Local_Dev_Example

A simple application to demonstrate deployment and program structure

What is this document?

To help you get started in using our tools, we created a very basic application that you can find here:

https://github.com/SMPirelli/local_dev_example

The Application is running here:

http://10.130.4.101/Local_Dev_Example/

This document will explain in detail how the application works, and how it was deployed on the server.

Assumptions

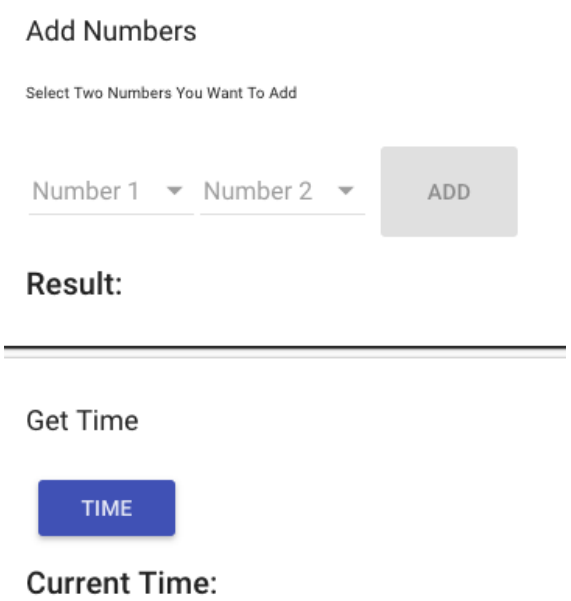
Some assumptions:

- You know what JSON is
- You've seen Python before, you know what functions are
- You have heard of Flask
- You've seen an Angular 1 Web App, using Angular Material
- You don't like Internet Explorer
- Backend, Frontend, RESTApi mean something to you (or you've at least heard of them)

Even without all of the above, it may well make sense (unless you like Internet Explorer).

The Application

It's a simple simple interface with two parts.



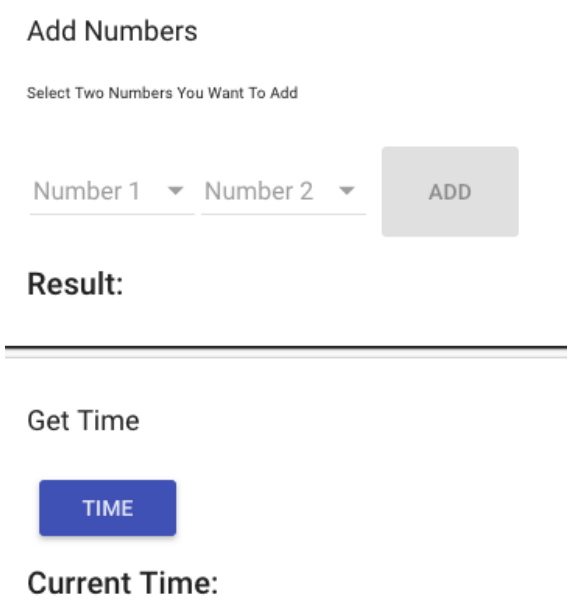
This part asks the user to choose two numbers. When two are chosen, the ADD button becomes active.

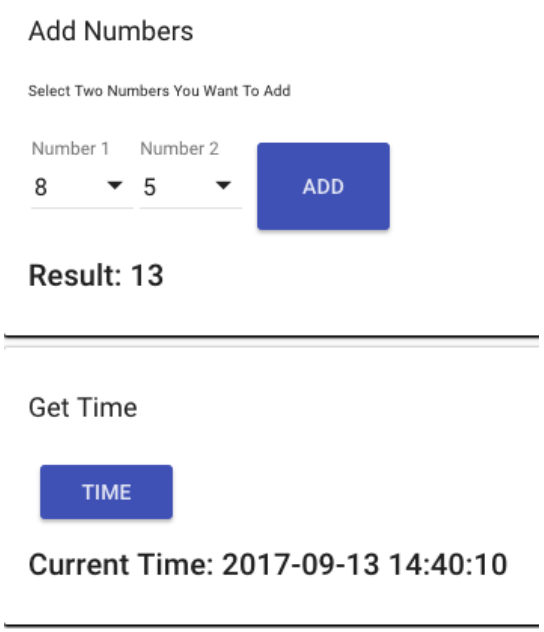
Upon Pressing 'ADD' the result of adding the two numbers together will be displayed.

Down here, the user can click the TIME button to get the current time (of the server running the app)

Action!

Here's a start to finish in all its glory.





‘Under the hood’

What actually happened?

Add Numbers

Select Two Numbers You Want To Add

Number 1 Number 2 ADD

At the start the ADD button is disabled, and the number choice lists are empty.

Add Numbers

Select Two Numbers You Want To Add

Number 1 Number 2

8 5 ADD

Result:

When you select a number in BOTH lists, the ADD button is enabled

The ADD button is clicked and a request is sent to the backend

The backend code reads the request, runs the code to add the numbers and returns the result

code.py

Add Numbers

Select Two Numbers You Want To Add

Number 1 Number 2

8 5 ADD

Result: 13

The front end receives the answer from the back and and displays the result, in this case ‘13’

Get Time

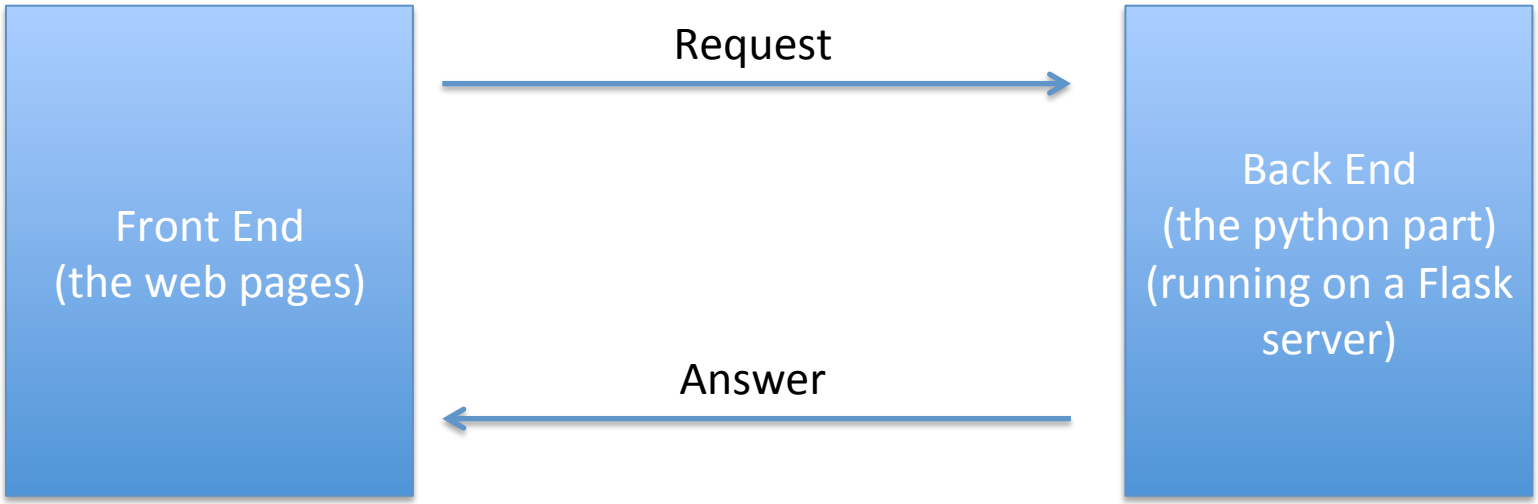
TIME

Current Time:

The Time button does much the same, instead asking the backend for the time

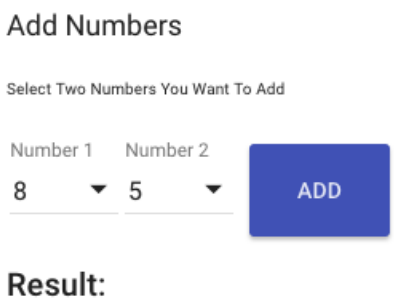
Concept

What was described in the previous slide is a concept seen in all our applications.

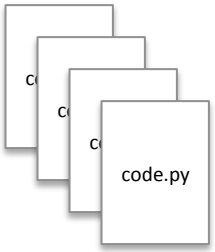


Concept

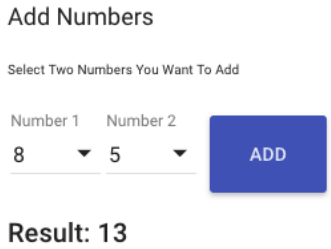
What was described in the previous slide is a concept seen in all our applications.



Request, please add 5 and 8



Calculate
5 + 8



Answer: 13

File Structure

On GITHUB, the application looks like this:

repository name for the application

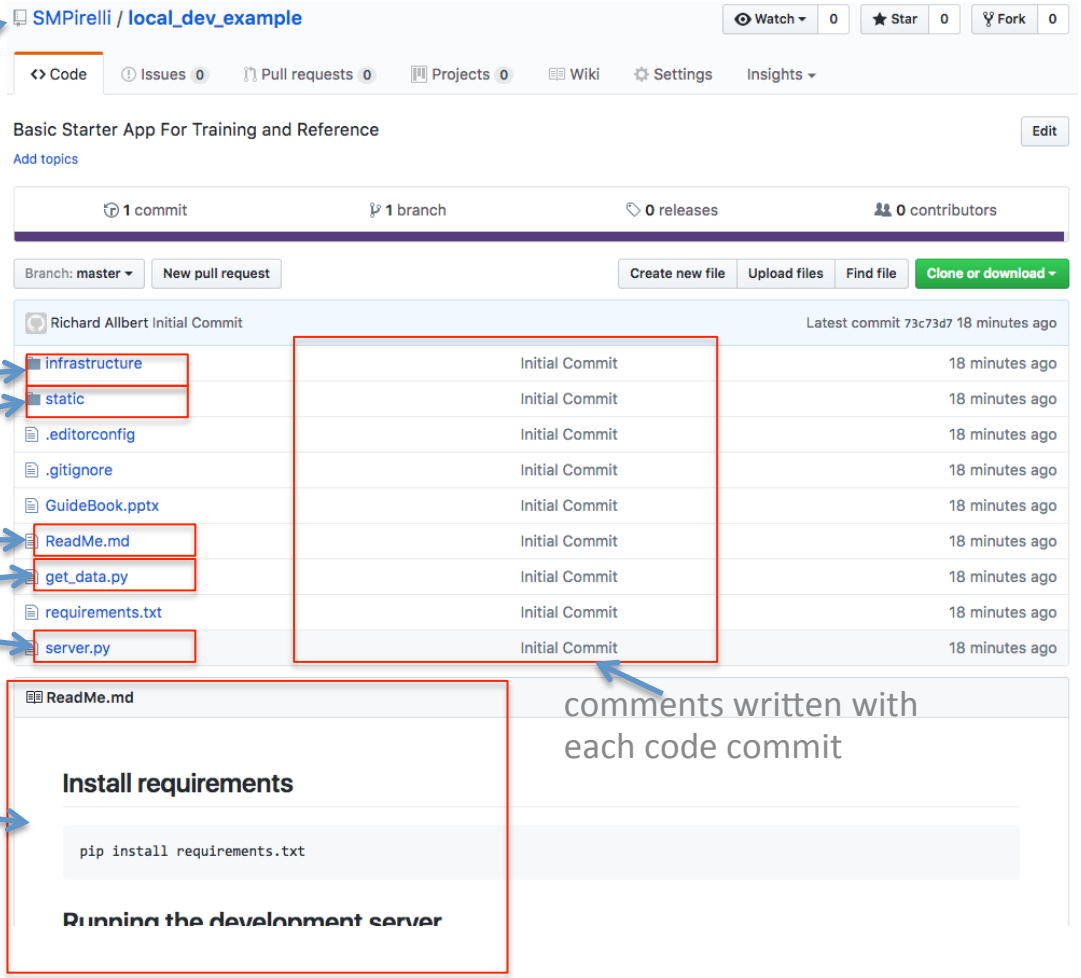
settings files for deployment

front end web application

important notes for this repo

backend python files

the ReadMe.md is automatically displayed here



comments written with each code commit

The Front End

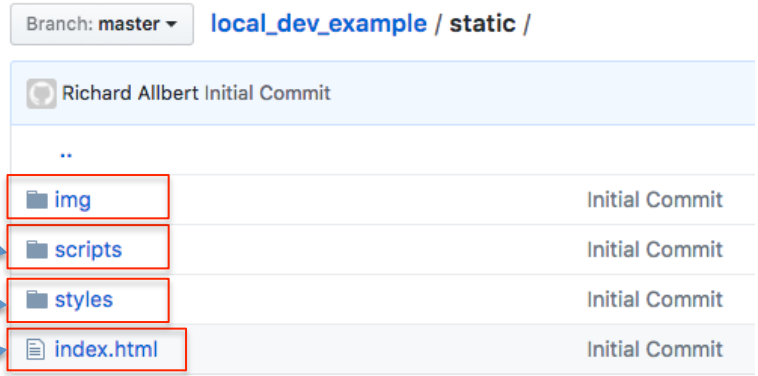
Inside /static/

images for toolbars

javscript files

style files (css)

the web page, always called index.html



Note, here we are looking at an app with one web page file, index.html. Multiple pages are covered in other documents (or will be). The structure you see in this document is the 'bare minimum'

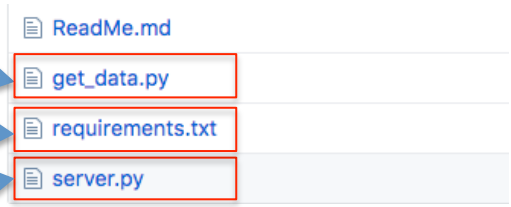
The Back End

Inside /

python code for the app

python libraries required for the application

the flask server python code



Infrastructure

Inside /infrastructure/

We have three files which contain settings to configure the application, and run the flask server

App configuration for Apache

Command to run the flask server (using gunicorn)

Configuration for a the process supervisor

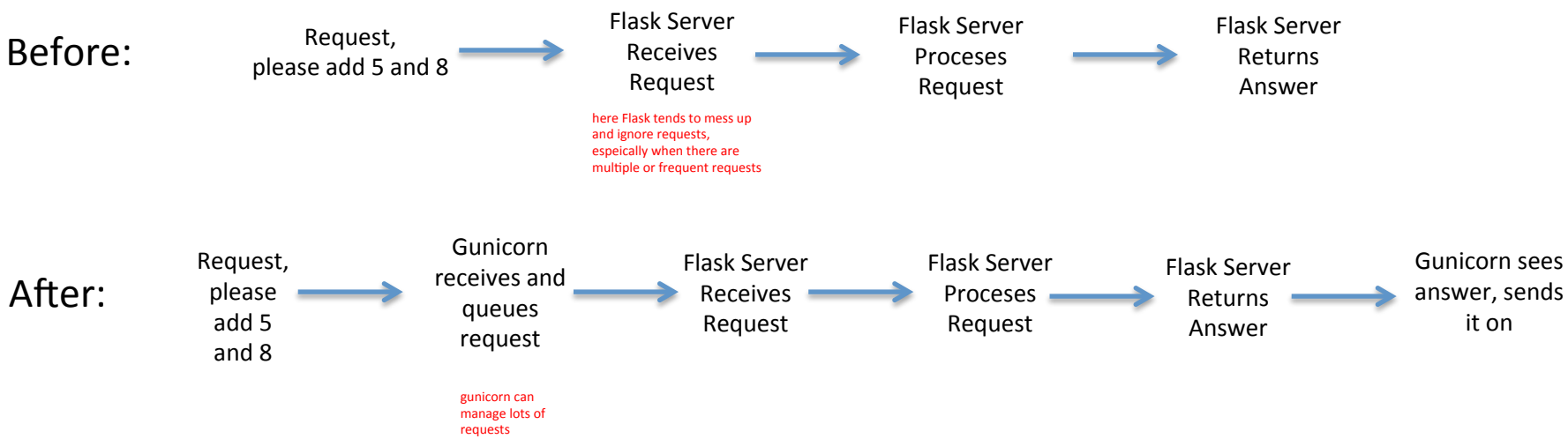
How To Deploy

How To Deploy

Our Flask Server

You can read online about Flask – it's a python server that responds to requests. We run our backend on a Flask server.

We use something called gunicorn to act as a gateway between the flask server and the request. The reason is that Flask does not handle lots of requests very well. gunicorn does.



How To Deploy

Our Flask Server

Here is our server code. For our deployment, we can see that the server listens for requests with two different urls.

/api/addition

/api/time

important is to stick to the naming convention of /api

This makes things easier and consistent between applications.

(the rest of the code we will cover later)

Branch: master ▾ local_dev_example / server.py

Richard Allbert Initial Commit

0 contributors

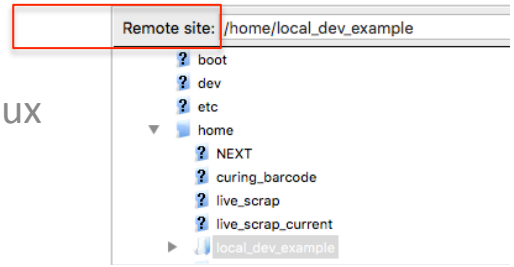
34 lines (22 sloc) | 726 Bytes

```python
from flask import Flask, request
from flask import Response
from get_data import add_numbers, get_time_string
import json

app = Flask(__name__)


# /api/addition?numone=22&numtwo=3
@app.route('/api/addition', methods=['GET'])
def addition():

    # get(key, default=None, type=None)

    num_one = request.args.get('numone', default=0, type=int)
    num_two = request.args.get('numtwo', default=0, type=int)

    dict_data = {
        'num_one' : num_one,
        'num_two' : num_two,
        'output'  : add_numbers(num_one, num_two)
    }

    return json.dumps(dict_data)


@app.route('/api/time')
def time():
    return json.dumps({'time_str' : get_time_string()})


if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

How To Deploy

Step 1 - Set The Configuration Files.

In this example application, we will put our application in the /home directory on the Linux application server, inside a directory called local_dev_example



Before Loading, we need to set the configuration files

How To Deploy

Step 1.a - apache.conf

We set the Apache web server configuration

Here, we specify what alias we want for our web app, in other words what address in the browser.

In this case we are telling the web server we want www.mymachine.com/Local_Dev_Example and that the web server should look inside /home/local_dev_example/static for the web application files

Here, we set the directory settings for this web applications, specifying the web application directory

Here, we set our api routeproxy – our_alias/api will proxy for localhost:port/api (where our flask server is listening)

Here you need to set the Alias,

and the port number. The port number has to be set to be able to listen to requests, and should not be in use by another application!

Finally, for our Alias set Require all granted (allows all ip addresses to access this location)

Branch: master ▾ local_dev_example / infrastructure / apache.conf

Richard Allbert Initial Commit

0 contributors

16 lines (11 sloc) | 383 Bytes

```
Alias "/Local_Dev_Example" "/home/local_dev_example/static/"

<Directory /home/local_dev_example/static/>
    AllowOverride All
    Require all granted
    Options -Indexes
</Directory>


ProxyPass /Local_Dev_Example/api/ http://localhost:5004/api/
ProxyPassReverse /Local_Dev_Example/api/ http://localhost:5004/api/

<Location /Local_Dev_Example/>
  Require all granted
</Location>
```

In other words: Where there is a red box you need to write the setting for your app. Either:

The port

The directory

The alias

How To Deploy

Step 1.b – supervisor.conf

We set the process supervisor configuration

Branch: master ▾ local_dev_example / infrastructure / supervisor.conf

Richard Allbert Initial Commit

0 contributors

12 lines (11 sloc) | 320 Bytes

```
[program:local_dev_example]
command = /home/local_dev_example/infrastructure/gunicorn.sh
directory = /home/local_dev_example/
autostart = true
autorestart = true
redirect_stderr = true
stdout_logfile = /home/logs/local_dev_example.log
stdout_logfile_maxbytes = 10MB
stdout_logfile_backups = 3
startsecs = 3
numprocs = 1
```

Here we specify the name for our process, any name you like, probably best to use something easy to recognize.

Here, set the command to run the flask server via gunicorn. This is the path to the gunicorn.sh script

Here we tell the supervisor the location of our application

Here set the log file (usually just use the process name we set on the top line)

In other words: Where there is a red box you need to write the setting for your app. Either:

The process name you want

The directory to the gunicorn.sh or the app itself

How To Deploy

Step 1.c – gunicorn.sh

We set the script to run our flask server (under gunicorn)

Set the port, same as in the supervisor.conf

Branch: master ▾ local_dev_example / infrastructure / gunicorn.sh

Richard Allbert Initial Commit

0 contributors

Executable File | 9 lines (7 sloc) | 109 Bytes

```
1  #!/bin/bash
2
3  # run gunicorn
4  exec gunicorn \
5      -b "localhost:5004" \
6      -t 300 \
7      -w 5 \
8      server:app
```

Replace 'server' with the name of your flask server file (in our case it is 'server.py'

Branch: master ▾ New pull reques

Richard Allbert Initial Commit

- infrastructure
- static
- .editorconfig
- .gitignore
- GuideBook.pptx
- ReadMe.md
- get_data.py
- requirements.txt
- server.py

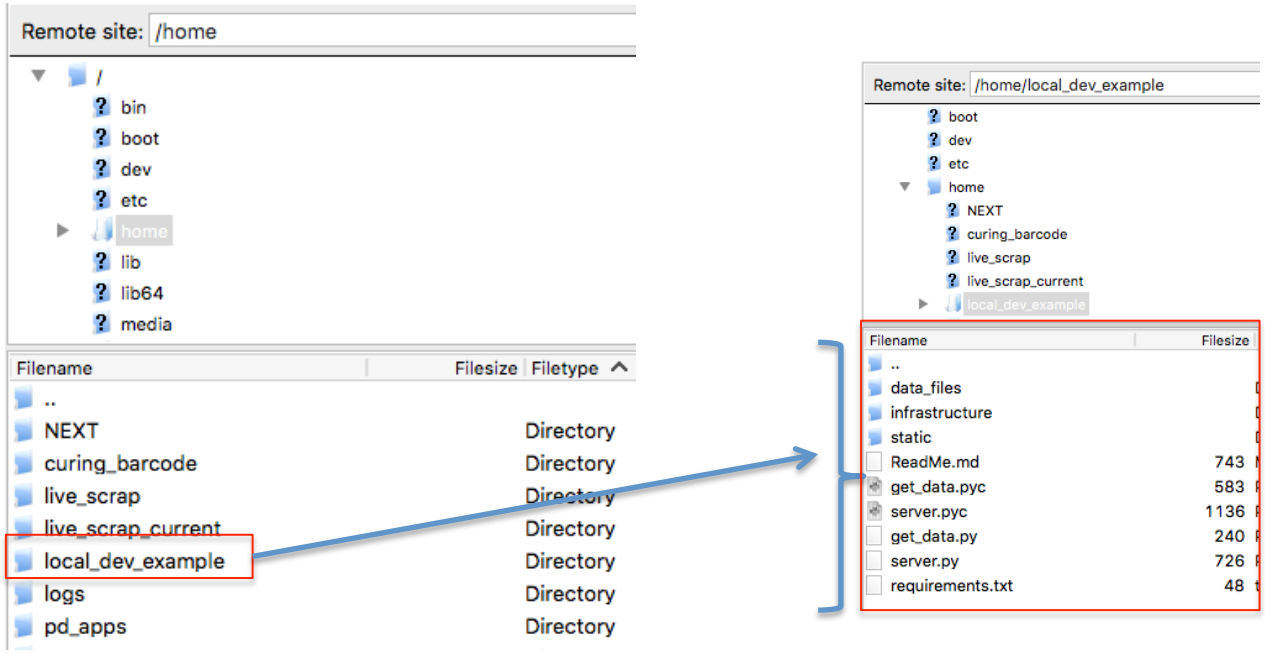
In other words: ... err this one is very simple.

How To Deploy

Step 2 - Load the application onto the server.

In this example application, we will put our application in the /home directory on the Linux application server.

It is an almost an exact copy of the github repository (removing unneeded documents like this document)



How To Deploy

Step 3 – Start the Application.

In the ReadMe.md, there are exact instructions on how to start the application. We will follow this step by step.

Deployment

The files in infrastructure can be used to deploy the site, using gunicorn + supervisor with apache + mod_proxy.

For a fresh installation, the files will need to be copied to the correct locations, and services restarted eg.

Apache

```
ln -s /home/local_dev_example/infrastructure/apache.conf /etc/httpd/conf.d/local_dev_example.conf

service httpd restart
```

Supervisor

```
ln -s /home/local_dev_example/infrastructure/supervisor.conf /etc/supervisor/conf.d/local_dev_example.conf

supervisorctl reread

supervisorctl reload

supervisorctl restart all
```

How To Deploy

Step 3 – Start the Application.

Apache

create a new web server config file in the apache directory by symbolically linking to our config file. Be sure to give an obvious name!

```
ln -s /home/local_dev_example/infrastructure/apache.conf /etc/httpd/conf.d/local_dev_example.conf
```

restart the web server to load the new configuration file

```
service httpd restart
```

Specify the name here

Supervisor

create a new supervisor config file in the supervisor directory by symbolically linking to our config file. Be sure to give an obvious name!

```
ln -s /home/local_dev_example/infrastructure/supervisor.conf /etc/supervisor/conf.d/local_dev_example.conf
```

Specify the name here

tell supervisor to reread all config files

```
supervisorctl reread
```

tell supervisor to reload all applications

```
supervisorctl reload
```

tell supervisor to restart all applications

```
supervisorctl restart all
```

How To Deploy

Step 3 – Start the Application.

verify the app files are present

Run the apache symbolic link cmmand

Restart Apache

Run the supervisor symbolic link command

Run the reread, reload, restart all command to start all applicatoins configured with the supervisor

The applications are started

```
[root@uccio ~]# pwd
/root
[root@uccio ~]# cd ..
[root@uccio /]# cd home/local_dev_example/
[root@uccio local_dev_example]# ls
data_files   get_data.pyc    ReadMe.md         server.py   static
get_data.py  infrastructure  requirements.txt  server.pyc
[root@uccio local_dev_example]# cd
[root@uccio ~]# pwd
/root
[root@uccio ~]# ln -s /home/local_dev_example/infrastructure/apache.conf /etc/httpd/conf.d/local_d
ev_example.conf
[root@uccio ~]# service httpd restart
Redirecting to /bin/systemctl restart  httpd.service
[root@uccio ~]# ln -s /home/local_dev_example/infrastructure/supervisor.conf /etc/supervisor/conf.
d/local_dev_example.conf
[root@uccio ~]# supervisorctl reread
No config updates to processes
[root@uccio ~]# supervisorctl reload
Restarted supervisord
[root@uccio ~]# supervisorctl restart all
local_dev_example: stopped
scrap_dashboard: stopped
live_scrap: stopped
scrap_dashboard: started
live_scrap: started
local_dev_example: started
[root@uccio ~]#
```

How To Deploy

Step 3 – Start the Application.

Now we can test the API, looks likt it works!!

10.130.4.101/Local_Dev_Example/api/time

n Dashboard... GitHub The Only Raspberry... Home - Domino Angular Material on... Dashboard - DSAA...

```
// 20170914145447
// http://10.130.4.101/Local_Dev_Example/api/time

{
  "time_str": "2017-09-14 14:54:45"
}
```