How to deploy your spring boot Java application on Heroku cloud ?

Used Link:

<https://devcenter.heroku.com/articles/deploying-spring-boot-apps-to-heroku>

Steps:

**1.**Create your spring boot application either with ‘maven’ or ‘gradle’ build tool. You can do it either using STS(Spring Tool Suit) or by springinitializer.io site . Create some rest controller to expose the endpoints. Create Database table and pojo mapping with it. Write CRUD operations on it.

**2.**Now install heroku cli .exe from <https://devcenter.heroku.com/articles/deploying-spring-boot-apps-to-heroku> and check if installed properly by firing command

**>heroku login**

heroku login

Enter your Heroku credentials.

Email: java@

Generating example.com

Password:

Could not find an existing public key.

Would you like to generate one? [Yn]

new SSH public key.

Uploading ssh public key /Users/java/.ssh/id\_rsa.pub

**3.** Go to your projects root directory and commit your project to git repository.

Before you can deploy the app to Heroku, you’ll need to create a Git repository for the application and add all of the code to it by running these commands:

$ git init

$ git add .

$ git commit -m "first commit"

You’ll deploy the app by pushing this Git repo to Heroku..

**4.** In order to deploy to Heroku, you’ll first need to provision a new Heroku app. Run this command:

$ heroku create <your app name>

Creating nameless-lake-8055 in organization heroku... done, stack is cedar-14

http://nameless-lake-8055.herokuapp.com/ | git@heroku.com:nameless-lake-8055.git

Git remote heroku added

This also creates a remote repository called heroku in your local git repo. Heroku generates a random name (in this case nameless-lake-8055) for your app. You can rename it later with the heroku apps:rename command.

**5. Now deploy your code:**

Deploying your code means moving your git pushed code to heroku cloud and running over there.

So , whatever code is commited on git only that will go to heroku , so each time you make code changes in your local make sure you push it first to your git repo and then push it to heroku repo.

git push heroku master

Initializing repository, done.

Counting objects: 110, done.

Delta compression using up to 4 threads.

Compressing objects: 100% (87/87), done.

Writing objects: 100% (110/110), 212.71 KiB | 0 bytes/s, done.

Total 110 (delta 30), reused 0 (delta 0)

-----> Java app detected

-----> Installing OpenJDK 1.8... done

-----> Installing Maven 3.3.3... done

-----> Executing: mvn -B -DskipTests=true clean install

[INFO] Scanning for projects...

...

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 11.417s

[INFO] Finished at: Thu Sep 11 17:16:38 UTC 2014

[INFO] Final Memory: 21M/649M

[INFO] ------------------------------------------------------------------------

-----> Discovering process types

Procfile declares types -> web

Heroku automatically detects the application as a Maven/Java app due to the presence of a pom.xml file. It installed Java 8 by default

Note: Now we have deployed the code but this project do not have any database configured. Oh heroku MySQL is not free, so instead we can configure postgresql db which is free.

**6.Configure PostgresSQL :**

>login to your heroku account on web. Now there it shows our newly pushed project in personal tab. Click on it.

>Click on resources tab and in Add-ons, search postres it will show you Heroku Postgres -> Select it ->  
Hobby Dev –Free > click on provision and it will create a dyno (db instance for your project).

> That’s it its done for heroku db config.

> Now this postgresql db has to configure in your springboot project and test with local postgresql db.

>SO install postgersql db on your local machine and create db ‘sachintestdb’.

> So add postgesql gradle dependency in **build.gradle** file.

**runtime('mysql:mysql-connector-java') //for mysql**

**runtime('org.postgresql:postgresql') //for PostgreSQL**

>in application.properties file add following config for local db test:

##################FONFIG FOR LOCAL POSTGRESQL DB ENABLE THIS SECTION ##############

#spring.jpa.database=POSTGRESQL

#spring.datasource.platform=postgres

#spring.jpa.show-sql=true

#spring.jpa.hibernate.ddl-auto=update

#spring.database.driverClassName=org.postgresql.Driver

#spring.datasource.url=jdbc:postgresql://localhost:5432/sachintestdb

#spring.datasource.username=postgres

# spring.datasource.password=root

>now clean build project and run it , test it by pushing some records in local db using Rest client.

>if local db testing is successful , then you just have to replace **datasource.url,username,and password**  properties with your heroku database url, username and password. To get these credientials select your project in heroku app in web. There it will show our configured db in resources tab > click on the db name > then go to settings tab for that db >click on view credentials button> It will show you Host,DBName,User,Port,Password and database URI.

> Copy DB URI, username and password and replace it in your local application.properties file.

##################### CONFIG FOR HEROKU POSTGRESQL DB ENABLE THIS ##############

spring.jpa.database=POSTGRESQL

spring.datasource.platform=postgres

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

spring.database.driverClassName=org.postgresql.Driver

spring.datasource.url=jdbc:postgresql://postgres://msvbaxsahvsnnn:9883fd8732379de0a40d81fbf4174a19f1ff3d70bbe5e0dc00c257988aa5f560@ec2-174-129-221-240.compute-1.amazonaws.com:5432/df3nluhdd61gc2

spring.datasource.username=msvbaxsahvsnnn

spring.datasource.password=9883fd8732379de0a40d81fbf4174a19f1ff3d70bbe5e0dc00c257988aa5f560

################### /CONFIG FOR HEROKU POSTGRESQL DB ENABLE THIS ##############

>Its done , it will work for heroku db also now.

> Now again commit your latest changes to git repo and then push those changes again to heroku repo using  **$git push heroku master** command and **$heroku open** .

>It will open your application. Copy the url and you can access it anywhere now.

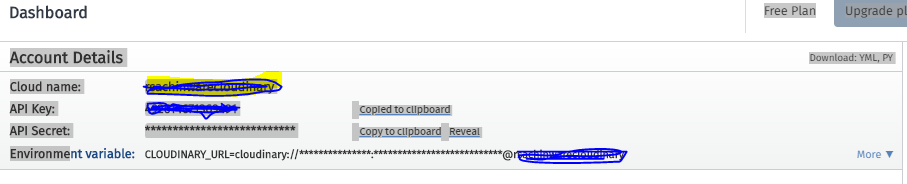
e.g. <https://sachin-ware-sb-rest-server.herokuapp.com>

# Cloudinary integration to upload image and video files to cloud

Steps:

1. Create your own cloudinary account at : <https://cloudinary.com>

After creation of account you will get details on cloudinary dashboard like following:



1. Put cloudinary dependency in your build.gradle file and run gradlew clean build OR gradelw eclipse eclipse :

// https://mvnrepository.com/artifact/com.cloudinary/cloudinary-http43

compile group: 'com.cloudinary', name: 'cloudinary-http43', version: '1.2.2'

**For Maven:**

<!-- https://mvnrepository.com/artifact/com.cloudinary/cloudinary-core -->

<dependency>

<groupId>com.cloudinary</groupId>

<artifactId>cloudinary-core</artifactId>

<version>1.19.0</version>

</dependency>

1. Create REST End-Point Like:

@PostMapping("/uploadPic")

**public** ResponseEntity<Object> upload(@RequestParam("file") MultipartFile multipartFile){

String cloudinaryImgURL=**null**;

**try** {

File fileDir = **new** File("rowFiles");

**if** (! fileDir.exists()){

fileDir.mkdir();

// If you require it to make the entire directory path including parents,

// use directory.mkdirs(); here instead.

}

String fileName=multipartFile.getOriginalFilename();

File physicalFile=**new** File(multipartFile.getOriginalFilename());

FileOutputStream fout=**new** FileOutputStream(fileDir.getName()+"/"+physicalFile);

fout.write(multipartFile.getBytes());

fout.close();

/\*Cloudinary cloudinary = new Cloudinary(ObjectUtils.asMap(

"cloud\_name", "your\_cloud\_name",

"api\_key", "your\_api\_key",

"api\_secret", "your\_secret\_key"));\*/

//OR

//used this option when you have set environment variable CLOUDINARY\_URL

File toUpload = **new** File("rowFiles/"+fileName);

//This type will fetch cloudinary account details from environment variable **CLOUDINARY\_URL**

Cloudinary cloudinary = **new** Cloudinary();

System.***out***.println("API Key:"+cloudinary.config.apiKey);

Map params = ObjectUtils.*asMap*("public\_id", "SRWRestImageBase/"+fileName);

Map uploadResult = cloudinary.uploader().upload(toUpload, params);

//Map uploadResult =Singleton.getCloudinary().uploader().upload(toUpload, params);

toUpload.delete();

System.***out***.println("==============>>"+uploadResult.get("url"));

cloudinaryImgURL=uploadResult.get("url").toString();

} **catch** (Exception e) {

System.***out***.println("upload:"+e.getMessage());

// **TODO**: handle exception

}

**return** **new** ResponseEntity<Object>("File uploaded successfully:"+cloudinaryImgURL,HttpStatus.***OK***);

}

Note: If you don’t want to expose your cloudinary account details to public repository of git , set these values to environment variable “CLOUDINARY\_URL” like

CLOUDINARY\_URL=cloudinary://my\_api\_key:my\_api\_secret@my\_cloud\_name

//You can find these details on dashboard page of cloudinary after successfully created the account.

And restart your eclipse to reflect environment variable changes.

Then you can create cloudinary object simply like :

Cloudinary cloudinary = **new** Cloudinary();

This will fetch cloudinary account details from environment variable.

1. Then you can use postman to test this endpoint

* Open Postman rest client
* Select post method in method type and put your endpoint url in url section
* Go to body tab and select key type as ‘file’
* File upload option will appear, so select any image file less that 1 mb as its restricted to 1mb.
* Hit the send button, you will get response.