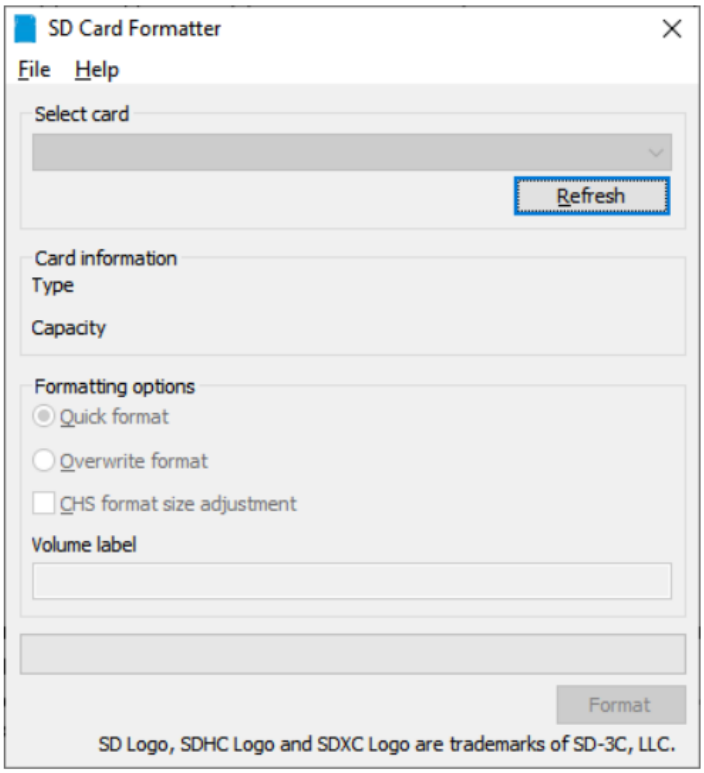
Для форматирования microSD-карты использовалась утилита SD Card Formatter. После того, как microSD карта вставлена в ПК, необходимо выбрать ее в выпадающем списке, затем нажать «Format» и согласиться с форматированием. Этот процесс представлен на «рисунке 13».



Для загрузки образа Armbian на microSD-карту использовалась утилита Win32DiskImager. Необходимо указать путь до образа для Orange Pi, выбрать отформатированную microSD-карту из выпадающего списка и нажать «Write». После завершения процесса, нажать «Exit», и вставить microSD-карту в Orange Pi.Этот этап показан на «рисунке 14».

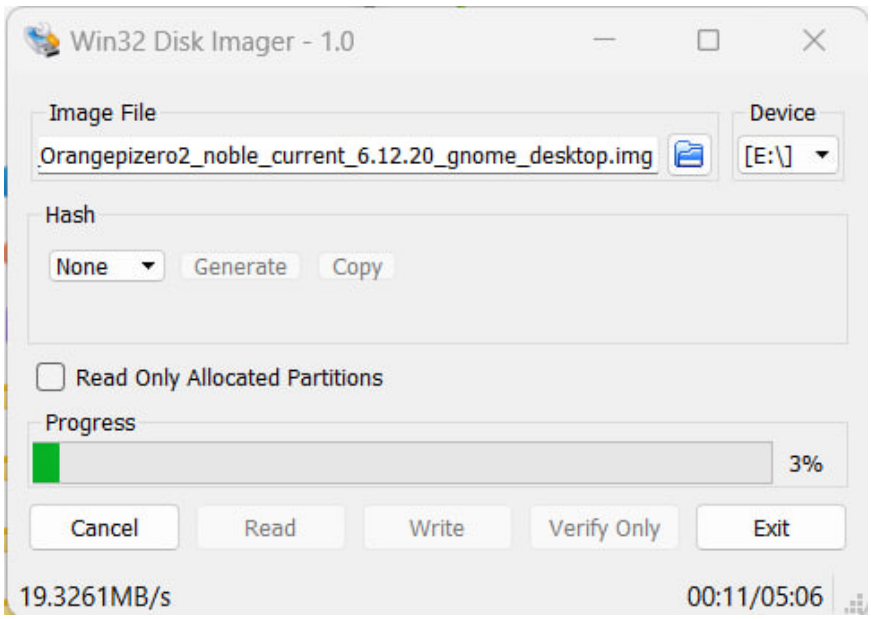


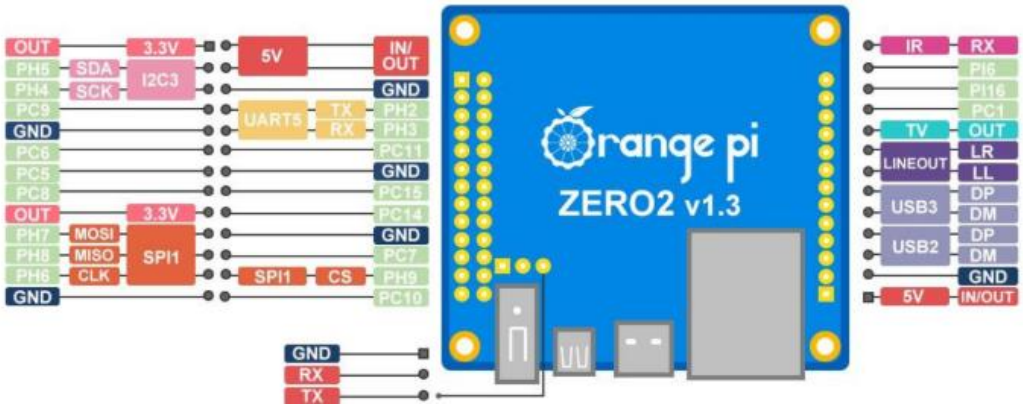
Рисунок 14 – Запись образа на карту памяти

Для того, чтобы подключить плату к ноутбуку необходимо использовать USB-TTL адапатер. Преобразователь USB-TTL на базе микросхемы PL2303 представляет собой компактный USB-модуль, эмулирующий последовательный интерфейс UART (RS-232), представленный на «рисунке 15». При подключении к USB-порту компьютера устройство создает виртуальный COM-порт. Компания Prolific в октябре 2012 года прекратила выпуск и поддержку микросхем PL2303X (Chip Rev A) и PL2303HX (Chip Rev A), заменив их новой моделью — PL2303TA. Тем не менее, на рынке до сих пор можно встретить преобразователи с маркировкой PL2303HX, однако они часто содержат поддельные чипы.

При использовании таких устройств на операционных системах Windows 7/8/8.1/10 автоматически загружается последняя версия драйвера, которая не поддерживает устаревшие или поддельные микросхемы. В результате в диспетчере устройств появляется желтый восклицательный знак с кодом ошибки 10, указывающий на проблему совместимости. Для корректной работы преобразователя на данных ОС необходимо установить устаревшую версию драйвера (v. 1.5.0), поддерживающую эти чипы.

PL2303HX

На «рисунке 16» представлена схема распиновки. Используя эту схему, можно понять, где находится необходимые нам точки подключения. Проводами необходимо соединить между собой RX и TX, TX и RX, GND и GND соответственно.



После успешного сопряжения необходимо установить актуальные драйвера для нашего адаптера. После их установки заходим в «Диспетчер устройств» и определяем какой COM порт имеет наш конвертер. На «рисунке 17» видно, что в нашем случае отображается COM3.

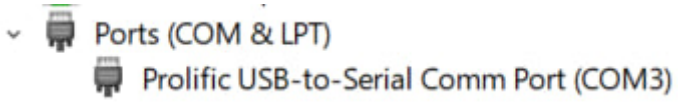


Рисунок 17 – Отображение порта в диспетчере устройств

На «рисунке 18» представлены настройки порта. Необходимо установить значение 115200 бит/секунду - скорость передачи данных по последовательному порту.

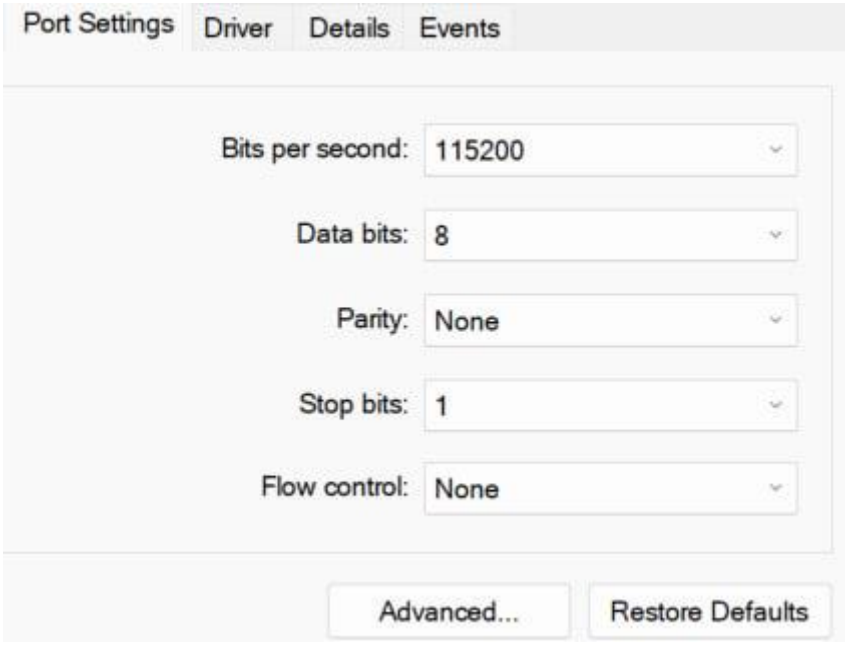


Рисунок 18 – Интерфейс настройки последовательного порта

На «рисунке 19» демонстрируется процесс создания нового пользователя н через командную строку . В ходе создания аккаунт вводим имя пользователя и пароль.

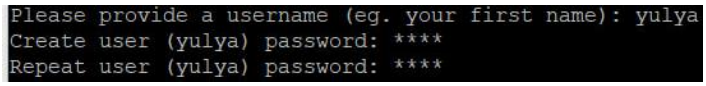


Рисунок 19 – Настройка системы и пользователя

На «рисунке 20» видим доступные сети и выбираем нужную нам после чего вводим от нее пароль для подключения.

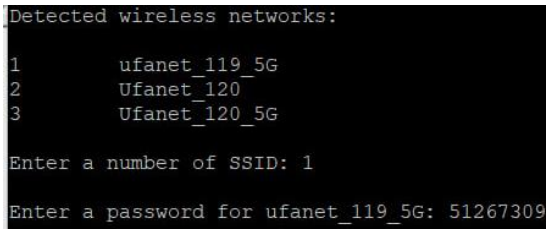


Рисунок 20 – Настройка доступа к wi-fi

Командой nmcli radio wifi on включаем адаптер беспроводной сети (Wi-Fi) на компьютере или одноплатном устройстве (например, Orange Pi). Команда не подключает устройство к конкретной сети, а только активирует возможность использования Wi-Fi. На «рисунке 21» активируем Wi-Fi интерфейс:

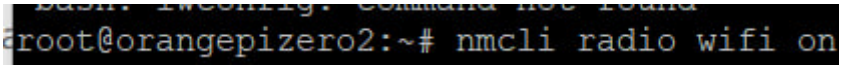


Рисунок 21 – Команда nmcli radio wifi on

Командой cat /etc/modprobe.d/r8723bs.conf просматриваем текущие настройки модуля ядра для Wi-Fi адаптера Realtek RTL8723BS, включая отключение энергосбережения и USBSS для обеспечения более стабильной работы Wi-Fi(рисунок 22).

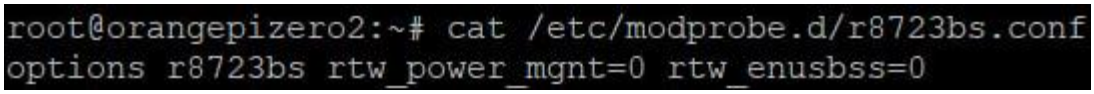


Рисунок 22 – Команда cat /etc/modprobe.d/r8723bs.conf

На «рисунке 23» проверяем командой ip a, что orange pi подключен к wi-fi и получил ip адрес.

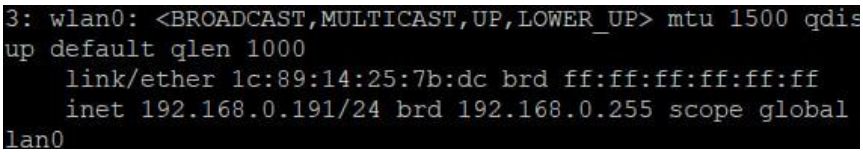


Рисунок 23 – Вывод команды ip a

После установки ОС важно обновить систему до последней версии командами apt update и apt upgrade, а также установить необходимые пакеты, такие как ssh, git и другие зависимости для работы Ansible. На «рисунке 24» показано выполнение вышеуказанной команды.

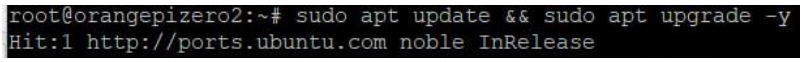


Рисунок 24 – Обновление системы до последней версии

После этого устанавливаем Python и его зависимости, так как Ansible работает через Python: `sudo apt install python3 python3-pip -y`. На «рисунке 25» показана установка нужных зависимостей.

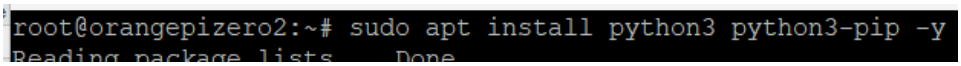


Рисунок 25 – Установка python

Создадим нового пользователя yulua командой useradd и после чего посмотрим в каких группах находится дефолтный пользователь. На «рисунке 26» показан этот процесс.

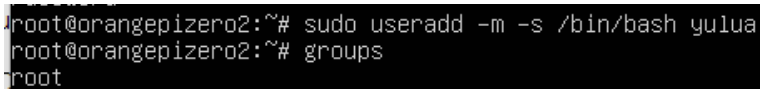


Рисунок 26 – Создание нового пользователя

Для корректной работы всех компонентов системы пользователю были назначены дополнительные системные группы, что позволило обеспечить полный доступ к необходимым ресурсам, таким как логирование, работа с дисками, сетевые интерфейсы и SSH. На «рисунке 27» командой usermod присваиваем пользователю нужные группы, командой passwd задаем пароль пользователя.

```
root@orangepizero2:~# sudo usermod -aG tty,disk,dialout,sudo,audio,video,plugdev,games,users,s
-journal,input,netdev,ssh yulua
root@orangepizero2:~# sudo passwd yulua
```

Рисунок 27 – Добавление нового пользователя в системные группы

Для полного удаления пользователя orangepi из системы была выполнена команда sudo userdel -r orangepi, которая удаляет аккаунт и все связанные с ним данные. Так как в некоторых случаях почтовый ящик пользователя может остаться в системе, была дополнительно использована команда sudo rm -rf /var/mail/orangepi — она гарантирует полное удаление всех данных пользователя. Этот процесс представлен на «рисунке 28».

```
root@orangepizero2:~# sudo userdel orangepi
root@orangepizero2:~# sudo rm -rf /var/mail/orangepi
```

Рисунок 28 – Удаление старого пользователя

Настраиваем время соответствующего нашему часовому поясу для корректной работой командой sudo timedatectl set-timezone. После чего проверяем правильность настроек. Этот процесс представлен на «рисунке 29».

```
root@orangepizero2:~# sudo timedatectl set-timezone Europe/Moscc
root@orangepizero2:~# timedatectl
               Local time: Tue 2025-04-29 10:31:14 MSK
           Universal time: Tue 2025-04-29 07:31:14 UTC
                 RTC time: Tue 2025-04-29 07:31:13
                Time zone: Europe/Moscow (MSK, +0300)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

Рисунок 29 – Установление корректного часового пояса

На «рисунке 30» показан процесс генерации rsa ключа на Orange Pi для подключения к серверам.

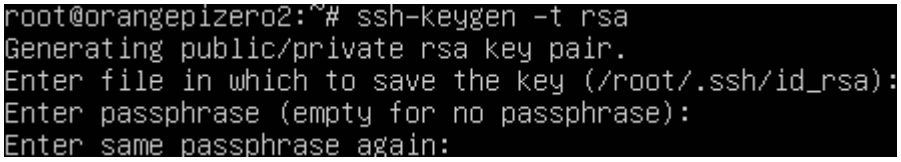


Рисунок 30 – Генерация ssh ключа

3.2 Создание инвентаря и плейбуков

На «рисунке 30» команда добавляет PPA (Personal Package Archive) для Ansible в систему. PPA — это дополнительный репозиторий, который содержит пакеты, которые не входят в стандартные репозитории Ubuntu или Debian. В данном случае, ppa:ansible/ansible — это официальный репозиторий Ansible, где можно установить актуальную версию Ansible.

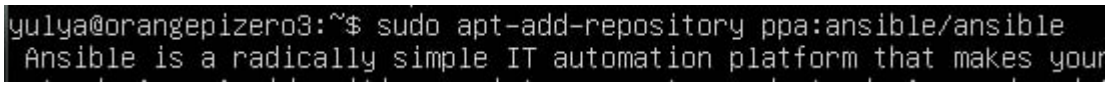


Рисунок 31 – Добавления репозитория ansible

На «рисунке 32» показан сам процесс установки ansible командой sudo apt-get install ansible.

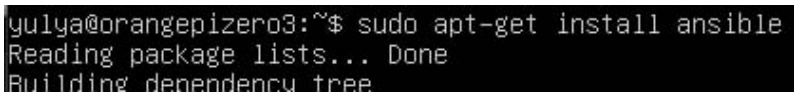


Рисунок 32 – Проверка установленной версии ansible

В файле ansible.cfg необходимо прописать данные, которые упрощают работу с ansible. Через него отключаем проверку SSH-ключей хостов, указываем пути к нашим ролям, ключам, методам авторизации, а также способу хранения кэша, что сокращает время тестирования новых ролей. Также для того, чтобы в будущем избежать ошибок с интерпретацией Python, указываем конкретную версию. Все эти настройки представлены на «рисунке 33».

Рисунок 33 – Отредактированный файл ansible.cfg

В файле инвентаря group_vars/all.yml удобно указывать переменные, общие для всех ролей. Это могут быть адреса серверов, имена служб, прокси и, в общем, все, что периодически нужно в наших ролях и что не хотелось бы дублировать. В нашем случае мы в него определяем и записываем, что Ansible будет использовать протокол SSH для подключения к удалённым хостам и включаем выполнение команд с повышенными правами, что необходимо для административных задач. После чего также указываем путь к приватному SSH-ключу, который будет использоваться для аутентификации при подключении к удалённым серверам. Этот файл показан на «рисунке 34».

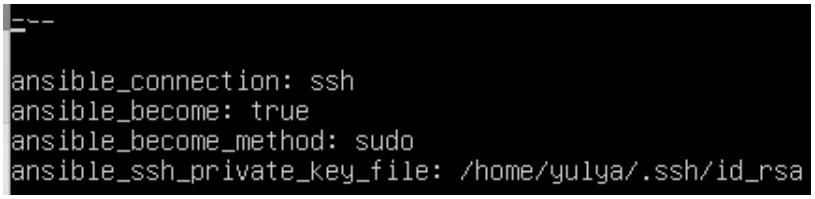


Рисунок 34 – Отредактированный файл инвентаря group_vars/all.yml

В hosts файле мы укажем хосты, а также группы, в которые входят эти хосты. Также в этом файле можно указать адреса и пароли наших хостов, но с точки зрения безопасности это не самое лучшее решение, поэтому их мы укажем в отдельных файлах, соответствующих именам хостов в директории host_vars. Я распределила сервера на несколько групп в зависимости от их функций, это очень удобно, так как позволяет все структурировать и в будущем при расширении инфраструктуры – будет легче заниматься перенастройкой при необходимости. Этот файл представлен на «рисунке 35».

Рисунок 35 – Файл hosts

После завершения настроек файлов инвентаря можем просмотреть как выглядит структура каталогов в этой папке командой tree. Результат команды показан на «рисунке 36».

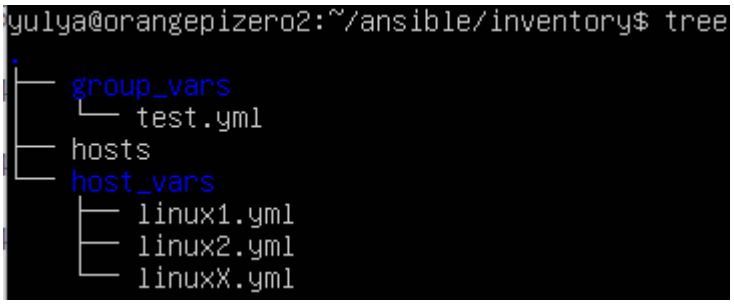


Рисунок 36 – Структура каталогов в папке inventory

В директории host_vars мы создаем файл для каждого сервера, где будут храниться его данные: ip-адрес, имя пользователя для подключения и папка для хранения ключа для подключения по ssh. Файл для linux1 представлен на «рисунке 37».

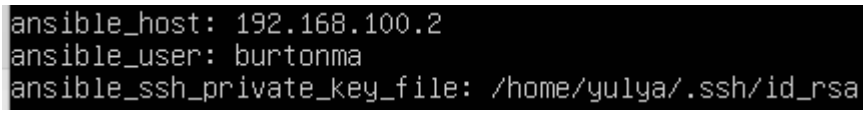


Рисунок 37 – Файл с конфигурацией для linux1

Аналогично прошлому файлу мы создаем подобный для linux2. Этот файл представлен на «рисунке 38».

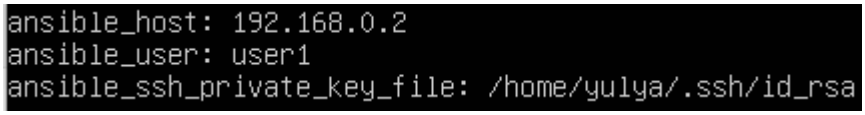


Рисунок 38 – Файл с конфигурацией для linux2

После чего создаем файл для linuxX.Он представлен на «рисунке 39».

Рисунок 39 – Файл с конфигурацией для linuxX

Теперь мы готовы проверить, насколько правильно настроен наш инвентарь. Для проверки доступности хостов и корректной работы Ansible была выполнена команда ansible all -m ping.Флаг –m используется для указания модуля, в данном случае это ping. Эта команда позволяет убедиться, что все необходимые системы доступны и готовы к выполнению задач. Результат на «рисунке 40» показал успешное подключение ко всем хостам, что подтверждает правильную настройку инвентаря и SSH-подключения

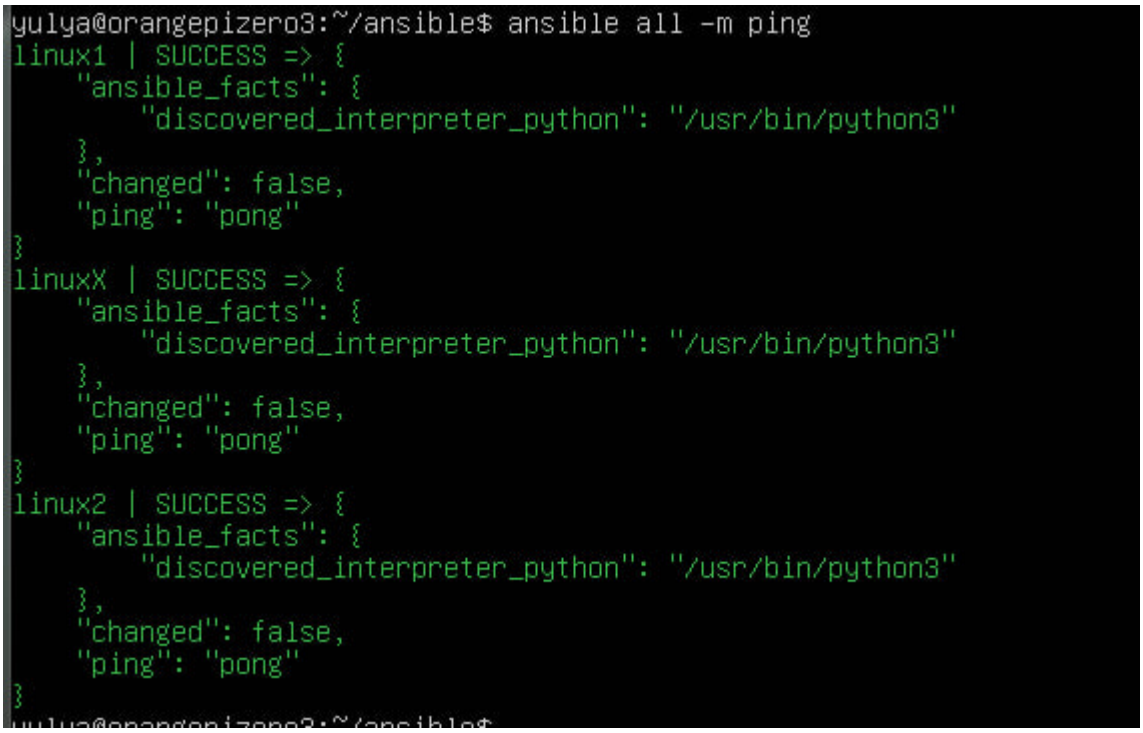


Рисунок 40 – Успешный отчет о тестировании связи с хостами

В Ansible существует два подхода к организации автоматизации, выбор между которыми зависит от масштаба ваших задач. Если требуется выполнить простое действие - например, установить пакеты, скопировать файлы или перенести данные - достаточно использовать один файл Ansible, в котором указать хосты, переменные и плейбуки. Такой файл легко переносить, просто оформлять, и он отлично подходит для несложных задач. Однако по мере роста сложности и объёма работ подобные файлы становятся громоздкими и трудными в поддержке, и тогда на помощь приходят роли.

Роли помогают структурировать плейбуки и упростить управление конфигурацией, разбивая её на более мелкие и понятные модули. Каждая роль - это набор задач, обработчиков, переменных и других файлов, объединённых по определённому признаку. Это способствует повторному использованию кода и повышает его читаемость. Важно, чтобы каждая роль выполняла конкретное, завершённое действие и была максимально универсальной. В сложных ролях для большей гибкости можно добавлять переменные с опциями, что позволяет объединять несколько смежных задач в одну роль и избегать дублирования кода. На «рисунке 41» для создания всей структуры каталогов роли используем команду ansible-galaxy role init roles/secure/ufw.

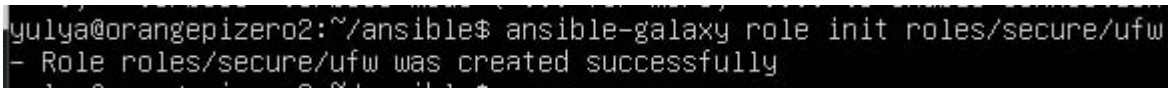


Рисунок 41 – Создание структуры каталогов для ufw

После чего перейдем в каталог роли командой cd и изучим структуру, выведя ее командой tree. Этот процесс показан на рисунке «рисунке 42».

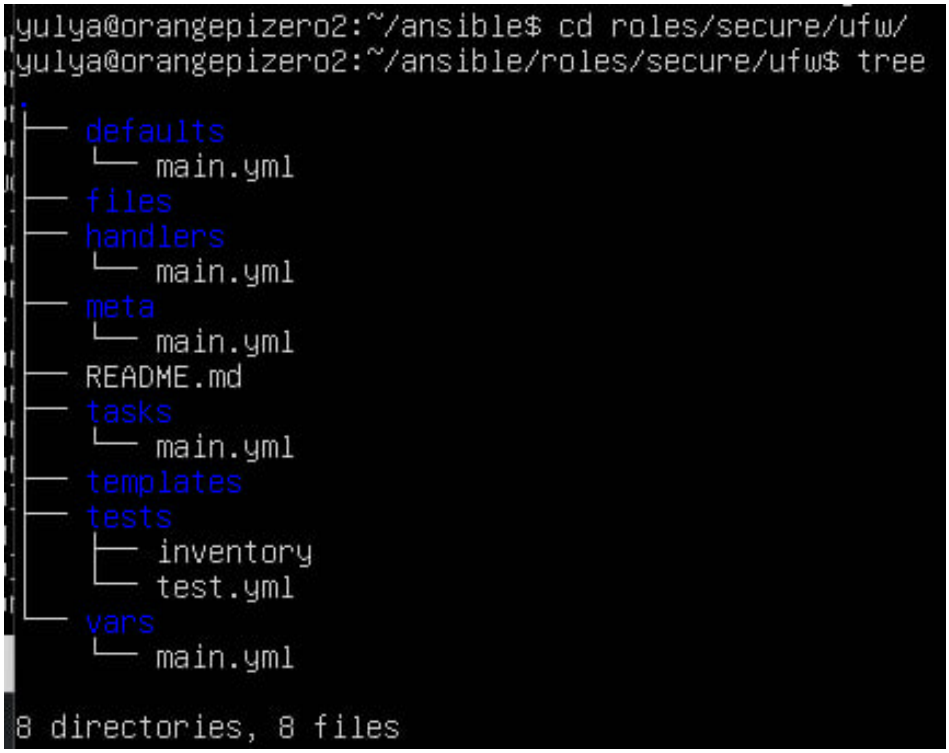


Рисунок 42 – Просмотр созданной структуры каталогов

Необходимо отредактировать файл с задачами tasks/main.yml, задав задачи. В первой заданной задаче мы проверяем, установлен ли ufw на нашей системе. Если его нет, Ansible автоматически установит нужный пакет. Это возможно

благодаря замечательному свойству Ansible - идемпотентности, которое гарантирует, что одно и то же действие не будет выполняться повторно. Поэтому при повторном запуске плейбука не возникнет ошибок из-за дублирования операций. Однако важно учитывать, что данное правило действует только для задач, выполняемых стандартными модулями Ansible. Если же используются команды, запускаемые через модуль терминала, идемпотентность не обеспечивается автоматически. Данный пример представлен на «рисунке 43».

```
# tasks file for roles/secure/ufw
- name: Ensure UFW is installed
  ansible.builtin.apt:
    name: ufw
    state: present
  become: true
```

Рисунок 43 – Задача с установкой ufw

Задача, представленная на «рисунке 44» настраивает политику по умолчанию для исходящих соединений как "разрешить". Это значит, что все исходящие соединения - то есть те, которые сервер инициирует к другим ресурсам - будут разрешены, если не задано отдельных правил брандмауэра, ограничивающих их.

```
- name: Set default outgoing policy to allow
  community.general.ufw:
    default: allow
    direction: outgoing
  become: true
```

Рисунок 44 – Задача, которая задает все исходящие соединения по умолчанию как «разрешить»

В данном примере на «рисунке 45» мы открываем 22 порт для работы по SSH, а также добавляем ufw в автозагрузку и сразу же запускаем его с уже применёнными правилами.

Рисунок 45 – Задачи, настраивающие запуск ufw

На «рисунке 46» показана задача, которая будет использовать цикл для разрешения указанных портов, из переменной ufw__rules, которую укажем по умолчанию и сможем переназначать при запуске плейбука. Она особенно полезна при масштабировании конфигурации, так как позволяет легко добавлять новые правила без изменения кода плейбука.

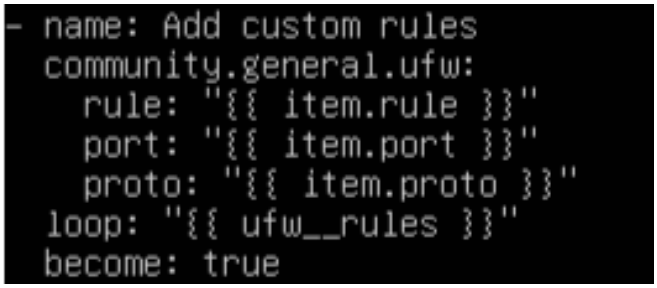


Рисунок 46 – Задача, использующая переменные, указанные по умолчанию

На «рисунке 47» представлен файл, в который можно задать переменные по умолчанию.В данном случае указаны два правила, разрешающие входящие подключения на порт 80 и 443 через протокол TCP.

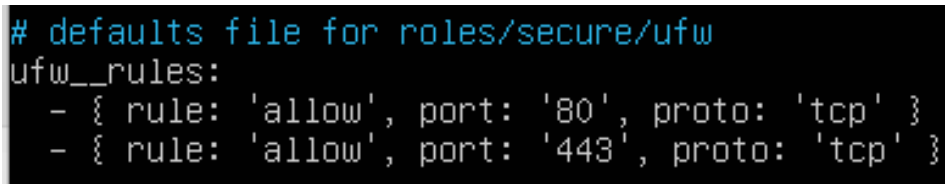


Рисунок 47 – Файл defaults/main.yml с переменными по умолчанию

После того, как роль готова – ее осталось протестировать. Для этого создадим плейбук командами mkdir -p playbooks/test и touch playbooks/test/test.yml. После чего заполним сам файл playbooks/test/test.yml. На «рисунке 48» мы открываем порт для службы DNS, чтобы продемонстрировать, как игнорировать переменные по умолчанию из роли UFW.

Рисунок 48 – Создание тестового плейбука

После написания плейбука, запустим его с помощью следующей команды: ansible-playbook playbooks/test/test.yml --ask-vault-pass. На «рисунке 49» видно, что плейбук запустился и есть связь с серверами.

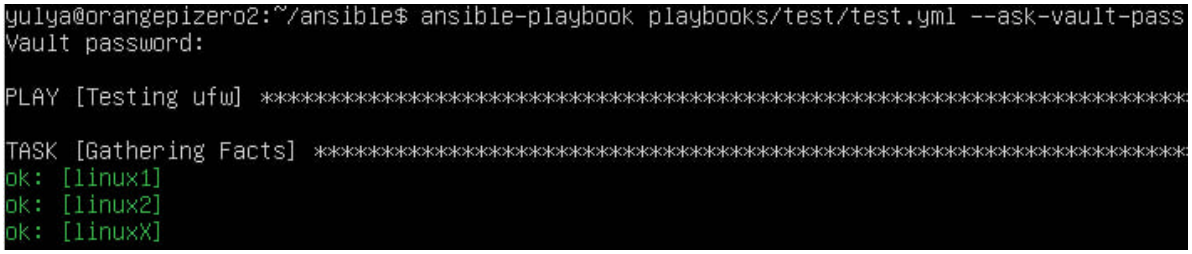


Рисунок 49 – Успешный запуск плейбука

Из вывода на «рисунке 50» команды можно сделать вывод, что все прошло успешно и все порты, кроме необходимых у нас закрыты.

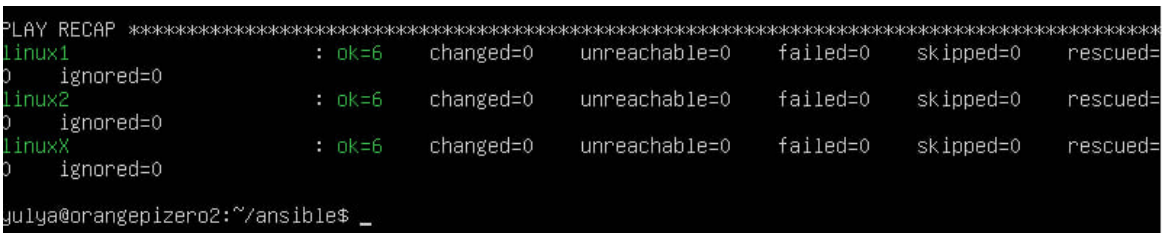


Рисунок 50 – Результат выполнения плейбука

Роль "fail2ban" предназначена для автоматической установки и настройки программы Fail2ban на серверах. Эта программа защищает сервер от атак, когда кто-то пытается подобрать пароль или получить доступ путём множества попыток (так называемые брутфорс-атаки). Fail2ban анализирует логи сервисов, находит подозрительные попытки входа и временно блокирует IP-адреса, с которых они поступают.

На «рисунке 51» для создания всей структуры каталогов роли используем команду ansible-galaxy role init, которая помогает организовать все необходимые файлы и настройки в одном месте. Всё, что связано с установкой

и настройкой Fail2ban (задачи, переменные, файлы, шаблоны), будет храниться внутри этой роли. Роль "fail2ban" - это готовый "блок" для автоматизации установки и настройки защиты от брутфорс-атак, который можно легко использовать в любом проекте Ansible.

```
yulya@orangepizero2:~/ansible/roles/secure/fail2ban$ ansible-galaxy role init roles/secure/fail2ban
- Role roles/secure/fail2ban was created successfully
```

Рисунок 51 – Установка структуры каталогов для fail2ban

Далее необходимо отредактировать конфигурационный файл templates/jail.local.j2. Параметры в нем я разделяю на две группы default и sshd. В первой группе я передаю такие параметры, как: время блокировки ip, промежуток времени для отслеживания неудачных попыток и количество, необходимое для блокировки пользователя. Во второй группе я настраиваю правила для защиты ssh-сервера: путь к лог.файлу, порт, фильтр для анализа логов и действие при обнаружении атаки. Этот файл показан на «рисунке 52»

```
[DEFAULT]
bantime     = {{ fail2ban__bantime }}
findtime    = {{ fail2ban__findtime }}
maxretry    = {{ fail2ban__maxretry }}
allowipv6   = true
[sshd]
enabled     = true
port        = ssh
filter      = sshd
action      = iptables-multiport[name==sshd, port=ssh, protocol=tcp]
logpath     = /var/log/sshd/sshd.log
```

Рисунок 52 – Отредактированный конфигурационный файл templates/jail.local.j2

Сами переменные я задаю по умолчанию в файле defaults/main.yml. Он представлен на «рисунке 53».

```
# defaults file for roles/secure/fail2ban
fail2ban__bantime: 600
fail2ban__findtime: 600
fail2ban__maxretry: 5
```

Рисунок 53 – Указание переменных в defaults/main.yml

После чего в плейбук добавляем роль fail2ban для ее корректной работы.

Также в нем же можно переопределить ключевые переменные в плейбуках для более гибкого управления ролями. Содержание плейбука представлено на «рисунке 54».

```
- name: Testing ufw + fail2ban
  hosts: test
  roles:
    - ufw
    - fail2ban
  vars:
# ufw
    ufw__rules:
      - { rule: 'allow', port: '53', proto: 'udp' }
#fail2ban
    fail2ban__bantime: 600
    fail2ban__findtime: 600
    fail2ban__maxretry: 3
```

Рисунок 54 – Добавление роли fail2ban в плейбук

Запускаем плейбук на «рисунке 55» со всеми ролями и видим, что все прошло успешно и запустилось корректно.

```
yulya@orangepizero2:~/ansible$ ansible-playbook playbooks/test/test.yml

PLAY [Testing ufw + fail2ban] **************************************************************************

TASK [Gathering Facts] *********************************************************************************
ok: [linux1]
ok: [linux2]
ok: [linuxX]

TASK [ufw : Ensure UFW is installed] *******************************************************************
ok: [linux1]
ok: [linuxX]
ok: [linux2]

TASK [ufw : Set default outgoing policy to allow] ******************************************************
ok: [linux1]
ok: [linuxX]
ok: [linux2]

TASK [ufw : Allow SSH connections] *********************************************************************
ok: [linux1]
ok: [linuxX]
ok: [linux2]

TASK [ufw : Ensure UFW is enabled and set to start on boot] ********************************************
ok: [linux1]
ok: [linux2]
ok: [linuxX]

TASK [ufw : Add custom rules] **************************************************************************
ok: [linux1] => (item={'rule': 'allow', 'port': '53', 'proto': 'udp'})
ok: [linuxX] => (item={'rule': 'allow', 'port': '53', 'proto': 'udp'})
ok: [linux2] => (item={'rule': 'allow', 'port': '53', 'proto': 'udp'})

TASK [fail2ban : Update_apt_cache] *********************************************************************
```

Рисунок 55 – Запуск плейбука с новыми ролями

После того, как основные задачи запустились верно, я решила продумать расширение функций бота для того, чтобы максимально автоматизировать

работу серверов и облегчить работу пользователя, который пользуется ботом. Для категории мониторинга я добавляю плейбук, которые будет выводить информацию о процессоре и памяти.Он позволяет быстро получить важные метрики без необходимости логиниться на каждый сервер вручную. Этот плейбук показан на «рисунке 56».

```
- name: Сбор информации о CPU и RAM (для всех хостов)
  hosts: test
  become: yes
  gather_facts: yes

  tasks:
    - name: Отображение информации о процессоре
      ansible.builtin.debug:
        msg: |
          Модель CPU: {{ ansible_processor }}
          Количество ядер: {{ ansible_processor_cores }}
          vCPU: {{ ansible_processor_vcpus }}

    - name: Отображение информации о памяти
      ansible.builtin.debug:
        msg: |
          Общая память: {{ ansible_memory_mb.real.total / 1024 / 1024 }} GB
          Используется: {{ ansible_memory_mb.real.used / 1024 / 1024 }} GB
          Свободно: {{ ansible_memory_mb.real.free / 1024 / 1024 }} GB

    - name: Показ текущей нагрузки системы
      ansible.builtin.shell: uptime
      register: load_avg

    - name: Вывод нагрузки
      ansible.builtin.debug:
        msg: "Текущая нагрузка: {{ load_avg.stdout }}"
```

Рисунок 56 – Плейбук для вывода CPU и RAM

К важным задачам по проверке серверов я также отношу отслеживание свободного места. Под нее создаем плейбук, который позволит командой df -h следить за контролем заполнения файловых систем. Этот файл показан на «рисунке 57».

```
- name: Проверка свободного места на дисках (для всех хостов)
  hosts: test
  become: yes

  tasks:
    - name: Получить информацию о дисковом пространстве
      ansible.builtin.shell: df -h
      register: disk_output

    - name: Вывести результат
      ansible.builtin.debug:
        msg: "{{ disk_output.stdout }}"
```

Рисунок 57 – Плейбук для проверки свободного места

При поиске нужных зависимостей особенно важно отслеживать список установленных пакетов на каждом хосте. Созданный под эту задачу плейбук выполняет команду dpkg --get-selections для получения списка установленных пакетов и выводит первые 15 в удобночитаемом формате. Он показан на «рисунке 58».

```
- name: Получение списка установленных пакетов (для всех хостов)
  hosts: test
  become: yes

  tasks:
    - name: Получить первые 15 установленных пакетов
      ansible.builtin.shell: dpkg --get-selections | head -n 15
      register: packages_list

    - name: Вывести список пакетов
      ansible.builtin.debug:
        msg: |
          Установленные пакеты (первые 15):
          {{ packages_list.stdout }}
```

Рисунок 58 – Плейбук для получения списка установленных пакетов

Для отслеживания активных ssh-подключений я использую созданный плейбук, который при помощи модуля shell и фильтрации через grep ssh получает список активных ssh-сессий. На «рисунке 59» показано содержимое этого плейбука.

```
- name: Проверка активных SSH-подключений (для всех хостов)
  hosts: test
  become: yes

  tasks:
    - name: Поиск активных SSH сессий
      ansible.builtin.shell: who | grep ssh
      register: ssh_connections
      ignore_errors: yes

    - name: Вывести результат
      ansible.builtin.debug:
        msg: |
          {% if ssh_connections.stdout != '' %}
          Активные SSH-сессии:\n{{ ssh_connections.stdout }}
          {% else %}
          Нет активных SSH-соединений.
          {% endif %}
```

Рисунок 59 – Плейбук для проверки ssh-подключений

Также важный пункт при удаленной работе - это отслеживать время непрерывной работы сервера, для того чтобы избежать сбоев и перегревов в системе. С помощью модуля shell и команды uptime –p можно получить и вывести время работы системы. Этот плейбук показан на «рисунке 60».

```
- name: Проверка времени работы сервера (для всех хостов)
  hosts: test
  become: yes

  tasks:
    - name: Получить аптайм сервера
      ansible.builtin.shell: uptime -p
      register: uptime_result

    - name: Вывести аптайм
      ansible.builtin.debug:
        msg: "Сервер работает: {{ uptime_result.stdout }}"
```

Рисунок 60 – Плейбук для проверки времени работы сервера

Для того, чтобы избежать сбоев в работе fail2ban добавим возможность перезапустить его. Этот плейбук показан на «рисунке 61».

```
- name: Перезапуск службы fail2ban
  hosts: test
  become: yes

  tasks:
    - name: Перезапустить fail2ban
      ansible.builtin.service:
        name: fail2ban
        state: restarted

    - name: Проверить статус после перезапуска
      ansible.builtin.service:
        name: fail2ban
        state: started

    - name: Сообщение о успешном перезапуске
      ansible.builtin.debug:
        msg: "[ok] Служба fail2ban успешно перезапущена"
```

Рисунок 61 – Плейбук для перезапуска fail2ban

На «рисунке 62» для подготовки окружения используются несколько задач, которые обновляют список пакетов, скачивают необходимые утилиты и подготавливают необходимое окружение для дальнейшей работы и настройки системы безопасности.

```
- name: Настройка базовой безопасности
  hosts: test
  become: yes

  vars:
    ssh_log_dir: /var/log/sshd/
    ssh_log_file: "{{ ssh_log_dir }}sshd.log"

  tasks:
    - name: Обновить список пакетов
      ansible.builtin.apt:
        update_cache: yes

    - name: Установка rsyslog
      ansible.builtin.apt:
        name: rsyslog
        state: present

    - name: Установка iptables
      ansible.builtin.apt:
        name: iptables
        state: present

    - name: Создать директорию для SSH-логов
      ansible.builtin.file:
        path: "{{ ssh_log_dir }}"
        state: directory
        mode: '0755'
```

Рисунок 62 – Задачи для подготовки окружения к настройке базовой безопасности

На «рисунке 63» этот фрагмент плейбука предназначен для настройки логирования SSH , чтобы перенаправлять логи SSH в отдельный файл и обеспечивать корректное логирование событий аутентификации через facility AUTH. А также гарантировать, что изменения вступят в силу , перезапуская

службу rsyslog.

```
- name: Настроить rsyslog для логирования SSH
  ansible.builtin.copy:
    dest: /etc/rsyslog.d/50-ssh.conf
    content: |
      if $programname == 'sshd' then {{ ssh_log_file }}
      & stop
    owner: root
    group: root
    mode: '0644'

- name: Перезапустить rsyslog
  ansible.builtin.systemd:
    name: rsyslog
    state: restarted

- name: Включить логирование AUTH
  ansible.builtin.lineinfile:
    path: /etc/ssh/sshd_config
    regexp: '^#SyslogFacility AUTH'
    line: 'SyslogFacility AUTH'
    state: present
```

Рисунок 63 – Задачи для настройки логирования ssh

На «рисунке 64» этот фрагмент плейбука предназначен для настройки уровня логирования SSH на INFO, чтобы собирать подробные данные о подключениях и перезапуска службы SSH , чтобы изменения вступили в силу. А также установки программы fail2ban , которая защищает сервер от атак.

```
- name: Установить уровень логирования INFO
  ansible.builtin.lineinfile:
    path: /etc/ssh/sshd_config
    regexp: '^#LogLevel INFO'
    line: 'LogLevel INFO'
    state: present

- name: Перезапустить SSH
  ansible.builtin.systemd:
    name: ssh
    state: restarted

- name: Установка Fail2Ban
  ansible.builtin.apt:
    name: fail2ban
    state: present
```

Рисунок 64 – Задачи для настройки логирования

На «рисунке 65» этот фрагмент плейбука предназначен для генерации файла конфигурации jail.local с помощью шаблона Jinja2 и запуска службы fail2ban , чтобы она работала постоянно. Также необходимо прописать настройку перезапуска fail2ban , если были изменения в конфигурации.

```
  - name: Копировать конфигурацию jail.local
    ansible.builtin.template:
      src: jail.local.j2
      dest: /etc/fail2ban/jail.local
      owner: root
      group: root
      mode: '0644'
    notify:
      - restart fail2ban

  - name: Запустить и включить Fail2Ban
    ansible.builtin.service:
      name: fail2ban
      state: started
      enabled: yes

handlers:
  - name: restart fail2ban
    ansible.builtin.service:
      name: fail2ban
      state: restarted
```

Рисунок 65 – Задачи для изменения конфигурации через файл jail.local

«Рисунок 66» представляет собой плейбук для вывода последних SSH – логов. В нем настраиваем выполнение команды tail -n 20 /var/log/sshd.log для получения последних 20 строк логов SSH, а затем сохраняем результат в переменную ssh_logs. Далее выводится результат в удобочитаемом формате, проверяя, есть ли данные в логах.

```
- name: Просмотр последних SSH-логов
  hosts: test
  become: yes

  tasks:
    - name: Показать последние 20 строк SSH-логов
      ansible.builtin.shell: tail -n 20 /var/log/sshd/sshd.log
      register: ssh_logs
      ignore_errors: yes

    - name: Отобразить результат
      ansible.builtin.debug:
        msg: |
          {% if ssh_logs.stdout != '' %}
          Последние SSH-подключения:\n{{ ssh_logs.stdout }}
          {% else %}
          Логи SSH не найдены или пусты.
          {% endif %}
```

Рисунок 66 – Плейбук для вывода логов ssh

На «рисунке 67» создаем плейбук для того, чтобы убедиться, что служба fail2ban запущена и работает.Плейбук выполняет команду systemctl status fail2ban | grep Active через модуль shell и сохраняет результат в переменную fail2ban_status. После чего выводит результат в удобочитаемом формате.

```
- name: Проверка статуса Fail2Ban
  hosts: test
  become: yes

  tasks:
    - name: Получить статус службы fail2ban
      ansible.builtin.shell: systemctl status fail2ban | grep Active
      register: fail2ban_status

    - name: Отобразить статус fail2ban
      ansible.builtin.debug:
        msg: |
          Служба fail2ban:
          {{ fail2ban_status.stdout }}
```

Рисунок 67 – Плейбук для проверки статуса fail2ban

На «рисунке 68» создан плейбук для того, чтобы оптимизировать процессы, который будет обновлять все пакеты до последней последней версии, снимая тем самым необходимость делать все вручную. В этом плейбуке мы настраиваем задачу командой upgrade: dist которое выполняет расширенное обновление и которое может переустанавливать пакеты для обеспечения максимальной совместимости.

```
- name: Обновление системы
  hosts: test
  become: yes

  tasks:
    - name: Обновить кэш пакетов
      ansible.builtin.apt:
        update_cache: yes

    - name: Обновить все установленные пакеты
      ansible.builtin.apt:
        upgrade: dist

    - name: Сообщение о завершении обновления
      ansible.builtin.debug:
        msg: "[ok] Все пакеты успешно обновлены"
```

Рисунок 68 – Плейбук для обновления пакетов

3.3 Реализация мобильного бота

После того, как все плейбуки созданы и настроены, переходим к настройке

бота. На «рисунке 69» мы прописываем функцию, которая будет обрабатывать команду /start от пользователя и создавать кнопки для выбора групп устройств.После чего оправлять приветственное сообщение с клавиатурой.

```
@bot.message_handler(commands=['start'])
def start(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    for group in GROUPS.keys():
        markup.add(types.KeyboardButton(group))
    bot.send_message(message.chat.id,
                     text=f"Hello, {message.from_user.first_name}! Select a device group.",
                     reply_markup=markup)
```

Рисунок 69 – Функция для обработки команды /start

На «рисунке 70» для того, чтобы у пользователя была возможность выбирать конкретную группу устройств мы распределяем их в зависимости от функционального назначения. Так пользователь сможет ускорить процесс работы, выполняя определенный плейбук для определенной группы.

```
GROUPS = {
    "Database": "database",
    "Webserver": "webservers",
    "Appserver": "appservers",
    "Production servers": "prod_servers",
    "Test(all)": "test"
}
```

Рисунок 70 – Настройка групп серверов для бота

Для удобства выбора разделим группы действий на 2 категории: Мониторинг и безопасность. Этот процесс показан на «рисунке 71».

```
CATEGORIES = {
    "Monitoring": "monitoring",
    "Security": "security"
}
```

Рисунок 71 – Создание категорий для бота

Далее на «рисунке 72» создадим функцию, которая после обработки группы устройств, будет предлагать пользователю выбрать категорию действий. То есть создаем две кнопки и создаем текст, который будет отправляться ботом

после выбора группы.

```
@bot.message_handler(func=lambda message: message.text in GROUPS.keys())
def select_group(message):
    selected_group = GROUPS[message.text]
    user_id = message.from_user.id
    user_groups[user_id] = selected_group

    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    for category in CATEGORIES.keys():
        markup.add(types.KeyboardButton(category))
    bot.send_message(
        message.chat.id,
        f"You selected the group: {selected_group}. Choose an action category:",
        reply_markup=markup
    )
```

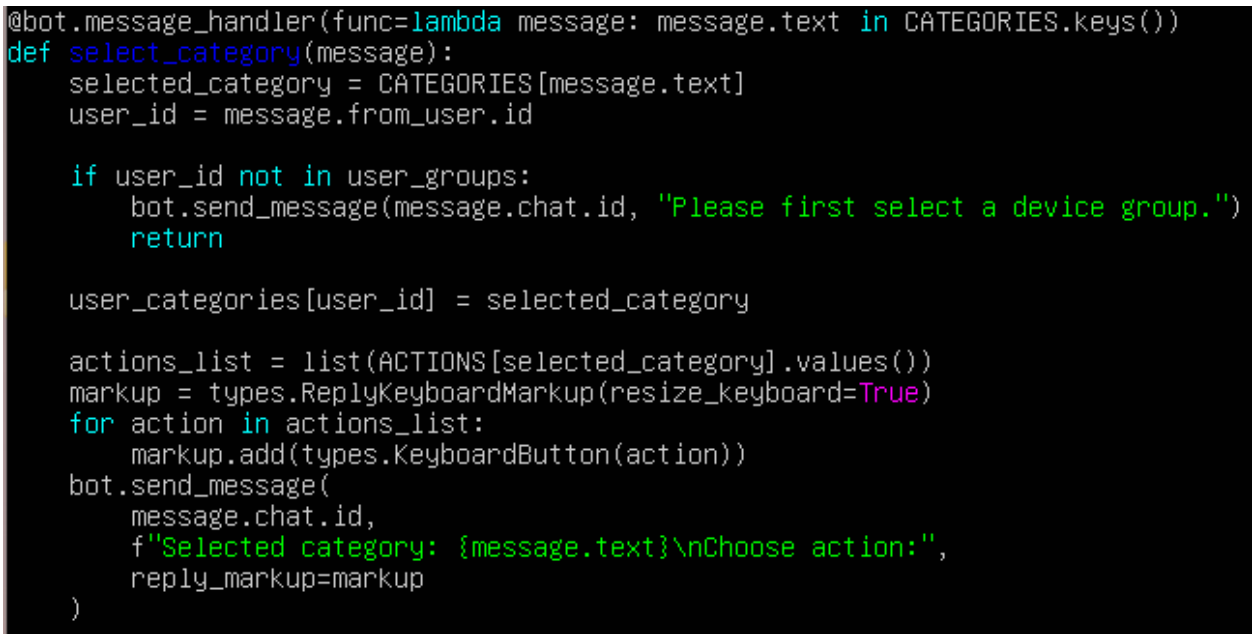
Рисунок 72 – Функция с кнопками выбора категорий

Далее создадим действия к каждому из плейбуков, распределяя их по категориям. Эта часть показана на «рисунке 73».

```
ACTIONS = {
    "security": {
        1: "setup_security",
        2: "status_fail2ban",
        3: "show_ssh_logs",
        4: "restart_service",
        5: "update_packages"
    },
    "monitoring": {
        1: "check_cpu_ram",
        2: "disk_usage",
        3: "active_ssh_connections",
        4: "server_uptime",
        5: "installed_packages"
    }
}
```

Рисунок 73 – Распределение действий по категориям

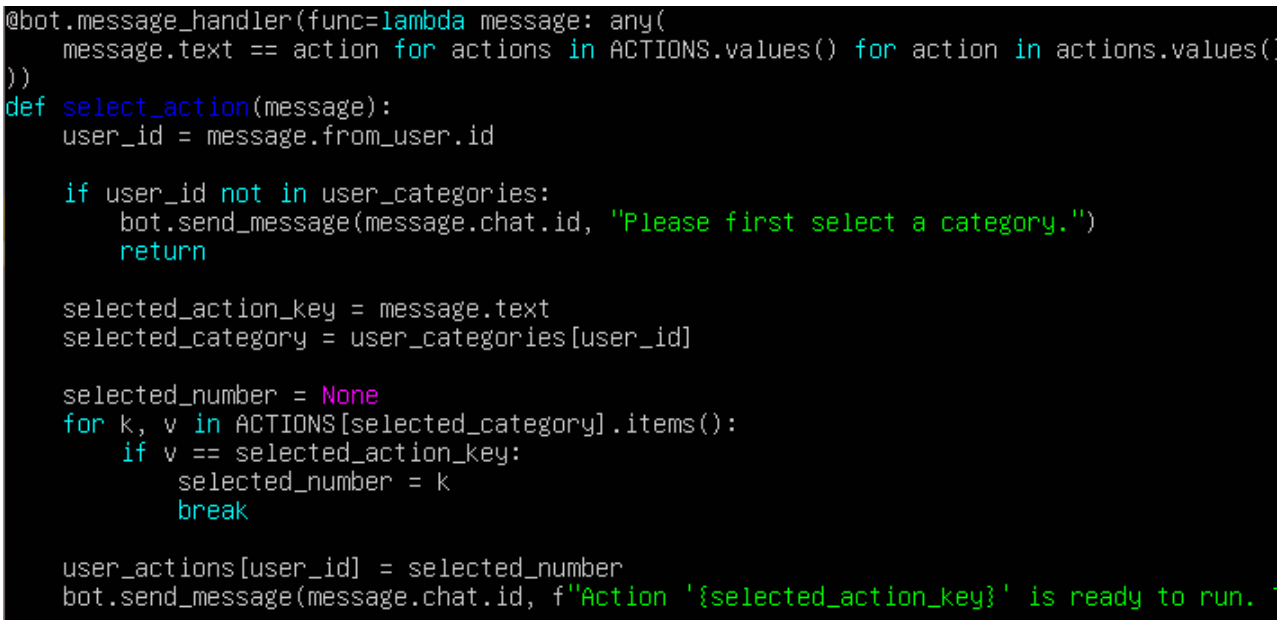
Также необходимо создать функцию, которая будет обрабатывать категорию действий, которую выбрал пользователь и на основе этого у пользователя должны появляться кнопки с конкретными действиями(рисунок 74).



```
@bot.message_handler(func=lambda message: message.text in CATEGORIES.keys())
def select_category(message):
    selected_category = CATEGORIES[message.text]
    user_id = message.from_user.id

    if user_id not in user_groups:
        bot.send_message(message.chat.id, "Please first select a device group.")
        return

    user_categories[user_id] = selected_category

    actions_list = list(ACTIONS[selected_category].values())
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    for action in actions_list:
        markup.add(types.KeyboardButton(action))
    bot.send_message(
        message.chat.id,
        f"Selected category: {message.text}\nChoose action:",
        reply_markup=markup
    )
```

Рисунок 74 – Функция для отображения кнопок с выбором действия

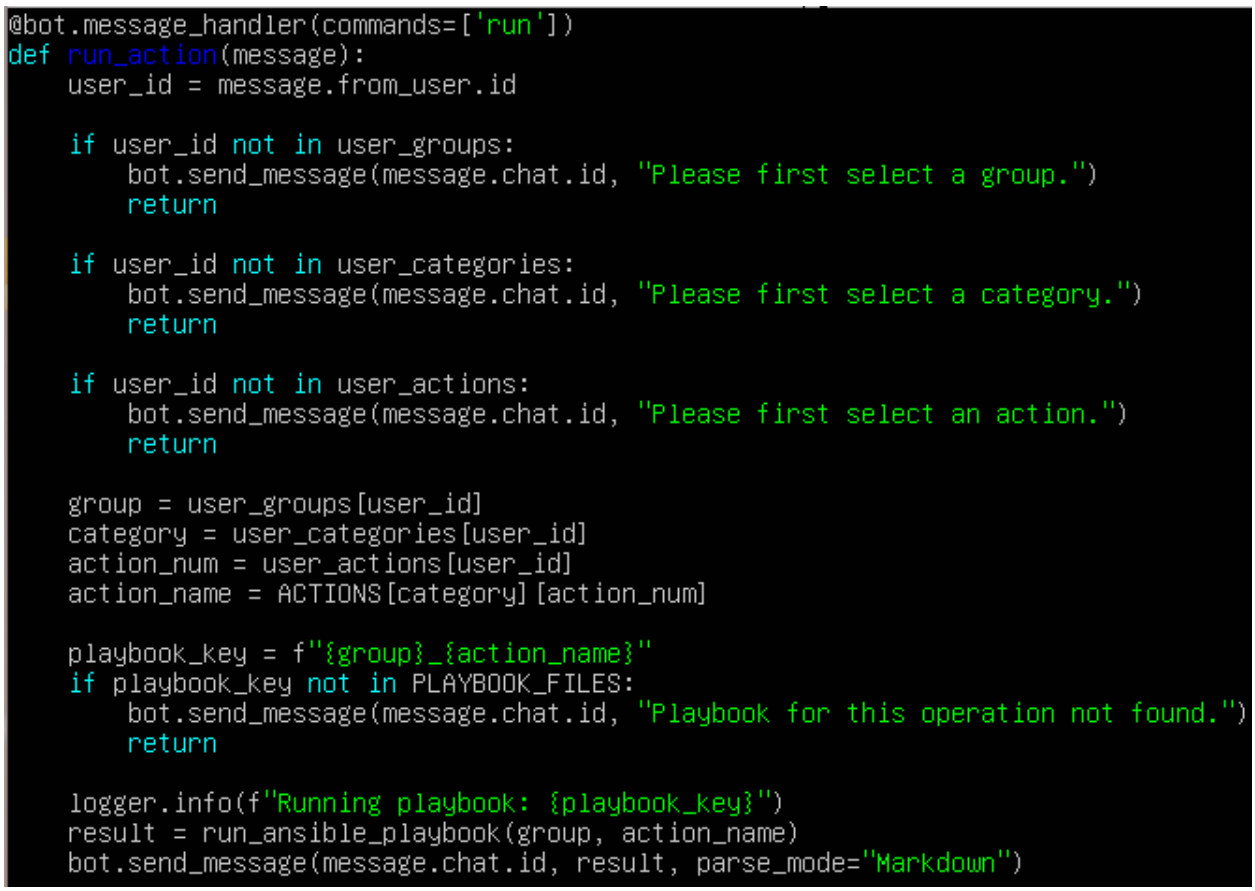
После того, как пользователь нажал кнопку с определенным действием, бот обрабатывает запрос и ждет команды пользователя /run для того чтобы запустить плейбук. Эта функция показана на «рисунке 75».



```
@bot.message_handler(func=lambda message: any(
    message.text == action for actions in ACTIONS.values() for action in actions.values(
))
def select_action(message):
    user_id = message.from_user.id

    if user_id not in user_categories:
        bot.send_message(message.chat.id, "Please first select a category.")
        return

    selected_action_key = message.text
    selected_category = user_categories[user_id]

    selected_number = None
    for k, v in ACTIONS[selected_category].items():
        if v == selected_action_key:
            selected_number = k
            break

    user_actions[user_id] = selected_number
    bot.send_message(message.chat.id, f"Action '{selected_action_key}' is ready to run. 
```

Рисунок 75 – Функция для обработки запроса пользователя

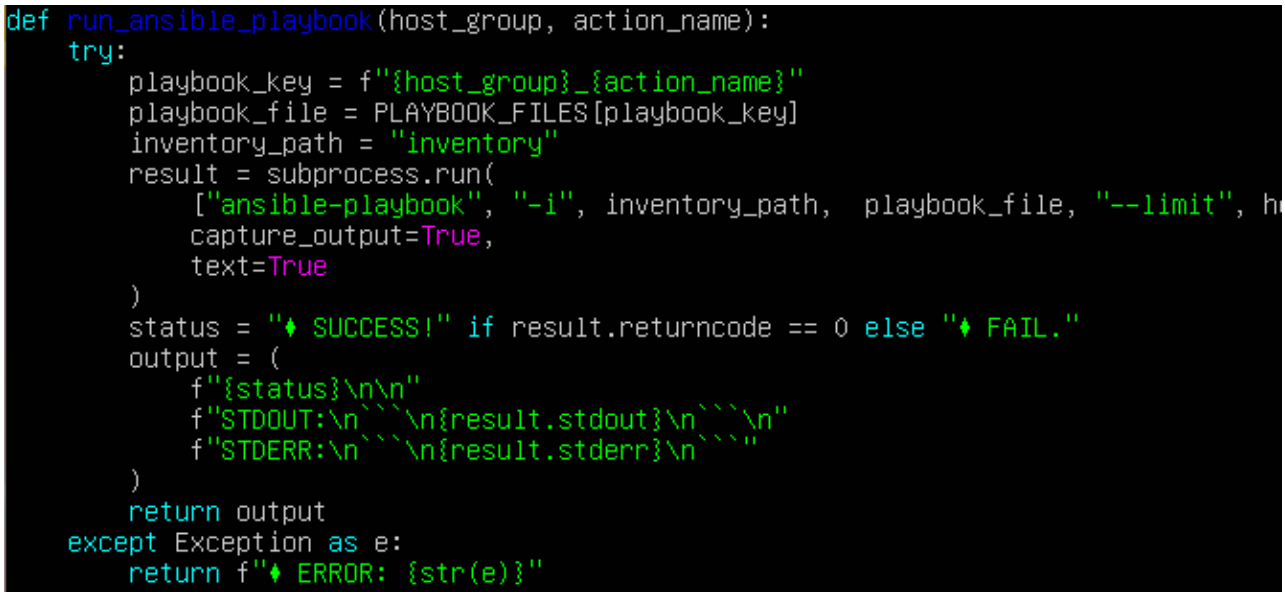
Создаем функцию, которая будет обрабатывать команду /run(рисунок 76). Прежде всего, нужно добавить условия, которые будут следить за тем, чтобы все необходимые группы были выбраны. После чего будет обрабатываться вывод в языке разметки markdown для наиболее читаемого вывода.



```python
@bot.message_handler(commands=['run'])
def run_action(message):
    user_id = message.from_user.id

    if user_id not in user_groups:
        bot.send_message(message.chat.id, "Please first select a group.")
        return

    if user_id not in user_categories:
        bot.send_message(message.chat.id, "Please first select a category.")
        return

    if user_id not in user_actions:
        bot.send_message(message.chat.id, "Please first select an action.")
        return

    group = user_groups[user_id]
    category = user_categories[user_id]
    action_num = user_actions[user_id]
    action_name = ACTIONS[category][action_num]

    playbook_key = f"{group}_{action_name}"
    if playbook_key not in PLAYBOOK_FILES:
        bot.send_message(message.chat.id, "Playbook for this operation not found.")
        return

    logger.info(f"Running playbook: {playbook_key}")
    result = run_ansible_playbook(group, action_name)
    bot.send_message(message.chat.id, result, parse_mode="Markdown")
```

Рисунок 76 – Функция для обработки команды /run

На «рисунке 77» создаем функцию, которая будет запускать плейбук со всеми введенными ранее значениями.



```python
def run_ansible_playbook(host_group, action_name):
    try:
        playbook_key = f"{host_group}_{action_name}"
        playbook_file = PLAYBOOK_FILES[playbook_key]
        inventory_path = "inventory"
        result = subprocess.run(
            ["ansible-playbook", "-i", inventory_path,  playbook_file, "--limit", h
            capture_output=True,
            text=True
        )
        status = "♦ SUCCESS!" if result.returncode == 0 else "♦ FAIL."
        output = (
            f"{status}\n\n"
            f"STDOUT:\n```\n{result.stdout}\n```\n"
            f"STDERR:\n```\n{result.stderr}\n```"
        )
        return output
    except Exception as e:
        return f"♦ ERROR: {str(e)}"
```

Рисунок 77 – Функция для запуска плейбука

На «рисунке 78» в самом файле бота мы прописали путь до запускаемого плейбука, чтобы при запуске бот обращался конкретно к нему.

```
# Monitoring / test
"test_check_cpu_ram": "playbooks/monitoring/test_cpu_ram.yml",
"test_disk_usage": "playbooks/monitoring/test_disk.yml",
"test_active_ssh_connections": "playbooks/monitoring/test_ssh_conns.yml"
"test_server_uptime": "playbooks/monitoring/test_uptime.yml",
"test_installed_packages": "playbooks/monitoring/test_packages.yml",

# Security / test
"test_setup_security": "playbooks/security/test_setup_security.yml",
"test_status_fail2ban": "playbooks/security/test_status_fail2ban.yml",
"test_show_ssh_logs": "playbooks/security/test_show_ssh_logs.yml",
"test_restart_service": "playbooks/security/test_restart_service.yml",
"test_update_packages": "playbooks/security/test_update_packages.yml",
```

Рисунок 78 – Указание пути до запускаемых плейбуков

На «рисунке 79» представлены библиотеки которые нужны для корректной работы бота. Например, модуль logging поможет отслеживать работу бота и упростить отладку. Модуль telebot обеспечивает связь с Telegram API, a types используется для создания пользовательского интерфейса. Модуль subprocess может быть полезен для запуска плейбуков Ansible или других скриптов.

```
import os
import logging
import subprocess
import telebot
from telebot import types
```

Рисунок 79 – Используемые библиотеки в работе бота

Для того, чтобы работу бота сделать стабильной и изолировать от основной системы. На «рисунке 80» представлен настроенный файл docker-compose.yml. В нем мы создаем контейнер ansible-telegram-bot, который будет обращаться к нашему боту через токен. Настраиваем работу бота даже после перезагрузки или сбоя системы. А также монтируем локальные директории внутрь контейнера.

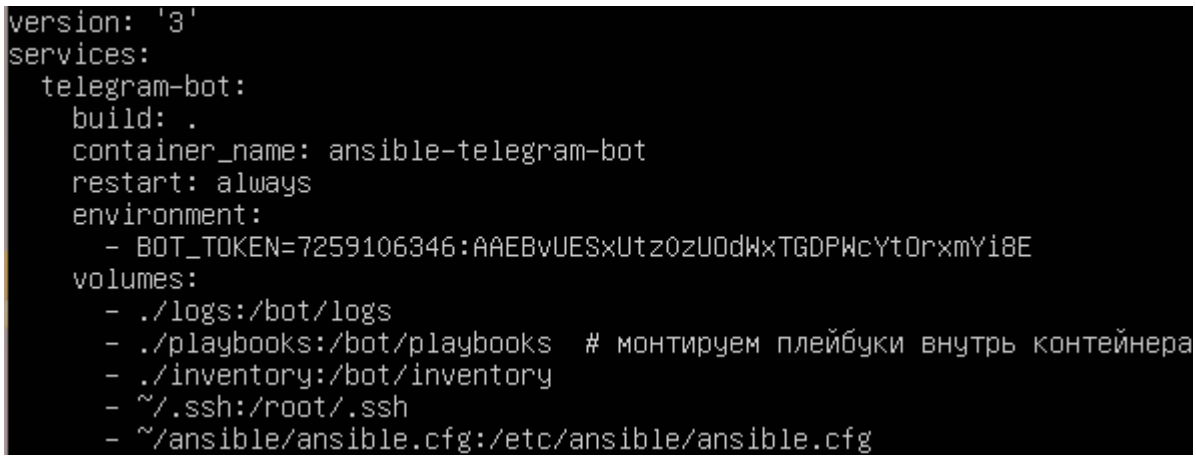


Рисунок 80 – Файл docker-compose.yml

На «рисунке 81» в самом Dockerfile мы настраиваем необходимые зависимости, задаем рабочую директорию /bot, и указываем команду запуска бота, которая будет запускаться после сборки образа

```
FROM python:3.10-slim


RUN apt-get update && apt-get install -y openssh-client


WORKDIR /bot
COPY bot/requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY bot /bot

CMD ["python3", "bot.py"]
```

Рисунок 81 – Файл Dockerfile

На «рисунке 82» показан файл зависимостей requirements.txt. В данном случае pyTelegramBotAPI нужен для работы с Telegram API, ansible — для взаимодействия с плейбуками Ansible.

```
pyTelegramBotAPI==4.16.1
ansible
```

Рисунок 82 – Файл зависимостей requirements.txt

На «рисунке 83» показано добавление cron задачи для регулярного запуска

бота telegram, используя скрипт start_bot.sh из домашней директории нашего пользователя.

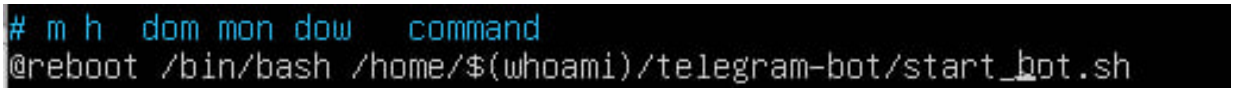


Рисунок 83 – настройка автоматического запуска телеграм-бота через cron

На «рисунке 84» показан процесс сборки Docker-образа для Telegram-бота с помощью Docker Compose командой docker-compose up --build.

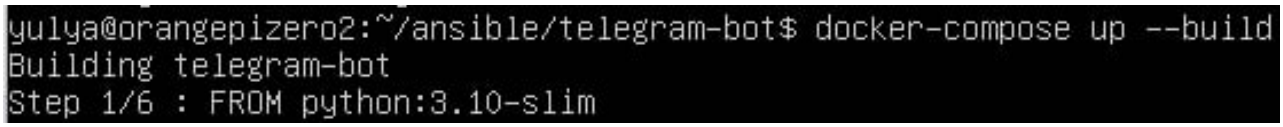


Рисунок 84 – Сборка образа

3.4 Тестирование системы управления на базе Ansible

На рисунке 85 показан начальный этап создания Telegram-бота через BotFather. Чтобы создать нового бота пишем команду /newbot. После чего выбираем имя бота Ansible for orange pi, соответствующего правилам Telegram. В результате получаем токен API для дальнейшей работы с ботом через Telegram Bot API.

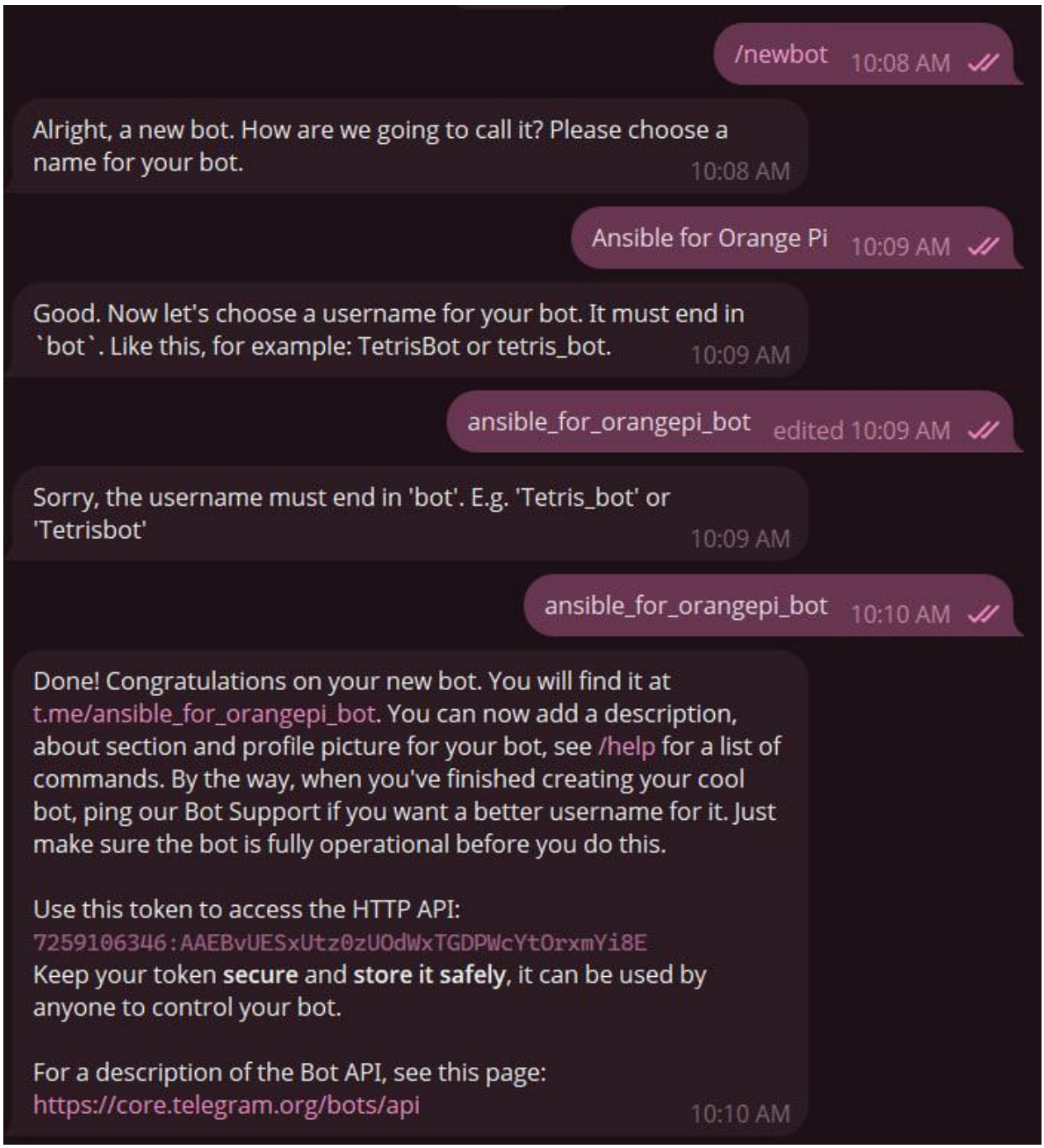


Рисунок 85 – Создание бота

На «рисунке 86» представляется сама работа бота, после отправленного сообщения с командой /start ему отправляется приветственное сообщение, в котором говорится о том, что нужно выбрать группу устройств.

Рисунок 86 – Работа бота при отправке /start

На «рисунке 87» показан процесс работы бота. Пользователь отправляет группу, а далее бот предлагает выбрать категорию действий для выполнения.

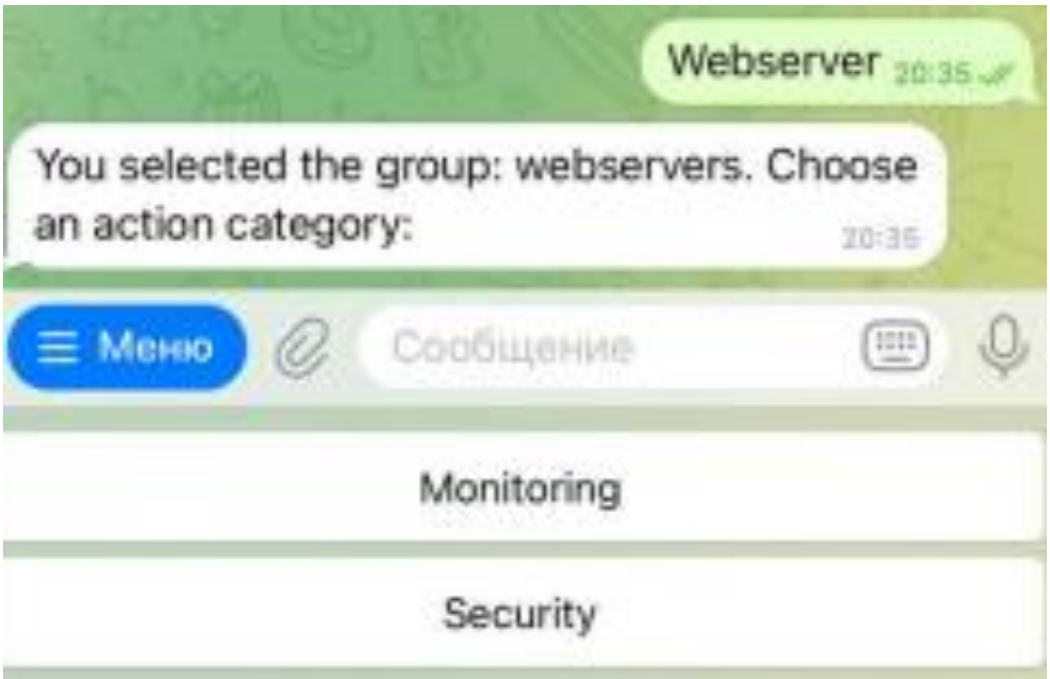


Рисунок 87 – Процесс работы бота после выбора устройств

На «рисунке 88» показана работа бота после выбора категории действий. У пользователя появляется 5 кнопок с названиями их действий.

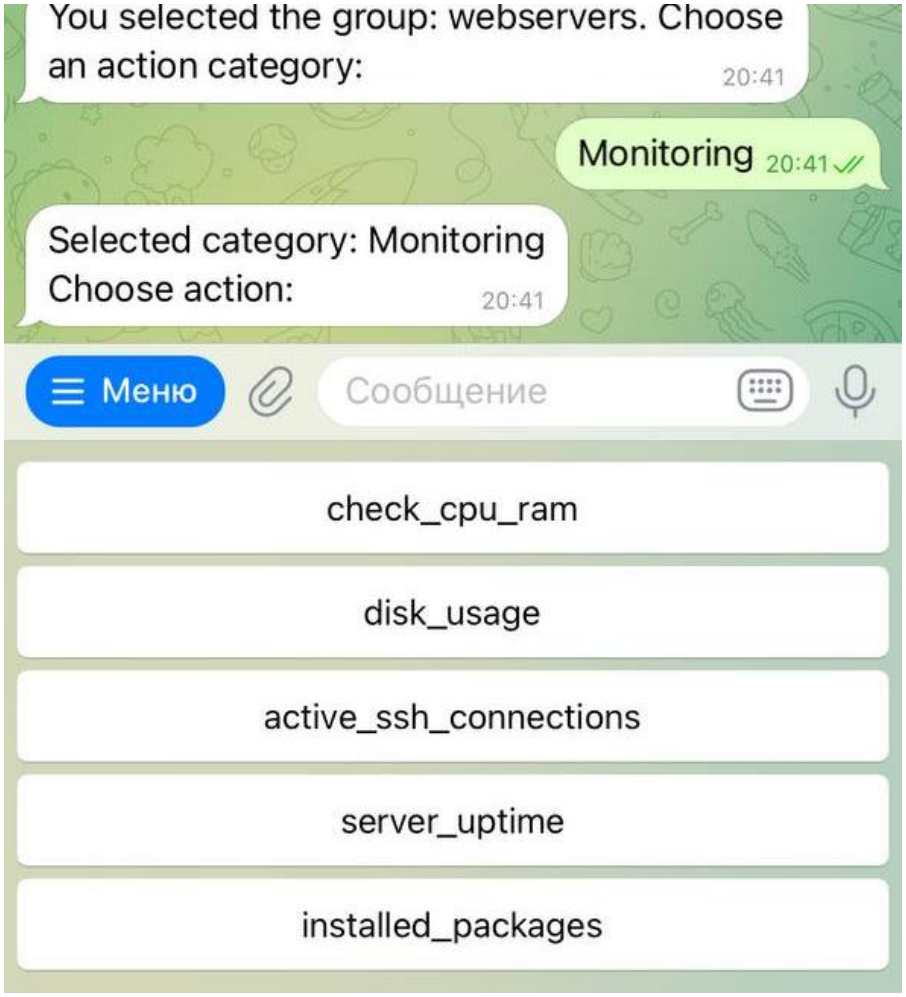


Рисунок 88 – Работа бота с кнопками выбора плейбука

На «рисунке 89» после того как действие выбрано, бот предлагает его запустить, введя команду /run.

Рисунок 89 – Работа бота после нажатия кнопки

На «рисунке 90» показан вывод бота, отображающий информацию о cpu и

памяти

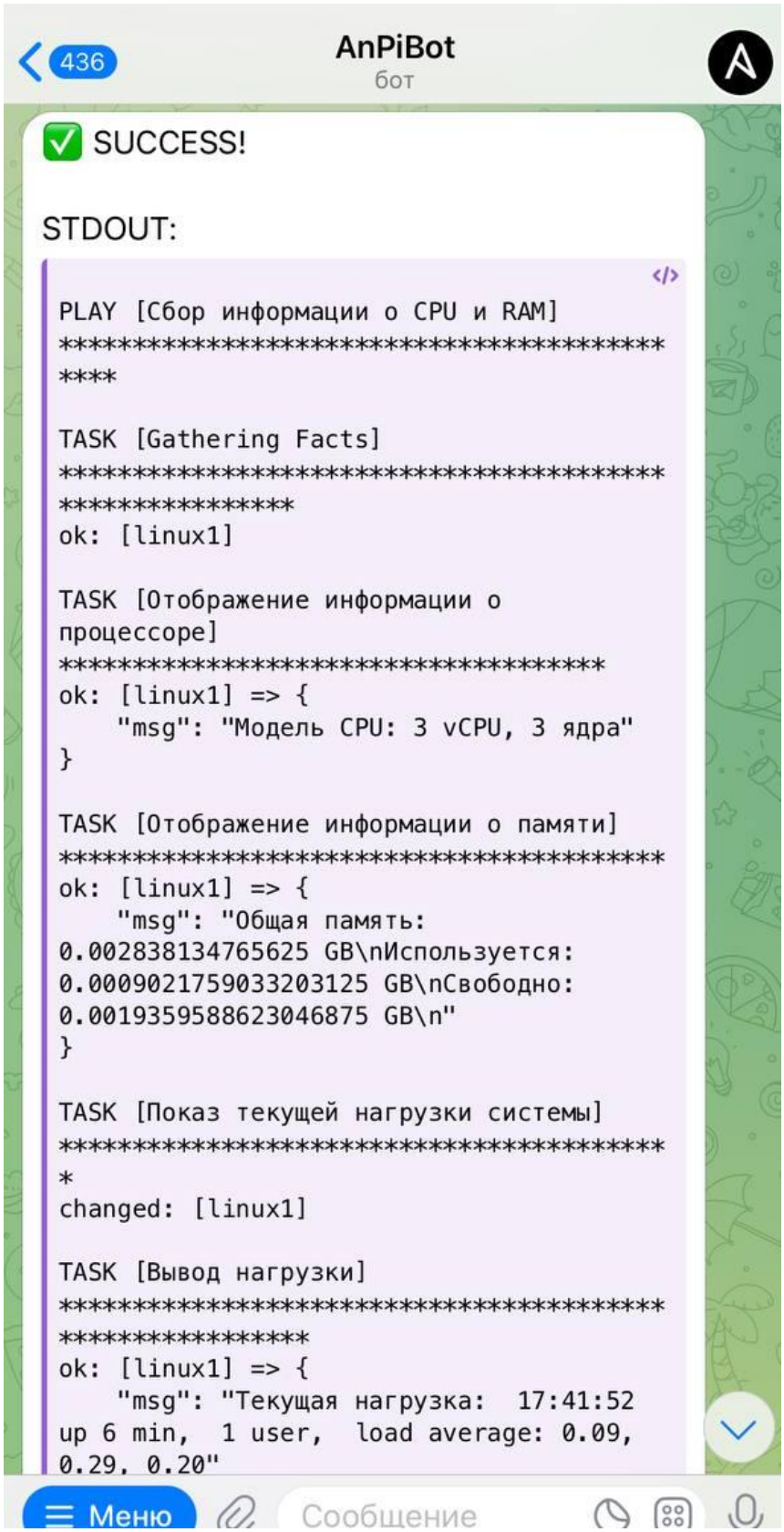


Рисунок 90 – Запуск плейбука и вывод