

DevOps



Build and Deploy Python Web Part 2



by Rom Freiman

Jun 08, 2016

In our previous post, [How to Deploy with Jenkins: Best Practices](#) we discussed security, backup and other topics. In this post we will provide the instructions required to use Jenkins to build and deploy Python web applications. We will use a UNIX-based OS, and in this specific case, Ubuntu Server 15.10 and latest version of Jenkins.

Jenkins Installation

To enable the Jenkins installation (on Ubuntu) to proceed smoothly, make sure you have installed JDK and JRE on your machine. The versions that are suggested on the Jenkins site are `openjdk-7-jre` and `openjdk-7-jdk` ([Learn how to install JDK and JRE](#)).

The commands to install Jenkins on Ubuntu

Proceed to install [Jenkins](#), which should be pretty easy using the following 4 commands:

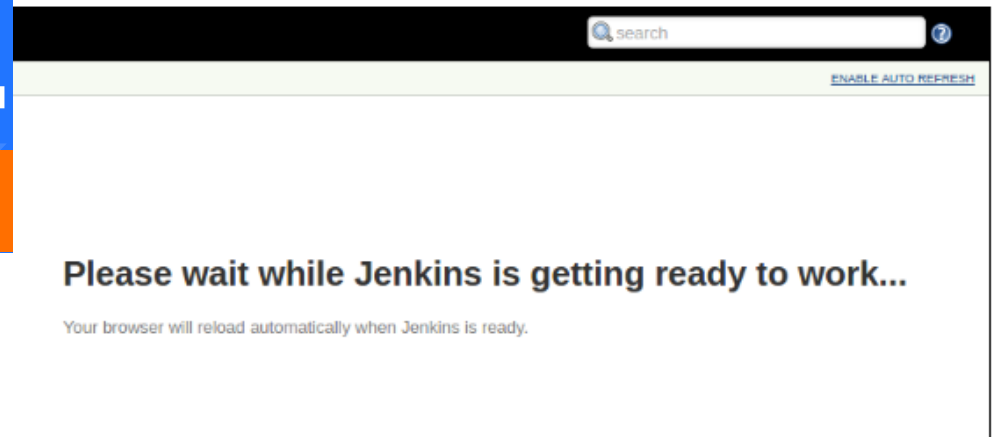
```
wget -q -O - https://jenkins-ci.org/debian/jenkins-ci.org.key |
```

```
sudo apt-key add -
```

```
sudo apt-get update sudo apt-get install jenkins
```



on port 8080. After installing Jenkins, enter the IP address of the machine. You will notice the message about booting up Jenkins and preparing for work,

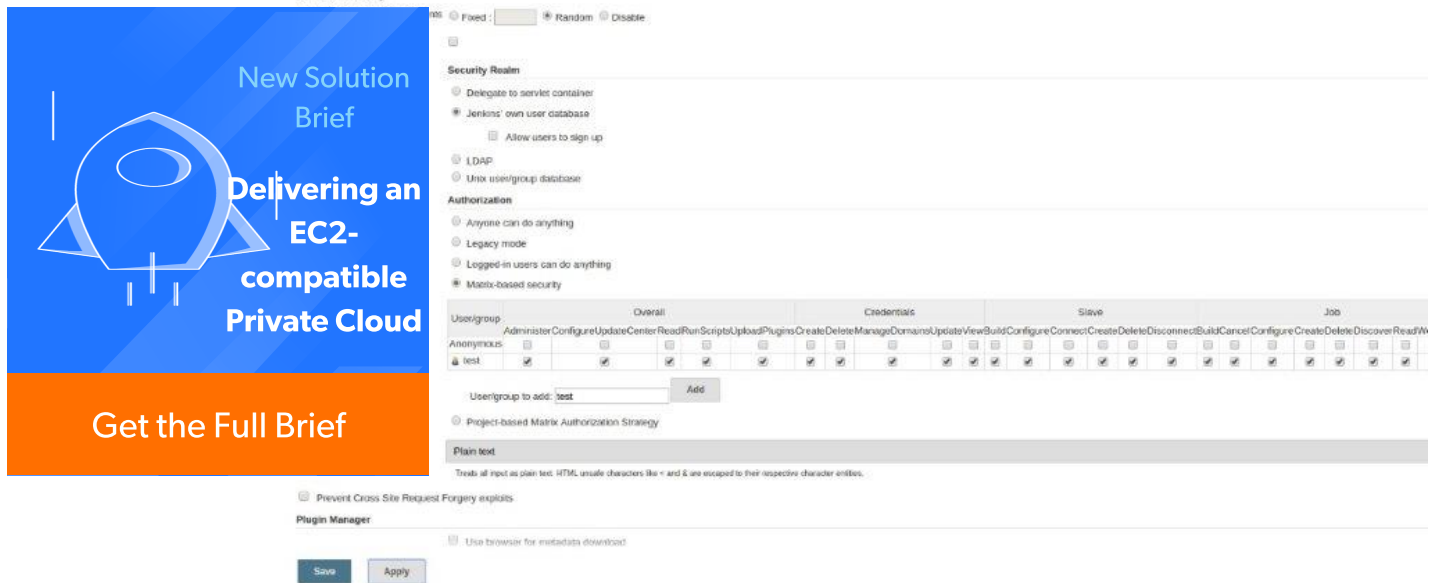


Now we're ready to move to the next step, which involves applying one of the basic and well known practices to secure Jenkins.

Under Manage Jenkins there is a Configure Global Security section where it is possible to configure different types of authentication. For this use case we will use the Jenkins user database after enabling security. Of importance here is the option "Allow users to sign up" where you can enable or disable new users to use your Jenkins. As mentioned above, Jenkins should be deployed in a private network and be restricted to a specific set of users. One option is for the administrator to add users. Another option is to use GitHub and similar organizations. You can add GitHub users to your organizations on the [GitHub](#) site and then you can configure the Jenkins plugin. This plugin reads the organization's structure and allows specific GitHub users to login into Jenkins.

Under Authorization there are several ways of configuring the permissions that you want to provide to particular users. In this section we will not go into detail about securing Jenkins, instead we will use the Matrix-based security where you can easily enable or disable permissions with simple checkboxes. More details can be [found on the Jenkins site – standard security setup](#).

Thinking about incorporating microservices? Find out more [here](#).



The screenshot shows the Jenkins Global Security Configuration page. On the left, there is a blue banner with a rocket icon and the text "New Solution Brief Delivering an EC2-compatible Private Cloud Get the Full Brief". The main content area is titled "Configure Global Security" and includes sections for "Enable security", "Security Realm", "Authorization", "User/group", "Project-based Matrix Authorization Strategy", and "Plugin Manager".

Security Realm

- ☐ Delegate to server container
- ☒ Jenkins' own user database
 - ☐ Allow users to sign up
- ☐ LDAP
- ☐ Unix user/group database

Authorization

- ☐ Anyone can do anything
- ☐ Legacy mode
- ☐ Logged-in users can do anything
- ☒ Matrix-based security

User/group	Overall	Credentials	Slave	Job
Administrator	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
test	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

User/group to add:

☐ Project-based Matrix Authorization Strategy

Plain text

☐ Prevent Cross Site Request Forgery exploits

Plugin Manager

☐ Use browser for metadata download

Jenkins Global Security Section

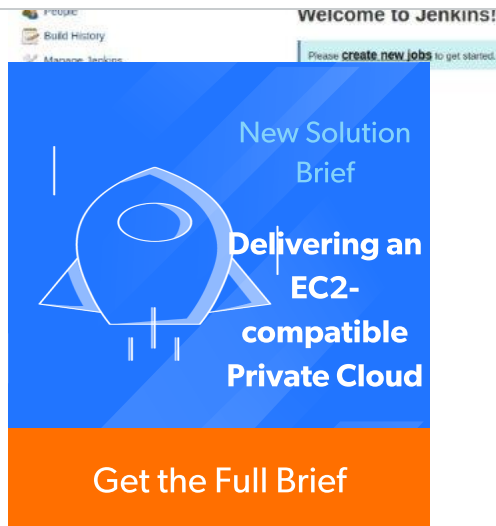
Now let's move to create the jobs that will prepare and and deploy your web application. Creating jobs is possible right after installing Jenkins on your server, but Jenkins support distributes builds in master-slave manner; more information is available via this [link](#).

While your integration and deployment processes should be divided into small jobs, since this example is relatively a small project we will not show this step. For this use case, just create the job for pulling the latest code from the repository, start the tests and then deploy and start it on your test server. Also this use-case has a simple architecture with Jenkins and the application located on same server. With complex architectures we might use the scripts or plugins to move the builds between servers.

Build

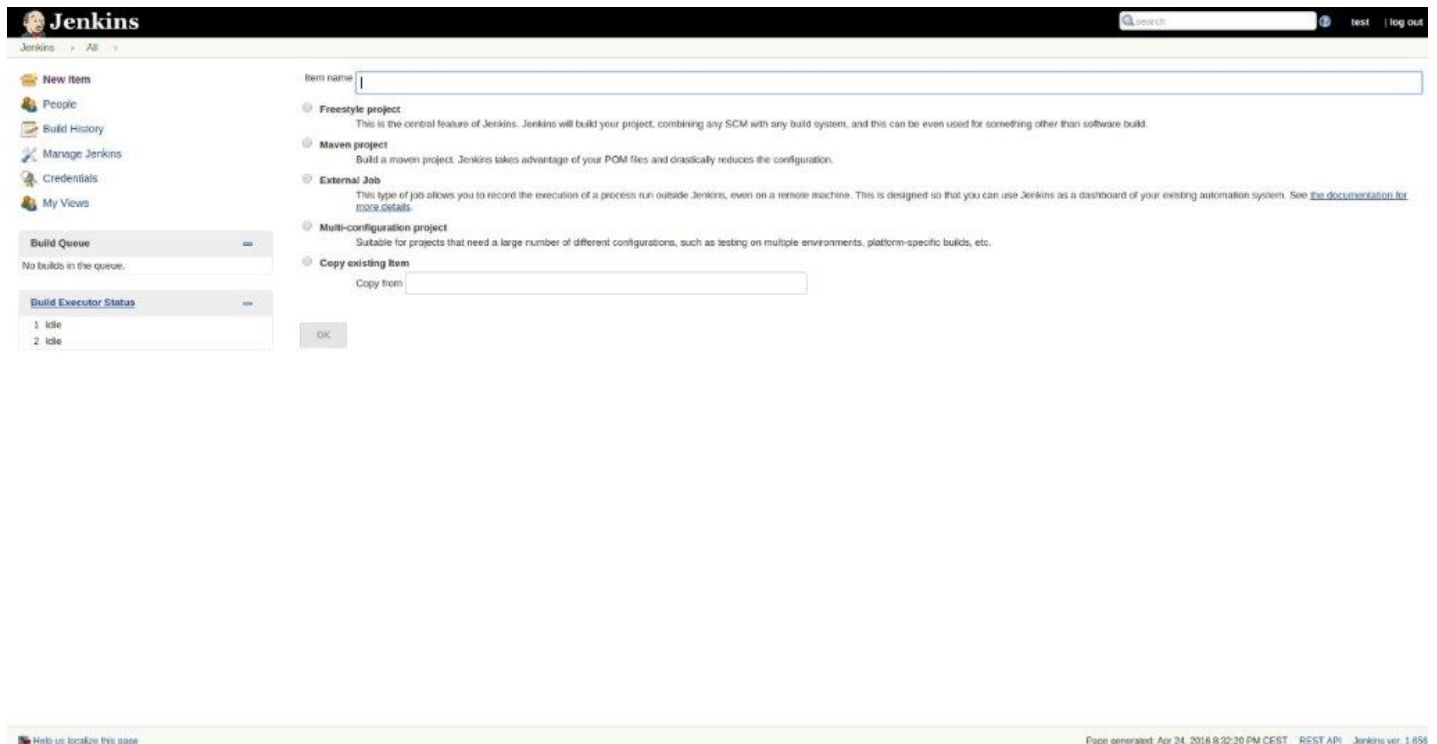
The first job which we created is the job which will execute shell commands, to create a [virtual Python environment](#) for your web application.

Enter the Jenkins dashboard and click New Item on the left menu as shown below:


[Help us localize this page](#)
Page generated: Apr 24, 2016 9:52:19 PM CEST [REST API](#) [Jenkins ver. 1.654](#)

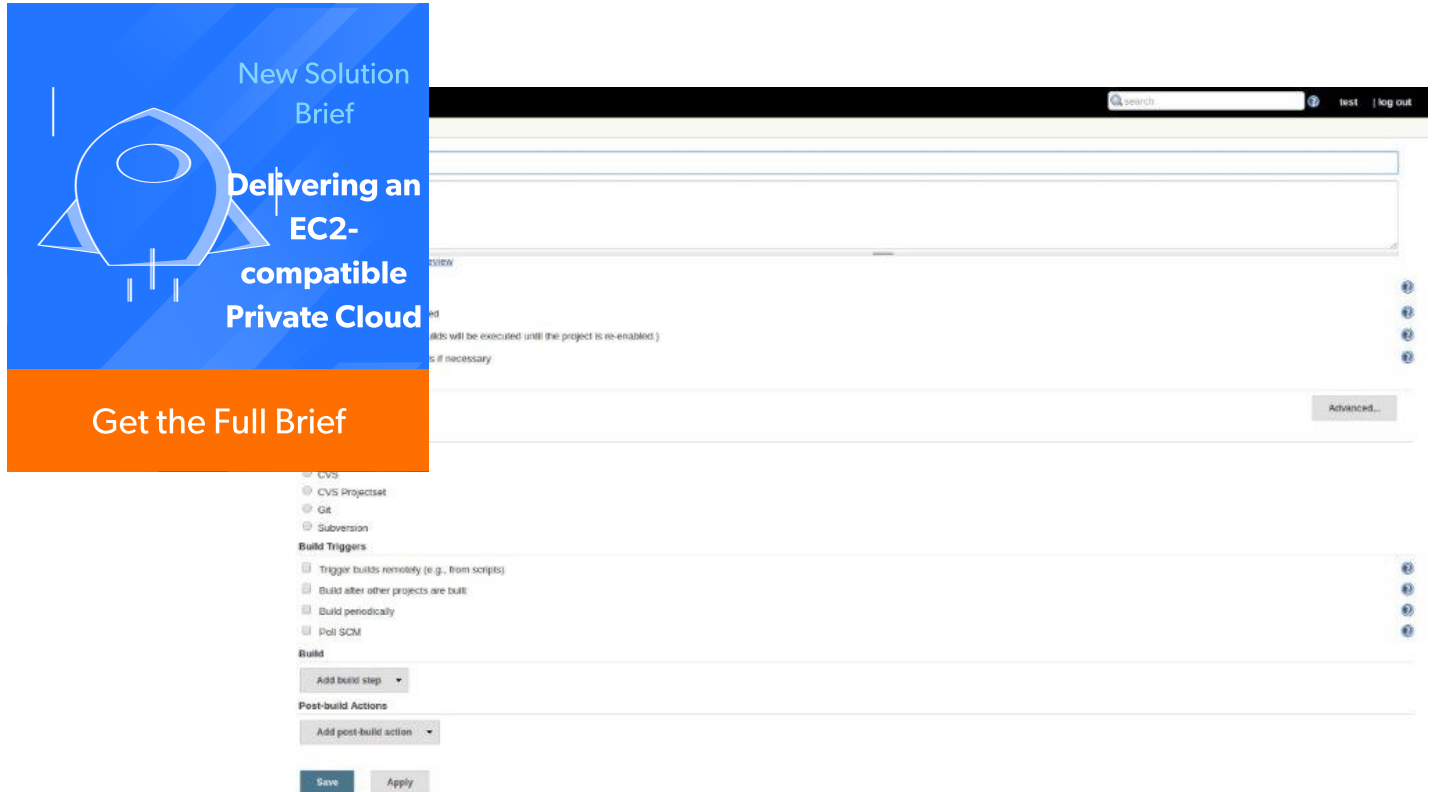
Jenkins Dashboard Screen

The form below displays; in this form, enter the item name and select the type of the project. For our purposes, the freestyle project is one which we selected and has the item name we used for PythonCreateEnv. The freestyle project provides enough options and features to build the complex jobs that you will need in your projects.



Jenkins form screen

project. This file will be used as source list, and notes which pip to use to install the required libraries for



The image shows a Jenkins job configuration page. On the left, there is a blue sidebar with a rocket icon and the text "New Solution Brief", "Delivering an EC2-compatible Private Cloud", and a button "Get the Full Brief". The main content area is titled "Job Configuration" and includes sections for "Source Code Management" (with radio buttons for None, CVS, CVS Projectset, and Git), "Build Triggers" (with checkboxes for Trigger builds remotely, Build after other projects are built, Build periodically, and Poll SCM), "Build" (with an "Add build step" button), and "Post-build Actions" (with an "Add post-build action" button). At the bottom, there are "Save" and "Apply" buttons.

Job config page

Before configuring the shell script, we will enter the code Repository URL under the Source Code Management –


Source Code Management


- ☐ None
☐ CVS
☐ CVS Projectset
☒ Git
- Repositories

Repository URL

source code management section

The next and final step under the Build section is to enter the shell script (below) which will create or recreate (this depends on the previous state of your virtual environment), the virtual environment for your web application and install the required libraries. This job also can be avoided in the Jenkins job and executed in your terminal, but also we want to show that jobs like this can be automated with Jenkins.


Execute shell



```

WORKSPACE/.sample1/
--no-site-packages $PYENV_HOME
bin/activate
--with-xunit tests

environment variables

Copy/paste)

PYENV_HOME=$WORKSPACE/.sample1/
if [ -d $PYENV_HOME ]; then
  rm -rf $PYENV_HOME
fi

virtualenv --no-site-packages $PYENV_HOME
. $PYENV_HOME/bin/activate
pip install -r envs/req.txt

```

After saving this item, the job is ready to create the virtual environment and will appear on the Jenkins dashboard. Our next move is to create a job to start the tests and interpret the results.

Test

Jenkins supports interpreting and presenting different kinds of unit tests for various programming languages. In our project we used the [nose](#) tool for Python testing. If we pass `--with-xunit` flag, nose will create the XML file as a report of the test cases which were covered. Jenkins knows how to read this file and it will create nice charts for test results, so let's create the job.

After clicking 'New item' from the menu and choosing the name and the type of the project, there are two additional steps to flow:

1. As in the previous job, you will have to enter the URL of the repository under the Source Code Management.
1. The next step is to create the shell script which Jenkins will execute. This shell script will start a new virtual environment, and then execute the nose unit test.



New Solution Brief

Delivering an EC2-compatible Private Cloud

Get the Full Brief

```
ME=$WORKSPACE/.sample1/
virtualenv --no-site-packages $PYENV_HOME
. $PYENV_HOME/bin/activate
nosetests --with-xunit tests
```

available environment variables

The shell script for starting the tests

```
PYENV_HOME=$WORKSPACE/.sample1/
virtualenv --no-site-packages $PYENV_HOME
. $PYENV_HOME/bin/activate
nosetests --with-xunit tests
```

For more information about how to use virtualenv, see: <https://www.google.com/url?q=https://virtualenv.pypa.io/en/stable/>

The last step is adding the post build actions.

Under the section Post-build Action, click the Add post-build action button.

Select the *Publish JUnit test result report*, then two new input fields will appear. In first one which is labeled *Test report XMLs*, enter the filename of generated report. The second field is optional – *health report amplification factor*, which allows you to specify a percentage of failed tests. Using this field you can measure the quality of the release.

Post-build Actions

Publish JUnit test result report

Test report XMLs

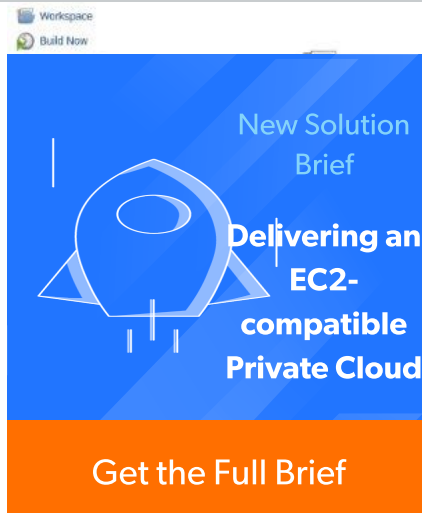
Fileset 'includes' setting that specifies the generated raw XML report files, such as 'myproject/target/test-reports/*.xml'. Basedir of the fileset is [the workspace root](#).

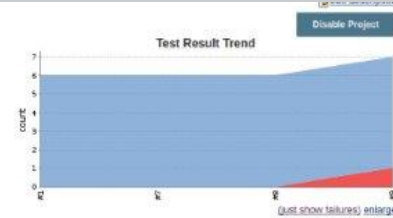
☐ Retain long standard output/error

The view for the post build actions of test job

Now we're done with configuring this job; after saving, it will appear on the Jenkins dashboard. The outcome results charts are generated after the job is done and will appear in the detailed view of the job, after clicking on the job name in the Jenkins jobs table on the dashboard.



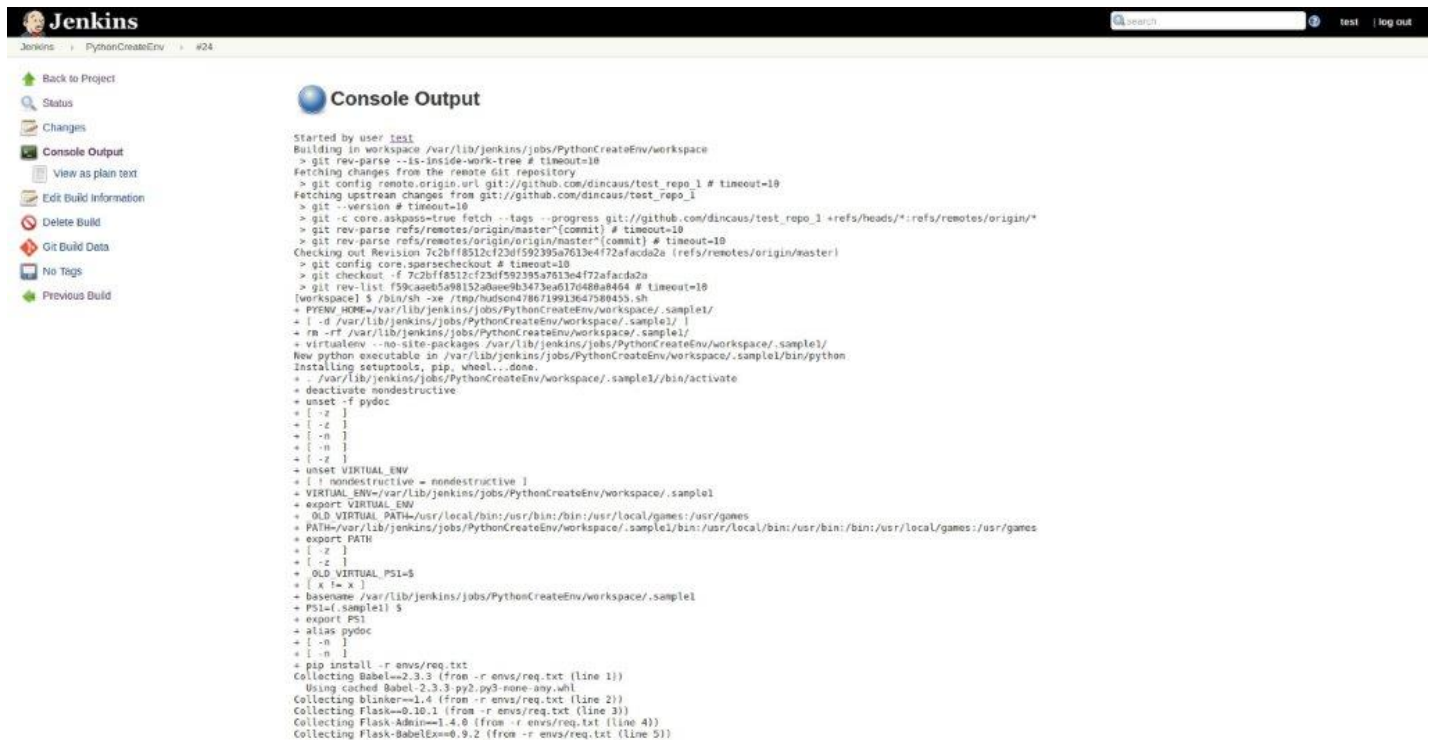




the chart on the right is the output test results

the icon on the far right, which is located in same row as the project on allows you to monitor the console output for each job once a job is done.

you can get the console output by clicking the job which appears in the side, or by clicking the number which appeared in the *Last Success* or *Last Failure* column in the Jenkins jobs table.



```

Jenkins
PythonCreateEnv #24

Back to Project
Status
Changes
Console Output
View as plain text
Edit Build Information
Delete Build
Git Build Data
No Tags
Previous Build

Started by user test1
Building in workspace /var/lib/jenkins/jobs/PythonCreateEnv/workspace
> git rev-parse --is-inside-work-tree # timeout=10
# fetching changes from the remote git repository
> git config remote.origin.url git://github.com/dincaus/test_repo_1 # timeout=10
Fetching upstream changes from git://github.com/dincaus/test_repo_1
> git --version # timeout=10
> git -c core.askpass=true fetch --tags --progress git://github.com/dincaus/test_repo_1 +refs/heads/*:refs/remotes/origin/*
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 7c2bf8512cf23df592395a7613e4f72afacda2a (refs/remotes/origin/master)
> git config core.sshCommand # timeout=10
> git checkout -f 7c2bf8512cf23df592395a7613e4f72afacda2a
> git rev-list f59caset65a98152a8aee9b3473ea6170488a8464 # timeout=10
[workspace] $ /bin/sh -xe /tmp/hudson18612991364788455.sh
+ PFW_HOME=/var/lib/jenkins/jobs/PythonCreateEnv/workspace/.sample1/
+ cd /var/lib/jenkins/jobs/PythonCreateEnv/workspace/.sample1/
+ rm -rf /var/lib/jenkins/jobs/PythonCreateEnv/workspace/.sample1/
+ virtualenv --no-site-packages /var/lib/jenkins/jobs/PythonCreateEnv/workspace/.sample1/
New python executable in /var/lib/jenkins/jobs/PythonCreateEnv/workspace/.sample1/bin/python
Installing setuptools, pip, wheel... done.
+ /var/lib/jenkins/jobs/PythonCreateEnv/workspace/.sample1/bin/activate
+ deactivate nondestructive
+ unset -f pydoc
+ [ -z ]
+ [ -z ]
+ [ -n ]
+ [ -n ]
+ [ -z ]
+ unset VIRTUAL_ENV
+ [ ! nondestructive = nondestructive ]
+ VIRTUAL_ENV=/var/lib/jenkins/jobs/PythonCreateEnv/workspace/.sample1
+ export VIRTUAL_ENV
+ OLD_VIRTUAL_PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
+ PATH=/var/lib/jenkins/jobs/PythonCreateEnv/workspace/.sample1/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
+ export PATH
+ [ -z ]
+ [ -z ]
+ OLD_VIRTUAL_PS1=$
+ [ x != x ]
+ basename /var/lib/jenkins/jobs/PythonCreateEnv/workspace/.sample1
+ PS1=(.sample1) $
+ export PS1
+ alias pydoc
+ [ -n ]
+ [ -n ]
+ pip install -r envs/req.txt
Collecting Babel==2.3.3 (from -r envs/req.txt (line 1))
Using cached Babel-2.3.3-py2.py3-none-any.whl
Collecting blinker==1.4 (from -r envs/req.txt (line 2))
Collecting Flask==0.10.1 (from -r envs/req.txt (line 3))
Collecting Flask-Admin==1.4.0 (from -r envs/req.txt (line 4))
Collecting Flask-BabelEx==0.9.2 (from -r envs/req.txt (line 5))

```

The console output of the virtual environment job

Questions about DevTest in the cloud? Click [here](#) to learn about how they go together.

The last job to close this use-case is the job which will start the web application. Starting a web application written in Python is as simple as creating the shell script which will start the HTTP server.

via the command line is accomplished with the command `python run.py &`.



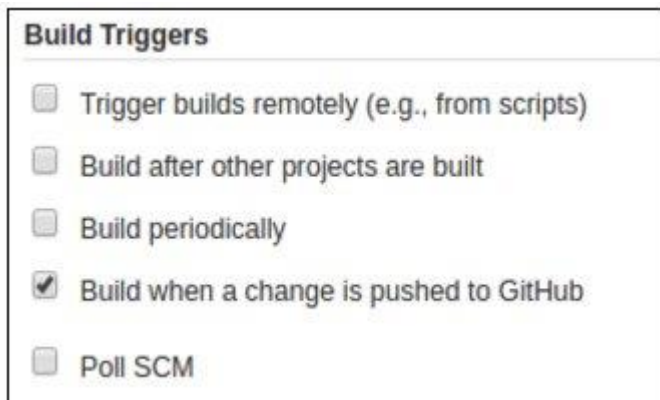
The recommended way of deploying web applications for a production environment is using [WSGI Servers](#) is used for this purpose.

You can schedule jobs periodically as an automation process so that you can automate your deployment. In Jenkins, the build can be triggered in several ways like using the cron syntax. The configuration settings for build triggers is located under the Build Triggers section of the job configuration page.



The build trigger section of job configuration page

One of the popular ways to trigger a job is after each commit to the repository. One of the easiest ways to do this is via the [GitHub Jenkins plugin](#). Once you have this plugin installed, a new option Build when a change is pushed to GitHub will appear under the *Build Triggers* section which we discussed above.




The build trigger section with option for triggering the build after the GitHub repository changed

In order to complete this way of triggering, a [webhook](#) needs to be added, so go to your GitHub repository then click *Settings* (see the image below).

Then click *Add Service* and find the *Jenkins (GitHub plugin)*. Now in the *Jenkins hook url* field enter the your Jenkins URL (see the image below).

Options

Services / Add Jenkins (GitHub plugin)



Jenkins is a popular continuous integration server.

By installing the Jenkins GitHub Plugin you can automatically trigger build jobs when pushes are made to GitHub.

Install Notes

"Jenkins Hook Url" is the URL of your Jenkins server's webhook endpoint. For example: `http://ci.jenkins-ci.org/github-webhook/`.

For more information see <https://wiki.jenkins-ci.org/display/JENKINS/GitHub+plugin>.

Jenkins hook url

☒ **Active**
We will run this service when an event is triggered.

Add service

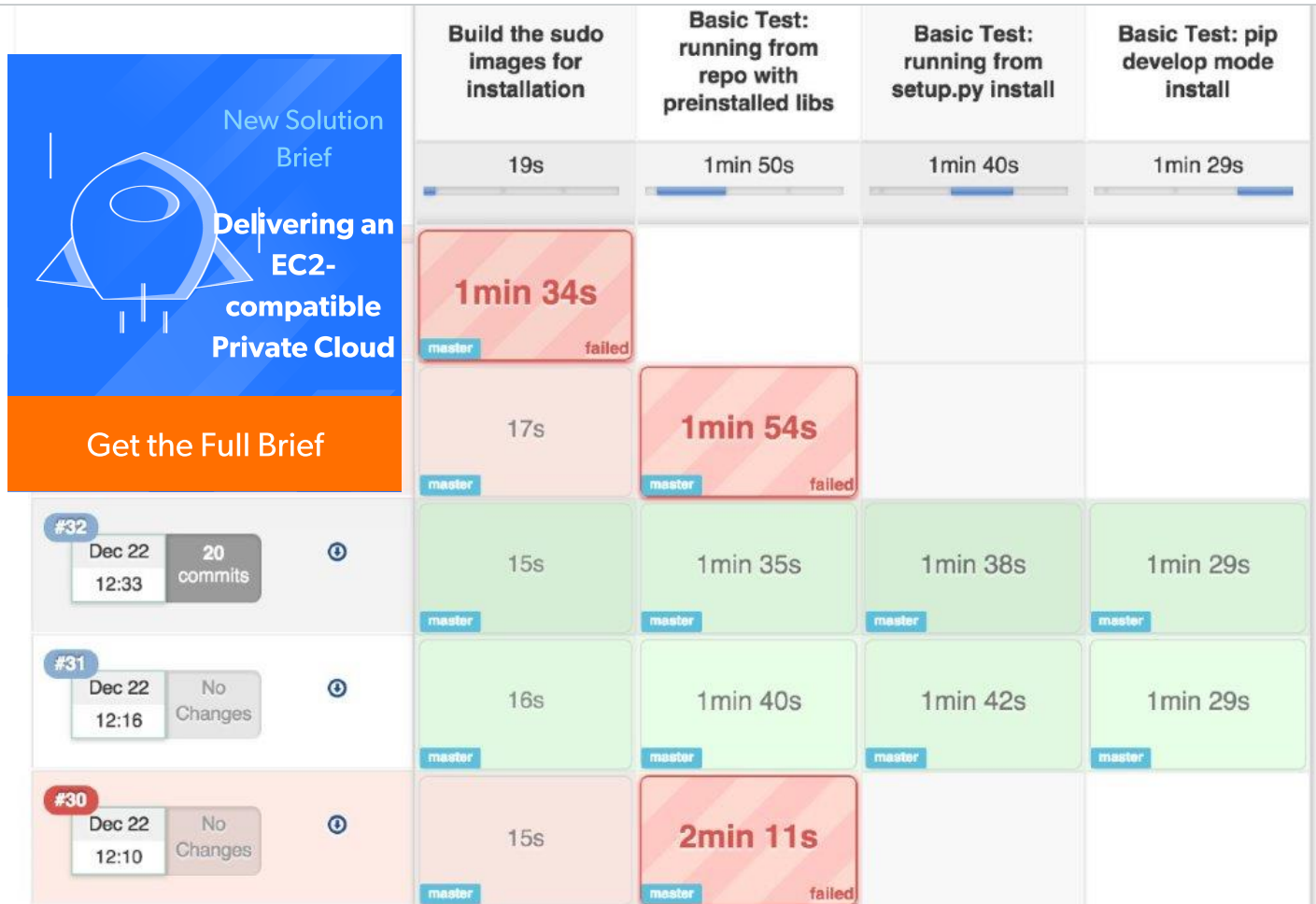
Webhooks settings on GitHub

Final Notes: Jenkins 2.0

Using Jenkins is the simplest and most common way to implement CI in today's R&D environments. In both parts of this article (part 1 can be found [here](#)) we discussed the theoretical side and best practices, as well as give you a hands-on example. However, there are a number of suitable alternatives to Jenkins, such as [TravisCI](#), [Go](#) and [TeamCity](#).

In addition, [Jenkins 2.0](#) was just released, presenting an important new capability – the Pipeline. The new version takes Jenkins beyond CI and CD, and includes new capabilities which enable users to plan, orchestrate and visualize their entire delivery pipeline.

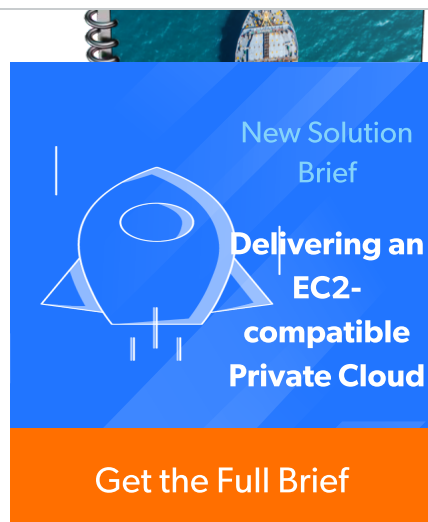
"For me as a practitioner and Jenkins administrator, the out-of-the-box defaults we've changed in this release are also important, especially for new users. New users of Jenkins are going to find it much easier to get set up with the tools they need, and Jenkins will be more secure out of the box." [Said R.Tyler Croy](#), [Jenkins Community Leader](#)



We also encourage you to get involved and join the active Jenkins community. Contribute by creating your own plugins, or at least extend the ones you are using. Fix bugs and share the fixes with the community; stay tuned to the Jenkins [mailing lists](#) and [IRC](#). After all, Jenkins is open source.

[Back to Blog >](#)

Download the [Ultimate Guide to deploying, managing and scaling Kubernetes](#)



EXPLORE

[Request Demo](#)[Solutions](#)

COMPANY

[About Us](#)[Press](#)



Resources

[Blog](#)[New Solution Brief](#)[SUPPORT](#)[Support Portal Login](#)
**Delivering an
EC2-
compatible
Private Cloud**[Get the Full Brief](#)

CONNECT

[Contact Us](#)[Careers](#)

© Copyright 2019 Stratoscale "AWS (Amazon Web Services)" is a trademark of Amazon.com, Inc