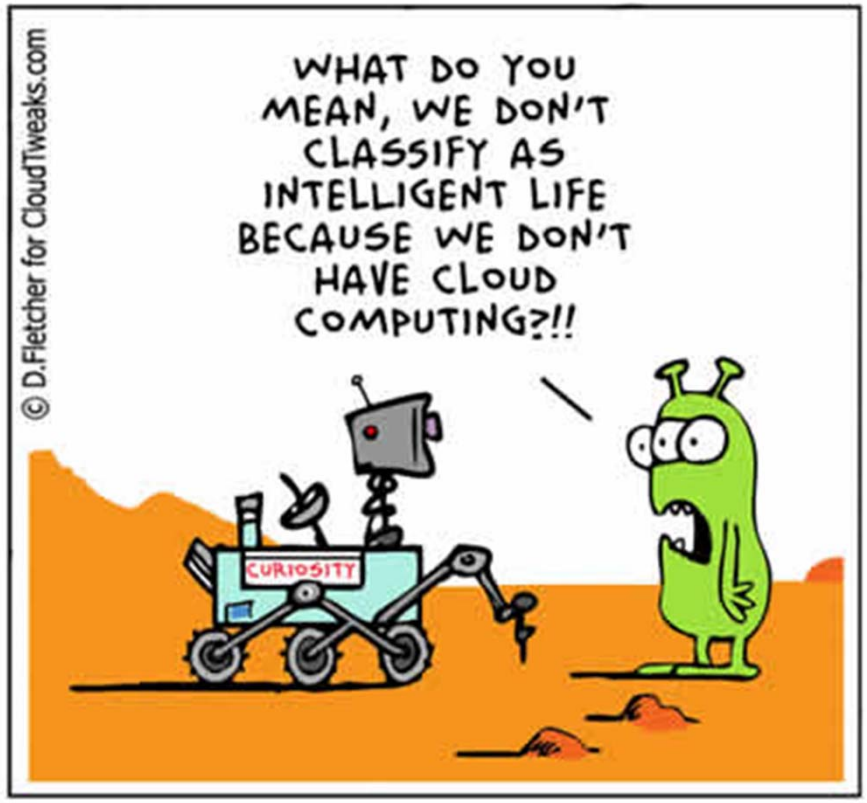
P02: IBM Cloud Services (PaaS, SaaS) - Lab

IBM Bluemix



IBM Watson

IBM DB2

IBM Data Science Experience

Outline

- Objective
- Bluemix
  - High-Level Architecture
  - Bluemix Cloud Foundry
  - Watch video – Demo: Getting Started with Node.js on Bluemix 2016
- Examples
  - Web Application to Query in-Cloud Database
  - Web Application with Watson Personality Insights Service
  - In-cloud Data Analytics using Simple SQL and R Scripting
- Summary

Objective

- develop a simple SaaS application using IBM Bluemix PaaS and SaaS services

- Example 1: A web server application for querying an in-cloud database
  - setting up an in-cloud database SaaS (Db2) and load data from the local computer
  - develop a simple SaaS web server application (using Node.js boilerplate PaaS) and linking with the in-cloud database SaaS

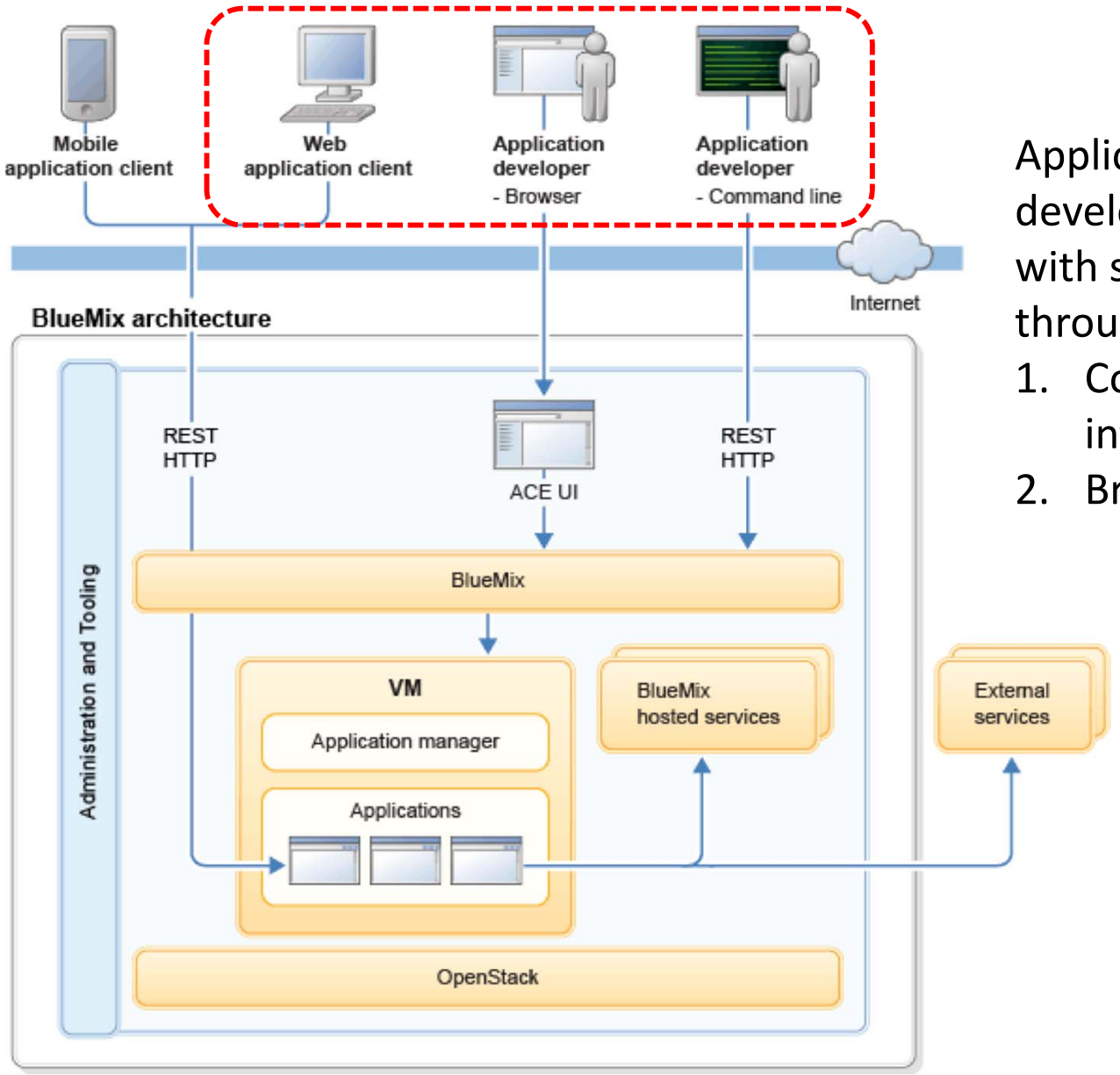
- Example 2: A web server application for generating personality insights using cognitive analysis SaaS
  - perform cognitive analysis using Watson service from a Node.js web server application and retrieve results

- Example 3: Using SaaS services for data analytics on cloud
  - use IBM Db2 Data Warehousing and DSX SaaS services for in-cloud data analytics with SQL and R scripting languages

Milestones

1. Successful login
2. Install CLI
3. Load web app
4. Load Watson web app
5. Run SQL query
6. Run R script and visualize data

Bluemix High-level Architecture



Application developer interacts with systems through:

1. Command line interface
2. Browser interface

Bluemix Cloud Foundry

- Users focus on application code, i.e., don't have to worry about the OS and infrastructure layers

- Cloud Foundry is an open source PaaS
  - Corporate users: SAS, Cisco, Rakuten, Baidu, SAP Verizon, ..
  - Leverages on the broad community
  - Push, extend and manage applications using
    - a command line tool – Bluemix CLI implements Cloud Foundry operations
    - eclipse plugin
    - DevOps …
  - Addresses PaaS lack of cross-compatibility among different cloud providers

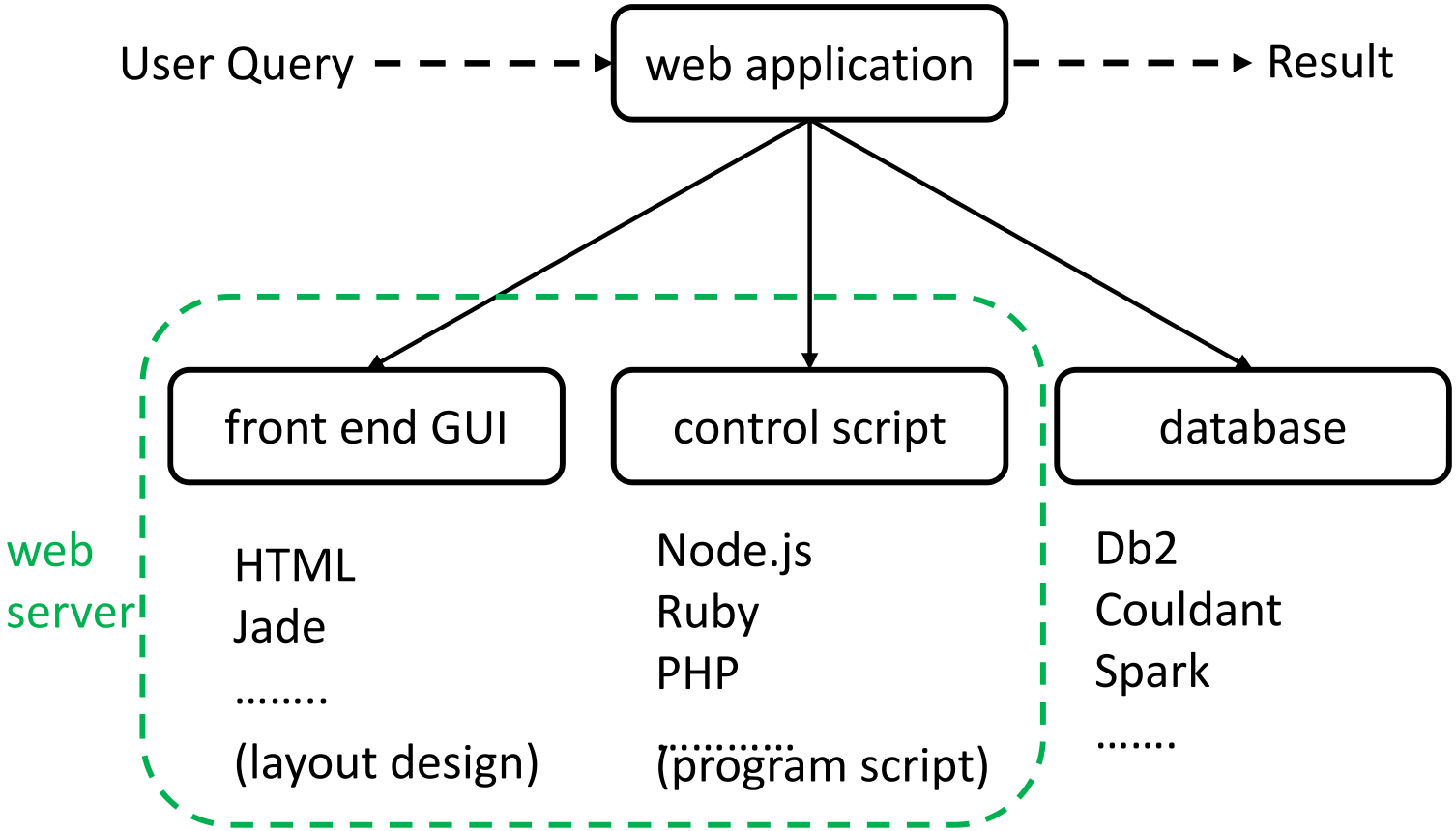
Video

Demo: Getting Started with Node.js on Bluemix 2016

Developing a Web Application

- Objective: set up a web server application using to retrieve data from an in-cloud database and display on a browser
- Example: Web application to retrieve and display economic data of countries stored in a IBM Db2 database, running on a cloud-based web server using node.js boilerplate
- Services can be added on top of the boilerplate runtime (eg. database service)
- Data file: World bank economic statistics

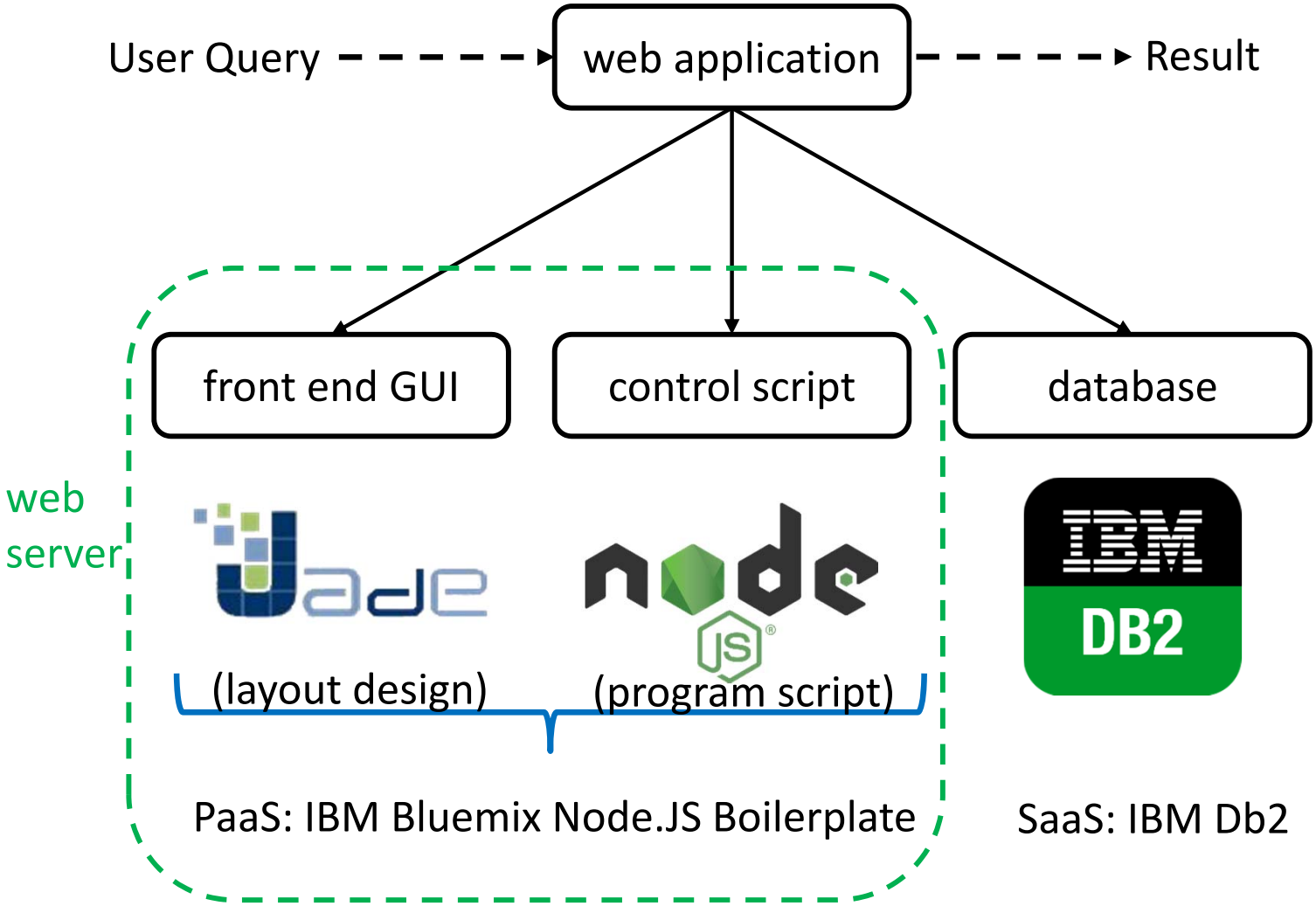
Developing a Web Application



Different technologies are offered by IBM Bluemix cloud

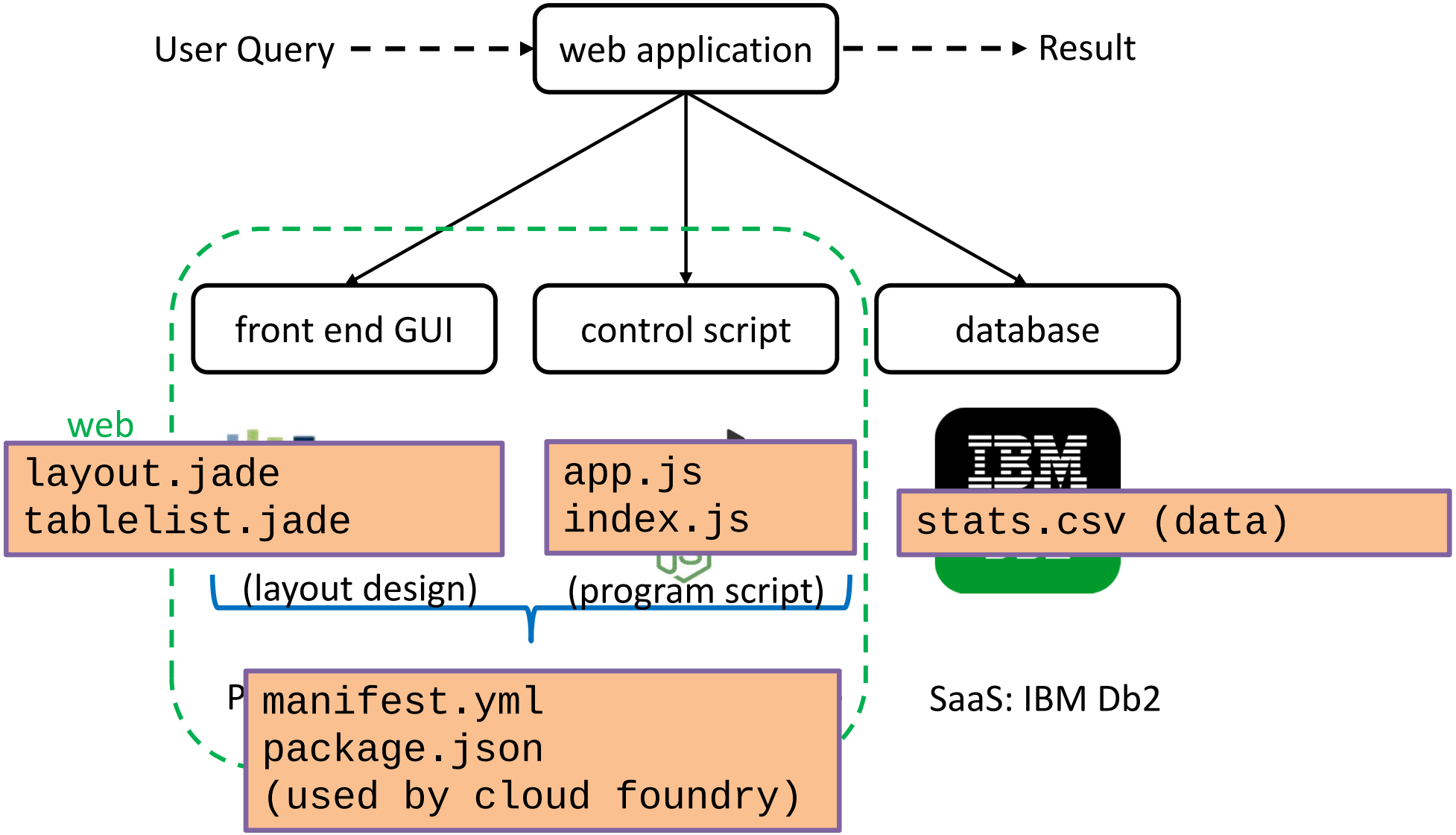
Developing a Web Application

What we are going to use for Example 1:

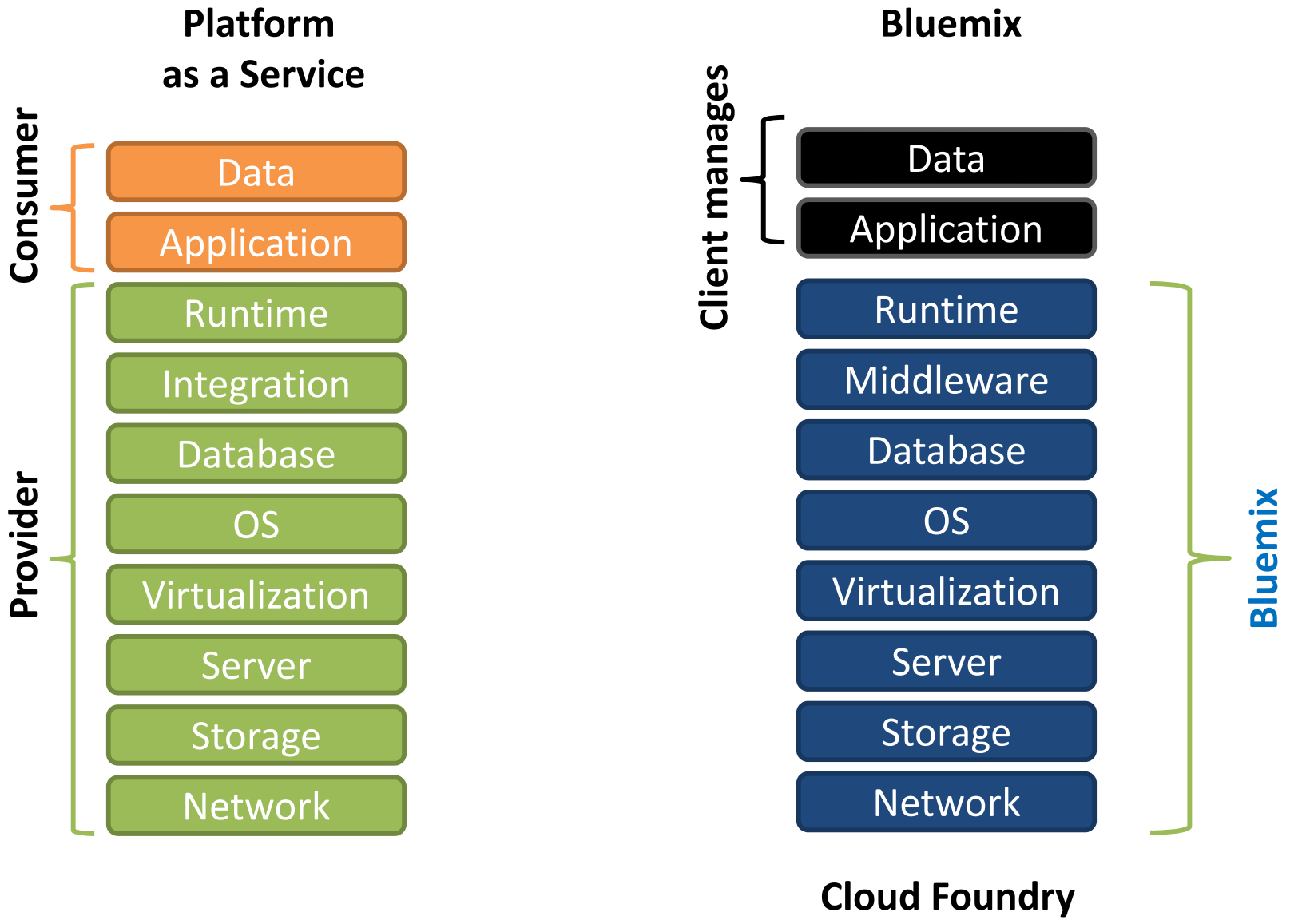


Developing a Web Application (relevant files)

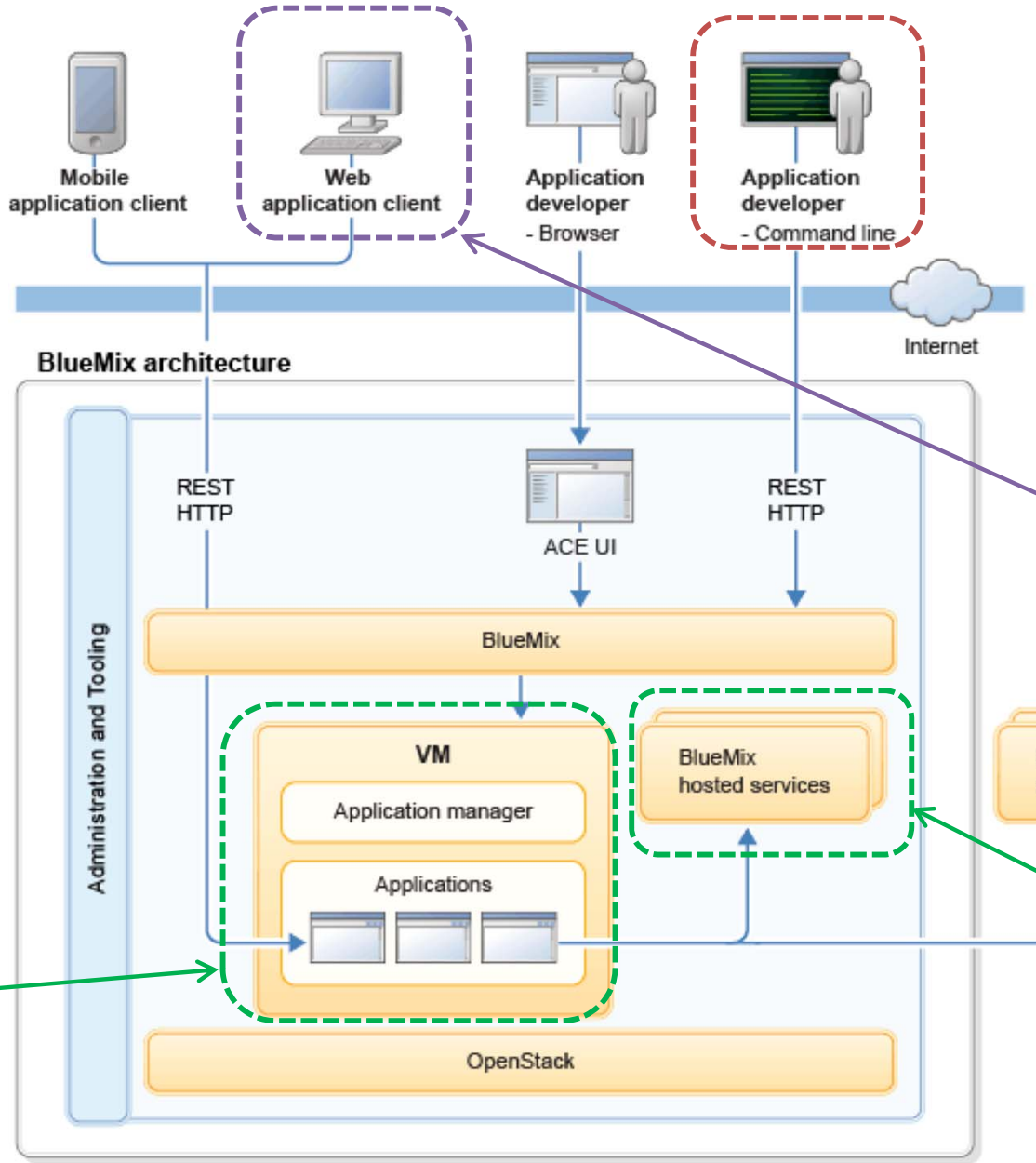
What we are going to use for Example 1:



What is PaaS?


Platform
as a Service
Consumer
Data
Application
Provider
Runtime
Integration
Database
OS
Virtualization
Server
Storage
Network
Bluemix
Client manages
Data
Application
Runtime
Middleware
Database
OS
Virtualization
Server
Storage
Network
Bluemix
Cloud Foundry

Web Application Architecture



Use node.js and push the app to the Cloud using CloudFoundry CLI

Use web-browser to browse the created web application

Db2 database

web server

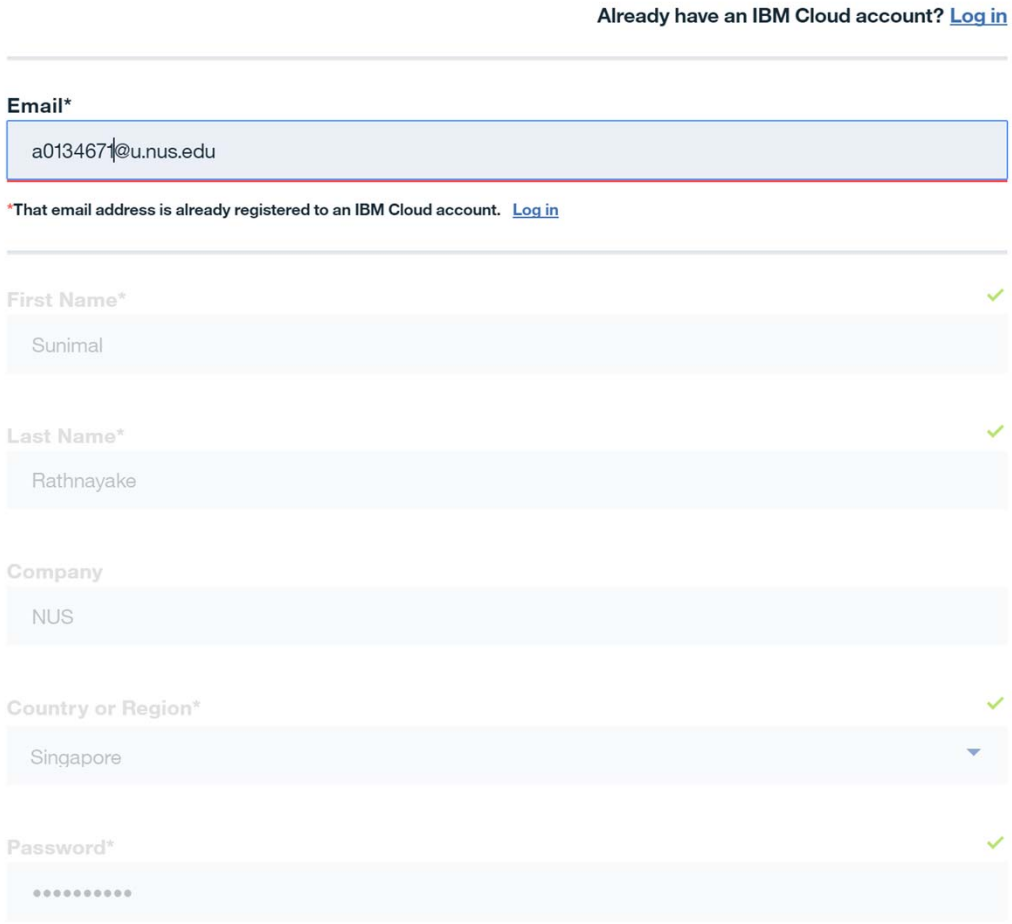
Main Steps: Web Application using Node.js

1. Sign up for IBM Bluemix
2. Select Db2 for Transactions from the catalog – only necessary if the application needs a database
3. Upload data to the newly created Db2 database create necessary tables
4. Modify/review data fields and get information necessary for remote access
5. Select Cloud Foundry Node.js app from the catalog – this is needed for every new application
6. Provide a hostname that serves as the web application URL – hostname.mybluemix.net
7. Install the Bluemix CLI (command line interface) client on your local machine - this is required if you have not done so previously
8. Add a connection to the database created in steps 2-4
9. To start a new web application, download the code (index.js, manifest.yml, app.js, package.json,..) onto your local directory
10. Modify manifest.yml with database host name
11. Deploy/upload application on Bluemix. Using the Bluemix CLI, connect the Bluemix client to the selected Bluemix region (server)
12. Using the Bluemix CLI, login to Bluemix and, upload your application to Bluemix!
13. To access the web application, enter your host URL (in step 5)

Step 1: Signing up

- IBM Cloud sign up
  https://console.bluemix.net/

- Fill in the form with your details.
  Make sure to use the NUS school email.



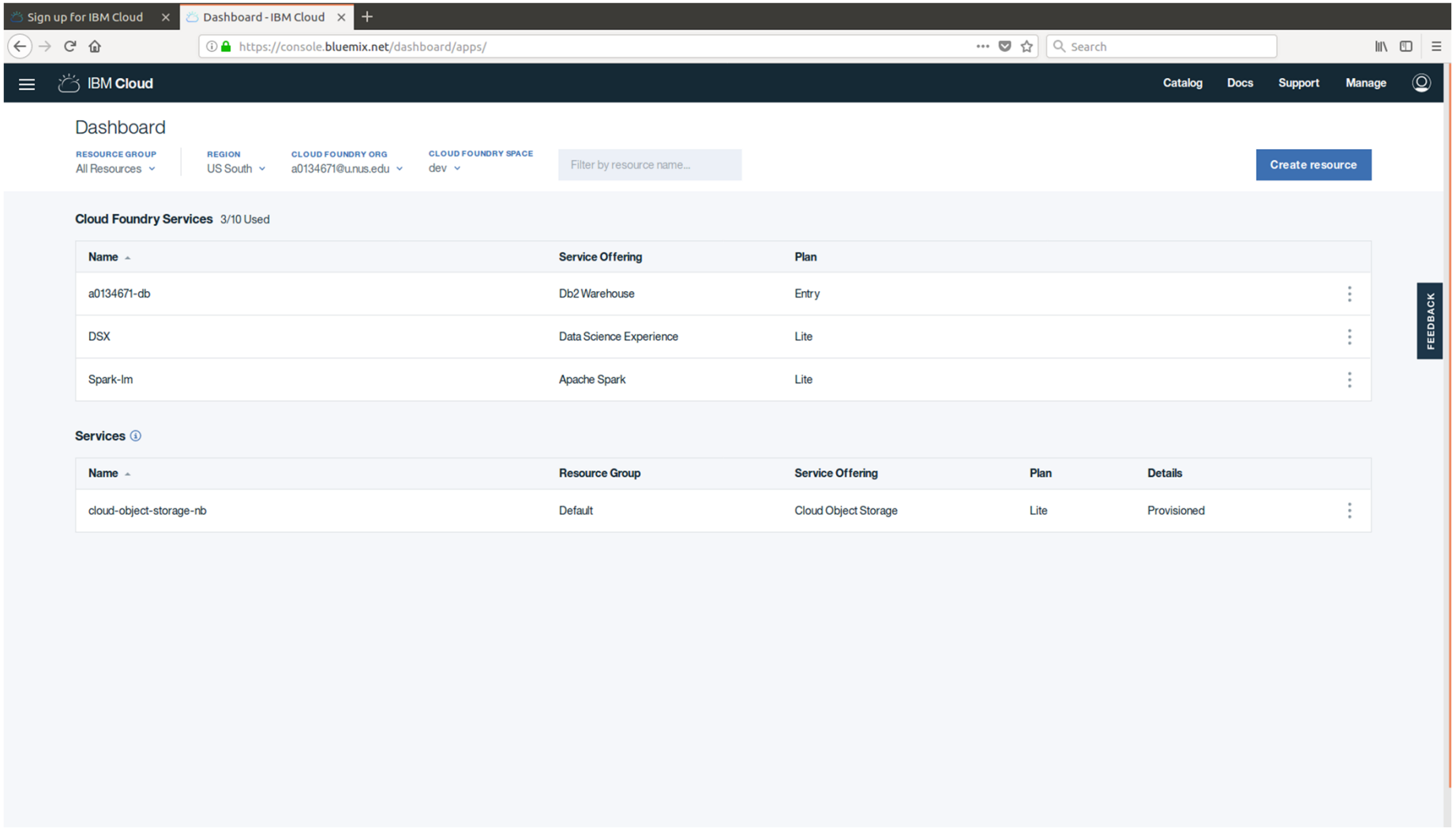
Milestone 1: Successful Login

- Did you successfully log in to IBM Bluemix?

- Let’s wait until everyone is done!

- In the meantime:
  - Explore the array of services offered in Bluemix

IBM Cloud Platform – Bluemix

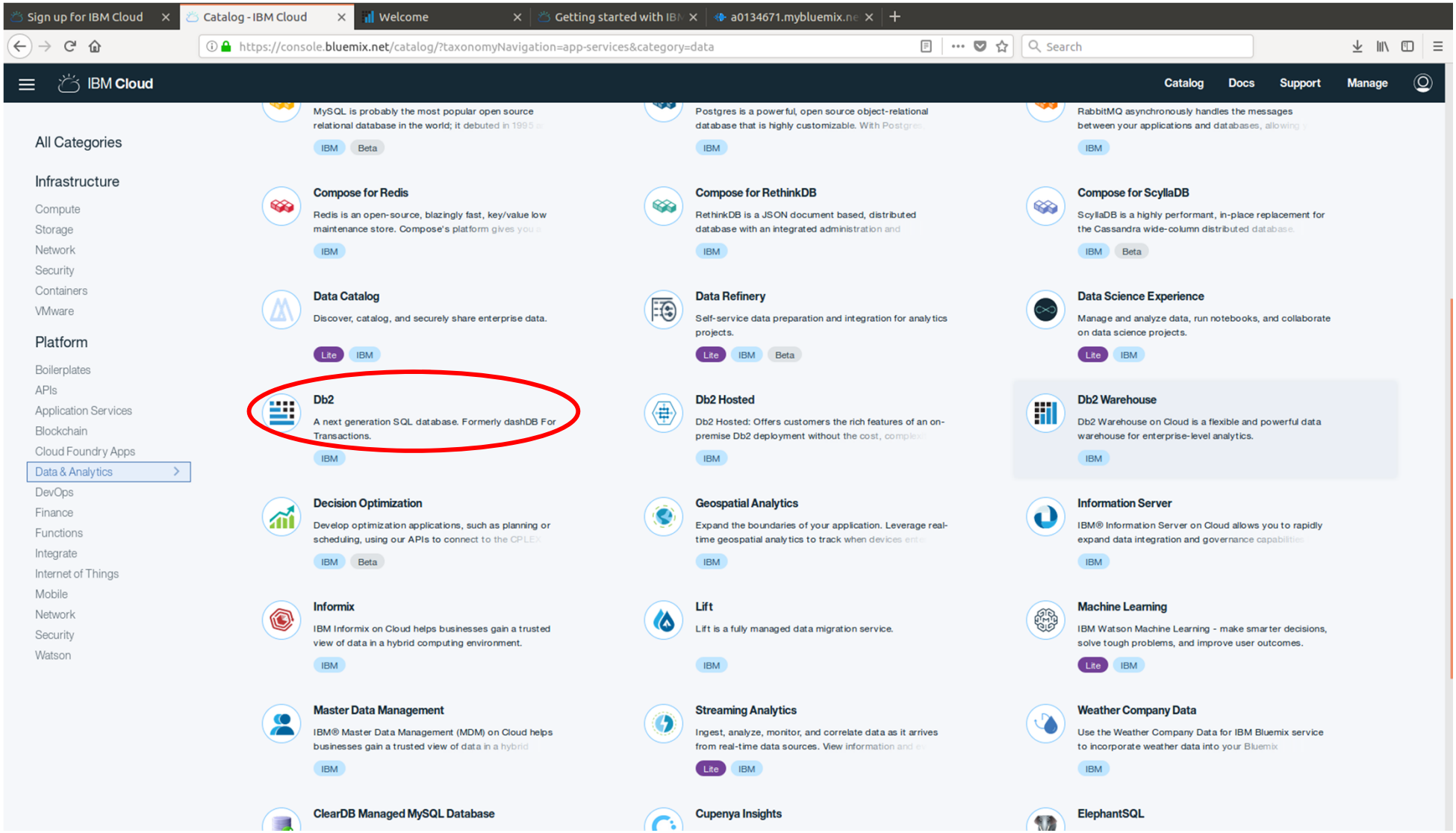


How to instantiate Bluemix services?

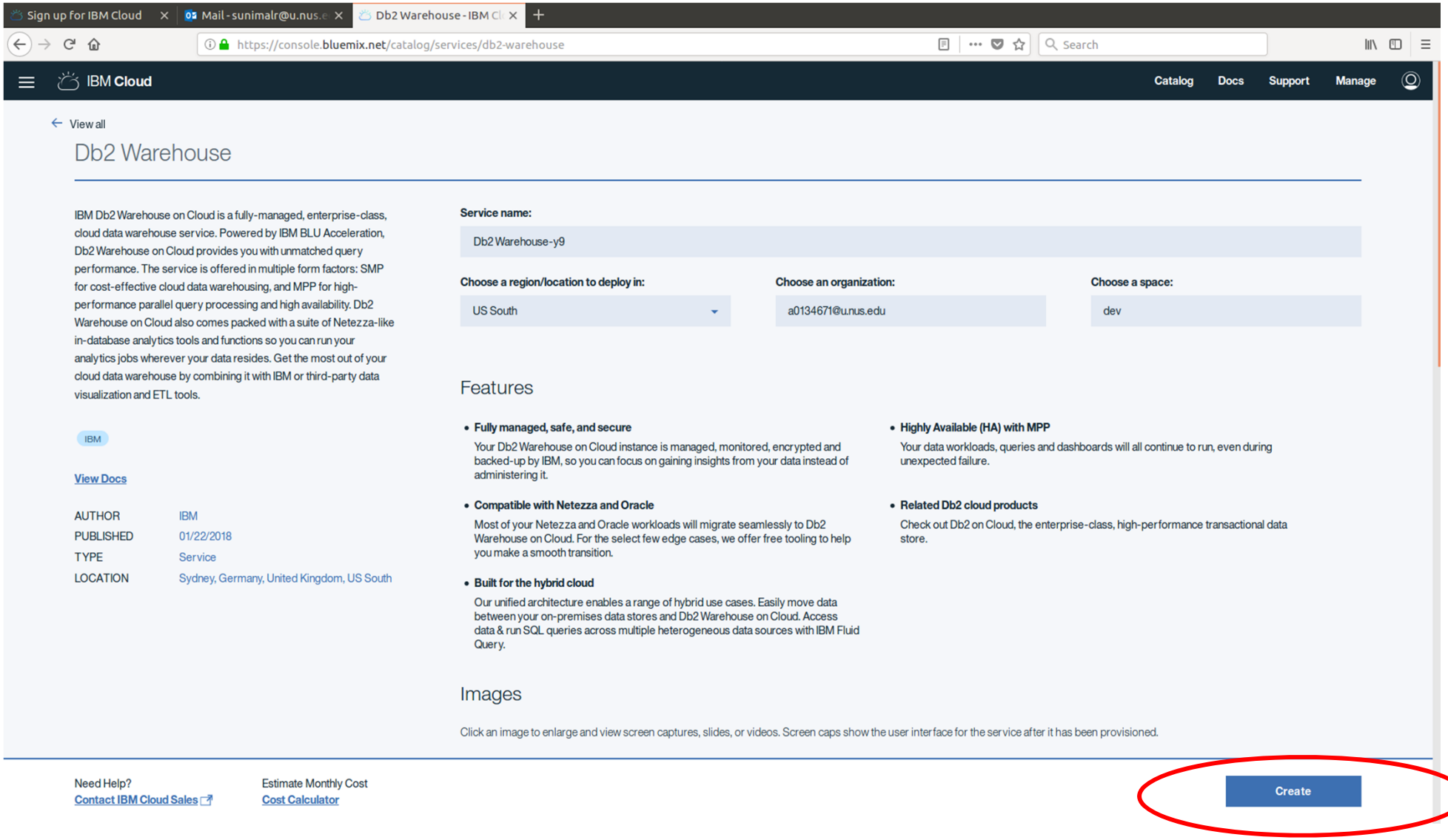
- Two ways to instantiate a Bluemix service
  1. Using the web browser interface (Bluemix web console)
     - eg. select Db2 on the web interface and do the setup through the web browser.
     - created service can be later connected with other application such as web applications
     - we demonstrate this in Example 1 (Db2 service)

  2. Using the command line interface (IBM Cloud CLI)
     - eg. issue a command through the CLI to instantiate a service (managed by cloud foundry software)
     - we demonstrate this in Example 2 (Watson personality insights service)

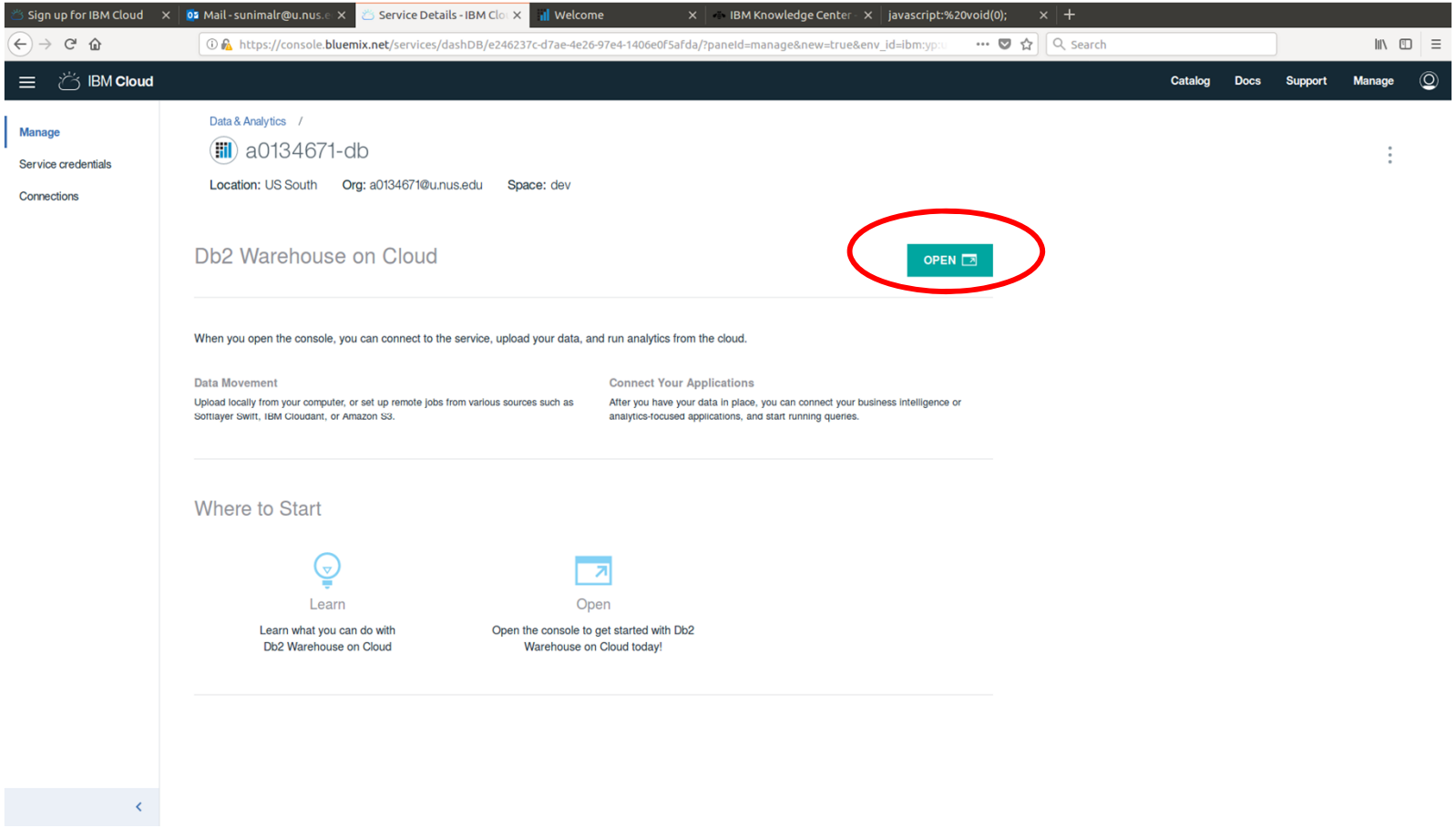
Step 2: Select Db2 from Catalog



Step 2: Db2 Setup

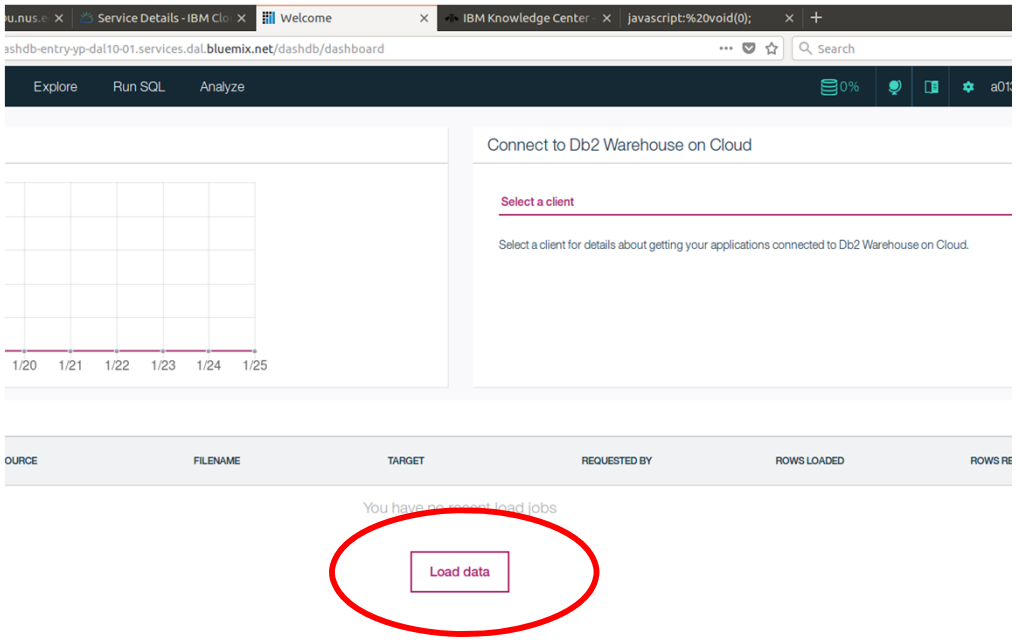


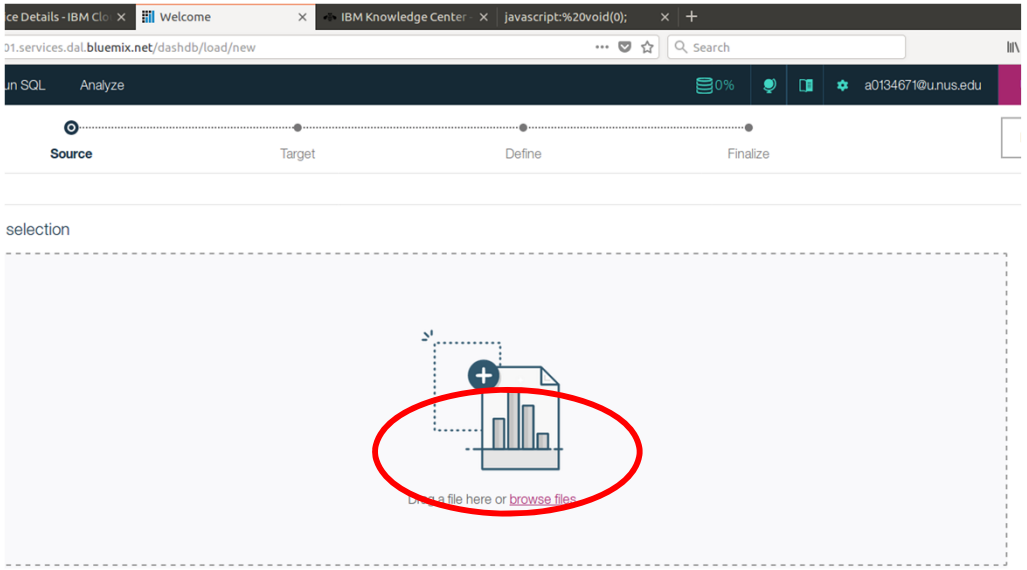
Step 2: Open Db2 Console



Step 3: Upload data

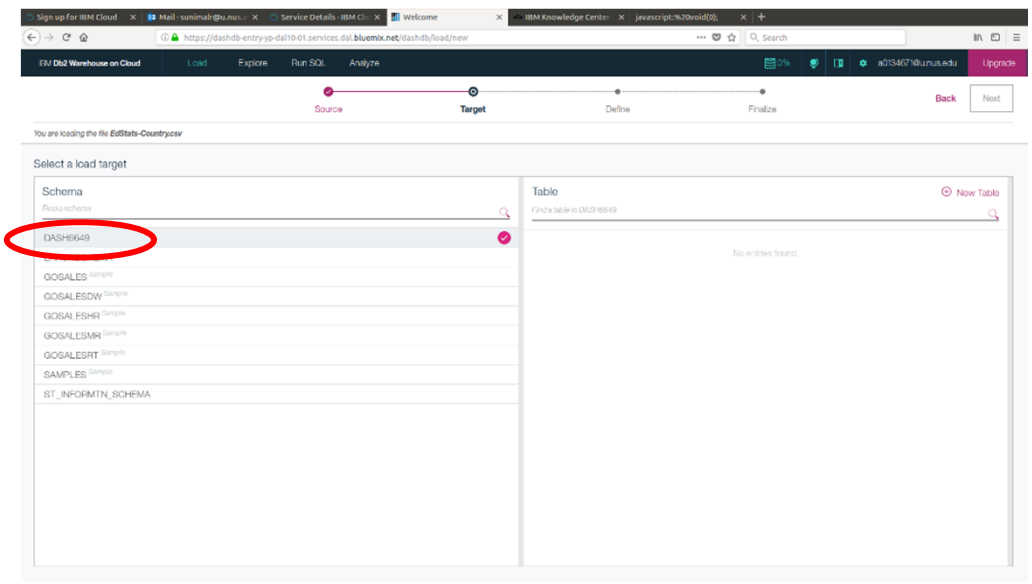
- First download stats.csv data file from the course webpage, then upload to Db2

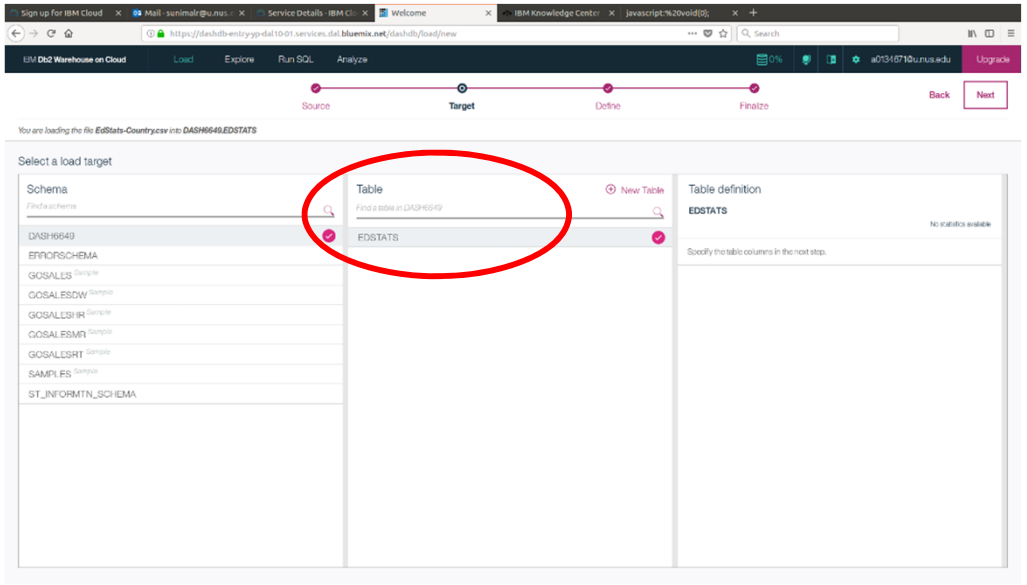




Step 4: Select Scheme, Table Name and, Review Table

- Create a new table “STATS”

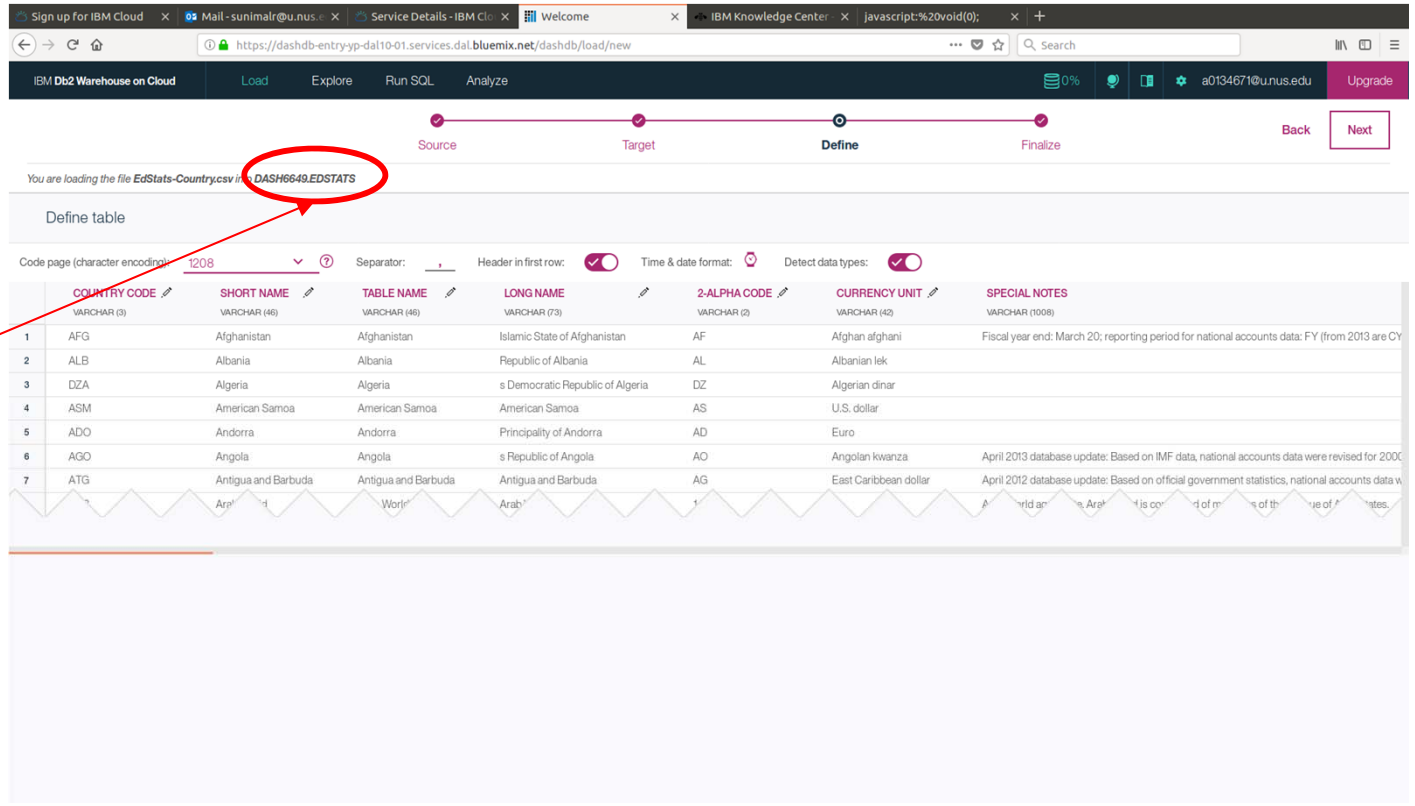




To connect database with another app, we need to note the “scheme” and the “tables” name.

Step 4: Select Scheme and Table Name

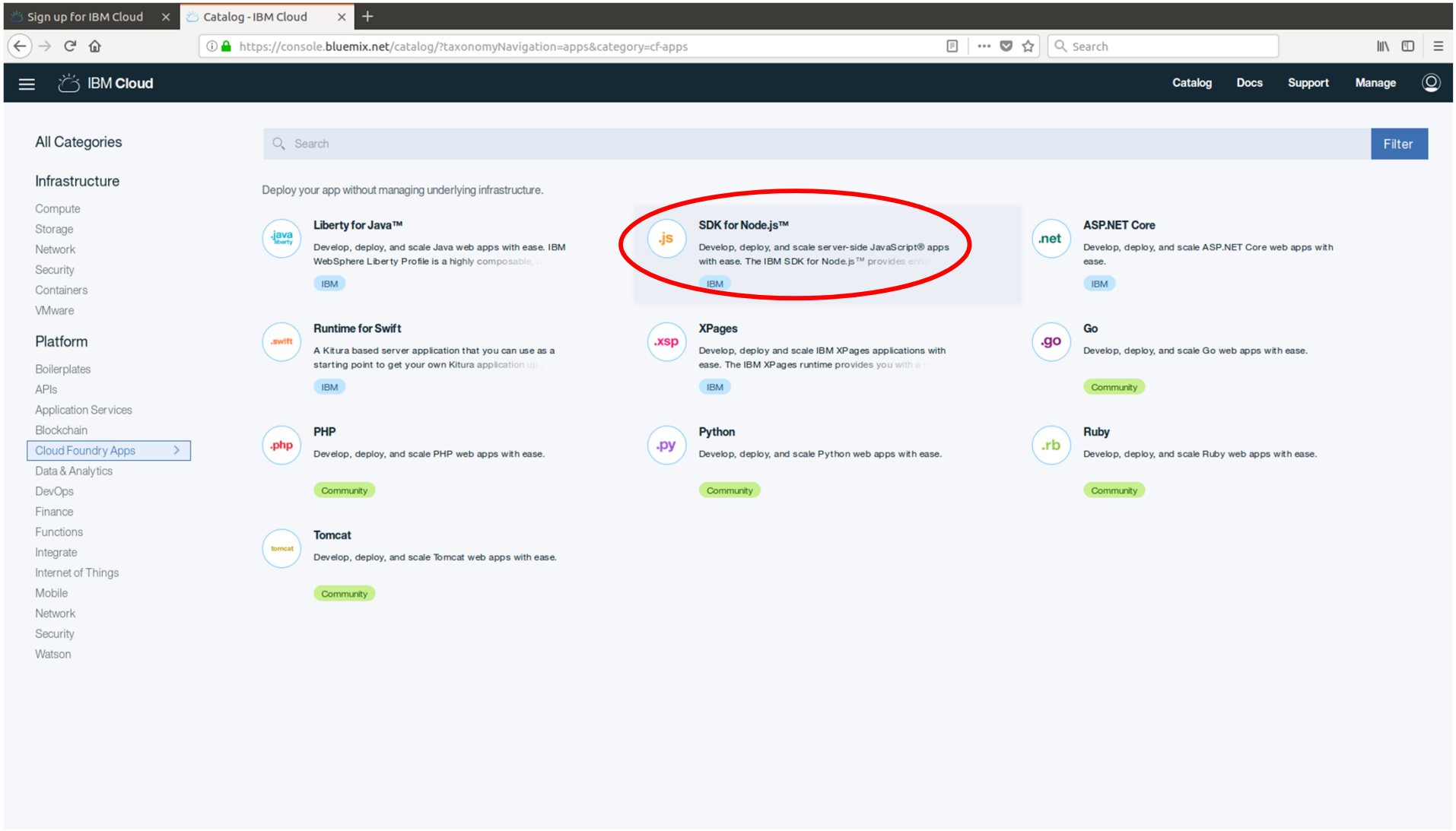
- Preview
  - You can change the column names if necessary
- Review settings
- Click Begin Load



Note the table name.
We need this in next steps

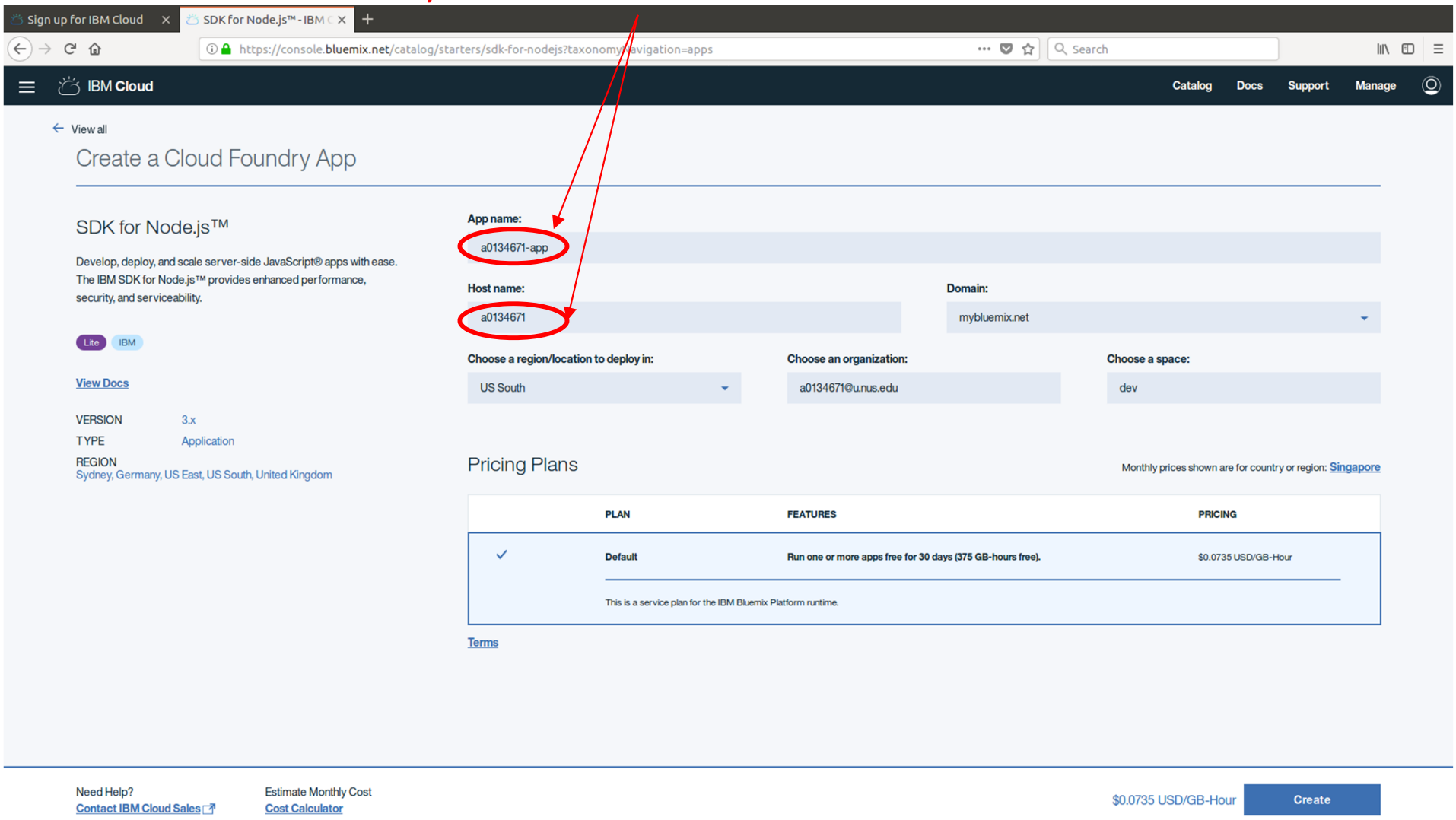
Step 5: Select Cloud Foundry Node.js

- Go back to catalog and select SDK for Node.js under CloudFoundry apps

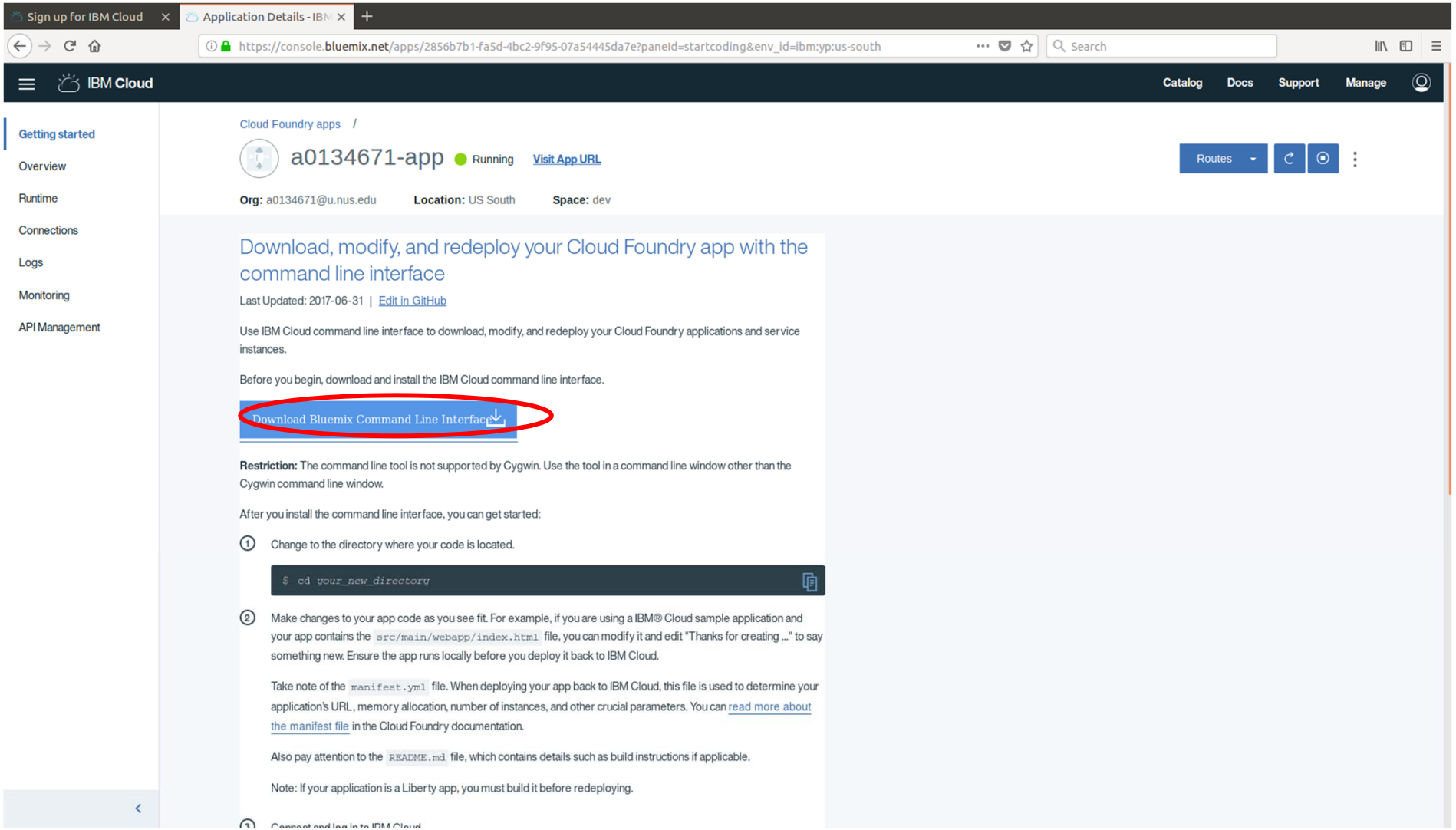


Step 6: Provide required app details

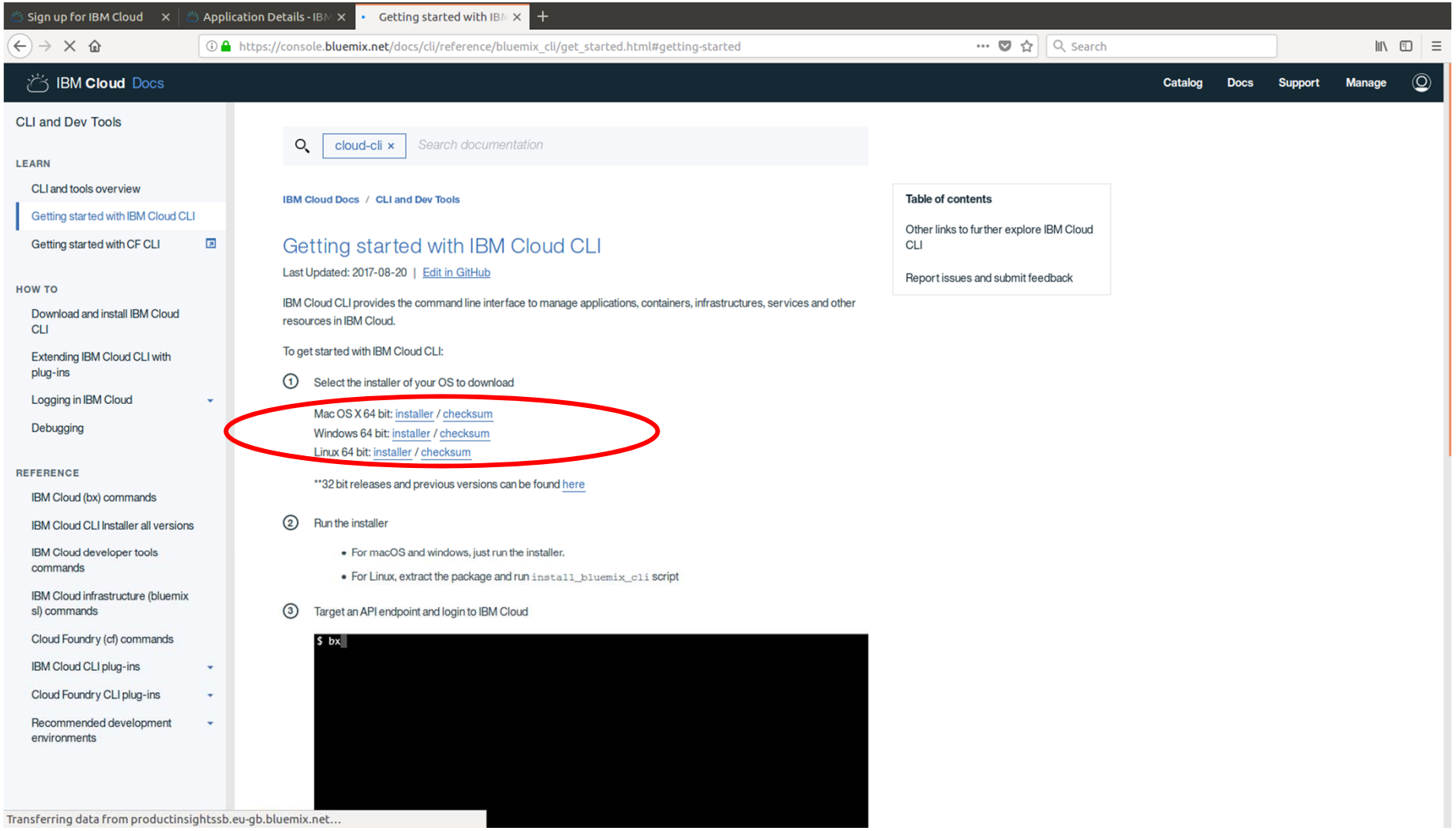
Use a unique app name and hostname.
I use my NUSNET id.



Step 7 : Install Bluemix CLI

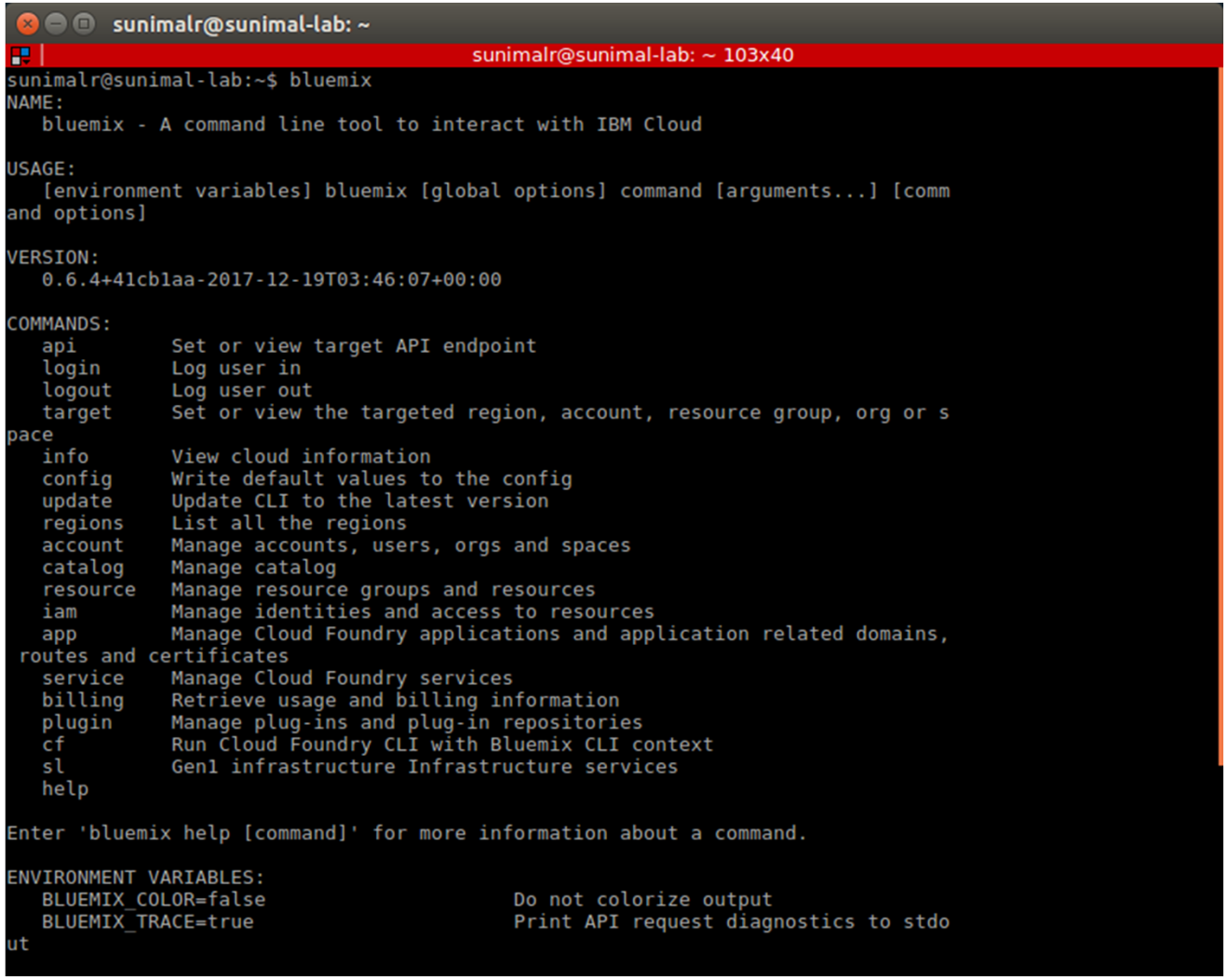


Step 7 : Install Bluemix CLI



Step 7 : Install Bluemix CLI

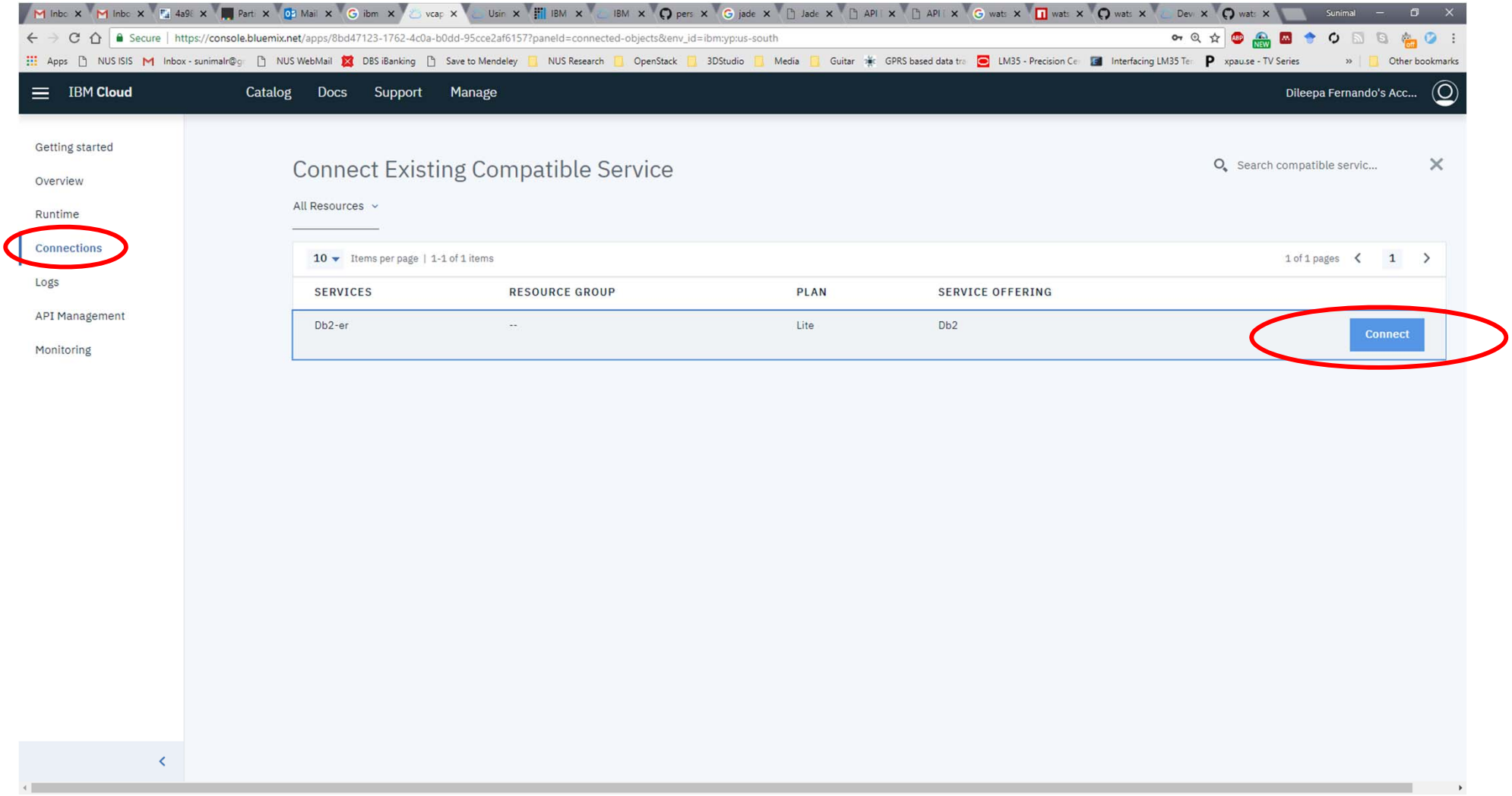
- Open command line and enter ‘bluemix’ and see if installation is a success. (as below)



Milestone 2: Install CLI

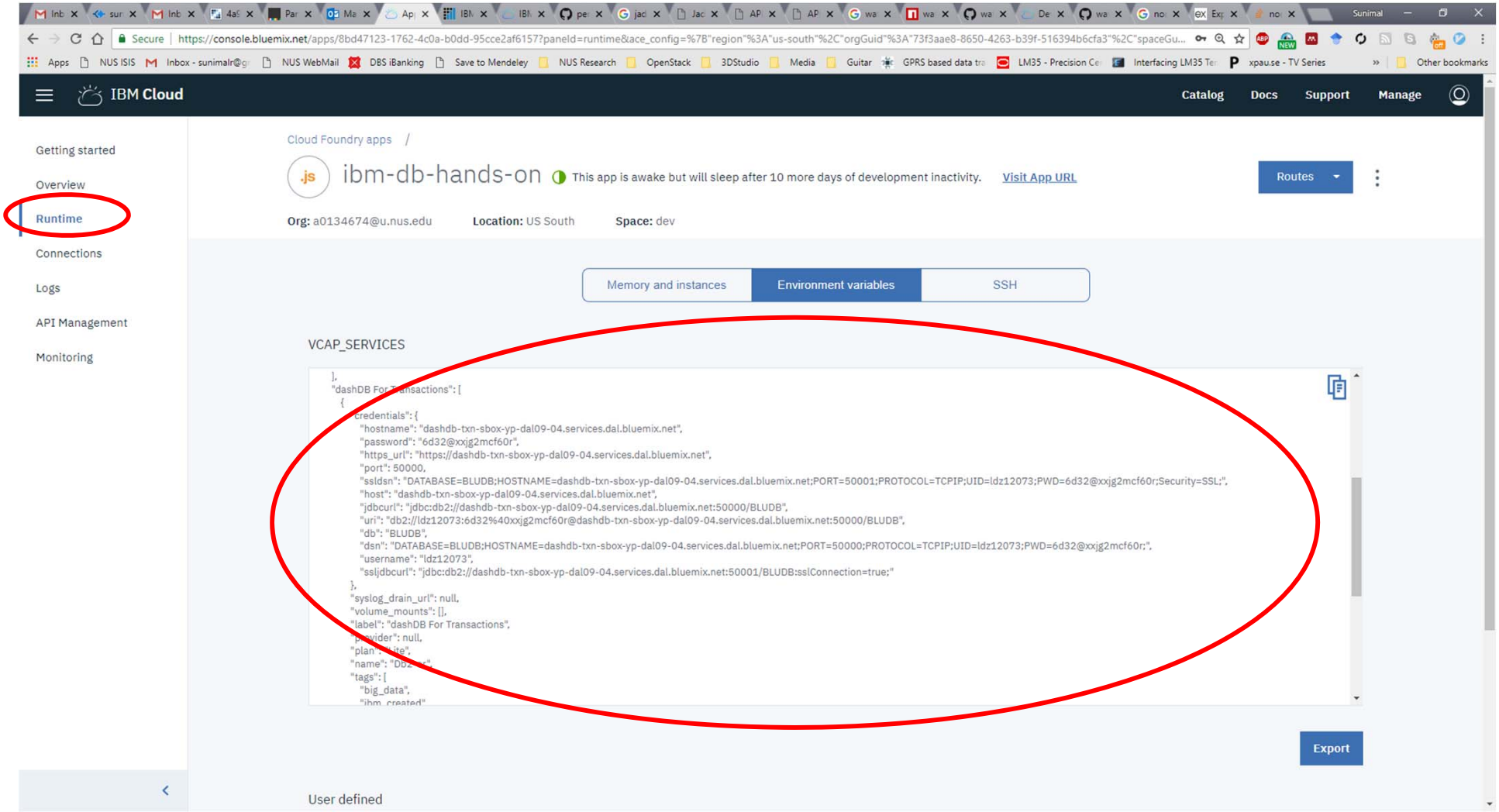
- Did you successfully install Bluemix CLI?
- Bluemix CLI implements Cloud Foundry services.
- Let's wait until everyone is done!
- In the meantime:
  - Read more about cloud foundry at: http://searchcloudcomputing.techtarget.com/definition/Cloud-Foundry

Step 8: Connect the web app to Db2 database



Step 8: Note the environment variables

Under runtime tab, you can see VCAP_SERVICES environment variables. This information is needed for external connections.



Step 9: Download Node.js content files

- Download the db-app.zip file from the course web page to your local computer and extract

- File structure
  - app.js – web application core
  - manifest.yml – deployment parameters for Cloud Foundry
  - package.json – app dependencies
  - routes/index.js – functions to connect to Database

- Take note of manifest.yml and package.json which sets deployment parameters and resolve dependencies when deploying applications via Cloud Foundry and app.js file which sets up server variables and initiates the web server

List of data and code with location (Example 1)

- Data files
  - stats.csv (downloaded at above Step3)

- Source files(db-app.zip file downloadable from course webpage)
  - db-app
    - views
      - layout.jade
      - tablelist.jade
    - routes
      - index.js
    - app.js
    - manifest.yml
    - package.json

layout design

control logic

dependencies and app properties

app.js

```
var express = require('express');
var routes = require('./routes');
var http = require('http');
var path = require('path');
var ibmdb = require('ibm_db');
require('cf-deployment-tracker-client').track();


var app = express();

// all environments
app.set('port', process.env.PORT || 3000);
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');
app.use(express.favicon());
app.use(express.logger('dev'));
app.use(express.json());
app.use(express.urlencoded());
app.use(express.methodOverride());
app.use(express.cookieParser('your secret here'));
app.use(express.session());
app.use(app.router);
app.use(express.static(path.join(__dirname, 'public')));
var db2;
var hasConnect = false;

// development only
if ('development' == app.get('env')) {
  app.use(express.errorHandler());
}

if (process.env.VCAP_SERVICES) {
    var env = JSON.parse(process.env.VCAP_SERVICES);
                if (env['dashDB']) {
        hasConnect = true;
                                db2 = env['dashDB'][0].credentials;
                }

}

if ( hasConnect == false ) {

   db2 = {
        db: "BLUDB",
        hostname: "xxxx",
        port: 50000,
        username: "xxx",
        password: "xxx"
    };
}

var connString = "DRIVER={DB2};DATABASE=" + db2.db + ";UID=" + db2.username + ";PWD=" + db2.password + ";HOSTNAME=" + db2.hostname + ";port=" +
db2.port;

app.get('/', routes.listSysTables(ibmdb,connString));

http.createServer(app).listen(app.get('port'), function(){
  console.log('Express server listening on port ' + app.get('port'));
});
```

package.json

```
{
  "engines": {
    "node": "^4"
  },
  "name": “db2dbnodesample",
  "version": "0.0.1",
  "description": "A sample node app for connecting to db2 service on Bluemix",
  "dependencies": {
    "cf-deployment-tracker-client": "^0.0.8",
    "express": "^3.5.3",
    "ibm_db": "^2.0.0",
    "jade": "^1.11.0"
  },
  "scripts": {
    "start": "node app.js"
  },
}
}
```

manifest.yml

```
---
declared-services:
  dashDB-nodesample:
    label: "dashDB For Transactions"
    plan: Lite
applications:
# replace the host variable below with your own unique one, as this
one can be already taken
- name: ibm-db-hands-on
  memory: 64M
  instances: 1
  path: .
  host: sunimal
  framework: node
  command:    node app.js
  buildpack:  sdk-for-nodejs
  services:
  - Db2-er
```

index.js

```
exports.listSysTables = function(ibmdb,connString) {
    return function(req, res) {


       ibmdb.open(connString, function(err, conn) {
               if (err ) {
                res.send("error occurred " + err.message);
               }
               else {
               conn.query("SELECT * FROM LDZ12073.STATS FETCH FIRST 10 ROWS ONLY", function(err, tables, moreResultSets) {


               if ( !err ) {
               res.render('tablelist', {
                               "tablelist" : tables,
                               "tableName" : "10 rows from the STATS table",
                               "message": "Congratulations. Your connection to Db2 is successful."

                                                                                  });
               } else {
                  res.send("error occurred " + err.message);
               }
               /*
               Close the connection to the database
               param 1: The callback function to execute on completion of close function.
               */
               conn.close(function(){
               console.log("Connection Closed");
               });
               });
               }
               } );

               }
}
```

Note: index.js is located in the ‘routes’ folder

layout.jade

```
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    block content
```

tablelist.jade

```
extends layout

block content
    h1=message
    h2=tableName

    form#list-form
      div#main
        table#list
          thead
            tr
              for hdr in Object.keys(tablelist[0])
                th #{hdr}
          tbody
            if tablelist && tablelist.length > 0
              each row in tablelist
                tr
                  each col in row
                    td #{col}
```

Note: located in the ‘views’ folder

Step 10: Modify index.js

Use the schema.table name from step 4

```
exports.listSysTables = function(ibmdb,connString) {
    return function(req, res) {


       ibmdb.open(connString, function(err, conn) {
               if (err ) {
                res.send("error occurred " + err.message);
               }
               else {
               conn.query("SELECT * FROM LDZ12073.STATS FETCH FIRST 10 ROWS ONLY", function(err, tables, moreResultSets) {


               if ( !err ) {
               res.render('tablelist', {
                              "tablelist" : tables,
                              "tableName" : "10 rows from the STATS table",
                              "message": "Congratulations. Your connection to Db2 is successful."

                                                                                  });
               } else {
                  res.send("error occurred " + err.message);
               }
               /*
               Close the connection to the database
               param 1: The callback function to execute on completion of close function.
               */
               conn.close(function(){
               console.log("Connection Closed");
               });
               });
               }
               } );

               }
}
```

Note: index.js is located in the ‘routes’ folder

Step 10: Modify manifest File

```
---
declared-services:
  dashDB-nodesample:
    label: "dashDB For Transactions“
    plan: Lite
applications:
# replace the host variable below with your own unique one, as this
one can be already taken
- name: your app name here
  memory: 64M
  instances: 1
  path: .
  host: your host name here
  framework: node
  command:    node app.js
  buildpack:  sdk-for-nodejs
  services:
  - name for the database (eg: db-<your nusnet id>) – we did this in
step 2
```

this is the maximum allowed RAM for the app. When you create the app in web console (in step 2 and 3), you may be asked to select memory size. Use that number here.

Steps 11, 12: Cloud Foundry and Bluemix

- Open command line and change directory into the folder with code downloaded in the previous step
- Use the Bluemix CLI to connect to the Bluemix API (Depending on the region you are logged on to. ‘ng’ is for US South)
  - bluemix api https://api.ng.bluemix.net
- Login to Bluemix
  - >bluemix login -u <bluemix_id> -o <organization> -s <space>
  - use the IBM ID and password
- upload your application to Bluemix
  - bluemix app push <appname>

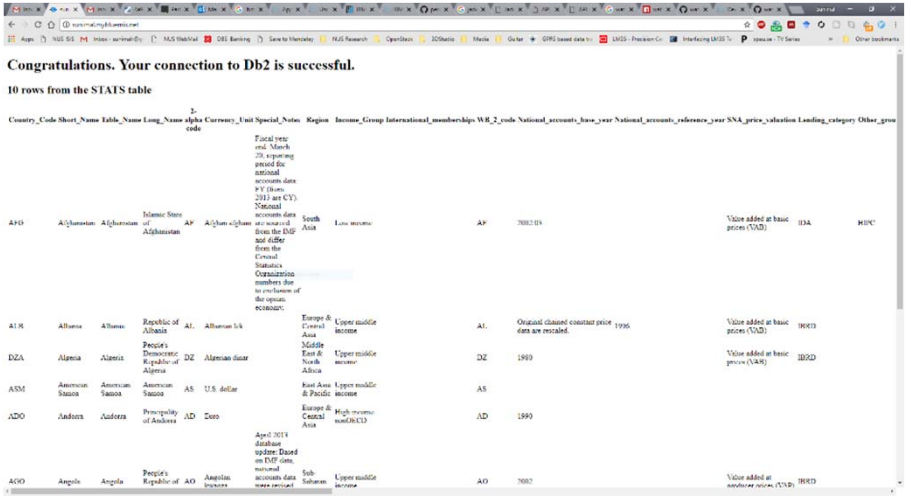
Step 13: View App

- Go to your favorite browser and enter the URL:

  http://your_host_name.mybluemix.net

- Alternatively, you can try mine at

http://sunimal.mybluemix.net



Congratulations. Your connection to Db2 is successful.

10 rows from the STATS table

Country_Code	Short_Name	Table_Name	Long_Name	3-alpha code	Currency_Unit	Special_Notes	Region	Income_Group	International_memberships	WB_2-code	National_accounts_base_year	National_accounts_reference_year	SNA_price_valuation	Lending_category	Other_grou
AFG	Afghanistan	Afghanistan	Islamic State of Afghanistan	AF	Afghani	Fiscal year end: March 20; reporting period for national accounts data: FY (from 2013 are CY). National accounts data are sourced from the IMF and differ from the Central Statistics Organization numbers due to exclusion of the opium economy.	South Asia	Low income		AF	2002/03		Value added at basic prices (VAB)	IDA	HIPC
ALB	Albania	Albania	Republic of Albania	AL	Albanian lek		Europe & Central Asia	Upper middle income		AL	Original chained constant price data are rescaled.	1996	Value added at basic prices (VAB)	IBRD	
DZA	Algeria	Algeria	People's Democratic Republic of Algeria	DZ	Algerian dinar		Middle East & North Africa	Upper middle income		DZ	1980		Value added at basic prices (VAB)	IBRD	
ASM	American Samoa	American Samoa	American Samoa	AS	U.S. dollar		East Asia & Pacific	Upper middle income		AS					
ADO	Andorra	Andorra	Principality of Andorra	AD	Euro		Europe & Central Asia	High income: nonOECD		AD	1990				
AGO	Angola	Angola	People's Republic of	AO	Angolan kwanza	Aged; 2015 database update: Based on IMF data; national accounts data were revised	Sub-Saharan	Upper middle income		AO	2002		Value added at producer prices (VAP)	IBRD	

Milestone 3: Load Web App

- Did you successfully load the web app?
- Let's wait until everyone is done!
- In the meantime:
  - Read more about core node.js files at: https://docs.cloudfoundry.org/buildpacks/node/node-tips.html

Web Application using Watson Analytics

- Watson is a cognitive analysis service powered by different AI techniques

- Watson could be used for different cognitive tasks such as
  - summarize a text to filter important points
  - analyze a person's personality based on a their resume
  - build interactive chat-bots to communicate with clients and answer their questions
  - …….

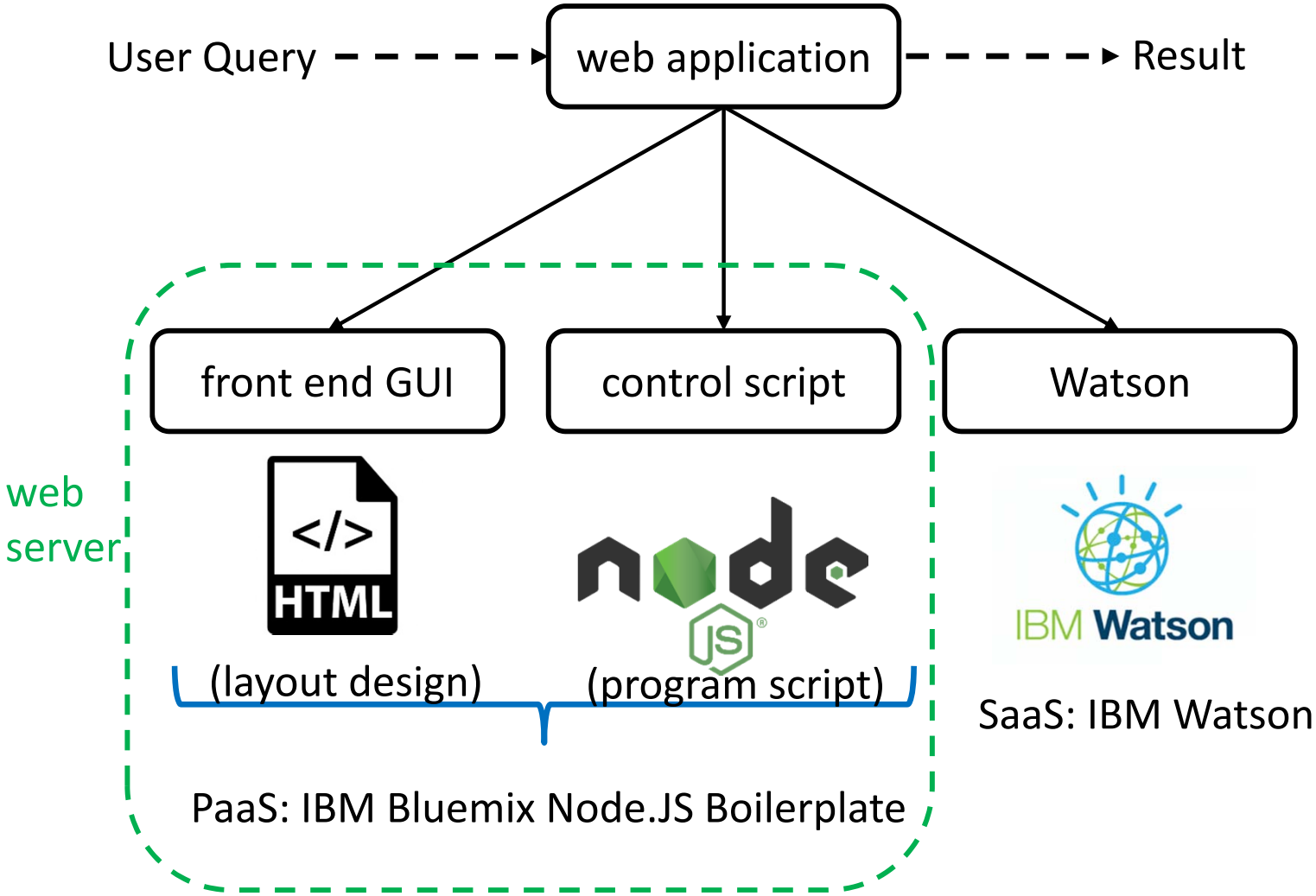
- Watson comes with a comprehensive API that helps integrate it with other IBM tools and services such as database, machine learning, web applications, among others

Watson Analytics

- Objective: use SaaS service (Watson) to get personality insights from a piece of text
- Example: a web server application to upload a part of a text and call IBM Watson services to get personality insights based on the uploaded text
- Data: part of a text from US former president Barak Obama's speech

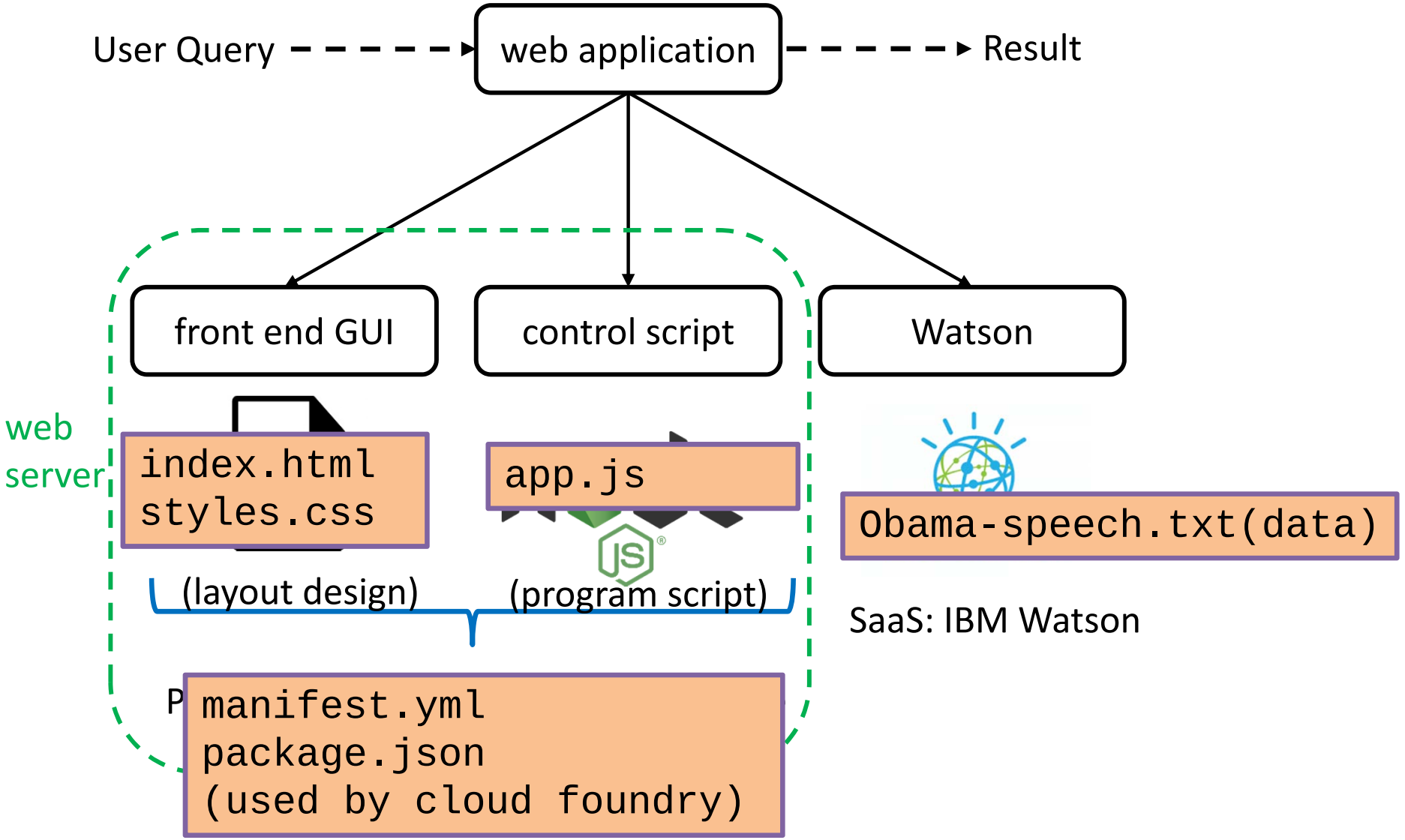
Developing a Web Application

What we are going to use for Example 2



Developing a Web Application

What we are going to use for Example 2



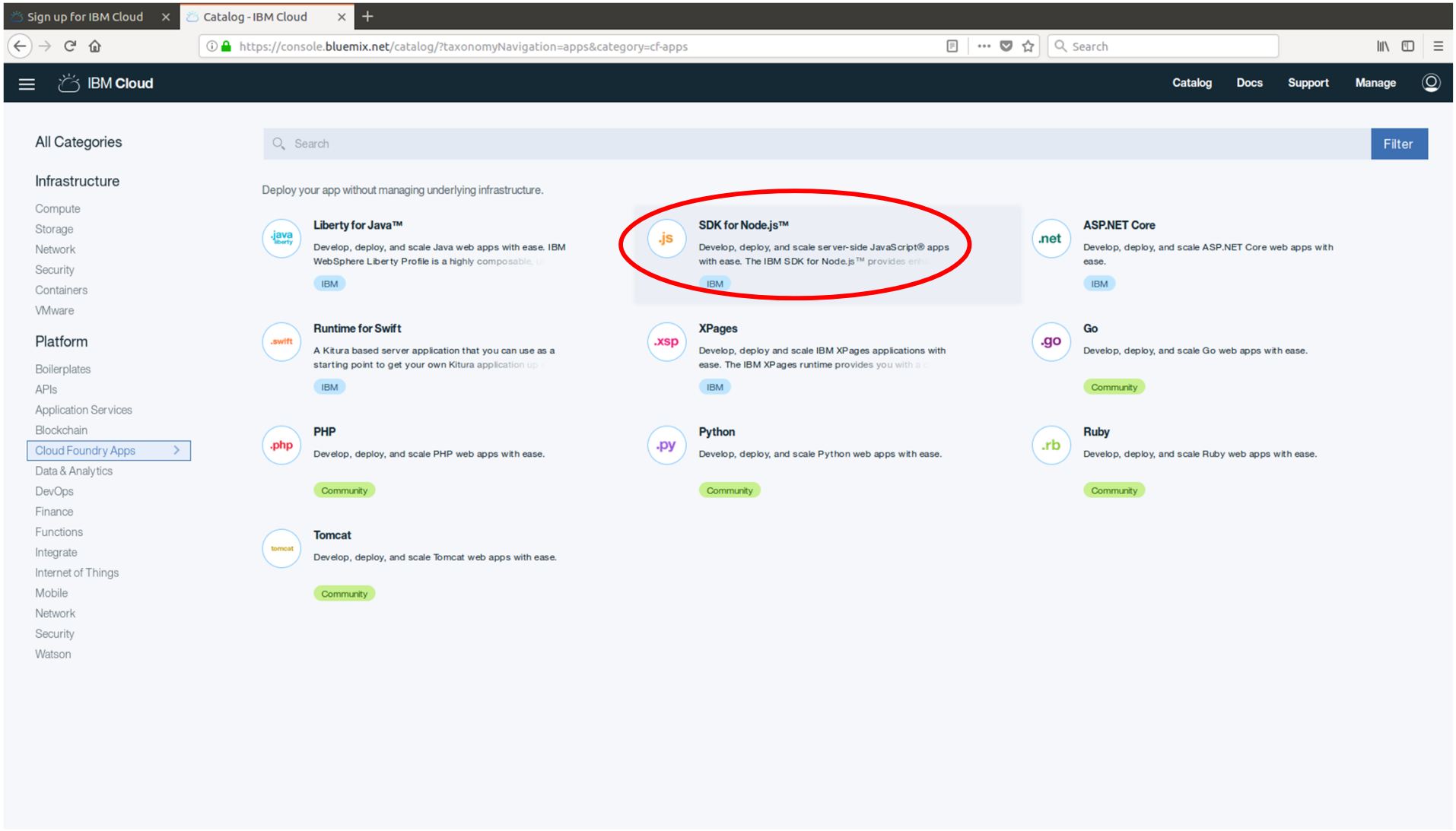
Watson Analytics: Main Steps in Setting up Web Server Application

1. Sign up for IBM Bluemix and go to the Catalog
2. Select Cloud Foundry Node.js app from the catalog – this is needed for every new application
3. Provide a hostname that serves as the web application URL – hostname.mybluemix.net
4. Install the Bluemix CLI (command line interface) client on your local machine - this is required only if you have not done so previously. We did this in Example 1.
5. To start a new web application, download the code (index.js, manifest.yml, app.js, package.json,..) onto your local directory from course web site
6. Modify manifest.yml (and other source code files as necessary)
7. Using Bluemix CLI, login to Bluemix
8. Create Watson personality insights service using CLI
9. Using the Bluemix CLI, upload your application to Bluemix!
10. To access the web application, enter your host URL (in step 3)

In Example 2 we demonstrate how to create a service with the CLI without using the web browser interface (step 8)

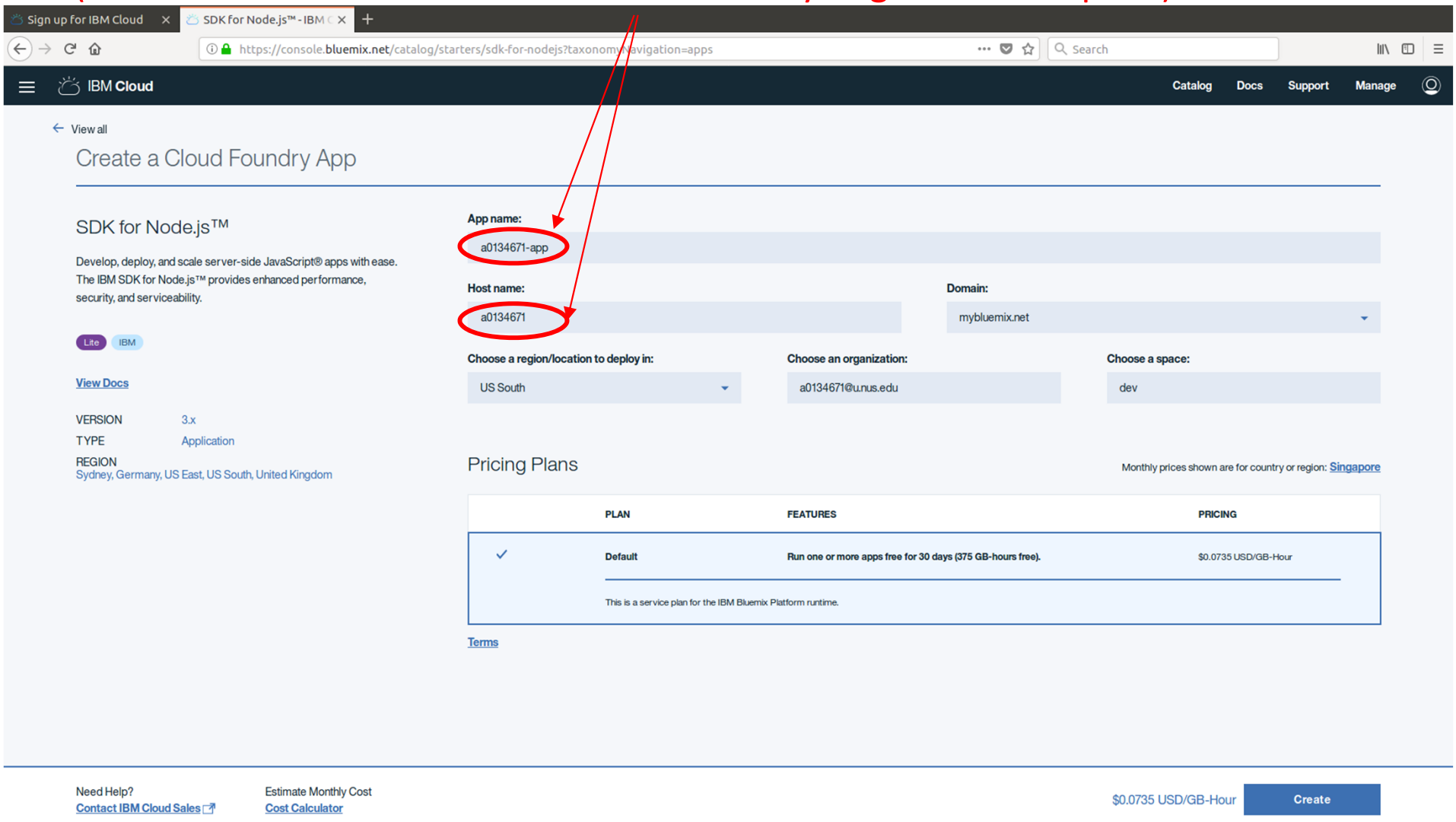
Step 2: Select Cloud Foundry Node.js

- Go back to catalog and select SDK for Node.js under CloudFoundry apps



Step 3: Provide required app details

Use a unique app name and hostname.
(this should be different from the hostname you gave in Example 1)



Step 5: Download Node.js content files

- Download the watson-app.zip file from the course web page to your local computer and extract it

- File structure
  - app.js – web application core
  - manifest.yml – deployment parameters for Cloud Foundry
  - package.json – app dependencies
  - public/index.html – web layout

- Take note of manifest.yml and package.json which sets deployment parameters and resolve dependencies when deploying applications via Cloud Foundry and app.js file which sets up server variables and initiates the web server

List of data and code with location (Example 2)

- Data files (watson-data.zip downloadable from course webpage)
  - watson-data
    - obama-speech.txt

- Source files (watson-app.zip downloadable from course webpage)
  - watson-app
    - public
      - stylesheets
        - style.css
      - index.html
    - routes
    - app.js
    - manifest.yml
    - package.json

layout design

control logic

dependencies and app properties

app.js

```
/*eslint-env node*/

//------------------------------------------------------------------------------
// node.js starter application for Bluemix
//------------------------------------------------------------------------------

// This application uses express as its web server
// for more info, see: http://expressjs.com
var express = require('express');

// cfenv provides access to your Cloud Foundry environment
// for more info, see: https://www.npmjs.com/package/cfenv
var cfenv = require('cfenv');

// create a new express server
var app = express();

// serve the files out of ./public as our main files
app.use(express.static(__dirname + '/public'));

// get the app environment from Cloud Foundry
var appEnv = cfenv.getAppEnv();

// start server on the specified port and binding host
app.listen(appEnv.port, '0.0.0.0', function() {

                    // print a message when the server starts listening
  console.log("server starting on " + appEnv.url);
});

var watson = require('watson-developer-cloud'),
    multer = require('multer');

var creds = appEnv.getServiceCreds('personality-insights-tutorial');
creds.version = 'v2';
var personalityInsights = watson.personality_insights(creds);

var uploading = multer({
    storage: multer.memoryStorage()
});

app.set('json spaces', 4);

app.post('/upload', uploading.single('file'), function (request, response) {
    console.log("file");
    var txtFile = request.file.buffer.toString();
    console.log(request.file.buffer);

    personalityInsights.profile({
        text: txtFile },
        function (error, result) {
            if (error) {
                response.send(error);
            }
            else {
                                                                                response.json(result);

            }
        }
    );
});


//---Deployment Tracker---------------------------------------------------------
require('cf-deployment-tracker-client').track();
```

Open the file in a text editor to clearly see/modify the content

package.json

```
{
  "name": "NodejsStarterApp",
  "version": "0.0.1",
  "description": "A sample nodejs app for Bluemix",
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {
    "body-parser": "^1.14.2",
    "cfenv": "1.0.x",
    "express": "4.12.x",
    "multer": "^1.1.0",
    "watson-developer-cloud": "1.x",
    "cf-deployment-tracker-client": "*"
  },
  "engines": {
    "node": "0.12.x"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/IBM-Bluemix/personality-insights-
nodejs-tutorial.git"
  }
}
```

public/index.html

```
<!DOCTYPE html>
<html>

  <head>
    <title>NodeJS Watson Application</title>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1">
    <link rel="stylesheet" href="stylesheets/style.css">
  </head>

  <body>
    <form action="/upload" method="POST"
enctype="multipart/form-data">
      Select a txt file to upload:
      <input type="file" name="file">
      <input type="submit" value="Upload File">
    </form>

  </body>

</html>
```

Step 6: Modify manifest.yml

```
---
declared-services:
  personality-insights-tutorial:
    label: personality_insights
    plan: lite
applications:
- path: .
  memory: 64M
  instances: 1
  domain: mybluemix.net
  name: sunimal-watson
  host: sunimal-watson
  disk_quota: 1024M
  services:
  - personality-insights-tutorial
```

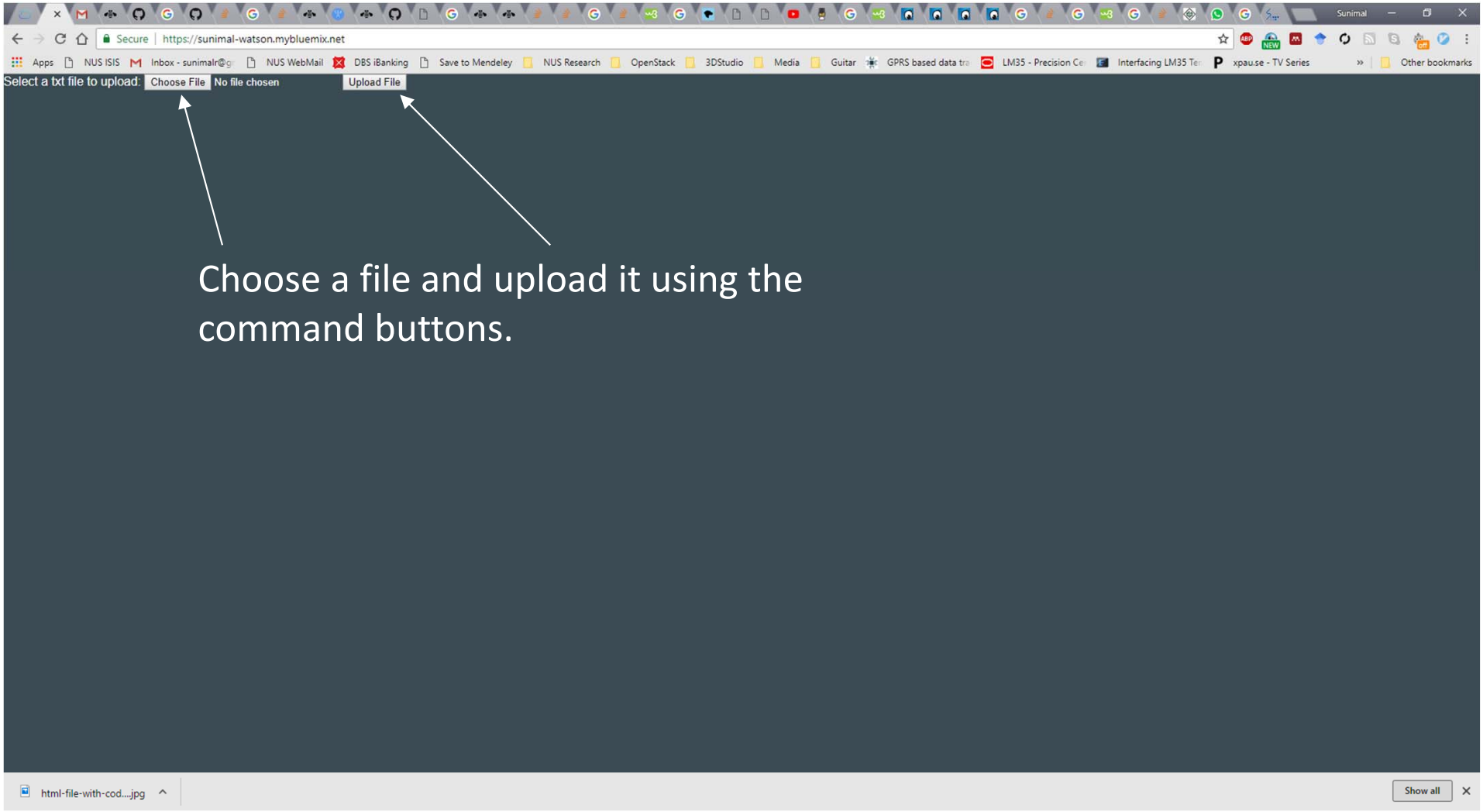
this is the maximum allowed RAM for the app. When you create the app in web console (in step 2 and 3), you may b e asked to select memory size. Use that number here.

use the hostname and app name you selected

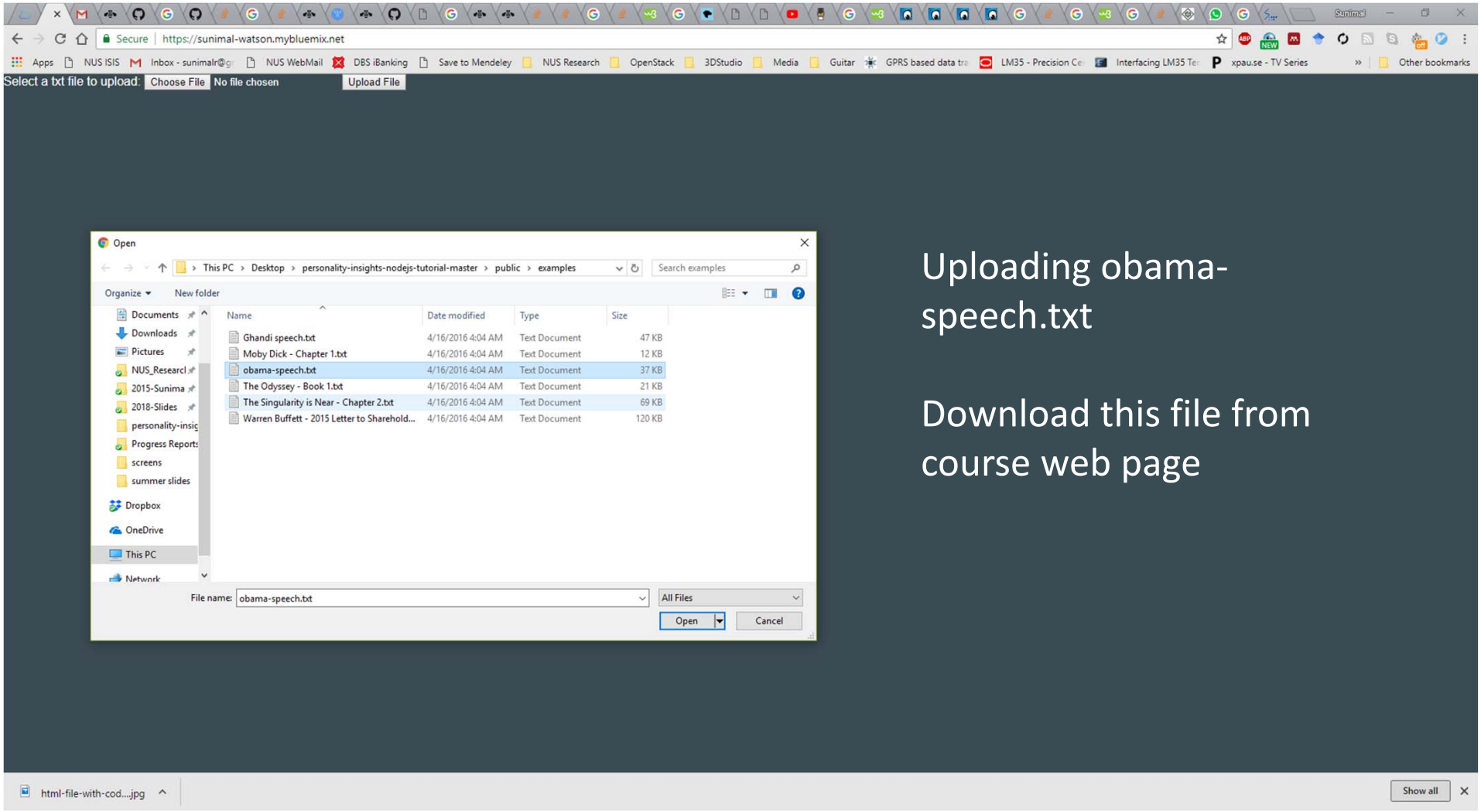
Step 7-9: Bluemix CLI Steps

- Open command line and change directory into the folder with code downloaded in the previous step
- Use the Bluemix CLI to connect to the Bluemix API (Depending on the region you are logged on to. 'ng' is for US South)
  - bluemix api https://api.ng.bluemix.net
- Login to Bluemix
  - bluemix login -u <bluemix_id> -o <organization> -s <space>
  - use the IBM ID and password
- Create the Personality Insights service in Bluemix
  - bluemix create-service personality_insights tiered personality-insights-tutorial
- upload your application to Bluemix
  - bluemix app push <appname>

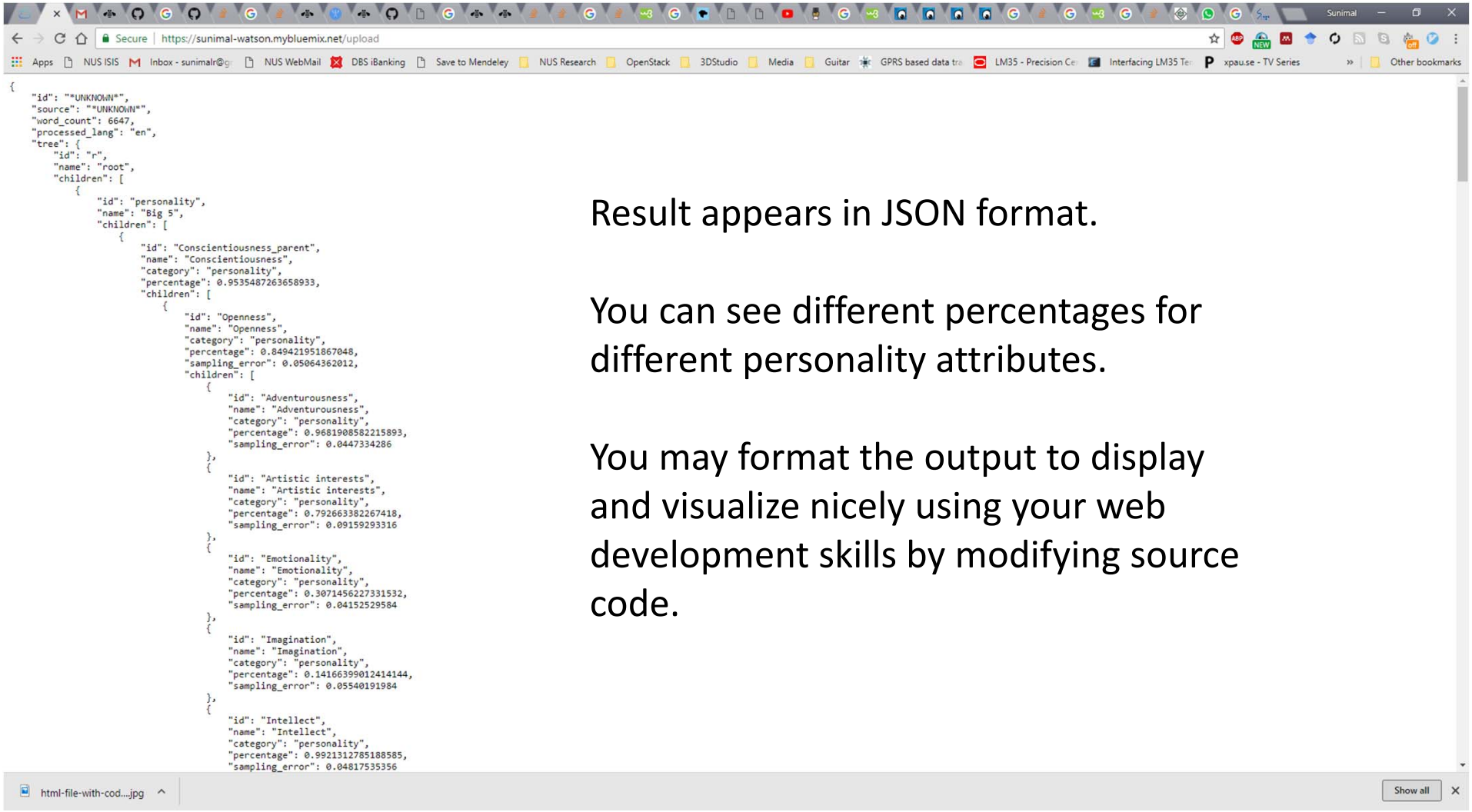
Step 10: View Result



Step 10: View Result



Step 10: View Result



Result appears in JSON format.

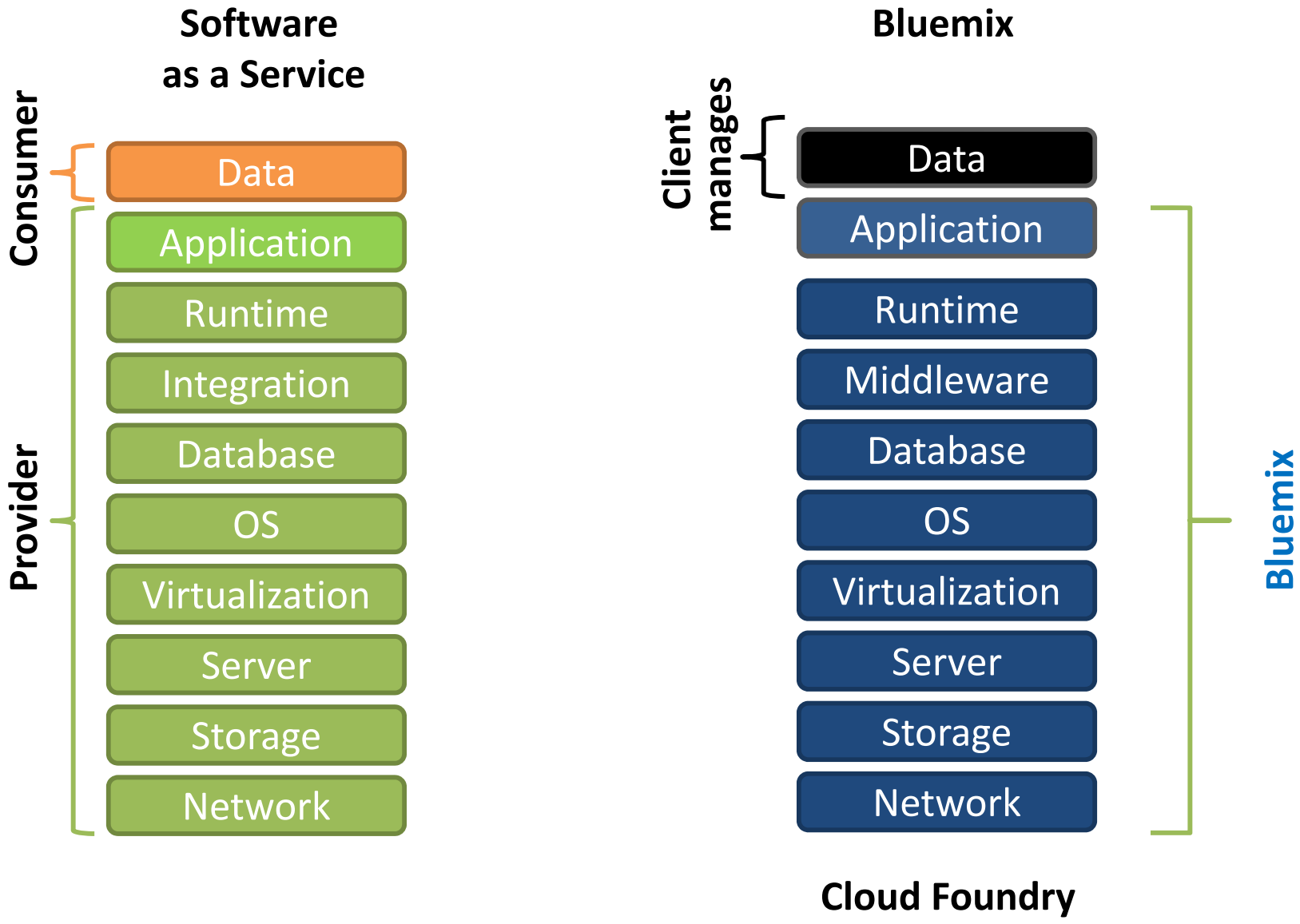
You can see different percentages for different personality attributes.

You may format the output to display and visualize nicely using your web development skills by modifying source code.

Milestone 4: Load Watson Web App

- Did you successfully load the web app?
- Let’s wait until everyone is done!
- In the meantime:
  - Learn more about available services under IBM Watson on the Bluemix console

What is SaaS?


Software
as a Service
Consumer
Provider
Data
Application
Runtime
Integration
Database
OS
Virtualization
Server
Storage
Network
Bluemix
Client
manages
Data
Application
Runtime
Middleware
Database
OS
Virtualization
Server
Storage
Network
Bluemix
Cloud Foundry

EXAMPLE 3 QUERYING WITH SQL SCRIPTING

- SQL is a querying language used for accessing and querying databases

- IBM Cloud offers in cloud SQL language support with Db2 (e.g. IBM Db2 Warehousing)

Objective: Storing and querying data with a database query language

Example: Load economic data of countries to a an IBM Db2 database, and, run simple SQL queries on-cloud on the stored dataset

Data: Economic data of countries from United Nations (stats.csv)

Example 3-Part 1: Querying with SQL Scripting

- SQL is a querying language used for accessing and querying databases

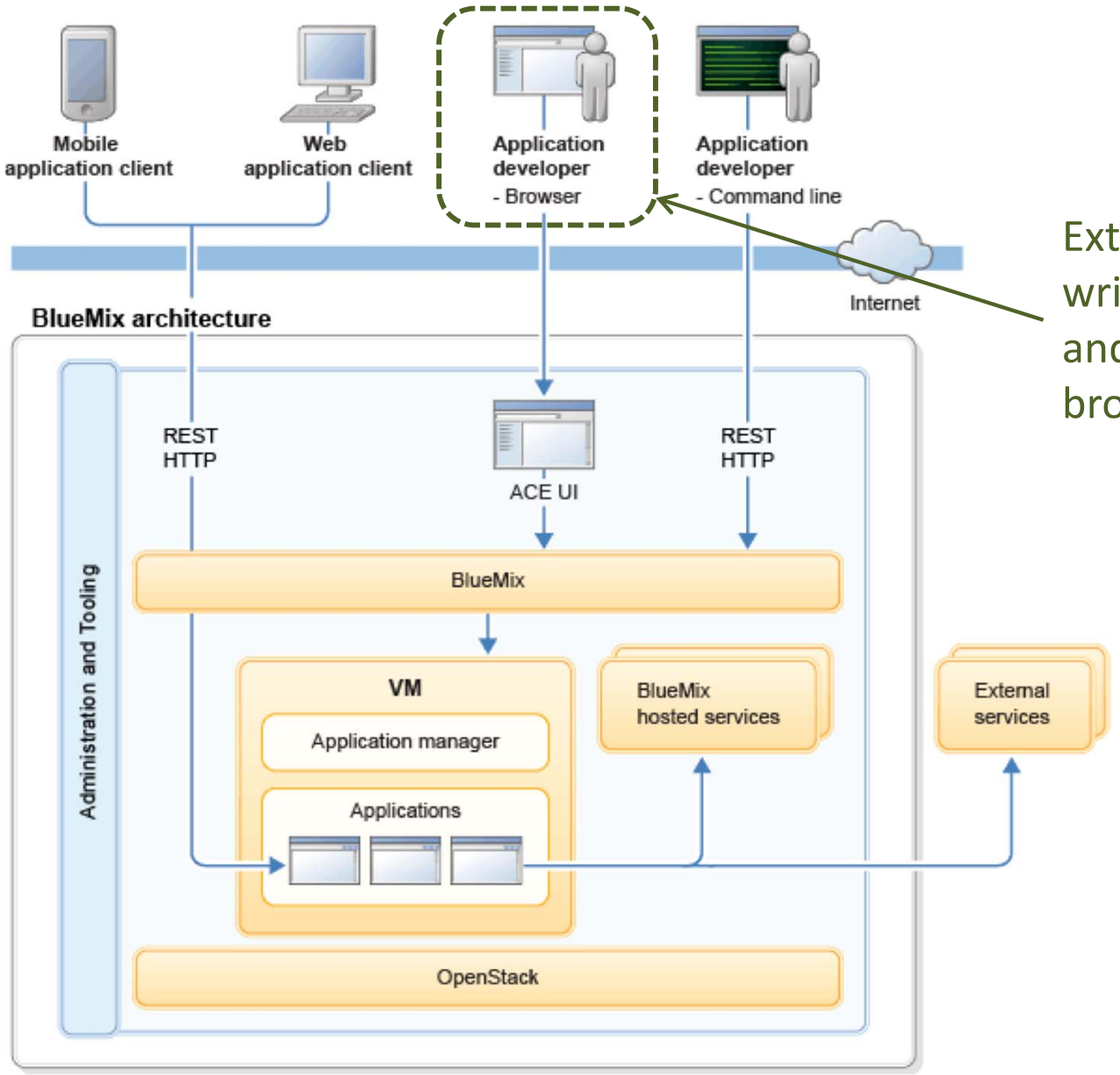
- IBM Cloud offers in cloud SQL language support with Db2 (e.g. IBM Db2 Warehousing)

Objective: Storing and querying data with a database query language

Example: Load economic data of countries to a an IBM Db2 database, and, run simple SQL queries on-cloud on the stored dataset

Data: Economic data of countries from United Nations (stats.csv)

SaaS: Analytics using DB2 Service



Extend SaaS to write SQL query and R code in web-browser

Main Steps: Analytics using DB2 Warehouse

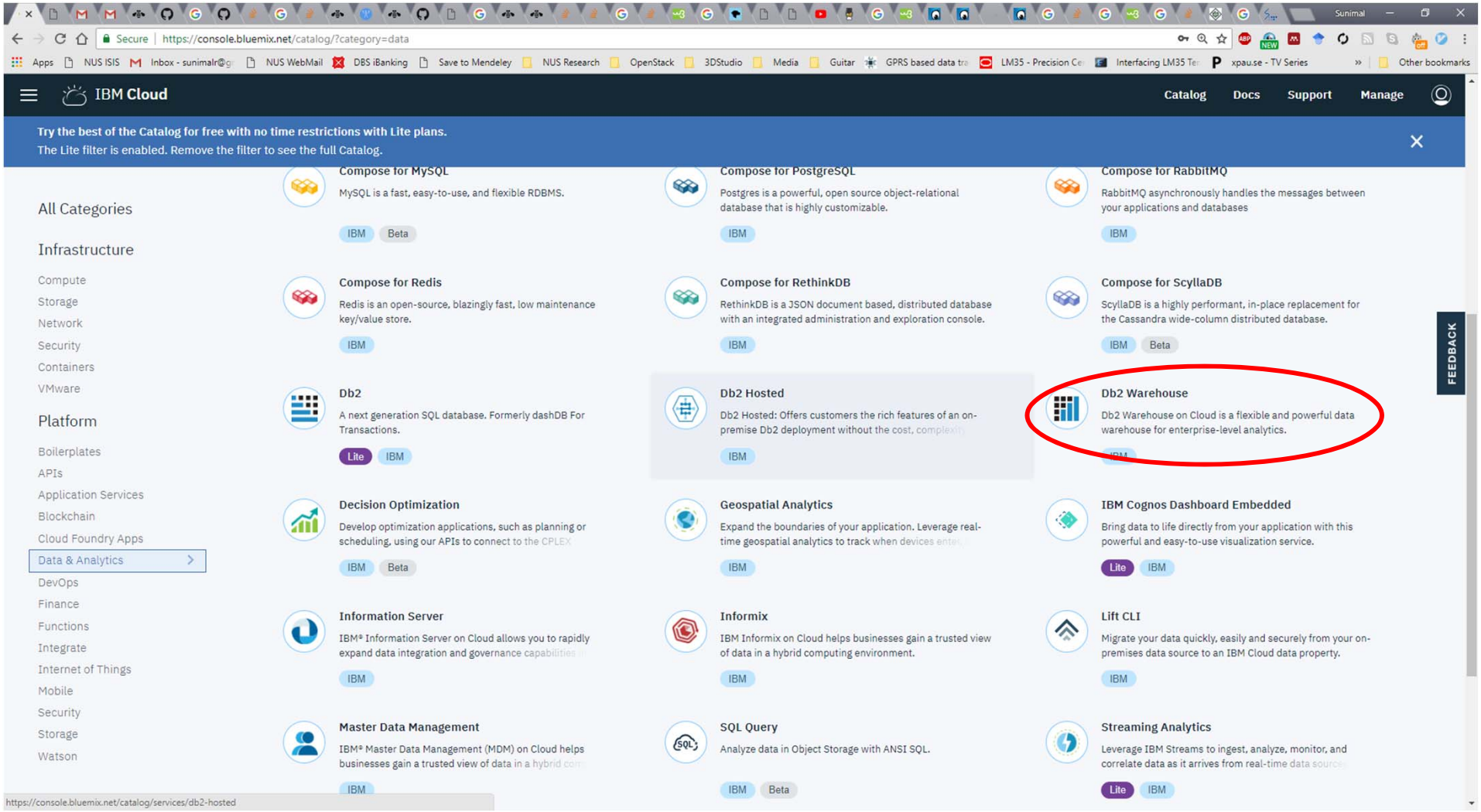
Example 2-Part1: SQL Scripting

1. Login to Bluemix and go to Catalog
2. Select DB2 Warehouse under Data & Analytics and setup
3. Open DB2 console
4. Upload data (CSV or Excel) from local computer
5. Review upload data on the cloud
6. Run SQL

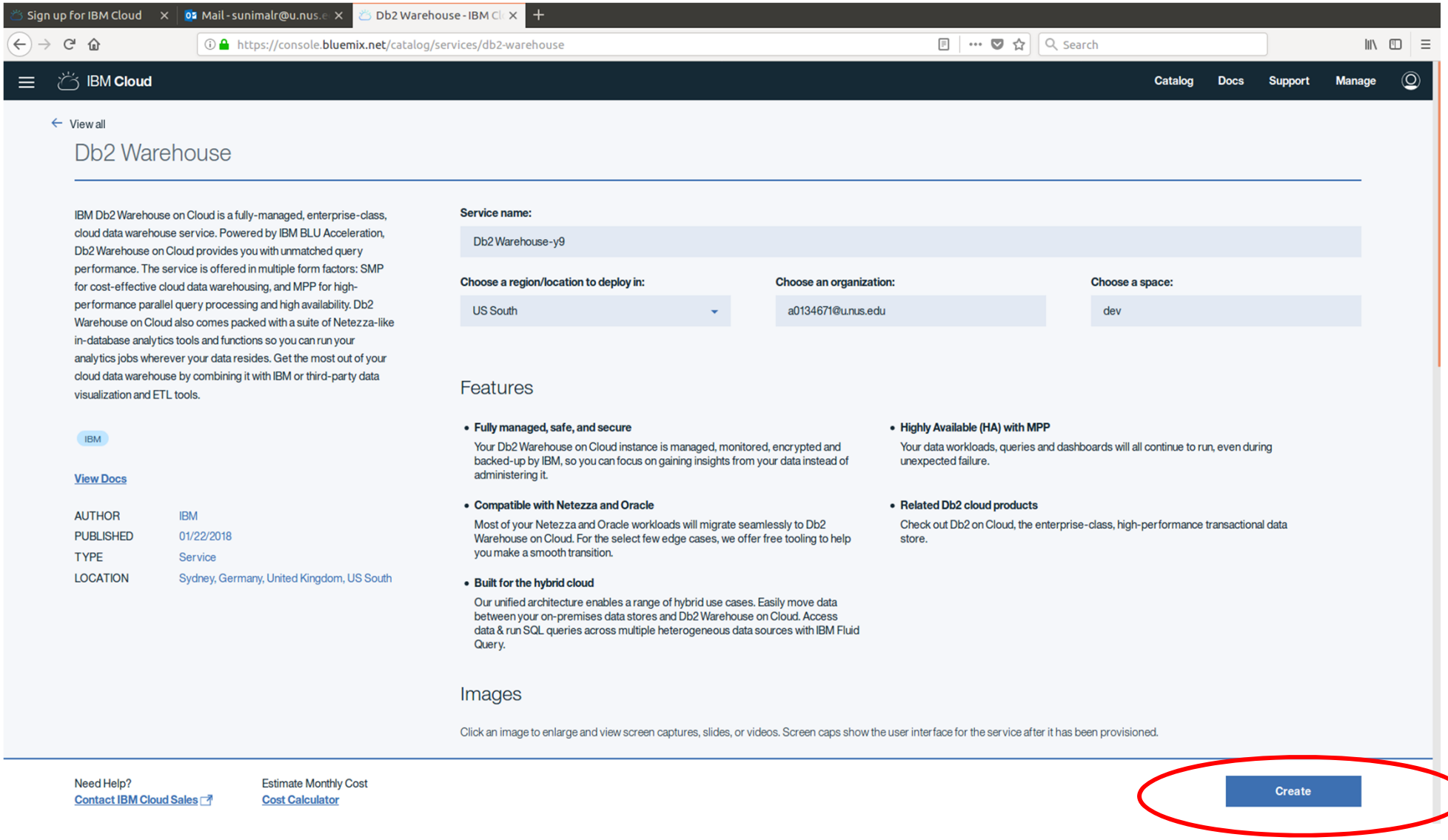
Example 2-Part2: R Scripting

7. Start an 'IBM Data Science Experience - DSX' account
8. Create and setup new DSX project
9. Connect DSX to Db2 database
10. Create a 'R' notebook
11. Insert R data frame
12. Write R script to analyze data
13. Output the plot

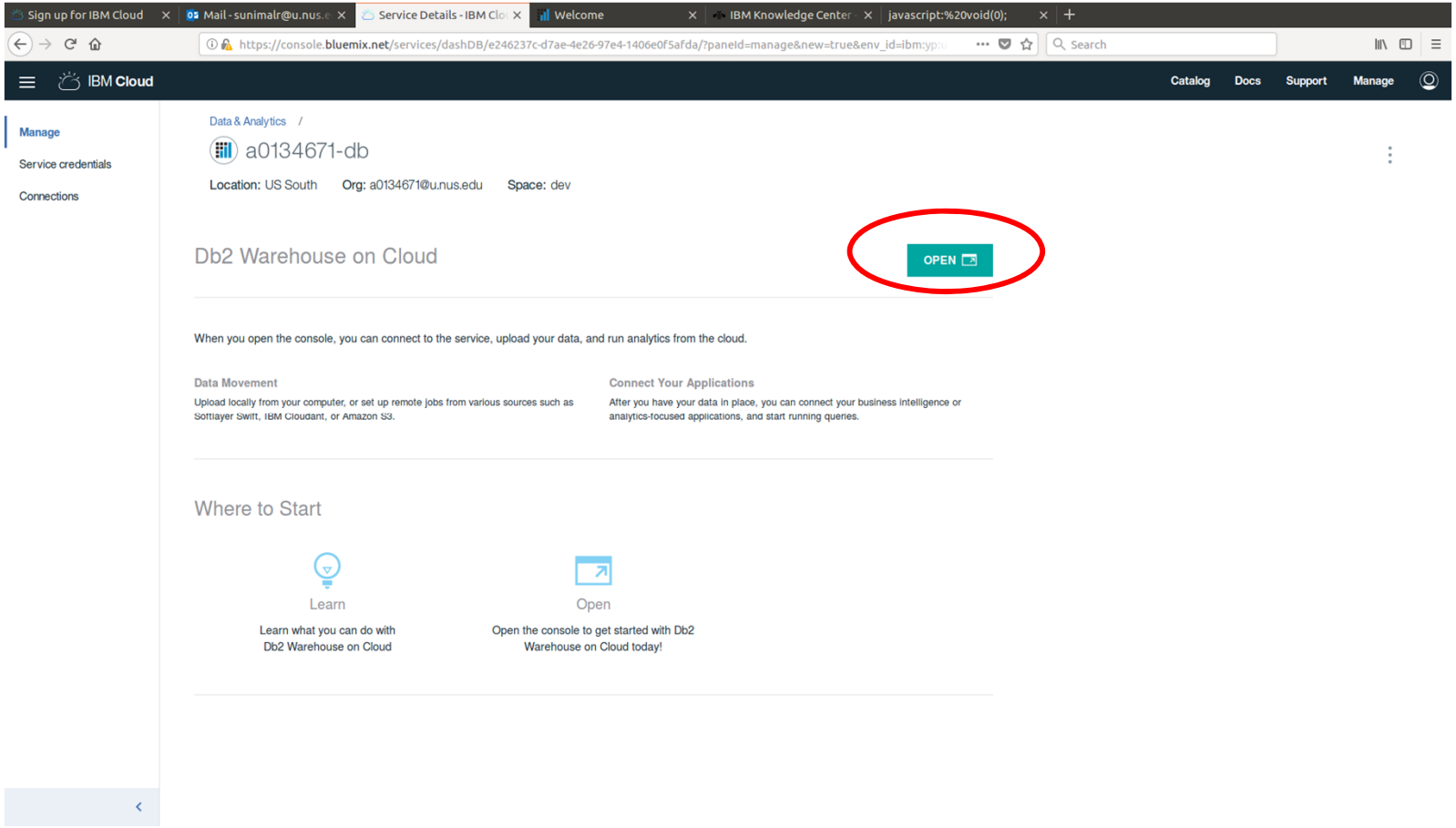
Step 1: Select Db2 Warehouse



Step 2: Db2 Warehouse Setup

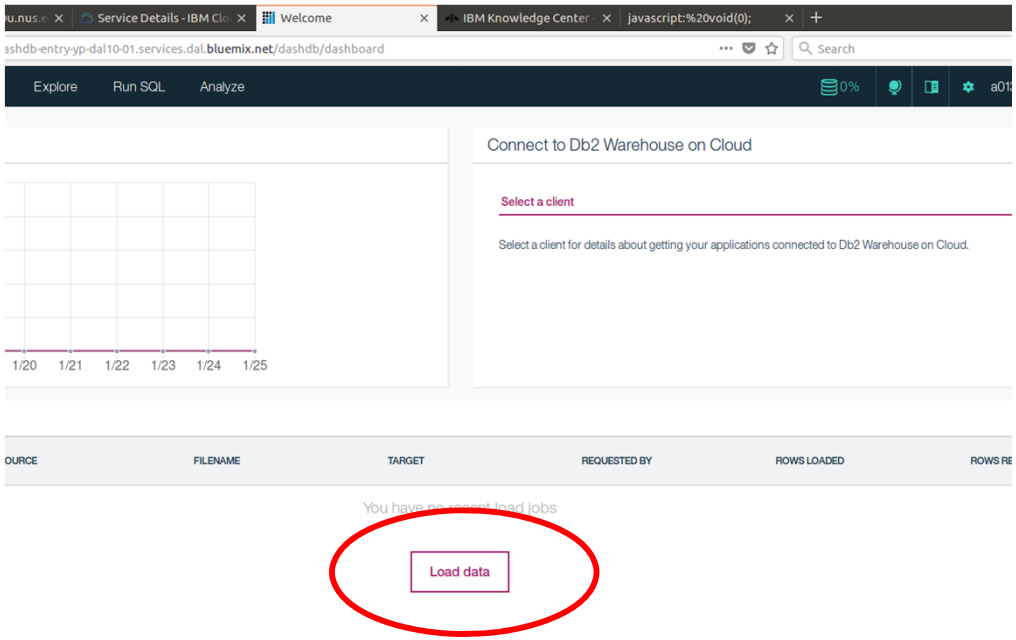


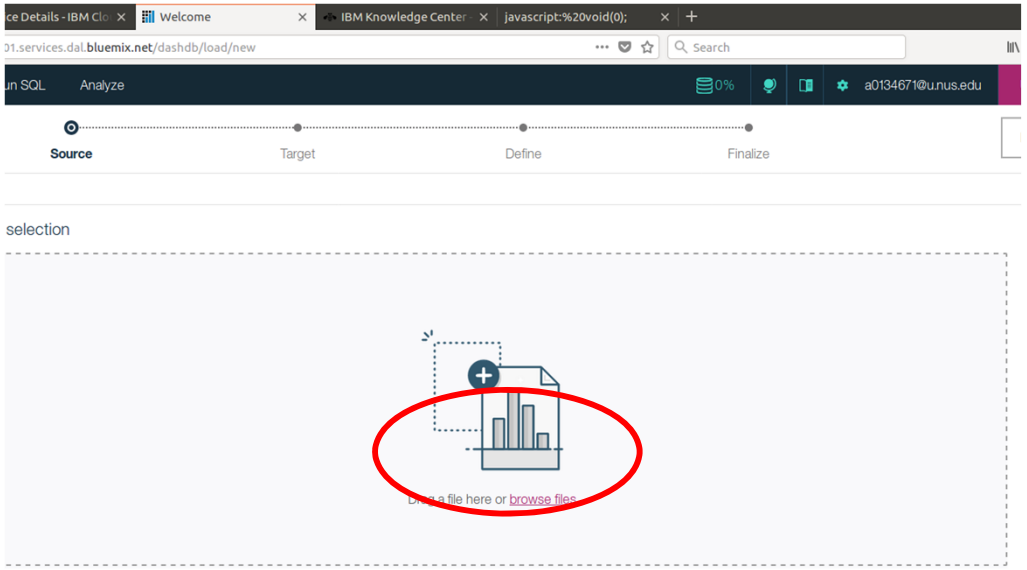
Step 3: Open Db2 Warehouse Console



Step 4: Upload data

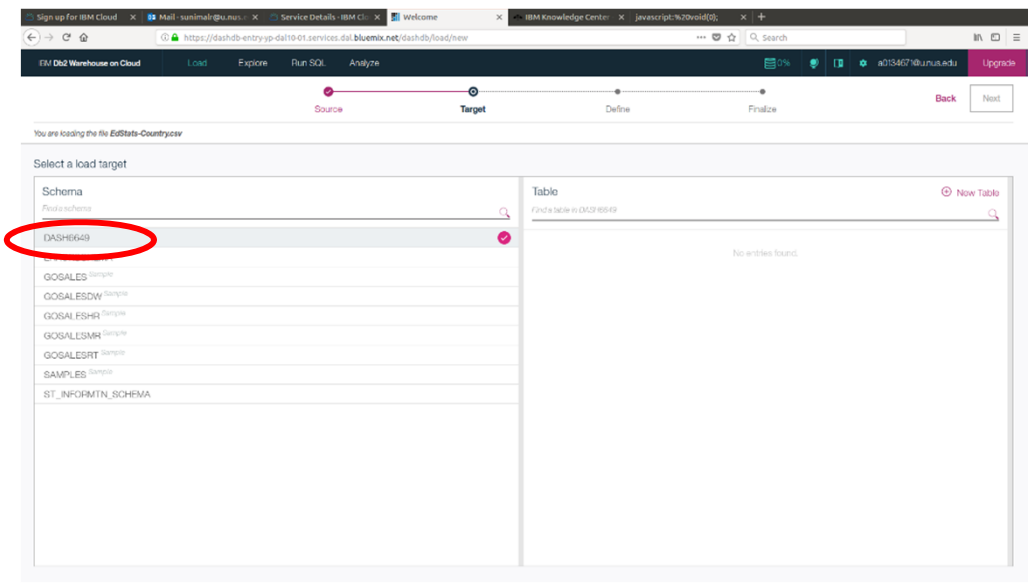
- First download stats.csv data file from the course webpage, then upload to Db2

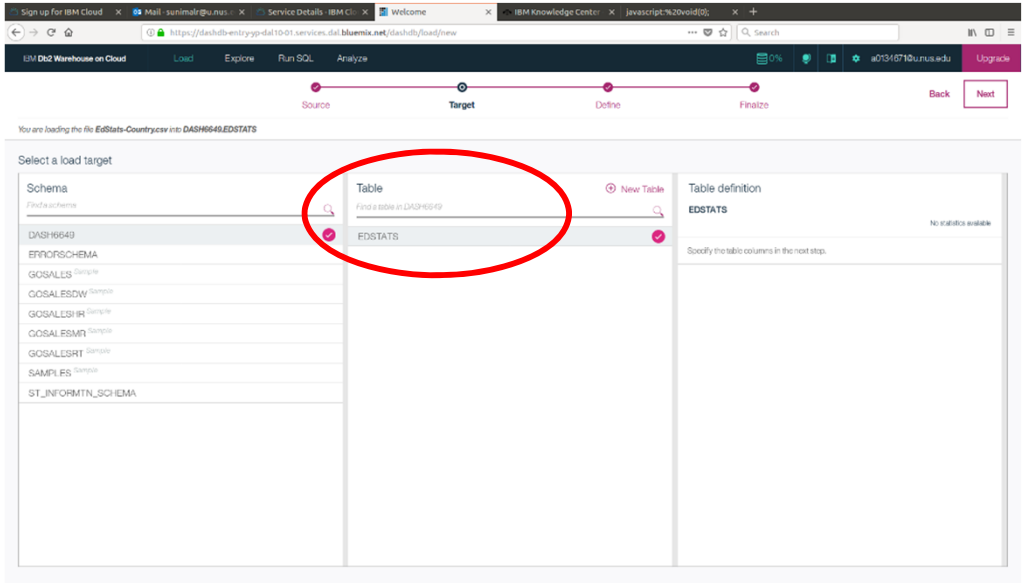




Step 5: Select Scheme, Table Name and, Review Table

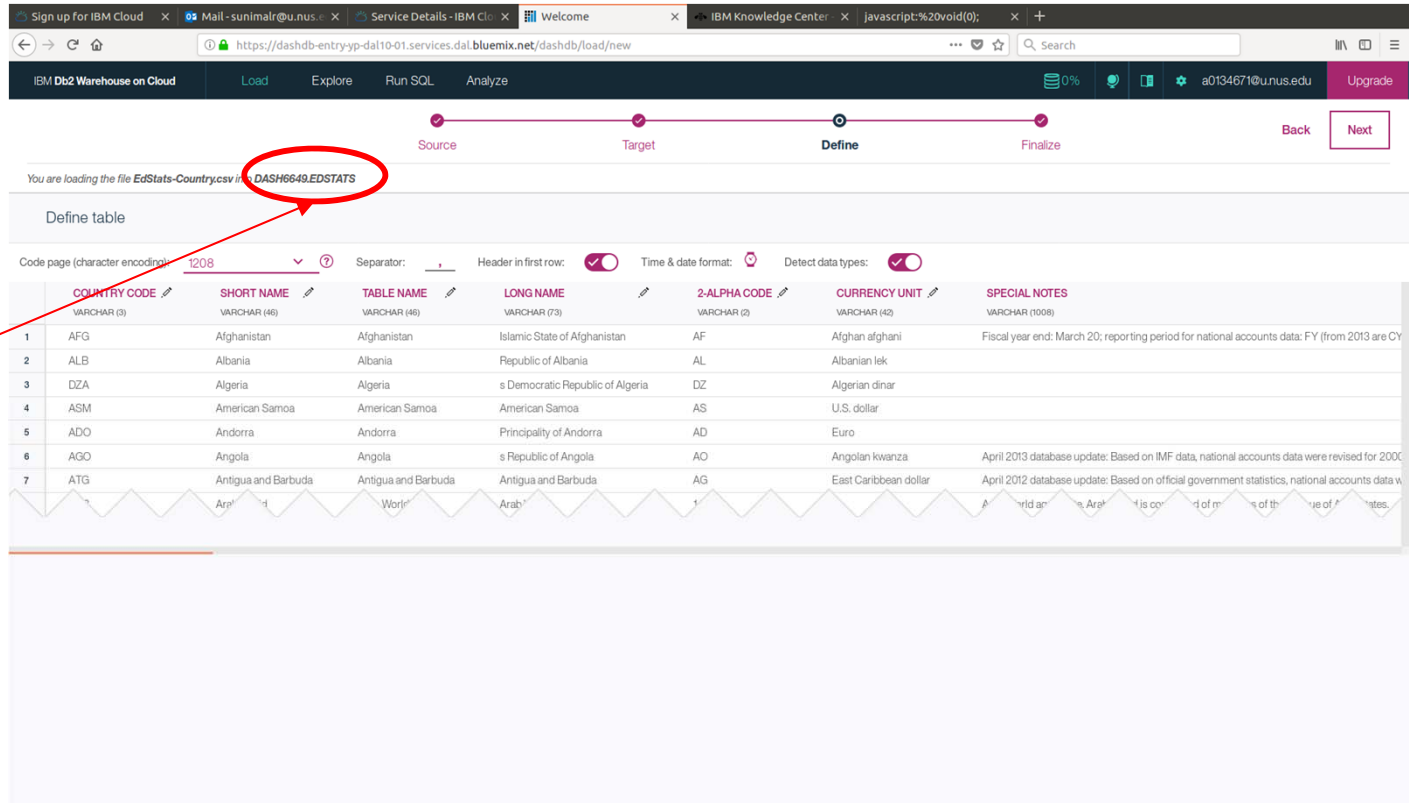
- Create a new table “EDSTATS”





Step 5: Select Scheme and Table Name

- Preview
  - You can change the column names if necessary
- Review settings
- Click Begin Load



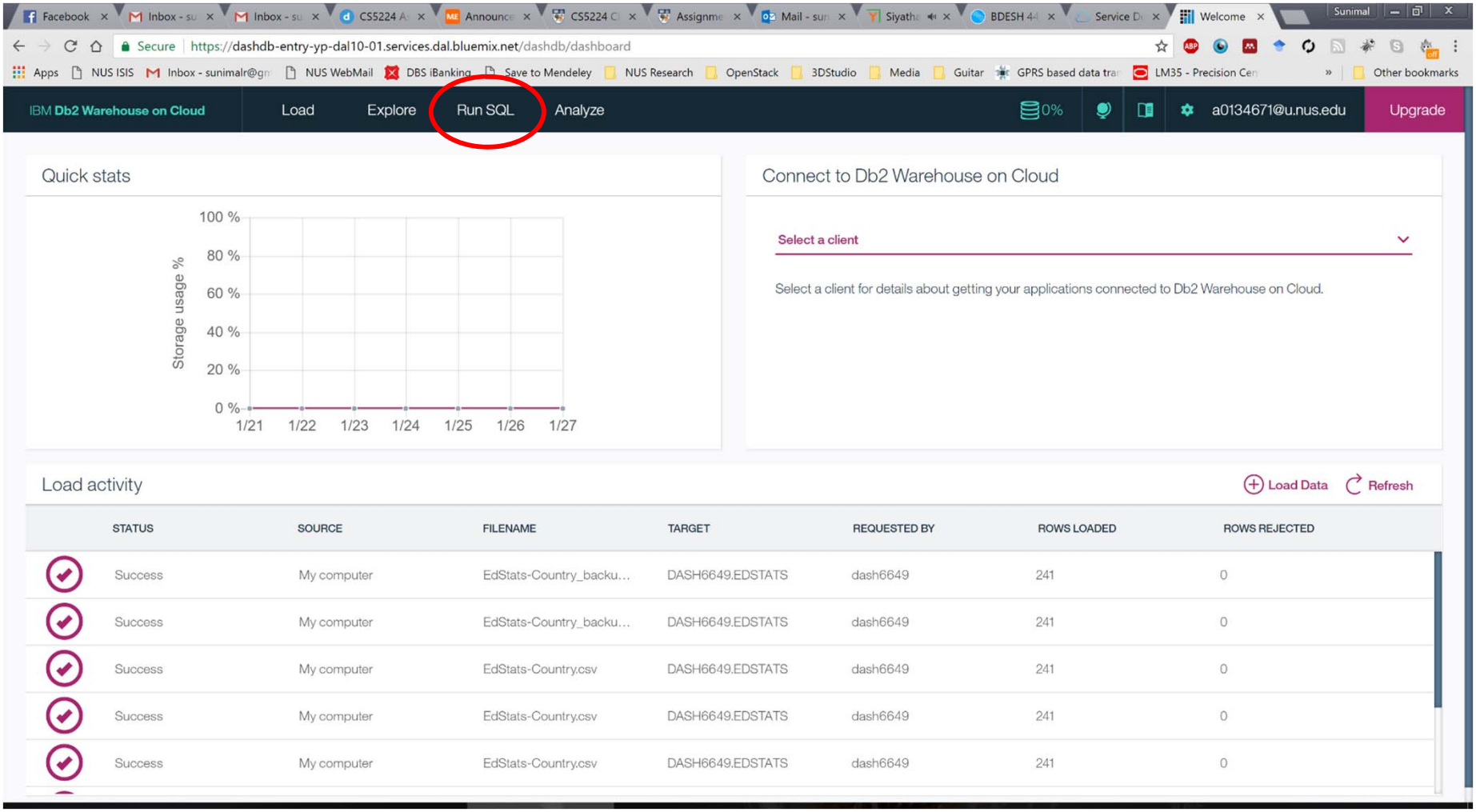
Note the table name.
We need this in next steps

Step 6: Run SQL Query

- Few things to note
  - SQL syntaxes slightly change across implementations (mysql, mssql, db2)
  - While loading file (step 5), the field names are shown in ALL CAPS, but this is in accurate
  - To see correct column names, go to ‘Explore’ tab

Step 6: Run SQL Query

- Click “Run SQL” to enter the SQL console



Step 6: Run SQL Query

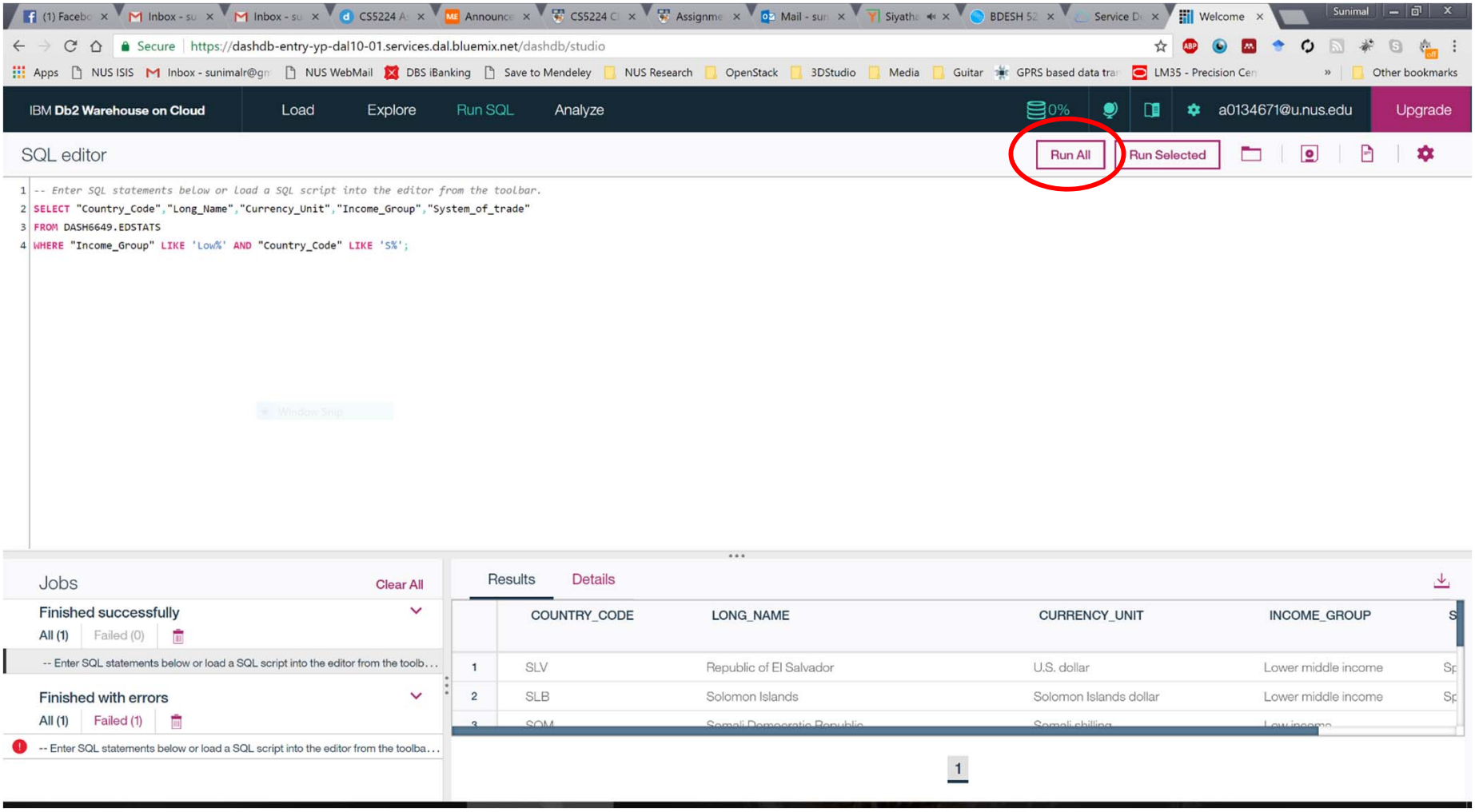
- Find out countries that fall under the low income group and has their country code starting with S

```
SELECT
"Country_Code","Long_Name","Currency_Unit","Income_Group","System_of
_trade"
FROM <INSERT Database table name here eg. DASH6649.EDSTATS>
WHERE "Income_Group" LIKE 'Low%' AND "Country_Code" LIKE 'S%';
```

Database name should be set as required by referring to the database tables.

Step 6: Run SQL Query

- Click “Run All” and get the result



Milestone 5: Run SQL Query

- Did you successfully run the SQL?
- Let's wait until everyone is done!
- In the meantime:
  - Explore more about Db2 at: https://www.youtube.com/watch?v=zm4-VZolYV0

Example 3-Part 2: Analytics with R scripting

- R is a scripting language used for statistical computing including big data analysis, and data mining

- IBM Cloud offers R language support with IBM Data Science Experience service also known as DSX service

- Data loaded in an existing on-cloud database can be analyzed in a DSX project after properly linking the database to the DSX project

Objective: Perform simple analytics and visualization of data on cloud

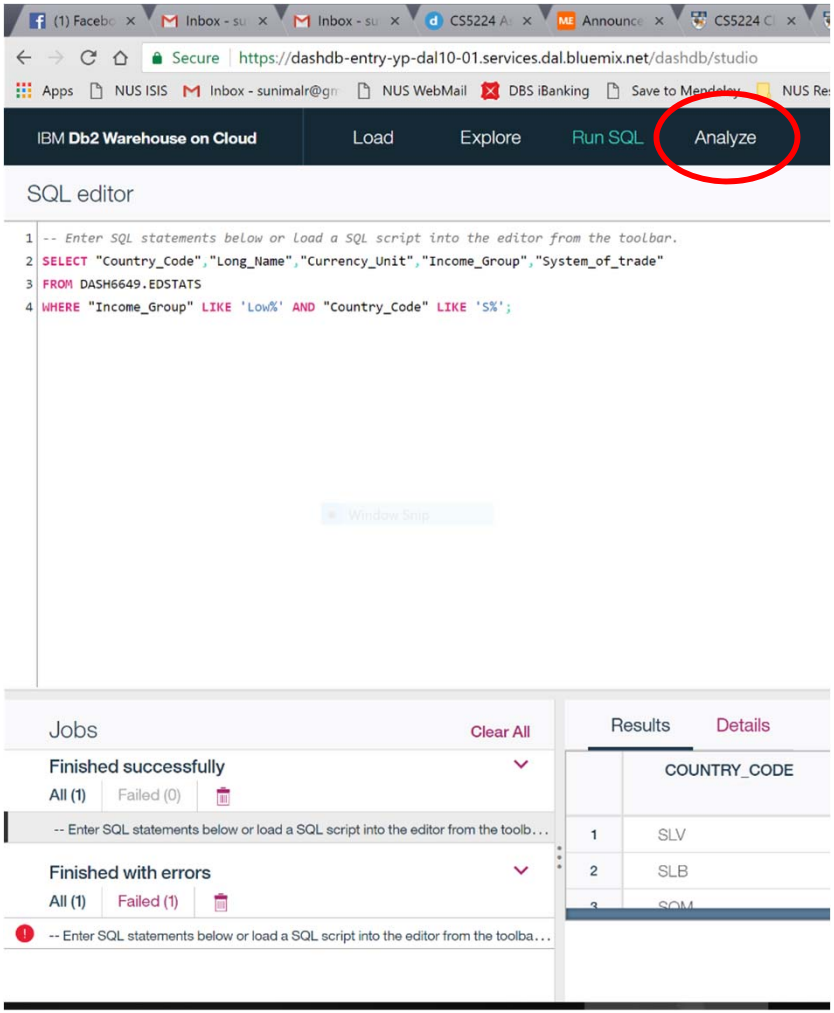
Example: Load economic data of countries to a an IBM Db2 database, and, execute simple R scripts for filtering data and visualization with plots

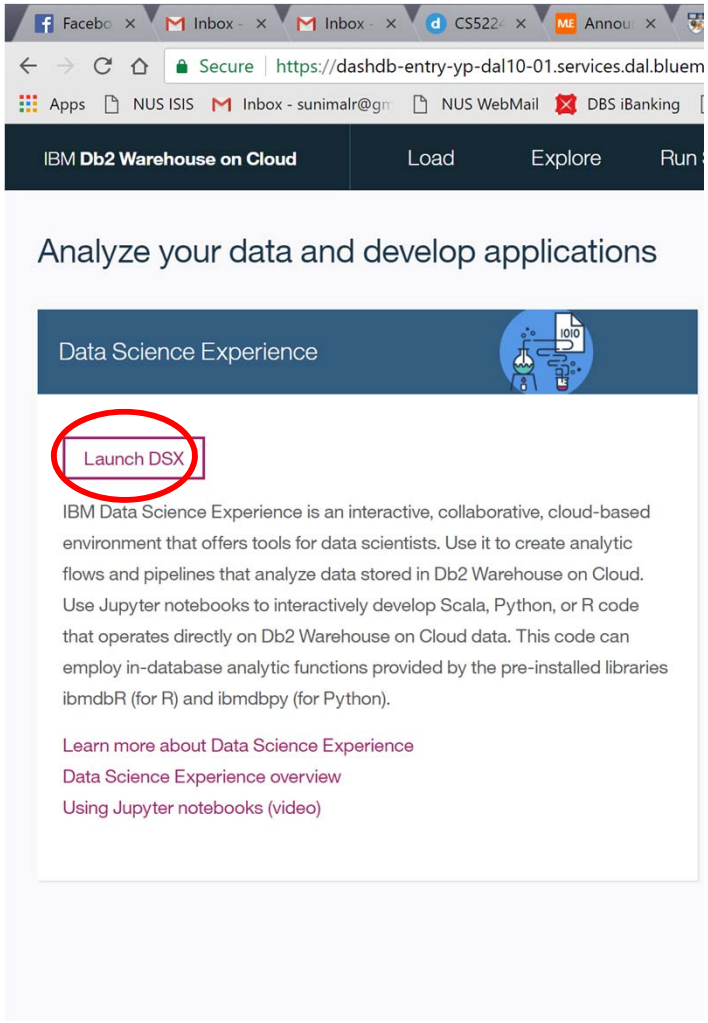
Data: Economic data of countries from United Nations (stats.csv)

This is a continuation of Example 3-Part 1

Step 7: Setup DSX

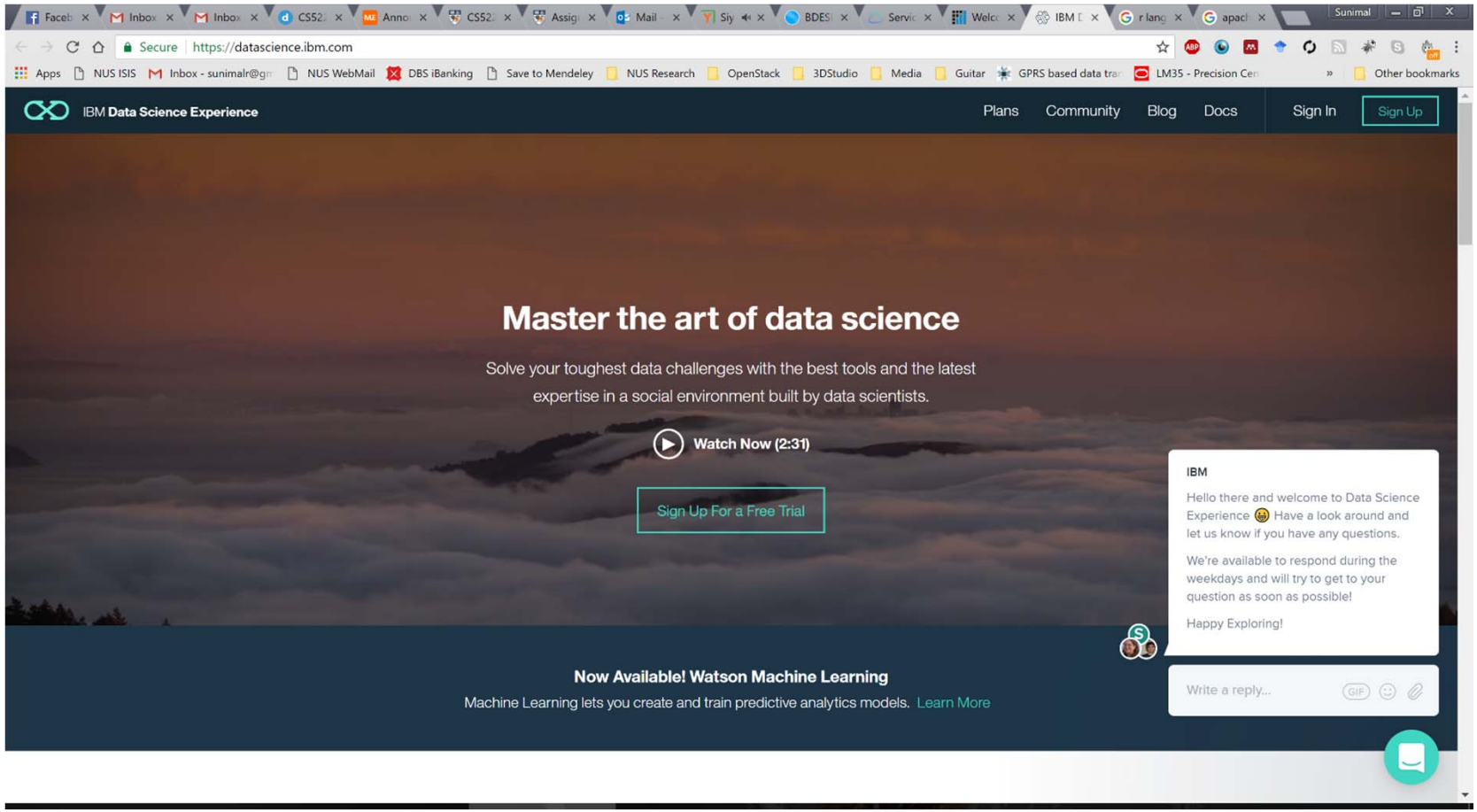
- Click on “Analyze” in the Db2 console
- Click “Launch DSX”





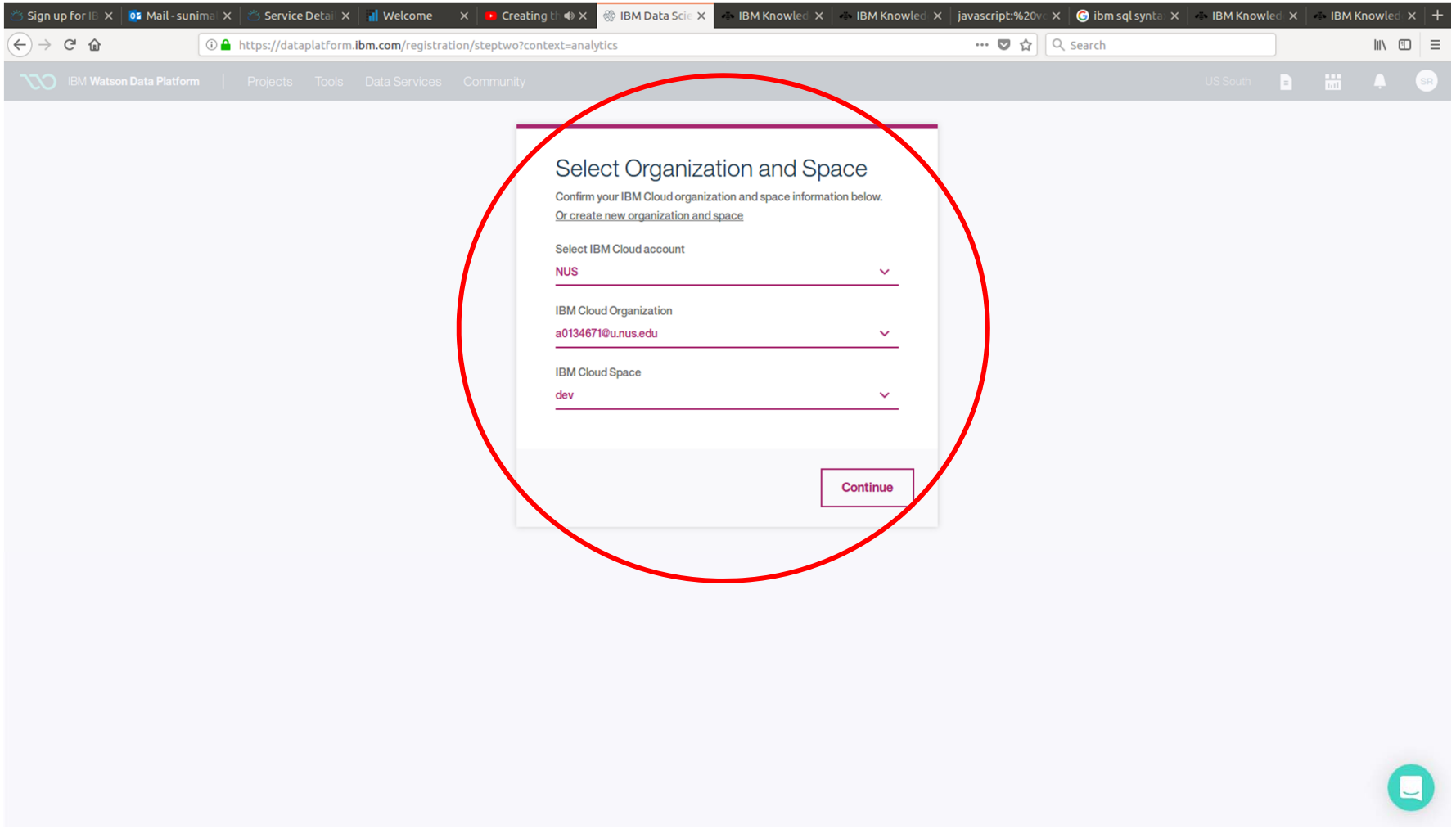
Step 7: Setup DSX

- Log in to DSX (You may be asked to sign up the first time)



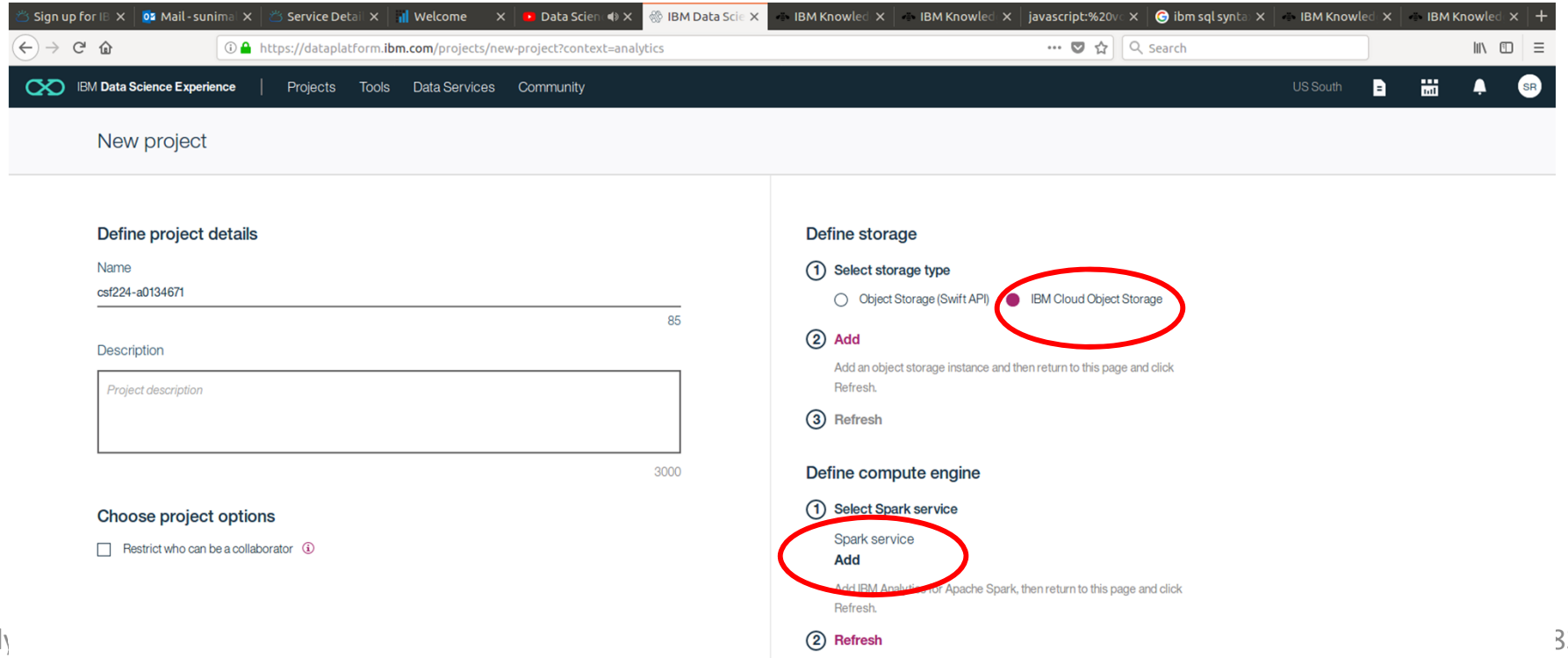
Step 7: Setup DSX

- Select the relevant account details from drop down menus



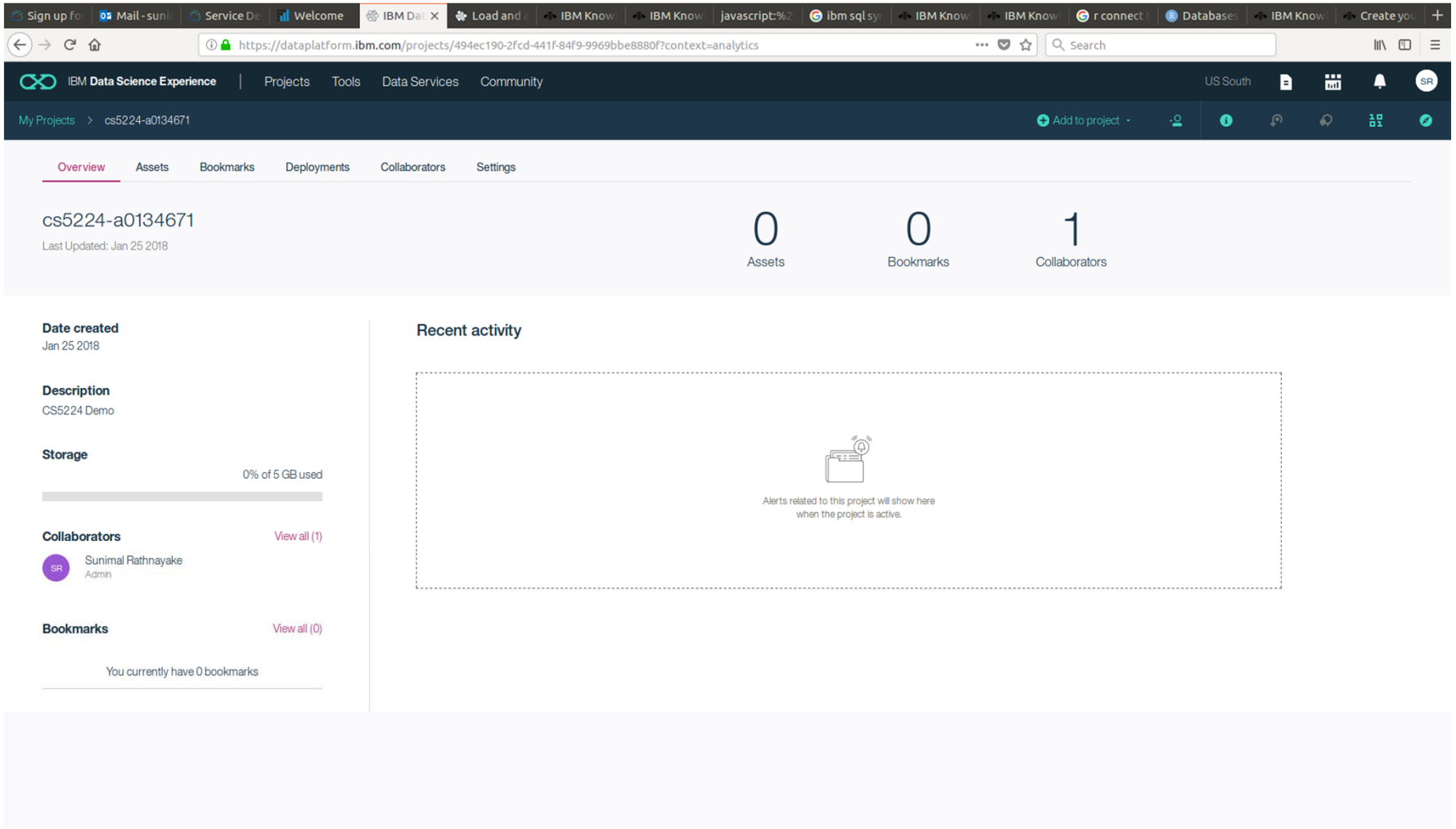
Step 8: Create new DSX project

- Give a unique name for the DSX project
- Select storage type – IBM Cloud Storage
- Add a “Spark” service as the compute engine – *Spark is a fast and general cluster computing system for Big Data. It provides high-level APIs in Scala, Java, Python, and R, and an optimized engine that supports general computation graphs for data analysis.*



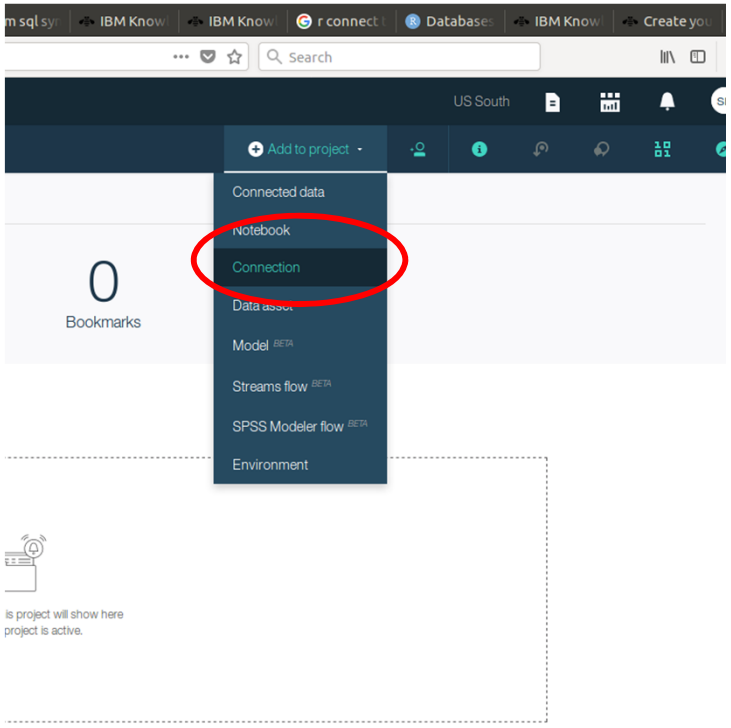
Step 8: Create new DSX project

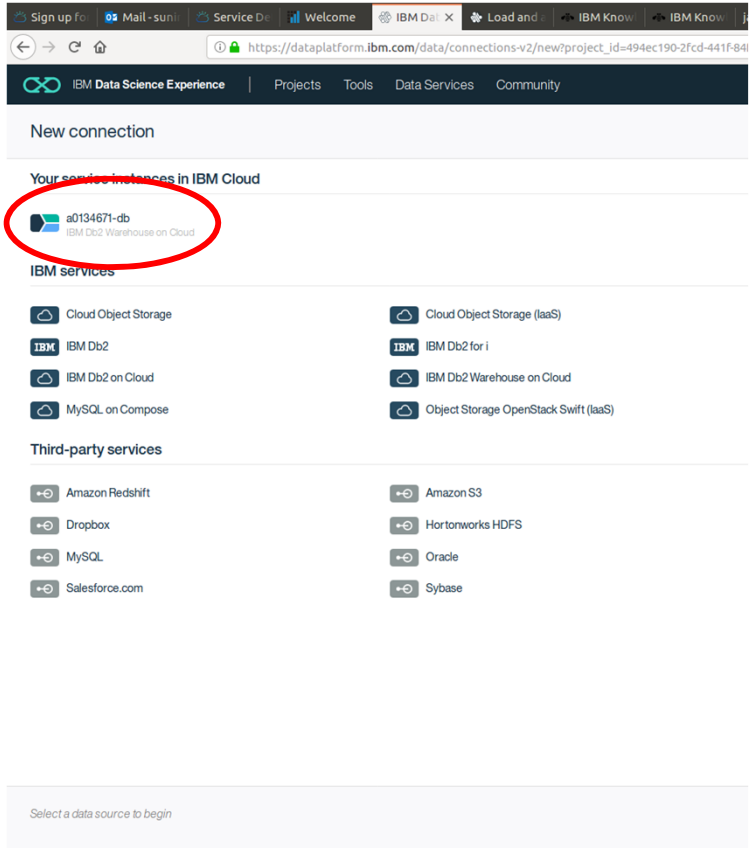
- Once the project is setup successfully, you should arrive at this portal



Step 9: Connect to Db2 database

- We need to connect DSX project to our previously created Db2 database

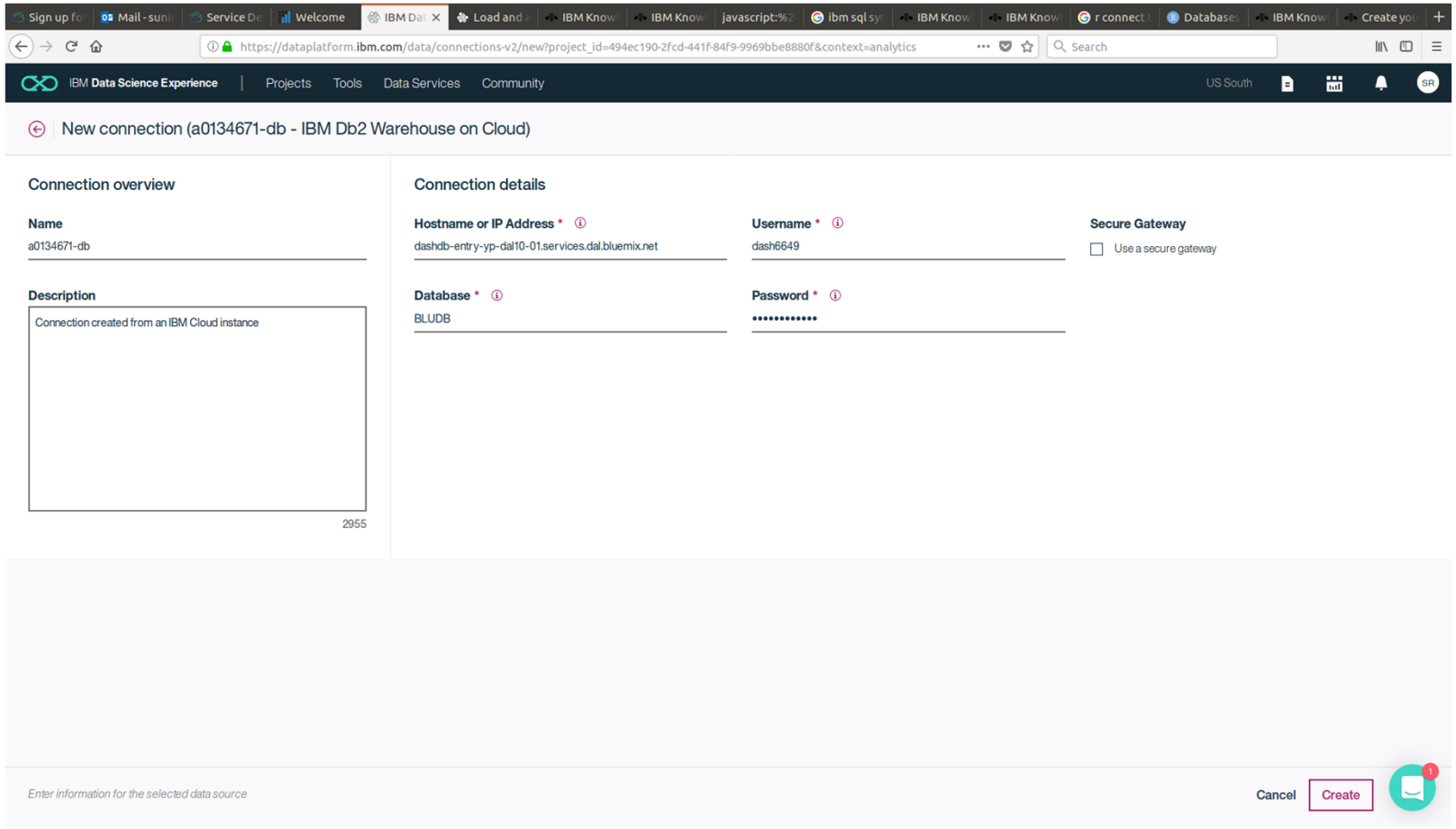




If there are multiple databases in your IBM Cloud account, select the correct one that we created with EDSTATS csv file.

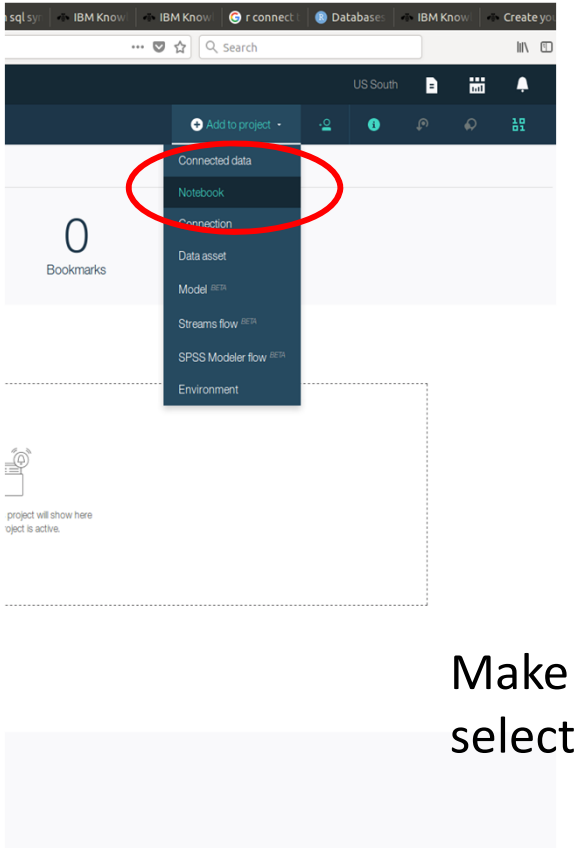
Step 9: Connect to Db2 database

- Give a name to data connection. Other fields should have been filled automatically.

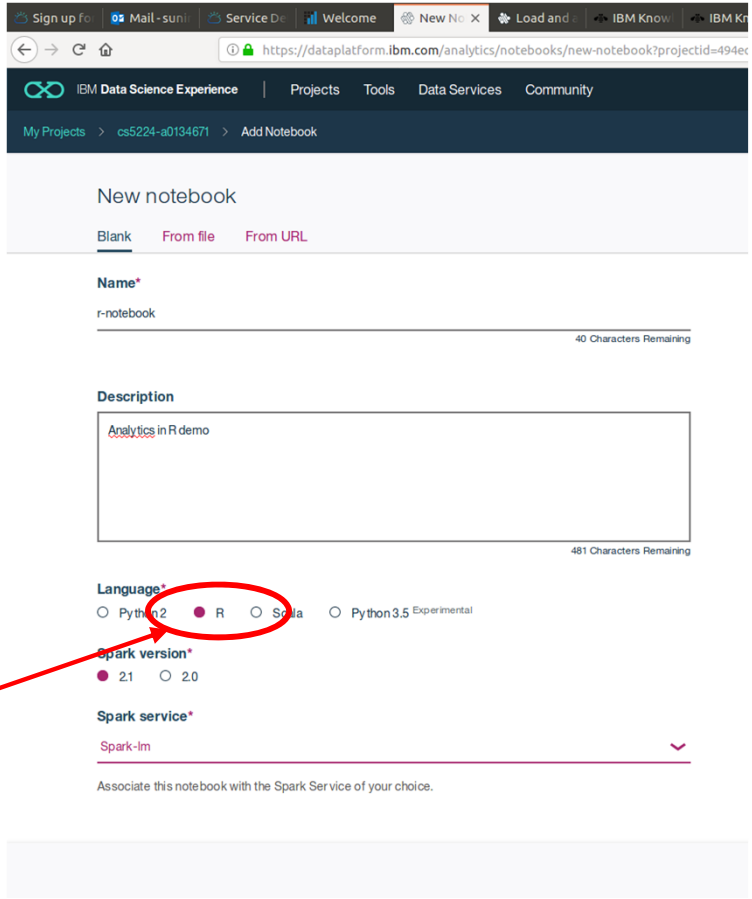


Step 10: Create a new R Notebook

- We need to connect DSX project to our previously created Db2 database



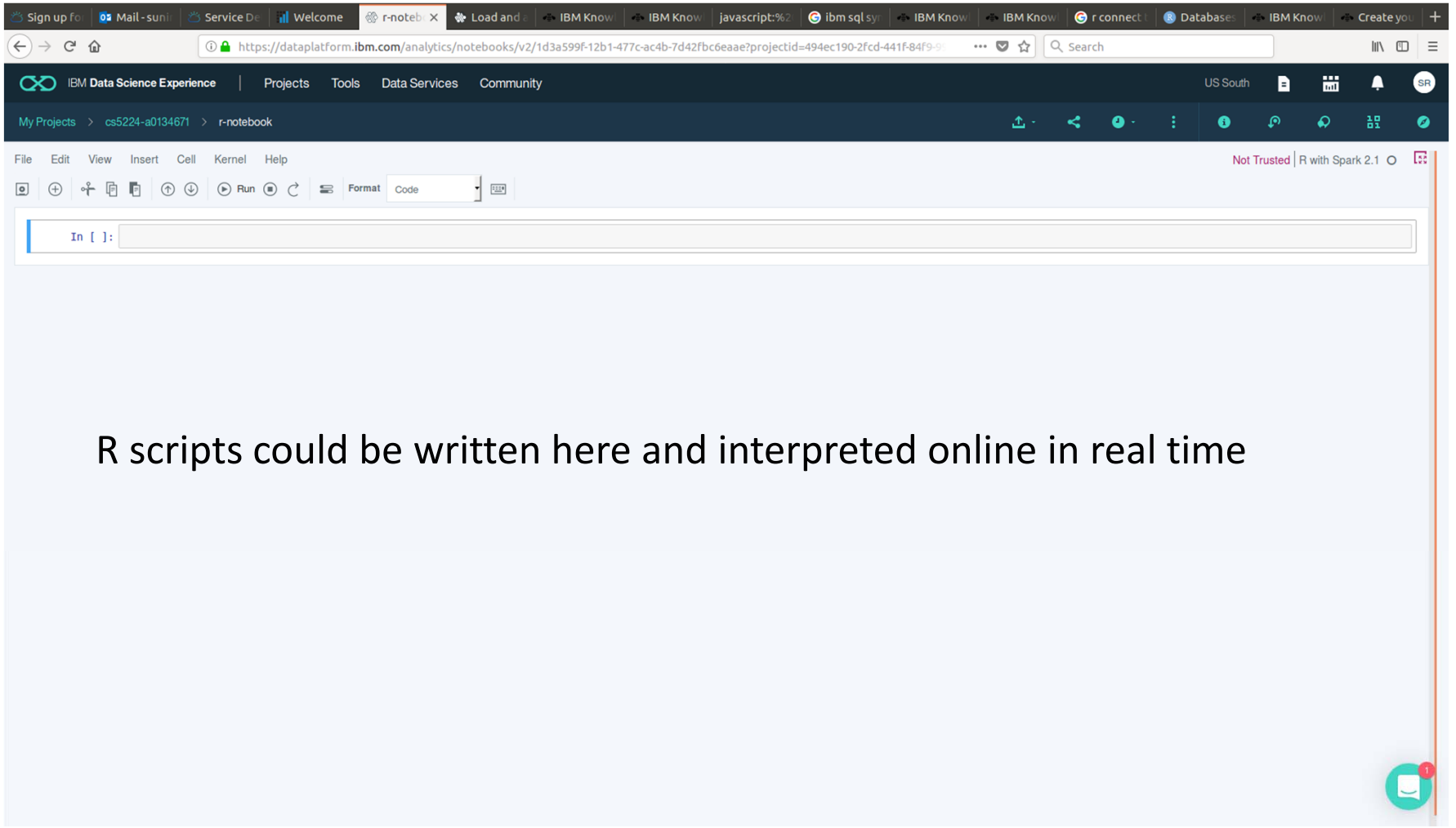
Make sure that you select R



Notebook is an online interpreter that allows scripting in different languages like R, Python etc.

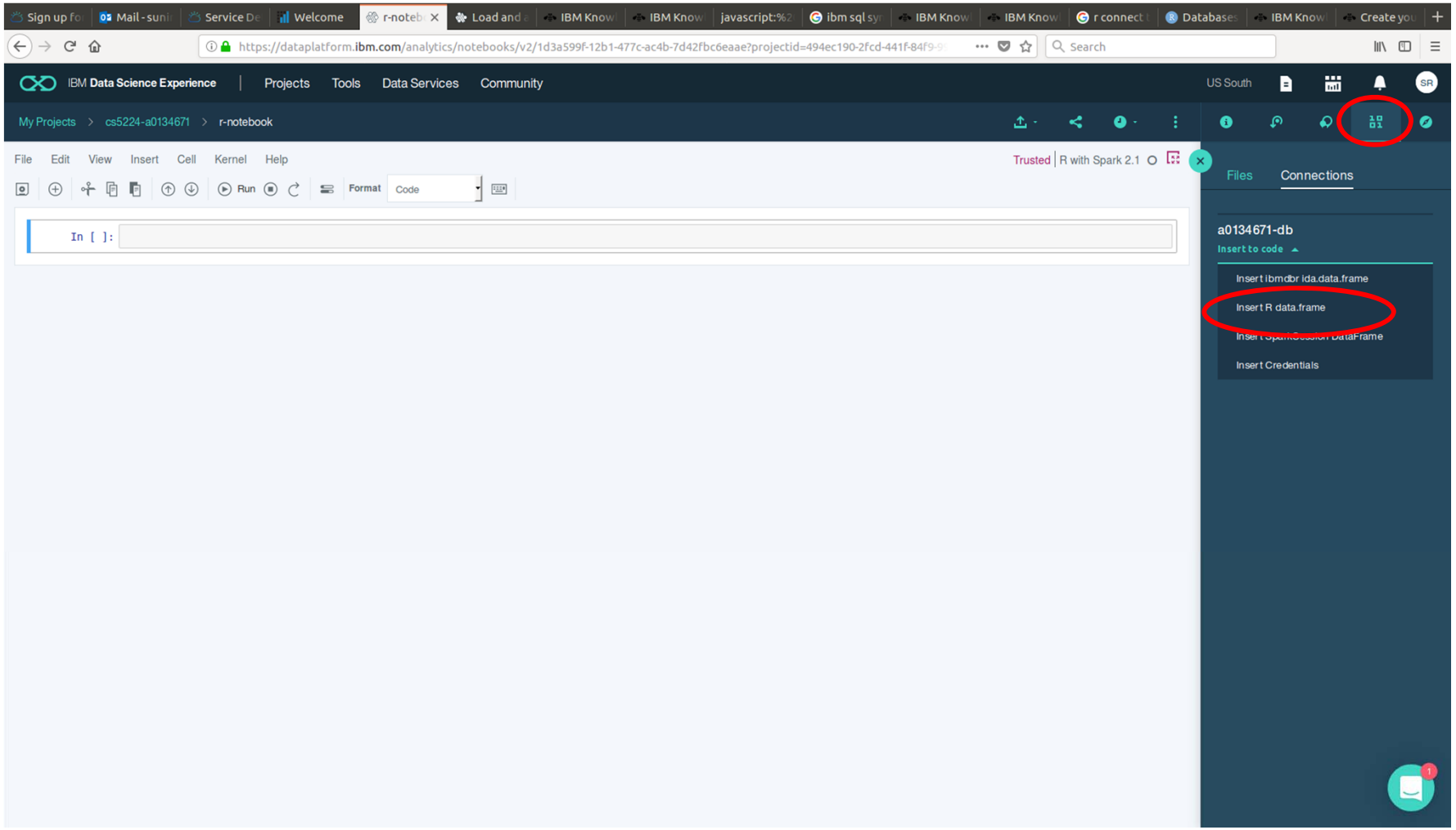
Step 10: Create a new R Notebook

- Created R notebook should look like this



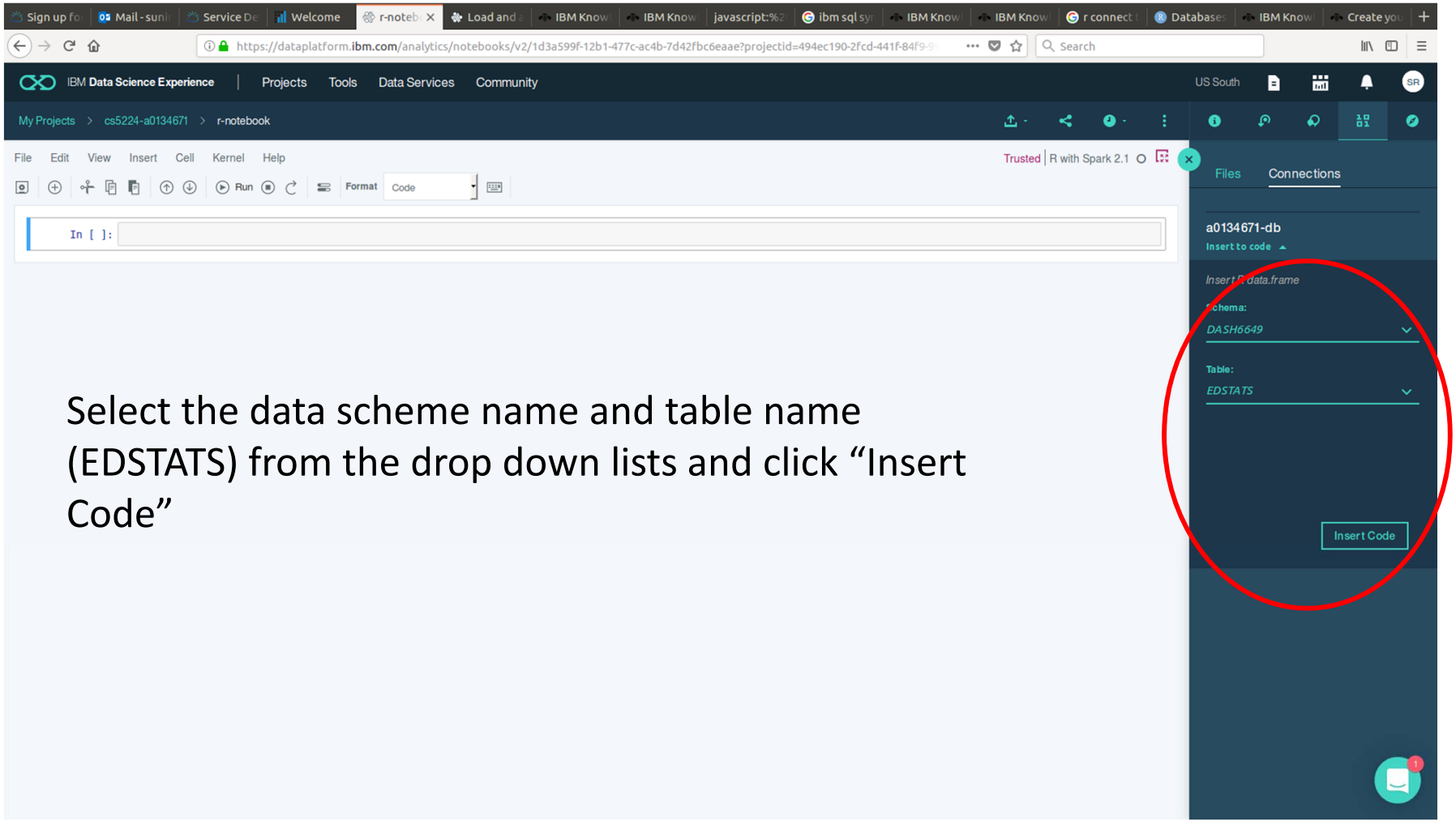
Step 11: Insert R data frame

- Insert a R data frame from the data connection.



Step 11: Insert R data frame

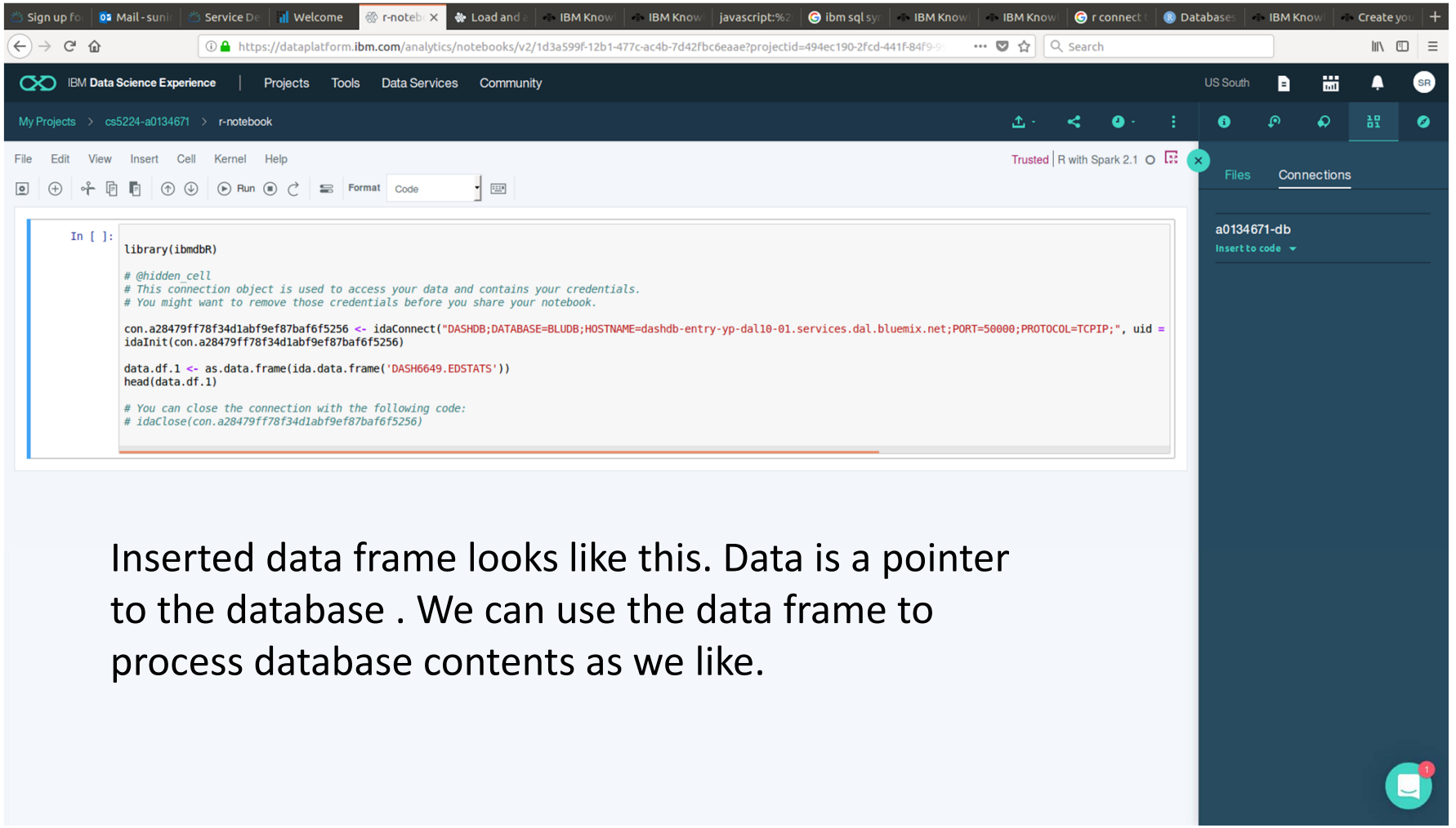
- Insert a R data frame from the data connection.



Select the data scheme name and table name (EDSTATS) from the drop down lists and click “Insert Code”

Step 11: Insert R data frame

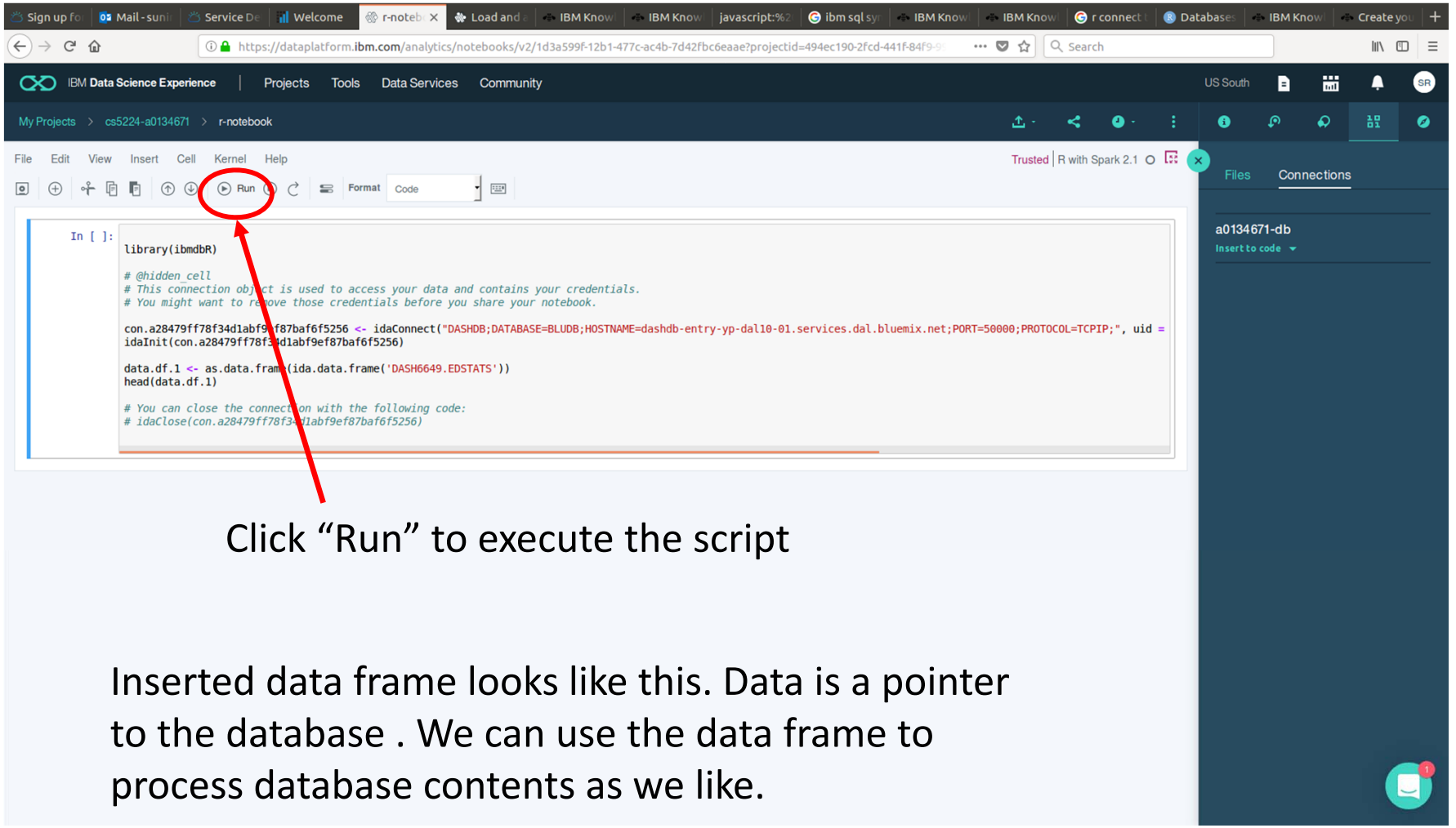
- Insert a R data frame from the data connection.



Inserted data frame looks like this. Data is a pointer to the database . We can use the data frame to process database contents as we like.

Step 11: Insert R data frame

- Insert a R data frame from the data connection.



Click “Run” to execute the script

Inserted data frame looks like this. Data is a pointer to the database . We can use the data frame to process database contents as we like.

Step 12: Write R script to Plot Data

- # Load the plotting package
  - library(ggplot2)

- # Plot the count of countries across different income groups
  - #Classify the levels and label them: Create a factor

```
data_frame1<- as.data.frame(ida.data.frame('DASH6649.EDSTATS'))[
,c('Currency_Unit', 'Income_Group', 'Region', 'Short_Name')]

data_frame1$"Income_Group" <-
factor(data_frame1$"Income_Group",levels=c("Low income","Lower
middle income","Upper middle income","High income:
nonOECD","High income: OECD"),
labels=c("LI","LMI","UMI","HIN","HIO"))
```

  - #Count the income groups

```
counts <- table(data_frame1$"Income_Group")
```

  - #Plot counts versus the labelled income groups

```
barplot(counts, main="Income Distribution",xlab="Income Group")
```

Step 12: Write R script to Plot Data

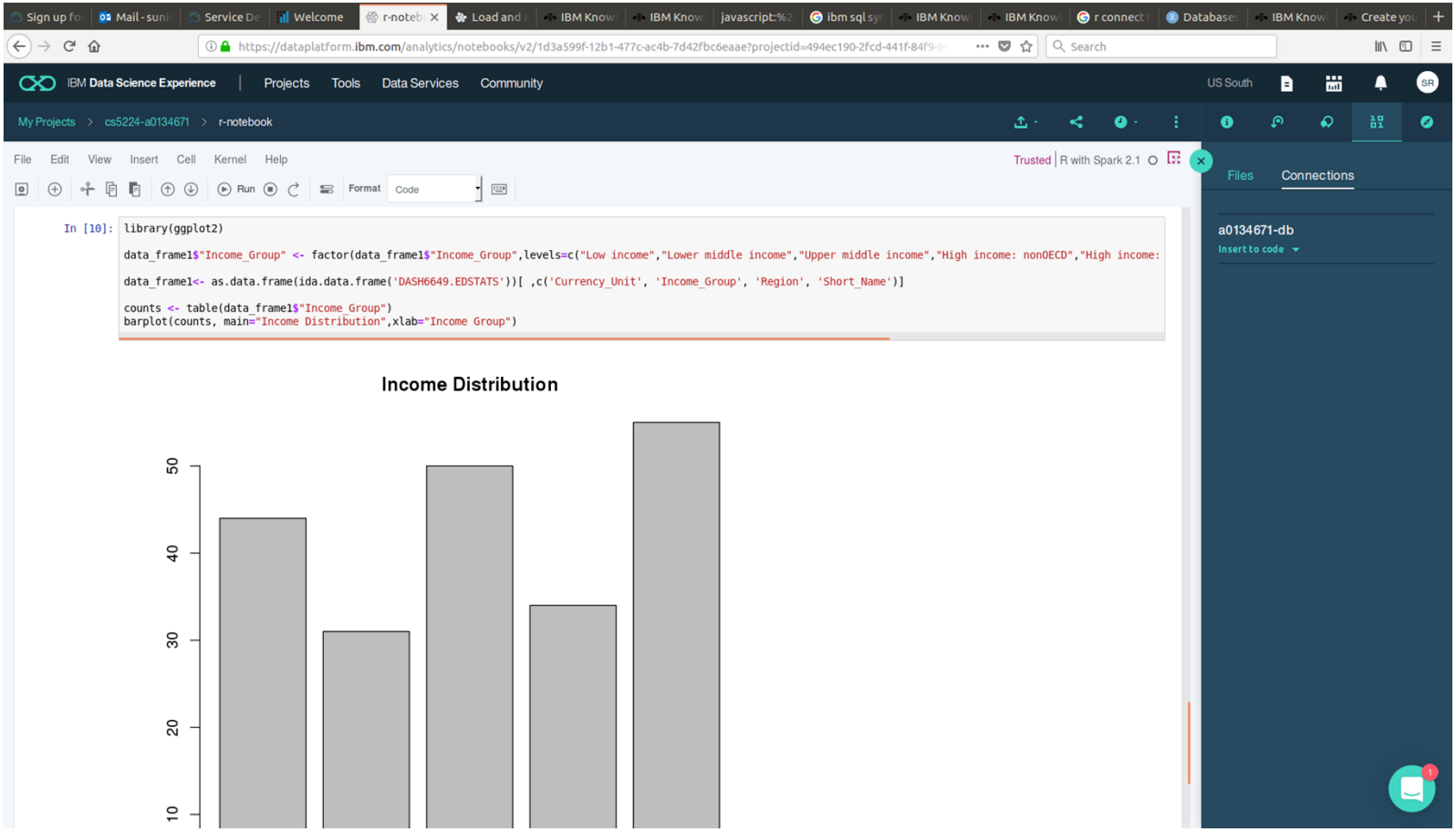
```
library(ggplot2)

data_frame1<- as.data.frame(ida.data.frame('DASH6649.EDSTATS'))[
,c('Currency_Unit', 'Income_Group', 'Region', 'Short_Name')]

data_frame1$"Income_Group" <-
factor(data_frame1$"Income_Group",levels=c("Low income","Lower
middle income","Upper middle income","High income:
nonOECD","High income: OECD"),
labels=c("LI","LMI","UMI","HIN","HIO"))

counts <- table(data_frame1$"Income_Group")
barplot(counts, main="Income Distribution",xlab="Income Group")
```

Step 13: Output the Plot



Milestone 6: Visualize data

- Did you successfully visualize the data retrieved from the database?

- Congratulations! This is the last step of the hands on activity. You may refer to the references section at the end of the presentation slides to learn more or search on the web.

- Next step: Lab Assessment

Summary

- hands-on with BlueMix
- simple web server application using node.js and Db2
- integrating IBM Watson personality insights service with a node.js web application
- simple SQL query using Db2 Warehouse SaaS
- simple R script to plot results of data on Db2 and DSX

References

- NodeJS-DashDB example
- Demo: Getting Started with Node.js on Bluemix 2016- https://www.youtube.com/watch?v=sHhNoV-sS_I&list=PLJxa6IsF8C5qFbRinR2ZaEZ3AiIFtdLqx (13 mins)
- IBM Bluemix The Cloud Platform for Creating and Delivering Applications, IBM Redbooks, 2015.
- Bluemix users: https://www.ibm.com/cloud-computing/bluemix/case-studies
- Watson SDK help: https://github.com/watson-developer-cloud/node-sdk
- Watson startup guide: https://console.bluemix.net/docs/services/watson/developing-nodejs.html#developing-a-watson-application-in-node-js
- IBM dashDB:
  http://www-01.ibm.com/support/knowledgecenter/SS6NHC/com.ibm.swg.im.dashdb.kc.doc/welcome.html
- SQL Query: http://www.w3schools.com/sql/
- R Language:
  http://cran.r-project.org/doc/manuals/r-release/R-intro.html
- Plotting: http://docs.ggplot2.org/current
- IBM DSX Getting Started Guide: https://datascience.ibm.com/docs/content/getting-started/get-started.html
- IBM DSX Tutorials: https://github.com/IBMDataScience/dsx-tutorials