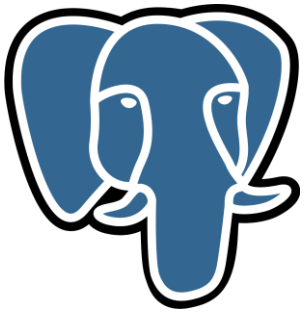
Introduction to backend development with Node.js, C++, MongoDB and Amazon Web Service

AGENDA

- Introduction
- Mongo and Node.js
  - MongoDB native driver
  - Mongoose
- Child Processes
  - spawn(), exec(), fork()
  - Cluster
- C++ Addons
  - Linker
  - node-gyp
  - nan
- Amazon Web Services
  - API Gateway
  - CloudWatch
  - AWS Lambda

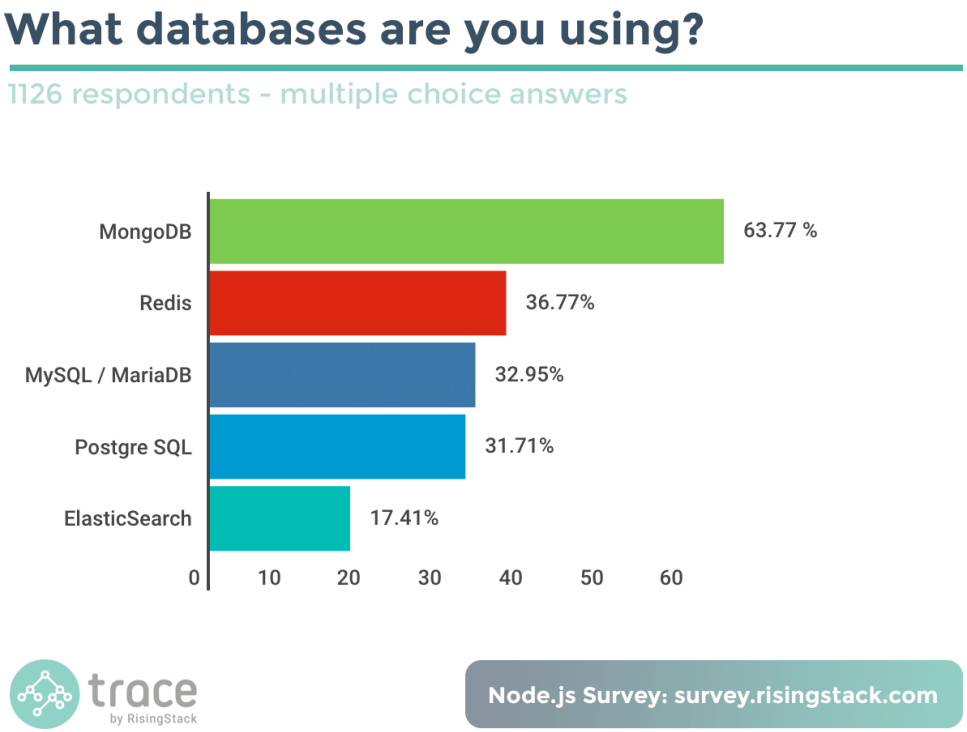
No single “best” database



Database Integration

Connecting to database is just a matter of loading an appropriate Node.js driver for the database in your app.

- Cassandra
- Couchbase
- CouchDB
- LevelDB
- MySQL
- MongoDB
- Neo4j
- Oracle
- PostgreSQL
- Redis
- SQL Server
- SQLite
- ElasticSearch



Ways to interact

There are two approaches for interacting with a database:

- Using the databases' native query language (e.g. SQL)
- Using an Object Data Model ("ODM") / Object Relational Model ("ORM")

	Mongoose	Mongodb driver
Schema *	Mandatory	No
Performance / processing time	Not bad	Excellent
Development time	Fast	Average
Maintainability	Easy	Little hard
Learning curve	Little high	Low
Community	Good	Good

AGENDA

- Introduction
- Mongo and Node.js
  - MongoDB native driver
  - Mongoose
- Child Processes
  - spawn(), exec(), fork()
  - Cluster
- C++ Addons
  - Linker
  - node-gyp
  - nan
- Amazon Web Services
  - API Gateway
  - CloudWatch
  - AWS Lambda

MongoDB Native Driver

```
$ npm install mongodb --save
```

```
const MongoClient = require('mongodb').MongoClient;
const assert = require('assert');

const url = 'mongodb://localhost:27017';
const dbName = 'myproject';

MongoClient.connect(url, {useNewUrlParser: true}, (err, client) => {
  assert.equal(null, err);
  console.log("Connected successfully to server");

  const db = client.db(dbName);

  client.close();
});
```

mongodb: insert documents

```
21  const insertDocuments = (db) => {
22    return new Promise((resolve, reject) => {
23      const collection = db.collection('documents');
24      collection.insertMany([
25        {a : 1}, {a : 2}, {a : 3}
26      ], (err, result) => {
27        if (err) {
28          return reject(err)
29        }
30        console.log("Inserted 3 documents into the collection");
31        return resolve(result)
32      })
33    })
34  }
```

mongodb: query database

```
36  const findDocuments = (db) => {
37    return new Promise((resolve, reject) => {
38      const collection = db.collection('documents');
39      collection.find({'a': 3}).toArray((err, docs) => {
40        if (err) {
41          return reject(err);
42        }
43        console.log(`Found ${docs.length} records`);
44        return resolve(docs)
45      })
46    })
47  }
```

mongodb: update documents

```
49  const updateDocument = (db) => {
50    return new Promise((resolve, reject) => {
51      const collection = db.collection('documents');
52      collection.updateOne({ a : 2 }, { $set: { b : 1 } },
53        (err, result) => {
54          if (err) {
55            return reject(err)
56          }
57          console.log("Updated the document with the field a equal to 2");
58          return resolve(result)
59      })
60    })
61  }
```

mongodb: remove documents

```
65  const removeDocument = (db) => {
66    return new Promise((resolve, reject) => {
67      const collection = db.collection('documents');
68      collection.deleteOne({ a : 3 }, (err, result) => {
69        if (err) {
70          return reject(err)
71        }
72        console.log("Removed the document with the field a equal to 3");
73        return resolve(result)
74      })
75    })
76  }
```

mongodb: remove documents

```
8  MongoClient.connect(url, {useNewUrlParser: true}, (err, client) => {
9    assert.equal(null, err);
10   console.log("Connected successfully to server");
11
12   const db = client.db(dbName);
13
14   return Promise.resolve()
15     .then(() => insertDocuments(db))
16     .then(() => findDocuments(db))
17     .tap(data => console.log('chain log:', data.length))
18     .then(() => updateDocument(db))
19     .then(() => removeDocument(db))
20     .finally(() => client.close())
21 });
```

https://github.com/mongodb/node-mongodb-native

http://mongodb.github.io/node-mongodb-native/3.1/api/

Wake UP

```
var foo = {n: 1};
var bar = foo;
foo.x = foo = {n: 2};
```

1. foo = {n:1}
2. foo = {n:2}
3. foo = {n:1, x:foo}
4. foo = {n:2, x:foo}
5. It’s a tricky question without right answer xd

Mongoose

mongoose

elegant mongodb object modeling for node.js

Mongoose

Mongoose - **schema-based** solution to model your application data.

- type casting
- validation
- query building
- business logic hooks
- and more …

```
1  const mongoose = require('mongoose');
2
3  const url = 'mongodb://localhost:27017';
4  const dbName = 'mypets';
5
6  mongoose.connect(`${url}/${dbName}`, {useNewUrlParser: true});
7
8  const Cat = mongoose.model('Cat', { name: String });
9
10 const kitty = new Cat({ name: 'Zildjian' });
11 kitty.save().then(() => console.log('meow'));
12
```

Mongoose Schema

Everything in Mongoose starts with a Schema. Each schema maps to a MongoDB collection and defines the shape of the documents within that collection.

Models are fancy constructors compiled from our Schema definitions. Instances of these models represent documents which can be saved and retrieved from our database. All document creation and retrieval from the database is handled by these models.

```
8  const kittySchema = new mongoose.Schema({
9    name: String
10 });
11
12 const Kitten = mongoose.model('Kitten', kittySchema);
13
```

Schema: Instance methods

You may also define custom document instance methods using schema.

```
8  const kittySchema = new mongoose.Schema({
9    name: String
10 });
11
12 // NOTE: methods must be added to the schema before compiling it with mongoose.model()
13 kittySchema.methods.speak = function () {
14   var greeting = this.name
15     ? "Meow name is " + this.name
16     : "I don't have a name";
17   console.log(greeting);
18 }
19
20 const Kitten = mongoose.model('Kitten', kittySchema);
21
```

Model: Constructing documents

Model.create() - Shortcut for saving one or more documents to the database.
MyModel.create(docs) does new MyModel(doc).save() for every doc in docs.

```
31  Kitten.create({
32    name: 'Boris'
33  }, (err, res) => {
34    if (err) return console.error(err);
35    console.log(res);
36  });
```

Model: Querying documents

Documents can be retrieved using each models find, findById, findOne, or where static methods.

```
35  // executes immediately, passing results to callback
36  Kitten.find({ name: 'fluffy' }, (err, docs) => {});
37
38  // find kitty by id and execute immediately
39  Kitten.findById(id, (err, kitty) => {});
40
41  // find only 1 kitty
42  Kitten.findOne({ name: 'fluffy' }, (err, kitty) => {});
43
44  // return Promise but
45  Kitten.findOne({ name: 'fluffy' }).then();
46
47  // fully-fledged promise
48  Kitten.where('age').gte(2).lte(5).exec().then();
```

Model: Updating documents

Model.update() - updates one/many document in the database without returning it.
Model.updateOne() - will update only the first document that matches criteria.
Model.updateMany() - will update all documents that match criteria.

Model.findByIdAndUpdate() and Model.findOneAndUpdate() - finds a matching document, updates it, passing any options, and returns the found document (if any) to the callback.

```
58  Kitten.findOneAndUpdate({ name: 'fluffy' }, { name: 'fluffy2' })
59    .then(callback)
60    .catch(errCallback)
```

Model: Removing documents

Model.remove() - removes all documents that match conditions from the collection.
Model.deleteOne() - deletes the first document that matches conditions.
Model.deleteMany() - deletes all of the documents that match conditions.

Model.findOneAndRemove() and Model.findByIdAndRemove() - finds a matching document, removes it, passing the found document (if any) to the callback.

```
63  Kitten.findOneAndRemove({ name: 'fluffy' })
64    .then(callback)
65    .catch(errCallback)
```

Mongoose document

Mongoose documents represent a one-to-one mapping to documents as stored in MongoDB. Each document is an instance of its Model.

Document.prototype.save() – saves the document.
Document.prototype.update() - sends an update command with this document _id as the query selector.
Model.prototype.remove() - Removes this document from the db.

```
68  Tank.findById(id, (err, tank) => {
69    if (err) return handleError(err);
70
71    tank.size = 'large';
72    tank.save((err, updatedTank) => {
73      if (err) return handleError(err);
74      res.send(updatedTank);
75    });
76  });
```

Mongoose document

By default, find() will return documents as Mongoose Documents, which costs a lot. By adding lean() to the query, documents are returned as plain JavaScript objects and for this case of 10k+ returned documents, the query time got **reduced 3-5 times**.

```
68  Sales.find()
69    .where('author').equals(author)
70    .where('date').gt(startDate.unix()).lt(endDate.unix())
71    .lean()
72    .exec(function(err, results) {
73        callback();
74    });
```

Mongoose Queries

A Query enables you to build up a query using chaining syntax, rather than specifying a JSON object. The below 2 examples are equivalent.

```
Person
  .find({
    occupation: /host/,
    'name.last': 'Ghost',
    age: { $gt: 17, $lt: 66 },
    likes: { $in: [
      'vaporizing',
      'talking'
    ]}
  })
  .limit(10)
  .sort({ occupation: -1 })
  .select({ name: 2, occupation: 1 })
  .exec(callback)
```

```
Person
  .find({ occupation: /host/ })
  .where('name.last').equals('Ghost')
  .where('age').gt(17).lt(66)
  .where('likes').in([
    'vaporizing',
    'talking'
  ])
  .limit(10)
  .sort('-occupation')
  .select('name occupation')
  .exec(callback)
```

Mongoose Validation

- Validation is defined in the SchemaType
- Validation is middleware (*pre('save')* - hook)
- *doc.validate(callback)* or *doc.validateSync()* to run manually
- Validators are not run on undefined values (* required)
- Validation is customizable

Mongoose Middleware

Middleware - pre and post *hooks* - functions which are passed control during execution.

Mongoose has 4 types of middleware:

- document middleware: init, validate, save, remove
- model middleware: insertMany
- aggregate middleware: aggregate
- query middleware: count, find, findOne, findOneAndRemove, findOneAndUpdate, update

Mongoose Middleware

Middleware are useful for atomizing model logic. Here are some other ideas:

- complex validation
- removing dependent documents (removing a user removes all his blogposts)
- asynchronous defaults
- asynchronous tasks that a certain action triggers

```
68  schema.pre('save', function() {
69    return doStuff().
70      then(() => doMoreStuff());
71  });
72
73  // Or, in Node.js >= 7.6.0:
74  schema.pre('save', async function() {
75    await doStuff();
76    await doMoreStuff();
77  });
```

Error Handling Middleware

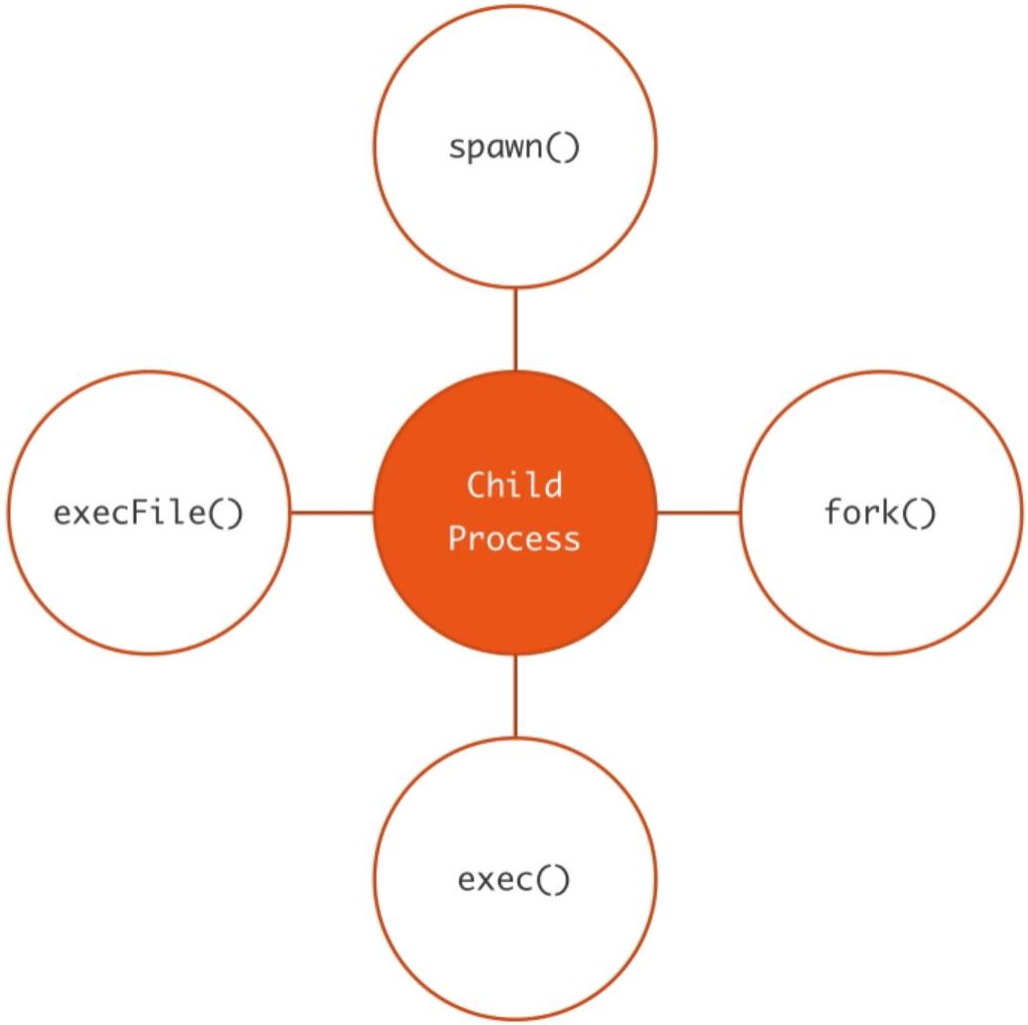
Error handling middleware is defined as middleware that takes one extra parameter: the 'error' that occurred as the first parameter to the function.

```
68  schema.pre('save', function(next) {
69    const err = new Error('something went wrong');
70    // If you call `next()` with an argument, that argument is assumed to be
71    // an error.
72    next(err);
73  });
74
75  schema.pre('save', function() {
76    // You can also return a promise that rejects
77    return new Promise((resolve, reject) => {
78      reject(new Error('something went wrong'));
79    });
80  });
81
82  schema.pre('save', function() {
83    // You can also throw a synchronous error
84    throw new Error('something went wrong');
85  });
86
87  schema.pre('save', async function() {
88    await Promise.resolve();
89    // You can also throw an error in an `async` function
90    throw new Error('something went wrong');
91  });
```

AGENDA

- Introduction
- Mongo and Node.js
  - MongoDB native driver
  - Mongoose
- Child Processes
  - spawn(), exec(), fork()
  - Cluster
- C++ Addons
  - Linker
  - node-gyp
  - nan
- Amazon Web Services
  - API Gateway
  - CloudWatch
  - AWS Lambda

Child Processes



Child Processes

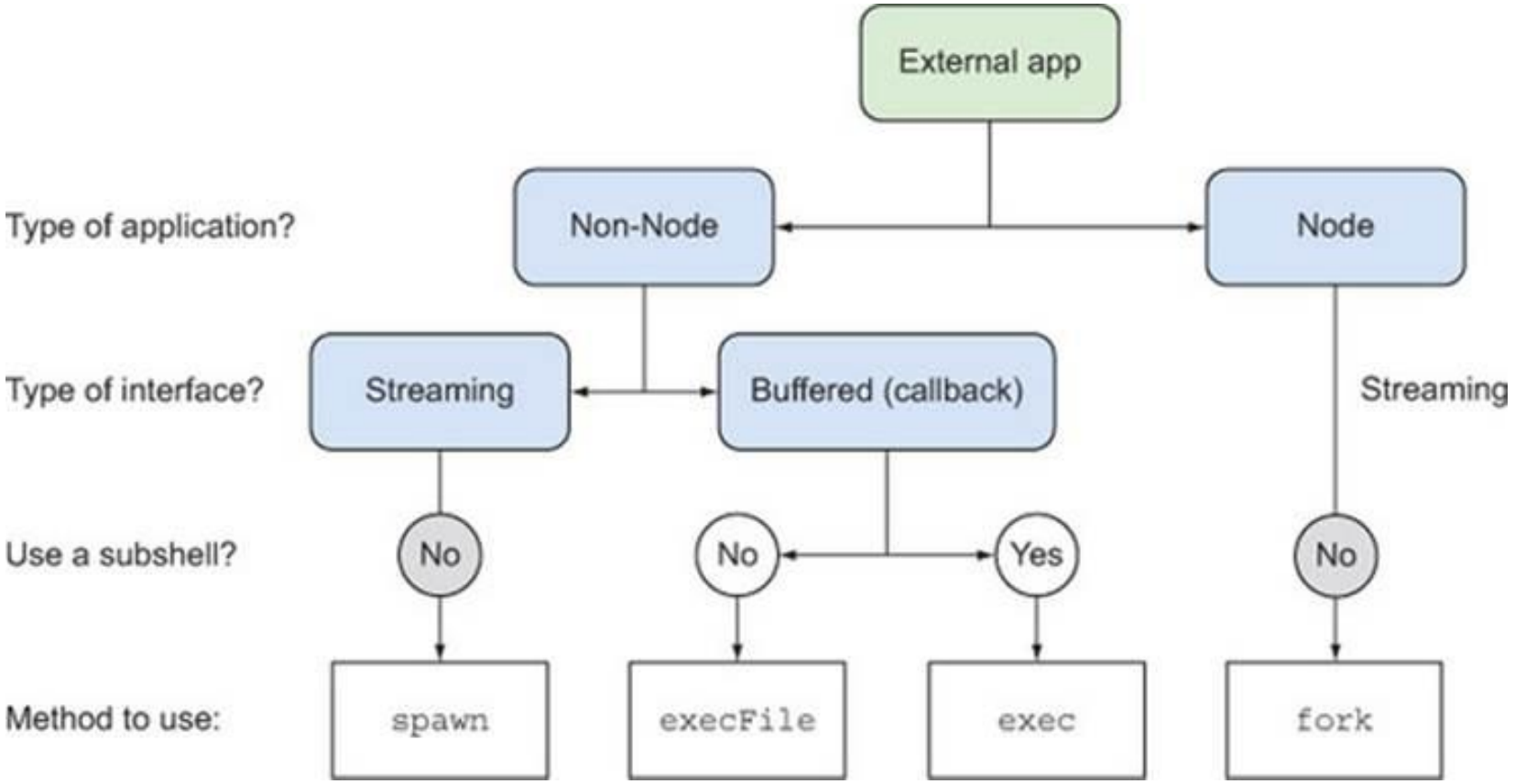
Instances of the ChildProcess class are EventEmitters that represent spawned child processes.

```
1  const { spawn } = require('child_process');
2  const ls = spawn('ls', ['-la', '..']);
3
4  ls.stdout.on('data', (data) => {
5    console.log(`stdout: ${data}`);
6  });
7
8  ls.stderr.on('data', (data) => {
9    console.log(`stderr: ${data}`);
10 });
11
12 ls.on('close', (code) => {
13   console.log(`child process exited with code ${code}`);
14 });
15
```

```
➜  child_processes node ls.js
stdout: total 0
drwxr-xr-x   4 vinfinit  staff  128 Jan 23 19:23 .
drwxr-xr-x  22 vinfinit  staff  704 Jan 23 11:11 ..
drwxr-xr-x   4 vinfinit  staff  128 Jan 23 11:08 c_addons
drwxr-xr-x   3 vinfinit  staff   96 Jan 23 19:23 child_processes

child process exited with code 0
➜  child_processes
```

Child Processes



Cluster

The worker processes are spawned using the child_process.fork() method, so that they can communicate with the parent via IPC and pass server handles back and forth.

```
const cluster = require('cluster');
const http = require('http');
const numCPUs = require('os').cpus().length;

if (cluster.isMaster) {
  console.log(`Master ${process.pid} is running`);

  for (let i = 0; i < numCPUs; i++) {
    cluster.fork();
  }

  cluster.on('exit', (worker, code, signal) => {
    console.log(`worker ${worker.process.pid} died`);
  });
} else {
  http.createServer((req, res) => {
    res.writeHead(200);
    res.end('hello world\n');
  }).listen(8000);

  console.log(`Worker ${process.pid} started`);
}
```

```
➜ child_processes node cluster.js
Master 7129 is running
Worker 7130 started
Worker 7133 started
Worker 7132 started
Worker 7131 started
```

AGENDA

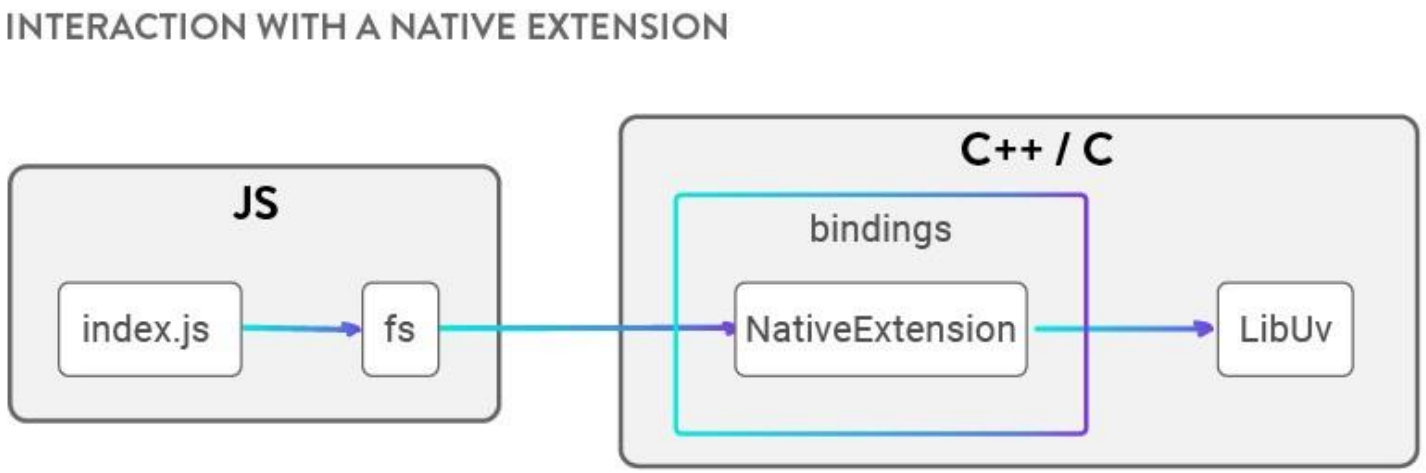
- Introduction
- Mongo and Node.js
  - MongoDB native driver
  - Mongoose
- Child Processes
  - spawn(), exec(), fork()
  - Cluster
- C++ Addons
  - Linker
  - node-gyp
  - nan
- Amazon Web Services
  - API Gateway
  - CloudWatch
  - AWS Lambda

I'm a JavaScript developer, why would I ever want to mess with C++?

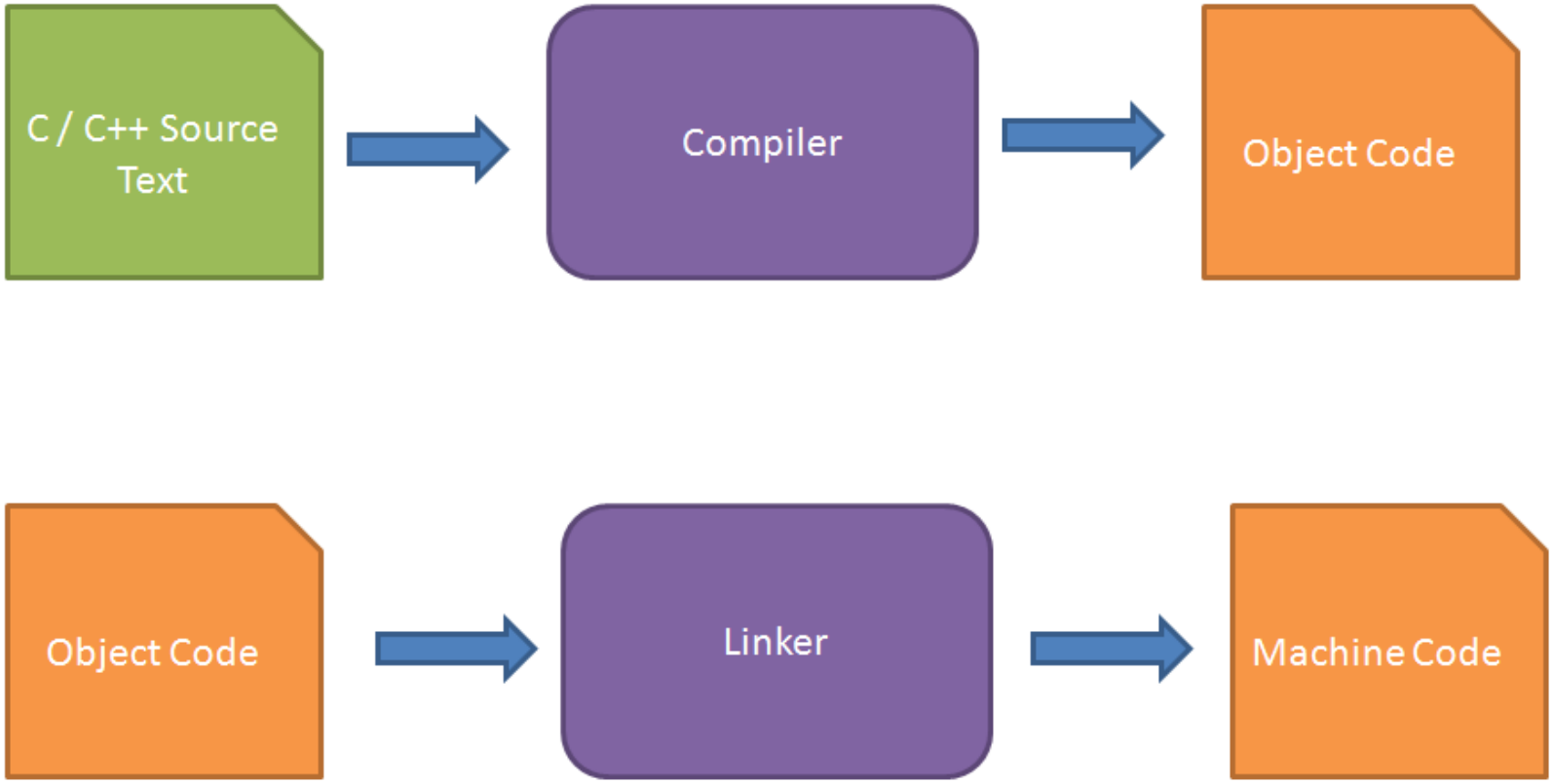
1. Direct access to existing (legacy) C/C++ libraries. Instead of calling these as external applications in "execute command" style, get your hands directly on the existing source code and pass the results back to Node.js in a form that is comprehensible for your JavaScript runtime. This way you can also get an access to low-level API of the operating system.
2. Performance. In many situations, a **well-written** native code might prove faster and more performant than the JavaScript equivalent.

C++ Addons

Node.js Addons are dynamically-linked shared objects, written in C++, that can be loaded into Node.js using the require() function, and used just as if they were an ordinary Node.js module. They are used primarily to provide an interface between JavaScript running in Node.js and C/C++ libraries.



Why linker?


C / C++ Source Text
Compiler
Object Code
Object Code
Linker
Machine Code

Static vs Dynamic linking

1. All the library modules are copied to the final executable image.
2. When the program is loaded, OS places only a single file to the memory.
3. Done by the linkers in the final step of the compilation.
4. If external source program is changed then they have to be recompiled and relinked.
5. Faster
6. Constant time

1. The names of external or shared libraries is placed into the memory.
2. Lets many programs use single copy of executable module.
3. Done at run time by the OS.
4. Only a single module needs to be updated and recompiled.
5. Since the library files are separately stored there may be compatibility issues
6. The time is variable

How does it look like

hello.cc

```cpp
#include <node.h>

namespace demo {

using v8::FunctionCallbackInfo;
using v8::Isolate;
using v8::Local;
using v8::NewStringType;
using v8::Object;
using v8::String;
using v8::Value;

void Method(const FunctionCallbackInfo<Value>& args) {
  Isolate* isolate = args.GetIsolate();
  args.GetReturnValue().Set(String::NewFromUtf8(
      isolate, "world", NewStringType::kNormal).ToLocalChecked());
}

void Initialize(Local<Object> exports) {
  NODE_SET_METHOD(exports, "hello", Method);
}

NODE_MODULE(NODE_GYP_MODULE_NAME, Initialize)

}
```

hello.js

```js
const addon = require('./build/Release/addon');

console.log(addon.hello());
```

binding.gyp

```
{
  "targets": [
    {
      "target_name": "addon",
      "sources": [ "hello.cc" ]
    }
  ]
}
```

What is important

Normal

```
void Initialize(Local<Object> exports);
NODE_MODULE(NODE_GYP_MODULE_NAME, Initialize)
```

Multi env

```
void AddEnvironmentCleanupHook(v8::Isolate* isolate,
                               void (*fun)(void* arg),
                               void* arg);
```

Worker support

```
using namespace v8;

extern "C" NODE_MODULE_EXPORT void
NODE_MODULE_INITIALIZER(Local<Object> exports,
                        Local<Value> module,
                        Local<Context> context) {
  /* Perform addon initialization steps here. */
}
```

C++ Addons

npm install -g node-gyp

node-gyp configure

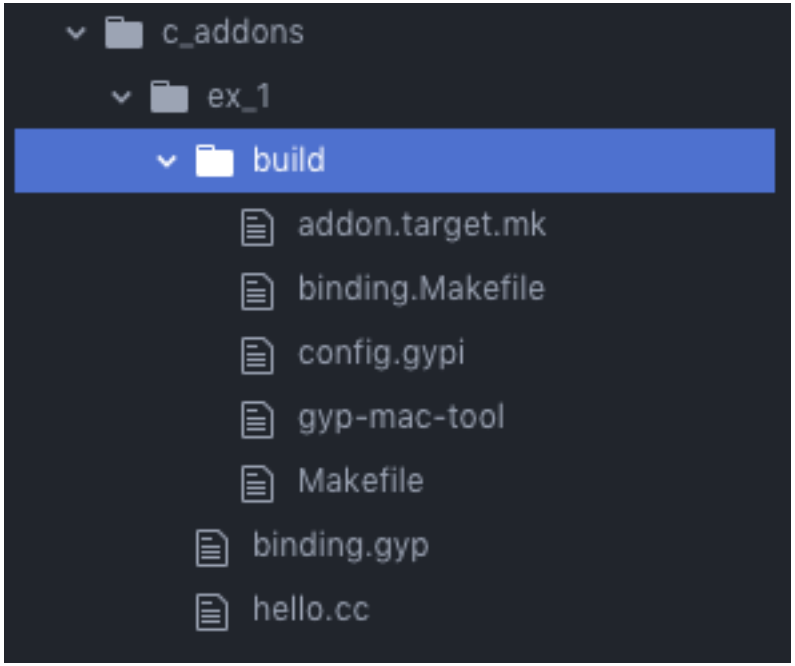
node-gyp build

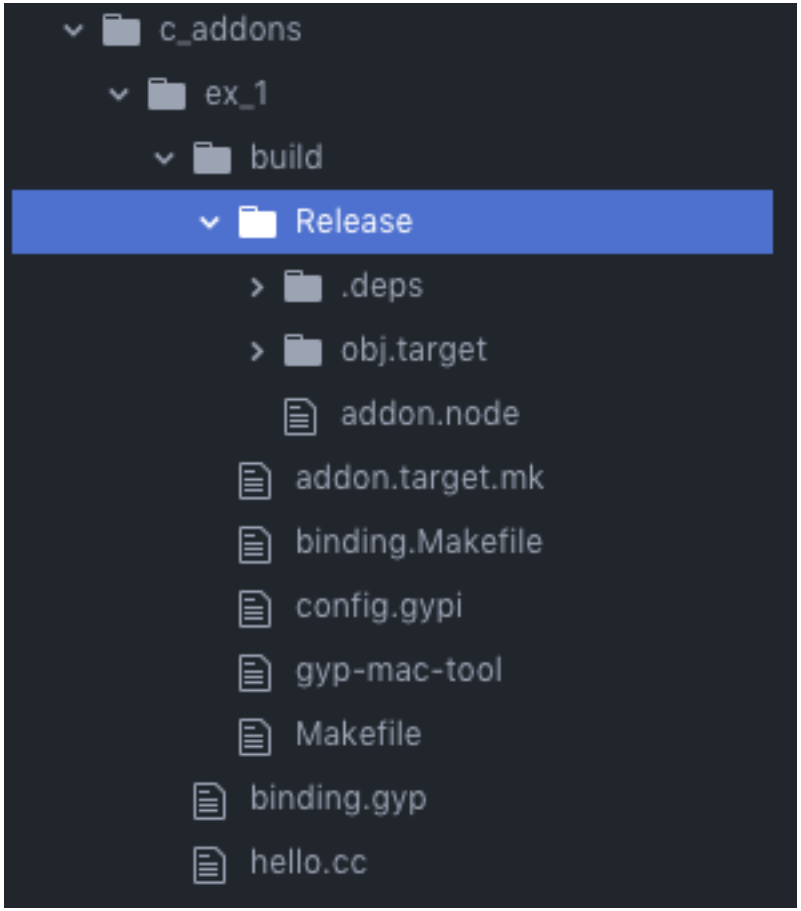
node hello.js

node-gyp configure

```
➜  ex_1 node-gyp configure
gyp info it worked if it ends with ok
gyp info using node-gyp@3.8.0
gyp info using node@10.9.0 | darwin | x64
gyp ERR! configure error
gyp ERR! stack Error: Command failed: /Users/vinfinit/app/anaconda3/bin/python -c import sys; print "%s.%s.%s" % sys.versio
n_info[:3];
gyp ERR! stack   File "<string>", line 1
gyp ERR! stack     import sys; print "%s.%s.%s" % sys.version_info[:3];
gyp ERR! stack                                  ^
gyp ERR! stack SyntaxError: invalid syntax
gyp ERR! stack
gyp ERR! stack     at ChildProcess.exithandler (child_process.js:289:12)
gyp ERR! stack     at ChildProcess.emit (events.js:182:13)
gyp ERR! stack     at maybeClose (internal/child_process.js:961:16)
gyp ERR! stack     at Socket.stream.socket.on (internal/child_process.js:380:11)
gyp ERR! stack     at Socket.emit (events.js:182:13)
gyp ERR! stack     at Pipe._handle.close (net.js:599:12)
gyp ERR! System Darwin 18.0.0
gyp ERR! command "/usr/local/bin/node" "/usr/local/bin/node-gyp" "configure"
gyp ERR! cwd /Users/vinfinit/projects/frontcamp/node_2/c_addons/ex_1
gyp ERR! node -v v10.9.0
gyp ERR! node-gyp -v v3.8.0
gyp ERR! not ok
➜  ex_1
```

node-gyp configure build





node-gyp configure build

```
➜  ex_1 node-gyp configure
gyp info it worked if it ends with ok
gyp info using node-gyp@3.8.0
gyp info using node@10.9.0 | darwin | x64
gyp info spawn /usr/bin/python
gyp info spawn args [ '/usr/local/lib/node_modules/node-gyp/gyp/gyp_main.py',
gyp info spawn args   'binding.gyp',
gyp info spawn args   '-f',
gyp info spawn args   'make',
gyp info spawn args   '-I',
gyp info spawn args   '/Users/vinfinit/projects/frontcamp/node_2/c_addons/ex_1/build/config.gypi',
gyp info spawn args   '-I',
gyp info spawn args   '/usr/local/lib/node_modules/node-gyp/addon.gypi',
gyp info spawn args   '-I',
gyp info spawn args   '/Users/vinfinit/.node-gyp/10.9.0/include/node/common.gypi',
gyp info spawn args   '-Dlibrary=shared_library',
gyp info spawn args   '-Dvisibility=default',
gyp info spawn args   '-Dnode_root_dir=/Users/vinfinit/.node-gyp/10.9.0',
gyp info spawn args   '-Dnode_gyp_dir=/usr/local/lib/node_modules/node-gyp',
gyp info spawn args   '-Dnode_lib_file=/Users/vinfinit/.node-gyp/10.9.0/<(target_arch)/node.lib',
gyp info spawn args   '-Dmodule_root_dir=/Users/vinfinit/projects/frontcamp/node_2/c_addons/ex_1',
gyp info spawn args   '-Dnode_engine=v8',
gyp info spawn args   '--depth=.',
gyp info spawn args   '--no-parallel',
gyp info spawn args   '--generator-output',
gyp info spawn args   'build',
gyp info spawn args   '-Goutput_dir=.' ]
gyp info ok
➜  ex_1
```

```
➜  ex_1 node-gyp build
gyp info it worked if it ends with ok
gyp info using node-gyp@3.8.0
gyp info using node@10.9.0 | darwin | x64
gyp info spawn make
gyp info spawn args [ 'BUILDTYPE=Release', '-C', 'build' ]
  CXX(target) Release/obj.target/addon/hello.o
  SOLINK_MODULE(target) Release/addon.node
gyp info ok
➜  ex_1
```

nodejs/nan

https://github.com/nodejs/nan - Native Abstraction for Node.js

Usage

Simply add **NAN** as a dependency in the *package.json* of your Node addon:

```
$ npm install --save nan
```

Pull in the path to **NAN** in your *binding.gyp* so that you can use `#include <nan.h>` in your *.cpp* files:

```
"include_dirs" : [
    "<!(node -e \"require('nan')\")"
]
```

This works like a `-I<path-to-NAN>` when compiling your addon.

HelloWorld with nan

```cpp
#include <nan.h>

NAN_METHOD(Hello) {
    auto message = Nan::New("Hello from C++!").ToLocalChecked();
    info.GetReturnValue().Set(message);
}

NAN_MODULE_INIT(Initialize) {
    NAN_EXPORT(target, Hello);
}

NODE_MODULE(addon, Initialize);
```

main.cpp

```cpp
#include <nan.h>

NAN_METHOD(IsPrime) {
    if (!info[0]->IsNumber()) {
        Nan::ThrowTypeError("argument must be a number!");
        return;
    }

    int number = (int) info[0]->NumberValue();

    if (number < 2) {
        info.GetReturnValue().Set(Nan::False());
        return;
    }

    for (int i = 2; i < number; i++) {
        if (number % i == 0) {
            info.GetReturnValue().Set(Nan::False());
            return;
        }
    }

    info.GetReturnValue().Set(Nan::True());
}

NAN_MODULE_INIT(Initialize) {
    NAN_EXPORT(target, IsPrime);
}

NODE_MODULE(addon, Initialize);
```

binding.gyp

```
{
  "targets": [
    {
      "include_dirs": [
        "<!(node -e \"require('nan')\")"
```

isPrime.js

```js
module.exports = (number) => {
  if (typeof number !== 'number') {
    throw new TypeError('argument must be a number!');
  }

  if (number < 2) {
    return false;
  }

  for (let i = 2; i < number; i++) {
    if (number % i === 0) {
      return false;
    }
  }

  return true;
};
```

main.js

```js
const {IsPrime} = require('./build/Release/addon');
const isPrime = require('./isPrime');

// thirty-fifth million first prime number (see https://pr
const number = 654188429;
const NATIVE = 'native';
const JS = 'js';

console.time(NATIVE);
console.log(`${NATIVE}: checking whether ${number} is prim
console.timeEnd(NATIVE);
console.log('');
console.time(JS);
console.log(`${JS}: checking whether ${number} is prime...
console.timeEnd(JS);
```

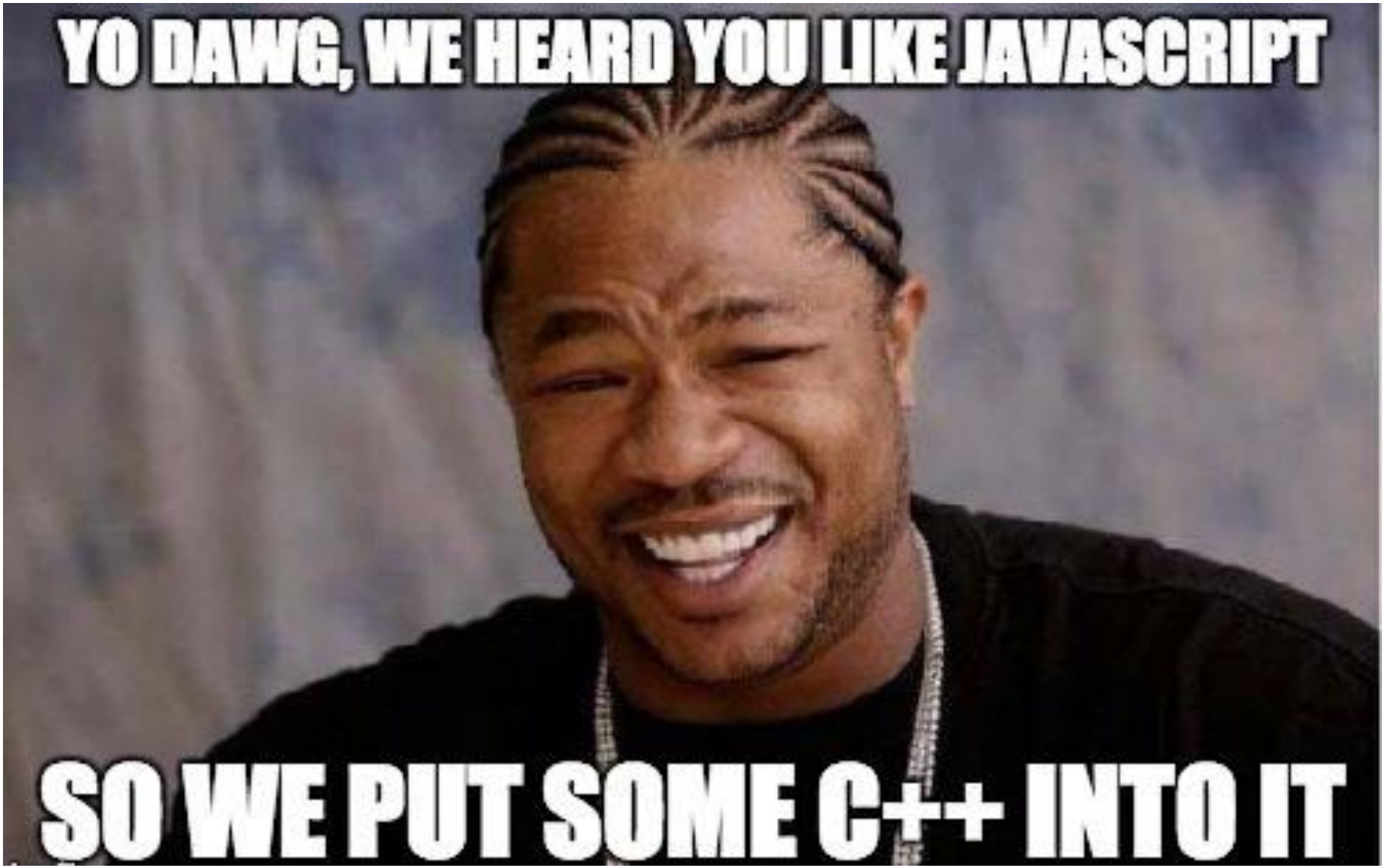
npm init

npm init

```
About to write to /Users/vinfinit/projects/frontcamp/node_2/c_addons/ex_2/package.json:

{
  "name": "ex_2",
  "version": "1.0.0",
  "description": "",
  "main": "main.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "install": "node-gyp rebuild"
  },
  "author": "",
  "license": "ISC",
  "gypfile": true
}


Is this OK? (yes)
```


YO DAWG, WE HEARD YOU LIKE JAVASCRIPT
SO WE PUT SOME C++ INTO IT

Node.js vs Native implementation

main.js

```
1  const {IsPrime} = require('./build/Release/addon');
2  const isPrime = require('./isPrime');
3
4  // thirty-fifth million first prime number (see https
5  const number = 654188429;
6  const NATIVE = 'native';
7  const JS = 'js';
8
9  console.time(NATIVE);
10 console.log(`${NATIVE}: checking whether ${number} is prime... ${IsPrime(number)}`);
11 console.timeEnd(NATIVE);
12 console.log('');
13 console.time(JS);
14 console.log(`${JS}: checking whether ${number} is prime... ${isPrime(number)}`);
15 console.timeEnd(JS);
16
```

..c_addons/ex_2

```
➜  ex_2 node main.js
native: checking whether 654188429 is prime... true
native: 2243.088ms

js: checking whether 654188429 is prime... true
js: 3374.973ms
➜  ex_2
```

Big Primes

AGENDA

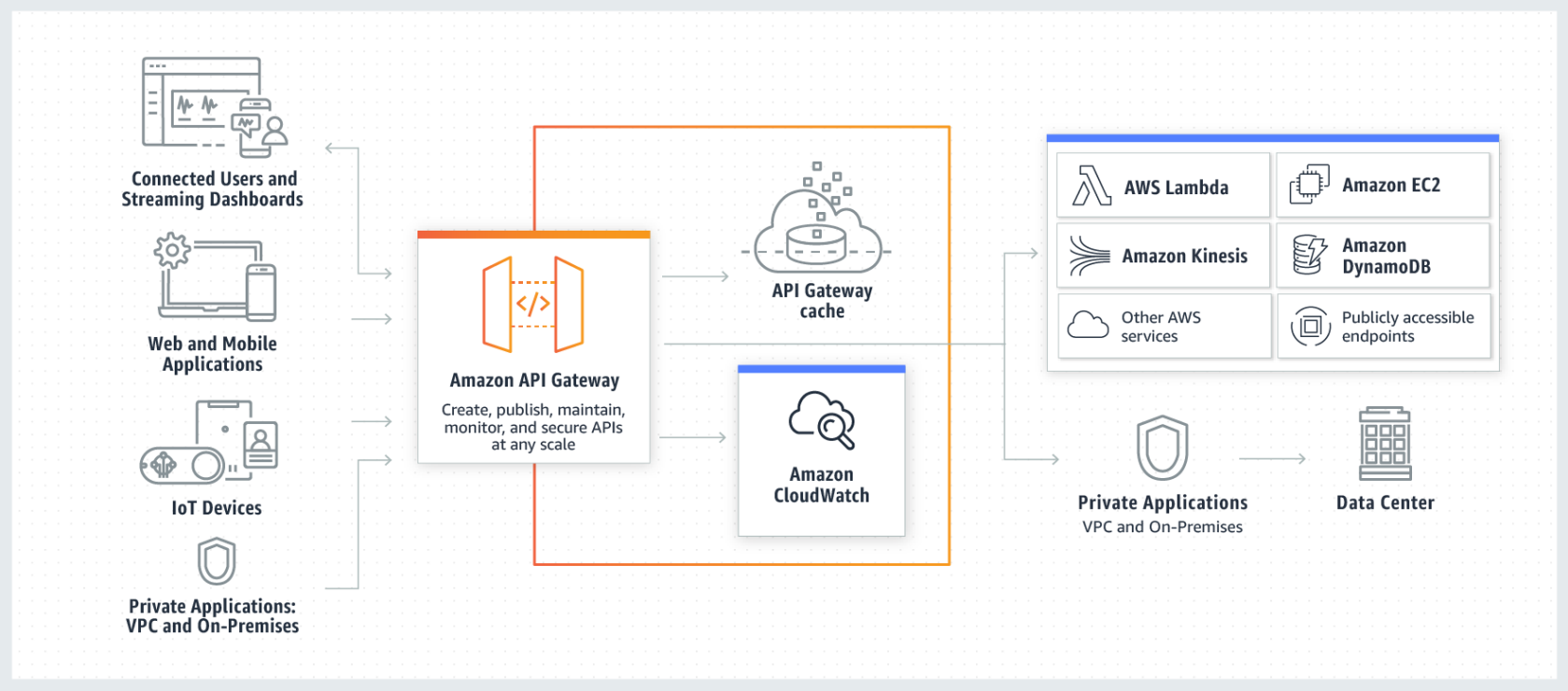
- Introduction
- Mongo and Node.js
  - MongoDB native driver
  - Mongoose
- Child Processes
  - spawn(), exec(), fork()
  - Cluster
- C++ Addons
  - Linker
  - node-gyp
  - nan
- Amazon Web Services
  - API Gateway
  - CloudWatch
  - AWS Lambda

Amazon Web Services

AWS Services included in the AWS Service Broker:

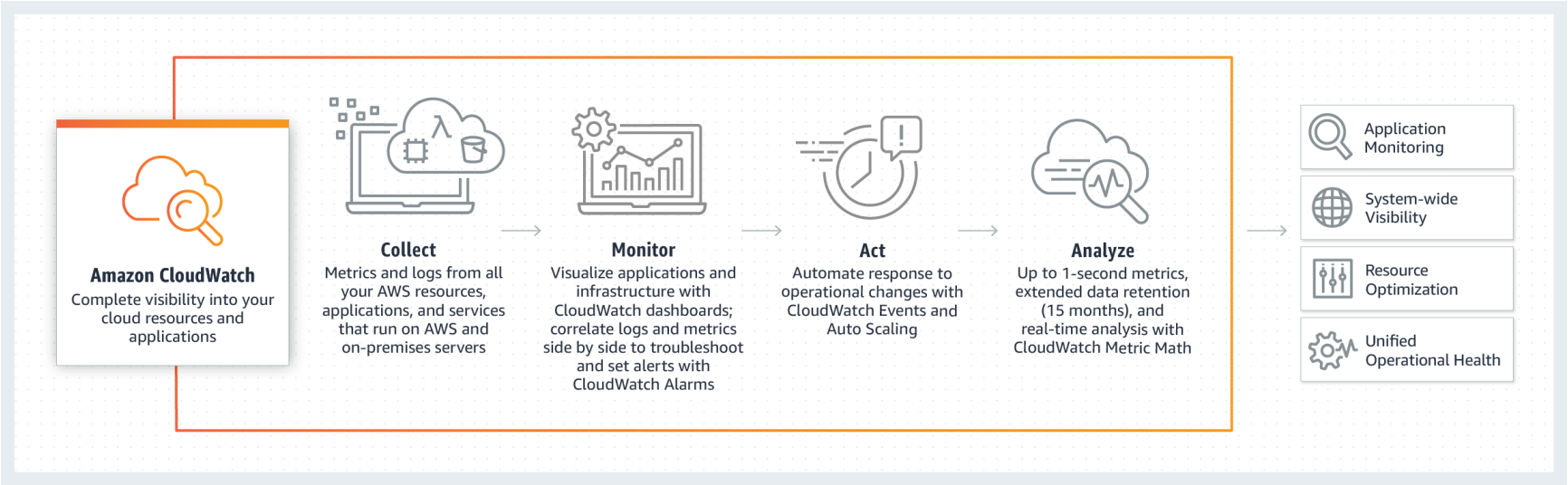
Amazon API Gateway

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale.



Amazon CloudWatch

Amazon CloudWatch is a monitoring and management service built for developers, system operators, site reliability engineers (SRE), and IT managers.

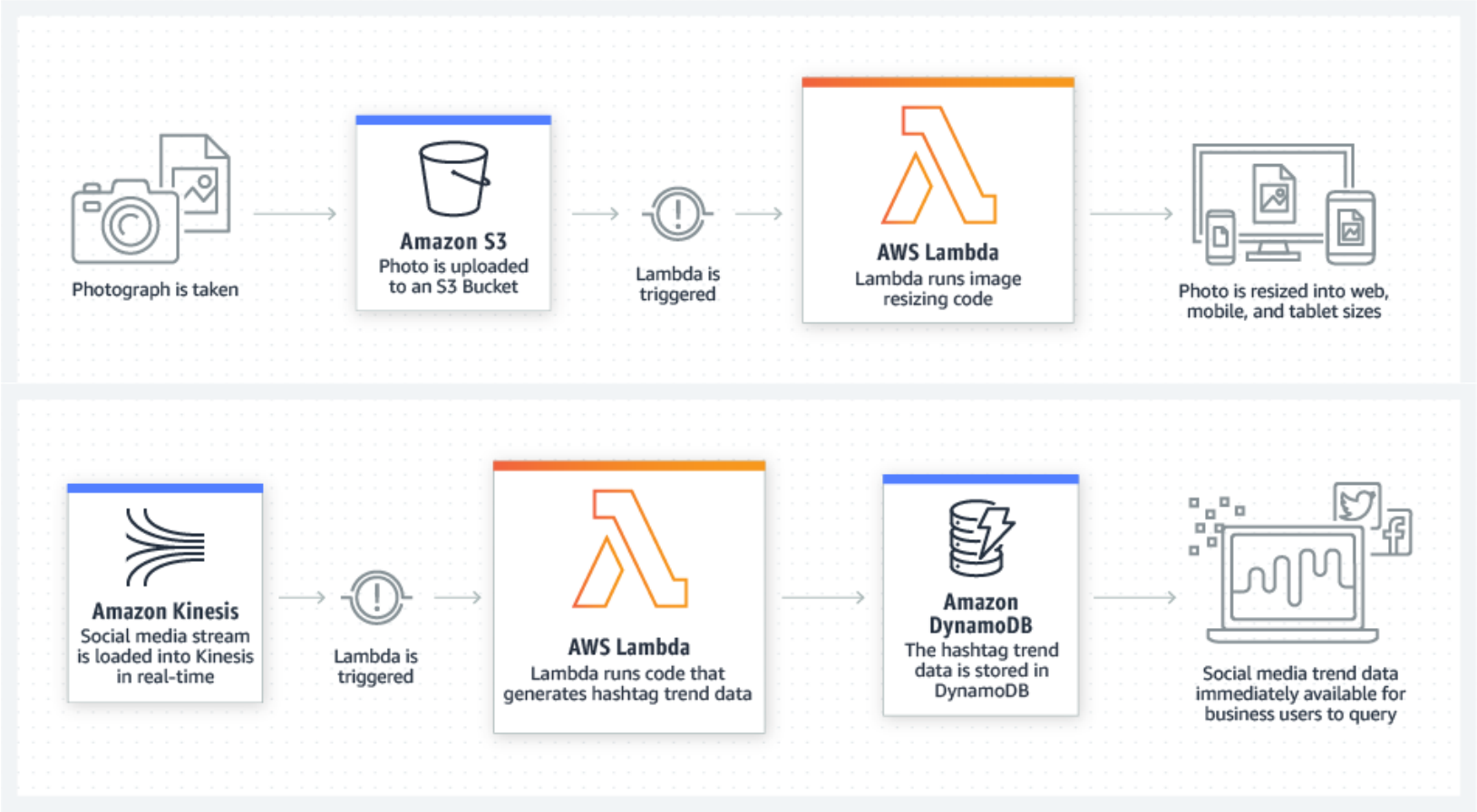


AWS Lambda

How it works

AWS Lambda

Real-time file/stream processing



AWS Lambda (local)

```
npm install -g lambda-local
```

```
# Simple usage

lambda-local -l index.js -h handler -e examples/s3-put.js
```

```
# Input environment variables

lambda-local -l index.js -h handler -e examples/s3-put.js -E '{"key":"value","key2":"value2"}'
```

https://github.com/ashiina/lambda-local

AWS Lambda and C++ Addons

The process isn’t much different than creating normal AWS Lambda functions with Node.js - you’ll just need to get your development environment to match the requirements for AWS.

Details are here.


It’s better to see something
once, than to hear about it
a thousand times.
- Anonymous

Click here

Hometask

Part 2:

1. Install and setup mongoose.
2. Create a mongoose scheme for news entity.
3. Replace "console logs"/stubs from part 1 to real communication with database.

    - Find all news
    - Find news by ID
    - Insert news
    - Update news record
    - Delete news from DB

4. Describe mongoose scheme for User model. Add registration / authorization functionality (passportjs) for accessing functionality edit / delete news.

*Advanced:

Add Facebook authentication (passport.js).


ARE THERE
ANY QUESTIONS?