Лабораторная работа №9

Управление конфигурациями хостов с помощью Ansible

Теоретическая часть

Однотипная конфигурация множества хостов – одна из важных задач системного администрирования. Предположим, перед системным администратором стоит задача установить на все пользовательские машины организации определенную программу и сконфигурировать её. Очевидное решение – подключиться к каждому из компьютеров и вручную проделать все необходимые действия, однако более правильным стало бы написание скрипта, который будет запущен на всех машинах и проведет необходимую настройку за вас. Централизованным решением данной задачи, без использования сторонних скриптов является система управлением конфигурациями, наиболее распространенной из которых является Ansible.

Ansible — система управления конфигурациями, предназначенная для решения широкого круга задач по автоматизации подготовке и развертыванию IT инфраструктуры организации. Простыми словами, Ansible предназначен для автоматизации настройки серверов, сетевых соединений для большого числа машин. Главное отличие Ansible от существующих аналогичных систем — не нужна установка дополнительного ПО на машины клиентов – взаимодействие происходит посредством SSH. Такая система называется безагентной. В предыдущих лабораторных работах все настройки утилит проводились посредством ручного внесения изменений в конфигурационные файлы. Недостатком такого подхода является невозможность использования системы контроля версий, отсутствие централизации и возможность допустить ошибку при конфигурировании. Система Ansible предлагает другой подход для управления, когда желаемое состояние инфраструктуры описывается с помощью единого программного кода. Такой подход называется IaC - Инфраструктура как код. Сценарии Ansible представляют собой декларативные сценарии на языке yaml.

Алгоритм работы Ansible, в общем случае, выглядит так:

- Серверная машина соединяется с клиентами через SSH, информация о которых берется из файла инвентаря
- Сервер отправляет на клиенты небольшие программы (модули), предназначенные для решения поставленной задачи.
- На основе отправленных модулей с заданными параметрами на клиенте генерируется исполняемый код на языке Python, который реализует поставленную задачу.
- По завершению работы модули на клиентах удаляются.

Особенность Ansible заключается в том, что при каждом запуске сценария система будет проверяться на соответствие желаемому состоянию и при необходимости подвергаться изменениям со стороны Ansible. Другими словами, если мы запустим скрипт с установкой утилиты дважды – то в первый раз она установится, а во второй раз изменения не произойдут, т. е. не будет проходить повторной установки, т.к. система уже находится в желаемом состоянии.

Настройка Ansible на сервере.

Для установки программы потребуется выполнить следующую команду:

```
#Установка пакета Ansible (если не установлен)
sudo apt-get install ansible
```

Ansible разработан на языке Python, поэтому для его корректной работы на клиенте необходимо проверить наличие интерпретатора Python совместимой версии с набором системных библиотек.

Работа с Ansible ведется локально на сервере, посредством установленных утилит, которые получают доступ к хостам через SSH соединение. Поэтому перед использованием, необходимо проверить доступность клиентских машин с помощью протокола ssh и в случае необходимости настроить подключение. Самым простым способом является настройка подключения без использования пароля при создании открытого ключа.

```
#Установка openssh-server (если не установлен)
sudo apt-get install openssh-server
#Создание ключа для SSH соединения (без пароля)
ssh-keygen
#Копирование ключа на машину клиента
ssh-copy-id hostname
```

Настройка файла инвентаря

Для задания адресов машин, на которых необходимо выполнить задачу используется файл инвентаря. Указанный файл называется hosts и может располагаться в одной из директорий, установленных Ansible. В их числе домашняя директория, /etc/ansible/ или директория, указанная пользователем с помощью конфигурационных параметров.

Чтобы добавить список хостов, на которых будут выполняться команды, необходимо перечислить их ip адреса в столбик.

```
192.168.122.1
192.168.122.2
###########
```

Машины возможно объединять в группы. Для этого необходимо перед адресами указать название, под которым они будут объединены.

```
[server]
192.168.122.1
[clients]
192.168.122.2
192.168.122.3
192.168.122.4
```

Далее возможно будет вызывать машины все вместе с помощью ключевого слова all или отдельно, по названию группы.

Запуск скриптов Ansible из командной строки

Самым простым способом отправить команду на все машины – указать её при запуске команды ansible, передав в качестве параметра вместо имени хоста ключевое слово - all. Например, для того, чтобы проверить соединение со всеми хостами необходимо выполнить команду:

```
ansible -m ping all
```

В случае, если необходимо выполнить команду только на группе машин, то вместо ключевого слова all указывается название группы

```
ansible -m ping server
```

Для выполнения команды на удаленных клиентах возможно воспользоваться ключом -a после которого будет следовать команда оболочки bash. Например:

```
ansible -a 'cat /etc/astra_version' all
```

Для того, чтобы запустить команду от администратора, необходимо добавить ключи -b и -K.

```
ansible -m shell -a 'cat /etc/astra_version' –b -K
-b – запустить программу от админа
-K – запросить пароль админа при запуске скрипта
```

Создание и выполнение сценариев (playbook)

Для решения более сложных задач по конфигурации существует возможность описывать их решение в файле сценария (в английской литературе playbook). Сценарии пишутся на языке YAML. Рассмотрим подробно пример сценария

```
- name: Simple playbook
  hosts: all
  become: no
  tasks:
        - name: Whoami
          ansible.builtin.shell:
              cmd: whoami
```

В первых трех строчках сценария задается комментарий, описывающий операцию (name), на каких машинах будет выполняться (all) и требуется ли выдача прав администратора (no). По умолчанию программы запускаются без прав суперпользователя.
Далее, после ключевого слова tasks идет перечисление выполняемых задач. В указанном примере она одна – запуск команды whoami в оболочке каждого из хостов. Вместо ключевого слова shell возможно использовать название одного из поддерживаемых ansible модулей. Ниже приведен список наиболее популярных модулей:

- ansible.builtin.apt (yum) - управление ПО
- ansible.builtin.copy - копирование файла
- ansible.builtin.template - тиражирование шаблонных файлов
- ansible.builtin.file - создание, удаление файлов, изменение атрибутов файлов
- ansible.builtin.lineinfile - вставка, замена, удаление строки в текстовом файле

- ansible.builtin.service - управление службами
- ansible.builtin.user - управление учетными записями пользователей
- ansible.builtin.group - управления учетными записями групп
- ansible.builtin.debug - вывод отладочной информации, значений переменных
- ansible.builtin.command, ansible.builtin.shell - выполнение внешних команд ОС. Рекомендуется применять только в тех случаях, когда задачу невозможно выполнить с использованием модулей ansible.

Обратите внимание на синтаксис YAML файлов. Файлы YAML начинаются с трех дефисов, обозначающих начало документа. Однако Ansible не посчитает ошибкой, если вы забудете указать три дефиса в начале сценария. Комментарии начинаются со знака «решетка» и продолжаются до конца строки, как в сценариях на языке командной оболочки, Python и Ruby. Обычно строки в YAML не заключаются в кавычки (если не используются переменные), даже если они включают пробелы.

В YAML существует понятие списка – перечисление, начинающееся со знака дефис. В данном случае, в начале нашего сценария находится знак «–» для обозначения первого перечисления. Далее указываются переменные и их значения через знак двоеточия. Такой объект называется отображением или словарем.

Обратите внимание, что для отделения задач от описания заголовочной части сценария происходит с помощью **пробелов**, не табуляции!

Для запуска разработанного сценария необходимо выполнить команду:

```
ansible-playbook script.yml
```

В случае успешного запуска команды мы получим сообщение приблизительно следующего содержания:

```
PLAY [Simple playbook]
**********************************************************************

TASK [Gathering Facts]
**********************************************************************
ok: [192.168.122.2]

TASK [Whoami]
**********************************************************************
changed: [192.168.122.2]

PLAY RECAP
**********************************************************************
192.168.122.2              : ok=2    changed=1    unreachable=0    failed=0
```

Несмотря на то, что файл содержит всего одну задачу, было выполнено две – Gathering Facts и Whoami. В первой задаче был произведен сбор фактов о удаленных хостах. В Ansible существует несколько типов специальных зарезервированных переменных: магические переменные, переменные соединения и факты (magic variables, connection variables, facts). Магические переменные автоматически создаются Ansible и не могут быть изменены пользователем. Эти переменные всегда будут отражать внутреннее состояние Ansible. Переменные соединения используются для определения того, как машина, на которой работает Ansible, подключается к удаленным хостам во время выполнения задач и

плейбуков. Факты используются для получения сведений о системе и оборудовании, собранных о текущем хосте во время выполнения плейбука. За их сбор отвечает модуль setup. Посмотреть все собираемые факты с хоста можно командой:

```
ansible -m setup
```

В результате её работы на основную машину будет возвращена структура, содержащая большое число параметров удаленного устройства - операционная система, процессор, память, настройка сети, переменные окружения, интерфейсы и т.д. Эти параметры возможно использовать в дальнейшем, чтобы производить индивидуальную настройку каждого из хостов на основе шаблонов.

В результате работы скрипта сценарий был выполнен успешно. Однако обратите внимание, результат его выполнения не был возвращен на экран. Для этого необходимо использовать переменные.

Переменные в ansible

Переменные в ansible возможно задать с помощью ключа *vars*. Переменные могут указываться для конкретного хоста или группы хостов внутри файла *inventory*:

- Переменная для хоста указывается после имени хоста в виде:
  переменная=значение;
- Переменные для группы задаются внутри секции
  [имя_группы:vars] в формате переменная=значение.

Для вывода значений переменных необходимо создать отдельную задачу с ключом *debug*. Обращение к переменной происходит с помощью двойных фигурных скобок.
Чтобы получить значение из выполненного на другом хосте скрипта, используется ключ *register*, после которого следует название переменной, в которую сохраняется значение.
Рассмотрим следующий пример:

```
- name: Simple playbook
  hosts: client
  become: no
  vars:
    X: Result
  tasks:
        - name: Whoami
          ansible.builtin.shell:
              cmd: whoami
          register: output_data

        - name: Print result
          debug:
              msg: “{{ X }} = {{ output_data.stdout }}”
```

В данном примере создается переменная X, содержащая строку «Result». Её значение подставляется в задаче Print result. Результат выполнения команды *whoami* записывается в объект *output_data,* содержащий несколько полей. Кроме непосредственно самого результата выполнения, в переменную заносится информация об успешности

выполнения операции и некоторые другие служебные данные. Т.к. нас интересует только результат – то мы обращаемся к полю *stdout*.

Результат выполнения скрипта:

```
PLAY [Simple playbook]
*********************************************************************

TASK [Gathering Facts]
*********************************************************************
ok: [192.168.122.2]

TASK [Whoami]
*********************************************************************
changed: [192.168.122.2]

TASK [Print result]
*********************************************************************
ok: [192.168.122.2] => {
    "msg": "Result = adminstd"
}

PLAY RECAP
*********************************************************************
192.168.122.2              : ok=3    changed=1    unreachable=0    failed=0
```

Обратите внимание, что было выполнено две задачи - Whoami и Print result.

Строки можно преобразовать в массив, в котором переменные отделены определенным символом разделителем. Для получения определенного значения необходимо выполнить метод split('/'), где в скобках указан символ разделения, а далее обратиться к переменной, как к элементу массива с помощью номера в квадратных скобках. Например: `result.stdout.split('.')[2].` Для объединения элементов массива обратно в строку возможно использовать метод join(). Например:

```
- hosts: localhost
  vars:
    my_list_of_strings:
      - "Hello"
      - "World"
      - "Ansible"
  tasks:
    - name: Join the list of strings
      debug:
        msg: "{{ my_list_of_strings | join(', ') }}"
```

Теги

Для того, чтобы запустить (или исключить запуск) части тасков в плейбуке используется понятие тега. Тегировать можно как play так и task. Для каждого элемента может быть назначено более одной метки.

Синтаксис тега:

```
tags: [ tag1, tag2, … ]
```

Для вызова задач по определенному тегу необходимо указать его при запуске команды:

```
ansible-playbook playbook.yml --tags “tag1,tag2”
```

Примеры сценариев

Рассмотрим несколько примеров сценариев Ansible.

Установка утилиты

```
- name: Install DNS server package
  ansible.builtin.apt:
    name: bind9
    state: latest
```

Создание директории

```
      - name: Create directory
        ansible.builtin.file:
              path: ~/work
              state: directory
```

Создание файла

```
       - name: Create file
         ansible.builtin.file:
               path: ~/test
               state: touch
```

Копирование файла с сервера на клиент

```
       - name: Copy file on client
         ansible.builtin.copy:
               src: ~/test
               dest: ~/work/test
```

Копирование файла на клиенте

```
       - name: Copy file on client
         ansible.builtin.copy:
               src: ~/test
               dest: ~/work/test
               remote_src: yes
```

Условный оператор

В ansible возможно выполнять или пропускать некоторые задачи в зависимости от выполнения условия. Для этого используется ключевое слово when.

```
- name: Simple playbook
  hosts: client
  become: no
  vars:
    X: 10
  tasks:
    - name: if/when X > 5
      ansible.builtin.debug:
        msg: "> 5"
```

```
      when: X|int > 5

    - name: if/when X < 5
      ansible.builtin.debug:
        msg: "< 5"
      when: X|int < 5
```

Обратим внимание на строку «when: X|int > 5». В данном случае с помощью конструкции X|int мы приводим переменную X к целому типу и далее сравниваем с числом 5. Т.к. значение X=10 > 5, то данное задание выполняется. Следующее задание напротив, будет пропущено.

```
TASK [if/when X > 5]
*********************************************************************
ok: [192.168.122.2] => {
    "msg": "> 5"
}

TASK [if/when X < 5]
*********************************************************************
skipping: [192.168.122.2]
```

Оператор цикла

Очень часто в ansible необходимо выполнить ряд однотипных задач – скопировать множество файлов с сервера на клиент, установить несколько утилит и т.п. Каждую задачу возможно обернуть в отдельный таск, однако правильным решением является применение циклов.

Для использования циклов в конструкции task необходимо добавить строку «loop:», после которой перечисляются переменные, к котором будет проходить обращение. Например,

```
---
- hosts: all
  tasks:
    - name: creates files
      ansible.builtin.file:
        path: “/tmp/{{ item }}”
        state: touch
      loop:
        - poems.txt
        - novels.txt
```

Редактирование текста

Для редактирования текстовых файлов используется ключ lineinfile

Добавить строку в файл:

```
    - name: Place line
      ansible.builtin.lineinfile:
        line: Hello World
```

```
    path: hello.txt
    create: true
```

Удалить строку из файла:

```
- name: Remove line
  ansible.builtin.lineinfile:
    line: Hello World
    path: hello.txt
    state: absent
```

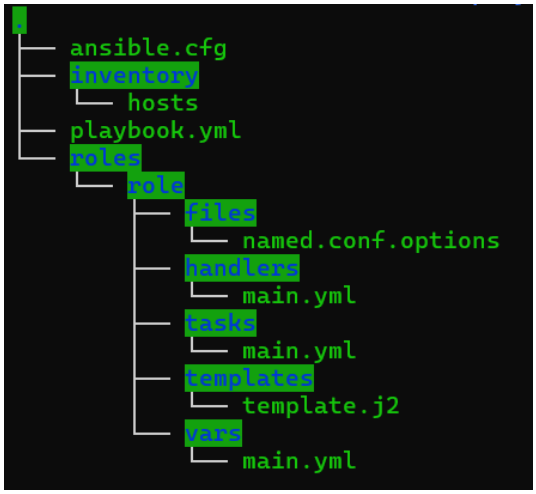
Изменение файла:

```
- name: Change file
  ansible.builtin.lineinfile:
   line: Passwords no
   regexp: ^Passwords
   path: ~/work/secret_file
```

Предположим, существует файл, содержащий строку Passwords yes. Для изменения значения этой строки найдем строку, начинающуюся со слова Passwords (regexp: ^Passwords) и изменим значение на Passwords no (line: Passwords yes)

Разработка ansible-playbooks для решения промышленных задач.

Для решения задач промышленного масштаба, когда число настраиваемых сервисов велико и их настройка не является тривиальной задачей, рекомендуется придерживаться определенного стиля разработки и размещения скриптов в файловой системе. Рассмотрим такую систему подробнее.



В директории, содержащей playbook создается файл ansible.cfg, содержащий основные настройки для работы ansible. Ниже приведен пример подобного файла:

```
[defaults]
inventory = ./inventory/hosts
verbosity = 0
remote_user = user
private_key_file = ~/.ssh/id_ed25519
host_key_checking = False

[privilege_escalation]
```

```
become = True
become_method = sudo
become_user = superuser
```

В разделе defaults находятся настройки по-умолчанию. Указывается путь до файла inventory, уровень детализации вывода (verbosity, 0 – выводится без технических подробностей, 3 – максимальный уровень), имя пользователя, из под которого происходит работа на удаленной системе (remote_user, по умолчанию совпадает с пользователем на устройстве, от которого происходит запуск ansible), путь до закрытого SSH ключа (private_key_file), и осуществление проверки ключей при подключении по SSH (host_key_checking). Отдельно возможно указать параметры, выполняемые при повышении привилегий доступа (privilege_escalation).

Как было указано выше, директория inventory содержит файл hosts, в котором указывается список всех клиентов. Например:

```
[server_host]
localhost ansible_connection=local ansible_user=user
ansible_become_pass=12345

[clients]
client ansible_host=192.168.122.4 ansible_user=user
ansible_become_pass=qwerty22
```

Первым параметром указывается доменное имя устройства, далее IP адрес, логин и пароль для подключения и выдачи привилегий.
Директория roles содержит описания ролей ansible – рассмотрим их подробнее.

Роли

Роль в контексте Ansible — это набор задач, обработчиков событий, переменных, файлов для копирования, шаблонов и т. д., которые распространяются и подключаются как единое целое к основному сценарию (playbook). Роли обычно отвечают за высокоуровневые задачи: установку и конфигурирование какого-либо сервиса (баз данных, веб-серверов и др.) В зависимости от сложности выполняемых задач, файлы конфигурации роли могут быть разнесены по нескольким каталогам. Вот некоторые из них:

- defaults: значения по умолчанию для используемых переменных (не обязательно всех) для включенных или зависимых ролей.
- files: статические файлы, которые копируются на конфигурируемый сервер без изменения содержимого.
- handlers: обработчики событий, описывается как обычный таск, но, который должен быть связан с каким-то таском из перечня выполняемых в рамках роли, вызываются только по необходимости, при изменении состояния системы основным таском.
- meta: метаданные роли, часто используются для управления зависимостями. Например, какие роли должны быть применены на конфигурируемом хосте до вызова текущей роли.

- templates: содержит файлы шаблонов для шаблонизатора jinja2, который генерирует конечные файлы согласно прописанным в шаблонах условиям и правилам используя переменные.
- tasks: содержит один или несколько файлов с задачами, которые определяются подобно оным в разделе tasks обычного плейбука Ansible. Эти задачи могут напрямую ссылаться на файлы и шаблоны, содержащиеся в соответствующих каталогах внутри роли, без необходимости указывать полный путь к файлу.
- vars: содержит файл(ы), содержащий(е) переменные для роли.

Практическая работа

1. Настройте ansible на серверной машине. В качестве клиентов выберете обе машины – сервер и клиент.
2. Проверьте доступность всех устройств с помощью команды ping используя запуск скрипта ansible
3. Используя ansible, запустите на машине клиента скрипт, выводящий объем свободной оперативной машины
4. Создайте playbook, выполняющий следующие задания:
    1.1. Создайте директории ServerВАШИИНИЦИАЛЫ на сервере и ClientВАШИИНИЦИАЛЫ на машине клиента соответственно. Данные директории создаются в домашней директории пользователя.
    2.1. Создайте файлы с названием info в домашней директории. Добавьте проверку на существование файла. В случае его наличия файл повторно не создается.
    3.1. Заполните данные файлы информацией о системе, включающей в себя имя машины, вашу фамилию, ip адрес, объем занятой оперативной памяти (в Mb), среднюю нагрузку за последние 15 минут работы (см. файл /proc/loadavg). Формат записи: astra001 | Ivanov | 192.168.122.1 | 722 | 1.58
    4.1. Скопируйте данный файл в созданную в п. 1.1 директорию.
    5.1. Измените в перемещенном файле значение вашей фамилии на ваше имя.
    6.1. В зависимости от значения нагрузки в файле выведите сообщение на экран. Если нагрузка больше 1: state NAME_MACHINE bad. Если меньше 1, то state NAME_MACHINE good.

*Дополнительные задания (со *)*
Указанные ниже задания рекомендуются к выполнению всеми студентами, но допустимо их пропустить. Задания обязательны для тех, кто будет выполнять 8-ю лабораторную работу. Их возможно сдавать вместе с 8ой лабораторной работой.

1. Возьмите из приложения файлы для ansible-playbook, разворачивающий DNS-сервер BIND9. Ознакомьтесь с содержимым файлов. При необходимости поправьте файл инвентаря и файл переменных так, чтобы после развертывания система имела имя хоста и домен, соответствующий предыдущим лабораторным работам.
2. Запустите ansible-playbook с вашей основной машины, на которой установлен менеджер виртуальных машин. Убедитесь, что DNS сервер установился корректно.
3. Добавьте в файл с переменными необходимые данные для того, чтобы в зону DNS была добавлена запись с произвольным именем, указывающая на основную A-запись Вашего сервера (алиас). Для проверки работоспособности запустите плейбук с дополнительным параметром ansible-playbook ./playbook.yml --tags

“untagged,my_dns”. Дополнительно, с хостовой ВМ, вытупающей сервером Ansible проверьте, что разрешаются оба FQDN.

4. Самостоятельно на основе примера вышеупомянутой роли DNS, создайте собственную роль, предназначенную для развертывания веб-сервера NGINX. К роли предъявляются следующие требования:
    1.1. Действия в тасках должны реализовываться специализированными модулями ansible, использование модулей command и shell, команд bash без веской причины запрещается.
    2.1. Для nginx должны создаваться на основе шаблонов три файла конфигурации - основной nginx.conf и два файла с конфигурацией сайтов (секция server) для каждого из FQDN (A-записи и CNAME-записи в DNS)
    3.1. В основном файле должны быть параметризованы следующие параметры:
        i. Количество процессов (workers) выставляется по количеству ядер процессора
        ii. Обработчик соединений (use) выставляется в epoll
        iii. Параметр worker_connections также задается переменной (произвольно, но ОСМЫСЛЕННО)
    4.1. В конфиге сайтов параметризуются порт и адрес прослушивания (могут быть одинаковы для обоих сайтов), директория (root) и адрес сайта (один из fqdn). Оба файла должны генерироваться и копироваться на клиент за один таск. Не забудьте, что изменение файлов конфигурации должно сопровождаться перезапуском сервиса. Разные сайты должны отдавать разные по содержанию странички (HTML файлы), чтобы было видно отличие при проверке.
    5.1. Обратите внимание, код вашей роли, подобно примеру роли DNS, должен по возможности использовать факты и другие встроенные переменные Ansible. Не забудьте добавить задание на проверку корректности конфигурации nginx (в этом таске использовать команды bash можно).
    6.1. На сервере Ansible проверьте через браузер, что NGINX отдает по разным именам разные странички.

Вопросы для проверки:

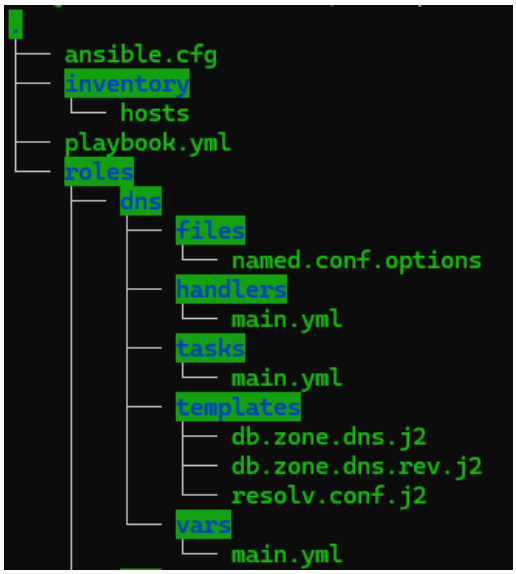
1. Какая команда используется для проигрывания сценариев?
2. Как называется ключ в play, в котором указываются хосты для выполнения заданий (task)?
3. Что нужно сделать для того, чтобы задания в сценарии выполнялись с привилегиями администратора?

Список литературы

[1] Л. Хоштейн и Р. Мозер, Запускаем Ansible, Москва: ДМК, 2018.

[2] «Документация ansible,» 07 11 2023. [В Интернете]. Available: https://docs.ansible.com/.

Приложение



Файлы ansible.cfg и inventory/hosts совпадают с файлами, приведенными выше.

playbook.yaml

```
---
- name: Provision dns server
  hosts: server.mpsu.stu
  become: true

  pre_tasks:
    - name: Set hostname on your provisioned server
      block:
        - name: Set static hostname
          ansible.builtin.hostname:
            name: "{{ inventory_hostname_short }}"

        - name: Add FQDN to /etc/hosts
          ansible.builtin.lineinfile:
            dest: /etc/hosts
            regexp: '^127\.0\.1\.1'
            line: "127.0.1.1 {{ inventory_hostname }} {{
inventory_hostname_short }}"
  roles:
    - dns
    #- web

  post_tasks:
    - name: Save new DNS server ip_address
      ansible.builtin.set_fact:
        server_ip: "{{ ansible_host }}"
        cacheable: yes
      tags: always

- name: Set new DNS server address on Ansible server
  hosts: localhost
```

```
  become: true
  vars:
    dns_ip: "{{
hostvars['server.dns.lab']['ansible_facts']['server_ip'] }}"

  tasks:
    - name: set new dns server on localhost
      ansible.builtin.lineinfile:
        path: /etc/resolv.conf
        line: "nameserver {{ dns_ip }}"
        insertbefore: '^nameserver*'
        firstmatch: yes
        state: present
      tags: [ never, my_dns ]

    - name: unset new dns server on localhost
      ansible.builtin.lineinfile:
        path: /etc/resolv.conf
        line: "nameserver {{ dns_ip }}"
        state: absent
      tags: [ never, default_dns ]
```

files/named.conf.options

```
options {
        directory "/var/cache/bind";

        forwarders {
            77.88.8.8;
        };

        // dnssec-validation auto;

        listen-on {
            any;
        };
        listen-on-v6 { none; };
};
```

handlers/mail.yaml

```
---
- name: restart bind
  service:
    name: "{{ service_name }}" #bind9
    state: restarted
```

tasks/main.yaml

```yaml
---
- name: Install DNS server package
  ansible.builtin.apt:
    name: "{{ item }}"
    state: latest
    update_cache: yes
    cache_valid_time: 36000
  loop:
    - "{{ service_name }}"

- name: Ensure BIND service is started and enabled
  ansible.builtin.service:
    name: "{{ service_name }}"
    state: started
    enabled: yes

- name: Generate zone files from templates
  ansible.builtin.template:
    src: "templates/{{ item }}"
    dest: "/etc/bind/{{ item.split('.')[:-1] | join('.') }}"
    owner: root
    group: bind
    mode: 0660
  loop:
    - db.zone.dns.j2
    - db.zone.dns.rev.j2
  notify:
    - restart bind

- name: Add info about new zone in config
  ansible.builtin.blockinfile:
    path: /etc/bind/named.conf.local
    insertbefore: BOF
    marker: "// {mark} ANSIBLE MANAGED BLOCK"
    block: |
      zone "{{ domain_zone }}"  { type master; file
"/etc/bind/db.zone.dns"; };
      zone "{{ network.split('.0')[0].split('.')[-1::-1] | join('.')
}}.in-addr.arpa"  { type master; file "/etc/bind/db.zone.dns.rev"; };
    state: present
  notify:
    - restart bind

- name: Copy general BIND config
  ansible.builtin.copy:
    src: files/named.conf.options
    dest: /etc/bind/
    owner: root
    group: bind
    mode: 0644
  notify:
```

```
    - restart bind

- name: Check validity of named.conf
  ansible.builtin.command: named-checkconf
  register: bind9_reg_named_checkconf
  changed_when: False

- name: Check validity of zone file
  ansible.builtin.command: named-checkzone {{ domain_zone }}
/etc/bind/db.zone.dns
  register: bind9_reg_named_checkzone
  changed_when: False

- name: Check validity of reverse zone file
  ansible.builtin.command: named-checkzone {{
network.split('.0')[0].split('.')[-1::-1] | join('.') }}.in-addr.arpa.
/etc/bind/db.zone.dns.rev
  register: bind9_reg_named_checkzone
  changed_when: False
```

templates/db.zone.dns.j2

```
$TTL 3600
$ORIGIN {{ domain_zone }}.
@               IN      SOA {{ host }}.{{ domain_zone }}. root.{{
domain_zone }}. (
                            2711201408 ; serial
                            3600       ; refresh (1 hour)
                            600        ; retry (10 minutes)
                            86400      ; expire (1 day)
                            600        ; minimum (10 minutes)
                        )

                IN      NS      {{ host }}.{{ domain_zone }}.

; DNS Servers
{{ host }}      IN      A      {{ ip_address }}

; Web servers and etc
{% if dns_records is defined  %}
{% for dict_item in dns_records %}
{{ dict_item['host'] }}           IN      {{ dict_item['type']
}}     {{ dict_item['value'] }}
{% endfor %}
{% endif %}
```

templates/db.zone.dns.rev.j2

```
$TTL 3600
```

```
$ORIGIN {{ network.split('.0')[0].split('.')[-1::-1] | join('.')
}}.in-addr.arpa.
@               IN      SOA     {{ host }}.{{ domain_zone }}. root.{{
domain_zone }}. (
                            2711201407 ; serial
                            3600       ; refresh (1 hour)
                            600        ; retry (10 minutes)
                            86400      ; expire (1 day)
                            600        ; minimum (10 minutes)
                        )

                IN      NS      {{ host }}.{{ domain_zone }}.

{{ ip_address.split(network.split('.0')[0]+'.')[1]
}}              IN      PTR     {{ host }}.{{ domain_zone }}.
{% if dns_records is defined  %}
{% for dict_item in dns_records %}
{% if dict_item['type'] == 'A' %}
{{ dict_item['value'].split(network.split('.0')[0]+'.')[1]
}}          IN          PTR       {{ dict_item['host'] }}.{{ domain_zone
}}.
{% endif %}
{% endfor %}
{% endif %}
```

templates/resolv.conf.j2

```
domain {{ domain_zone }}
search {{ domain_zone }}
{% if inventory_hostname == 'server.mpsu.stu' %}
nameserver {{ ip_address }}
{% else %}
nameserver 127.0.0.1
{% endif %}
```

vars/main.yaml

```
---
host: "{{ inventory_hostname_short }}"
dns_records:
   - { host: 'web', type: 'CNAME', value: "{{ inventory_hostname }}."
}
domain_zone: "{{ inventory_hostname.split('.')[1::] | join('.') }}"
ip_address: "{{ ansible_default_ipv4.address }}"
network: "{{ ansible_default_ipv4.network }}"
service_name: bind9
```