (hay gọi là phía front-end).

Đứng trước một thực tế rõ ràng về sự mạnh mẽ của V8, ý tưởng bỗng nảy ra trong đầu các nhà sáng lập là tách V8 này riêng ra khỏi Chrome và tạo thành một ứng dụng có khả năng thực thi các đoạn mã Javascript ở phía server side, và người ta gọi ứng dụng đó là Node. Nếu bạn nào biết Java, thì chỗ này bạn sẽ rất dễ hiểu khi được nói rằng Node có vai trò tương đương như JVM (Java Virtual Machine).

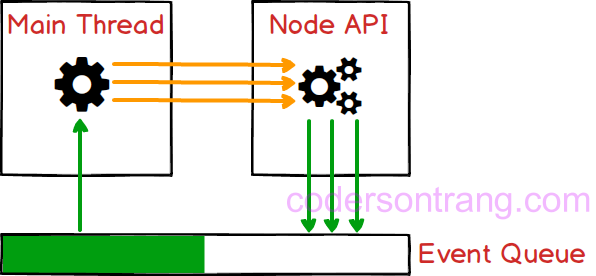
Cũng giống như JVM, Node được coi là một nền tảng, tức là một môi trường cho các ứng dụng chạy trên đó. Nếu như JVM thực thi các đoạn mã byte code, thì Node thực thi các đoạn mã Javascript. Về phía lập trình ứng dụng, khi làm trên JVM thì bao giờ các lập trình viên cũng dùng các ngôn ngữ bậc cao như Java, Scala, Groovy hay Kotlin để viết mã nguồn, sau đó mã nguồn này sẽ được biên dịch thành byte code. Nhưng trên nền tảng Node, bản thân Node có thể hiểu và thực thi Javascript luôn, nên ngôn ngữ viết ở phía ứng dụng có thể là Javascript, hoặc các ngôn ngữ có khả năng biên dịch thành Javascript như Typescript chẳng hạn.

Sự ra đời của Node mang đến một luồng gió mới trong giới lập trình. Nếu trước đây ai cũng chỉ nghĩ Javascript chỉ dùng cho việc phát triển các ứng dụng web phía người dùng cuối, thì giờ đây cùng với Node, javascript đã có mặt tại phía Server. Điều này cũng có nghĩa là nếu bạn chỉ biết mỗi Javascript, thì bạn vẫn có khả năng để xây dựng nên một hệ thống hoàn thiện từ đâu chí cuối, hay nôm na bạn có thể đang bước trên con đường trở thành full-stack developer cùng với Node và Javascript.

II. Lập trình Non-Blocking trong Node

Nền tảng Node khiến người ta phải chú ý có lẽ không phải vì ở engine V8 hay khả năng hỗ trợ ngôn ngữ Javascript, mà là ở phong cách lập trình theo hướng Non-Blocking. Các thao tác liên quan đến Non-Blocking ở trong Node đa phần là các thao tác liên quan đến IO, ví dụ như việc đọc và ghi dữ liệu xuống đĩa, hay gọi các Web API chẳng hạn. Và việc sử dụng các thao tác Non-Blocking khiến cho các ứng dụng viết trên nền tảng Node có khả năng sử dụng tài nguyên tính toán (CPU) một các hiệu quả nhất.

Để tạo được cơ chế Non-Blocking, các ứng dụng Node hoạt động theo mẫu thiết kế Event Loop. Trước khi giải thích chi tiết hơn về mẫu thiết kế này, bạn hãy nhìn xuống hình vẽ minh họa dưới đây:



Trong mô hình trên, ta có thể kể tên đến ba thành phần chính như sau:

Event Queue: Đóng vai trò một cái kho để lưu trữ các Event (sự kiện), Event ở đây có thể hiểu là một xử lý nào đó trong chương trình, mỗi Event bao gồm thông tin để phân loại xem Event đó là gì và các dữ liệu đi kèm. Vì là cấu trúc hàng đợi nên Event Queue sẽ hoạt động theo nguyên tắc vào trước ra trước (First In First Out – FIFO). Điều này đảm bảo rằng thứ tự của các Event được lấy ra để xử lý cũng giống như các thứ tự mà chúng được đưa vào bên trong hàng đợi Event Queue.

Main Thread: là Thread chính của ứng dụng NodeJS, nói đây là Thread chính vì nó sẽ là ngọn nguồn của chương trình cũng như sự kết thúc của chương trình. Thread chính này có nhiệm vụ là xử lý các tính toán khi nhận được các Event từ Event Queue. Đây cũng chính là Thread duy nhất của ứng dụng Node mà lập trình viên có quyền điều khiển. Đó cũng chính là lý do vì sao mà người ta vẫn nói ứng dụng Node là đơn tuyến (single-thread). Và vì là đơn tuyến nên khi lập trình các lập trình viên sẽ không phải đau đầu về các vấn đề xử lý đồng bộ (concurrency) giữa các thread như trong một số các nền tàng khác như Java.

Node API: Là đơn vị chịu trách nhiệm xử lý các thao tác IO. Các thao tác IO sẽ được xử lý ở đây bằng cơ chế đa tuyến (multi-threading). Và mỗi một thao tác IO sau khi hoàn thành sẽ trả về kết quả dưới dạng một Event và Event này sẽ được đưa vào hàng đợi Event Queue.

Với ba thành phần trên, thì cách ứng dụng Node xử lý sẽ như sau:

Main thread sẽ chạy các câu lệnh xử lý tính toán được khai báo trong mã nguồn. Ở bất cứ chỗ nào có liên quan đến thao tác IO, Main thread sẽ tạo ta một lời gọi Non-Blocking sang cho Node API và sau đó tiếp tục thực hiện các lệnh khác mà không cần phải chờ thao tác IO kia hoàn thành

Node API khi nhận được yêu cầu từ Main Thread sẽ trực tiếp xử lý các thao tác IO bằng các Thread khác nhau. Một khi có thao tác IO hoàn thành, Node API sẽ gói kết quả dưới dạng một Event và đặt kết quả vào hàng đợi Event Queue

Event ở trong hàng đợi Event Queue sẽ lại lần lượt được xử lý bởi Main Thread. Trong Main Thread, đoạn mã dùng để xử lý các Event thường được khai báo ở dưới dạng các callback.

Quá trình trên cứ như vậy lặp đi lặp lại, tạo nên một vòng tuần hoàn của các Event trong ứng dụng. Việc lập trình sẽ được áp đặt theo phong cách hướng đến xử lý các sự kiện thay vì kiểu nghĩ xử lý tuần tự truyền thống. Và phong cách lập trình như vậy được đặt một cái tên là Reactive Programming.

III. Ngôn ngữ cho các ứng dụng trên nền tảng Node

Ngôn ngữ cho việc xây dựng các ứng dụng Node chính là Javascript, một ngôn ngữ rất thân quen đối với các lập trình viên web. Javascript là một bản thể của ECMA Script (ES) là một chuẩn ngôn ngữ với các tính năng trong cú pháp được cập nhật khá thường xuyên. Phiên bản mới nhất của ECMA Script tính đến thời điểm này là phiên bản ES 6.

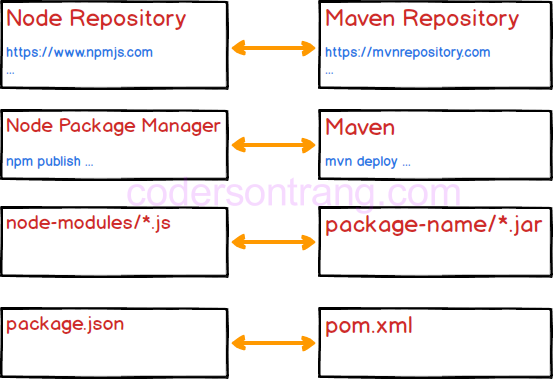
Tuy nhiên Javascript ở phía các trình duyệt Web thường không mang đầy đủ tất cả các tính năng mới nhất của ES. Vì thế khi lập trình Web, để có thể tạo ra ứng dụng có khả năng tương thích trên nhiều trình duyệt khác nhau, các lập trình viên thường phải sử dụng các cú pháp cũ hơn. Rất may mắn là điều này không xảy ra trong thế giới của Node. Vì vấn đề ở đây chỉ là làm sao viết ứng dụng tương thích với một engine V8 duy nhất mà thôi. Hay nói cách khác, khi viết ứng dụng Node, lập trình viên luôn có thể sử dụng ngôn ngữ Java Script với các tính năng mới nhất của ECMA Script.

Java Script là một ngôn ngữ thực sự linh hoạt, tinh gọn và phải thừa nhận rằng viết chương trình bằng ngôn ngữ này, lập trình viên sẽ tránh được khá nhiều các đoạn mã dư thừa hay còn gọi là boilerplate code. Tuy nhiên có một vấn đề đó là Javascript là một ngôn ngữ định nghĩa kiểu yếu, cho nên sẽ gây ra tiềm ẩn về các lỗi mà ta không thể phát hiện được ngay từ khi biên dịch ứng dụng. Để đến khi chạy ứng dụng rồi mới phát hiện ra thì đã là quá muộn. Điều này đặc biệt rất dễ xảy ra nếu ứng dụng có số lượng mã nguồn lớn, và một thiệt thòi nữa là các IDE rất khó để phát hiện ra các lỗi tiềm ẩn rồi cảnh báo cho các ngôn ngữ có định nghĩa kiểu yếu.

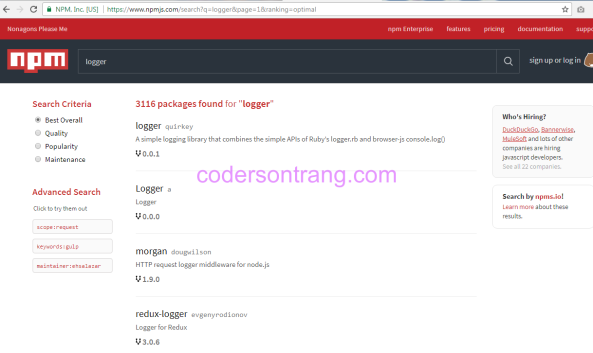
Để khắc phục vấn đề trên thì người ta có thể dùng một ngôn ngữ định nghĩa kiểu mạnh để viết ứng dụng. Sau đó dùng bộ biên dịch để dịch từ ngôn ngữ kiểu mạnh này sang Javascript. Một ví dụ về ngôn ngữ kiểu mạnh như nói ở trên có thể kể đến là Type Script là một ngôn ngữ được phát triển bởi Microsoft. Có một số bạn hay nhầm tưởng rằng Node có thể hiểu được Type Script, nhưng câu trả lời sẽ là “Không Đúng”, Node với engine V8 sẽ chỉ có thể hiểu Javascript mà thôi. Nó mặc kệ cho lập trình viên chúng ta viết ứng dụng bằng gì thì viết, sau cùng tất cả sẽ phải chuyển thể về Javascript để có thể thực thi được trên Node. Điều này cũng giống như JVM chỉ hiểu được bytecode vậy. Tất cả các ứng dụng viết bằng đủ thể loại ngôn ngữ như Java, Scala, Groovy, Kotlin rồi sau cùng vẫn phải chuyển thể về bytecode.

IV. Hệ sinh thái các thư viện trong nền tảng Node

Node cho phép tạo ra các thư viện dưới dạng các node-modules để có được sử dụng đi sử dụng lại ở các ứng dụng khác nhau. Điều này giống như bên phía Java có các file \*.jar vậy. Mỗi một node-module là tổ hợp của các file mã nguồn javascript \*.js và nó cũng được đánh số phiên bản (version) để phục vụ cho việc quản lý cấu hình. Việc quản lý các phiên bản của từng thư viện, cũng như quản lý sự phụ thuộc giữa các thư viện với nhau bên trong Node là nhờ một chương trình quản lý có tên là Node Package Manager hay được viết tắt là npm. Npm có khả năng quản lý và lưu trữ các thư viện trên các repository và nếu so sánh thì vai trò của nó sẽ tương tự với Maven trong Java vậy. Hình dưới đây sẽ thể hiện một ánh xạ giữa những gì trong thế giới của Node sang những thứ tương tự của Java.



Một trong những repository nổi tiếng của những lập trình viên Node là https://www.npmjs.com/.Có thể lên trang chủ repository này và tìm kiếm sẽ thấy có rất nhiều thư viện giúp giải quyết một loạt các vấn đề khi lập trình. Thậm chí đối với một vấn đề cụ thể thì cũng có nhiều các thư viện khác nhau. Từ đó có lẽ việc khó khăn đối với các lập trình viên Node chính là việc sẽ lựa chọn thư viện nào để có thể áp dụng vào trong bài toán của riêng mình. Hình vẽ dưới đây là việc ta thử tìm bộ thư viện hỗ trợ cho logging, kết quả là ta có thể tìm thấy hàng nghìn sự lựa chọn khác nhau trên repository của Node.



V. So sánh nền tảng Node và nền tảng lập trình truyền thống Java

Cái khác biệt đáng chú ý đầu tiên giữa hai nền tảng lập trình Node và Java chính là ở kiến trúc của ứng dụng. Như đã miêu tả ở trên, đối với các ứng dụng Node, lập trình viên chỉ có thể can thiệp vào 1 thread duy nhất là Main Thread, trong khi đó các ứng dụng Java lại cho phép lập trình viên tùy biến số lượng Thread sử dụng qua các API về Thread và xử lý đồng bộ. Vì vậy cho nên về mặt lý thuyết thì cùng là một ứng dụng nhưng viết bằng Java dưới dạng multi-threading và chạy trên máy có CPU nhiều nhân thì sẽ có năng suất tốt hơn ứng dụng tương tự nhưng viết trên Node.

Tiếp đến nữa là JVM và Node V8 đều được viết bằng C++, nhưng bản thân JVM ra đời trước V8, đã trưởng thành qua hàng thập kỉ và được các chuyên gia nghiên cứu tối ưu liên tục. Vì thế không thể phủ nhân rằng đây cũng là một nhân tố góp phần làm tăng hiệu suất hoạt động cho các ứng dụng Java so với các ứng dụng Node. Đấy là còn chưa kể đến JVM thực hiện bytecode là ngôn ngữ đã được sàng lọc chỉnh sửa qua một trình biên dịch nên cú pháp luôn ở mức tối ưu so với mã nguồn. Còn với Node thì Javascript engine V8 sẽ thực hiện ngay Javascript được viết trực tiếp bằng lập trình viên mà không qua trình biên dịch nào, vì thế hiệu suất của ứng dụng Node không phải lúc nào cũng tốt, đôi khi cũng phụ thuộc vào trình độ của lập trình viên nữa.

Tuy nhiên không phải vì thế mà Node trở nên thua kém, bằng chứng là ngày nay thì các ứng dụng trên nền tảng Node ngày càng phổ biến, len lỏi mọi ngóc ngách của các hệ thống thông tin. Trong thực tiễn, hiệu năng xử lý và tốc độ chỉ là một phần của câu chuyện, còn rất nhiều các khía cạnh khác nữa mà ngành công nghiệp phần mềm phải chú ý tới với mục tiêu cuối cùng là làm sao có được sản phẩm đáp ứng được nhu cầu sử dụng mà giá rẻ nhất.

Như đã nói ở trên, Java cho phép lập trình viên làm các ứng dụng multi-threading nhưng điều này cũng mang lại một rủi ro tiềm ẩn là độ phức tạp của mã nguồn sẽ tăng lên, từ đó dễ có nguy cơ gây lỗi. Thay vì rất tốn kém để thuê các lập trình viên giỏi Java làm nên một ứng dụng multi-threading với sự tối ưu về hiệu năng xử lý, thì người ta hoàn toàn có thể chấp nhận làm ứng dụng tương tự trên Node với hiệu năng thấp hơn một chút, nhưng có khả năng mở rộng bằng cách nhân rộng số lượng ứng dụng lên. Nói cách khác, tiền thuê lập trình viên giỏi Java có thể để dành để thuê nhiều server hơn để chạy ứng dụng Node.

Tiếp nữa là bản thân V8 được đánh giá là “nhẹ” (light weight) hơn JVM, vì thế mà Node được chọn lựa để làm các ứng dụng cho các thiết bị IoT, khi mà ở đó mọi tài nguyên (CPU, RAM … ) đều rất là hạn chế.

Còn một số các so sánh khác như là về IDE hỗ trợ quá trình lập trình (coding, debugging …), các IDE được đánh giá là hỗ trợ cho Java tốt hơn so với Node. Một phần vì bản thân Java là một ngôn ngữ lâu đời, một phần nữa vì nó là ngôn ngữ định nghĩa kiểu mạnh (strong type) so với Javascript là một ngôn ngữ định nghĩa kiểu yếu. Tuy nhiên những sự khác biệt này theo thời gian có lẽ sẽ dần không đúng nữa. Và ngay tại thời điểm hiện tại thì lập trình viên hoàn toàn có thể viết ứng dụng Node bằng ngôn ngữ định nghĩa kiểu mạnh như Typescript để có được sự hỗ trợ tốt hơn từ các IDE.

Về sự phong phú các thư viện cũng như cộng đồng mã nguồn mở thì cả Node và Java đều đang sở hữu những đế chế riêng hết sức to lớn của mình. Mặc dù ra đời sau nhưng hệ sinh thái các thư viện của Node cũng không hề kém Java một chút nào, cộng đồng các lập trình viên tham gia vào phát triển trên nền tảng Node cũng đông khủng khiếp. Tuy nhiên ở một số các tác vụ đặc biệt, nền tảng Java một lần nữa khẳng định vị thế qua sự thâm niên của nó trong việc phát triển các bộ thư viện có thể coi là những báu vật như thư viện về Search Engine ( Lucene ), về Computer Vision, hay liên quan đến tài chính với các vấn đề về số lớn, số thập phân mà ở đó bản thân ngôn ngữ kiểu yếu như Javascript trở nên khá khó khăn để đáp ứng.

Có lẽ còn rất nhiều khía cạnh khác để đem ra so sánh giữa hai nền tảng, nhưng tựu chung lại thì tất cả đều tuân theo quy luật muôn thủa là không có một giải pháp cố định nào cho mọi bài toán cả. Mọi sự so sánh đều là khập khiễng vì nó còn phụ thuộc vào bối cảnh sử dụng. Sự tài tình trong lựa chọn của các chuyên gia giải pháp đôi khi có thể nâng tầm lên đến mức được gọi nghệ thuật.

**Các tính năng của Node js**

Sau đây là một số tính năng quan trọng giúp Node.js trở thành lựa chọn hàng đầu cho các kiến trúc sư phần mềm.

Lập trình hướng sự kiện và không đồng bộ: Toàn bộ API trong thư viện Node.js đều không đồng bộ, hay không bị chặn. Về cơ bản điều này có nghĩa là một server sử dụng Node.js sẽ không bao giờ chờ một API trả về dữ liệu. Server sẽ chuyển sang API kế tiếp sau khi gọi API đó và cơ chế thông báo của Events trong Node js giúp server nhận được phản hồi từ lần gọi API trước.

Cực kỳ nhanh chóng: Được xây dựng trên Công cụ JavaScript V8 của Google Chrome, thư viện Node.js có khả năng xử lý mã vô cùng nhanh.

Đơn luồng/Single thread nhưng có khả năng mở rộng cao: Node.js sử dụng một mô hình luồng đơn với vòng lặp sự kiện/event. Cơ chế event cho phép máy chủ phản hồi non-blocking và cũng cho phép khả năng mở rộng cao hơn so với các server truyền thống hỗ trợ giới hạn các thread để xử lý yêu cầu. Node.js sử dụng một chương trình đơn luồng, cùng một chương trình có thể cung cấp dịch vụ cho một số lượng yêu cầu lớn hơn so với các máy chủ truyền thống như Apache HTTP Server.

Không có buffer - Các ứng dụng Node.js không có vùng nhớ tạm thời (buffer) cho bất kỳ dữ liệu nào. Các ứng dụng này chỉ đơn giản xuất dữ liệu theo khối.

License - Node.js được phát hành theo giấy phép MIT.

Đối tượng sử dụng Node js

Danh sách các công ty đang sử dụng Node js có thể kể đến một số tên tuổi lớn như eBay, General Electric, GoDaddy, Microsoft, PayPal, Uber, Wikipin, Yahoo !, và Yammer. Bạn có thể tham khảo link sau nếu muốn biết toàn bộ các dự án, ứng dụng và công ty đang sử dụng Node js trên thế giới.

Nên và không nên sử dụng Node.js

Dưới đây là một số lĩnh vực mà Node.js đang cho thấy những đóng góp hết sức quan trọng:

- Các ứng dụng liên quan đến I/O

- Các ứng dụng Data Streaming

- Các ứng dụng dữ liệu chuyên sâu theo thời gian thực (DIRT)

- Các ứng dụng dựa trên API JSON

- Các ứng dụng Single Page

Không nên sử dụng Node.js cho các ứng dụng CPU chuyên sâu.

Các ưu điểm của Node js

- Node.js là một framework nguồn mở được thông qua theo MIT license. (Giấy phép MIT là giấy phép phần mềm miễn phí có nguồn gốc tại Viện Công nghệ Massachusetts (MIT).)

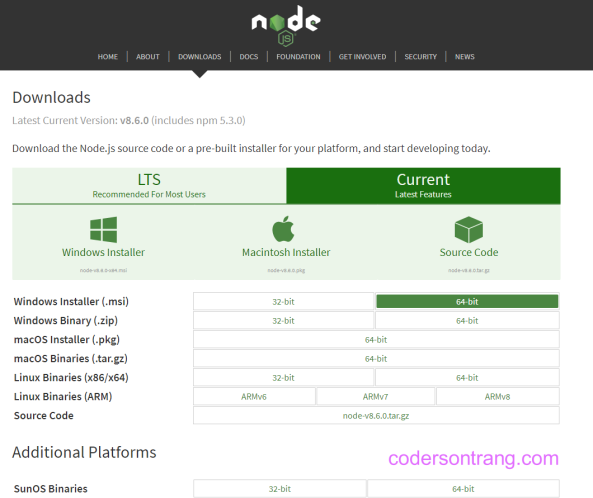
- Sử dụng JavaScript để xây dựng toàn bộ các ứng dụng server side.

- Framework gọn nhẹ bao gồm các module tối thiểu. Các module khác có thể được thêm theo nhu cầu của từng ứng dụng.

- Không đồng bộ theo mặc định, nhờ vậy, hoạt động nhanh hơn các framework khác.

- Framework đa nền tảng chạy trên Windows, MAC hoặc Linux.

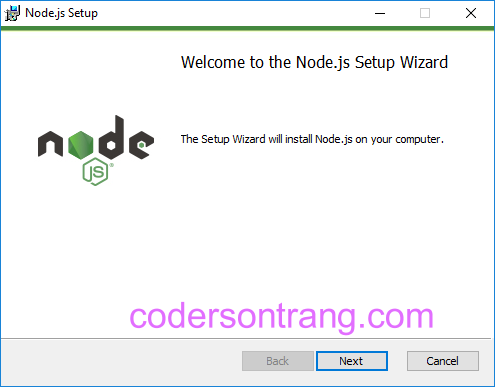
Cài đặt Node trên Window



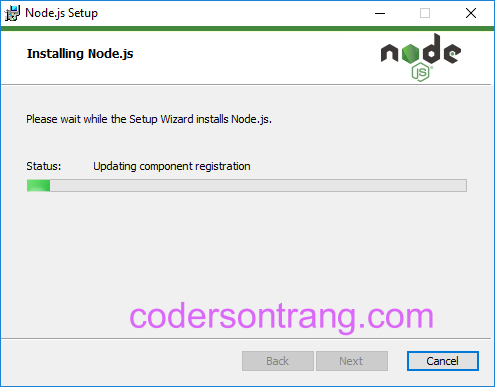
Nhìn vào hình trên các bạn sẽ thắc mắc vì sao lại có 2 mục là LTS và Current. Nhân tiện thì cũng xin giải thích luôn về sự khác biệt giữa hai phiên bản cài đặt này.

Đối với phiên bản LTS (là viết tắt của Long Term Support) là một phiên bản Node JS mà ở đó yếu tố ổn định được đặt lên hàng đầu. Phiên bản này đều có lịch cập nhật và nâng cấp rõ ràng, thêm vào nữa là các tính năng được cập nhật và nâng cấp sẽ là các tính năng được kiểm duyệt kĩ càng sau một quá trình thử nghiệm dài đã giải quyết hết các lỗi lầm trước đó. Chính vì thế mà phiên bản LTS thường được khuyến cáo dùng cho các môi trường chạy thật (production).

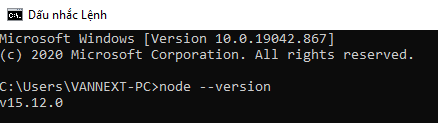
Đối với phiên bản Current, lúc nào cũng là phiên bản chứa đựng những tính năng mới nhất của Node. Phiên bản này được cập nhật thường xuyên hơn phiên bản LTS và thường được khuyến cáo dùng cho việc nghiên cứu, học tập đáp ứng mong muốn trải nghiệm công nghệ mới nhất của các lập trình viên. Nói như vậy không có nghĩa là phiên bản này không dùng được cho môi trường chạy thật, đối với các ứng dụng không quá khẩn thiết về mức độ rủi ro, thì phiên bản mới nhất của Node hoàn toàn có thể là một sự lựa chọn phù hợp.



Cứ bấm vào Next và đợi cho quá trình cài đặt hoàn thành.



Sau khi cài đặt xong thì ta có thể kiểm tra rằng việc cài đặt có thành công hay không bằng cách mở của sổ command line hay power shell sau đó thực hiện các lệnh như hình dưới đây:



Các lệnh ở hình trên được mô tả ngắn gọn như sau:

-node -v: hiển thị phiên bản của Node JS

-node -p process.versions.v8: hiển thị phiên bản của engine V8 được dùng ở trong phiên bản Node đang cài đặt

-npm -v: hiển thị phiên bản của Node Package Manager

-node: Bật chế độ Node REPL, REPL là viết tắt của Read Evaluate Print Loop, khi ở chế độ này, ta có thể viết các đoạn mã Javascript, sau đó các đoạn mã Javascript này sẽ được thực thi bởi Node rồi sau đó kết quả sẽ được in ra màn hình nếu có. Trong ví dụ này, ta nhập vào một biểu thức toán học là (1 + 2) \* 3 và nhận được kết quả trả về từ Node là 9.