Development Guidelines

Overview

This document outlines the development practices, standards, and workflows for the HX-Infrastructure Ansible project. All contributors must follow these guidelines to ensure code quality, maintainability, and consistency.

Code Standards

Ansible Best Practices

1. **Idempotency**: All tasks must be idempotent
2. **Error Handling**: Implement proper error handling and rollback mechanisms
3. **Documentation**: Document all roles, playbooks, and complex tasks
4. **Testing**: Include appropriate tests for all code changes

YAML Formatting

- Use 2 spaces for indentation
- Maximum line length: 120 characters
- Use descriptive names for tasks, handlers, and variables
- Follow consistent naming conventions (snake_case for variables)

Variable Management

- Use group_vars for environment-specific variables
- Use host_vars for host-specific configurations
- Encrypt sensitive data using Ansible Vault
- Document all variables in defaults/main.yml

Role Structure

Follow the standard Ansible role structure:

```
roles/role_name/
├── defaults/main.yml      # Default variables
├── vars/main.yml          # Role variables
├── tasks/main.yml         # Main task list
├── handlers/main.yml      # Handlers
├── templates/             # Jinja2 templates
├── files/                 # Static files
├── meta/main.yml          # Role metadata
└── README.md              # Role documentation
```

Development Workflow

1. Branch Strategy

- `main` : Production-ready code
- `develop` : Integration branch for features

- `feature/*`: Feature development branches
- `hotfix/*`: Critical fixes for production

2. Development Process

1. Create feature branch from `develop`
2. Implement changes following coding standards
3. Write/update tests
4. Run linting and testing locally
5. Create pull request to `develop`
6. Code review and approval
7. Merge to `develop`

3. Testing Requirements

All changes must include:
- Syntax validation (`ansible-playbook --syntax-check`)
- Linting checks (`yamllint`, `ansible-lint`)
- Molecule tests for roles (where applicable)
- Integration tests for complex workflows

Quality Assurance

Pre-commit Checks

Before committing code, run:

```
# YAML linting
yamllint .

# Ansible linting
ansible-lint

# Syntax check
ansible-playbook --syntax-check playbooks/site/main.yml

# Dry run test
ansible-playbook --check -i inventories/dev playbooks/site/main.yml
```

Code Review Checklist

- [ ] Code follows established patterns and conventions
- [ ] All variables are properly documented
- [ ] Sensitive data is encrypted with Ansible Vault
- [ ] Tasks are idempotent and include proper error handling
- [ ] Documentation is updated for significant changes
- [ ] Tests are included and passing

Security Guidelines

Credential Management

- Never commit plaintext passwords or API keys
- Use Ansible Vault for sensitive data

- Implement least-privilege access principles
- Regularly rotate credentials

Access Control

- Use SSH keys for authentication
- Implement proper sudo configurations
- Audit access logs regularly
- Follow principle of least privilege

Documentation Standards

Code Documentation

- Document all roles with comprehensive README.md files
- Include variable descriptions and examples
- Document complex logic and business rules
- Maintain up-to-date architecture diagrams

Commit Messages

Follow conventional commit format:

```
type(scope): description

[optional body]

[optional footer]
```

Types: feat, fix, docs, style, refactor, test, chore

Environment Management

Development Environment

- Use consistent development tools and versions
- Maintain environment-specific configurations
- Test changes in development before promoting
- Document environment setup procedures

Deployment Process

1. Test in development environment
2. Promote to test environment for integration testing
3. User acceptance testing
4. Deploy to production with proper change management

Troubleshooting

Common Issues

1. **Connectivity Issues**: Check SSH access and network connectivity
2. **Permission Errors**: Verify sudo configurations and user permissions

3. **Idempotency Failures**: Review task logic and state management
4. **Variable Conflicts**: Check variable precedence and scoping

Debugging Techniques

- Use `-vvv` flag for verbose output
- Enable debug tasks for complex logic
- Use `ansible-playbook --check` for dry runs
- Implement proper logging and monitoring

Performance Optimization

Best Practices

- Use `gather_facts: false` when facts are not needed
- Implement fact caching for better performance
- Use `serial` keyword for controlled deployments
- Optimize task execution with proper conditionals

Monitoring

- Monitor playbook execution times
- Track resource utilization during deployments
- Implement alerting for failed deployments
- Regular performance reviews and optimization

Contribution Guidelines

Getting Started

1. Fork the repository
2. Set up development environment
3. Read through existing code and documentation
4. Start with small, focused contributions

Pull Request Process

1. Ensure all tests pass
2. Update documentation as needed
3. Follow the established code review process
4. Address feedback promptly and professionally

For questions or clarification on these guidelines, please contact the HX Infrastructure Team.