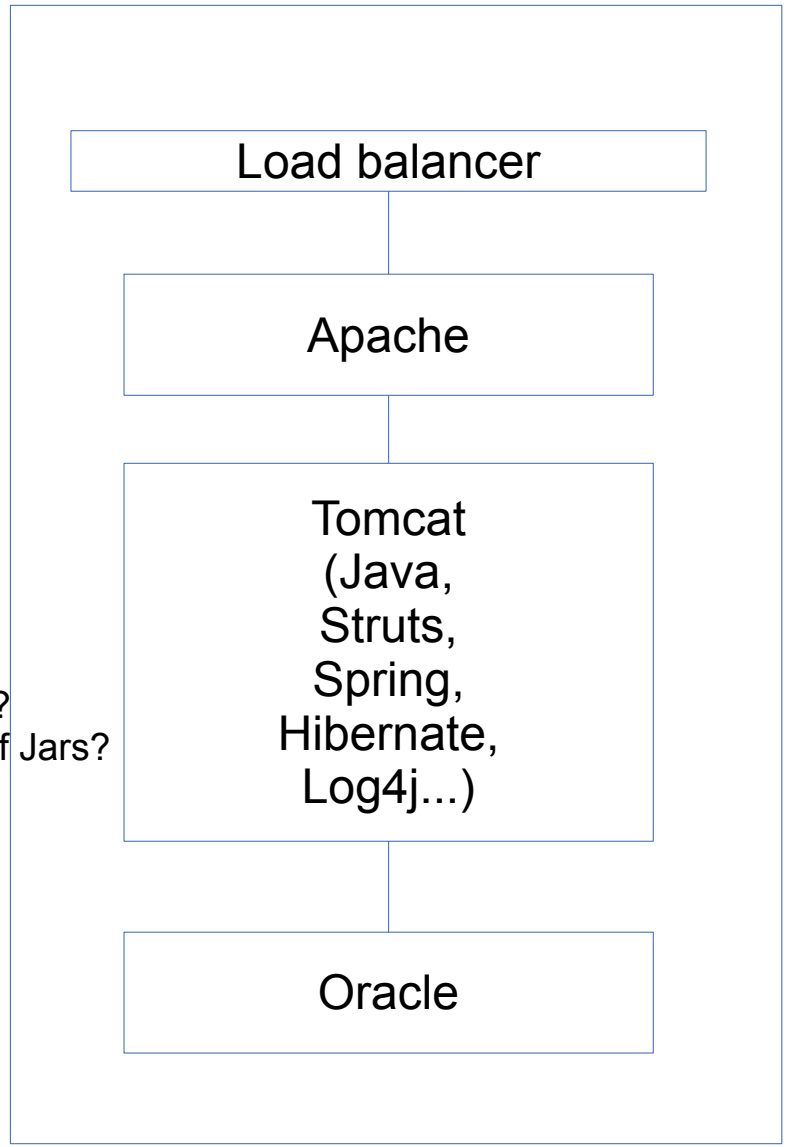
NodeJS, MongoDB, AWS: pitfalls and issues

Dave Cleal

Technology stack

- Computer program written in some language
- Some libraries
- Database to put the data in
- Other stuff (computers, load balancers, firewalls, message queues, networks, scaling up, scaling down, running out of space, DNS, deployment processes, certificates, ...)

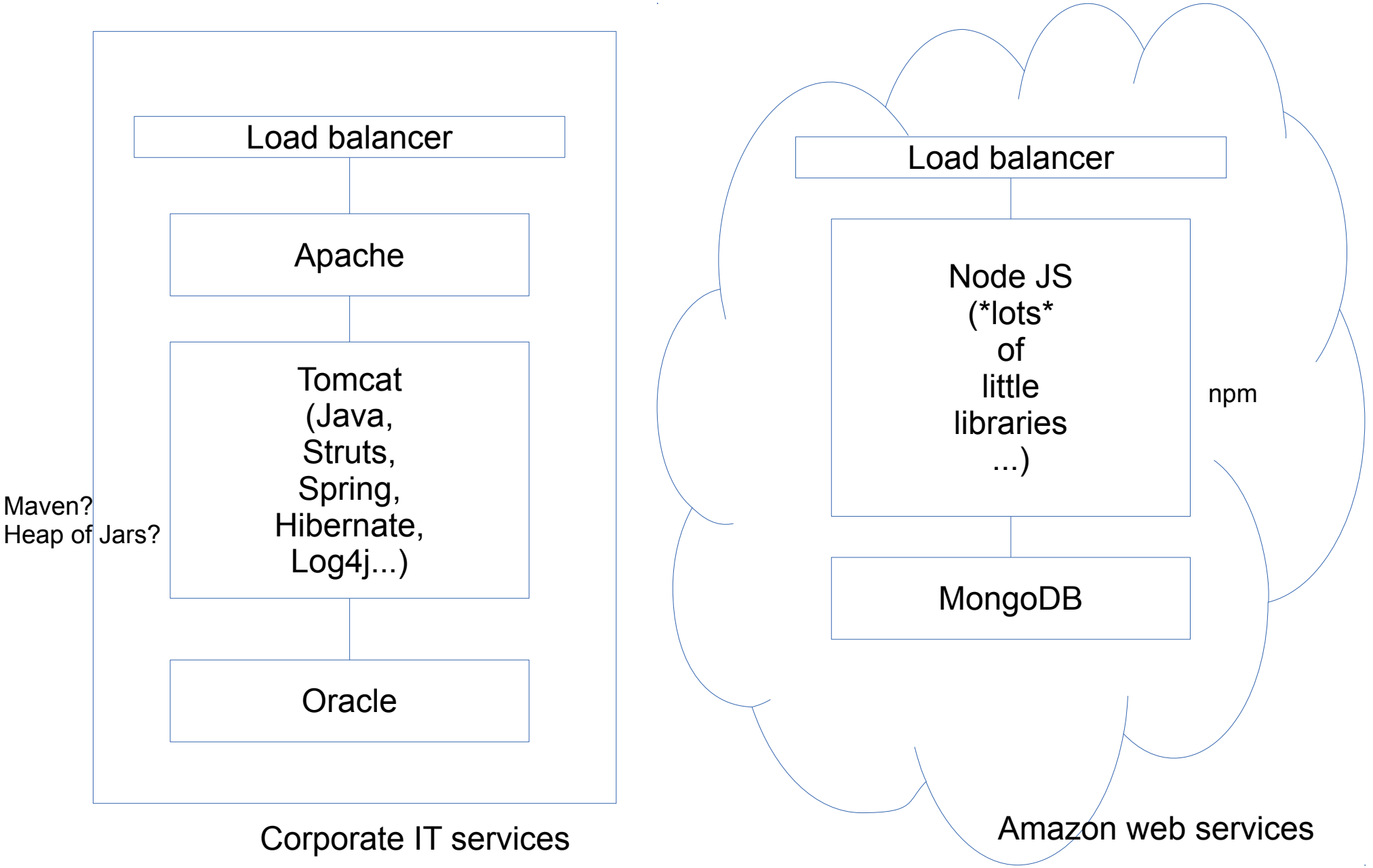
Tech stack



Maven?
Heap of Jars?

Corporate IT services

Tech stack


Load balancer
Apache
Tomcat
(Java,
Struts,
Spring,
Hibernate,
Log4j...)
Oracle
Maven?
Heap of Jars?
Corporate IT services
Load balancer
Node JS
(*lots*
of
little
libraries
...)
npm
MongoDB
Amazon web services

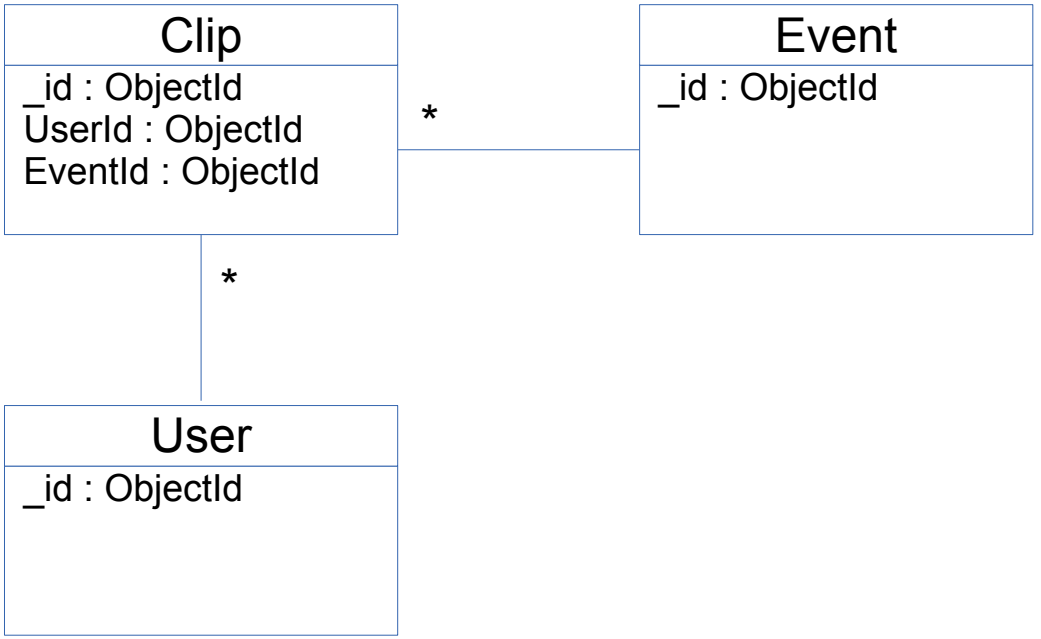
AWS – what's not to like

- It's all good
- CDN needs a bit of thought

Mongo DB

- Essentially a document store without SQL or transactions
- Fast
- Scalable
- Geospatial queries
- Data can be unstructured, or partly structured – you can index fields that only some of the records have, for example
- Did I mention that it's fast

joins

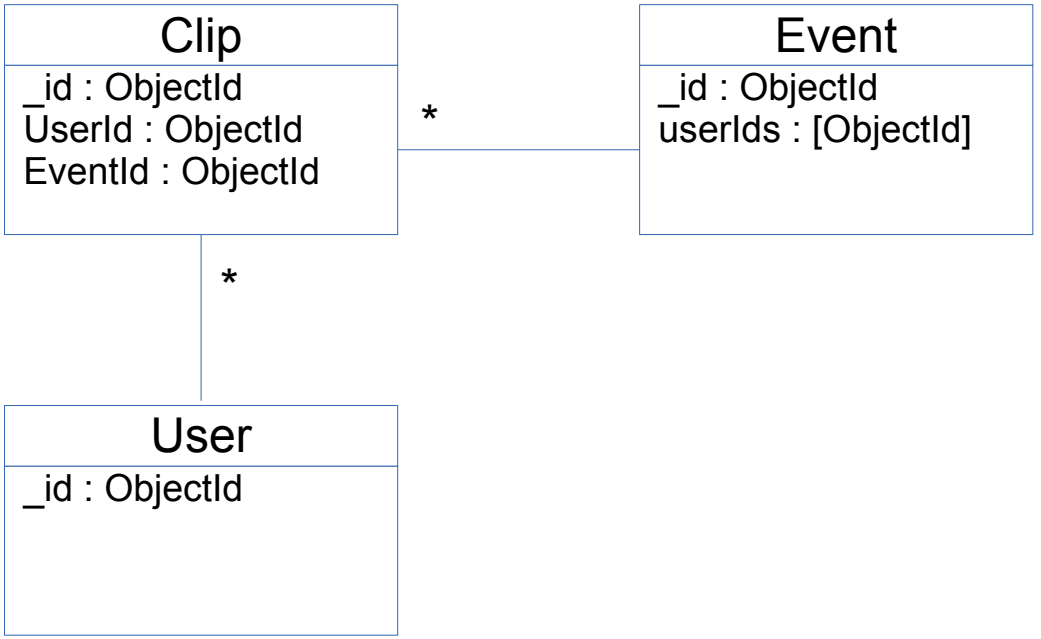


Select distinct (e.id) from clip c inner join event e on c.eventId = e.id
where clip.userId = <theUserId>

db.clips.find({userId : <theUserId>}, {eventId : 1}) → <aSetOEventIds>

db.events.find({ _id : { $in : <aSetOfEventIds> } })

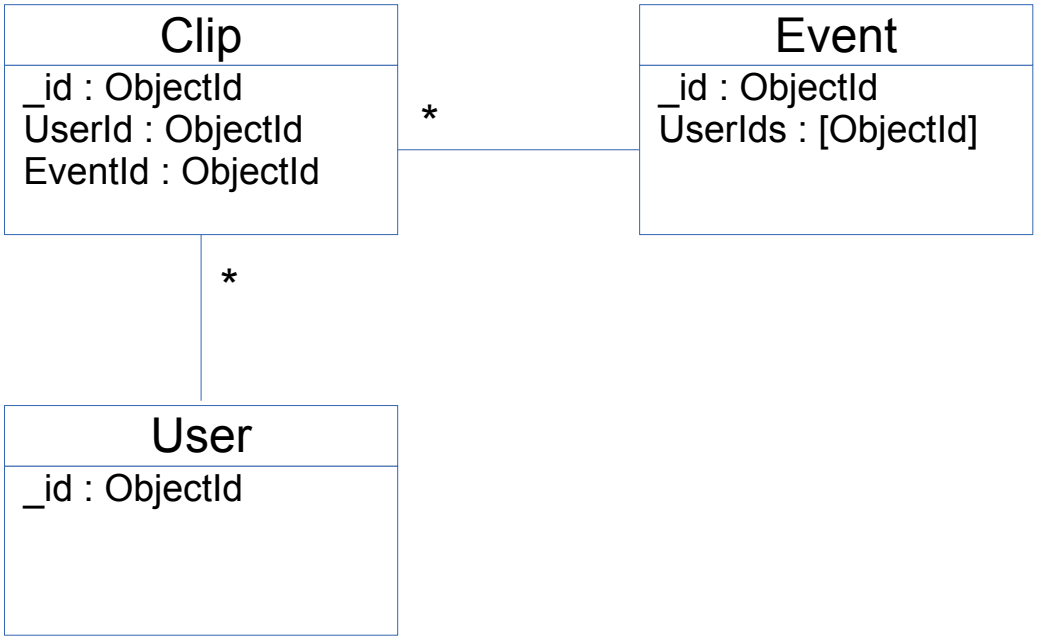
denormalise



Select distinct (e.id) from clip c inner join event e on c.eventId = e.id
where clip.userId = <theUserId>

db.events.find({ userIds : <theUserId> })

deletion



delete * from clip where _id = <theClipId>

db.clips.find({ id : <theClipId> }, { eventId : 1, userId : 1 })
→ <theEventId>, <theUserId>

db.events.update({ id : <theEventId> }, { $pull : { userIds : <theUserId> } })

db.clips.remove({ id : <theClipId> })

MongoDB – transactional hoops

- upserts
- Mongoose version incrementing
- findByIdAndRemove (and return the old one)
- findOneAndUpdate (return new or old)
- deletion and race conditions

MongoDB – lessons learnt

- Immutability is good
- Nested documents are good
- It's really fast – don't prematurely optimise – don't worry too much about the number of queries
- Transaction tricks

Node JS

- Event-driven, non-blocking I/o
- Continuation passing
- Fast
- Active community with small, focussed libraries served from **github**.
- Simple parallelism

Node JS - style

```
function confirmPartUploaded(uploadId, partNumber, etagCode, callback) {
    access.storeEtag(uploadId, partNumber, etagCode, function(err) {
        access.tryToLockForUpload(uploadId, function(err, gotLock) {
            if (gotLock) {
                access.findUploadById(uploadId, function(err, upload) {
                    s3.endUpload(upload, function(err) {
                        if (err && err.invalidPart) {
                            access.removeInvalidPart(upload, err.invalidPart, function(e
                        } else {
                            callback();
                        }
                    });
                });
            } else {
                callback();
            }
        });
    });
}
```

Node JS - style

```
function confirmPartUploaded(uploadId, partNumber, etagCode, callback) {
	access.storeEtag(uploadId, partNumber, etagCode, tryForLock);

	function tryForLock(err) {
		access.tryToLockForUpload(uploadId, testLock);
	}
	function testLock(err, gotLock) {
		if (gotLock) {
			access.findUploadById(uploadId, doEnd);
		} else {
			callback();
		}
	}
	function doEnd(err, upload) {
		s3.endUpload(upload, checkResults);
	}
	function checkResults(err) {
		if (err && err.invalidPart) {
			access.removeInvalidPart(upload, err.invalidPart, recordErrorThen(callback);
		} else {
			callback();
		}
	}
}
```

Node JS – error handling

```
function getJson(aHost, aPath, callback) {
  getUrl(aHost, aPath, parseJson);

  function parseJson(statusCode, responseBody) {
    callback(JSON.parse(responseBody));
  }
}
```

Node JS – error handling

```
function getJson(aHost, aPath, callback) {
  getUrl(aHost, aPath, parseJson);

  function parseJson(statusCode, responseBody) {
    try {
      callback(null, JSON.parse(responseBody));
    } catch (e) {
      callback(e);
    }
  }
}
```

Node JS – error handling

```
function getJson(aHost, aPath, callback) {
  getUrl(aHost, aPath, parseJson);

  function parseJson(err, statusCode, responseBody) {
    if (err) {
      callback(err);
    } else {
      try {
        callback(null, JSON.parse(responseBody));
      } catch (e) {
        callback(e);
      }
    }
  }
}
```

Node JS – error handling

```
function getJson(aHost, aPath, callback, errorCallback) {
  var tryCatch = tryCatchFunc(errorCallback);

  tryCatch(getUrl)(aHost, aPath, tryCatch(parseJson), errorCallback);

  function parseJson(err, statusCode, responseBody) {
    handleErrors(err);
    if (statusCode !== 200) {
      throw new StatusCodeError(statusCode);
    } else {
      callback(JSON.parse(responseBody));
    }
  }
}
```

Node JS – promises

```
function getJson(aHost, aPath) {
  return getUrlP(aHost, aPath).then(
    function(results) {
      var statusCode = results[0]
        , responseBody = results[1];
      if (statusCode !== 200) {
        throw new Error("Unexpected status code: " + statusCode);
      } else {
        return JSON.parse(responseBody);
      }
    }
  );
}

function getUrlP(aHost, aPath) {
  var deferred = require('q').defer();
  getUrl(aHost, aPath, deferred.makeNodeResolver());
  return deferred.promise;
}
```

Node JS – promises - 2

```
...
getJson(host, path, successCallback, errorCallback);
...

...
getJson(host, path).then(successCallback, errorCallback);
...
```

Node JS – async – the commercial

- the best reason to skip promises is that you get to use async instead
- Parallel threads (in the OS) become an everyday occurrence serving a single request
- All the usual functional suspects (map, filter, reduce, etc)
- Asynchronous control flow: while, until, parallel, series, waterfall...
-

Node JS – async – example

```
async.parallel(
  [thumbAndSend, profileSource],
  tryCatch(postProcess));
```

```
async.mapLimit(users, doCount, 50, next);
```

```
function doCount(user, callback) {
  access.countLogins({
    userId: user._id,
    date: {$lt: dateUtil.addHours(user.timestamp, window)}
  }, callback);
}

function next(err, userCounts) {
  // ...
)
```

Node JS – domains

```
function getJson(aHost, aPath, callback) {
  getUrl(aHost, aPath, parseJson);

  function parseJson(statusCode, responseBody) {
    if (statusCode !== 200) {
      throw new StatusCodeError(statusCode);
    } else {
      callback(JSON.parse(responseBody));
    }
  }
}

...
var d = domain.create();
d.on('error', errorCallback);

d.run(function() {
  getJson(aHost, aPath, storeCuts);
});
...
```

Node JS – weak typing and weak IDE support

- Renaming operations aren't guaranteed to work correctly, nor are – for example – method extractions
- Weak navigation especially for common names.
- A deal-breaker (at least for me when I get hired into the fifth year and third generation of contractors in the corporate world).

Duplicate callbacks alert

```
function handleWorkResult(err, logString) {
    if (callbackCalledOnceAlready) {
        logger.error("Got a second callback processing message "
+ sqsMessage.body + " in queue " + queue.name);
        // not calling for another message because the first time
the callback was called should have taken care of that.
      } else {
        callbackCalledOnceAlready = true;
        thread.currentMessage = null;
         ...
```

Node JS – error handling

```
function getJson(aHost, aPath, callback) {
  getUrl(aHost, aPath, parseJson);

  function parseJson(err, statusCode, responseBody) {
    if (err) {
      callback(err);
    } else if (statusCode !== 200) {
      callback(new StatusCodeError(statusCode));
    } else {
      try {
        callback(null, JSON.parse(responseBody));
      } catch (e) {
        callback(e);
      }
    }
  }
}
```

Node JS – more on errors and logging

- Stack traces !
  - Try longjohn (but not on jade rendering)
- Just let the errors go
  - Not for web servers, but processors...

Further reading

- http://substack.net/node_aesthetic
- http://journal.paul.querna.org/articles/2011/12/18/the-switch-python-to-node-js/
- http://docs.mongodb.org/manual/tutorial/perform-two-phase-commits/
- http://docs.mongodb.org/manual/tutorial/isolate-sequence-of-operations/
- http://stella.laurenzo.org/2011/03/bulletproof-node-js-coding/
- http://howtonode.org/promises
- http://stackoverflow.com/questions/5683916/node-js-express-vs-geddy

Node JS – lessons learned

- Use the libraries, use github, use npm
- Name functions
- Choose an error handling strategy
- Use async!
- Be aware of limited refactoring