Background Jobs + Node.JS

Async Processing for your Async Language

{{ BioPage }}

@evantahler

www.evantahler.com

We are Hiring! Talk to me later!

- Director of Technology @ TaskRabbit
- Maintainer of node-resque
- Maintainer of actionhero.js

DISCLAIMER!

Most of what you will see is a terrible idea.

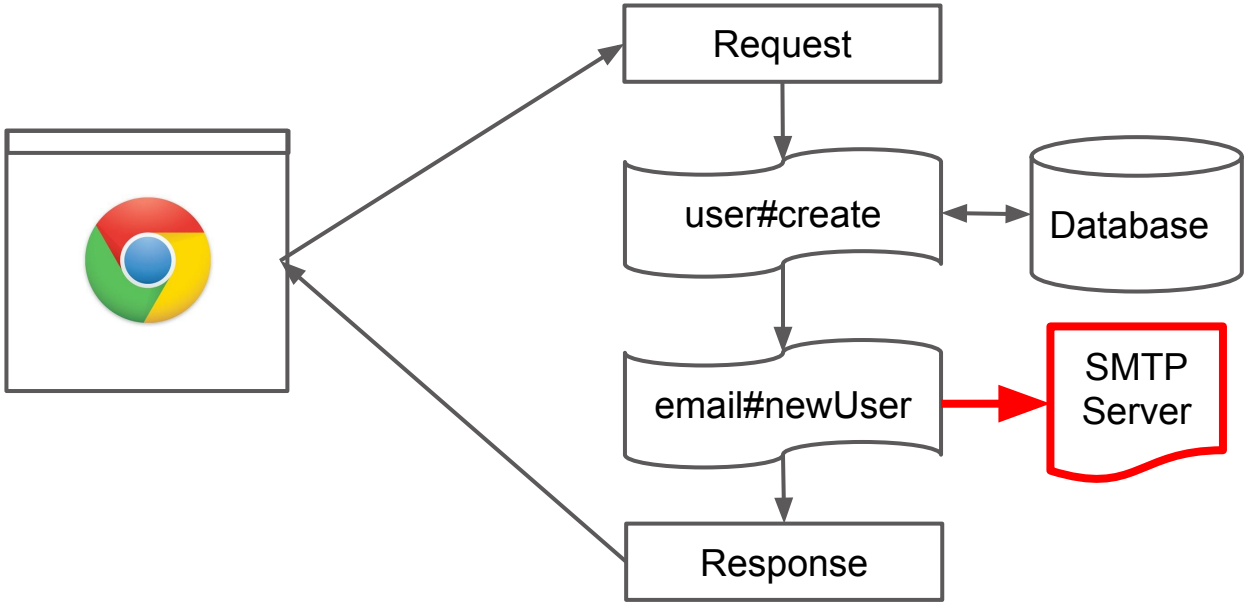
Try this at ~, not on production

The Point:

Everything is better/faster/stronger in node.

...even the bad ideas!

So you have a website...



Possible Task Strategies

1. Foreground (in-line)
2. Parallel (threaded-ish)
3. Local Messages (fork-ish)
4. ~~Remote Messages~~
5. Remote Queues (Resque-ish)
6. ~~Event Bus (Kafka-ish)~~

Strategy 1: Foreground


```php
<?php
$to      = 'nobody@example.com';
$subject = 'the subject';
$message = 'hello';
$headers = 'From: webmaster@example.com' . "\r\n" .
    'Reply-To: webmaster@example.com' . "\r\n" .
    'X-Mailer: PHP/' . phpversion();

mail($to, $subject, $message, $headers);
?>
```

Strategy 1: Foreground

```
var http          = require('http');
var nodemailer    = require('nodemailer');
var httpPort      = 8080 || process.env.port;

var transporter = nodemailer.createTransport({
    service: 'gmail',
    auth: { user: require('./.emailUsername'), pass: require('./.emailPassword') }
});
```

```
var server = function(req, res){
  var start = Date.now();
  var responseCode = 200;
  var response     = {};
  sendEmail(req, function(error, email){
    response.email = email;
    if(error){
      console.log(error);
      responseCode = 500;
      response.error = error;
    }
    res.writeHead(responseCode, {'Content-Type': 'application/json'});
    res.end(JSON.stringify(response, null, 2));
    var delta = Date.now() - start;
    console.log('Sent an email to in ' + delta + 'ms');
  });
};

http.createServer(server).listen(httpPort, '0.0.0.0');

console.log('Server running at http://0.0.0.0:' + httpPort);
console.log('send an email and message to /TO_ADDRESS/SUBJECT/YOUR_MESSAGE');
```

Strategy 1: Foreground

```
var sendEmail = function(req, callback){
  var urlParts      = req.url.split('/');
  var email         = {
    from:    require('./.emailUsername'),
    to:      urlParts[1],
    subject: urlParts[2],
    text:    urlParts[3],
  };
  transporter.sendMail(email, function(error, info){
    callback(error, email);
  });
};
```

Async!

Strategy 1: Foreground

- Why it is better in node:
  - The client still needs to wait for the message to send, but you won't block any other client's requests
  - Avg response time of ~2 seconds from my couch
- Why it is still a bad idea:
  - Slow
  - Spending "web server" resources on sending email
  - Error / Timeout to the client for "partial success"
    - IE: Account created but email not sent
    - Confusing to the user, dangerous for the DB

Strategy 2: Parallel

- “Threading”
  - But if it were real threading, the client would still have to wait
  - I guess this might help you catch errors…
  - But you could use domains?
  - **note: do not get into a discussion about threads*
- Lets get crazy:
  - **Ignore the Callback**

DISCLAIMER!

Most of what you will see is a terrible idea.

Try this at ~, not on production

```
var sendEmail = function(req, callback){
  var urlParts     = req.url.split('/');
  var email        = {
    from:    require('./.emailUsername'),
    to:      urlParts[1],
    subject: urlParts[2],
    text:    urlParts[3],
  };
  transporter.sendMail(email, function(error, info){
    if(typeof callback === 'function'){ callback(error, email); }
  });
};

var server = function(req, res){
  var start = Date.now();
  var responseCode = 200;
  var response     = {};
  sendEmail(req);
  res.writeHead(responseCode, {'Content-Type': 'application/json'});
  res.end(JSON.stringify(response, null, 2));
};
```

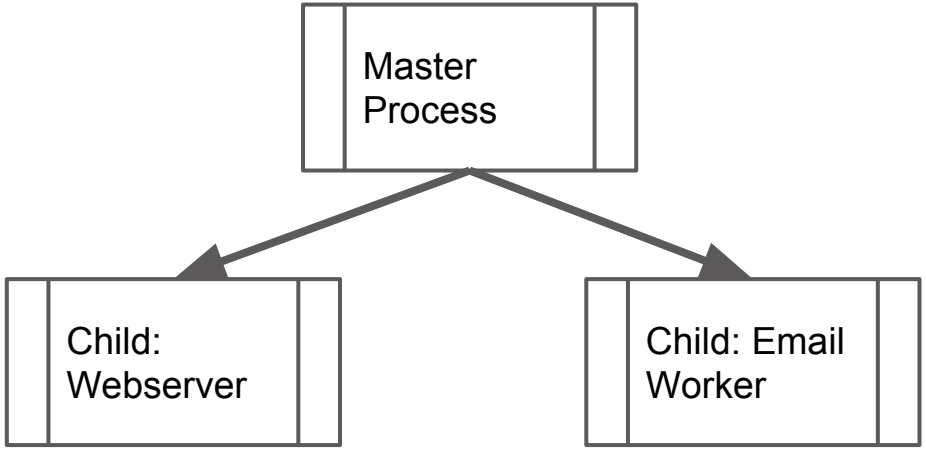
Async!

Strategy 2: Parallel

- Why it is better in node:
  - It’s rare you can actually do this in a language… without threading!
  - Crazy-wicked-fast.
- Why it is still a bad idea:
  - 0 callbacks, 0 data captured
  - I guess you could log errors?
    - But what would you do with that data?
  - The client has no idea what happened

Strategy 3: Local Messages

- “Forking”
  - or: *“The part of the talk where we grossly over-engineer some stuff”*



Strategy 3: Local Messages

```
if(cluster.isMaster){
  doMasterStuff();
}else{
  if(process.env.ROLE === 'server'){ doServerStuff(); }
  if(process.env.ROLE === 'worker'){ doWorkerStuff(); }
}
```

```
var doServerStuff = function(){
  var server = function(req, res){
    var urlParts = req.url.split('/');
    var email    = {
      to:      urlParts[1],
      subject: urlParts[2],
      text:    urlParts[3]
    };

    var response = {email: email};
    res.writeHead(200, {'Content-Type': 'application/json'});
    res.end(JSON.stringify(response, null, 2));

    process.send(email);
  };

  http.createServer(server).listen(httpPort);
};
```

IPC!

```
var doMasterStuff = function(){
  log('master', 'started master');

  var masterLoop = function(){
    checkOnWebServer();
    checkOnEmailWorker();
  };

  var checkOnWebServer = function(){ ...
  };

  var checkOnEmailWorker = function(){ ...
  };

  setInterval(masterLoop, 1000);
};
```

```
var checkOnWebServer = function(){
  if(children.server === undefined){
    log('master', 'starting web server');
    children.server = cluster.fork({ROLE: 'SERVER'});
    children.server.name = 'web server';
    children.server.on('online',    function(){ log(children.server, 'ready on port ' + httpPort); });
    children.server.on('exit',      function(){
      log(children.server, 'died :(');
      delete children.server;
    });
    children.server.on('message',   function(message){
      log(children.server, 'got an email to send from the webserver: ' + JSON.stringify(message));
      children.worker.send(message);
    });
  }
};
```

It's really all just message passing and monitoring…

```
var checkOnEmailWorker = function(){
  if(children.worker === undefined){
    log('master', 'starting email worker');
    children.worker = cluster.fork({ROLE: 'WORKER'});
    children.worker.name = 'email worker';
    children.worker.on('online',    function(){ log(children.worker, 'ready!'); });
    children.worker.on('exit',      function(){
      log(children.worker, 'died :(');
      delete children.worker;
    });
    children.worker.on('message',   function(message){
      log(children.worker, JSON.stringify(message));
    });
  }
};
```

It's really all just message passing and monitoring…

```
var doWorkerStuff = function(){
  process.on('message', function(message){
    emails.push(message);
  });

  var sendEmail = function(to, subject, text, callback){
    var email = {
      from:    require('./.emailUsername'),
      to:      to,
      subject: subject,
      text:    text,
    };

    transporter.sendMail(email, function(error, info){
      callback(error, email);
    });
  };

  var workerLoop = function(){
    if(emails.length === 0){
      setTimeout(workerLoop, 1000);
    }else{
      var e = emails.shift();
      process.send({msg: 'trying to send an email...'});
      sendEmail(e.to, e.subject, e.text, function(error){
        if(error){
          emails.push(e); // try again
          process.send({msg: 'failed sending email, trying again :('});
        }else{
          process.send({msg: 'email sent!'});
        }
        setTimeout(workerLoop, 1000);
      });
    }
  };

  workerLoop();
};
```

Message Queue!

Throttling!

Retry!

IPC!

Strategy 3: Local Messages

- Notes:
  - the children never log themselves
    - the master does it for them
  - Each process has it’s own “main” loop:
    - web server
    - worker
    - master
  - AND we can kill the child processes…

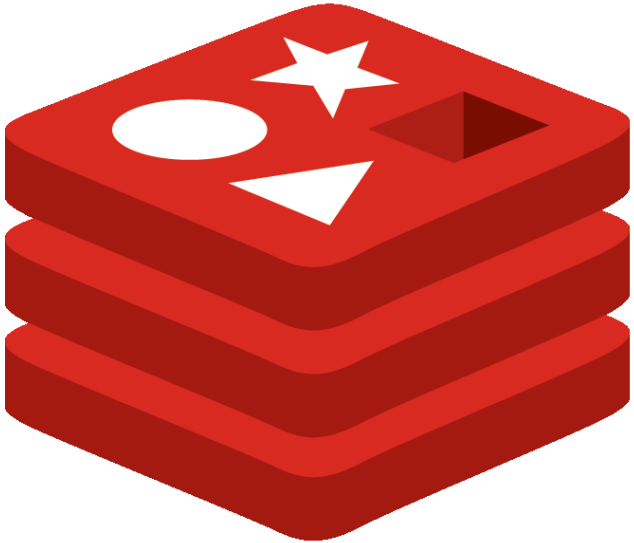
Strategy 3: Local Messages

- Why it is better in node:
  - In ~100 lines of JS...
    - Messages aren’t lost when server dies
    - Webserver process unbound by email sending
    - Error handling, Throttling, Queuing and retries!
    - Offline support?
- Why it is still a bad idea:
  - Bound to one host

Strategy 5: Remote Queues

- Observability
  - how long is the queue?
  - how long does an item wait in the queue?
  - ops stuff
- Redundancy
  - Backups
  - Clustering
  - ops stuff

A Quick Aside



redis

REDIS IS REALLY AWESOME

Data Structures for a MVP Queue:

- Array
  - Push, Pop, Length

I guess that’s it...

Data Structures for a <u>good</u> Queue:

- Array
  - Push, Pop, Length
- Hash (key types: string, integer, hash)
  - Set, Get, Exists
- Sorted Set
  - Exists, Add, Remove

Data Structures for a Good Queue

RESQUE (**node-resque**)

```
var queue = new NR.queue({connection: connectionDetails}, jobs, function(){
  queue.enqueue('math', "add", [1,2]);
  queue.enqueue('math', "add", [2,3]);
  queue.enqueueIn(3000, 'math', "subtract", [2,1]);
});
```

Data Structures for a Good Queue

redis Commander

Refresh | Commands | More...

- 127.0.0.1:6379:0
  - resque:* (2)
    - queue:* (1)
      - emailQueue (1)
    - queues (1)

Add New Value... | Delete Key | 0 | Goto Index

#	Value
0	{"class":"sendEmail","queue":"emailQueue","args": [{"to":"evantahler@gmail.com","subject":"hello_from_node","text":"hello_again"}]}

Queues

The list below contains all the registered queues with the number of jobs currently in the queue. Select a queue from above to view all jobs currently pending on the queue.

Name	Jobs
emailQueue	1
failed	0

0 of 0 Workers Working

The list below contains all workers which are currently running a job.

	Where	Queue	Processing
Nothing is happening right now...			

Live Poll

```
var jobs = {
  sendEmail: function(data, callback){
    var email = {
      from:    require('./.emailUsername'),
      to:      data.to,
      subject: data.subject,
      text:    data.text,
    };

    transporter.sendMail(email, function(error, info){
      callback(error, {email: email, info: info});
    });
  }
};
```

```
if(process.env.ROLE === 'web'){
  var queue;

  var server = function(req, res){
    var urlParts = req.url.split('/');
    var email    = {
      to:      urlParts[1],
      subject: urlParts[2],
      text:    urlParts[3]
    };

    queue.enqueue('emailQueue', "sendEmail", email);

    var response = {email: email};
    res.writeHead(200, {'Content-Type': 'application/json'});
    res.end(JSON.stringify(response, null, 2));
  };

  queue = new NR.queue({connection: connectionDetails}, jobs, function(){
    http.createServer(server).listen(httpPort);
    console.log('server running');
  });
}
```

IPC!

Connect before server start

```
if(process.env.ROLE === 'worker'){
  var worker = new NR.worker({connection: connectionDetails, queues: ['emailQueue']}, jobs, function(){
    worker.workerCleanup();
    worker.start();
  });

  worker.on('start',           function(){ console.log("worker started"); });
  worker.on('end',             function(){ console.log("worker ended"); });
  worker.on('cleaning_worker', function(worker, pid){ console.log("cleaning old worker " + worker); });
  worker.on('poll',            function(queue){ console.log("worker polling " + queue); });
  worker.on('job',             function(queue, job){ console.log("working job " + queue + " " + JSON.str
  worker.on('reEnqueue',       function(queue, job, plugin){ console.log("reEnqueue job (" + plugin + ")
  worker.on('success',         function(queue, job, result){ console.log("job success " + queue + " " +
  worker.on('failure',         function(queue, job, failure){ console.log("job failure " + queue + " " +
  worker.on('error',           function(queue, job, error){ console.log("error " + queue + " " + JSON.st
  worker.on('pause',           function(){ console.log("worker paused"); });
}
```

Really Simple

Tons of optional status events

So what is special about node.js here?

Queue Workers @ Node

- The event loops is great for processing **all** non-blocking events, not just web servers.
- Most Background jobs are non-blocking events
  - Update the DB, Talk to this external service, etc
- So node can handle **many** of these at once per process!

```
var multiWorker = new NR.multiWorker({
  connection: connectionDetails,
  queues: ['slowQueue'],
  minTaskProcessors:   1,
  maxTaskProcessors:   100,
  checkTimeout:        1000,
  maxEventLoopDelay:   10,
  toDisconnectProcessors: true,
}, jobs, function(){

  // normal worker emitters
  multiWorker.on('start',              function(workerI
  multiWorker.on('end',                function(workerI
  /// ...

  multiWorker.on('internalError',      function(error){
  multiWorker.on('multiWorkerAction', function(verb, d

  multiWorker.start();
});
```

How can you tell the CPU is pegged?

```
// inspired by https://github.com/tj/node-blocked

module.exports = function(limit, interval, fn) {
  var start = process.hrtime();

  setInterval(function(){
    var delta = process.hrtime(start);
    var nanosec = delta[0] * 1e9 + delta[1];
    var ms = nanosec / 1e6;
    var n = ms - interval;
    if (n > limit){
      fn(true, Math.round(n));
    }else{
      fn(false, Math.round(n));
    }
    start = process.hrtime();
  }, interval).unref();
};
```

+ **process.setImmediate()**

Example Time!

```
var blockingSleep = function(naptime){
  var sleeping = true;
  var now = new Date();
  var alarm;
  var startingMSeconds = now.getTime();
  while(sleeping){
    alarm = new Date();
    var alarmMSeconds = alarm.getTime();
    if(alarmMSeconds - startingMSeconds > naptime){ sleeping = false }
  }
}
```

```
var jobs = {
  "slowSleepJob": {
    plugins: [],
    pluginOptions: {},
    perform: function(callback){
      var start = new Date().getTime();
      setTimeout(function(){
        callback(null, (new Date().getTime() - start) );
      }, 1000);
    },
  },
  "slowCPUJob": {
    plugins: [],
    pluginOptions: {},
    perform: function(callback){
      var start = new Date().getTime();
      blockingSleep(1000);
      callback(null, (new Date().getTime() - start) );
    },
  },
};
```

Strategy 5: Remote Queues

- Why it is better in node:
  - In addition to persistent storage and multiple server/process support, you get get CPU scaling and Throttling very simply!
  - Node also has tooling (domains) around async exceptions which other languages lack
    - Integrates well with the resque/sidekiq pattern
- This might finally be a good idea!

THANKS!

- These Slides
  - goo.gl/yUuApo
- Supporting Project:
  - https://github.com/evantahler/background_jobs_node
- Node-Resque:
  - https://github.com/taskrabbit/node-resque
  - MultiWorker Example

Bonus Slides to follow

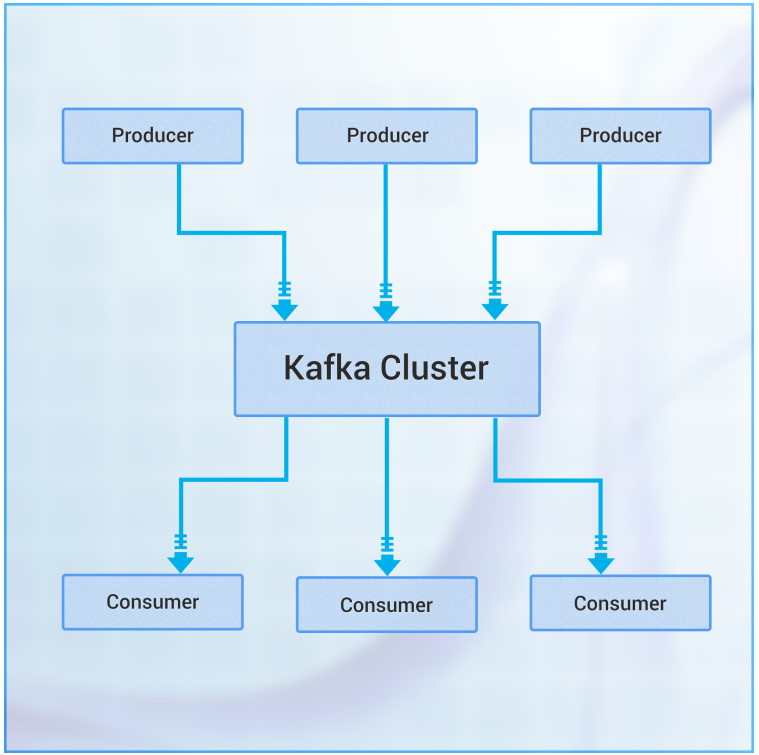
Strategy 4: Remote Messages

- Styles:
  - **Synchronous-processing**
    - Can provide messaging to the client about success
    - But the client still has to wait...
  - **Asynchronous-processing**
    - Just like our cluster example, but now we can separate **servers** and not just **processes**

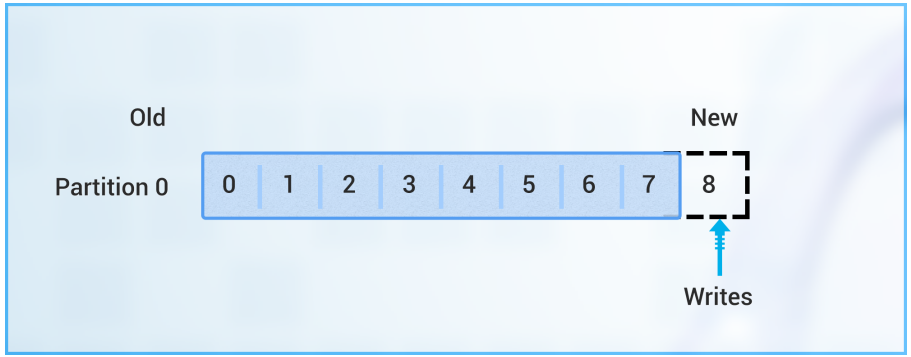
Strategy 4: Remote Messages

- **Synchronous-processing** doesn't seem help too much (unless there are *OPS* considerations)
- How can we build a persistent **Asynchronous-processing** app?
  - We'll need that app to respond with status
    - Job Started, job failed, job succeeded...
  - We'll use a **Remote Queue!**

Strategy 6: Event Bus



(Watch or Poll) vs Push



http://blog.qburst.com/2014/06/apache-kafka/