ansible node1 -m setup #Makina ile ilgili bütün bilgiler(facts)

Playbook1:

- name: show facts

  hosts: all

  tasks:

    - name: print facts

      ansible.builtin.debug:

        var: ansible\_facts

Yukarıda kod ile yazdığımızı playbook1 haline çevirdik.

Playbook2:Burada makinanın bütün bilgilerinden ziyade sadece ipv4 addressini getirecek bize

- hosts: all

  tasks:

  - name: show IP address

    ansible.builtin.debug:

      msg: >

       This host uses IP address {{ ansible\_facts.default\_ipv4.address }}

ansible-vault create secret.yml #Bu kodda bir şifreli bir dosya oluşturuyoruz ve bu dosyaya ne kaydedersek onlar bu şifreli dosya içinde olacak mesela aşağıdaki örnekte gibi:

username: tyler

password: 99abcd

cat secret.yml #içeriğini okuyunca bu yml dosyasının bir sürü rakam gelecek

Playbook create-user.yml:

- name: create a user

  hosts: all

  become: true

  vars\_files:

    - secret.yml

  tasks:

    - name: creating user

      ansible.builtin.user:

        name: "{{ username }}"

        password: "{{ password }}"

Yukarıda yaml kodunda biz değişken olaak secret.yml verdik.Bunun içine gir ve ordan bize

        name: "{{ username }}"

        password: "{{ password }}" bunları getir dedik.

ansible-playbook create-user.yml #terminalde bunu yazdık playbook’un çalışması için

çıktı:

ERROR! Attempting to decrypt but no vault secrets found

ansible-playbook --ask-vault-pass create-user.yml #Yukardaki playbook böyle çalıştırmamız gerekiyor.

ansible all -b -m command -a "grep tyler /etc/shadow"

çıktı:

node1 | CHANGED | rc=0 >>

tyler:99abcd:18569:0:99999:7:::

şifre

#Bilgileri şifrelesek bile böyle yazınca şifreler gözüküyor.Yani vault kodu ile şifrelemek pek güvenli değil onun yerine şu playbooku yazarız.

Yeni bir kullanıcının bilgilerine gizlersek:

ansible-vault create secret-1.yml

New Vault password: xxxx

Confirm Nev Vault password: xxxx

Şimdi create-user-1.yml adında bir playbook oluşturalım:

- name: create a user

  hosts: all

  become: true

  vars\_files:

    - secret-1.yml

  tasks:

    - name: creating user

      ansible.builtin.user:

        name: "{{ username }}"

        password: "{{ pwhash | password\_hash ('sha512') }}"

ansible-playbook --ask-vault-pass create-user-1.yml

ansible all -b -m command -a "grep Oliver /etc/shadow" #bu komutu yazınca artık gözükmez şifre.

node1 | CHANGED | rc=0 >>

tyler:#665fffgkg6&fkg689##2£6466?%^^+&%+:18569:0:99999:7:::

haslenmiş ne oluğu anlaşılmayan şifre

**Dinamik Enventory**

 go to AWS Management Consol and select the ```IAM roles```:

- click the  "create role" then create a role with "AmazonEC2FullAccess" #rol vermemizin nedeni dinamik olduğu için sadece control node değil bütün konsoldaki makinalara erişecek stop eden ve çalışan

- go to EC2 instance Dashboard, and select the control-node instance

- select actions -> security -> modify IAM role

- select the role thay you have jsut created for EC2 full access and save it.

- install "boto3 and botocore"

# sudo yum install pip

pip install --user boto3 botocore

```

**niye dinamik invertory oluştururken boto3 kurmak zorundayız**

ChatGPT

Dinamik bir envanter (inventory) oluştururken AWS (Amazon Web Services) kaynaklarını yönetmek için Boto3 kütüphanesini kullanmanız gerektiği birkaç neden vardır:

AWS ile Etkileşim: Boto3, AWS hizmetleriyle etkileşimde bulunmanıza yardımcı olan bir Python SDK'sıdır. AWS hizmetlerini programatik olarak yönetmek için kullanılır. Bu nedenle, AWS kaynaklarınızı dinamik olarak izlemek ve yönetmek istediğinizde Boto3'ü kullanmanız gerekebilir.

Otomasyon ve Yönetim: AWS hizmetlerini otomatikleştirmek ve AWS kaynaklarınızı dinamik bir şekilde yönetmek için Boto3 çok güçlü bir araçtır. Örneğin, EC2 örneklerini başlatmak, durdurmak, yeniden başlatmak veya diğer kaynakları yönetmek için Boto3 kullanabilirsiniz. Bu, dinamik bir envanter oluşturmanıza olanak tanır.

Güvenilirlik ve Performans: Boto3, AWS hizmetleri ile güvenilir ve performanslı bir şekilde etkileşimde bulunmak için tasarlanmıştır. AWS hizmetlerine doğrudan HTTP istekleri göndermek yerine Boto3 kullanarak işlerinizi daha güvenilir ve optimize edilmiş bir şekilde yapabilirsiniz.

AWS API Entegrasyonu: AWS, birçok hizmet için API'lar sağlar. Boto3, bu API'ları kullanarak AWS hizmetleri ile etkileşim kurmanıza olanak tanır. Bu, AWS kaynaklarınızı programatik olarak izlemeniz ve yönetmeniz için önemlidir.

Sonuç olarak, dinamik bir envanter oluştururken Boto3 kullanmak, AWS hizmetleri ile etkileşimde bulunmanızı, otomasyonu sağlamanızı ve kaynakları dinamik bir şekilde yönetmenizi kolaylaştırır. Bu nedenle, AWS ortamınızı etkili bir şekilde yönetmek ve dinamik bir envanter oluşturmak için Boto3'ü tercih etmek yaygın bir uygulamadır.

- Create another file named ```inventory\_aws\_ec2.yml``` in the project directory.

#Dinamik envantory için inventory\_aws\_ec2 böyle adlandırmak mecburi.

nano inventory\_aws\_ec2.yml #içine aşağıdaki yml kodunu yaz

```

```yml

plugin: aws\_ec2

regions:

  - "us-east-1"

keyed\_groups:                        # grplr

  - key: tags.Name #tage göre bu makinaları gruplandıracak

compose:                             # vrb çk

  ansible\_host: public\_ip\_address

```

#compose boto3 ansible\_host: public\_ip\_address çkm

- see the inventory

```bash

ansible-inventory --graph

ansible-inventory -i inventory\_aws\_ec2.yml --graph

```

1. **plugin: aws\_ec2**: Bu satır, Ansible için kullanılacak envanterin AWS EC2 örneklerini kullanacağını belirtir. Yani bu envanter, AWS'deki EC2 örneklerini otomatik olarak çekmek için AWS API'larını kullanacaktır.
2. **regions**: Bu satır, envanterin hangi AWS bölgesindeki EC2 örneklerini kullanacağını belirtir. Burada, "us-east-1" bölgesi belirtilmiş, yani bu bölgedeki EC2 örnekleri envanterde kullanılacaktır. İhtiyaca göre farklı bölgeler ekleyebilirsiniz.
3. **keyed\_groups**: Bu bölüm, EC2 örneklerini gruplandırmak için kullanılacak bir anahtar belirler. Burada, örneklerin etiketlerine ("tags.Name") göre gruplandırılacağı belirtilmiştir. Yani, örneğin, bir örnek bir etiketle "webserver" olarak etiketlenmişse, bu grup "webserver" adıyla oluşturulacaktır.
4. **compose**: Bu bölüm, Ansible tarafından kullanılacak değişkenleri belirtir. **ansible\_host** değişkeni, her EC2 örneğinin nasıl erişileceğini belirtir. Burada, örneğin her örneğin "public\_ip\_address" adlı bir değişkenle erişilebileceği belirtilmiştir. Bu, her örneğin uzak erişim adresini belirtmek için kullanılır.

Bu envanter dosyasını kullanarak, Ansible ile AWS EC2 örneklerini gruplandırabilir ve bu örnekler üzerinde görevler çalıştırabilirsiniz. Örneklerinizi bu envanter yapısı üzerinden belirli gruplara veya özel değişkenlere dayalı olarak işleyebilirsiniz. Bu, AWS altyapınızdaki otomasyon ve yönetimi kolaylaştırabilir.

```

@all:

  |--@aws\_ec2:

  |  |--ec2-34-201-69-79.compute-1.amazonaws.com

  |  |--ec2-54-234-17-41.compute-1.amazonaws.com

  |--@ungrouped:

```

- Change the inventory's value in ansible.cfg file to inventory.txt. 'inventory=/home/ec2-user/dynamic-inventory/inventory\_aws\_ec2.yml'

- To make sure that all our hosts are reachable with dynamic inventory, we will run various ad-hoc commands that use the ping module.

```bash

ansible all -m ping --key-file "~/<pem file>"

```

- create a playbook name ``user.yml``

```yml

---

- name: create a user using a variable

  hosts: all

  become: true

  vars:

    user: lisa

    ansible\_ssh\_private\_key\_file: "/home/ec2-user/<pem file>"

  tasks:

    - name: create a user {{ user }}

      ansible.builtin.user:

        name: "{{ user }}"

```

- run the playbook

```bash

ansible-playbook user.yml -i inventory\_aws\_ec2.yml

```

```bash

ansible all -a "tail -2 /etc/passwd"

```