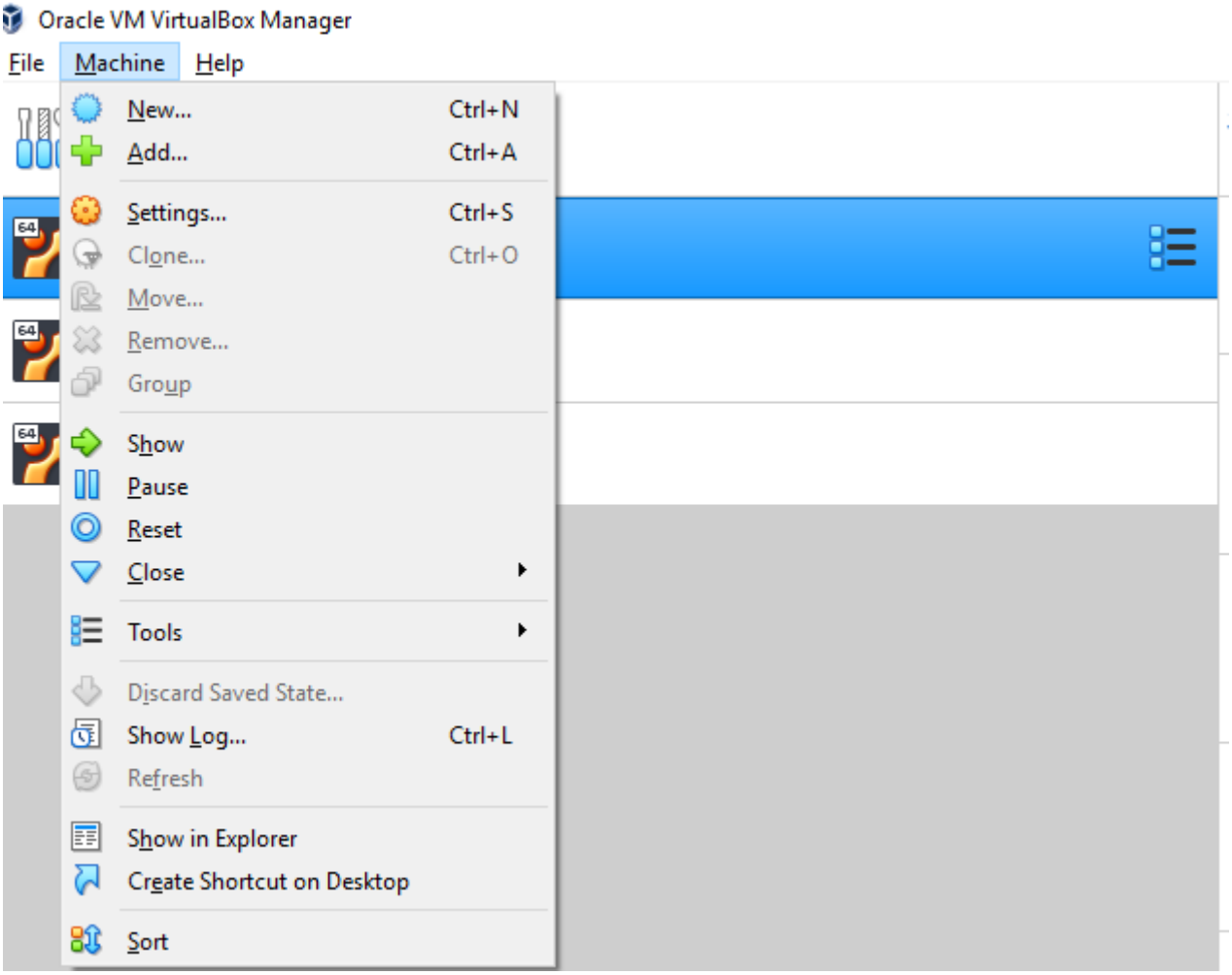
Guide to Setup CI/CD Tools

Table of Contents

Setup VM in VirtualBox

The below steps guides through to create Virtual Machine in VirtualBox.

- In VirtualBox home screen go to Machine Menu and click on New (Machine→New), which would open VM creation form window.



- In VM creation Window set VM name, location to store machine config and logs. and OS type desired(In our case it will be Linux) and version (Ubuntu 18.04).

  [VM Creation Window] | *vm_create_2.png*

- Click on expert mode and set desired RAM(min 4GB)
- Select Option `Create new Virtual Harddisk now` and click `create`.

  [VM set HDD] | *vm_create_2.png*

- Provide directory location to store VM image, set Hard disk space required(min 20GB) and set Fixed Size Mode so that we get better performance. Let the image type be default VDI. Then click Create Button(It would take couple of minutes to create Allocate Image)

  [VM Creation Window] | *vm_create_3.png*

- Now, Double click on the VM Name in VBox home page. which start the VM.

- A option will popup to provide OS Image. Browse to OS image location and provide OS image (ubuntu-18.04.1.0-live-server-amd64.iso). Then click on button start.

  [VM HDD configuration] | *vm_create_4.png*

- After image is loaded, you will be asked to choose the language, set the desired language.

  [VM setup] | *vm_create_5.png*

- Next you will be prompted to select keyboard layout, let the default configuration be as it is. Navigate to Done using tab and click enter.

  [VM Creation Window] | *vm_create_6.png*

- Next you will be prompted to select Network interface, let the default configuration be as it is will change it later on. Navigate to Done using tab and click enter.

  [VM Creation Window] | *vm_create_7.png*

- Next setup hard drive by selecting option `entire disk with LVM`. CAUTION: When choosing LVM make sure you've selected added entire disk to VG by default in 18.04 4GB is selected. You can add the Disk at later point as well.

  [VM Creation Window] | *vm_create_8.png*

- Navigate through `Configure Ubuntu Archive Mirror` with default values.

  [VM Creation Window] | *vm_create_9.png*

- A summary window will shown describing the Disk Partitions, Navigate to `Done` option and press Enter.

  [VM Creation Window] | *vm_create_10.png*

- Next Profile setup window comes up, fill all the details.

  [VM Creation Window] | *vm_create_11.png*

- Once you navigate from profile setup window, it would take sometime to setup OS. Once Setup completes, would prompt to install grub, select yes to install grub, finally after installation click on `Reboot`

  [VM Creation Window] | *vm_create_12.png*

Setup VM Network

We will add 3 Networks to VM, with first network being `host only`, second network being `NAT` and final network being `Bridged`

- Shutdown the VM.
- Select the VM, Navigate to Machine→Settings and select tab Network.

[VM Creation Window] | *vm_create_13.png*

- Select Adapter 1 tab and set `attached-to` Host-Only Adapter and fill the details as show below.

  [VM Network Adapter 1] | *vm_create_14.png*

- Select Adapter 2 tab and set `attached-to` NAT and fill the details as show below.

  [VM Network Adapter 2] | *vm_create_15.png*

- Select Adapter 3 tab and set `attached-to` Bridged and fill the details as show below.

  [VM Network Adapter 3] | *vm_create_16.png*

- Start the VM, login to the VM, execute below command and note down list of interface names.

```
$ ifconfig -a
```

- Open netplan yaml file at `/etc/netplan/50-cloud-init.yaml` and set static IP for host-only adapter and dhcp IP auto discovery for NAT and Bridged, For Host-only network before setting up note down the subnet for Host only adapter from VirtualBox Network. You can also setup a static IP from Bridge Network. A sample configuration is as shown below.

  [VM Network Adapter 3] | *vm_create_17.png*

- Reboot the VM.

Install and setup GitLab on Ubuntu 18.04 server.

Install Gitlab

- Update packages

```
$ sudo apt-get update
```

- Install additional packages

```
$ sudo apt-get install -y curl openssh-server ca-certificates
```

- Add the GitLab package repository and install the package

  Add the GitLab package repository.

```
$ curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.deb.sh
\
        | sudo bash
```

- Next, install the GitLab package. Change https://gitlab.example.com to the URL at which you want to access your GitLab instance. Installation will automatically configure and start GitLab at that URL. For https:// URLs GitLab will automatically request a certificate with Let's Encrypt, which requires inbound HTTP access and a valid hostname. You can also use your own certificate or just use http://.

```
$ sudo EXTERNAL_URL="https://gitlab.example.com" apt-get install gitlab-ce
```

- Browse to the hostname and login.

  On your first visit, you'll be redirected to a password reset screen. Provide the password for the initial administrator account and you will be redirected back to the login screen. Use the default account's username `root` to login.

Setup Email Address using Gmail.

- Enable 2-way Authentication by following the steps here
- Generate App code.
  - Go to your Google Account.
  - On the left navigation panel, click Security.
  - On the "Signing in to Google" panel, click App passwords. If prompted, enter your password.

    If you can't get to the page, 2-Step Verification is:

    - Not set up for your account
    - Set up for security keys only
  - At the bottom, click Select app and choose the app you're using.
  - Click Select device and choose the device you're using.
  - Click Generate.
  - Follow the instructions to enter the App password (the 16 character code in the yellow bar) on your device.
  - Copy the app password and click Done.
- Setup SMTP server on `Gitlab`
  - Open `/etc/gitlab/gitlab.rb`

    ```
    $ sudo vim /etc/gitlab/gitlab.rb
    ```

  - Copy the below content and update email and password generated in previous step

```
gitlab_rails['smtp_enable'] = true
gitlab_rails['smtp_address'] = "smtp.gmail.com"
gitlab_rails['smtp_port'] = 587
gitlab_rails['smtp_user_name'] = "my.email@gmail.com"
gitlab_rails['smtp_password'] = "my-gmail-password"
gitlab_rails['smtp_domain'] = "smtp.gmail.com"
gitlab_rails['smtp_authentication'] = "login"
gitlab_rails['smtp_enable_starttls_auto'] = true
gitlab_rails['smtp_tls'] = false
gitlab_rails['smtp_openssl_verify_mode'] = 'peer'

gitlab_rails['gitlab_email_from'] = 'gitlab@example.com'
gitlab_rails['gitlab_email_reply_to'] = 'noreply@example.com'
```

- Reload the configuration be executing below command

```
$ sudo gitlab-ctl reconfigure
```

- Testing gitlab's ability to send email
  - Open Ruby console

```
$ sudo gitlab-rails console
```

  - Execute below command after updating destination email address

```
> Notify.test_email('destination_email@address.com', 'Message Subject',
'<html><body>Message Body</body></html>').deliver_now
```

Install JDK-8 as Jenkins LTS does not support JDK-11

Install OpenJDK 8/11 in Ubuntu 18.04

Below are the steps to install openjdk 8/11 in ubuntu 18.04. The default jdk shipped in ubuntu 18.04 is JDK-10.

- Add Java PPA Repository.

```
$ sudo add-apt-repository ppa:webupd8team/java
```

- Update apt cache.

```
$ sudo apt update
```

- Now you can install `openjdk-8` or `openjdk-11` using below command.

```
$ sudo apt install openjdk-8-jdk
```

- Update Java alternatives.

```
$ sudo update-alternatives --install /usr/bin/java java usr/lib/jvm/java-8-openjdk-amd64/bin/java 1000
```

Install Jenkins Server on Ubuntu 18.04

Prerequiste

- JDK-8

Installtion

- Update Jenkins key cache.

```
$ wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -
```

- Add Repository.

```
$ sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ >
/etc/apt/sources.list.d/jenkins.list'
```

- Update package

```
$ sudo apt-get update
```

- Install Jenkins server

```
$ sudo apt-get install jenkins
```

This package installation will:

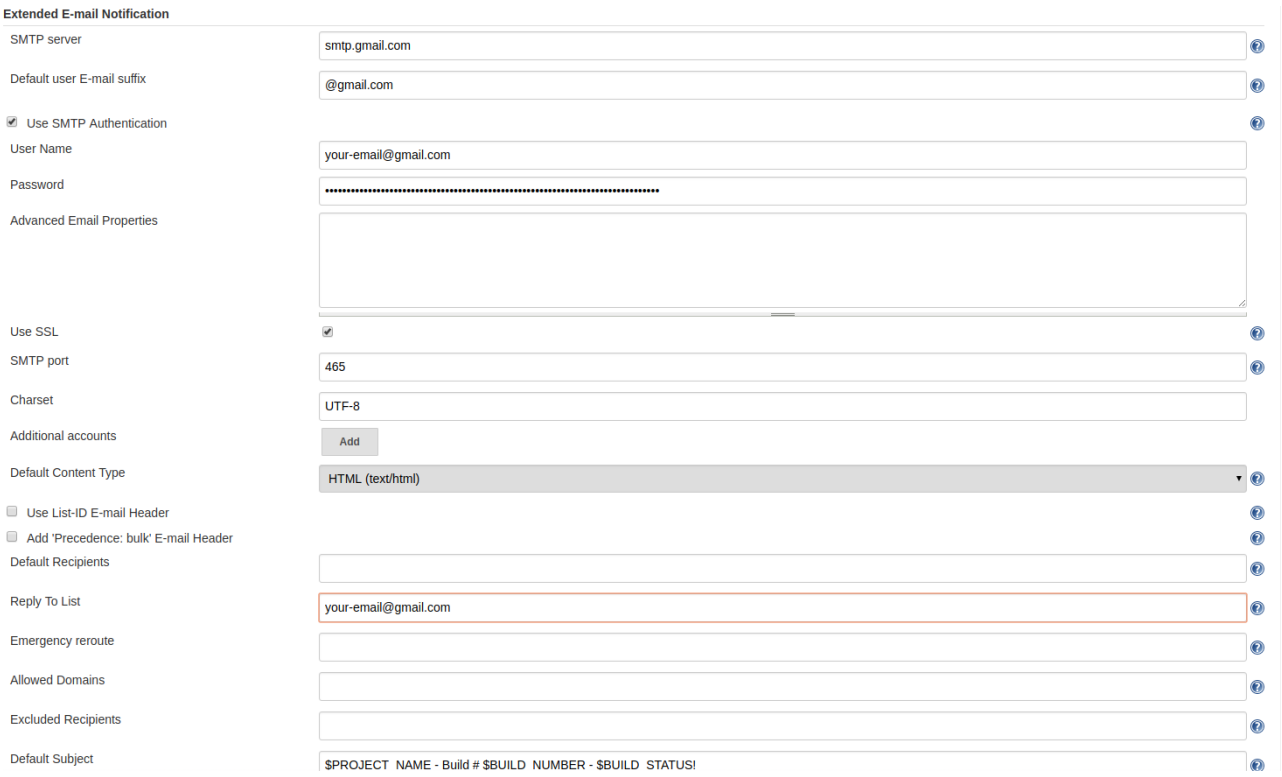
- Setup Jenkins as a daemon launched on start. See `/etc/init.d/jenkins` for more details.
- Create a `jenkins` user to run this service.
- Direct console log output to the file `/var/log/jenkins/jenkins.log`. Check this file if you are troubleshooting Jenkins.
- Populate `/etc/default/jenkins` with configuration parameters for the launch, e.g `JENKINS_HOME`
- Set Jenkins to listen on port `8080`. Access this port with your browser to start configuration.

If your `/etc/init.d/jenkins` file fails to start Jenkins, edit the `/etc/default/jenkins` to replace the line `----HTTP_PORT=8080----` with `----HTTP_PORT=8081----` Here, "8081" was chosen but you can put another port available.

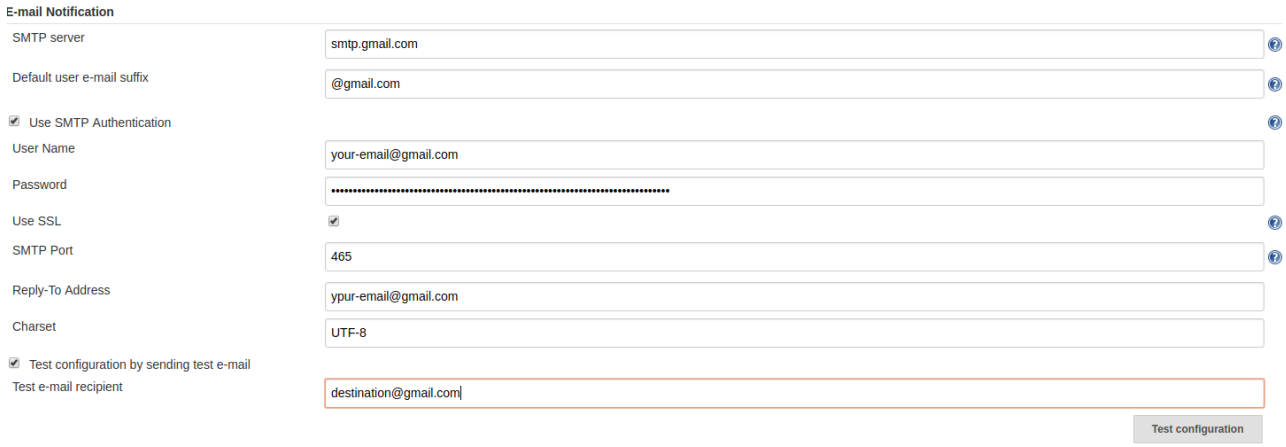
- While configuring on UI select `Install recommended plugins`.

Configure Email Notification using Gmail.

- Go to Manager Jenkins -→ Configure System.
- Goto Extending Email Notification section. Fill details as shown in below screenshot with your email address. SMTP Server is `smtp.gmail.com`, select adavanced option and set username and password(password should be your app code) set `USE SSL` with Port `465`



- Goto E-mail Notification section. Fill details as shown in below screenshot with your email address. SMTP Server is `smtp.gmail.com`, select adavanced option and set username and password(password should be your app code) set `USE SSL` with Port `465`



Use jenkins user while adding user to docker group.

Installation and configuration of docker in Ubuntu 18.04

Install docker package

Below are the steps to install docker in ubuntu 18.04

- First update existing packages.

```
$ sudo apt update ; sudo apt upgrade
```

- Install utility packages

```
$ sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

- Add GPG key

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

- Add docker repository to apt sources.

```
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"
```

- Update apt sources and install docker

```
$ sudo apt update ; sudo apt install docker-ce
```

- Verify docker service is started.

```
$ sudo systemctl status docker
```

Enable docker command execution without sudo.

- We need to add user to docker group to enable docker command execution without sudo user.
  - To add current user

    ```
    $ sudo usermod -aG docker ${USER}
    ```

  - Alternatively to add any other user

    ```
    $ sudo usermod -aG docker <username>
    ```

- Verify if user is added to docker group by listing the groups the user is added to.

```
$ id -nG ${USER}
```

- Login to new session, now you can execute `docker` commands without need `sudo` access.

```
$ su - ${USER}
```

Steps to Integrate Gitlab with Jenkins

Jenkins Server configuration

Install Plugins

- Goto `Manage Jenkins` → `Manage Plguins` → `Available Tab` and install following plugins.
  - Violation Comments to GitLab
  - GitLab
  - Blue Ocean
  - Common API for Blue Ocean
  - Config API for Blue Ocean

- Dashboard for Blue Ocean
- Events API for Blue Ocean
- Git Pipeline for Blue Ocean
- JWT for Blue Ocean
- Personalization for Blue Ocean
- Pipeline implementation for Blue Ocean
- REST API for Blue Ocean
- REST Implementation for Blue Ocean
- Web for Blue Ocean
- Blue Ocean Pipeline Editor
- Blue Ocean Executor Info
- Pipeline SCM API for Blue Ocean
- docker
- HTML Publisher

Create User for Gitlab

We need to create a user in Jenkins, such that when a JOBs executed by this user, when JOB is triggered by Webhooks from Gitlab.

- Goto `Manager Jenkins` → `Manage Users` → `Create User` and add user details.

Jenkins-to-GitLab authentication

This auth configuration is only used for accessing the GitLab API for sending build status to GitLab. It is not used for cloning git repos. The credentials for cloning (usually SSH credentials) should be configured separately, in the git plugin. This plugin can be configured to send build status messages to GitLab, which show up in the GitLab Merge Request UI. To enable this functionality:

- Create a new user in GitLab
- Give this user 'Developer' permissions on each repo you want Jenkins to send build status to
- Log in or 'Impersonate' that user in GitLab, click the user's icon/avatar and choose Settings
- Click on 'Access Tokens'
- Create a token named e.g. 'jenkins' with 'api' scope; expiration is optional
- Copy the token immediately, it cannot be accessed after you leave this page
- On the Global Configuration page in Jenkins, in the GitLab configuration section, supply the GitLab host URL, e.g. http://your.gitlab.server
- Click the 'Add' button to add a credential, choose 'GitLab API token' as the kind of credential, and paste your GitLab user's API key into the 'API token' field

- Click the 'Test Connection' button; it should succeed.

Jenkins-to-Gitlab Clone AUthentication

We need to Add SSH key to enable Jenkins to clone repositories.

- Generate SSH public/private key using below command

```
$ ssh-keygen -t rsa
```

- Goto `Manage Jenkins` → `Configure Credentials` → `Credentials` → ` System` → `Global Credentials` → `Add Credentials`
- Select Kind as `ssh username with private key`
- Set the username generate in previous section.
- Set `ID`
- Select option `private key enter directly` and add key from `cat ~/.ssh/id_rsa`
- Provide pass phrase if you've added
- And click ok
- Now login to Gitlab with user created in previous section.
- On Top right corner click on User Avatar → Settings and then on 'SSH Keys'.
- Add the public key from output of `cat ~/.ssh/id_rsa/pub`

Create Project and configure project settings in Gitlab.

- Now create a sample Project with Jenkinsfile in Gitlab.

```
pipeline {
    agent any

    stages {
        stage('Hello') {

            steps {
                echo "Hello from Jenkins pipeline"
            }

        }
    }
}
```

- Add Build User to the project with role `Developer`
- Goto Jenkins Server and create new multibranch project say `k-sample-ci-project`
- In Branch Sources section select `Git`

- Provide the git clone URL of the Gitlab project. `git@localhost:k-projects/sample-ci-project.git`
- Select Credentials as Git SSH User added in previous section.
- Click Save

Adding Webhook.

- Log in as that user (this is required even if you are a Jenkins admin user), then click on the user's name in the top right corner of the page
- Click `Configure` → `API Token` → `Add new Token` then 'generate', and note/copy the User ID and API Token
- Now goto Gitlab Project → <code>settings</code> → <code>Integration</code> add the webhook <code><a href="http://localhost:8182/project/<jenkins_project_name&gt" class="bare">http://localhost:8182/project/<jenkins_project_name&gt;</code>, and api token from previous step. click the 'Test' button, and it should succeed and trigger a JOB.

To prevent this type of exploitation from happening, starting with GitLab 10.6, all Webhook requests to the current GitLab instance server address and/or in a private network will be forbidden by default. That means that all requests made to 127.0.0.1, ::1 and 0.0.0.0, as well as IPv4 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16 and IPv6 site-local (ffc0::/10) addresses won't be allowed.
This behavior can be overridden by enabling the option "Allow requests to the local network from hooks and services" in the "Outbound requests" section inside the Admin area under Settings (`/admin/application_settings/network`)