Odoo DevOps Documentation

IT-Projects LLC

Nov 11, 2020

Contents:

Note: The project is moved to https://itpp.dev/ops/ and will be shutdown here soon

CHAPTER 1

Docker

CHAPTER 2

Kubernetes

2.1 Kubernetes solutions

There is a lot of ways to run your Kubernetes cluster on different platforms including single-node Minikube with completely automated setup on your own laptop or managed cluster on Google Compute Engine.

In this documentation we will consider intallation of Minikube cluster on your server or personal computer to give you an idea of how quickly configure minimal working cluster on one machine.

Different platforms and solutions you can find in official kubernetes documentation

There should be no difference in where and how you set up your cluster. So you can pick up any of the solutions presented instead.

2.2 Minikube

Minikube is easiest way to run single-node Kubernetes cluster locally. Setup is completely automated so it is just matter of installation and starting the cluster.

2.2.1 Installing Minikube

In order to install Minikube you need to:

- Enable Intel Virtualization Technology or AMD virtualization in your computer's BIOS
- Install VirtualBox or alternatively you can install other hypervisors: VMware Fusion, HyperKit, KVM or Hyper-V depending on your OS
- Install kubectl according to the instructions
- Install latest Minikube

2.2.2 Starting Minukube

To start cluster you can just run:

```
minikube start
```

Depending on the hypervisor you want to use you can specifiy it by –vm-driver option and choose amount of memory you want Minikube to use:

```
minikube start --memory 4096 --vm-driver virtualbox
```

Minikube also supports a –vm-driver=none option that runs the Kubernetes components on the host and not in a VM. In this case you should have Docker installed.

2.2.3 Iteract with your cluster

Now you can access your cluster with kubectl proxy:

```
kubectl proxy --port=8001 &
```

And you can get the API with curl or any browser:

```
curl http://localhost:8001/api/
```

2.2.4 Dashboard

Minikube automaticly have Kubernetes Dashboard - web-based UI for Kubernetes clusters. It allows you to monitor and manage aplications on your cluster.

To access dashboard you can just type in console:

```
minikube dashboard
```

And it will open in your default browser.

Or to get url you can run:

```
minikube dashboard --url
```

2.2.5 Stopping Minikube

To stop your cluster just run:

```
minikube stop
```

CHAPTER 3

GitLab CI/CD

3.1 Gitlab - Kubernetes integration

You can easily connect existing Kubernetes cluster to your GitLab project. With connected cluster you can use Review Apps, deploy your applications and run your pipelines.

3.1.1 Adding an existing Kubernetes cluster

In order to add your existing Kubernetes cluster to your project:

- Navigate to your project's Operations > Kubernetes page.
- Click on Add Kubernetes cluster.
- Click on Add an existing Kubernetes cluster and fill in the details:
  - Kubernetes cluster name (required) - The name you wish to give the cluster.
  - Environment scope (required) - The associated environment to this cluster. You can leave it with "*".
  - API URL (required) - It's the URL that GitLab uses to access the Kubernetes base API. You can access it locally with cubectl proxy and need to make it accessible externially. In the end you should have something like "https://kubernetes.example.com".
  - CA certificate (optional) - If the API is using a self-signed TLS certificate, you'll also need to include the ca.crt contents here.
  - Token - GitLab authenticates against Kubernetes using service tokens, which are scoped to a particular namespace. If you don't have a service token yet, you can follow the Kubernetes documentation to create one. You can also view or create service tokens in the Kubernetes dashboard (under Config > Secrets). The account that will issue the service token must have admin privileges on the cluster.
  - Project namespace (optional) - You don't have to fill it in; by leaving it blank, GitLab will create one for you.
- Click on Create Kubernetes cluster.

After a couple of minutes, your cluster will be ready to go.

If you using Minukube cluster or just have Kubernetes Dashboard you can get CA certificate and token in Dashboard. You need to choose default namespace and click on secrets. There should be default token with CA and token inside.

3.1.2 Installing applications

GitLab provides a one-click install for some applications which will be added directly to your connected Kubernetes cluster.

To one-click install applications:

- Navigate to your project's Operations > Kubernetes page.
- Click on your connected cluster.
- Click install button beside the application you need.

You need to install Helm Tiller before you install any other application

3.2 GitLab Runner

There is a different ways to install GitLab Runner on your Kubernetes cluster.

3.2.1 One-click install

If your Kubernetes cluster is connected to your GitLab project you can just:

- Navigate to your project's Operations > Kubernetes page.
- Click on your connected cluster.
- Install Helm Tiller by clicking the install button beside it.
- Install GitLab Runner by clicking the install button beside it.

3.2.2 Deploy GitLab Runner manually

If you want to cofigure everything yourself, you can deploy runner manually.

First you need to create namespace for your future deployment:

```
kubectl create namespace gitlab-runner-ns
```

To check your current namespaces:

```
kubectl get namespaces
```

Now set created namespace as default:

```
kubectl config set-context $(kubectl config current-context) --namespace=gitlab-
↪runner-ns
```

To deployment we will need to create a deployment.yaml, config-map.yaml and secret.yaml.

Start with config-map.yaml:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: gitlab-runner-cm
  namespace: gitlab-runner-ns
data:
  config.toml: |
    concurrent = 10
    check_interval = 30

  entrypoint: |
    #!/bin/bash

    set -xe

    cp /scripts/config.toml /etc/gitlab-runner/

    # Register the runner
    /entrypoint register --non-interactive \
      --url $GITLAB_URL \
      --executor kubernetes

    # Start the runner
    /entrypoint run --user=gitlab-runner \
      --working-directory=/home/gitlab-runner
```

And create config map with:

```
kubectl create -f config-map.yaml
```

For sake of not showing your token in clear in your deployment file we need to create secret.yaml with token as base 64 string:

```
echo -n "your_token" | base64
```

```
apiVersion: v1
kind: Secret
metadata:
  name: gitlab-runner-secret
  namespace: gitlab-runner-ns
type: Opaque
data:
  runner-registration-token: <your token as base 64 string>
```

Now, create secret with:

```
kubectl create --validate -f secret.yaml
```

And finally deployment.yaml file:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: gitlab-runner
  namespace: gitlab-runner-ns
spec:
  replicas: 1
```

(continues on next page)

(continued from previous page)

```
  selector:
    matchLabels:
      name: gitlab-runner
  template:
    metadata:
      labels:
        name: gitlab-runner
    spec:
      containers:
        - name: gitlab-runner
          image: gitlab/gitlab-runner:alpine-v9.3.0
          command: ["/bin/bash", "/scripts/entrypoint"]
          env:
            - name: GITLAB_URL
              value: "https://gitlab.com/"
            - name: REGISTRATION_TOKEN
              valueFrom:
                secretKeyRef:
                  name: gitlab-runner-secret
                  key: runner-registration-token
          imagePullPolicy: Always
          volumeMounts:
            - name: config
              mountPath: /scripts
            - name: cacerts
              mountPath: /etc/gitlab-runner/certs
              readOnly: true
      restartPolicy: Always
      volumes:
        - name: config
          configMap:
            name: gitlab-runner-cm
        - name: cacerts
          hostPath:
            path: /var/mozilla
```

For creating runners gitlab needs ClusterRoleBinding with cluster-admin role. So before deploying we creating cluster role:

```
kubectl create clusterrolebinding gitlab-cluster-admin --clusterrole=cluster-admin --
↪group=system:serviceaccounts --namespace=gitlab-runner-ns
```

And now creating deployment:

```
kubectl create --validate -f deployment.yaml
```

CHAPTER 4

Github

4.1 Creating Pull Requests in batch

4.1.1 Prerequisites

- Add a SSH key to your GitHub account. See: https://help.github.com/en/articles/adding-a-new-ssh-key-to-your-github-account
- Install hub. Look this: https://github.com/github/hub#installation

4.1.2 Script

Make a script `make-prs.sh` with following content

```
#!/bin/bash

# ORGANIZATION GITHUB URL
ORG=it-projects-llc
UPSTREAM_URL_GIT=https://github.com/$ORG

# DEVELOPER INFO
USERNAME=yelizariev

# WHERE TO CLONE
DIRECTORY_CLONE=$(pwd)

# DESCRIPTION OF THE UPDATES
MSG=":shield: travis.yml notifications webhook travis"
BRANCH_SUFFIX=travis-notifications

REPOS=(
    misc-addons
    saas-addons
```

(continues on next page)

(continued from previous page)

```
    pos-addons
    access-addons
    mail-addons
    website-addons
    sync-addons
    )
BRANCHES=(
    10.0
    11.0
    12.0
    )

for REPO in "${REPOS[@]}"; do
    if [ ! -d $DIRECTORY_CLONE/$REPO ]
    then
        git clone $UPSTREAM_URL_GIT/$REPO.git $DIRECTORY_CLONE/$REPO
        cd $DIRECTORY_CLONE/$REPO
        git remote rename origin upstream
        git remote add origin git@github.com:$USERNAME/$REPO.git
    fi
    cd $DIRECTORY_CLONE/$REPO
    for BRANCH in "${BRANCHES[@]}"; do
        git fetch upstream $BRANCH
        git checkout -b $BRANCH-$BRANCH_SUFFIX upstream/$BRANCH

        # CHECK THAT UPDATES ARE NOT DONE YET
        if grep -qx '    on_failure: change' .travis.yml
        then
            echo "File are already updated in $REPO#$BRANCH"
        else
            # MAKE UPDATE
            { echo '  webhooks:'; echo '    on_failure: change'; echo '  urls:'; echo
↪'    - "https://ci.it-projects.info/travis/on_failure/change"';} >> ./.travis.yml
        fi
        git commit -a -m "$MSG"
        git push origin $BRANCH-$BRANCH_SUFFIX
        hub pull-request -b it-projects-llc:$BRANCH -m "$MSG"
    done
done
```

Update script according to you needs

Run it with `bash make-prs.sh`

4.2 Merge bot for GitHub

The script gives the right to a certain circle of people to merge branches in the repository by sending the certain comment in the pull request.

4.2.1 Prepare IFTTT's hooks

- Log in / Sign up at https://ifttt.com/
- Click on `Documentation` button here: https://ifttt.com/maker_webhooks

- Replace `{event}` with event name, for example `travis-not-finished-pr`, `travis-success-pr` and `travis-failed-pr`. Save the links you got.

4.2.2 Create AWS Lambda function

Create lambda function with following settings:

- Runtime

  Use `Python 3.6`

- Environment variables
  - `GITHUB_TOKEN` – generate one in https://github.com/settings/tokens . Select scope `repo`.
  - `USERNAMES` – use comma-separated list of Github's usernames without @.
  - `LOG_LEVEL` – optional. Set to DEBUG to get detailed logs in AWS CloudWatch.
  - `MSG_RQST_MERGE` – message-request for merge. Default: `I approve to merge it now`
  - `IFTTT_HOOK_RED_PR`, `IFTTT_HOOK_GREEN_PR`, `IFTTT_HOOK_NOT_FINISHED_PR` – use IFTTT's hooks
- Trigger

  Use `API Gateway`. Once you configure it and save, you will see `API endpoint` under Api Gateway **details** section. Use option `Open`

  Now register the URL as webhook at github: https://developer.github.com/webhooks/creating/. Use following webhook settings:
  - **Payload URL** – the URL
  - **Content Type**: application/json
  - **Which events would you like to trigger this webhook?** – *Let me select individual events* and then select `[x] Issue comments`
- Function Code
  - Copy-paste this code: https://gitlab.com/itpp/odoo-devops/raw/master/tools/github-merge-bot/lambda_function.py
- Basic settings
  - Change time running function by 15 sec – `Timeout` (default 3 sec)

4.2.3 Create IFTTT applets

- **If** – Service *Webhooks*.

  Use `{event}` from `Prepare IFTTT's hooks` of this instruction. For example: `Event Name` = `travis-not-finished-pr`, `Event Name` = `travis-failed-pr`.
- **Then** – whatever you like. For actions with text ingredients use following for failed, success and not finished checks:
  - `Value1` – Author of the merge
  - `Value2` – Author of the pull-request
  - `Value3` – Link to pull-request

4.2.4 Logs

- AWS CloudWatch: https://console.aws.amazon.com/cloudwatch . Choice tab `Logs`
- IFTTT logs: https://ifttt.com/activity

4.3 Review bot for GitHub

This github bot posts review of pull-requests with odoo modules: list of updated files (installable and non-installable), new features to test (according to doc/changelog.rst file)

4.3.1 Create AWS Lambda function

Create lambda function with following settings:

- Runtime

  Use `Python 3.6`

- Environment variables
  - `GITHUB_TOKEN` – generate one in https://github.com/settings/tokens . Select scope `repo`.
  - `LOG_LEVEL` – optional. Set to DEBUG to get detailed logs in AWS CloudWatch.
- Trigger

  Use `API Gateway`. Once you configure it and save, you will see `API endpoint` under Api Gateway **details** section. Use option `Open`

  Now register the URL as webhook at github: https://developer.github.com/webhooks/creating/. Use following webhook settings:
  - **Payload URL** – the URL
  - **Content Type**: application/json
  - **Which events would you like to trigger this webhook?** – *Let me select individual events* and then select `[x] Pull request`
- Function Code
  - Use this commands:

```
mkdir /tmp/github-review-bot
cd /tmp/github-review-bot

pip3 install pyGithub -t .
wget https://gitlab.com/itpp/odoo-devops/raw/master/tools/github-review-bot/
↪lambda_function.py
wget https://gitlab.com/itpp/odoo-devops/raw/master/tools/github-review-bot/
↪text_tree.py
zip -r /tmp/github-review-bot.zip *
```

  - Then set **Code Entry type** to `Upload a .zip file` and select the created zip file
- Basic settings
  - Change time running function to 50 sec – `Timeout` (default 3 sec)

4.3.2 Logs

- AWS CloudWatch: https://console.aws.amazon.com/cloudwatch . Choose tab `Logs`

4.3.3 Roadmap

- TODO: Deleted files should be listed with tag `[DELETED]`
- TODO: Renamed files should be listed with tag `[RENAMED from path/to/original-file]` (for new files) and `[RENAMED]` (for original place of the file)
- TODO: New modules (e.g. root `__init__.py` didn't exist) should be marked with tag `[NEW]`, e.g. `├─ [NEW] pos_debt_notebook/`
- TODO: Ported modules (`installable` attribute is changed from False to True) should be marked with tag `[PORT]`, e.g. `├─ [PORT] pos_debt_notebook/`
- Updating review doesn't work without write access to the repo: github API returns 404. See https://gitlab.com/itpp/odoo-devops/issues/3

4.4 Notifications to Telegram Group

In this example we make a bot, that will send notifications to telegram group on new issues. You can slightly change the script to use other type of events.

4.4.1 Telegram Bot

- In telegram client open BotFather
- Send /newbot command to create a new bot
- Follow instruction to set bot name and get bot token
- Keep your token secure and store safely, it can be used by anyone to control your bot

4.4.2 Telegram Group

Add created bot to the group, where it will send notifications

You will need Group ID. To get one, temporarly add Get My ID bot to the group.

4.4.3 Secrets

Add following secrets

- `TELEGRAM_TOKEN` – bot token
- `TELEGRAM_CHAT_ID` – Group ID. Normally, it's negative integer

4.4.4 Github Actions

Create `.github/workflows/main.yml` file (you can also use `[Set up a workflow yourself]` button at `Actions` tab of the repository page)

```
name: Telegram Notifications

on:
  issues:
    types: [opened, reopened, deleted, closed]

jobs:
  notify:

    runs-on: ubuntu-latest

    steps:
    - name: Send notifications to Telegram
      run: curl -s -X POST https://api.telegram.org/bot${{ secrets.TELEGRAM_TOKEN }}/
↪sendMessage -d chat_id=${{ secrets.TELEGRAM_CHAT_ID }} -d text="${MESSAGE}" >> /dev/
↪null
      env:
        MESSAGE: "Issue ${{ github.event.action }}: \n${{ github.event.issue.html_url␣
↪}}"
```

4.4.5 Try it out

- Create new issue
- RESULT: bot sends a notification

CHAPTER 5

IFTTT

5.1 GitHub Integration with IFTTT

5.1.1 Trigger Travis Success / Failure

Prepare IFTTT's hooks

- Log in / Sign up at https://ifttt.com/
- Click on `Documentation` button here: https://ifttt.com/maker_webhooks
- Replace `{event}` with event name, for example `travis-success-pr`. Do the same for another event, for example `travis-failed-pr` and `travis-failed-branch`. Save the links you got.

Create AWS Lambda function

Create lambda function with following settings:

- Runtime

  Use `Python 2.7`
- Environment variables
  - `GITHUB_TOKEN` – generate one in https://github.com/settings/tokens . No settings are needed for public repositories.
  - `IFTTT_HOOK_GREEN_PR`, `IFTTT_HOOK_RED_PR`, `IFTTT_HOOK_RED_BRANCH` – use IFTTT's hooks.
  - `IGNORE_BRANCHES` – optional. List of branches separated by comma to ignore to notify.
  - `LOG_LEVEL` – optional. Set to `DEBUG` to get detailed logs in AWS CloudWatch.

- Trigger

  Use `API Gateway`. Once you configure it and save, you will see `API endpoint` under Api Gateway **details** section. Use option `Open`

  Now register the URL as webhook at github: https://developer.github.com/webhooks/creating/. Use following webhook settings:

  - **Payload URL** – the URL
  - **Content Type**: application/json
  - **Which events would you like to trigger this webhook?** – *Let me select individual events* and then select `[x] Check runs`

- Function Code
  - Copy-paste this code: https://gitlab.com/itpp/odoo-devops/raw/master/tools/github-ifttt/lambda_function.py

Create IFTTT applets

- **If** – Service *Webhooks*

  Use `{event}` from `Prepare IFTTT's hooks` of this instruction. For example: `Event Name = travis-success-pr`, `Event Name = travis-failed-pr` and `Event Name = travis-failed-branch`

- **Then** – whatever you like. For actions with text ingredients use following:
  - `Value1` – Author of the pull-request
  - `Value2` – Link to pull-request
  - `Value3` – Link to the travis check

  and for checks of stable branch:
  - `Value1` – Name of the branch
  - `Value2` – Name of the repo
  - `Value3` – Link to the travis check

Travis settings

- Update `.travis.yml` to get a notification in lambda when travis check is finished. You can configure either always notify on failure or only when previous check was successful. Check Travis Documentation for details: https://docs.travis-ci.com/user/notifications/#configuring-webhook-notifications
- Look it for example:

```
notifications:
    webhooks:
        on_failure: change
    urls:
      - "https://9ltrkrik2l.execute-api.eu-central-1.amazonaws.com/default/
↪TriggerTravis/"
```

Logs

- AWS CloudWatch: https://console.aws.amazon.com/cloudwatch . Choice tab `Logs`
- IFTTT logs: https://ifttt.com/activity

CHAPTER 6

Lint Checks

6.1 Preparation

Execute once per computer

```
cd
git clone https://github.com/it-projects-llc/maintainer-quality-tools.git
cd maintainer-quality-tools/travis
LINT_CHECK="1" sudo -E bash -x travis_install_nightly 8.0

echo "export PATH=\$PATH:$(pwd)/" >> ~/.bashrc
source ~/.bashrc
```

6.2 Running checks

```
cd YOUR-PATH/TO/REPOSTORY
LINT_CHECK="1" TRAVIS_BUILD_DIR="." VERSION="12.0" travis_run_tests 12.0
```

CHAPTER 7

Remote Development

The section contains instructions to setup remote development environment. That is developer runs odoo and probably other tools on remote server rather on his machine. Advantages of this approach are:

- easy way to provide big computing capacity
- the same environment from any device
- easy way to demonstrate work

7.1 Usage

7.1.1 SSH agent forwarding

To send commit or get access to private repositories you can use either login-password authentication or ssh keys. In later case you can face a problem to do it on remote server, because your private ssh key is not installed there. The good news is that you don't need to do it. You can "forward ssh keys". Just add `-A` to your ssh command or add following lines to your ssh config (`~/.ssh/config`) on your (local) computer:

```
Host your.dev.server.example.com
  ForwardAgent yes
```

Then connect to your server and type to test:

```
ssh -T git@github.com
```

For more information see: https://developer.github.com/guides/using-ssh-agent-forwarding/

Putty users (Windows)

- install Pageant SSH agent (pageant.exe) https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html
- add your keys to Pageant SSH

- enable ssh agent forwarding in putty settings

7.1.2 How to mount local files on a server

sshfs

On your local machine:

```
# Step 1. Install ssh server on your local machine
# TODO
# Step 2. Configure ssh keys on you local machine
cat cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
# Step 3. Connect to your server
ssh USERNAME@SERVER -p 22 -A -R 2222:localhost:22
```

On your remote server:

```
# Step 4. Mount your directory on remote server
# about allow_other check this: https://github.com/moby/moby/issues/27026
↪#issuecomment-253579983
sshfs -p 2222 -o idmap=user,nonempty,allow_other \
             LOCALUSERNAME@127.0.0.1:/PATH/TO/LOCAL/FOLDER /PATH/TO/REMOTE/FOLDER

# to unmount:
fusermount -u /PATH/TO/REMOTE/FOLDER
```

References

- https://superuser.com/questions/616182/how-to-mount-local-directory-to-remote-like-sshfs

7.1.3 How to edit server files locally

```
sshfs -p 22 -o idmap=user,nonempty USERNAME@REMOTE-SERVER:/path/to/REMOTE/folder /
↪path/to/LOCAL/folder
```

7.1.4 Remote desktop via X2GO

Deploying X2GO server

x2go allows you to run remotely browser (or any other application on x-server)

- Connect to your server:
- install x2go server :

```
sudo add-apt-repository ppa:x2go/stable && \
sudo apt-get update && \
sudo apt-get install -y x2goserver x2goserver-xsession
```

- install desktop environment you prefer, e.g. LXDE:

```
sudo apt-get install lubuntu-desktop
# choose lightdm
```

- Install browser Pale Moon

```
# http://linux.palemoon.org
sudo sh -c "echo 'deb http://download.opensuse.org/repositories/home:/stevenpusser/
↪xUbuntu_18.04/ /' > /etc/apt/sources.list.d/home:stevenpusser.list" && \
sudo apt-get update && \
sudo apt-get install palemoon
```

X2GO Client

- install `x2goclient`

  Ubuntu:

  ```
  sudo add-apt-repository ppa:x2go/stable && \
  sudo apt-get update && \
  sudo apt-get install x2goclient
  ```

  References:

  - https://www.howtoforge.com/tutorial/x2go-server-ubuntu-14-04/
  - http://wiki.x2go.org/doku.php/doc:installation:x2goclient

- Run client:

```
x2goclient
```

- create a new session with the settings below and connect to it (we assume that you have user named "noroot" with ssh keys configured):

```
Host : YOUHOST
Port : 22
Session type: LXDE
[x] Try auto Login
Input / Output: Use Whole Display
Username: noroot
```

7.2 Containers administration

7.2.1 LXD Containers

```
# For understanding LXC see https://wiki.debian.org/LXC

# Based on:
# lxd + docker: https://stgraber.org/2016/04/13/lxd-2-0-docker-in-lxd-712/
# lxd network (static ip): https://stgraber.org/2016/10/27/network-management-with-
↪lxd-2-3/
LXD_NETWORK="dev-network2"

# install lxd 2.3+
```

(continues on next page)

(continued from previous page)

```
apt-get install software-properties-common iptables-persistent

add-apt-repository ppa:ubuntu-lxc/lxd-stable
apt-get update
apt-get dist-upgrade
apt-get install lxd

# init lxd
lxd init

# init network
lxc network create ${LXD_NETWORK}
lxc network show ${LXD_NETWORK}  # check ipv4.address field


############################
# Per each Developer
GITHUB_USERNAME="yelizariev"
CONTAINER="${GITHUB_USERNAME}"
SERVER_DOMAIN="${GITHUB_USERNAME}.dev.it-projects.info"
NGINX_CONF="dev-${GITHUB_USERNAME}.conf"
LOCAL_IP="10.37.82.100"  # use one from network subnet
PORT="10100"  # unique per each developer

# https://discuss.linuxcontainers.org/t/docker-cannot-write-to-devices-allow/998/3
read -r -d '' RAW_LXC <<EOF
lxc.apparmor.profile=unconfined
lxc.mount.auto="proc:rw sys:rw cgroup:rw"
lxc.cgroup.devices.allow=a
lxc.cap.drop=
EOF
lxc init ubuntu-daily:18.04 ${CONTAINER} -p default && \
lxc network attach ${LXD_NETWORK} ${CONTAINER} eth0 && \
lxc config device set ${CONTAINER} eth0 ipv4.address ${LOCAL_IP} && \
lxc config set ${CONTAINER} security.privileged true && \
# allow run docker in previliged mode.
# https://discuss.linuxcontainers.org/t/failed-to-write-a-rwm-to-devices-allow-
↪operation-not-permitted-in-privileged-container/925/3
lxc config set ${CONTAINER} raw.lxc "$RAW_LXC"


# forward ssh port
iptables -t nat -A PREROUTING -p tcp --dport ${PORT} -j DNAT \
  --to-destination ${LOCAL_IP}:22

# save iptables record. Otherwise it's disappeared after rebooting
sudo netfilter-persistent save
sudo netfilter-persistent reload

PASS="$(< /dev/urandom tr -dc _A-Za-z0-9 | head -c${1:-32};echo;)"
lxc start ${CONTAINER}

lxc exec ${CONTAINER} -- apt-get update && \
lxc exec  ${CONTAINER} -- apt dist-upgrade -y

# colorize prompt:
lxc exec ${CONTAINER} -- sed -i "s/#force_color_prompt=yes/force_color_prompt=yes/" /
↪root/.bashrc && \
```

(continues on next page)

(continued from previous page)

```
lxc exec ${CONTAINER} -- sed -i "s/01;32m/01;36m/" /root/.bashrc && \
# install some packages
lxc exec  ${CONTAINER} -- apt install docker.io htop python3-pip -y && \
lxc exec  ${CONTAINER} -- ln -s /usr/bin/pip3 /usr/bin/pip && \
lxc exec  ${CONTAINER} -- pip install odooup && \
# https://docs.docker.com/v17.09/compose/install/#install-compose
lxc exec  ${CONTAINER} -- curl -L https://github.com/docker/compose/releases/download/
↪1.18.0/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose && \
lxc exec  ${CONTAINER} -- chmod +x /usr/local/bin/docker-compose && \
# update git. See https://github.com/xoe-labs/odooup/issues/8
# TODO: this may not be needed in ubuntu 18
lxc exec ${CONTAINER} -- add-apt-repository ppa:git-core/ppa -y && \
lxc exec ${CONTAINER} -- apt-get update && \
lxc exec ${CONTAINER} -- apt-get install git -y && \
lxc exec ${CONTAINER} -- adduser noroot --disabled-password --gecos "" && \
lxc exec ${CONTAINER} -- mkdir -p /root/.ssh && \
lxc exec ${CONTAINER} -- bash -c "curl --silent https://github.com/${GITHUB_USERNAME}.
↪keys >> /root/.ssh/authorized_keys" && \
# access for noroot
lxc exec ${CONTAINER} -- bash -c "echo $PASS > /root/noroot-password"     && \
lxc exec ${CONTAINER} -- bash -c "echo noroot:$PASS | chpasswd "     && \
lxc exec ${CONTAINER} -- sudo -u "noroot" bash -c "mkdir -p /home/noroot/.ssh" && \
lxc exec ${CONTAINER} -- sudo -u "noroot" bash -c "curl --silent https://github.com/$
↪{GITHUB_USERNAME}.keys >> /home/noroot/.ssh/authorized_keys" && \
lxc exec ${CONTAINER} -- sudo -u "noroot" sed -i "s/01;32m/01;93m/" /home/noroot/.
↪bashrc && \
# Manage Docker as a non-root user https://docs.docker.com/install/linux/linux-
↪postinstall/
lxc exec ${CONTAINER} -- usermod -aG docker noroot && \
lxc exec ${CONTAINER} -- usermod -aG sudo noroot && \
lxc exec ${CONTAINER} -- locale-gen --purge en_US.UTF-8 && \
lxc exec ${CONTAINER} -- bash -c "echo -e 'LANG=\"en_US.UTF-8\"\nLANGUAGE=\"en_US:en\
↪"\n' > /etc/default/locale"

lxc config device add ${CONTAINER} sharedcachenoroot disk path=/home/noroot/.cache␣
↪source=/var/lxc/share/cache && \
lxc stop ${CONTAINER} && \
lxc start ${CONTAINER}

## nginx on host machine
cd /tmp/
curl -s https://gitlab.com/itpp/odoo-devops/raw/master/docs/remote-dev/lxd/nginx.conf␣
↪> nginx.conf
sed -i "s/NGINX_SERVER_DOMAIN/.${SERVER_DOMAIN}/g" nginx.conf
sed -i "s/SERVER_HOST/${LOCAL_IP}/g" nginx.conf
cp nginx.conf /etc/nginx/sites-available/${NGINX_CONF}
ln -s /etc/nginx/sites-available/${NGINX_CONF} /etc/nginx/sites-enabled/${NGINX_CONF}
# then restart nginx in a usual way

####################
# Control commands

# delete container
lxc delete CONTAINER-NAME

# see iptables rules
iptables -L -t nat
```

(continues on next page)

(continued from previous page)

```
# delete nat rule
iptables -t nat -D PREROUTING POSITION_NUMBER
```