Microservices Continuous Deployment Tutorial

Table of Contents

[Objective 3](#_Toc450574596)

[Conventions Used in This document 3](#_Toc450574597)

[Continuous Deployment Overview 4](#_Toc450574598)

[Continuous Deployment Cycle 5](#_Toc450574599)

[Microservices Continuous Deployment 6](#_Toc450574600)

[Microservices Continuous Deployment Tools 7](#_Toc450574601)

[Version Control (Git) 7](#_Toc450574602)

[Continuous Integration Software (Jenkins) 7](#_Toc450574603)

[Unit Test & Code Coverage Report 8](#_Toc450574604)

[Application Deployment (Docker) 8](#_Toc450574605)

[Notification (Slack) 8](#_Toc450574606)

[Version Control (Git) 9](#_Toc450574607)

[Gitflow Model 10](#_Toc450574608)

[Main branch 11](#_Toc450574609)

[Supporting branches 11](#_Toc450574610)

[Github 12](#_Toc450574611)

[Github & Jenkins Integration 12](#_Toc450574612)

[Continuous Integration Software (Jenkins) 13](#_Toc450574613)

[Login to Jenkins 13](#_Toc450574614)

[Create New Project Job – Freestyle project 15](#_Toc450574615)

[Step 1 – Setting project name & job type 15](#_Toc450574616)

[Step 2 – Setting project-based security (optional setting) 15](#_Toc450574617)

[Step 3 – Setting Source Code Management details 16](#_Toc450574618)

[Step 4 – Setting the Build Trigger 16](#_Toc450574619)

[Step 5 – Setting Build Step 18](#_Toc450574620)

[Step 6 – Setting Post-build Actions 19](#_Toc450574621)

[Create New Project Job – Freestyle multi-branch project 22](#_Toc450574622)

[Step 1 – Setting project name & job type 22](#_Toc450574623)

[Step 2 – Branch Management 23](#_Toc450574624)

[Step 3 – Per Branch Configuration (optional settings) 24](#_Toc450574625)

[Step 4 – Sync Branches (Automatic process by Jenkins) 26](#_Toc450574626)

[Unit Test & Code Coverage Report 27](#_Toc450574627)

[Setting sbt-scoverage plugin in project 27](#_Toc450574628)

[Setting Code Coverage in Jenkins 27](#_Toc450574629)

[Application Deployment (Docker) 30](#_Toc450574630)

[Basic Deployment to Docker 30](#_Toc450574631)

[Deployment to Docker with LM script 31](#_Toc450574632)

[Notification (Slack) 33](#_Toc450574633)

[Install Jenkins CI in Slack (One time setup) 33](#_Toc450574634)

[Add New Configuration for Jenkins 34](#_Toc450574635)

[References 40](#_Toc450574636)

[Continuous Deployment Concept 40](#_Toc450574637)

[Jenkins 40](#_Toc450574638)

[Git 40](#_Toc450574639)

[Slack 40](#_Toc450574640)

[Unit Test & Coverage Code 40](#_Toc450574641)

[Docker 40](#_Toc450574642)

# Objective

To overview about continuous deployment for Microservices project. This document is the starter guide how to integrate tools to support the automatic process of software deployment with Jenkins.

# Conventions Used in This document

The following typographical conventions are used in this tutorial:

Gray text

Indicate the key terms, page name, or specific topic in the screenshot.

Constant width

Indicate the programming code, script, or command line.

[Blue color with underline](http://localhost)

Indicate the hyperlink. The text is clickable to browse to the related content or website.

or (green-color arrow)

Mean that the process is generated automatically

or (orange-color arrow)

Mean that the process is generated manually

# Continuous Deployment Overview

Continuous deployment is related to continuous integration and delivery to provide the faster development cycles, and getting code written, integrated, tested, and deployed as quickly as possible.

#### Continuous Integration

Continuous Integration is a software development practice in which you build and unit-test software every time a developer checks in new code.

#### Continuous Delivery

Continuous Delivery (CD) is a software development practice in which continuous integration, automated testing, and automated deployment capabilities allow software to be developed and deployed rapidly, reliably and repeatedly with minimal manual overhead.

**Application Test**

**Dev**

**Acceptance**

**Test**

**Integration**

**Test**

**Production**

Auto

Auto

Auto

Manual

**Continuous Delivery**

#### Continuous Deployment

Continuous Deployment is a software development practice in which every code change goes through the entire pipeline and is put into production, automatically, resulting in many production deployments every day.

**Application Test**

**Dev**

**Acceptance**

**Test**

**Integration**

**Test**

**Production**

Auto

Auto

Auto

Auto

**Continuous Deployment**

The benefits of deploy automation are:

* Deployment modeling controls variability and reduces errors, resulting in higher product quality
* Automated and accelerated product release processes lead to on-time and frequent product releases
* Consolidated access to all tools, process and resource data leads to faster troubleshooting
* Effective collaboration between Dev, QA, and Ops teams leads to higher quality and higher customer satisfaction
* Centralized view of all deployment activities and outcomes leads to faster and lower effort audits

# Continuous Deployment Cycle



**Pass**

**Test  
Build/Packing  
Deploy**

push code changes



Jenkins detect new commit  
and pull latest code



deliver to servers



notify testers,  
project managers, etc.

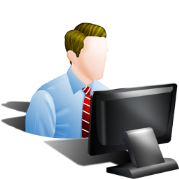
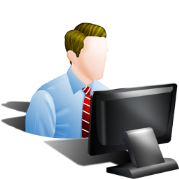




Jenkins run test  
generate test report  
build / package  
deploy

notify  
developer  
team





Dev

webhook



**Fail**

# Microservices Continuous Deployment



**Jenkins  
(Master / Slaves)**



**Microservice A**

**Microservice B**



**Microservice C**



**Running Servers**



**Docker Cluster**



**Docker Single Node**



**Server Clusters**

Developer  
push code changes  
to  
Version Control (Git)

Jenkins detect code changes with webhook  
and pull latest code from Git

Jenkins run test, build, package, deploy  
according to the job configuration

Jenkins can deploy the project into various containers or server environments according to the job configuration.

* Deploy to tomcat server
* Deploy to apache httpd server

We are also able to set Jenkins job to deploy a project into many targets in the same time.

# Microservices Continuous Deployment Tools

Following are the software that we refer in this document for Continuous Deployment of microservices project. There are also many alternative tools or further useful software to integrate to Jenkins.

* Version Control : Git
* Continuous Integration Software : Jenkins
* Unit Test & Code Coverage Report : sbt-scoverage
* Application Deployment : Docker
* Notification : Slack

## Version Control (Git)

Git is a version control system that is widely used for software development. It is a distributed revision control system with an emphasis on speed, data integrity, and support for distributed, non-linear workflows.

Developer commits the code written or code changed and push to remote repository. When Jenkins detect the new commit, it would add queue to pull the latest code, test, build and deploy according to the configuration to its job.

## Continuous Integration Software (Jenkins)

Jenkins is an open source continuous integration tool written in Java. The project was forked from Hudson after a dispute with Oracle.

Jenkins provides continuous integration services for software development. It is a server-based system running in a servlet container such as Apache Tomcat. It supports SCM tools including AccuRev, CVS, Subversion, Git, Mercurial, Perforce, Clearcase and RTC, and can execute Apache Ant and Apache Maven based projects as well as arbitrary shell scripts and Windows batch commands.

##### Recommended Configuration to a job in Jenkins

* Authorize User if enable project-based security
* Git repository & Branch
* Build Step
  + Test
  + Run Scoverage report
  + Build & Package
* Deploy
* Notification

##### General Deployment Pipeline

1. Jenkins automatically check for new commit with Git webhook
2. Pull the latest code
3. Download dependency
4. Run Unit Test
5. Build
6. Publish Scoverage Report
7. Deploy
8. Send notification of build result

## Unit Test & Code Coverage Report

It is required to write the unit test for each microservice. Specs2 and Scalacheck is the recommended library for automated testing for Scala project.

Scoverage is the tool to check the coverage of unit test on the project. Unit Test is very important for continuous deployment to make sure that the software can work properly after it’s automatically deployed. If the test is failed, Jenkins would stop and don’t deploy to the production.

## Application Deployment (Docker)

Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating-system-level virtualization on Linux.

Generally Jenkins can deploy the software to many environment or servers by using its plugins or shell script. Our Microservices Foundation Template provide the script to create docker image and run it as container both single node or cluster.

## Notification (Slack)

Slack is a cloud-based team collaboration tool. Slack provides mobile apps for iOS, Android, and Windows Phone in addition to their web browser client and native desktop clients for Mac OS X, Windows, and Linux (beta). Slack is also available for the Apple Watch, allowing users to send direct messages, see mentions, and make simple replies.  
  
Slack provide the secure way to integrate with Jenkins to enable the notification of build status.

# Version Control (Git)

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. Git is a Distributed Version Control Systems (DVCSs). Git clients don’t just check out the latest snapshot of the files, they fully mirror the repository. Every clone is really a full backup of all the data. With Git, you can collaborate with different groups of people in different ways simultaneously within the same project. This allows you to set up several types of workflows.

|  |  |
| --- | --- |
| https://upload.wikimedia.org/wikipedia/commons/1/1e/Curation_bar_icon_info_35x35.png | Git Documentation & Tutorials:  <https://git-scm.com/doc>  <https://www.atlassian.com/git/tutorials>  <https://try.github.io>  <http://pcottle.github.io/learnGitBranching> |

## Gitflow Model

Gitflow Model is the agreed standard workflow to use Git in microservice development.

The purpose of this branching model is to provide easy development and test codebases that leads to convenient deployment on various server environments.



### Main branch

* master = production-ready state
* develop = latest delivered development changes for the next release

When the source code in the develop branch reaches a stable point and is ready to be released, all of the changes should be merged back into master somehow and then tagged with a release number.

Therefore, each time when changes are merged back into master, this is a new production release. We could use a Git hook script or Jenkins to automatically build and roll-out our software to our production servers once there is a new commit on master.

### Supporting branches

* Feature branches branch of from: develop merge back into: develop
* Release branches branch of from: develop merge back into: develop & master
* Hotfix branches branch of from: master merge back into: develop & master

##### Feature branches

Feature branches are used to develop new features for the upcoming or a future release. The essence of a feature branch is that it exists as long as the feature is in development, but will eventually be merged back into develop (to definitely add the new feature to the upcoming release) or discarded (in case of a disappointing experiment).

##### Release branches

Release branches support preparation of a new production release. The key moment to branch off a new release branch from develop is when develop (almost) reflects the desired state of the new release. At least all features that are targeted for the release-to-be-built must be merged in to develop at this point in time. All features targeted at future releases may not—they must wait until after the release branch is branched off.

##### Hotfix branches

When a critical bug in a production version must be resolved immediately, a hotfix branch may be branched off from the corresponding tag on the master branch that marks the production version.

The essence is that work of team members (on the develop branch) can continue, while another person is preparing a quick production fix.

## Github

GitHub is a web-based Git repository hosting service. It offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features.

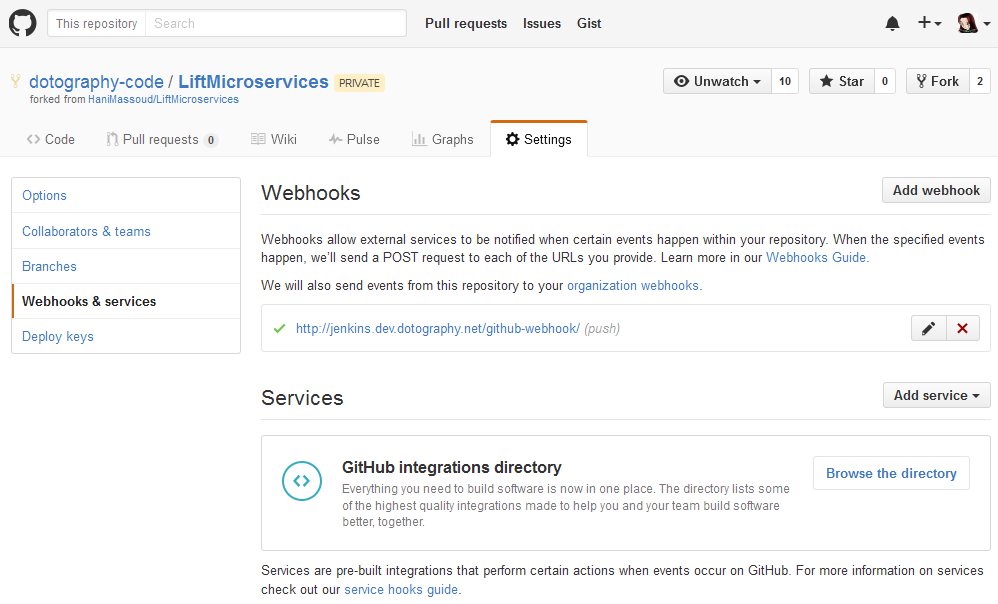
Github Site: <https://github.com/dotography-code>

It also provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.

## Github & Jenkins Integration

Github provide Webhook service to allow external service to be notified when certain events happen within a repository e.g. when there is a new commit, github will send a POST request to the specified URL.

After first pull source successful with Jenkins, the URL will be updated in Github Webhooks page automatically as screenshot:



Jenkins Github plugin provides integration of Jenkins with Github repository. Following are its functionalities:

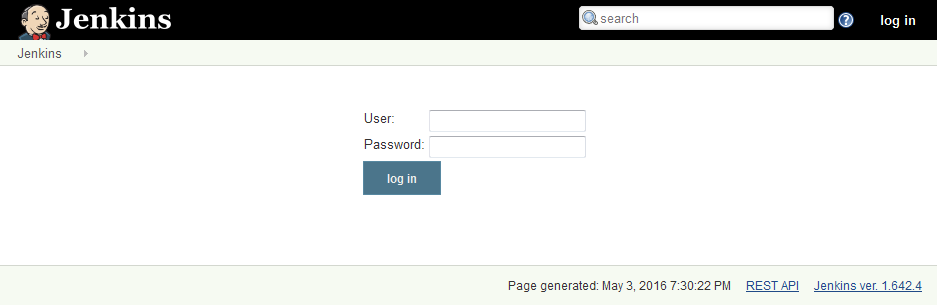
* Create hyperlinks between Jenkins projects and GitHub
* Trigger a job when a change is pushed to GitHub, notified by Github webhook
* Report build status result back to github as Commit Status (optional setting)

# Continuous Integration Software (Jenkins)

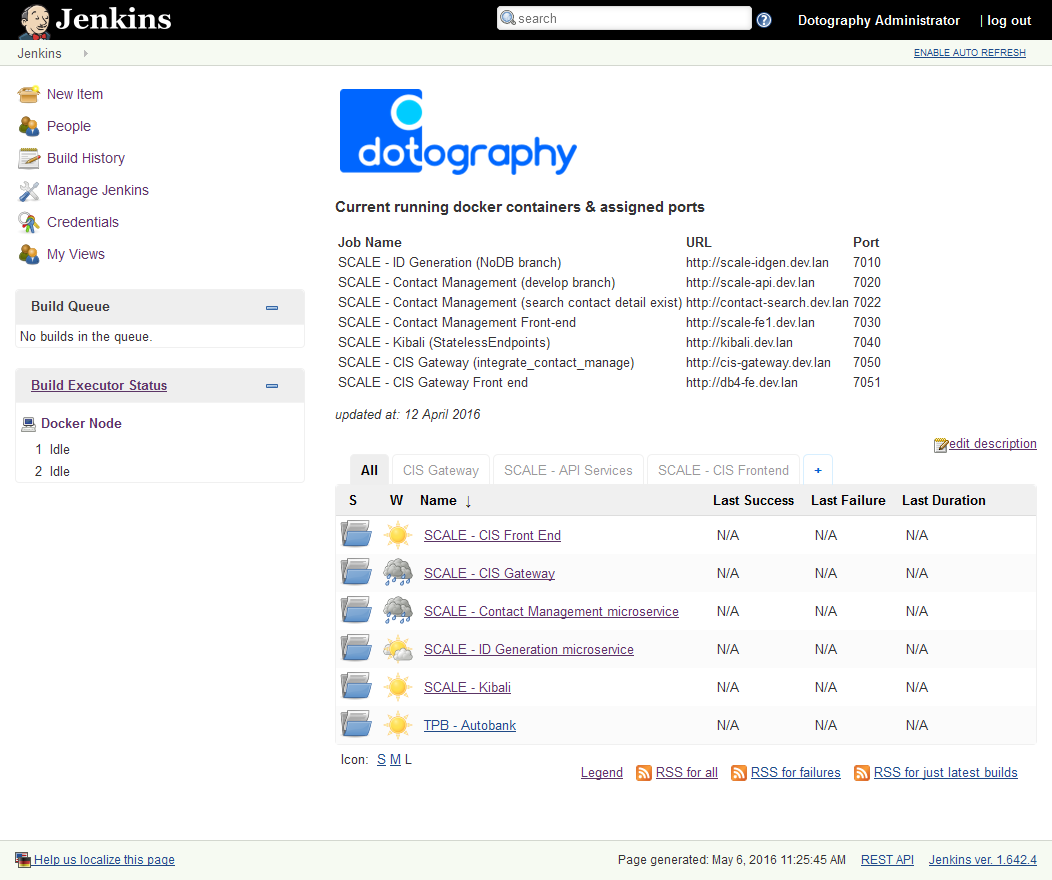
Jenkins is a powerful application that allows continuous integration and continuous delivery of projects, regardless of the platform you are working on. It is a free source that can handle any kind of build or continuous integration. You can integrate Jenkins with a number of testing and deployment technologies. In this tutorial, we would explain how you can use Jenkins to build and test your software projects continuously.

Dotography Jenkins site: <http://jenkins.dev.dotography.net>

## Login to Jenkins



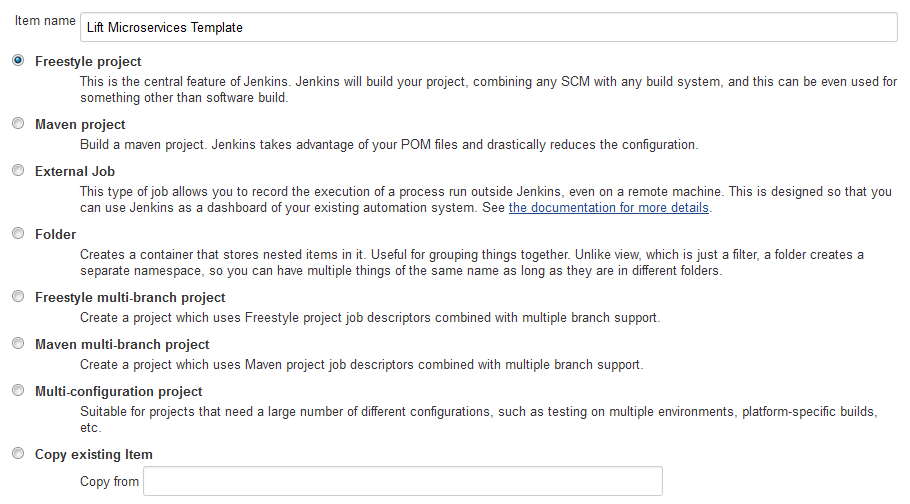
After login successfully, default landing page is Jenkins Dashboard



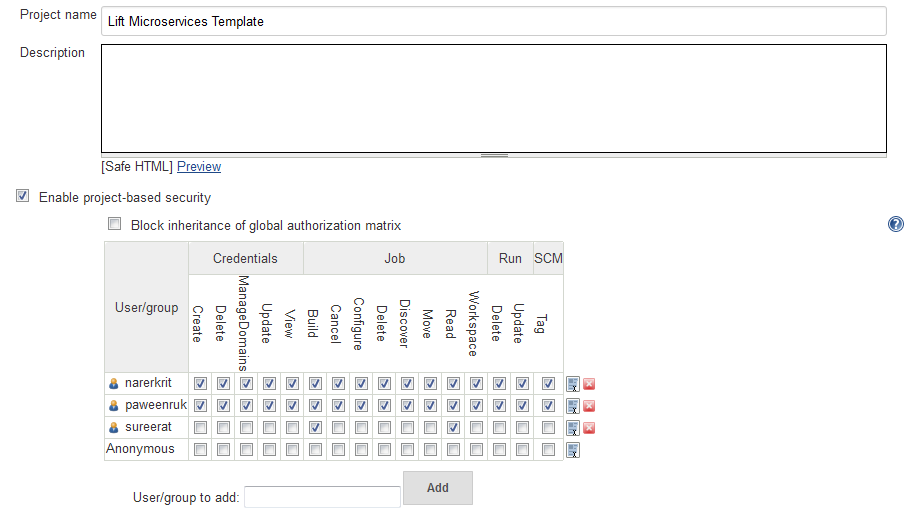
|  |  |
| --- | --- |
|  |  |

## Create New Project Job – Freestyle project

### Step 1 – Setting project name & job type



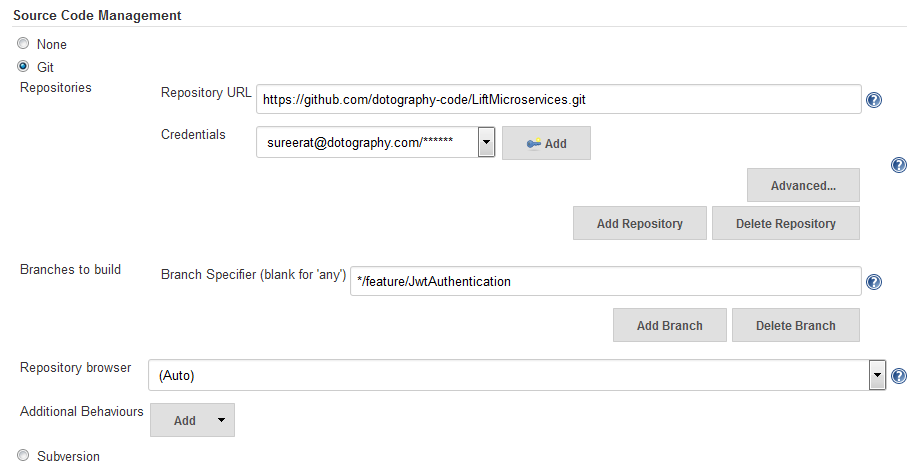
### Step 2 – Setting project-based security (optional setting)



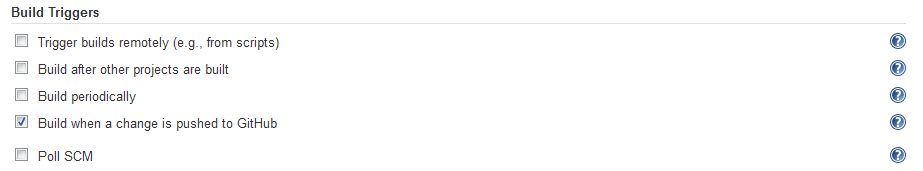
If this part is not enabled, it would apply the global security setting to this project.

### Step 3 – Setting Source Code Management details

* Git Repository URL
* Credentials
* Branch



### Step 4 – Setting the Build Trigger



#### Available Build Triggers:

##### Trigger builds remotely (e.g., from scripts)

This option is for trigger new builds by accessing a special predefined URL (convenient for scripts).

One typical example for this feature would be to trigger new build from the source control system's hook script, when somebody has just committed a change into the repository, or from a script that parses your source control email notifications.

You'll need to provide an authorization token in the form of a string so that only those who know it would be able to remotely trigger this project's builds.

##### Build after other projects are built

Set up a trigger so that when some other projects finish building, a new build is scheduled for this project. This is convenient for running an extensive test after a build is complete, for example.

This configuration complements the Build other projects section in the Post-build Actions of an upstream project, but is preferable when you want to configure the downstream project.

##### Build periodically

Provides a cron-like feature to periodically execute this project.

This feature is primarily for using Jenkins as a cron replacement, and it is not ideal for continuously building software projects. When people first start continuous integration, they are often so used to the idea of regularly scheduled builds like nightly/weekly that they use this feature. However, the point of continuous integration is to start a build as soon as a change is made, to provide a quick feedback to the change. To do that you need to hook up SCM change notification to Jenkins.

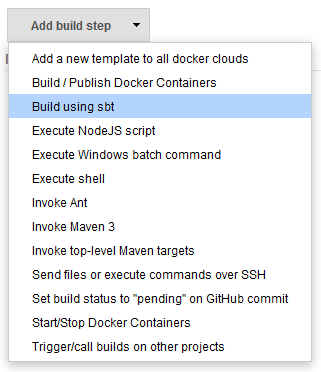
##### Build when a change is pushed to GitHub

This job will be triggered if Jenkins receive PUSH GitHub hook from repository defined in Source Code Management section

##### Poll SCM

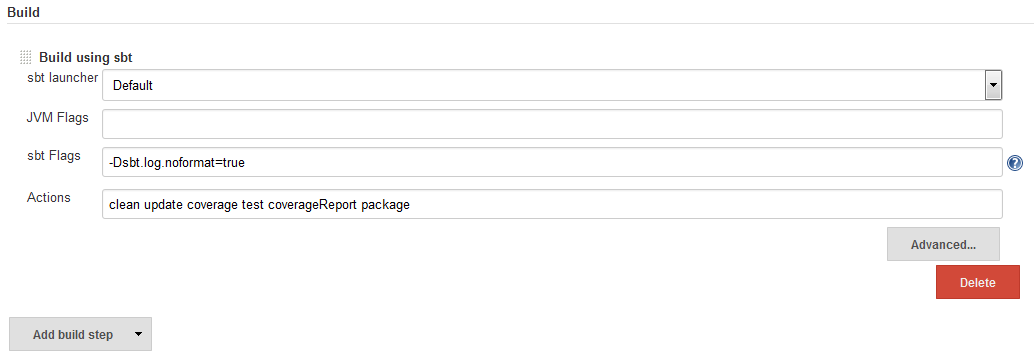
Configure Jenkins to poll changes in SCM

### Step 5 – Setting Build Step

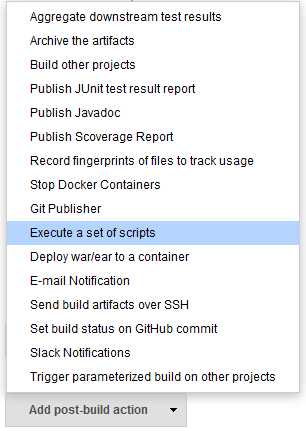


##### Sample Build Step - Build using sbt

1. clean = Clean all previous build in project
2. update = Force update dependency
3. coverage = Enable coverage
4. test = Run Unit Test
5. coverageReport = Generate coverage reports
6. package = Build project



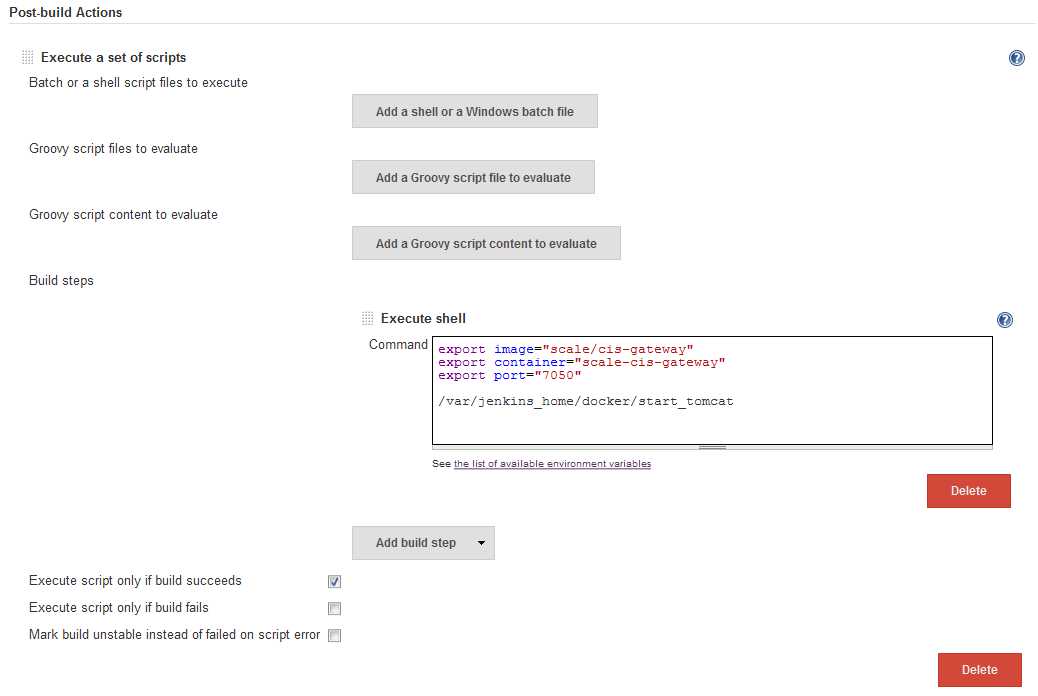
### Step 6 – Setting Post-build Actions



##### Sample Post-build Action – Execute a set of scripts

The sample is to deploy the project into tomcat docker container:

* create docker image by adding the build package
* run the docker image as container with the specified port



Sample shell command to deploy package to docker of tomcat server

export image="scale/cis-gateway"  
export container="scale-cis-gateway"  
export port="7050"

/var/Jenkins\_home/docker/start\_tomcat

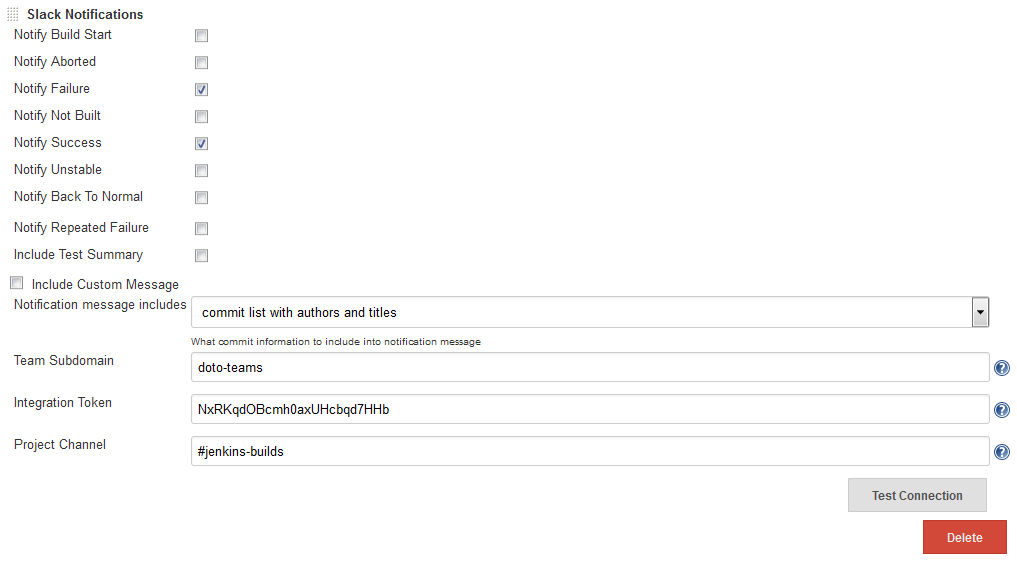
The commands set would perform the following actions:

1. create docker image named scale/cis-gateway
2. Run docker container named scale-cis-gateway from image scale/cis-gateway
   * Run tomcat server
   * Run at port 7050

##### Sample Post-build Action – Slack Notifications

Following is the integration settings to Slack:

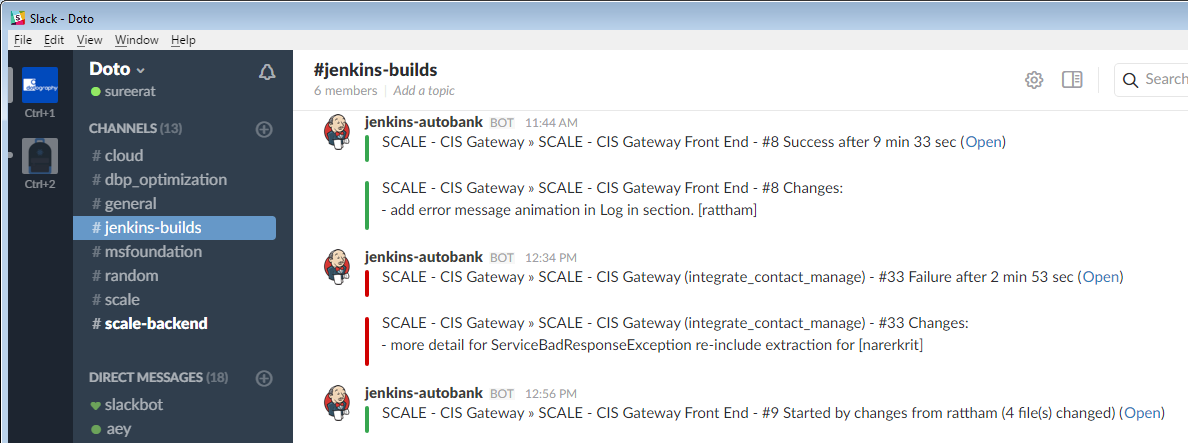
* Team Subdomain
* Integration Token
* Project Channel



##### Screenshot of messages that Jenkins send to Slack

Build Success message

Team Subdomain – doto-teams

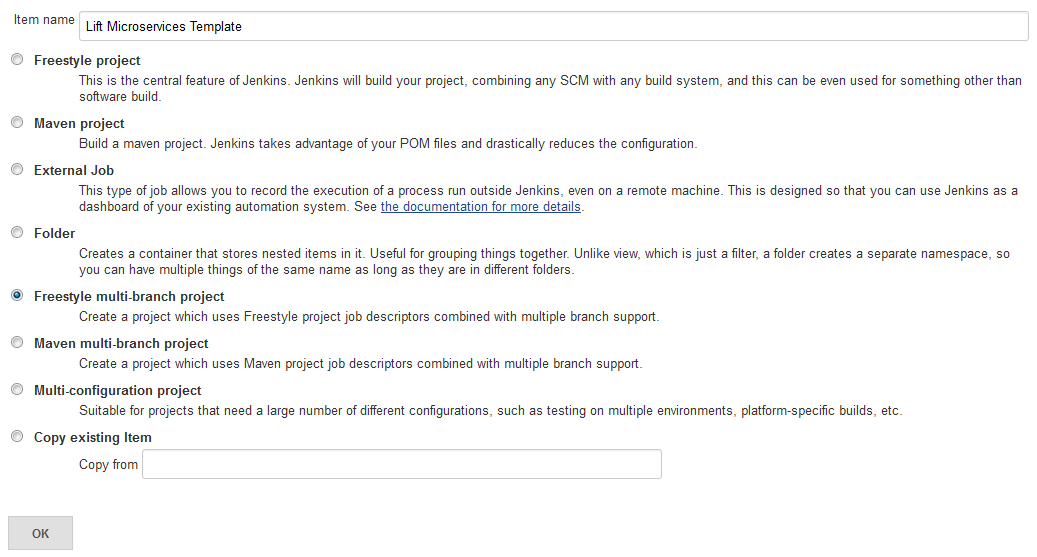


Build Fail message

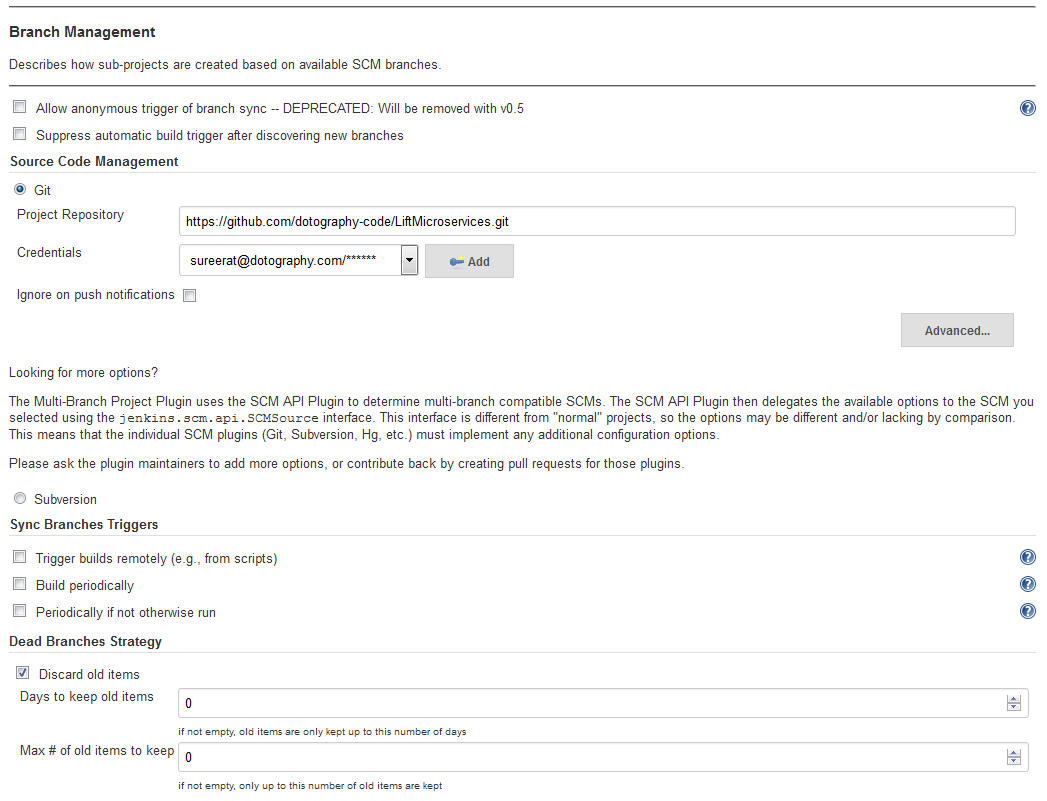
Project Channel - #jenkins-builds

## Create New Project Job – Freestyle multi-branch project

### Step 1 – Setting project name & job type



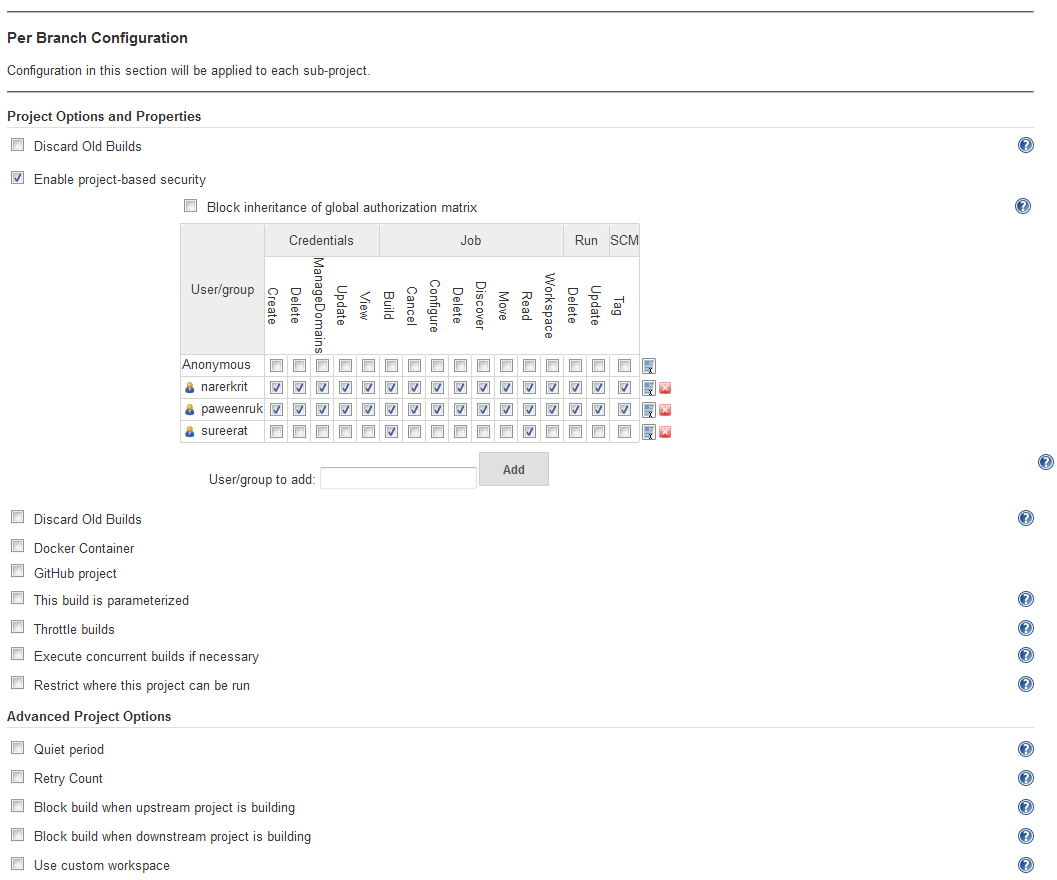
### Step 2 – Branch Management



### Step 3 – Per Branch Configuration (optional settings)

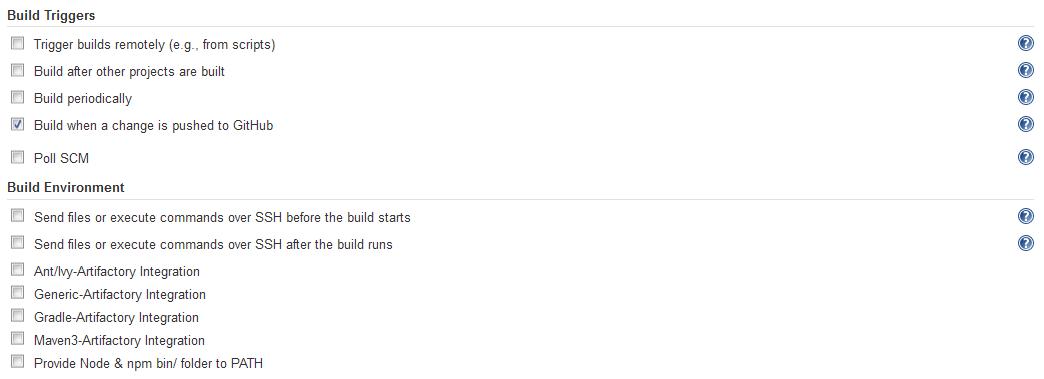
This will set as default configuration for each branch. If skip setting this section, you can set these configurations individually one by one after Jenkins sync all branch successfully.

#### Setting project-based security (optional setting)

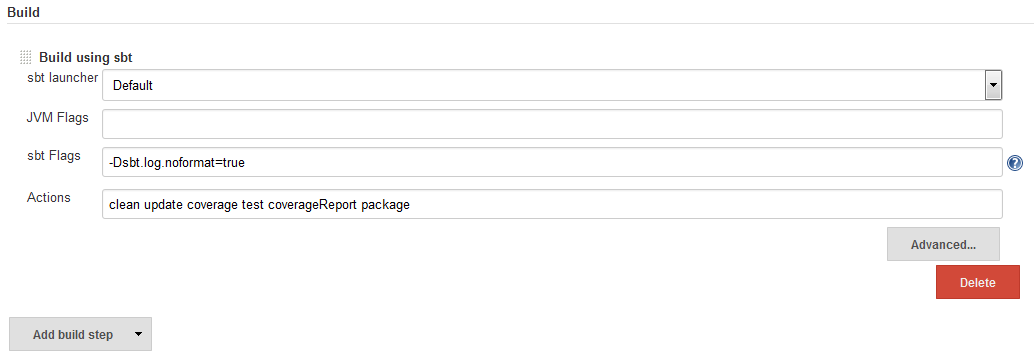


If this part is not enabled, it would apply the global security setting to this project.

#### Setting the Build Trigger



#### Setting Build Step



Please refer to [**Create New Project Job – Freestyle project > Step 5 – Setting Build Step**](#_Step_5_–) for the details about Build Step

|  |  |
| --- | --- |
| https://upload.wikimedia.org/wikipedia/commons/1/1e/Curation_bar_icon_info_35x35.png | If you don’t want to have Jenkins build to every branch, it’s recommended to skip these settings and do it for each branch individually after Jenkins sync branches successfully. |

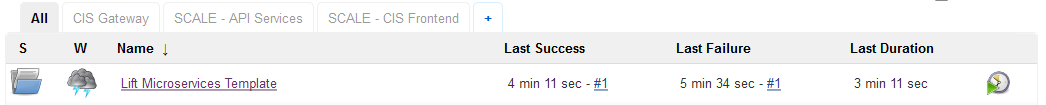
#### Setting Post-build Actions



|  |  |
| --- | --- |
| https://upload.wikimedia.org/wikipedia/commons/1/1e/Curation_bar_icon_info_35x35.png | It’s recommended to skip these settings and do it for each branch individually after Jenkins sync branches successfully. |

### Step 4 – Sync Branches (Automatic process by Jenkins)

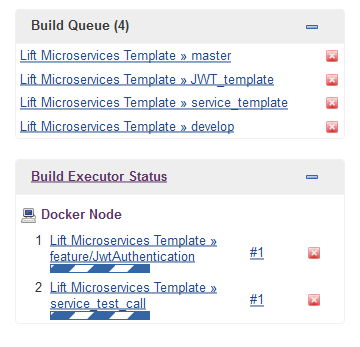
After you save the project, Jenkins will create the project as folder.



Then Jenkins will start sync branches and build each job according to your settings. If you don’t configure the “Build Step”, Jenkins will just sync branches.

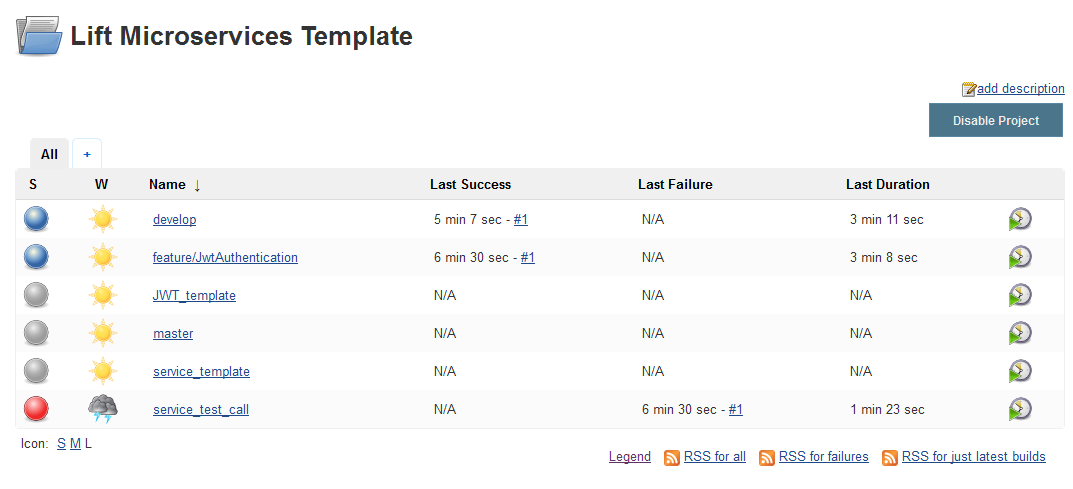
Pending Job in Queue

Jobs that is being built



Occurs at Sidebar  
under  
navigation menu

After Jenkins sync branch successfully, it would create each branch as a job and you can configure each job individually. Please refer chapter [**Create New Project Job – Freestyle project**](#_Create_New_Project) on how to configure.



# Unit Test & Code Coverage Report

Code coverage is a measure used to describe the degree to which the source code of a program is tested by a particular test suite. A program with high code coverage has been more thoroughly tested and has a lower chance of containing software bugs than a program with low code coverage. Many different metrics can be used to calculate code coverage; some of the most basic are the percent of program subroutines and the percent of program statements called during execution of the test suite.

## Setting sbt-scoverage plugin in project

sbt-scoverage is a plugin for SBT that integrates the scoverage code coverage library.

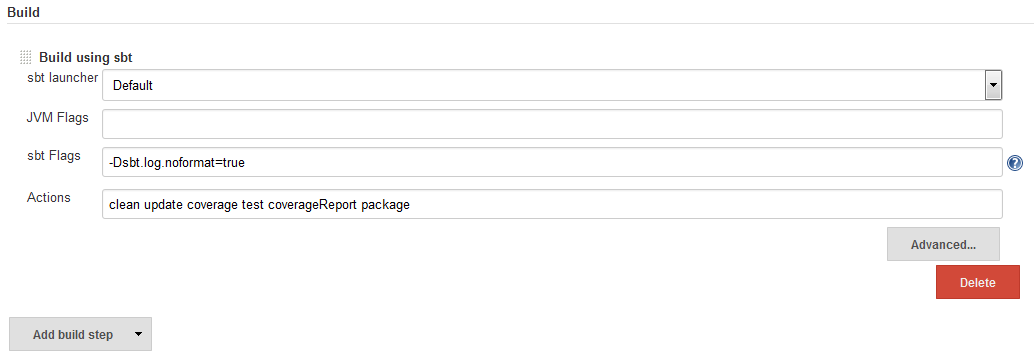
At the project directory, edit the following files:

|  |
| --- |
| project/plugins.sbt – add to the end of file |
| //scoverage addSbtPlugin("org.scoverage" % "sbt-scoverage" % "1.3.5") |

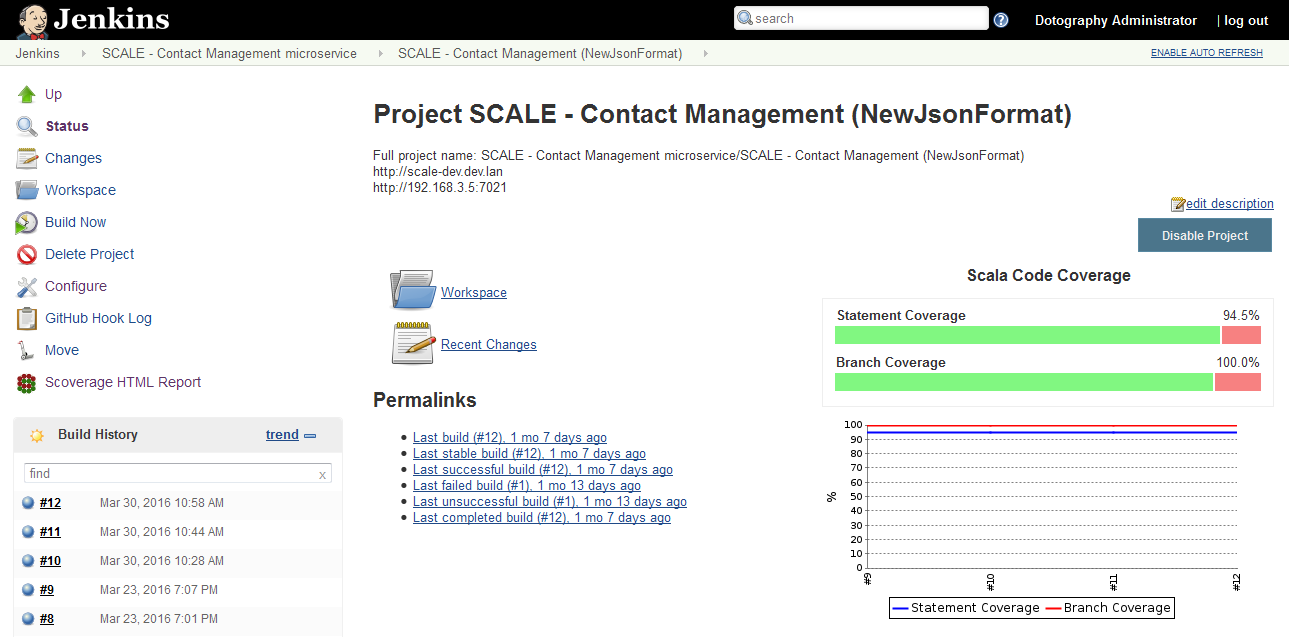
|  |
| --- |
| build.sbt – add to the end of file |
| coverageExcludedPackages := "<empty>;bootstrap.liftweb.Boot.\*" |

## Setting Code Coverage in Jenkins

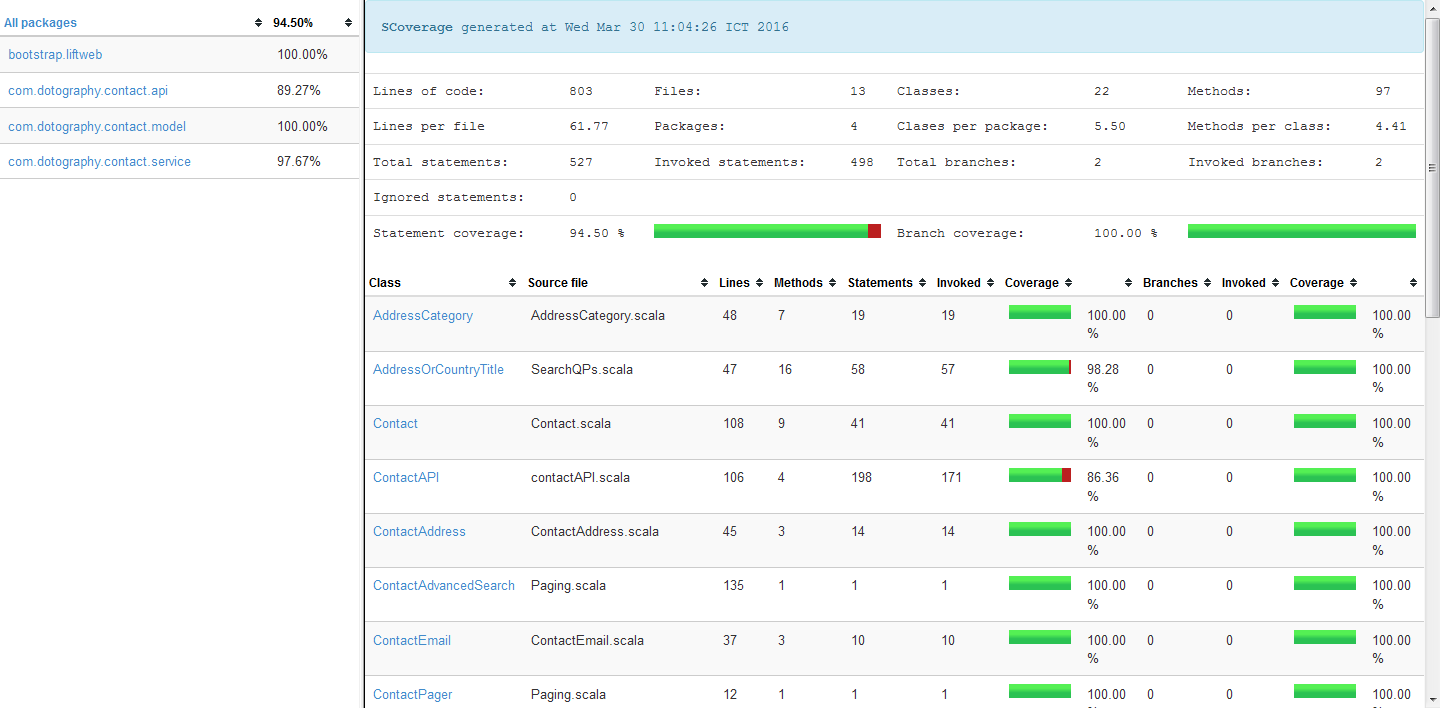
1. Enable code coverage by “coverage” before run unit test
2. Generate report by “coverageReport” after unit test is done



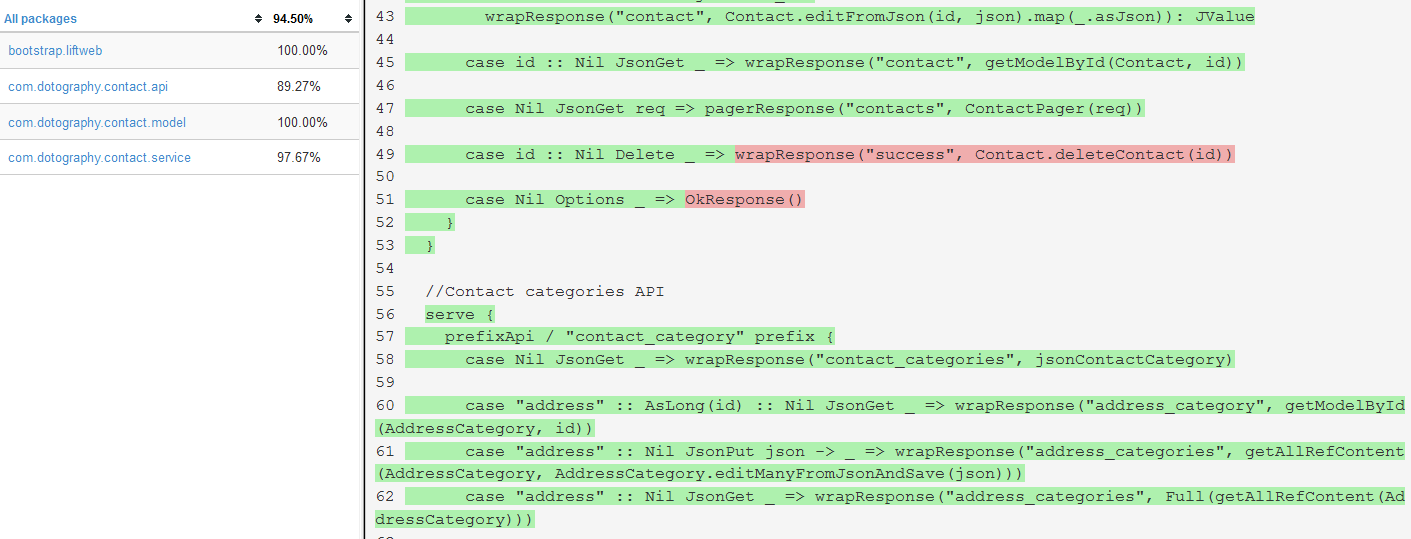
After test & build are successful, the project page will provide the Code Coverage Result as following:



At the left side navigation menu, Scoverage HTML Report will browse to the report detail as this screenshot.



The report will calculate the summary and provide the link at each class which browse to the source code and highlight the code that is tested or not tested.



# Application Deployment (Docker)

Jenkins provide tools or plugins to deploy the package to various server environment e.g. deploy war file to the running tomcat server, publish web components files to httpd server, deploy package to a docker container.

For microservices, we recommend to deploy the project into docker.

## Basic Deployment to Docker

Target container: Tomcat

This deployment method will perform the following action:

1. Stop and remove the existed container with same name
2. Remove the existed docker image with same name
3. Create docker image with copy the new build package to the image
4. Run the container at the specified port



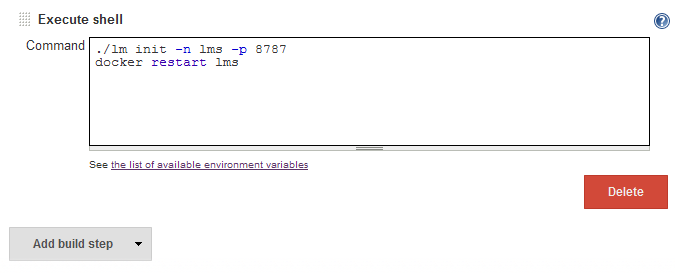
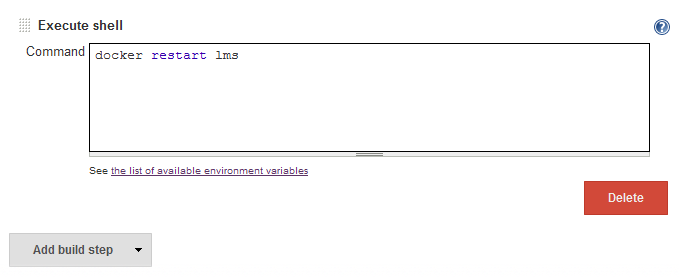
## Deployment to Docker with LM script

LiftMicroservices template provides a script named “lm” that can easy deploy the project as docker container.

LiftMicroservices Repository & Quick Guide: <https://github.com/dotography-code/LiftMicroservices>

Target Container: Jetty

At the build step,

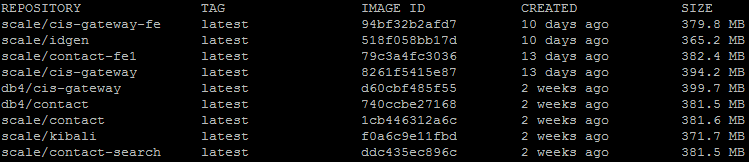
1. using the Execute shell to run the lm script as the screenshot  
   
2. Run first build
3. If build is successful, edit the configuration by remove the first lm script line then save the change.  
   

LM script will perform the following action:

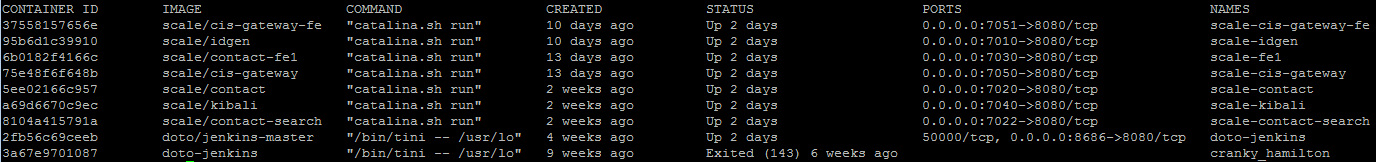
* Run the container from jetty image by mount volume to project’s target webapp directory
* The container run at the specified port. (default port is 8080 if not specified)

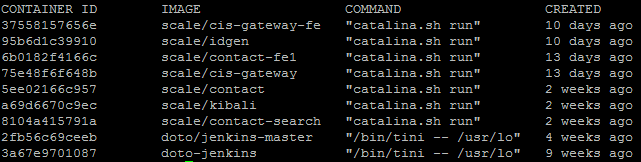
So it’s required to remove this command after first build success. Next build is required only docker restart to reload new files from host.

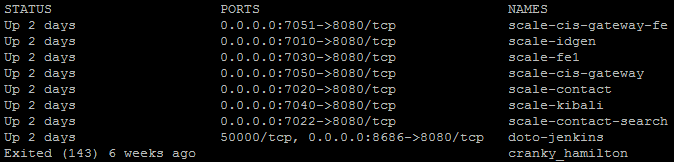
###### Screenshot are the sample of docker images



###### Screenshot are the sample of docker containers







# Notification (Slack)

Slack is a team collaboration tool. It brings all communication together in one place. It's real-time messaging, archiving and search for modern teams.

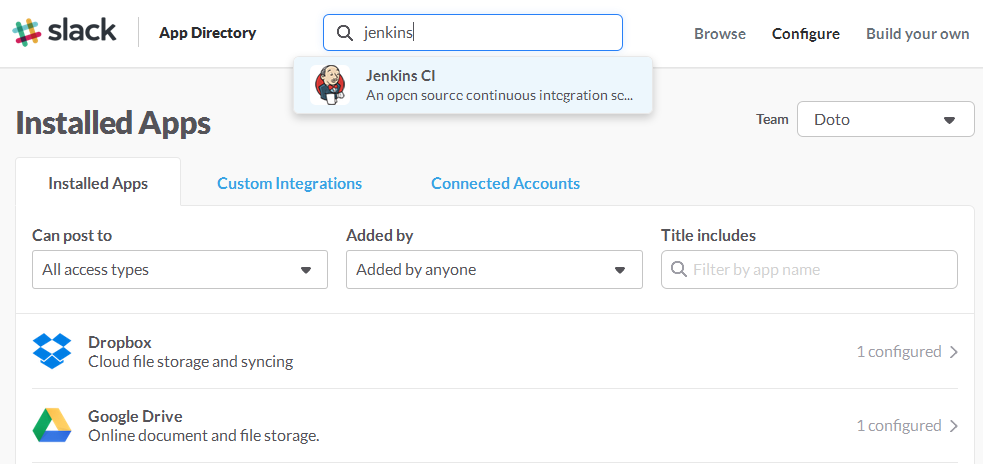
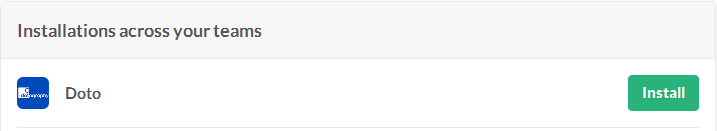
Slack provide the integration tool to integrate with Jenkins so we can automatically get notification of the build status.

Slack Setup

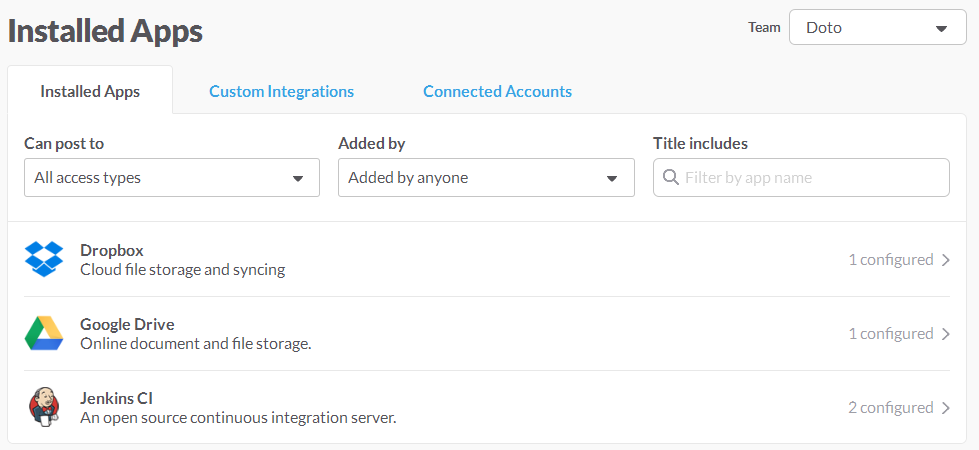
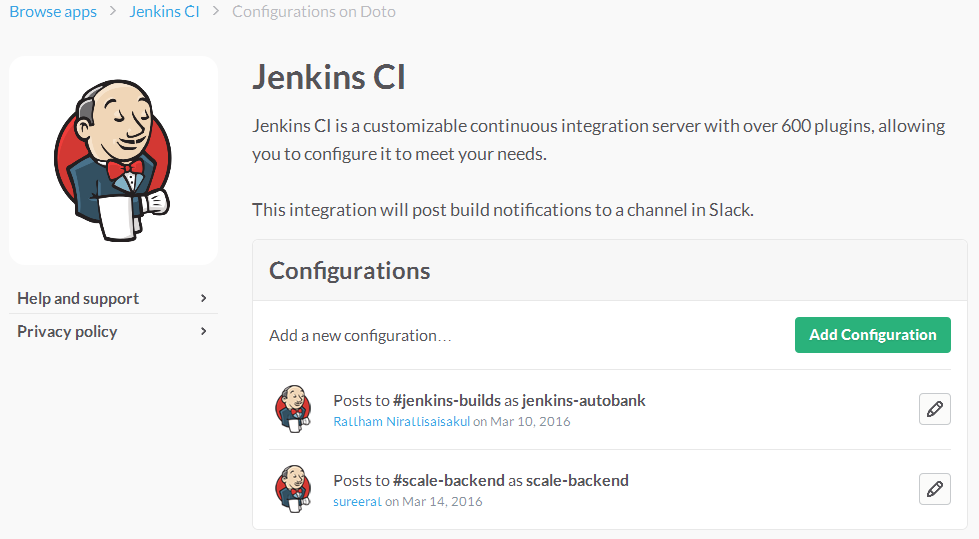
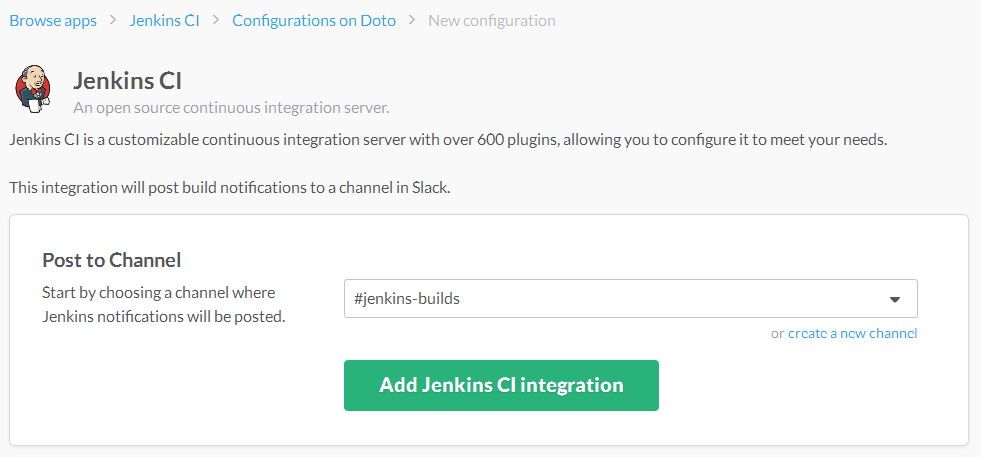
It’s required to install “Jenkins CI” app in Slack account via its application management tool if it’s not existed.

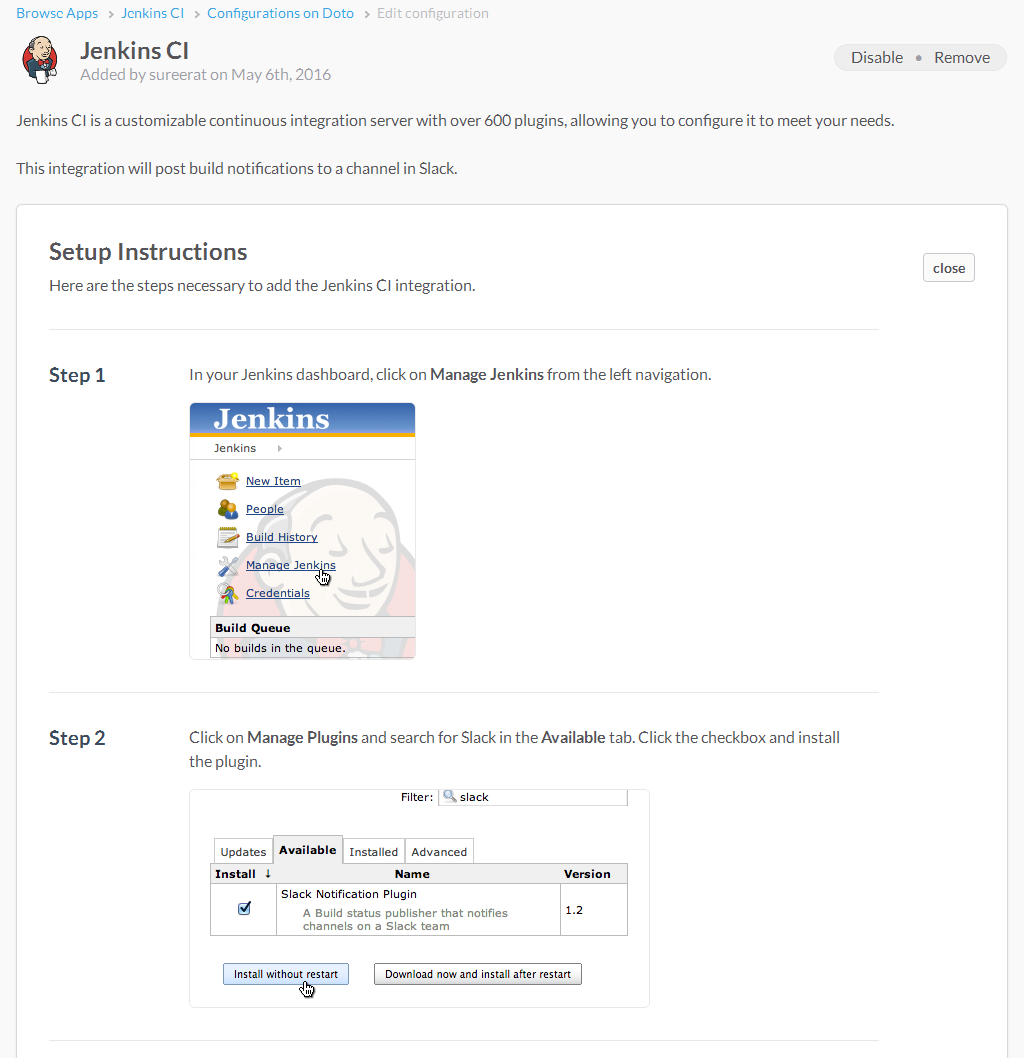
## Install Jenkins CI in Slack (One time setup)

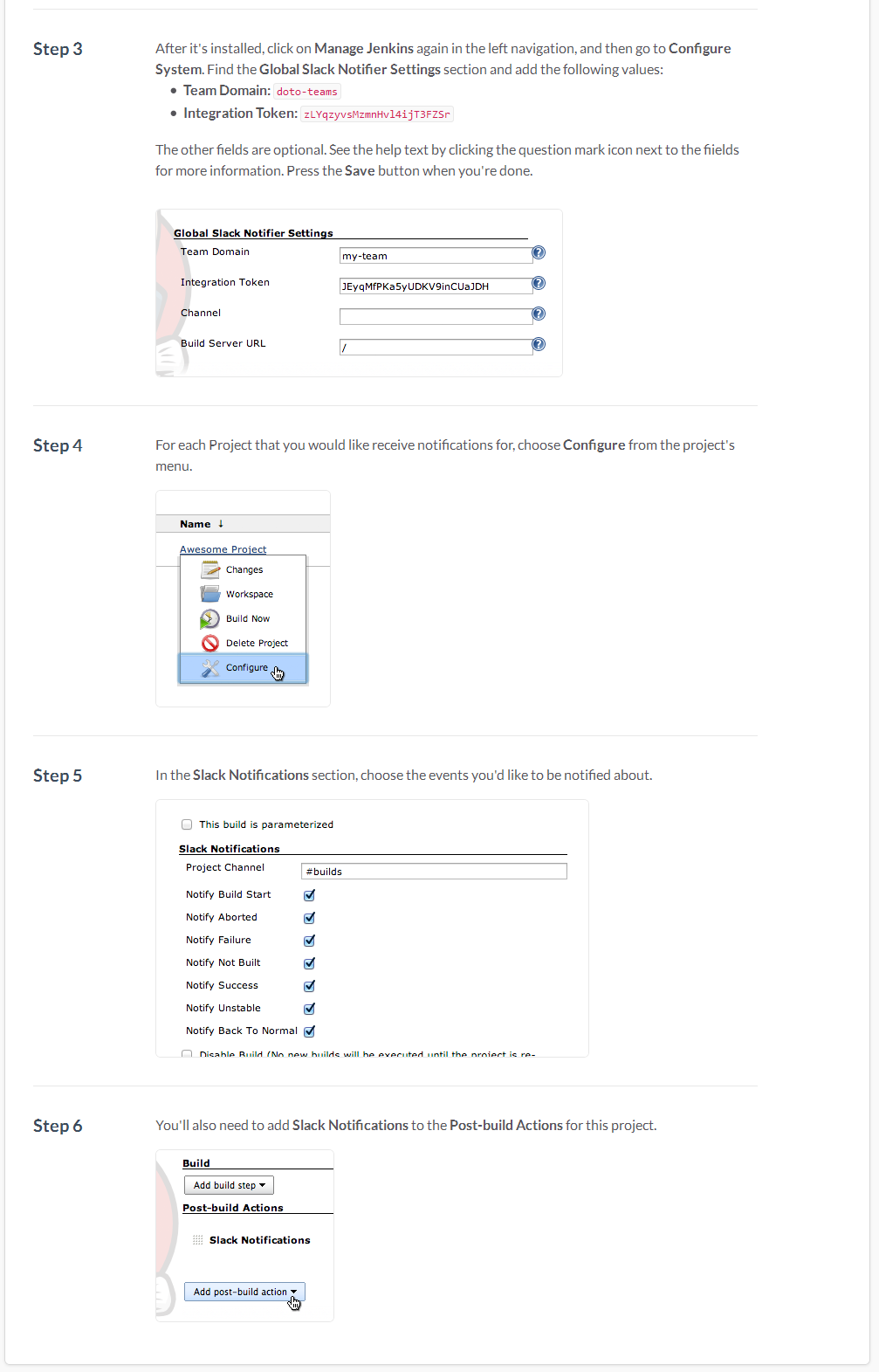
You can skip these step, if Jenkins CI is already existed.

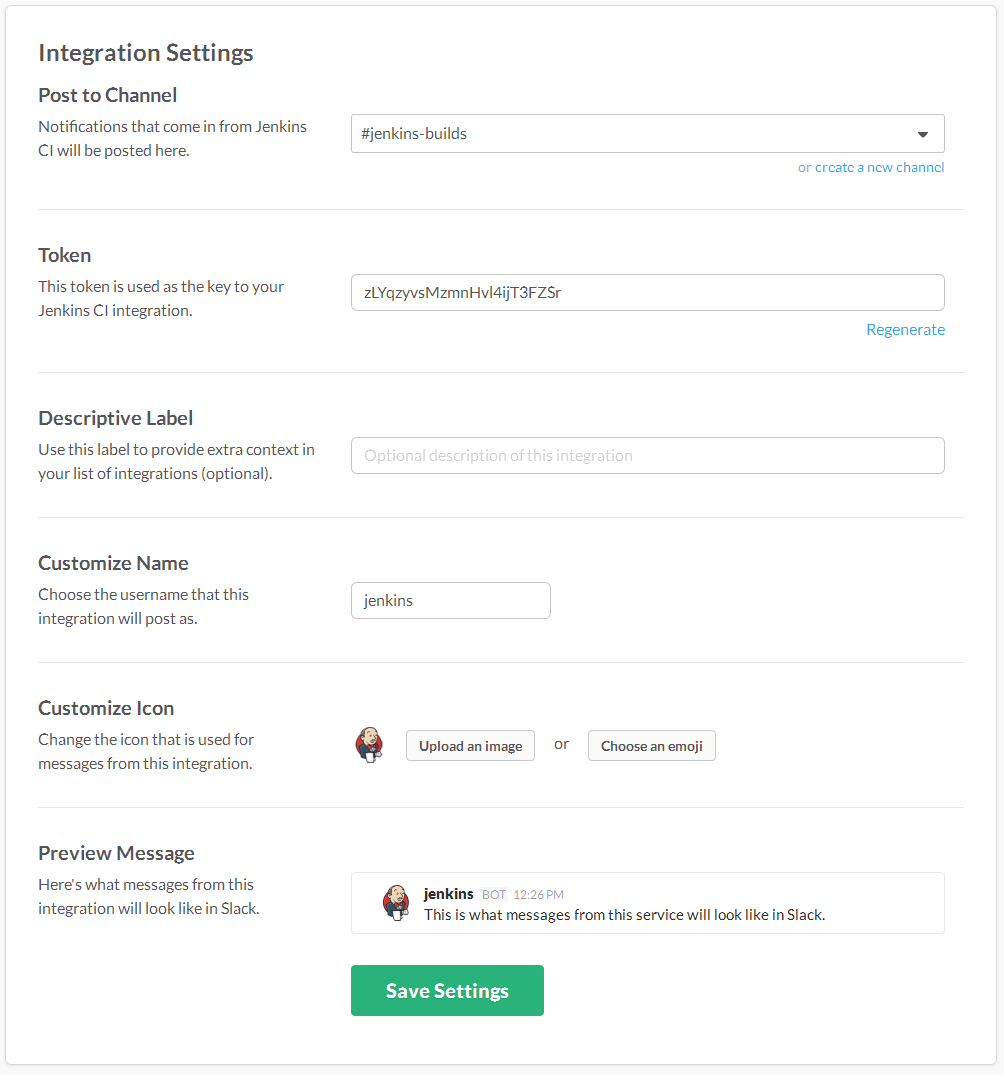
1. Login to your slack account via web browser
2. Browse to Application Management page: <https://doto-teams.slack.com/apps/manage>
3. Enter “Jenkins” at “Search” area in the top of page  
     
   
4. It would browse to the Application detail page, click “Install” at the “Team Subdomain” you want  
     
   

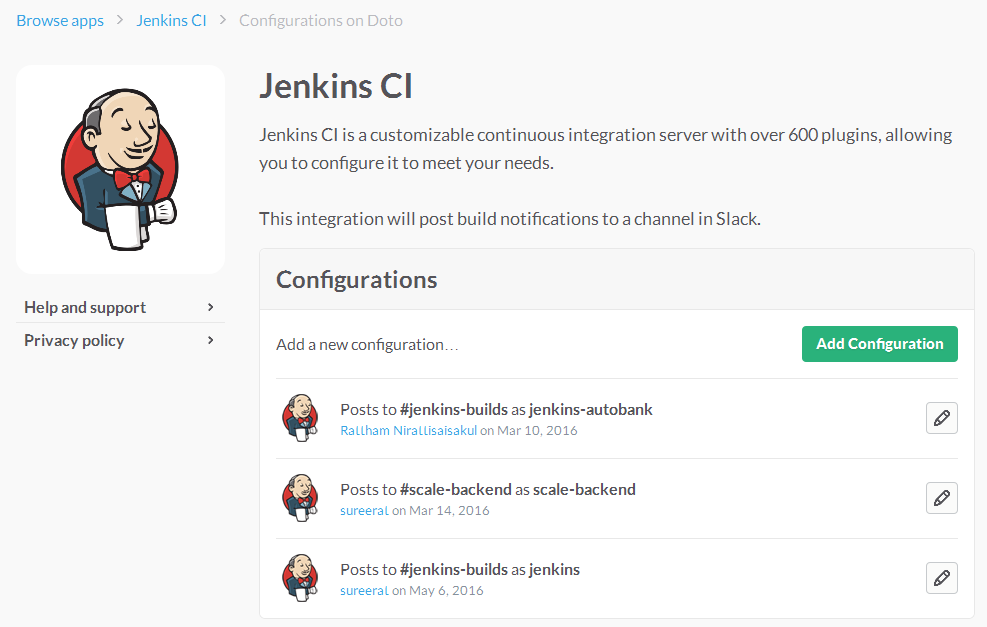
## Add New Configuration for Jenkins

1. Login to your slack account via web browser
2. Browse to Application Management page: <https://doto-teams.slack.com/apps/manage>
3. Enter Configuration page by clicking Jenkins CI  
     
   
4. At the Jenkins CI Configurations, click the button Add Configuration  
     
   
5. Enter the channel that the message will be sent to (or you can create new channel here)  
     
     
   then click Add Jenkins CI integration
6. Next step will come up with the Setup Instructions that you need to do in Jenkins & more Integration Settings





  
You can review, edit and submit the configuration by press Save Settings button.

After adding new configuration is completed, the new item will be added in the configurations list.  
  


# References

## Continuous Deployment Concept

* <http://electric-cloud.com/wiki/display/releasemanagement/Continuous+Deployment>
* <https://en.wikipedia.org/wiki/Continuous_delivery>

## Jenkins

* <https://wiki.jenkins-ci.org>
* <http://www.tutorialspoint.com/jenkins/index.htm>

## Git

* <https://git-scm.com>
* <https://github.com>
* <https://en.wikipedia.org/wiki/GitHub>
* <http://nvie.com/posts/a-successful-git-branching-model>

## Slack

* [https://www.slack.com](https://www.slack.com/)

## Unit Test & Coverage Code

* <https://github.com/scoverage>
* <https://github.com/scoverage/sbt-scoverage>

## Docker

* <https://www.docker.com>
* <https://en.wikipedia.org/wiki/Docker_%28software%29>