Ansible Cheatsheet

Essential operations for infrastructure automation and configuration management

This cheatsheet provides a quick reference to fundamental Ansible operations, syntax, and advanced features, ideal for both beginners and experienced DevOps engineers for efficient infrastructure automation.

Installation & Setup
Install and configure Ansible

Inventory Management
Define and organize target hosts

Playbooks & Tasks
Create automation workflows

Modules & Commands
Execute specific operations

Variables & Templates
Manage dynamic configurations

Installation & Setup

Ubuntu/Debian: `apt install ansible`

Install Ansible on Debian-based Linux systems.

```
# Add Ansible repository
sudo apt-add-repository ppa:ansible/ansible
# Update package lists
sudo apt-get update
# Install Ansible
sudo apt-get install ansible
# Verify installation
ansible --version
```

CentOS/RHEL: `yum install ansible`

Install Ansible on Red Hat-based systems.

```
# Install EPEL repository
sudo yum install epel-release -y
# Install Ansible
sudo yum install ansible -y
# Verify installation
ansible --version
```

macOS: `brew install ansible`

Install Ansible on macOS using Homebrew.

```
# Install using Homebrew
brew install ansible
# Verify installation
ansible --version
```

Configuration: `/etc/ansible/ansible.cfg`

Configure Ansible settings and defaults.

```
# View current configuration
ansible-config list
# View effective configuration
ansible-config view
# Custom configuration file
export ANSIBLE_CONFIG=/path/to/ansible.cfg
```

SSH Setup: Key-based Authentication

Ansible uses SSH to communicate between nodes.

```
# Generate SSH key
ssh-keygen -t rsa -b 4096
# Copy public key to remote hosts
ssh-copy-id user@hostname
# Test SSH connection
ssh user@hostname
```

Environment Setup

Set up Ansible environment variables and paths.

```
# Set inventory file location
export ANSIBLE_INVENTORY=/path/to/inventory
# Set host key checking
export ANSIBLE_HOST_KEY_CHECKING=False
# Set remote user
export ANSIBLE_REMOTE_USER=ubuntu
```

Inventory Management

The inventory file lists all platforms you want to automate across.

01

Basic Inventory: `/etc/ansible/hosts`

Host groups can be created by giving a group name within square brackets.

```
# Basic hosts file (INI format)
[webservers]
web1.example.com
web2.example.com

[databases]
db1.example.com
db2.example.com

[all:vars]
ansible_user=ubuntu
ansible_ssh_private_key_file=~/.ssh/id_rsa
```

02

YAML Inventory Format

Inventory files can be in INI or YAML format.

```
# inventory.yml
all:
  children:
    webservers:
      hosts:
        web1.example.com:
        web2.example.com:
    databases:
      hosts:
        db1.example.com:
      vars:
        mysql_port: 3306
```

03

Host Variables & Groups

Define host-specific variables and group configurations.

```
# Inventory with variables
[webservers]
web1.example.com http_port=80
web2.example.com http_port=8080

[webservers:vars]
ansible_user=nginx
nginx_version=1.18

# Test inventory
ansible-inventory --list
ansible-inventory --graph
```

Ad-Hoc Commands

Basic Command Structure

Basic structure of an Ansible command: ansible -m -a ""

```
# Test connectivity
ansible all -m ping
# Check specific group
ansible webservers -m ping
# Run command on all hosts
ansible all -m command -a "uptime"
# Run with sudo privileges
ansible all -m command -a "systemctl status nginx" --become
```

File Operations

Create directories, files, and symbolic links on hosts.

```
# Create directory
ansible all -m file -a "path=/tmp/test state=directory mode=0755"
# Create file
ansible all -m file -a "path=/tmp/test.txt state=touch"
# Delete file/directory
ansible all -m file -a "path=/tmp/test state=absent"
# Create symbolic link
ansible all -m file -a "src=/etc/nginx dest=/tmp/nginx state=link"
```

Package Management

Install, update, and remove packages across different systems.

```
# Install package (apt)
ansible webservers -m apt -a "name=nginx state=present" --become
# Install package (yum)
ansible webservers -m yum -a "name=httpd state=present" --become
# Update all packages
ansible all -m apt -a "upgrade=dist" --become
# Remove package
ansible all -m apt -a "name=apache2 state=absent" --become
```

Service Management

Start, stop, and manage system services.

```
# Start service
ansible webservers -m service -a "name=nginx state=started" --become
# Stop service
ansible webservers -m service -a "name=apache2 state=stopped" --become
# Restart service
ansible webservers -m service -a "name=ssh state=restarted" --become
# Enable service at boot
ansible all -m service -a "name=nginx enabled=yes" --become
```

Playbooks & Tasks

Basic Playbook Structure

YAML files that define which tasks should be run and on which hosts.

```
---
- name: Web server setup
  hosts: webservers
  become: yes
  vars:
    nginx_port: 80

  tasks:
    - name: Install nginx
      apt:
        name: nginx
        state: present

    - name: Start nginx service
      service:
        name: nginx
        state: started
        enabled: yes
```

Running Playbooks

Run playbooks with various options and configurations.

```
# Run playbook
ansible-playbook site.yml
# Run with specific inventory
ansible-playbook -i inventory.yml site.yml
# Dry run (check mode)
ansible-playbook site.yml --check
# Run on specific hosts
ansible-playbook site.yml --limit webservers
# Run with extra variables
ansible-playbook site.yml --extra-vars "nginx_port=8080"
```

Task Options & Conditionals

Add conditions, loops, and error handling to tasks.

```
tasks:
  - name: Install packages
    apt:
      name: "{{ item }}"
      state: present
    loop:
      - nginx
      - mysql-server
      - php
    when: ansible_os_family == "Debian"

  - name: Create user
    user:
      name: webuser
      state: present
    register: user_result

  - name: Show user creation result
    debug:
      msg: "User created: {{ user_result.changed }}"
```

Handlers & Notifications

Define handlers that run when notified by tasks.

```
tasks:
  - name: Update nginx config
    template:
      src: nginx.conf.j2
      dest: /etc/nginx/nginx.conf
    notify: restart nginx

handlers:
  - name: restart nginx
    service:
      name: nginx
      state: restarted
```

Variables & Templates

Manage dynamic configurations and data across your infrastructure.

Variable Definition

Define variables at different levels and scopes.

```
# In playbook
vars:
  app_name: myapp
  app_port: 8080

# In group_vars/all.yml
database_host: db.example.com
database_port: 5432

# In host_vars/web1.yml
server_role: frontend
max_connections: 100

# Command line variables
ansible-playbook site.yml -e "env=production"
```

Jinja2 Templates

Create dynamic configuration files using templates.

```
# Template file: nginx.conf.j2
server {
    listen {{ nginx_port }};
    server_name {{ server_name }};

    location / {
        proxy_pass http://{{ backend_host }}:{{ backend_port }};
    }
}

# Using template module
- name: Deploy nginx config
  template:
    src: nginx.conf.j2
    dest: /etc/nginx/sites-available/default
  notify: reload nginx
```

Facts & System Information

Gather and use system facts in playbooks.

```
# Gather facts manually
ansible all -m setup
# Gather specific facts
ansible all -m setup -a "filter=ansible_eth*"

# Use facts in playbooks
- name: Show system info
  debug:
    msg: "{{ ansible_hostname }} runs {{ ansible_distribution }}"

- name: Install package based on OS
  apt:
    name: apache2
  when: ansible_os_family == "Debian"
```

Vault & Secrets Management

Encrypt sensitive data using Ansible Vault.

```
# Create encrypted file
ansible-vault create secrets.yml
# Edit encrypted file
ansible-vault edit secrets.yml
# Encrypt existing file
ansible-vault encrypt passwords.yml
# Run playbook with vault
ansible-playbook site.yml --ask-vault-pass
# Use vault password file
ansible-playbook site.yml --vault-password-file .vault_pass
```

Roles & Organization

Structure and reuse Ansible code with roles and collections.

Role Structure

Organize playbooks into reusable roles.

```
# Create role structure
ansible-galaxy init webserver

# Role directory structure
webserver/
 ├── tasks/
 │   └── main.yml
 ├── handlers/
 │   └── main.yml
 ├── templates/
 ├── files/
 ├── vars/
 │   └── main.yml
 ├── defaults/
 │   └── main.yml
 └── meta/
     └── main.yml
```

Using Roles in Playbooks

Apply roles to hosts in your playbooks.

```
---
- hosts: webservers
  roles:
    - common
    - webserver
    - { role: database, database_type: mysql }

# Or with include_role
- hosts: webservers
  tasks:
    - include_role:
        name: webserver
      vars:
        nginx_port: 8080
```

Ansible Galaxy

Download and manage community roles from Ansible Galaxy.

```
# Install role from Galaxy
ansible-galaxy install geerlingguy.nginx
# Install specific version
ansible-galaxy install geerlingguy.nginx,2.8.0
# Install from requirements file
ansible-galaxy install -r requirements.yml
# List installed roles
ansible-galaxy list
# Remove role
ansible-galaxy remove geerlingguy.nginx
```

Collections

Work with Ansible Collections for extended functionality.

```
# Install collection
ansible-galaxy collection install community.general
# Use collection in playbook
collections:
  - community.general

tasks:
  - name: Install package
    community.general.snap:
      name: code
      state: present
```

Debugging & Troubleshooting

Debugging Tasks

Debug and troubleshoot playbook execution.

```
# Add debug tasks
- name: Show variable value
  debug:
    var: my_variable

- name: Show custom message
  debug:
    msg: "Server {{ inventory_hostname }} has IP {{ ansible_default_ipv4.address }}"

# Verbose execution
ansible-playbook site.yml -v
ansible-playbook site.yml -vvv  # Maximum verbosity
```

Error Handling

Handle errors and failures gracefully.

```
- name: Task that might fail
  command: /bin/false
  ignore_errors: yes

- name: Task with rescue
  block:
    - command: /bin/false
  rescue:
    - debug:
        msg: "Task failed, running rescue"
  always:
    - debug:
        msg: "This always runs"
```

Testing & Validation

Test playbooks and validate configurations.

```
# Check syntax
ansible-playbook site.yml --syntax-check
# List tasks
ansible-playbook site.yml --list-tasks
# List hosts
ansible-playbook site.yml --list-hosts
# Step through playbook
ansible-playbook site.yml --step

# Test with check mode
ansible-playbook site.yml --check --diff
```

Performance & Optimization

Optimize playbook performance and execution.

```
# Run tasks in parallel
- name: Install packages
  apt:
    name: "{{ packages }}"
  vars:
    packages:
    - nginx
    - mysql-server

# Use async for long-running tasks
- name: Long running task
  command: /usr/bin/long-task
  async: 300
  poll: 5
```

Best Practices & Tips

Follow these practices for maintainable and secure Ansible automation.

Security Best Practices

Secure your Ansible infrastructure and operations.

```
# Use Ansible Vault for secrets
ansible-vault create group_vars/all/vault.yml
# Disable host key checking cautiously
host_key_checking = False
# Use become only when necessary
become: yes
become_user: root
# Limit playbook scope
ansible-playbook site.yml --limit production
```

Code Organization

Structure your Ansible projects effectively.

```
# Recommended directory structure
ansible-project/
 ├── inventories/
 │   ├── production/
 │   └── staging/
 ├── group_vars/
 ├── host_vars/
 ├── roles/
 ├── playbooks/
 └── ansible.cfg

# Use meaningful names and documentation
- name: Descriptive task name
  # Add comments for complex logic
```

Version Control & Testing

Manage Ansible code with proper version control.

```
# Use Git for version control
git init
git add .
git commit -m "Initial Ansible setup"

# Test in staging before production
ansible-playbook -i staging site.yml
# Use tags for selective execution
ansible-playbook site.yml --tags "nginx,ssl"
```

Configuration & Advanced Features

Ansible provides idempotent execution ensuring re-running tasks results in consistent system state.

Ansible Configuration

Customize Ansible behavior with configuration options.

```
# ansible.cfg
[defaults]
inventory = ./inventory
remote_user = ansible
host_key_checking = False
timeout = 30
forks = 5

[ssh_connection]
ssh_args = -o ControlMaster=auto -o ControlPersist=60s
pipelining = True
```

Callback Plugins

Enhance output and logging with callback plugins.

```
# Enable callback plugins in ansible.cfg
[defaults]
stdout_callback = yaml
callbacks_enabled = profile_tasks, timer

# Custom callback configuration
[callback_profile_tasks]
task_output_limit = 20
```

Filters & Lookups

Use Jinja2 filters and lookup plugins for data manipulation.

```
# Common filters in templates
{{ variable | default('default_value') }}
{{ list_var | length }}
{{ string_var | upper }}
{{ dict_var | to_nice_yaml }}

# Lookup plugins
- name: Read file content
  debug:
    msg: "{{ lookup('file', '/etc/hostname') }}"

- name: Environment variable
  debug:
    msg: "{{ lookup('env', 'HOME') }}"
```

Dynamic Inventories

Use dynamic inventories for cloud and container environments.

```
# AWS EC2 dynamic inventory
ansible-playbook -i ec2.py site.yml
# Docker dynamic inventory
ansible-playbook -i docker.yml site.yml
# Custom inventory script
ansible-playbook -i ./dynamic_inventory.py site.yml
```

Reference: This cheatsheet covers essential Ansible commands and modern practices for efficient infrastructure automation and configuration management in DevOps workflows.