

---

# Essential Python Tools Documentation

*Release 3.7*

**Agiliq**

**Jul 31, 2018**



---

## Table of Contents:

---

<b>1</b>	<b>Interactive environments and debugging</b>	<b>3</b>
1.1	Ipython . . . . .	3
1.2	Jupyter . . . . .	4
1.3	pdb and ipdb . . . . .	5
<b>2</b>	<b>Linters and formatters</b>	<b>7</b>
2.1	PEP8 . . . . .	7
2.2	pycodestyle . . . . .	7
2.3	pylint . . . . .	9
2.4	pyflakes . . . . .	10
2.5	flake8 . . . . .	11
2.6	black . . . . .	13
2.7	autopep8 . . . . .	14
2.8	yapf . . . . .	14
2.9	Conclusion . . . . .	15
<b>3</b>	<b>Environment Management</b>	<b>17</b>
3.1	virtualenv and virtualenvwrapper . . . . .	17
3.2	pipenv . . . . .	18
3.3	pip, requirement.txt and pipfile . . . . .	19
3.4	poetry . . . . .	20
3.5	A comparison of the tools . . . . .	21
<b>4</b>	<b>Source code management</b>	<b>23</b>
4.1	git . . . . .	23
4.2	github . . . . .	23
4.3	gitlab . . . . .	23
4.4	Continous Integration . . . . .	24
<b>5</b>	<b>Documentation Tools</b>	<b>27</b>
5.1	Markdown (.md) . . . . .	27
5.2	Restructured Text (.rst) . . . . .	27
5.3	Sphinx . . . . .	28
<b>6</b>	<b>Deployment</b>	<b>29</b>
6.1	Docker . . . . .	29
6.2	Fabric . . . . .	31

6.3	Ansible . . . . .	31
6.4	Google Cloud Platform . . . . .	31
6.5	Amazon Web Services . . . . .	45
6.6	Deploying a Django application . . . . .	47
<b>7</b>	<b>Indices and tables</b>	<b>53</b>





# ESSENTIAL PYTHON TOOLS

Supercharge your Python  
development using professional tools

---

## Interactive environments and debugging

---

### 1.1 Ipython

**IPython** (Interactive Python) is a command shell for interactive computing in python, originally developed to make python easy and interactive, it offers introspection, rich media, shell syntax, tab completion, and history.

You can install Ipython via pip( `pip install ipython` ) but we suggest installing IPython using the [Python Anaconda distribution](#).

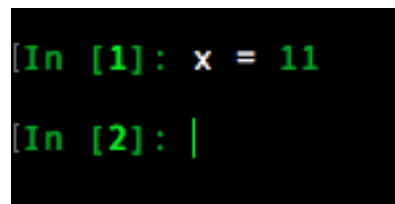
Anaconda is an open-source Python distribution which provides us with many python tools. When installed, Anaconda includes: core Python language, IPython, Jupyter Notebook and many data-science packages.

To use Ipython

```
$ ipython
```

IPython has more features than python like Syntax highlighting, Proper Intendation, tab-completion, documentation

#### Syntax highlighting

A screenshot of an IPython terminal window with a black background. The prompt '[In [1]:' is followed by the code 'x = 11', where 'x' is green, '=' is white, and '11' is green. The next line shows the prompt '[In [2]:' followed by a vertical bar '|', where the prompt is green and the bar is white.

#### Tab-completion

```
[In [2]: x.  
x.bit_length  x.from_bytes  x.real  
x.conjugate    x.imag       x.to_bytes  
x.denominator x.numerator
```

**Documentation** can be viewed by putting a `?` after the command.

```
[In [3]: x.conjugate?  
Docstring: Returns self, the complex conjugate of any int.  
Type:      builtin_function_or_method  
[In [4]: ]
```

## 1.2 Jupyter

**Jupyter Notebook** is a open source application for writing and sharing code and text. It is one of the popular data science tool in python.

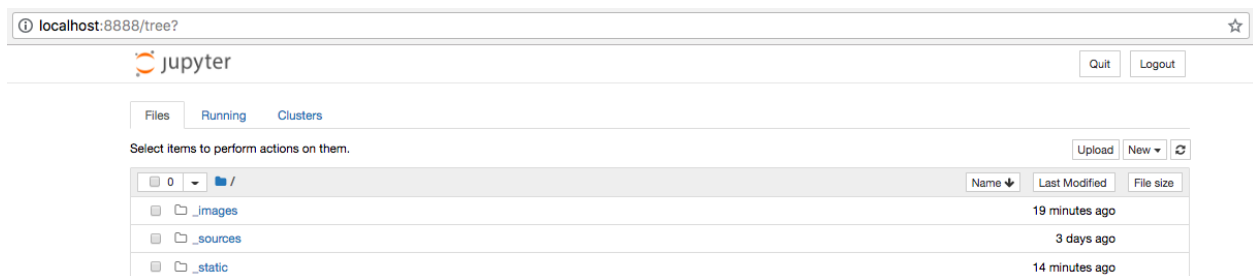
It has replaced IPython notebook, which initially supported only python but now later started supporting many languages. IPython notebooks was started to make working with python easy and interactive. IPython provides python backend (also known as kernel) for Jupyter

If you installed Python using Anaconda, then you have the Jupyter Notebook installed. To start the notebook

```
$ jupyter notebook
```

After starting Jupyter notebook, we'll notice that a tab will open in a web browser open. It will run the Jupyter Notebook on a local port, such as <http://localhost:8888>. It will list out the contents of the system in a directory format. To create new “notebooks” just click “New” and then selecting the python version.

Using Jupyter Notebook we can create shareable files that can support live code, charts, graphs, math, different forms of markup (Markdown, etc.), and much more.



To know more about ‘Jupyter Notebook’ check - <http://jupyter-notebook.readthedocs.io/en/stable/>



## 1.3 pdb and ipdb

### 1.3.1 pdb

`pdb` is a debugging tool that is part of python's standard library. It is an interactive source code debugger for Python programs.

Using `pdb`, we can set [breakpoints](#) at any point of our program to stop it and check for errors or the status of our running program.

**`pdb` help speed up the debugging process a lot faster than using simple `print()` statements everywhere.**

The easiest way to use `pdb` is to call it in the code you're working on.

```
import pdb; pdb.set_trace()
```

As soon as the interpreter reaches this line, we'll receive a command prompt on the terminal where we're running the program. This is a general Python prompt, but with some new commands.

- `l` (list) - Display 11 lines around the current line.
- `r` (return) - Continue execution until the current function returns.
- `b` (break) - Set a breakpoint (depending on the argument provided).
- `n` (next) - Continue execution until the next line in the current function is reached.
- `s` (step) - Execute the current line, stop at the first possible occasion.
- `j` (jump) - Jump to the next line to be executed.
- `c` (continue) - Creates a breakpoint in the program execution.

for more commands [check](#)

If we want to run the application from the debugger and set breakpoints without any changes in the source code, then we need to execute the application with the debugger, use the command

```
$ python -m pdb hello.py
```

### 1.3.2 ipdb

`ipdb`, the IPython-enabled python debugger, with all `pdb`'s features and adds ipython support for the interactive shell, like tab completion, color support, magic functions and more. We can use `ipdb` just as we use `pdb`.



### 2.1 PEP8

PEP8 is the official style guide for python. It is important to know the style-guide if you want to be a part of the python-community.

**PEP8 coding conventions are:**

- Spaces are the preferred indentation method.
- Use 4 spaces per indentation level.
- Limit all lines to a maximum of 79 characters.
- Separate top-level function and class definitions with two blank lines.
- Method definitions inside a class are surrounded by a single blank line.
- **Imports should be grouped in the following order:**
  - Standard library imports.
  - Related third party imports.
  - Local application/library specific imports.
  - A blank line between each group of imports.

### 2.2 pycodestyle

Pycodestyle (Formerly PEP8) is the official linter tool to check the python code against the style conventions of PEP8 python.

To install it: `pip install pycodestyle`.

Let us take a small example script to test *pycodestyle*

We will create a test script file `test_script.py` and use it as an example for all the linters.

```
from __future__ import print_function
import os, sys
import logging
from .. import views

class DoSomething(SomeCommand) :

    def __init__(self):
        for i in range(1,11):
            if self.number == i:
                print("matched")
            else:
                print('not matched')

    def check_user(self):
        if self.user: return True
        else : return False
```

If we run `pycodestyle`: `$ pycodestyle {source_file_or_directory}`

```
$ pycodestyle test_script.py
test_script.py:2:10: E401 multiple imports on one line
test_script.py:6:1: E302 expected 2 blank lines, found 1
test_script.py:6:31: E203 whitespace before ':'
test_script.py:9:25: E231 missing whitespace after ','
test_script.py:13:37: W291 trailing whitespace
test_script.py:16:21: E701 multiple statements on one line (colon)
test_script.py:16:34: W291 trailing whitespace
test_script.py:17:13: E271 multiple spaces after keyword
test_script.py:17:14: E203 whitespace before ':'
test_script.py:17:15: E701 multiple statements on one line (colon)
test_script.py:17:29: W291 trailing whitespace
```

To see the summary, use `--statistics -qq` `$ pycodestyle --statistics -qq {source_file_or_directory}`

```
$ pycodestyle --statistics -qq test_script.py
 2      E203 whitespace before ':'
 1      E231 missing whitespace after ','
 1      E271 multiple spaces after keyword
 1      E302 expected 2 blank lines, found 1
 1      E401 multiple imports on one line
 2      E701 multiple statements on one line (colon)
 3      W291 trailing whitespace
```

We can also make `pycodestyle` show the error and the description of how to solve the error by using `--show-source --show-pep8`

`$ pycodestyle --show-source --show-pep8 {source_file_or_directory}`

```
$ pycodestyle --show-source --show-pep8 test_script.py
test_script.py:2:10: E401 multiple imports on one line
import os, sys
    ^
    Place imports on separate lines.
...
...
...
```

To customise *pycodestyle* we can configure it at the project-level or in user-level. It is better to configure at the project level as the style usually varies for every-project.

To **configure** a project's *pycodestyle* create a `tox.ini` Or a `setup.cfg`

And add

```
[pycodestyle]
ignore = E501, W291
max-line-length = 88
statistics = True
```

In the above file ,

- `[pycodestyle]` tells this is the *pycodestyle* section
- we are telling to ignore the Error E501 (which is a line-length error) and Warning W291 (trailing whitespace warning).
- mentioning the `max-line-length` to be 88.
- and to show the statistics with every check

Table 1: **PEP8 Error/Warning code**

Error/ Warning	Meaning
Starting with E...	Errors
Starting with W...	Warnings
100 type ...	Indentation
200 type ...	Whitespace
300 type ...	Blank lines
400 type ...	Imports
500 type ...	Line length
600 type ...	Deprecation
700 type ...	Statements
900 type ...	Syntax errors

## 2.3 pylint

*Pylint* is a python linter which checks the source code and also acts as a bug and quality checker. It has more verification checks and options than just PEP8(Python style guide).

This is the most commonly used tool for linting in python.

**It includes the following features:**

- Checking the length of each line
- Checking if variable names are well-formed according to the project's coding standard
- Checking if declared interfaces are truly implemented.

To install it: `pip install pylint`.

Usage: `pylint {source_file_or_directory}`

Using the file *test\_script.py* as an example

```
$ pylint test_script.py
No config file found, using default configuration
***** Module test_script
C:  6, 0: No space allowed before :
class DoSomething(SomeCommand) :
    ^ (bad-whitespace)
C:  9, 0: Exactly one space required after comma
    for i in range(1,11):
        ^ (bad-whitespace)
C: 13, 0: Trailing whitespace (trailing-whitespace)
C: 16, 0: Trailing whitespace (trailing-whitespace)
C: 17, 0: Final newline missing (missing-final-newline)
C: 17, 0: No space allowed before :
    else : return False
        ^ (bad-whitespace)
C:  1, 0: Missing module docstring (missing-docstring)
C:  2, 0: Multiple imports on one line (os, sys) (multiple-imports)
E:  4, 0: Attempted relative import beyond top-level package (relative-beyond-top-
→level)
C:  6, 0: Missing class docstring (missing-docstring)
E:  6,18: Undefined variable 'SomeCommand' (undefined-variable)
C: 15, 4: Missing method docstring (missing-docstring)
R: 16, 8: The if statement can be replaced with 'return bool(test)' (simplifiable-if-
→statement)
R: 16, 8: Unnecessary "else" after "return" (no-else-return)
C: 16,22: More than one statement on a single line (multiple-statements)
R:  6, 0: Too few public methods (1/2) (too-few-public-methods)
W:  2, 0: Unused import sys (unused-import)
W:  2, 0: Unused import os (unused-import)
W:  3, 0: Unused import logging (unused-import)
W:  4, 0: Unused import views (unused-import)

-----
Your code has been rated at -10.00/10 (previous run: -10.00/10, +0.00)
```

As we can see *pylint* has more error/warning checks and options than *pep8*. And it is more descriptive.

**To customise *pylint*** we can configure it at the project-level, user-level or global-level .

- create a `/etc/pylintrc` for default global configuration
- create a `~/pylintrc` for default user configuration
- Or create a `pylintrc` file

To create a `pylintrc` file `pylint --generate-rcfile > pylintrc` , which creates a template `pylintrc`(with comments) which can be customised as required.

For example if we want the max line length to be 88, then we have to set the `max-line-length` to 88 .

## 2.4 pyflakes

*pyflakes* is a verification tool(linter) which checks for Python files for errors. *Pyflakes* doesn't verify the style at all but it verifies only logistic errors like the syntax tree of each file individually.

To install it: `pip install pyflakes`.

Let us take the same example script to test *pyflakes*

Usage: `pyflakes {source_file_or_directory}`

Using the file *test\_script.py* as an example

```
$ pyflakes test_script.py
test_script.py:2: 'sys' imported but unused
test_script.py:2: 'os' imported but unused
test_script.py:3: 'logging' imported but unused
test_script.py:4: '..views' imported but unused
test_script.py:6: undefined name 'SomeCommand'
```

It detected newly “library imported but unused” and “Undefined name”, it doesn’t verify the style but verify only logistic error.

If we like Pyflakes but also want stylistic checks, we can use `flake8`, which combines Pyflakes with style checks against PEP 8

## 2.5 flake8

`Flake8` is just a wrapper around *pyflakes*, *pycodestyle* and *McCabe script (circular complexity checker)* (which is used to detect complex-code).

If we like Pyflakes but also want stylistic checks, we can use `flake8`, which combines Pyflakes with style checks against PEP 8

To install it: `pip install flake8`.

Usage: `flake8 {source_file_or_directory}`

To get statics also `flake8 {source_file_or_directory} --statistics`

Using the file *test\_script.py* as an example

```
$ flake8 test_script.py --statistics
test_script.py:2:1: F401 'os' imported but unused
test_script.py:2:1: F401 'sys' imported but unused
test_script.py:2:10: E401 multiple imports on one line
test_script.py:3:1: F401 'logging' imported but unused
test_script.py:4:1: F401 '..views' imported but unused
test_script.py:6:1: E302 expected 2 blank lines, found 1
test_script.py:6:19: F821 undefined name 'SomeCommand'
test_script.py:6:31: E203 whitespace before ':'
test_script.py:9:25: E231 missing whitespace after ','
test_script.py:13:37: W291 trailing whitespace
test_script.py:16:21: E701 multiple statements on one line (colon)
test_script.py:16:34: W291 trailing whitespace
test_script.py:17:13: E271 multiple spaces after keyword
test_script.py:17:14: E203 whitespace before ':'
test_script.py:17:15: E701 multiple statements on one line (colon)
test_script.py:17:29: W291 trailing whitespace
2      E203 whitespace before ':'
1      E231 missing whitespace after ','
1      E271 multiple spaces after keyword
1      E302 expected 2 blank lines, found 1
1      E401 multiple imports on one line
2      E701 multiple statements on one line (colon)
4      F401 'os' imported but unused
```

(continues on next page)

(continued from previous page)

```
1      F821 undefined name 'SomeCommand'
3      W291 trailing whitespace
```

The output is formatted as:

```
file path : line number : column number : error code : short description
```

By adding the `--show-source` option, it'll be easier to find out what parts of the source code need to be revised.

```
$ flake8 test_script.py --show-source
test_script.py:2:1: F401 'os' imported but unused
import os, sys
^
test_script.py:2:1: F401 'sys' imported but unused
import os, sys
^
test_script.py:2:10: E401 multiple imports on one line
import os, sys
      ^
test_script.py:3:1: F401 'logging' imported but unused
import logging
^
...
...
...
```

We can see the result of pep8 (error code is Exxx and Wxxx) and pyflakes (error code is Fxxx) are output together.

Flake8 Error code meaning

The error code of flake8 are :

- E\*\*\*/W\*\*\*: Errors and warnings of pep8
- F\*\*\*: Detections of PyFlakes
- C9\*\*: Detections of circulate complexity by McCabe-script

Flake8 can be customised/configured in :

- Toplevel User directory, in `~/.config/flake8` Or
- In a project folder by one of `setup.cfg`, `tox.ini`, or `.flake8`.

To customize flake8

```
[flake8]
ignore = D203
exclude = .git,__pycache__,docs/source/conf.py,old,build,dist, *migrations*
max-complexity = 10
```

This is same as the below one line command

```
$ flake8 --ignore D203 \
  --exclude .git,__pycache__,docs/source/conf.py,old,build,dist \
  --max-complexity 10
```



## 2.6 black

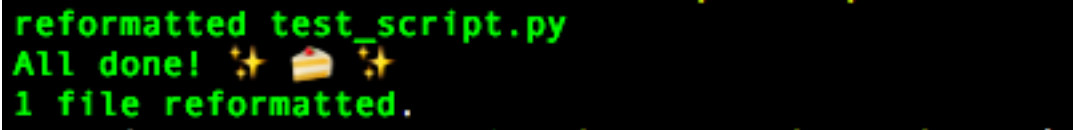
`black` is a python code auto-formatter. Black reformats entire files in place and also formats the strings to have double-quotes.

*Black* is not configurable(except for line-length).

To install it: `pip install black`.

Usage: `black {source_file_or_directory}`

The response we got when we did `black test_script.py` is



```
reformatted test_script.py
All done! ✨ 🍰 ✨
1 file reformatted.
```

Using the file `test_script.py` as an example

And the formatted code is

```
from __future__ import print_function
import os, sys
import logging
from .. import views

class DoSomething(SomeCommand):
    def __init__(self):
        for i in range(1, 11):
            if self.number == i:
                print("matched")
            else:
                print("not matched")

    def check_user(self):
        if self.user:
            return True
        else:
            return False
```

To customise *black* we have to add this section in `pyproject.toml`

```
[tool.black]
line-length = 90
py36 = true
include = '\.pyi?$'
exclude = '''
/(
    \.git
| \.mypy_cache
| \.tox
| \.venv
| _build
| buck-out
| build
| dist
```

(continues on next page)

(continued from previous page)

```
) /  
'''
```

In the above section, we have modified the line-length to 90, and specified the python version to 3.6

## 2.7 autopep8

`autopep8` automatically formats Python code to the PEP8 style. It fixes most of the formatting issues that are reported by `pycodestyle`.

To install it: `pip install autopep8`

Usage(to format a file): `autopep8 --in-place {file_name}`

here `--in-place` is to make changes to files in place.

Using the file `test_script.py` as an example

This is the formatted code.

```
from __future__ import print_function  
import os  
import sys  
import logging  
from .. import views  
  
class DoSomething (SomeCommand):  
  
    def __init__(self):  
        for i in range(1, 11):  
            if self.number == i:  
                print("matched")  
            else:  
                print('not matched')  
  
    def check_user(self):  
        if self.user:  
            return True  
        else:  
            return False
```

To configure `autopep8` we have to add this section `[pep8]` in `setup.cfg`:

```
[pep8]  
ignore = E226,E302,E41  
max-line-length = 160
```

## 2.8 yapf

Yet another Python formatter is another auto-formatter which is maintained by google. `yapf` is highly configurable and it has different base configurations, like pep8, Google and Facebook.

To install it: `pip install yapf`

Usage: `yapf -i {source_file_or_directory}`

here `-i` is to make changes to files in place.

This is the formatted code.

```
from __future__ import print_function
import os, sys
import logging
from .. import views

class DoSomething (SomeCommand):
    def __init__(self):
        for i in range(1, 11):
            if self.number == i:
                print("matched")
            else:
                print('not matched')

    def check_user(self):
        if self.user: return True
        else: return False
```

To configure *yapf* we have to add this section `[yapf]` in `setup.cfg`:

```
[yapf]
ignore = E226,E302
max-line-length = 96
```

---

## 2.9 Conclusion

Linting:

*Pylint* and *flake8* have the most detailed way of showing the error and warnings (and it also gives the code rating).



### 3.1 virtualenv and virtualenvwrapper

#### 3.1.1 virtualenv

*virtualenv* is a popular tool for creating isolated python environments without affecting other projects.

It is really helpful if you are having more than one project at a time, so that there won't be any version clashes among the packages of the projects. Example , if you want to work with more python2.7 for one project and python3.5 for another project, *virtualenv* solves the purpose.

It creates an environment that has its own isolated installation directories, that doesn't share libraries with other *virtualenv* environments.

We have to install it globally: `[sudo] pip install virtualenv`

Once we have *virtualenv* installed, we have to create the a directory for our *virtualenv* `mkdir ~/virtualenvironment`

#### 3.1.2 virtualenvwrapper

*virtualenvwrapper* is an just like an extension to *virtualenv* which simplifies the commands to use and manage.

**To install it:** `[sudo] pip install virtualenvwrapper`

After installing, Add these lines to your shell startup file (`.bashrc`, `.profile`, etc.)

```
export WORKON_HOME=$HOME/.virtualenvs
export PROJECT_HOME=$HOME/Devel
source /usr/local/bin/virtualenvwrapper.sh
```

After editing the file, reload the startup file (e.g., run `source ~/.bashrc`).

We tell the startup file to set the location where the virtual environments should live, the location of your development project directories, and the location of the script installed with this package:

Next, we can create the virtualenv with command `mkvirtualenv` This command creates the virtualenv and automatically loads it for us.

```
$ mkvirtualenv testEnv
New python executable in /Users/anmol/.virtualenvs/testEnv/bin/python2.7
Also creating executable in /Users/anmol/.virtualenvs/testEnv/bin/python
Installing setuptools, pip, wheel...done.
virtualenvwrapper.user_scripts creating /Users/anmol/.virtualenvs/testEnv/bin/
↳predeactivate
virtualenvwrapper.user_scripts creating /Users/anmol/.virtualenvs/testEnv/bin/
↳postdeactivate
virtualenvwrapper.user_scripts creating /Users/anmol/.virtualenvs/testEnv/bin/
↳preactivate
virtualenvwrapper.user_scripts creating /Users/anmol/.virtualenvs/testEnv/bin/
↳postactivate
virtualenvwrapper.user_scripts creating /Users/anmol/.virtualenvs/testEnv/bin/get_env_
↳details
(testEnv) $
```

To come out of the virtualenv use command `$ deactivate`

If you want to create a virtualenv with a different version of python like python3(which should be globally installed) then specify the python version using `-p python3`

My system created the virtualenv with my default python which is python2.7. If you also have python3 installed in your system and you want to create the virtualenv with python3 then create the virtualenv with this command `mkvirtualenv testEnv -p python3`

```
$ mkvirtualenv testEnv -p python3
Running virtualenv with interpreter /usr/local/bin/python3
Using base prefix '/usr/local/Cellar/python3/3.6.4_2/Frameworks/Python.framework/
↳Versions/3.6'
New python executable in /Users/anmol/.virtualenvs/testEnv/bin/python3
Installing setuptools, pip, wheel...done.
(testEnv) $ python -V
Python 3.6.4
(testEnv) $
```

To list all virtualenvs present in the system run command: `$ workon workon` lists all the virtualenvs present in the system.

To start working in a virtualenv: `$ workon <name_of_virtualenv>`

To remove/delete a virtualenv: `$ rmvirtualenv <name_of_virtualenv>`

## 3.2 pipenv

**Pipenv** is a tool for creating a separate/isolated working environment which manages the dependency versions. It creates virtualenv for every project in the project folder.

**To install it:** `[sudo] pip install pipenv`

Next, to create a pipenv for a project, go to the project directory and type

```
$ pipenv install <package>    // like pipenv install requests
```

```
$ pipenv install -r requirements.txt    // if our dependencies are listed in a file

$ pipenv --python python3 install <package> // with different version of python_
↪like python3
```

after creating a pipenv, 2 files will be created **Pipfile** and **Pipfile.lock** which lists all our packages and these files get updated whenever we install/update/delete any package.

If we want to add a package for only development/testing then use :code:‘pipenv install -d ‘

To activate this project’s virtualenv, run `pipenv shell`

And to run a command inside the virtualenv with `pipenv run . example` `pipenv run python hello.py`

And to exit the virtualenv run `exit`

### 3.3 pip, requirement.txt and pipfile

**Pip** (Python’s package manager) is a package management system used to install and manage software packages written in Python.

**To check pip version:** `pip -V`

To get pip:

First download pip

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
```

Then run this command

```
python get-pip.py
```

**List all packages installed :** `pip freeze`

To install/uninstall a package using pip:

```
pip install <pacakge>    // install
pip install <pacakge>==1.2.2    // install a specific version
pip uninstall <pacakge>    // uninstall
```

**Requirement.txt** is a text file which stores the list of all the pip packages with versions which are required to run the project.

To create a *requirements.txt* file do `pip freeze > requirements.txt`

A sample requirements.txt file

```
Django==2.0.3
djangorestframework==3.7.7
django-rest-swagger==2.1.2
coreapi==2.3.3
```

**Pipfile** is just a replacement to the requirement.txt file. pipfile is generated when using *pipenv*.

Pipfile lists all the packages by separating the development/testing packages from the main packages used and also mentions the python version it uses.

A sample Pipfile

```
[[source]]
url = "https://pypi.python.org/simple"
verify_ssl = true
name = "pypi"

[packages]
coverage = "*"
requests = "*"

[dev-packages]
pylint = "*"

[requires]
python_version = "3.6"
```

## 3.4 poetry

**Poetry** is a tool for dependency management and packaging in Python. It allows us to declare the libraries your project depends on and it will manage (install/update) them for us.

Poetry can be installed using pip, but the recommended way to install is

```
curl -sSL https://raw.githubusercontent.com/sdispater/poetry/master/
get-poetry.py | python
```

To use poetry run this command: `poetry init`

This command will help you create a `pyproject.toml` file interactively by prompting you to provide basic information about your package.

**pyproject.toml** is the main file which manages all the dependencies.

**pyproject.toml** file contains all the details of the project. It mentions the dependencies/dev-dependencies and also other details like name, description, author, version etc of project.

A sample *pyproject.toml*

```
[tool.poetry]
name = "base-ing"
version = "0.1.0"
description = ""
authors = ["anmol <anmol@agiliq.com>"]

[tool.poetry.dependencies]
python = "*"

[tool.poetry.dev-dependencies]
```

To add/remove a package : `poetry add <package>` `poetry remove <package>`

To add a package as a development-dependency: `poetry add <package> --dev`

To run a command in poetry `poetry run python hello.py`



## 3.5 A comparison of the tools

### 3.5.1 Python/pip Standard

Both `pipenv` and `virtualenvwrapper` are officially recommended and are considered as standards.

### 3.5.2 Easy of use.

`Pipenv` and `virtualenvwrapper` both are easy to use. .. Poetry is

For beginners I suggest, start with *virtualenv/virtualenvwrapper* and then use *pipenv* .



#### 4.1 git

**Git** is a version control system for tracking changes in computer files or a folder and coordinating work on those files among multiple people.

Git/Version Control System is the basic tool in software development.

The main purpose of Git is to manage software development projects and its files, as they are changing over time. It stores the information in a repository.

To install git check [this](#) .

To learn git check [official-tutorial](#) and [github-tutorial](#)

#### 4.2 github

**GitHub** is a web-based hosting service for version control using Git. It provides access control and several collaboration features such as issue tracking, feature requests, documentation, and wikis for every project.

With git, we can collaborate with other developers, track all our work via commits, and revert to any previous version of our code even if we accidentally delete something.

GitHub projects can be public or private and every publicly shared code is freely open to everyone. We can have private projects as well, though, they require a paid GitHub plan. Public repositories on Github are often used to share Open Source Software

It is the most-widely used web-based hosting service, and it has the largest number of open-source projects.

#### 4.3 gitlab

**GitLab** is a web-based and self-based hosting service for version control using Git. Gitlab is free and open-source.

Gitlab offers unlimited free public/private repositories, where as Github offers unlimited public repositories only(private repositories require a paid plan)

GitLab offers all the features of Github plus builtin CI/CD service (Continuous Integration & Continuous Delivery), and more authentication levels.

Gitlab(Community Edition) is open-source and can be installed in private servers.

## 4.4 Continous Integration

Continuous Integration (CI) is a development practice which automates the build and testing process of the code. Continuous Integration automates the build and testing processes. The main objective of CI is to test code for every push made to the repository. We need to have a testing framework for the test in CI.

**Each push is verified by an automated build and tests, individual changes are immediately tested and reported on when they are added to a repository, allowing developers to detect problems early.**

CI encourages developers to integrate their code into a main branch of a shared repository. Instead of building out features in isolation and integrating them at the end of a development cycle, code is integrated with the shared repository by each developer multiple times throughout the day.

CI makes code integration a simple, fast process that is part of the everyday development workflow in order to reduce integration costs and respond to defects early.

### 4.4.1 gitlab-ci

Gitlab Continuous Integration(CI) is a open-source CI service that is included in gitlab and can be used for all projects in gitlab.

To use GitLab CI, you first need to add a `.gitlab-ci.yml` file to the root directory of your repository, as well as configure your GitLab project to use a Runner. Afterwards, every commit or push will trigger the CI pipeline that has three stages: build, test and deploy.

To set up `.gitlab-ci.yml` file follow this [official doc](#)

### 4.4.2 circleci

Circleci is a cloud-based continuous integration server. It supports containers, OSX, Linux and can run within a private cloud or our own data center. that supports any language that builds on Linux or macOS.

It is available in both cloud-based and self-hosted.

**Setting up CircleCI:**

- Sign-in to circleci.com and link your project.
- After that activate the service hook to that repo in the profile page of circleci .
- Add circle.yml to the project

To set up `.circle-ci.yml` file follow this [official doc](#) and for python apps - [config circleci for python apps](#)

### 4.4.3 Travis CI

Travis CI is a hosted continous integration service used to build and test the projects hosted at github.com. It is free for all open-source projects hosted on Github.com .

Travis CI builds and runs the tests every time we push the code to the github repo or put a pull request to the repo.

To use *travis-ci* we must add a file `.travis.yml` to our repository. And link our github account with *travis-ci* by logging in to [travis-ci website](#)

Sample `.travis.yml` file

```
language: python
python:
- 2.7
- 3.6

# command to install dependencies
install:
- pip install -r requirements.txt

# command to run tests
script:
- pytest # or py.test To test for Python versions 3.5 and below

branches:
- master
- dev
```

This file will get our project tested on all the listed Python versions by running the given script, and it will build the master and dev branch only.

The CI Environment uses separate `virtualenv` instances for each Python version.

By default Travis CI uses `pip` to manage Python dependencies. If you have a `requirements.txt` file, Travis CI runs `pip install -r requirements.txt` during the `install` phase of the build.

Python projects need to provide the `script` key in their `.travis.yml` to specify what command to run tests with.



### 5.1 Markdown (.md)

`Markdown` is the most widely used markup language and it is often used to format readme files.

Text written in markdown are easy to read and can be easily converted to HTML.

There are different versions of markdown used , and `github` suggested one is the most popular.

### 5.2 Restructured Text (.rst)

`reStructuredText` is an easy-to-read, plaintext markup syntax and parser system.

When compared *reStructuredText* is more formalised and powerful than *markdown*.

It is useful for in-line program documentation (such as Python docstrings), for quickly creating simple web pages, and for standalone documents.

`reStructuredText` is designed for extensibility for specific application domains.

#### 5.2.1 When to use *markdown* and *reStructuredText*

- If you want to write a gist or a single page readme then it is better to use *markdown*.
- If you want to write a documentation consisting of many in-line programs and tables then it is better to use *reStructuredText*.

**So, which one to use ?** Generally it is - For small documentation use *markdown* otherwise use *reStructuredText*.

## 5.3 Sphinx

[Sphinx](#) is a documentation generator/tool which converts *reStructuredText* files (.rst) into HTML websites or to other formats like PDF, LaTeX, Epub, Texinfo.

Sphinx support Themes and extensions.

Btw this document is built using Sphinx.



### 6.1 Docker

**Docker** is an open-source tool for creating, deploying, and running applications by using containers.

Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

By using containers, the application will run on any machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

Containerization is an approach in which an application or service, its dependencies, and its configuration (abstracted as deployment manifest files) are packaged together as a container image.

The containerized application can be tested as a unit and deployed as a container image instance to the host operating system (OS).

Docker is an open-source tool for automating the deployment of applications.

#### 6.1.1 Docker Compose

**Compose** is a tool for running multi-container applications like for example a container with a DataBase and an application. We can start/stop multiple services using compose with a single command.

We can create multiple compose files each for production, staging, development, testing, as well as CI, and each will be isolated with each other.

To use compose

- Create a `Dockerfile` where all our environment configuration and initial packages are mentioned.
- Create a file `docker-compose.yml`, and mention all the services which we would be using.
- Finally run `docker-compose up`.

Dockerfile

```
FROM python:3.4-alpine # 1
ADD . /code #2
WORKDIR /code #3
ADD requirements.txt /code/ #4
RUN pip install -r requirements.txt #5
```

In the above file we

- In #1 we are building an image starting with the Python 3.4 image
- In #2 and #3 we are adding directory :code:‘ . ‘ into the path /code in the image and making it the working directory.
- In #4 and #5 we adding the requirements file to the /code/ directory and installing all requirements.

docker-compose.yml

```
version: '3' # 1

services: # 2
  web:
    build: . # 3
    command: python3 manage.py runserver 0.0.0.0:8000 #4
    volumes: # 5
    - ./code
    ports: #6
    - "8000:8000"
    depends_on: #7
    - db
  db:
    image: postgres
```

- In #1 we mention the docker version (which is 3)
- #2 defines two services, web and db.
  - The web service uses an image that’s built from the Dockerfile
  - The db service uses a public Postgres image pulled from the Docker Hub registry.
- #3 tells to find the the dockerfile in the current directory
- #4 is a command to run the service .
- #5 tells the host paths for that service.
- #6 forwards the exposed port 8000 on the container to port 8000 on the host machine.
- #7 mentions the dependency between services

To use the docker compose , we use commands

```
$ docker-compose up # to create and start the containers
$ docker-compose build #to build or rebuild services
$ docker-compose up --build # build the services and start the containers
```

**We write different ‘docker-compose’ files for each Development, Testing, & Production.**

## 6.2 Fabric

**Fabric** is a high level Python (2.7, 3.4+) library designed to execute shell commands remotely over SSH for application deployment.

It provides a basic suite of operations for executing local or remote shell commands (normally or via `sudo`) and uploading/downloading files, as well as auxiliary functionality such as prompting the running user for input, or aborting execution.

We can execute shell commands over SSH, so we only need to have SSH running on the remote machine.

It interact with the remote machines that we specify as if they were local.

**Fabric can be used for many things, including deploying, restarting servers, stopping and restarting processes.**

To install fabric `pip install fabric`

Fabric, provides a command line utility, `fab` which looks for a `fabfile.py`, which is a file containing Python code.

example fabfile:

```
from fabric.api import run
def diskspace():
    run('df')
```

The above example provides a function to check free disk space, `run` command executes a remote command on all specific hosts, with user-level permissions.

The fabfile should be in the same directory where we run the Fabric tool. We write all our functions, roles, configurations, etc in a fabfile.

## 6.3 Ansible

**Ansible** is an automation tool to deploy our applications. It gives us the power to deploy multi-tier applications reliably and consistently, all from one common framework.

Ansible is a configuration management and provisioning tool used to automate deployment tasks over SSH.

Ansible helps to automate :

- **Application deployment:** Make DevOps easier by automating the deployment of internally developed applications to our production systems.
- **Configuration management:** Change the configuration of an application, OS, or device; start and stop services; install or update applications; implement a security policy; or perform a wide variety of other configuration tasks.
- Set up the various servers needed in our infrastructure.

## 6.4 Google Cloud Platform

**Google Cloud Platform(GCP)** is a suite of cloud computing services offered by Google.

The platform includes a range of hosted services for compute, storage and application development that run on Google hardware. Google Cloud Platform services can be accessed by software developers, cloud administrators and other enterprise IT professionals over the public internet or through a dedicated network connection.

The core cloud computing products in Google Cloud Platform include:

- **Google Compute Engine**, is an infrastructure-as-a-service (IaaS) offering that provides users with virtual machine instances for workload hosting.
- **Google App Engine**, is a platform-as-a-service (PaaS) offering that gives software developers access to Google's scalable hosting. Developers can also use a software developer kit (SDK) to develop software products that run on App Engine.
- **Google Cloud Storage**, is a cloud storage platform designed to store large, unstructured data sets. Google also offers database storage options, including Cloud Datastore for NoSQL nonrelational storage, Cloud SQL for MySQL fully relational storage and Google's native Cloud Bigtable database.
- **Google Kubernetes Engine**, is a managed, production-ready orchestrated environment for deploying containerized applications that run within Google's public cloud. It brings our latest innovations in developer productivity, resource efficiency, automated operations, and open source flexibility to accelerate your time to market.

Let us see an example tutorial below which deploys a django application on GCP using Kubernetes

This tutorial should help to deploy a django application on a Kubernetes Cluster. Before starting this tutorial, the user is expected to have basic knowledge of GKE, Django, PostgreSQL and Docker

### 6.4.1 Understanding Kubernetes

Before we jump into the tutorial, lets have a basic understanding of what kubernetes is and how will it be useful for us to deploy our django application.

#### What is Kubernetes?

**Kubernetes**, at its basic level, is a system for running & co-ordinating containerized applications across a cluster of machines. It is a platform designed to completely manage the life cycle of containerized applications and services using methods that provide predictability, scalability, and high availability.

To know more about kubernetes, visit [here](#)

Moving on, as a part of this tutorial we'll be deploying Polls API, from [here](#)

#### Local Deployment of Polls API

Let's first clone our sample django application from

```
git clone https://github.com/yvssantosh/django-polls-rest.git
```

Just to make sure we're on master branch, run the command `git checkout master`

To test the application locally, let's create a virtual environment, and test the server

```
# Creating a virtual environment
mkvirtualenv pollsapl

# Installing current project requirements
pip install -r requirements.txt

# Connect to postgres
export POSTGRES_USER=pollsdb
export POSTGRES_DB=polls_admin
export POSTGRES_PASSWORD=polls_password
export POLLSAPI_PG_HOST=127.0.0.1
```

(continues on next page)

(continued from previous page)

```
# Running migrations
python manage.py migrate

# Start the local server
python manage.py runserver 0.0.0.0:8000
```

Now that we have tested on local server, lets create a new kubernetes cluster and deploy our application on it.

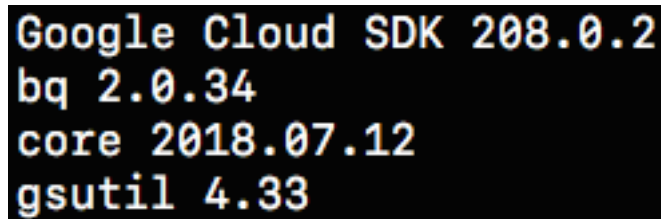
## Setting up Google Cloud SDK

For instructions to setup Google Cloud SDK navigate to <https://cloud.google.com/sdk>

Read the installation instructions and setup accordingly. Once done, check the installation status by running the command

```
$ gcloud -v
```

It should show an output similar to this:



```
Google Cloud SDK 208.0.2
bq 2.0.34
core 2018.07.12
gsutil 4.33
```

## Setting up kubectl command line

To manage and maintain a kubernetes cluster from our desktop/laptop we need to setup kubectl command line. It can be done using the command

```
gcloud components install kubectl
```

```
Your current Cloud SDK version is: 208.0.2
Installing components from version: 208.0.2
```

These components will be installed	
Name	Version
kubectl	1.9.7
kubectl	

```
For the latest full release notes, please visit
https://cloud.google.com/sdk/release_notes
```

```
Do you want to continue (Y/n)? Y
```

```
= Creating update staging area
= Installing: kubectl
= Installing: kubectl
= Creating backup and activating new installation
```

```
Performing post processing steps...done.
```

```
Update done!
```

Once the installation finishes, we should see something like this:

### Creating a kubernetes cluster

Navigate to <https://console.cloud.google.com>. Select an existing project or create a new one, based on requirement.

Then click on Kubernetes Engine from the navigate menu which would result in the following page

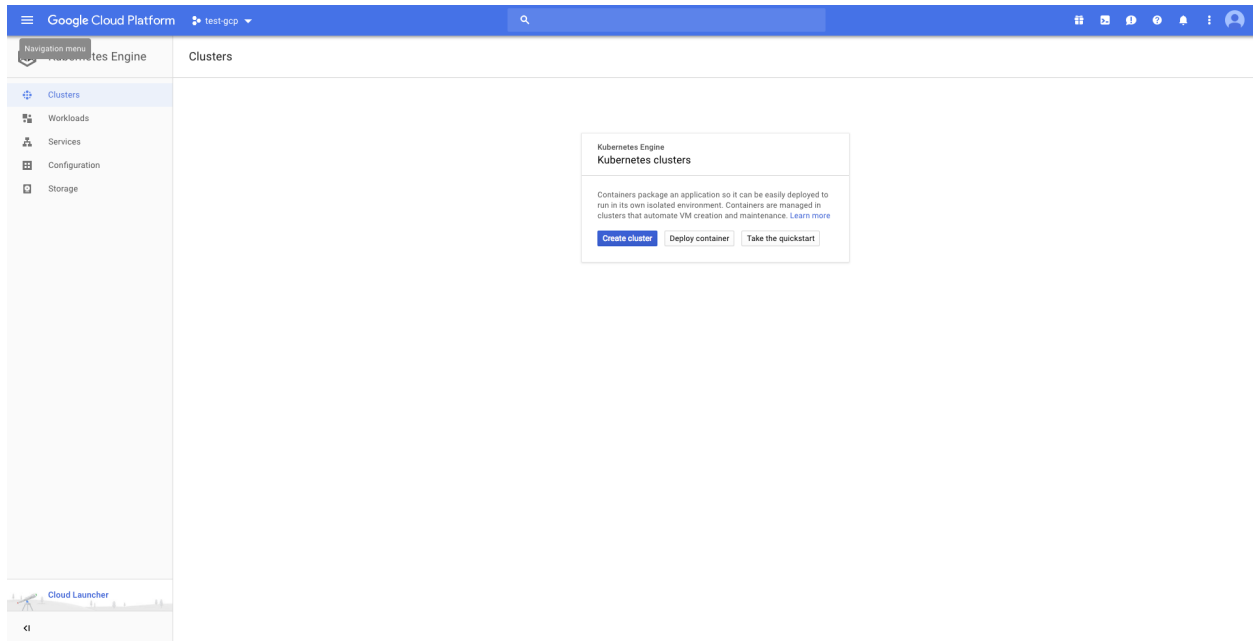
Create a new cluster. I've created a cluster based on the following settings

```
Cluster name      : pollsapl
Location          : Zonal
Zone              : asia-south1-a
Cluster Version   : 1.9.7-gke.3 (default)
Machine Type      : Small (1 shared CPU with 1.7GB Memory)
Node Image        : Core OS (cos)
Size              : 2
Boot Disk Size    : 20GB per Node
```

```
#####
```

```
→ #
```

(continues on next page)



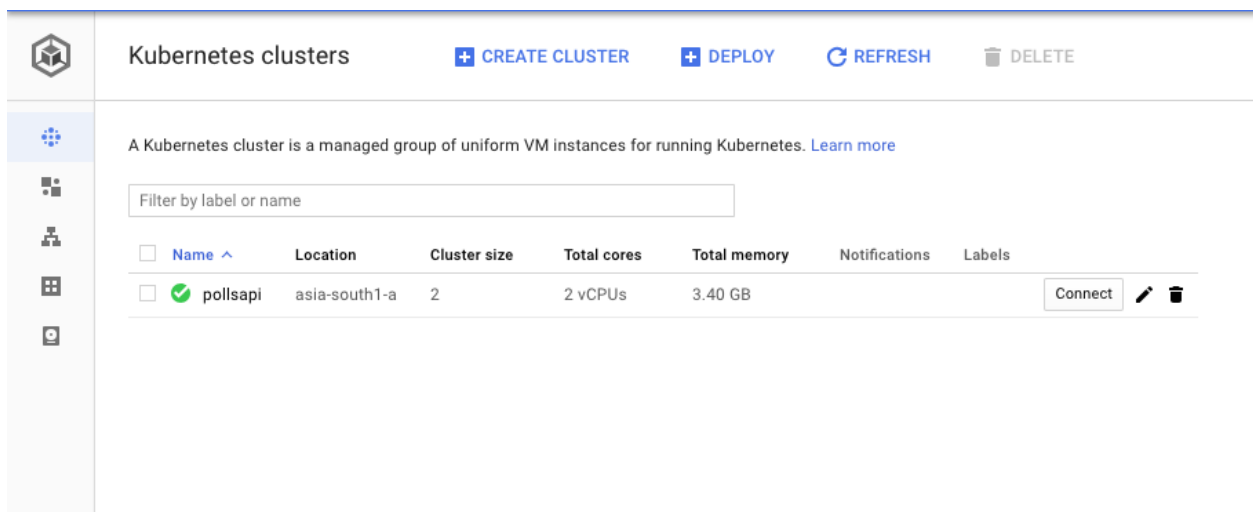
(continued from previous page)

```
## Only for testing purposes as preemptible nodes are NOT recommended for Production #
↪ #
#####
↪ #

Preemptible Nodes      :    Yes
```

Please be patient as it takes some time to create the cluster

Once the cluster is created, we'll be able to see a tick mark beside the name of the cluster.



Now click on `Connect` and copy the command shown, and paste it in terminal.

Once connected run the command `kubectl get all`.

Now that the cluster is up and running, let's package our application into a containerized one using docker.

```
yvssantosh@Sivas-MacBook-Air:~$ gcloud container clusters get-credentials pollsapi --zone asia-south1-a --project test-gcp-208915
Fetching cluster endpoint and auth data.
kubeconfig entry generated for pollsapi.
yvssantosh@Sivas-MacBook-Air:~$
```

```
yvssantosh@Sivas-MacBook-Air:~$ kubectl get all
NAME                TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
svc/kubernetes      ClusterIP     10.51.240.1    <none>         443/TCP    30m
yvssantosh@Sivas-MacBook-Air:~$
```

## Setting up Google Container Registry using Docker

Configuring docker with gcloud:

```
gcloud auth config-docker
```

The following settings will be added to your Docker config file located at [/Users/yvssantosh/.docker/config.json]:

```
{
  "credHelpers": {
    "gcr.io": "gcloud",
    "us.gcr.io": "gcloud",
    "eu.gcr.io": "gcloud",
    "asia.gcr.io": "gcloud",
    "staging-k8s.gcr.io": "gcloud"
  }
}
```

Do you want to continue (Y/n)? Y

Docker configuration file updated.

Once docker is configured, we are ready to build the image.

```
# Build the image
# Common format to push an image to google container registry is gcr.io/$PROJECT_ID/
# → $IMAGE_NAME:$TAG

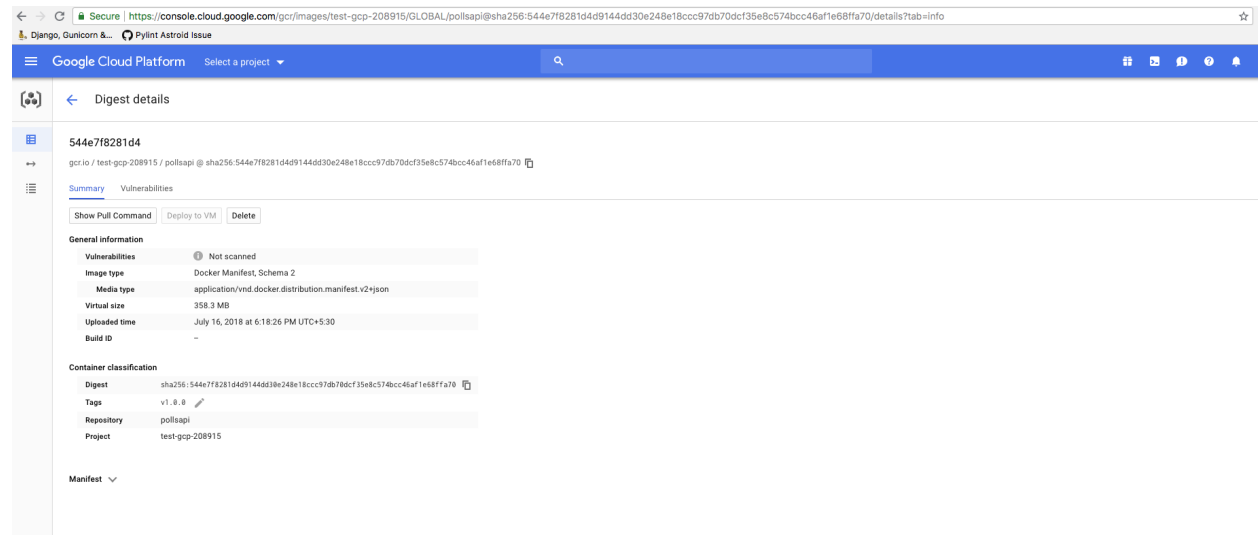
export PROJECT_ID=YOUR_PROJECT_ID_HERE
export IMAGE_NAME=YOUR_IMAGE_NAME_HERE
export TAG=YOUR_IMAGE_TAG (optional, default is `latest`)

# In my case, giving the tag as v1.0.0 (default is latest)
docker build -t gcr.io/test-gcp-208915/pollsapi:v1.0.0 .
# (Note the . in the end)

# Push the image
docker push gcr.io/test-gcp-208915/pollsapi:v1.0.0
```



Once the image has been pushed, paste the push URL in browser. It will ask you to sign in into google account which has been used to configure this cluster (if not already signed in).



Since our image has been uploaded successfully, we need to setup the database next.

## Setting up Helm Package Manager

The simplest way of setting up PostgreSQL on kubernetes is with the help of [Helm Package Manager](#)

For mac users, the command to install helm (using brew) is:

```
# Install Helm
brew install kubernetes-helm

# Setup Helm
helm init
```

- Note: Often during package installation i.e., `helm install --name MY_RELEASE stable/PACKAGE_NAME` a common error is generated explaining tiller not having access to create cluster role bindings. This usually happens if the user logged inside Google Cloud SDK doesn't have proper access to create role bindings or issues with helm installation.

If that error occurs, then run the following commands:

```
# Completely uninstall helm
helm reset --force

# Remove directories created by helm
sudo rm -r ~/.helm
```

Once helm is completely removed, create a clusterrolebinding and a serviceaccount for helm using the code below

rbac-config.yaml

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: tiller
```

(continues on next page)

(continued from previous page)

```
namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: tiller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: tiller
  namespace: kube-system
```

Create a file named `rbac-config.yaml` and run the following using `kubectl` command line

```
# Creating Service Account and ClusterRoleBinding Tiller
kubectl create -f rbac-config.yaml
```

Once this is successfully done, initialize helm using

```
helm init --service-account tiller
```

And then run the command to install PostgreSQL in our cluster, as previously mentioned.

## Setting up PostgreSQL

Before we setup PostgreSQL, let's create a namespace databases

```
# Why create namespace databases?
# This command is totally optional, but this is preferred this because I place all the
# databases created in a single namespace so that they'll be easy to access.
kubectl create namespace databases

# Before creating PostgreSQL using helm, let's understand few basics.

# Default command of creation enables Persistent Volume Claim (PVC)
# Instead of default postgres username, we are setting custom user.
# So replace YOUR_POSTGRES_USER with desired username, in my case polls_admin &
# MY_RELEASE_NAME, which in my case is pollssdb &
# MY_DATABASE_NAME, which in my case is pollssdb

# helm install --name MY_RELEASE_NAME stable/postgresql --set postgresUser=YOUR_
↪POSTGRES_USER,postgresDatabase=MY_DATABASE_NAME --namespace databases

helm install --name pollssdb stable/postgresql --set postgresUser=polls_admin,
↪postgresDatabase=pollssdb --namespace databases

# If user wishes not to have a separate namespace then just ignore the last two words
# i.e. --namespace databases
```

For more options on customizing postgres with custom parameters, see [here](#)

```

yvssantosh@Sivas-MacBook-Air:~/Desktop/workspace/django-on-k8s$ helm install --name pollsd stable/postgresql --namespace databases
NAME: pollsd
LAST DEPLOYED: Mon Jul 16 18:04:25 2018
NAMESPACE: databases
STATUS: DEPLOYED

RESOURCES:
==> v1/Pod(related)
NAME                                READY  STATUS   RESTARTS  AGE
pollsd-postgresql-7f4f796999-2mmtt  0/1    Pending  0          0s

==> v1/Secret
NAME                                TYPE  DATA  AGE
pollsd-postgresql                   Opaque 1      0s

==> v1/ConfigMap
NAME                                DATA  AGE
pollsd-postgresql                   0      0s

==> v1/PersistentVolumeClaim
NAME                                STATUS  VOLUME  CAPACITY  ACCESS MODES  STORAGECLASS  AGE
pollsd-postgresql                   Pending standard 0s

==> v1/Service
NAME                                TYPE        CLUSTER-IP  EXTERNAL-IP  PORT(S)  AGE
pollsd-postgresql                   ClusterIP   10.61.254.75 <none>      5432/TCP   0s

==> v1beta1/Deployment
NAME                                DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
pollsd-postgresql                   1        1        1            0          0s

NOTES:
PostgreSQL can be accessed via port 5432 on the following DNS name from within your cluster:
pollsd-postgresql.databases.svc.cluster.local
To get your user password run:

PGPASSWORD=$(kubectl get secret --namespace databases pollsd-postgresql -o jsonpath="{.data.postgres-password}" | base64 --decode; echo)

To connect to your database run the following command (using the env variable from above):

kubectl run --namespace databases pollsd-postgresql-client --restart=Never --rm --tty -i --image postgres \
--env "PGPASSWORD=$PGPASSWORD" \
--command -- psql -U postgres \
-h pollsd-postgresql postgres

To connect to your database directly from outside the K8s cluster:
PGHOST=127.0.0.1
PGPORT=5432

# Execute the following commands to route the connection:
export POD_NAME=$(kubectl get pods --namespace databases -l "app=postgresql,release=pollsd" -o jsonpath="{.items[0].metadata.name}")
kubectl port-forward --namespace databases $POD_NAME 5432:5432

```

**DO NOT FORGET** to take a note of PGPASSWORD as seen in the NOTES section (above image) once postgres has been created

```

# Saving password of PostgreSQL into environment variable $PGPASSWORD
PGPASSWORD=$(kubectl get secret --namespace databases pollsd-postgresql -o jsonpath="
↪{.data.postgres-password}" | base64 --decode; echo)

# Why save the password?
# Since we have created a separate namespace for databases, secrets from one_
↪namespaces cannot be accessed from another
# So in order to access the postgres password in the default namespace, we must_
↪create a new secret
# Let's first convert our password into base64 encoding.

echo -n $PGPASSWORD | base64

# MUST DO : Copy the generated value and replace it with `YOUR_ENCODED_PASSWORD` in_
↪the `polls-password-secret.yml`. Then create the secret.

kubectl create -f pollsd-password-secret.yml

# Now that the secret has been setup, let's migrate the data.
kubectl create -f polls-migration.yml

# Wait for a minute and check the status of the migration using following commands.
kubectl get jobs

# In order to check the logs, identify the pod running the pod running migration.
kubectl get pods --show-all

# Check the logs of the pod

```

(continues on next page)

(continued from previous page)

```
# kubectl logs POD_NAME
kubectl logs polls-migration-5tf8z

# Since the jobs have passed, there is no need for them to exist.
# We can just delete the jobs using
kubectl delete -f polls-migration.yml
```

```
yvssantosh@Sivas-MacBook-Air:~/Desktop/workspace/pollsapi$ kubectl get jobs
NAME                DESIRED    SUCCESSFUL    AGE
polls-migration      1          1             7s
yvssantosh@Sivas-MacBook-Air:~/Desktop/workspace/pollsapi$ kubectl get po --show-all
NAME                READY      STATUS      RESTARTS    AGE
polls-migration-5tf8z 0/1        Completed   0           18s
yvssantosh@Sivas-MacBook-Air:~/Desktop/workspace/pollsapi$ kubectl logs polls-migration-5tf8z
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, polls, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying polls.0001_initial... OK
  Applying sessions.0001_initial... OK
yvssantosh@Sivas-MacBook-Air:~/Desktop/workspace/pollsapi$
```

## Serving Static Files

Now that we have the database up and running with our migrations, let's setup our static files. Rather than setting up a separate NGINX server to serve static files, it'd be much simpler, secure & faster to use Google Cloud Storage as a provider to serve static files.

Let's first create a bucket in Google Cloud Storage. Visit <https://console.cloud.google.com/storage>

Make sure to check if the right project is selected.

I've created a bucket using the following settings:

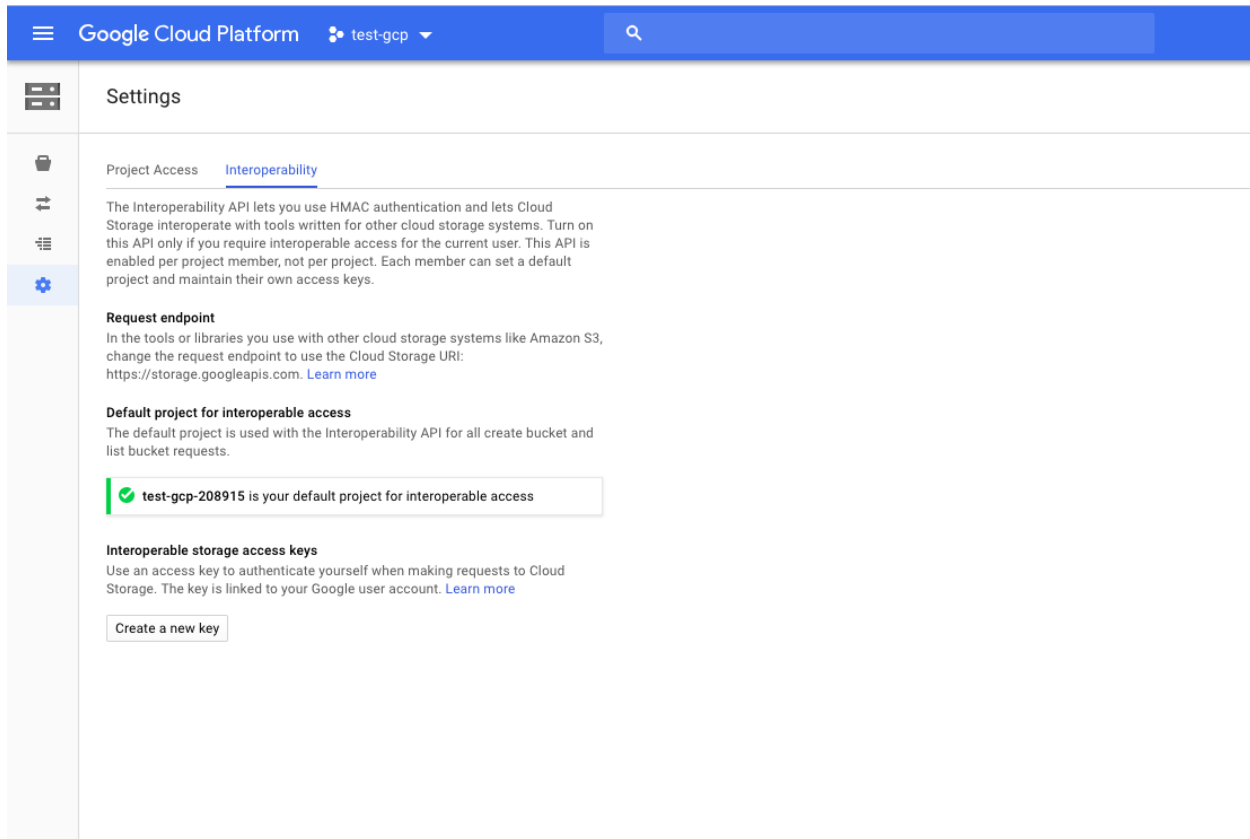
Name of Bucket	:	pollsapi-storage
Default Storage Class	:	Regional
Location	:	asia-south1 (Closest to my location)

Once the bucket is created, navigate to the settings icon as shown below

In the interoperability tab, create a new key. This key is required to let our django application send static files to our bucket.

Now that we have ACCESS\_KEY, ACCESS\_SECRET and BUCKET\_NAME, let's create a secrets file in kubernetes, so that we can directly use these as environment variables in our django application.





```
# Lets first encode our secrets into base64 format

echo -n 'YOUR_SECRET_ACCESS_KEY_ID_HERE' | base64

## Repeat the same for SECRET_ACCESS_KEY and BUCKET_NAME
```

Once we have the three generated values, replace them in `cloud-storage-secrets.yml`. After replacing the values with appropriate ones, lets create our secret in kubernetes.

```
# Creating cloud storage secret

kubectl create -f cloud-storage-secrets.yml
```

Now that the secrets are setup sucessfully, lets run the Job `polls-collect-static.yml` in order to collect static files.

```
kubectl create -f polls-collect-static.yml

# Note : It will take some time to collect the static files, as they are being
↪ uploaded
# to our bucket from the batch job which we created just now.
# We can just check the status of static files by either checking the logs
# or by checking the job status itself
```

```

yvssantosh@Sivas-MacBook-Air:~/Desktop/workspace/pollsapi$ kubectl create -f cloud-storage-secrets.yml
secret "pollsapi-static-serve" created
yvssantosh@Sivas-MacBook-Air:~/Desktop/workspace/pollsapi$ kubectl create -f polls-collect-static.yml
job "polls-collect-static" created
yvssantosh@Sivas-MacBook-Air:~/Desktop/workspace/pollsapi$ kubectl get pods --show-all
NAME                                READY    STATUS    RESTARTS   AGE
polls-api-6656bb4d9b-74k7b          1/1      Running   0           1d
polls-collect-static-mj4jk          1/1      Running   0           11s
yvssantosh@Sivas-MacBook-Air:~/Desktop/workspace/pollsapi$ kubectl logs polls-collect-static-mj4jk
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/fonts/README.txt'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/fonts/Roboto-Light-webfont.woff'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/fonts/Roboto-Regular-webfont.woff'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/fonts/Roboto-Bold-webfont.woff'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/fonts/LICENSE.txt'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/img/README.txt'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/img/icon-unknown-alt.svg'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/img/search.svg'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/img/tooltag-arrowright.svg'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/img/LICENSE'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/img/icon-clock.svg'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/img/selector-icons.svg'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/img/icon-calendar.svg'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/img/icon-alert.svg'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/img/icon-no.svg'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/img/icon-yes.svg'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/img/icon-unknown.svg'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/img/icon-addlink.svg'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/img/sorting-icons.svg'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/img/inline-delete.svg'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/img/icon-changelink.svg'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/img/icon-deletelink.svg'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/img/calendar-icons.svg'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/img/tooltag-add.svg'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/img/gis/move_vertex_on.svg'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/img/gis/move_vertex_off.svg'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/css/base.css'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/css/login.css'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/css/rtl.css'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/css/changelists.css'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/css/forms.css'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/css/responsive.css'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/css/autocomplete.css'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/css/responsive_rtl.css'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/css/fonts.css'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/css/widgets.css'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/css/dashboard.css'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/css/vendor/select2/select2.min.css'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/css/vendor/select2/LICENSE-SELECT2.md'
Copying '/usr/local/lib/python3.7/site-packages/django/contrib/admin/static/admin/css/vendor/select2/select2.css'

Copying '/usr/local/lib/python3.7/site-packages/rest_framework/static/rest_framework/js/prettify-min.js'
Copying '/usr/local/lib/python3.7/site-packages/rest_framework/static/rest_framework/js/jquery-3.3.1.min.js'
Copying '/usr/local/lib/python3.7/site-packages/rest_framework/static/rest_framework/js/csrf.js'
Copying '/usr/local/lib/python3.7/site-packages/rest_framework/static/rest_framework/js/default.js'
Copying '/usr/local/lib/python3.7/site-packages/rest_framework/static/rest_framework/js/ajax-form.js'
Copying '/usr/local/lib/python3.7/site-packages/rest_framework/static/rest_framework/js/coreapi-0.1.1.js'
Copying '/usr/local/lib/python3.7/site-packages/rest_framework/static/rest_framework/js/bootstrap.min.js'
Copying '/usr/local/lib/python3.7/site-packages/rest_framework_swagger/static/rest_framework_swagger/favicon-16x16.png'
Copying '/usr/local/lib/python3.7/site-packages/rest_framework_swagger/static/rest_framework_swagger/favicon-32x32.png'
Copying '/usr/local/lib/python3.7/site-packages/rest_framework_swagger/static/rest_framework_swagger/logo_small.png'
Copying '/usr/local/lib/python3.7/site-packages/rest_framework_swagger/static/rest_framework_swagger/bundles/vendors.bundle.css'
Copying '/usr/local/lib/python3.7/site-packages/rest_framework_swagger/static/rest_framework_swagger/bundles/app.bundle.js'
Copying '/usr/local/lib/python3.7/site-packages/rest_framework_swagger/static/rest_framework_swagger/bundles/app.bundle.css'

158 static files copied.
yvssantosh@Sivas-MacBook-Air:~/Desktop/workspace/pollsapi$ kubectl get jobs
NAME            DESIRED    SUCCESSFUL    AGE
polls-collect-static 1            1             6m
yvssantosh@Sivas-MacBook-Air:~/Desktop/workspace/pollsapi$

```

We have successfully setup static files in our application. But the major question is:

How are the static files being served?

To answer that question, let's see a small code snippet below

```

# First, the packages Boto & Django Storages are required. Lets install them
# These packages help us to connect to Google Cloud Storage
pip install boto django-storages

# Check the following snippet now (from settings.py file under the STATIC_FILES_
↪ SETTINGS)
DEFAULT_FILE_STORAGE = 'storages.backends.gs.GSBotoStorage'
STATICFILES_STORAGE = 'storages.backends.gs.GSBotoStorage'

GS_ACCESS_KEY_ID = os.environ.get('GS_ACCESS_KEY_ID', None)
GS_SECRET_ACCESS_KEY = os.environ.get('GS_SECRET_ACCESS_KEY', None)
GS_BUCKET_NAME = os.environ.get('GS_BUCKET_NAME', None)

```

(continues on next page)

(continued from previous page)

```
# Here we are configuring Google Cloud Storage as our default storage provider.
# So whenever we run python manage.py collectstatic, all the static files
# will be uploaded/updated in our Google Cloud Storage.
# This also makes sure that all the static files (when required), will be served
# from the location specified.

# GS_ACCESS_KEY_ID, GS_SECRET_ACCESS_KEY and GS_BUCKET_NAME are the environment
# variables which were created in `cloud-storage-secrets.yml` and are passed to our
# application when the yaml file has been created.
```

## Setting up Django Application

Now that we have the database ready with migrations, collected staticfiles, lets start our application.

```
# Start application
kubectl create -f pollsapi.yml
```

```
yvsssantosh@Sivas-MacBook-Air:~/Desktop/workspace/pollsapi$ kubectl create -f pollsapi.yml
service "polls-api" created
deployment "polls-api" created
yvsssantosh@Sivas-MacBook-Air:~/Desktop/workspace/pollsapi$ kubectl get all
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
deploy/polls-api	1	1	1	1	9s

NAME	DESIRED	CURRENT	READY	AGE
rs/polls-api-6656bb4d9b	1	1	1	9s

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
deploy/polls-api	1	1	1	1	9s

NAME	DESIRED	CURRENT	READY	AGE
rs/polls-api-6656bb4d9b	1	1	1	9s

NAME	DESIRED	SUCCESSFUL	AGE
jobs/polls-collect-static	1	1	19m

NAME	READY	STATUS	RESTARTS	AGE
po/polls-api-6656bb4d9b-qb9w7	1/1	Running	0	9s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
svc/kubernetes	ClusterIP	10.51.240.1	<none>	443/TCP	1d
svc/polls-api	NodePort	10.51.246.101	<none>	8000:30145/TCP	9s

```
yvsssantosh@Sivas-MacBook-Air:~/Desktop/workspace/pollsapi$ kubectl logs polls-api-6656bb4d9b-qb9w7
[2018-07-18 08:30:17 +0000] [1] [INFO] Starting gunicorn 19.9.0
[2018-07-18 08:30:17 +0000] [1] [INFO] Listening at: http://0.0.0.0:8000 (1)
[2018-07-18 08:30:17 +0000] [1] [INFO] Using worker: sync
[2018-07-18 08:30:17 +0000] [7] [INFO] Booting worker with pid: 7
yvsssantosh@Sivas-MacBook-Air:~/Desktop/workspace/pollsapi$
```

Note: 1. Usually, we run the server using `python manage.py runserver`. This is NOT RECOMMENDED for production purposes because of security concerns and extra memory usage. More on this can be found [here](#) Keeping that in mind, this tutorial uses `gunicorn` server to run the application.

- The service type is NodePort for our application, which means that we'll be able to access our application once we expose it using an Ingress.



## Exposing our Application

Lets create an Ingress to expose our application.

```
kubectl create -f pollsapi-ingress.yml
```

Note that creating an ingress may take atleast 5 minutes, or sometimes even more. Please be patient while an ingress is being created

```

yvssantosh@Sivas-MacBook-Air:~/Desktop/workspace/pollsapi$ kubectl create -f pollsapi-ingress.yml
ingress "pollsapi-ingress" created
yvssantosh@Sivas-MacBook-Air:~/Desktop/workspace/pollsapi$ kubectl get ingress
NAME                HOSTS                ADDRESS        PORTS        AGE
pollsapi-ingress    *                   80             53s
yvssantosh@Sivas-MacBook-Air:~/Desktop/workspace/pollsapi$ kubectl describe ingress pollsapi-ingress
Name:                pollsapi-ingress
Namespace:           default
Address:
Default backend:     polls-api:8000 (10.48.3.10:8000)
Rules:
  Host  Path  Backends
  ----  ---  -
  *     *   polls-api:8000 (10.48.3.10:8000)
Annotations:
Events:
  Type    Reason      Age    From                      Message
  ----    -
Normal    ADD         1m     loadbalancer-controller   default/pollsapi-ingress
yvssantosh@Sivas-MacBook-Air:~/Desktop/workspace/pollsapi$ kubectl get ingress
NAME                HOSTS                ADDRESS        PORTS        AGE
pollsapi-ingress    *                   35.241.42.232  80           13m
yvssantosh@Sivas-MacBook-Air:~/Desktop/workspace/pollsapi$ kubectl describe ingress pollsapi-ingress
Name:                pollsapi-ingress
Namespace:           default
Address:             35.241.42.232
Default backend:     polls-api:8000 (10.48.4.12:8000)
Rules:
  Host  Path  Backends
  ----  ---  -
  *     *   polls-api:8000 (10.48.4.12:8000)
Annotations:
  backends:      {"k8s-be-30145--f683f49d9c23ceca":"HEALTHY"}
  forwarding-rule: k8s-fw-default-pollsapi-ingress--f683f49d9c23ceca
  target-proxy:   k8s-tp-default-pollsapi-ingress--f683f49d9c23ceca
  url-map:        k8s-um-default-pollsapi-ingress--f683f49d9c23ceca
Events:
  Type    Reason      Age    From                      Message
  ----    -
Normal    ADD         13m     loadbalancer-controller   default/pollsapi-ingress
Normal    CREATE      11m     loadbalancer-controller   ip: 35.241.42.232
Normal    Service     55s (x6 over 11m) loadbalancer-controller   default backend set to polls-api:30145
yvssantosh@Sivas-MacBook-Air:~/Desktop/workspace/pollsapi$

```

To check the status of the ingress, see below

As expected, it took around 10 minutes for the ingress to setup properly. Navigate to the ingress address generated i.e. <http://35.241.42.232/> in order to access our application.

## Documentation for Polls API

For any queries, please create a new issue [here](#)

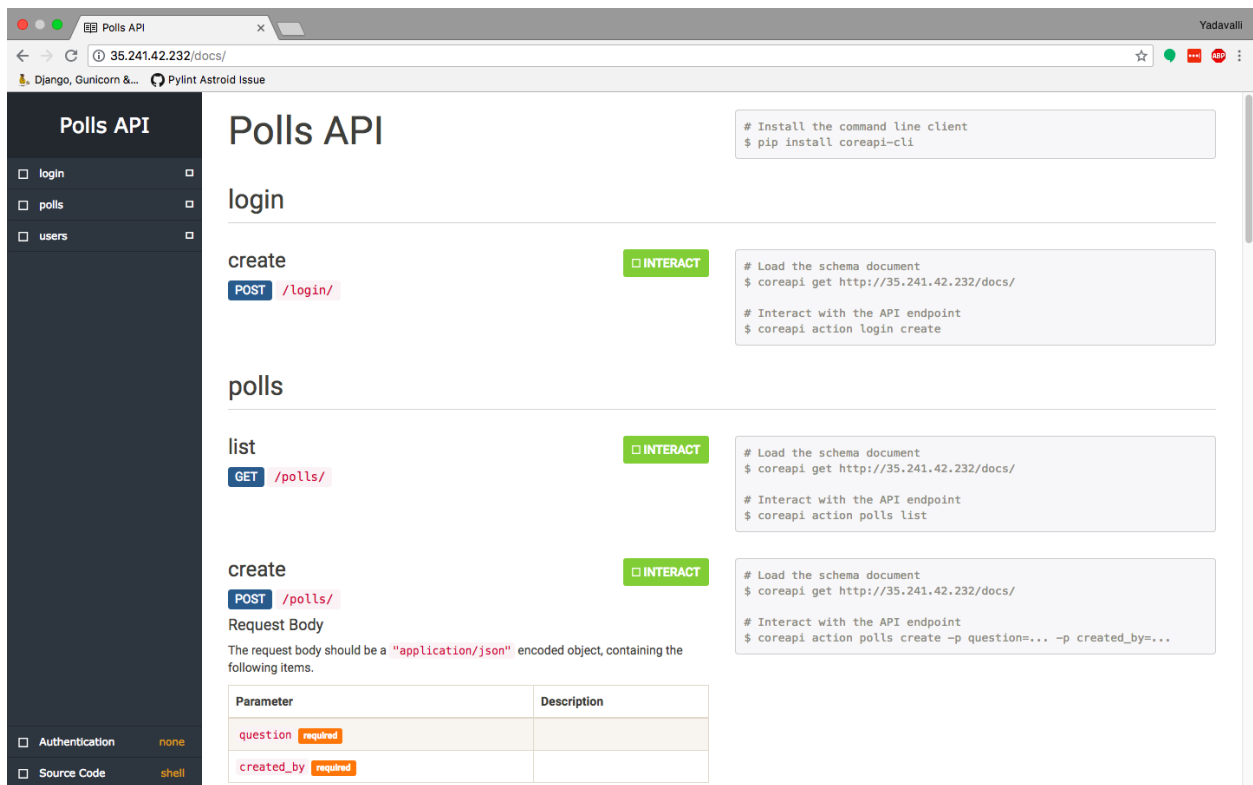
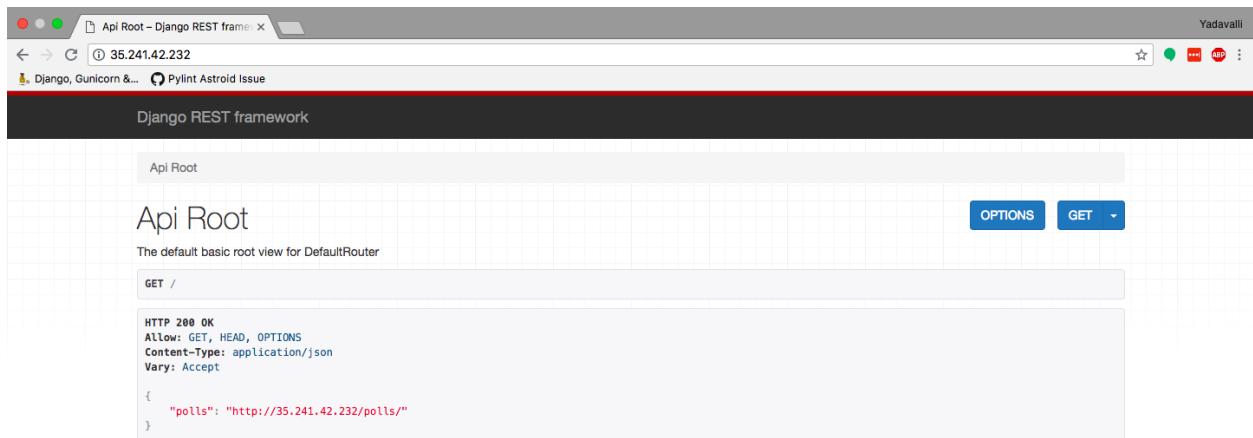
## 6.5 Amazon Web Services

AWS is a cloud platform service from amazon, used to create and deploy any type of application in the cloud.

AWS is a Cloud platform service offering compute power, data storage, and a wide array of other IT solutions and utilities for modern organizations. AWS was launched in 2006, and has since become one of the most popular cloud platforms currently available.

We should have an account in [AWS](#) to use aws services. It offers many featured services for compute, storage, networking, analytics, application services, deployment, identity and access management, directory services, security and many more cloud services.

To use AWS for python, check <https://aws.amazon.com/developer/language/python/>



We can use [Boto3](#) (python package) which provides interfaces to Amazon Web Services, it makes us easy to integrate our Python application, library, or script with AWS services

Boto3 is the Amazon Web Services (AWS) Software Development Kit (SDK) for Python, which allows Python developers to write software that makes use of services like Amazon S3(Simple storage service) and Amazon EC2(Elastic Compute Cloud).

### 6.5.1 Amazon Elastic Cloud Compute (EC2)

[Amazon Elastic Compute Cloud \(Amazon EC2\)](#) is a web service that provides resizable computing capacity.

We use Amazon EC2 to launch a virtual servers and also configure security , networking, and manage storage. It enables us to scale up or down depending the requirement.

- It provides virtual computing environments called as *instances*
- Various configurations of CPU, memory, storage, and networking capacity are available for our instances, known as instance types.

### 6.5.2 Amazon Elastic Beanstalk

[AWS Elastic Beanstalk](#) is a service for deploying and scaling web applications and services. Elastic Beanstalk will also run instances (Computing environments) EC2, and it has some additional components like Elastic Load Balancer, Auto-Scaling Group, Security Group.

We pay only for the EC2 instances or S3 buckets and aws-DB we use and the other features like Elastic Load Balancer, Auto-Scaling Group, Security Group in Elastic Beanstalk do not cost anything.

### 6.5.3 Amazon Lambda

[Amazon Lambda](#) is a computing service which automatically manages the server. AWS Lambda executes our code only when needed and scales automatically, from a few requests per day to thousands per second.

We only pay for the compute time we consume , and there will be no charge if the code is not running.

The initial purpose of lambda is to simplify building on-demand applications that are responsive to events. AWS starts a Lambda instance within milliseconds of an event.

#### Deployment in AWS services:

Once we connect with the server using ssh , then the deployment will be same for all services. Which is same as in the example mentioned in the next chapter

---

## 6.6 Deploying a Django application

This chapter tells the basics of deploying a [django application](#) using gunicorn, nginx and supervisord.

## 6.6.1 Prerequisites

### Knowledge

- Django basics
- (Optional) finish the book about [django-rest-apis](#) .

### Resources

A Unix server for deploying the app , connected with a SSH(preferred).

---

[Django](#) is a free and open source web application framework, written in Python. Django has a lot of inbuilt set of components that helps you to develop websites faster and easier.

[Gunicorn](#) is a simple, light-weight Python WSGI HTTP Server for UNIX. [WSGI](#) is the Web Server Gateway Interface. It is a specification that describes how a web server communicates with web applications, and how web applications can be chained together to process one request.

[Nginx](#) is a high-performance HTTP server, reverse proxy, load balancer and static files loader.

[Supervisord](#) is a process-control system which allows us to monitor and control a number of processes on UNIX operating system.

---

### Let's start with our server

Once we create our server and let's login to the server via SSH,

```
$ ssh root@IP_ADDRESS_OF_SERVER
```

Now we have to install the prerequisites, run these commands

```
$ sudo apt-get update
$ sudo apt-get install git python-pip python-dev virtualenv virtualenvwrapper
$ sudo apt-get install postgresql postgresql-contrib
$ pip install --upgrade pip
```

Now let's [configure the virtual-env wrapper](#)

After setting-up the virtualenvwrapper, create a virtualenv

```
$ mkvirtualenv env-name
```

From within our virtual-env, install:

```
(env-name) $ pip install django gunicorn psycpg2
```

Let's clone the repo in home folder, pull the application from Git, we use this repo <https://github.com/anmolakhilesh/django-polls-rest>

```
$ cd ~
$ git clone https://github.com/anmolakhilesh/django-polls-rest
```

Now we have to add permissions to the `manage.py` file

```
$ cd /django-polls-rest/  
$ chmod 755 manage.py
```

Now install the requirements

```
(env-name) $ pip install -r requirements.txt
```

Now set up PostgreSQL

Create a file `.env` and add these lines in that

```
$ export POSTGRES_DB = pollsdB  
$ export POSTGRES_USER = polls_admin  
$ export POSTGRES_PASSWORD = polls_password  
$ export POLLSAPI_PG_HOST = 127.0.0.1
```

Create a postgres Database

```
$ sudo -u postgres psql
```

After running the above command, we will be logged inside PostgreSQL terminal, now lets create our db and user

```
> CREATE DATABASE pollsdB;  
> CREATE USER polls_admin WITH PASSWORD 'polls_password';  
> ALTER ROLE polls_admin SET client_encoding TO 'utf8';  
> ALTER ROLE polls_admin SET default_transaction_isolation TO 'read committed';  
> ALTER ROLE polls_admin SET timezone TO 'UTC';  
> ALTER USER polls_admin CREATEDB;  
> GRANT ALL PRIVILEGES ON DATABASE pollsdB TO polls_admin;  
  
> \q  # to quit the shell
```

Make sure that these details match the details in the `.env` file. Exit the PostgreSQL shell by typing `\q`.

Now as the DB is ready , we can run migrations command inside the repo folder.

```
# migrations  
(env-name) $ python manage.py migrate  
  
# Create a supervisor, let's  
(env-name) $ python manage.py createsuperuser
```

Now postgres-db is setted, now we have to set up the server

---

## Using gunicorn

```
(env-name) $ pip install gunicorn
```

After installing gunicorn , now run it

```
# starts the server  
(env-name) $ gunicorn polls_rest.wsgi
```

It will run the app , we can check `IP_ADDRESS_OF_SERVER:8000` , `IP_ADDRESS_OF_SERVER:8000/admin` . It will not have any css , as the gunicorn only serves the application. We will be serving static files using *nginx* .

To exit it press `Ctrl+C` .

```
# starts the server by binding it to a specific port
(env-name) $ gunicorn --bind 0.0.0.0:8888 polls_rest.wsgi

# running with a config file
(env-name) $ gunicorn -c /path/to/config/file polls_rest.wsgi

# running in daemon mode
(env-name) $ gunicorn --daemon polls_rest.wsgi
```

If it is in daemon-mode, then exit it with `kill gunicorn` , which will kill the gunicorn process.

To have a *gunicorn config file* for gunicorn , we write the config file in a `.py` .

### Using nginx

By using gunicorn, we were able to run the application, but without styles as the gunicorn only runs the application and does not serve the static files django does not serve static file except in development.

We will use *nginx* to serve the static files , *nginx* will first get the request, and it will send it to gunicorn.

To install *nginx*

```
$ sudo apt-get install nginx
```

let's configure *nginx*

So, **create a file** `/etc/nginx/sites-available/pollsapp` and add the following

```
server {
    listen 80;          #L1
    server_name SERVER_DOMAIN_OR_IP_ADDRESS_OF_SERVER;  #L2

    location = /favicon.ico { access_log off; log_not_found off; }  #L3

    location /static/ {  #L4
        root /home/django-polls-rest;
    }

    location / {        #L5
        include proxy_params;
        proxy_pass http://unix:/home/django-polls-rest/polls_rest.sock;
    }
}
```

- `#L1` and `#L2` lines defines where our *nginx* server should run.
- `#L3` line ignores any errors related to the favicon.
- `#L4` block `location /static/` defines the location of static files.
- `#L5` block `location /` tells the socket(gunicorn socket) to communicate.

After this, we have to enable this config file by linking with the `sites-enabled` folder.

```
$ ln -s /etc/nginx/sites-available/pollsapp /etc/nginx/sites-enabled
```

We link the above file to `sites-enabled`, so that it will be included in the main nginx settings file `/etc/nginx/nginx.conf`

After enabling the config file, we can check nginx configuration by

```
$ sudo nginx -t
```

If the configuration file is correct, then we should see this

```
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

Now we have to mention the static files directory of our app in `settings.py` file. So add this line in `settings.py`

```
STATIC_ROOT = os.path.join(BASE_DIR, 'static/')
```

After adding this line, we have to perform run `collectstatic` command

```
(env-name) $ python manage.py collectstatic
```

Let's run the app

```
(env-name) $ gunicorn --daemon --workers 3 --bind unix:/home/django-polls-rest/polls_
↪rest.sock polls_rest.wsgi
```

The `/home/django-polls-rest/polls_rest.sock` file is a unix-socket file which will be created automatically. And this file will enable Gunicorn and Nginx to communicate with each other.

Now Restart Nginx for changes to take effect.

```
$ sudo service nginx restart
```

This will run our app in the `http://IP_ADDRESS`

Point to remember, check `ALLOWED_HOSTS` in `settings.py` to have you host name or ip address of server.

## Configuring Gunicorn with Supervisor

**Supervisor** is a process monitoring tool, which can restart any process if the process dies or gets killed for some reason.

At present we are manually starting gunicorn in daemon to run our app, Suppose if this gunicorn process closes or gets killed due to some reason then we have to manually start it again. To monitor our processes we use *Supervisor*, So that supervisor controls the gunicorn process.

To install supervisor

```
$ sudo apt-get install supervisor
```

Let's add a configuration file `pollsapi.conf` for our application in `/etc/supervisor/conf.d/` folder, the `conf.d` folder will have all our config files.

```
[program:pollsapi]      #L1
directory=/home/django-polls-rest/polls_rest      #L2
command=/home/.virtualenvs/demo-polls-1/bin/gunicorn --workers 3 --bind unix:/home/
↳django-polls-rest/polls_rest.sock polls_rest.wsgi      #L3
autostart=true          #L4
autorestart=true        #L5
stderr_logfile=/var/log/pollsapi.err.log          #L6
stdout_logfile=/var/log/pollsapi.out.log          #L7
```

Let's understand the config file we have written,

- #L1 line `[program:pollsapi]` names the program( or process ) as *pollsapi*, which can be used as

```
$ sudo supervisorctl start pollsapi
```

- #L2 line `directory` is the path to our project.
- #L3 line `command` is the command to start our project
- #L4 lines `autostart` tells the script to start on system boot.
- #L5 line `autorestart` tells the script to restart when it closes for some reason
- #L6 `stderr_logfile` which will store the error logs & #L7 `stdout_logfile` will store the non-error logs.

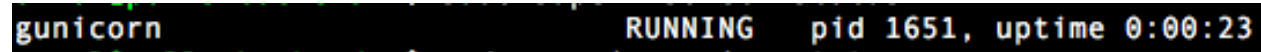
Now lets save this file and update supervisor

```
$ sudo supervisorctl reread
$ sudo supervisorctl update
$ sudo supervisorctl reload
```

Check the supervisor status .

```
$ sudo supervisorctl status
```

This will show

A terminal screenshot showing the output of the supervisorctl status command. It displays 'gunicorn' in green, followed by 'RUNNING' in white, and 'pid 1651, uptime 0:00:23' in white. The background is black with some faint, colorful text from other processes visible in the background.

To check gunicorn processes

```
$ ps ax | grep gunicorn
```

This command lists all the processes running with gunicorn

To check if the app is running , let's do curl

```
$ curl 0.0.0.0:8000
```

After configuring gunicorn with supervisor, let's restart our nginx

```
$ systemctl restart nginx
```

Now our app should be running on `http://IP_ADDRESS_OF_SERVER`



## CHAPTER 7

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`