

# ОСНОВНЫЕ ШАГИ

## 1. Создаем 2 виртуальные машины

По инструкции из файла на сайте <https://ds1.digitalenergy.online/portal/#/dashboard>

Логин irina\_kozlova\_1

## 2. Копируем сохраняем информацию о машинах

BPM1

NnrBtH2qU

локальный ip 192.168.3.11

публичный ip 176.118.165.63 2225

BPM2

KUcuDu7zE

локальный ip 192.168.3.10

публичный ip 176.118.165.63 2226

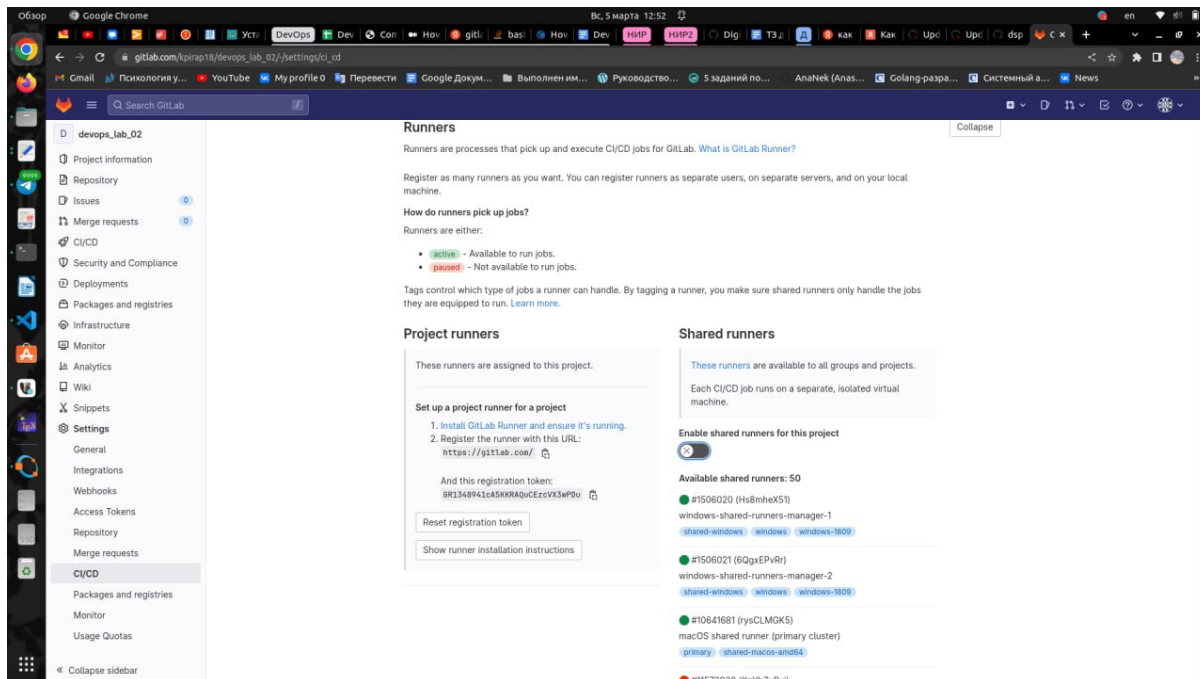
## 3. Создаем проект на гите

Создаем ssh (чтобы клонировать репозиторий по ssh)

Клонировем проект

Setting – Ci CD — runners

убираем флажок – чтобы гитлаба свои раннеры (не на серверах гитлаба)



## 4. На вмл

shell раннер – все команды, которые мы указываем выполняются на машине, где запущен раннер. Когда вставили токен, то мы привязали раннер к репозиторию.

### 4.1. Подключаемся через терминал

4.1.1. `ssh -p 2225 user@176.118.165.63`

4.1.2.

### 4.2. Устанавливаем nginx

4.2.1. `sudo apt update`

4.2.2. `sudo apt install nginx`

4.2.3. на все тыкаем окей и как появляеся кенсел, то тыкаем туда

### 4.3. Устанавливаем imagemagick, который будет генерировать статику (картинку из текста)

4.3.1. `sudo apt install imagemagick`

### 4.4. Создаем gitlab-runner

<https://kyosenaharidza.medium.com/how-to-create-your-own-gitlab-runner-a37cd54bae33>

4.4.1. `mkdir Downloads`

4.4.2. `cd Downloads`

4.4.3. скачиваем установщик гитлаб раннера

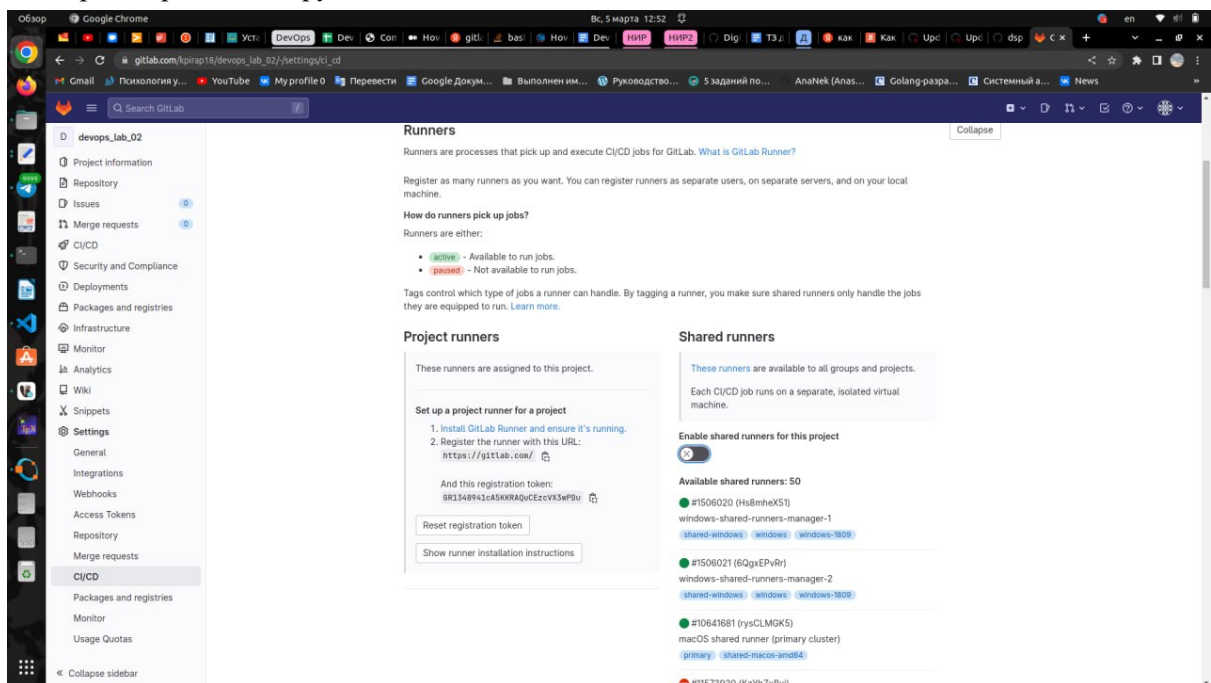
`curl -LJO "https://gitlab-runner-downloads.s3.amazonaws.com/latest/deb/gitlab-runner_amd64.deb"`

4.4.4. `sudo dpkg -i ./gitlab-runner_amd64.deb` – устанавливаем гитлаб раннер

4.4.5. `sudo gitlab-runner register` – регистрируем гитлаб раннер

4.4.6. далее вводим команду гитлаба (с которым раннер будет работать) <https://git.iu7.bmstu.ru/>

4.4.7. токен регистрации копируем отсюда



4.4.8. потом ничего не вводим

4.4.9. `comma-separated`

`sh, linux` – теги по которому он будет выбирать что ему делать

### 4.5. Все описанные команды:

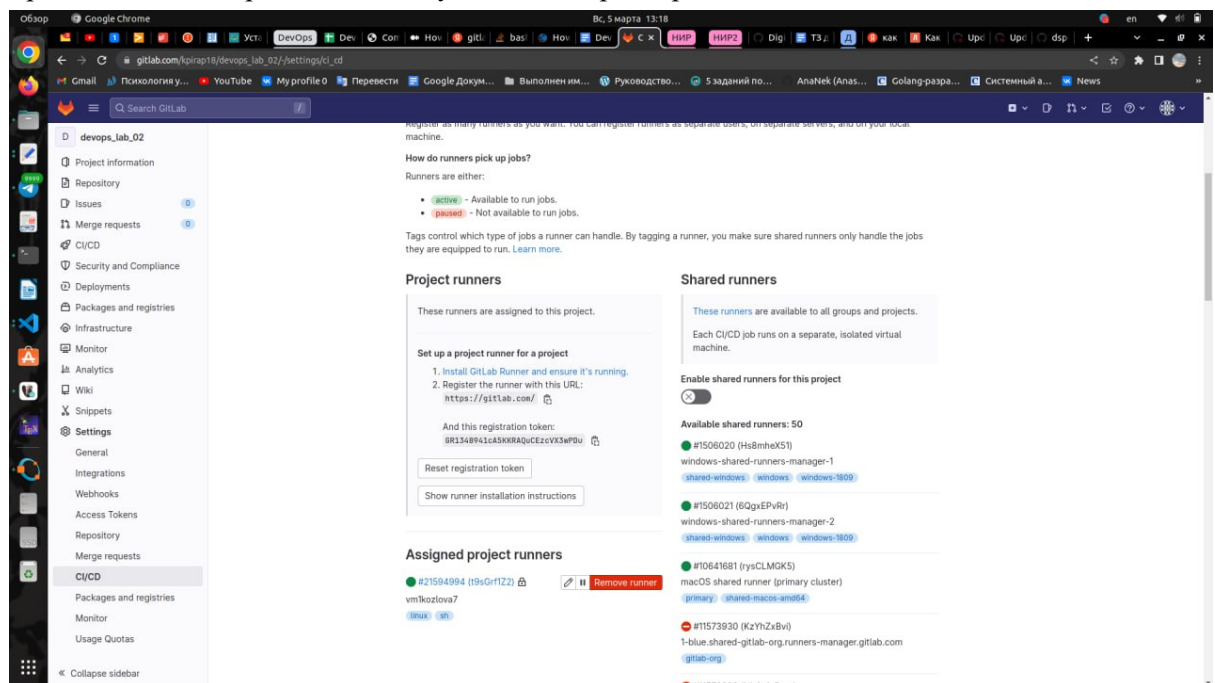
```
Обзор Терминал
user@vm1kozlova7: ~/Downloads
Selecting previously unselected package gitlab-runner.
(Reading database ... 96776 files and directories currently installed.)
Preparing to unpack ./gitlab-runner_amd64.deb ...
Unpacking gitlab-runner (15.9.1) ...
Setting up gitlab-runner (15.9.1) ...
GitLab Runner: creating gitlab-runner...
Home directory skeleton not used
Runtime platform
gitlab-runner: the service is not installed arch=amd64 os=linux pid=24757 revision=d540b510 version=15.9.1
Runtime platform
gitlab-ci-multi-runner: the service is not installed arch=amd64 os=linux pid=24764 revision=d540b510 version=15.9.1
Runtime platform
Runtime platform arch=amd64 os=linux pid=24784 revision=d540b510 version=15.9.1
Runtime platform arch=amd64 os=linux pid=24846 revision=d540b510 version=15.9.1
INFO: Docker installation not found, skipping clear-docker-cache
user@vm1kozlova7: ~/Downloads$ sudo gitlab-runner register
Runtime platform arch=amd64 os=linux pid=24864 revision=d540b510 version=15.9.1
Running in system-mode.

Enter the GitLab instance URL (for example, https://gitlab.com/):
https://gitlab.com/
Enter the registration token:
GR1348941cA5KKRAQuCEzcVX3wP0u
Enter a description for the runner:
[vm1kozlova7]:
Enter tags for the runner (comma-separated):
sh,linux
Enter optional maintenance note for the runner:

WARNING: Support for registration tokens and runner parameters in the 'register' command has been deprecated in GitLab Runner 15.6 and will be replaced with support for authentication tokens. For more information, see https://gitlab.com/gitlab-org/gitlab/-/issues/380872
Registering runner... succeeded runner=GR1348941cA5KKRAQ
Enter an executor: docker+machine, docker-ssh+machine, kubernetes, docker-ssh, parallels, shell, ssh, virtualbox, custom, docker, instance:
shell
Runner registered successfully. Feel free to start it, but if it's running already the config should be automatically reloaded!

Configuration (with the authentication token) was saved in "/etc/gitlab-runner/config.toml"
user@vm1kozlova7: ~/Downloads$
```

4.6. При обновлении страницы можно увидеть, что раннер появился



## 5. Стадия build

На стадии build мы создаем динамическую статику и записываем переменные Gitlab CI, чтобы потом их вывести в index.html.

Стадия build выполняется на всех ветках.

### 5.1. Создаем файл .gitlab-ci.yml в репозитории гитлаба

5.2. touch .gitlab-ci.yml

5.3. вставляем туда

```
stages:
  - build
  - tests
```

- deploy

build:

stage: build

script:

- mkdir build

- mkdir build/img

- echo "The job's stage is '\$CI\_JOB\_STAGE'. The commit's author '\$CI\_COMMIT\_AUTHOR'. The commit's ref\_name '\$CI\_COMMIT\_REF\_NAME'. The commit's short\_sha '\$CI\_COMMIT\_SHORT\_SHA'" > build/variables.txt

- convert -size 400x400 xc:white -font "FreeMono" -pointsize 12 -fill black -draw @texts/pic1.txt build/img/pic1.png

- convert -size 400x400 xc:white -font "FreeMono" -pointsize 12 -fill black -draw @texts/pic2.txt build/img/pic2.png

- convert -size 400x400 xc:white -font "FreeMono" -pointsize 12 -fill black -draw @texts/pic3.txt build/img/pic3.png

tags:

- sh

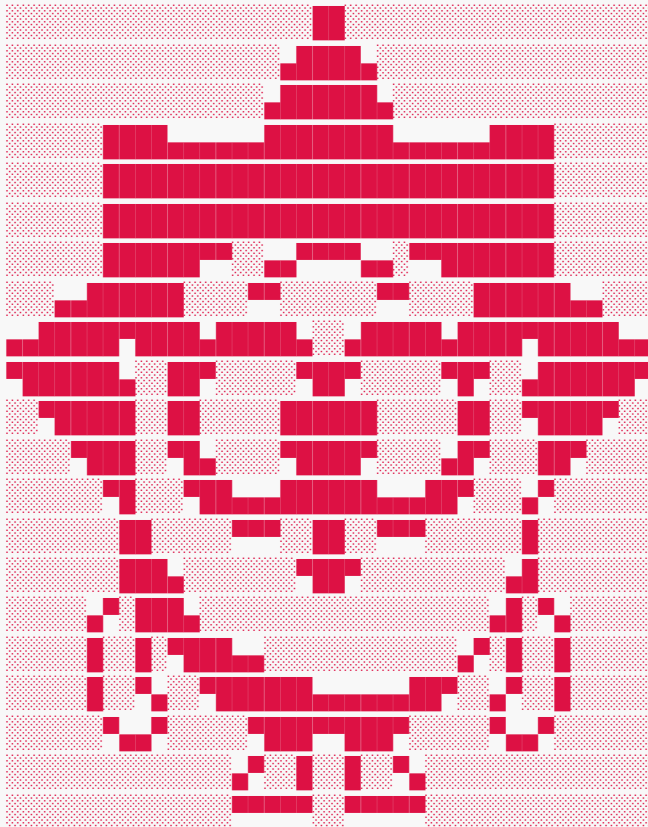
artifacts:

paths:

- build/

5.4. В качестве текста из которого будем формировать картинки берем изображение ежика/лисы/кота и сохраняем в файлы pic1 pic2 pic3 соответственно в папку texts

text 40,40 "



"

### 5.5. Создаем ветку lab\_02,

так как “Стадии build и test должны быть автоматическими, доступными для любой ветки”. Поэтому создаем ветку, в примере это lab\_02.

### 5.6. Пушим изменения в эту ветку (add commit push)

5.7. Зашли в результаты pipeline и видим, что команда build сработала

## 6. Стадия test

На стадии тестирования (выполняется автоматически на всех ветках) будет проверяться корректность конфиг-файла nginx.

Данный конфиг хранится в репозитории, а когда раннер будет выполнять стадию deploy, он скопирует его на вторую машину: сначала в /home/user/nginx.conf, а потом оттуда в /etc/nginx/nginx.conf. После этого на второй машине будет запущен nginx с эти конфигом.

test должен запускаться из всех веток

### 6.1. Создаем конфиг nginx/nginx.conf в репозитории (на локальной машине)

```
user user;

events {
    worker_connections 768;
    # multi_accept on;
}

http {
    server {
        location / {
            root /home/user/static/;
        }

        location /status {
            proxy_no_cache 1;
            stub_status;
        }
    }
}
```

### 6.2. В .gitlab-ci.yml добавляем

```
validate_config:
  stage: tests
  script:
    - sudo nginx -t -c 'pwd'/nginx/nginx.conf # -t проверит синтаксис -с путь до файла
  tags:
    - sh
```

```
needs:
  - build
```

### 6.3. Проблема – мы используем `sudo` для которого нужен пароль

- 6.4. Надо разрешить `gitlab-runner` выполнять `sudo` без пароля. Для этого на `vm1` (так как раннер будет выполняться там) выполняем:

### 6.5. Добавляем пользователя `gitlab-runner` в группу `sudo`

```
sudo usermod -a -G sudo gitlab-runner
```

### 6.6. Открываем файл `etc/sudoers` (файл с описанием прав)

```
sudo visudo
```

- 6.7. В конец файла вставляем

```
gitlab-runner ALL=(ALL) NOPASSWD: ALL
```

– без пароль вызывать `sudo` на все команды (работаем через `nano`)

- 6.8. Пушим изменения в ветку `lab_02`

- 6.9. Проверяем, что стадия `test` сработала

## 7. Стадия `deploy`

При подключении к 2 машине нас спрашивают – доверяем ли мы подключению. Нам надо это сделать автоматически, чтобы показать, что мы доверяем. (это ниже)

### 7.1. Заходим на вторую машину

```
ssh -p 2225 user@176.118.165.63
```

### 7.2. На 2 машине надо поставить `nginx`

- 7.3. `sudo apt install nginx`

- 7.4. Добавляем в `.gitlab-ci.yml`

```
deploy_image:
  stage: deploy
  script:
    - scp -r static/ user@192.168.3.11:/home/user/
    - scp -r build/img/ user@192.168.3.11:/home/user/static/
    - scp build/variables.txt user@192.168.3.11:/home/user/static/variables.txt
    #- scp nginx/nginx.conf user@192.168.3.11:/home/user/nginx.conf
    #- ssh user@192.168.3.11 'sudo cp /home/user/nginx.conf /etc/nginx/nginx.conf'
```

```
- ssh user@192.168.3.11 'sudo nginx -s reload'
tags:
- sh
needs:
- build
rules:
- if: ($CI_COMMIT_BRANCH == "main" || $CI_COMMIT_BRANCH == "develop")
  when: manual
```

- 7.5. Можно закомментировать строки последние, чтобы понять, что это все работает (что просто ветка deploy реально копирует данные с одной машины на другую)
- 7.6. Создаем файл index.html и кладем его в папке static в репозиторий

```
<!DOCTYPE html>
<meta charset="utf-8">
<html>
<head>
  KPIAP18
</body>
</html>
```

## 8. Устанавливаем доверие между машинами

### 8.1. Обнаружили проблему

- спрашивает разрешения на подключение (доверять ли ей)
- спрашивает пароль от машины

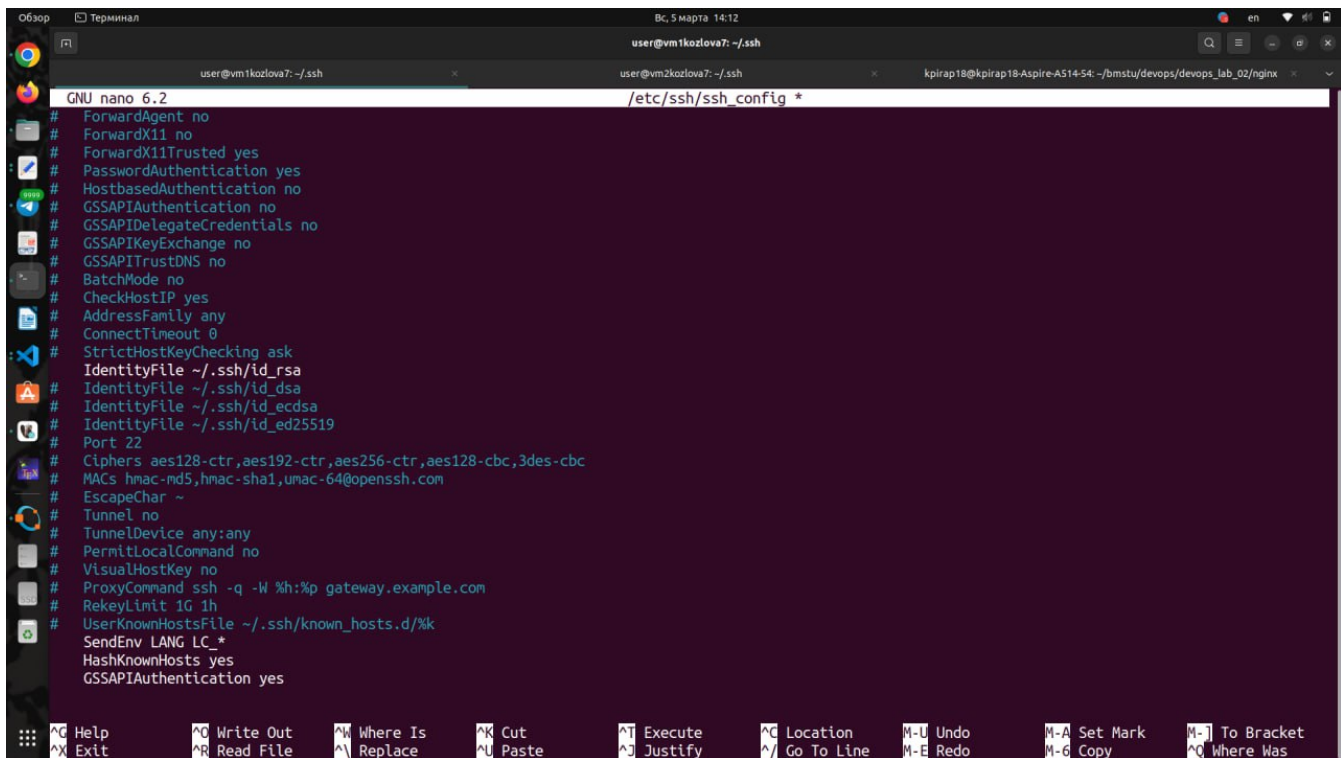
### 8.2. Решаем проблему “спрашивает пароль от машины”

- 8.3. На wrm1
- 8.4. `sudo su - gitlab-runner` – подключаемся под пользователем гитлаб раннер
- 8.5. создаем папку `.ssh` (`mkdir`)
- 8.6. заходим в эту папку
- 8.7. Пишем `ssh-keygen` – создать ssh ключи
- 8.8. везде нажимаем энтер
- 8.9. Избавляемся от ввода пароля – для этого надо указать `authorized key` (которыми он доверяет)
- 8.10. На wrm2 – надо добавить публичный ключ, который создали на wrm1 (берем его с первой машины – там где создали его)
- 8.11. открываем через nano файл `authorize_keys` и вставляем туда нам ключ
  - 8.11.1. если к wrm2 подключаются с таким-то публичным ключом, то она доверяет и пароль вводить не просит

### 8.12. Решаем проблему “спрашивает разрешения на подключение (доверять ли ей)”

- 8.13. На wrm1 надо сделать
  - 8.13.1. – открыть файл `/etc/ssh/ssh_config` через `судо`
  - 8.13.2. – раскомментировать строку





```
GNU nano 6.2 /etc/ssh/ssh_config *
# ForwardAgent no
# ForwardX11 no
# ForwardX11Trusted yes
# PasswordAuthentication yes
# HostbasedAuthentication no
# GSSAPIAuthentication no
# GSSAPIDelegateCredentials no
# GSSAPIKeyExchange no
# GSSAPITrustDNS no
# BatchMode no
# CheckHostIP yes
# AddressFamily any
# ConnectTimeout 0
# StrictHostKeyChecking ask
# IdentityFile ~/.ssh/id_rsa
# IdentityFile ~/.ssh/id_dsa
# IdentityFile ~/.ssh/id_ecdsa
# IdentityFile ~/.ssh/id_ed25519
# Port 22
# Ciphers aes128-ctr,aes192-ctr,aes256-ctr,aes128-cbc,3des-cbc
# MACs hmac-md5,hmac-sha1,umac-64@openssh.com
# EscapeChar ~
# Tunnel no
# TunnelDevice any:any
# PermitLocalCommand no
# VisualHostKey no
# ProxyCommand ssh -q -W %h:%p gateway.example.com
# RekeyLimit 1G 1h
# UserKnownHostsFile ~/.ssh/known_hosts.d/%k
# SendEnv LANG LC_*
# HashKnownHosts yes
# GSSAPIAuthentication yes
```

– это мы сказали ssh чтобы он использовал этот ключ, который мы создали

8.14. На врм 1 вводим команду

8.15. `ssh user@192.168.3.46`

8.16. один раз вводим yes

8.17. Теперь у нас на vm1 в файле `~/.ssh/known_hosts` появилась информация о vm2. При подключении к штукам из этого файла, vm1 не задает вопросов о том, можно ли ей подключиться. (сохранили хост, к которому можно подключаться 100%) .

И если мы напишем команду подключения ко 2 машине – то у нас будет все работать и никакой пароль и вопрос о доверии не будут спрашивать

8.18. Раскомментируем

```
#- ssh user@192.168.3.20 'sudo cp /home/user/nginx.conf /etc/nginx/nginx.conf'
```

```
#- ssh user@192.168.3.20 'sudo nginx -s reload'
```

1 — файл `nginx.conf` копируем в стандартный файл `nginx.conf`

2 — перезапускаем `nginx`

8.19. Опять проблема, что используем `sudo`

8.20. На врм2

```
sudo usermod -a -G sudo user
```

8.21. Запускаем и вводим пароль

8.22. Далее `sudo visudo`

8.23. В конец файла добавляем

```
user ALL=(ALL) NOPASSWD: ALL
```

8.24. Теперь мы можем запускать команды и у нас не будет просить пароль



## 9. Продолжение стадии deploy

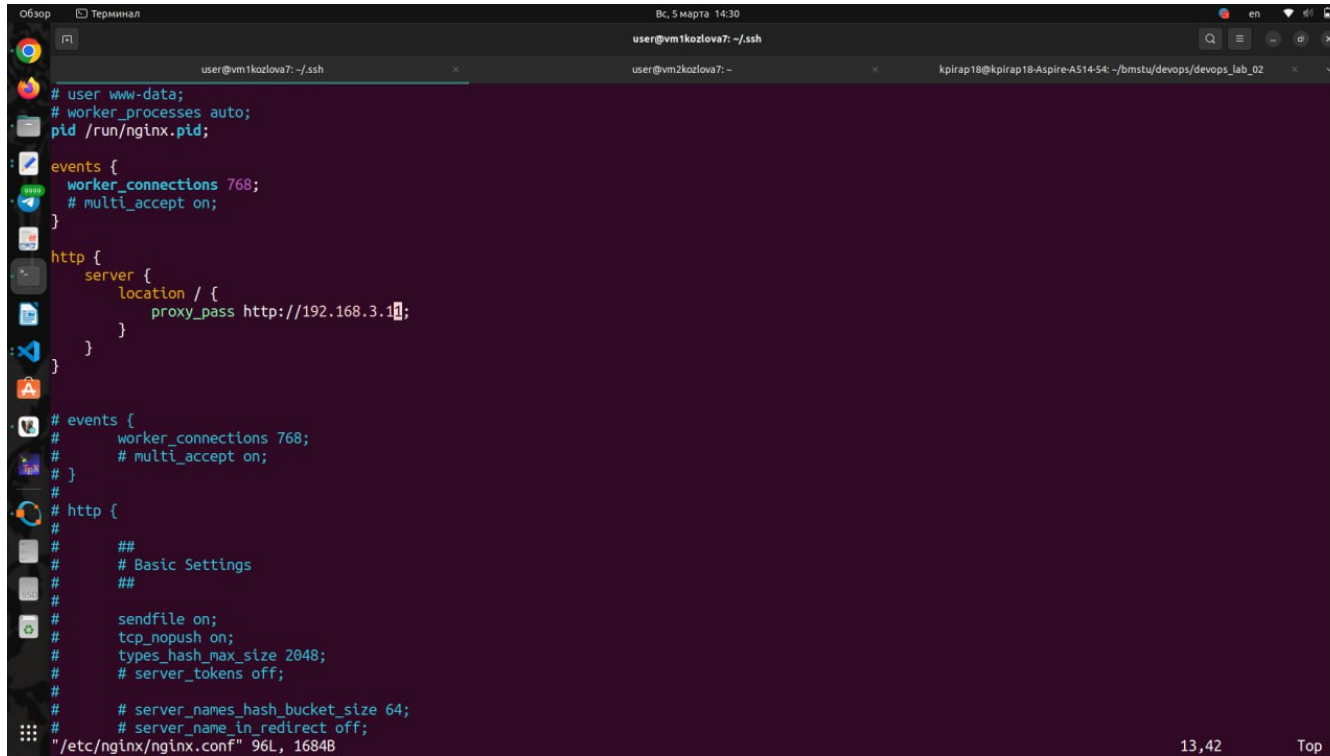
### 9.1. Проверка, что файлы копируются с врм1 на врм2

- 9.2. На врм2 создаем папку статик, куда мы копируем с первой машины
- 9.3. Запускаем
- 9.4. Пушим все в ветку lab\_02 – для того, чтобы проверить что это просто работает и работает верно
- 9.5. Немного ждем и ОП магия, у нас на врм2 появились все файлы, которые мы копировали с врм1

### 9.6. Теперь надо сделать как в первой лабе

- 9.7. на первой машине меняем nginx.conf

9.8.



```
Обзор Терминал BC, 5 марта 14:30 en
user@vm1kozlova7: ~/ssh
user@vm1kozlova7: ~
kpirap18@kpirap18-Aspire-A514-54: ~/bmstu/devops/devops_lab_02

# user www-data;
# worker_processes auto;
pid /run/nginx.pid;

events {
    worker_connections 768;
    # multi_accept on;
}

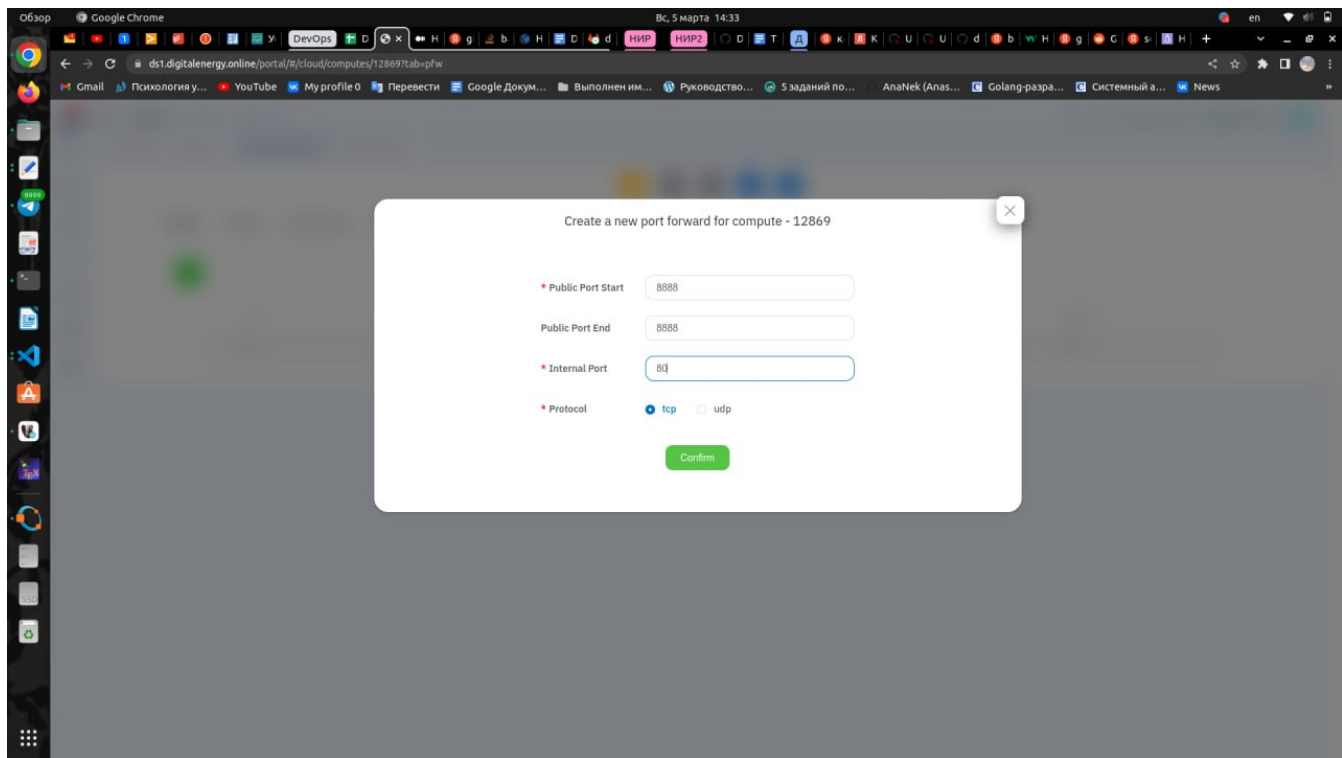
http {
    server {
        location / {
            proxy_pass http://192.168.3.10;
        }
    }
}

# events {
#     worker_connections 768;
#     # multi_accept on;
# }
#
# http {
#
#     ##
#     # Basic Settings
#     ##
#
#     sendfile on;
#     tcp_nopush on;
#     types_hash_max_size 2048;
#     # server_tokens off;
#
#     # server_names_hash_bucket_size 64;
#     # server_name_in_redirect off;
#
# }
"/etc/nginx/nginx.conf" 96L, 1684B
13,42 Top
```

- 9.9. Перезагружаем nginx на 1 машине

### 9.10. Нам надо из глоб инета попасть на врм2

- 9.11. nginx на первой машине слушает 80 порт. Нам надо на сайте добавить



9.12. На локальной машине пишем `http://176.118.165.63:8888` (порт)

9.13. Раскомментируем последние строки на локальной машине – чтобы запускать только из ветки `devops`

9.14. Пушим в репозиторий

9.15. Смотрим pipeline

9.16. А в ветке `lab_02` выполняется только 2 стадии

9.17. Создаем ветку `deploy` и пушим туда

9.18. Смотрим и он нам запустил pipeline из 3х стадий

9.19. `deploy` автоматом не запускается только если нажать, чтобы она запустилась

## 10. Отключаем `vm2` от интернета

1. Последнее – надо закрыть вторую машину из внешнего интернета

На второй машине делаем команду

```
sudo iptables -A INPUT -p tcp -s 192.168.3.0/24 --dport 22 -j ACCEPT
```

– все входящие запросы по протоколу `tcp` из локальной сети `192.168.3.0/24` принимать на порт `22`

```
user@vm2kozlova7:~$ sudo iptables -A INPUT -p tcp -s 192.168.3.0/24 --dport 22 -j ACCEPT
```

добавляем правило в таблицу `INPUT` (входящие пакеты) по протоколу `tcp` с маской `192.168.3.0/24` с портом назначения (куда хочет обратиться пакет) Что мы делаем – принимаем

```
user@vm2kozlova7:~$ sudo iptables -A INPUT -p tcp -s 192.168.3.0/24 --dport 80 -j ACCEPT
```

```
user@vm2kozlova7:~$ sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

все пакеты, которые на `22` порт приходят – принимаем

```
user@vm2kozlova7:~$ sudo iptables -A INPUT -j DROP
```

все остальные отклонить

Проверить – пингануть что-нибудь

Создать порт 80 и попробовать подключиться – не срабатывает