
Ansible Dynamic Inventory on AWS

---

1. Introduction to Ansible Inventory

Ansible needs an **inventory** to know *which servers* to manage.

Static Inventory Example

[web]
10.0.0.10
10.0.0.11

Static inventory works for small, fixed environments.
But in cloud environments like AWS, it becomes a problem because:

- New EC2 instances get created dynamically
- IPs change frequently
- Auto-scaling launches and terminates servers
- Tags define roles (web, devops, db)
- Manual updates are not feasible

This is where **Dynamic Inventory** becomes essential.

---

2. What is Dynamic Inventory?

Dynamic Inventory means Ansible **automatically discovers servers** by querying AWS APIs.

Instead of writing IPs manually, Ansible uses:

- AWS tags
- AWS metadata
- AWS regions

Ansible fetches:

- Public IP
- Private IP
- Instance ID
- Tags (Name, Role, Application, etc.)
- State (running/stopped)

This enables **fully automated, scalable, cloud-native deployments**.

---

3. Use-Case Example

You created an EC2 instance with:

- Tag: Name = centos

- Tag: Role = devops

You want Ansible to:

- Automatically detect this instance
- Group it under role_devops and name_centos
- Connect to it over SSH
- Run playbooks on it

All **without manually updating any inventory file**.

---

4. Prerequisites

On Ansible Master (Ubuntu Server):

- Python 3
- Virtual environment support
- boto3 & botocore
- Ansible AWS collection
- AWS CLI credentials
- SSH private key (PEM file)

Your SSH key is:

~/worklab.pem

---

5. Installing Requirements (Ubuntu 24.04 & 22.04)

Ubuntu now blocks global pip installs (PEP 668).
So we use a virtual environment.

Step 1 — Install python venv

sudo apt update
sudo apt install python3.12-venv -y

Step 2 — Create Virtual Environment

python3 -m venv myenv
source myenv/bin/activate

Step 3 — Install boto3 & botocore

pip install boto3 botocore

Step 4 — Install AWS Collection for Ansible

ansible-galaxy collection install amazon.aws

Step 5 — Configure AWS credentials

aws configure

Provide:

- AWS Access Key
- AWS Secret Key
- Default region (e.g., us-east-2)
- Output format json

---

6. Creating the Dynamic Inventory File

Create:

/etc/ansible/aws_ec2.yml

Paste this **correct, production-ready configuration**:

```
plugin: amazon.aws.aws_ec2

regions:
  - us-east-2

filters:
  tag:Name: centos
  tag:Role: devops

hostnames:
  - ip-address       # Correct field for public IP
```

```
keyed_groups:
  - key: tags.Role
    prefix: role
  - key: tags.Name
    prefix: name
```

✔ **Why ip-address?**

Because this is the correct AWS metadata field the plugin uses to fetch public IPs.

---

7. Validate Dynamic Inventory

Run:

```
ansible-inventory -i /etc/ansible/aws_ec2.yml --list
```

You should now see:

```
"hosts": ["18.218.218.228"]
```

This means dynamic inventory is working properly.

---

8. Fixing SSH Authentication

Initially, the error appeared:

```
no such identity: /home/ubuntu/.ssh/worklab.pem
```

Meaning Ansible was searching in the wrong folder.

Your key is located at:

~/worklab.pem

So use:

--private-key ~/worklab.pem

---

9. Determine the Correct SSH Username

Your AMI ID:

ami-028bd5cea26d40561

This AMI = **Ubuntu 22.04 LTS**

Therefore, the correct user is:

ubuntu

Not centos.

---

10. Test SSH Connectivity Using Ansible

Final working ping command:

```
ansible -i /etc/ansible/aws_ec2.yml all -m ping \
  -u ubuntu --private-key ~/worklab.pem
```

Successful output:

```
18.218.218.228 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Now your dynamic inventory is verified.

---

11. Running a Playbook Using Dynamic Inventory

Create a file:

deploy.yml

```
- hosts: all
  become: yes
  tasks:
    - name: Install Git
      apt:
        name: git
        state: present
```

Run it:

```
ansible-playbook -i /etc/ansible/aws_ec2.yml deploy.yml \
  -u ubuntu --private-key ~/worklab.pem
```

This installs git on the AWS instance automatically discovered via dynamic inventory.

---

12. Understanding AWS Dynamic Inventory Internals

When you run Ansible:

1. Plugin connects to AWS using boto3
2. AWS returns all EC2 instances
3. Ansible filters only:
    - Name=centos
    - Role=devops
4. Ansible extracts the IP using:
    - ip-address
5. Groups servers into:
    - role_devops
    - name_centos
6. Runs modules/playbooks on discovered instances
7. No manual host updates required

---

13. Best Practices

✔ Use IAM roles for EC2 Ansible Masters
✔ Never hardcode AWS credentials in playbooks
✔ Secure your PEM file:

chmod 400 ~/worklab.pem

✔ Use tagging strategy:

Environment = dev
Application = web
Role = backend

✔ Use separate dynamic inventories per environment:

- aws_dev.yml
- aws_stage.yml
- aws_prod.yml

✔ Always test with:

ansible-inventory --graph -i aws_ec2.yml

---

14. Common Errors & Fixes

Error	Cause	Fix
public_ip host shown	Wrong hostname field	Use ip-address
Permission denied	Wrong user	Use ubuntu
key not found	Wrong path	Use ~/worklab.pem
PEP 668 pip error	Ubuntu restriction	Use python3 -m venv

Inventory empty	Wrong tags/region	Verify tags in AWS console
unreachable	SG restriction	Allow SSH from Ansible master IP

---

15. Final Summary

You have successfully completed:

✔ Python virtual env setup
✔ boto3 & AWS plugin installation
✔ AWS credential configuration
✔ Dynamic inventory creation
✔ Correct hostname mapping (ip-address)
✔ SSH authentication fix
✔ Successful Ansible ping
✔ Running playbooks with dynamic inventory

Your Ansible environment is now fully cloud-ready and scalable.
