Практическая работа №5

По дисциплине «Технологии разработки программных приложений»

Тема «Системы конфигурационного управления»

Ansible

Цель работы: получить навыки настройки вычислительной инфраструктуры при помощи системы конфигурационного управления Ansible

Подготовка инфраструктуры

Прежде всего, необходимо создать 2 виртуальные машины Ubuntu или Debian. Этими машинами необходимо будет управлять в ходе практики. Имя пользователя должно быть указано как user. Это важно для дальнейшей работы. В качестве образа ОС рекомендуется использовать дистрибутив Debian, который можно загрузить по ссылке: https://www.debian.org/distrib/

Также при установке пакетов необходимо обязательно выбрать SSH-сервер.

Примечание: *у кого машина слабая, можно создать одну виртуалку.*

После создания и установки ОС необходимо настроить доступ к сети. Для начала в VirtualBox измените тип сетевого подключения на сетевой мост. Это позволит вашей виртуальной машине находится в локальной сети на ровне с физическими устройствами. (Имя сетевого моста у каждого будет отличаться)

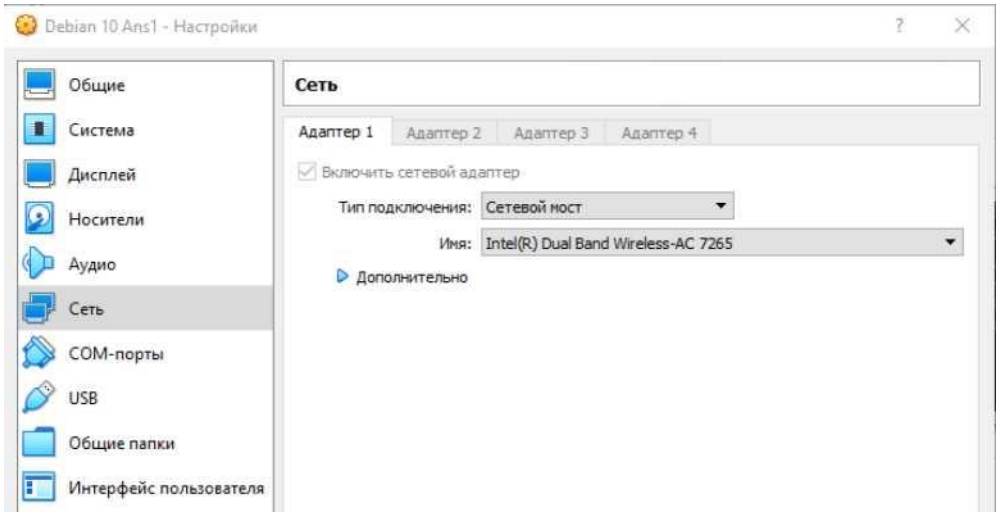

Рис.1 Настройка сети VM

Теперь необходимо настроить ip-адрес для каждой машины.

Сначала необходимо узнать адрес той сети, в которой расположена хостовая машина. Сделать это можно, зайдя в консоль Windows и прописав команду **`ipconfig`**. Для пользователей Linux есть команда **`hostname -I`**. Для пользователей MacOS команды **`ipconfig getifaddr en0`** (при

подключении по Wi-fi) и **`ipconfig getifaddr en1`** (при подключении по Ethernet). Теперь можно приступить к настройке виртуальных машин.

Примечание: Настройку необходимо выполнять из-под пользователя root.

Будем производить настройку при помощи файла **`/etc/network/interfaces`**. Открыв его в редакторе nano, увидите набор параметров для двух интерфейсов lo и что-то похожее на enp0s3. Второй интерфейс необходимо настроить по следующему подобию.

```
allow-hotplug enp0s3
iface enp0s3 inet static
address 192.168.1.5
mask 255.255.255.0
gateway 192.168.1.1
```

- Параметр address назначается по ip, полученному для сети хостовой машины. К примеру, для ip 192.168.1.220 для физической машины, можно настроить ip 192.168.1.5.
- Параметр mask обычно равен 255.255.255.0.
- Параметр gateway чаще всего является первым адресом в сети.

После заполнения перезапустите машину командой reboot.

Ещё одним важным шагом является установка ssh-сервера, который позволит удалённо подключаться к машине. Установить его можно при помощи команды:

apt install ssh

Также необходимо создать управляющую машину, на которой непосредственно будет установлен Ansible. Это может быть как виртуальная машина, тогда необходимо будет проделать те же самые манипуляции, что и для управляемых машин, так и хостовая машина. Для пользователей Windows рекомендуется воспользоваться WSL2, который позволяет поднять Linux прямо на хостовой машине.

Ссылка на инструкцию по установке WSL2: https://docs.microsoft.com/ru-ru/windows/wsl/install-win10#manual-installation-steps

После этого можно переходить к конфигурации машин под Ansible.

Установка Ansible

Для начала необходимо установить сам Ansible на управляющую машину. Сделать это можно при помощи следующих команд:

`sudo apt install ansible` - будет установлена версия Ansible,

содержащаяся в текущем выпуске дистрибутива

`sudo pip3 install ansible` - установка при помощи пакетного менеджера Python 3 (поскольку Ansible написан на Python)

Для установки через pip может потребоваться его установить. Сделать это можно командой **`sudo apt install python3-pip`**.

Наличие Ansible можно проверить командой **`ansible --version`**.

После чего попробуем подключиться к управляемым машинам при помощи ssh.

`ssh root@ip_address`

где **`ip_address`** - то адрес управляемого узла в сети.

Если соединение удастся установить, после подключения система попросит ввести пароль пользователя.

Если нет, необходимо разрешить доступ на машину от пользователя root. Для этого на управляемых машинах необходимо открыть файл /etc/ssh/sshd_config, найти строку PermitRootLogin, раскомментировать её и изменить значение на yes. Итоговая строка должна выглядеть так:

PermitRootLogin yes

После этого необходимо перезапустить службу ssh при помощи команды:

service ssh restart

Теперь на машину должен быть доступ.

Чтобы выйти из ssh соединения необходимо прописать команду exit.

Далее необходимо сгенерировать ключ при помощи команды **`ssh-keygen`**. Эта команда создаст ключ для возможности подключения к удаленному хосту без пароля.

После этого необходимо воспользоваться командой ssh-copy-id для передачи ключа на обе виртуальные машины.

`ssh-copy-id root@ip_address`

Также на каждом управляемом хосте необходимо установить утилиту **`sudo`**. Для этого необходимо сначала залогиниться под суперпользователем при помощи команды **`su -`**, либо же при старте системы сразу выбрать пользователя root. После чего необходимо ввести команду **`apt install sudo`**.

После выполнения всех перечисленных манипуляций инфраструктура готова к управлению через Ansible.

Вся настройка самого Ansible может производится через файл **`/etc/ansible/ansible.cfg`**

Настройка Ansible

Все действия также необходимо выполнять из под пользователя root.

Настроим inventory-файл. Создадим директорию ansible в домашнем каталоге пользователя для хранения репозитория инфраструктуры, будем считать эту директорию рабочей, и в ней создадим файл hosts. Заполните файл по следующему образу. IP-адреса приведены только для примера.

```
[webservers]
server-1 ansible_host=192.168.1.5
server-2 ansible_host=192.168.1.6

[webservers:vars]
ansible_python_interpreter=/usr/bin/python3
```

`webservers` - это название группы хостов, управляемых при помощи Ansible.

`webservers:var` - список переменных для группы хостов `webservers`. В указанном примере задается переменная `ansible_python_interpreter`, которая указывает, какой интерпретатор python должен использоваться на управляемых узлах. Без указания этой переменной может возникнуть ошибка, если хост в качестве интерпретатора по умолчанию используется python второй версии, а не третьей.

Inventory-файл содержит в себе всю необходимую информацию о хостах, которые должны управляться системой Ansible.

Выполним самую первую команду для проверки работы Ansible:

```
ansible -i ./hosts -m ping all
```

Данная команда выполнит команду ping для всех хостов в inventory и выдаст результат выполнения.

Конкретную команду на хостах можно выполнить при помощи следующей конструкции:

```
ansible -i ./hosts -m command -a free all
```

К примеру, данная команда покажет объемы свободной оперативной памяти на всех хостах.

Вместо **`all`** можно указать имя хоста из файла **`hosts`**, чтобы выполнить команду на конкретной машине. Попробуйте самостоятельно выполнить команду uptime на одной из машин.

Использование Ansible для конфигурации хостов

Установка Ansible

Ansible - это инструмент автоматизации без агентов, который вы устанавливаете на одном хосте (называемом узлом управления).

С управляющего узла Ansible может удаленно управлять целым парком машин и других устройств (называемых управляемыми узлами) с помощью SSH, удаленного управления Powershell и множества других транспортных средств, и все это с помощью простого интерфейса командной строки, не требующего баз данных или демонов. Руководство по эксплуатации: https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html#id22

Факты

Факты - это параметры, которыми можно управлять в реализуемых сценариях.

Соберем возможные факты с управляемого хоста server-1:

```
ansible server-1 -i ./hosts -m command -m setup
```

Playbook

Это конфигурационный сценарии, написанный на языке YAML, который впоследствии будет выполнятся на управляемых хостах.

Напишем playbook, который установит веб-сервер Nginx на управляемые хосты. Следующий текст необходимо внести в файл ---

```
- name: Install Nginx to Webservers
  hosts: webservers
  become: yes

  tasks:
  - name: Install Nginx
    apt:
      name=nginx state=present update_cache=yes
    when:
      ansible_os_family == "Debian"
    notify:
      - Nginx Systemd
```

```
handlers:
- name: Nginx Systemd
  systemd:
    name: nginx
    enabled: yes state: started
```

Сам файл с playbook’OM состоит из сценариев. Каждый сценарий начинается с ключевого слова **name**.

tasks - это список задач, которые необходимо выполнить на управляемой машине в ходе сценария. Каждый элемент списка задач начинается с символа дефис (-). Этот символ в языке YAML обозначает начало ассоциативного массива.

При помощи ключевого слова **name** задается имя задачи. При помощи ключевого слова **when** задаются условия для запуска задачи, к примеру в указанном примере задаётся условие, что задача по установке nginx должна выполнятся только на хосте под управлением Debian.

handlers - это действия, которые будут выполняться после завершения задачи. Они запускаются на выполнения только 1 раз и после завершения всего сценария.

Ключевое слово **notify** позволяет запускать handler’bi. В указанном примере после установки nginx будет запущен handler, активирующий автозапуск сервиса nginx.

После написания p1ayЬook’a его можно выполнить при помощи следующей команды **ansible-playbook -i hosts <имя файла playbook’a>**

В результате успешного запуска вы должны увидеть следующий вывод в терминале.

```
PLAY [Install Nginx to Webservers] *****************************************************************

TASK [Gathering Facts] *****************************************************************************
ok: [server-2]
ok: [server-1]

TASK [Install Nginx] *******************************************************************************
changed: [server-1]
changed: [server-2]

RUNNING HANDLER [Nginx Systemd] ********************************************************************
ok: [server-1]
ok: [server-2]

PLAY RECAP *****************************************************************************************
server-1                   : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
server-2                   : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Рис.2 Результат выполнения p1ayЬook’a

Если же попробовать запустить выполнения playbook’a ещё раз, никаких изменений произведено не будет, а будут выведены лишь статусы ok,

означающие соответствие состояния систем с указанным playbook'ом.

После этого можно перейти на любой из двух узлов (при помощи ssh или же непосредственно открыв окно виртуальной машины) и убедиться в том, что nginx действительно установлен при помощи команды **systemctl status nginx**.

Теперь остановим и удалим с данного узла nginx при помощи следующих команд непосредственно через терминал.

systemctl stop nginx

apt remove nginx

Теперь повторно необходимо запустить playbook.

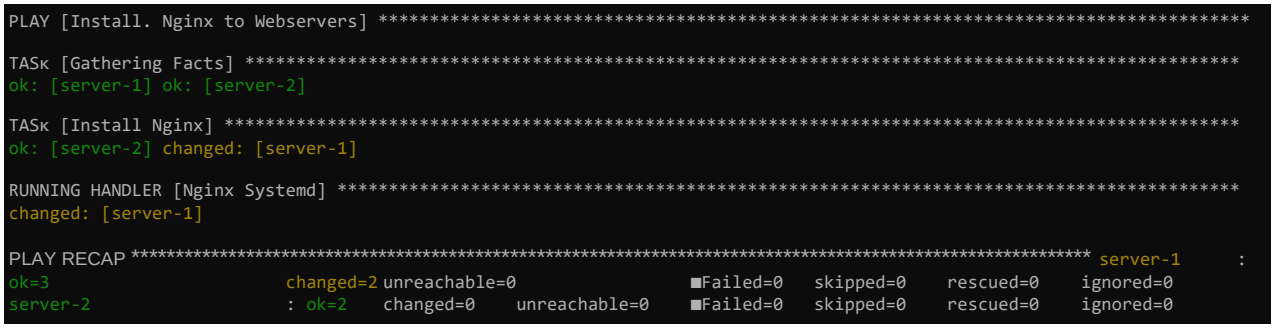

Рис.3 Результат повторного выполнения playbook'а

В результате выполнения можно видеть, что на первом сервере вновь запущен nginx.

Более сложный playbook

Начнем с написания более сложного playbook'а и рассмотрим отдельные его составляющие.

```
- name: Install and config Nginx hosts: webservers become:
  yes

  vars:
    html_dir: /usr/share/nginx/html greeting: "Hello
    Everybody!"

  tasks:
  - name: Install Nginx
    apt:
      name=nginx state=present update_cache=yes when:
      ansible_os_family == "Debian"
    notify:
```

```
      - Nginx Systemd

  - name: Delete default HTML files
    shell: /bin/rm -rf /usr/share/nginx/html/*.html

  - name: Replace config file
    vars:
      nginx_user: user
      worker_processes: 2
      worker_connections: 256
    template:
      src: templates/nginx.conf.j2
      dest: /etc/nginx/nginx.conf
      mode: 0644
    register: result
    failed_when: result.failed == true notify: Reload Nginx

  - name: Copy index file
    copy: src=files/index.html dest={{ html_dir }} mode=0644
    notify: Reload Nginx

  - name: Generate dynamic HTML from template
    template:
      src=templates/hello.html.j2 dest={{ html_dir }}/hello.
html owner=root mode=0644
    notify: Reload Nginx

  handlers:
  - name: Nginx Systemd
    systemd:
      name: nginx
      enabled: yes
      state: started
  - name: Reload Nginx
    systemd: name=nginx state=reloaded
```

В разделе vars описываются переменные, которые затем можно использовать при запуске задач. Для использования переменных в задачах используется синтаксис шаблонизатора - {{ имя_переменной }}.

Ещё одним инструментом является использование шаблонных файлов. Для этого используется шаблонизатор Jinja2. Шаблонные файлы добавляются

в директорию templates с дополнительных расширением j2, то есть итоговое имя файла должно выглядеть, например, следующим образом index.html.j2. При реализации вышеописанного playbook'а используется 2 шаблонных файла: nginx.conf.j2, hello.html.j2.

```
- name: Replace config file
  vars:
    nginx_user: nginx
    worker_processes: 2
    worker_connections: 256
  template:
    src: templates/nginx.conf.j2
    dest: /etc/nginx/nginx.conf
    mode: 0644
  register: result
  failed_when: result.failed == true
  notify: Reload Nginx
```

Использовать шаблонизированные файлы можно при помощи модуля **template**. В качестве параметров задается scr - путь к шаблонному файлу, dest - пусть к файлу на управляемом узле, mode - режим доступа в 8-ричной системе счисления. Помимо этого, используется модуль **register**, позволяющий получить и проверить результат выполнения команды. Все результаты записываются в переменную **result** и при истинном значении параметра **failed**, означающего, что процесс переноса шаблона провалился, весь task становится проваленным благодаря конструкции **failed_when**.

```
- name: Copy index file
  copy: src=files/index.html dest={{ html_dir }} mode=0644
  notify: Reload Nginx
```

Для копирования статического файла используется модуль **copy**. Ему так же задается параметр **scr**, указывающий путь до копируемого файла, параметр **dest**, указывающий путь для файла на целевом хосте, и параметр mode, указывающий режим доступа. Следует отметить, что параметры могут быть записаны в одну строку, как это сделано в данном случае.

```
- name: Generate dynamic HTML from template

  template:
    src=templates/hello.html.j2 dest={{ html_dir }}/hello.ht
ml owner=root group=wheel mode=0644
  notify: Reload Nginx
```

В модуле template также могут быть заданы параметры **owner** и **group**,

обозначающие владельца и группу для переносимого файла.

Содержимое файла **hello.html.j2**

```
"Server {{ ansible_hostname }} ( ip
{{ansible_default_ipv4.address }} ) greets you: {{ greeting
| default("Hello") }}!"
```

Содержимое файла **nginx.conf.j2**

```
# Пользователь, из-под которого будет запущен nginx user {{
nginx_user }};
# Количество рабочих процессов, которые будет задействовать
nginx
worker_processes {{ worker_processes }};

error_log /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

events {
    # Число соединений, которое может поддерживать каждый пр
оцесс
    worker_connections {{ worker_connections }};
}


http {
    include       /etc/nginx/mime.types;
    default_type application/octet-stream;

    keepalive_timeout 60;
    gzip on;

    server {
      access_log off;

      location / {
          root /usr/share/nginx/html;

         try_files $uri $uri/ $uri.html /index.html;
      }
    }
}
```

Также необходимо создать папку **files**, в которой будут храниться все

статические файлы, к примеру файл **index.html**, которым будут заменены стандартные файлы nginx.

Содержимое файла **index.html**

```
Web server is working!
```

В результате содержимое директории должно иметь следующий вид.

Рис.4 Содержимое директории ansible

Теперь выполним команду для запуска playbook’a с пробным прогоном, который позволит проверить корректность написанного playbook’a без внесения изменений на целевые узлы.

ansible-playbook -i hosts <имя файла playbook’a> --check

Примечание: *при использовании флага --check Ansible задействует модуль python, которого может не оказаться на управляемых узлах. Для его установки необходимо выполнить следующую команду:*

ansible -i hosts -m "apt name=python-apt state=latest" all

Вывод покажет, какие изменения будут произведены на узле после применения данной команды, однако на текущий момент никаких изменение произведено не было. В этом можно убедиться, обратившись к одному из узлов при помощи утилиты **curl** для получения базовой html страницы nginx. Сделать это можно командой **curl -L http://192.168.1.6**.

Как можно видеть, полученная страница отличается от той, что должна была быть загружена при помощи Ansible.

Помимо этого, можно задать выполнение playbook’a только на одном узле при помощи флага **--limit**. Например, следующая команда позволит выполнить проверку только на втором сервере.

`ansible-playbook -i hosts playbook-param-nginx.yml -check --limit server-2`

Теперь внесём изменения в конфигурацию первого сервера, убрав ключ **`--check`** из предыдущей команды, а затем запросим базовую страницу при помощи всё той же утилиты **`curl`**.

`curl -L http://192.168.1.6`

Также можно запросить созданную при помощи шаблона страницу hello.

`curl -L http://192.168.1.6/hello`

Как можно увидеть, изменения успешно применились. Можете самостоятельно поменять переменную greetings, затем загрузить повторно playbook на машину и понаблюдать за изменениями.

После этого применим playbook для всех серверов в группе.

Как можно заметить, при небольшом расширении задачи, решаемой при помощи Ansible, файл playbook'а начинает разрастаться, что вызывает неудобство при его чтении.

Роли Ansible

Данный механизм позволяет систематизировать конфигурации путём выделения каждого механизма (задач, переменных, обработчиков и т. д.) в отдельные части.

Ansible имеет похожий на GitHub сервис, называемый Ansible Galaxy. Там находится множество ролей для Ansible, которыми можно воспользоваться. Для использования данного сервиса в Ansible встроена команда ansible-galaxy.

Загрузим в систему роль для установки MySQL при помощи команды

`ansible-galaxy install geerlingguy.mysql`

Найти файлы установленной роли можно по следующему пути **`~/.ansible/roles/geerlingguy.mysql/`**. Рассмотрим более подробно содержимое данной роли. При помощи команды **`tree .`** можно вывести дерево каталога для большой наглядности (если команды tree нет, её можно установить при помощи команды **`apt install tree`**).

- Директория **`defaults`** содержит значения переменных по умолчанию.
- Директория **`handlers`** содержит описание обработчиков.
- Директория **`meta`** содержит информацию о роли, то есть создателе роли, её описание, используемая лицензия, зависимости и т. д. Данный файл используется в дальнейшем для Ansible Galaxy, который будет описан далее.

- Директория **molecule** содержит сценарии тестирования для роли Ansilbe.
- Директория **tasks** содержит непосредственно сценарии для конфигураций.
- Директория **templates**, как можно догадаться, содержит шаблонные параметризованные файлы.
- Директория **vars** содержит файлы описания различных переменных.

Попробуем создать свою роль для установки ранее разобранного playbook'а nginx. Сперва вернёмся в рабочую папку со всеми файлами для Ansible и создадим там директорию **roles**. После чего перейдём в эту директорию и инициируем роль стандартной структуры при помощи команды **ansible-galaxy init nginx**. При помощи команды tree опять же можно опять же посмотреть структуру директории.

Можно заметить, что появилась директория **tests**, она предназначена для тестов роли и запуска её на исполнение и является аналогом директории **molecule**. Также появилась директория **files**, в которой будут храниться статические файлы роли.

Помимо этого можно заметить файл **.travis.yml**. Данный файл предназначен для запуска автоматического тестирования роли через Travis CI.

Перейдём к созданию роли. Заполним соответствующие файлы данными из секций p1ayЬook'а и соответствующие директории ранее созданными файлами.

Это значит, что в директории **tasks** файл **main.yml** должен быть заполнен данными из секции **tasks**. В директории **vars** - из секции **vars**. В директории **handlers** - из секции **handlers**. Также файлы из директорий **files** и **templates** должны быть перемещены в директории **files** и **templates** в папке роли.

Теперь напишем playbook, запускающий созданную роль. Для вызова составленной роли в p1ayЬook'е используется секция **roles**. Создадим файл **nginx-role.yml** в рабочей директории ansible и заполним его следующим образом.

```
- name: Install and config Nginx via Role hosts: webservers
  become: yes

  roles:
    - nginx
```

После этого выполним данный playbook на первом сервере.

Роль должна отработать и показать полное соответствие текущего

состояния требуемому.

Задание

Написать роль для запуска сервера nginx, написать playbook для применения роли, провести тестовый запуск playbook’а, в случае успешного прохождения теста, применить playbook к серверам.

Необходимо добавить переменную, содержащую ФИО, номер группы и номер варианта. Данная переменная должна выводиться в шаблонный файл nginx.

Установка пакета выполняется при помощи модуля APT, используемого для установки nginx в базовой роли.

Добавьте в playbook task по установке пакета согласно варианту:

1. nano
2. imagemagick
3. wget
4. patch
5. php-cli
6. mysql-client
7. jq
8. emacs-nox
9. zip
10. git
11. cowsay
12. apache
13. postgresql-client
14. gpg
15. figlet