# Jenkins Continuous Integration Admin Guide

This article describes how to install and set up Jenkins for continuous integration (e.g. build automation).

# Expected Outcome

Whenever engineers push out code changes to GitHub or GitLab, Jenkins server will pull the latest copy of our source codes (e.g. FOSS mission operations software, Julia for internal engineering), build the software, run test automation, and publish to the target testing or production machine within minutes without manual intervention.

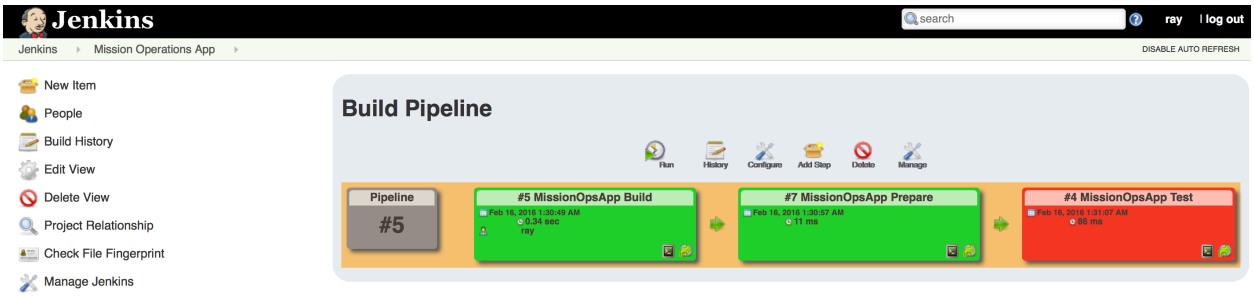
# Overview

### When to Use

Whenever there is any code change, you want to re-build the software codes, and push them to your target system environment automatically.

### Use Case Scenario

Open source developers have checked in their new code changes to https://github.com/audacyDevOps/missionOpsWidgets (Mission Operations widgets). Jenkins server will automatically detect the code changes, start the build job to pull the codes from GitHub (#5 MissionOpsApp Build from the build pipeline screenshot below), retain a copy of the relevant artifacts or binary files (#7 MissionOpsApp Prepare), execute any automated tests (#4 MissionOpApp Test). If these 3 steps are executed successfully, Jenkins will push the final build to the target server host (e.g. production Web server).



### Benefits

- Continuous integration (e.g. test automation)
- Customized workflow by projects

### Jenkins Capabilities

In DevOps context, the entire workflow to pull code changes, run test automation and push final binaries to the target system environment is called "Build Pipeline". In other words, build pipeline is a visual representation of the build and deployment workflow process. Build pipeline is part of the Development Operations capabilities to achieve continuous integration and continuous delivery.

Jenkins server plays the role of workflow orchestration, by defining each workflow step, and customizing scripts and processing logic. With recent new technology updates in DevOps space, Jenkins server is used to:

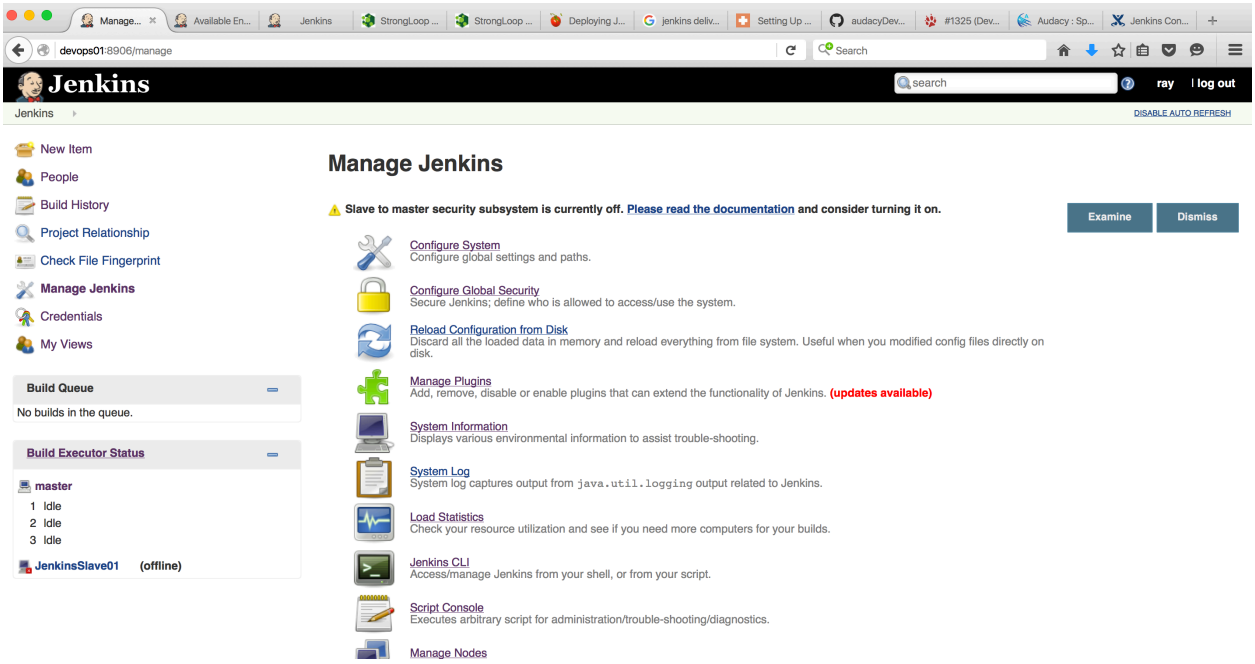
- Integrate with GitHub or GitLab (e.g. execute a build if there is a code change in GitHub)
- Integrate with NodeJS Web server
- Integrate with configuration management tools such as Ansible
- Customize security test automation tools such as w3af

The current scope of Jenkins setup is focused on the integration with GitHub only.

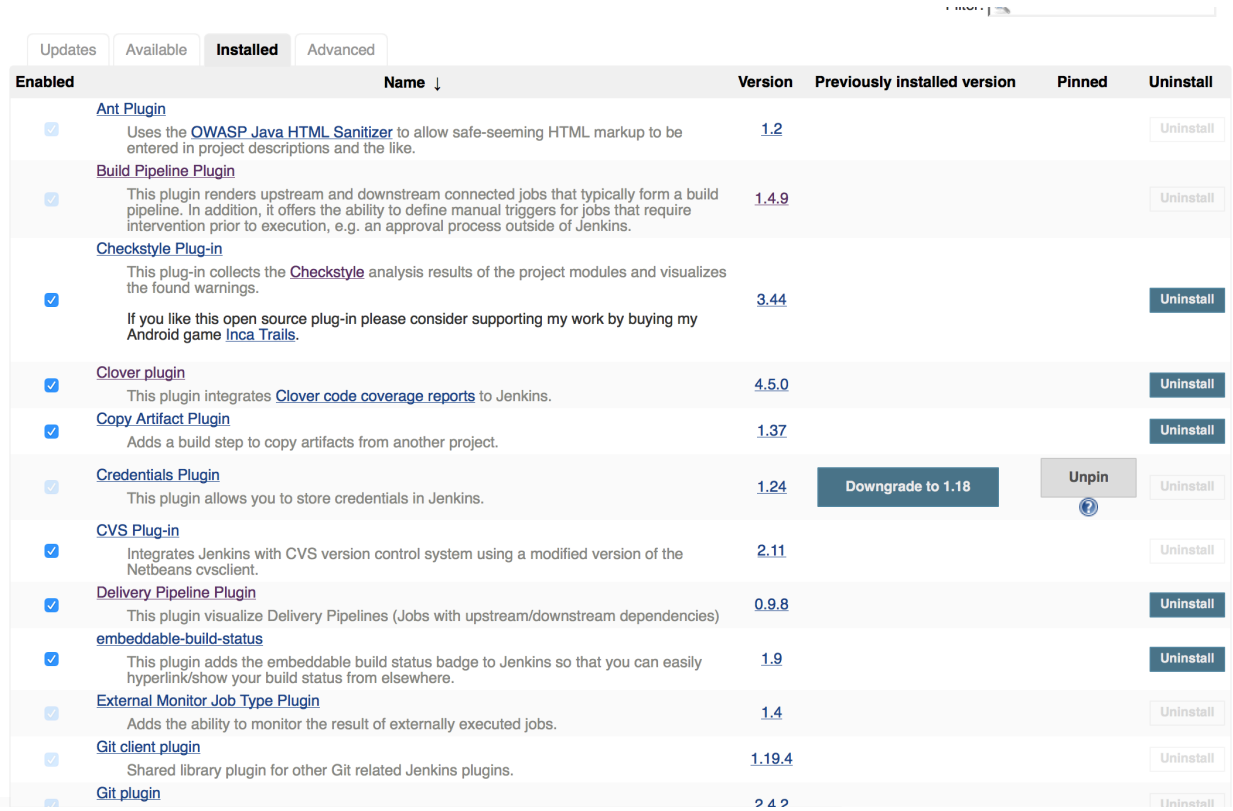
# Software Installation

- Pre-requisites:
    - Create a Linux machine (aka Docker Host) from RackSpace (or Amazon AWS in future).
    - Create a shared storage volume /mnt/data01 and attach to the Linux host. Refer to RackSpace help for details of how to create shared storage volume.
    - Select CentOS 7 which is Redhat Enterprise compatible because it is widely used for enterprise and commercial software deployment, particularly for high availability support.
    - Install Docker RPM package. Refer to https://docs.docker.com/engine/installation/linux/centos/ for details.
    - Assumptions: use "sudo" (root privilege) to execute docker commands.
    - Example: the Linux host has IP address 192.168.1.10. I add this alias devops01 in my /etc/hosts for ease of SSH access. We can create subdomain name later.
- Create a docker instance
    - Issue the command from docker host: "sudo docker run --name jenkins01 -d -p 8906:8080 -p 50000:50000 -v /mnt/data01/jenkins:/var/jenkins_home jenkins"

- This docker command will create a docker container with the name "jenkins01". You can access the Jenkins server via port 8906 of the docker host (Linux machine set up earlier, e.g. 192.168.1.10). Internally, docker will do a port forwarding from port 8906 of the docker host (parent Linux machine) to the port 8080 of the docker container (Jenkins instance). It will store the Jenkins configuration and working data under /mnt/data01/jenkins of the LInux machine (docker host), but the docker container will map this shared storage volume to its local folder /var/jenkins_home.
- From a Web browser, enter the URL http://192.168.1.10:8906. Refer to the following screenshot. There is no user login on startup, and you need to start configuration to enable security. Refer to the section How to Configure Jenkins.




- Install plugins
  - From Jenkins home page, select Manage Jenkins | Manage Plugins.
  - Select the following plugins.
    - Checkstyle
    - Github
    - Github authentication
    - Copy Artifact
    - NodeJS
    - Embeddable build status
    - TAP
    - Build pipeline
    - Delivery pipeline
  - You can select Restart server (by checking install with restart option) later.

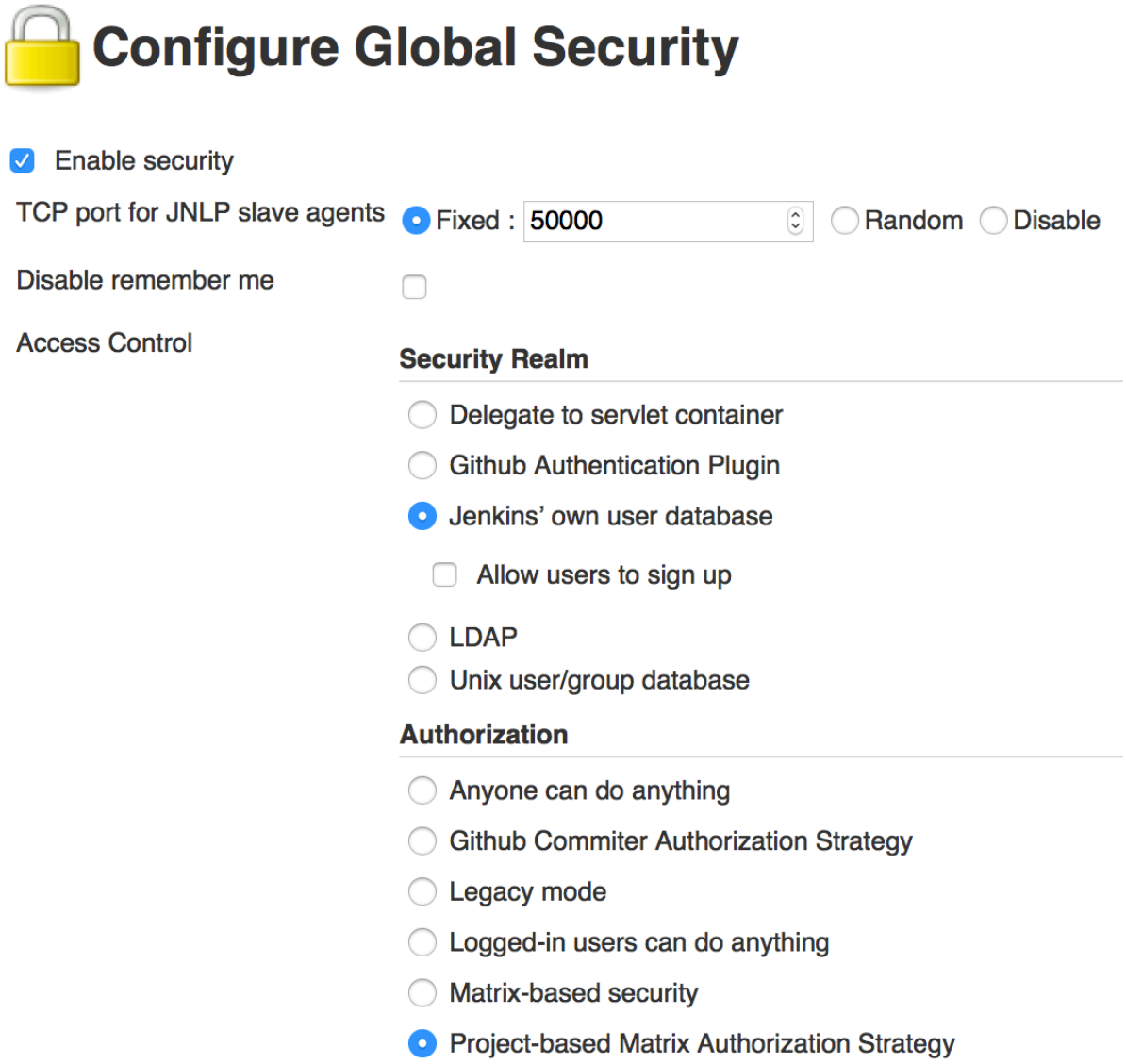


Updates | Available | **Installed** | Advanced

| Enabled | Name ↓ | Version | Previously installed version | Pinned | Uninstall |
|---|---|---|---|---|---|
| ☑ | Ant Plugin<br>Uses the OWASP Java HTML Sanitizer to allow safe-seeming HTML markup to be entered in project descriptions and the like. | 1.2 | | | Uninstall |
| ☑ | Build Pipeline Plugin<br>This plugin renders upstream and downstream connected jobs that typically form a build pipeline. In addition, it offers the ability to define manual triggers for jobs that require intervention prior to execution, e.g. an approval process outside of Jenkins. | 1.4.9 | | | Uninstall |
| ☑ | Checkstyle Plug-in<br>This plug-in collects the Checkstyle analysis results of the project modules and visualizes the found warnings.<br>If you like this open source plug-in please consider supporting my work by buying my Android game Inca Trails. | 3.44 | | | Uninstall |
| ☑ | Clover plugin<br>This plugin integrates Clover code coverage reports to Jenkins. | 4.5.0 | | | Uninstall |
| ☑ | Copy Artifact Plugin<br>Adds a build step to copy artifacts from another project. | 1.37 | | | Uninstall |
| ☑ | Credentials Plugin<br>This plugin allows you to store credentials in Jenkins. | 1.24 | Downgrade to 1.18 | Unpin | Uninstall |
| ☑ | CVS Plug-in<br>Integrates Jenkins with CVS version control system using a modified version of the Netbeans cvsclient. | 2.11 | | | Uninstall |
| ☑ | Delivery Pipeline Plugin<br>This plugin visualize Delivery Pipelines (Jobs with upstream/downstream dependencies) | 0.9.8 | | | Uninstall |
| ☑ | embeddable-build-status<br>This plugin adds the embeddable build status badge to Jenkins so that you can easily hyperlink/show your build status from elsewhere. | 1.9 | | | Uninstall |
| ☑ | External Monitor Job Type Plugin<br>Adds the ability to monitor the result of externally executed jobs. | 1.4 | | | Uninstall |
| ☑ | Git client plugin<br>Shared library plugin for other Git related Jenkins plugins. | 1.19.4 | | | Uninstall |
| ☑ | Git plugin | 2.4.2 | | | Uninstall |

# How to Configure Jenkins Server

- After DevOps engineer launches the docker instance of Jenkins server, enter the URL http://192.168.1.10:8906 from a Web browser.
- Create your admin user id
  - From top of the page, select Sign-up.
  - Create your first admin user id, e.g. audacy.
  - Select a strong password, e.g. at least 8 alpha-numeric characters, mixed with special characters.
  - Optional: You can add a backup user id, or a few more user ids. You can assign them role-based access later without the need to share 1 root password.
- Enable security
  - From main menu (Manage Jenkins), select Configure Global Security, check Enable Security. Select Access Control | Security Realm | Jenkins' own user database.
  - Uncheck "Allow users to sign up" (note: make sure you have all admin user ids created in last step)
  - In future, we can integrate with LDAP (Crowd LDAP server).
  - For Authorization, select Project-based Matrix Authorization Strategy




- Set up role-based access control
  - From Manage Jenkins | Configure Global Security | Enable Security | Access Control | Authorization | Project-based Matrix Authorization Strategy, add your user id.
  - You can grant appropriate access rights. At a minimum, you need to check Overall | Administer for the target user id.
  - You can check "Job | ViewStatus" for the user Anonymous. This allows build status to be displayed on the Web page without relying on specific user permission after login.

**Authorization**

- ○ Anyone can do anything
- ○ Github Commiter Authorization Strategy
- ○ Legacy mode
- ○ Logged-in users can do anything
- ○ Matrix-based security
- ◉ Project-based Matrix Authorization Strategy

| User/group | Overall | | | | | Credentials | | | | | Slave | | | | | | Job | | | | | | | | | | Run | | View | | | | SCM | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Administer | ConfigureUpdateCenter | Read | RunScripts | UploadPlugins | Create | Delete | ManageDomains | Update | View | Build | Configure | Connect | Create | Delete | Disconnect | Build | Cancel | Configure | Create | Delete | Discover | Read | ViewStatus | Workspace | Delete | Update | Configure | Create | Delete | Read | Tag | | |
| audacyAdmin | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☑ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | | |
| ray | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☑ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | | |
| Anonymous | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☑ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | | |

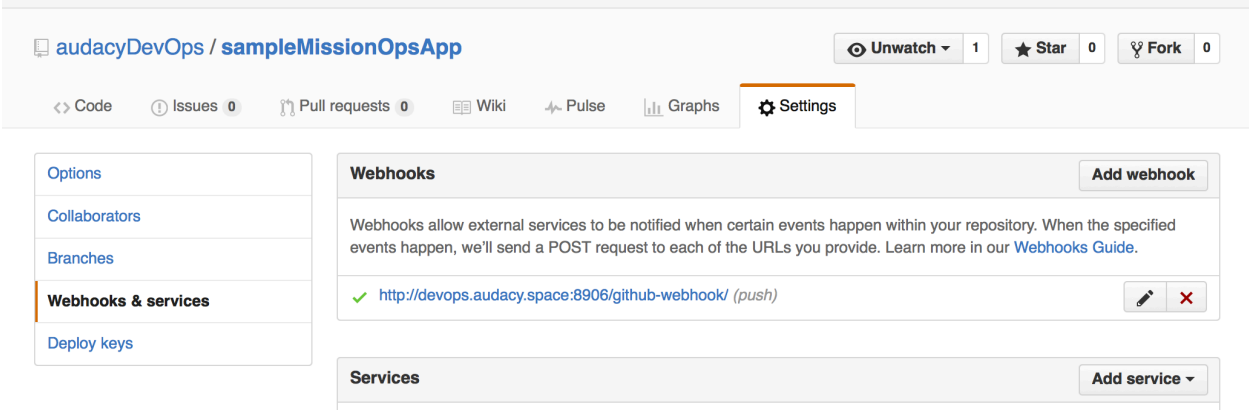
User/group to add: ralph [Add]

Refer to https://strongloop.com/strongblog/roll-your-own-node-js-ci-server-with-jenkins-part-1/ for details.

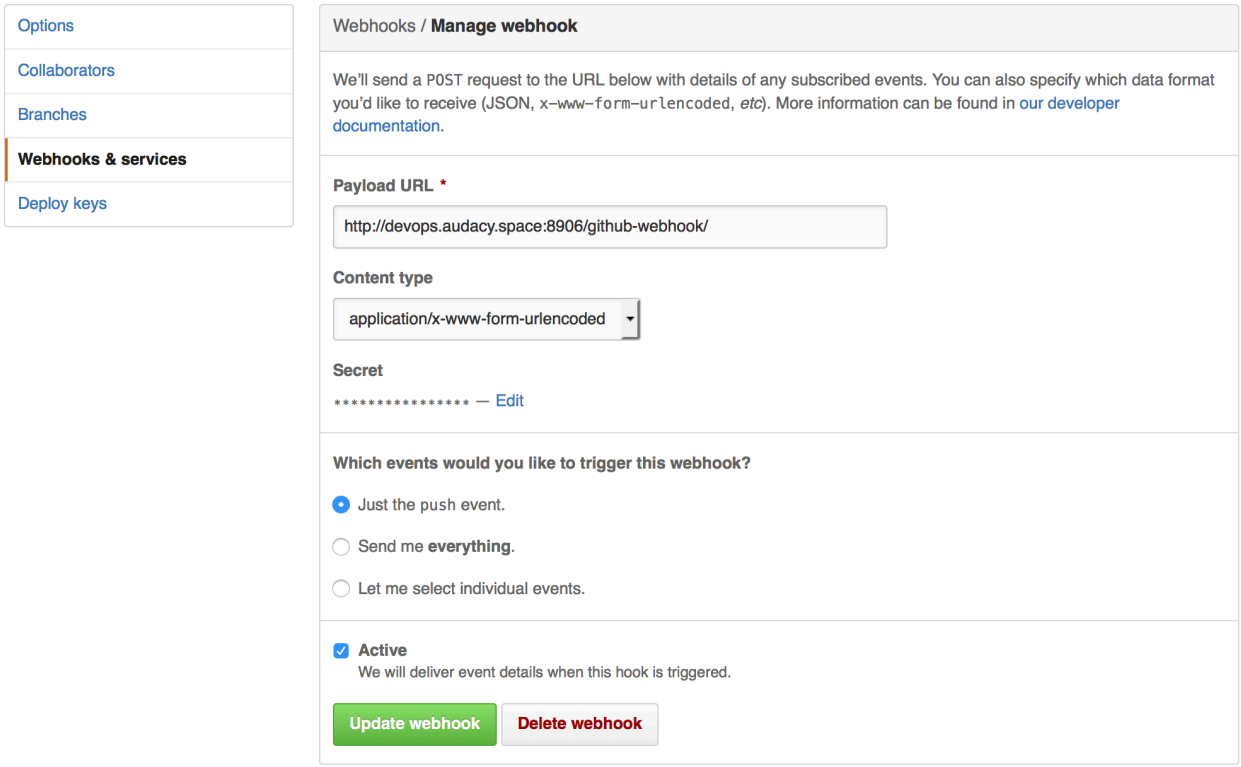
# How to Set Up Trigger in GitHub

In order for Jenkins to kick off any build job right after developers check in codes to GitHub, we need to configure GitHub Webhook. This is a followup task after Jenkins has configured GitHub hook/integration.

- Login to GitHub
- Pre-requisite: you create a personal access token (aka OAuth access token) for your Jenkins for access. Refer to previous section on Jenkins configuration.
- Select your GitHub repo (project), select Settings tab | Webhooks & services (do not confuse with the overall repository settings, which is an icon on top right corner of the GitHub home page)




- Click Add webhook.
    - Enter the name of your Jenkins server, with the suffix "/github-webhook/". In our case, http://devops.audacy.space:8906/github-webhook/
    - Copy the personal access token to the field "Secret". This is the security token that you generated from earlier step when setting up GitHub and Jenkins GitHub plugin.




# How to Set up Build Pipeline

This section explains how to configure Jenkins to set up a build pipeline to automatically pull new codes changes from GitHub, and push to a shared folder that is used by your target test Web server. It is based on a single docker host scenario. We need separate documentation for the multiple docker hosts scenario, where not all your target production servers can share the same shared folder.

## Sample Scenario #1

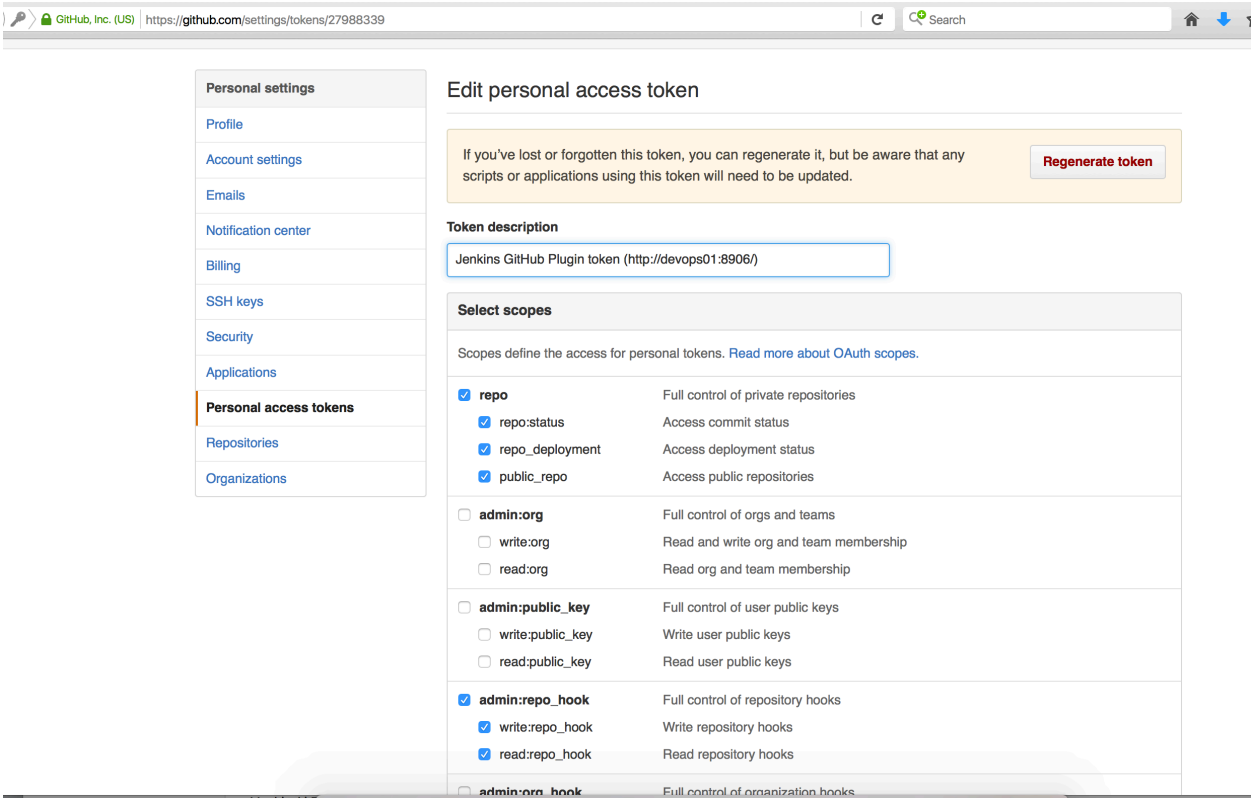
- Developers make code changes in GitHub (e.g. Open Source FOSS mission operations project)
- You want Jenkins to automatically pull the code changes, and to copy the codes to the target Web servers

## Sample Scenario #2

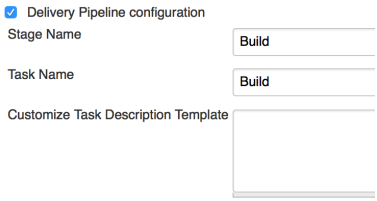
- Audacy engineers complete orbit simulation codes using Julia in GitLab private project.
- You want Jenkins to automatically pull the Julia codes, test it with automated test scripts, and push the codes to the target Julia server.

## Build Pipeline Instructions

- Pre-requisites - set up GitHub
    - Create GitHub repository (or GitLab private project)
    - Create personal access token in GitHub (from top right menu, select Settings icon. Then pick Personal settings | Personal access tokens) for Jenkins to use. When you create personal access token, check the scope for (1) repo, (2) admin repo hook. Refer to the screenshot example below.




| Steps | Purpose | Instructions | |
|---|---|---|---|
| 1 | Design how many steps you need for your GitHub project | N/A | e.g. Git pull -> Build -> Copy |
| 2 | Create GitHub personal access token | login to GitHub<br>From Personal settings create a personal access token.<br>Select Scopes for (1) repo, (2) admin:repo hook | Refer to the GitHub screensh |




3 Create #1 Build step

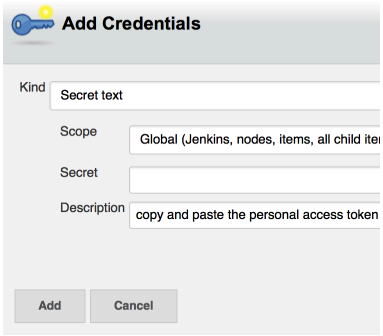
From Jenkins home page, click New Item.

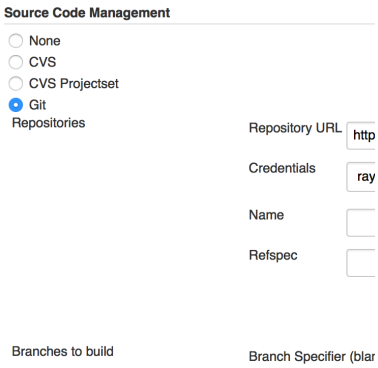
Edit description to describe this step for pulling code changes from GitHub to build the application.

Check Delivery Pipeline configuration, and enter the stage name and task name as "Build".

Under Source Code Management section, select Git. Enter the repository URL for your GitHub repo. For credentials, select Add to add your personal access token (you need to specify "Secret Text" from the Add Credentials pop-up box).

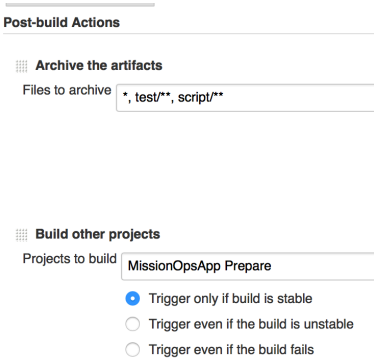
By default, your GitHub will pull from your master unless you replace the "Branch to build" with a different branch name.



Under the section Post-build actions, add 2 steps:

1. Archive the artifacts - this step will retain a copy of your previous build step (which gets a copy of GitHub codes in the workspace temporarily)
2. Build other projects - this will start the next task (#2 Prepare)



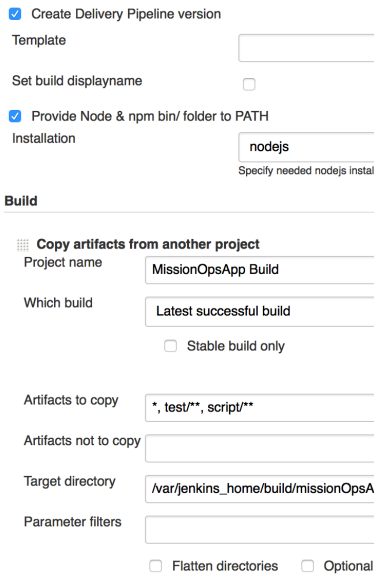

4 Create #2 Prepare

From Jenkins home page, click New Item.

Edit description to describe this step for pulling code changes from GitHub to copy the GitHub codes to the target shared folder.

Create build steps:

- Check "Create Delivery Pipeline version" (the status will show up in Delivery pipeline view).
- Under Build section, select Copy artifacts from another project. Choose the project name from step #1 (e.g. MissioniOpsApp Build). Specify which files you want to copy, and the target folder.




In this example, the folder /var/jenkins_home/build/missionOpsApp is a shared folder, which is or will be used by production Web server.

5 Create #3 Test

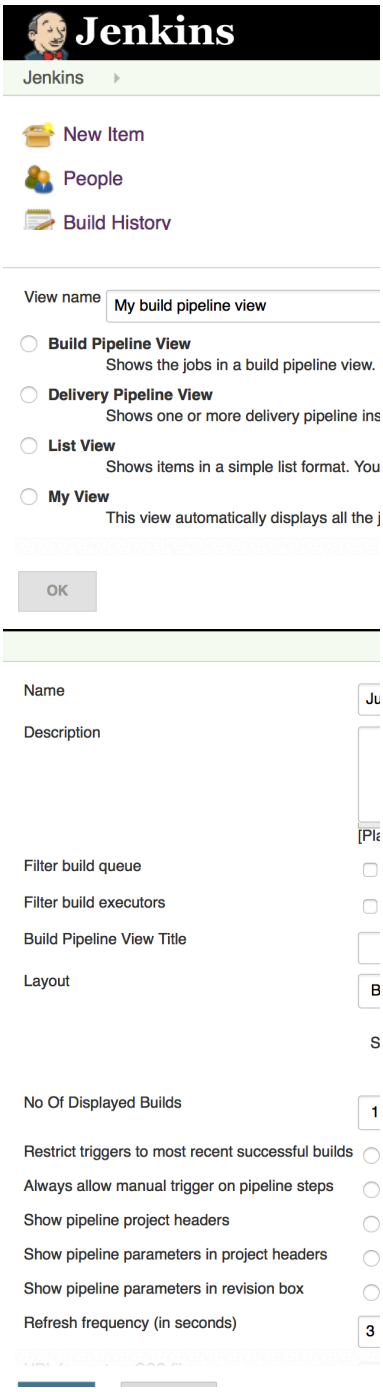
From Jenkins home page, click New Item.

Edit description to describe this step for executing test automation scripts.

Create build steps:

- Execute shell to run test scripts, e.g. ./test.sh.

| | | |
|---|---|---|
| 6 | Create Build Pipeline View | From Jenkins home page, click the "+" icon next to the tab All.<br><br>Create a build pipeline view by specifying the name, and check "Build Pipeline View". |




Add description for your build pipeline.

Under the item "Layout" | Select Initial Job, select your tasks or jobs. you can add more jobs later.

Your build pipeline should look something like:


# Open Items

The following issues are either "followup tasks" or "open questions" to be addressed. They may not be any immediate issues.

- When Jenkins build software apps after pulling from GitHub, it will write to a shared folder. For demo, we set up the shared folder /mnt/data01 to be world writeable. This is NOT appropriate for production use. Followup actions are:
    - Create docker containers with appropriate R/W access rights for the shared folder
    - We may re-configure the docker command to be non-root, and make the user docker to have write/read access to the shared folder only
- Next step is to set up nginx as reverse proxy, and haproxy as load balancer.