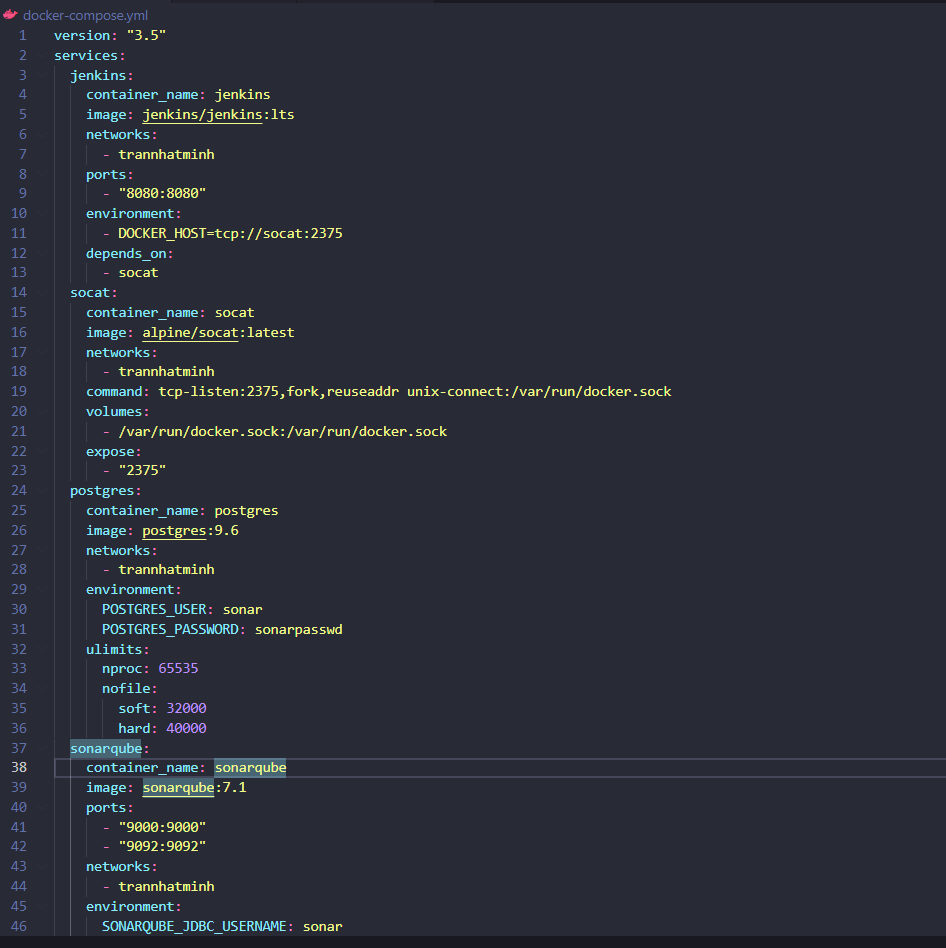
This project experience involves Amazon Web Services (AWS), and I will use the AWS Management Console to complete all the project steps.

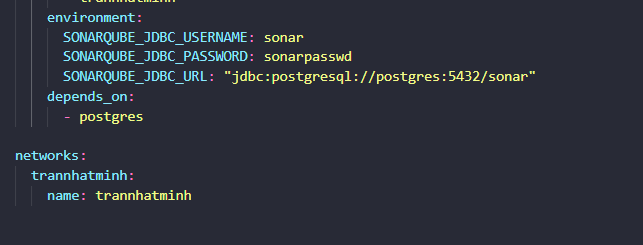
# Launch Jenkins and SonarQube Docker Containers

In this project step, I will complete the provisioning of your Jenkins and SonarQube CICD environment. To do so, I will need to SSH into the ec2 instance, and perform a git clone of the repository located on GitHub, at the followinglocation https://github.com/TranNhatMinh23/Devops\_Maven\_Jenkins\_Sonarqube\_Jfrog\_Jira.git

Next, I will use the docker-compose command to launch the Jenkins and SonarQube CICD docker containers. Two additional docker containers are launched to support the environment, Postgres and Socat. Postgres is the backend SQL database for SonarQube, and Socat provides a bi-directional TCP comms link for running "docker inside docker" - this is only required, for example, if the Jenkins docker container is configured to build docker containers itself, etc. With this completed, I will then use the docker client to query and inspect several environment values that are required for the remainder of this Lab. In particular, I will use it to query for the:

cat docker-compose.yml

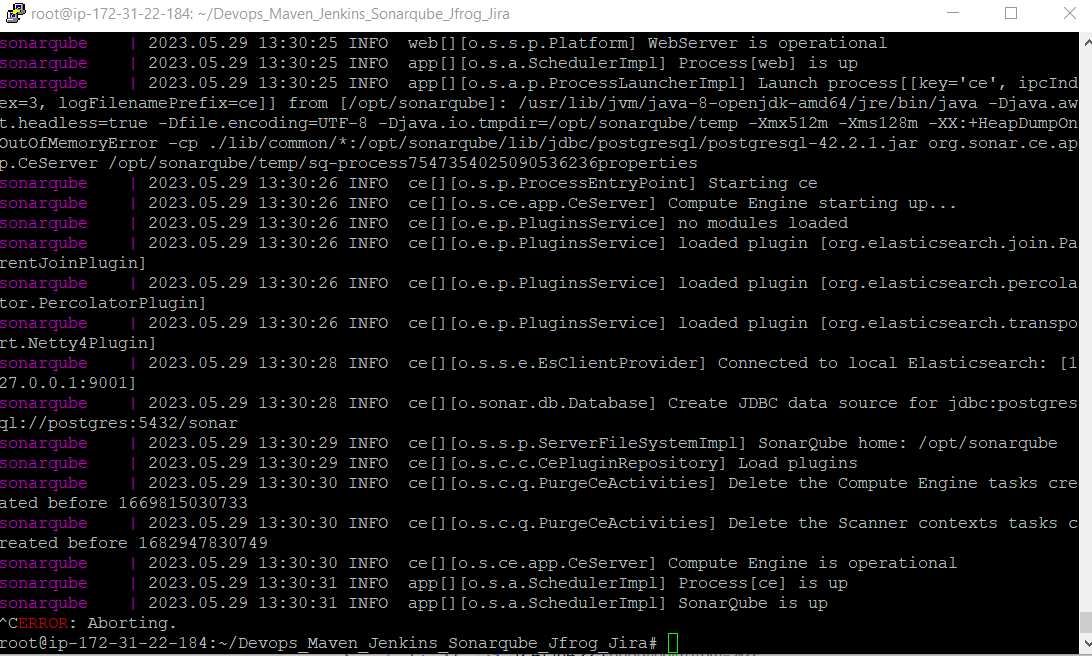




Launch the Jenkins and SonarQube docker environment. Running these commands will grab the latest Jenkins docker image, and then launch the docker environment with logging enabled. The docker compose container provisioning process typically takes 1-2 minutes to complete. Upon conclusion four (4) docker containers will be launched Jenkins, SonarQube, Postgres, and Socat:

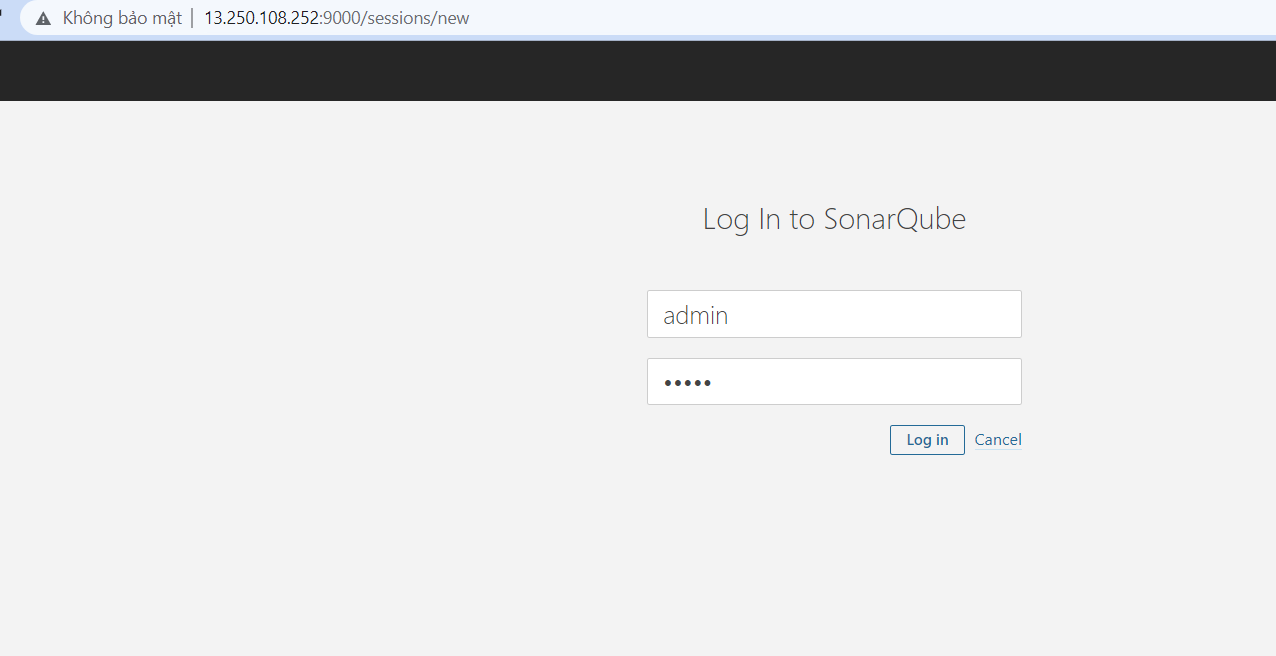
docker pull jenkins/jenkins:lts

docker-compose up -d && docker-compose logs -f



docker exec -it jenkins cat /var/jenkins\_home/secrets/initialAdminPassword

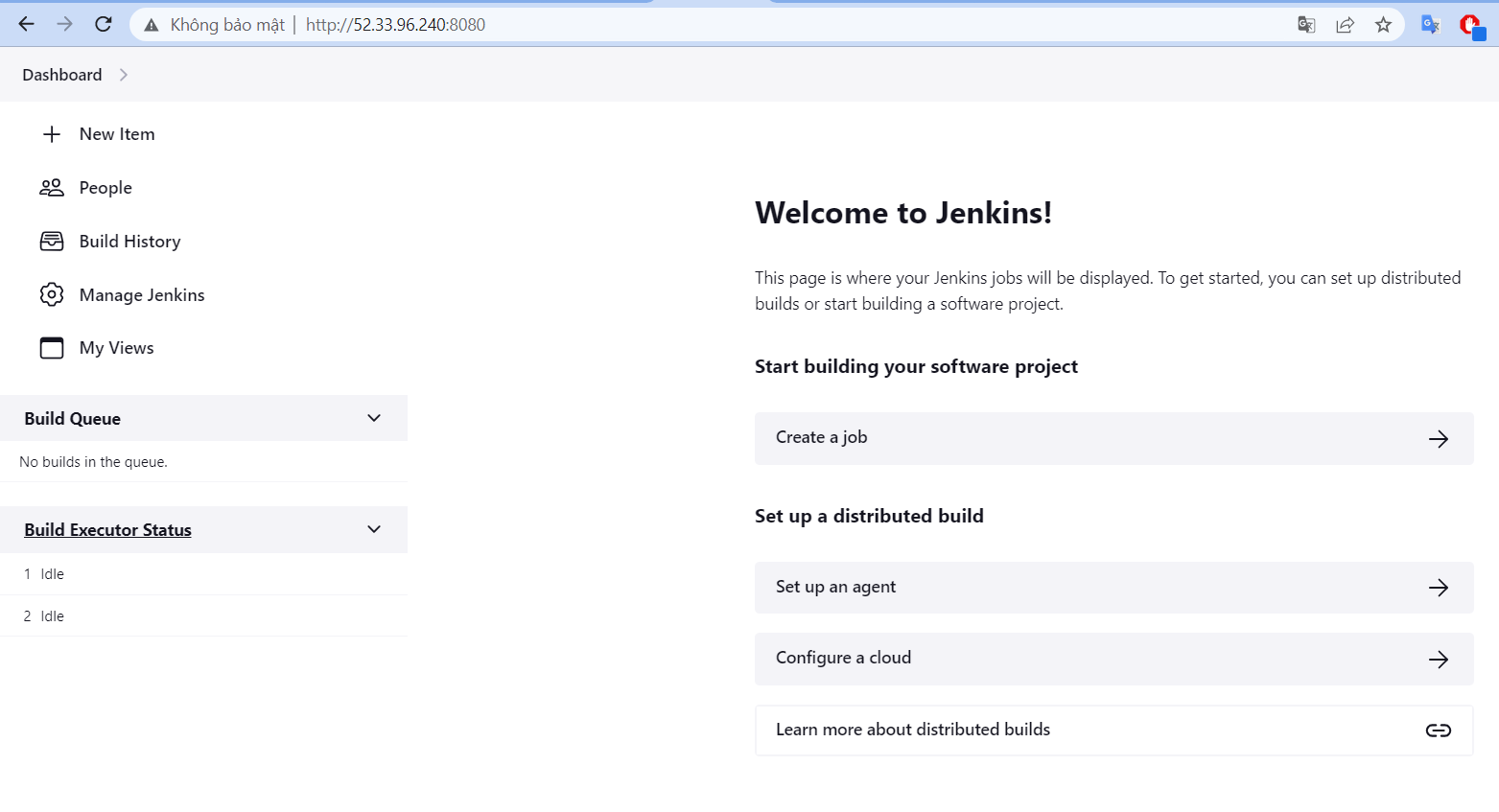
The SonarQube docker container has been configured to listen for inbound connections on port **9000**. Using your browser, navigate to the SonarQube home page: http://PUBLIC\_IP\_CICD\_PLATFORM\_INSTANCE:9000.



After completing the Jenkins installation and configuration (in the following Project steps), and initiating a Jenkins CICD build of a sample Java servlet web application, I will revist this page. I will see that Jenkins automatically forwards the respective source code into SonarQube for static code analysis, resulting in a new project being registered here.

The Jenkins docker container has been configured to listen for inbound connections on port **8080**.

Enter the initial administrator password, captured previously using the command: docker exec -it jenkins cat /var/jenkins\_home/secrets/initialAdminPassword, then click **Continue**:



Click on the **Manage Jenkins** menu item taking I into the **Manage Jenkins** screen. Ignore any update and/or security messages that Jenkins may announce as this is a Project environment only:  
Click on the **Manage Plugins** menu item taking I into the **Plugin Manager**:

Click the **Available** tab and enter SonarQube within the search input field associated with the Plugins Manager pane. Select the **SonarQube Scanner** option and click **Install without restart**:

Click on the **Manage Jenkins** > **Global Tool Configuration** menu item:

Under **SonarQube Scanner**, click **Add SonarQube Scanner**. Enter sonar in the **Name** field and ensure that the **Install automatically** checkbox is ticked. Then select the **SonarQube Scanner 3.2.0.1227**version under **Install from Maven Central**. Click **Apply** to commit the **SonarQube Scanner** plugin installation settings:

Staying on the **Global Tool Configuration** screen, scroll back up. Under Gradle, click **Add Gradle**. Enter gradle-4.10.2 in the **Name** field and ensure the **Install automatically** checkbox is ticked. Then select the **Gradle 4.10.2** version under **Install from Gradle.org**. Click **Apply** to commit the **Gradle** plugin installation settings:

Click **Manage Jenkins**> **Credentials** to be taken into the Credentials screen:

Set **Kind** to **Secret text**, and ensure that the **Scope** is set to **Global**, then set:

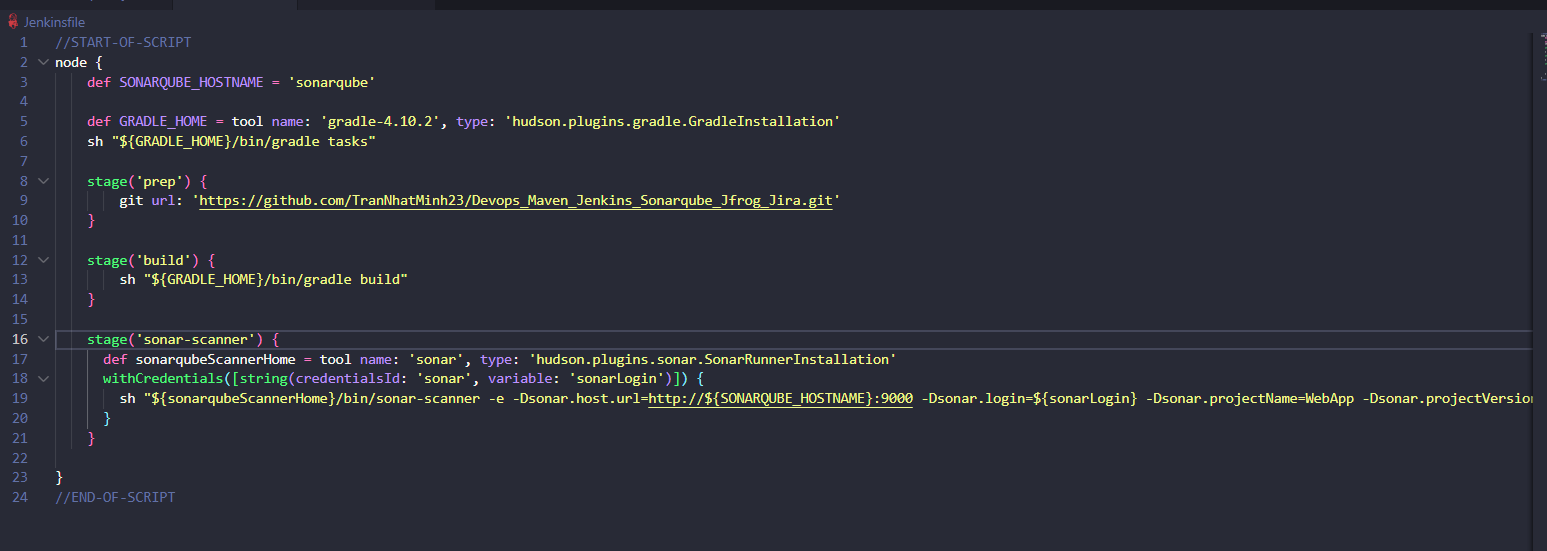
Secret: Enter the secret token value I generated in the SonarQube application

I will continue using the Jenkins administration web console. I will create a new build pipeline job. Gradle is used to manage and download all the Java library dependencies, perform a compilation, and package the outputs into a deployable WAR (web archive) file. Finally, the build pipeline will publish the Java source code into SonarQube for static code analysis.

Click **Create a job** on the Jenkins home page:

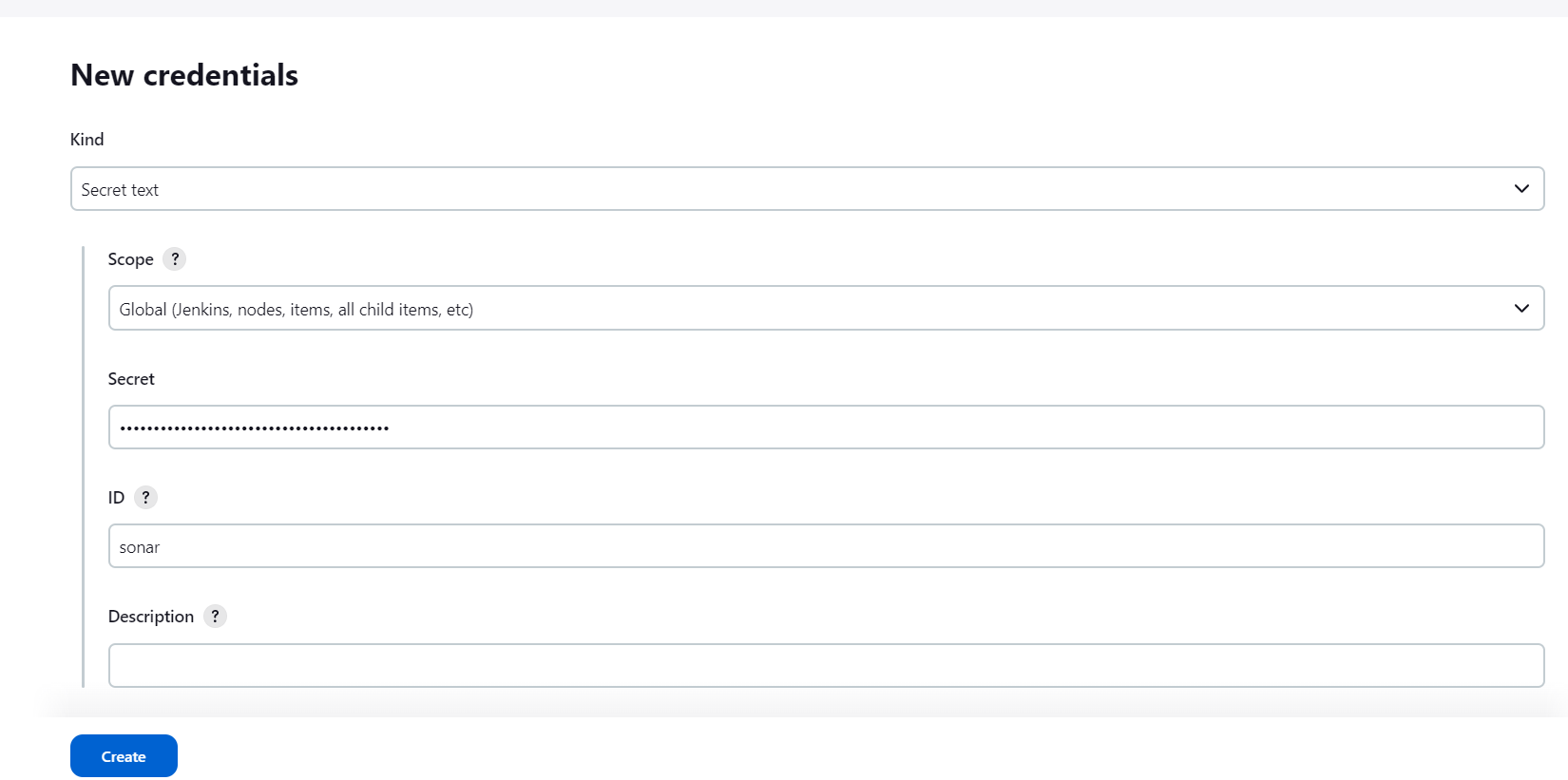
Enter a build job name such as buildjob1, and select **Pipeline**, followed by clicking **OK**:

Change the Pipeline **Definition** to **Pipeline script from SCM** and set **SCM** to **Git**. Set the **Repository URL** to https://github.com/TranNhatMinh23/Devops\_Maven\_Jenkins\_Sonarqube\_Jfrog\_Jira.git and leave all remaining settings as defaults, ensuring in particular that the **Script Path** setting is set to **Jenkinsfile**. Click **Save**:



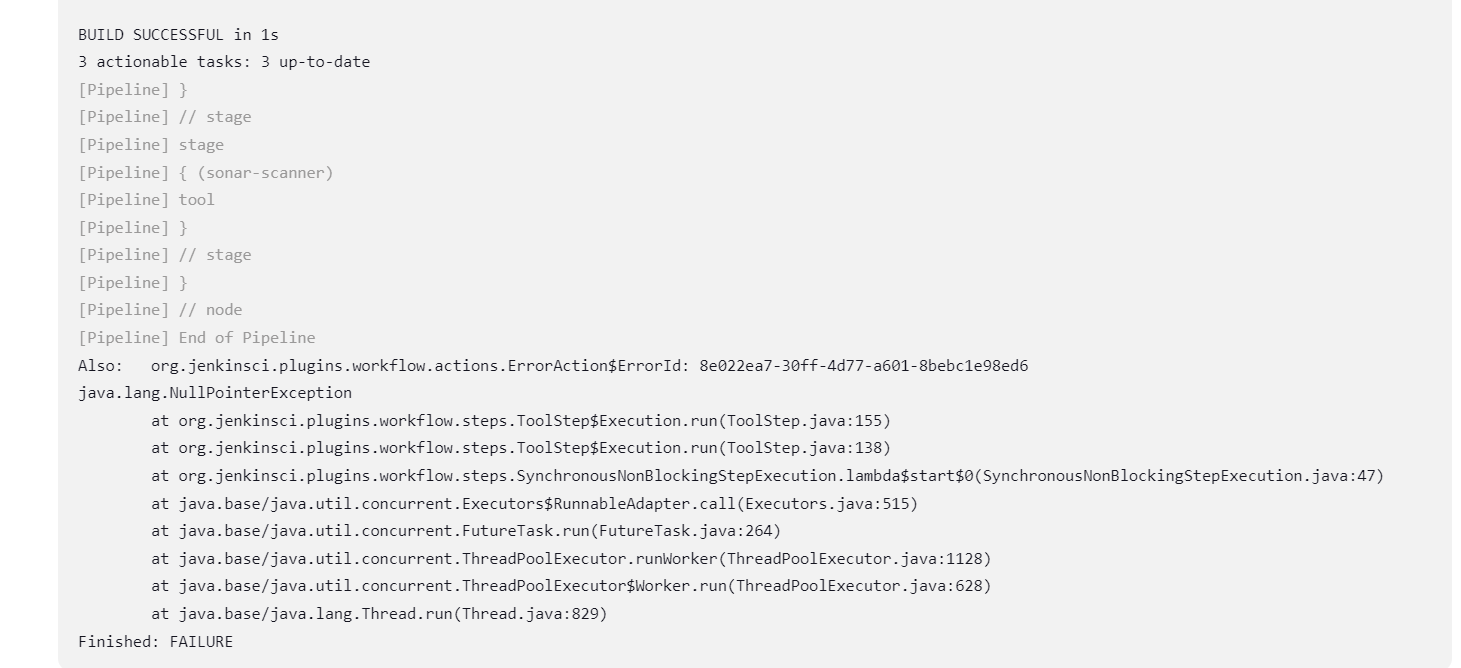
The Jenkinsfile is a scripted pipeline and drives the overall Jenkins CICD build process. The **prep** stage git clones the java webapp, which is then compiled using Gradle in the **build** stage. Finally, the **sonar-scanner** stage sends the Java source code over to SonarQube for static analysis.

Scroll through the Console Output build details all the way to the bottom, confirming there are no build errors, and that the build has completed successfully as per the Finished: SUCCESS message at the end:





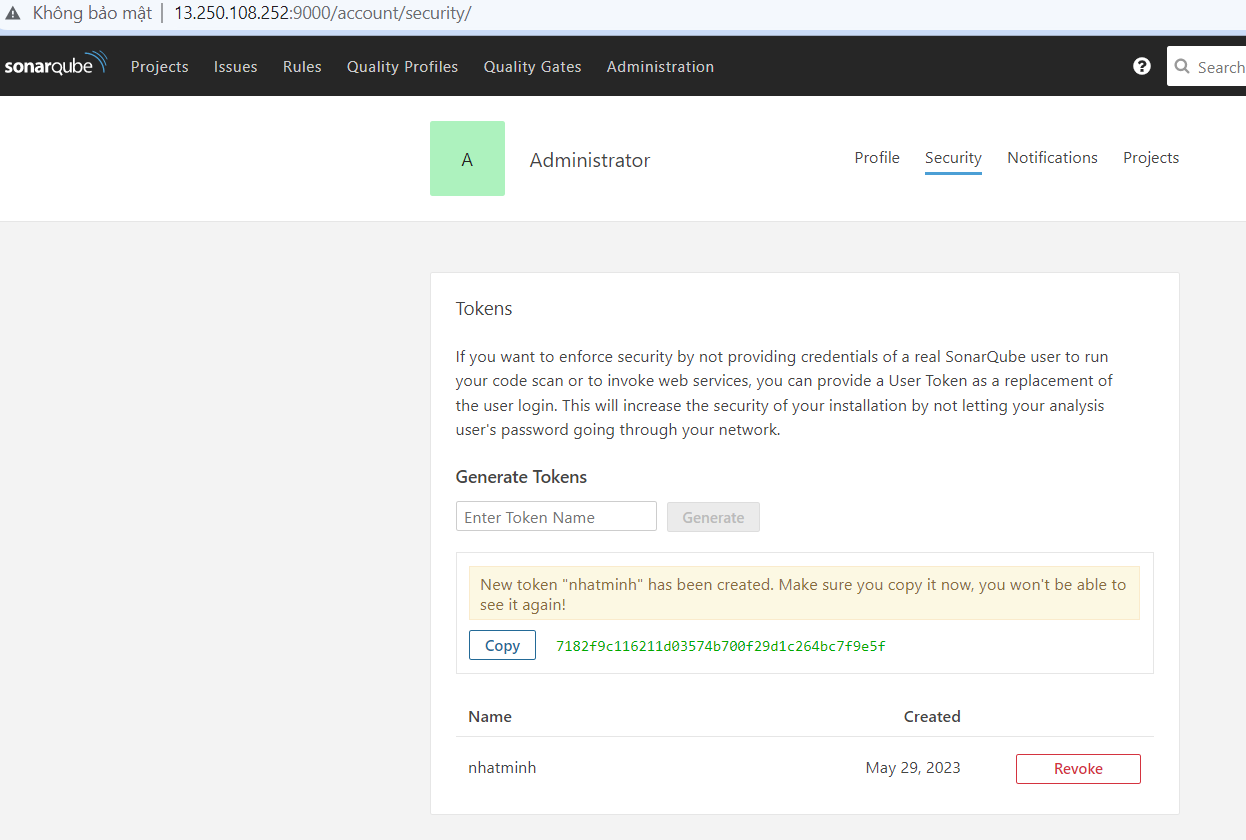
error branche git (Fix by master -> main, search error google)

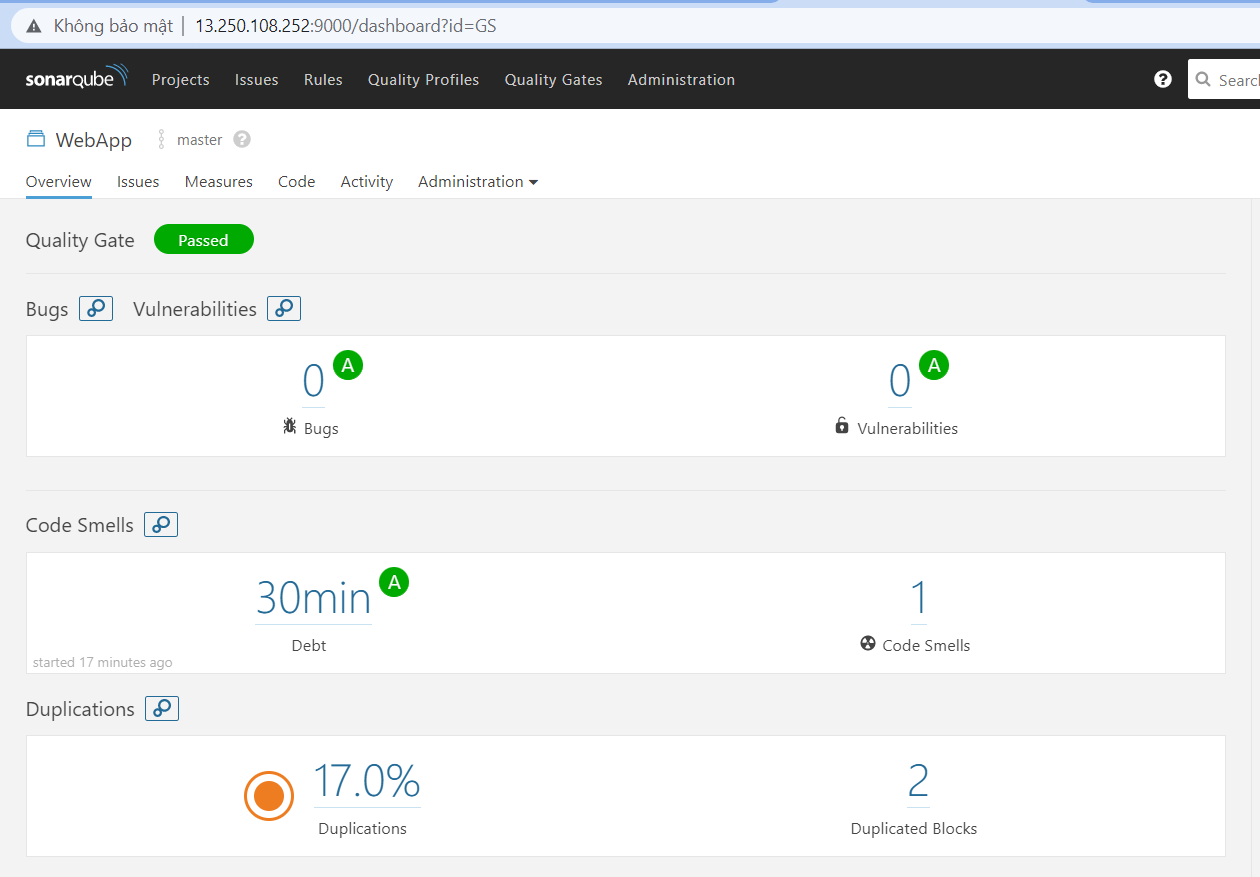


Error because forget not name sonarqube in global config



Success after fix 2 error





**Create Jenkins CICD Pipeline with JFROG Artifactory Integration for Build Artifact Management**

Integrating Jenkins with Jfrog Artifactory provides I with an automated platform for building and managing deployable artifacts (binaries).

In this lab, I will launch a Jenkins and Artifactory CICD environment using Docker containers on a provided EC2 instance. I will then configure a Jenkins build pipeline to build, compile, The build pipeline will publish the build artifacts into Artifactory, which in turn will catalog and register them.

This project is aimed at DevOps and CICD practitioners, and, in particular, build and release engineers interested in managing and configuring Jenkins together with Artifactory to manage and maintain build artifacts.

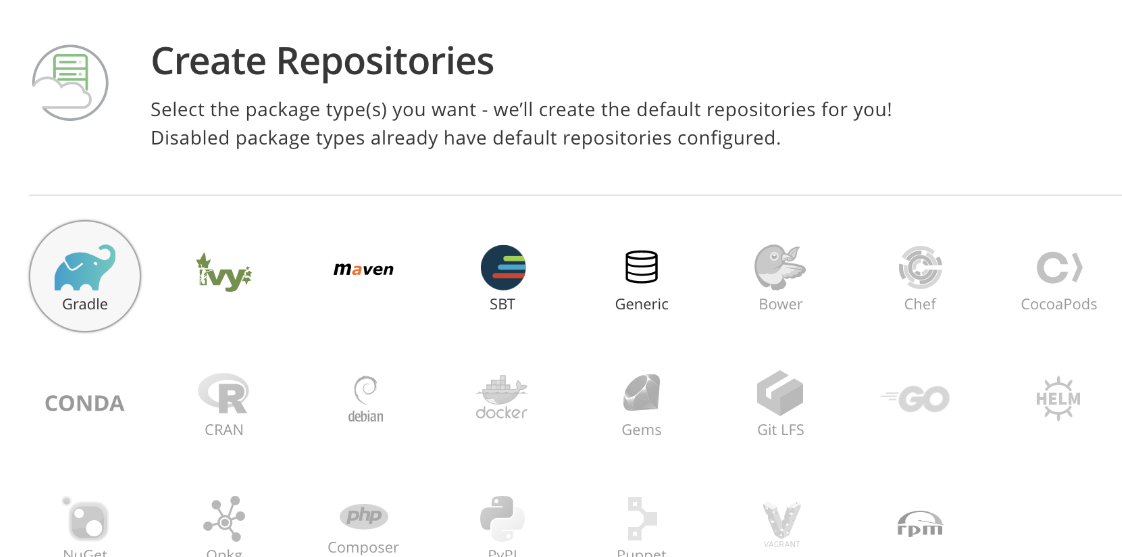
I will login into the Artifactory administration web console and complete the default installation. I will enable the default Gradle artifact repositories.

The Artifactory docker container has been configured to listen for inbound connections on port **8081**. Using your browser, navigate to the Artifactory home page: http://PUBLIC\_IP\_CICD\_PLATFORM\_INSTANCE:8081. Remember to use the public IP address assigned to the *cicd.platform.instance* EC2 instance.

Login using the default Artifactory credentials:

Username: admin

Password: password

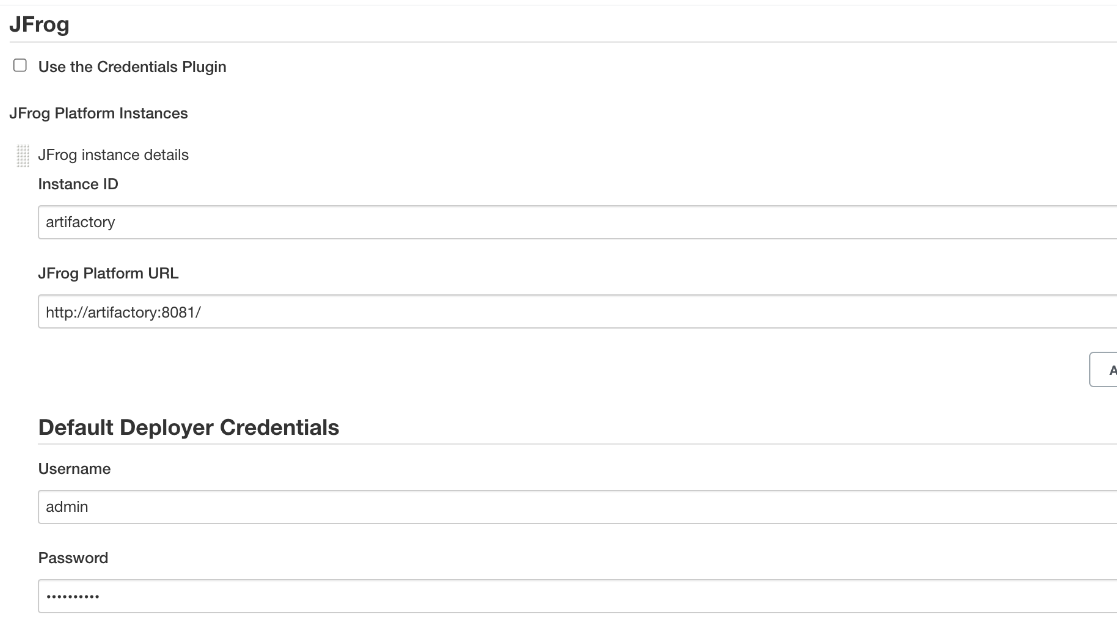
On the **Create Repositories** screen, select **Gradle** and click **Create**: 

I will continue using the Jenkins administration web console. I will now install the Gradle and Artifactory plugins, required to first perform a build, and then publish the resulting artifact into Artifactory.

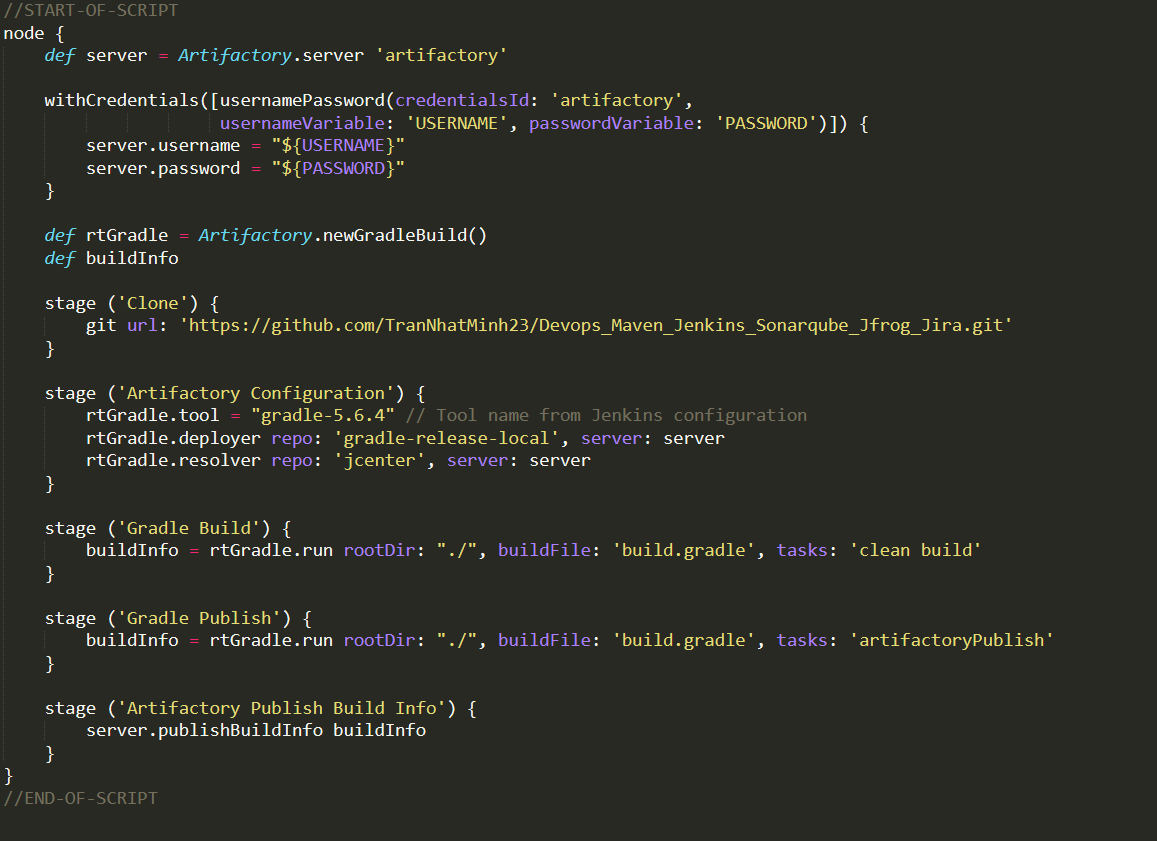
Click the **Available** tab and enter Artifactory within the search input field associated with the **Plugins Manager** pane. Select the **Artifactory** option and click **Install without restart**:

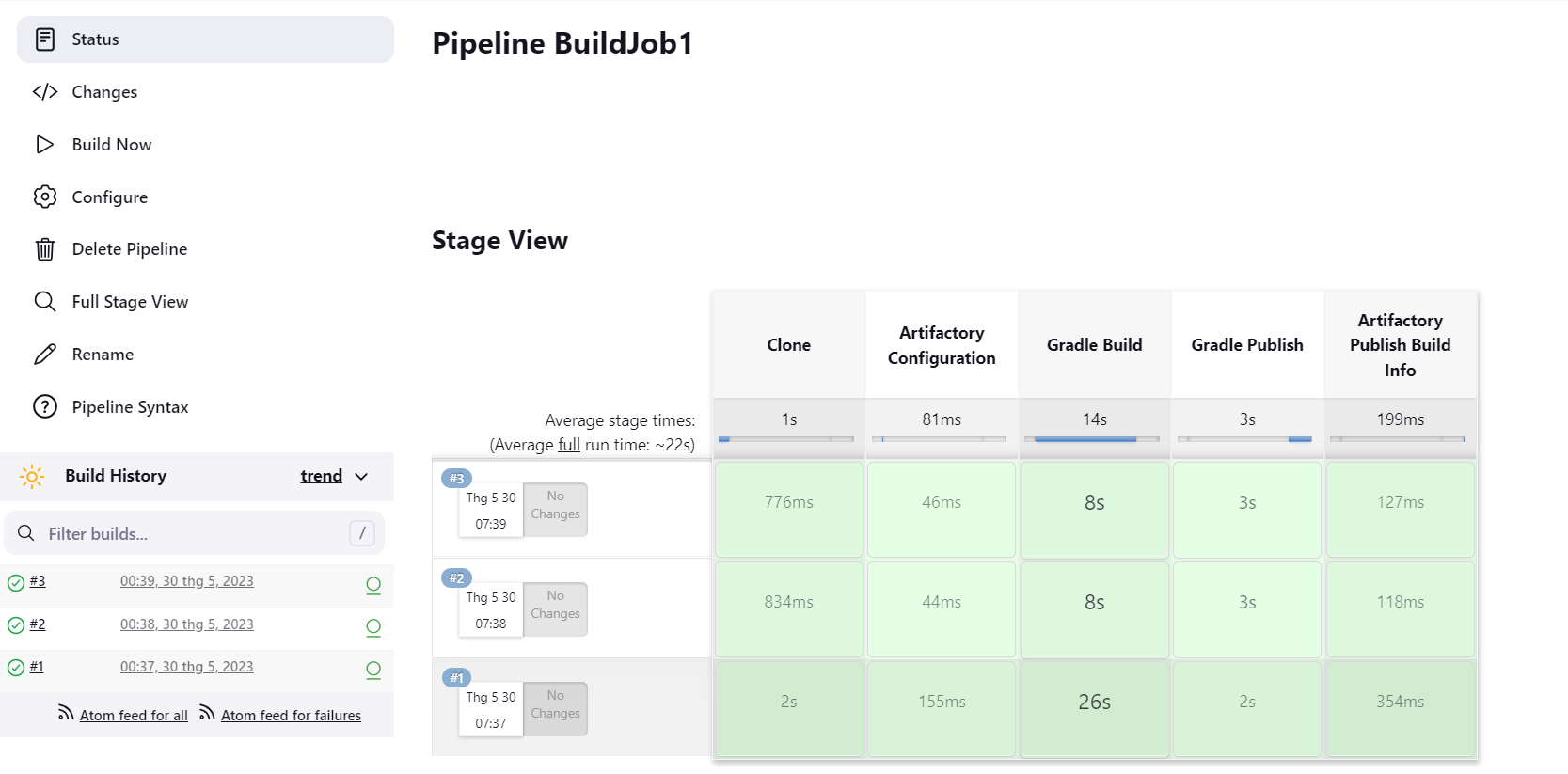
Click the **Manage Jenkins** menu item and then **Configure System**:

Under **JFrog** click the **Add JFrog Platform Instance** button. Enter the following Artifactory Server settings:



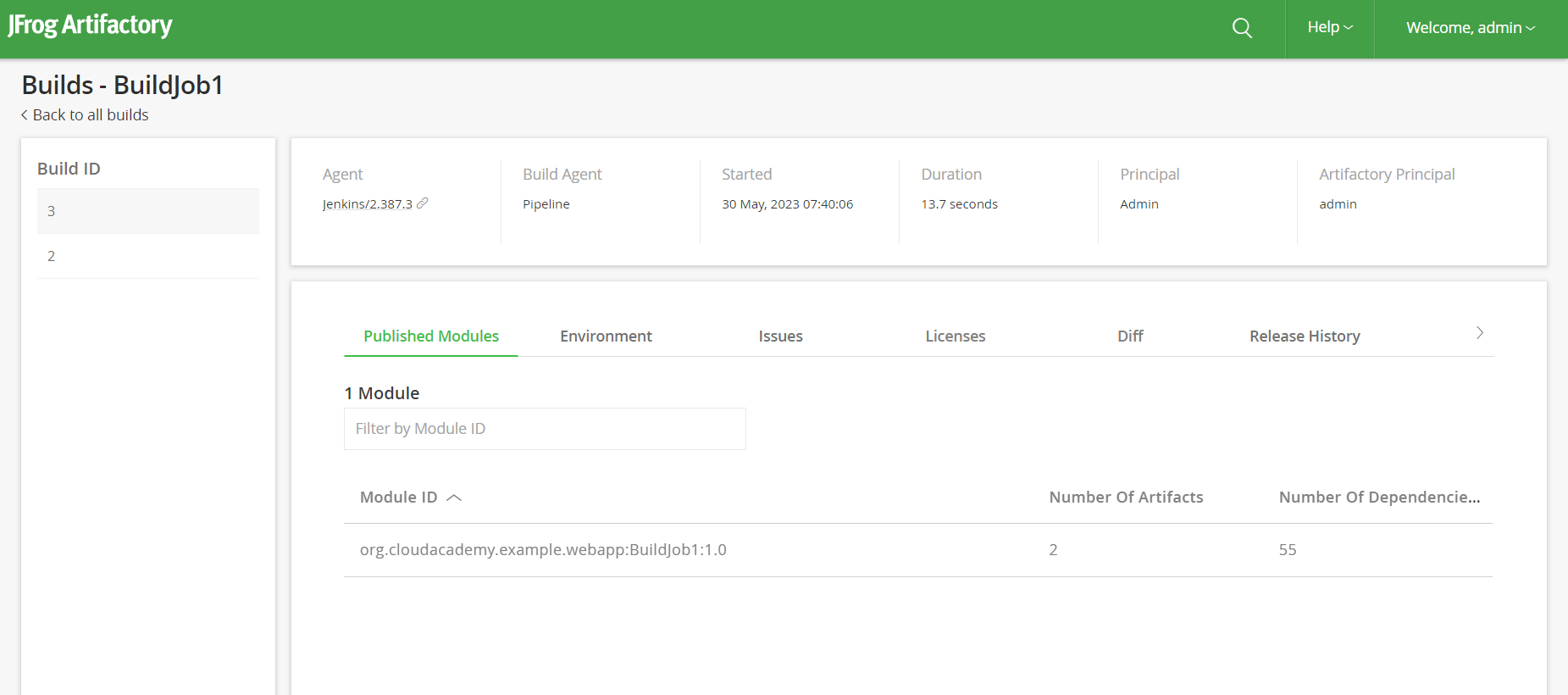
Within the current build job, click the **Pipeline**tab to be taken directly to the Pipeline section:





Notice how there are *Artifactory Build Info* menu links that will take I directly into the build information registered within Artifactory. These links currently use the internal Artifactory docker host name within the URL, so it will not work from your browser. To get these links to work, replace the internal Artifactory docker host name with the Public IP address that is assgined to your *cicd.platform.instance* EC2 instance. For example:

http://artifactory:8081/artifactory/webapp/builds/BuildJob1/1

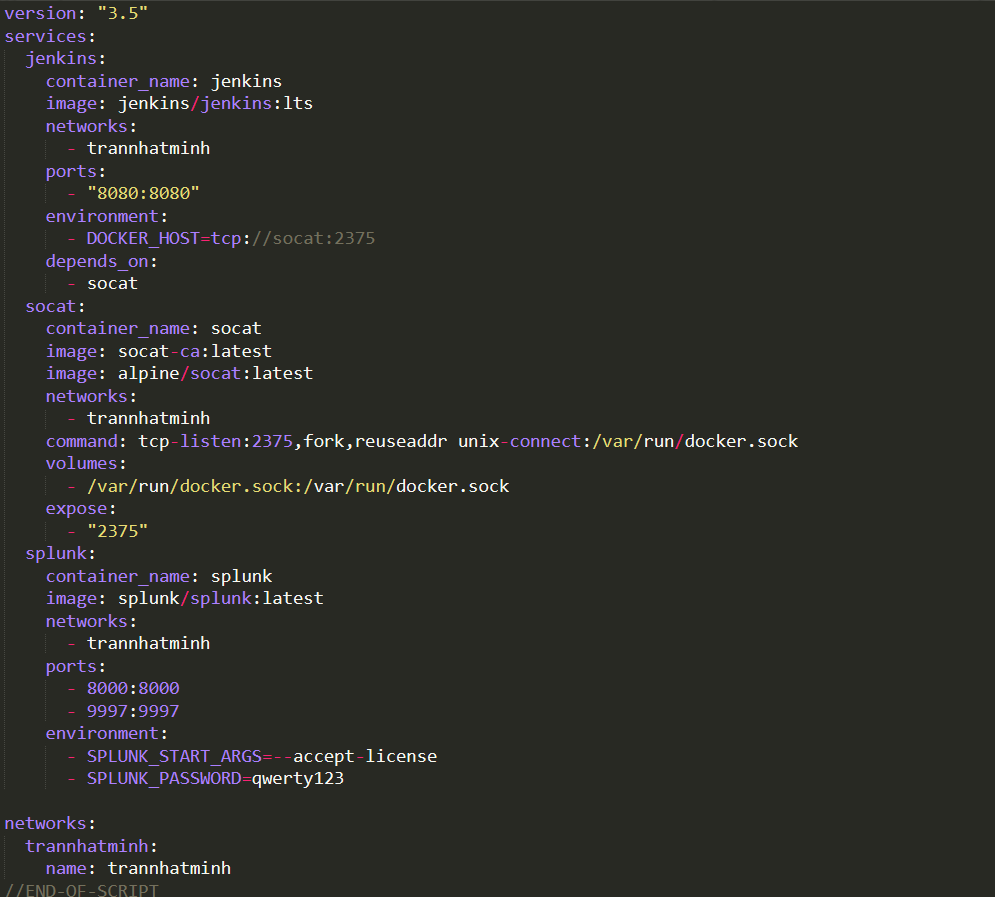


**Create a Jenkins CICD Pipeline to Build a Docker Image with Splunk Integration**

Integrating Jenkins with Docker provides I with a robust CICD method for building and packaging runnable containers. Building and deploying your docker containers is only half the challenge. Once they are in production, I will want to monitor and assess their operational performance. Splunk can provide I with this capability, through its support for enterprise-grade monitoring, logging, and diagnostics collection.

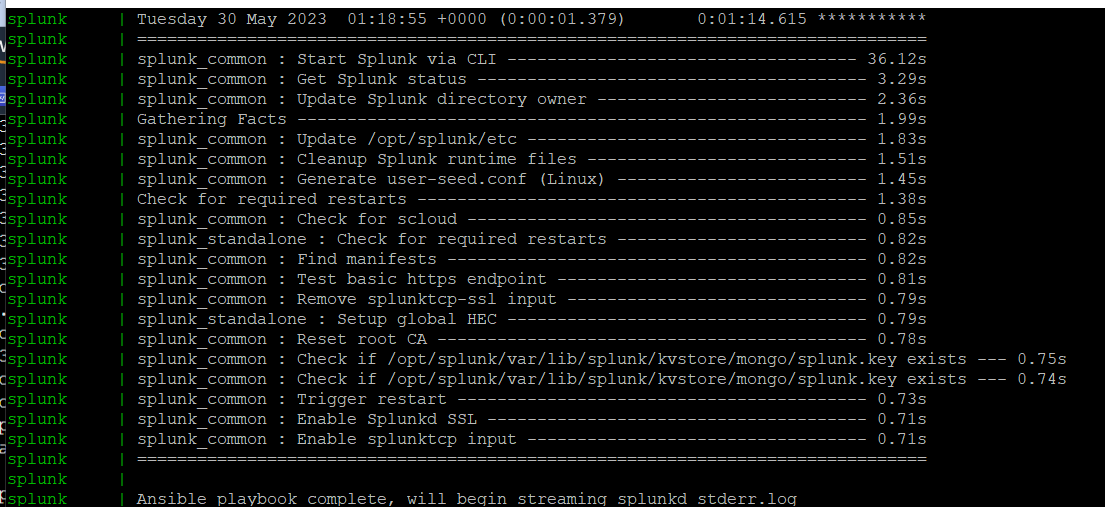
In this lab, I will launch a Jenkins and Splunk CICD and monitoring environment using Docker containers on a provided EC2 instance. I will then configure a Jenkins build pipeline to build, compile, and package project into a runnable Docker image. The Jenkins build process will create the Docker image using a Dockerfile. The build process integrates Splunk logging into the Docker image, by installing the Splunk Forwarding Agent at build time. At runtime, the docker container will then have the ability to publish runtime logging information into the Splunk service.

I will use the docker-compose command to launch the Jenkins and Splunk docker containers. One additional docker container named Socat is also launched and will be used to provide a bi-directional TCP comms link for running docker inside docker. This is required by the Jenkins docker container, which will be later configured to build docker containers itself within a build pipeline. With this completed, I will then use the docker client to query and inspect several environment values that I will require in the remainder of this lab. In particular, I will use it to query for the:



docker pull jenkins/jenkins:lts

docker-compose up -d && docker-compose logs -f



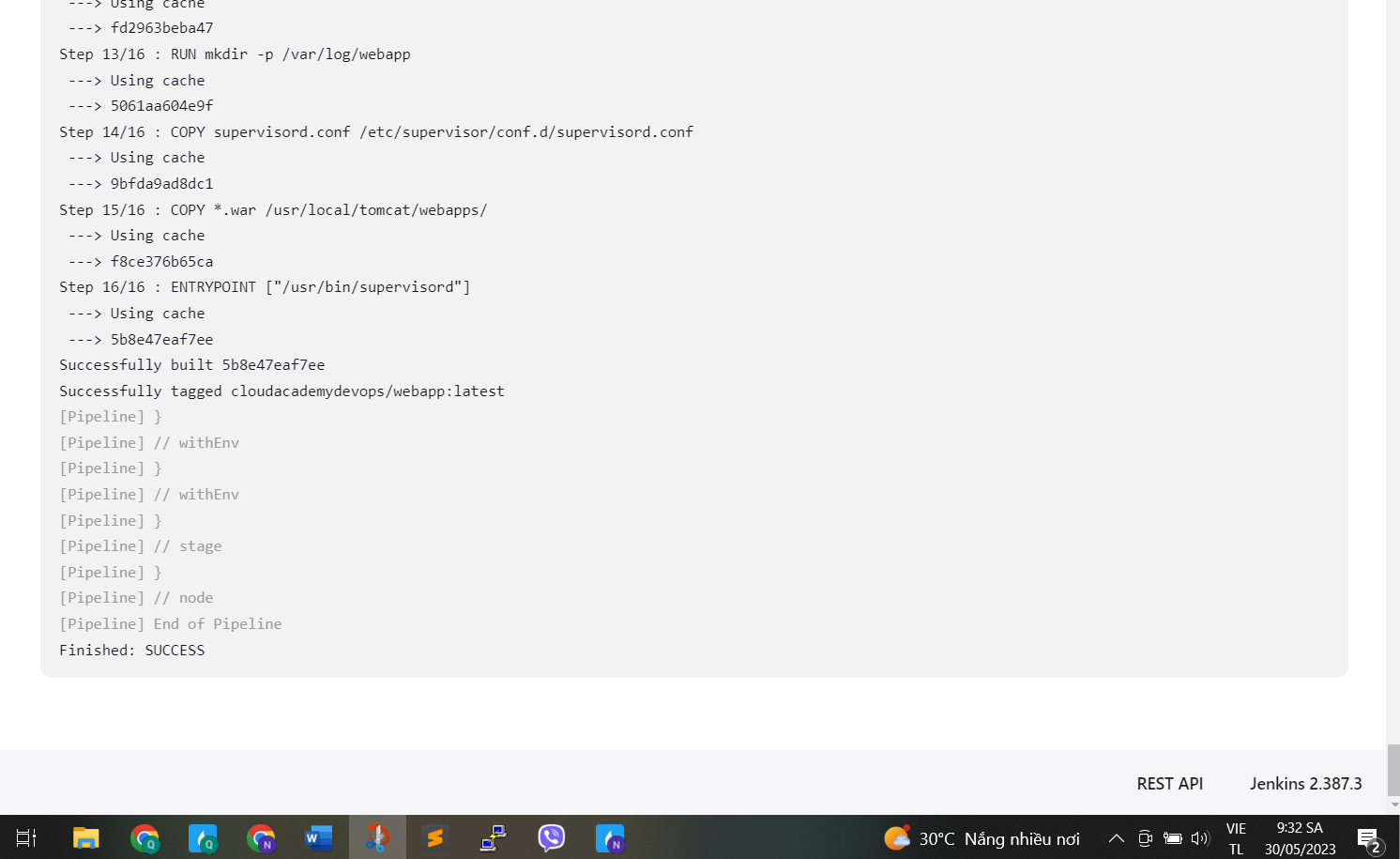
The Splunk docker container has been configured to listen for inbound connections on port **8000**. Using your browser, navigate to the Splunk home page:

In the **Change license group** section, choose the **Free license** option, and then click the **Save** button:

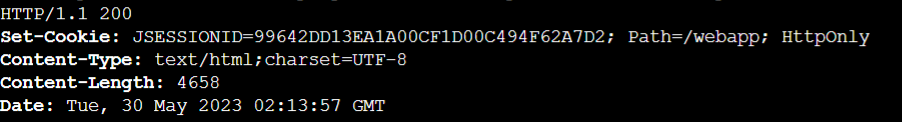
I will begin by installing the **Docker Pipeline Plugin**. To do so, navigate to the **Manage Jenkins** > **Manage Plugins** section like so:

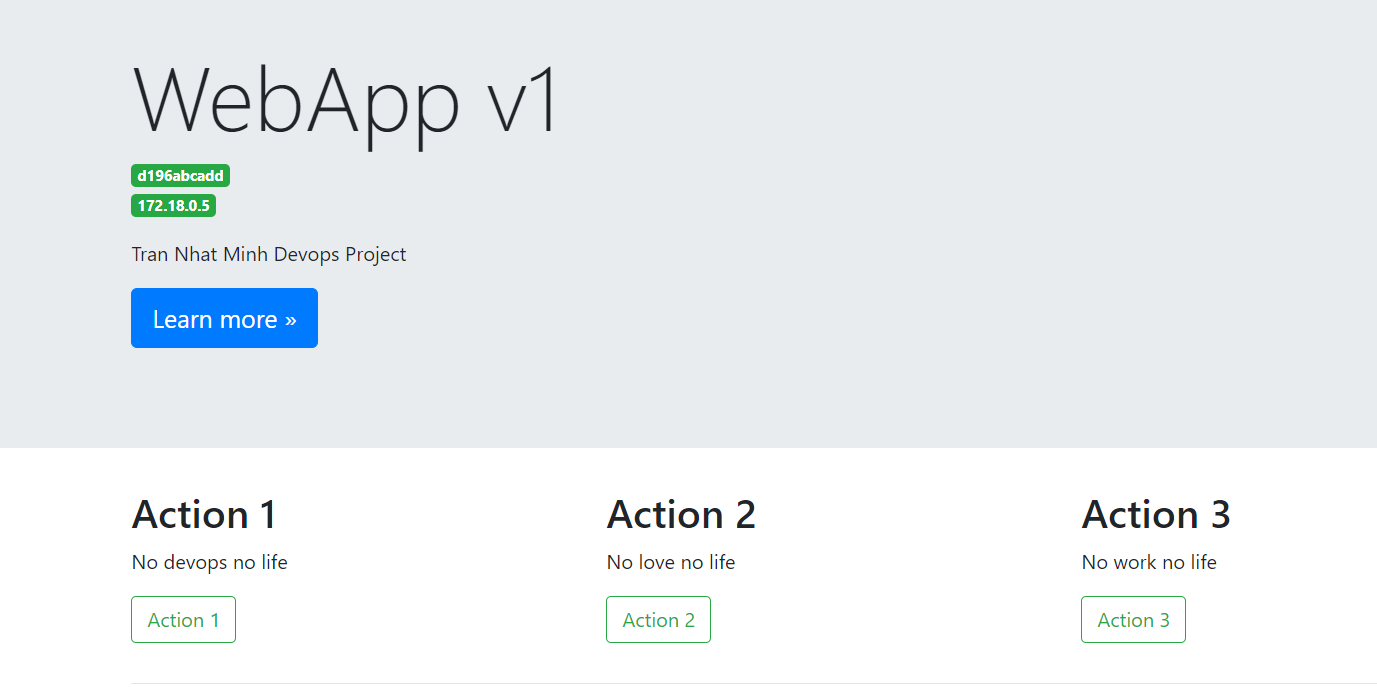
Select the **Available plugins** from the left navigation pane, and then type Docker Pipeline in the **Search**input. This will result in the **Docker Pipeline** plugin being displayed in the Install list. Enable the **Docker Pipeline** plugin and then click the **Install without restart** button:

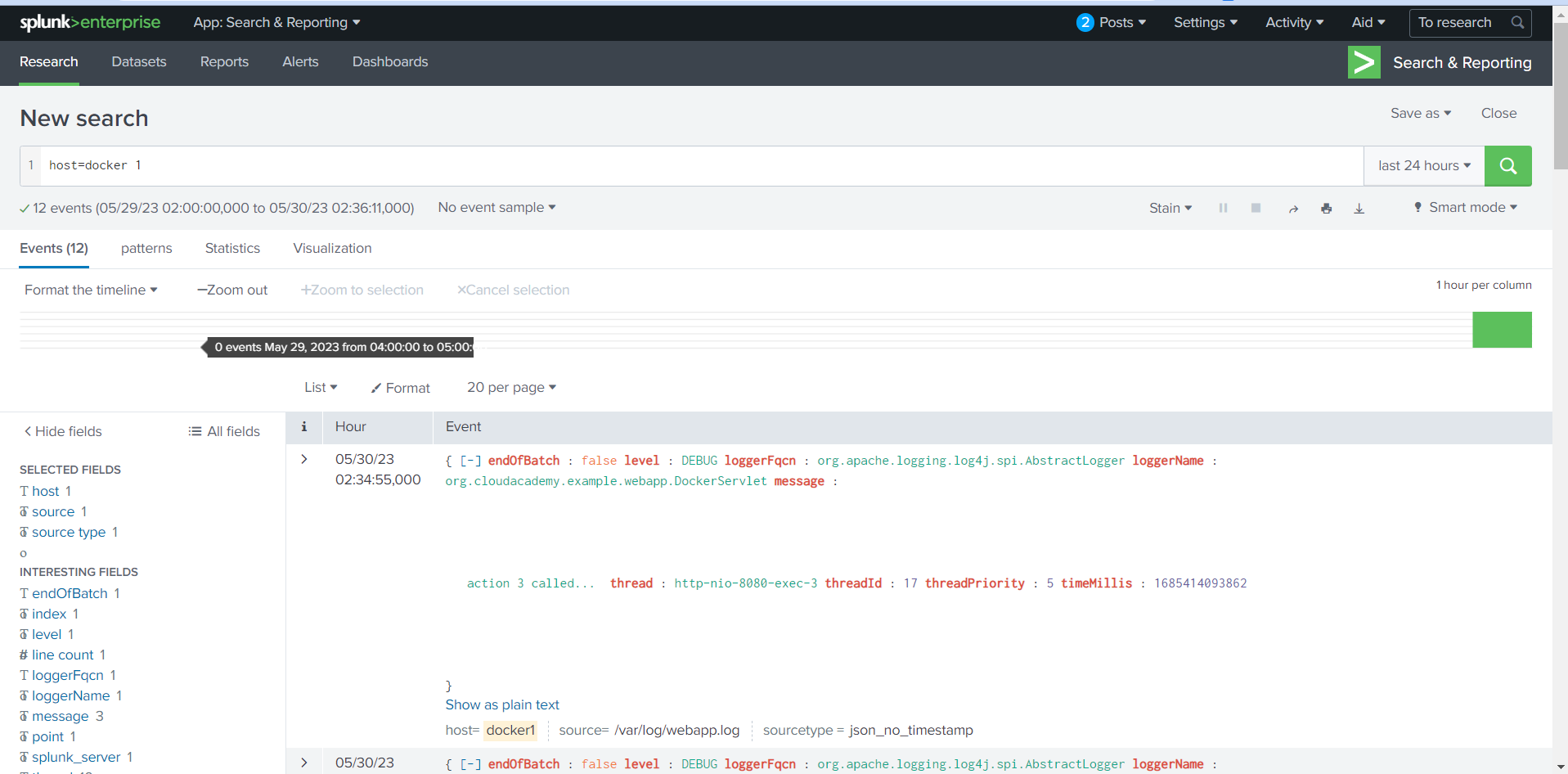
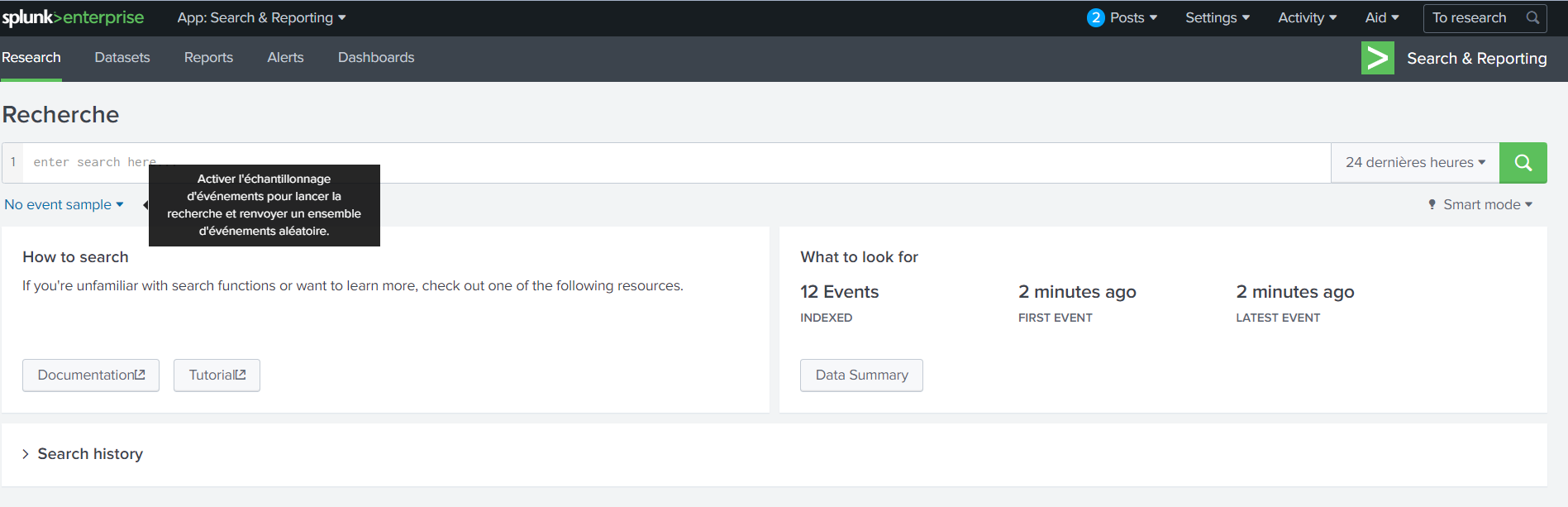
Staying on the same page, scroll down to the **Docker** section, and click **Add Docker**. Enter docker-latest in the **Name** field and then select the **Install automatically** checkbox. Click **Add Installer** and select **Download from docker.com**. Leave the **Docker version** field set to **latest**.



Next, run a simple curl command to test the webapp Tomcat endpoint. The webapp docker container has been configured to listen for inbound connections on port **9999**. Using the curl command, navigate to the webapp home page: http://PUBLIC\_IP\_CICD\_PLATFORM\_INSTANCE:9999/webapp/home. Remember to use the public IP address assigned to the cicd.platform.instance EC2 instance:





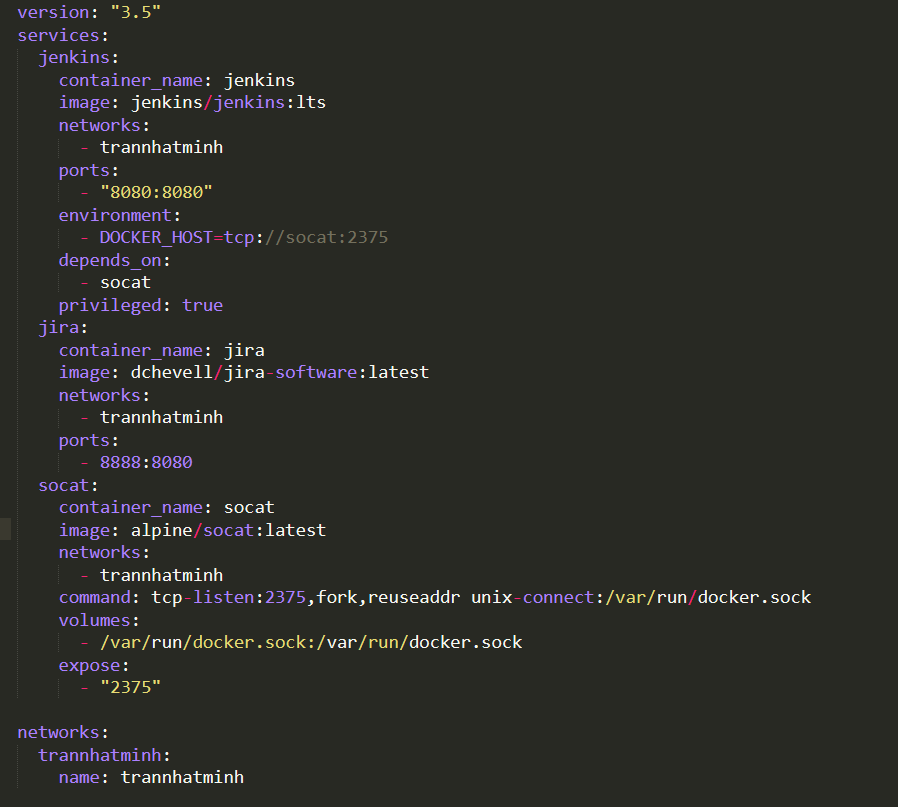


I used the Splunk administration web console to navigate back into the Search & Reporting area. I then confirmed that several data events had been collected, and that I were able to perform an analysis on them using the **Search & Reporting** features.

**Create a Jenkins CICD Pipeline to Publish Build Results into Jira**

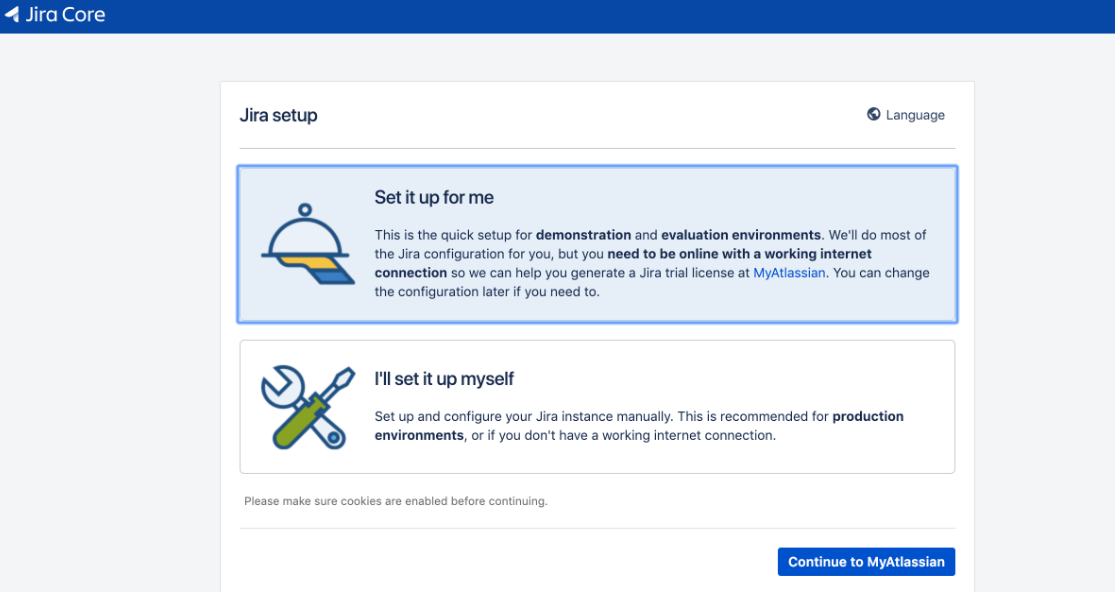
Integrating Jenkins and Jira together provides I with a solution that can be used to report issues as they happen for any CICD pipeline build job. Jenkins can be configured to publish any and all build results, artifacts, and/or bugs directly into Jira. DevOps teams can then use Jira to project manage the required fixes necessary to get their applications into production.

In this lab, I will launch a Jenkins and Jira CICD and issue management environment using Docker containers on a provided EC2 instance. I will then configure a Jenkins build pipeline to build, compile, and package a project, with the resulting build information being published automatically into Jira. I will then use Jira to observe the details about the Jenkins build just performed.



The Jira docker container has been configured to listen for inbound connections on port **8888**. Using your browser, navigate to the Jira home page:

On the **Create project with sample data** popup window, select **Kanban software development** and click **Next**:



Click the **Manage Jenkins** menu item taking I into the **Manage Jenkins** screen. Ignore any update and/or security messages that Jenkins may announce as this is a Project environment only:

Click the **Available** tab and enterJira Pipeline within the search input field associated with the **Plugins Manager** pane. Select the **Jira Pipeline Steps** option and click **Install without restart**:

After build Jenkins Pipeline, reloading the Kanban board, I should then see that a new issue has been automatically created and added to the Jira project. I will then manage the issue by moving it between states within the Kanban board. I will also open up the issue and download the attached Gradle build log.

