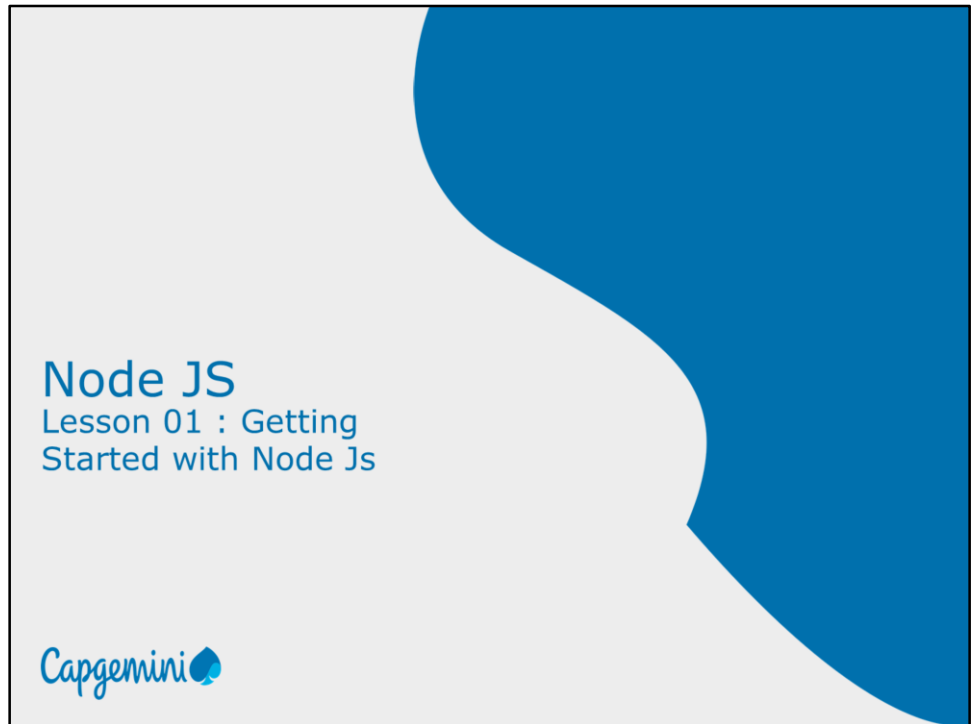


**Instructor Notes:**

Add instructor notes here.



**Instructor Notes:**

Add instructor notes here.

### Lesson Objectives

- JavaScript Essentials
- How JavaScript works
- Event loop
- Stack, Heap and Queue
- Node.js Fundamentals
- Introduction to Node.js
- Why Node.js?
- Traditional Programming Limitations
- Creating more call stacks
- Event-Driven Programming
- Node.js Official website
- Downloading and Installing Node.js
- Node.js Globals



## Instructor Notes:

Add instructor notes here.

### 1.1: JavaScript Essentials JavaScript Essentials



- JavaScript is Single Threaded.
- A JavaScript engine exists in a single OS process and consumes a single thread.
- A JavaScript engine is a program or an interpreter which executes JavaScript code. A JavaScript engine can be implemented as a standard interpreter, or just-in-time compiler that compiles
- JavaScript is getting more and more popular, teams are leveraging its support on many levels in their stack - front-end, back-end, hybrid apps, embedded devices
- When the application is running, CPU execution is never performed in parallel, since the JavaScript engine uses this method, it is impossible for users to get the Deadlocks and Race Conditions which actually makes Multi Threaded applications so complex.

July 31, 2018 Proprietary and Confidential - 3 -

If an application is built to be Multi Threaded, it will make use of several of your CPU cores simultaneously. This means it can do number of tasks in different places at the same time and we refer to this as Concurrency.

An application built in this manner can be a single process within the Operating System. The Operating System itself usually gets to choose which cores an application will run on (even which core a single threaded application will run on).

At present we have more CPU cores at the computer. In order to truly scale and use the hardware to its fullest, one needs to build applications which make use of all CPU cores.

Even though Multi threaded programming concurrently perform the execution in parallel we have some issues particularly Deadlocks and Race Condition.

One such example of these kinds of issues is that if an application is running on two separate threads, both threads reads a variable from memory at the same time, and both attempt to update the value by adding 2 to it. If the existing value is 10, and thread A adds 2, it does so by writing 12 to the memory location. If thread B also wants to add 2, it still thinks the value is 10, and writes 12. The programmer would expect it to be 14 and ends up with 12, and there are no errors. This type of bug can be very hard to track down, and the worst part is that it will happen in an unpredictable way.

JavaScript is a single threaded programming language, single threaded Runtime, it has a single call stack. And it can do one thing at a time, that's what a single thread means, the program can run one piece of code at a time.

**Instructor Notes:**

Add instructor notes here.

### 1.1: JavaScript Essentials

## How JavaScript works



List of popular projects that are implementing a JavaScript engine:

- V8—open source, developed by Google, written in C++
- Rhino—managed by the Mozilla Foundation, open source, developed entirely in Java
- SpiderMonkey—the first JavaScript engine, which back in the days powered Netscape Navigator, and today powers Firefox
- JavaScriptCore—open source, marketed as Nitro and developed by Apple for Safari
- KJS—KDE's engine originally developed by Harri Porten for the KDE project's Konqueror web browser
- Chakra (JScript9)—Internet Explorer

July 31, 2018

Proprietary and Confidential

- 4 -

## Instructor Notes:

Add instructor notes here.

1.1: JavaScript Essentials  
How JavaScript works

The diagram shows a yellow box labeled 'JS' at the top. Below it are two boxes: 'Memory Heap' on the left and 'Call Stack' on the right. The 'Memory Heap' box contains several small colored squares (blue, red, green, blue). The 'Call Stack' box contains several horizontal lines representing stack frames.

A popular example of a JavaScript Engine is Google's V8 engine. The V8 engine is used inside Chrome and Node.js

The Engine consists of two main components:

- \* Memory Heap—this is where the memory allocation happens
- \* Call Stack—this is where your stack frames are as your code executes

July 31, 2018
Proprietary and Confidential
- 5 -

V8 was first designed to increase the performance of JavaScript execution inside web browsers. In order to obtain speed, V8 translates JavaScript code into more efficient machine code instead of using an interpreter. It compiles JavaScript code into machine code at execution by implementing a **JIT (Just-In-Time) compiler** like a lot of modern JavaScript engines do such as SpiderMonkey or Rhino (Mozilla). The main difference here is that V8 doesn't produce bytecode or any intermediate code.

### V8 used to have two compilers

Before version 5.9 of V8 came out (released earlier this year), the engine used two compilers:

full-codegen—a simple and very fast compiler that produced simple and relatively slow machine code.

Crankshaft—a more complex (Just-In-Time) optimizing compiler that produced highly-optimized code.

The V8 Engine also uses several threads internally:

The main thread does what you would expect: fetch your code, compile it and then execute it

There's also a separate thread for compiling, so that the main thread can keep executing while the former is optimizing the code

A Profiler thread that will tell the runtime on which methods we spend a lot of time so that Crankshaft can optimize them

A few threads to handle Garbage Collector sweeps


When first executing the JavaScript code, V8 leverages **full-codegen** which directly translates the parsed JavaScript into machine code without any transformation. This allows it to start executing machine code **very fast**. Note that V8 does not use intermediate bytecode representation this way removing the need for an interpreter.

When your code has run for some time, the profiler thread has gathered enough data to tell which method should be optimized.

**Instructor Notes:**

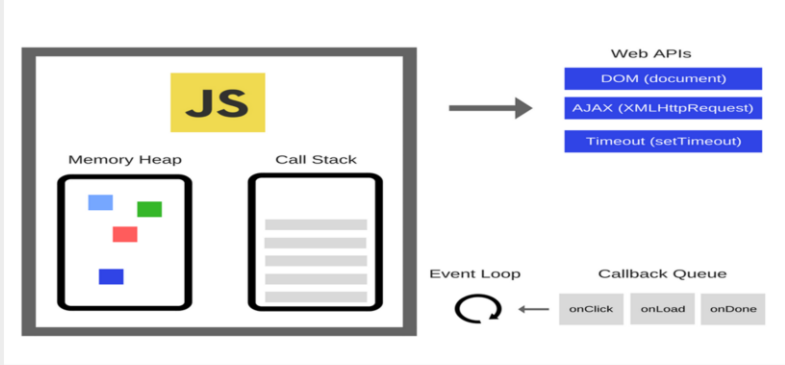
Add instructor notes here.

1.1: JavaScript Essentials  
How JavaScript works



➤The Runtime

- There are APIs in the browser that have been used by almost any JavaScript developer out there
- Those APIs, however, are not provided by the Engine.



The diagram illustrates the JavaScript runtime environment. A central box labeled 'JS' contains two sub-components: 'Memory Heap' (depicted as a box with four colored squares) and 'Call Stack' (depicted as a box with horizontal lines). An arrow points from the 'JS' box to a 'Web APIs' section, which lists 'DOM (document)', 'AJAX (XMLHttpRequest)', and 'Timeout (setTimeout)'. Below this, an 'Event Loop' is shown as a circular arrow, and a 'Callback Queue' is shown as a box containing 'onClick', 'onLoad', and 'onDone'. An arrow points from the 'Callback Queue' to the 'Event Loop'.

July 31, 2018

Proprietary and Confidential

- 6 -

In the stack when we step into a function, we put something on to the stack, if we return from a function, we pop off the top of the stack.

DOM, Ajax, timeout are the APIs provided to us by the browser, it doesn't live in the V8 source. Web API can't just start modifying your code, it can't chuck stuff onto the stack when it's ready. If it did, it would appear randomly in the middle of your code so this is where the task queue or callback queue kicks in. Web APIs pushes the callback on to the task queue when it's done.

The event loops job is to look at the stack and look at the task queue. If the stack is empty it takes the first thing on the queue and pushes it on to the stack which effectively run it.

## Instructor Notes:

Add instructor notes here.

1.1: JavaScript Essentials

Call stack

- JavaScript is a single-threaded programming language, which means it has a single Call Stack.
- The Call Stack is a data structure which records basically where in the program we are. If we step into a function, we put it on the top of the stack. If we return from a function, we pop off the top of the stack.

```
function multiply(x, y) {return x * y;}  
function printSquare(x) {  
  var s = multiply(x, x);  
  console.log(s);  
  printSquare(5);  
}
```

Call Stack

Step 1

Step 2

Step 3

Step 4

Step 5

Each entry in the Call Stack is called a **Stack Frame**.

And this is exactly how stack traces are being constructed when an exception is being thrown—it is basically the state of the Call Stack when the exception happened.

**“Blowing the stack”**—this happens when you reach the maximum Call Stack size. And that could happen quite easily, especially if you’re using recursion without testing your code very extensively.

## Instructor Notes:

Add instructor notes here.

### 1.1: JavaScript Essentials Stack, Heap and Queue



- JavaScript engine has three important features. They are Stack, Heap and Queue
- Each browser will implement these features differently, but all does the same.
- Stack
  - Currently running functions gets added to the stack(frame). Pops out from the once it completes its execution.
- Heap
  - Memory allocation happened here. It is a bunch of memory where object's live in a unordered manner.
- Queue
  - function calls are queued up which gets added to the stack once it is empty.

July 31, 2018

Proprietary and Confidential

- 8 -

**Heap:** This is a bunch of memory where your objects live (e.g. variables and functions and all those things you instantiate). It is referred as Chaotic, only because the order doesn't really matter and there's no guarantee with how they will live. In this heap, different browsers will perform different optimizations, e.g., if an object is duplicated many times, it may only exist in memory once, until a change needs to happen, at which point the object is copied.

**Stack:** This is where the currently running functions get added. If function A() runs function B(), well you're two levels deep in the stack. Each time one of these functions is added to the stack, it is called a frame. These frames contain pointers to the functions in the heap, as well as the objects available to the function depending on its current scope, and of course the arguments to the function itself. Different JavaScript engines likely have different maximum stack sizes, and unless you have a runaway recursive function, you've probably never hit this limit. Once a function call is complete, it gets removed from the stack. Once the stack is empty, we're ready for the next item in the Queue.

**Queue:** This is where function calls which are queued up for the future go. If you perform a `setTimeout(function() { console.log('hi'); }, 10);`, that anonymous function is living in the next available queue slot. No items in the queue will be run until the current stack is complete. So, if you have some work that might be slow that you want to run after you get your data, try a `setTimeout()` with a delay of zero milliseconds.

**Garbage Collection:** JavaScript will keep an eye on the current stack and the items in the Queue, and see what objects in the Heap are being pointed to. If an object no longer has pointers to it, it is safe to assume that object can be thrown away.



## Instructor Notes:

Add instructor notes here.

### 1.1: JavaScript Essentials Event loop



➤An event loop is a construct that mainly performs two functions in a continuous loop

- **Event detection** : In any run of the loop, it has to detect which events just happened.
- **Event handler triggering** : When an event happens, the event loop must determine the event callback and invoke it.

➤Event loop is just one thread running inside one process, which means that, when an event happens, the event handler can run without interruption

- There is at most one event handler running at any given time
- Any event handler will run to completion without being interrupted

➤This allows the programmer to relax the synchronization requirements and not have to worry about concurrent threads of execution changing the shared memory state.

July 31, 2018

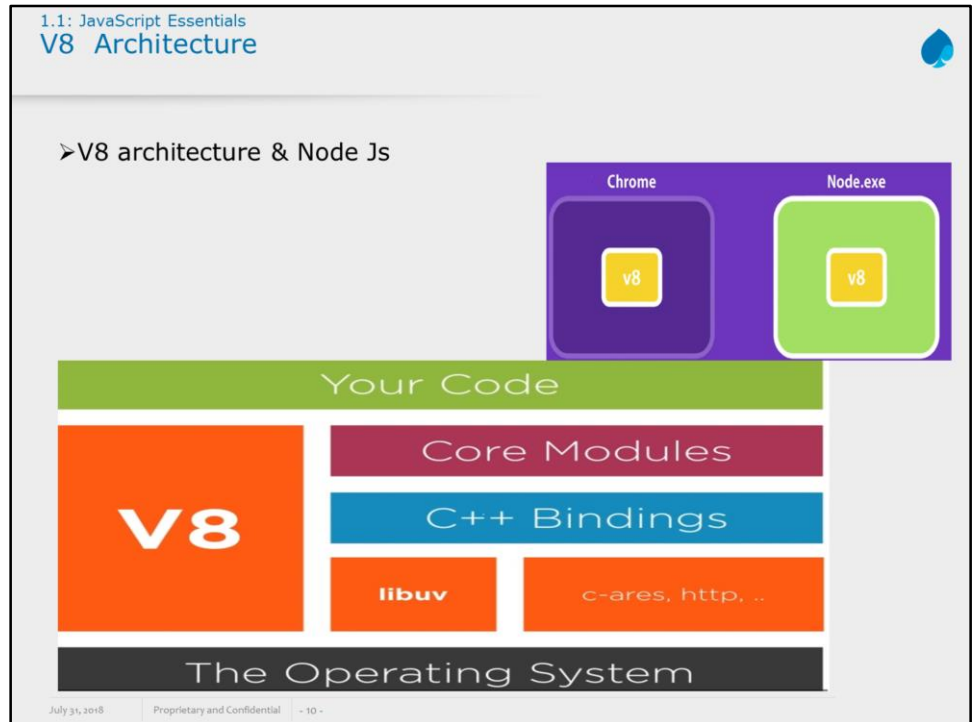
Proprietary and Confidential

- 9 -

One of the key concepts "Node" brings from the browser to "JavaScript" on the server is the "Event Loop". In the browser, the "Event Loop" is constantly listening for DOM events, so just key presses or mouse clicks. Similarly, Node's "Event Loop" is constantly listening for events on the server side. These events can be externally generated, such as incoming HTTP requests or TCP connections, or they can be timers and other internal events generated by your "Node" application

**Instructor Notes:**

Add instructor notes here.



**V8** - Google's open source JavaScript engine that resides in Chrome/Chromium browsers. Instead of interpreting JavaScript code on the fly like typical web browsers do, V8 translates your JS code into machine code so that it's blazing fast. V8 is written in C++. Read more about how V8 works [here](#).

**libuv** - libuv is originally developed to provide asynchronous I/O that includes asynchronous TCP & UDP sockets, (famous) event loop, asynchronous DNS resolution, file system read/write, and etc. libuv is written in C. Here is a good video to check out to learn more about libuv.

**Other Low-Level Components** - such as c-ares, http parser, OpenSSL, zlib, and etc, mostly written in C/C++.

**Application** - here is our code, modules, and Node.js' built in modules, written in JavaScript (or compiled to JS through TypeScript, CoffeeScript, etc.)

**Binding** - a binding basically is a wrapper around a library written in one language and expose the library to codes written in another language so that codes written in different languages can communicate.

**Instructor Notes:**

1.1: JavaScript Essentials

**How to write Javascript Optimized code?****How to write optimized JavaScript**

- Order of object properties: always instantiate our object properties in the same order so that hidden classes, and subsequently optimized code, can be shared.
- Dynamic properties: adding properties to an object after instantiation will force a hidden class change and slow down any methods that were optimized for the previous hidden class. Instead, assign all of an object's properties in its constructor.
- Methods: code that executes the same method repeatedly will run faster than code that executes many different methods only once (due to inline caching).
- Arrays: avoid sparse arrays where keys are not incremental numbers. Sparse arrays which don't have every element inside them are a hash table. Elements in such arrays are more expensive to access.
- Tagged values: V8 represents objects and numbers with 32 bits.

July 31, 2018

Proprietary and Confidential

- 11 -

**Instructor Notes:**

Demo

EventLoop



July 31, 2018

Proprietary and Confidential

- 12 -

**Instructor Notes:**

Add instructor notes here.

1.2: Node.js Fundamentals  
Introduction to Node.js



- Node.js is an open source, real time communication, cross platform, server side language which is written by Javascript on Google's v8 Javascript engine.
- In 2009 Ryan Dahl created Node.js or Node, a framework primarily used to create highly scalable servers for web applications. It is written in C++ and JavaScript.
- It's a highly scalable system that uses asynchronous, non-blocking I/O model (input/output), rather than threads or separate processes
- It is not a framework like jQuery nor a programming language like C# or JAVA . It's a new kind of web server like has a lot in common with other popular web servers, like Microsoft's Internet Information Services (IIS) or Apache
- IIS / Apache processes only HTTP requests, leaving application logic to be implemented in a language such as PHP or Java or ASP.NET. Node removes a layer of complexity by combining server and application logic in one place.

July 31, 2018

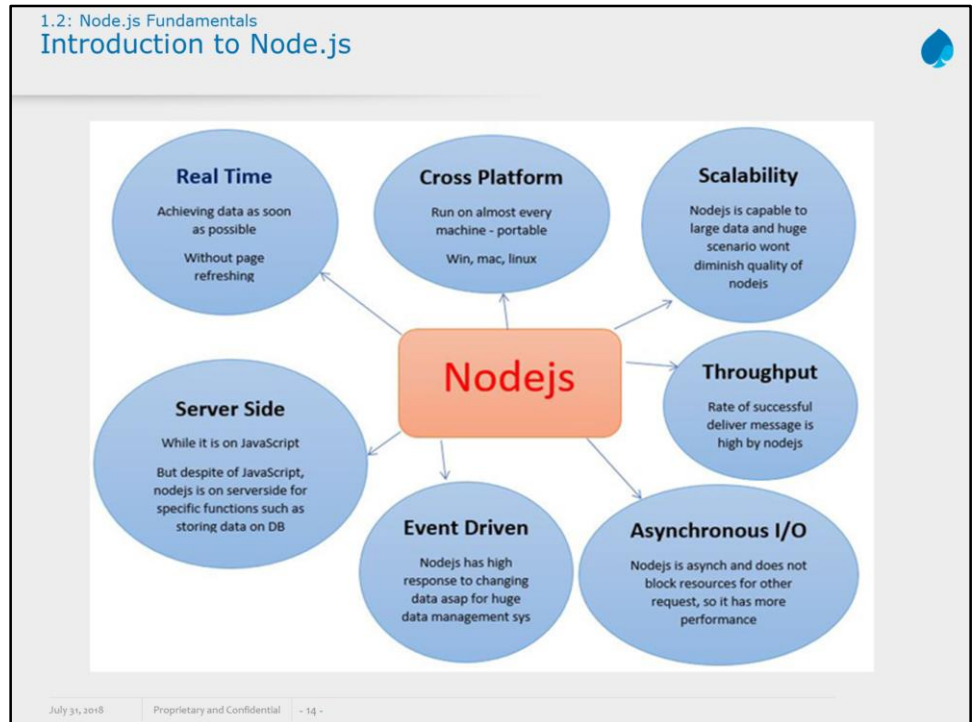
Proprietary and Confidential

- 13 -

Node.js is ideal for web applications that are frequently accessed but computationally simple

## Instructor Notes:

Add instructor notes here.



### Real Time Communication

Real time web solutions are web technologies which enable clients to achieve and receive information as soon as they are published by their authors instead of checking by software or refreshing pages by users manually whether there is new data to expose or not.

### Cross Platform

Cross platform means that Nodejs is almost independent from platform and is enabled to be installed on every machine with different operating systems such as Windows, Linux, Macintosh. So Nodejs does not need specific preparation and it is portable.

### Server Side Application

There are situations where it is not possible to give all responsibilities to the client due to the fact that the client does not have the ability to accept them. For instance, if we want to store data on a database or do some special processing on a program we need to put these kind of functionalities on the server which is remote from the client and the client can try to achieve their purpose.

### Event Driven Architecture

If we are working with some data management systems which have many nodes with different statuses, event driven enables us to find out other statuses very quickly so we are able to respond properly. Think about real estate trading, where houses are for sale or rent and customers want to be aware which house is still available. As soon as a house is selected and is sold its status should be changed for others.

### Asynchronous I/O

There are two approaches for I/O, the simple way is synchronous which blocks resources and progress until communication is completed which causes waiting and wastes a lot of resources especially if we have too many I/O.

Another way is asynchronous, which allows critical operations to do their job while waiting for I/O

### Throughput

In such a scenario which we need to send and receive messages, there are possibilities that some messages cannot reach the stations so the rate of successful messages which are received correctly on one channel is called throughput.

### Scalability

Scalability is a capability of an application that can grow by encountering large scenarios such as large amounts of data or nodes. So unexpected increase of quantity will not decrease quality of system.

**Instructor Notes:**

Add instructor notes here.

## 1.2: Node.js Fundamentals

### Why Node.js?



- JavaScript everywhere i.e. Server-side and Client-side applications in JavaScript.
- Node is very easy to set up and configure.
- Vibrant Community
- Small core but large community so far we have 60,000 + packages on npm
- Real-time/ high concurrency apps (I/O bound)
- API tier for single-page apps and rich clients(iOS, Android)
- Service orchestration
- Top corporate sponsors like Microsoft, Joyent, PayPal etc..
- Working with NOSQL(MongoDB) Databases

July 31, 2018

Proprietary and Confidential

- 15 -

**Instructor Notes:**

Add instructor notes here.

1.2: Node.js Fundamentals

**Traditional Programming Limitations**

- In traditional programming I/O (database, network, file or inter-process communication) is performed in the same way as it does local function calls. i.e. Processing cannot continue until the operation is completed.
- When the operation like executing a query against database is being executed, the whole process/thread idles, waiting for the response. This is termed as "Blocking"
- Due to this blocking behavior we cannot perform another I/O operation, the call stack becomes frozen waiting for the response.
- We can overcome this issue by creating more call stacks or by using event callbacks.

July 31, 2018

Proprietary and Confidential

- 16 -



**Instructor Notes:**

Add instructor notes here.

### 1.2: Node.js Fundamentals Creating more call stacks



- To handle more concurrent I/O, we need to have more concurrent call stacks.
- Multi-threading is one alternative to this programming model.
  - Makes use of separate CPU Cores as "Threads"
  - Uses a single process within the Operating System
  - Ran out of Ghz, hardware adds more cores
- If the application relies heavily on a shared state between threads accessing and modifying shared state increase the complexity of the code and It can be very difficult to configure, understand and debug.

July 31, 2018 | Proprietary and Confidential | - 17 -

A thread is a kind of lightweight process that shares memory with every other thread within the same process. Threads were created as an ad hoc extension of the previous model to accommodate several concurrent threads of execution. When one thread is waiting for an I/O operation, another thread can take over the CPU. When the I/O operation finishes, that thread can wake up, which means the thread that was running can be interrupted and eventually be resumed later. Furthermore, some systems allow threads to execute in parallel in different CPU cores

**Instructor Notes:**

Add instructor notes here.




Instructor Notes:

Add instructor notes here.


1.2: Node.js Fundamentals

Downloading Node.js




Windows Installer

node-v0.12.0-x86.msi



Macintosh Installer

node-v0.12.0.pkg



Source Code

node-v0.12.0.tar.gz

Windows Installer (.msi)	32-bit	64-bit
Windows Binary (.exe)	32-bit	64-bit
Mac OS X Installer (.pkg)	Universal	
Mac OS X Binaries (.tar.gz)	32-bit	64-bit
Linux Binaries (.tar.gz)	32-bit	64-bit
SunOS Binaries (.tar.gz)	32-bit	64-bit
Source Code	node-v0.12.0.tar.gz	

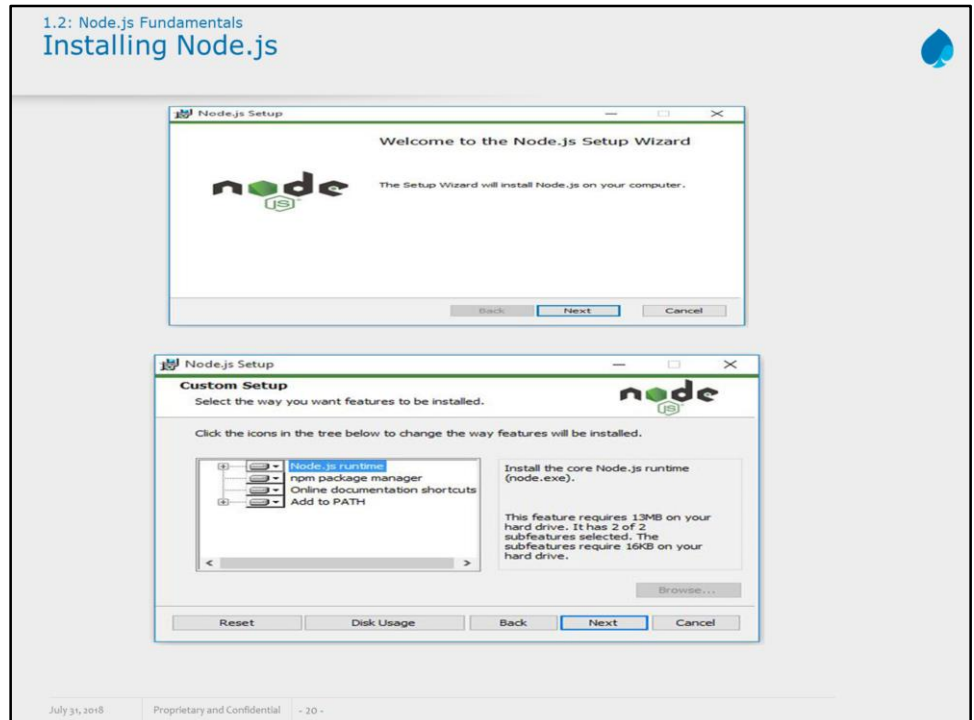
July 31, 2018

Proprietary and Confidential

- 19 -

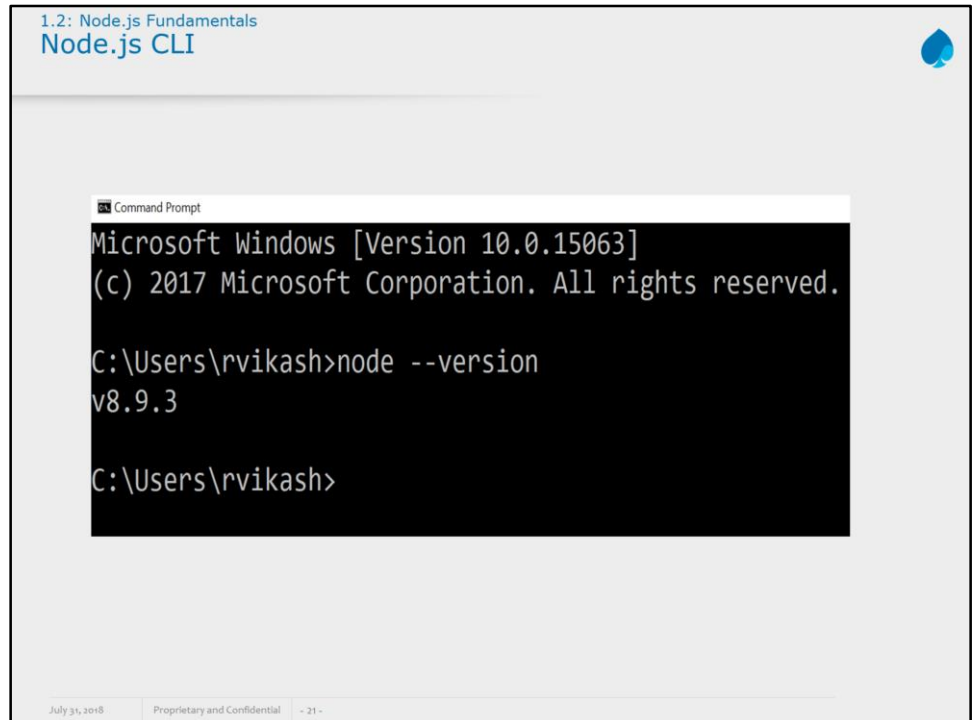
Page 01 - 19

## Instructor Notes:



**Instructor Notes:**

Add instructor notes here.

**Environment Variables**

**NODE\_PATH** : C:\Users\<username>\AppData\Roaming\npm\node\_modules

**PATH** : C:\Users\<username>\AppData\Roaming\npm

Variable	Value
NODE_PATH	C:\Users\714709\AppData\Roaming\npm\node_modules
PATH	C:\Users\714709\AppData\Roaming\npm

**Instructor Notes:**

Add instructor notes here.

## 1.2: Node.js Fundamentals Node.js Globals



### ➤ global

- Any variables or members attached to global are available anywhere in our application. GLOBAL is an alias for global
  - `global.companyName = 'IGATE'`
  - `global['companyName']` // We can directly access the members attached to global.

### ➤ process

- The process object is a global object which is used to Inquire the current process to know the PID, environment variables, platform, memory usage etc.
  - `process.platform`
  - `process.exit( )`

### ➤ console

- It provides two primary functions for developers testing web pages and applications
  - `console.log('IGATE')`

July 31, 2018


Proprietary and Confidential

- 22 -

Instructor Notes:

Demo

Basic



July 31, 2018

Proprietary and Confidential


- 23 -

**Instructor Notes:**

Add instructor notes here.

Lab

Lab 1.1



July 31, 2018

Proprietary and Confidential

- 24 -

Add the notes here.