CONFIGURATION MANAGEMENT WITH ANSIBLE

WHAT IS CONFIGURATION MANAGEMENT?

Configuration management is a way in which a DevOps engineer manages configuration of servers or infrastructure.

THE PROBLEM: MANAGING SERVERS BEFORE CONFIGURATION MANAGEMENT

Scenario:

Role: System Administrator

Company setup: On-premises servers

What is on-premises? On-premises means we have servers in your own data center (not cloud).

Server count: 100 servers

- 50 servers: Linux
- 25 servers: CentOS
- 25 servers: Ubuntu

System Admin Responsibilities:

Three important tasks:

1. Upgrades
2. Security patches
3. Installations

The Challenge:

Doing these tasks on 20-30 or in this case 100 servers is extremely difficult.

Manual approach problems:

- Login to each server individually
- Repeat same tasks 100 times
- Time consuming
- Error prone
- Not scalable

Previous solution: People used to write scripts to loop over these servers and do the tasks. This was very hectic for the system admin.

THE PROBLEM BECOMES BIGGER WITH CLOUD

What changed:

1. Migration to cloud: Entire configuration was moved to cloud

2. Microservice architecture: Applications broken into smaller services

3. Increased server count: Number of servers increased dramatically

4. Reduced compute per server: Resources inside each server reduced

Result: The problem has become even bigger because number of servers has increased.

THE SOLUTION: CONFIGURATION MANAGEMENT TOOLS

What was needed: There was a need for a tool to be invented. Somebody needs to write an application to solve this issue.

Solution developed: A concept was developed called configuration management. It helps in solving the problem of managing multiple servers.

CONFIGURATION MANAGEMENT TOOLS

Famous tools:

- Puppet
- Chef
- Ansible
- Salt

The winner: Ansible

Most number of users and DevOps engineers found Ansible to be better than others.

WHAT IS ANSIBLE?

Developer: Ansible is a tool constantly developed by people at Red Hat.

Full name: Red Hat Ansible

Status: It is the go-to tool for configuration management.

WHY ANSIBLE? (IMPORTANT INTERVIEW QUESTION)

If there are multiple tools like Puppet and Ansible, then why Ansible?

To explain this, we need to know the architecture of both tools.

ANSIBLE VS PUPPET ARCHITECTURE

Key difference:

- Puppet: Pull mechanism model
- Ansible: Push mechanism model

UNDERSTANDING PULL VS PUSH MECHANISM

Scenario:

Role: DevOps engineer managing AWS configuration of an organization

Setup: Created 10 EC2 instances

Responsibility: Manage the configuration of these 10 instances

- Downloading things
- Updating configs
- Installing software
- Security patches

PUPPET (PULL MODEL)

How it works:

Architecture: Master-Slave

Setup required:

1. Create master server (Puppet master)
2. Configure each EC2 as slave:
    - Slave 1
    - Slave 2
    - ...
    - Slave 10
3. Install Puppet agent on each slave
4. Slaves periodically pull configuration from master

Pull mechanism:

- Slaves check master for updates
- Slaves pull configuration
- Slaves apply configuration

Disadvantages:

- Need to set up master server
- Need to install agent on every server
- More infrastructure to manage
- Slaves pull at intervals (not immediate)

ANSIBLE (PUSH MODEL)

How it works:

Architecture: Agentless

Setup:

1. Write Ansible script on laptop
2. Push configuration to 10 EC2 instances

Push mechanism: All you have to do is execute the Ansible playbook and the configuration is updated in a push mechanism and everything is updated.

Advantages:

- No master server needed
- No agent installation needed
- Immediate execution
- Simpler architecture

ADVANTAGE 1: PUSH MODEL

Ansible uses push model:

DevOps engineer workflow:

1. Write Ansible playbook on laptop
2. Execute playbook
3. Configuration pushed to all servers immediately
4. All servers updated

Benefits:

- Control from one location
- Immediate execution
- No delay in updates
- Simpler workflow

ADVANTAGE 2: AGENTLESS MODEL

What is agentless model? Ansible uses agentless model. This means when someone uses Puppet, they have to follow master-slave architecture.

Puppet (Agent-based):

1. Engineer creates master server
2. For all instances, configure each EC2 as slave 1, slave 2, slave 10
3. Install Puppet agent on each slave
4. Once we do this configuration, Puppet does the work

Ansible (Agentless):

1. Put the name of the server or IP address or DNS of servers in the inventory file
2. Enable passwordless authentication
3. That's it - ready to go

What is passwordless authentication? The laptop on which we are running our Ansible playbook should be able to connect to instances without password.

Why is this very important? In this dynamic world, you can anytime scale up or scale down the number of servers.

With Ansible: Because it uses agentless approach, even if number of servers are scaled:

1. Just add the IP address of server to inventory file
2. If we have passwordless authentication enabled
3. We are good to go

No need to:

- Install agents
- Configure master-slave
- Manage agent updates

ADVANTAGE 3: DYNAMIC INVENTORY

What is dynamic inventory? Ansible has come up with a new topic called dynamic inventory.

How it works: With dynamic inventory, Ansible will auto-detect any new server created and manage it.

Benefits:

- Automatic server discovery
- No manual inventory updates
- Adapts to auto-scaling
- Seamless integration with cloud

Example:

Auto-scaling creates 5 new EC2 instances

Dynamic inventory detects them automatically

Ansible manages them without manual configuration

ADVANTAGE 4: EASY TO USE WITH BOTH WINDOWS AND LINUX

Ansible works with:

- Linux servers
- Windows servers
- Unix systems
- Network devices
- Cloud platforms

Cross-platform support: Same Ansible playbook can manage different operating systems.

ADVANTAGE 5: SIMPLE YAML SYNTAX

Ansible is pretty simple.

Puppet: You have to write configuration files in Puppet language (Puppet DSL).

Example Puppet code:

```
node 'webserver' {
 package { 'apache2':
  ensure => installed,
 }
 service { 'apache2':
  ensure => running,
  enable => true,
 }
}
```

Ansible: Simple YAML manifest.

Example Ansible playbook:

```yaml
- name: Install Apache
  hosts: webservers
  tasks:
    - name: Install apache2
      apt:
        name: apache2
        state: present
    - name: Start apache2
      service:
        name: apache2
        state: started
```

Why YAML is better:

- Ansible is using a global language that is known by DevOps engineers already
- YAML is the programming language that most people are familiar with
- Ansible has allowed engineers to write their playbooks in YAML manifest
- In the previous case (Puppet), you had to use Puppet language

---

ADVANTAGE 6: EXTENSIBILITY - CUSTOM MODULES

We can write our own Ansible modules.

How it works: Ansible is basically written in Python.

Use case: If we have some load balancer, we can write some Ansible modules to:

- Install configurations
- Delete configurations
- Remove configurations from load balancer

Sharing modules: These Ansible modules can be shared in the organization with the help of Ansible Galaxy.

What is Ansible Galaxy?

- Repository for Ansible roles and modules
- Community-contributed content
- Reusable automation components
- Easy sharing and distribution

DISADVANTAGES OF ANSIBLE

Despite being advanced, Ansible has some disadvantages:

Disadvantage 1: Windows Support

Issue: Windows users still find it slightly difficult even if you have this advanced Ansible tool for configuration management.

Details:

- Ansible was designed for Linux first
- Windows support added later
- Some modules work differently on Windows
- PowerShell remoting required
- More complex setup for Windows

Disadvantage 2: Debugging

Issue: When we talk about debugging, if there is a mechanism to look into debug logs to check the problem with playbook execution, this is not done well with Ansible.

Problems:

- Error messages can be unclear
- Difficult to trace execution flow
- Limited debugging tools
- Hard to identify exact failure point

Workarounds:

- Use verbose mode (-v, -vv, -vvv)
- Add debug tasks

- Check log files
- Use register and debug modules

Disadvantage 3: Performance Issues

Issue: There is also an issue with the performance of Ansible.

Details: When we do some parallel execution, there might be some performance issues.

Problems:

- Slower than some alternatives
- Sequential execution by default
- Network latency affects performance
- Large inventories can be slow

What Ansible is doing: Ansible is dealing with these problems and continuously improving performance.

COMPARISON TABLE: ANSIBLE VS PUPPET

Aspect: Architecture

- Ansible: Agentless
- Puppet: Agent-based (Master-Slave)

Aspect: Mechanism

- Ansible: Push model
- Puppet: Pull model

Aspect: Setup Complexity

- Ansible: Simple (no master server needed)
- Puppet: Complex (requires master server and agents)

Aspect: Configuration Language

- Ansible: YAML (simple, widely known)
- Puppet: Puppet DSL (proprietary language)

Aspect: Learning Curve

- Ansible: Easy to learn
- Puppet: Steeper learning curve

Aspect: Scalability

- Ansible: Good (with dynamic inventory)
- Puppet: Good (mature product)

Aspect: Windows Support

- Ansible: Limited
- Puppet: Better

Aspect: Execution Speed

- Ansible: Can be slower
- Puppet: Generally faster

Aspect: Debugging

- Ansible: Limited debugging tools
- Puppet: Better debugging capabilities

Aspect: Community

- Ansible: Large, active community
- Puppet: Established community

Aspect: Use Case

- Ansible: Cloud-native, dynamic environments
- Puppet: Enterprise, stable environments

ANSIBLE COMPONENTS

1. Control Node:

- Machine where Ansible is installed
- Where you run Ansible commands
- Your laptop or dedicated server

2. Managed Nodes:

- Servers you manage with Ansible

- Also called hosts
- No Ansible installation needed

3. Inventory:

- List of managed nodes
- Can be static (file) or dynamic (script)
- Groups servers for organization

4. Playbook:

- YAML file with automation tasks
- Defines what to do and where
- Reusable and version-controlled

5. Modules:

- Units of code Ansible executes
- Pre-built for common tasks
- Can write custom modules

6. Tasks:

- Single action to execute
- Uses a module
- Part of a playbook

7. Roles:

- Way to organize playbooks
- Reusable components
- Include tasks, variables, files

ANSIBLE WORKFLOW EXAMPLE

Scenario: Install Nginx on 10 web servers

Step 1: Create inventory file (inventory.ini)

```
[webservers]

web1.example.com
```

```
web2.example.com
web3.example.com
web4.example.com
web5.example.com
web6.example.com
web7.example.com
web8.example.com
web9.example.com
web10.example.com
```

Step 2: Create playbook (install-nginx.yml)

```yaml
---
- name: Install and start Nginx
  hosts: webservers
  become: yes

  tasks:
    - name: Install Nginx
      apt:
        name: nginx
        state: present
        update_cache: yes

    - name: Start Nginx service
      service:
        name: nginx
        state: started
        enabled: yes
    - name: Copy custom config
      copy:
        src: nginx.conf
        dest: /etc/nginx/nginx.conf
      notify: Restart Nginx

  handlers:
    - name: Restart Nginx
      service:
        name: nginx
        state: restarted
```

Step 3: Execute playbook

```
ansible-playbook -i inventory.ini install-nginx.yml
```

Result:

- Ansible connects to all 10 servers
- Installs Nginx on each
- Starts the service
- Copies configuration
- All servers configured identically
- Total time: Few minutes

PASSWORDLESS AUTHENTICATION SETUP

Why needed: Ansible needs to connect to servers without password prompt for automation.

How to set up:

Step 1: Generate SSH key on control node

```
ssh-keygen -t rsa -b 4096
```

Step 2: Copy public key to managed nodes

```
ssh-copy-id user@web1.example.com

ssh-copy-id user@web2.example.com

...
```

Step 3: Test connection

```
ssh user@web1.example.com
```

Should connect without password prompt.

Step 4: Configure Ansible to use key

ansible.cfg:

```
[defaults]

private_key_file = ~/.ssh/id_rsa
```

Now Ansible can connect to all servers automatically.

ANSIBLE USE CASES IN DEVOPS

Use Case 1: Application Deployment

- Deploy code to multiple servers
- Update configuration files
- Restart services
- All servers updated simultaneously

Use Case 2: Configuration Management

- Ensure all servers have same configuration
- Install required packages
- Set up users and permissions
- Configure firewalls

Use Case 3: Infrastructure Provisioning

- Create AWS EC2 instances
- Configure networking
- Set up load balancers
- Provision databases

Use Case 4: Security Hardening

- Apply security patches

- Update SSL certificates

- Configure firewall rules

- Enforce security policies

Use Case 5: Continuous Integration/Deployment

- Integrate with CI/CD pipelines

- Automated testing

- Staged deployments

- Rollback capabilities

ANSIBLE INTERVIEW QUESTIONS

Question 1: What is Ansible? Ansible is an open-source configuration management, deployment, and orchestration tool. It uses agentless architecture and push-based model to manage servers.

Question 2: Why Ansible over Puppet/Chef?

- Agentless (no software installation on managed nodes)
- Push model (immediate execution)
- YAML syntax (easy to learn)
- No master server required
- Dynamic inventory support
- Simpler architecture

Question 3: What is Ansible playbook? Playbook is a YAML file that defines automation tasks. It contains plays (groups of tasks) that execute on specific hosts.

Question 4: What is inventory in Ansible? Inventory is a file that lists managed nodes (servers). It can be static (text file) or dynamic (script that queries cloud providers).

Question 5: What is idempotency in Ansible? Idempotency means running the same playbook multiple times produces the same result. Ansible ensures operations are idempotent - applying configuration multiple times doesn't cause issues.

Question 6: How does Ansible connect to remote servers? Ansible uses SSH (for Linux) or WinRM (for Windows) to connect to remote servers. It requires passwordless authentication using SSH keys.

Question 7: What is the difference between Ansible ad-hoc commands and playbooks?

- Ad-hoc commands: One-time tasks executed from command line
- Playbooks: Reusable, version-controlled YAML files for complex automation

Question 8: What are Ansible roles? Roles are way to organize playbooks into reusable components. They include tasks, variables, files, templates, and handlers organized in a standard directory structure.

Question 9: What is Ansible Galaxy? Ansible Galaxy is a repository for sharing Ansible roles and modules. It's a community hub where users can download and share automation content.

Question 10: How do you handle secrets in Ansible? Use Ansible Vault to encrypt sensitive data like passwords, API keys, and certificates. Vault encrypts files and variables that can be decrypted during playbook execution.

Question 11: What is dynamic inventory? Dynamic inventory is a script or plugin that queries external sources (like AWS, Azure, GCP) to automatically discover and list managed nodes.

Question 12: What are handlers in Ansible? Handlers are special tasks that run only when notified by other tasks. Commonly used to restart services after configuration changes.

Question 13: How do you debug Ansible playbooks? Use verbose mode (-v, -vv, -vvv, -vvvv), debug module, register variable to capture output, and check ansible.log file.

Question 14: What is the difference between copy and template module?

- copy: Copies static files as-is
- template: Uses Jinja2 templating to generate dynamic content before copying

Question 15: How does Ansible handle parallel execution? Ansible can execute tasks on multiple hosts in parallel. The number of parallel processes is controlled by the forks parameter (default is 5).

QUESTION 1: WHICH PROGRAMMING LANGUAGE DOES ANSIBLE USE? CAN YOU WRITE THE MODULES?

Answer:

Programming language: Ansible is written in Python.

Can you write modules? Yes, you can write custom Ansible modules.

How to write modules:

- Modules are written in Python
- Can also be written in any language that returns JSON
- Python is preferred and most common

Why write custom modules?

- When built-in modules don't meet your needs
- To interact with custom applications
- To integrate with proprietary systems
- To extend Ansible functionality

Example use case: If you have a custom load balancer in your organization:

- Write Ansible module to configure it
- Module can install configurations
- Module can delete configurations
- Module can remove configurations from load balancer

Sharing modules:

- Share within organization using Ansible Galaxy
- Contribute to Ansible community
- Create private Galaxy server for company use

Module requirements:

- Must accept JSON input
- Must return JSON output
- Should be idempotent
- Should handle errors properly

QUESTION 2: DOES ANSIBLE SUPPORT ONLY LINUX OR WINDOWS? WHAT PROTOCOLS ARE USED?

Answer:

Ansible supports both Linux and Windows.

Operating systems supported:

- Linux (all distributions)
- Windows
- Unix systems
- macOS
- Network devices
- Cloud platforms

Protocols used:

For Linux: Protocol: SSH (Secure Shell)

How it works:

- Ansible connects via SSH
- Default port: 22
- Requires SSH key or password
- Passwordless authentication recommended

For Windows: Protocol: WinRM (Windows Remote Management)

How it works:

- Windows Remote Management protocol
- Default ports: 5985 (HTTP), 5986 (HTTPS)
- Requires PowerShell
- Uses NTLM or Kerberos authentication

Configuration differences:

Linux configuration:

Inventory file:

```
[linux_servers]

server1.example.com ansible_connection=ssh ansible_user=root
```

Requires:

- SSH access

- SSH key or password

- Python installed on target

Windows configuration:

Inventory file:

```
[windows_servers]

server1.example.com ansible_connection=winrm ansible_user=Administrator ansible_password=password
```

Requires:

- WinRM enabled

- PowerShell 3.0 or higher

- Proper authentication configured

Additional notes:

Windows challenges:

- WinRM setup more complex than SSH
- Requires additional configuration
- Some modules work differently
- PowerShell knowledge helpful

Why SSH for Linux:

- SSH is standard on Linux

- Secure by default
- No additional setup needed
- Widely used protocol

Why WinRM for Windows:

- Windows native protocol
- Integrated with Windows security
- Supports Windows authentication methods
- Required for remote management

QUESTION 3: WHAT IS THE DIFFERENCE BETWEEN PUPPET, ANSIBLE, AND CHEF? WHY ANSIBLE?

Answer:

PUPPET

Architecture:

- Master-Slave architecture
- Agent-based
- Pull mechanism

How it works:

1. Install Puppet master server
2. Install Puppet agent on each node
3. Agents pull configuration from master
4. Agents check master periodically

Language:

- Puppet DSL (Domain Specific Language)
- Proprietary language
- Steeper learning curve

Advantages:

- Mature and stable
- Large enterprise adoption
- Good Windows support
- Strong reporting

Disadvantages:

- Complex setup
- Requires master server
- Requires agent installation
- Pull mechanism (not immediate)

CHEF

Architecture:

- Master-Slave architecture
- Agent-based
- Pull mechanism

How it works:

1. Install Chef server
2. Install Chef client on each node
3. Clients pull configuration from server
4. Clients check server periodically

Language:

- Ruby DSL
- Recipes and cookbooks
- Requires Ruby knowledge

Advantages:

- Powerful and flexible
- Good for complex infrastructure
- Strong version control

Disadvantages:

- Steep learning curve
- Complex setup
- Requires master server
- Ruby knowledge needed

ANSIBLE

Architecture:

- Agentless
- No master-slave
- Push mechanism

How it works:

1. Install Ansible on control node only
2. No installation on managed nodes
3. Push configuration immediately
4. Execute on demand

Language:

- YAML
- Simple and readable
- Easy to learn

Advantages:

- Simple setup
- No master server needed
- No agent installation
- Immediate execution
- Easy to learn

Disadvantages:

- Limited Windows support

- Debugging challenges
- Performance issues at scale

WHY ANSIBLE?

Reason 1: Agentless architecture

- No software installation on managed nodes
- Reduces management overhead
- No agent updates needed
- Simpler infrastructure

Reason 2: Push mechanism

- Immediate execution
- No waiting for nodes to pull
- On-demand configuration
- Real-time updates

Reason 3: Simple YAML syntax

- Easy to learn
- Human-readable
- No proprietary language
- Quick adoption

Reason 4: No master server

- Less infrastructure to manage
- No single point of failure
- Lower costs
- Simpler architecture

Reason 5: Dynamic inventory

- Auto-discovers new servers
- Adapts to auto-scaling
- Cloud-native approach

- No manual updates

Reason 6: Fast setup

- Install Ansible on one machine
- Create inventory file
- Write playbook
- Start automating

Reason 7: Community and ecosystem

- Large active community
- Ansible Galaxy for sharing
- Extensive module library
- Good documentation

COMPARISON TABLE

Aspect: Architecture

- Puppet: Master-Slave
- Chef: Master-Slave
- Ansible: Agentless

Aspect: Agent Required

- Puppet: Yes
- Chef: Yes
- Ansible: No

Aspect: Mechanism

- Puppet: Pull
- Chef: Pull
- Ansible: Push

Aspect: Language

- Puppet: Puppet DSL
- Chef: Ruby DSL

- Ansible: YAML

Aspect: Learning Curve

- Puppet: Medium
- Chef: Steep
- Ansible: Easy

Aspect: Setup Complexity

- Puppet: High
- Chef: High
- Ansible: Low

Aspect: Master Server

- Puppet: Required
- Chef: Required
- Ansible: Not required

Aspect: Windows Support

- Puppet: Good
- Chef: Good
- Ansible: Limited

Aspect: Execution Speed

- Puppet: Fast
- Chef: Fast
- Ansible: Medium

Aspect: Best For

- Puppet: Enterprise, Windows
- Chef: Complex infrastructure
- Ansible: Cloud, DevOps, simplicity

WHY ANSIBLE IS PREFERRED IN DEVOPS:

1. **Cloud-native approach:**

    - Works well with dynamic infrastructure
    - Easy integration with AWS, Azure, GCP
    - Dynamic inventory support

2. **DevOps culture:**
    - Infrastructure as code
    - Version control friendly
    - Easy collaboration

3. **Speed of adoption:**
    - Team can learn quickly
    - Start automating fast
    - Less training required

4. **Modern workflows:**
    - CI/CD integration
    - Container orchestration
    - Microservices deployment

5. **Cost-effective:**
    - No additional infrastructure
    - No license costs (open source)
    - Lower operational overhead

QUESTION 4: IS ANSIBLE PUSH OR PULL MECHANISM?

Answer:

Ansible is a PUSH mechanism.

What is push mechanism?

Definition: The control node (where Ansible is installed) pushes configuration to managed nodes.

How it works:

Step 1: DevOps engineer writes playbook on control node (laptop or server)

Step 2: Engineer executes playbook:

ansible-playbook deploy.yml

Step 3: Ansible immediately connects to all managed nodes

Step 4: Ansible pushes configuration to nodes

Step 5: Configuration applied immediately

Step 6: Ansible returns results

Workflow example:

```
Control Node (Your Laptop)
  |
  | Execute playbook
  |
  +---> Push to Server 1
  +---> Push to Server 2
  +---> Push to Server 3
  +---> Push to Server 4
  +---> Push to Server 5
  |
  | All configured simultaneously
  |
  Results returned
```

Advantages of push mechanism:

1. Immediate execution:

- Run when you want
- No waiting for nodes to check in
- Real-time updates

2. On-demand:

- Execute as needed
- No scheduled intervals
- Full control over timing

3. Predictable:

- Know exactly when changes happen
- See results immediately
- Easy troubleshooting

4. Efficient:

- No periodic checking
- No unnecessary network traffic
- Resources used only when needed

Comparison with pull mechanism:

Pull mechanism (Puppet, Chef):

Managed Nodes periodically check master:

Every 30 minutes:

  - Node 1 checks master

  - Node 2 checks master

  - Node 3 checks master

If configuration changed:

  - Node pulls new configuration

  - Node applies configuration

If no change:

  - Node does nothing

  - Wasted check

Push mechanism (Ansible):

When you execute playbook:

Immediately:

- Ansible connects to all nodes
- Pushes configuration
- Applies changes
- Returns results

When not executing:

- No activity
- No resource usage
- No network traffic

Why push is better for DevOps:

1. **Immediate deployment:**
   - Deploy when code is ready
   - No waiting period
   - Faster release cycles
2. **Better control:**
   - Execute on demand
   - Scheduled deployments
   - Emergency patches
3. **Efficient resource usage:**
   - No continuous polling
   - No wasted checks
   - Pay for execution only
4. **Easier debugging:**
   - See results immediately

    - Identify issues quickly
    - Fix and retry instantly

QUESTION 5: DOES ANSIBLE SUPPORT ALL CLOUD PROVIDERS?

Answer:

YES, Ansible supports all major cloud providers.

Supported cloud providers:

1. Amazon Web Services (AWS)

- EC2 instances
- S3 buckets
- RDS databases
- Lambda functions
- VPC networking
- All AWS services

2. Microsoft Azure

- Virtual Machines
- Storage accounts
- Azure SQL
- App Services
- All Azure services

3. Google Cloud Platform (GCP)

- Compute Engine
- Cloud Storage
- Cloud SQL
- Kubernetes Engine
- All GCP services

4. DigitalOcean

- Droplets
- Spaces
- Databases
- Load Balancers

5. IBM Cloud

6. Oracle Cloud

7. Alibaba Cloud

8. VMware

9. OpenStack

10. Private clouds

What matters for Ansible to work:

Requirements:

1. Public IP address: Managed nodes must have public IP address that Ansible can reach.

2. SSH access (for Linux):

- SSH must be allowed from Ansible control node to managed machines
- Port 22 must be open
- Firewall must allow SSH connections

3. WinRM access (for Windows):

- WinRM must be enabled
- Ports 5985/5986 must be open
- Firewall must allow WinRM connections

4. Network connectivity:

- Control node must be able to reach managed nodes
- No network restrictions between them
- Proper routing configured

5. Authentication:

- SSH key for Linux
- Username/password for Windows
- Proper credentials configured

How Ansible connects to cloud servers:

Example: AWS EC2 instances

Step 1: Create EC2 instances

Ansible can create EC2 instances:

- Use ec2 module

- Define instance type, AMI, security group

- Ansible provisions instances

Step 2: Configure inventory

Add public IP addresses to inventory:

```
[aws_servers]

54.123.45.67

54.123.45.68

54.123.45.69
```

Step 3: Ensure SSH access

Security group must allow:

- Inbound SSH (port 22)

- From Ansible control node IP

Step 4: Run playbook

```
ansible-playbook -i inventory deploy.yml
```

Ansible connects via:

- Public IP address
- SSH protocol
- SSH key authentication
- Manages EC2 instances

Cloud-specific considerations:

AWS:

- Use AWS access keys
- Dynamic inventory from EC2 API
- Auto-discover instances by tags
- Manage AWS services via modules

Azure:

- Use Azure credentials
- Dynamic inventory from Azure API
- Auto-discover VMs
- Manage Azure services

GCP:

- Use GCP credentials
- Dynamic inventory from GCP API
- Auto-discover instances
- Manage GCP services

Dynamic inventory with cloud providers:

What is dynamic inventory? Instead of manually listing IP addresses, Ansible queries cloud provider API to automatically discover instances.

Example: AWS dynamic inventory

Install AWS CLI and configure credentials

Create dynamic inventory script:

```
#!/usr/bin/env python

# Queries AWS API

# Returns list of EC2 instances
```

Groups by tags, regions, etc.

Run playbook:

ansible-playbook -i aws_ec2.yml deploy.yml

Ansible automatically finds all EC2 instances

No manual IP address management needed

Benefits:

- Auto-discovery of new instances
- Works with auto-scaling
- Always up-to-date inventory
- No manual updates

Network requirements summary:

For Ansible to work with ANY cloud provider:

1. Connectivity: Control node must reach managed nodes over network

2. Protocol access:

- Linux: SSH allowed (port 22)
- Windows: WinRM allowed (ports 5985/5986)

3. IP addressing:

- Public IP for internet-based control
- Private IP for VPN/VPC connectivity
- DNS names also work

4. Firewall rules:

- Allow inbound SSH/WinRM
- From Ansible control node IP
- Security groups configured properly

5. Authentication:

- SSH keys for Linux
- Credentials for Windows
- Proper permissions

If these conditions are met, Ansible works with ANY cloud provider.

Example: Multi-cloud setup

Scenario: Managing servers across AWS, Azure, and GCP

Inventory file:

```
[aws_servers]
aws1.example.com ansible_host=54.123.45.67
aws2.example.com ansible_host=54.123.45.68

[azure_servers]
azure1.example.com ansible_host=20.30.40.50
azure2.example.com ansible_host=20.30.40.51

[gcp_servers]
gcp1.example.com ansible_host=35.40.50.60
gcp2.example.com ansible_host=35.40.50.61

[all_servers:children]
aws_servers
azure_servers
gcp_servers
```

Single playbook manages all clouds:

```
---
- name: Configure all cloud servers
  hosts: all_servers
```

```
 tasks:
  - name: Install common packages
    package:
      name: nginx
      state: present
```

Result: One Ansible playbook manages servers across AWS, Azure, and GCP simultaneously.