



Lecture 09



presentation

DAD – Distributed Applications Development

Cristian Toma

D.I.C.E/D.E.I.C – Department of Economic Informatics & Cybernetics

www.dice.ase.ro



Cristian Toma – Business Card



Cristian Toma

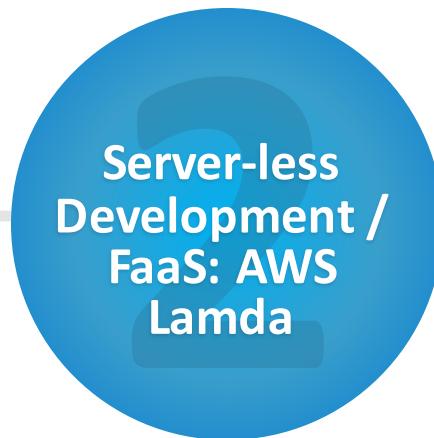
IT&C Security Master

Dorobantilor Ave., No. 15-17
010572 Bucharest - Romania

<http://ism.ase.ro>
cristian.toma@ie.ase.ro
T +40 21 319 19 00 - 310
F +40 21 319 19 00



Agenda for Lecture 09





Java Script: CallBack Functions and Async Programming, Buffers, Promise, Event-Emitter, node.js modules

ECMAScript/node.js Overview

1. node.js Intro / RECAP



About node.js ...

- Created by Ryan Dahl in 2009
- Development && maintenance sponsored by Joyent
- License: MIT
- Latest LTS Version (Y2020): 12.16.1 (includes npm 6.13.4)
- The syntax is compliant with Java Script / ECMAScript 2015+
- Based on Google V8 Engine
- +100 000 packages
- it has interpreter on x86, ARM, ESP8266, etc. controllers, on various OS-es: MacOS, Linux, Windows, Linux Embedded, etc.

<https://nodejs.org> | <https://nodejs.org/en/download/>

1. node.js Intro / RECAP

Other projects as complementary to node.js ...

- Vert.x => Polygot programming
- Akka => Scala and Java/Kotlin
- Tornado => Python
- Libevent => C
- EventMachine => Ruby

Node.js is not...

- Another Web framework
- For beginners
- Supporting Multi-thread

1. node.js Intro / RECAP

Development of Server-side for the Web & others ...

Traditional desktop applications have a user interface wired up with background logic

- user interfaces are based on Windows Forms, Jswing, WPF, Gtk, Qt, etc. and dependant on operating system
- On the web user interfaces are standardized
 - HTML for markup
 - CSS for style
 - JavaScript for dynamic content
- In the past proprietary technologies existed on the web, e.g. Adobe Flash, ActiveX: They are slowly dying ...

Server Side technologies include: **Java Servlet/JSP**, PHP, ASP .NET, Rails, Django, and: **node.js**

1. node.js Intro / RECAP

Why node.js?

- Non Blocking I/O
- V8 Java-Script Engine
- Single Thread with Event Loop
- 40,000+ modules
- 1 Language for Frontend and Backend
- Active community

1. node.js Intro / RECAP

What is node.js?

- Asynchronous I/O framework
- Core in C++ on top of v8
- Rest of it, in JavaScript / ECMAScript
- Swiss army knife for network Related stuffs
- Can handle thousands of Concurrent connections with Minimal overhead (CPU/Memory) on a single process
- It's NOT a web framework, and it's also NOT a language
- It is a development platform basing on interpreting JavaScript / ECMAScript code in native C/C++

« A platform built on Chrome's
JavaScript runtime for easily
building fast, scalable network
applications. » <http://nodejs.org/>

1. node.js Intro / RECAP

Why JavaScript/ECMAScript?

- Friendly call-backs
- Ubiquitous
- No I/o Primitives
- One language to RULE them all

JavaScript is well known for client-side scripts running inside the browser

node.js is JavaScript running on the server-side

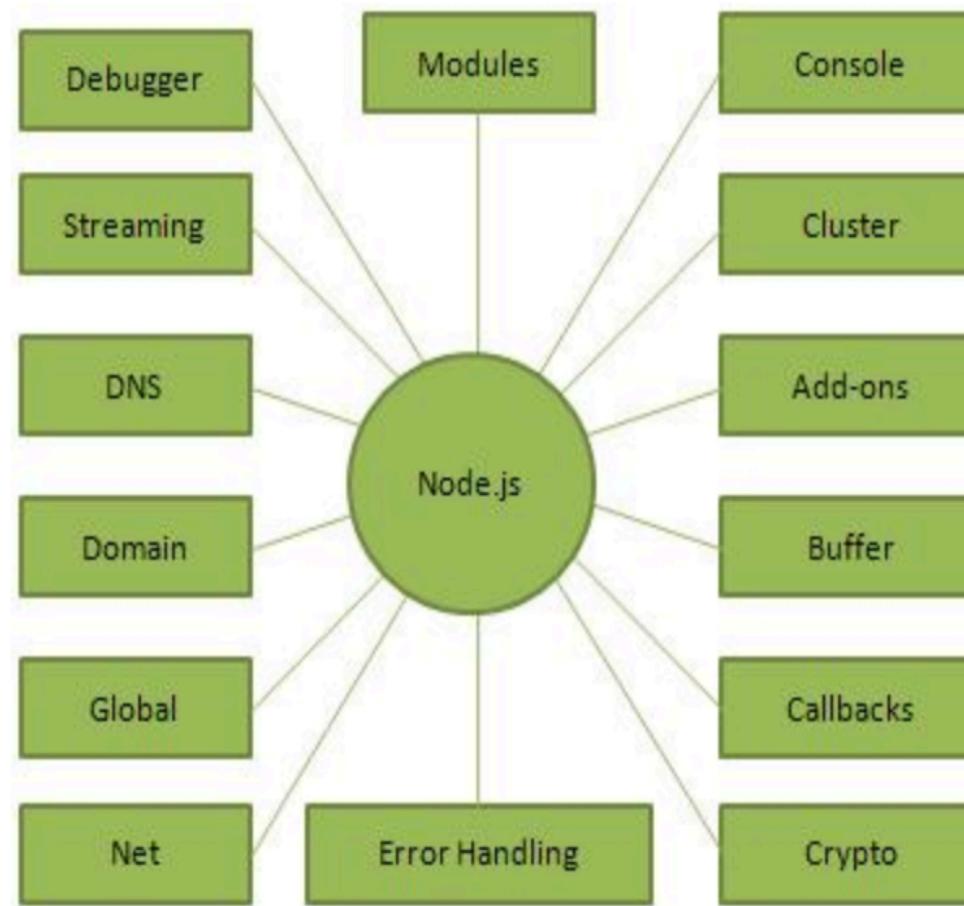
SSJS -> Server-Side JavaScript

Use a language you know

Use the same language for client side and server side

Concepts

The following diagram depicts some important parts of Node.js which we will discuss in detail in the subsequent chapters.



JavaScript Engines.....

Every browser has **its own VM**

Firefox? **Spidermonkey**

Internet Explorer? **Chakra**

Chrome? **V8**

Safari? **JavaScriptCore**

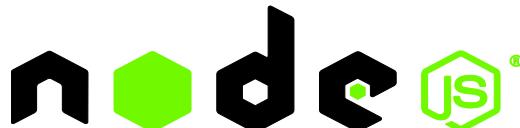
Opera? **Carakan**

Also **Rhino**, stand alone



- In simple words Node.js is '**server-side JavaScript**'.
- In *not-so-simple* words Node.js is a high-performance **network applications framework**, well optimized for high concurrent environments.
- It's a **command line** tool.
- In 'Node.js' , '**.js**' doesn't mean that its solely written JavaScript. It is 40% JS and 60% C++.
- From the official site:

'Node's goal is to provide an easy way to build scalable network programs' - (from nodejs.org!)



INTRODUCTION: ADVANCED (& CONFUSING)

- Node.js uses an **event-driven, non-blocking I/O** model, which makes it lightweight. ([from nodejs.org!](#))
- It makes use of **event-loops** via JavaScript's **callback** functionality to implement the non-blocking I/O.
- Programs for Node.js are written in JavaScript but not in the same JavaScript we are used to. There is no DOM implementation provided by Node.js, i.e. you **can not** do this:

```
var element = document.getElementById("elementId");
```

- Everything inside Node.js runs in a **single-thread**.



EXAMPLE-1: GETTING STARTED & HELLO WORLD

- Install/build Node.js.
 - (Yes! Windows installer is available!)
- Open your favorite editor and start typing JavaScript.
- When you are done, open cmd/terminal and type this:
`'node YOUR_FILE.js'`
- Here is a simple example, which prints '*hello world*'

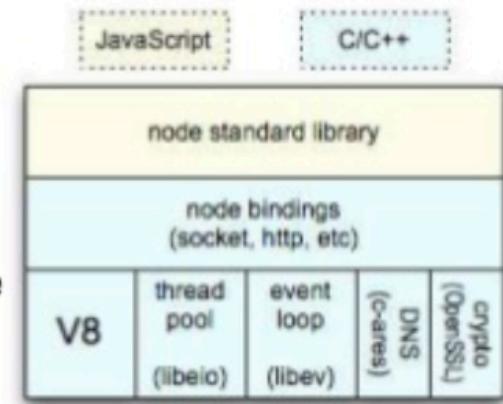
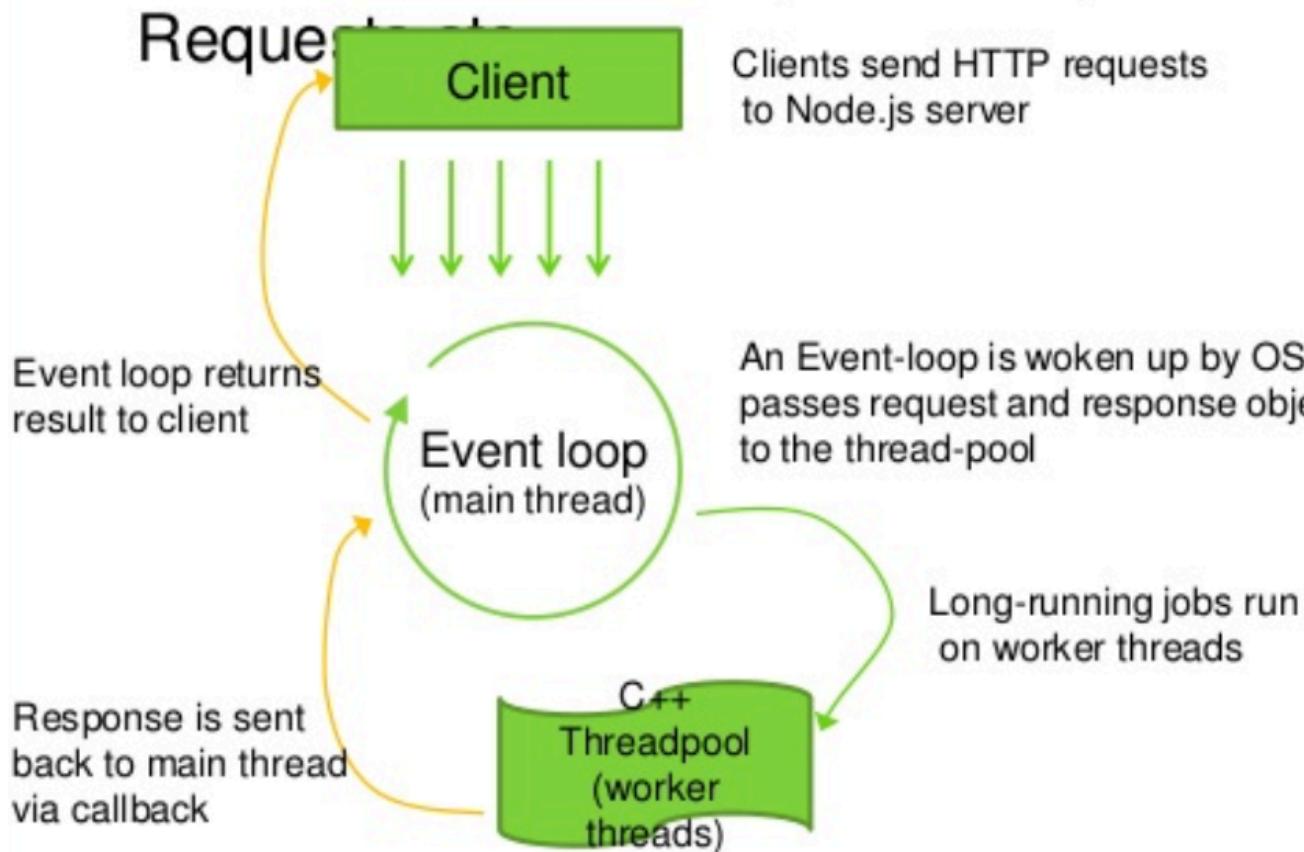
```
var sys = require("sys");
setTimeout(function(){
  sys.puts("world");},3000);
sys.puts("hello");

//it prints 'hello' first and waits for 3 seconds and then
  prints 'world'
```



SOME THEORY: EVENT-LOOPS

- Event-loops are the core of event-driven programming, almost all the UI programs use event-loops to track the user event, for example: Clicks, Ajax

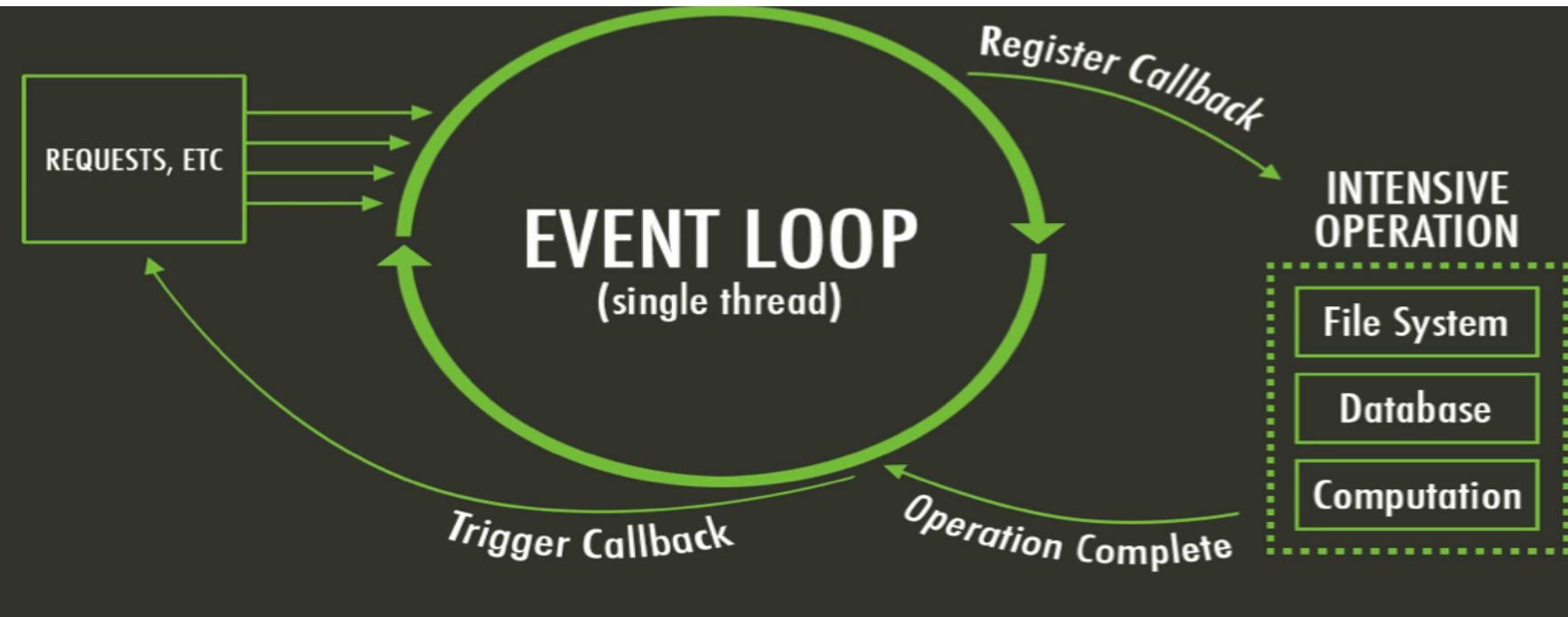


The idea behind node.js....

- Perform asynchronous processing on single thread instead of classical multithread processing, minimize overhead & latency, maximize scalability
- Scale horizontally instead of vertically
- Ideal for applications that serve a lot of requests but dont use/need lots of computational power per request
- Not so ideal for heavy calculations, e.g. massive parallel computing
- Also: Less problems with concurrency



Node.js Event Loop



There are a couple of implications of this apparently very simple and basic model

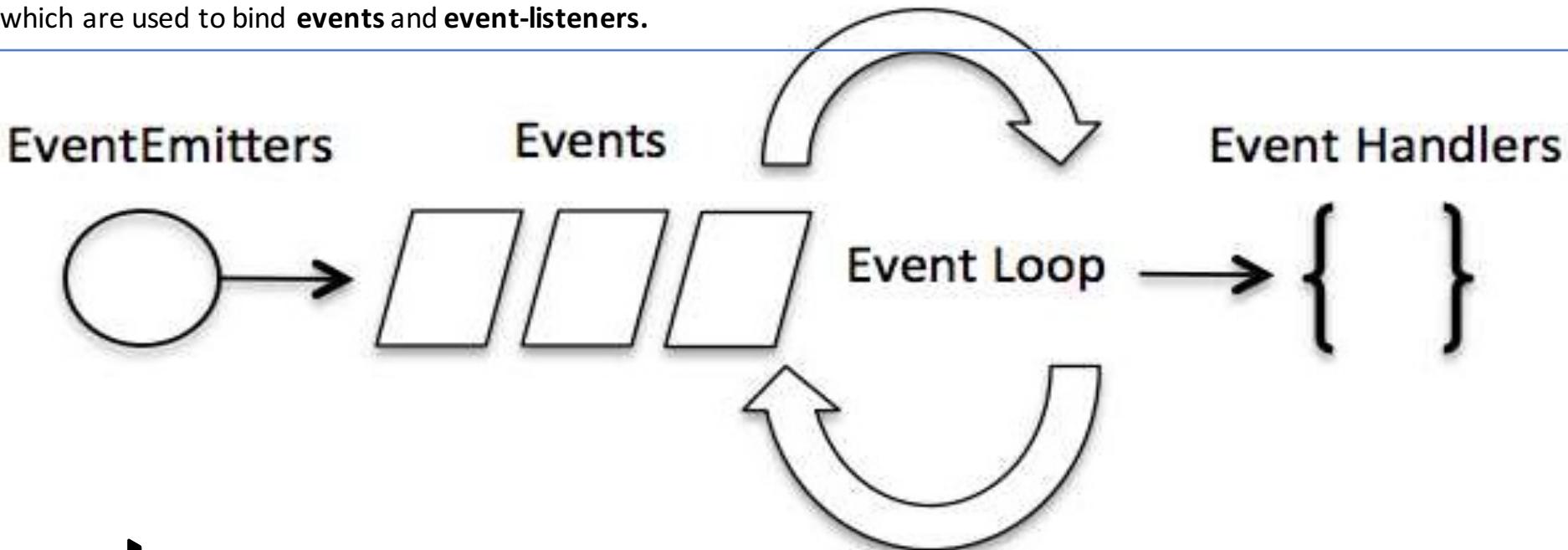
- Avoid synchronous code at all costs because it blocks the event loop
- Which means: callbacks, callbacks, and more callbacks

Node.js is a **single-threaded application**, but it can support concurrency via the concept of **event** and **callbacks**. Every API of Node.js is asynchronous and being single-threaded, they use **async function calls** to maintain concurrency. Node uses observer pattern. Node thread keeps an event loop and whenever a task gets completed, it fires the corresponding event which signals the event-listener function to execute.

Event-Driven Programming

Node.js uses events heavily and it is also one of the reasons why **Node.js** is pretty fast compared to other similar technologies. As soon as **Node** starts its server, it simply initiates its variables, declares functions and then simply waits for the event to occur.

In an event-driven application, there is generally a main loop that listens for events, and then triggers a callback function when one of those events is detected. Although events look quite similar to callbacks, the difference lies in the fact that callback functions are called when an asynchronous function returns its result, whereas event handling works on the observer pattern. The functions that listen to events act as **Observers**. Whenever an event gets fired, its listener function starts executing. Node.js has multiple in-built events available through events module and **EventEmitter** class which are used to bind **events** and **event-listeners**.





Node.js Event driven Programming (Node v6.11.2)

Execute

Share

main.js

STDIN

Result

```
1 // Import events module
2 var events = require('events');
3
4 // Create an eventEmitter object
5 var eventEmitter = new events.EventEmitter();
6
7 // Create an event handler as follows
8 var connectHandler = function connected() {
9     console.log('connection succesful.');
10
11     // Fire the data_received event
12     eventEmitter.emit('data_received');
13 }
14
15 // Bind the connection event with the handler
16 eventEmitter.on('connection', connectHandler);
17
18 // Bind the data_received event with the anonymous function
19 eventEmitter.on('data_received', function(){
20     console.log('data received succesfully.');
21 });
22
23 // Fire the connection event
24 eventEmitter.emit('connection');
25
26 console.log("Program Ended.");
```

```
$node main.js
connection succesful.
data received succesfully.
Program Ended.
```

Blocking vs Non-Blocking.....

Example :: Read data from file and show data

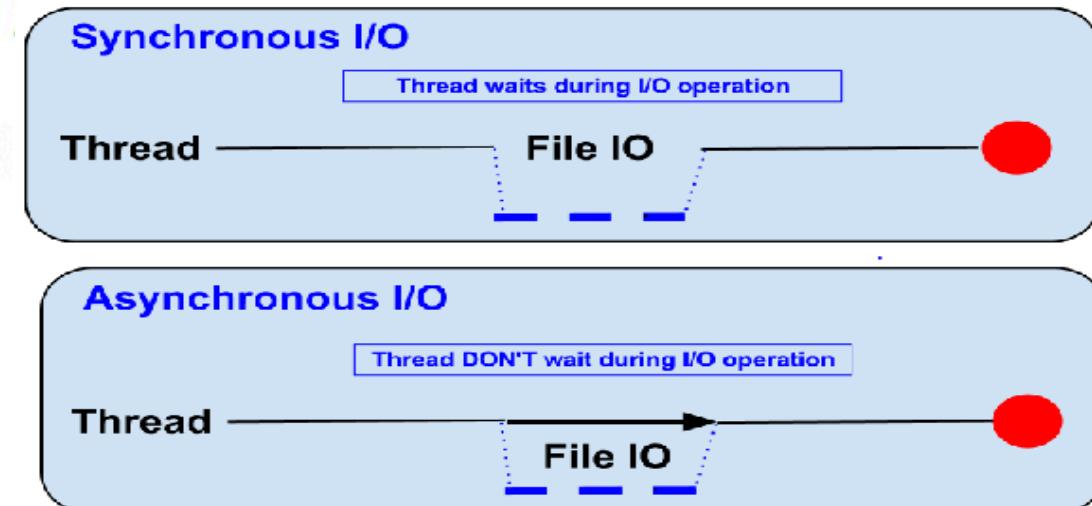
SOME THEORY: NON-BLOCKING I/O

- Traditional I/O

```
var result = db.query("select x from table_Y");
doSomethingWith(result); //wait for result!
doSomethingWithoutResult(); //execution is blocked!
```

- Non-traditional, Non-blocking I/O

```
db.query("select x from table_Y",function (result){
    doSomethingWith(result);
});
doSomethingWithoutResult();
delay!
```



Node.js for....

- Web application
- Web-socket server
- Ad server
- Proxy server
- Streaming server
- Fast file upload client
- Any Real-time data apps
- Anything with high I/O

Where to Use Node.js?

- Following are the areas where Node.js is proving itself as a perfect technology partner.
- I/O bound Applications
- Data Streaming Applications
- Data Intensive Real-time Applications (DIRT)
- JSON APIs based Applications
- Single Page Applications



In NodeJS

Standard JavaScript with

- Buffer
- C/C++ Addons
- Child Processes
- Cluster
- Console
- Crypto
- Debugger
- DNS
- Domain
- Events
- File System
- Globals
- [HTTP](#)
- [HTTPS](#)
- [Modules](#)
- [Net](#)
- [OS](#)
- [Path](#)
- [Process](#)
- [Punycode](#)
- [Query Strings](#)
- [Readline](#)
- [REPL](#)
- [Stream](#)
- String Decoder
- Timers
- TLS/SSL
- TTY
- UDP/Datagram
- URL
- Utilities
- VM
- ZLIB

... but without DOM manipulation



WHEN TO NOT USE NODE.JS

- When you are doing heavy and CPU intensive calculations on server side, because event-loops are CPU hungry.
- Node.js API is still in beta, it keeps on changing a lot from one revision to another and there is a very little backward compatibility. Most of the packages are also unstable. Therefore is not yet production ready.
- Node.js is a no match for enterprise level application frameworks like Spring(java), Django(python), Symfony/php) etc. Applications written on such platforms are meant to be highly user interactive and involve complex business logic.



- Read the free O'reilly Book '*Up and Running with Node.js*' @
<http://ofps.oreilly.com/titles/9781449398583/>
- Visit www.nodejs.org for Info/News about Node.js
- Watch Node.js tutorials @ <http://nodetuts.com/>
- For Info on MongoDB:
<http://www.mongodb.org/display/DOCS/Home>
- For anything else **Google!**

Section Conclusion

Fact: **node.js is used in DAD**

In few **samples** it is simple to remember:
ECMAScript / JavaScript RECAP and node.js best
practice for DAD / REST Services development and
FaaS – Function as a Service implementation





FaaS: Function as a Service – Amazon AWS Lambda example

Server-less Development – FaaS: AWS Lambda

2. AWS Lambda – FaaS Cloud

<https://docs.aws.amazon.com/lambda/latest/dg/getting-started-create-function.html>

The screenshot shows a web browser displaying the AWS Lambda Developer Guide. The left sidebar contains a navigation menu with sections like 'Getting Started' (which is expanded), 'Create a Function' (highlighted in orange), and other topics such as 'Code Editor', 'AWS CLI', 'Concepts', 'Features', 'Tools', 'Limits', and several collapsed sections under 'Permissions', 'Managing Functions', 'Invoking Functions', and 'Lambda Runtimes'. The main content area has a title 'Create a Lambda Function with the Console' and a sub-section 'To create a Lambda function' with a numbered list of four steps. Step 3 highlights the 'Function name' field with the value 'my-function'. Below the list, a note explains that Lambda creates a Node.js function and an execution role. At the bottom of the page, there are links to 'nodejs-handler.html' and 'getting-started-create-function.html'.

Create a Lambda Function with the Console

PDF | Kindle | RSS

In this Getting Started exercise you create a Lambda function using the AWS Lambda console. Next, you manually invoke the Lambda function using sample event data. AWS Lambda executes the Lambda function and returns results. You then verify execution results, including the logs that your Lambda function created and various CloudWatch metrics.

To create a Lambda function

1. Open the [AWS Lambda console](#).
2. Choose **Create a function**.
3. For **Function name**, enter **my-function**.
4. Choose **Create function**.

Lambda creates a Node.js function and an execution role that grants the function permission to upload logs. Lambda assumes the execution role when you invoke your function, and uses it to create credentials for the AWS SDK and to read data from event sources.

2. AWS Lambda – FaaS Cloud

us-east-2.console.aws.amazon.com/lambda/home?region=us-east-2#/functions/my-function?tab=configuration

Apps Bookmarks Mozilla Firefox Oracle / Sun Java / Android / O... Java / Kotlin / And... Python JavaScript - Node... Objective-C / Swift Cloud (AWS EC2 I... REST IoT / Embedded Other B

Lambda > Functions > my-function ARN - arn:aws:lambda:us-east-2:235126029635:function:my-function

my-function

Throttle Qualifiers Actions MyEventName ▾ Test Save

Configuration Permissions Monitoring

▼ Designer

my-function

Layers (0)

API Gateway X + Add destination

+ Add trigger

The screenshot shows the AWS Lambda function configuration page for 'my-function'. The top navigation bar includes links for Services, Resource Groups, and various programming languages. The main header displays the function name and ARN. Below the header, there are tabs for Configuration, Permissions, and Monitoring, with 'Configuration' currently selected. The 'Designer' section is expanded, showing the function's name, a 'Layers' section with '(0)', and a 'Destinations' section with an 'API Gateway' entry and a '+ Add destination' button. A 'Triggers' section at the bottom has a '+ Add trigger' button. The URL in the browser is https://docs.aws.amazon.com/lambda/latest/dg/nodejs-handler.html | https://docs.aws.amazon.com/lambda/latest/dg/getting-started-create-function.html.

2. AWS Lambda – FaaS Cloud

The screenshot shows the AWS Lambda console interface for a function named "my-function".

Code entry type: Edit code inline

Runtime: Node.js 12.x

Handler: index.handler

Function code:

```
// TODO implement
const response = {
  statusCode: 200,
  body: JSON.stringify('Hello')
};
```

Execution Result: No execution results yet.

2. AWS Lambda – FaaS Cloud

1. go to <https://console.aws.amazon.com/apigateway>

2. select api link (which you have deployed on aws lambda).

The screenshot shows the AWS API Gateway console interface. At the top, there is a navigation bar with the AWS logo, 'Services' dropdown, 'Resource Groups' dropdown, a bell icon, 'N. Virginia' region selector, and 'Support' dropdown. Below the navigation bar, the main header has 'Amazon API Gateway' and 'APIs' tabs, with 'Show all hints' and a help icon on the right.

The left sidebar contains a list of items: APIs (selected), insureriskma-dev, insureriskmanagement-prod, Usage Plans, API Keys, Custom Domain Names, Client Certificates, VPC Links, and Settings.

The main content area displays two API entries:

- insureriskma-dev**: Created on 5/4/2019. Created automatically by Zappa. Protocol: HTTP. Endpoint Configuration: Endpoint Type ⓘ Edge optimized.
- insureriskmanagement-prod**: Created on 5/4/2019. Created automatically by Zappa. Protocol: HTTP. Endpoint Configuration: Endpoint Type ⓘ Edge optimized.

A red box highlights the 'insureriskmanagement-prod' API entry.

2. AWS Lambda – FaaS Cloud

3. select **stages** in left side panel and see the **invoke url**.

The screenshot shows the AWS API Gateway Stage Editor for the 'prod' stage of an API named 'insuranceriskmanagement-prod'. The left sidebar is collapsed, and the main area is titled 'prod Stage Editor'. At the top, there's a 'Create' button and a 'Delete Stage' link. Below the title, the 'Invoke URL' is displayed as a blue link: [https://\[REDACTED\].amazonaws.com/prod](https://[REDACTED].amazonaws.com/prod). A navigation bar below the title includes tabs for 'Settings', 'Logs/Tracing', 'Stage Variables', 'SDK Generation', 'Export', and 'Deployment History', with 'Settings' being the active tab. Under 'Settings', there are sections for 'Cache Settings' (with an 'Enable API cache' checkbox), 'Default Method Throttling' (describing rate and burst limits), and 'Documentation History' (with a 'Canary' tab). The bottom of the page shows browser navigation and search bars.

The screenshot shows the AWS API Gateway Stages list for an API named 'httpapitest01'. The left sidebar is expanded, showing 'APIs', 'Custom domain names', 'VPC links', and 'Develop' sections with 'Routes' and 'Authorization'. The main area shows a list of stages under 'Stages for httpapitest01'. The '\$default' stage is selected, and its details are shown in the 'Stage details' panel on the right. The 'Details' section shows the stage's name, creation date (March 15, 2020 2:45 AM), and last update date (March 15, 2020 2:49 AM). The 'Invoke URL' is also listed as [https://\[REDACTED\].execute-api.us-east-2.amazonaws.com](https://[REDACTED].execute-api.us-east-2.amazonaws.com). Action buttons for 'Delete' and 'Edit' are available at the bottom of the stage list and details panel.

2. AWS Lambda – FaaS Cloud

Adding a http listener can be done by going to your lambda function, selecting the 'triggers' tab and 'add trigger', finally selecting API Gateway - but as others mentioned this does create a public facing url.

The screenshot shows a user interface for managing triggers for a Lambda function. At the top, there are four tabs: 'Code', 'Configuration', 'Triggers', and 'Monitoring'. The 'Triggers' tab is highlighted with an orange border. Below the tabs, a message reads 'You do not have any triggers for this function.' A blue button labeled '+ Add trigger' is visible at the bottom left.

Section Conclusions

JMS – Java Message Service is JEE implementation for
MOM – Message Oriented Middleware

The chosen Java Web & Application Server with JMS
implementation for testing is Apache TomEE

Apache Kafka demo

JMS DEMO

for easy sharing



Distributed Application Development

Communicate & Exchange Ideas





Questions & Answers!

But wait...

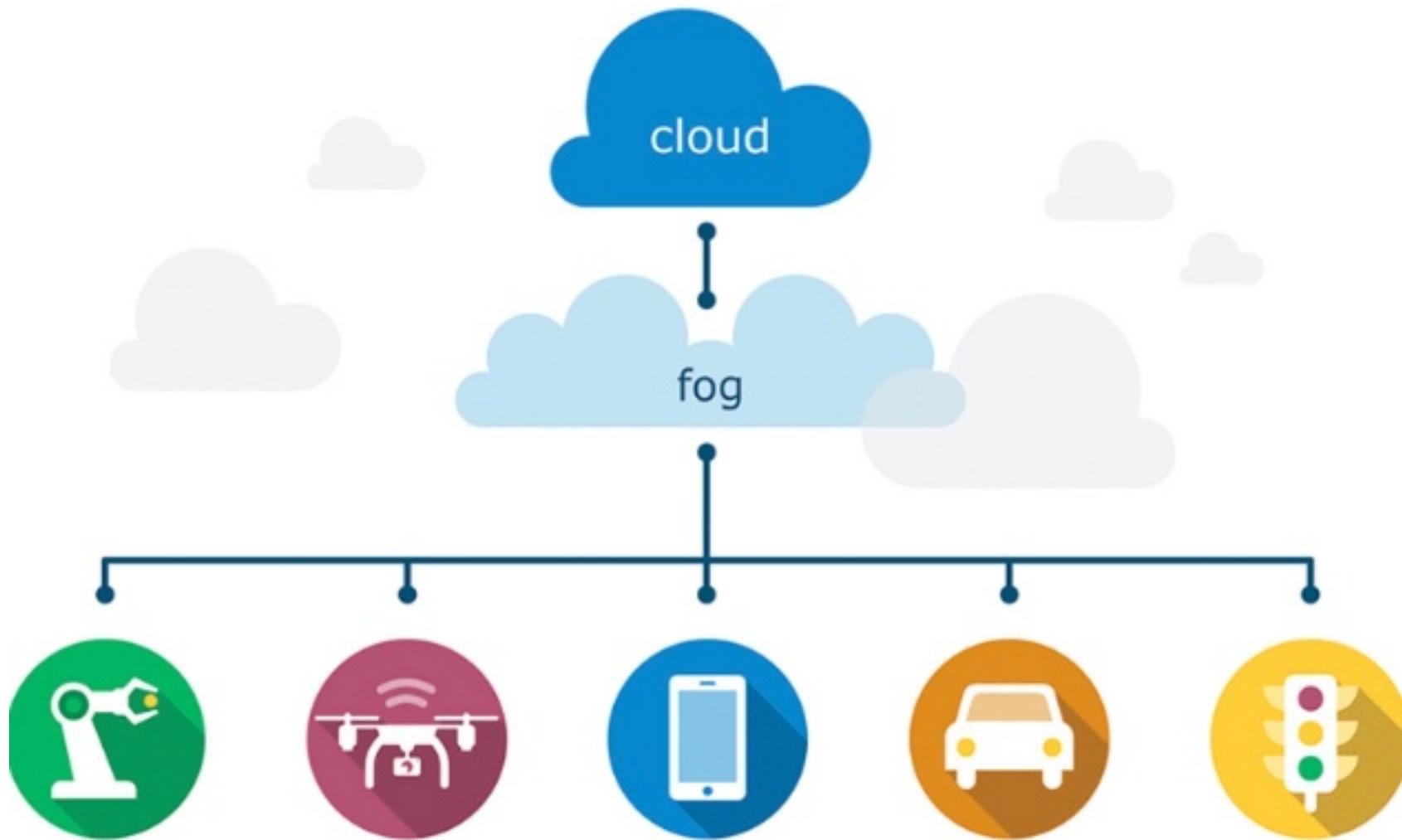
There's More!



Fog Computing vs. FaaS



Fog Computing

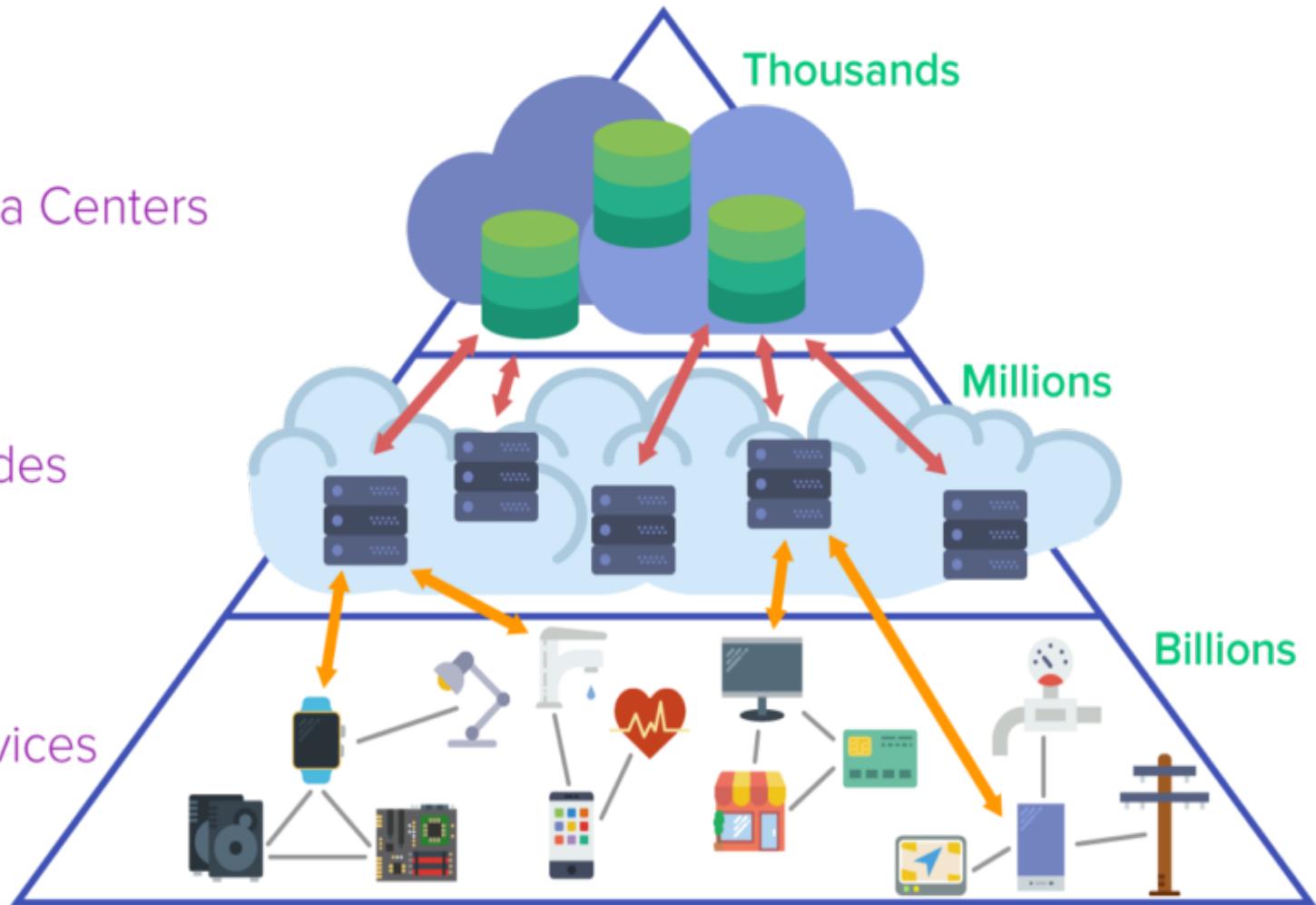


Fog Computing

CLOUD | Data Centers

FOG | Nodes

EDGE | Devices





Thanks!



DAD – Distributed Application Development
End of Lecture 09

