



# Cloud 300 Student Lab Manual



## Contents

|   |    |
|---|----|
| Lab: Implement a Protocol with Spring Boot.....           | 5  |
| Lab: Open Source – REST Setup .....                       | 8  |
| Lab: Open Source – RESTful Time Tracker Web Service ..... | 12 |
| Discussion: Analyzing an Application for Resiliency ..... | 16 |
| Discussion: Designing a Resilience Application.....       | 17 |
| Lab: Limits Service.....                                  | 18 |
| Lab: Spring Cloud Config Server .....                     | 24 |
| Lab: Pulling a Container .....                            | 26 |
| Lab: Manipulating a Container.....                        | 29 |
| Lab: Ports .....  | 35 |
| Lab: Dockerfile .....                                     | 39 |

This page intentionally left blank.

## Lab: Implement a Protocol with Spring Boot

### Overview

**Time: 15-30 Minutes**

In this lab, you will:

- Implement Protocol with Spring Boot

### Introduction

#### Step 1: Examine the new code of the pom.xml

We are going to modify the pom.xml in order to add some new components.

Take a look at the following:

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Notice that there are two new dependencies, both related to security:

1. spring-boot-starter-security
2. spring-security-test

These are going to introduce security into our applications. The way this will work is that it will force authentication.

We haven't specified a username or password. In fact, we haven't specified anything in our code. So how will it work?

The username will be "user", and the password will be an autogenerated UUID. We will get the password once we run the app.

#### Step 2: Copy the pom.xml to your directory

```
cp configs/1.2-pom.xml api-lab-1
```

### Step 3: Re-build

Go to the command line, and type:

```
cd api-lab-1
./mvnw clean package # Linux / Mac
```

Windows users should do the following:

```
cd api-lab-1
mvnw.cmd clean package
```

Again, we will see a lot of output.

If things went well, you should see something like this:

```
[INFO] --- spring-boot-maven-plugin:2.0.2.RELEASE:repackage (default) @ api-
lab-1 ---
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 11.896 s
[INFO] Finished at: 2018-06-12T22:58:35-07:00
[INFO] -----
```

### Step 4: Run my App

Let's again run our application. We can do that with Maven as well.

```
./mvn spring-boot:run
```

Followed by a bunch of more messages. This time, watch out for an extremely important message, which is the password.

```
Using generated security password: 594d8fc9-17e2-4b46-929b-4a9c125305a5
```

Of course, your security password will be different. Cut and paste the password. Because you will need it in the next step.

### Step 5: Test my app with authentication

Let's test it! Open your browser to <http://localhost:8080>

You will need to login as **user**, and the password will be the generated password you saw before.

If you correctly authenticate, you will see the following:

```
Hello from Spring Boot!
```

### Step 6: Use the command line

That is fantastic! But, wait a minute! Do I really have to open a browser up every time I want to authenticate? That could be a real pain.

Fortunately, no, you don't have to do that. Let's test our web service with "curl" -- or your favorite command line tool for testing web services. You will need to change your password credential to match the one you used in Step 5.

```
curl -i --user user:594d8fc9-17e2-4b46-929b-4a9c125305a5  
http://localhost:8080/
```

If you correctly authenticate, you will see the following:

```
Hello from Spring Boot!
```

## Lab: Open Source – REST Setup

### Overview

**Time: 30-45 Minutes**

In this lab, you will:

- Set up development environment for RESTful calls.

### Step 1: Install Anaconda

Open this URL in your browser: <https://conda.io/docs/user-guide/install/index.html>  
Follow the instructions for your operating system.

### Step 2: Check Flask

Flask should be installed in Anaconda.

Check by:

```
$ python
> import flask
```

If not, you can install as follows:

#### Option 1: Anaconda environment

```
$ conda install flask
```

#### Option 2: Python environment

```
$ pip install flask
```

### Step 3: Run Basic Flask Server

Source code for '300-Lab-Files' will be provided. Download it to your computer.

Execute the following file:

```
$ cd 300-Lab-Files
$ cd REST-labs
$ python ./simple-web-server.py
```

```
> * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```



```
> * Restarting with stat
> * Debugger is active!

> * Debugger PIN: 257-977-014
```

## Step 4: Testing Web Server / Service

For this we are going to use various methods to test the web server.

### 4.1 - Browser (Simplest)

In a browser go to <http://127.0.0.1:5000/>

You should see 'hello world' response.

### 4.2 - Command Line

This works best on Linux / Mac environments where we have good toolset installed.

#### url

Curl is old-school.

```
$ curl http://127.0.0.1:5000/
```

#### httpie

This is more modern version.

<https://github.com/jakubroztocil/httpie>

```
$ pip install httpie
```

#### Usage

```
$ http http://127.0.0.1:5000/
```

```
HTTP/1.0 200 OK
  Content-Length: 13

  Content-Type: text/html; charset=utf-8

  Date: Wed, 18 Apr 2018 18:29:49 GMT
```

```
Server: Werkzeug/0.14.1 Python/3.6.4
```

```
Hello, World!
```

### **4.3 - Rest Clients**

These browser extensions are really great to test web services very easily.

You can try the following Chrome extensions:

- Advanced Rest Client
- Postman
- Rest client

To install go to this URL:

<https://chrome.google.com/webstore/category/extensions>

Go ahead and install one of the clients and try to go to the URL.

<http://127.0.0.1:5000/>

### **4.4: Python program**

And finally, we need to be able to test our webservice from Python.

#### **Option 1: Requests library**

<http://docs.python-requests.org/en/latest/>

Try the following code:

```
import requests
import pprint

r = requests.get('http://127.0.0.1:5000/')
print("status ", r.status_code)
print()
print("headers \n", pprint.pformat(r.headers, indent=4))
```

```
print()

print("content:\n", r.text)

print()

print("content as JSON:\n", r.json())

print()
```

## Option 2: urllib2 library

<https://github.com/httplib2/httplib2>

Install it as:

```
$ pip install httplib2
```

And try the following python code.

Then try going to google.com to see the response.

```
import httplib2

from pprint import pprint

h = httplib2.Http('.cache')

response, content = h.request('http://127.0.0.1:5000/')

#response, content = h.request('http://www.google.com/')

print("-----headers-----")

pprint(response)

print('-----content-----')

pprint(content)
```

Done!

## Lab: Open Source – RESTful Time Tracker Web Service

### Overview

**Time: 15-30 Minutes**

In this lab, you will:

- Use your setup to create and deploy web services.

We are going to build a web service that keeps track of time spent on various tasks.

We are keeping our tasks as a simple list in memory. In reality, you will store this into a database.

```
tasks = [
    {
        'id': 1,
        'name': u'fixing bug #1',
        'time_spent': 20 # in mins
    },
    {
        'id': 2,
        'name': u'customer support',
        'time_spent': 30
    }
]
```

We will build a web server and a client.

### Step 1: Run Web Server

Inspect this file: **time-tracker-webservice.py**

This is our web service server.

We are going to build up the service by fixing TODO items in the code.

We are building REST api calls by using annotations as follows:

```
import flask

tasks = [
    {
        'id': 1,
        'name': u'fixing bug #1',
        'time_spent': 20 # in mins
    },
    {
        'id': 2,
        'name': u'customer support',
        'time_spent': 30
    }
]

@app.route('/tasks/list', methods=['GET'])
def get_tasks_list():
    ret_value = {'status' : 'ok',\
                 'tasks' : tasks}
    return flask.jsonify(ret_value)
```

Launch the server from a terminal, as it is a long running process.

```
$ cd 300-Lab-Files/REST-labs
$ python ./time-tracker-webservice.py
```

Go to url <http://127.0.0.1:5000/> to make sure server is running.

## Step 2: Client

We will use httpie for our testing.

Install httpie as follows:

```
$ pip install httpie
```

## Step 3: Quick Test

Execute the following from command line:

```
$ http http://127.0.0.1:5000
```

## Step 4: List Tasks

```
$ http http://127.0.0.1:5000/tasks/
$ http http://127.0.0.1:5000/tasks/list
```

## Step 5: GET

```
$ http http://127.0.0.1:5000/task/1
## Get a task that doesn't exist
$ http http://127.0.0.1:5000/task/100
## Get a task that doesn't exist
```

```
$ http http://127.0.0.1:5000/task/xyz
```

**Step 6: POST**

```
$ http POST http://127.0.0.1:5000/task/new name='new task'
```

Step 7: UPDATE (PUT / POST)

```
$ http POST http://127.0.0.1:5000/task/update/1 time=10
```

**Step 8: DELETE**

```
$ http DELETE http://127.0.0.1:5000/task/delete/1
```

## Discussion: Analyzing an Application for Resiliency

### Overview

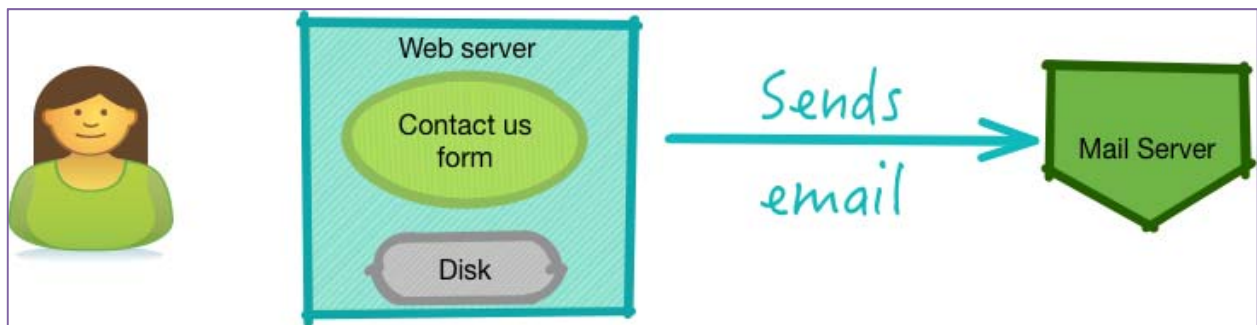
Time: 10-20 Minutes

In this exercise, we will examine an application for potential failure scenarios.

### Scenario: A form mailer application.

We have the web form: **Contact US**.

**Contact Us** collects information and sends that information, in an email, to the company.



### Step 1: What are the failure scenarios you can think of?

Here are some things to take into consideration:

- What if the remote mail server connection is refused?
- What if the web server fails?
- What if the web server disk becomes full?



## Discussion: Designing a Resilience Application

### Overview

**Time: 20-30 Minutes**

In this exercise, we are going to design a resilience IOT (Internet of Things) application.

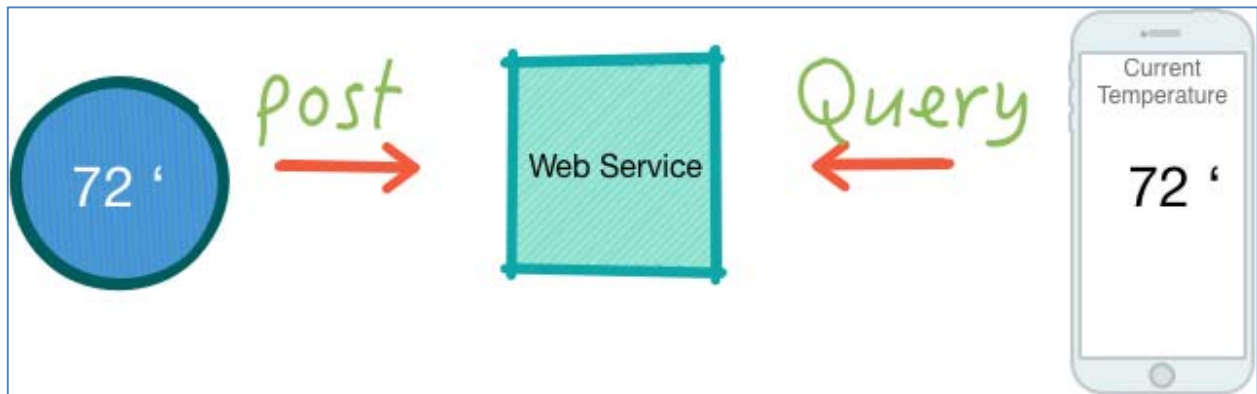
### Scenario: Our IOT application.

Imagine we have an IOT device – a thermostat - at home.

The IOT device reports temperature data to a web service.

We also have a web service that the user can access to see the current temperature at their home, and keeps the historical temperature readings.

Here is an overview:



### Step 1: Form Groups

Self-organize into 3 or 4 groups.

Sit together so you can discuss the scenario freely.

### Step 2: Group Design

Design a system that is highly available and resilient.

Think of all the components that could be needed for a service of this nature.

Do we need a datastore? What kind? etc.

### Step 3: Present Your Design to the Class

Each group will present their design.

We will learn together!

## Lab: Limits Service

### Overview

**Time: 30 Minutes**

In this lab, you will:

- Develop limits service to read configuration from a properties file.

### Step 1: Bootstrap Simple Rest Service

Open URL <https://start.spring.io/>

Pick latest boot version and add dependencies as shown below.

**SPRING INITIALIZR** bootstrap your application now

Generate a Maven Project with Java and Spring Boot 2.0.2

#### Project Metadata

Artifact coordinates

Group

com.optum.microservices

Artifact

limits-service

#### Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Web, Security, JPA, Actuator, Devtools...

Selected Dependencies

Web
DevTools
Actuator
Config Client

Generate Project

Don't know what to look for? Want more options? [Switch to the full version.](#)

After clicking **Generate Project**, a zip file will be downloaded, unzip the file and keep it in a folder of your choosing.

Open this project in your favorite IDE. The first time setup takes time for Spring Boot.

Run com.optum.microservices.limitsservice.LimitsServiceApplication to assure service is working fine. Output results after running should be similar to the below.

```
Started LimitsServiceApplication in 8.917 seconds (JVM running for 10.546)
```

## Step 2: Create a Bean to Serve Data through Rest

Create a bean to return a value from the controller.

com.optum.microservices.bean.LimitConfiguration

```
public class LimitConfiguration {

    private int maximum;

    private int minimum;

    protected LimitConfiguration() {

    }

    public LimitConfiguration(int maximum, int minimum) {

        this.maximum = maximum;

        this.minimum = minimum;

    }

    public int getMaximum() {

        return maximum;

    }

    public int getMinimum() {

        return minimum;

    }

}
```



### Step 3: Adding Configuration to Properties

```
spring.application.name=limits-service

limits-service.minimum=1

limits-service.maximum=999
```

### Step 4: Create Property Configuration Reader

Add `com.optum.microservices.limitsservice.Configuration` class.

Add the fields `maximum` and `minimum`, the same as was added in `application.properties` and generate getters/setters.

Also, add the prefix we have used in the properties `"limits-service"` to the annotation `ConfigurationProperties`.

```
@org.springframework.context.annotation.Configuration

@ConfigurationProperties("limits-service")

public class Configuration {

    private int maximum;

    private int minimum;

    public int getMaximum() {

        return maximum;

    }

    public void setMaximum(int maximum) {

        this.maximum = maximum;

    }

}
```

```
public int getMinimum() {

return minimum;

}

public void setMinimum(int minimum) {

this.minimum = minimum;

}

}
```

### Step 5: Create Rest Controller to Handle Incoming Requests

Create a class `com.optum.microservices.limitsservice.LimitsConfigurationController`.

Create a public method `retrieveLimitsFromConfiguration` and annotate this method for `GetMapping` as `/limits`.

Annotate the class with the annotation **RestController**.

```
@RestController

public class LimitsConfigurationController {

    @GetMapping("/limits")

    public LimitConfiguration retrieveLimitsFromConfiguration() {

    }

}
```

Autowire this class with an instance of configuration reader to read the value from the properties file.

```
@RestController
```

```
public class LimitsConfigurationController {

    @Autowired

    private Configuration configuration;

    @GetMapping("/limits")

    public LimitConfiguration retrieveLimitsFromConfiguration() {

        return new LimitConfiguration(configuration.getMaximum(),
configuration.getMinimum());

    }

}
```

Restart your application and open this URL <http://localhost:8080/limits>

```
{"maximum":999,"minimum":1}
```

## Lab: Spring Cloud Config Server

### Overview

**Time: 15 Minutes**

In this lab, you will:

- Generate and setup Spring Cloud Config Server to hold maximum and minimum configuration properties and will run on port 8888. This service will serve limits-service with properties.

### Step 1: Setup Git Local Repo with Config File

- Create a folder anywhere on your machine and perform git init.
- Add this folder as an external resource (optional step).
- Create a properties file inside this folder, since this will be your configuration server. Keep in mind that the file name has to be same as the application name. That is the application for which we are creating the configuration i.e., limits-service.properties.
- Copy the below entries from application.properties of limits-service to the new config project. Then the limits-service will get configuration from spring cloud config.

```
limits-service.minimum=1
limits-service.maximum=999
```

- Run **git init** inside the new folder and add the newly created file.
- Commit this file locally, no need to push.

### Step 2: Generate and Setup Spring Cloud Config Server on Port 8888

#### Setup

Create a new project from <https://start.spring.io/> and add Config Server as a dependency, as shown below.



Generate a Maven Project with Java and Spring Boot 2.0.2

### Project Metadata

Artifact coordinates

Group

Artifact

### Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Selected Dependencies

DevTools
Config Server

[Generate Project](#)

Don't know what to look for? Want more options? [Switch to the full version.](#)

- Click **Generate Project** and open with IDE.
- Make the below entries to **application.properties**:

```
spring.application.name=spring-cloud-config-server
server.port=8888
spring.cloud.config.server.git.uri=file://{git-localconfig-repo-path}
```

- Change `git-localconfig-repo-path` with full path on your machine, you can also use the github file.
- Add **@EnableConfigServer** to `com.optum.microservices.springcloudconfigserver.SpringCloudConfigServerApplication`

Run spring cloud config server and open URL <http://localhost:8888/limits-service/default>

```
{"name": "limits-
service", "profiles": [ "default"], "label": null, "version": "d7bbc8de7
b6d61e4da2f8f20dcfea8b2"}
```

### Step 3: Change to Limits-Service to Read from Spring Cloud Config Server

- Rename **application.properties** inside limits-service to **bootstrap.properties**
- Add `spring.cloud.config.uri=http://localhost:8888` to **bootstrap.properties**
- Run the application and now, **limit service** will read the property from spring-cloud-config-server running on port 8888.

## Lab: Pulling a Container

### Overview

When we run a docker container, we very rarely ever start from scratch. Docker provides us an OS kernel (such as Linux), but generally does **not** provide us with any of the other aspects of the OS we think of.

### Time: 30 Minutes

In this lab, you will be able to:

- Perform multiple Docker functions.

### Step 1: Do a Docker Pull

A `docker pull` will pull down a public repo from dockerhub. Think of it like a `git pull`.

We are going to use a lightweight Linux container called alpine. You can read more about it here: [https://hub.docker.com/r/\\_/alpine/](https://hub.docker.com/r/_/alpine/). It is only 4MB in size, making it perfect for a container.

```
docker pull alpine
```

You should get the following results:

```
Using default tag: latest
latest: Pulling from library/alpine
ff3a5c916c92: Pull complete
Digest:
sha256:7df6db5aa61ae9480f52f0b3a06a140ab98d427f86d8d5de0bedab9b8df6b1c
0
Status: Downloaded newer image for alpine:latest
```

### Step 2: See the docker image we downloaded

```
docker images
```

| REPOSITORY  | TAG    | IMAGE ID     | CREATED      |
|-------------|--------|--------------|--------------|
| hello-world | latest | e38bc07ac18e | 5 weeks ago  |
| alpine      | latest | 3fd9065eaf02 | 4 months ago |
| SIZE        |        |              |              |
| 1.85kB      |        |              |              |
| 4.15MB      |        |              |              |

There you see it, Alpine. But what do we do with it?

### Step 3: Run the Container

How do we run the container?

```
docker container run alpine ls
```

Running this at the root "/" directory, we see the directories there.

```
bin
dev
etc
home
lib
media
mnt
proc
root
run
sbin
srv
sys
tmp
usr
var
```

#### Step 4: Listing Containers

Let's try to list our container.

```
docker container ls
```

Resulting in the response below:

| CONTAINER ID | IMAGE | COMMAND | CREATED |
|--------------|-------|---------|---------|
| STATUS       | PORTS | NAMES   |         |

**Wait!** Where's our container? Well, our container isn't running at the moment. The command we gave it (ls) ended already. We can see the stopped container if we want like this.

```
docker container ls -a
```

| CONTAINER ID    | IMAGE       | COMMAND            | CREATED    |
|-----------------|-------------|--------------------|------------|
| STATUS          | PORTS       | NAMES              |            |
| d060f9a50c7c    | alpine      | "ls"               | About a    |
| minute ago      | Exited (0)  | About a minute ago |            |
| admirer_keldysh |             |                    |            |
| 887030cdbc7e    | hello-world | "/hello"           | 26 minutes |
| ago             | Exited (0)  | 26 minutes ago     |            |
| nostalgic_lewin |             |                    |            |

There we go. We have two containers, both stopped. Remember hello world? That's there. Just stopped.

Wait, should we keep creating all these containers? You can stop it from being created, if you just want to perform the one-and-done command.

```
docker container --rm ls -l
```

**\*\* TO DO:** See what containers we have. Does the new one show up?  
It will display the following message:

```
unknown flag: --rm
```

See 'docker container --help'.

## Step 5: Interactive shells

Wait, how do I `ssh` my container? Well, you don't usually do it that way. For one thing, your container isn't actually running right now, so if you try to `ssh` it won't respond. What you probably want is an interactive shell.

How do I do that?

```
docker container run -it --rm alpine /bin/ash
```

What does this mean?

- \* `-i` : interactive mode
- \* `-t` : terminal mode
- \* `--rm` : remove container after we are done
- \* `/bin/ash` : bash is big and needs to be installed. `ash` (almquist shell) is small. We also have old-school `sh`.

Here are the results with a few commands:

```
/ # echo hello
hello
/ # cd
~ # pwd
/root
~ # echo bye!
bye!
~ # exit
```

**\*\* TO DO:** You try some of your own commands.

## Summary

So, what's the point? That we can run a mini size linux? Well, we are about to see how we can build on what we have learned so far. This was a good place to start, and we needed some basic commands to build on.

## Lab: Manipulating a Container

### Overview

**Time: 30 Minutes**

In this lab, you will be able to:

- Manipulate our containers.

### Step 1: Start up a new alpine container

```
docker container run -it alpine
```

You will find yourself a root prompt in the new container. Try executing the command "ls /"

```
# ls /

bin  dev  etc  home  lib  media  mnt  proc
root run /sbin  srv  sys  tmp  usr  var
```

### Step 2: Delete all files in the container

Now, let's delete all files in the container. This is very dangerous, you must make sure you are doing this **IN THE CONTAINER**, and not on a root prompt on your local system or your cloud VM.

```
# rm -rf /
```

#### **DO NOT TRY THIS AT HOME!!**

Now, let's see if the files are gone.

```
# ls

/bin/sh ls: not found
```

Everything's gone. You've trashed your system. Congratulations!

**Exit** out of your container by hitting **CTRL-D**.

Now, let's see what happens if we run our container again.

```
docker container run -it alpine
```

Ok, let's see if our files are there.

```
ls -l

bin  dev  etc  home  lib  media  mnt  proc
root run /sbin  srv  sys  tmp  usr  var
```

**\*\* Why are all the files back after we deleted them all? \*\***

Go ahead and exit by hitting **CTRL-D** or typing **exit**.

### Step 3: Deleting Containers

Now, see if you can see your command:

```
docker container ls -as
```

| CONTAINER ID  | IMAGE  | COMMAND        | CREATED       | STATUS     |
|---------------|--------|----------------|---------------|------------|
| PORTS         | NAMES  | SIZE           |               |            |
| c183852a7215  | alpine | "/bin/sh"      | 6 minutes ago | Exited (0) |
| 2 minutes ago |        | adoring_joliot | 5B (virtual   | 4.15MB)    |

Notice the size around 5 bytes. Not exactly taking up a lot of space. The container IMAGE takes up around 4.15MB.

But, we may want to clean up our images anyway. We can use `docker container rm` for that.

```
docker container rm <container-id>
```

Now you can run:

```
docker container rm ls -as
```

| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS |
|--------------|-------|---------|---------|--------|
| PORTS        | NAMES | SIZE    |         |        |

It shouldn't be there anymore.

What if you want to run the docker container and just have it auto-delete after you are done.

```

Applications Places System
centos@ip-172-31-4-226:~
File Edit View Search Terminal Tabs Help
centos@ip-172-31-4-226:~
[centos@ip-172-31-4-226 ~]$ sudo docker pull alpine
Using default tag: latest
latest: Pulling from library/alpine
8e3b11ec2a2: Pull complete
Digest: sha256:78430763440f5040220d6ad703798fd8593a0918d06d3ce38c6c93be117e430
Status: Downloaded newer image for alpine:latest
[centos@ip-172-31-4-226 ~]$ sudo docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
alpine               latest          11c0b38bc3c    12 days ago     4.41MB
[centos@ip-172-31-4-226 ~]$ docker container run alpine ls
docker: Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post http://%2Fvar%2Frun%2Fdocker.sock/v1.38/containers/create: dial unix /var/run/docker.sock: connect: permission denied.
See 'docker run --help'.
[centos@ip-172-31-4-226 ~]$ sudo docker container run alpine ls
-bash: sudodocker: command not found
[centos@ip-172-31-4-226 ~]$ sudo docker container run alpine ls
bin
dev
etc
home
lib
media
mnt
proc
root
run
sbin
srv
sys
tmp
usr
var
[centos@ip-172-31-4-226 ~]$ docker container ls
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1.38/containers/json: dial unix /var/run/docker.sock: connect: permission denied
[centos@ip-172-31-4-226 ~]$ sudo docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS              NAMES
[centos@ip-172-31-4-226 ~]$ sudo docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS              NAMES
807ad7521475        alpine              "ls"                    57 seconds ago     Exited (0) 56 seconds ago              quirkky_newton
[centos@ip-172-31-4-226 ~]$ docker run hello-world
docker: Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post http://%2Fvar%2Frun%2Fdocker.sock/v1.38/containers/create: dial unix /var/run/docker.sock: connect: permission denied.
See 'docker run --help'.
[centos@ip-172-31-4-226 ~]$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
9db2c6a6cae8: Pull complete
Digest: sha256:4b8ff392a12e99e17784bd3c9a8b1fa3299cac44aca35a85c90c5e3c7afacdc
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.

```

Docker pull and docker images.

Run and List the containers and Interactive Shells.

Run the Container and remove all the files in it.

Confidential and proprietary | Optum





```

centos@ip-172-31-4-226:~$ rm: can't remove /sys/hypervisor/properties/pagelsize: Read-only file system
rm: can't remove /sys/hypervisor/properties/changeset: Read-only file system
rm: can't remove /sys/hypervisor/properties/virtual_start: Read-only file system
rm: can't remove /sys/hypervisor/properties/features: Read-only file system
rm: can't remove /sys/hypervisor/properties/capabilities: Read-only file system
rm: can't remove /sys/hypervisor/properties: Read-only file system
rm: can't remove /sys/hypervisor/version/extra: Read-only file system
rm: can't remove /sys/hypervisor/version/major: Read-only file system
rm: can't remove /sys/hypervisor/version/minor: Read-only file system
rm: can't remove /sys/hypervisor/version: Read-only file system
rm: can't remove /sys/hypervisor: Read-only file system
rm: can't remove /sys: Resource busy
rm: can't remove /: Resource busy
# ls
/bin/sh: ls: not found
# [centos@ip-172-31-4-226 ~]$
[centos@ip-172-31-4-226 ~]$ sudo docker container run -it alpine
# # ls -l
total 8
drwxr-xr-x 2 root root 4096 Jul 5 14:47 bin
drwxr-xr-x 5 root root 360 Jul 19 06:03 dev
drwxr-xr-x 1 root root 66 Jul 19 06:03 etc
drwxr-xr-x 2 root root 6 Jul 5 14:47 home
drwxr-xr-x 5 root root 278 Jul 5 14:47 lib
drwxr-xr-x 5 root root 44 Jul 5 14:47 media
drwxr-xr-x 2 root root 6 Jul 5 14:47 mnt
dr-xr-xr-x 102 root root 0 Jul 19 06:03 proc
drwxr-xr-x 1 root root 26 Jul 19 06:03 root
drwxr-xr-x 2 root root 6 Jul 5 14:47 run
drwxr-xr-x 2 root root 4096 Jul 5 14:47 sbin
drwxr-xr-x 2 root root 6 Jul 5 14:47 srv
dr-xr-xr-x 13 root root 0 Jul 19 04:54 sys
drwxrwxrwt 2 root root 6 Jul 5 14:47 tmp
drwxr-xr-x 7 root root 66 Jul 5 14:47 usr
drwxr-xr-x 11 root root 125 Jul 5 14:47 var
# [centos@ip-172-31-4-226 ~]$
[centos@ip-172-31-4-226 ~]$ docker container ls -as
#
Get permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get http://172.17.0.1:2375/v1.38/containers/json?all=1&size=1: dial unix /var/run/docker.sock: connect: permission denied
#
[centos@ip-172-31-4-226 ~]$ sudo docker container ls -as
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES                SIZE
2a1739f8093f        alpine              "/bin/sh"           30 seconds ago      Exited (0) 20 seconds ago                  thirsty_rosalind     60 (virtual 4.41MB)
a198863f3074        alpine              "/bin/sh"           2 minutes ago       Exited (127) About a minute ago            pedantic_payne       60 (virtual 4.41MB)
e1dc0e2ed5a5        alpine              "/bin/sh"           9 minutes ago       Exited (0) 9 minutes ago                   adoring_chatterjee   60 (virtual 4.41MB)
0d10baa159f5        hello-world         "/hello"            17 minutes ago      Exited (0) 17 minutes ago                   loving_poincaré      60 (virtual 1.85KB)
807ad7521475        alpine              "ls"                19 minutes ago      Exited (0) 19 minutes ago                   quirky_newton         60 (virtual 4.41MB)
[centos@ip-172-31-4-226 ~]$ sudo docker container rm 2a1739f8093f
2a1739f8093f
[centos@ip-172-31-4-226 ~]$ sudo docker container ls -as
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES                SIZE
a198863f3074        alpine              "/bin/sh"           3 minutes ago       Exited (127) 2 minutes ago                  pedantic_payne       60 (virtual 4.41MB)
e1dc0e2ed5a5        alpine              "/bin/sh"           10 minutes ago      Exited (0) 10 minutes ago                  adoring_chatterjee   60 (virtual 4.41MB)
0d10baa159f5        hello-world         "/hello"            19 minutes ago      Exited (0) 19 minutes ago                   loving_poincaré      60 (virtual 1.85KB)
807ad7521475        alpine              "ls"                20 minutes ago      Exited (0) 20 minutes ago                   quirky_newton         60 (virtual 4.41MB)
[centos@ip-172-31-4-226 ~]$ Write failed: Broken pipe

```

Deleting the Containers.

## Lab: Ports

### Overview

So, we have had fun executing Linux commands on our container. How do we do networking on our container? Networking is going to be picked up by our host. So, we need to know specify what ports we want forwarded to the container.

### Time: 30 Minutes

In this lab, you will be able to:

- Network with containers.

Here's how we do port forwarding.

```
docker container run -p HOSTPORT:CONTAINERPORT
```

### Step 1: Run

We run this, and this means port 8002 on the host is pointed to port 8002, which is mapped to port 80 on the container.

```
docker container run -p 8000:80 nginx
docker container run -p 8002:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
f2aa67a397c4: Pull complete
3c091c23e29d: Pull complete
4a99993b8636: Pull complete
Digest:
sha256:0fb320e2a1b1620b4905facb3447e3d84ad36da0b2c8aa8fe3a5a81d1187b88
4
Status: Downloaded newer image for nginx:latest
172.17.0.1 - - [24/May/2018:04:25:57 +0000] "GET / HTTP/1.1" 200 612
 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
 Gecko) Chrome/65.0.3325.181 Safari/537.36" "-"
2018/05/24 04:25:57 [error] 6#6: *1 open()
"/usr/share/nginx/html/favicon.ico" failed (2: No such file or
directory), client: 172.17.0.1, server: localhost, request: "GET
/favicon.ico HTTP/1.1", host: "localhost:8002", referrer:
"http://localhost:8002/"
172.17.0.1 - - [24/May/2018:04:25:57 +0000] "GET /favicon.ico
HTTP/1.1" 404 572 "http://localhost:8002/" "Mozilla/5.0 (X11; Linux
x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181
Safari/537.36" "-"
```

## Step 2: Go to Browser

Open up a browser and go to the following url:

```
http://localhost:8002
```

If you are running locally.

If you are running in the cloud, go to your IP address:

```
http://YOURIPADDRESS:8002
```

You should see the following:

```
Welcome to nginx!

If you see this page...

Thank you for using nginx.
```

## Step 3: Stop the container

Press **Ctrl-C** to stop the container.

Now, try going back to the browser and re-loading. Do you see the page?

The page doesn't work while the container is stopped!

## Step 4: Run the container in the background

```
docker container run -p 8002:80 -d nginx
```

You will see the new container id copied to the screen, as this runs in the background.

```
ffcee5395fc4fc4ca97c48b03b3510ec973c6fa8601e9b299e306a6d36f6ff74
```

## Step 5: Go back to the page

As in step 2, go back to the page. You should see nginx running again.

## Step 6: Stop the container

```
docker stop <paste-container-id-here>
```

It should stop your container. If it doesn't recognize the container id you can paste "sudo docker ps" to get the list of container ids.

```

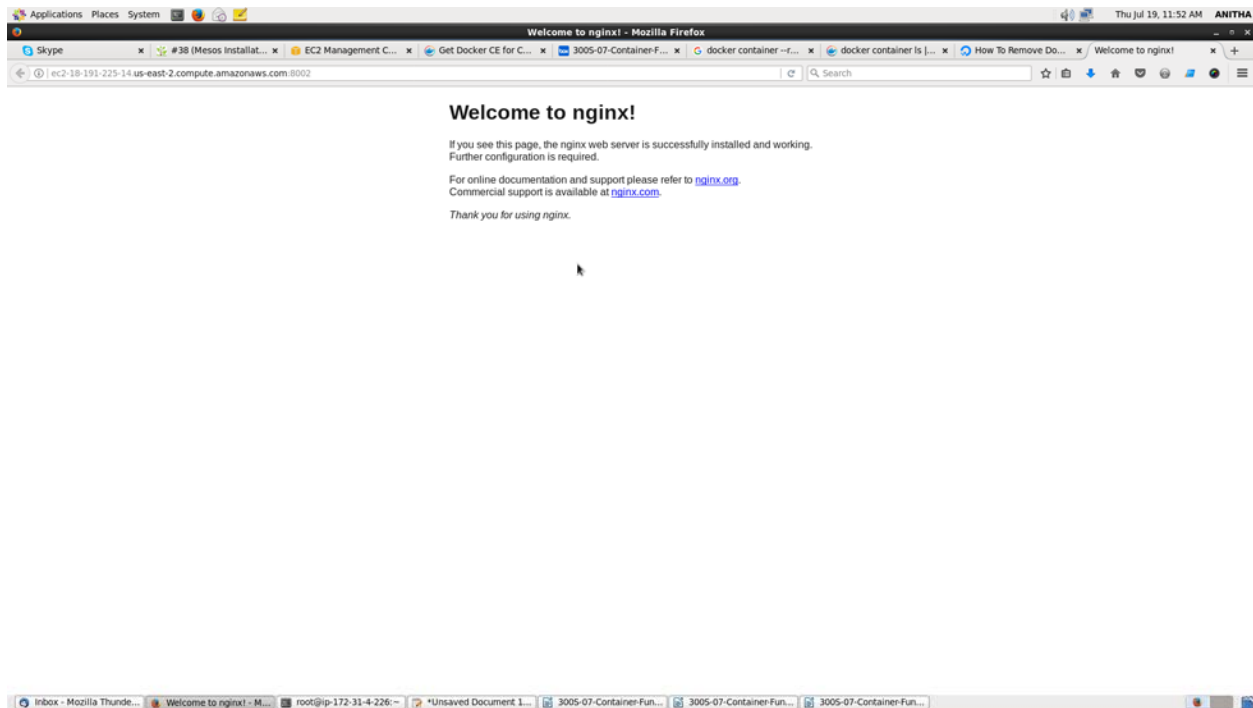
Applications Places System root@ip-172-31-4-226:~ Thu Jul 19, 4:00 PM ANITHA
File Edit View Search Terminal Tabs Help
centos@ip-172-31-4-226:~ root@ip-172-31-4-226:~ anitha@anitha:~/figo_dev_compressed
[anitha@anitha ~]$ ssh -i access centos@ec2-18-191-225-14.us-east-2.compute.amazonaws.com
Last login: Thu Jul 19 05:02:06 2018 from 122.165.130.117
[centos@ip-172-31-4-226 ~]$ sudo docker container run -p 8002:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
be8881be156: Pull complete
f2127ed9664f: Pull complete
54ff137eb1b2: Pull complete
Digest: sha256:4a5573037f358b6cdfaf2f3e8a9c33a5cf11bcd1675ca72ca76f8e5bd770d0682
Status: Downloaded newer image for nginx:latest
122.165.130.117 - - [19/Jul/2018:06:14:37 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0" "-"
122.165.130.117 - - [19/Jul/2018:06:14:37 +0000] "GET /favicon.ico HTTP/1.1" 404 169 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0" "-"
2018/07/19 06:14:37 [error] 6#6: *1 open() "/usr/share/nginx/html/favicon.ico" failed (2: No such file or directory), client: 122.165.130.117, server: localhost, request: "GET /favicon.ico HTTP/1.1", host: "ec2-18-191-225-14.us-east-2.co
mpute.amazonaws.com:8002"
122.165.130.117 - - [19/Jul/2018:06:14:38 +0000] "GET /favicon.ico HTTP/1.1" 404 169 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0" "-"
2018/07/19 06:14:38 [error] 6#6: *1 open() "/usr/share/nginx/html/favicon.ico" failed (2: No such file or directory), client: 122.165.130.117, server: localhost, request: "GET /favicon.ico HTTP/1.1", host: "ec2-18-191-225-14.us-east-2.co
mpute.amazonaws.com:8002"
122.165.130.117 - - [19/Jul/2018:06:15:03 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0" "-"
122.165.130.117 - - [19/Jul/2018:06:15:10 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0" "-"
[centos@ip-172-31-4-226 ~]$
[centos@ip-172-31-4-226 ~]$ sudo docker container run -p 8002:80 -d nginx
598228e5cd09d1d4c82b0ca176f09c241859bb3c9d27e71e8f29aed5711e7b74
[centos@ip-172-31-4-226 ~]$ sudo docker stop 598228e5cd09d1d4c82b0ca176f09c241859bb3c9d27e71e8f29aed5711e7b74
598228e5cd09d1d4c82b0ca176f09c241859bb3c9d27e71e8f29aed5711e7b74
[centos@ip-172-31-4-226 ~]$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS             PORTS             NAMES
[centos@ip-172-31-4-226 ~]$ sudo su -
[root@ip-172-31-4-226 ~]# docker container run -p 8002:80 nginx
122.165.130.117 - - [19/Jul/2018:06:18:38 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0" "-"
[root@ip-172-31-4-226 ~]# docker container run -p 8002:80 -d nginx
0a91e23e3d5997686df87b1d6cf6092657d54ad7e49dad462326992e34b9538f
0a91e23e3d5997686df87b1d6cf6092657d54ad7e49dad462326992e34b9538f
[root@ip-172-31-4-226 ~]# docker stop 0a91e23e3d5997686df87b1d6cf6092657d54ad7e49dad462326992e34b9538f
0a91e23e3d5997686df87b1d6cf6092657d54ad7e49dad462326992e34b9538f
[root@ip-172-31-4-226 ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS             PORTS             NAMES
[root@ip-172-31-4-226 ~]# docker container run -p 8002:80 -d nginx
1d2179822070005a35f41023a30e5464a734d0481df3409e02f627f535f8c4
[root@ip-172-31-4-226 ~]# docker stop 1d2179822070005a35f41023a30e5464a734d0481df3409e02f627f535f8c4
1d2179822070005a35f41023a30e5464a734d0481df3409e02f627f535f8c4
[root@ip-172-31-4-226 ~]#
Message from syslog@ip-172-31-4-226 at Jul 19 06:22:19 ...
kernel:unregister_netdevice: waiting for lo to become free. Usage count = 1
^C
[root@ip-172-31-4-226 ~]# vi myfirstapp
[root@ip-172-31-4-226 ~]# ls
anaconda-ks.cfg  myfirstapp  original-ks.cfg
[root@ip-172-31-4-226 ~]# docker build -t myfirstapp
"docker build" requires exactly 1 argument.
See 'docker build --help'.

Usage: docker build [OPTIONS] PATH | URL | -

```

Run the container.

In the browser.



## Lab: Dockerfile

### Overview

**Time: 30 Minutes**

In this lab, you will be able to:

- Create a Python app using flask, and deploy it using docker.

### Instructions:

Here is the Dockerfile we are going to use:

```
# Use an official Python runtime as a parent image
FROM python:2.7-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
ADD . /app

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

Then create file “requirements.txt” and place the following contents into it.

```
Flask
Redis
```

Also create file “app.py” and place the following contents into it.

```
from flask import Flask
from redis import Redis, RedisError
import os
import socket

# Connect to Redis
redis = Redis(host="redis", db=0, socket_connect_timeout=2,
socket_timeout=2)

app = Flask(__name__)
```

```
@app.route("/")
def hello():
    try:
        visits = redis.incr("counter")
    except RedisError:
        visits = "<i>cannot connect to Redis, counter disabled</i>"
    html = "<h3>Hello {name}!</h3>" \
        "<b>Hostname:</b> {hostname}<br/>" \
        "<b>Visits:</b> {visits}"
    return html.format(name=os.getenv("NAME", "world"),
        hostname=socket.gethostname(), visits=visits)
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80)
```

Create a directory “newapp” and these 3 files are placed into this directory.  
Note what we're about to do. We are going to start with building the file and give it the name “mypythonnew”.

```
cd newapp
docker build -t mypythonnew
```

You will get a very long output, which will be Docker loading all of your Dockerfile commands onto your container.

You should look at your Dockerfile

```
Sending build context to Docker daemon   5.12kB
Step 1/7: FROM python:2.7-slim
2.7-slim: Pulling from library/python
be8881be8156: Already exists
e87541f2e904: Pull complete
Digest:
sha256:a3ae315a6bc8cd58b4b60bda9eac6640795359b4970230fc5bf4eef3de6a802
2
Status: Downloaded newer image for python:2.7-slim
---> 42967d04ddc5
Step 2/7: WORKDIR /app
---> Running in f848e7c9eef8
```



```
Removing intermediate container f848e7c9eef8
---> 5d5ce4911ed9
Step 3/7: ADD . /app
---> a3408e25d84f
Step 4/7 : RUN pip install --trusted-host pypi.python.org -r
requirements.txt
---> Running in a9c9057ac5c3
Collecting Flask (from -r requirements.txt (line 1))
  Downloading
https://files.pythonhosted.org/packages/7f/e7/08578774ed4536d3242b14da
cb4696386634607af824ea997202cd0edb4b/Flask-1.0.2-py2.py3-none-any.whl
(91kB)
```

I've snipped this for clarity but pay attention to what's going on.

## List your containers

```
docker image ls
```

| REPOSITORY  | TAG    | IMAGE ID     | CREATED    |
|-------------|--------|--------------|------------|
| mypythonnew | latest | cc01dd8497dd | 36 seconds |
| ago         | 132MB  |              |            |

## Run the container

```
docker container run -p 4000:80 mypythonnew
```

This will run our app. console output should look like the following

```
* Running on http://0.0.0.0:4000/ (Press CTRL+C to quit)
172.17.0.1 - - [24/May/2018 15:55:11] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [24/May/2018 15:55:13] "GET /favicon.ico HTTP/1.1" 404
-
```

## Go to browser

Open your browser and go to YOURMACHINE:4000. If you are running on localhost, then go to localhost:5000

You should see something like the following in your browser:

```
Hello World!
Hostname: c67c11a0399a
Visits: cannot connect to Redis, counter disabled
```

This indicates your Flask app is running properly. You can now close your container by typing control-c

## Docker Build

```

Applications Places System root@ip-172-31-4-226:~ Thu Jul 19, 4:08 PM ANITHA
File Edit View Search Terminal Tabs Help
centos@ip-172-31-4-226:~ root@ip-172-31-4-226:~ anitha@anitha:~/figo_dev_compressed
[root@ip-172-31-4-226 ~]# docker build -t mypythonnew .
Sending build context to Docker daemon 5.12kB
Step 1/7 : FROM python:2.7-slim
2.7-slim: Pulling from library/python
be0881be0156: Already exists
e075412e094: Pull complete
81dc5ba046c8: Pull complete
b411fa192c9: Pull complete
Digest: sha256:a3ae315a6c8cd58b4b6bd9eac6640795359b4970230fc3bf4ee73de6a8022
Status: Downloaded newer image for python:2.7-slim
--> 429670046dc3
Step 2/7 : WORKDIR /app
--> Running in f848e7c9eef8
Removing intermediate container f848e7c9eef8
--> 5d5ce491ed9
Step 3/7 : ADD . /app
--> a3408e250b4f
Step 4/7 : RUN pip install --trusted-host pypi.python.org -r requirements.txt
--> Running in a9c9057ac5c3
Collecting Flask (from -r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/7f/e7/08570774e4536d3242b14dcb4696386634607af824ea997202c0ed04b/Flask-1.0.2-py2.py3-none-any.whl (91kB)
Collecting Redis (from -r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/3b/f6/7a76333cf0b9251ecf49eff63501517184390977e4ffcf59f9c4428052/redis-2.10.6-py2.py3-none-any.whl (64kB)
Collecting itsdangerous==0.24 (from Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/dc/b4/a60bcd0a945c00f6d608d897513ab3f25b2f2bcfe1da221165194b2d4/itsdangerous-0.24.tar.gz (46kB)
Collecting Jinja2==2.10 (from Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/7f/ff/ae64bacdfc95f27a016a7bede8686763ba4d277a78ca76f32659220a731/Jinja2-2.10-py2.py3-none-any.whl (126kB)
Collecting Werkzeug==0.14 (from Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/20/c4/12e3e56473e52375aa29c4764e70d1b8f3efa6682b6f8daae04fe335243/Werkzeug-0.14.1-py2.py3-none-any.whl (322kB)
Collecting click==5.1 (from Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/34/c1/8806f99713d0d993c5366c362b2f908f18269f8d792aff1abfd700775a77/click-6.7-py2.py3-none-any.whl (71kB)
Collecting MarkupSafe==0.23 (from Jinja2==2.10->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/4d/de/32d741db316d8fbd76808226d37001ef7a448255de9699ab4bfc0df4172b/MarkupSafe-1.0.tar.gz
Building wheels for collected packages: itsdangerous, MarkupSafe
Running setup.py bdist_wheel for itsdangerous: started
Running setup.py bdist_wheel for itsdangerous: finished with status 'done'
Stored in directory: /root/.cache/pip/wheels/2c/4a/61/5599631c1554768c6290b08c02c7207317918374ca0827f1e5
Running setup.py bdist_wheel for MarkupSafe: started
Running setup.py bdist_wheel for MarkupSafe: finished with status 'done'
Stored in directory: /root/.cache/pip/wheels/33/56/20/ebe49a5c6127ffe1c5a632140b16596f9e64676768661e4e46
Successfully built itsdangerous MarkupSafe
Installing collected packages: itsdangerous, MarkupSafe, Jinja2, Werkzeug, click, Flask, Redis
Successfully installed Flask-1.0.2 Jinja2-2.10 MarkupSafe-1.0 Redis-2.10.6 Werkzeug-0.14.1 click-6.7 itsdangerous-0.24
Removing intermediate container a9c9057ac5c3
--> 2703446d08a2
Step 5/7 : EXPOSE 80
--> Running in 81807e6ac04e
Removing intermediate container 81807e6ac04e
--> 0f42dc941d2c
Step 6/7 : ENV NAME World
--> Running in 7fa0bea2900f
Removing intermediate container 7fa0bea2900f
--> a367b43ee006
Step 7/7 : CMD ["python", "app.py"]
--> Running in bda793c550e0

```

## Run the Container

[illegible]

## In the Browser

