Installation Guide  
Native Open edX Installation on Ubuntu 16.04 (Xenial Xerus)

horizontal line

# Prerequisites

* + Ubuntu 16.04 amd64

## Set-up configurations

Ubuntu package sources need to be updated.

sudo apt-get update -y

sudo apt-get upgrade -y

sudo reboot

## Installation Procedure

1. Firstly, we’ll set the OPENEDX\_RELEASE variable to the Git Tag corresponding to the Ironwood release of Open edX.
2. Next, we create a config.yml file, which states the hostname of the Learning Management System (LMS) and the Content Management System (CMS).

EDXAPP\_LMS\_BASE: “127.0.0.1”

EDXAPP\_CMS\_BASE: “127.0.0.1:8080”

(works only when the LMS hostname and the CMS hostnames are the same, or the CMS hostname is a subdomain of the LMS hostname).

1. Further, we will now bootstrap the Ansible installation.
2. Firstly, we will retrieve the ansible-bootstrap.sh bash script from the Github server.

wget https://raw.githubusercontent.com/edx/configuration/$OPENEDX\_RELEASE/util/install/ansible-bootstrap.sh

1. The bash script will initialize the variables (in case they haven’t been initialized).

if [[ -z "${ANSIBLE\_REPO}" ]]; then

ANSIBLE\_REPO="https://github.com/edx/ansible.git"

fi

if [[ -z "${ANSIBLE\_VERSION}" ]]; then

ANSIBLE\_VERSION="master"

fi

if [[ -z "${CONFIGURATION\_REPO}" ]]; then

CONFIGURATION\_REPO="https://github.com/edx/configuration.git"

fi

if [[ -z "${CONFIGURATION\_VERSION}" ]]; then

CONFIGURATION\_VERSION=${OPENEDX\_RELEASE-master}

fi

if [[ -z "${UPGRADE\_OS}" ]]; then

UPGRADE\_OS=false

fi

if [[ -z "${RUN\_ANSIBLE}" ]]; then

RUN\_ANSIBLE=true

fi

1. Next, the bootstrapping constants (Python Virtual Environment Version, PIP Version, etc.) are set. (will be needed later).

VIRTUAL\_ENV\_VERSION="15.2.0"

PIP\_VERSION="9.0.3"

SETUPTOOLS\_VERSION="39.0.1"

VIRTUAL\_ENV="/tmp/bootstrap"

PYTHON\_BIN="${VIRTUAL\_ENV}/bin"

ANSIBLE\_DIR="/tmp/ansible"

CONFIGURATION\_DIR="/tmp/configuration"

EDX\_PPA\_KEY\_SERVER="keyserver.ubuntu.com"

EDX\_PPA\_KEY\_ID="B41E5E3969464050"

1. Note that PPAs (Personal Package Archives) are software repositories which are used for non-standard updates or to obtain non-standard softwares. One must be cautious while adding a PPA.
2. IIT Bombay’s firewall blocks the default port. This can be fixed by forcing the use of port 80. Initialize EDX\_PPA\_KEY\_SERVER to “keyserver.ubuntu.com:80” instead of “keyserver.ubuntu.com”.
3. The script returns an exit status 1 in case it runs without root privileges.

if [[ $(id -u) -ne 0 ]] ;then

echo "Please run as root";

exit 1;

fi

1. Thereafter, the script uses grep to check if the os-release file in the etc folder contains the string ‘Trusty Tahr’, ‘Xenial Xerus’, or ‘Bionic Beaver’, and sets the SHORT\_DIST variable accordingly.

if grep -q 'Trusty Tahr' /etc/os-release

then

SHORT\_DIST="trusty"

elif grep -q 'Xenial Xerus' /etc/os-release

then

SHORT\_DIST="xenial"

elif grep -q 'Bionic Beaver' /etc/os-release

then

SHORT\_DIST="bionic"

else

cat << EOF

This script is only known to work on Ubuntu Trusty, Xenial, and Bionic;

exiting. If you are interested in helping make installation possible

on other platforms, let us know.

EOF

exit 1;

fi

1. The SHORT\_DIST variable will be used to check which dependencies need to be installed.
2. Next, the script installs GnuPG for the Bionic release for secure key exchange. (Necessary for apt-key update in bionic).

if [[ "${SHORT\_DIST}" == bionic ]] ;then

apt-get install -y gnupg

fi

1. The script then updates the list of keys used by apt to authenticate packages.

apt-key update -y

1. Software-properties-common provides some useful scripts to add or remove PPAs (used in our case). Hence, the script executes apt-get install -y software-properties-common command.

apt-get install -y software-properties-common

1. Python-software-properties is an alternative for software-properties-common and is used for ubuntu releases older than Xenial Xerus (12.04). Therefore, the script checks for this situation and executes the apt-get install -y python-software-properties command (if needed).

if [[ "${SHORT\_DIST}" != bionic ]] && [[ "${SHORT\_DIST}" != xenial ]];then

apt-get install -y python-software-properties

fi

1. Now the script uses the add-apt-repository -y ppa:git-core/ppa command to add the git PPA. This PPA provides the latest stable upstream git version.

add-apt-repository -y ppa:git-core/ppa

1. For older software, we need to install our own PPA. Incase the ubuntu release being used isn’t bionic, the script uses the apt-key adv --keyserver "${EDX\_PPA\_KEY\_SERVER}" --recv-keys "${EDX\_PPA\_KEY\_ID}" command to download the key with keyid B41E5E3969464050 from the ubuntu keyserver directly into the trusted set of keys. Along with that, we add the recently installed PPA to the list of sources in our system so that Ubuntu checks for updates from this PPA as well.

if [[ "${SHORT\_DIST}" != bionic ]] ;then

apt-key adv --keyserver "${EDX\_PPA\_KEY\_SERVER}" --recv-keys "${EDX\_PPA\_KEY\_ID}"

add-apt-repository -y "${EDX\_PPA}"

fi

1. Along with that, the script installs the required dependencies (python 2.7 etc.).

apt-get install -y python2.7 python2.7-dev python-pip python-apt python-yaml python-jinja2 build-essential sudo git-core libmysqlclient-dev libffi-dev libssl-dev

1. The script then updates the PyPI repository.

pip install --upgrade pip=="${PIP\_VERSION}"

1. Further, the script adds the /usr/local/bin/virtualenv directory to the list of possible directories containing executable files. To achieve this, we use the PATH environment variable.

PATH=/usr/local/bin:${PATH}

1. Now, the script installs setuptools (used to package python projects) and virtualenv (a tool to create isolated python environments) with the pre-defined versions.

pip install setuptools=="${SETUPTOOLS\_VERSION}"

pip install virtualenv=="${VIRTUAL\_ENV\_VERSION}"

1. Next, the script creates a python virtual environment in the /tmp/bootstrap directory. Along with that, the path to the python packages in the virtual environment is added to the list of possible paths containing executables (using the PATH environment variable).

/usr/local/bin/virtualenv "${VIRTUAL\_ENV}"

PATH="${PYTHON\_BIN}":${PATH}

1. Later on, the repository located at \_ is fetched (cloned) to the \_ directory. The script also uses the git checkout command to switch to the desired local branch.

git clone ${CONFIGURATION\_REPO} ${CONFIGURATION\_DIR}

cd ${CONFIGURATION\_DIR}

git checkout ${CONFIGURATION\_VERSION}

1. The script uses the make utility to execute the commands listed under the requirements tag in the makefile. Essentially, what it does is install the required python packages with their corresponding versions listed in the pre-requirements.txt and requirements.txt file.

make requirements

1. Finally, the edx\_ansible playbook is executed. Thereafter, the temporary directories used during the installation process are removed.

cd "${CONFIGURATION\_DIR}"/playbooks

"${PYTHON\_BIN}"/ansible-playbook edx\_ansible.yml -i '127.0.0.1,' -c local -e "configuration\_version=${CONFIGURATION\_VERSION}"

rm -rf "${ANSIBLE\_DIR}"

rm -rf "${CONFIGURATION\_DIR}"

rm -rf "${VIRTUAL\_ENV}"

rm -rf "${HOME}/.ansible"

Submitted by:

Mridul Mahjan, Harshit Mogalapalli and Soumyakant Mohakul