PRJ4 - Cloud

IBM Bluemix - Cloud Foundry

Gregor Schwake PhD

17.02.2021

Contents

Date	Comment
17.02.2021	Initial Version
06.03.2021	Update “Remote Database” - Alternative to Remote MqSQL
06.03.2021	Update “Remote Database” - Error “Missing Table” solved

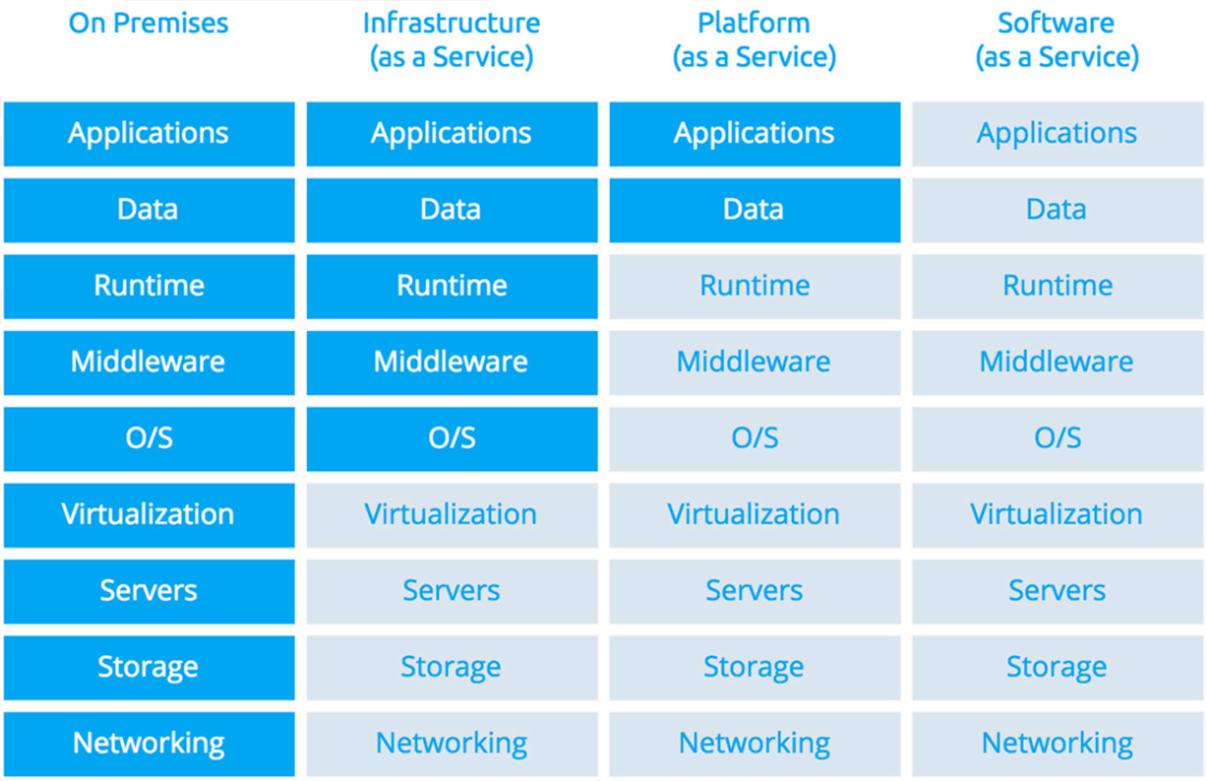
PRJ4-Cloud

Approach

With PRJ4 the topic Cloud is approached in a pragmatic way. The basic concepts are introduced and some hands on experience is gained by orchestrating different tools to deploy an app to the cloud and manage it.

In a first step a simple app is created via LoopBack 4. This app runs locally and uses an external database provided by RemoteMySQL. The second step introduces the IBM Cloud. While still using the database provided by RemoteMySQL, the app is pushed to IBM Bluemix und runs there. The final step is to follow the LoopBack 4 tutorial on deploying the Todo example app to the IBM Cloud with having persistency provided by an IBM Cloudant database.

It might be a good idea to first have a look at different Cloud approaches like *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS) and *Software as a Service* (SaaS).



The approach is as follows. With all topics there will be a tutorial to run through or a link to a resource to familiarize with. There will also be so called **Deliverables**. A deliverable states what might be addressed in the assessment. Usually knowledge and expertise will be shown via the app developed by the group. However, sometimes not all areas are covered by a group's app. In these cases tutorials

could offer a fallback scenario.

In general, you have to be careful with the tutorials because sometimes not all pitfalls are addressed. Whenever something you need to look out for is not mentioned in the tutorial, there will be a **Be careful:** telling how to cope.

Remote Database

With the *Todo* LoopBack 4 tutorial an in memory database was used. It would be better if a real database could be used for persistency but without the work involved in setting up one. The solution is to use a database that is setup and maintained by somebody else.

It doesn't matter what provider is used for this example.

- *FreeMySQLHosting.net* is recommended
- *RemoteMqSLQ.com* is cost free but has lots of ads. In some cases it was impossible to successfully do the surveys, which are necessary to create a database.
- Others like *DB4Free.net* have not been tested

No matter what provider is used, the typical approach is to create an account and then create a database. With *DB4Free.net* the database is automatically created together with the account. *FreeMySQLHosting.net* will send you an email with access information.

Be careful:

- FreeMySQLHosting.net has been tested to work. You might be asked to send a weekly email to indicate that the database is still in use.
- In order to be able to create the database at RemoteMySQL.com, ie before the button to create the database is shown, a survey has to be done (two to be honest). Read the questions carefully as there are some that advise to pick a specific answer. If that advised answer isn't selected, the survey will have to be redone. Once the create button is shown, click it and your MySQL database is created.
- RemoteMySQL might remove your database after a specific period of inactivity.

Super App

In order to procees with the example, an app needs to be created. Of course, this will be done using LoopBack 4. Later on this app will be pushed to the cloud.

Be careful:

- With IBM Bluemix, which we will use, each app has to have a unique name. Therefore you should include something like part of your student number or phone number to make the app name unique. Instead of of using *lb4 app* it is possible to directly use the name of the app.

```
lb4 <app name>
```

```
greg@Gregs-MacBook-Pro apps2021 % lb4 app89271
? Project description: LoopBack 4 App with RemoteMqSQL database
? Project root directory: app89271
? Application class name: App89271Application
? Select features to enable in the project Enable eslint, Enable prettier, Enable mocha, Enable loopbackBuild, Enable vscode, Enable docker, Enable repositories, Enable services
```

Figure 1: App Creation

All following activities need to be done inside the app's directory.

```
cd <app name>
```

MySQL Connector

Before MySQL databases can be accessed, the corresponding LoopBack connector needs to be installed.

```
npm install loopback-connector-mysql --save
```

For details on MySQL usage with LoopBack 4 have a look at https://loopback.io/doc/en/lb4/MySQL-connector.html.

Model

The app's model will be very simple with managing names and surnames.

```
greg@Gregs-MacBook-Pro app89271 % lb4 model
? Model class name: names
? Please select the model base class Entity (A persisted model with an ID)
? Allow additional (free-form) properties? No
Model Names will be created in src/models/names.model.ts

Let's add a property to Names
Enter an empty property name when done

? Enter the property name: surname
? Property type: string
? Is surname the ID property? No
? Is it required?: Yes

Let's add another property to Names
Enter an empty property name when done

? Enter the property name: name
? Property type: string
? Is name the ID property? No
? Is it required?: Yes

Let's add another property to Names
Enter an empty property name when done

? Enter the property name: id
? Property type: number
? Is id the ID property? Yes
? Is id generated automatically? Yes
```

Datasource

Once the model is created the datasource, specifying where the model's data comes from, needs to be created. It is important to note that specific access data is not given at this point, instead later on the corresponding file will be modified.

```
greg@Gregs-MacBook-Pro app89271 % lb4 datasource
? Datasource name: names
? Select the connector for names:  MySQL (supported by StrongLoop)
? Connection String url to override other settings (eg: mysql://user:pass@host/db):
? host:
? port:
? user:
? password: [hidden]
? database:
   create src/datasources/names.datasource.ts
   update src/datasources/index.ts
```

Repository

Following the datasource, the next step is to create the repository.

```
greg@Gregs-MacBook-Pro app89271 % lb4 repository
? Please select the datasource NamesDatasource
? Select the model(s) you want to generate a repository for Names
? Please select the repository base class DefaultCrudRepository (Juggler bridge)
   create src/repositories/names.repository.ts
   update src/repositories/index.ts

Repository NamesRepository was/were created in src/repositories
```

Controller

As controller a simple CRUD controller will suffice as there is no specific business logic.

```
greg@Gregs-MacBook-Pro app89271 % lb4 controller
? Controller class name: names
Controller Names will be created in src/controllers/names.controller.ts

? What kind of controller would you like to generate? REST Controller with CRUD functions
? What is the name of the model to use with this CRUD repository? Names
? What is the name of your CRUD repository? NamesRepository
? What is the name of ID property? id
? What is the type of your ID? number
? Is the id omitted when creating a new instance? Yes
? What is the base HTTP path name of the CRUD operations? /names
   create src/controllers/names.controller.ts
   update src/controllers/index.ts

Controller Names was/were created in src/controllers
```

Datasource Configuration

The datasource needs to be updated with the RemoteMySQL access data.

Be careful:

- As soon as something is given for *url* all other entries are obsolete.
- The correct sequence needs to be used *“url”:“https//user:password@host:port/database”*. An example is given below.

```
const config = {
  name: 'names',
  connector: 'mysql',
  url: 'https://xVhg7CVnGQ:aPassword@remotemysql.com:3306/xVhg7CVnGQ'
  host: '',
  port: 0,
  user: '',
  password: '',
  database: ''
};
```

Figure 2: Datasource Configuration

Test

Now it has to be checked that the app is really using RemoteMySQL as persistency store.

- Start the server with *npm start*.
- Using the App Explorer (http://localhost:3000/explorer/) insert something into the remote database. With *POST/names* that is easily done.

```
{
  "names": "Dean",
  "surname": "James"
}
```

- Query the database. With *GET/names* that is easily done.

Be careful:

- With the API Explorer's *filter* and *where* data might not be shown correctly. It is probably best to remove them.
- Sometimes the table *Names* is not automatically created during start up. If there is an error message like table *Names* not found, *app.migrateSchema()* needs to be added to *src/index.ts*. An example of how to do that can be found at *Auto-update database at start* in the LoopBack 4 documentation (https://loopback.io/doc/en/lb4/Database-migrations.html#auto-update-database-at-start)

Response body


```
[
  {
    "names": "Dean",
    "surname": "James",
    "id": 1
  }
]
```


Download

Response headers

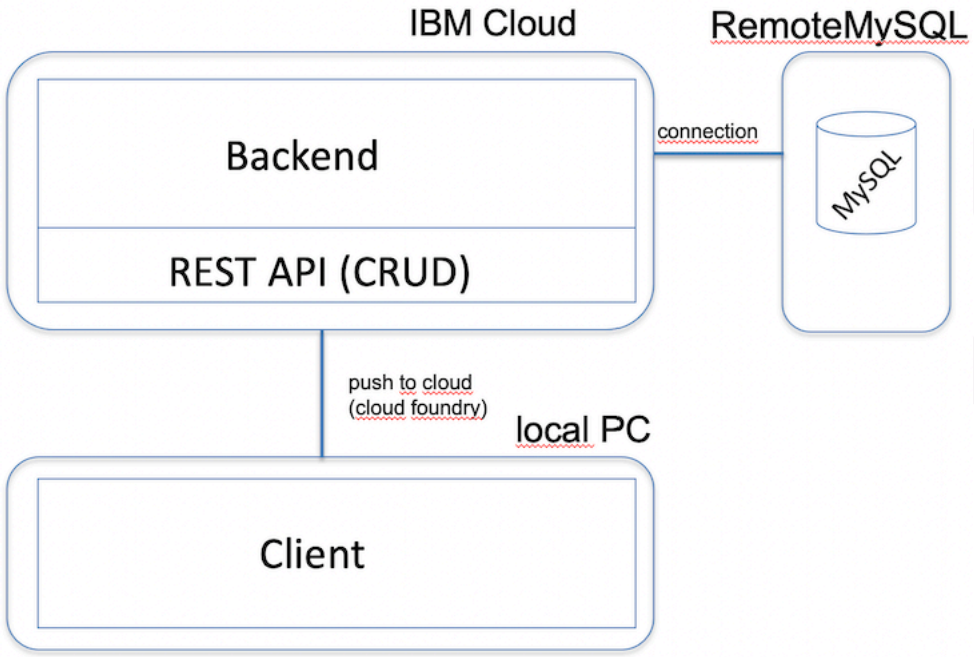

```
access-control-allow-credentials: true
access-control-allow-origin: *
connection: keep-alive
content-length: 43
content-type: application/json
date: Fri,05 Feb 2021 23:05:31 GMT
keep-alive: timeout=5
x-powered-by: Express
```


Deliverables:

- You are able to create a remotely accessible MySQL database (provider doesn't matter).
- You are able to create and test an app that uses a remote MySQL database using LoopBack 4.

IBM Cloud

In this section we are going to familiarize with the IBM Cloud. The previous example is pushed to IBM Bluemix. With the app running in the IBM Cloud, the database will still be with the external provider.



Prerequisites

Before the IBM Cloud can be used, an account has to be created. These accounts can typically be used for 6 months and are cost free. The Cloud Foundry CLI needs to be installed as this will later on be used to push apps to the IBM Cloud.

- Sign up for IBM Cloud (https://cloud.ibm.com/login)
- As we are using the previous *Super App* example, change to the corresponding directory if necessary.
- Install the *Cloud Foundry CLI* (https://docs.cloudfoundry.org/cf-cli/install-go-cli.html)
- Install the module *cfenv* to simplify Cloud Foundry related operations.
- If there are problems pushing the app to the cloud have a look at section *Push to IBM Cloud - Pitfalls*

```
1 npm i cfenv
```

Be careful:

- An IBM Lite account is only accessible for at most 6 months.
- If you already have an account and at sign on you are prompted to provide billing information in order to upgrade your account you will have to create a new account with another email.
- IBM does not accept all email providers. In case there is an error message during account creation, try with an email address from another provider. Fontys, gmail and icloud accounts are proven to work.
- Install the Cloud Foundry CLI version 6, that is the one where no version is given with the installation.
- With macOS, if XCode Command Line Tools are not current go to Apple Developer (https://developer.apple.com/download/more) and download the XCode Command Line Tools for your XCode version.
- If there are problems pushing the app to the cloud have a look at section *Push to IBM Cloud - Pitfalls*

Prepare App

There are some steps that need to be taken to prepare the app for the IBM Cloud. First service binding needs to be enabled. In *src/index.ts* add the following snippets.

- After the import statements add:

```
// --------- ADD THIS AFTER IMPORTS ---------
import {DbDataSource} from './datasources/db.datasource';
const cfenv = require('cfenv');
const appEnv = cfenv.getAppEnv();
// --------- ADD THIS SNIPPET ---------
```

- As first entry of mains add:

```
// --------- ADD THIS SNIPPET AFTER: export async function main(options
    ?: ApplicationConfig) {---------
  // Set the port assigned for the app
  if (!options) options = {};
  if (!options.rest) options.rest = {};
  options.rest.port = appEnv.isLocal ? options.rest.port : appEnv.port;
  options.rest.host = appEnv.isLocal ? options.rest.host : appEnv.host;
  // --------- ADD THIS SNIPPET ---------
```

- Remove the *prestart* entry from package.json.

Be careful:

- Do not forget that with any change in the code, *npm run build* needs to be done before deploying the code.

Push App

- Set the endpoint and then do the login as shown:

```
greg@Gregs-MacBook-Pro app89271cloud % cf api https://api.eu-gb.bluemix.net
Setting API endpoint to https://api.eu-gb.bluemix.net...
OK

API endpoint:   https://api.eu-gb.bluemix.net
API version:    3.93.0

Not logged in. Use 'cf login' or 'cf login --sso' to log in.
greg@Gregs-MacBook-Pro app89271cloud % cf login
API endpoint: https://api.eu-gb.bluemix.net

Email: gwj.schwake@gmail.com
Password:

Authenticating...
OK

Targeted org gwj.schwake@gmail.com.

Targeted space dev.

API endpoint:   https://api.eu-gb.bluemix.net
API version:    3.93.0
user:           gwj.schwake@gmail.com
org:            gwj.schwake@gmail.com
space:          dev
```

Screenshot

Be careful:

- With the push command it is important to not exceed any quotas. Therefore it is better to first limit the app to 256 MB:

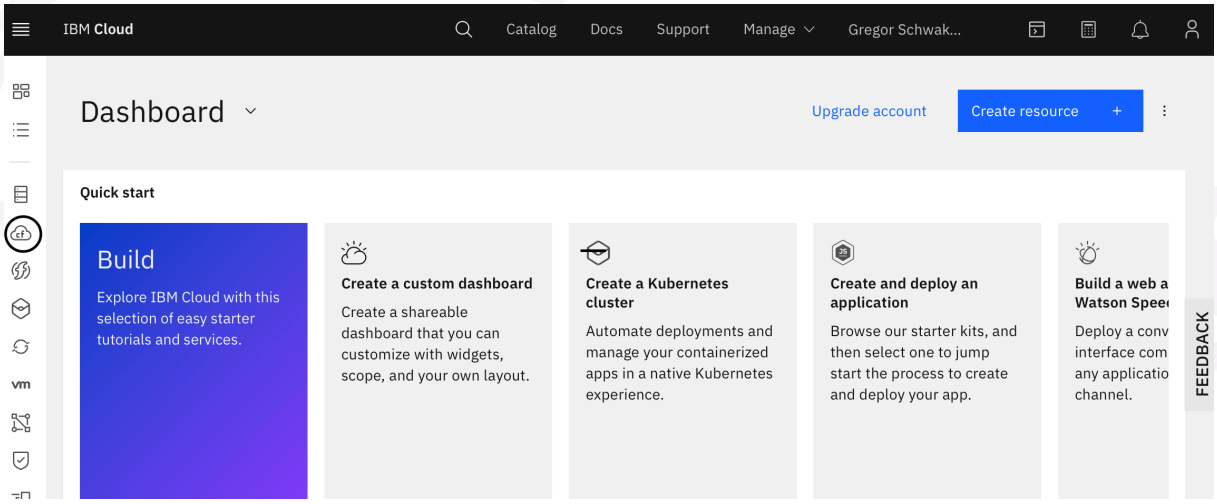
```
greg@Gregs-MacBook-Pro app89271cloud % cf push app89271 -m256MB
Pushing app app89271 to org gwj.schwake@gmail.com / space dev as gwj.schwake@gmail.com...
Packaging files to upload...
Uploading files...
 14.82 MiB / 14.82 MiB [=============================================================================

Waiting for API to complete processing files...

Staging app and tracing logs...
   Downloading xpages_buildpack...
   Downloading python_buildpack...
```

A successful push looks like this:

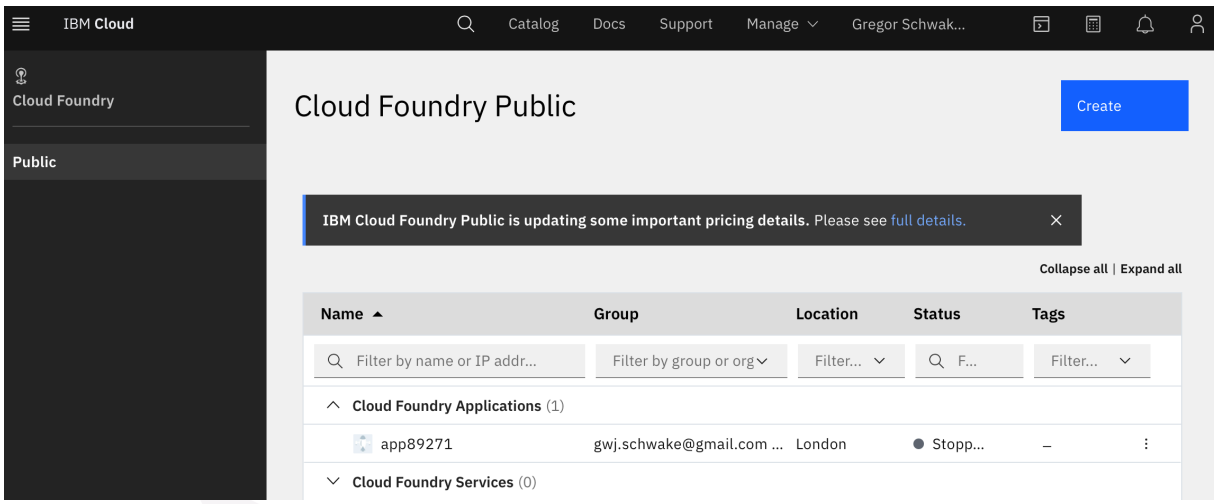
```
name:              app89271
requested state:   started
routes:            app89271.eu-gb.mybluemix.net
last uploaded:     Sat 06 Feb 23:42:54 CET 2021
stack:             cflinuxfs3
buildpacks:
       name              version              detect output    buildpack name
       sdk-for-nodejs    v4.5-20201130-1530   sdk-for-nodejs   sdk-for-nodejs

type:            web
sidecars:
instances:       1/1
memory usage:    256M
start command:   npm start
     state     since                  cpu     memory           disk           details
#0   running   2021-02-06T22:43:3     Screenshot  69.4M of 256M    308.9M of 1G
```

Check Result

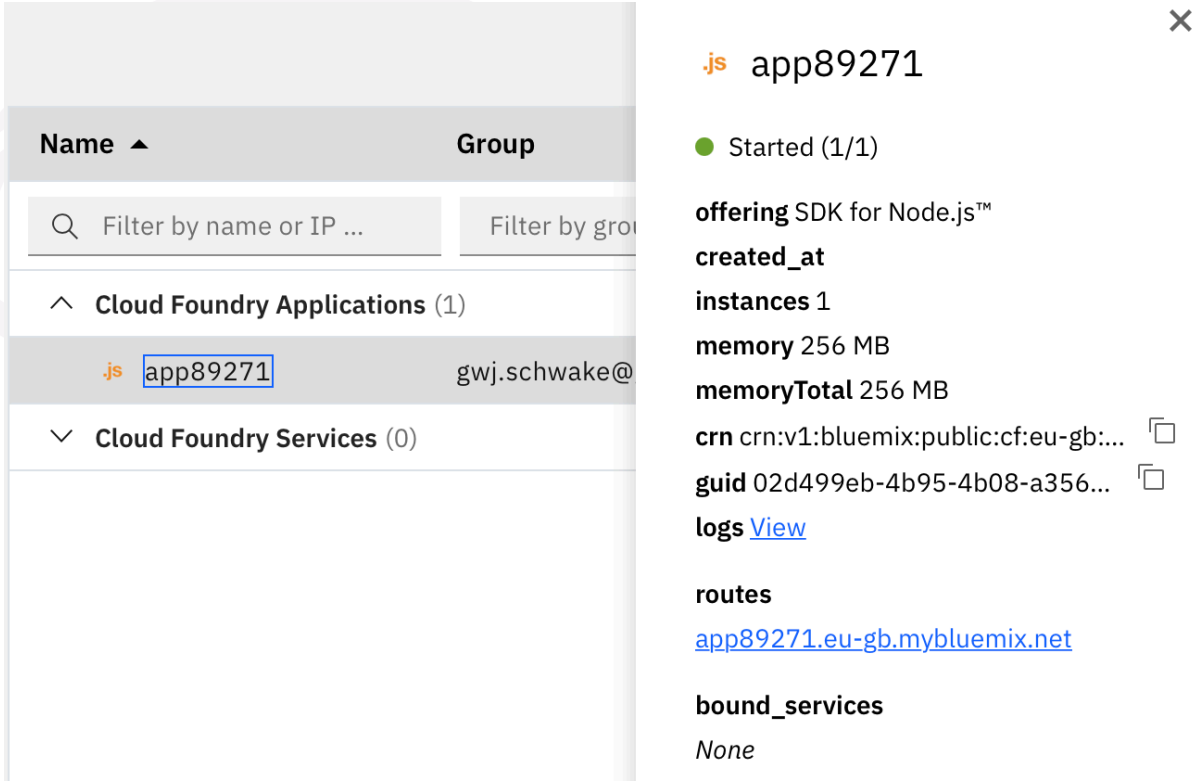
- Login to the IBM Cloud (https://cloud.ibm.com/login).
- In the *Dashboard* on the left, select the cloud that has *cf* inside:



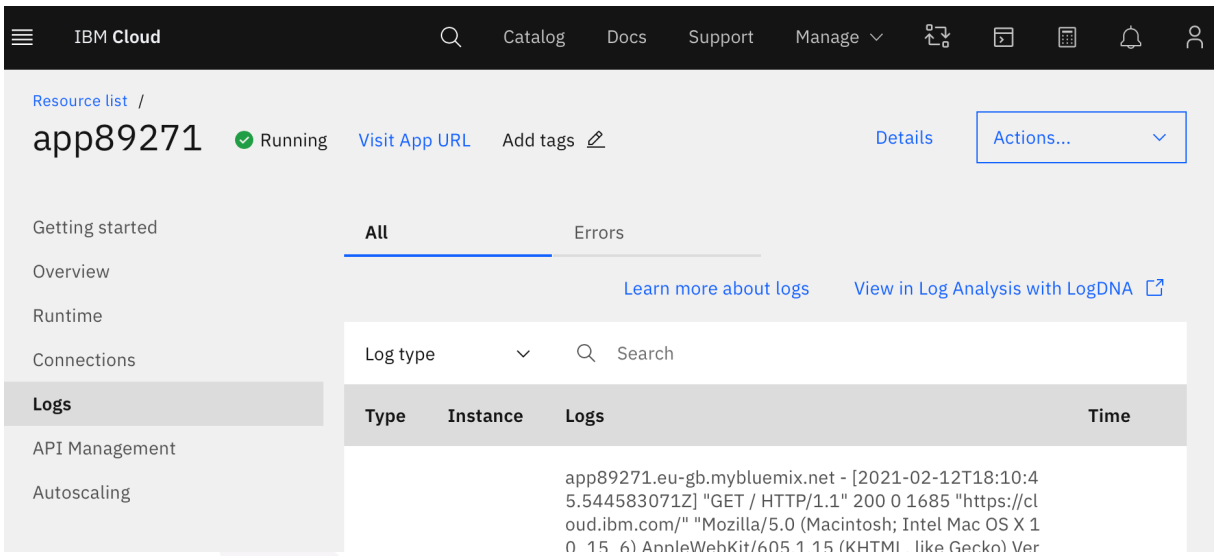
- On the next screen select *public*



- Select the app to see app details, a popup will appear.
- Select *logs View* to see log information.



- Select *VisitAppUrl* on top to check the app's REST API



Deliverables:

- You are able to push an app to the IBM Cloud using cloud foundry.
- You are able to test an app that is running in the IBM Cloud with the API Explorer.

Cloudant Database

This is an example that clones the *Todo* example and configures the app to use a Cloudant database and pushes it to the IBM Cloud. Several things like IBM Cloud account creation and cf CLI (Cloud Foundry Command Line Interface) installation have already covered with the previous example. This tutorial gives further information about how to create a Cloudant database within the IBM cloud as well as how to link that database to an app.

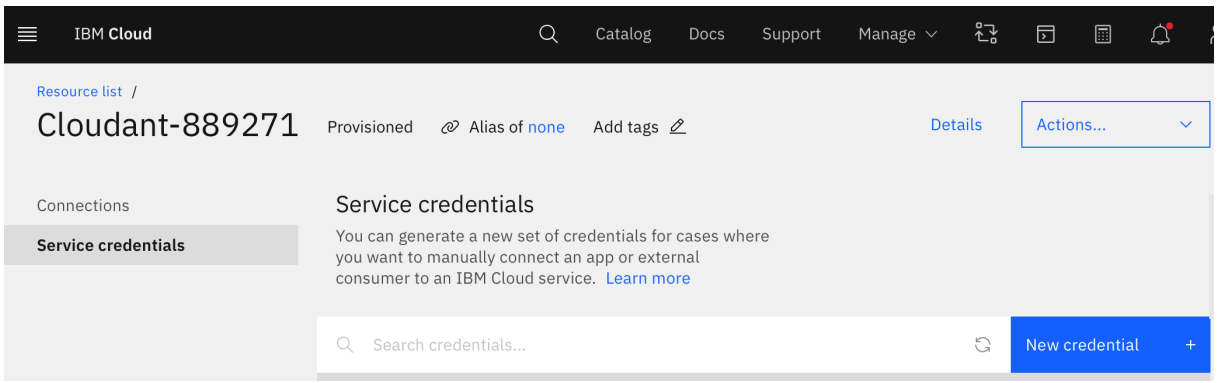
- Do the LoopBack 4 tutorial *Deploying to IBM Cloud* (https://loopback.io/doc/en/lb4/Deploying-to-IBM-Cloud.html)

Deliverables:

- Being able to push an app to the IBM Cloud with connection to an IBM Cloud database.

Be careful:

- Step 1: The IBM Cloud GUI has changed but configuration content is unchanged.
- Step 1: For *region* use *London*.
- Step 2: Select *non-partitioned*.
- Step 3: Install cloudant connector from app directory, e.g. loopback4-example-todo.
- Step 3: Get the service's credentials and use them in *db.datasource.ts* (see example below)



- Step 3: Put the credentials in *db.datasource.ts* (see example below)

```
const config = {
  name: 'db',
  connector: 'cloudant',
  database: 'todo89271',
  modelIndex: '',
  apikey: 'je24trCDBLnMHH8W4EyCXFmsBYHBrkh0yJm3YygFHuES',
  host: '00fe23de-f4fc-476d-b167-37d4bc5aaa65-bluemix.cloudantnosqldb.appdomain.cloud',
  iam_apikey_description: 'Auto-generated for key 3803a115-9225-4f1c-903d-8dd3d605edb5',
  iam_apikey_name: 'Service credentials-1',
  iam_role_crn: 'crn:v1:bluemix:public:iam::::serviceRole:Manager',
  iam_serviceid_crn: 'crn:v1:bluemix:public:iam-identity::a/8c4d8465de8a475a9d09dddfb0445
  username: 'apikey-v2-kghn52rmtm6c057o68uspop4reecb30llme9wcwr5f689',
  password: '54ba2bc834b7e5d6f35d8988d311ded5',
  port: 443,
};
```

* Step 5: From Europe use the endpoint *api.eu-gb.bluemix.net* with region *London*. * If something goes wrong, check the next section *Push to IBM Cloud - Pitfalls*.

Push to IBM Cloud - Pitfalls

Prepare the Push to IBM Cloud

- Provisioning a Cloudant database service on IBM Cloud – Stay with the standard region, which is *London*
- Always (!) run* npm run build* before deploying
- Mandatory: Create the *.cfignore* file with corresponding content
- To show hidden files in Finder (Mac) press *Cmd* + *Shift* + *.*
- Update url or (password, username and port) in *db.datasource.config.json* with the values from the Cloudant database credentials. Regarding credentials see step 3 from section *Cloudant Database*.

Push to IBM Cloud

- Make sure you have got the right endpoint: *cf api https://api.eu-gb.bluemix.net*
- Login, have IBM Cloud login credentials ready: *cf login*
- Push the application to the cloud, set memory constraints to 256 MB or 512 MB: *cf push <> -m256MB*

Push to IBM Cloud - Errors

- Errors about Org and Space – Check on endpoint and region
- Errors about Route – Check on unique route (api endpoint + unique(!) app name)
- App does not start in IBM Cloud / memory errors / TCP connection error – Did you do *npm run build* before the push to the cloud? – Did you restrict memory?,e.g. push -m256M – There are memory restrictions with your IBM Cloud account,check that you have not already pushed several apps to the cloud - if so, remove them.
- 500 Error: Internal Server Error – Probably a database error. Is the application connected to the Cloudant DB? Is *db.datasource.configuration.json* correctly updated? If the database is changed: Is migrate added to *index.ts*?

Ressources

- IBM Cloud (https://cloud.ibm.com/login)
- Cloud Foundry CLI Installation (https://docs.cloudfoundry.org/cf-cli/install-go-cli.html)