**Introduction**

Jenkins is an open source automation server intended to automate repetitive technical tasks involved in the continuous integration and delivery of software. Jenkins is Java-based and can be installed from Ubuntu packages or by downloading and running its Web application ARchive (WAR) file — a collection of files that make up a complete web application which is intended to be run on a server.

In this tutorial we will install Jenkins by adding its Debian package repository, then using that repository to install the package using apt-get.

**Prerequisites**

To follow this tutorial, you will need:

**One Ubuntu 16.04 server** configured with a non-root sudo user and a firewall by following the [Ubuntu 16.04 initial server setup guide](https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-16-04). We recommend starting with at least 1 GB of RAM. See [Choosing the Right Hardware for Masters](https://jenkins.io/doc/book/hardware-recommendations/) for guidance in planning the capacity of a production Jenkins installation.

When the server is set up, you're ready to follow along.

**Step 1 — Installing Jenkins**

The version of Jenkins included with the default Ubuntu packages is often behind the latest available version from the project itself. In order to take advantage of the latest fixes and features, we'll use the project-maintained packages to install Jenkins.

First, we'll add the repository key to the system.

* wget -q -O - https://pkg.jenkins.io/debian/jenkins-ci.org.key | sudo apt-key add -

When the key is added, the system will return OK. Next, we'll append the Debian package repository address to the server's sources.list:

* echo deb https://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list

When both of these are in place, we'll run update so that apt-get will use the new repository:

* sudo apt-get update

Finally, we'll install Jenkins and its dependencies, including Java:

* sudo apt-get install jenkins

Now that Jenkins and its dependencies are in place, we'll start the Jenkins server.

**Step 2 — Starting Jenkins**

Using systemctl we'll start Jenkins:

sudo systemctl start jenkins

Since systemctl doesn't display output, we'll use its status command to verify that it started successfully:

* sudo systemctl status jenkins

If everything went well, the beginning of the output should show that the service is active and configured to start at boot:

Output

● jenkins.service - LSB: Start Jenkins at boot time

Loaded: loaded (/etc/init.d/jenkins; bad; vendor preset: enabled)

Active:active (exited) since Thu 2017-04-20 16:51:13 UTC; 2min 7s ago

Docs: man:systemd-sysv-generator(8)

Now that Jenkins is running, we'll adjust our firewall rules so that we can reach Jenkins from a web browser to complete the initial set up.

**Step 3 — Opening the Firewall**

By default, Jenkins runs on port 8080, so we'll open that port using ufw:

* sudo ufw allow 8080

We can see the new rules by checking UFW's status.

* sudo ufw status

We should see that traffic is allowed to port 8080 from anywhere:

Output

Status: active

To Action From

-- ------ ----

OpenSSH ALLOW Anywhere

8080 ALLOW Anywhere

OpenSSH (v6) ALLOW Anywhere (v6)

8080 (v6) ALLOW Anywhere (v6)

**Note:** If the firewall is inactive, the following commands will make sure that OpenSSH is allowed and then enable it.

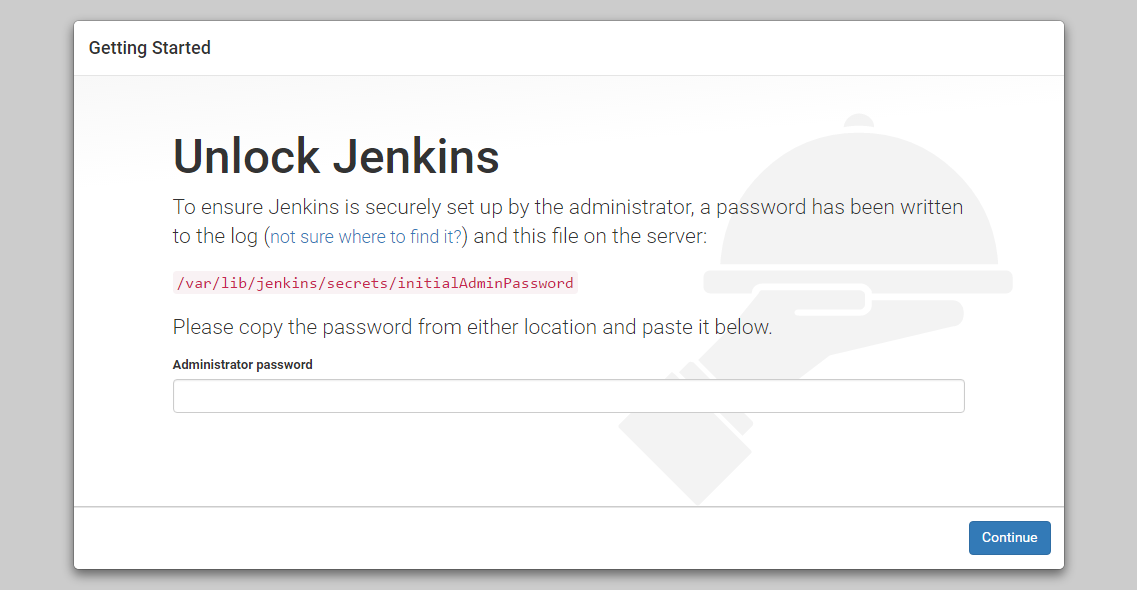
* sudo ufw allow OpenSSH
* sudo ufw enable

Now that Jenkins is installed and the firewall allows us to access it, we can complete the initial setup.

**Step 4 — Setting up Jenkins**

To set up our installation, we'll visit Jenkins on its default port, 8080, using the server domain name or IP address: http://ip\_address\_or\_domain\_name:8080

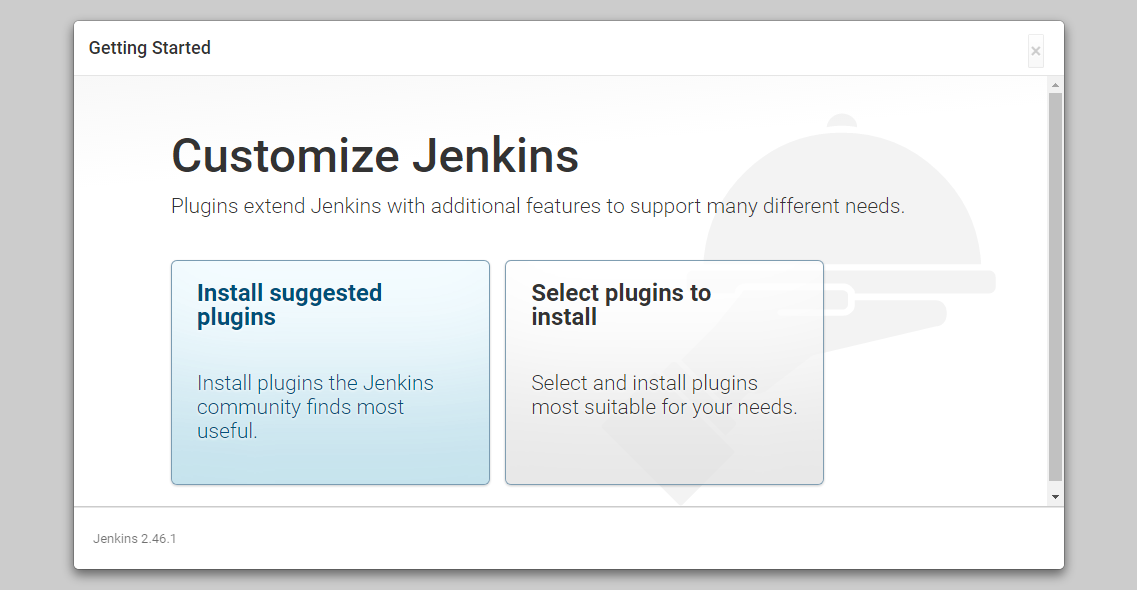
We should see "Unlock Jenkins" screen, which displays the location of the initial password



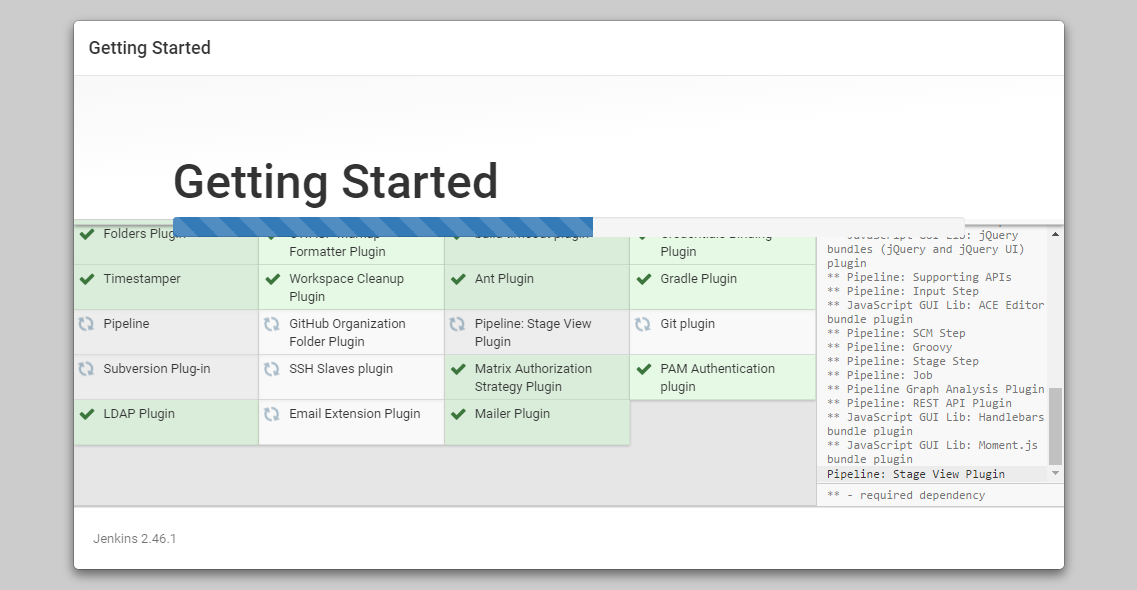
In the terminal window, we'll use the cat command to display the password:

* sudo cat /var/lib/jenkins/secrets/initialAdminPassword

We'll copy the 32-character alphanumeric password from the terminal and paste it into the "Administrator password" field, then click "Continue". The next screen presents the option of installing suggested plugins or selecting specific plugins.

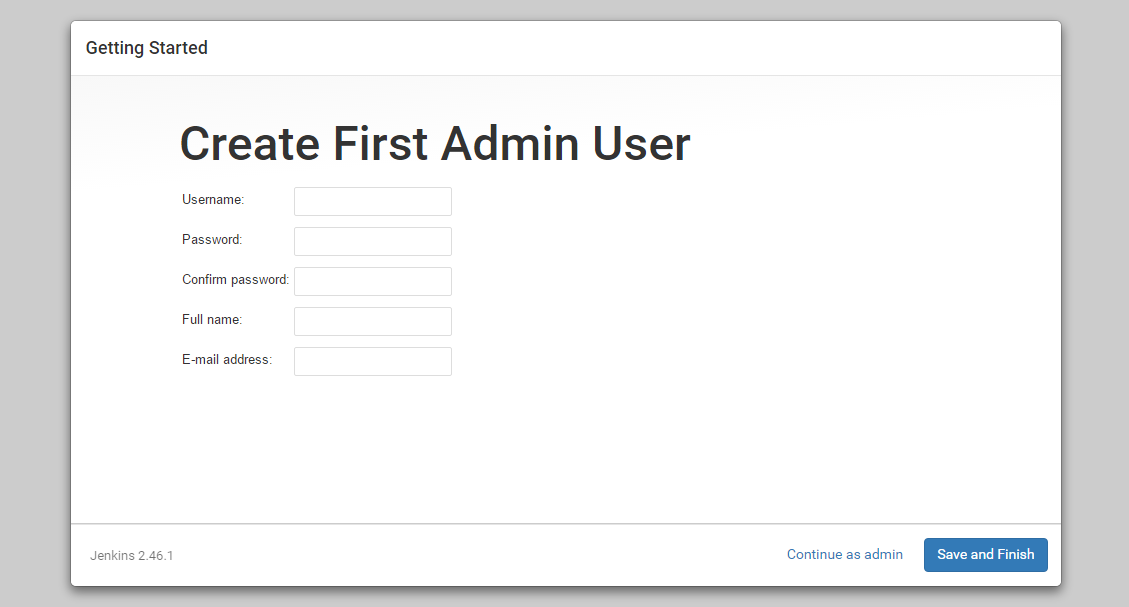


We'll click the "Install suggested plugins" option, which will immediately begin the installation process:

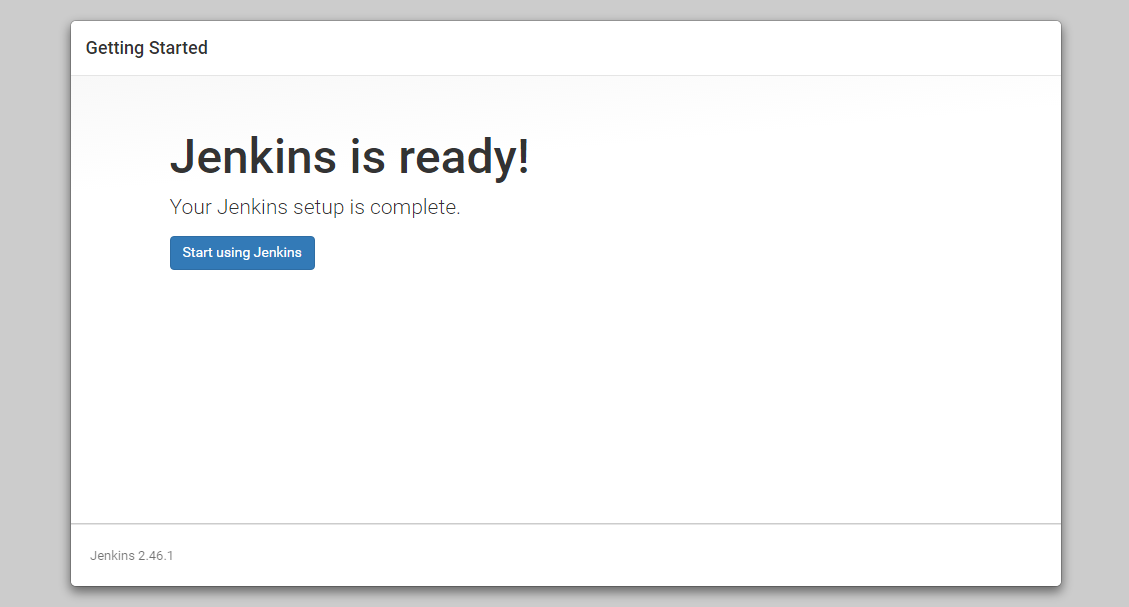


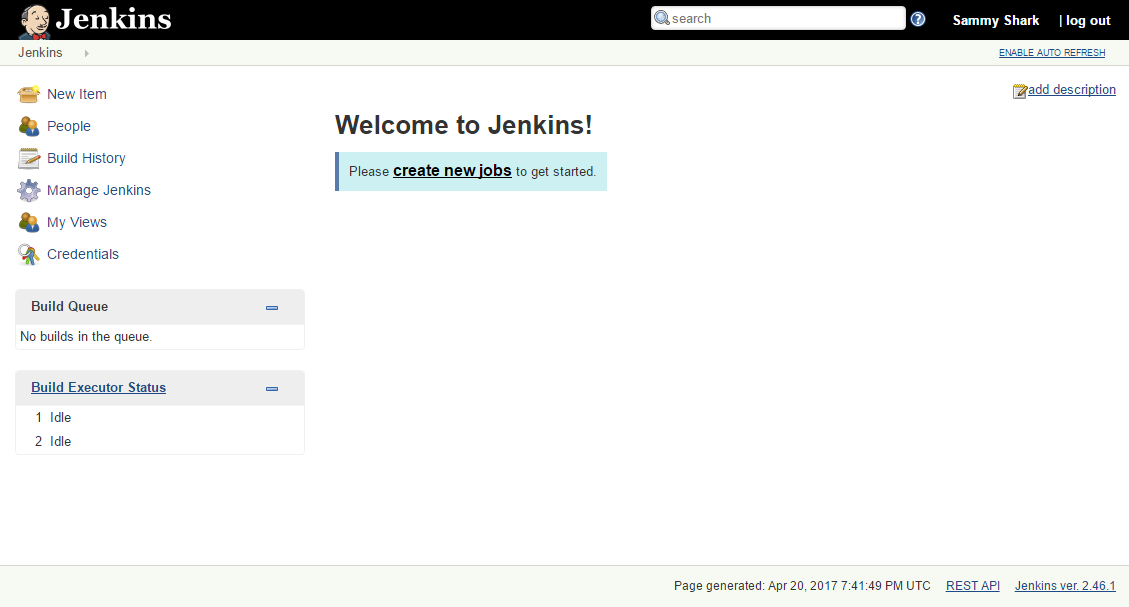
When the installation is complete, we'll be prompted to set up the first administrative user. It's possible to skip this step and continue as admin using the initial password we used above, but we'll take a moment to create the user.

**Note:** The default Jenkins server is NOT encrypted, so the data submitted with this form is not protected. When you're ready to use this installation, follow the guide [How to Configure Jenkins with SSL using an Nginx Reverse Proxy](https://www.digitalocean.com/community/tutorials/how-to-configure-jenkins-with-ssl-using-an-nginx-reverse-proxy). This will protect user credentials and information about builds that are transmitted via the Web interface.



Once the first admin user is in place, you should see a "Jenkins is ready!" confirmation screen.

  
Click "Start using Jenkins" to visit the main Jenkins dashboard:



At this point, Jenkins has been successfully installed.

Running Node Service in Jenkins

**Configuring Jenkins with the Necessary Plugins**

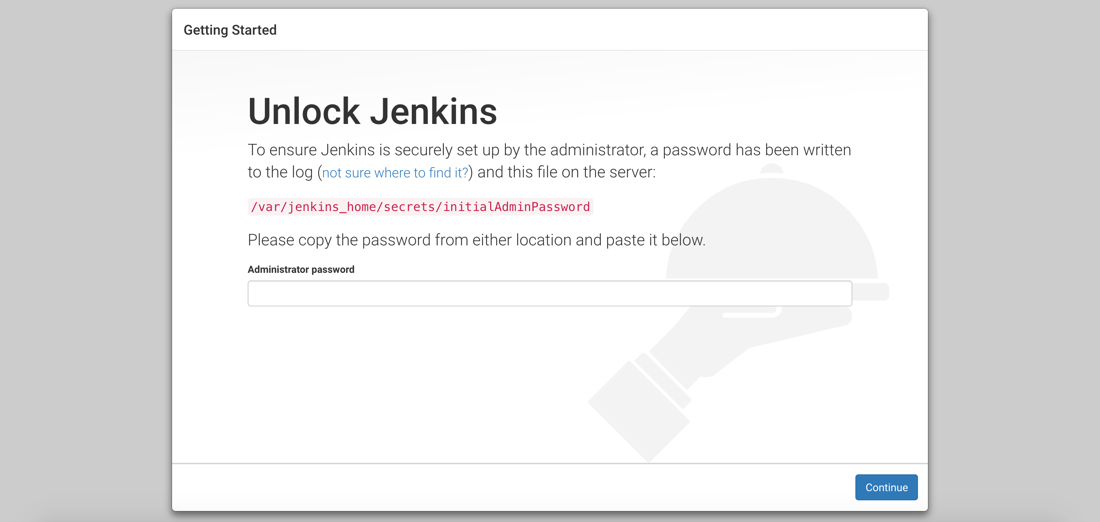
You should have already downloaded Jenkins by now.  If you haven’t go ahead and obtain the WAR file from the Jenkins website.

To start Jenkins, execute the following command from your Command Prompt or Terminal:

java -jar jenkins.war -httpPort=8080

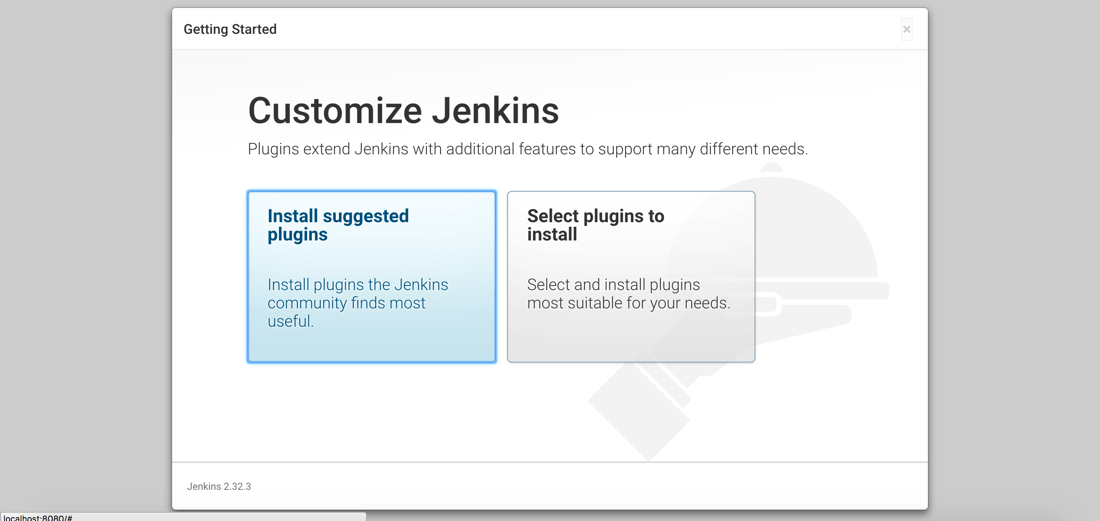
|  |  |
| --- | --- |
| 1 | java -jar jenkins.war -httpPort=8080 |

This will make Jenkins accessible from a web browser at **http://localhost:8080**.  Upon first launch, you’ll be placed in a configuration wizard.



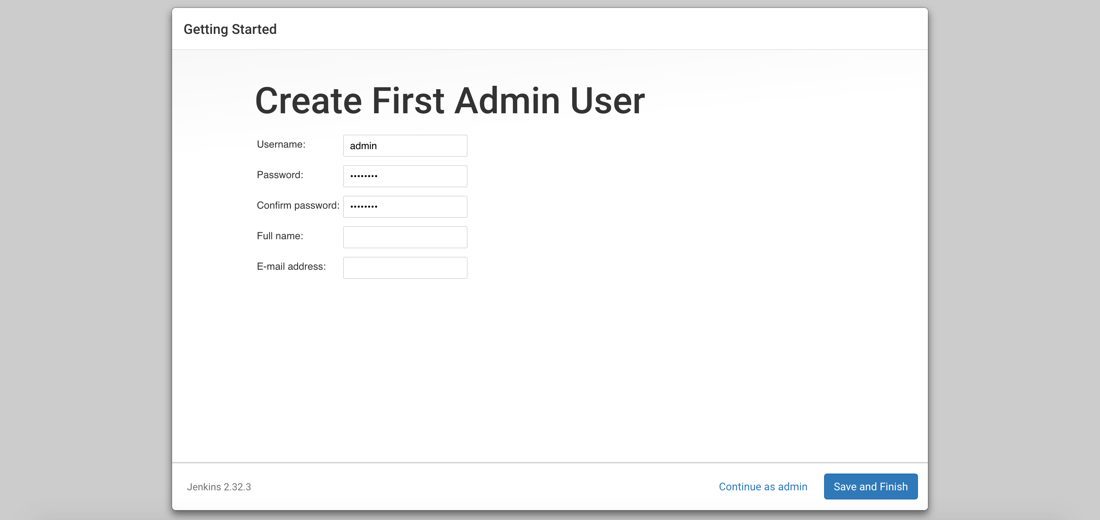
The first screen in this configuration wizard will ask you for the password that Jenkins generates.  Find it in the location presented on the screen.

The second screen will ask you which plugins you’d like to install.



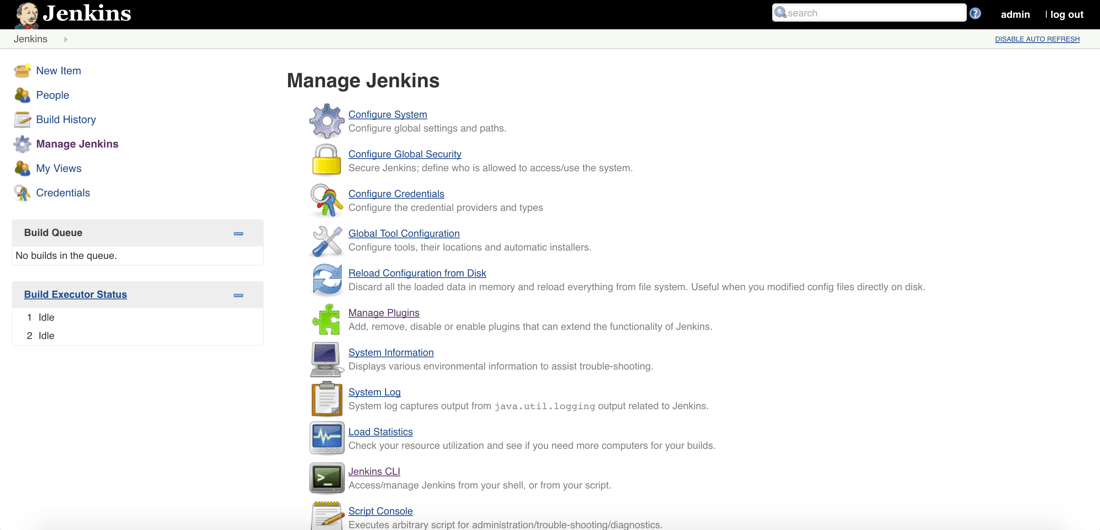
For now, we’re going to install the suggested plugins.  We’ll be installing extra plugins later.

The third screen will ask us to create our first administrative user.  Technically, the generated password you’re using is an administrative user, but you may want to create a new one.

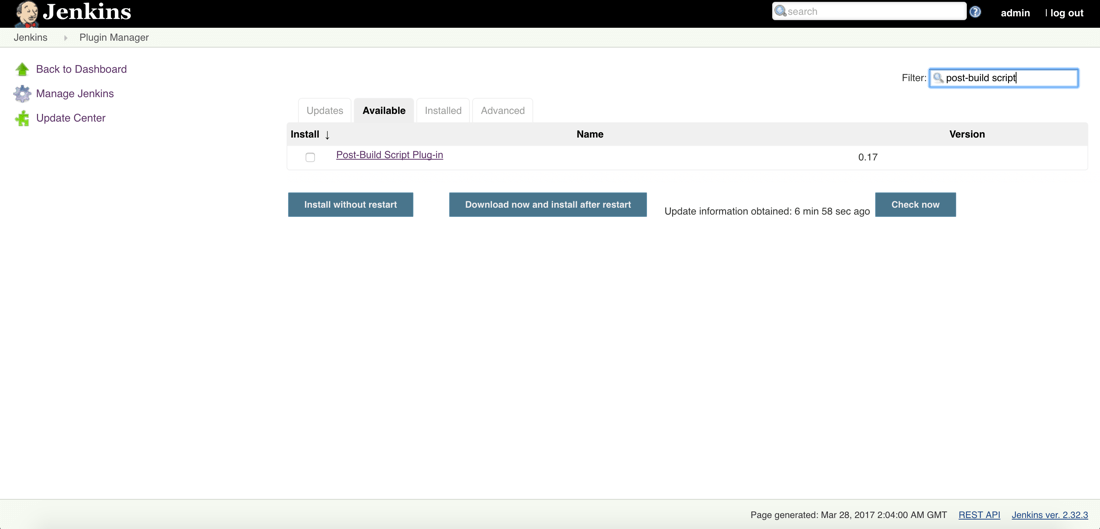


After you create a user, Jenkins is ready to go.  However, we are going to need another plugin and it can vary depending on how we wish to build and deploy the Node.js application.

From the main Jenkins screen, choose to **Manage Jenkins** to see a list of administration options.



What we care about is managing the available plugins.  After choosing **Manage Plugins** we want to search for and install a plugin by the name of **Post-Build Script**.



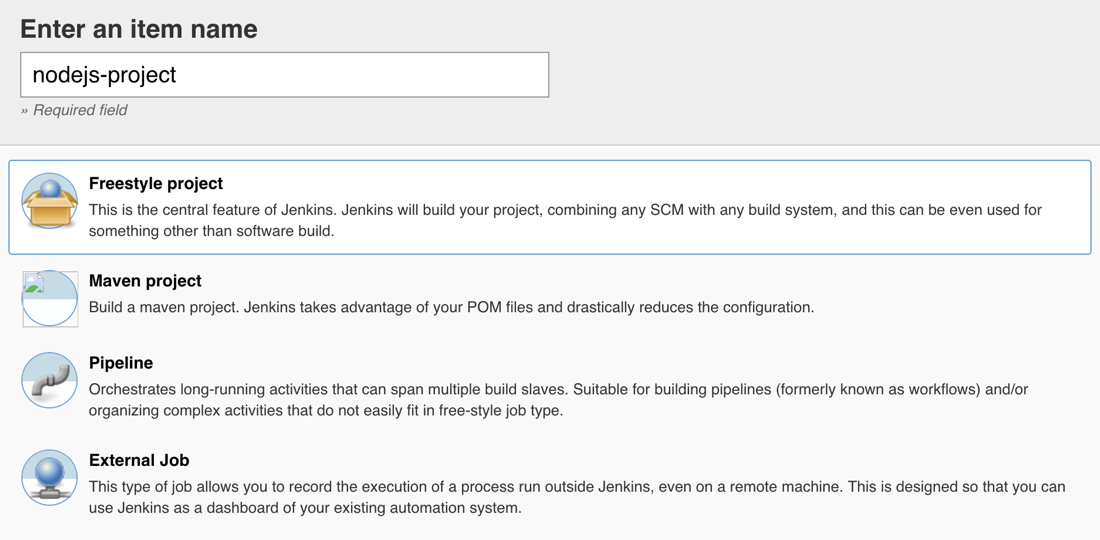
This plugin allows us to execute shell commands or scripts after the build stage has completed.  In this example we’ll be building and deploying to the same host, we can run everything locally via shell commands.  In a production environment you might want to use the SSH plugin to migrate the code to a remote server and run it there.

With the plugins available, let’s create our continuous deployment workflow for Node.js in Jenkins.

**Creating a Jenkins Continuous Deployment Workflow for Node.js**

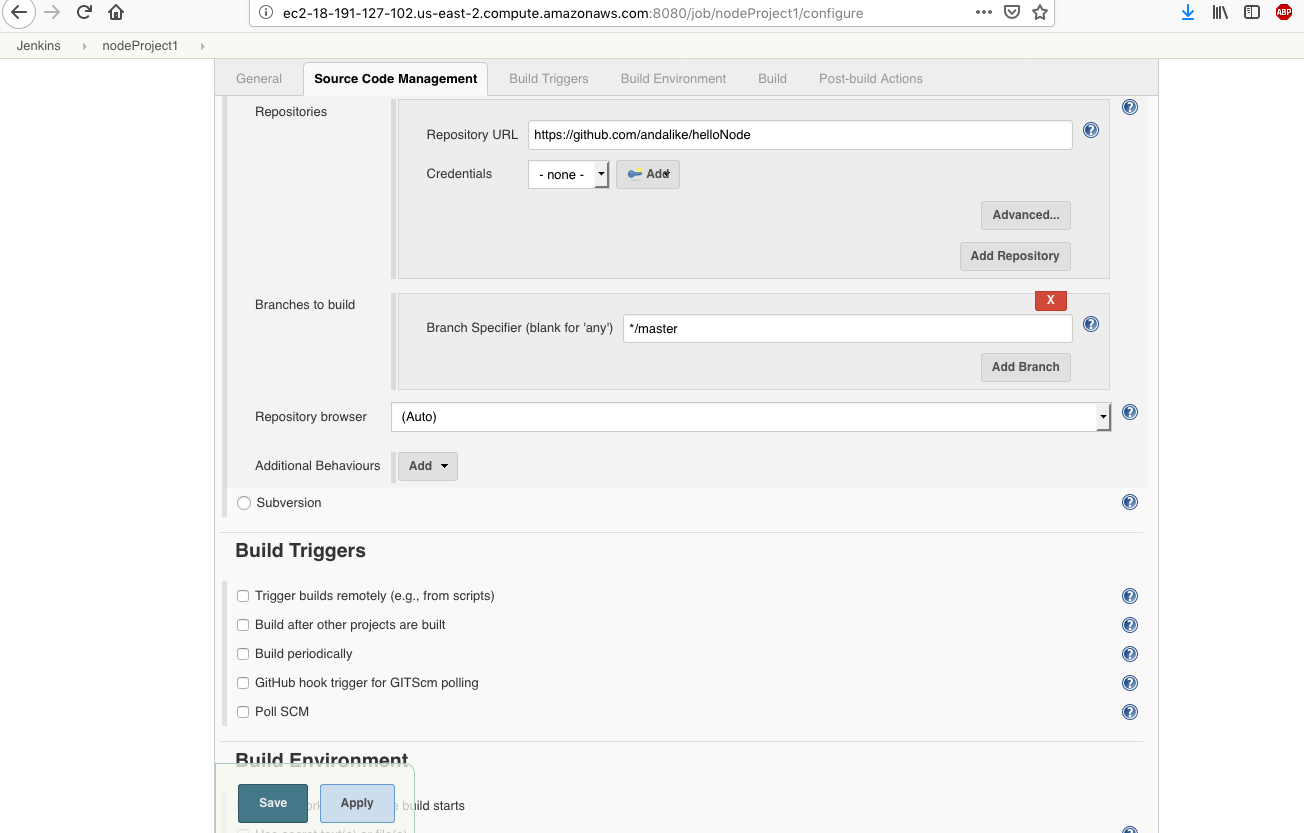
Just to reiterate, our goal here is to create a workflow that will pull a project from GitHub, build it by installing all the dependencies, and deploy it by running it on a server, in this case our local machine.

Start by creating a new item, otherwise known as a new job or workflow.



We’re going to be creating a **Freestyle Project**, but you can give it any name you want.  There are three things that need to be done on the next screen.

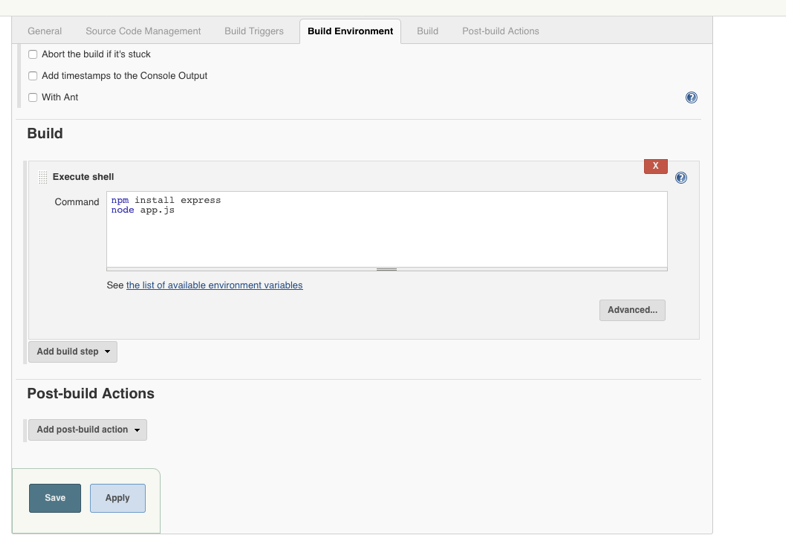
The source of our workspace will come from GitHub.  In your own project it can come from elsewhere, but for this one we need to define our source control information.

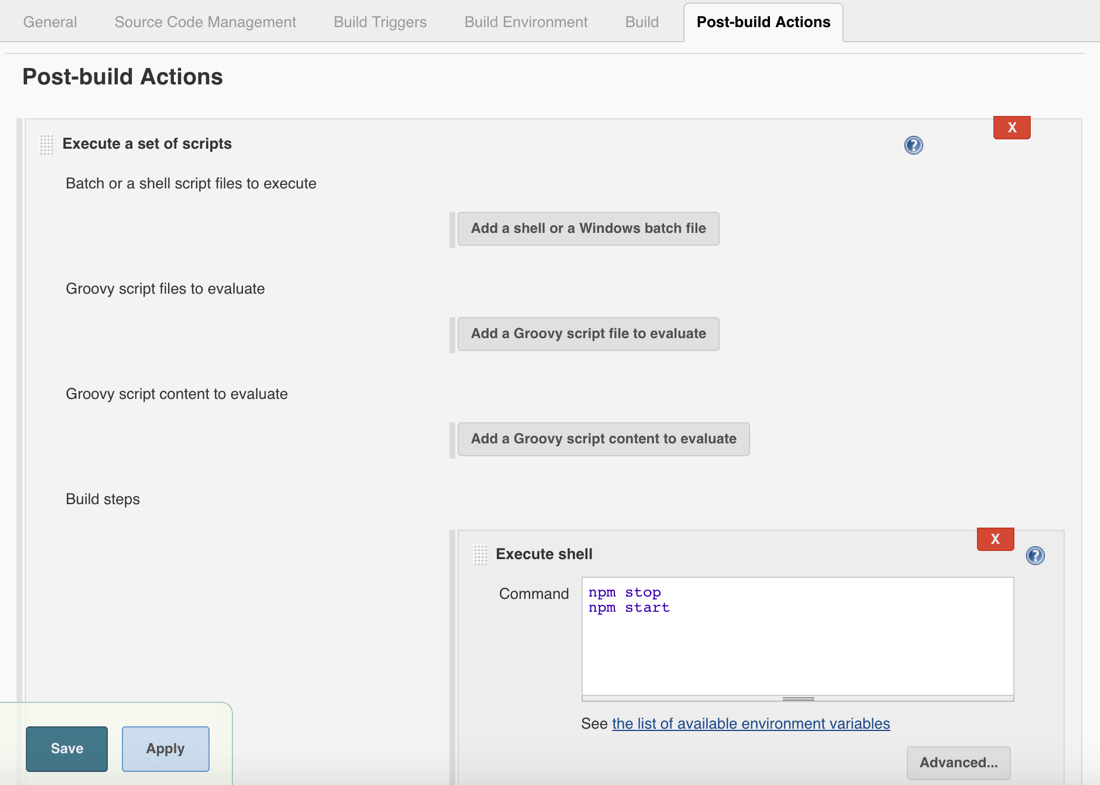


The GitHub project is one that I had previously created and written about, like mentioned before.  The project can be found at:

https://github.com/andalike/helloNode

Now in a production environment you’ll probably want to set up GitHub hooks to trigger the job process, but since this is all on localhost, GitHub won’t allow it.  Instead we’ll be triggering the job manually.





In this example we will be deploying the application locally on our machine.