

**IMPORTANT:**  
Specification described on the homework description takes  
precedence over the instructions here  
(i.e., follow the homework description when there is a conflict)  
(Example app: <https://github.com/tobbetubil/myDemoApp>)

## Preparation

- Example instruction

```
$ example_command  
  
example_output_for_the _command
```

- Example fill in the blanks:

```
abc <xyz> def
```

This means you need to replace <xyz> with your information. So if it is 123, then the whole expression should be abcxyzdef.

## Sign up for a github account

<https://github.com/>

## Sign up for an account on Heroku

<http://heroku.com/>

## Operating System

You need to use the terminal on this homework. The instructions are provided for Windows gitbash. If you have Ubuntu or Mac installed on your computer then the instructions should still work.

- If you have Windows, and but do not have gitbash, then you can install it following these instructions: <https://gitforwindows.org/>.

## Java

Set up Java8 or above (You will use lambda expressions which requires  $\geq$  JDK8)

## Git

Set up git if you do not have it already. Type (\$ represents the cursor so do not type \$)

```
$ git
```

on your terminal and see if it responds.

- If you do not have git, then this is how you install it on Ubuntu:

```
$ sudo apt-get install git
```

(Read this or similar resource for more information on using git on ubuntu):

<https://www.howtoforge.com/tutorial/install-git-and-github-on-ubuntu-14.04/>

## Maven

Set up maven on Windows:

- <http://maven.apache.org/download.cgi>
- <http://maven.apache.org/install.html>

Set up maven on Ubuntu:

```
$ sudo apt-get install maven
```

Verify the installation:

```
$ mvn -version
Apache Maven 3.5.2 (138edd61fd100ec658bfa2d307c43b76940a5d7d;
2017-10-18T10:58:13+03:00)
```

(You may see a more recent version printed)

**When following the instructions, keep the following documents close by in case you have a question. So now, make yourself familiar with what is in the following documents:**

git:

Read [Git from bottom up](#)

Play with [try.github.io](http://try.github.io)

Maven

<https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>

Spark

<http://sparkjava.com/>

Mustache

<http://coenraets.org/blog/2011/12/tutorial-html-templates-with-mustache-js/>

**Create the maven project:**

From the (gitbash) terminal:

Create a mvn project.

```
$ mvn -B archetype:generate \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DgroupId=com.mycompany.app \
-DartifactId=myDemoApp
```

This created a folder called *myDemoApp*. Verify the folder exists:

```
$ ls myDemoApp/  
pom.xml src
```

Verify that the folder structure is created: (Use `tree.com //a //f` on Windows and `tree` on Ubuntu)

```
$ tree.com //a //f  
.  
├── myDemoApp  
│   ├── pom.xml  
│   └── src  
│       ├── main  
│       │   ├── java  
│       │   │   ├── com  
│       │   │   │   ├── mycompany  
│       │   │   │   │   ├── app  
│       │   │   │   │   └── App.java  
│       │   └── test  
│       │       ├── java  
│       │       │   ├── com  
│       │       │   │   ├── mycompany  
│       │       │   │   │   ├── app  
│       │       │   │   │   └── AppTest.java
```

Initialize a git repo inside myDemoApp (**make sure** that you are inside myDemoApp folder before executing “git init”)

```
$ cd myDemoApp  
$ git init
```

Verify that there is a `.git` folder and it contains all the necessary folders and files.

```
$ ls .git
```

branches   config   description   HEAD   hooks   info   objects   refs
--

Verify that git commands work correctly

```
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    pom.xml
    src/

nothing added to commit but untracked files present (use "git add" to track)
```

Add your username and email for git. As described in

<https://help.github.com/articles/setting-your-username-in-git/>

The name and email you use should match with what you provided when you sign up for github.com account.

Verify that you set it up correctly:

```
$ git config --global user.name
YOUR NAME SHOULD PRINT HERE

$ git config --global user.email

YOUR EMAIL SHOULD PRINT HERE
```

## Add a small implementation to App.java

Write a method implementing a simple algorithm. It should accept some inputs and return some result. Open the file at "src/main/java/com/mycompany/app/App.java" with an editor.

Some popular options are:

- <https://code.visualstudio.com/>
- <https://www.sublimetext.com/>
- <http://www.eclipse.org/>
- <https://www.jetbrains.com/idea/>

Add a method that accepts some parameters and returns a value inside  
src/main/java/com/mycompany/app/App.java

For example, we will use the following method in the rest of this document:

```
public static boolean search(ArrayList<Integer> array, int e) {  
    System.out.println("inside search");  
    if (array == null) return false;  
  
    for (int elt : array) {  
        if (elt == e) return true;  
    }  
    return false;  
}
```

You need to add the necessary import statements so that the code compiles.

Make sure the project compiles:

```
$ mvn compile  
[INFO] Scanning for projects...  
[INFO]  
[INFO] -----  
[INFO] Building myDemoApp 1.0-SNAPSHOT  
[INFO] -----  
[INFO]  
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ myDemoApp ---  
[WARNING] Using platform encoding (Cp1254 actually) to copy filtered resources, i.e. build is  
platform dependent!  
[INFO] skip non existing resourceDirectory C:\Users\PC\481\myDemoApp\src\main\resources  
[INFO]  
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ myDemoApp ---  
[INFO] Changes detected - recompiling the module!  
[WARNING] File encoding has not been set, using platform encoding Cp1254, i.e. build is  
platform dependent!  
[INFO] Compiling 1 source file to C:\Users\PC\481\myDemoApp\target\classes  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 1.032 s  
[INFO] Finished at: 2018-09-24T05:55:47+03:00  
[INFO] Final Memory: 13M/371M  
[INFO] -----
```

Add the following *properties* section to the pom.xml file (after the *dependencies* section):

```
<project>
...
  </dependency>
</dependencies>

  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

</project>
```

This assumes that you have Java8 or above

```
$ java -version
```

```
java version "1.8.0_151"
Java(TM) SE Runtime Environment (build 1.8.0_151-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.151-b12, mixed mode)
```

Make sure your code still compiles (Fix the compilation errors - such as importing necessary packages)

```
$ mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building myDemoApp 1.0-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ myDemoApp ---
[WARNING] Using platform encoding (Cp1254 actually) to copy filtered resources, i.e. build is
platform dependent!
[INFO] skip non existing resourceDirectory C:\Users\PC\481\myDemoApp\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ myDemoApp ---
[INFO] Nothing to compile - all classes are up to date
[INFO] -----
```

```
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 0.768 s
[INFO] Finished at: 2018-09-24T05:59:40+03:00
[INFO] Final Memory: 9M/491M
[INFO] -----
```

If it does not, then read the output and fix the compilation errors.

## Write some unit tests

Open “src/test/java/com/mycompany/app/AppTest.java” and add test coverage with JUnit tests.

```
public void testFound() {
    ArrayList<Integer> array = new ArrayList<>(Arrays.asList(1, 2, 3, 4));
    assertTrue(new App().search(array, 4));
}

public void testNotFound() {
    ArrayList<Integer> array = new ArrayList<>(Arrays.asList(1, 2, 3, 4));
    assertFalse(new App().search(array, 5));
}

public void testEmptyArray() {
    ArrayList<Integer> array = new ArrayList<>();
    assertFalse(new App().search(array, 1));
}

public void testNull() {
    assertFalse(new App().search(null, 1));
}
```

Make sure to add the necessary imports.

```
import java.util.ArrayList;
import java.util.Arrays;

import junit.framework.TestCase;
```



Check your pom.xml and see what junit your project uses. In pom.xml, see <dependencies> section. Depending on the version, you may need to import Test class and use annotation for each Test method. <https://stackoverflow.com/questions/16062910/annotation-type-expected>

For example, if it is 3.8, then you start each method with test... But if you have >=4 then you need to use @Test annotation.

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Run the unit tests and make sure all tests pass:

```
$ mvn test
.....
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.632 s
```

Create a file named **README.md** inside the top folder (the same folder where pom.xml is). Write some description inside the file describing what this project is about.

Add a file named “**.gitignore**” inside the top folder (the same folder where pom.xml is)

Add the following content:

```
target/
```

This will exclude target/ folder (all compilation results) from your git repository history.

Verify you have the correct output for git status. If you have other file or folders in the output, then add each of these to .gitignore file.

```
$ git status

On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    .gitignore
    README.md
    pom.xml
    src/
```

For instance, let's say there is a folder “.settings/” in the git status output. Then append a line “.settings/” to the .gitignore file.

Add all the new files for tracking

```
$ git add .gitignore
$ git add README.md
$ git add pom.xml
$ git add src/
```

Check git status again

```
$ git status
```

On branch master

Initial commit

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: .gitignore

new file: README.md

new file: pom.xml

new file: src/main/java/com/mycompany/app/App.java

new file: src/test/java/com/mycompany/app/AppTest.java

Commit the changeset

```
$ git commit -m "I have started my awesome project!!!! It will be great."
```

```
[master (root-commit) 009cc98] ....
```

```
5 files changed, 98 insertions(+)
```

```
create mode 100644 .gitignore
```

```
create mode 100644 README.md
```

```
create mode 100644 pom.xml
```

```
create mode 100644 src/main/java/com/mycompany/app/App.java
```

```
create mode 100644 src/test/java/com/mycompany/app/AppTest.java
```

Check the git status again. You should see nothing to commit.

```
$ git status
```

On branch master

nothing to commit, working tree clean

Check your git commit history:

```
$ git log
```

```
commit 0edf1ccc7fdc1a86e72fbe01d03e8e3ca2089508
```

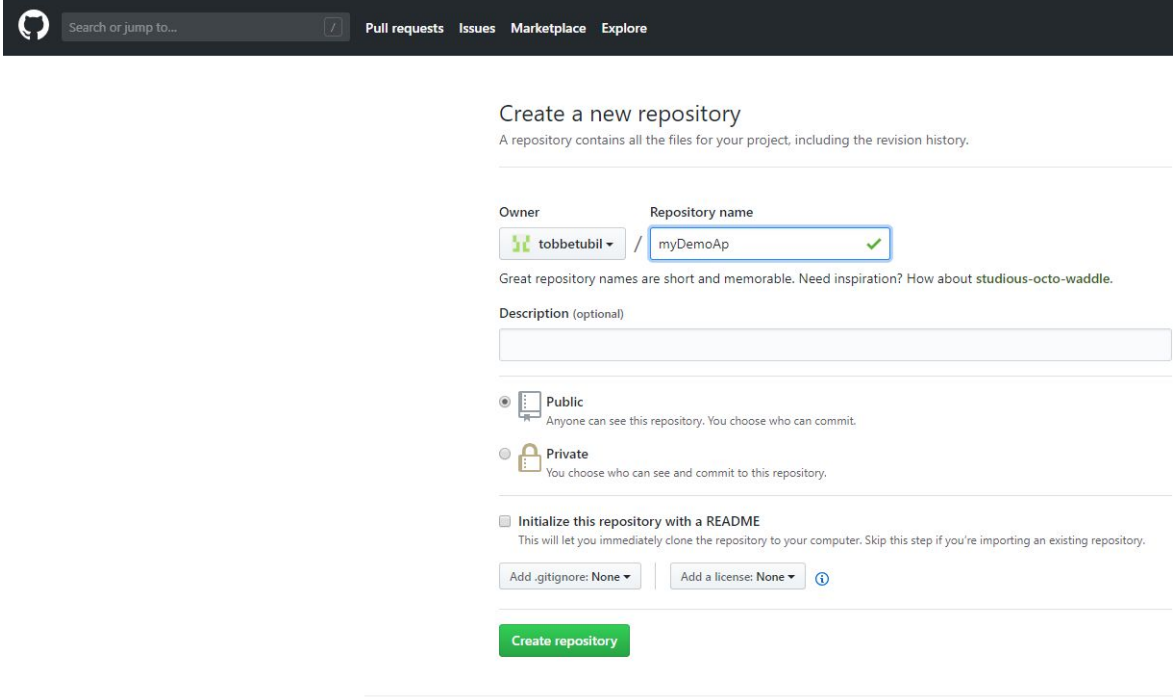
```
Author: .....
```

```
Date: .....
```

```
I have started my awesome project. It will be great.
```

## Uploading your changes to github


From the github.com, create a repository called “myDemoApp”.



Search or jump to... Pull requests Issues Marketplace Explore

### Create a new repository

A repository contains all the files for your project, including the revision history.

Owner:  tobbetubil / Repository name:

Great repository names are short and memorable. Need inspiration? How about [studious-octo-waddle](#).

Description (optional):

☒ **Public**  
Anyone can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository.

☒ **Initialize this repository with a README**  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** | Add a license: **None** ⓘ

**Create repository**

Add a remote target for your local git repository.

```
$ git remote add origin https://github.com/<your account name>/myDemoApp.git
```

Verify that you added correctly:

```
$ git remote -v
```

```
origin https://github.com/<your account>/myDemoApp.git (fetch)
origin https://github.com/<your account>/myDemoApp.git (push)
```

Push your local repo to github:

```
$ git push origin master

Username for 'https://github.com': *****
Password for 'https://*****@github.com':
Counting objects: 18, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (18/18), 1.75 KiB | 0 bytes/s, done.
Total 18 (delta 0), reused 0 (delta 0)
To https://github.com/*****/myDemoApp.git
* [new branch]    master -> master
```

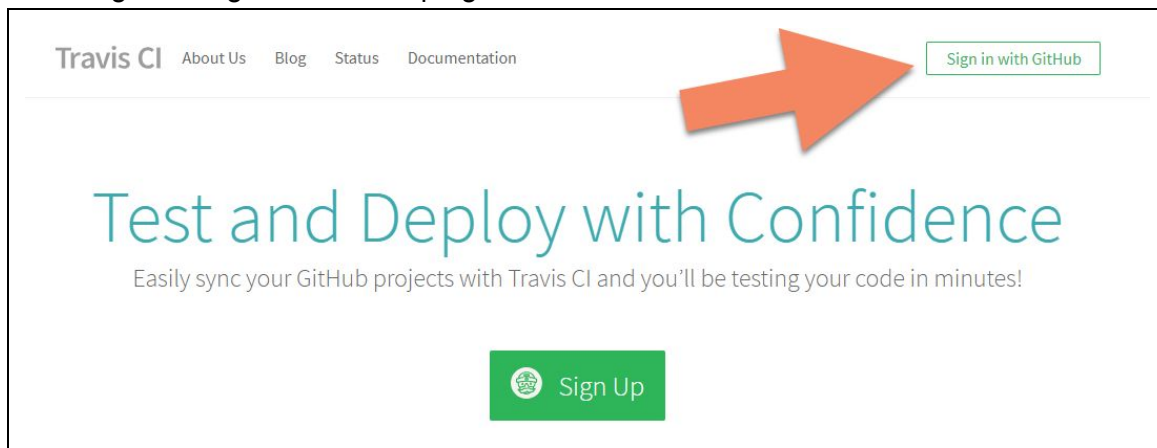
Check your github.com repo and make sure the files are listed there.

Go to <https://github.com/<YOUR ACCOUNT NAME>/mydemoapp>




## Setting up continuous integration:

Enable Travis build

- Go to <https://travis-ci.org/>
- Click Log in with github at the top right




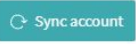

- Find your github repository under “Legacy Services Integration” and enable it by using the toggle.

Travis CI  [Changelog](#) [Documentation](#)  

---

MY ACCOUNT

 **tobbetubil**


 

ORGANIZATIONS

You are not currently a member of any organization.

MISSING AN ORGANIZATION?




[Review and add your authorized organizations.](#)




 **tobbetubil**  
@tobbetubil

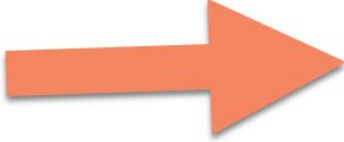
[Repositories](#) [Settings](#)

We're only showing your public repositories. You can find your private projects on [travis-ci.com](https://travis-ci.com).

Legacy Services Integration

 collapsing-puzzle   Settings

 myDemoApp   Settings



Go back to your project, create a new file named “.travis.yml” and add the following content:

```
language: java
jdk:
  - oraclejdk8
```

Check git status:

```
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    .travis.yml

nothing added to commit but untracked files present (use "git add" to track)
```

Add the file for tracking

```
$ git add .
```

Check the git status

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   .travis.yml
```

Commit the changeset

```
$ git commit -m "Setting up continuous integration"
[master cba3d8d] Setting up continuous integration
 1 file changed, 5 insertions(+)
 create mode 100644 .travis.yml
```

Check the git status and make sure it is clean

```
$ git status
On branch master
nothing to commit, working directory clean
```

Verify your git log

```
$ git log
commit 5f213ba285018763ce5f1047ab40cb594a5b36bb
Author: XXXXXXXXXXXX
Date:  XXXXXXXXXXXX

    Setting up continous integration

commit 0edf1ccc7fdc1a86e72fbe01d03e8e3ca2089508
Author: XXXXXXXXXXXX
Date:  XXXXXXXXXXXX

    I have started my awesome project. It will be great.
```

Push the local repo to github

```
$ git push origin master
```

```
Username for 'https://github.com': *****
Password for 'https://*****@github.com':
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 351 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local objects.
To https://github.com/*****/myDemoApp.git
    17c0de8..cba3d8d  master -> master
```

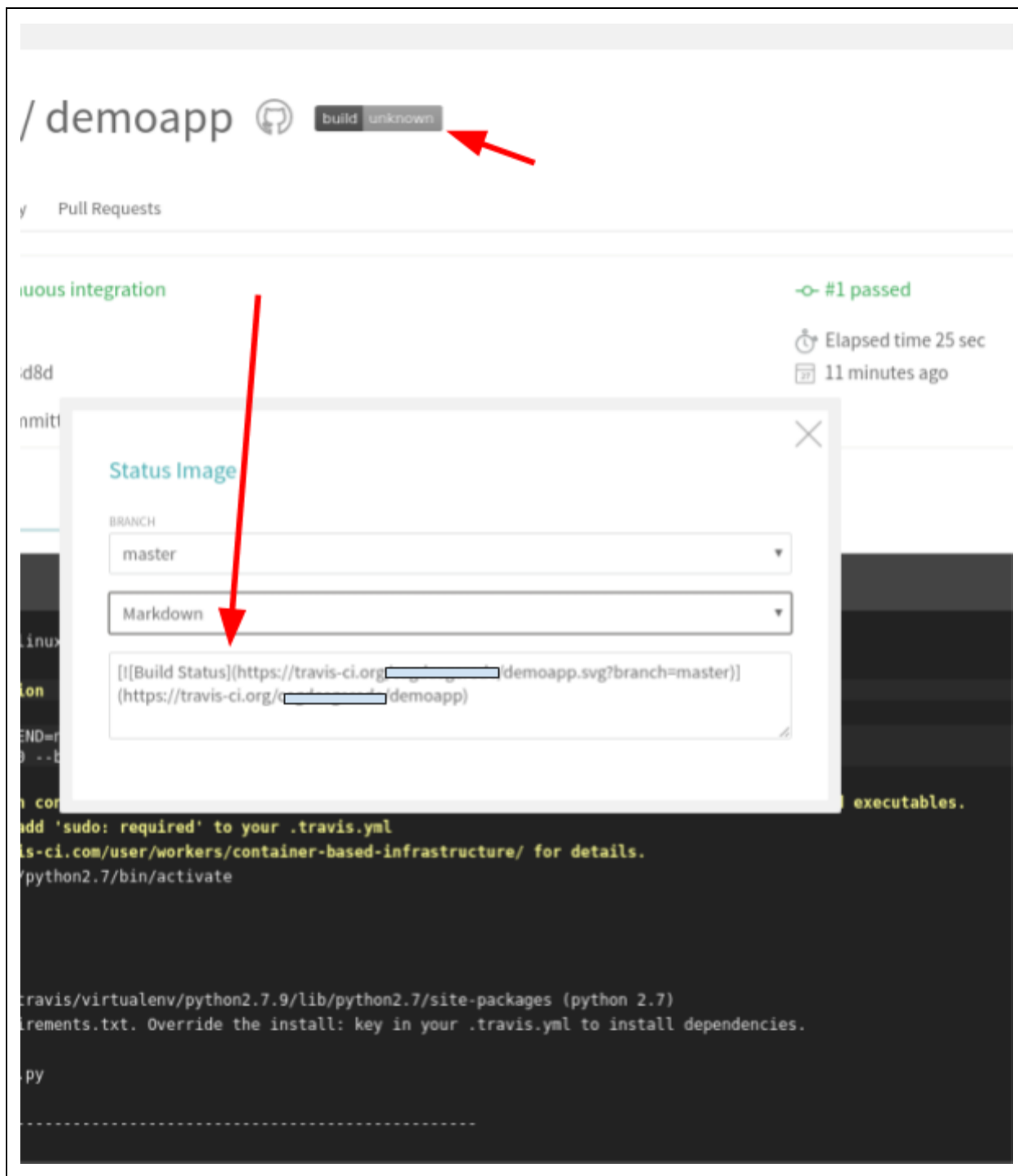
## Adding build status to your repository page

- In travis, go to your repository:

*<https://travis-ci.org/<your account name>/myDemoApp>*

Copy the link that is displayed when you click to [build | unknown] image as shown below (Make sure to select **Markdown**):





Append the text to the README.md file in your computer.

Commit the README change.

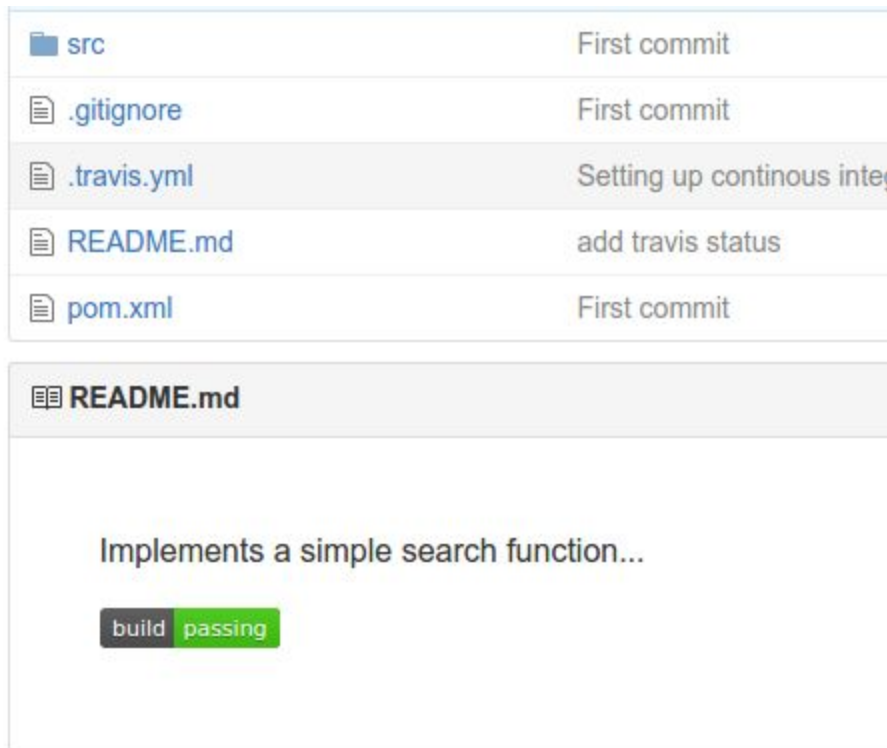
```
$ git add README.md
```

```
$ git commit -m "Added build status image to README file"  
[master db64332] Added build status image to README file  
1 file changed, 2 insertions(+)
```

Push the local repo to github

```
$ git push origin master
```

Check the repo in github and make sure that it has the build passing image inside README.md file.



## Deploying the app to a server on the internet

Add dependency to Spark and mustache

- Add the following snippet to your pom.xml file *inside the <dependencies> section*:

```
<dependency>
  <groupId>com.sparkjava</groupId>
  <artifactId>spark-core</artifactId>
  <version>2.5.4</version>
</dependency>

<dependency>
  <groupId>com.sparkjava</groupId>
  <artifactId>spark-template-mustache</artifactId>
  <version>2.3</version>
</dependency>

<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
  <version>2.10.0</version>
</dependency>

<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-api</artifactId>
  <version>2.10.0</version>
</dependency>

<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-slf4j-impl</artifactId>
  <version>2.10.0</version>
</dependency>
```

(For more information on Spark installation see <http://sparkjava.com/download.html>)

Add a mustache template by creating a file named compute.mustache in  
"src/main/resources/templates/compute.mustache"

Add the following content to the file

```
<title>My application</title>
<h1>Hello, World!</h1>
  Result is {{result}}

<form action="compute" method="post">
  <textarea name="input1" rows="10" cols="30"></textarea>
  <textarea name="input2" rows="10" cols="30"></textarea>
  <br>
  <input type="submit">
</form>
```

Update the App file (src/main/java/com/mycompany/app/App.java) with Spark routes by updating the file content as shown below. Note that the main method is where the requests arrive. See how each route is mapped to a big lambda expression. For instance, for the line *get("/", (req, res) -> "Hello, World")* says that for the default requests to the web application, we respond with "Hello World".

```
package com.mycompany.app;

import static spark.Spark.get;
import static spark.Spark.port;
import static spark.Spark.post;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import spark.ModelAndView;
import spark.template.mustache.MustacheTemplateEngine;

public class App
{
    public static boolean search(ArrayList<Integer> array, int e) {
        System.out.println("inside search");
        if (array == null) return false;

        for (int elt : array) {
            if (elt == e) return true;
        }
        return false;
    }
}
```

```

public static void main(String[] args) {
    port(getHerokuAssignedPort());

    get("/", (req, res) -> "Hello, World");

    post("/compute", (req, res) -> {
        //System.out.println(req.queryParams("input1"));
        //System.out.println(req.queryParams("input2"));

        String input1 = req.queryParams("input1");
        java.util.Scanner sc1 = new java.util.Scanner(input1);
        sc1.useDelimiter("[;\r\n]+");
        java.util.ArrayList<Integer> inputList = new java.util.ArrayList<>();
        while (sc1.hasNext())
        {
            int value = Integer.parseInt(sc1.next().replaceAll("\\s", ""));
            inputList.add(value);
        }
        System.out.println(inputList);

        String input2 = req.queryParams("input2").replaceAll("\\s", "");
        int input2AsInt = Integer.parseInt(input2);

        boolean result = App.search(inputList, input2AsInt);

        Map map = new HashMap();
        map.put("result", result);
        return new ModelAndView(map, "compute.mustache");
    }, new MustacheTemplateEngine());

    get("/compute",
        (rq, rs) -> {
            Map map = new HashMap();
            map.put("result", "not computed yet!");
            return new ModelAndView(map, "compute.mustache");
        },
        new MustacheTemplateEngine());
}

static int getHerokuAssignedPort() {
    ProcessBuilder processBuilder = new ProcessBuilder();
    if (processBuilder.environment().get("PORT") != null) {
        return Integer.parseInt(processBuilder.environment().get("PORT"));
    }
}

```

```
        return 4567; //return default port if heroku-port isn't set (i.e. on localhost)
    }
}
```

Note that the main method's `post("/compute",` route parses the http query parameters and calls `search` method. In your case, you need to change this to call your method and also change the way the http query parameters are parsed.

Package the application as follows. If there are errors, then fix the errors until it packages successfully (Use google and [stackoverflow.com](https://stackoverflow.com) to search for solutions the error messages)

```
$ mvn package

...

Results :

Tests run: 4, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ myDemoApp ---
[INFO] Building jar:
/home/cagdass/git/zimbirti/myDemoApp/target/myDemoApp-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 3.302 s
[INFO] Finished at: 2018-02-08T19:30:33+03:00
[INFO] Final Memory: 19M/188M
[INFO] -----
```

Verify that the jar file is created under target/ folder

```
$ ls target
classes          generated-test-sources  maven-status          surefire-reports
generated-sources maven-archiver         myDemoApp-1.0-SNAPSHOT.jar test-classes
```

Next we will create a virtual machine on heroku and deploy the app there.

(For more information, see this:

<https://devcenter.heroku.com/articles/getting-started-with-java#introduction>)

First, add a file named "Procfile" at the top folder where the README.md is (the Procfile will tell Heroku how to start the app)

```
web: java -jar target/myDemoApp-1.0-SNAPSHOT.jar
```

Install the dependencies

```
$ mvn clean install
...

[INFO]
-----
[INFO] BUILD SUCCESS
[INFO]
-----
[INFO] Total time: 4.695 s
[INFO] Finished at: 2018-02-08T19:35:05+03:00
[INFO] Final Memory: 22M/263M
[INFO]
-----
```

Check you have all the modified/new files and folders reported in git status:

```
$ git status

On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
```

```
modified: pom.xml
modified: src/main/java/com/mycompany/app/App.java
```

Untracked files:  
(use "git add <file>..." to include in what will be committed)

```
Procfile
src/main/resources/
```

```
$ git add .
```

```
$ git commit -m "Getting ready for deployment to heroku"
```

Check git status again

```
$ git status
On branch master
nothing to commit, working directory clean
```

Check git log:

```
$ git log
commit 719c171ec613b8edbe11185ccbb9a3391c8f7d5e
Author: XXXXXXXX
Date: XXXXXXXX

    Getting ready for deployment to heroku

commit c73fad7c2f273a3c881fc477373e53f28465a913
Author: XXXXXXXX
Date: XXXXXXXX

    Added build status image to README file

commit 4f2577d6f7a0d8fec3aaaefe5af566df54c9c27b
Author: XXXXXXXX
Date: XXXXXXXX

    Trigger travix

commit 5f213ba285018763ce5f1047ab40cb594a5b36bb
Author: XXXXXXXX
```



Date: XXXXXXXX

Setting up continuous integration

commit 0edf1ccc7fdc1a86e72fbe01d03e8e3ca2089508

Author: XXXXXXXX

Date: XXXXXXXX

I have started my awesome project. It will be great.

## Heroku setup

- Install command line interface by following the article:  
<https://devcenter.heroku.com/articles/heroku-cli>

Add the following plugins to pom.xml **inside <project> tag**:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.5.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
        <optimize>true</optimize>
        <debug>true</debug>
      </configuration>
    </plugin>
    <plugin>
      <artifactId>maven-assembly-plugin</artifactId>
      <version>2.3</version>
      <configuration>
        <descriptorRefs>
          <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
      </configuration>
    </plugin>
  </plugins>
</build>
```

```
</descriptorRefs>
<finalName>myDemoApp-1.0-SNAPSHOT</finalName>
<appendAssemblyId>false</appendAssemblyId>
<archive>
  <manifest>
    <mainClass>com.mycompany.app.App</mainClass>
  </manifest>
</archive>
</configuration>
<executions>
  <execution>
    <id>build-jar-with-dependencies</id>
    <phase>package</phase>
    <goals>
      <goal>single</goal>
    </goals>
  </execution>
</executions>
</plugin>
</plugins>
</build>
```

Add a file named “system.properties” at the top folder with the following line in it:

```
java.runtime.version=1.8
```

Package your app

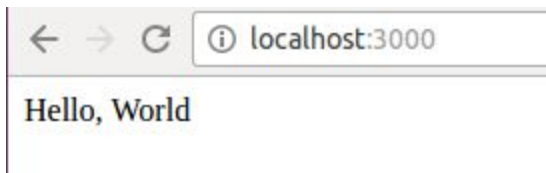
```
mvn clean package
```

Start your app through heoku CLI locally

```
$ heroku local -p 3000
```

```
[WARN] No ENV file found
20:01:21 web.1 | ERROR StatusLogger No log4j2 configuration file found. Using default
configuration: logging only errors to the console. Set system property 'log4j2.debug' to show
Log4j2 internal initialization logging.
```

In your browser, go to <http://localhost:3000>). You should see your demo app’s hello world page.



Stop the app from the terminal by pressing *Ctrl + C*

Check git status

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   pom.xml

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        system.properties

no changes added to commit (use "git add" and/or "git commit -a")
```

Add the new files to git

```
$ git add pom.xml  
$ git add system.properties  
$ git commit -m "Finished heroku deployment configuration"
```

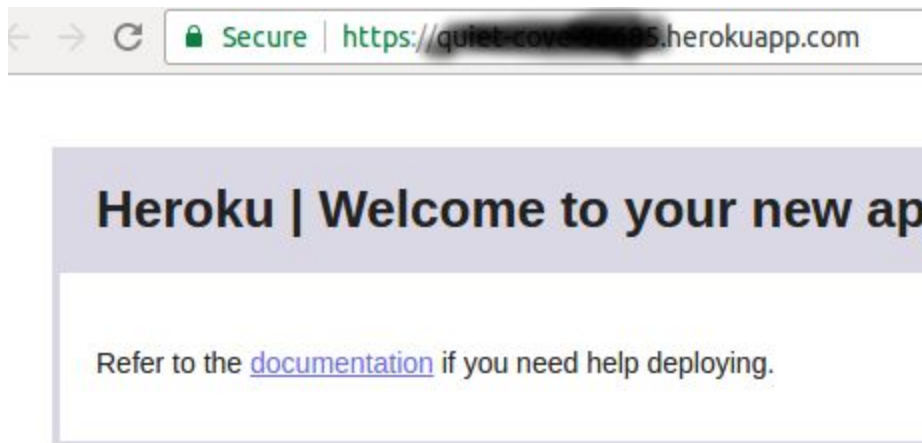
## Deploying to remote server in the cloud

Create a heroku instance

```
$ heroku create  
  
Creating app... done, 🍷 quiet-cove-123456  
https://quiet-cove-12345.herokuapp.com/ | https://git.heroku.com/quiet-cove-12345.git
```

There are two urls at the bottom. The first one is the address of your virtual machine on heroku cloud. The next url is your git repository on heroku.

If you go to the **first** url on the browser, you should see a default web page:



Push your demo app code to heroku:

```
$ git push heroku master
```

```
Counting objects: 9, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (9/9), 817 bytes | 0 bytes/s, done.
Total 9 (delta 2), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Java app detected
remote: -----> Installing OpenJDK 1.8... done
remote: -----> Installing Maven 3.3.9... done
remote: -----> Executing: mvn -DskipTests clean dependency:list install
remote:      [INFO] Scanning for projects...
remote:      [INFO]
remote:      [INFO]
....
....
....
remote: Verifying deploy... done.
To https://git.heroku.com/stormy-woodland-33416.git
   24e63f4..dee4756  master -> master
```

Open the first url again on the browser. You should see your demo app's hello world page that you have seen before in your local machine.

Now go to your url but add /compute to the end of it  
(Something like **<https://quiet-cove-12345.herokuapp.com/compute>**)

This will show the html forms for entering the parameters for the method you implemented.

Open the server log to see the log message from your app on heroku.

```
$ heroku logs --tail
```

You can enter values in the forms on the web page and then click the submit button to run the method. If there are problems, then check the log messages.

The default algorithm accepts a list of integers like below in the first box, and another integer on the second box. It searches the integer on the second box in the list. If the search is successful, it prints true at the result page as shown below

Enter the input values	Result
<p><b>Hello, World!</b></p> <p>Result is not computed yet!</p> <div> <div> 1 2 3 4 5 </div> <div>3 </div> </div> <p>Submit</p>	<p><b>Hello, World!</b></p> <p>Result is true</p> <div> <div></div> <div></div> </div> <p>Submit</p>

Next we will add the heroku app address to README.md file.

Open README.md file and append your heroku app address to the end of the file (whatever url you have been visiting to see your app on heroku):

Demo site: <https://quiet-cove-12345.herokuapp.com/>

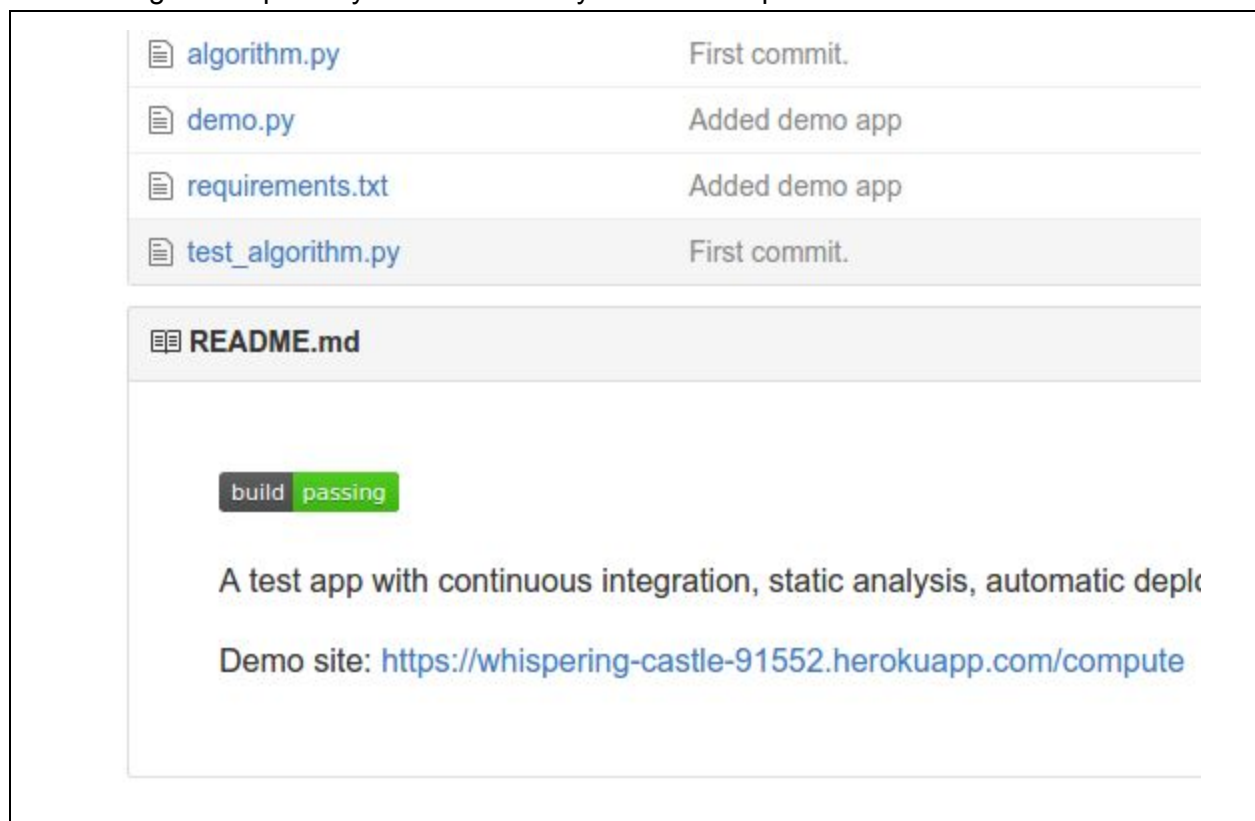
Commit the change and push the new version of the app to github:

```
$ git add README.md

$ git commit -m "Added demo site to README"
[master 3f93116] Added demo site to README
1 file changed, 3 insertions(+)

$ git push origin master
Username for 'https://github.com': *****
Password for 'https://*****@github.com':
Counting objects: 48, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (27/27), done.
Writing objects: 100% (48/48), 4.83 KiB | 0 bytes/s, done.
Total 48 (delta 10), reused 0 (delta 0)
remote: Resolving deltas: 100% (10/10), completed with 2 local objects.
To https://github.com/*****/myDemoApp.git
b0188fa..c8b21a2 master -> master
```

Check the github repository and make sure your see the updated README file content



The screenshot shows a GitHub repository interface. At the top, there is a list of files: `algorithm.py`, `demo.py`, `requirements.txt`, and `test_algorithm.py`. Each file has a commit message: `algorithm.py` and `test_algorithm.py` are marked as "First commit.", while `demo.py` and `requirements.txt` are marked as "Added demo app". Below the file list, the `README.md` file is selected and its content is displayed. The README content includes a "build passing" status badge, a description of the app as a test app with continuous integration, static analysis, and automatic deployment, and a demo site URL: <https://whispering-castle-91552.herokuapp.com/compute>.

File	Commit Message
<code>algorithm.py</code>	First commit.
<code>demo.py</code>	Added demo app
<code>requirements.txt</code>	Added demo app
<code>test_algorithm.py</code>	First commit.

**README.md**

build **passing**

A test app with continuous integration, static analysis, automatic deplk

Demo site: <https://whispering-castle-91552.herokuapp.com/compute>

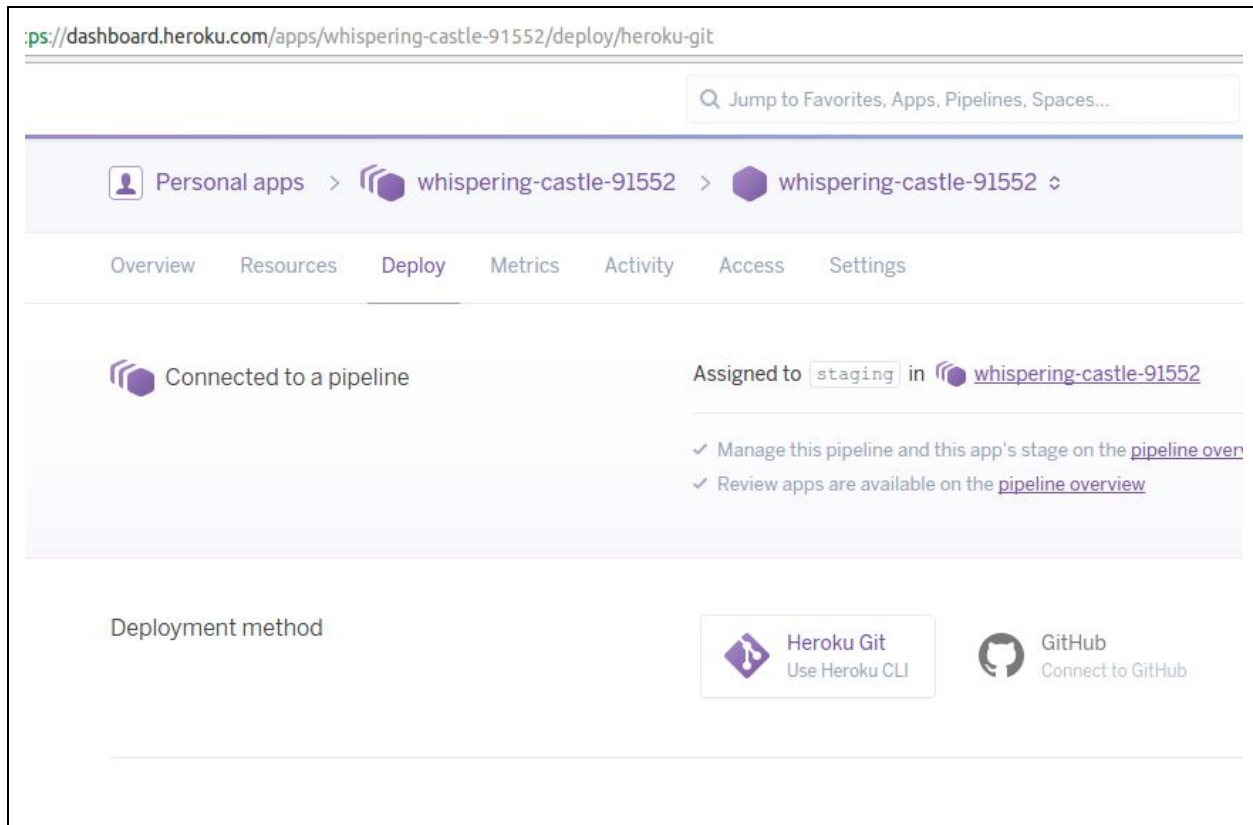
## Setting up continuous deployment

Go to

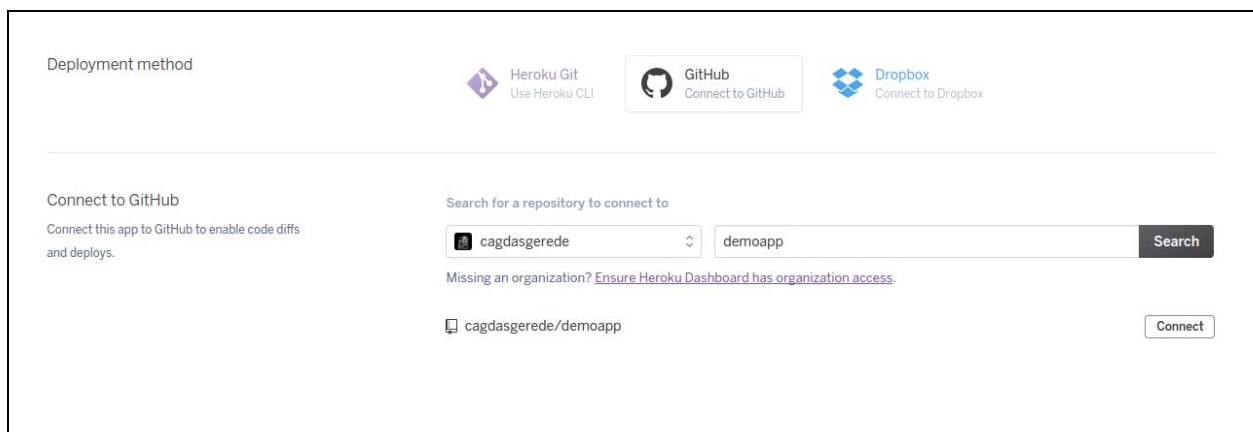
<https://dashboard.heroku.com/>

This page shows your virtual machines on heroku cloud.

Select the machine you just used. Go to Deploy tab. Then go to deployment method.

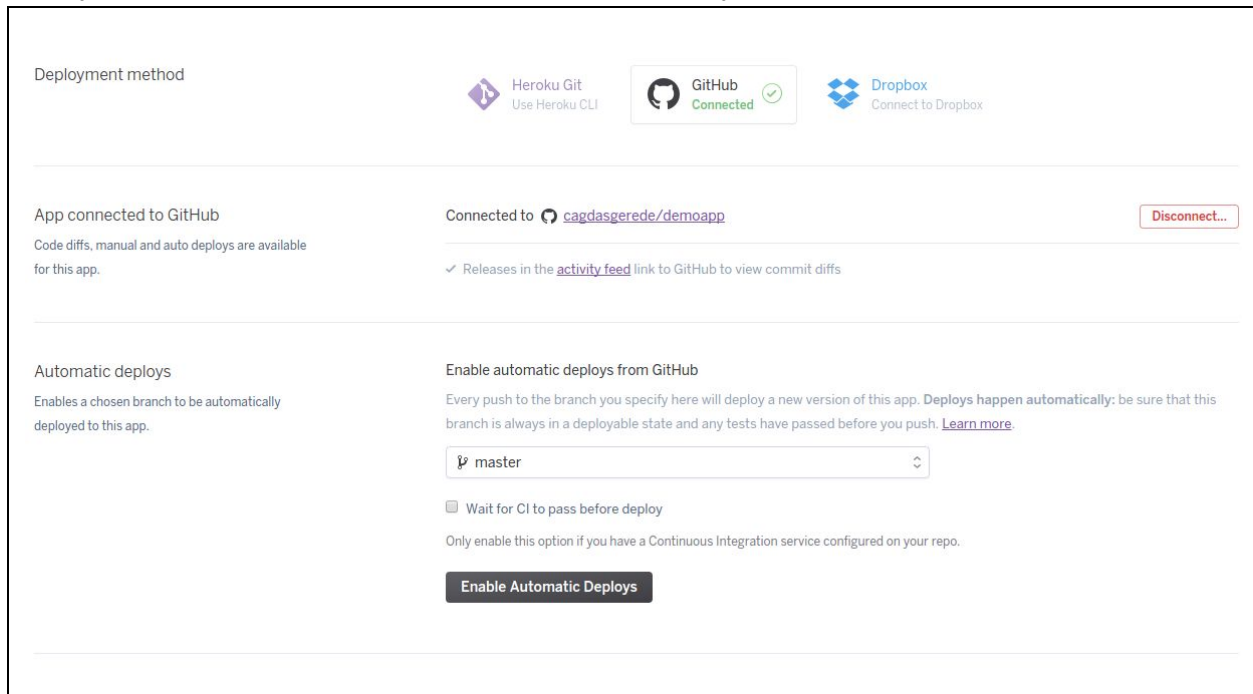


On Deployment method, click “GitHub” to connect your heroku app to github. Enter your repository’s name (demoapp) and search. Then, select “connect”.





Right below that section, you see “Automatic deploys” section. Click “Wait for CI to pass before deploy” checkbox, then press “Enable Automatic Deploys” button.



The screenshot shows the Heroku deployment settings for a GitHub repository. At the top, under "Deployment method", there are three options: Heroku Git (Use Heroku CLI), GitHub (Connected), and Dropbox (Connect to Dropbox). Below this, the "App connected to GitHub" section shows the repository "cagdasgerede/demoapp" with a "Disconnect..." button. A checkmark indicates that releases are in the activity feed. The "Automatic deploys" section is expanded, showing the "Enable automatic deploys from GitHub" option. A dropdown menu shows "master" as the selected branch. A checkbox labeled "Wait for CI to pass before deploy" is checked. Below this, a note states: "Only enable this option if you have a Continuous Integration service configured on your repo." At the bottom of this section is a button labeled "Enable Automatic Deploys".

Now, every time you deploy to github, you should see that version deployed to heroku app automatically. In other words, demo app will be updated automatically as you commit a new changeset.

To test this, go to *compute.mustache* file in your computer. Change the title of the page

```
<title>My application deployed automatically</title>
```

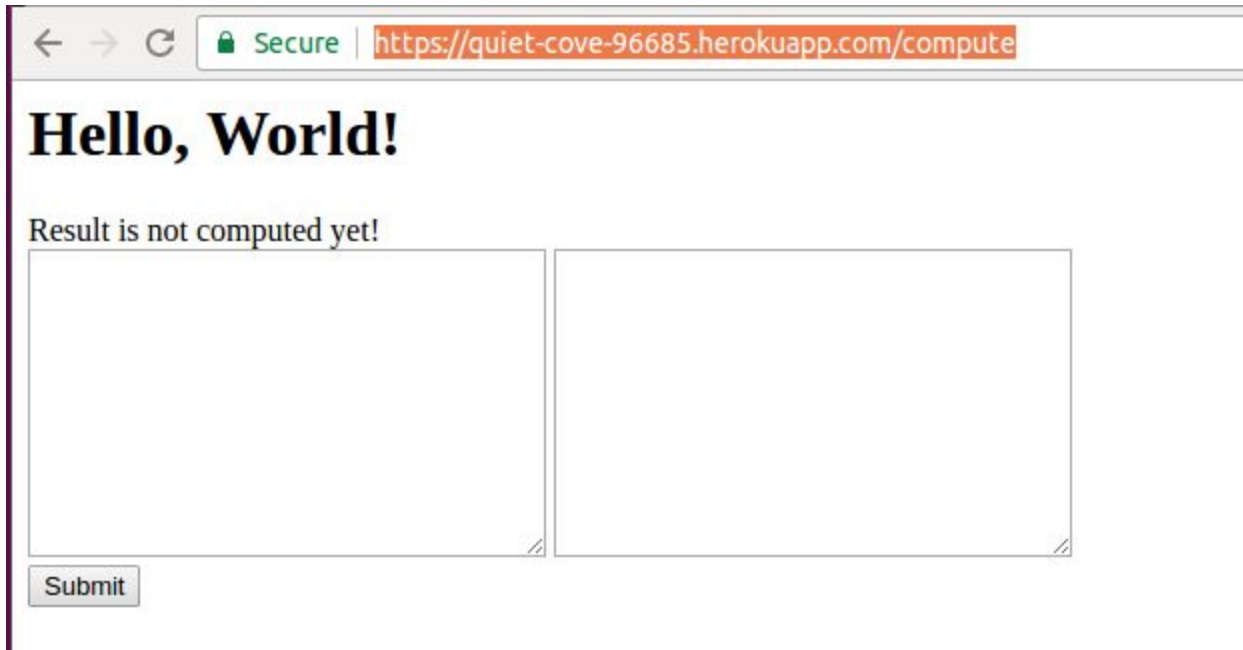
Then push the change to heroku

```
$ git add src/main/resources/templates/compute.mustache
```

```
$ git commit -m "Changing the title to mention continuous deployment"
```

```
$ git push origin master
```

Open your heroku app using the url you have been using (the one that ends with /compute). You should see the new title (It may take a couple of minutes).



The screenshot shows a web browser window with the address bar displaying "Secure | https://quiet-cove-96685.herokuapp.com/compute". The main content of the page features the text "Hello, World!" in a large, bold, black serif font. Below this, the text "Result is not computed yet!" is displayed in a smaller, regular black serif font. Underneath the text, there are two side-by-side empty text input fields with thin gray borders. At the bottom left of the form area, there is a "Submit" button with a gray background and black text.

(You can go to travis and heroku sites to monitor the state of the build and deploy process. You will see in travis that there is a new build process running. Once that is complete you will see on heroku that a new deployment happened).



Search all repositories



[My Repositories](#) +

✓ [cagdasgerede/mydemoapp2](#) # 5

🕒 Duration: 53 sec

📅 Finished: about 2 hours ago

# cagdasgerede / mydemoapp

[Current](#)

[Branches](#)

[Build History](#)

[Pull Requests](#)

✓ **master** Updating title to mention continous deployment

🔗 Commit e62c357 [🔗](#)

🔗 Compare 74ba2a7..e62c357 [🔗](#)

🔗 Branch master [🔗](#)