Installing Jenkins

Raúl Estrada

Octubre 2020

Jenkins is also modular, which enables developers to write plugins to extend functionalities, for example, sending messages to a Slack channel if the build fails or running Node.js scripts as a part of a job.

On the scalability side, Jenkins, like Bamboo, can be scaled to hundreds of nodes through a master/slave configuration so that we can add more power to our CI server in order to execute some tasks in parallel.

On its own, Jenkins will be enough to provide contents for a couple of books, but we are going to visit what we need to set up automated jobs for testing our applications. It is also possible to write plugins for Jenkins, so virtually, there is no limit to what it can do.

Let's focus on the operational side of Jenkins for now. In order to run Jenkins, we have two options:

- Running it as a Docker container
- Installing it as a program in your CI server

For now, we are going to install Jenkins, using its Docker image as it is the simplest way of running it and it fits our purpose. Let's start. The first thing is running a simple instance of Jenkins from the command line:

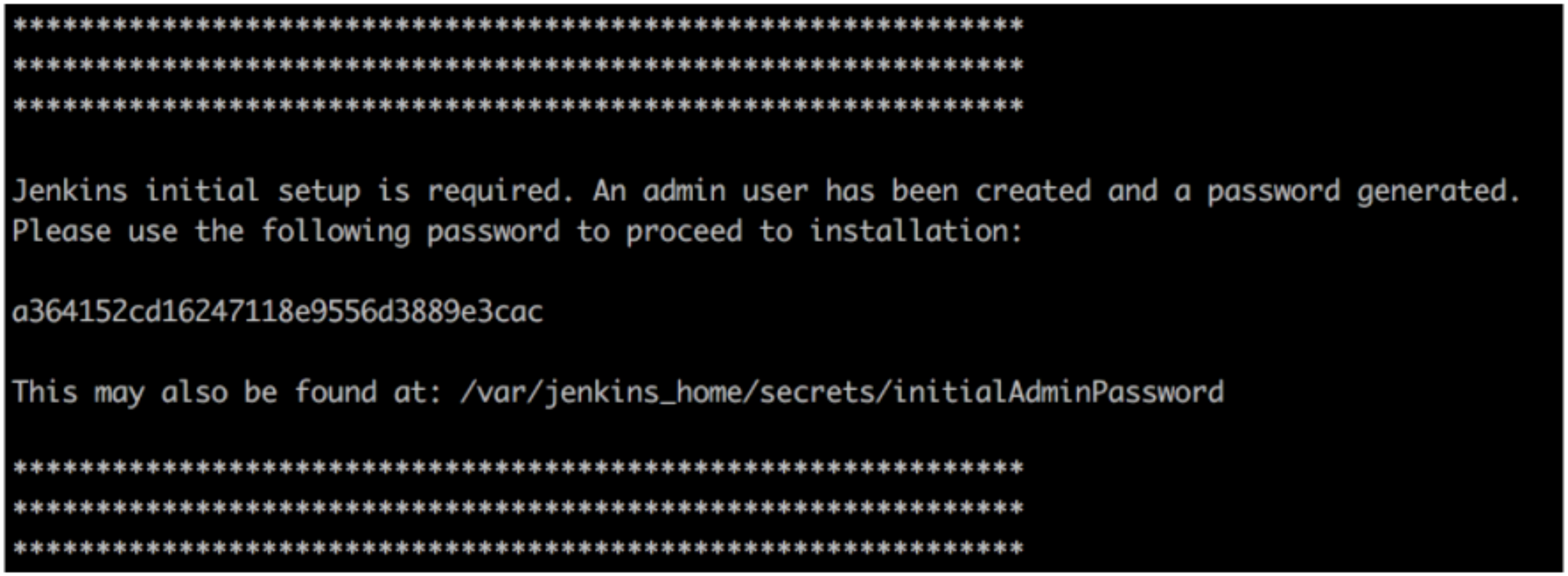
```
docker run -p 8080:8080 -p 50000:50000 jenkins
```

This will run Jenkins, but be aware that all the information about configuration and builds executed will be stored within the container, so if you lose the container, all the data is lost as well. If you want to use a volume to store the data, the command that you need to execute is as follows:

Copy

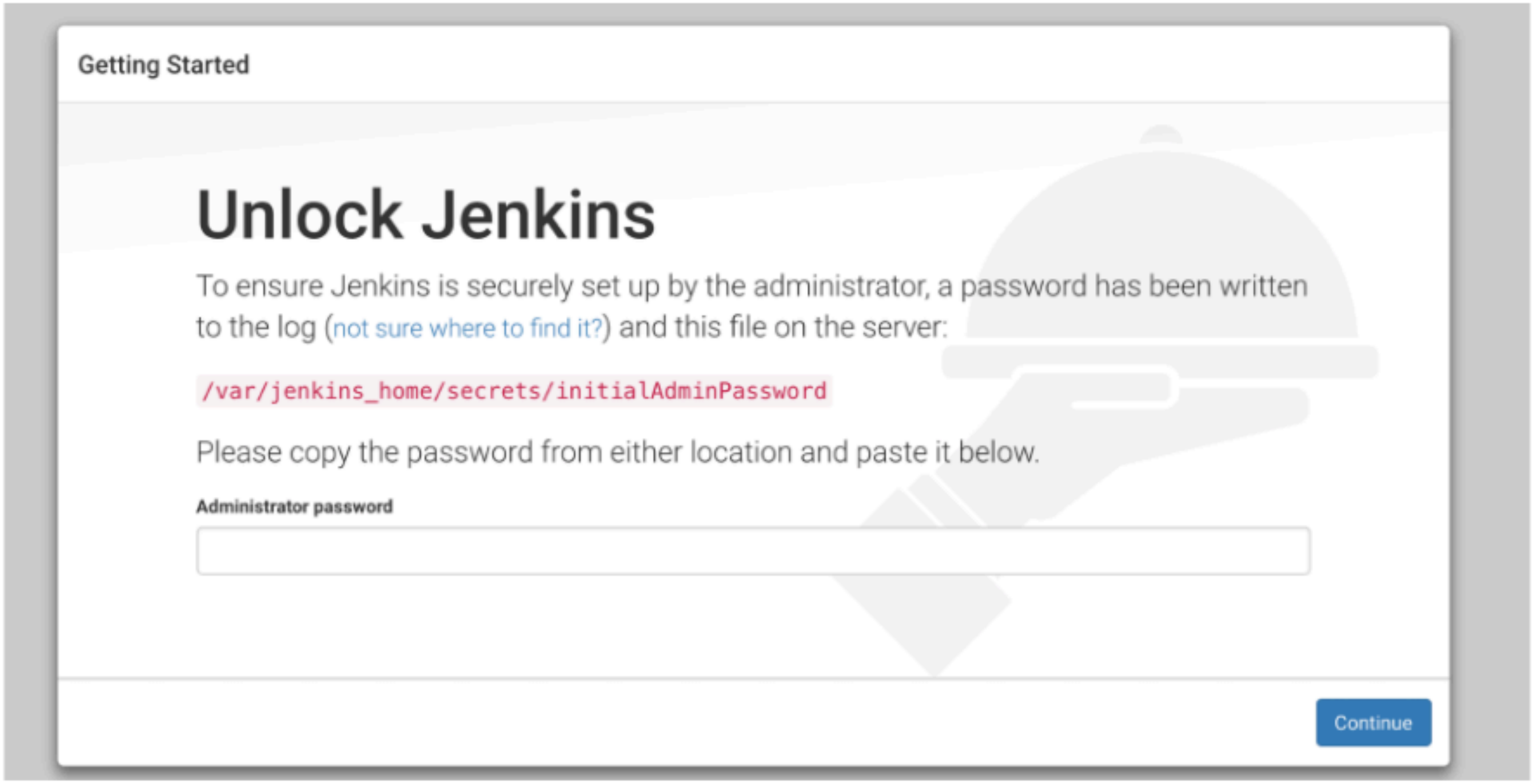
```
docker run --name myjenkins -p 8080:8080 -p 50000:50000 -v /var/jenkins_home jenkins
```

This will create a volume that you can reuse later on when upgrading to new versions of Jenkins or even restarting the same container. After running the command, the logs will show something similar to what is shown in the following figure:



```
*************************************************************
*************************************************************
*************************************************************

Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

a364152cd16247118e9556d3889e3cac

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

*************************************************************
*************************************************************
*************************************************************
```

This is the initial password for Jenkins and it is necessary in order to set up the instance. After a few seconds, the logs of the container will stop, which means your Jenkins server is ready to be used. Just open the browser and go to `http://localhost:8080/` , and you will see something similar to this:



Unlock Jenkins

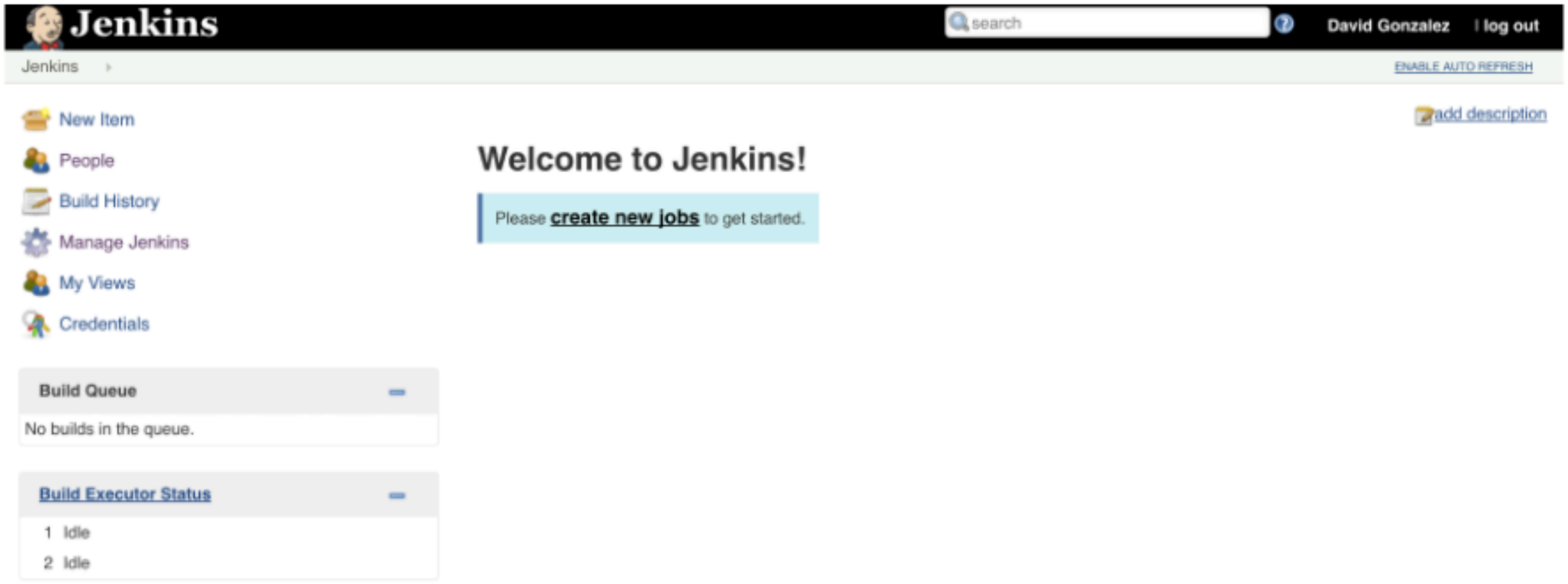
To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

`/var/jenkins_home/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

Administrator password

This is where you can enter `Administrator password`, which we saved earlier, and click on the C`ontinue button. The next screen will ask you whether it should install the suggested plugins or whether you want to select which plugins to install. Choose the suggested plugins. After a few minutes, it will let you create a user and that's it. Jenkins is up and running in a container:`

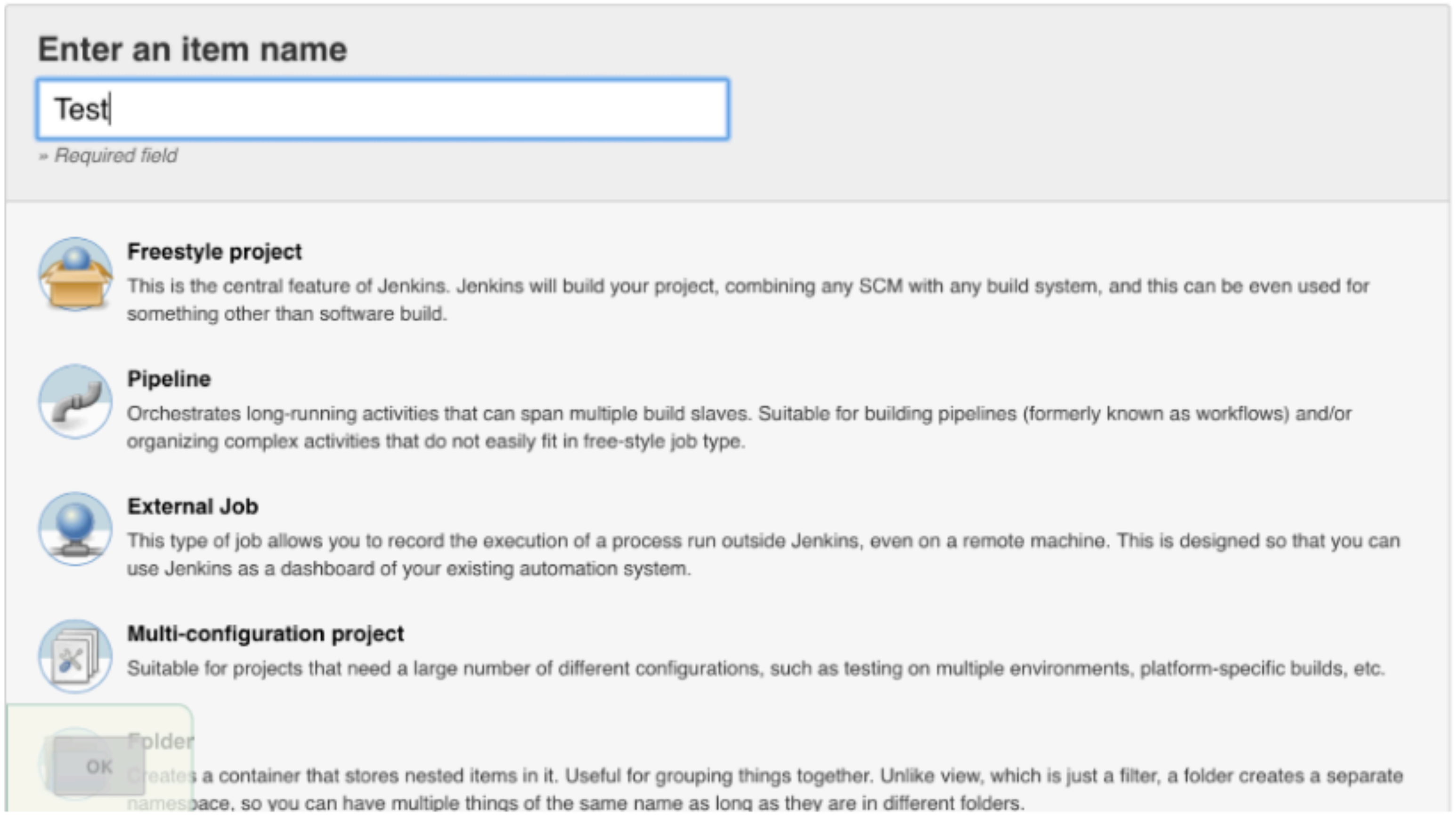


Welcome to Jenkins!

Please **create new jobs** to get started.

No builds in the queue.

Now we are going to create a new job. We are going to use the same repository as we used with Bamboo so we can compare the two integration servers. Let's click on `Create a new project`. You should be presented with the following form:


Enter an item name

Test

» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Pipeline
Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

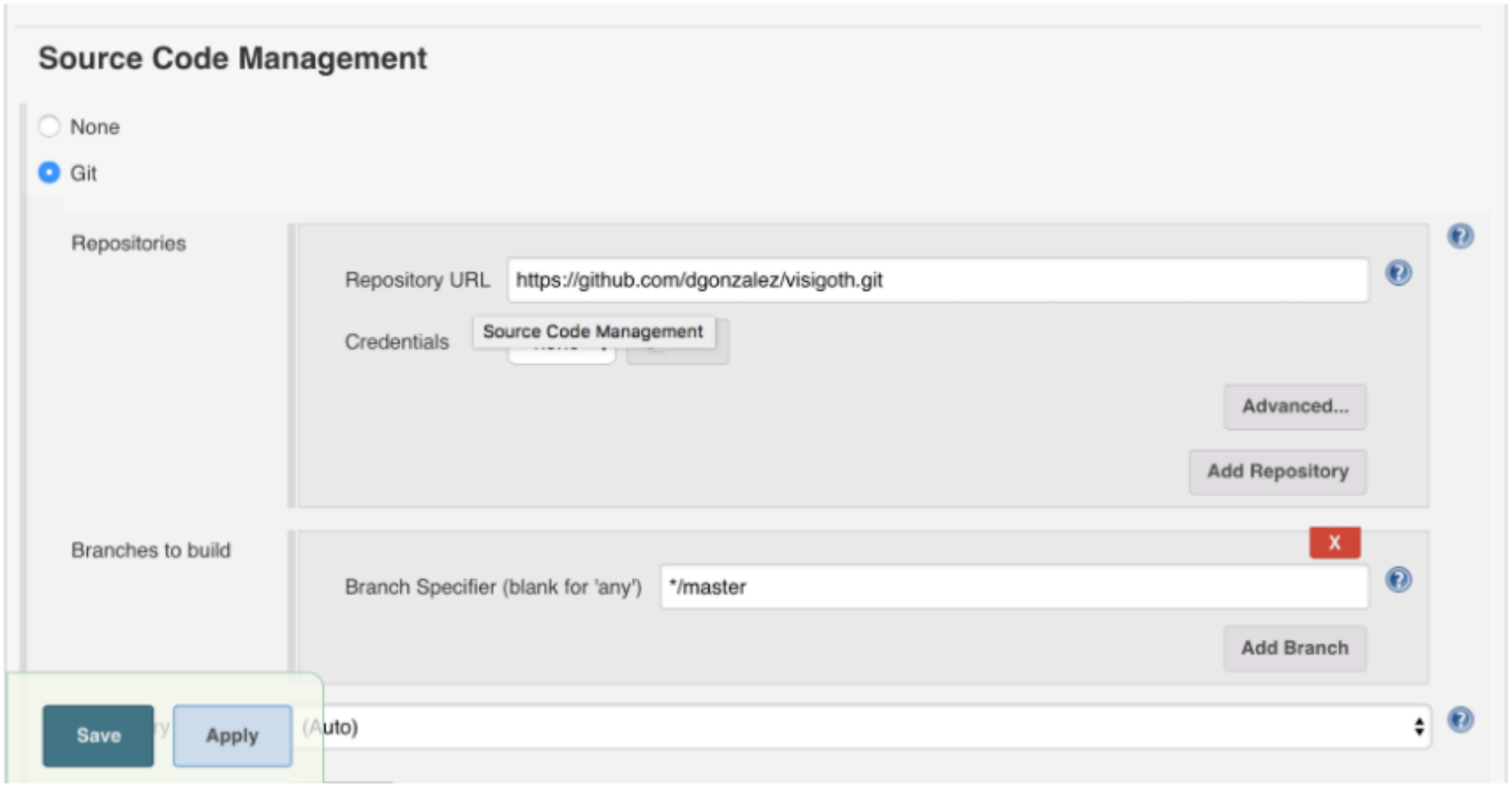
External Job
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

Just enter a name for the project and choose the first option: `Freestyle project`. Jenkins has different types of projects. Freestyle project is a type of project where we can define the steps, as we did in Bamboo. Another interesting option is the type `Pipeline` where we can, through a **DSL** (known as **Domain Specific Language**), define a set of steps and stages, creating a pipeline that can be saved as code.

The following screen is where we configure the project. We are going to use Git with the repository hosted at https://github.com/dgonzalez/visigoth.git. You can use your fork if you previously forked it while working with Bamboo. Your configuration should be similar to the what is shown in the following screenshot:



Source Code Management

None

[✓ Git]

Repositories

Repository URL: https://github.com/dgonzalez/visigoth.git

Credentials

Branches to build

Branch Specifier (blank for 'any'): */master

(Auto)

Now we need to install the dependencies of Visigoth with the `npm install --development` command and execute the tests with the `npm test` command, but we are running Jenkins from a container and this container does not have Node.js installed. We are going to use our Docker knowledge to install it. Inspecting the Dockerfile of the Jenkins image in the Docker Hub, we can verify that it is based on Debian Jessie (it is based on OpenJDK but that is based on Debian Jessie) so we can install the required software in it. The first thing that

Copy

```
docker exec -u 0 -it eaaef41f221b /bin/bash
```

This command executes `/bin/bash` in the container with the ID `eaaef41f221b` (it will change in your system as it is unique per container) but with the user that matches the user ID `0`, in this case, root. We need to do this because the Jenkins image defines and uses a new user called `jenkins` with a known UID and GID so if the `-u 0` flag is not passed, the `/bin/bash` command will be executed by the user `jenkins`.

Once we are root in the container, proceed to install Node.js:

Copy

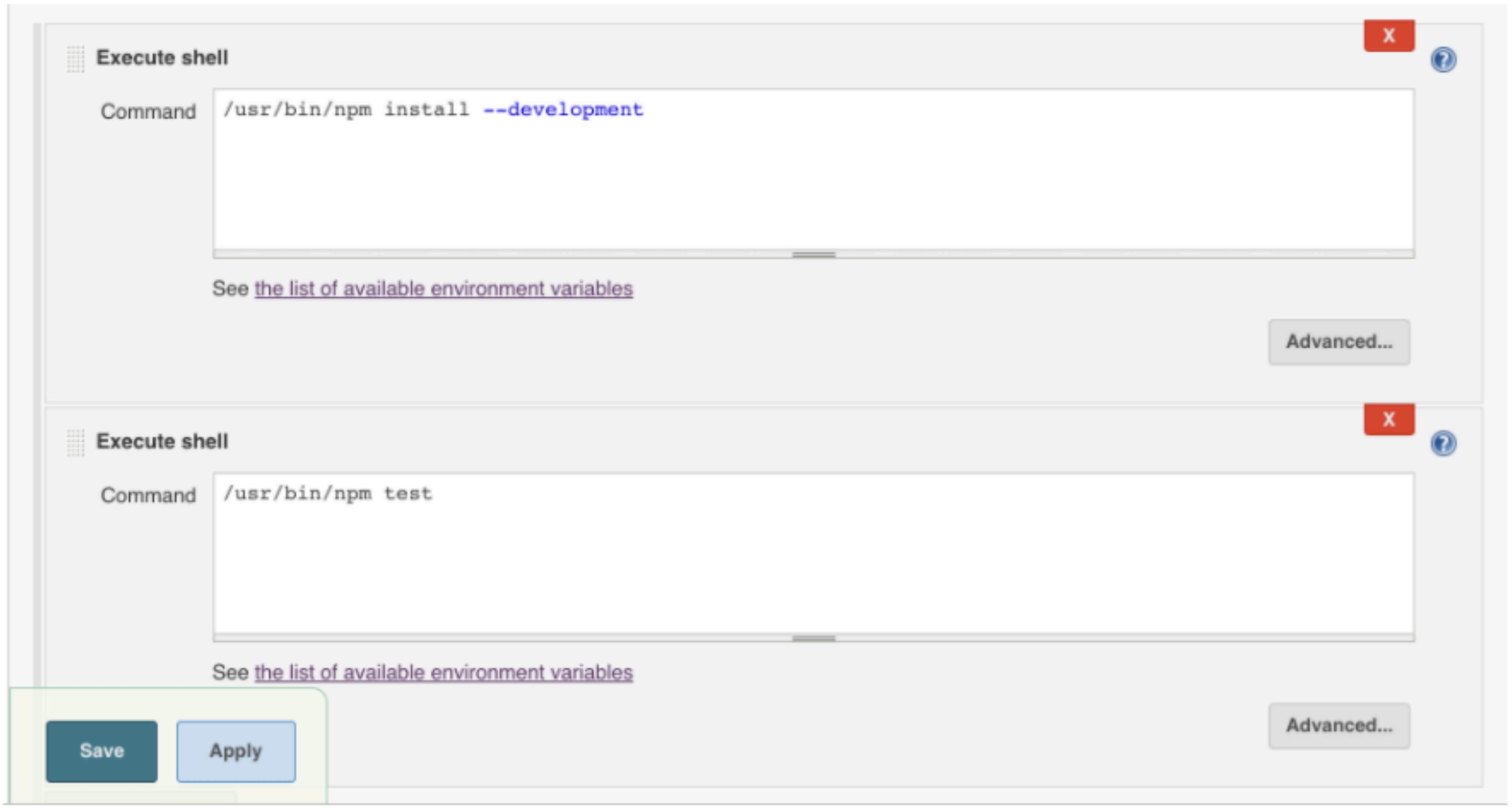
```
curl -sL https://deb.nodesource.com/setup_7.x | bash -
```

And once the execution of the previous command is finished, run the following one:

Copy

```
apt-getinstall -y nodejs build-essentials
```

Once Node.js is installed, we can just exit the root shell within the container and go back to Jenkins to complete our tasks. As we did with Bamboo, here are our tasks in order to run our tests:


Execute shell
Command
See the list of available environment variables
Advanced...
Execute shell
Command
See the list of available environment variables
Advanced...
Save
Apply

```
/usr/bin/npm install --development
```

```
/usr/bin/npm test
```

In the very bottom of the job configuration, there is a section called **post-build** actions. This section allows you to execute actions once the job is finished. These actions include sending e-mails, adding commit messages to the Git repository, among others. As we previously mentioned, Jenkins is extensible and new actions can be added by installing new plugins.

Once you have added these two steps to the build, just click on `Save` and we are all set: you now have a fully functional Jenkins job. If we run it, it should successfully run the tests on Visigoth.

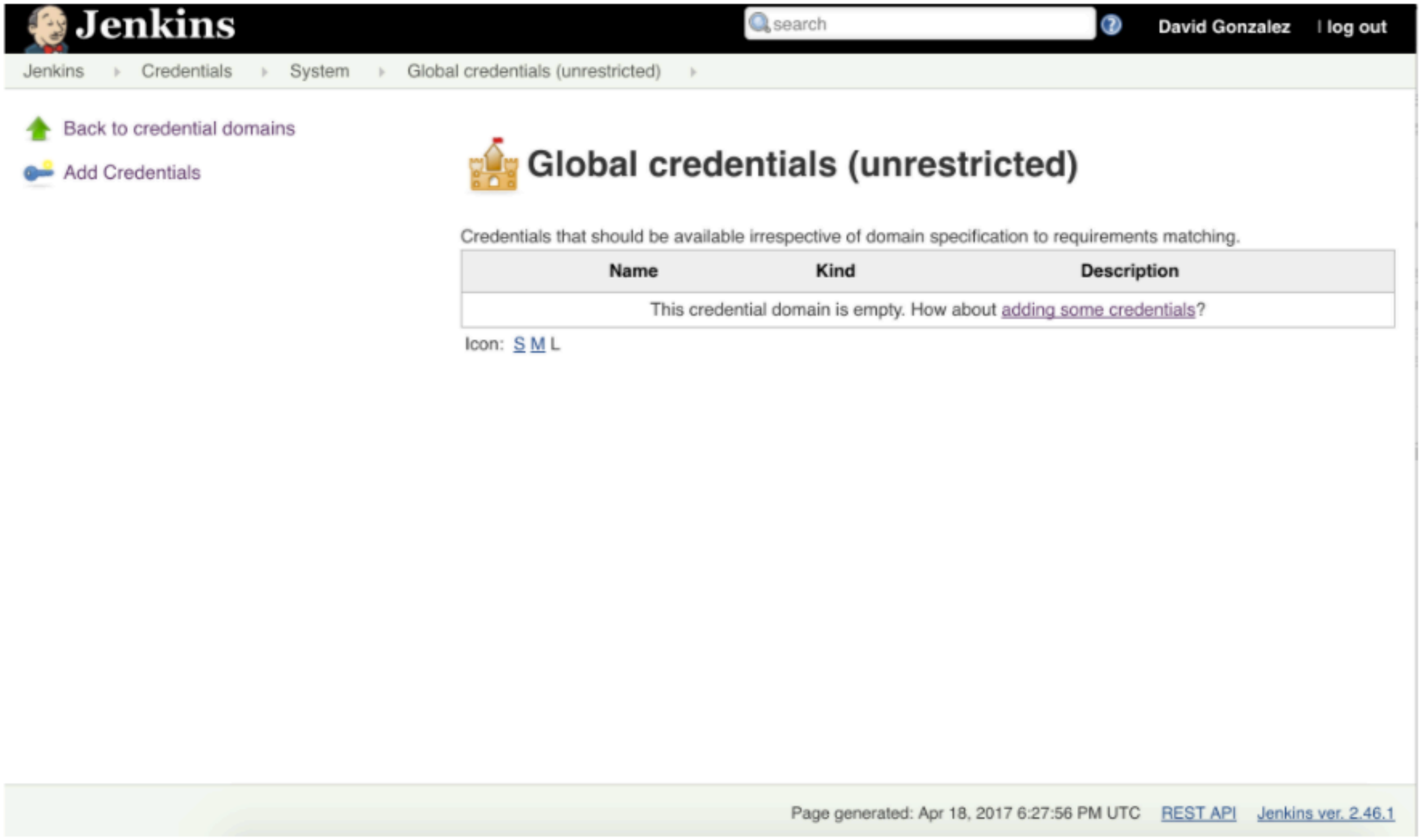
Secrets Management

One of the possibilities of the CI server is the ability to talk to third-party services that usually rely on some sort of credentials (such as access tokens or similar) to authenticate the user. Exposing these secrets would be discouraged as they could potentially cause major harm to our company.

Jenkins handles this in a very simple way: it provides a way to store credentials in a safe way that can be injected into the build as environment variables so that we can work with them.

Let's look at some examples. First, we need to create the secrets in Jenkins. In order to do that, we have to go to `Manage Jenkins` from the home page.

Once we are there, you should see a screen very similar to this one:


Jenkins ▸ Credentials ▸ System ▸ Global credentials (unrestricted) ▸

Back to credential domains
Add Credentials

Global credentials (unrestricted)

Credentials that should be available irrespective of domain specification to requirements matching.

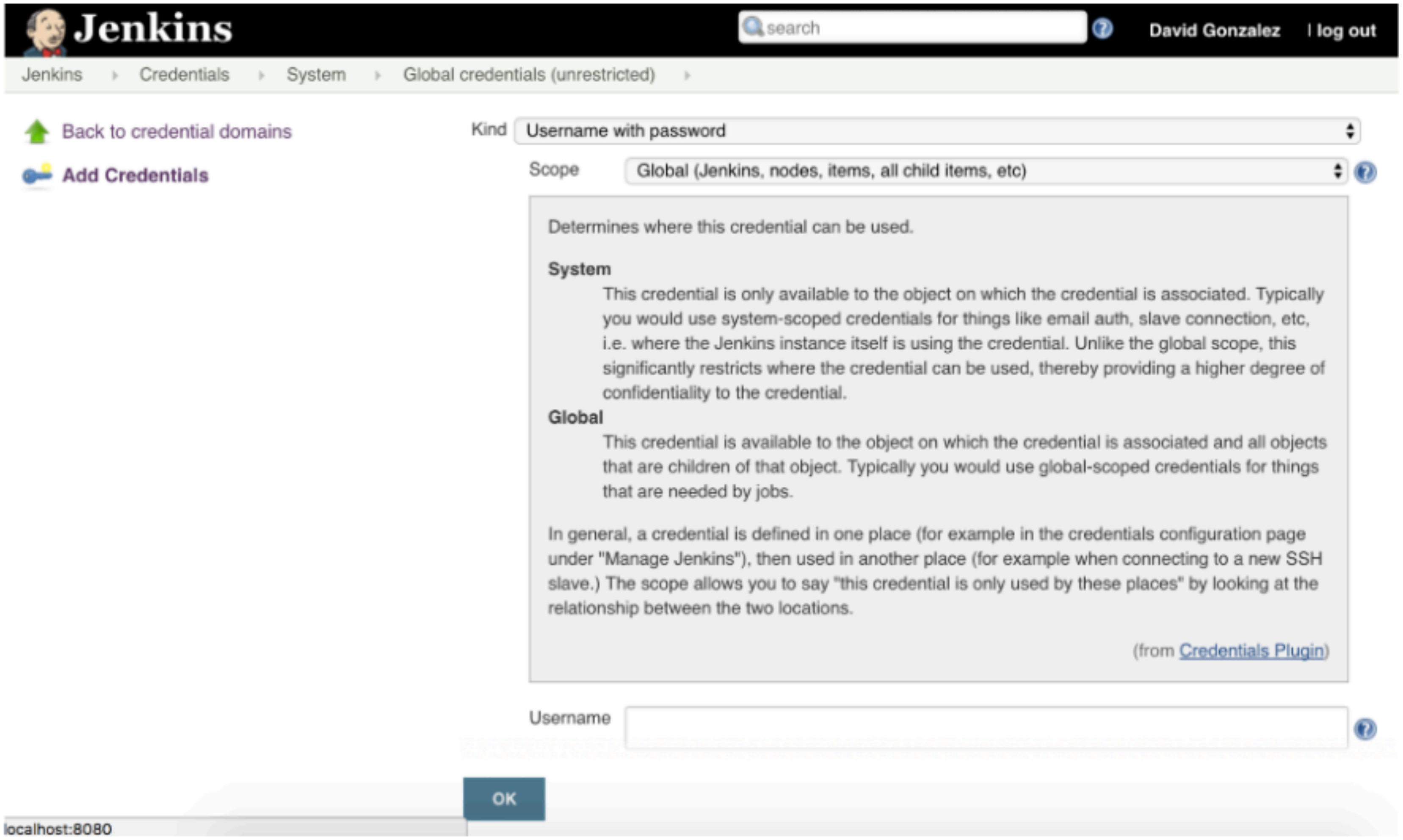
Name	Kind	Description
This credential domain is empty. How about adding some credentials?		

Icon: S M L

Page generated: Apr 18, 2017 6:27:56 PM UTC REST API Jenkins ver. 2.46.1

We are using the `Global credentials` store as we just want to showcase how it works, but Jenkins allows you to box credentials so you can restrict access across different usages. In Jenkins, credentials, aside from being injected into the build context, can be connected to plugins and extensions so that they can authenticate against third-party systems.

Now, we click on `Add Credentials` on the left-hand side:


Jenkins
search
David Gonzalez
| log out
Jenkins › Credentials › System › Global credentials (unrestricted) ›
Back to credential domains
Add Credentials

Kind: Username with password

Scope: Global (Jenkins, nodes, items, all child items, etc)

Determines where this credential can be used.

System
This credential is only available to the object on which the credential is associated. Typically you would use system-scoped credentials for things like email auth, slave connection, etc, i.e. where the Jenkins instance itself is using the credential. Unlike the global scope, this significantly restricts where the credential can be used, thereby providing a higher degree of confidentiality to the credential.

Global
This credential is available to the object on which the credential is associated and all objects that are children of that object. Typically you would use global-scoped credentials for things that are needed by jobs.

In general, a credential is defined in one place (for example in the credentials configuration page under "Manage Jenkins"), then used in another place (for example when connecting to a new SSH slave.) The scope allows you to say "this credential is only used by these places" by looking at the relationship between the two locations.

(from Credentials Plugin)

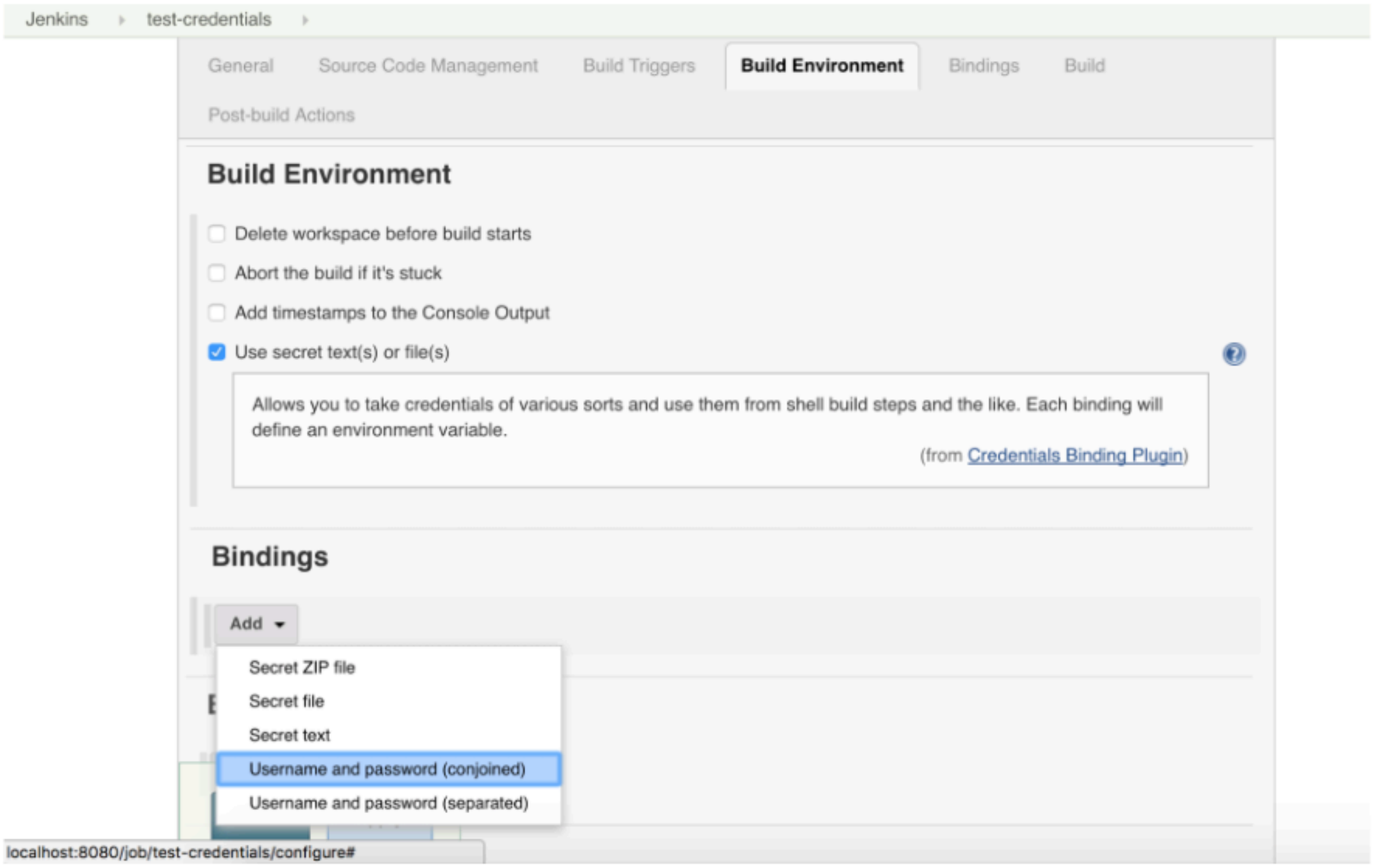
Username:

There are some fields that we need to fill before proceeding, but they are very basic:

- `Kind` : This is the type of secret that we want to create. If you open the drop-down, there are several types, from files to certificates, walking through usernames and passwords.
- `Scope` : This is the scope of our secret. The documentation is not 100% clear (at least not in the first read) but it allows us to hide the secret from certain stances. There are two options: `Global` and `System` . With `Global` , the credentials can be exposed to any object within Jenkins and its child, whereas with `System` , the credentials can be exposed only to Jenkins and its nodes.

The rest of the fields are dependant on the type of secret. For now on, we are going to create a `Username with password` secret. Just select it in the dropdown and fill in the rest of the details. Once it is created, it should show in the list of credentials.

The next step is to create a job that is bound to those credentials so we can use them. Just create a new freestyle project, as we saw in the beginning of this section, but we are going to stop on the screen where we can configure the job, precisely in the `Build Environment` section:



Build Environment

- [ ] Delete workspace before build starts
- [ ] Abort the build if it's stuck
- [ ] Add timestamps to the Console Output
- [✓] Use secret text(s) or file(s)

Allows you to take credentials of various sorts and use them from shell build steps and the like. Each binding will define an environment variable.

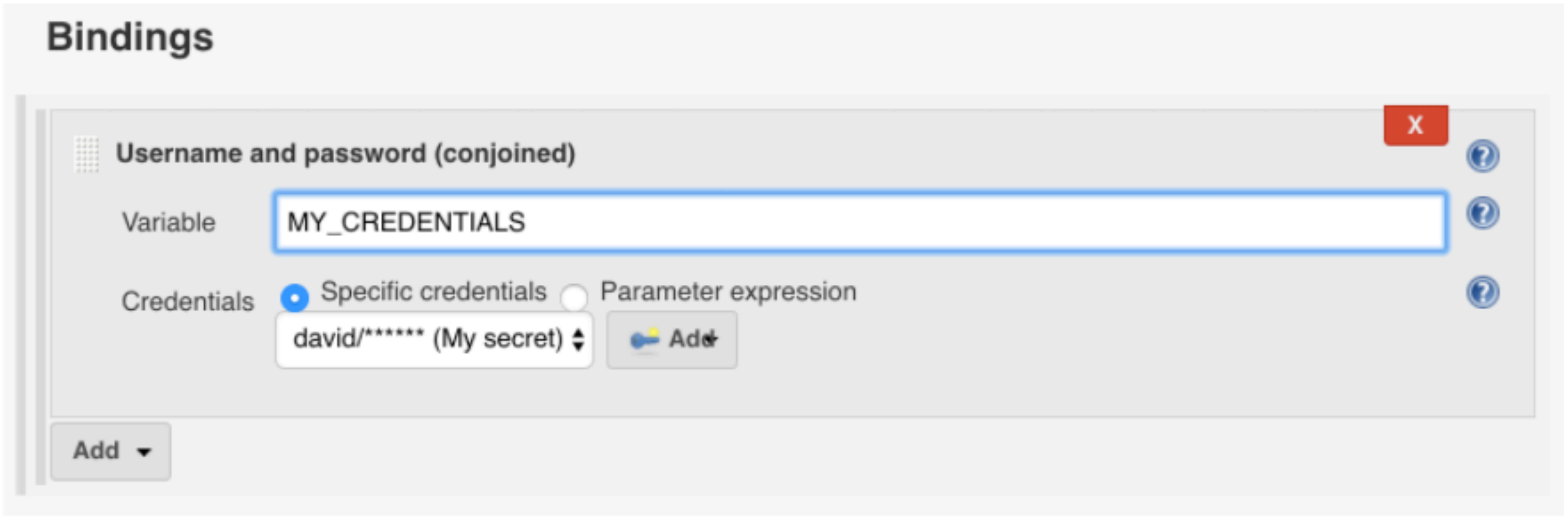
(from Credentials Binding Plugin)

Bindings

Add
Secret ZIP file
Secret file
Secret text
Username and password (conjoined)
Username and password (separated)
localhost:8080/job/test-credentials/configure#

Now select `Username and password (conjoined)`. Conjoined username and password means that we get the full secret (the username and the password) in a single variable, whereas with separated, we get the secret split in two variables: one for the username and another one for the password.

Once we select it, the form to create the binding is fairly simple:



Bindings

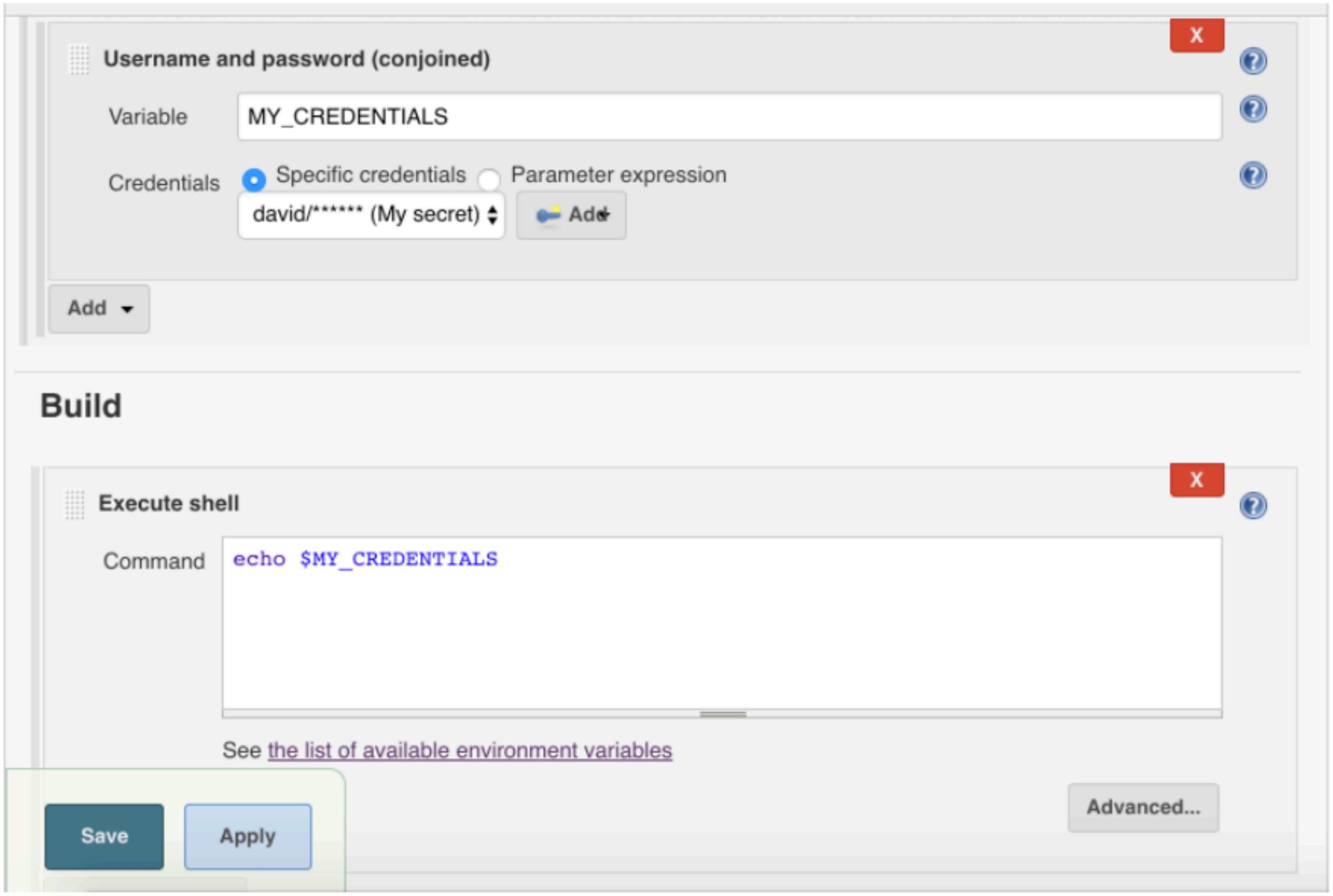
Username and password (conjoined)

Variable: MY_CREDENTIALS

Credentials: [✓ Specific credentials] Parameter expression

david/****** (My secret)

We get to choose the `Variable` where we want to store the secret and we also get to choose which secret. There is a radio button that lets you choose between `Parameter expression` or `Specific credentials` as we can parametrize the job to get input from the user on the triggering screen. In order to showcase how well thought Jenkins is, we are going to add a `Build` step that uses the secret by just echoing it into the logs:



Username and password (conjoined)

Variable: MY_CREDENTIALS

Credentials: [✓ Specific credentials] Parameter expression

david/****** (My secret)

Build

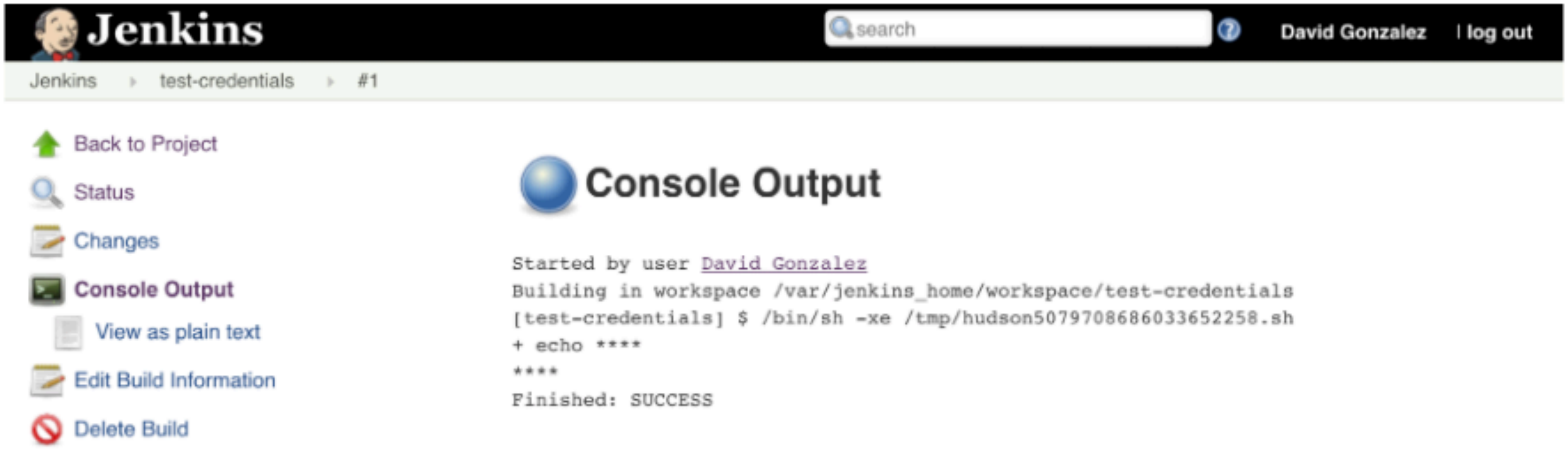
Execute shell

Command:

```
echo $MY_CREDENTIALS
```

See the list of available environment variables

Click on the `Save` button to save the job and run it. Once the job execution finishes, go to the result and click on `Console Output`. If you were expecting to see the secret in here, Jenkins has a surprise for you:



```
Started by user David Gonzalez
Building in workspace /var/jenkins_home/workspace/test-credentials
[test-credentials] $ /bin/sh -xe /tmp/hudson5079708686033652258.sh
+ echo ****
****
Finished: SUCCESS
```