Google Cloud

Edge NodeJS

NodeJS Integration with Edge

NodeJS* runtime

NodeJS runtime on Edge is Trireme

```
http://github.com/apigee/
trireme
```

*Node.js is a trademark of Joyent, Inc. and is used with its permission. We are not endorsed by or affiliated with Joyent.

What is Trireme

Edge open-source project

Set of libraries for running node.js scripts inside JVM

Specifically designed to be embeddable within any Java program

- “HTTP Adapter” lets it run inside existing containers
- “Sandbox” restricts file and network I/O access

Why?

We wanted to add node.js capabilities to our existing product which is already built using Java

We wanted to use Java code from node.js

We didn't want to assemble a node.js PaaS

We wanted script isolation – there is no way for one script to affect the heap of others

We wanted sandboxed execution

- Prevent script from gaining access to file system and local network
- Limit execution time of a script – preventing infinite loops

NodeJS implementation

Node.js = JavaScript shell + native modules in C++

Trireme exposes Java modules that mimic the interfaces of the C++ native modules in node.js.

Architecture

One thread per Node.js application

- Async I/O handled via NIO within that thread

Additional thread pool for blocking operations

- File I/O
- DNS lookups

Replace native code from Node.js with Java alternatives

- Internal modules such as “tcp_wrap”, etc.

Implement a few popular native modules with Java code

- “iconv”, “node_xslt”, etc.

Why would you care

It is an open-source project that can be utilized outside Edge

- If you want to embed Node.js apps inside existing Java application
- If you want to run Node.js apps that take advantage of Java libraries you can't live without, e.g. JDBC, XML parsers, XSLT engines

As it is the Node.js runtime used, it has significant impact on design and architecture of your solution that you need to know about

- Node.js 0.10 is supported
- Uses Rhino (JavaScript implementation for JVM) which only implements JavaScript 1.8
- Newer features of V8, particularly the primitive array types, are not supported in Rhino
- Trireme does not support 100% Node.js APIs

NodeJS vs. Trireme

NodeJS

- Single-threaded event engine
  - Non-blocking TCP I/O
  - Non-blocking UDP datagrams
  - Non-blocking File I/O
  - Timers
- “Buffer” object
- Module loading system
- Utility modules
- Third-party components
  - V8 JavaScript engine
  - OpenSSL
  - ZLib

Trireme

- Single-threaded event engine
  - Non-blocking TCP I/O
  - Non-blocking UDP datagrams
  - Non-blocking File I/O
  - Timers
- “Buffer” object
- Module loading system
- Utility modules
- Third-party components
  - Rhino JavaScript engine
  - Java SE
  - Bouncy Castle (crypto, optional)

Compatibility

Can’t load native code (can’t load C code)

https://github.com/apigee/trireme#how-complete-is-trireme

Module	Status	Source
assert	Complete	node.js
child_process	Partial	Trireme
cluster	Not Implemented Yet	node.js
console	Complete	node.js
crypto	Complete	node.js + Trireme
debugger	Not Supported	
dgram	Complete	node.js + Trireme
dns	Partial	Trireme
domain	Complete	node.js + Trireme
events	Complete	node.js
fs	Complete	node.js + Trireme
globals	Complete	node.js + Trireme
http	Complete	node.js + Trireme
https	Complete but See Notes	Trireme
module	Complete	node.js

Edge restrictions

Script to write files or to read outside of the current local directory tree where it was deployed

Script to listen on an incoming port (but it can open outgoing ports all it wants)

child_process

cluster

debugger

dgram

readline

repl

tty

NodeJS as a Target Endpoint

Target Endpoint

```
<HTTPTargetConnection>

  <Properties>
    <Property name="success.codes">2XX,3XX,4XX</
Property>
  </Properties>

  <LoadBalancer>
    <Algorithm>RoundRobin</Algorithm>
    <Server name="server1"/>
    <Server name="server2"/>
  </LoadBalancer>

  <Path>/resource</Path>

</HTTPTargetConnection>
```

NodeJS

```
<ScriptTarget>

  <Properties>
    <Property name="success.codes">2XX,3XX,4XX</
Property>
  </Properties>

  <ResourceURL>node://app.js</ResourceURL>

</ScriptTarget>
```

NodeJS as a Target Endpoint

Other languages are extension policies and placed on the flow as a step

- Java
- JavaScript
- Python

Node.js implemented as a target not as a policy

NodeJS as a Target Endpoint

```
<ProxyEndpoint>
  …

  <RouteRule name="node">
    <TargetEndpoint>node</
TargetEndpoint>
  </RouteRule>

</ProxyEndpoint>
```

```
<TargetEndpoint name="node">
  <Description>Node Target</Description>

  <ScriptTarget>
    <Properties>
      <Property name="success.codes">2XX,3XX,4XX</Property>
    </Properties>

    <EnvironmentVariables>
      <!-- process.env.var -->
      <EnvironmentVariable name="var">VALUE</
EnvironmentVariable>
    </EnvironmentVariables>

    <Arguments>
      <Argument>argument</Argument>
    </Arguments>

    <ResourceURL>node://app.js</ResourceURL>
  </ScriptTarget>

</TargetEndpoint>
```

NodeJS as a Target Endpoint

```
<ProxyEndpoint>
  …

  <RouteRule name="node">
    <Condition>proxy.pathsuffix MatchesPath “/
queue”</Condition>
    <TargetEndpoint>node</TargetEndpoint>
  </RouteRule>

  <RouteRule name="directToBackend">
    <TargetEndpoint>backend</TargetEndpoint>
  </RouteRule>

</ProxyEndpoint>
```

Apigee Access

apigee-access open source Node.js module

Provides Node.js applications running on the Edge platform a way to access Edge specific functionality. You can use this module to:

Access and modify "flow variables" within the Edge message context.

Retrieve sensitive data from the the secure store.

Use the built-in distributed cache.

Use the built-in distributed quota service.

Use the OAuth service.

http://apigee.com/docs/api-services/content/using-apigee-access

```
https://github.com/apigee/apigee-access
```

When to use NodeJS?

Try out of the box policies first

- Configuration over code, aka policy over code
- Speed / Agility
  - Let's implement a distributed quota
  - Use an Edge policy implement from scratch
- Quality
  - "Developed/Tested" once, "configured" everywhere
  - Maintained centrally

```
<Quota name="Quota.DailyPerApp">
    <Interval>1</Interval>
    <TimeUnit>day</TimeUnit>
    <Distributed>true</Distributed>
    <Synchronous>true</Synchronous>
    <Identifier
ref="request.queryparam.apikey" />
    <Allow count="100" />
</Quota>
```

When to use NodeJS?

Then consider callout policies

- Simple functionality which can be visualized as a step?
  - Get/set variable
  - Make custom/simple modifications to request/response?
  - Request/response data validations?
- Edge supports Java, JavaScript, Python as callout policies

```
//offset parameter validation
var offset =
context.getVariable('request.queryparam.offset');
if (offset != null) {
    if (offset < 1) {
        context.setVariable('errorCode',
'400.02.001');
        context.setVariable('errorMessage', 'offset
query parameter value is invalid');
    }
}
```

When to use NodeJS?

Consider Node.js if

- Edge doesn't have an existing policy to do the work
- You need intelligent asynchronous processing logic
- Implementation can be visualized as an API proxy target, rather than as a step
- Legacy backend protocols (non HTTP)

Non-http backend

Async execution

Complex mashups

Job scheduling

Bulk operations

Retry logic

Mockups

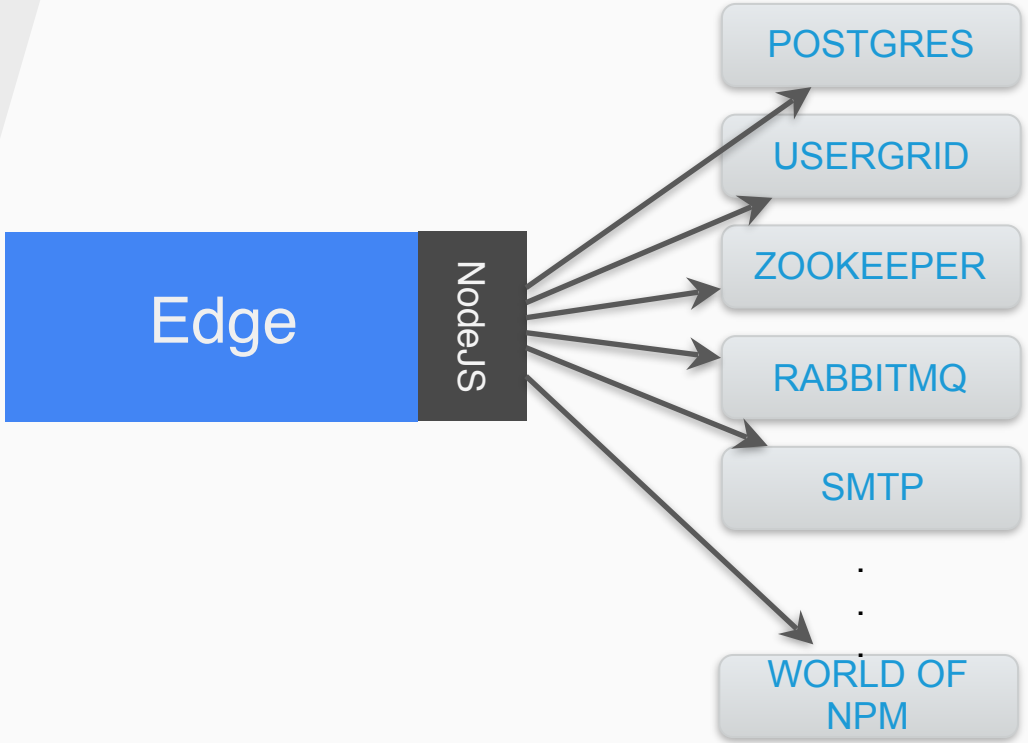
User interface

Quick demonstrations

PoC development

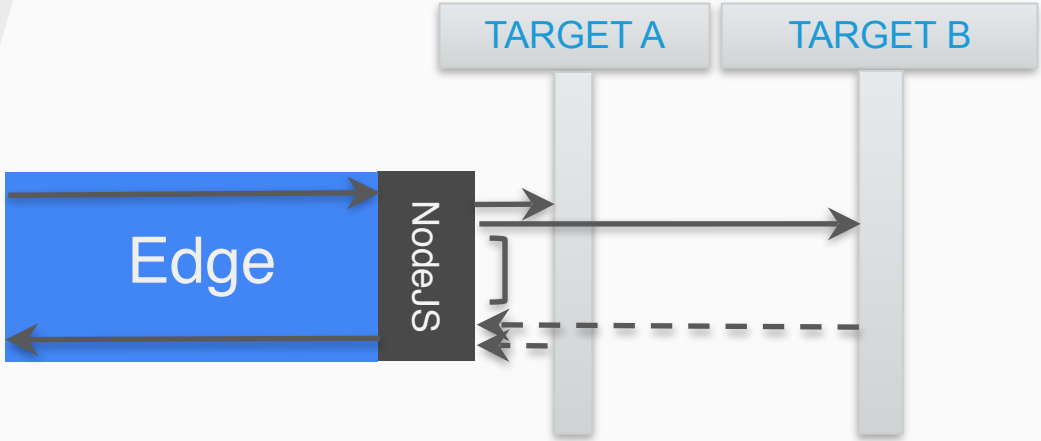
When to use NodeJS?

Non-HTTP backend



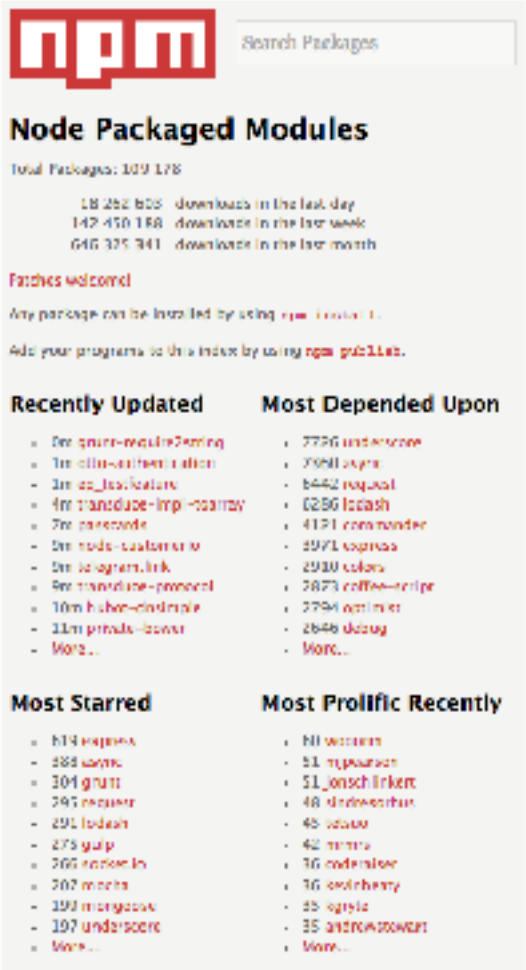
When to use NodeJS?

Async operations



When to use NodeJS?

Module reuse



When to use NodeJS?

Module reuse - async example

```
async.waterfall([
    function(callback){
        callback(null, 'one', 'two');
    },
    function(arg1, arg2, callback){
      // arg1 now equals 'one' and arg2 now equals
'two'
        callback(null, 'three');
    },
    function(arg1, callback){
        // arg1 now equals 'three'
        callback(null, 'done');
    }
], function (err, result) {
   // result now equals 'done'
});
```

```
async.parallel([
    function(callback){
        setTimeout(function(){
            callback(null, 'one');
        }, 200);
    },
    function(callback){
        setTimeout(function(){
            callback(null, 'two');
        }, 100);
    }
],
// optional callback
function(err, results){
    // the results array will equal ['one','two'] even
though
    // the second function had a shorter timeout.
});
```

When NOT to use NodeJS?

There are existing Edge policies that can do the job

Cannot be visualized as an API proxy “target”

Node.js is implemented as a target, so

- You cannot execute an Edge policy in the middle of your Node.js script
- You cannot execute full Node.js script multiple times

Testing NodeJS

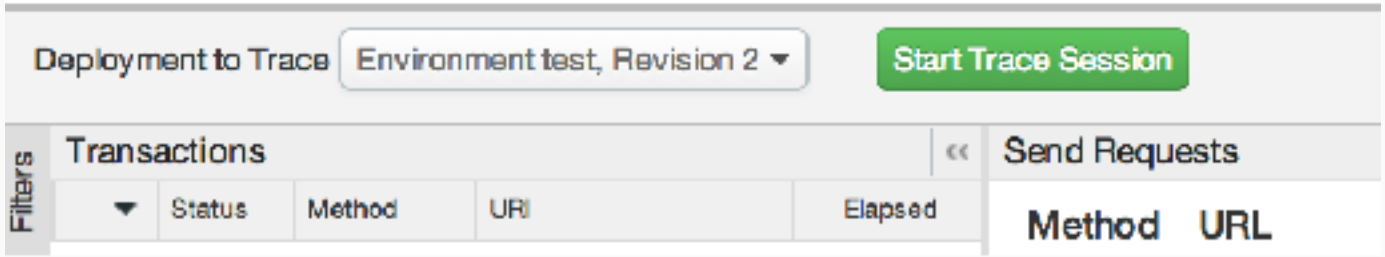
Testing Node.js applications

- Grunt.js
- Mocha
- Chai

TDD

Troubleshooting NodeJS

Troubleshooting Node.js using the Edge Trace tool { hands on }



Inspecting requests, responses, and HTTP status codes

console.log()

Troubleshooting NodeJS

Handling dependencies and dependency conflicts

- Express 3.7 and Connect 3.0 (deprecation notices and removal of middleware)
- Conflicting module versions in Edge

Handling response errors and node script failures/crashes

- Error handling in JavaScript (try/catch)
- Syntax errors in Node.js
- nodejitsu forever

Troubleshooting NodeJS

The node-inspector tool

- Browser-based Node.js debugger
- Navigate in your source files
- Set breakpoints (and specify trigger conditions)
- Step over, step in, step out, resume (continue)
- Inspect scopes, variables, object properties

IntelliJ IDEA

- A feature-rich GUI builder for Node.js
- (Should be) free with the community edition

Troubleshooting NodeJS

Getting help

StackOverflow has an incredible wealth of Node.js knowledge

How to ask: http://stackoverflow.com/help/how-to-ask

Edge docs

http://apigee.com/docs/api-services/content/getting-started-nodejs-apigee-edge

THANK YOU