???? inventory ? ??????????

Список хостов с тэгами (группами), на которые впоследствии ссылаются в задачах.

Последовательность поиска файла

По умолчанию: /etc/ansible/hosts

Однако проще указывать файл при запуске playbook

```
ansible-playbook -i my_inventory.ini playbook.yml
```

Наследование переменных

- переменные группы all
- переменные родительской группы
- переменные дочерней группы
- переменные хоста

Варианты запуска inventory файла

```
ansible-playbook -i prod.ini


ansible.cfg:
[defaults]
inventory = /root/ansible_playbooks/inventory/cloud_inv.py
```

Проверка интерпретации файла:

```
# Покажет структуру inventory в JSON
ansible-inventory -i ваш_файл --list


# Покажет в удобном для человека виде
ansible-inventory -i ваш_файл --graph
```

Можно в директории хранить несколько файлов и использовать их по необходимости.

```
# Посмотреть все хосты из inventory
ansible all -i inventory.ini --list-hosts
```

```
# Посмотреть переменные конкретного хоста
ansible -i inventory.ini web1 -m debug -a "var=hostvars[inventory_hostname]"

# Выполнить команду на группе серверов
ansible web_servers -i inventory.ini -m command -a "df -h"
```

Форматы файлов:

Есть динамический, ini и yaml формат. Определяет формат по содержанию файла, расширение файла (.ini, .yaml, .yml) не играет роли. Порядок анализа:

- Является ли файл исполняемым — если да, то запускает его как динамический inventory
- Пытается распарсить как INI — если получается, использует INI-формат
- Пытается распарсить как YAML — если получается, использует YAML-формат
- Если ничего не подошло — выдает ошибку

Важно:

1. Нельзя смешивать форматы в одном файле
2. Способ отличия форматов:
   INI-формат: начинается с секций в квадратных скобках [group]
   YAML-формат: начинается с --- (опционально) или со слова all: / children: / hosts:
3. Автодетект может ошибаться. Если будет файл, валидный как INI и как YAML, то Ansible выберет INI.

ini формат:

Ожидается, что hosts в формате ini.

```
[example]
www.example.com
192.168.0.1
192.168.0.2:2222
```

Внутри [] название группы, далее состав этой группы (доменное имя или IP). Указание порта определяет порт ssh сервера.

Группы по умолчанию

- Группа all - все хосты
- Группа ungrouped - хосты без группы

Вложенные группы

При указании :children создается метагруппа. То есть

```
[staging:children]
web_servers
db_servers
```

создаст группу staging, которая будет состоять из хостов группы web_servers и db_servers. Это именно группы, хосты указывать в метагруппе нельзя. Можно городить любую иерархию, одна метагруппа внутри другой.

Диапазоны

```
web[01:05].example.com
```

Переменные

Для хоста переменные через пробел в одной строке

```
[databases]
192.168.1.20 ansible_user=ubuntu ansible_ssh_private_key_file=~/.ssh/ubuntu-key.pem
```

При помощи :vars задаются переменные для группы

```
[staging:vars]
ansible_user=staging_user
env=staging
app_version=latest
```

yaml формат

Когда использовать YAML	Когда проще INI
Сложная иерархия групп (3+ уровня) Много переменных со сложными типами (списки, словари) Храните инвентарь в Git (лучше для code review) Команда привыкла к YAML	Мало хостов (<20) Простая структура (1-2 уровня групп) Быстрые временные инвентари

Корневая группа одна — all. Группа all является специальной встроенной группой Ansible:

- в неё входят все хосты inventory
- все группы являются её потомками (напрямую или косвенно)

Поддерживает диапазоны аналогично ini

Если будет несколько корневых групп, то будет ошибка. Структура файла настроек:

```
all:
  hosts:
  vars:
  children:
```

Хосты:

Внутри hosts указываются переменные для хоста.

```
all:
  hosts:
    server1:
      ansible_host: 192.168.1.10
    server2:
      ansible_host: 192.168.1.11
```

Переменные:

```
vars:
  http_port: 80
  app_user: nginx
```

Переменные из all.vars наследуются всеми хостами

Теперь переменные доступны в playbook

```
- name: Show port
  debug:
    msg: "{{ http_port }}"
```

Переменные переопределяются по иерархии.

Создание дальнейшей вложенной структуры создает вложенную структуру переменной:

```
all:
  hosts:
    db-master:
      replication:
        role: master
        sync_mode: synchronous
```

То есть у хоста db-master переменные replication.role и replication.sync_mode. Можно также использовать списки:

```
vars:
  allowed_ips:
    - 192.168.1.0/24
    - 10.0.0.0/8
```

Группы:

Внутри children указываются имена дочерних групп и их настройки. Структура дочерних групп аналогичная. Допускается множественное вложение.

```
all:
  children:
    webservers:
      hosts:
        web1:
          ansible_host: 10.0.0.10
        web2:
          ansible_host: 10.0.0.11

    dbservers:
      hosts:
        db1:
          ansible_host: 10.0.0.20
```

Множественное вложение:

```
all:
  children:
    msk_web:
      hosts:
        msk-web-01:
        msk-web-02:
    msk_db:
      hosts:
        msk-db-01:
    spb_web:
      hosts:
        spb-web-01:
```

```
    spb_db:
      hosts:
        spb-db-01:

    msk:
      children:
        msk_web
        msk_db
      vars:
        datacenter: moscow
        region: central

    spb:
      children:
        spb_web
        spb_db
      vars:
        datacenter: st_petersburg
        region: north-west

    russia:
      children:
        msk
        spb

    production:
      children:
        russia
      vars:
        env: production
        monitoring_enabled: true
```

Однако при усложнении правил (субъективно) теряется сама идея простоты настройки. То есть приходится объяснять правила объединения, рисовать зависимости, ... А когда переменные еще и отличаются для разных групп, и этих групп много, то все становится совсем плохо.

Выполнение playbook только для группы:

```
ansible-playbook -i inventory.yml site.yml --limit webservers
```

Можно использовать якоря

```
defaults: &defaults
  ansible_user: admin
  ansible_port: 22

webservers:
  hosts:
    web1:
      <<: *defaults
      ansible_host: 192.168.1.10
    web2:
      <<: *defaults
      ansible_host: 192.168.1.11
```

Разделение данных на файлы

Предыдущий текст относится к ситуации, когда хосты и переменные расположены в одном файле. Есть возможность разбить проект на специально названные файлы. Например:

```
inventory/
└── testing/
    ├── hosts.yml
    ├── group_vars/
    │   ├── all.yml
└── production/
    ├── hosts.yml
    ├── group_vars/
    │   ├── all.yml
    │   ├── webservers.yml
    │   └── dbservers.yml
    └── host_vars/
        ├── web1.yml
        └── db1.yml
```

Создана директория inventory, внутри окружения testing и production, в каждом окружении файл hosts.yml и директории group_vars и host_vars. Запуск playbook будет выглядеть так:

```
ansible-playbook -i inventory/production/hosts.yml site.yml
```

hosts.yml: только хосты/группы, стандартная структура без переменных.

```
all:
  children:
    webservers:
      hosts:
        web1:
        web2:

    dbservers:
      hosts:
        db1:
```

group_vars/all.yml: Переменные для всех групп, все пишется на одном уровне

```
ansible_user: ubuntu
timezone: Europe/Amsterdam
```

group_vars/<group_name>.yml: Переменные для группы group_name, все пишется на одном уровне

host_vars/<host_name>.yml: Переменные только для хоста <host_name>

Динамический файл

Это обычный исполняемый файл, который:

- Может быть написан на любом языке (Python, Bash, Go, Ruby и т.д.)
- При запуске возвращает JSON с описанием хостов и групп
- Ansible запускает его вместо чтения статического файла

```
ansible-playbook -i dynamic_inventory.py playbook.yml
```

Пример структуры JSON:

Например API отдает следующий JSON

```
{
  "servers": [
    {
      "id": "i-12345abc",
      "ip_address": "192.168.1.10",
      "type": "web",
      "environment": "production",
```

```
    "name": "web-server-01",
    "status": "running",
    "region": "us-east-1",
    "instance_type": "t2.medium"
  },
  {
    "id": "i-12345def",
    "ip_address": "192.168.1.11",
    "type": "web",
    "environment": "production",
    "name": "web-server-02",
    "status": "running",
    "region": "us-east-1",
    "instance_type": "t2.medium"
  },
  {
    "id": "i-67890ghi",
    "ip_address": "192.168.1.20",
    "type": "db",
    "environment": "production",
    "name": "db-server-01",
    "status": "running",
    "region": "us-east-1",
    "instance_type": "r5.large"
  },
  {
    "id": "i-67890jkl",
    "ip_address": "192.168.1.21",
    "type": "db",
    "environment": "staging",
    "name": "db-server-02",
    "status": "running",
    "region": "us-west-2",
    "instance_type": "r5.large"
  },
  {
    "id": "i-99999mno",
    "ip_address": "192.168.1.30",
    "type": "web",
    "environment": "staging",
```

```
      "name": "web-server-03",
      "status": "stopped",
      "region": "us-west-2",
      "instance_type": "t2.micro"
    }
  ]
}
```

Тогда скрипт, разбирающий этот JSON в формат Ansible

```
#!/usr/bin/env python3
import json
import requests

def get_servers():
    # Получаем список серверов из API
    response = requests.get('https://api.mycloud.com/v1/servers',
                            headers={'Authorization': 'Bearer token123'})
    servers = response.json()

    inventory = {
        'all': {
            'hosts': [],
            'vars': {}
        },
        'web': {
            'hosts': [],
            'vars': {'role': 'web'}
        },
        'db': {
            'hosts': [],
            'vars': {'role': 'database'}
        },
        '_meta': {
            'hostvars': {}
        }
    }

    for server in servers:
        ip = server['ip_address']
```

```
        inventory['all']['hosts'].append(ip)

        if server['type'] == 'web':
            inventory['web']['hosts'].append(ip)
        elif server['type'] == 'db':
            inventory['db']['hosts'].append(ip)

        # Переменные для конкретного хоста
        inventory['_meta']['hostvars'][ip] = {
            'ansible_host': ip,
            'environment': server['environment'],
            'instance_id': server['id']
        }

    print(json.dumps(inventory, indent=2))

if __name__ == '__main__':
    get_servers()
```

Использование скрипта

```
# Сделать скрипт исполняемым
chmod +x inventory.py

# Проверить, что выдает скрипт
./inventory.py

# Использовать с Ansible
ansible-inventory -i inventory.py --list
ansible-inventory -i inventory.py --graph

# Выполнить команду на всех хостах
ansible all -i inventory.py -m ping

# Выполнить команду только на web-серверах
ansible web -i inventory.py -m shell -a "nginx -v"

# Выполнить команду только на db-серверах
ansible db -i inventory.py -m shell -a "psql --version"
```

Важные правила

- Хост может быть в нескольких группах
- Переменные переопределяются. Приоритет (от низшего к высшему):
  - all vars
  - Групповые vars
  - _meta hostvars (самый высокий приоритет)
- Имена хостов и групп
  - Могут содержать буквы, цифры, _, -
  - Чувствительны к регистру
  - Не должны начинаться с _ (зарезервировано для _meta)

Элементы JSON

Группы ("group_name")

Каждая группа может содержать три ключа:

```
{
  "group_name": {
    "hosts": ["host1", "host2"],     // список хостов в группе
    "vars": {                        // переменные для всей группы
      "var1": "value1",
      "var2": "value2"
    },
    "children": ["child_group1", "child_group2"]  // вложенные группы
  }
}
```

Специальная группа "all"

Обязательная группа, которая содержит все хосты:

```
{
  "all": {
    "hosts": ["web1", "web2", "db1"],
    "vars": {
      "ansible_python_interpreter": "/usr/bin/python3",
      "ntp_server": "pool.ntp.org"
    }
  }
}
```

Специальная группа "_meta"

Содержит переменные для конкретных хостов:

```
{
  "_meta": {
    "hostvars": {
      "web1": {
        "ansible_host": "10.0.0.1",
        "ansible_user": "deploy"
      },
      "web2": {
        "ansible_host": "10.0.0.2"
      }
    }
  }
}
```

При использовании _meta, в группах можно просто перечислять имена хостов без переменных.

Revision #8
Created 13 June 2026 15:10:04 by Admin
Updated 23 June 2026 13:47:07 by Admin