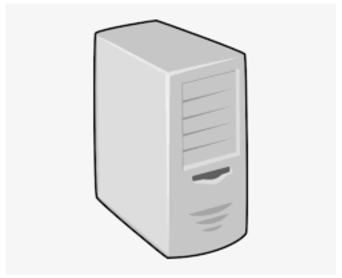
Configuration Management

Question:

Describe the architecture and communication flow in an Ansible setup?

Configuration Management

Ansible Architecture



Control node

Inventory

Modules

Playbooks

Roles

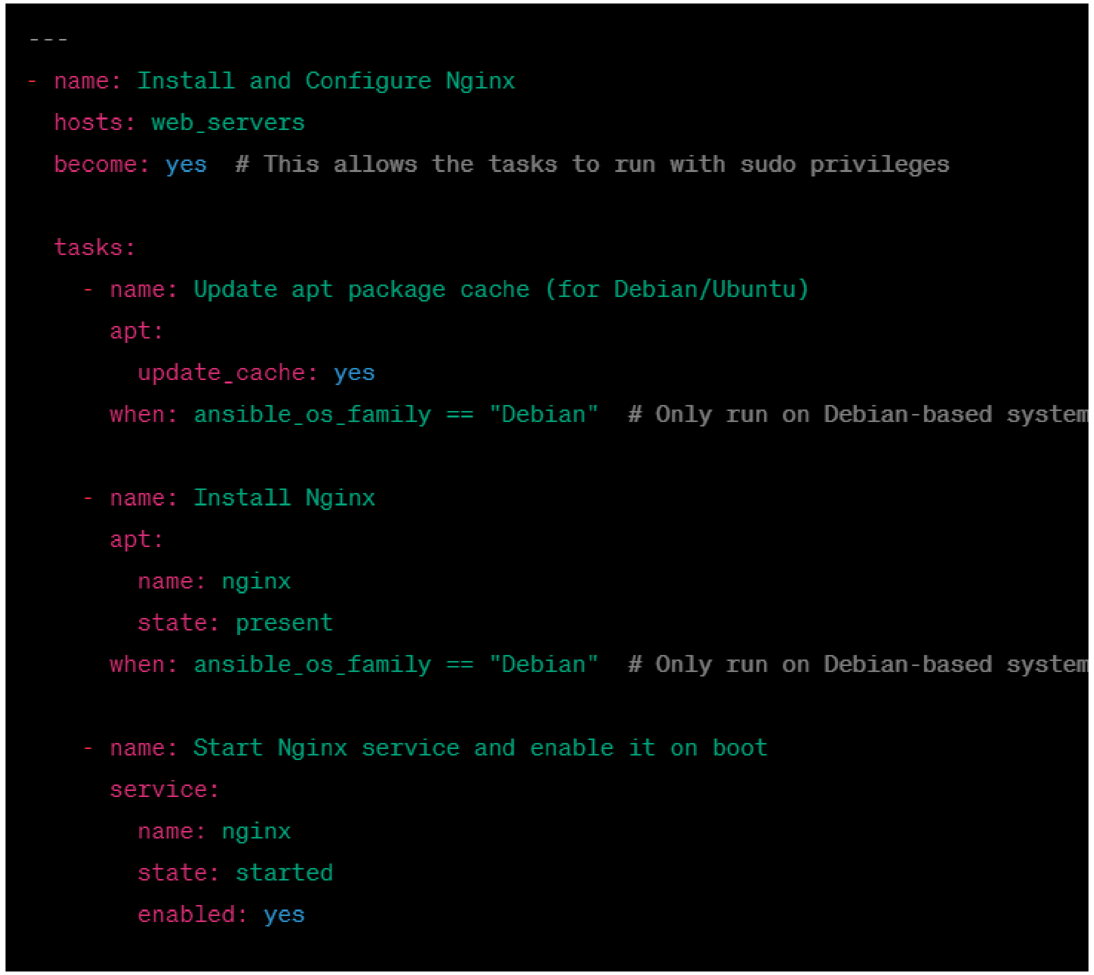
Configuration Management

Question:

How are ansible playbooks used in defining infrastructure configurations and automation tasks?

Configuration Management

- Task Definition
- Host and Group Specs
- Variables Usage
- Task Execution Control
- Roles and Reusability
- Handler Invocation
- Playbook Execution



```yaml
---
- name: Install and Configure Nginx
  hosts: web_servers
  become: yes  # This allows the tasks to run with sudo privileges

  tasks:
    - name: Update apt package cache (for Debian/Ubuntu)
      apt:
        update_cache: yes
      when: ansible_os_family == "Debian"  # Only run on Debian-based system

    - name: Install Nginx
      apt:
        name: nginx
        state: present
      when: ansible_os_family == "Debian"  # Only run on Debian-based system

    - name: Start Nginx service and enable it on boot
      service:
        name: nginx
        state: started
        enabled: yes
```

Configuration Management

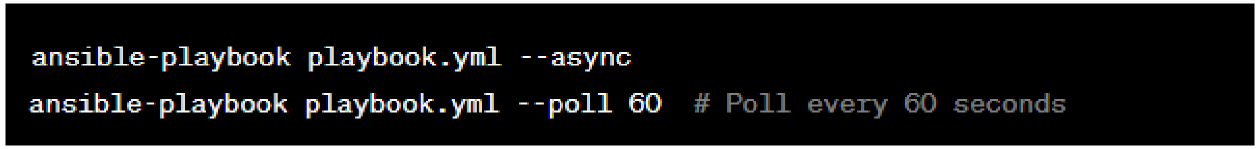
Question:

How to optimize ansible performance and reduce execution time?

Configuration Management

Asynchronous mode

This mode allows you to run tasks in the background



```
ansible-playbook playbook.yml --async
ansible-playbook playbook.yml --poll 60  # Poll every 60 seconds
```

Configuration Management

Forks

Parallel processing

```
ansible-playbook -i inventory.ini playbook.yml --forks=10
```

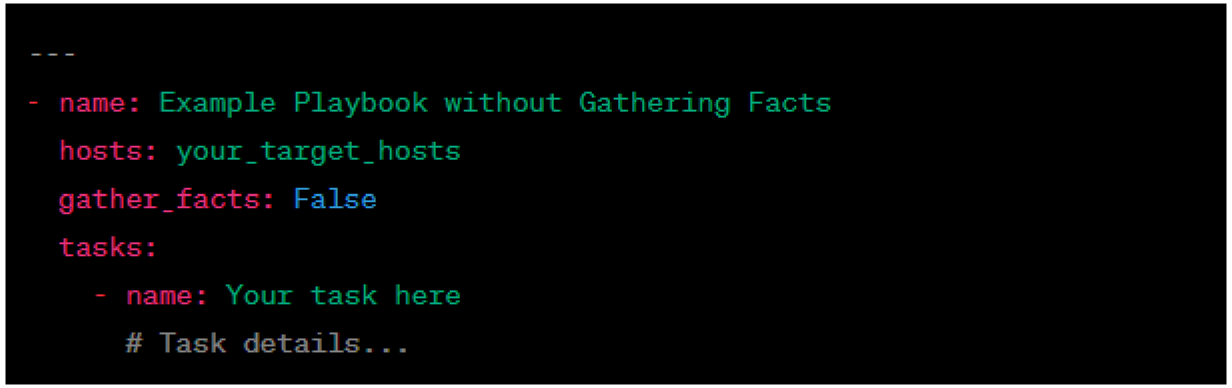
- Resource Usage
- Network Impact
- Target Environment

Configuration Management

Skip gathering facts

Avoid unwanted information collection

- Speed
- Resource Usage
- Minimize network traffic



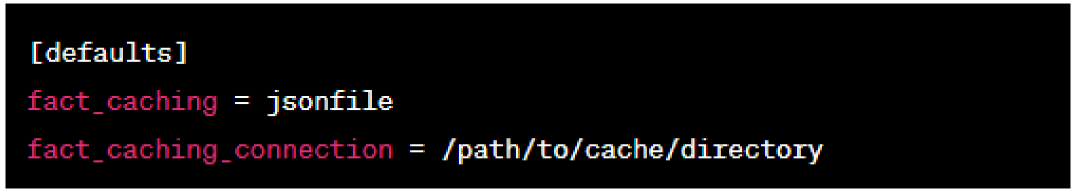
```yaml
---
- name: Example Playbook without Gathering Facts
  hosts: your_target_hosts
  gather_facts: False
  tasks:
    - name: Your task here
      # Task details...
```

Configuration Management

Caching facts

Store previously gathered facts

- Performance improvement
- Reduced load
- Optimized for Idempotence
- Customization



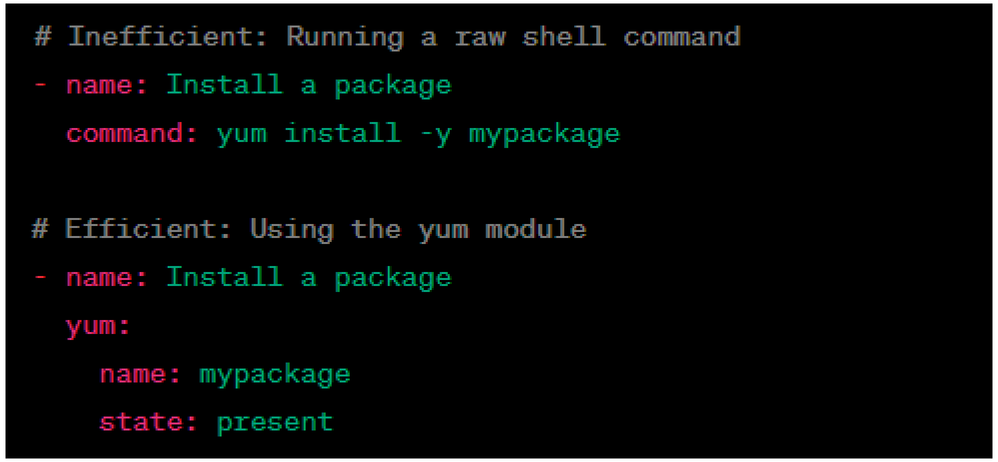
```
[defaults]
fact_caching = jsonfile
fact_caching_connection = /path/to/cache/directory
```

Configuration Management

Use Efficient Modules

Avoid shell commands and use modules instead

- Idempotence



```yaml
# Inefficient: Running a raw shell command
- name: Install a package
  command: yum install -y mypackage

# Efficient: Using the yum module
- name: Install a package
  yum:
    name: mypackage
    state: present
```

Configuration Management

Question:

What is Idempotence and why is it an important feature?

Configuration Management

Idempotence

Applying same configuration multiple times produces the same result as applying it once.

Configuration Management

- Predictable Behavior
- Corrective and Preventive Actions
- Efficiency
- Safety
- Consistency
- Simplified Maintenance

Configuration Management

Question:

How do you handle secrets and sensitive data in Ansible, such as password or API Keys?

Configuration Management

Ansible Vault

Sooraj Mohammed

Configuration Management

Ansible Vault

Ansible Vault allows you to encrypt sensitive information, such as passwords, API keys, or any other confidential data, within your playbooks or variable files.

Configuration Management

```
ansible-vault encrypt secrets.yml

ansible-vault edit secrets.yml

ansible-playbook --ask-vault-pass playbook.yml

ansible-playbook --vault-password-file=vault_pass.txt
playbook.yml
```

Configuration Management

Question:

You have a web application running on a cluster of servers. Explain how you would implement a rolling update strategy using Ansible to minimize downtime during updates?

Configuration Management

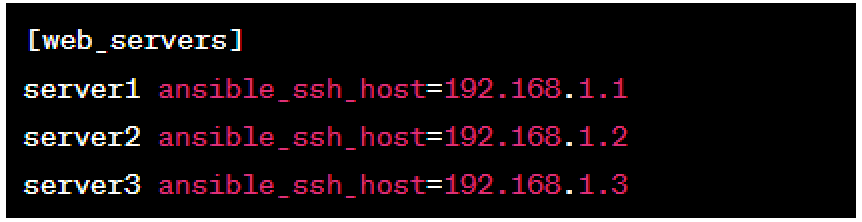
Rolling Update Strategy

A rolling update strategy is a deployment technique used to update a system with minimal disruption to its availability.

Configuration Management

Rolling Update Strategy

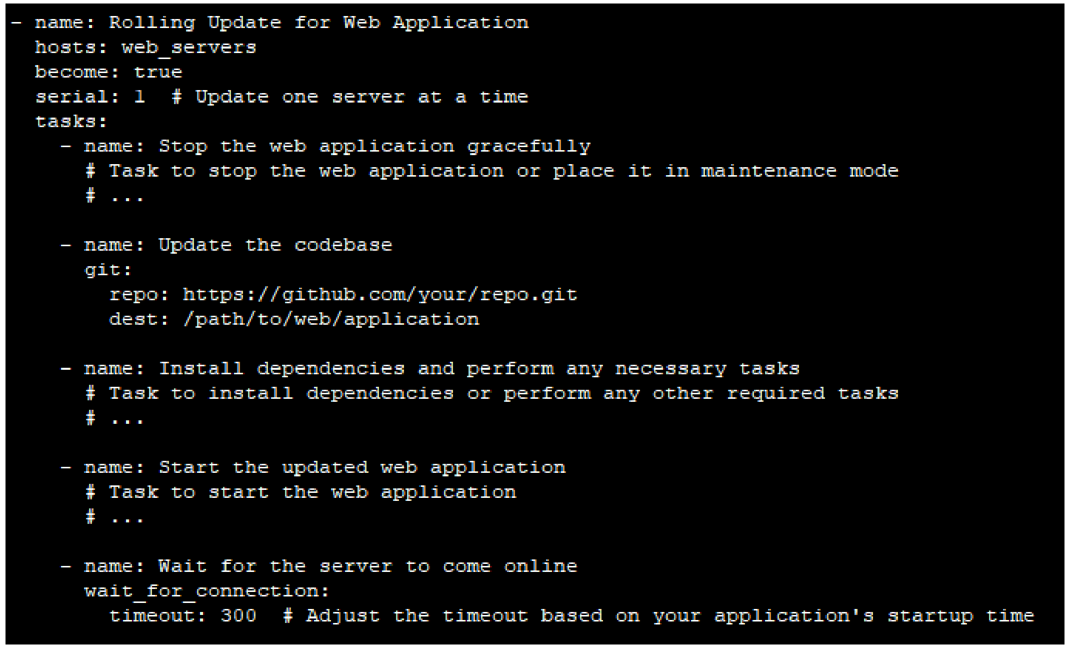
Inventory Setup



```
[web_servers]
server1 ansible_ssh_host=192.168.1.1
server2 ansible_ssh_host=192.168.1.2
server3 ansible_ssh_host=192.168.1.3
```

Configuration Management

Rolling Update Strategy

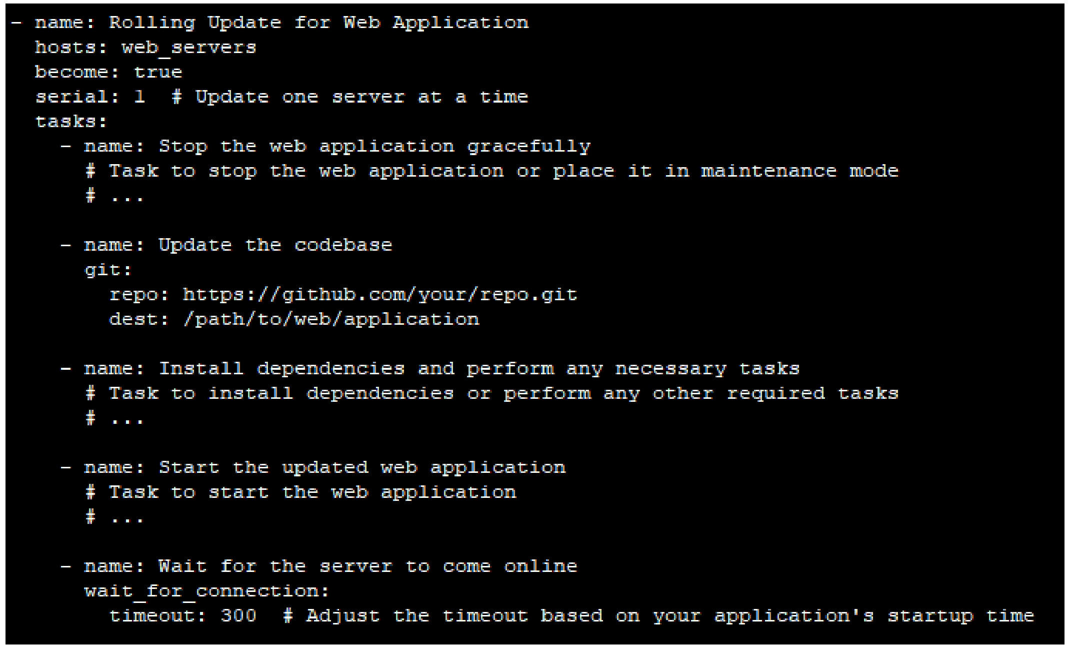


```
- name: Rolling Update for Web Application
  hosts: web_servers
  become: true
  serial: 1  # Update one server at a time
  tasks:
    - name: Stop the web application gracefully
      # Task to stop the web application or place it in maintenance mode
      # ...

    - name: Update the codebase
      git:
        repo: https://github.com/your/repo.git
        dest: /path/to/web/application

    - name: Install dependencies and perform any necessary tasks
      # Task to install dependencies or perform any other required tasks
      # ...

    - name: Start the updated web application
      # Task to start the web application
      # ...

    - name: Wait for the server to come online
      wait_for_connection:
        timeout: 300  # Adjust the timeout based on your application's startup time
```

Serial Execution

Configuration Management

Rolling Update Strategy

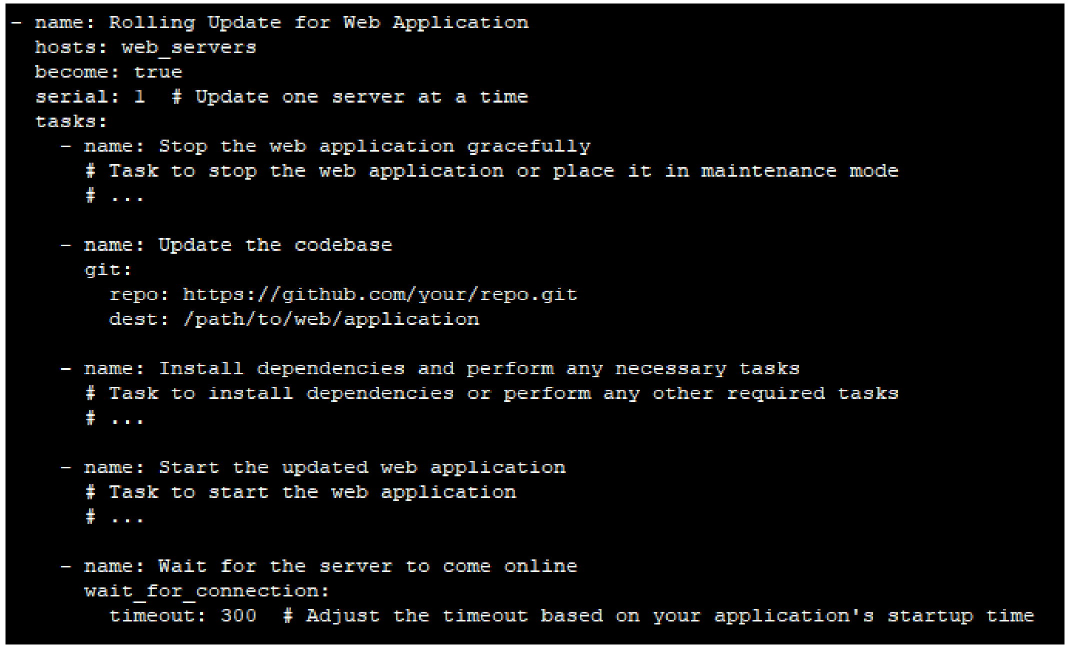


```
- name: Rolling Update for Web Application
  hosts: web_servers
  become: true
  serial: 1  # Update one server at a time
  tasks:
    - name: Stop the web application gracefully
      # Task to stop the web application or place it in maintenance mode
      # ...

    - name: Update the codebase
      git:
        repo: https://github.com/your/repo.git
        dest: /path/to/web/application

    - name: Install dependencies and perform any necessary tasks
      # Task to install dependencies or perform any other required tasks
      # ...

    - name: Start the updated web application
      # Task to start the web application
      # ...

    - name: Wait for the server to come online
      wait_for_connection:
        timeout: 300  # Adjust the timeout based on your application's startup time
```

Graceful Stop/Start

Configuration Management

Rolling Update Strategy

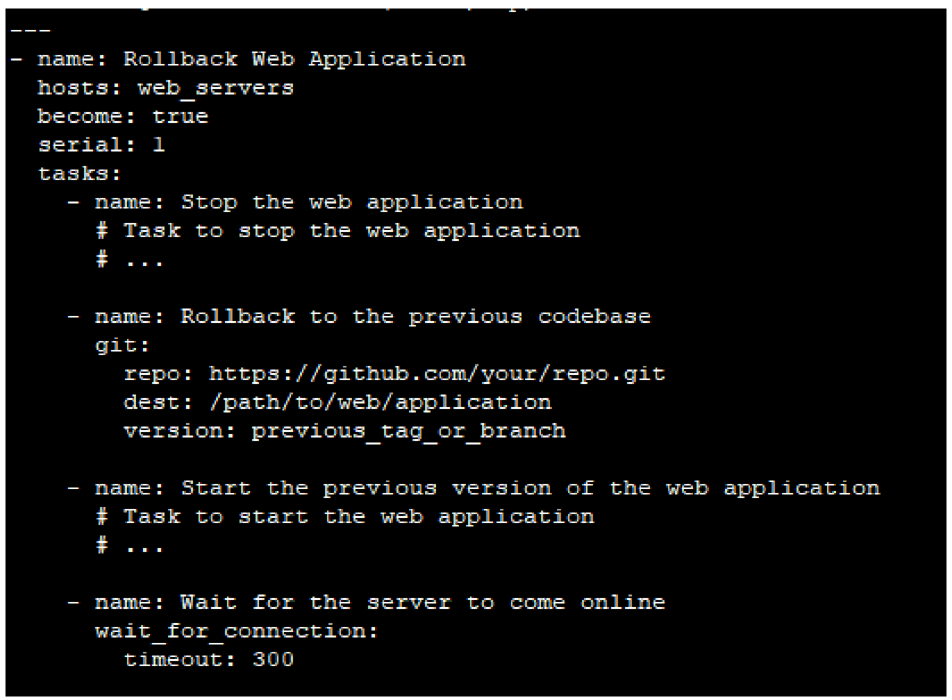


```
- name: Rolling Update for Web Application
  hosts: web_servers
  become: true
  serial: 1  # Update one server at a time
  tasks:
    - name: Stop the web application gracefully
      # Task to stop the web application or place it in maintenance mode
      # ...

    - name: Update the codebase
      git:
        repo: https://github.com/your/repo.git
        dest: /path/to/web/application

    - name: Install dependencies and perform any necessary tasks
      # Task to install dependencies or perform any other required tasks
      # ...

    - name: Start the updated web application
      # Task to start the web application
      # ...

    - name: Wait for the server to come online
      wait_for_connection:
        timeout: 300  # Adjust the timeout based on your application's startup time
```

wait_for_connection

Configuration Management

Rolling Update Strategy



```
---
- name: Rollback Web Application
  hosts: web_servers
  become: true
  serial: 1
  tasks:
    - name: Stop the web application
      # Task to stop the web application
      # ...

    - name: Rollback to the previous codebase
      git:
        repo: https://github.com/your/repo.git
        dest: /path/to/web/application
        version: previous_tag_or_branch

    - name: Start the previous version of the web application
      # Task to start the web application
      # ...

    - name: Wait for the server to come online
      wait_for_connection:
        timeout: 300
```

Rolling back playbook

Configuration Management

Question:

You have a dynamic environment with servers being added and removed frequently. How would you set up and use dynamic inventory in Ansible to ensure it always includes the latest server information?

Configuration Management

Automatically discover and manage inventory

Pull information from cloud providers, configuration management databases, or custom scripts

Configuration Management

Cloud Providers

aws

Azure

Google Cloud

Configuration Management

Custom Script

ansible.cfg

```
[defaults]
inventory = /path/to/your/custom_inventory.py
```

Configuration Management

Question:

In the event of a system failure, how would you use Ansible to automate the recovery process, including restoring data and configurations to a predefined state?

Configuration Management

Recovery process involves restoring not only the application code but also data, configurations, and dependencies

Configuration Management

Recovery Steps

Define the Predefined State

Write Ansible Playbooks

Backup Strategies

Error Handling

Utilize Roles and Variables

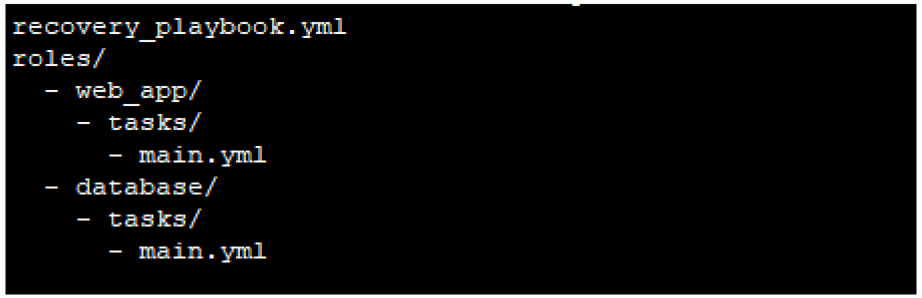
Test and Validate

Integration with Monitoring Systems

Secure Credentials

Configuration Management

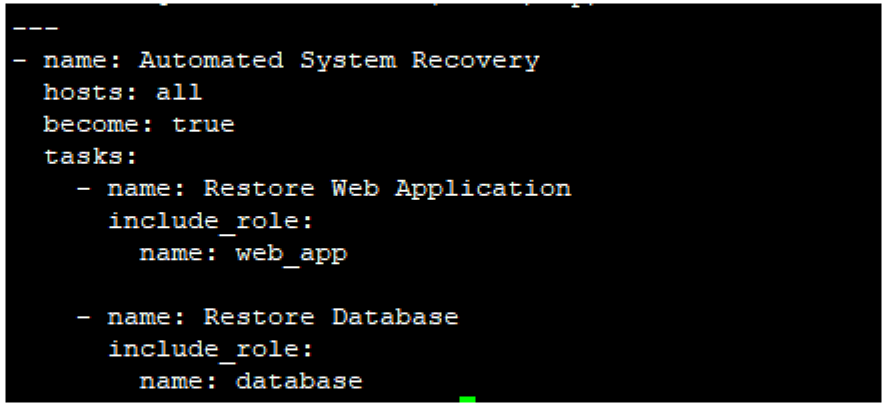
Example Directory Structure



```
recovery_playbook.yml
roles/
  - web_app/
    - tasks/
      - main.yml
  - database/
    - tasks/
      - main.yml
```

Configuration Management

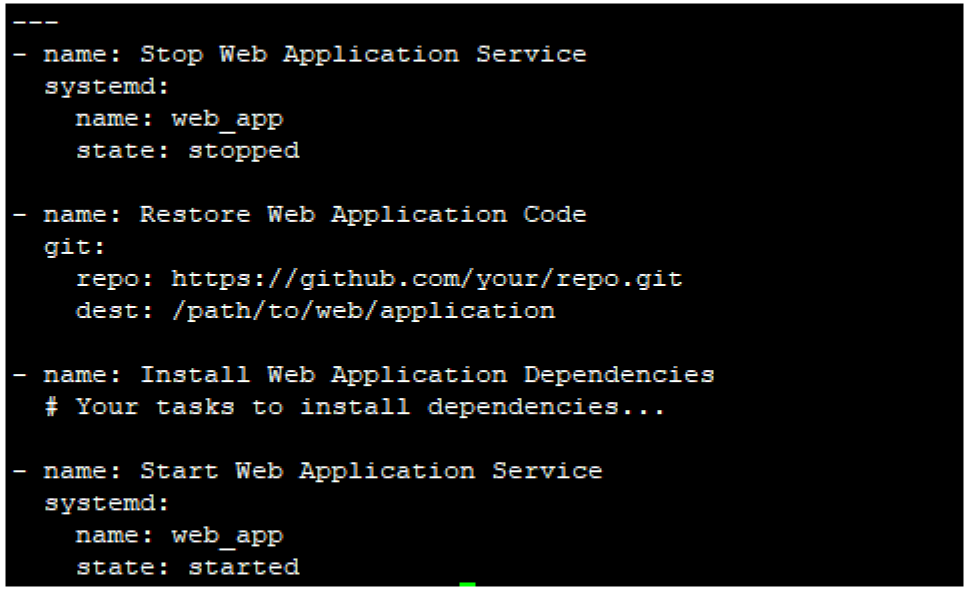
recovery_playbook.yaml



```
---
- name: Automated System Recovery
  hosts: all
  become: true
  tasks:
    - name: Restore Web Application
      include_role:
        name: web_app

    - name: Restore Database
      include_role:
        name: database
```

Configuration Management

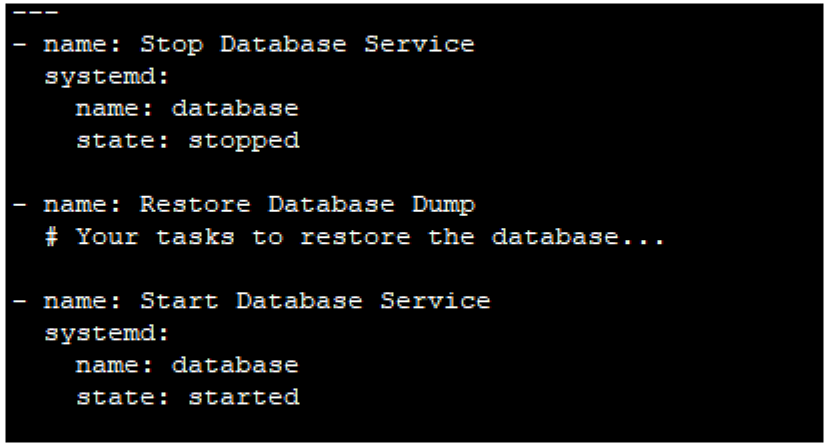
Web Application Role
(roles/webapp/tasks/main.yaml)



```
---
- name: Stop Web Application Service
  systemd:
    name: web_app
    state: stopped

- name: Restore Web Application Code
  git:
    repo: https://github.com/your/repo.git
    dest: /path/to/web/application

- name: Install Web Application Dependencies
  # Your tasks to install dependencies...

- name: Start Web Application Service
  systemd:
    name: web_app
    state: started
```

Configuration Management

Database Role
(roles/database/tasks/main.yaml)



```
---
- name: Stop Database Service
  systemd:
    name: database
    state: stopped

- name: Restore Database Dump
  # Your tasks to restore the database...

- name: Start Database Service
  systemd:
    name: database
    state: started
```

Configuration Management

Question:

You're implementing a blue-green deployment strategy for a web application. Explain how you would use Ansible to manage and switch between the blue and green environments while minimizing risks and ensuring rollback capabilities.

Configuration Management

Blue/Green Deployment

Blue/green deployment is a release management strategy where two identical environments, "blue" (production) and "green" (new version), coexist. Only one environment serves live traffic at a time.

Configuration Management

Deployment process

Verification and Testing

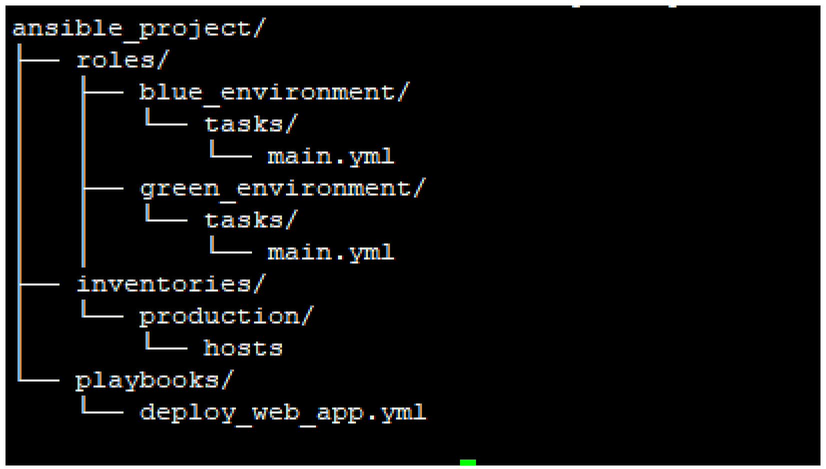
Switching Traffic

Rollback Capability

Continuous Availability

Configuration Management

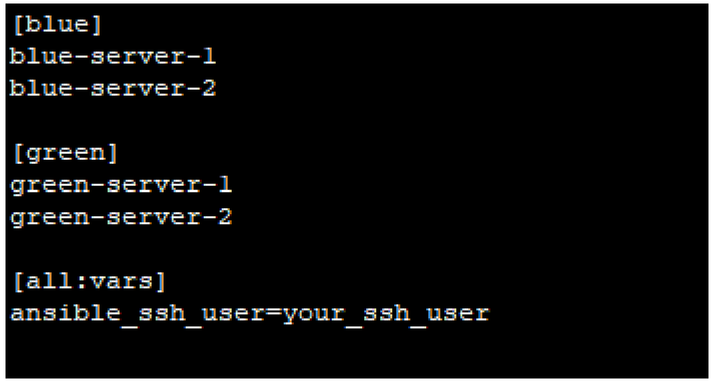
Directory Structure



```
ansible_project/
├── roles/
│   ├── blue_environment/
│   │   └── tasks/
│   │       └── main.yml
│   ├── green_environment/
│   │   └── tasks/
│   │       └── main.yml
├── inventories/
│   └── production/
│       └── hosts
└── playbooks/
    └── deploy_web_app.yml
```

Configuration Management

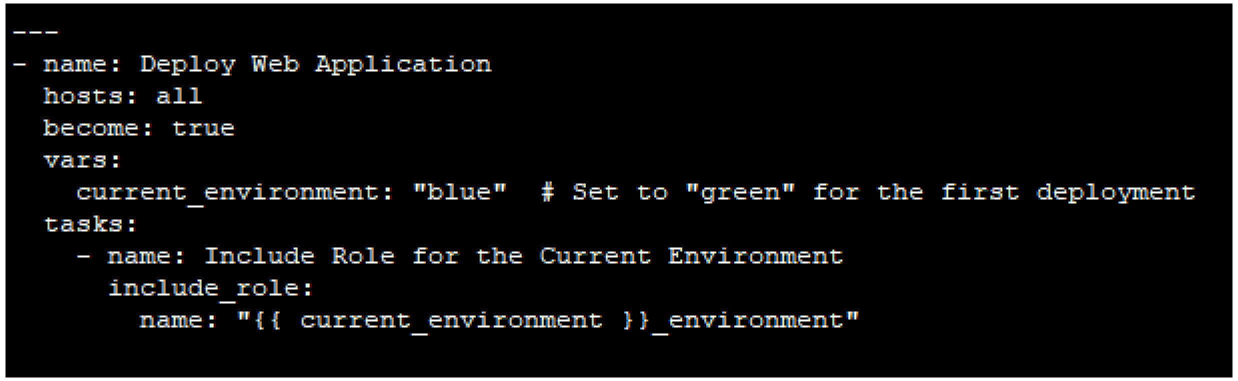
Inventory Configuration



```
[blue]
blue-server-1
blue-server-2

[green]
green-server-1
green-server-2

[all:vars]
ansible_ssh_user=your_ssh_user
```

Configuration Management

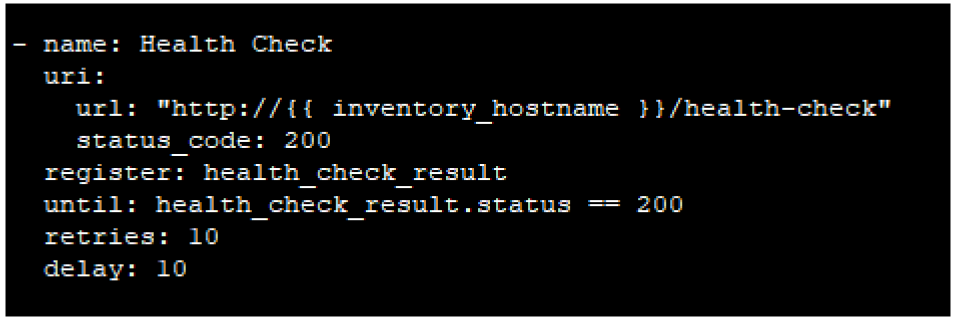
Playbook



```
---
- name: Deploy Web Application
  hosts: all
  become: true
  vars:
    current_environment: "blue"  # Set to "green" for the first deployment
  tasks:
    - name: Include Role for the Current Environment
      include_role:
        name: "{{ current_environment }}_environment"
```

Configuration Management

Health Checks



```
- name: Health Check
  uri:
    url: "http://{{ inventory_hostname }}/health-check"
    status_code: 200
  register: health_check_result
  until: health_check_result.status == 200
  retries: 10
  delay: 10
```