Project-based authorization

Project-based authorization in Jenkins allows administrators to control access to specific projects or jobs based on user roles and permissions. This means that users or groups can be granted different levels of access to different projects within Jenkins. Here are the key concepts and steps for implementing project-based authorization in Jenkins:

Steps to Set Up Project-Based Authorization:

1. **Configure Global Security:**
    - In the Jenkins dashboard, go to "Manage Jenkins" > "Configure Global Security."
    - Under "Authorization," select "Project-based Matrix Authorization Strategy."
    - Configure the desired roles and permissions for "Anonymous," "Authenticated," and other users or groups.
2. **Configure Project-Based Authorization:**
    - Open the configuration page for a specific project by clicking on the project's name in the Jenkins dashboard.
    - Scroll down to the "Access Control" section.
    - Select "Project-based Matrix Authorization Strategy" as the security model.
    - Configure the matrix of roles and permissions for specific users or groups associated with the project.
3. **Assign Users to Roles:**
    - In the "Manage Jenkins" > "Manage Users" section, create or edit user accounts.
    - Assign users to specific roles (e.g., "Developer," "Administrator") based on their responsibilities.
4. **Test Authorization:**
    - Log in as different users with various roles to verify that the configured matrix accurately reflects the desired access control for the project.

Example Matrix Authorization Configuration:

- **Example Matrix:**

User/Permission	Read	Configure	Build	Cancel	Delete	Workspace
Administrator	✓	✓	✓	✓	✓	✓
Developer	✓		✓			✓
Tester	✓					
Anonymous						

In the example matrix, administrators have full control over the project, developers can read, build, and access the workspace, testers can only read, and anonymous users have no access.

Project-based authorization allows for fine-grained control over who can perform various actions within each project, contributing to a more secure and organized Jenkins environment.

Role-based authorization

Role-based authorization in Jenkins involves assigning specific roles to users or groups and defining permissions for each role. This approach allows administrators to control access to Jenkins resources based on user roles, simplifying the management of permissions and ensuring that users have the appropriate level of access for their responsibilities. Jenkins provides plugins such as the "Role-based Authorization Strategy" plugin to implement role-based access control.

Here are the steps to set up role-based authorization in Jenkins:

Prerequisites:

- Ensure you have administrative access to Jenkins.
- Install the "Role-based Authorization Strategy" plugin.

Steps to Set Up Role-Based Authorization:

1. **Install Role-based Authorization Strategy Plugin:**
    - Navigate to "Manage Jenkins" > "Manage Plugins" > "Available" tab.
    - Search for "Role-based Authorization Strategy" and install it.
    - Restart Jenkins after installing the plugin.

2. **Configure Global Security:**
    - Go to "Manage Jenkins" > "Configure Global Security."
    - Under the "Authorization" section, select "Role-Based Strategy."
    - Click "Save" to apply the changes.
3. **Manage Roles:**
    - After enabling the Role-Based Strategy, a new "Manage and Assign Roles" link will appear in the sidebar under "Manage Jenkins."
    - Click on "Manage and Assign Roles" to access the role configuration.
4. **Create Roles:**
    - Click on "Manage Roles" to define roles based on your organizational needs.
    - Create roles such as "Administrator," "Developer," "Tester," etc.
    - Assign appropriate permissions to each role.
5. **Assign Users or Groups to Roles:**
    - In the "Assign Roles" section, assign users or groups to specific roles.
    - You can assign multiple roles to a user or group, and users can have different roles on different projects.
6. **Configure Project-Based Roles (Optional):**
    - If you want to have project-specific roles, configure roles within each project.
    - In the project configuration, you can assign roles to specific users or groups.
7. **Test Permissions:**
    - Log in as different users or use the "Check Permissions" feature to verify that the configured roles accurately reflect the desired access control.

Example Role Configuration:

- **Example Roles:**
    - **Administrator:**
        - Overall access to Jenkins.

- Manage users, create and configure jobs, and administer Jenkins.

- **Developer:**
  - Build jobs, read and view job configurations.
  - Limited access compared to administrators.
- **Tester:**
  - Read-only access to jobs.
  - Limited access for viewing build results and configurations.

Benefits of Role-Based Authorization:

- **Simplified Management:**
  - Roles abstract permissions, making it easier to manage access control.
  - Changes to roles are reflected globally across all projects.
- **Fine-Grained Control:**
  - Assign different roles to users or groups based on their responsibilities.
  - Control access at both the global and project levels.
- **Scalability:**
  - Easily scale access control as your Jenkins instance grows by adding or modifying roles.

Role-based authorization provides a flexible and scalable approach to managing access control in Jenkins, ensuring that users have the right permissions for their roles while maintaining security and compliance.

Jenkins Dashboard Overview:

1. **Accessing the Dashboard:**
   - When you log in to your Jenkins instance, the default landing page is the dashboard.
2. **Job Status:**
   - The main section of the dashboard displays a list of jobs along with their status.
   - Jobs may have different statuses such as success, unstable, failure, or aborted.

3. **Build History:**
    - The dashboard typically includes a build history for each job, showing recent builds and their status.
    - You can click on a specific build number to view detailed information about that build.
4. **Quick Navigation:**
    - The dashboard provides quick navigation links to different sections of Jenkins, such as "New Item" for creating a new job, "Build Executor Status" for node information, and "Manage Jenkins" for system configuration.
5. **Viewing Specific Jobs:**
    - Click on a job name to go to the job's detail page, where you can see build history, configure the job, and access other details.

Example Job Dashboard Components:

In this example:

- **Job List:**
    - The list of jobs displays their names, status, and weather icons indicating recent build stability.
- **Build History:**
    - Each job shows a history of recent builds with build numbers, status indicators, and timestamps.
- **Navigation Links:**
    - Links to other Jenkins sections like "New Item," "Build Executor Status," and "Manage Jenkins" are available on the left sidebar.

Customizing the Dashboard:

Jenkins provides a range of plugins that can enhance or customize the dashboard. Some plugins allow you to create custom views, widgets, or dashboards tailored to specific needs.

Additional Features:

1. **Blue Ocean:**
    - Jenkins Blue Ocean is a modern user interface for Jenkins that provides a more visual and intuitive way to interact with Jenkins pipelines and job status. It includes a redesigned dashboard.
2. **Viewing Specific Job Details:**
    - Navigate to a specific job's detail page for comprehensive information about that job, including build history, console output, and configuration.
3. **Plugins:**
    - Additional plugins may introduce new dashboard components, widgets, or views. Check your Jenkins instance for installed plugins that may enhance the dashboard.

JENKINS SERVER AUTOMATION SCRIPT

```
#!/bin/bash
sudo yum update –y
sudo wget -O /etc/yum.repos.d/jenkins.repo \
    https://pkg.jenkins.io/redhat-stable/jenkins.repo
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
sudo yum upgrade
sudo sudo dnf install java-17-amazon-corretto -y
sudo yum install jenkins -y
sudo systemctl enable jenkins
sudo systemctl start jenkins
```

- **Use above userdata and launch ec2 instances for automation of Jenkins server**

Jenkins jobs are the building blocks of continuous integration and automation, allowing you to define tasks such as building, testing, deploying, and more. Here's an explanation of key elements in the Jenkins job configuration:

1. General Configuration:

- **Job Name:** Assign a unique and descriptive name to the job.
- **Description:** Provide a brief description of the job's purpose or function.
- **Discard Old Builds:** Specify how many builds to keep to manage disk space.

2. Source Code Management (SCM):

- **Repository URL:** Specify the URL of the version control repository (e.g., Git, Subversion) containing your source code.
- **Branches to Build:** Define the branches to include or exclude during the build process.

3. Build Triggers:

- **Build Periodically:** Schedule builds at specific times using cron syntax.
- **Build when a change is pushed to GitHub or other SCM:** Trigger builds automatically when changes are pushed to the repository.
- **Build after other projects are built:** Chain jobs together by triggering one job after the completion of another.

4. Build Environment:

- **Delete workspace before build starts:** Optionally clean the workspace before the build.
- **Use secret text(s) or file(s):** Inject secrets or credentials into the build environment securely.

5. Build:

- **Build Steps:** Define the sequence of commands or tasks to execute during the build.
- **Build using Shell/Batch script:** Write shell or batch scripts to execute build steps.
- **Invoke Ant/Maven/Gradle:** Execute builds using popular build tools.
- **Windows PowerShell:** For Windows environments, execute PowerShell scripts.

6. Post-Build Actions:

- **Archive the artifacts:** Specify files or directories to be archived as build artifacts.
- **Email Notification:** Send email notifications based on build status.
- **Publish JUnit test result report:** Report and display JUnit test results.
- **Publish HTML reports:** Display HTML reports generated during the build.
- **Deploy to container:** Deploy the built artifacts to a container or server.

7. Build Environment:

- **Set environment variables:** Define environment variables that will be available during the build.
- **Abort the build if it's stuck:** Specify a timeout to abort the build if it does not complete within a certain time.

8. Advanced Project Options:

- **Quiet period:** Specify a quiet period to delay the build after a triggering event.
- **Block build when downstream project is building:** Prevent concurrent builds of downstream projects.
- **Execute concurrent builds if necessary:** Allow multiple builds to run concurrently.

9. Save and Apply:

- After configuring the job, save the changes and apply the configuration.

10. Build Now:

- Manually trigger a build by clicking on "Build Now."

Example of a Freestyle Project Configuration:

Here is a simplified example of configuring a Freestyle Project in Jenkins:

1. **General:**
    - *Project Name:* MyProject
    - *Description:* This is a sample project.
2. **Source Code Management (SCM):**

- *Repository URL:* https://github.com/devopstraininghub/webapp1.git
- *Credentials:* None
- *Branches to Build: /main*

3. **Build Triggers:**
    - *Build periodically:* H/5 * * * *
4. **Build:**
    - *Build Steps:* Execute shell

      echo "Building the project..."
5. **Save and Apply:**
    - Save the configuration changes.

This is a basic example, and the configuration can be extended based on the complexity and requirements of your project. Jenkins supports various plugins that can enhance and customize job configurations based on your specific needs.

Upstream and Downstream In Jenkins

In Jenkins, the terms "downstream" and "upstream" refer to the relationships between jobs in a build pipeline. Understanding these relationships is crucial for implementing continuous integration and continuous delivery (CI/CD) workflows.

Upstream Job:

The upstream job is the job that triggers another job. In a CI/CD pipeline, it's the job that initiates the build process. When the upstream job is executed, it can trigger one or more downstream jobs to start. Upstream jobs are typically at the beginning of the build pipeline and are responsible for initiating the build process.

Downstream Job:

The downstream job is the job that gets triggered by an upstream job. When the upstream job completes its execution, it triggers the downstream job(s) to start. Downstream jobs depend on the successful completion of their upstream jobs. Downstream jobs are often responsible for tasks like testing, deployment, or other activities that follow the build process.

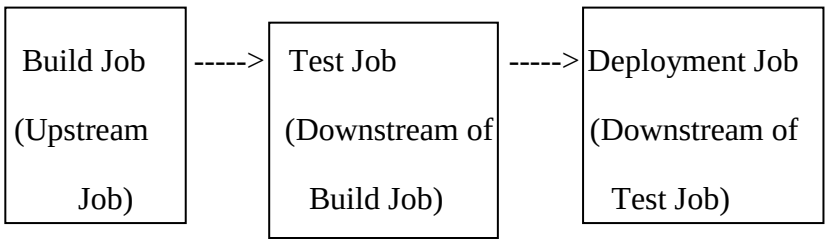
Example Scenario:

Consider a simple CI/CD pipeline with three jobs:

1. **Build Job (Upstream):**
    - This job compiles the source code, runs unit tests, and produces an executable or deployable artifact.
2. **Test Job (Downstream of Build Job):**
    - This job runs additional tests (integration tests, performance tests) on the artifact produced by the Build Job.
    - The Test Job is triggered automatically when the Build Job completes successfully.
3. **Deployment Job (Downstream of Test Job):**
    - This job deploys the artifact to a staging environment or production.
    - The Deployment Job is triggered automatically when the Test Job completes successfully.

In this example:

- The **Build Job** is the upstream job because it triggers the **Test Job**.
- The **Test Job** is both an upstream job (triggered by the Build Job) and a downstream job (triggering the Deployment Job).



Each arrow represents the triggering relationship between jobs. The sequence from left to right indicates the flow of the pipeline. Upstream jobs trigger downstream jobs upon successful completion.

Remember, Jenkins itself provides a visual representation of these relationships on the Jenkins web interface when you configure jobs and view the build history. Upstream jobs will show downstream jobs in the "Build Pipeline" view or similar views.

Benefits of Upstream and Downstream Relationships:

1. **Automated Pipelines:**
    - Upstream and downstream relationships allow the automation of entire build and deployment pipelines.
2. **Dependency Management:**
    - Downstream jobs depend on the success of upstream jobs, ensuring that tasks are executed in the correct order.
3. **Parallel Execution:**
    - Upstream jobs can trigger multiple downstream jobs in parallel, optimizing build times and resource utilization.
4. **Conditional Execution:**
    - Jobs can be configured to trigger downstream jobs based on specific conditions (e.g., only if the upstream job is successful).
5. **Easy Maintenance:**
    - Changes to one part of the pipeline (e.g., adding a new test) can be made in the downstream job without affecting the upstream job.

By establishing upstream and downstream relationships, Jenkins enables the creation of robust and automated CI/CD pipelines, promoting efficiency and consistency in the software development process.

Crontab

Cron is a time-based job scheduler in Unix-like operating systems. The cron service enables users to schedule jobs (commands or scripts) to run periodically at specified intervals. The configuration for cron jobs is managed using the crontab (cron table) command.

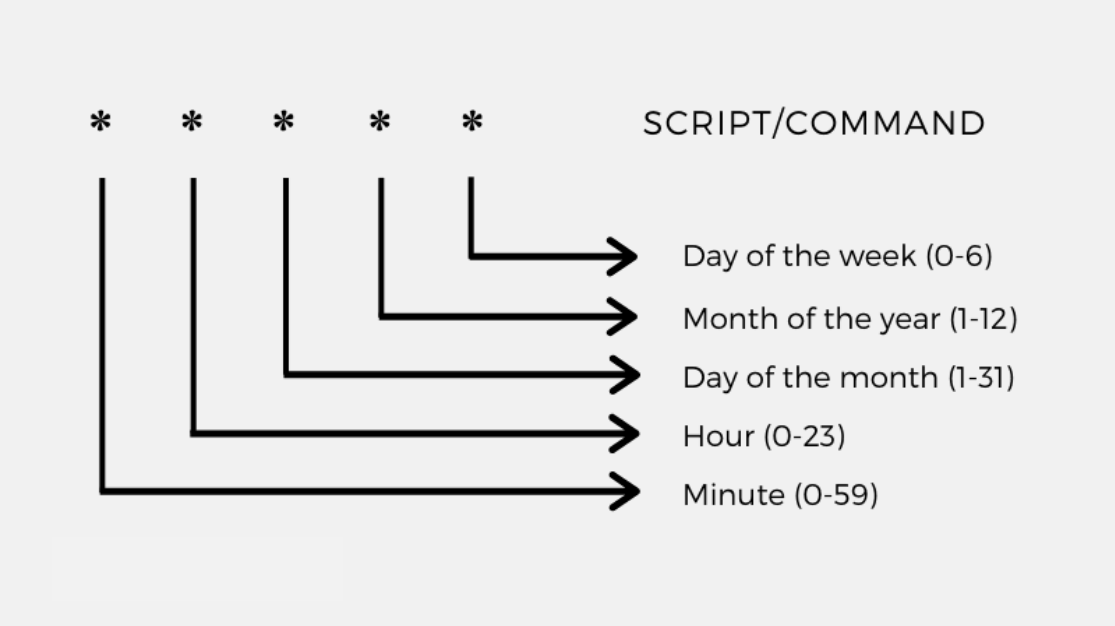
Crontab Syntax:

The crontab syntax consists of five fields followed by the command to be executed. The fields represent minute, hour, day of the month, month, and day of the week. The syntax is as follows:

* * * * * command-to-be-executed

- The asterisk (*) denotes "every" or "any."

- Each field can be a single number, a list of numbers, or an asterisk.



Examples:

1. Run a command every day at 3:30 AM:

   30 3 * * * command-to-be-executed

2. Run a script every Monday at 2 PM:

   0 14 * * 1 script-to-be-executed

3. Run a command every hour:

   0 * * * * command-to-be-executed

Apache Tomcat

Apache Tomcat is an open-source implementation of the Java Servlet, JavaServer Pages (JSP), and Java Expression Language technologies. It powers numerous large-scale, mission-critical web applications across various industries. Here's an overview of Apache Tomcat:

Key Features:

1. **Servlet and JSP Container:**
    - Tomcat provides a servlet container and a JSP container, enabling the development and execution of Java-based web applications.
2. **Open Source:**
    - Tomcat is an open-source project under the Apache Software Foundation. Its source code is freely available, and it has a large community of users and contributors.
3. **Java EE Compatibility:**
    - While Tomcat is not a full Java EE (Enterprise Edition) server, it supports a subset of Java EE specifications and is suitable for hosting Java web applications.
4. **Lightweight:**
    - Tomcat is designed to be lightweight and efficient. It is often used for deploying web applications where a full Java EE server might be considered too heavyweight.
5. **Cross-Platform:**
    - Tomcat is cross-platform and can run on various operating systems, including Windows, Linux, and macOS.
6. **Embeddable:**
    - Tomcat can be embedded in other applications, allowing developers to package and distribute their applications with an embedded Tomcat server.
7. **Security Features:**
    - Tomcat includes security features, such as support for SSL/TLS, user authentication, and authorization.

Basic Concepts:

1. **Connector:**
    - A connector is a component that enables Tomcat to communicate with different protocols, such as HTTP. The HTTP connector listens for incoming requests and forwards them to the appropriate components for processing.
2. **Catalina:**
    - Catalina is the servlet container for Tomcat. It is responsible for processing servlets and JSP pages.
3. **Web Application:**
    - A web application in Tomcat is a collection of servlets, JSP pages, HTML pages, and other resources organized in a specific directory structure. Each web application is contained in a separate directory within the Tomcat **webapps** directory.
4. **Context:**
    - A context represents a single web application in Tomcat. It defines the configuration and attributes associated with that application.
5. **Server and Service:**
    - Tomcat can host multiple services, and each service can have multiple connectors. The server is the top-level container that holds services.

Common Configuration Files:

1. **server.xml:**
    - The main configuration file for Tomcat, defining the server, services, connectors, and other global settings.
2. **web.xml:**
    - The deployment descriptor for a web application. It provides configuration information for the web application, including servlet mappings and initialization parameters.

Deployment:

1. **Manual Deployment:**
    - Copying the web application directory (WAR file or an unpacked directory) to the **webapps** directory of Tomcat.
2. **Manager Application:**
    - Tomcat provides a web-based manager application that allows deploying, undeploying, and managing web applications through a web interface.

Accessing Tomcat:

1. **Web Interface:**
    - Tomcat provides a web-based management interface that can be accessed using a web browser. It typically runs on port 8080.

        http://localhost:8080
2. **Logs:**
    - Tomcat logs can be found in the **logs** directory within the Tomcat installation.

Example Installation Steps (Linux):

1. Download Tomcat from the tomcat official website

    Cd /opt

    wget https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.84/bin/apache-tomcat-9.0.84.tar.gz
2. Extract the downloaded archive.

    tar -xvf apache-tomcat-9.0.84.tar.gz
3. Go to cd apache-tomcat-9.0.84.tar.gz/
4. Go to cd webapps/managers/META-NF
5. Edit the file vim context.xml ➔allow =".*"
6. Now come to apache-tomcat-9.0.84.tar.gz dir and go to cd conf/ dir

7. Edit the vim tomcat.xml file ➔remove all the data

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="manager-gui"/>
  <user username="tomcat" password="Tomcat" roles="manager-gui, manager-script,
manager-status"/>
</tomcat-users>
```

8. Navigate to the Tomcat bin directory.

   cd apache-tomcat-.9.0.84/bin

9. Start Tomcat.

   ./startup.sh

10. Access the Tomcat web interface in a web browser.

    http://public-ip-address:8080

Tomcat Server automation using user data

cd /opt

sudo wget https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.84/bin/apache-tomcat-9.0.84.tar.gz

sudo tar -xvf /opt/apache-tomcat-9.0.84.tar.gz

cd /opt/apache-tomcat-9.0.84/webapps/manager/META-INF

sudo sed -i 's/"127\\.\\d+\\.\\d+\\.\\d+|::1|0:0:0:0:0:0:0:1"/".*"/g' context.xml

cd /opt/apache-tomcat-9.0.84/conf

sudo mv tomcat-users.xml tomcat-users_bkup.xml

sudo touch tomcat-users.xml

sudo echo '<?xml version="1.0" encoding="utf-8"?>

<tomcat-users>

<role rolename="manager-gui"/>

<user username="tomcat" password="Tomcat" roles="manager-gui, manager-script,
manager-status"/>

</tomcat-users>' > tomcat-users.xml

cd /opt/apache-tomcat-9.0.84/conf/

sudo sed -i 's/Connector port="8080"/Connector port="8081"/g' server.xml

sudo /opt/apache-tomcat-9.0.84/bin/startup.sh

Deploying a web application to Apache Tomcat using Jenkins

Deploying a web application to Apache Tomcat using Jenkins , and it's commonly achieved through the use of Jenkins jobs and the deployment manager application provided by Tomcat.

Prerequisites:

1. **Jenkins Installation:**
    - Ensure that Jenkins is installed and configured on your server.
2. **Apache Tomcat:**
    - Have Apache Tomcat installed and running on the server.

Jenkins Job Configuration:

1. **Create a New Jenkins Job:**
    - In the Jenkins dashboard, click on "New Item" to create a new job.
2. **Configure General Settings:**
    - Enter a name for the job (e.g., "Deploy to Tomcat").
    - Choose the "Freestyle project" option.
3. **Source Code Management:**
    - Choose "Git" as the source code management option.
    - Provide the URL of your Git repository.
    - Specify the branch to build.
4. **Build Triggers:**
    - Configure build triggers based on your requirements (e.g., Poll SCM, GitHub webhook, etc.).

5. **Build:**
    - In the build section, add the necessary build steps to compile or package your web application. This might involve running build tools like Maven, Gradle, or a script.

```
# Example Maven Build Step
    mvn clean package
```

6. **Post-Build Actions:**
    - Add a post-build action to deploy the artifact to Tomcat.
    - Choose "Deploy war/ear to a container."(install Deploy to container plugin)
7. **Deploy to Container Configuration:**
    - Select "Tomcat-9" as the container.
    - Provide the Tomcat URL (e.g., **http://localhost:8080**).
    - Enter the Tomcat username and password.
    - Specify the context path where you want to deploy the application.
8. **Save the Job Configuration:**
    - Save your Jenkins job configuration.

Run the Jenkins Job:

1. **Trigger the Job:**
    - Manually trigger the Jenkins job or configure automatic triggers.
2. **Monitor the Build:**
    - Monitor the Jenkins build console output for any errors during the build and deployment process.
3. **Verify Deployment:**
    - Access your deployed web application through the Tomcat Manager or the context path you specified.

Important Notes:

- Ensure that the Tomcat Manager application is configured correctly with the necessary roles and permissions.
- Store sensitive information like usernames and passwords securely, preferably using Jenkins credentials or environment variables.
- Adjust the job configuration based on your specific project structure, build tools, and deployment requirements.
- Consider integrating Jenkins with a version control system and a continuous integration pipeline to automate the build and deployment processes further.