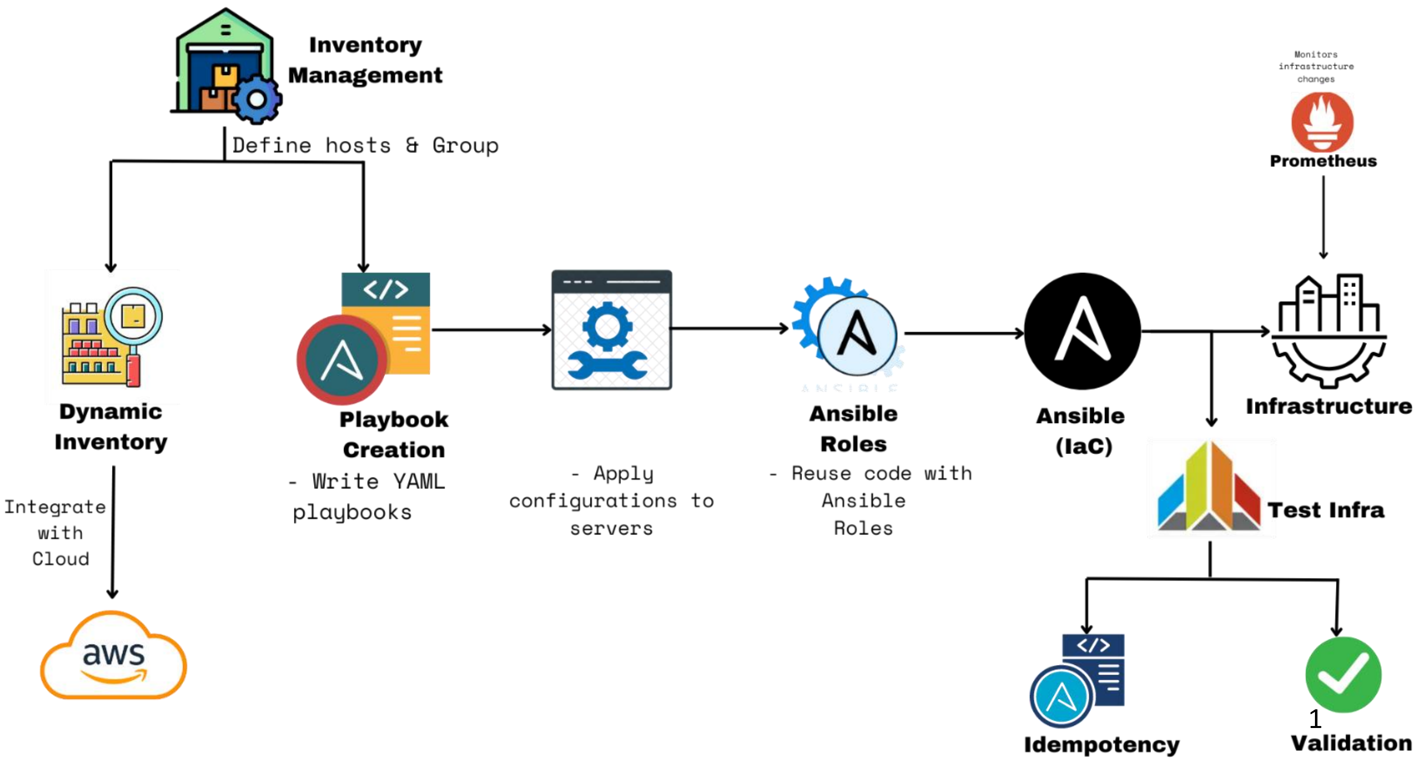
DevOps Shack

Automating Infrastructure Provisioning with Ansible

In today's fast-paced IT environments, manual infrastructure provisioning is no longer feasible due to its time-consuming and error-prone nature. Automation is crucial as it ensures consistency, reduces human errors, and significantly speeds up the deployment process. Automating infrastructure provisioning allows teams to manage large-scale environments efficiently, making it possible to handle complex configurations, scale operations, and ensure that environments are consistent across development, testing, and production.

Ansible is a powerful automation tool that simplifies infrastructure management through its declarative language and modular architecture. By leveraging Ansible, organizations can treat their infrastructure as code, enabling version control, auditing, and reproducibility.



Tools Used

Ansible Inventory: Defines hosts and groups of servers.

Ansible Playbooks: YAML files where tasks, roles, and handlers are defined.

Ansible Modules: Pre-built units of code that perform tasks like installing packages, managing files, or controlling services.

Dynamic Inventory Scripts/Plugins: Scripts or plugins that allow Ansible to interact with cloud providers, gathering information dynamically.

Ansible Roles: Reusable and modular units of playbooks that organize tasks for better management and reuse.

Nagios Monitoring tools that continuously observe the infrastructure for performance and operational issues.

Testinfra: Tools for validating and testing that the infrastructure is provisioned correctly according to the defined state.

Why Automation is Important?

Consistency: Automation ensures that every environment, whether it's development, testing, or production, is configured identically, reducing the likelihood of issues caused by discrepancies.

Speed: Automated provisioning can deploy complex environments in minutes, drastically reducing setup times compared to manual processes.

Scalability: Automation tools can easily scale to manage thousands of servers, which is critical for growing organizations.

Reproducibility: With infrastructure as code, environments can be easily recreated, aiding in disaster recovery and scaling operations.

Detailed Setup

1. Inventory Management

Ansible uses an inventory file to define and group hosts. This can be a static file listing IP addresses or hostnames, or a dynamic inventory script that pulls information from cloud providers.

Setup Example:

```
[web_servers]
web1.example.com
web2.example.com

[db_servers]
db1.example.com
```

2. Playbook Creation

Playbooks are YAML files where you define the tasks that Ansible will execute on the target machines. Each playbook can include multiple tasks, which are executed sequentially.

```
---
- name: Provision Infrastructure on AWS
  hosts: localhost
  gather_facts: no
  tasks:
    - name: Create an EC2 instance
      ec2:
        key_name: my_key
        instance_type: t2.micro
        image: ami-0c55b159cbfafe1f0
        region: us-east-1
        group: my_security_group
        count: 1
        instance_tags:
          Name: Ansible-EC2
        wait: yes
      register: ec2

    - name: Add new EC2 instance to host group
      add_host:
        hostname: "{{ item.public_dns_name }}"
        ansible_ssh_private_key_file: /path/to/your/private-key.pem
        ansible_user: ec2-user
        groups: launched
      loop: "{{ ec2.instances }}"

- name: Configure Web Server on EC2
  hosts: launched
  become: yes
  tasks:
    - name: Install Apache
      yum:
        name: httpd
        state: present

    - name: Start Apache service
      service:
        name: httpd
        state: started
        enabled: yes

    - name: Create index.html
      copy:
        content: "<h1>Hello World from Ansible</h1>"
        dest: /var/www/html/index.html

    - name: Open HTTP port 80 in security group
      ec2_group:
        name: my_security_group
        region: us-east-1
        rules:
          - proto: tcp
            from_port: 80
            to_port: 80
            cidr_ip: 0.0.0.0/0
        state: present

- name: Set Up Monitoring (Prometheus Node Exporter)
  hosts: launched
  become: yes
  tasks:

    - name: Download Prometheus Node Exporter
      get_url:
        url:
https://github.com/prometheus/node_exporter/releases/download/v1.3.1/
node_exporter-1.3.1.linux-amd64.tar.gz
        dest: /tmp/node_exporter.tar.gz

    - name: Extract Node Exporter
      unarchive:
        src: /tmp/node_exporter.tar.gz
        dest: /usr/local/bin/
        remote_src: yes

    - name: Create Node Exporter systemd service
      copy:
        dest: /etc/systemd/system/node_exporter.service
        content: |
          [Unit]
          Description=Prometheus Node Exporter
          After=network.target

          [Service]
          User=root
          ExecStart=/usr/local/bin/node_exporter-1.3.1.linux-
amd64/node_exporter

          [Install]
          WantedBy=default.target
```

```
  - name: Start Node Exporter service
    systemd:
      name: node_exporter
      enabled: yes
      state: started

  - name: Ensure Node Exporter is accessible on port 9100
    ec2_group:
      name: my_security_group
      region: us-east-1
      rules:
        - proto: tcp
          from_port: 9100
          to_port: 9100
          cidr_ip: 0.0.0.0/0
      state: present
```

3. Configuration Management

Ansible modules are used within playbooks to perform tasks. Modules are the building blocks of Ansible, and they ensure idempotency by only making changes when necessary.

Task:

```
- name: Ensure Nginx is running
  service:
    name: nginx
    state: started
```

4. Dynamic Inventory

For environments that change frequently, such as cloud environments, Ansible can use dynamic inventory scripts. These scripts query the cloud provider's API to gather real-time information about the infrastructure.

AWS:

```
ansible-playbook -i aws_ec2.yml playbook.yml
```

5. Role-based Architecture

Roles allow you to group tasks, handlers, variables, and other playbook components into reusable units. This modular approach simplifies complex playbooks.

Directory Structure:

```
roles/
├── common/
│   ├── tasks/
│   ├── handlers/
│   ├── templates/
│   └── files/
```

6. Continuous Monitoring

Continuous monitoring is a crucial aspect of ensuring the health and performance of your infrastructure. This involves setting up tools that can provide real-time monitoring and alerting in case of any issues.

Tool Used:

Prometheus: A popular open-source monitoring tool that can be used to collect and store metrics from various sources, such as servers, applications, and services.

Setup with Prometheus:

To set up Prometheus for continuous monitoring, follow these steps:

Install Prometheus: Install Prometheus on a monitoring server.

Configure Prometheus: Configure Prometheus to scrape metrics from the target servers. This can be done by creating a configuration file that specifies the metrics to collect and the targets to scrape.

Create Alerting Rules: Create alerting rules to notify administrators in case of any issues or anomalies detected by Prometheus.

Integrate with Grafana: Integrate Prometheus with Grafana, a popular open-source visualization tool, to create dashboards and visualizations of the monitored metrics.

Benefits:

Real-time monitoring and alerting

Ability to detect issues and anomalies quickly

Improved visibility into infrastructure performance

Ability to identify trends and patterns in infrastructure performance

7. Infrastructure as Code (IaC)

Infrastructure as Code (IaC) is a best practice in DevOps that involves managing infrastructure resources using code instead of manually configuring them. This approach provides several benefits, including:

Version Control: IaC allows you to manage your infrastructure configurations in a version-controlled manner, which makes it easier to track changes and collaborate with team members.

Repeatability: IaC enables you to recreate your infrastructure environment consistently, which is essential for testing and development.

Automation: IaC allows you to automate the deployment of infrastructure resources, which reduces the risk of human error and increases efficiency.

Tools Used:

Ansible: A popular open-source automation tool that can be used to manage infrastructure resources.

Version Control:

To set up Ansible for IaC, follow these steps:

Store Playbooks in Git Repository: Store your Ansible playbooks in a Git repository.

Create Branching Strategies: Create branching strategies to manage different environments (e.g., dev, staging, prod).

Use Git Hooks: Use Git hooks to automate the deployment of infrastructure resources.

Integrate with CI/CD Tools: Integrate Ansible with CI/CD tools, such as Jenkins or Travis CI, to automate the deployment of infrastructure resources.

8. Idempotency and Validation

Ansible ensures idempotency, meaning that applying the same playbook multiple times will not produce different results. To validate that the infrastructure is in the correct state, tools like Testinfra or Molecule can be used.

Validation Example with **Testinfra**:

```
def test_nginx_is_installed(host):
    nginx = host.package("nginx")
    assert nginx.is_installed
```

Conclusion

By following this guide, you can automate the entire process of infrastructure provisioning using Ansible, ensuring that your infrastructure is consistent, scalable, and reliable. Automation not only saves time but also reduces the risks associated with manual configurations, allowing your team to focus on more strategic tasks.