

<https://www.softwaretestinghelp.com/jenkins-tutorials/>

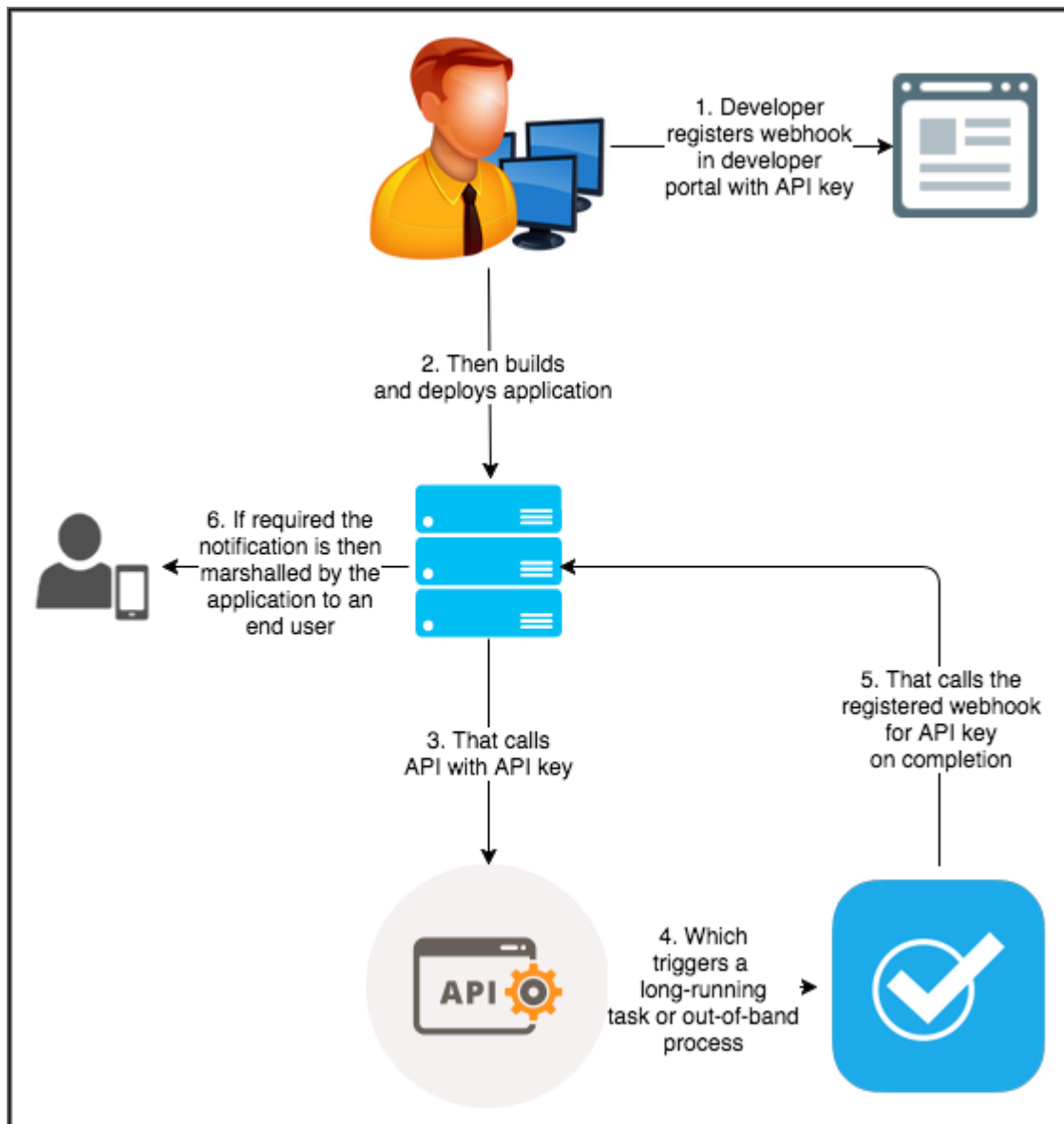
<https://www.jenkins.io/doc/book/security/managing-security/>

<https://www.softwaretestinghelp.com/jenkins-tutorials/>

<https://www.softwaretestinghelp.com/jenkins-job-tutorial/>

<https://learning.oreilly.com/library/view/learning-continuous-integration/9781788479356/1eefa49b-3abc-466e-8b47-1b50c384ba2f.xhtml>

Add a webhook in our pipeline.



First, let's see what is a webhook??

The concept of a WebHook is simple. A WebHook is an HTTP callback: an HTTP POST that occurs when something happens. GitHub webhooks in Jenkins is used to trigger our build whenever a developer commits something on the master branch.

Let's see how to add build a Webhook in GitHub and then add this webhook in Jenkins.

Steps:-

1. Go to the project repository.
2. Go to settings on the right corner.
3. Click on webhooks.
4. Click Add webhooks.
5. Write the Payload URL

<https://<Jenkins>Server>

Options	Webhooks / Manage webhook
Collaborators	We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in our developer documentation .
Branches	
Webhooks	Payload URL * <input type="text" value="https://228b9f82.ngrok.io/github-webhook/"/>
Integrations & services	Content type <input type="text" value="application/json"/>
Deploy keys	Secret <input type="text"/>
Moderation	SSL verification <p> By default, we verify SSL certificates when delivering payloads.</p> <p><input type="radio"/> Enable SSL verification <input checked="" type="radio"/> Disable (not recommended)</p>
Interaction limits	Which events would you like to trigger this webhook? <p><input type="radio"/> Just the push event.</p> <p><input checked="" type="radio"/> Send me everything.</p> <p><input type="radio"/> Let me select individual events.</p>

Here,

Payload URL:- Is the URL where our Jenkins is running.

Content type:- The kind of data we want in our webhook. JSON, XML, etc.

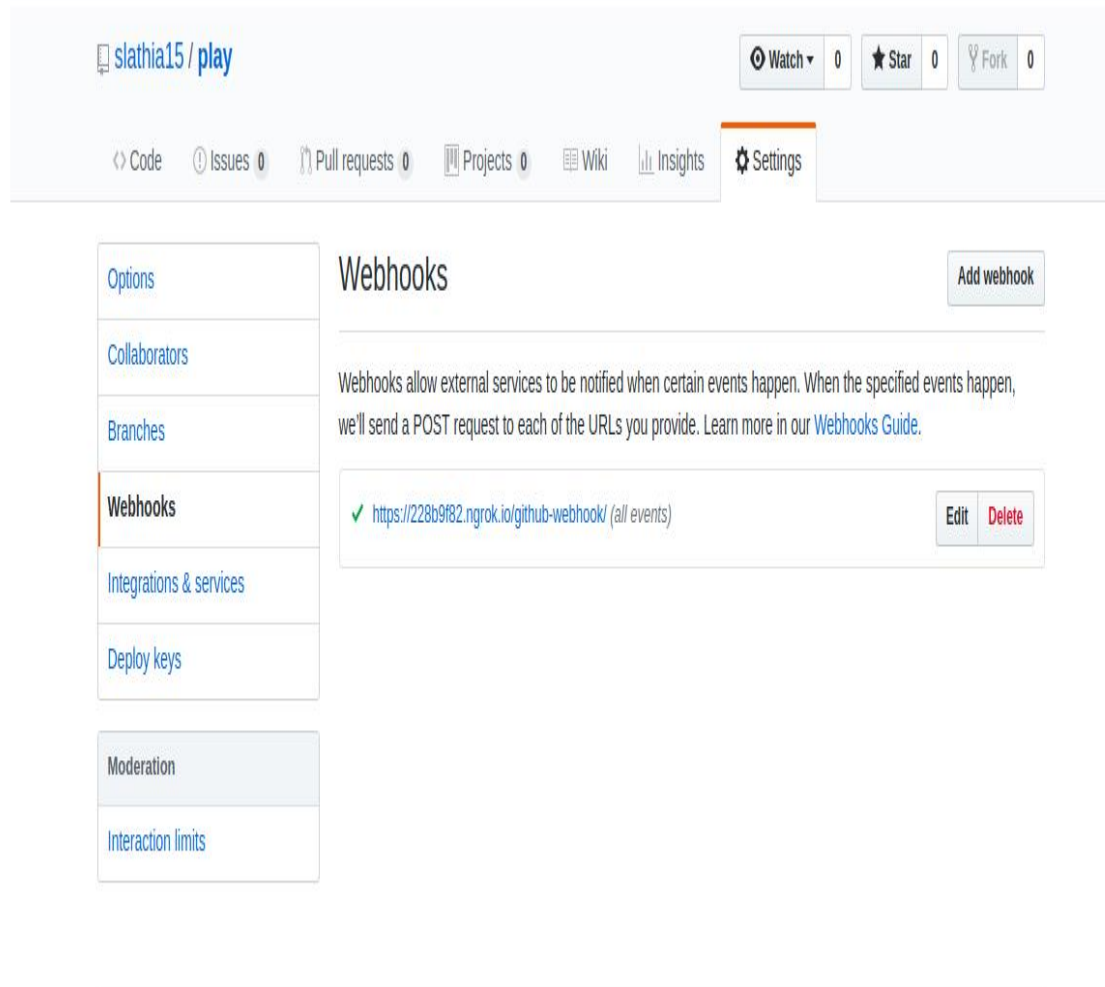
Secret:- Used to secure our webhook we can provide a secret in our webhook and ensure that only applications having this webhooks can use it.

SSL verification:- This SSL Checker will help us diagnose problems with SSL certificate installation. We can verify the SSL certificate on our web server to make sure it is correctly installed, valid, trusted and doesn't give any errors to any of our users.

When shall these webhooks be triggered?

1. *Just the push event:-* Only send data when someone pushed into my repository.
2. *Send me everything:-* If there is any pull or push event in our repository we will get notified.
3. *Let me select individual events:-* We can configure for what events we want our data.

Click Create and a webhook will be created.



But there is a problem, we have to take care of if we are running Jenkins on localhost then writing `https://localhost:8080/github-webhook/` will not work because webhooks can only work when they are exposed to the internet.

So, if we want to make our localhost to be exposed to the internet then we can use the **ngrok tool**. Refer to this [link](#).

Now let's see how to use this webhook into our Jenkins.

Steps:-

1. Go to Manage Jenkins -> **Configure System**
2. On scrolling down, we will find **GitHub Pull Requests** checkbox.
In **Published Jenkins URL**:- add the repository link
3. Click on Save.

Jenkins » configuration

☐ Enable Debug Mode

☐ Require Administrator for Template Testing

☐ Enable watching for jobs

☐ Allow sending to unregistered users

Content Token Reference

E-mail Notification

SMTP server

Default user e-mail suffix

Test configuration by sending test e-mail

GitHub Pull Requests

Published Jenkins URL

Actualise local repo on factory creation

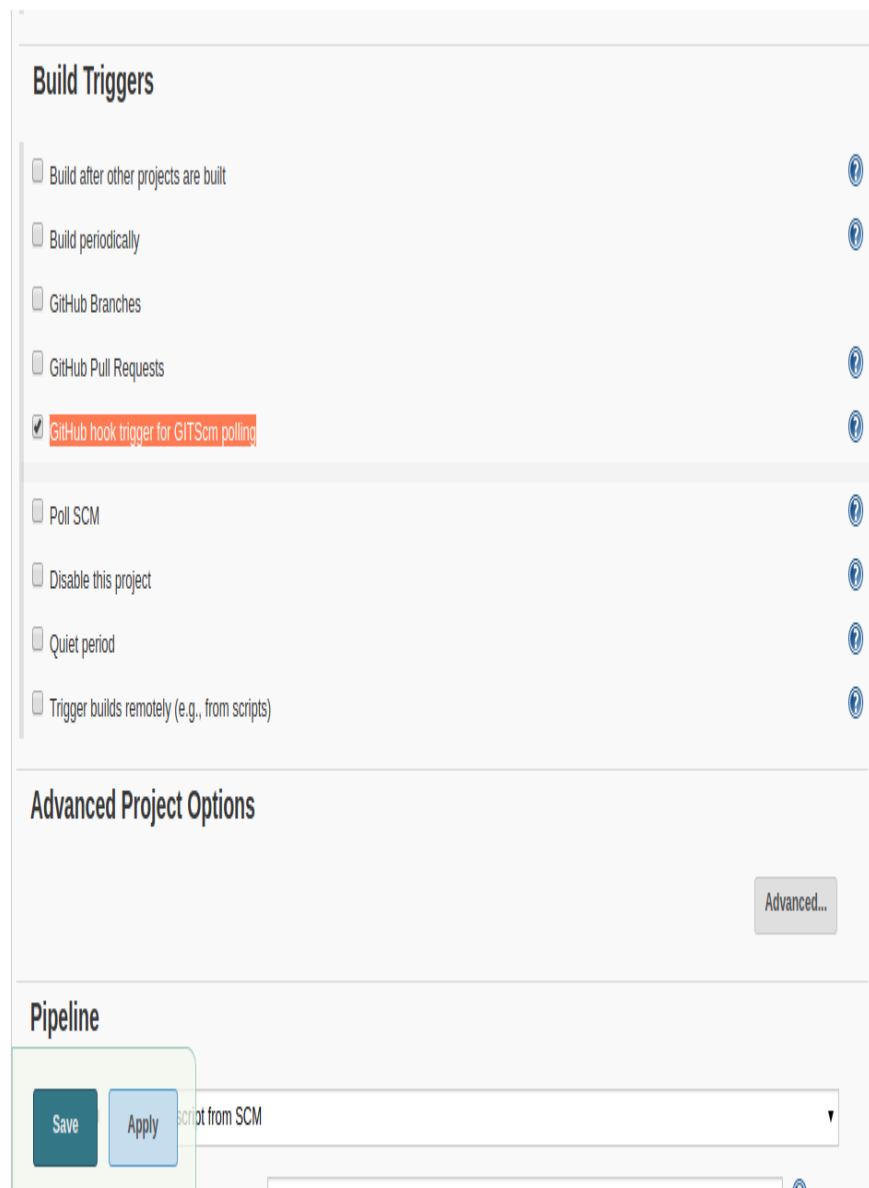
Save Apply

Default Triggers...

Advanced...

Page generated: 25-Oct-2018 14:32:11 IST [REST API](#) [Jenkins ver. 2.138.2](#)

Now go to Jenkins Pipeline and select *GitHub hook trigger for GITScm polling.*



In this way, we can add Webhook to our job and ensure that everytime a developer commits code to GitHub our build will get triggered.

Don't use input within an agent

While you can put an `input statement` within a stage that is within an agent, you definitely shouldn't.

Why? The input element pauses pipeline execution to wait for an approval - either automated or manual. Naturally these approvals could take some time. An agent, on the other hand, acquires and holds a lock on a workspace and heavy weight Jenkins executor - an expensive resource to hold onto while pausing for input. So, create your inputs outside of your agents.

```
pipeline {
  agent none
  stages {
    stage('Example Build') {
      agent {
        label "linux"
      }
      steps {
        sh 'echo Hello World'
      }
    }
    stage('Ready to Deploy') {
      steps {
        input(message: "Deploy to production?")
      }
    }
  }
}
```

```

    }
  }
  stage('Example Deploy') {
    agent {
      label "linux"
    }
    steps {
      sh 'echo Deploying'
    }
  }
}

```

7. Wrap your input in a timeout

Pipeline has an easy mechanism for timing out any given step of your pipeline. As a best practice, you should always plan for timeouts around your inputs.

Why? For healthy cleanup of the pipeline, that's why. Wrapping your inputs in a timeout will allow them to be cleaned-up (i.e., aborted) if approvals don't occur within a given window.

```

pipeline {
  agent none
}

```

```
stages {  
  stage('Example Build') {  
    agent {  
      label "linux"  
    }  
    steps {  
      sh 'echo Hello World'  
    }  
  }  
  stage('Ready to Deploy') {  
    options {  
      timeout(time: 1, unit: 'MINUTES')  
    }  
    steps {  
      input(message: "Deploy to production?")  
    }  
  }  
  stage('Example Deploy') {  
    agent {  
      label "linux"  
    }  
    steps {  
      sh 'echo Deploying'  
    }  
  }  
}
```

```
}  
  
}  
  
}
```

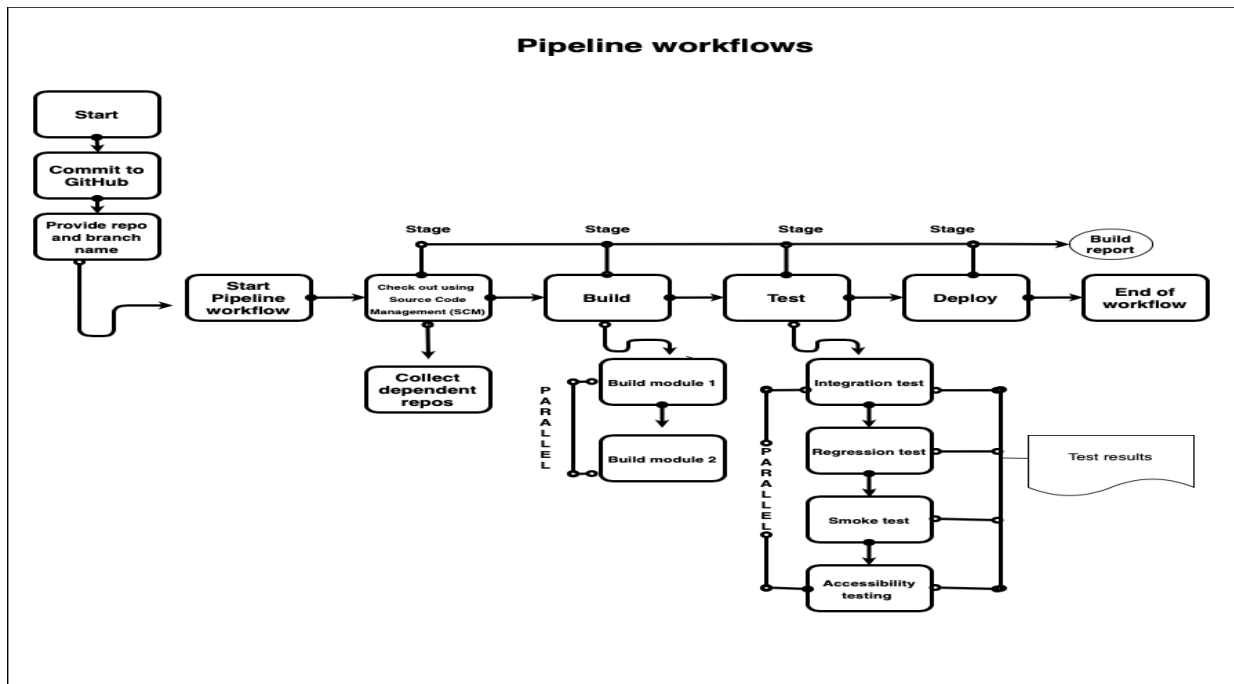
PIPELINE

Pipelines are sets of processes that are used to compile, build, and deploy code into a production environment. With Pipelines you can:

- Manage automated builds, tests, and deployments as one workflow.
- Deliver quality products frequently and consistently from testing through staging to production automatically.
- Automatically promote or prevent build artifacts from being deployed. If errors are discovered anytime during the process the Pipeline stops and alerts are sent to the appropriate team for their attention.

To ensure that the quality of the software being developed is not compromised for speed, appropriate checks and balances are built into Pipeline processes. They can be programmed to trigger with commits to the code thereby enabling frequent releases and consistent behavior.

Pipeline example



A Jenkins Pipeline defines the tasks required to build, test, and deploy your software. It is a suite of plugins that supports implementing and integrating Pipelines into Jenkins.

Understanding the key concepts in Pipelines helps you deliver improved code efficiently and reliably.

Declarative Pipeline

Declarative Pipelines provide a structured hierarchical syntax to simplify the creation of Pipelines and the associated Jenkinsfiles. In its simplest form, a Pipeline runs on an agent and contains stages, while each stage contains steps that define specific actions. Here is an example:

```

pipeline {
  agent {
    label '...'
  }
}

```

```

    stages {
        stage('Build') {
            steps {
                sh 'mvn install'
            }
        }
    }
}

```

agent

stages

steps

agent

An **agent** section specifies where tasks in a Pipeline run. An **agent** must be defined at the top level inside the **pipeline** block to define the default agent for all stages in the Pipeline. An **agent** section may optionally be specified at the stage level, overriding the default for this stage and any child stages. In this example, the agent is specified with an empty **label** filter, meaning this Pipeline will run on any available agent.

The code snippet above does not show an agent running on Kubernetes. An agent running on Kubernetes is configured differently.

stages

A **stage** represents a logical grouping of tasks in the Pipeline.

Each **stage** may contain a **steps** section with steps to be executed, or stages to be executed sequentially, in parallel, or expanded into a parallel matrix.

It contains commands that run processes like Build, Deploy, or Tests. These commands are included in the **steps** section in the stage.

It is advisable to have descriptive names for a **stage** as these names are displayed in the UI and logs.

There can be one or more **stage** sections inside a **stages** section. At least one **stage** must be defined in the **stages** section.

steps

The `steps` section contains a set of commands. Each command performs a specific action and is executed one-by-one.

```
pipeline {  
  agent any  
  stages {  
    stage('Example') {  
      steps {  
        sh 'mvn compile'  
      }  
    }  
  }  
}
```


For more information on Declarative Pipelines, see [Using Declarative Pipeline syntax](#).

Scripted Pipeline

Scripted Pipeline is a more general form of Pipeline as Code. It is a Domain Specific Language (DSL) based on Apache Groovy. Most functionality provided by the Groovy language is made available in Scripted Pipelines. This means Scripted Pipelines can offer flexibility and extensibility to users. However, the learning curve associated with Scripted Pipelines is very steep and it is possible to do things inside Pipelines that are better done in other tools called as part of Pipelines. CloudBees recommends the use of Declarative Pipelines rather than Scripted Pipelines.

For more information on Scripted Pipelines, see [Using Scripted Pipeline syntax](#).

Freestyle projects

Freestyle projects are used to implement, develop, or run simple jobs. They can span multiple operations like building and running scripts.

CloudBees recommends the use of Pipeline projects instead of Freestyle projects. For information about converting a Freestyle project to a Declarative Pipeline please see [Converting a Freestyle project to a](#)

[Declarative Pipeline.](#)

Multibranch Pipelines

A multibranch Pipeline can be used to automatically create Pipelines based on branches and pull requests in your repository. For more information on multi-branch Pipelines, see [Managing Multibranch Pipeline options in template.yaml](#).

Pipeline as Code

Pipeline as Code provides a consistent way to build and deploy services and applications by treating them as code. It is used as a tool to execute underlying tasks for various stages of a software delivery lifecycle and takes away the implementation details.

It is a set of features that allow pipelined job processes to be defined with code, stored, and versioned in a source repository. These features allow the discovery, management, and running of jobs for multiple source repositories and branches, eliminating the need for manual job creation and allowing for change management at job level without using the UI.

Both types of Pipelines as described in this topic are Pipeline as Code. Before Pipelines, jobs were configured using the UI, and were part of the `config.xml` files rather than the code.

Benefits of Pipeline as Code

- Pipeline as Code allows integration of your Pipeline with your code thereby making collaboration on it easier. When you update your Pipeline, your changes are automatically picked up.
- Storing Pipelines in a version control system enables tracking of changes to the Pipeline. If a change to the Pipeline causes a build to break, then it can be fixed before it is merged or can easily be reverted. It can also be restored more easily.

Pipeline development utilities

The following utilities are available to help you develop your Pipelines:

- **Command-line Pipeline Linter (jenkins CLI):** Command-line Pipeline Linter is a built-in command line interface that allows users and administrators to access Jenkins from a script or shell environment. This can be convenient for scripting of routine tasks, bulk updates, and troubleshooting.
- "Replay" Pipeline runs with modifications:** Use this tool to develop scripts incrementally. You can use the "Replay" tool to prepare and test your changes before committing them to Pipeline scripts.

For more information on any of these tools, please see

[Using Pipeline development tools.](#)

Pipeline terminology

controller: A controller is a computer, VM, or container where Jenkins is installed and run. It is used to serve requests and handle build tasks.

- **agent:** An agent is typically a machine, or container, which connects to a Jenkins controller and executes tasks when directed by the controller.
- **node:** A node is an operating system or container running Jenkins as an agent.
- **executor:** An executor is a computational resource for running builds and performing operations. It can run on any controller or agent. An executor can be parallelized on a specific controller or agent.

Using executors on a controller should be reserved to very specific cases as there are security and performance implications to doing so.

CloudBees proprietary features for Pipelines

CloudBees provides some additional features that enhance the functionality available in Pipelines.

- Features that help in remote collaboration.
- Features that help in bringing about consistency across Pipelines.

For details, please see the sections below.

Features for remote collaboration

CloudBees Pipelines have access to features that can be used to improve collaboration between different teams using different Pipelines.

Cross Team Collaboration

CloudBees' Cross Team Collaboration provides the ability to publish an event from a Jenkins job that triggers any other Jenkins job on the same controller or different controllers that are listening for that event. For more information, see [Cross Team Collaboration](#).

Cluster-wide job triggers

Cluster-wide job triggers provide the ability to trigger jobs across client controllers attached to the same CloudBees Operations Center. Depending on the scheme used to organize an Operations Center cluster, it may be necessary to trigger jobs that are on a remote client controller. For more information, see [Cluster-wide job triggers](#).

External HTTP endpoints

External HTTP endpoints work with Cross Team Collaboration to enable external systems such as GitHub or Nexus to generate notification events for Pipelines. For more information, see [External HTTP endpoints](#).

Features for Pipeline standardization

CloudBees Pipelines have access to features that can be used to bring consistency across Pipelines across your organization.

Pipeline Templates

Pipeline Templates can be used to define reusable Pipelines. This helps in bringing about consistency across your Pipelines. Pipeline

Templates are specific to CloudBees CI. For more information on Pipeline Templates, see [Pipeline Templates](#).

Pipeline Policies

Pipeline Policies are runtime validations that work for both Scripted and Declarative Pipelines and provide administrators a way to include warnings for or block the execution of Pipelines that do not comply with certain regulatory requirements, rules, or best practice guidelines. For more information on concepts related to Pipeline Policies, see [Using Pipeline Policies](#).

Organization folders

Organization folders are a way to automatically create Multibranch projects for every repository in a GitHub organization or Bitbucket team. For more information, please see [Organization folders](#).

How to move jenkins job from one folder to another

Goto manage Jenkins > manage plugins > search for cloudbees folder plugin > install without restart. this will add an option called move that will move your jobs from one folder to another.

HOW TO MOVE JENKINS JOB FROM ONE SERVES TO ANOTHER

Goto manage Jenkins > manageplugins> search for cloudbees folder plugin > install without restart. this will add an option called move that will move your jobs from one folder to another.

ANOTHER SOLUTION

1. Copy all the files in your JENKINS_HOME directory over to the new server.

2. Point JENKINS_HOME on the new server at the new directory.

3. Copy the Jenkins war file (or your servlet container setup if you have one) over to the new machine and start it up.

All Jenkins settings, jobs, plugins, config, etc. live in JENKINS_HOME.

You just need a copy of it to start it elsewhere.

Throttle Concurrent Builds Plugin

-
-
-

Plugin Information

View Throttle Concurrent Builds [on the plugin site](#) for more information.

This **z**

To set an unlimited value of concurrent builds for the project or node, use "0".

Per Single-Job

On the job-page, you select **Throttle Concurrent Builds** and then specify **Maximum Total** and/or **Maximum Per Node**:

☒ **Throttle Concurrent Builds**

Maximum Total Concurrent Builds	<input type="text" value="1"/>
Maximum Concurrent Builds Per Node	<input type="text" value="1"/>

These settings only **apply to this specific Job** and do not apply to any categories you may additionally select.

It should be noted that Jenkins, **by default**, **never executes the same Job in parallel**, so you do not need to actually throttle anything if you go with the default. However, there is the option **Execute concurrent builds if necessary (beta)**, which allows for running the same Job multiple time in parallel, and of course if you use the categories below, you will also be able to restrict multiple Jobs.)

Per Project-Category

When using **Multi-Project Throttle Categories** the maximum values are chosen in the **global configuration** where you define the category:

Throttle Concurrent Builds	
Category Name	<input type="text" value="category-x-one-per-node"/>
Maximum Total Concurrent Builds	<input type="text" value="0"/>
Multi-Project Throttle Categories	Maximum Concurrent Builds Per Node <input type="text" value="1"/>

Throttling of Pipeline jobs

`throttle()` step

Starting in `throttle-concurrents-2.0`, this plugin allows throttling of particular Pipeline blocks by categories. For this purpose you can use the `throttle()` step.

How does it work?

- If a `throttle()` step is used, all explicit and implicit `node()` invocations within this step are throttled.
- If a `node()` step is used within a `parallel()` block, each parallel branch is throttled separately.
- Throttling of Pipeline steps in `throttle()` takes precedence over other throttling logic, such as job properties in Pipeline and other job types.
- If the specified category is missing, `throttle()` execution fails the run.

Warning regarding restarting the Jenkins controller

Due to a deadlock (as described in [JENKINS-44747](#)), a change has been made which can theoretically result in throttle categories being ignored in running Pipelines immediately after the Jenkins controller has been restarted. This will be investigated further in [JENKINS-44756](#) but was considered necessary in order to resolve the deadlock scenario.

Examples

Example 1: Throttling of `node()` runs

```
// Throttle a single operation
```

```
throttle(['test_2']) {  
  node() {  
    sh "sleep 500"  
    echo "Done"  
  }  
}
```

Example 2: Throttling of parallel steps

```
// The script below triggers 6 subtasks in parallel.
```

```
// Then tasks are throttled according to the category settings.
```

```
def labels = ['1', '2', '3', '4', '5', '6']
```

```
def builders = [:]
```

```
for (x in labels) {
```

```
  def label = x // Need to bind the label variable before the closure
```

```
  // Create a map to pass in to the 'parallel' step so we can fire all the  
  builds at once
```

```
    builders[label] = {
```

```
      node('linux') {
```

```
        sh "sleep 5"
```

```
      }  
    }
```

```
  }
```

```
}
```

```
throttle(['myThrottleCategory1', 'myThrottleCategory2']) {  
  parallel builders  
}
```

Example 3: Throttling of declarative pipelines

To throttle concurrent builds to 1, configure a global category and add an options property to the pipeline.

Throttle Concurrent Builds

Multi-Project Throttle Categories

Category Name



Maximum Total Concurrent Builds

Maximum Concurrent Builds Per Node

Add Maximum Per Labeled Node

Delete

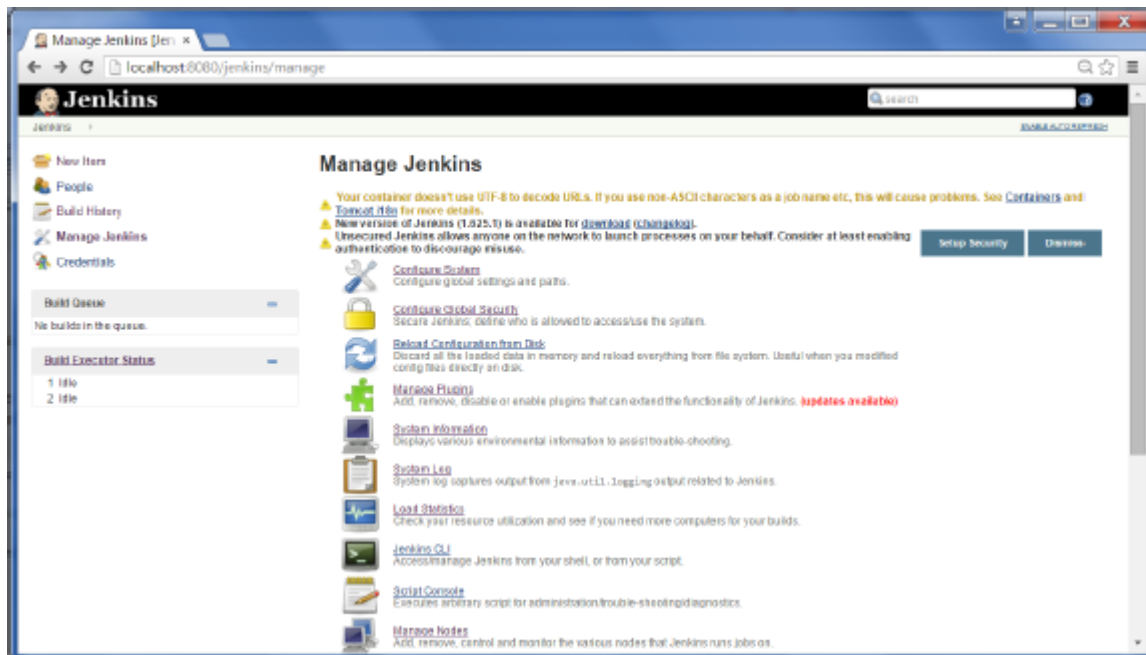
Add Category

```
pipeline {  
  agent any  
  
  // Throttle a declarative pipeline via options  
  options {  
    throttleJobProperty(  
      categories: ['test_3'],
```

```
    throttleEnabled: true,  
    throttleOption: 'category'  
  )  
}  
  
stages {  
  stage('sleep') {  
    steps {  
      sh "sleep 500"  
      echo "Done"  
    }  
  }  
}  
}
```

To configure Security in Jenkins, follow the steps given below.

Step 1 – Click on Manage Jenkins and choose the 'Configure Global Security' option.



Step 2 – Click on **Enable Security option**. As an example, let's assume that we want Jenkins to maintain its own database of users, so in the Security Realm, choose the option of **'Jenkins' own user database**.

By default you would want a central administrator to define users in the system, hence ensure the **'Allow users to sign up'** option is unselected. You can leave the rest as it is for now and click the Save button.



Step 3 – You will be prompted to add your first user. As an example, we are setting up an admin users for the system.

Sign up [Jenkins]

localhost:8080/jenkins/securityRealm/firstUser

Jenkins

Jenkins' own user database

Back to Dashboard

Manage Jenkins

Create User

Sign up

Username: admin

Password: *****

Confirm password: *****

Full name: Administrator

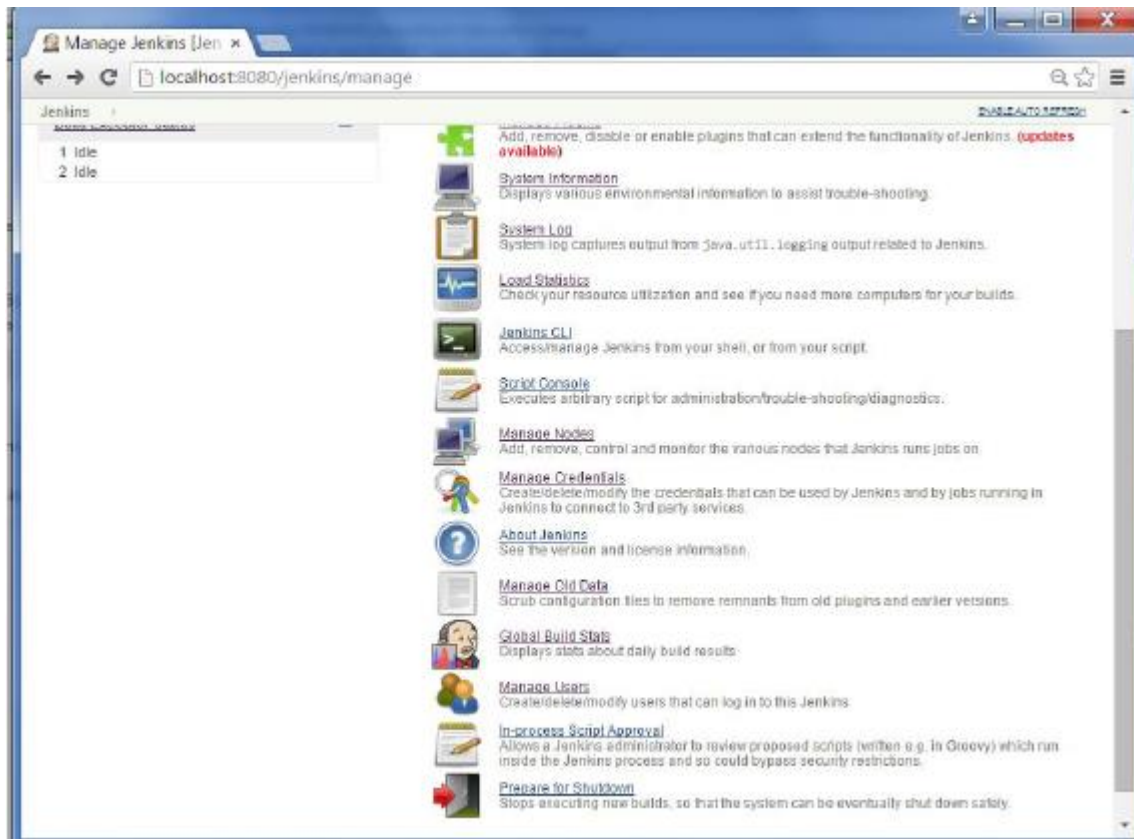
E-mail address: al@gmail.com

Sign up

Help us localize this page

Page generated: Oct 24, 2015 6:16:01 PM REST API Jenkins ver. 1.608.3

Step 4 – It's now time to setup your users in the system. Now when you go to Manage Jenkins, and scroll down, you will see a 'Manage Users' option. Click this option.



Step 5 – Just like you defined your admin user, start creating other users for the system. As an example, we are just creating another user called 'user'.

Step 7 – If you don't see the user in the user group list, enter the user name and add it to the list. Then give the appropriate permissions to the user.

Click on the **Save button once you have defined the relevant authorizations.**

Your Jenkins security is now setup.

Branch:

Job:

Pipeline:

Python:

Loops:

Decision making statements

Node:

Ansible:

Docker:

Kubernetes:

Groovy script:

Aws:

Enabling Security In Jenkins

Jenkins server supports several security models.

For smaller organizations, it may not be that important with close proximity within the developers. But still, security is required to protect the access of Jenkins for the outsiders.

The security for larger organizations becomes even stricter as there will be multiple teams and access needs to be given to developer teams and system admins.

The following are the simple steps to enable or activate security in Jenkins:

#1) Log in to Jenkins

#2) Click on Manage Jenkins and Configure Global Security in Jenkins dashboard as shown in Figure 1.

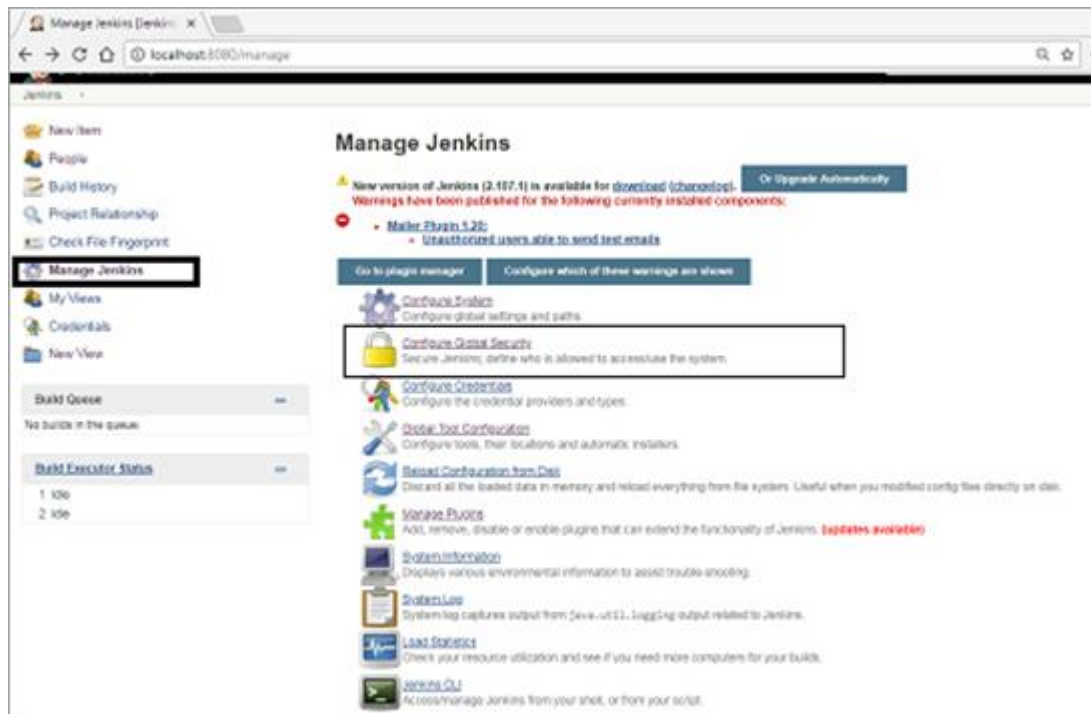


Figure 1: Jenkins Dashboard showing Manage Jenkins and Configure Global Security

#3) Check the **Enable security option**, check use **"Jenkins own user database"** under security realm or authentication, and Authorization check **"Logged in user can do anything"**. Also, check the **"Allow users signup"**. This security form is the simplest one and beneficial for smaller teams. Refer to the below Figure 2 for an understanding of security settings.

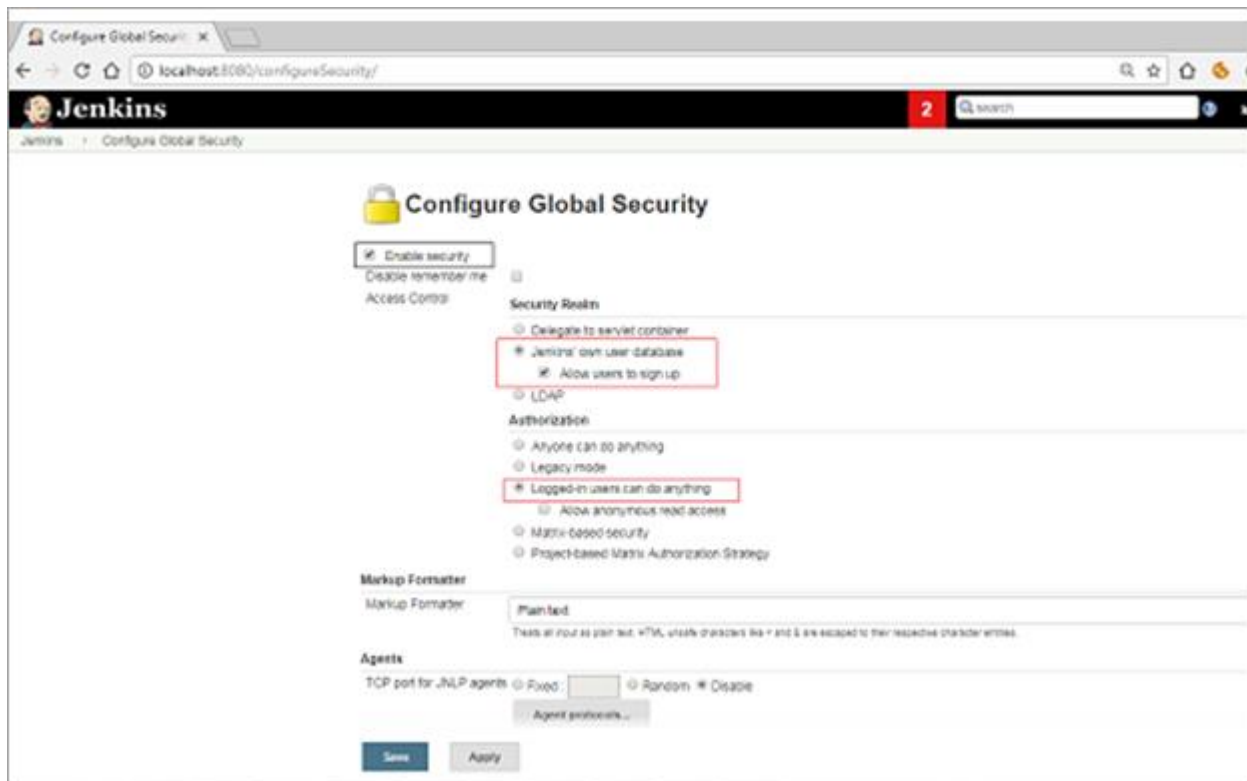


Figure 2: Settings in Configure global security

Authentication Or Security Realm

This lets us identify and manage users on Jenkins. There are many ways we can do this. The simplest way is using Jenkins's local database. This lets us set up the authentication for smaller organizations.

Others are being:

1. Jenkins own user database
2. Delegate to the servlet container
3. LDAP

Jenkins own user database: Here, the users can sign up using the signup link. All these users can be authenticated against the local database when

logged in. Based on security, we can gauge what the users can do. To find the users click on People link as shown in Figure 3 below.

Figure 3: Users list in Jenkins using people link

To find the build details of the users click on the User and then click on Builds as shown in Figure 4.

Build	Time Since	Status
JenkinsJobProject #1	11 days	stable
JenkinsJobProject #2	11 days	stable
JenkinsJobProject #3	11 days	stable
JenkinsJobProject #4	11 days	stable
JenkinsJobProject #5	12 days	stable
JenkinsJobProject #6	12 days	stable
JenkinsJobProject #7	13 days	stable
JenkinsJobProject #8	14 days	back to normal
JenkinsJobProject #9	14 days	broken since build #11
JenkinsJobProject #10	14 days	broken since this build
JenkinsJobProject #11	17 days	back to normal
JenkinsJobProject #12	17 days	broken for a long time
JenkinsJobProject #13	17 days	broken for a long time
JenkinsJobProject #14	17 days	broken for a long time

Figure 4: To find the builds triggered by people.

To configure the password, email details of the users, click on configure as shown in Figure 5.

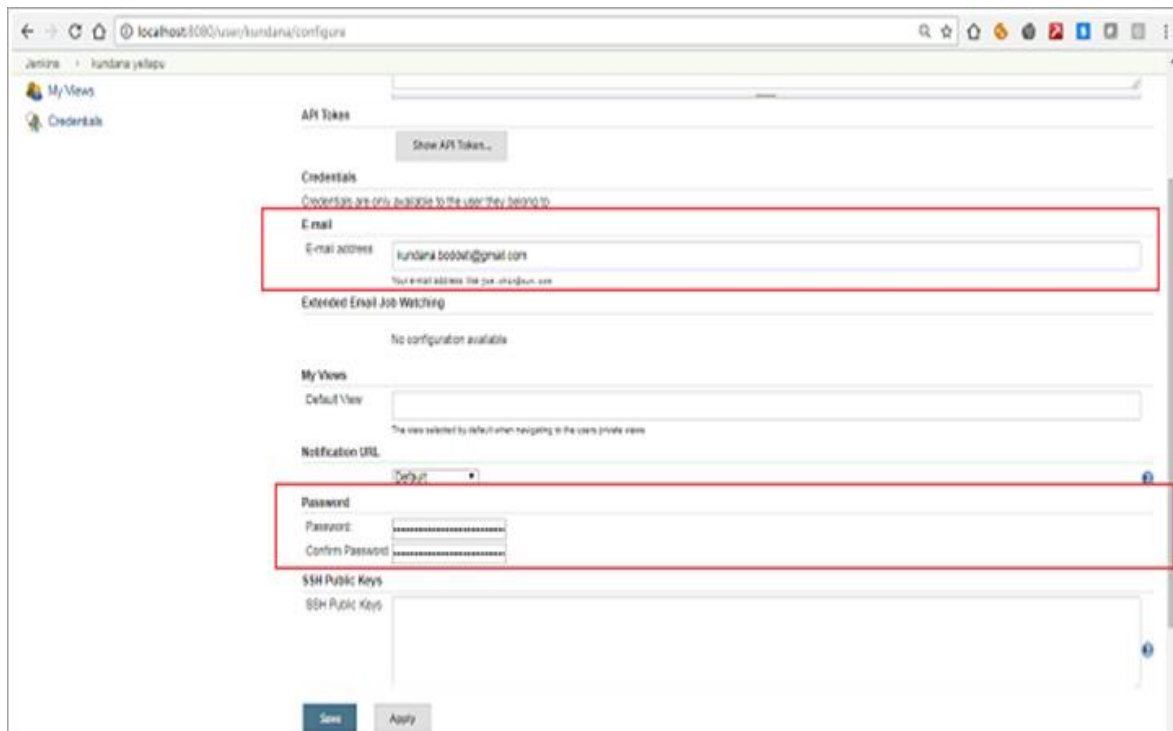


Figure 5: To set the Password and Email in configuring

Authorization

Once users are authenticated, it's time to grant the privilege to them. This process is called Authorization. There are many ways of authorization. The simplest being that the logged-in user can do anything. Other complicated ones are project-based authorization.

Various ways of authorization include:


- Anyone can do anything
- Legacy mode
- Logged-in users can do anything
- Matrix-based security
- Project-based Matrix authorization strategy

Enabling Security

Beginning with Jenkins 2.214 and Jenkins LTS 2.222.1, the "Enable Security" checkbox has been removed. Jenkins own user database is used as the default security realm.


In versions before Jenkins 2.214 and Jenkins LTS 2.222.1, when the Enable Security checkbox is checked, users can log in with a username and password in order to perform operations not available to anonymous users. Which operations require users to log in depends on the chosen authorization strategy and its configuration; by default anonymous users have no permissions, and logged in users have full control. The "Enable Security" checkbox should always be enabled for any non-local (test) Jenkins environment.

The "Configure Global Security" section of the web UI allows a Jenkins administrator to enable, configure, or disable key security features which apply to the entire Jenkins environment.

 Jenkins

search ?

Dashboard > Configure Global Security



Configure Global Security

Authentication

☐ Disable remember me

Security Realm

☐ Delegate to servlet container ?

☒ Jenkins' own user database ?

☐ Allow users to sign up ?

☐ None

Authorization

☐ Anyone can do anything ?

☐ Legacy mode ?

☒ Logged-in users can do anything ?

☐ Allow anonymous read access ?

Markup Formatter

Markup Formatter

Plain text

Treats all input as plain text. HTML unsafe characters like < and & are escaped to their respective character entities.

Save

Apply

TCP Port

Jenkins can use a TCP port to communicate with inbound (formerly known as "JNLP" Java Network Launch Protocol) files are used to launch applications from a hosted web server on a remote desktop client.

Software such as Java Plug-in and Java Web Start programs use JNLP files to run.) agents, such as Windows-based agents. As of Jenkins 2.0, by default this port is disabled.

For administrators wishing to use inbound TCP agents, the two port options are:

1. Random: The TCP port is chosen at random to avoid collisions on the Jenkins master. The downside to randomized ports is that they are chosen during the boot of the Jenkins controller, making it difficult to manage firewall rules allowing TCP traffic.
2. Fixed: The port is chosen by the Jenkins administrator and is consistent across reboots of the Jenkins controller. This makes it easier to manage firewall rules allowing TCP-based agents to connect to the controller.

As of Jenkins 2.217, inbound agents may instead be configured to use WebSocket transport to connect to Jenkins. In this case no extra TCP port need be enabled and no special security configuration is needed.

Access Control

Access Control is the primary mechanism for securing a Jenkins environment against unauthorized usage. Two facets of configuration are necessary for configuring Access Control in Jenkins:

1. A Security Realm which informs the Jenkins environment how and where to pull user (or identity) information from. Also commonly known as "authentication."
2. Authorization configuration which informs the Jenkins environment as to which users and/or groups can access which aspects of Jenkins, and to what extent.

Using both the Security Realm and Authorization configurations it is possible to configure very relaxed or very rigid authentication and authorization schemes in Jenkins.

Additionally, some plugins such as the [Role-based Authorization Strategy](#) plugin can extend the Access Control capabilities of Jenkins to support even more nuanced authentication and authorization schemes.

Security Realm

By default Jenkins includes support for a few different Security Realms:

Delegate to servlet container

For delegating authentication a servlet container running the Jenkins controller, such as [Jetty](#). If using this option, please consult the servlet container's authentication documentation.

Jenkins' own user database

Use Jenkins's own built-in user data store for authentication instead of delegating to an external system. This is enabled by default with new Jenkins 2.0 or later installations and is suitable for smaller environments.

LDAP

Delegate all authentication to a configured LDAP server, including both users and groups. This option is more common for larger installations in organizations which already have configured an external identity provider such as LDAP. This also supports Active Directory installations.

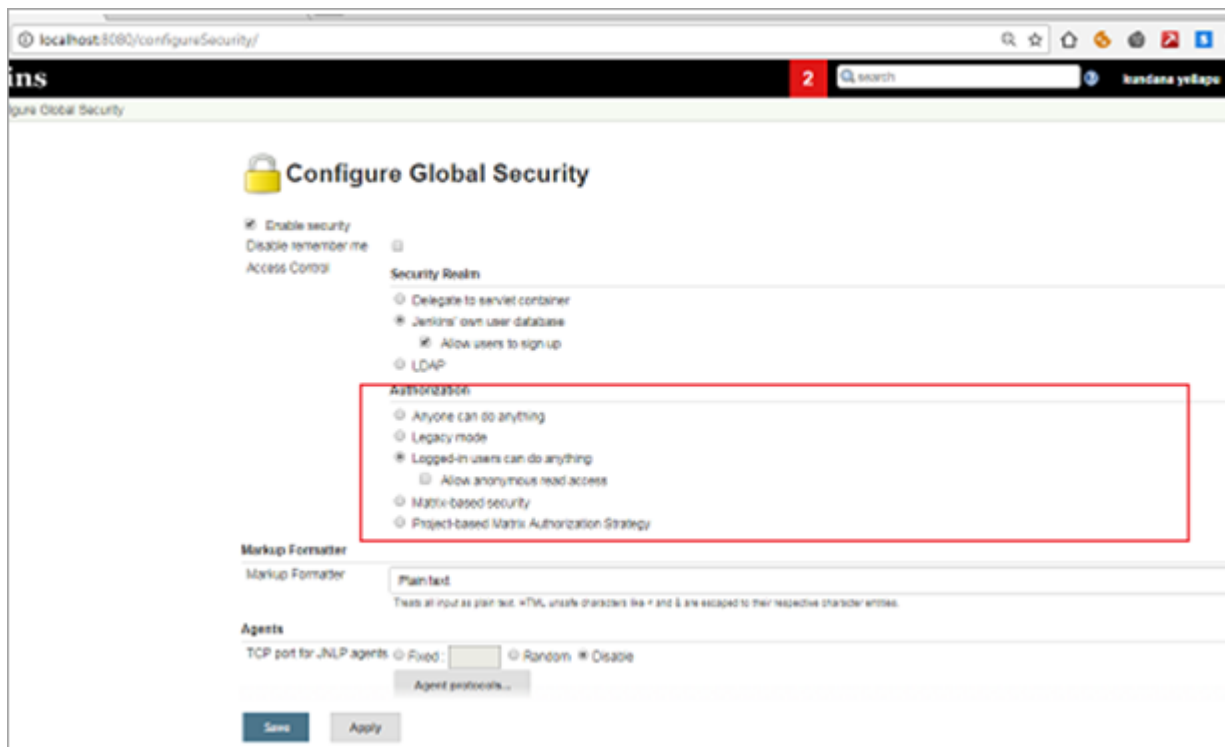


Figure 6: Various kinds of Authorization Roles

Creating An Admin User And Giving Privileges

To create an admin first click on the Signup link on the Jenkins dashboard and then fill in the details as shown in Figure 7 below.

The screenshot shows the Jenkins 'Sign up' page. The form fields are filled with: Username: admin, Password: admin, Confirm password: admin, Full name: administrator, and E-mail address: admin@gmail.com. The 'Sign up' button is visible at the bottom.

Figure 7: Creating the Administrator

Then click on the Sign up button. It will get you logged in as admin.

Now to grant privileges, click on:

- Manage Jenkins and Configure Global Security.
- In project-based Matrix Authorization Strategy, add admin we created and grant all the privileges to it as shown in Figure 8 below:

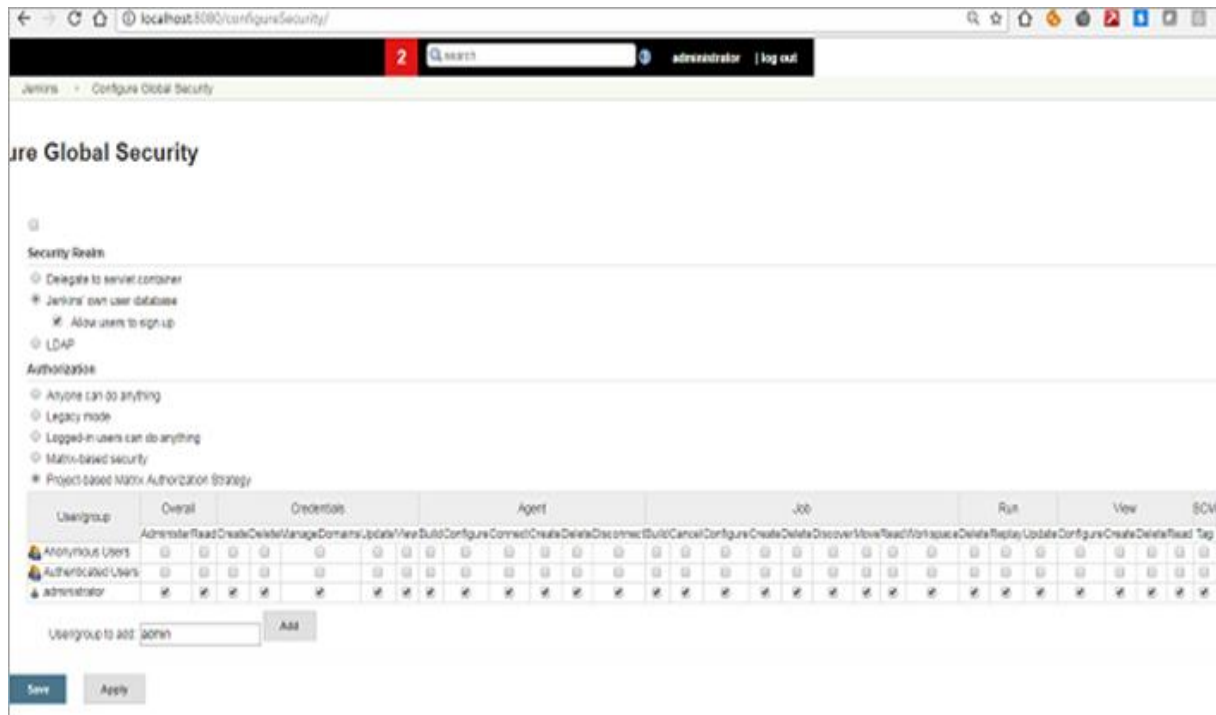


Figure 8: Granting privileges for administrator

Enabling Project Security Matrix

After adding the administrator, the users can be added and required roles can be assigned to them as shown in Figure 9.

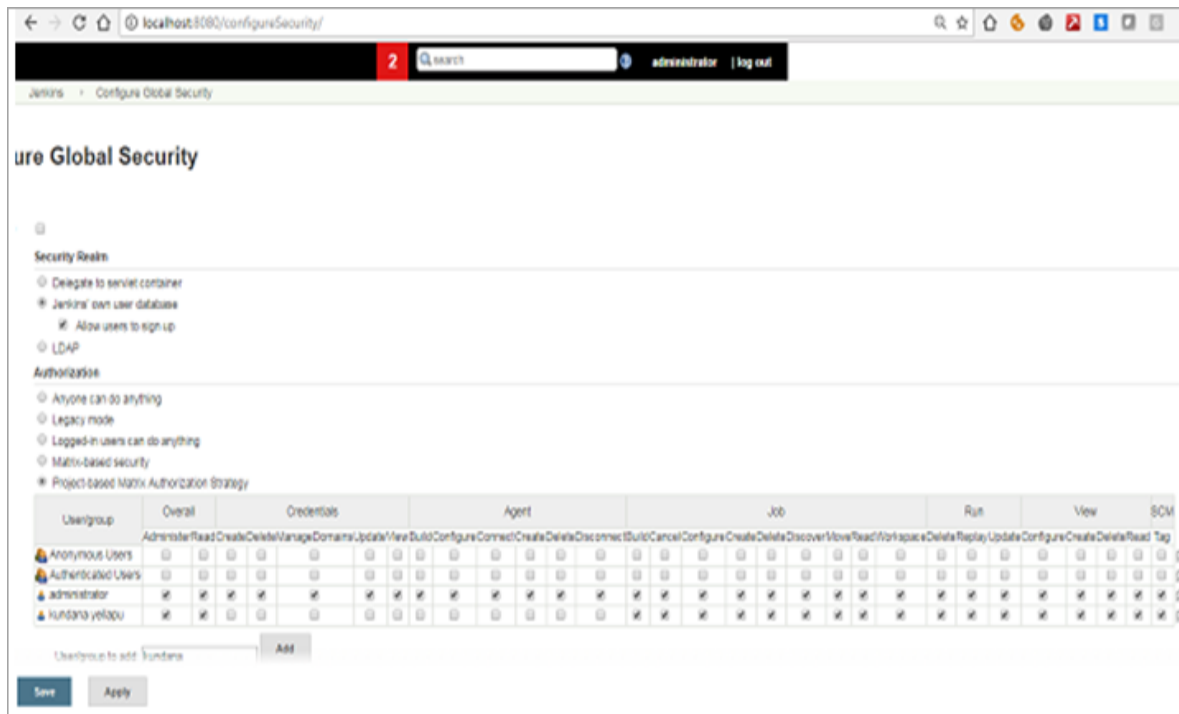


Figure 9: Assigning roles to the users

LDAP Integration with Jenkins

In this document, we are going to see the integration process of LDAP and Jenkins. The default installation of Jenkins server uses internal user database which doesn't work for large development and operations teams. You need to manually create each user.

If you want to access the Jenkins with LDAP credentials (i.e. with Active Directory Credentials), we need to install the LDAP plugin in Jenkins. Jenkins has a native LDAP plugin which can be used to authenticate users against an external LDAP server, such as OpenLDAP etc. So, please make sure to install LDAP plugin in Jenkins before configuring the setup.

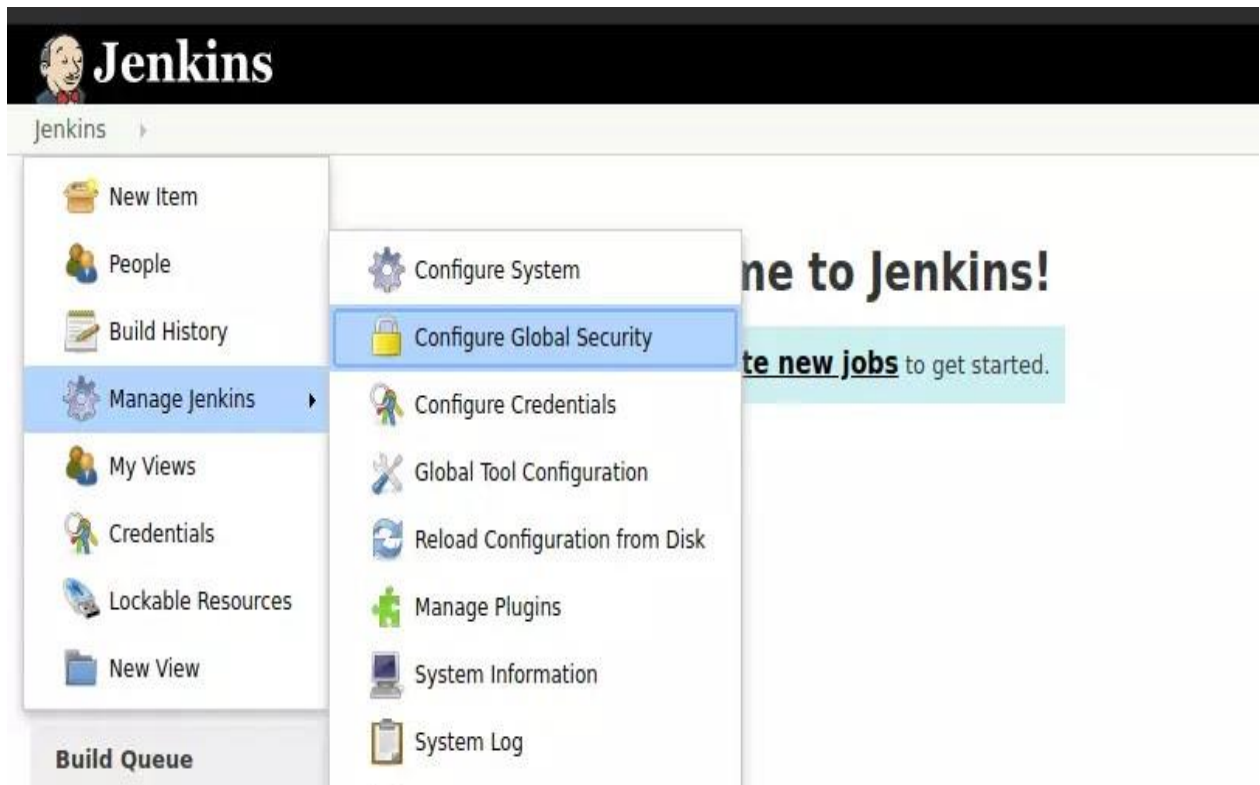
Filter:

Updates	Available	Installed	Advanced	
Enabled	Name ↓	Version	Previously installed version	Uninstall
<input checked="" type="checkbox"/>	bouncycastle API Plugin This plugin provides an stable API to Bouncy Castle related tasks.	2.17		Uninstall
<input checked="" type="checkbox"/>	Command Agent Launcher Plugin Allows agents to be launched using a specified command.	1.3		Uninstall
<input checked="" type="checkbox"/>	JDK Tool Plugin Allows the JDK tool to be installed via download from Oracle's website.	1.2		Uninstall
<input checked="" type="checkbox"/>	jQuery UI plugin This plugin allows you to use jQuery UI on view descriptions.	1.0.2		Uninstall
<input checked="" type="checkbox"/>	LDAP Plugin Adds LDAP authentication to Jenkins	1.20		Uninstall
<input checked="" type="checkbox"/>	Mailer Plugin This plugin allows you to configure email notifications for build results	1.23		Uninstall

Steps: -

The following are the steps to configure the LDAP – Jenkins setup.

1. In order to configure the LDAP – Jenkins setup, some parameter values are needed and need to be filled in. But before that, we need to login into Jenkins server as admin user and navigate to 'Jenkins > Configure Global Security '.



2. Select 'LDAP' under 'Access Control' and then enter the LDAP server details and press 'Save' button. Please fill out the parameters by providing the details of the image below.

Jenkins > Configure Global Security

☒ LDAP ?

Server

Server ?

root DN ?

☐ Allow blank rootDN

User search base ?

User search filter ?

Group search base ?

Group search filter ?

Group membership

☐ Parse user attribute for list of LDAP groups

☐ Search for LDAP groups containing user

Manager DN ?

Manager Password ?

Display Name LDAP attribute ?

Email Address LDAP attribute ?

Environment Properties

Ignore if Unavailable ☐ ?

☐ Unix user/group database ?

Authorization

Activate Windows

3. Test the connection establishment after configuring LDAP by pressing the 'Test LDAP settings' button.

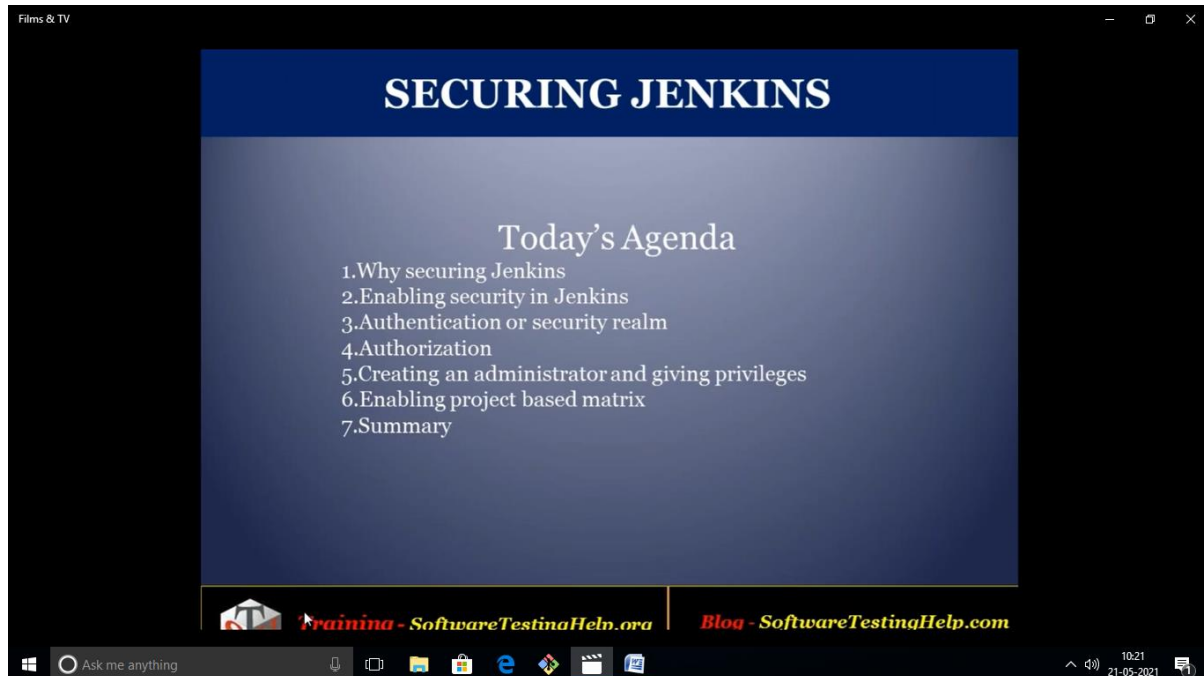
4. After the connection is established, basic authentication will be available. Users can then login using their directory credentials from Jenkins login page.

Conclusion: -

We should have configured Jenkins server successfully to authenticate user via LDAP Server. Please notice that we can not sign in with Jenkins database users once LDAP authentication is enabled because Jenkins local internal user database is disabled. Both cannot coexist, for some reason.

Delegating to a servlet container

This option can be used only when you are running your Jenkins server from a servlet container, such as Apache Tomcat and so on. Enabling this option will allow Jenkins to authenticate users using the servlet containers' realm.



Films & TV

Why Securing Jenkins

- Security is required.
- Outsiders should not login to Jenkins.
- Security can be given to smaller to larger organizations.

Twining SoftwareTestingHelp.com Blog - SoftwareTestingHelp.com

Ask me anything 10:21 21-05-2021

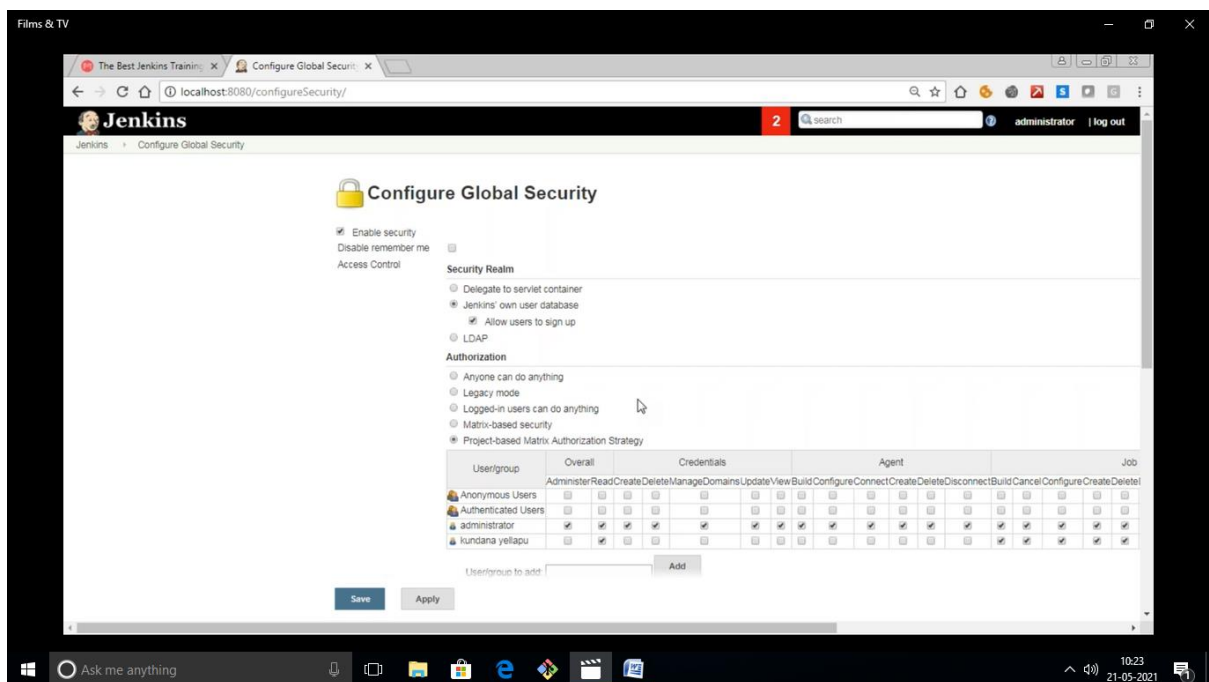
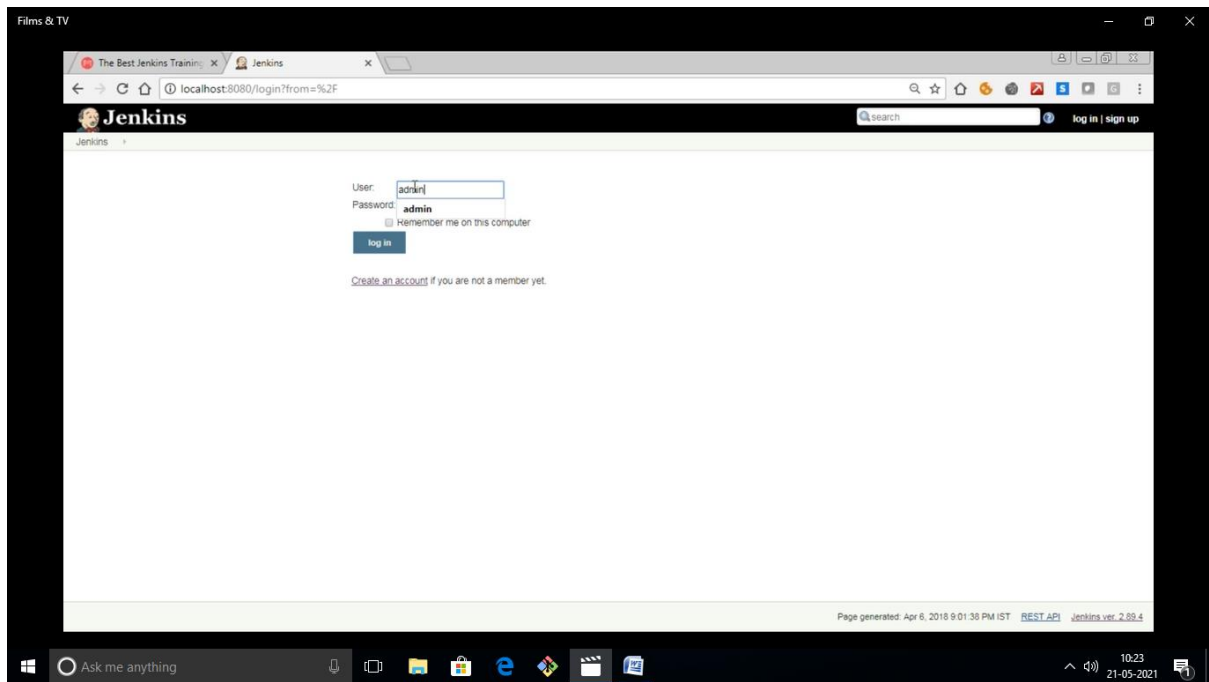
Films & TV

Enabling security in Jenkins

- Click on manage Jenkins.
- Click on configure global security.
- Check enable security.
- Check Jenkins own user database under authentication. Also check sign up.
- Check logged in user can do anything under authorization.
- This is the simplest security model.

Twining SoftwareTestingHelp.com Blog - SoftwareTestingHelp.com

Ask me anything 10:22 21-05-2021



Films & TV

Authentication

Lets Jenkins identify and manage the users who are logging in. The following are the authentication ways:

- 1) Jenkins own user database
- 2) LDAP
- 3) Delegate to servlet container

Twininga - SoftwareTestingHols.com | Blog - SoftwareTestingHols.com

Ask me anything

10:24 21-05-2021

Films & TV

Authorization

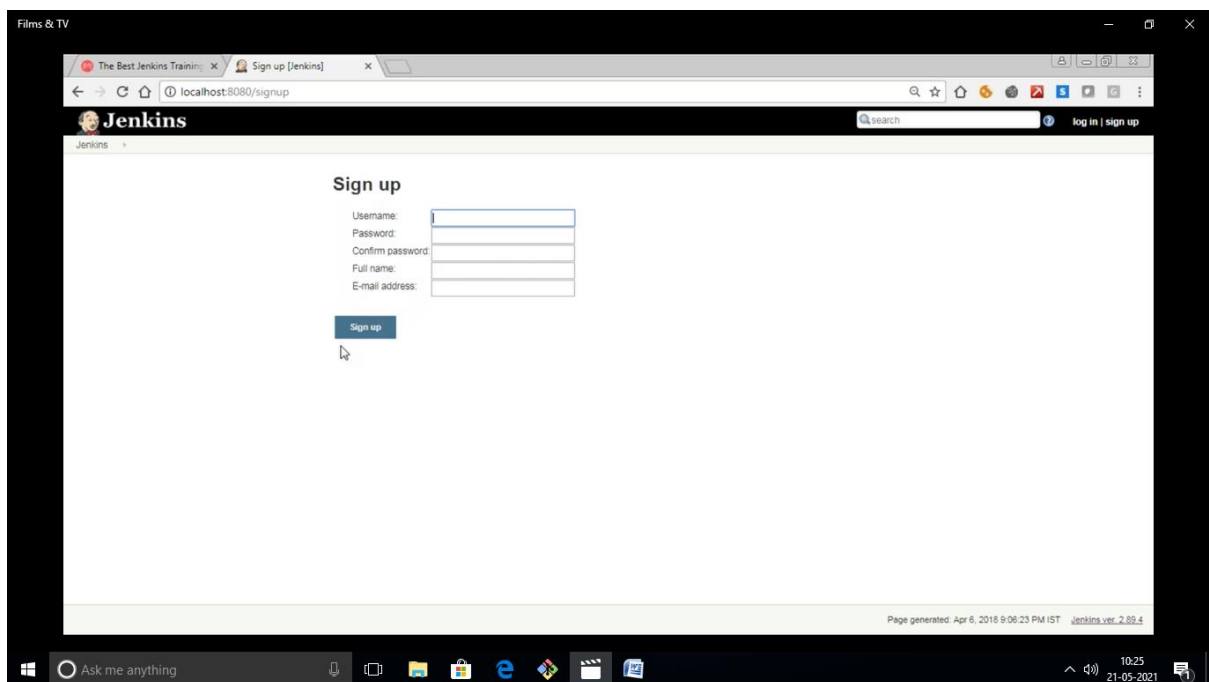
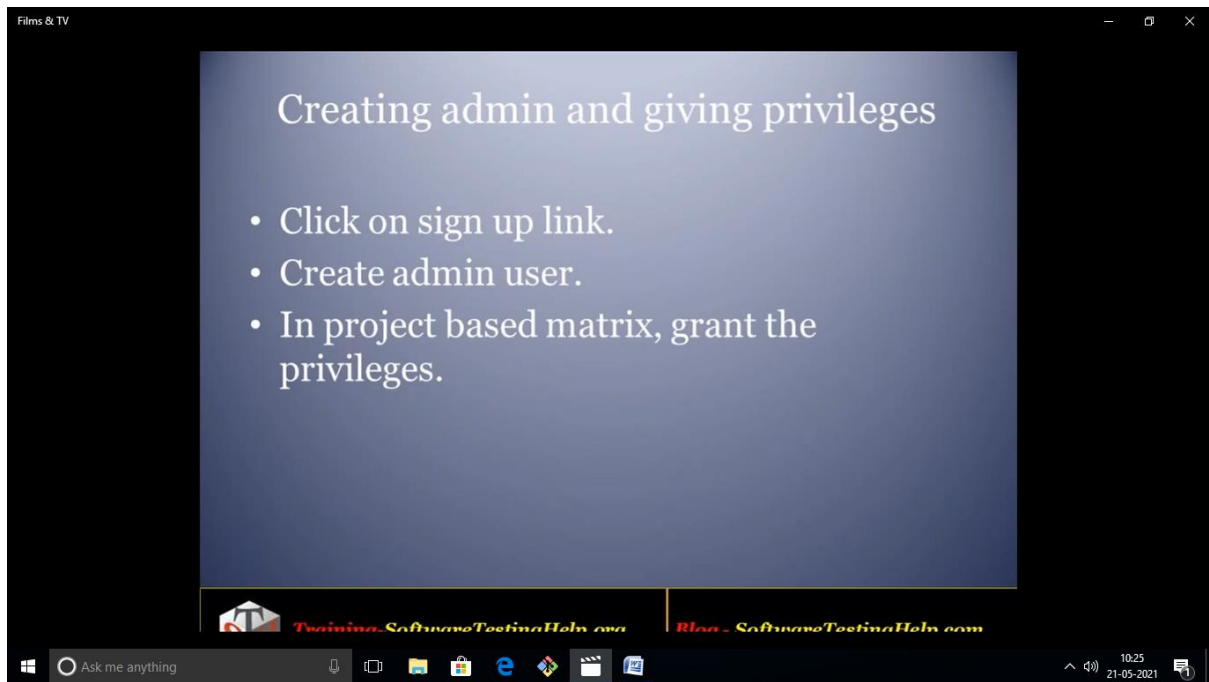
- The process of what the users can do once they are logged in is authorization.
- Various ways are:
- Anyone can do anything.
- Logged in users can do anything.
- Legacy mode
- Matrix based security
- Project based Matrix authorization strategy

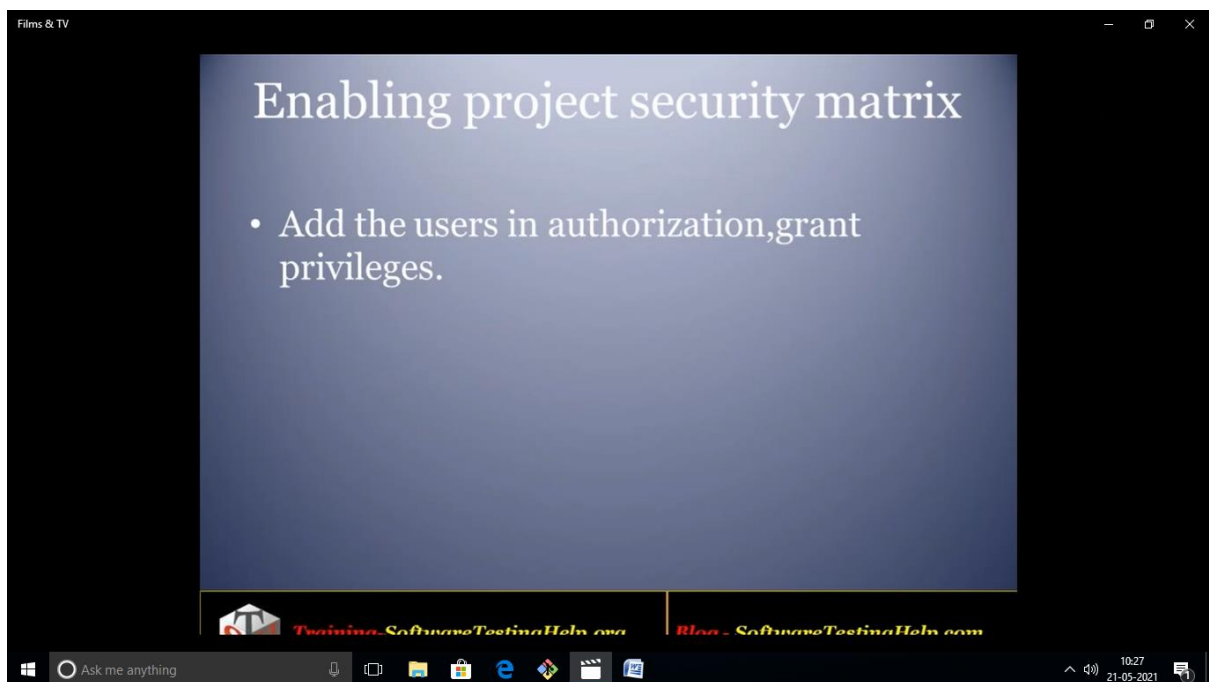
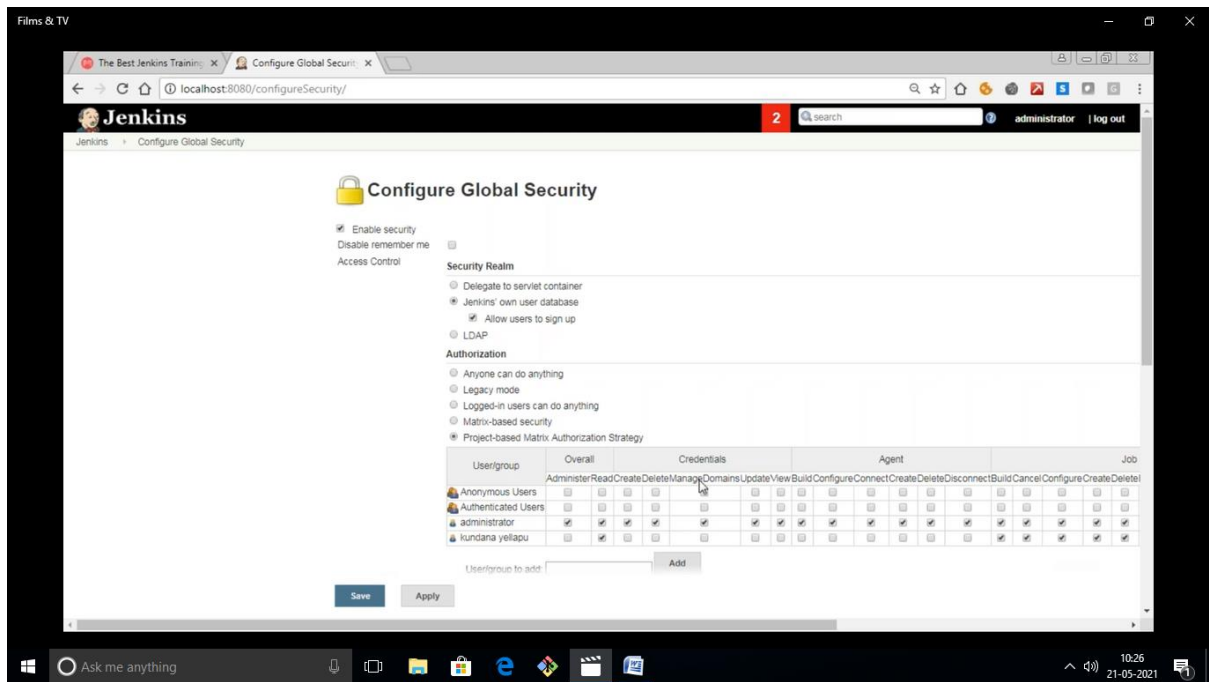
00:02:01 00:02:10

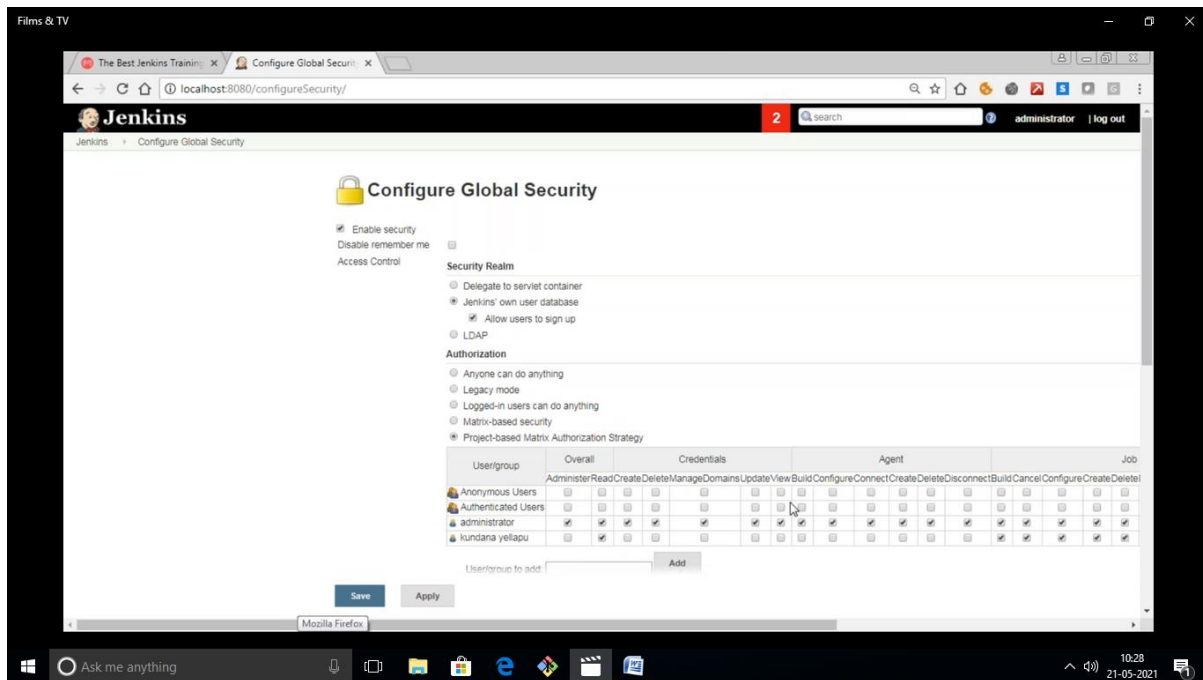
Twininga - SoftwareTestingHols.com | Blog - SoftwareTestingHols.com

Ask me anything

10:24 21-05-2021







CSRF Protection

Cross-Site Request Forgery (CSRF or XSRF) is a type of security vulnerability in web applications. Without protection from CSRF, a Jenkins user or administrator visiting some other web site would allow the operator of that site to perform actions in Jenkins as the victim.

CSRF Protection in Jenkins

CSRF protection uses a token (called *crumb* in Jenkins) that is created by Jenkins and sent to the user. Any form submissions or similar action resulting in modifications, like triggering builds or changing configuration, requires that the crumb be provided. The crumb contains information identifying the user it was created for, so submissions with another user's token would be rejected. All of this happens in the background and has no visible impact except in rare circumstances, e.g., after a user's session expired and they logged in again.

CSRF Protection

Crumb Issuer

Default Crumb Issuer

☐ Enable proxy compatibility



The *Default Crumb Issuer* encodes the following information in the hash used as crumb:

- The user name that the crumb was generated for
- The web session ID that the crumb was generated in
- The IP address of the user that the crumb was generated for
- A salt unique to this Jenkins instance

All of this information needs to match when a crumb is sent back to Jenkins for that submission to be considered valid.

The only supported option *Enable proxy compatibility* removes information about the user IP address from the token. This can be useful when Jenkins

is running behind a reverse proxy and a user's IP address as seen from Jenkins would regularly change.

Disabling CSRF Protection

Outdated plugins that send HTTP requests to Jenkins may not work with CSRF protection enabled. In this case, it may be necessary to disable CSRF protection temporarily.

It is strongly recommended that CSRF protection be left enabled, including on instances operating on private, fully trusted networks.

To disable CSRF protection, set the system property `hudson.security.csrf.GlobalCrumbIssuerConfiguration.DISABLE_CSRF_PROTECTION` to `true` on startup.

Markup Formatters

Jenkins allows users with the appropriate permissions to enter descriptions of various objects, like views, jobs, builds, etc. These descriptions are filtered by *markup formatters*. They serve two purposes:

1. Allow users to use rich formatting for these descriptions
2. Protect other users from [Cross-Site Scripting](#) (XSS) attacks

Configuring the Markup Formatter

The markup formatter can be configured in *Manage Jenkins » Configure Global Security » Markup Formatter*.

The default markup formatter *Plain text* renders all descriptions as entered:

Agent → Controller Access Control

The Jenkins controller and agents can be thought of as a distributed process which executes across multiple discrete processes and machines. This allows an agent to ask the controller process for information available to it, for example, the contents of files, etc., and even to have the controller run certain commands when requested by the agent.

So while not building on the master node is a good general practice to protect from bugs and less sophisticated attackers, an agent process taken over by a malicious user would still be able to obtain data or execute commands on the Jenkins controller. To prevent this, the Agent → Controller Access Control system prevents agent processes from being able to send malicious commands to the Jenkins controller. It is enabled by default, but can be disabled in the web UI by un-checking the box on the *Manage Jenkins » Configure Global Security* page.

It is strongly recommended not to disable the Agent → Controller Access Control system.

Agent → Controller Security

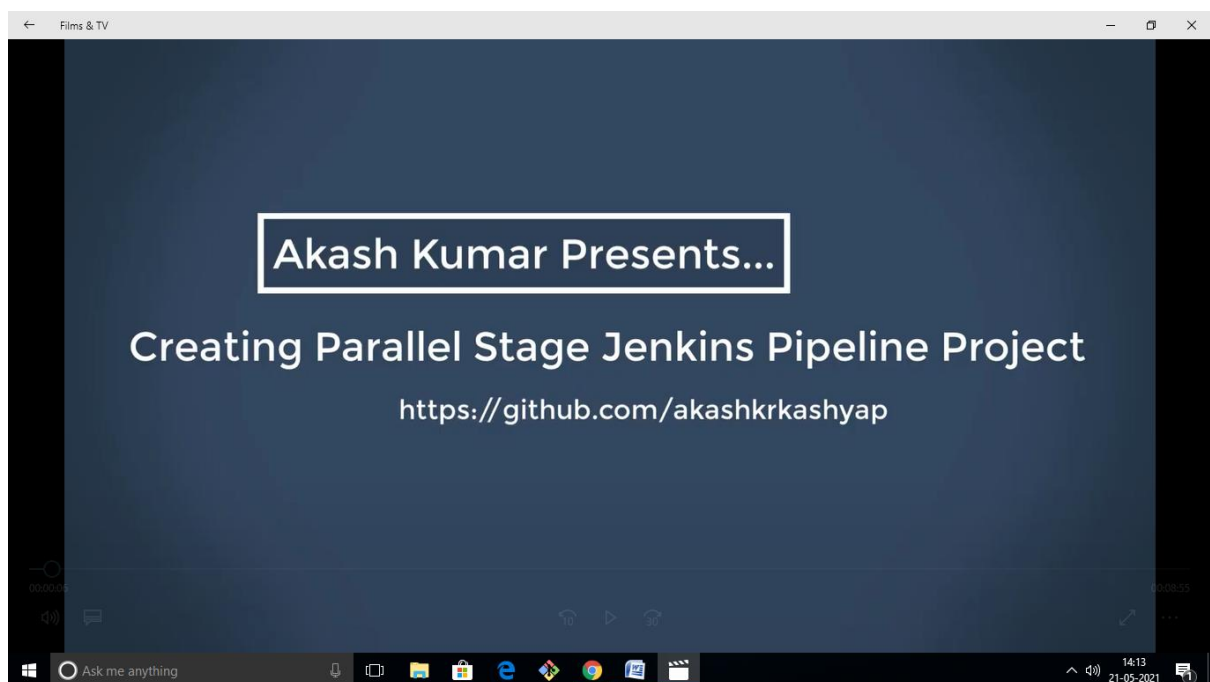
✓ Enable Agent → Controller Access Control

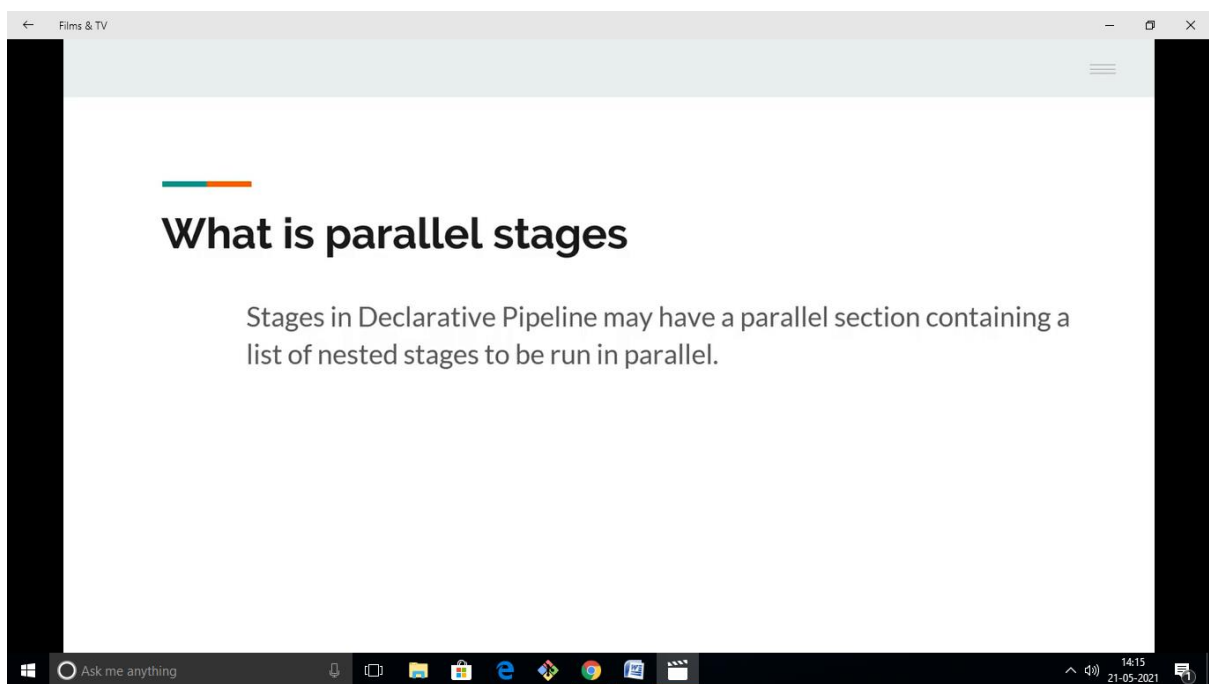
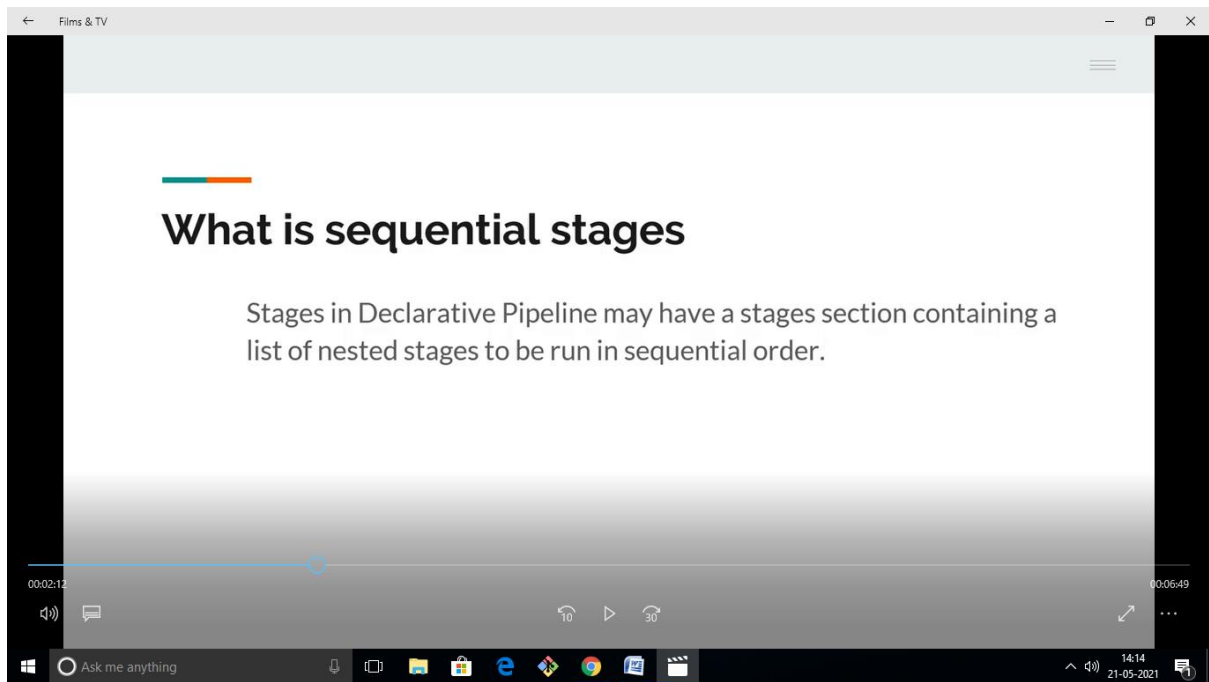


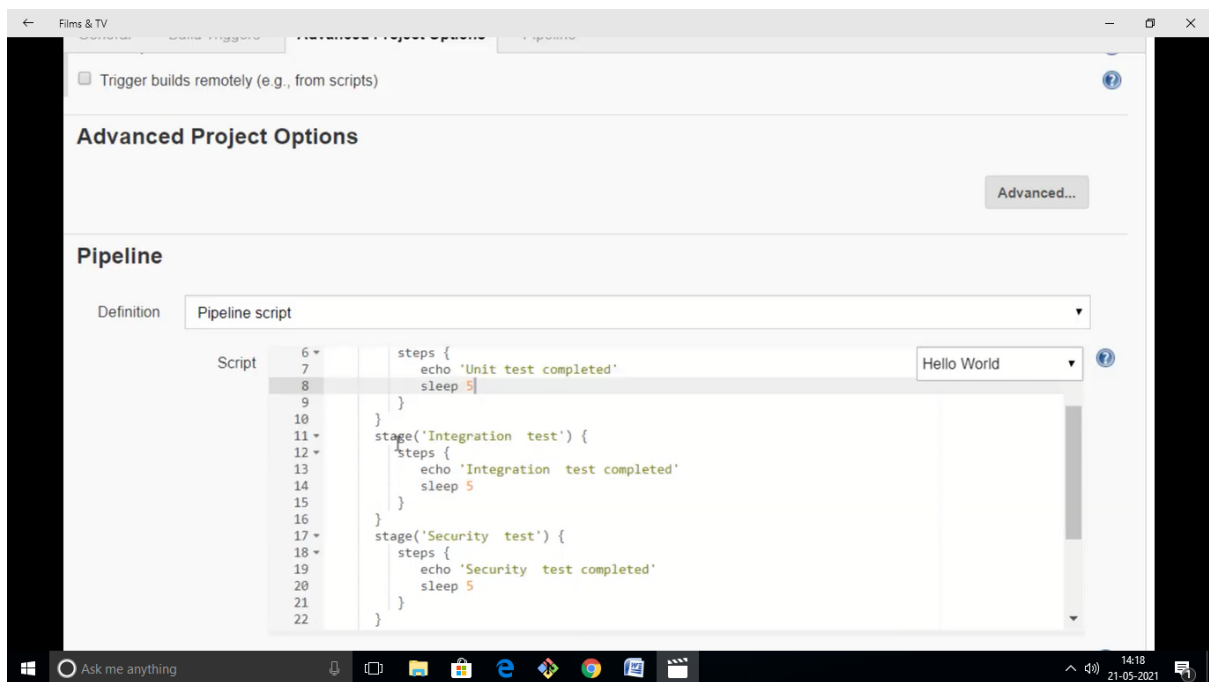
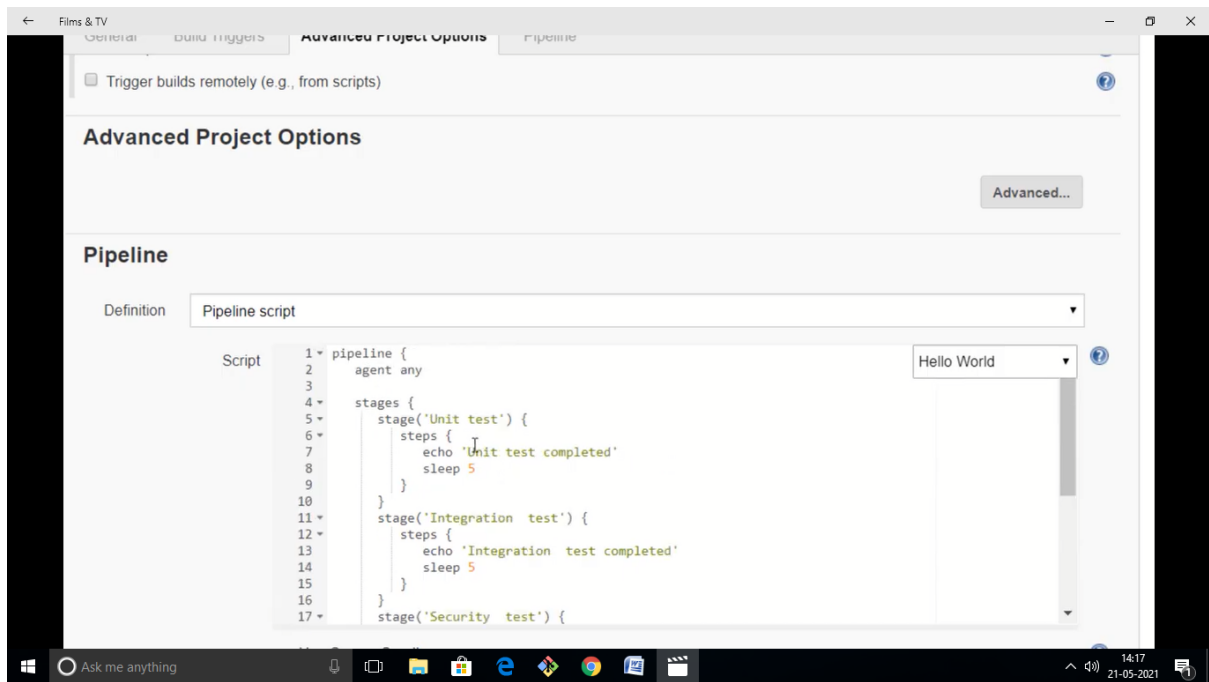
Rules can be tweaked [here](#)

As an alternative to disabling Agent → Controller Access Control, administrators can selectively allow greater access. See the [documentation](#) for details.

Parallel STAGE PIPELINE







test_sequential_pipeline [Jenkins] x

3.136.155.137:8080/job/test_sequential_pipeline/

Jenkins test_sequential_pipeline enable auto refresh

Back to Dashboard
Status
Changes
Build Now
Delete Pipeline
Configure
Full Stage View
Rename
Pipeline Syntax

Pipeline test_sequential_pipeline

Recent Changes

Stage View

Average stage times:
(Average full run time: ~21s)

Unit test	Integration test	Security test	Selenium UI test
5s	5s	5s	5s

Build History

#1 Apr 19, 2020 2:03 PM

Atom feed for all Atom feed for failures

Permalinks

00:05:39 00:03:22

Page generated: Apr 19, 2020 2:03:34 PM UTC BEST API Jenkins ver. 2.161.1

19:34 19-04-2020

14:19 21-05-2021

Films & TV

General build triggers Advanced Project Options pipeline

Pipeline

Definition Pipeline script

Script

```

1 pipeline {
2   agent any
3
4   stages {
5     stage ('My Java Project Test Scenarios') {
6       parallel {
7         stage('Unit test') {
8           steps {
9             echo 'Unit test completed'
10            sleep 5
11          }
12        }
13        stage('Integration test') {
14          steps {
15            echo 'Integration test completed'
16            sleep 5
17          }
18        }
19      }
20    }
21  }
22 }

```

☒ Use Groovy Sandbox

[Pipeline Syntax](#)

00:06:43 00:02:18

Save Apply

14:22 21-05-2021

