Try-try Documentation

Release 0.1

Try-try team

August 19, 2012

CONTENTS

CHAPTER

ONE

PROJECT CONCEPTS

Try-try is the Django application which allows for developers to quite easily create test flows. There is quite a lot of similar web services, but the majority of them use browser-side javascript interpretation of the language in question, which effectively limit the set of languages to be interpreted that way.

Our apprach is to use "fair server-side intepreter" in isolated environment. Every new session creates a virtual environment where users (believe you or not) have superuser privileges.

Test writer writes a specially crafted `steps.py` file, containing test steps, one by one, and define the environment, which everything should be executed.

See next modules of the documentation to get more detailed view about the application and feel free to visit our http://www.try-try.me reference installation.

CHAPTER

TWO

PROJECT INSTALLATION

We should admit. The project is quite hard to install. And the first thing, it is pretty much useless in Windows environment.

We tested its work on Debian, Ubuntu and Gentoo. Ubuntu should be considered as the best variant for installing on, no matter locally or remotely.

2.1 Local installation

Local installation differs from remote one in that sense that it is considered as "development" rather than production. Usually you don't need no LXC containers, no special webserver, nothing like this. Just a plain old:

```
$ ./manage.py runserver
```

Although even that it can be not easy to install.

So, there are the steps you need to make:

Clone the repository, create a new virtual environment, and install all the dependencies

```
$ git clone https://github.com/imankulov/trytry.git
$ cd ./trytry
$ mkvirtualenv --system-site-packages trytry
$ workon trytry
$ pip install -r requirements.pip
```

Copy localsettings.py.example to localsettings.py and adjust it according to your needs

```
$ cp -a trytry/localsettings.py.example trytry/localsettings.py
$ editor trytry/localsettings.py
```

Type `./manage.py trytry_sanity_check`. It is a special command which tries to estimate the envornment it works within, and give some advice on how to fix. Usually it asks to setup a timelimit system package, which is as trivial as

```
$ sudo apt-get install timelimit
```

And then do

```
$ ./manage.py syncdb --migrate
$ ./manage.py runserver
```

Then go to http://localhost:8000 . I hope it will work for you.

Okay, it's not very hard when it works, but if it doesn't .. yeah, it doesn't.

2.2 Remote installation

We wrote a fabric script which is intender to transform your bare Ubuntu server to a fully fledged try-try platform. The script and accompanying files reside in the `deploy` subdirectory of the project.

First before, make sure you have root ssh access to the server. Many Ubuntu-based virtual servers don't provide SSH root access, assuming that you use "sudo" when it is needed, but the fabric script we created won't work unless you run it as root.

Then, review `server_configs` directory, and change something according to your needs.

Then, launch a basic setup installation command:

```
fab -H root@servername setup
```

The same command can be used to update the project. Then, providing you need LXC setup, run:

```
fab -H root@servername lxc_setup
```

The command creates a number of LXC containers, as described in the to of `fabfile.py`.

Hopefully, it work out. If so, then visit the webpage of your remote server. Nginx should respond with a funny collage of some geeky guys.

CHAPTER

THREE

MODULE WRITING HOWTO

We hope you already familiarized with the system concept, and willing to write some test flows for the application.

You don't need to start from scratch here, to simplify your job we prepared an application template. Clone the repository from Github

```
$ git clone https://github.com/imankulov/try_app_template.git
```

Then, in the project directory, type:

```
$ ./manage.py startapp --template path/to/try_app_template my_app
```

A new app named *my_app* will be created, you should add it to the list of your installed apps in `settings.py` or `localsettings.py`.

The core of the test application is the `steps.py` file where a whole test flow is described.

Below is a summary description of the file contents:

3.1 *__flow__* variable

Flow variable describes the flow in a nutshell, and contains the links to steps which should be done to successfully pass the test.

```
from trytry.core.utils.lxc import lxc_setup, lxc_teardown

__flow__ = {
    # class names in this module, which will be used as steps
    'steps': ['Step1', 'Step2'],
    # the name of LXC container template which will be used to set up
    # a base container for your user.
    'lxc_container': 'python',
    # Setup and teardown functions. You can define them as functions or,
    # exactly like steps, as strings within your module.
    # Function accept one parameter: an initialized Flow object
    'setup': lxc_setup,
    'teardown': lxc_teardown,
    # the name of your module
    'name': 'Simple Bash',
    # The short name of your module, will be used as a part of urls in tests
    'url': 'simple_bash',
    # Detailed description of your module
    'description': __doc__,
}
```

3.2 Step classes

Step classes are ordinary classes, but it is more convenient to inherit them from any generic step. There are three generic steps at your service:

- `trytry.core.steps.GenericStep`: a generic step to execute an arbitrary command in a virtual LXC environment
- `trytry.simple_bash.steps.GenericStep` a generic step to execute bash one-liners. Can store the state of variables between commands
- `trytry.simple_python.steps.GenericStep` a generic step to execute Python one-liners. Can store the state of variables between commands

3.3 Setup and teardown functions

The test flow calls setup function before starting the first test in the flow. the `trytry.core.utils.lxc.lxc_setup()` is a good way to start.

Likewise, tye test flow calls teardown function after all tests have been completed, and the `trytry.core.utils.lxc.lxc_teartown()` should be used as a *lxc_setup* counterpart.

CHAPTER

FOUR

LXC CONFIGURATION

4.1 Default configuration

Try-try security model takes advantage of LXC containers. You shouldn't care about LXC installation, if you just installed the package to play around on your localhost intend to setup the web service in a *very* trusted environment.

We assume that you have a virtual or a real (bare-metal) server with operating system supporting LXC containers. The server is probably dedicated to try-try project, you have root privileges to it (it is required). Instructions below assume that configuration is made for Ubuntu distribuion.

First before, install the LXC package

```
$ sudo apt-get install lxc
```

On-boot container launch is redundant in our environment. Open the file `/etc/default/lxc` and change value `LXC_AUTO` from `"true"` to `"false"`

The next step is to create one or more templates to work with. Create a new file named `lxc.conf` with two lines:

```
lxc.network.type = empty
lxc.aa_profile = unconfined
```

Then create a new base template with minimal Ubuntu installation.

```
$ sudo lxc-create -n try-try -t ubuntu -f lxc.conf -- --trim
```

Option `--trim` creates minimalistic installation of the system.

Then create and configure a bunch of clones of this distibution. Feel free to create as much distributions as you like. It's fun

```
$ sudo lxc-clone -o try-try -n python
$ sudo lxc-clone -o try-try -n php
...
```

As these template images don't have access to network, if you need extra packages in there, you should install them by chrooting in the root directory.

For instance, below is a command which can be used to install PHP accessible via command line in the image with the name "php"

```
$ sudo chroot /var/lib/lxc/php/rootfs bash -c 'apt-get update && apt-get install --yes --force-yes ph
```

Then you can check how it works by issuing the command

```
$ lxc-start -n php -- php -r '$foo = "hello world\n"; echo $foo;'
hello world
```

For more information about LXC managing visit https://help.ubuntu.com/12.04/serverguide/lxc.html

4.2 Speed-up lxc cloning

By default cloning a new environment takes about 10 seconds, but this timespan can be significantly improved by leveraging btrfs snapshots.

```
# apt-get install btrfs-tools
# mkfs.btrfs /dev/<device-name>
# mount /dev/<device-name> /var/lib/lxc
# echo "/dev/<device-name> /var/lib/lxc/ btrfs defaults 0 0" >> /etc/fstab
```

Enjoy watching the list of btrfs subvolumes while creating new virtual images

```
# btrfs subvolume list /var/lib/lxc/
```

- GitHub repository is available at http://github.com/imankulov/trytry
- This is a template to write new tests http://github.com/imankulov/try_app_template
- Reference installation resides here http://try-try.me