IBM API Connect

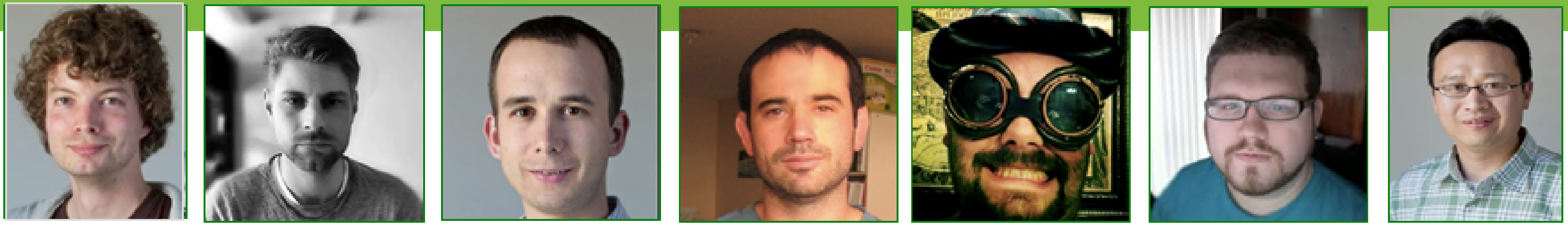
Building REST APIs in LoopBack

Asim Siddiqui | @asimASiddiqui

IBM apiconnect

Node Core

LoopBack & Express Core



Ben Noordhuis

Bert Belder

Miroslav Bajtos

Sam Roberts

Ryan Graham

Ritchie Martori

Raymond Feng

5 of the top 10 Node Contributors

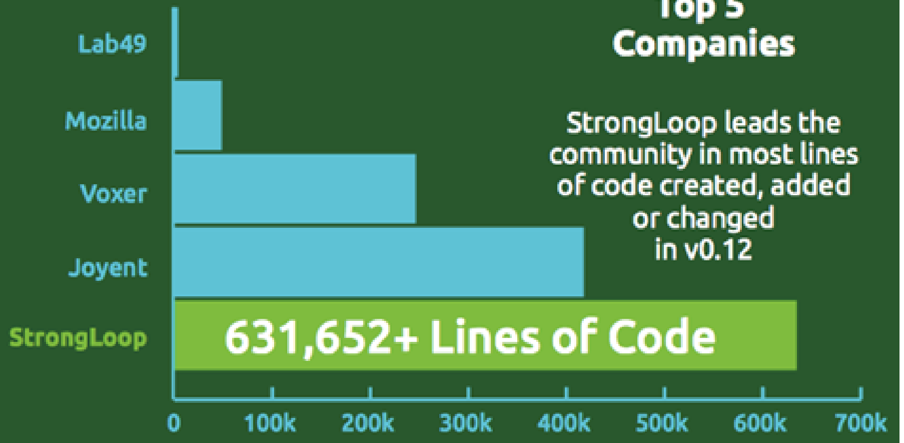
StrongLoop developers made **420+ Commits** to Node v0.12

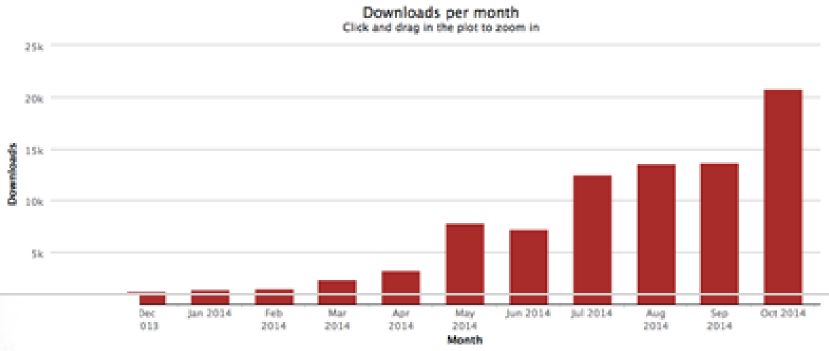
1	Ben Noordhuis
2	Fedor Indutny
3	Isaac Schlueter
4	Timothy Fontaine
5	Trevor Norris
6	Bert Belder
7	Domenic Denicola
8	Miroslav Bajtos
9	Sam Roberts
10	Ryan Graham

GitHub: March 28, 2013 - July 15, 2014

Top 5 Companies

StrongLoop leads the community in most lines of code created, added or changed in v0.12





LoopBack

Express

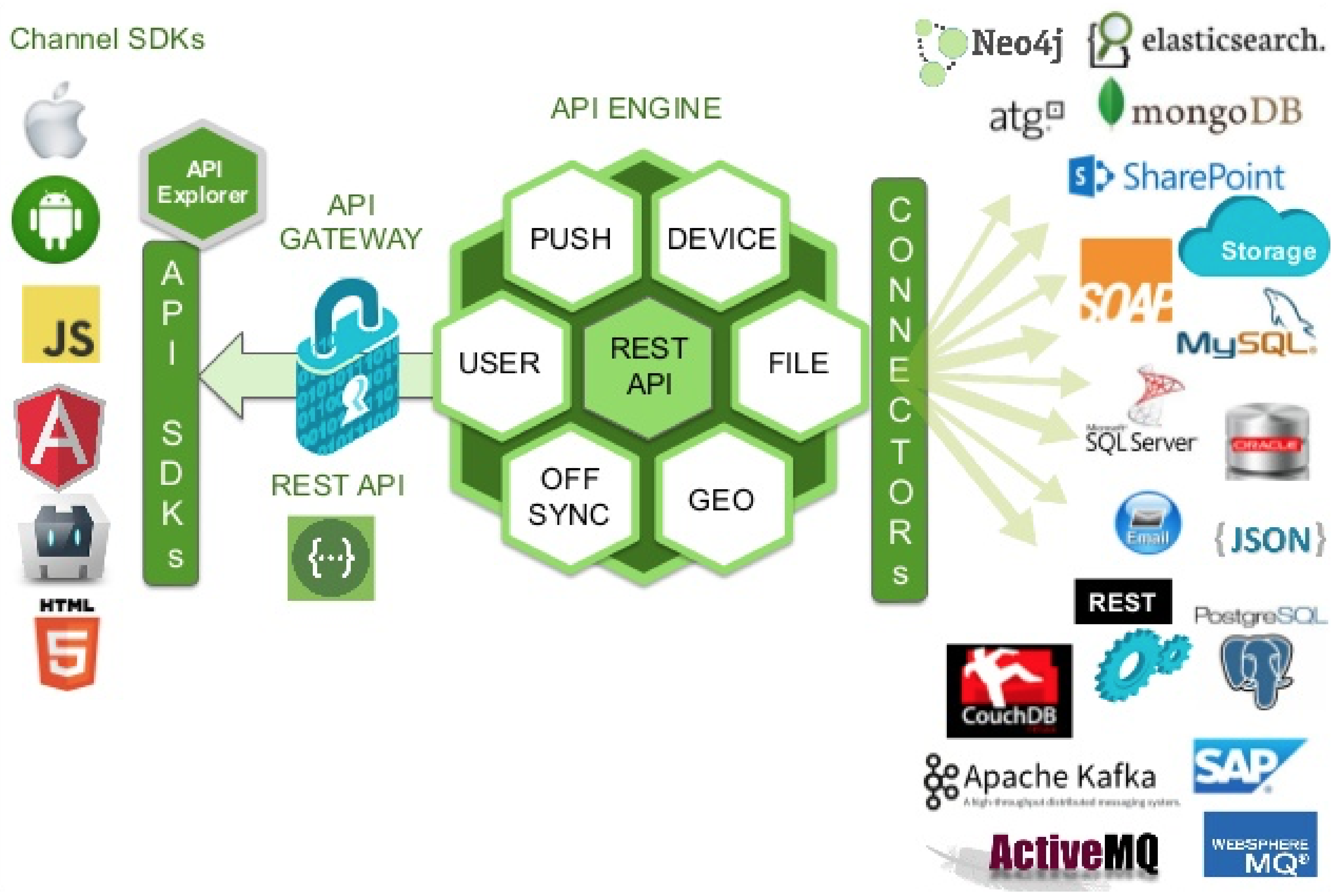
io

StrongLoop

What is LoopBack?

ORM, MBaaS, and API Server

- Model Driven Development
  - Multiple data sources
- Automatic REST endpoint generation
  - Includes routing & CRUD
- Built on Express
  - Create custom API Routes
  - Add customer middleware


Channel SDKs
JS
HTML
API Explorer
API SDKs
API GATEWAY
REST API
API ENGINE
PUSH
DEVICE
USER
REST API
FILE
OFF SYNC
GEO
CONNECTORS
Neo4j
elasticsearch.
atg
mongoDB
SharePoint
Storage
SOAP
MySQL
SQL Server
ORACLE
Email
{JSON}
REST
PostgreSQL
CouchDB
Apache Kafka
SAP
ActiveMQ
WEBSPHERE MQ

What else does it provide?

- Rich model relations
- Access controls (built in token authentication)
- Geolocation, push notifications, offline sync
- Angular, Android, and iOS SDKs

Let's build a loopback App
(and learn along the way of course!)

First a little something about me

Asim Siddiqui | @asimASiddiqui is a

We will be building a sneaker review app.

Step 1: Install API Connect Developer Toolkit

```
~$ npm install -g apiconnect
```

And scaffold your application:

```
~$ apic loopback
  ? What's the name of your application? kickReviews
  ? Enter name of the directory to contain the project: kickReviews
  ? What kind of application do you have in mind? (Use arrow keys)
    empty-server (An empty LoopBack API, without any configured models
  ❯ hello-world (A project containing a controller, including a single
    notes (A project containing a basic working example, including a m
```

Application Scaffolding

1. Initialize project folder structure
2. Creating default JSON config files
3. Creating default JavaScript files
4. Install initial dependencies

What do we have?

```
my-app/
  |_ client          # used for client app (if any)
  |_ node_modules
  |_ server
    |_ boot                   # app startup scripts
    |_ component-config.json  # primary API config
    |_ config.json            # primary API config
    |_ datasources.json       # data source config
    |_ middleware.json        # middleware config
    |_ model-config.json      # LB model config
    |_ server.js              # Server start script
  |_ package.json
```

Basic config

```
{
  "restApiRoot": "/kicksApi",
  "host": "0.0.0.0",
  "port": 3000,
  "remoting": {
    "context": {
      "enableHttpContext": false
    },
    "rest": {
      "normalizeHttpPath": false,
      "xml": false
    },
    "json": { ... },
    "urlencoded": { ... },
    "cors": false,
    "errorHandler": {
      "disableStackTrace": false
    }
  }
```

Working with Data Models

IBM apiconnect

What is a model in Loopback?

- representation of data in backend systems such as databases, REST Services, or SOAP Services.
- By default generates both Node and REST APIs.
- has a Model Definition JSON file
- has a Model JS file

Model Definition JSON file

```
{
  "name": "Shoe",
  "base": "PersistedModel",
    "idInjection": true,
    "options": {
      "validateUpsert": true
    },
    "properties": {
      "brand": {
        "type": "string",
        "required": true,
        "default": "   "
      },
      "model": {
        "type": "string",
        "required": true,
        "default": "  "
      },
      "color": {
                 :ing",
```

Built-in Models

Some built-in Models

- PersistedModel
- ACL
- AccessToken
- Role
- RoleMapping
- User

Creating Models

Different ways…

1. CLI: `~/my-app$ apic create --type model`.
2. Config files in `/common/models`, etc
3. Programmatically in a boot script
4. Programmatically with model discovery from data source

Creating a Model via CLI

```
$ apic create --type model
[?] Enter the model name: shoe
[?] Select the data-source to attach Shoe to: (Use arrow keys)
❯ db (memory)
[?] Select model's base class: (Use arrow keys)
  Model
❯ PersistedModel
  ACL
[?] Expose shoe via the REST API? (Y/n) Y
[?] Custom plural form (used to build REST URL):
```

Creating a Model via CLI

```
[?] Property name: brand
   invoke   loopback:property
[?] Property type: (Use arrow keys)
❯ string
  number
  boolean
  object
  array
  date
  buffer
  geopoint
  (other)
[?] Required? (y/N)
```

Run the Application

```
~/my-app$ node .
Web server's listening at http://0.0.0.0:3000
```

What is this?

http://localhost:3000

```
{ started: "2015-04-19T19:32:17.263Z", uptime: 5.001 }
```

This is the default "root page", which displays API status.

Remember, our API is served from the `/kicksApi` path!

The Shoe Route

http://localhost:3000/api/shoes

```
[ ]
```

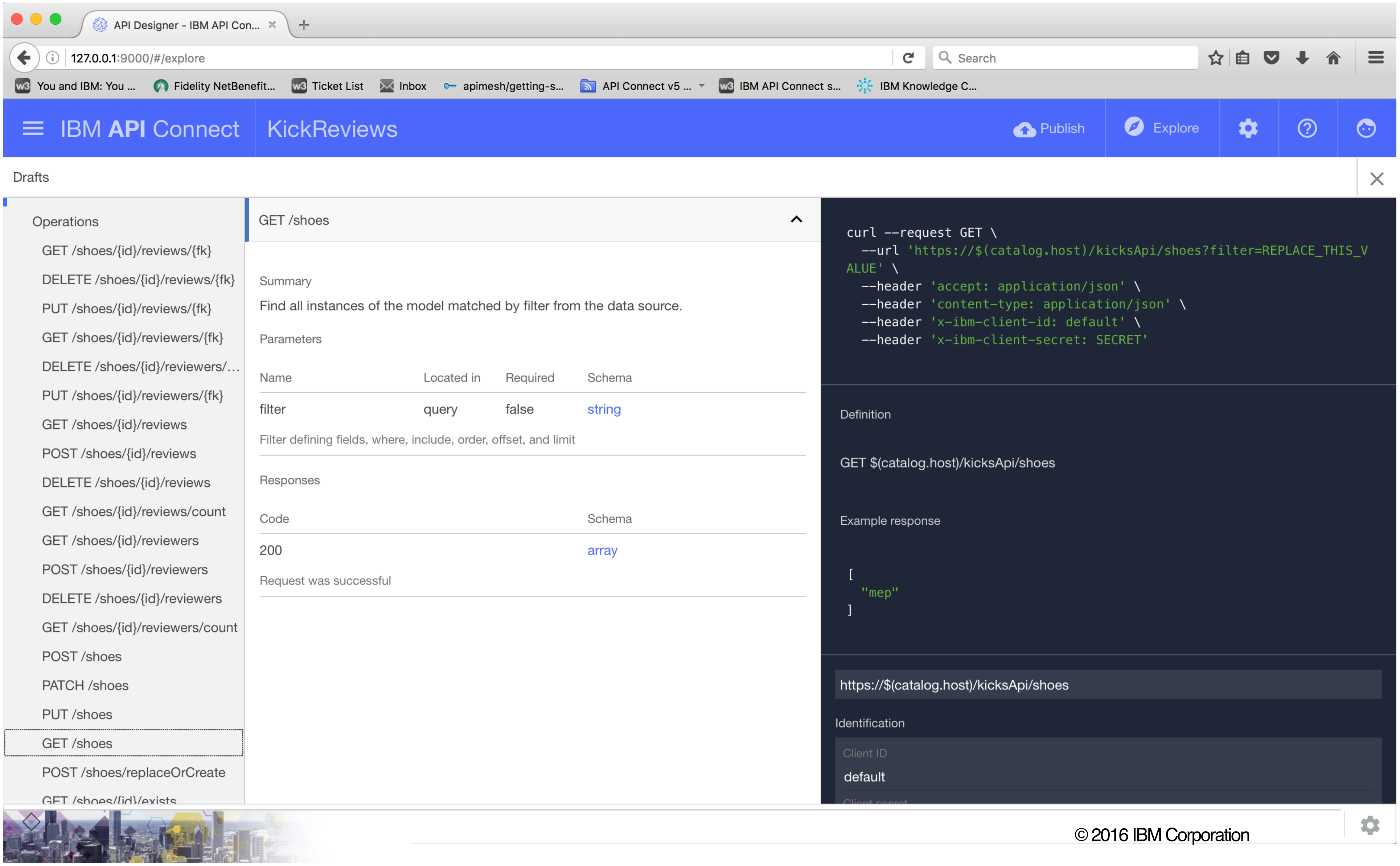
We don't have any kicks yet, but there we go!

View Models REST API Endpoints:

Start the API Designer to view the API endpoints

```
~/myapp$ apic edit
```

View Models REST API Endpoints:



Operations

- GET /shoes/{id}/reviews/{fk}
- DELETE /shoes/{id}/reviews/{fk}
- PUT /shoes/{id}/reviews/{fk}
- GET /shoes/{id}/reviewers/{fk}
- DELETE /shoes/{id}/reviewers/...
- PUT /shoes/{id}/reviewers/{fk}
- GET /shoes/{id}/reviews
- POST /shoes/{id}/reviews
- DELETE /shoes/{id}/reviews
- GET /shoes/{id}/reviews/count
- GET /shoes/{id}/reviewers
- POST /shoes/{id}/reviewers
- DELETE /shoes/{id}/reviewers
- GET /shoes/{id}/reviewers/count
- POST /shoes
- PATCH /shoes
- PUT /shoes
- GET /shoes
- POST /shoes/replaceOrCreate
- GET /shoes/{id}/exists

GET /shoes

Summary

Find all instances of the model matched by filter from the data source.

Parameters

Name	Located in	Required	Schema
filter	query	false	string

Filter defining fields, where, include, order, offset, and limit

Responses

Code	Schema
200	array

Request was successful

```
curl --request GET \
  --url 'https://$(catalog.host)/kicksApi/shoes?filter=REPLACE_THIS_V
ALUE' \
  --header 'accept: application/json' \
  --header 'content-type: application/json' \
  --header 'x-ibm-client-id: default' \
  --header 'x-ibm-client-secret: SECRET'
```

Definition

GET $(catalog.host)/kicksApi/shoes

Example response

```
[
  "mep"
]
```

https://$(catalog.host)/kicksApi/shoes

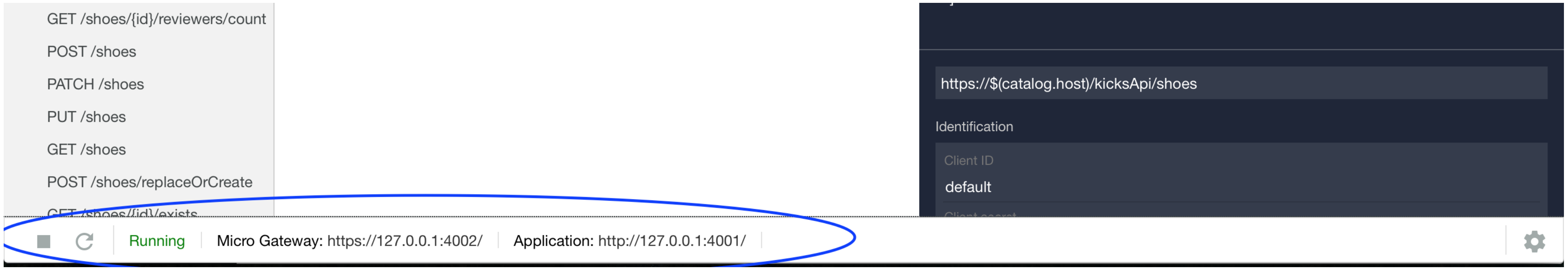
Identification

Client ID

default

Test your API instantly!

Click on the play button at the bottom of the designer to start your services.



GET /shoes/{id}/reviewers/count
POST /shoes
PATCH /shoes
PUT /shoes
GET /shoes
POST /shoes/replaceOrCreate

https://$(catalog.host)/kicksApi/shoes

Identification

Client ID
default

Running | Micro Gateway: https://127.0.0.1:4002/ | Application: http://127.0.0.1:4001/

PersistedModel generated CRUD capability

- Create model instance
- Update / insert instance
- Check instance existence
- Find instance by ID
- Find matching instances
- Find first instance
- Delete model instance
- Delete all matching instances
- Get instance count
- Update model instance attributes
- Update matching model instances

Querying Data Models

Remember a model generates both Node and REST API's

In code:

```
Shoe.find(
    { where: { name: 'Jordan' }, limit: 3 },
    function(err, shops) {
        // ...
    }
);
```

Or over HTTP:

```
/Shoes?filter=[where][name]=Jordan&filter[limit]=3
/Shoes?filter={"where":{"name":"Jordan"},"limit":3}
```

Let's create a few more models for the app

- Add some more properties to the shoe model using the designer
  - string model
  - string color
- create a "review" model with properties
  - string comments
  - number rating
  - Date reviewDate
- create a "reviewer" model with User as the base model
  - string name

The built-in User model

- Inherited from Persisted Model
- Additional built-in methods
  - Log in user
  - Log out user
  - Confirm email address
  - Reset password

Relationship Modeling

Various Relationship Types

- BelongsTo relations
- HasMany relations
- HasManyThrough relations
- HasAndBelongsToMany relations
- Polymorphic relations
- Embedded relations (embedsOne and embedsMany)

Relationships in kickReviews

- one shoe can have many reviews
- one review belongs to one shoe
- one reviewer can write many reviews
- one review belongs to one reviewer

Let's create Relationships

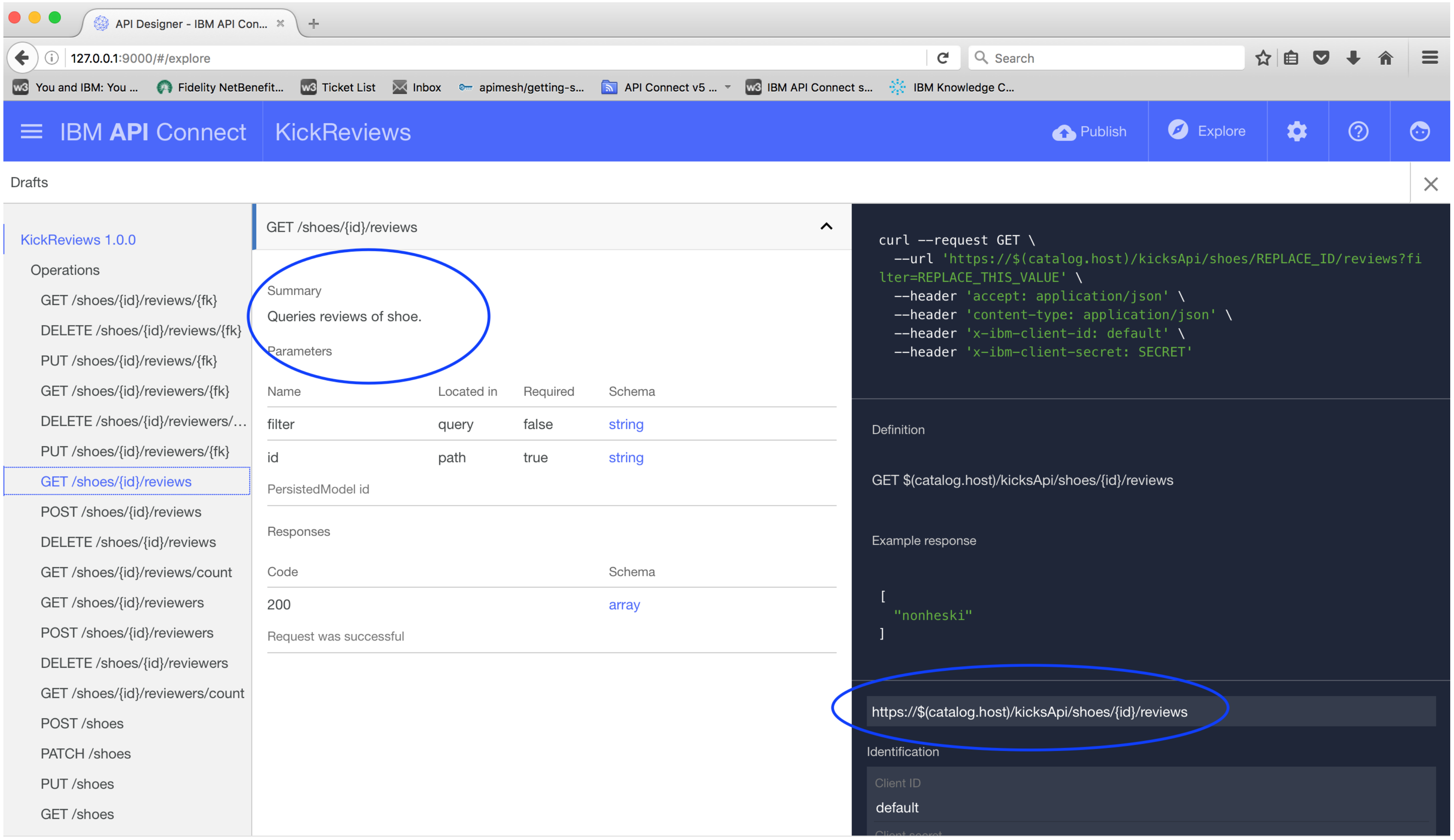
1. Via CLI

```
~/my-app$ apic loopback:relation
```

2. In model config file

```
// common/models/author.json
{
"name": "shoe",
// ...,
"reviews": {
      "type": "hasMany",
      "model": "review",
      "foreignKey": ""
   },
   "reviewers": {
      "type": "hasMany",
      "model": "reviewer",
      "foreignKey": ""
```

Now Look at the APIC Explorer again



API Designer - IBM API Con...

127.0.0.1:9000/#/explore

IBM API Connect | KickReviews | Publish | Explore

Drafts

KickReviews 1.0.0

Operations

- GET /shoes/{id}/reviews/{fk}
- DELETE /shoes/{id}/reviews/{fk}
- PUT /shoes/{id}/reviews/{fk}
- GET /shoes/{id}/reviewers/{fk}
- DELETE /shoes/{id}/reviewers/…
- PUT /shoes/{id}/reviewers/{fk}
- GET /shoes/{id}/reviews
- POST /shoes/{id}/reviews
- DELETE /shoes/{id}/reviews
- GET /shoes/{id}/reviews/count
- GET /shoes/{id}/reviewers
- POST /shoes/{id}/reviewers
- DELETE /shoes/{id}/reviewers
- GET /shoes/{id}/reviewers/count
- POST /shoes
- PATCH /shoes
- PUT /shoes
- GET /shoes

GET /shoes/{id}/reviews

Summary

Queries reviews of shoe.

Parameters

Name	Located in	Required	Schema
filter	query	false	string
id	path	true	string
PersistedModel id			

Responses

Code	Schema
200	array
Request was successful	

```
curl --request GET \
  --url 'https://$(catalog.host)/kicksApi/shoes/REPLACE_ID/reviews?fi
lter=REPLACE_THIS_VALUE' \
  --header 'accept: application/json' \
  --header 'content-type: application/json' \
  --header 'x-ibm-client-id: default' \
  --header 'x-ibm-client-secret: SECRET'
```

Definition

GET $(catalog.host)/kicksApi/shoes/{id}/reviews

Example response

```
[
  "nonheski"
]
```

https://$(catalog.host)/kicksApi/shoes/{id}/reviews

Identification

Client ID

default

Authentication and Authorization

Authentication

- Remember the built-in user model includes basic authentication.
- Login returns an access token which can be used for session management.

LoopBack Authentication & Authorization

- Principal
- Role
- RoleMapping
- ACL

Principals

An entity that can be identified or authenticated

- A user
- An application
- A role

Role

A group of principals with the same permissions

Some dynamic roles are included:

- $everyone
- $unauthenticated
- $authenticated
- $owner

RoleMapping

Holds the mapping of principal to role.

Access Control Layers

Define access a principal has to a certain operation against a model.

- Deny everyone to access the model
- Allow '$authenticated' role to create model instances
- Allow '$owner' to update an existing model instance

This is an example of "whitelisting", and is safer than excluding operations.

Roles in the kickReviews App

- Admin
- Reviewer

Creating New Roles

```
module.exports = function(app) {

    app.models.Role.create({

        name: 'admin'

    }, function(err, theAdminRole) {
        if (err) { cb(err); }

        // Maybe add some users to it?
    });

};
```

Typically this is done in a boot script.

Map Principal to a custom role

```
theAdminRole.principals.create({

    principalType: app.models.RoleMapping.USER,
    principalId: someUser.id

}, function(err, principal) {
    // handle the error!
    cb(err);
});
```

Typically this is also done in a boot script.

Defining an ACL

Use `apic loopback`, but this time you'll use the `acl` sub-command:

```
$ apic loopback:acl
```

And now we can start using the roles we just created.

ACLs in kickReviews

- allow unauthenticated users (anyone) to read reviews
- only allow authenticated users to create reviews
- only allow the review writer to make changes
- only allow admin to create new shoe entries

IBM apiconnect

Defining an ACL

Deny everyone all endpoints:

```
? Select the model to apply the ACL entry to: Review
? Select the ACL scope: All methods and properties
? Select the access type: All (match all types)
? Select the role: All users
? Select the permission to apply: Explicitly deny access
```

Defining an ACL

Now allow everyone to read Reviews:

```
? Select the model to apply the ACL entry to: Review
? Select the ACL scope: All methods and properties
? Select the access type: Read
? Select the role: All users
? Select the permission to apply: Explicitly grant access
```

Defining an ACL

Allow authenticated users to create reviews

```
? Select the model to apply the ACL entry to: Review
? Select the ACL scope: A single method
? Enter the method name: create
? Select the role: Any authenticated user
? Select the permission to apply: Explicitly grant access
```

Defining an ACL

Enable the creator ("$owner") to make changes:

```
? Select the model to apply the ACL entry to: Review
? Select the ACL scope: All methods and properties
? Select the access type: Write
? Select the role: The user owning the object
? Select the permission to apply: Explicitly grant access
```

Final step: enable Access Control

- call enableAuth() on the application object in a bootscript

Working with Data Sources

NoSQL

{JSON} Neo4j Apache Kafka
A high-throughput distributed messaging system.
elasticsearch.
ArangoDB

Database

MySQL ORACLE Microsoft SQL Server SAP HANA CouchDB relax

Services

REST SOAP SharePoint <xml/> twilio CLOUD COMMUNICATIONS atg

Messaging

ActiveMQ WEBSPHERE MQ® Email redis

Datasource Definition

1. CLI: `$ apic loopback:datasource`
2. in code
3. via API Designer

Via CLI (and code)

1. `~/my-app$ apic loopback:datasource`
2. Edit `server/datasources.json` with connection info
3. Update model-config as needed
4. Install connector npm module:

```
~/my-app$ npm install --save loopback-connector-mongodb
```

StrongLoop Supported Connectors

- MongoDB
- MySQL
- Oracle
- Postgres
- MSSQL
- IBM DB2
- IBM Cloudant DB

More LoopBack connectors (3rd patry)

Kafka, Elastic Search, CouchDB, Neo4j, RethinkDB, Riak, etc, etc, etc

https://github.com/pasindud/awesome-loopback

Configuring the Datasource for Cloudant DB on Bluemix

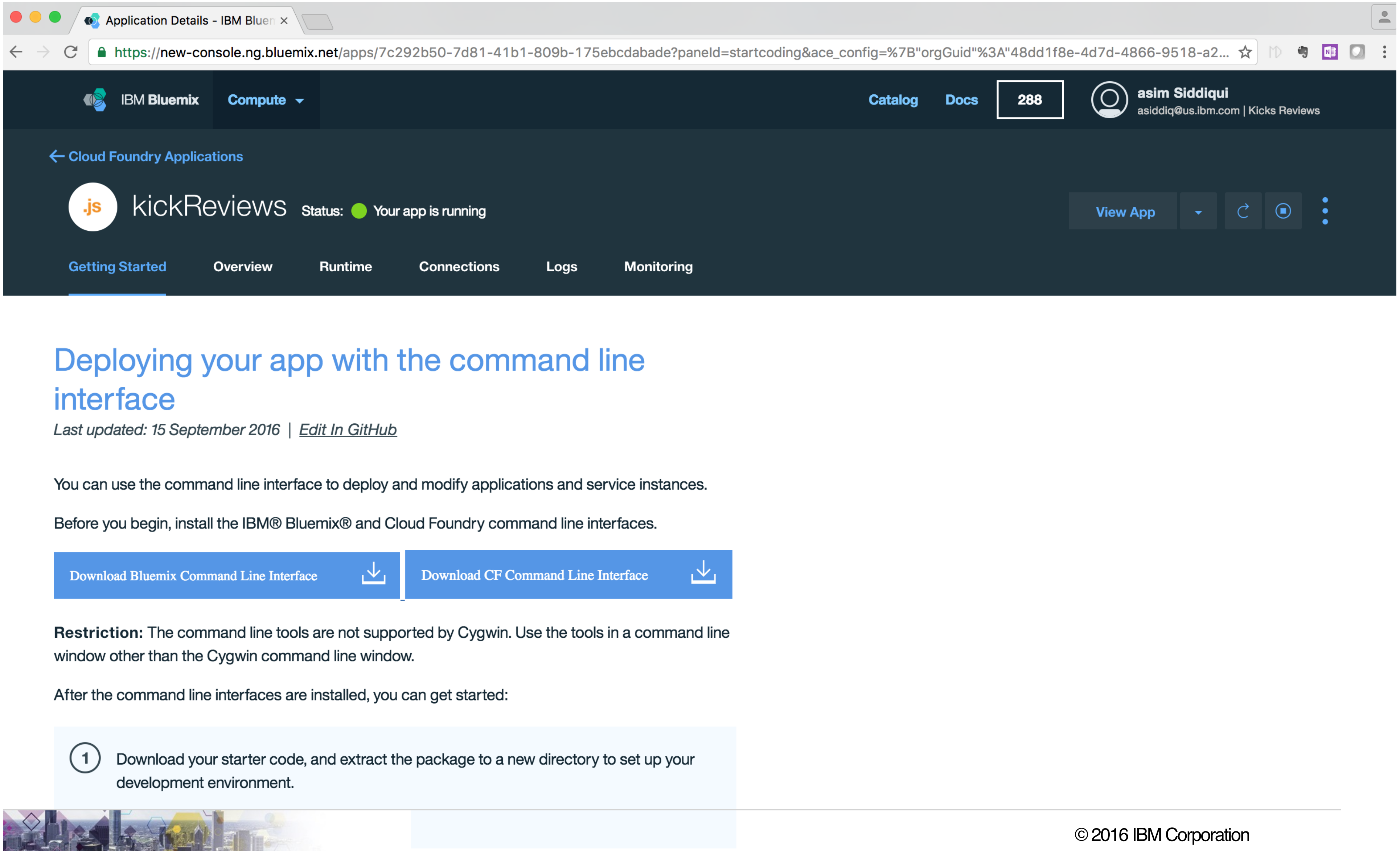
```json
{
  "cloudant_kr": {
      "port": 443,
      "url": "https://a41808b2-0e16-49a8-a38f-c183e7c89170-bluemix:39b
      "database": "kickreviews",
      "username": "a41808b2-0e16-49a8-a38f-c183e7c89170-bluemix",
      "password": "39ba7164f0737dd1bd60997facd7b97660cfe1154121b9e2bef
      "name": "cloudant_kr",
      "modelIndex": "",
      "connector": "cloudant"
    }
}
```

Deploy to Bluemix

Create an instant nodejs runtime

- Select the Compute menu
- Select IBM SDK for nodejs

Download Bluemix and CF CLI



Application Details - IBM Bluen
https://new-console.ng.bluemix.net/apps/7c292b50-7d81-41b1-809b-175ebcdabade?paneld=startcoding&ace_config=%7B"orgGuid"%3A"48dd1f8e-4d7d-4866-9518-a2...
IBM Bluemix
Compute
Catalog
Docs
288
asim Siddiqui
asiddiq@us.ibm.com | Kicks Reviews
Cloud Foundry Applications
.js
kickReviews
Status: Your app is running
View App
Getting Started
Overview
Runtime
Connections
Logs
Monitoring

Deploying your app with the command line interface

Last updated: 15 September 2016 | *Edit In GitHub*

You can use the command line interface to deploy and modify applications and service instances.

Before you begin, install the IBM® Bluemix® and Cloud Foundry command line interfaces.

Download Bluemix Command Line Interface
Download CF Command Line Interface

Restriction: The command line tools are not supported by Cygwin. Use the tools in a command line window other than the Cygwin command line window.

After the command line interfaces are installed, you can get started:

1 Download your starter code, and extract the package to a new directory to set up your development environment.

Use Bluemix CLI to login on terminal

```
~$ bluemix login
   Invoking 'cf login'...

   API endpoint: https://api.ng.bluemix.net

   Email> asiddiq@us.ibm.com

   Password>
   Authenticating...
   OK
```

IBM

IBM apiconnect

Use Cloudfoundry CLI to Deploy your app

```
~$  cf push kickReviews -c "node server/server.js"
```

THE END

Data Validation

Using Built-in Methods

```
CoffeeShop.validates[Method](
    'property',
    {
        option: value,
        message: 'What to tell the user...'
    }
);
```

Using Built-in Methods

```
// common/models/coffee-shop.js

CoffeeShop.validatesPresenceOf('name', 'address');

CoffeeShop.validatesLengthOf('name',{
    min: 5,
    max: 100,
    message: {
        min: 'Name is too short',
        max: 'Name is too long'
    }
});

CoffeeShop.validatesInclusionOf('beverages', {
    in: [ 'coffee', 'tea', 'juice', 'soda' ]
});
```

Custom Validation

```
CoffeeShop.validate('openingHour', hourCheck, {
    message: 'Opening hour must be before closing hour'
});

function hourCheck(errCallback) {
    if (this.openingHour > this.closingHour) {
        errCallback();
    }
});
```

Data Validation

Validation will automatically occur on "upsert"!

But you can also call it directly:

```
if (!CoffeeShop.isValid()) {
    // do something?
}
```

Advancing Your API

Using the REST Connector

The LoopBack REST connector enables applications to interact with other REST APIs using a template-driven approach

IBM

IBM apiconnect

REST DataSource

```
"geoRest": {
  "name": "geoRest",
  "connector": "rest",
  "operations": [{
    "template": {
      "method": "GET",
      "url": "http://maps.googleapis.com/maps/api/geocode/{format=json
      "headers": {
        "accepts": "application/json",
        "content-type": "application/json"
      },
      "query": {
        "address": "{street},{city},{zipcode}",
        "sensor": "{sensor=false}"
      },
      "responsePath": "$.results[0].geometry.location"
    },
    "functions": {
      "geocode": ["street", "city", "zipcode"]
```

Using a REST Datasource

Once you connect a model to a datasource (in `model-config.json`):

```
"Widget": {
    "dataSource": "geoRest",
    "public": true
}
```

You can call the function:

```
Widget.geocode('107 S B St', 'San Mateo', '94401', function(res) {
    // ... handle the response
});
```

Remote Methods

A way to create new, non-CRUD methods on your model.

Remote Methods

```
// common/models/person.js

module.exports = function(Person){
    Person.greet = function(msg, cb) {
      cb(null, 'Greetings... ' + msg);
    }

    Person.remoteMethod(
        'greet',
        {
          accepts: { arg: 'msg', type: 'string', http: { source: 'quer
          returns: { arg: 'greeting', type: 'string' }
          http: [
            { verb: 'get', path: '/greet' }
          ]
        }
    );
};
```

Remote Methods

Now, for example, a request to `GET /api/persons/greet?msg=John` will return:

```
{ greeting: "Greetings... John!" }
```

Remote and Operation Hooks

Hooks

LoopBack provides two kinds of hooks:

1. **Remote Hooks**: execute before or after a remote method is called.
2. **Operation Hooks**: execute when models perform CRUD operations.

NOTE: Operation hooks replace model hooks, which are now deprecated.

Remote Hooks

A remote hook enables you to execute a function before or after a remote method is called by a client:

- `beforeRemote()` runs before the remote method.
- `afterRemote()` runs after the remote method has finished successfully.
- `afterRemoteError()` runs after the remote method has finished with an error.

Remote Hooks

Both `beforeRemote()` and `afterRemote()` have the same signature; below syntax uses beforeRemote but afterRemote is the same:

```
Person.afterRemote( 'create', function( ctx, modelInstance, next) {
  // do some stuff... maybe send them a nice email?

  next();
}
```

Context AccessToken

The `accessToken` of the user calling the remote method.

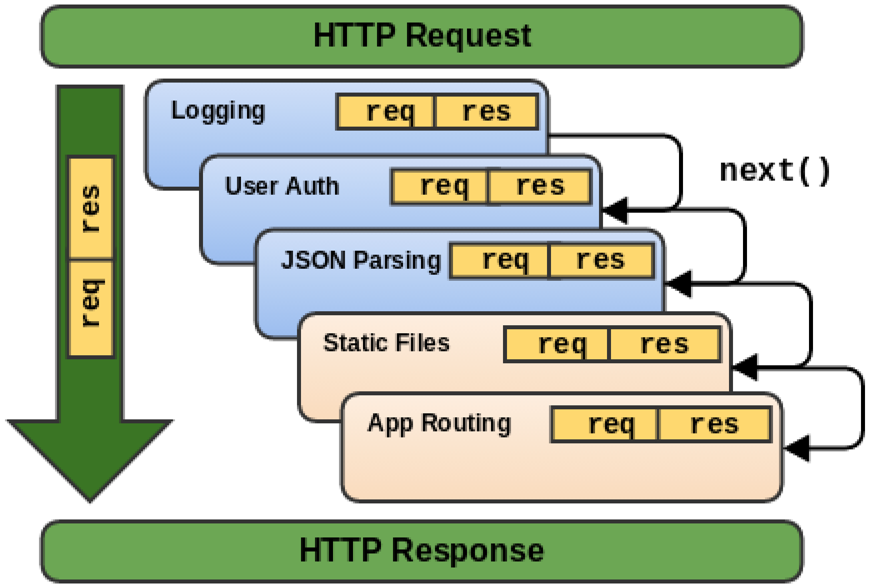
`ctx.req.accessToken` is undefined if the remote method is not invoked by a logged in user (or other principal).

You can use `ctx.req.accessToken.userId` to find the currently logged in user (if there is one).

Middleware

Introduction to Middleware

LoopBack is built on Express, which uses "Connect" style middleware.



LoopBack Phases

LoopBack defines a number of phases, corresponding to different aspects of application execution.

Use these phases to define where your middleware should be inserted!

This is necessary since middleware fires in the order it is added.

LoopBack Phases

1. `initial`
2. `session`
3. `auth`
4. `parse`
5. `routes`
6. `files`
7. `final`

LoopBack Phases

Each phase has "before" and "after" subphases in addition to the main phase, encoded following the phase name, separated by a colon. For example, for the "initial" phase, middleware executes in this order:

- `initial:before`
- `initial`
- `initial:after`

Defining Middleware

function which accepts 3 arguments (4 if error-handling):

```
function myMiddlewareFunc([err,] req, res, next) {
    // do some stuff

    next();
    // next(new Error());  // if there is an error during processing
};
```

Defining Middleware

You might want to use a factory for this function to encapsulate any options:

```
module.exports = function(options) {

  return function myMiddleware(req, res, next) {
    // ...
    next();
  }

}
```

Registering Middleware

```
// server/middleware.json
{
  // ...,
  "auth": {
    "../server/middleware/logger": {
      "enabled": true
    }
  },
  // ...
}
```

Registering Middleware

Additionally, you can use the shorthand format `{module}#{fragment}` where fragment is:

- A function exported by `{module}`
- A JS file in `{module}/server/middleware/` directory
- A JS file in `{module}/middleware/`

IBM apiconnect

Middleware configuration properties

You can specify the following properties in each middleware section:

```
"../server/middleware/logger": {
    "enabled": true,
    "params": { "some": "option" },
    "paths": "/foo/bar"
}
```

Registering Middleware

We can also register middleware in JavaScript code (a boot script perhaps):

```
app.middleware('routes:before', require('morgan')('dev'));
app.middleware('routes:after', '/foo', require('./foo/routes'));
app.middleware('final', errorhandler());
```

Thank You!

Questions?

Introduction to LoopBack

Asim Siddiqui l @asimASiddiqui

Join us for more events!

strongloop.com/developers/events