My Awesome Project

Andrew Chen Wang

Aug 15, 2020

CONTENTS:

CHAPTER

ONE

HOW TO - PROJECT DOCUMENTATION

1.1 Get Started

Documentation can be written as rst files in the *my_awesome_project/docs/_source*.

To build and serve docs, use the commands:

```
docker-compose -f local.yml up docs
```

Changes to files in *docs/_source* will be picked up and reloaded automatically.

Sphinx is the tool used to build documentation.

1.2 Docstrings to Documentation

The sphinx extension apidoc is used to automatically document code using signatures and docstrings.

Numpy or Google style docstrings will be picked up from project files and availble for documentation. See the Napoleon extension for details.

For an in-use example, see the page source for *Users*.

To compile all docstrings automatically into documentation source files, use the command:

```
make apidocs
```

This can be done in the docker container:

```
docker run --rm docs make apidocs
```

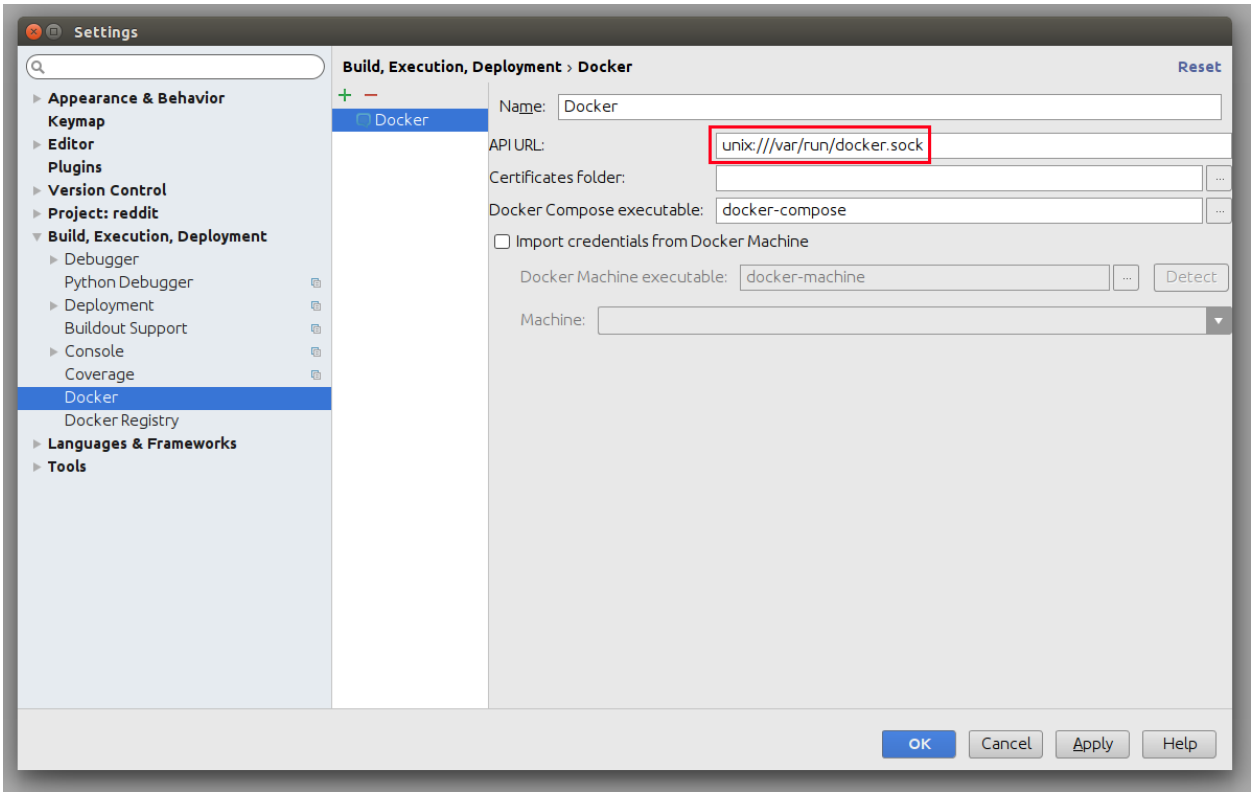
CHAPTER

TWO

DOCKER REMOTE DEBUGGING

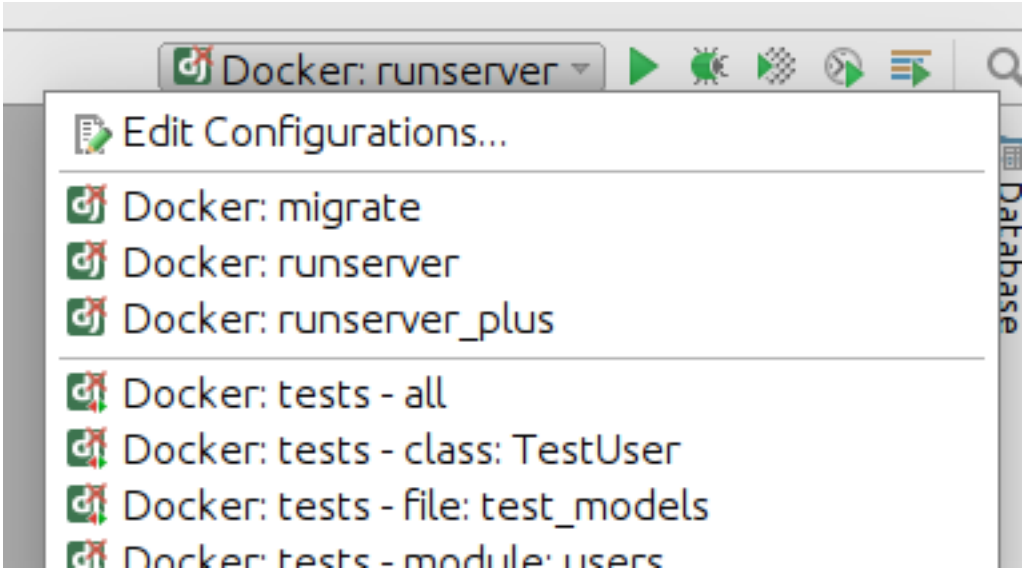
To connect to python remote interpreter inside docker, you have to make sure first, that Pycharm is aware of your docker.

Go to *Settings* > *Build, Execution, Deployment* > *Docker*. If you are on linux, you can use docker directly using its socket *unix:///var/run/docker.sock*, if you are on Windows or Mac, make sure that you have docker-machine installed, then you can simply *Import credentials from Docker Machine*.



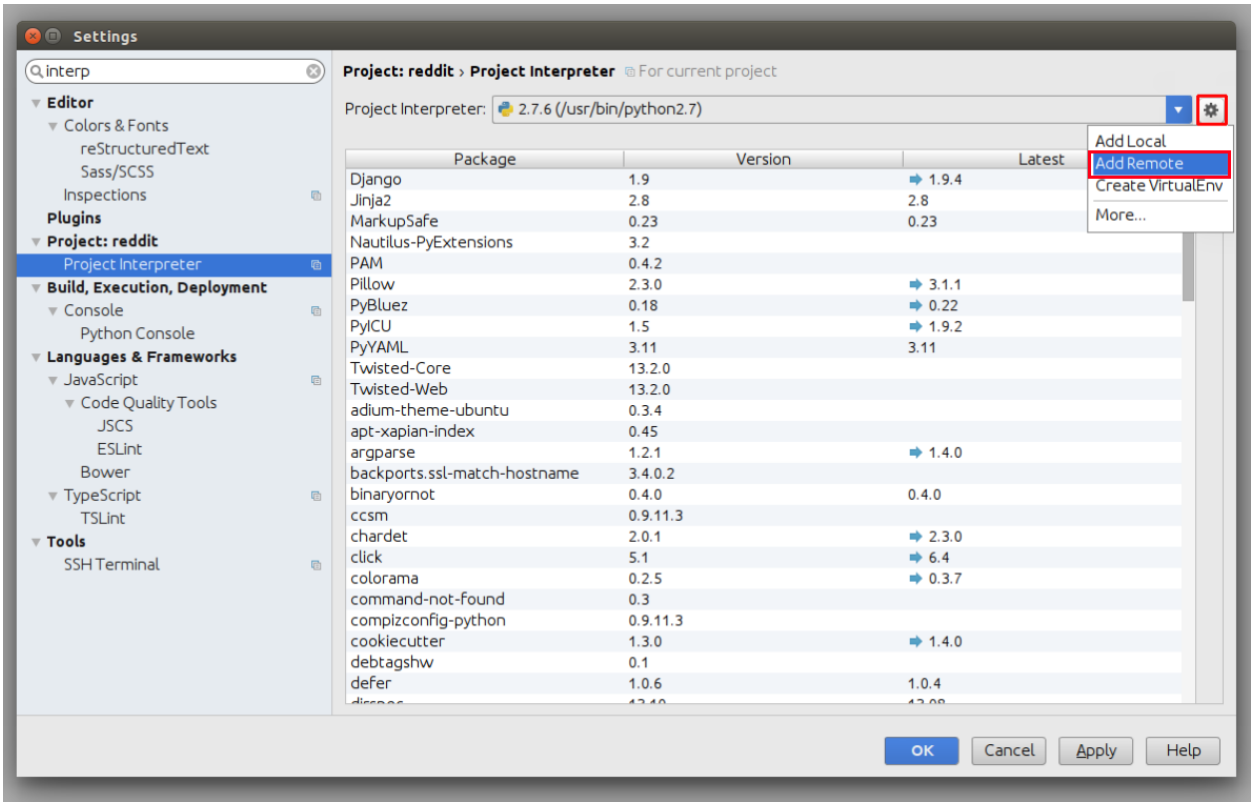
2.1 Configure Remote Python Interpreter

This repository comes with already prepared "Run/Debug Configurations" for docker.

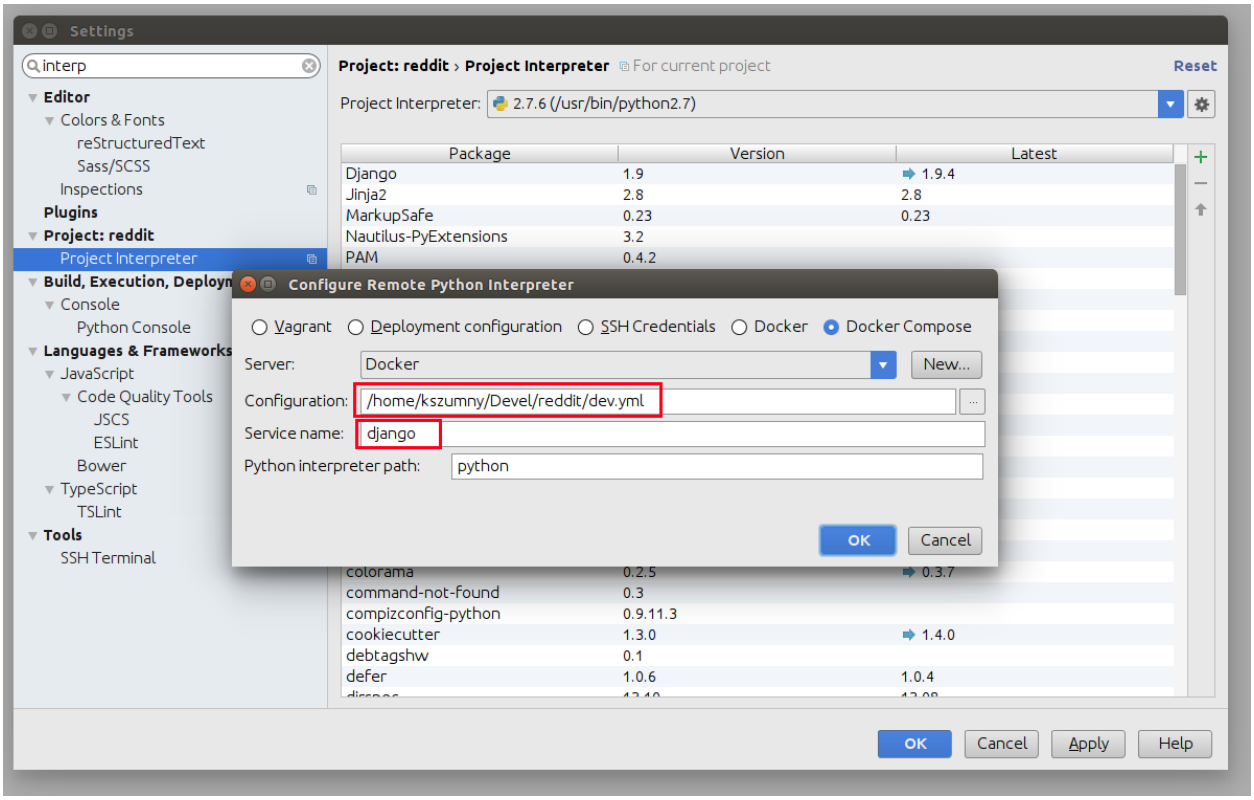


But as you can see, at the beginning there is something wrong with them. They have red X on django icon, and they cannot be used, without configuring remote python interpreter. To do that, you have to go to *Settings* > *Build, Execution, Deployment* first.

Next, you have to add new remote python interpreter, based on already tested deployment settings. Go to *Settings* > *Project* > *Project Interpreter*. Click on the cog icon, and click *Add Remote*.

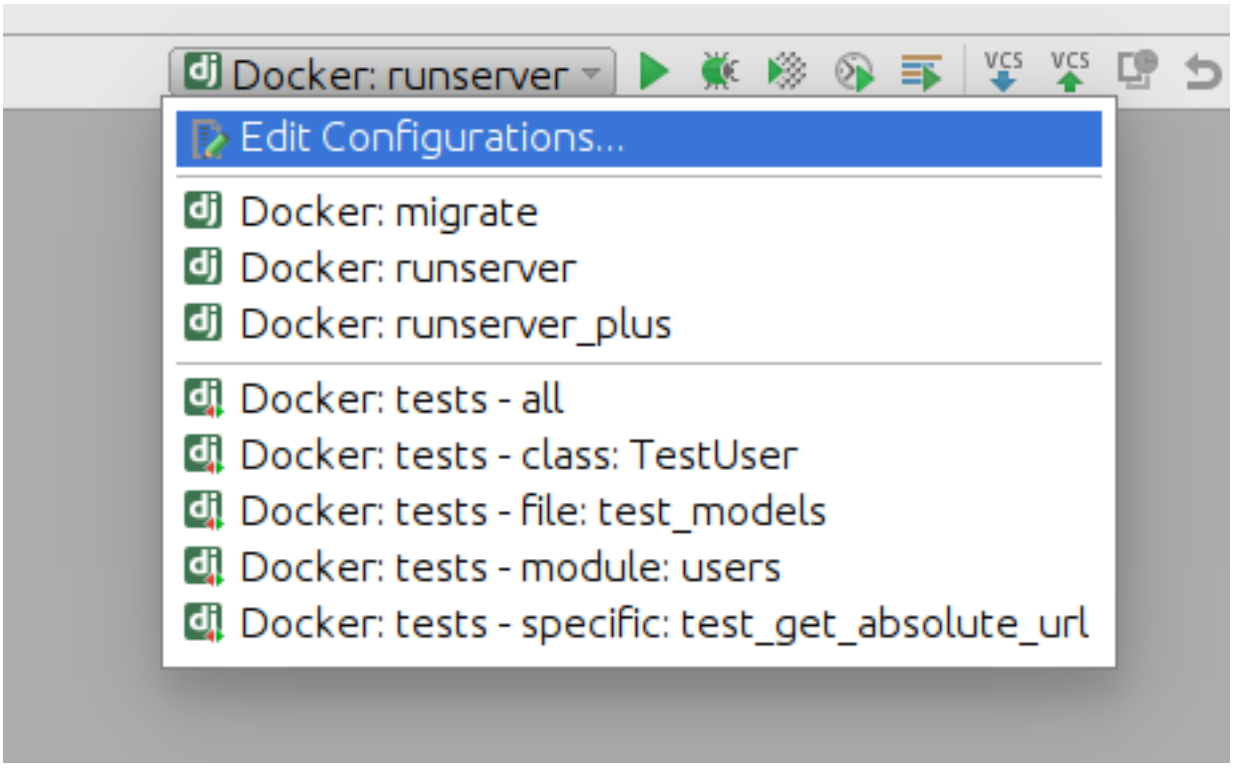


Switch to *Docker Compose* and select *local.yml* file from directory of your project, next set *Service name* to *django*



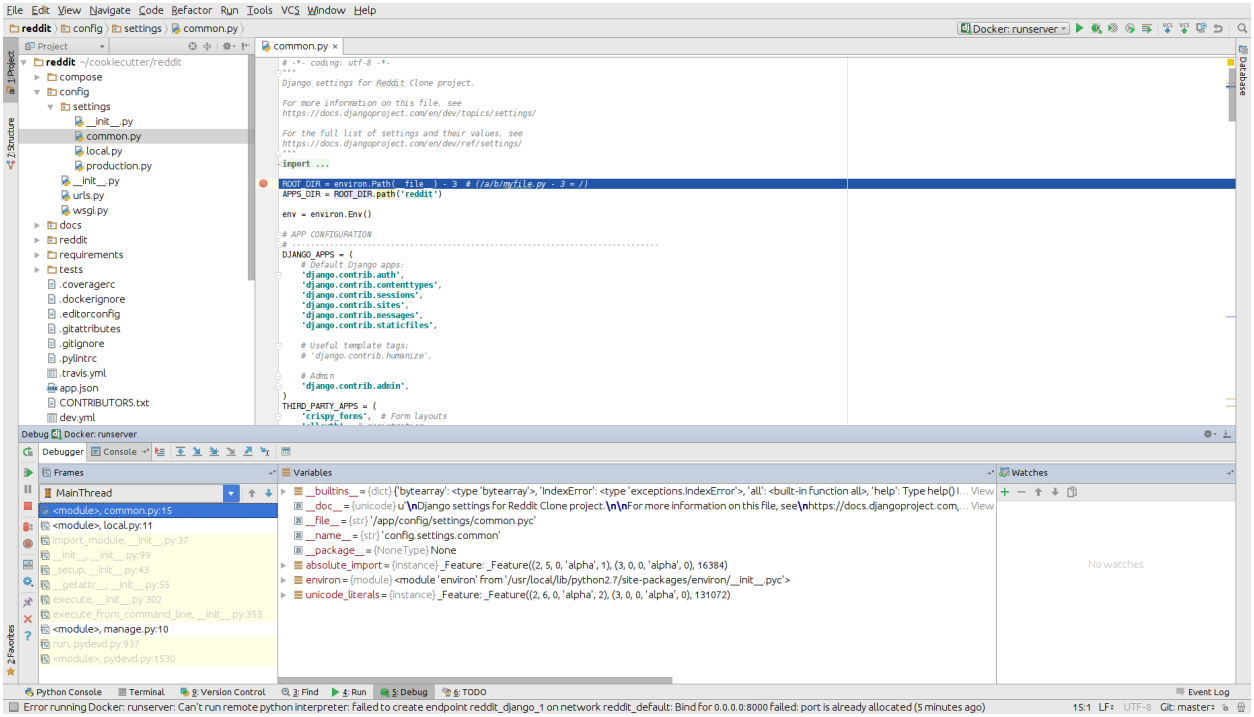
Having that, click *OK*. Close *Settings* panel, and wait few seconds...

After few seconds, all *Run/Debug Configurations* should be ready to use.

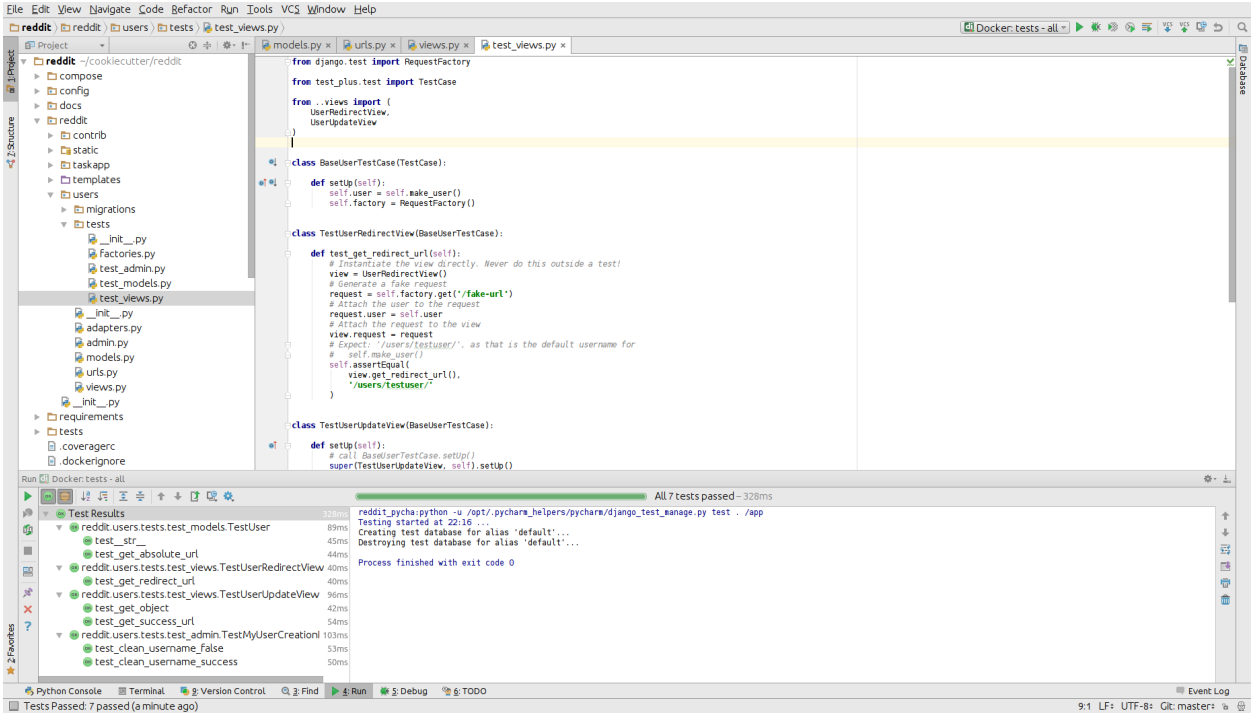


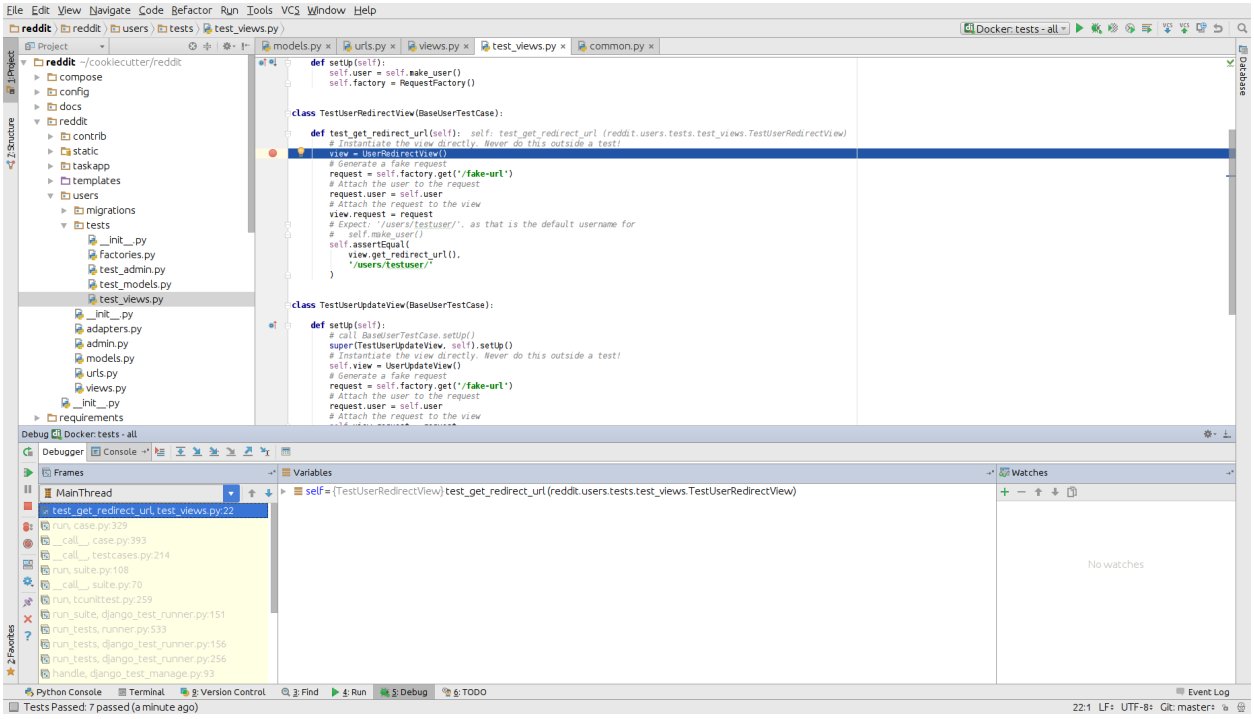
Things you can do with provided configuration:

- run and debug python code

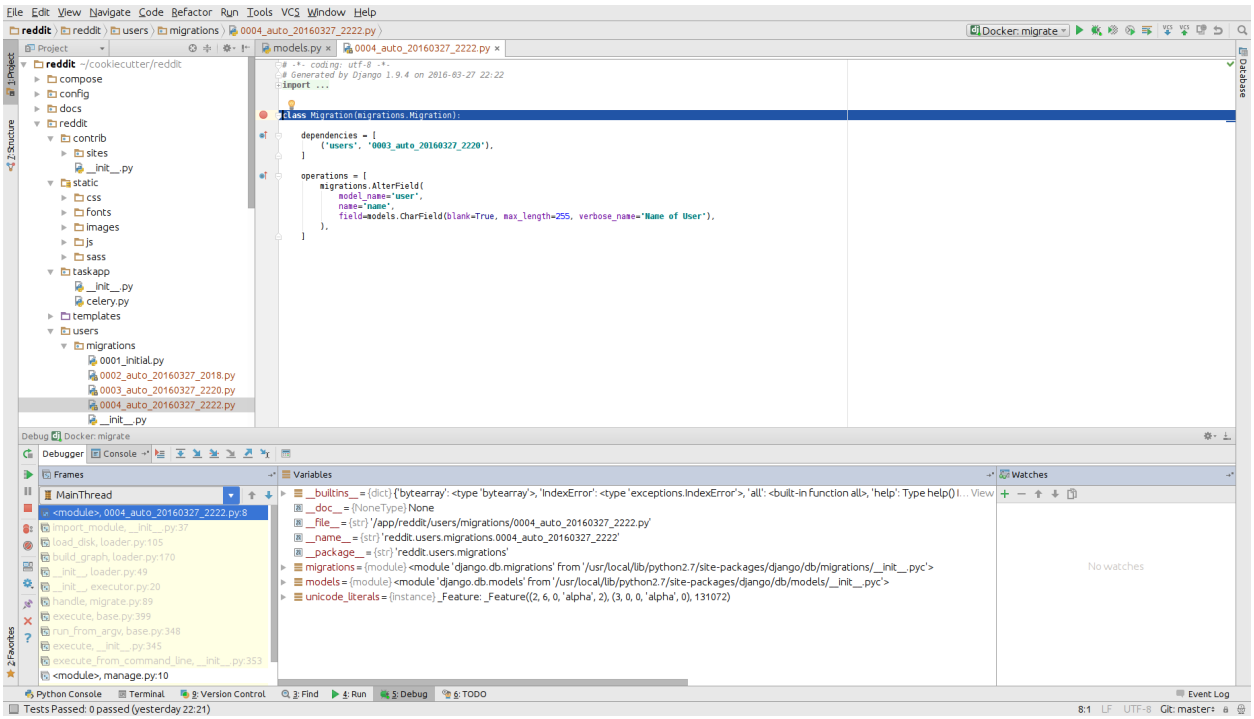


- run and debug tests





- run and debug migrations or different django management commands



- and many others..

2.2 Known issues

- Pycharm hangs on “Connecting to Debugger”

This might be fault of your firewall. Take a look on this ticket - https://youtrack.jetbrains.com/issue/PY-18913

- Modified files in *.idea* directory

Most of the files from *.idea/* were added to *.gitignore* with a few exceptions, which were made, to provide “ready to go” configuration. After adding remote interpreter some of these files are altered by PyCharm:

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

        modified:   .idea/project_name.iml

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

In theory you can remove them from repository, but then, other people will lose a ability to initialize a project from provided configurations as you did. To get rid of this annoying state, you can run command:

```
$ git update-index --assume-unchanged my_awesome_project.iml
```

CHAPTER

THREE

USERS

Starting a new project, it's highly recommended to set up a custom user model, even if the default User model is sufficient for you.

This model behaves identically to the default user model, but you'll be able to customize it in the future if the need arises.

class my_awesome_project.users.models.**User**(**args*, ***kwargs*)

Default user for My Awesome Project.

exception DoesNotExist

exception MultipleObjectsReturned

get_absolute_url()

Get url for user's detail view.

Returns URL for user detail.

Return type str

name

First and last name do not cover name patterns around the globe

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search