HX-Citadel Ansible Infrastructure Analysis

Analyzed by: DeepAgent
Date: October 12, 2025
Repository: hx-citadel-ansible
Analysis Scope: Core Ansible infrastructure, playbooks, roles, inventory, and configuration

Table of Contents

Executive Summary

The HX-Citadel Ansible infrastructure is a **well-organized, production-ready deployment framework** for a distributed AI/LLM orchestration platform. The repository demonstrates strong adherence to Ansible best practices with a clear separation of concerns, modular role design, and comprehensive variable management.

Key Highlights

✅ **Strengths:**
- Excellent FQDN-based infrastructure design preventing hardcoded IPs
- Well-structured role hierarchy with 24 specialized roles
- Comprehensive variable layering (global → group → host)
- Modular playbook design with clear deployment stages
- Strong testing framework integration
- Good documentation coverage

⚠️ **Areas Requiring Attention:**
- Vault file management needs consolidation (`vault.yml.broken` indicates issues)
- Some inventory hosts use different SSH keys (inconsistency)

- Missing role-level README.md files for documentation
- No explicit rollback/recovery playbooks (except MCP recovery)
- Potential IP hardcoding violations need verification

Infrastructure Scope

- **17 managed hosts** across 8 host groups
- **24 custom roles** + 2 external Galaxy roles
- **26 playbooks** covering deployment, validation, and recovery
- **217 YAML files** across the entire project
- **9 utility scripts** for automation and validation

Project Statistics

```
Total YAML Files:              217
Custom Roles:                  24
Galaxy Roles:                  2 (PostgreSQL, Redis)
Main Playbooks:                26
Inventory Files:               2
Group Variable Files:          3
Host Variable Files:           3
Utility Scripts:               9
Test Files:                    Multiple (see TESTS_ANALYSIS.md)
```

Role Breakdown

Category	Roles	Purpose
Base Infrastructure	base-setup	System configuration, logging, health checks
Database Layer	postgresql, redis, postgresql_role, redis_role	Data persistence and caching
Vector Database	qdrant, qdrant_web_ui	Embeddings and similarity search
LLM Services	ollama, litellm	Model hosting and API gateway
Orchestration	orchestrator_* (9 roles)	Core application logic and workflow
API Layer	fastapi, prisma	Application interfaces
Observability	observability	Monitoring and metrics
MCP Server	fastmcp_server	Model context protocol

Architecture Overview

```
graph TB
    subgraph "Entry Point"
        A[site.yml]
    end

    subgraph "Deployment Stages"
        B[deploy-base.yml]
        C[deploy-db.yml]
        D[deploy-vector.yml]
        E[deploy-llm.yml]
        F[deploy-api.yml]
        G[deploy-orchestrator-local.yml]
    end

    subgraph "Host Groups"
        H[db_nodes]
        I[vector_nodes]
        J[llm_nodes]
        K[orchestrator_nodes]
        L[api_nodes]
        M[ui_nodes]
        N[mcp_nodes]
    end

    subgraph "Roles Layer"
        O[base-setup]
        P[postgresql/redis]
        Q[qdrant]
        R[ollama/litellm]
        S[fastapi/prisma]
        T[orchestrator_*]
    end

    A --> B
    A --> C
    A --> D
    A --> E
    A --> F
    A --> G

    B --> O
    C --> P
    C --> H
    D --> Q
    D --> I
    E --> R
    E --> J
    F --> S
    F --> K
    F --> L
    G --> T
    G --> K
```

Infrastructure Layers

1. **Base Layer**: Common configuration, logging, health checks
2. **Data Layer**: PostgreSQL (app DB) + Redis (caching)
3. **Vector Layer**: Qdrant for embeddings and similarity search

4. **LLM Layer**: Ollama (model hosting) + LiteLLM (unified API)
5. **Application Layer**: FastAPI, Prisma ORM
6. **Orchestration Layer**: Core business logic with multiple framework integrations
7. **UI/Frontend Layer**: Web interfaces
8. **MCP Layer**: Model Context Protocol servers

Configuration Management

ansible.cfg

Location: `/home/ubuntu/hx-citadel-ansible/ansible.cfg`

```ini
[defaults]
inventory = ./inventory/prod.ini
roles_path = ./roles
host_key_checking = False
retry_files_enabled = False
gathering = smart
fact_caching = jsonfile
fact_caching_connection = ./.ansible_cache
fact_caching_timeout = 86400
vault_password_file = ~/.ansible_vault_pass

[ssh_connection]
pipelining = True
ssh_args = -o ControlMaster=auto -o ControlPersist=60m -o ControlPath=~/.ssh/cm-%r@%h:%p
```

Strengths ✅

- **Smart gathering** with fact caching (24-hour timeout) reduces execution time
- **SSH multiplexing** enabled for performance optimization
- **Pipelining** enabled for reduced SSH overhead
- **No retry files** prevents clutter
- **Vault password file** configured for automated operations

Concerns ⚠️

- `host_key_checking = False` is a **security risk** in production
- **Vulnerable to MITM attacks**
- Should only be used in isolated/trusted networks
- Recommendation: Enable host key checking and maintain `known_hosts`

site.yml - Master Playbook

```yaml
---
- import_playbook: playbooks/deploy-base.yml
- import_playbook: playbooks/deploy-db.yml
- import_playbook: playbooks/deploy-vector.yml
- import_playbook: playbooks/deploy-llm.yml
- import_playbook: playbooks/deploy-api.yml
- import_playbook: playbooks/deploy-orchestrator-local.yml
```

Design Pattern: Sequential import strategy ensures proper dependency ordering.

Evaluation:
- ✅ Clear separation of deployment stages
- ✅ Easy to run individual stages via tags or direct playbook execution
- ⚠️ No error handling between stages
- ⚠️ No pre/post deployment validation hooks
- ⚠️ Consider adding `--check` mode support verification

requirements.yml - External Dependencies

```
---
roles:
  - src: geerlingguy.postgresql
    name: postgresql_role
  - src: geerlingguy.redis
    name: redis_role
collections:
  - name: community.docker
  - name: community.general
```

Evaluation:
- ✅ Uses trusted community roles (geerlingguy)
- ✅ Version pinning would be beneficial
- ⚠️ Missing version constraints (e.g., `version: "3.4.0"`)
- ⚠️ No specification of collection versions

Recommendation:

```
roles:
  - src: geerlingguy.postgresql
    version: "3.5.0"  # Pin to specific version
    name: postgresql_role
```

Inventory Structure

Production Inventory (prod.ini)

Location: `inventory/prod.ini`

Host Groups

Group	Hosts	Purpose
`all`	17 hosts	Global scope
`db_nodes`	hx-sqldb-server	PostgreSQL + Redis
`vector_nodes`	hx-vectordb-server	Qdrant vector DB
`llm_nodes`	hx-ollama1, hx-ollama2	Model hosting
`orchestrator_nodes`	hx-orchestrator-server	Core orchestration
`api_nodes`	hx-litellm-server, hx-prisma-server	API services
`monitoring_nodes`	hx-metrics-server	Observability
`mcp_nodes`	hx-mcp1-server	MCP protocol
`ui_nodes`	hx-qwebui-server	Web UI

Global Variables

```
[all:vars]
ansible_user=agent0
ansible_ssh_private_key_file=~/.ssh/id_rsa
ansible_python_interpreter=/usr/bin/python3
ansible_connection=ssh
ansible_timeout=30
```

Strengths ✅

- Clear host-to-group mappings
- Consistent naming convention (`hx-*-server`)
- Logical grouping by function/service
- Global SSH settings for consistency

Issues ⚠️

Inconsistent SSH Key Usage:

```
# Most hosts use id_rsa
ansible_ssh_private_key_file=~/.ssh/id_rsa

# But these two use ed25519
hx-qwebui-server ansible_host=192.168.10.53 ansible_ssh_private_key_file=~/.ssh/id_ed25519
hx-mcp1-server ansible_host=192.168.10.59 ansible_ssh_private_key_file=~/.ssh/id_ed25519
```

Why this matters:

- Creates confusion about which key to use

- May indicate these hosts were added later
- Could lead to authentication failures if keys aren't synchronized

Recommendation:
- Standardize on one key type (preferably ed25519 for security)
- OR document why certain hosts require different keys
- Consider using `ansible_ssh_private_key_file` in group_vars instead

StrictHostKeyChecking Disabled:

```
ansible_ssh_common_args='-o StrictHostKeyChecking=no'
```

- This is a **security vulnerability**
- Only justified for initial provisioning or testing
- Should be removed in production

Variable Hierarchy

Ansible evaluates variables in this precedence order (highest to lowest):

```
1. Extra vars (-e on CLI)
2. Task vars
3. Block vars
4. Role vars
5. Play vars
6. Host vars
7. Group vars
8. Facts/registered vars
9. Role defaults
```

Current Structure

```
group_vars/
├── all/
│   ├── fqdn_map.yml        # FQDN mappings (EXCELLENT!)
│   ├── main.yml            # Global defaults
│   ├── vault.yml           # Encrypted secrets
│   └── vault.yml.broken    # ⚠️ Problem indicator
├── db_nodes.yml            # Database-specific vars
└── llm_nodes.yml           # LLM-specific vars

host_vars/
├── hx-ollama1.yml          # Ollama1-specific models
├── hx-ollama2.yml          # Ollama2-specific models
└── hx-orchestrator-server.yml  # Orchestrator-specific config
```

group_vars/all/main.yml

```yaml
---
ubuntu_version: "24.04"
python_version: "3.12"
app_dir: "/opt/hx-citadel-shield"
log_dir: "/var/log/hx-citadel"
services_to_restart: []
```

Evaluation:
- ✅ Clean, minimal global defaults
- ✅ Appropriate abstraction level
- ⚠️ `services_to_restart: []` seems unused (verify)

group_vars/all/fqdn_map.yml

This file is **OUTSTANDING** and represents a best practice for infrastructure management.

Key Features:

1. **Four mapping dictionaries:**
   - `hx_hosts_fqdn`: Short hostname → FQDN
   - `hx_hosts_ip`: FQDN → IP (for reporting only)
   - `hx_ip_to_fqdn`: IP → FQDN (for auto-fix tasks)
   - `hx_hosts_short_ip`: Short hostname → IP (legacy)

2. **Clear usage guidance in comments:**

```yaml
# Short hostname → FQDN mapping
# Usage: {{ hx_hosts_fqdn['hx-ollama1'] }}
```

1. **Prevents hardcoded IPs/hostnames in:**
   - Templates
   - Configuration files
   - Application code
   - Service definitions

Example Usage:

```yaml
# Instead of:
postgresql_host: "192.168.10.48"  # BAD

# Use:
postgresql_host: "{{ hx_hosts_fqdn['hx-sqldb-server'] }}"  # GOOD
```

Why This Matters:
- Makes infrastructure changes easy (update one file, not 100+ templates)
- Enables environment promotion (dev → staging → prod)
- Reduces errors from typos in hostnames/IPs
- Facilitates disaster recovery and migration

group_vars/db_nodes.yml

```yaml
---
db_name: "hx_citadel"
db_owner: "hx_pg"
db_state: "present"
db_encoding: "UTF8"
db_lc_collate: "en_US.UTF-8"
db_lc_ctype: "en_US.UTF-8"

pg_login_unix_socket: "/var/run/postgresql"
pg_login_user: "postgres"

postgresql_listen_address: "0.0.0.0"
postgresql_port: 5432

redis_bind_interface: "127.0.0.1 {{ redis_additional_bind_ip | default(ansible_default_ipv4.address) }}"
redis_additional_bind_ip: "192.168.10.48"
```

Issues:

1. **Hardcoded IP:**

```yaml
redis_additional_bind_ip: "192.168.10.48"  # ⚠️ Should use fqdn_map
```

Should be:

```yaml
redis_additional_bind_ip: "{{ hx_hosts_ip[hx_hosts_fqdn['hx-sqldb-server']] }}"
```

1. **Security Concern:**

```yaml
postgresql_listen_address: "0.0.0.0"  # Listens on ALL interfaces
```

- This exposes PostgreSQL to all network interfaces
- Should restrict to specific interfaces or use firewall rules
- Consider: `"127.0.0.1,{{ ansible_default_ipv4.address }}"`

1. **Missing Redis Password:**

```yaml
# redis_requirepass: "CHANGEME_STRONG"  # Commented out!
```

- Redis has **no authentication** enabled
- **Critical security vulnerability** if exposed to network
- Must be configured via vault

host_vars Examples

```yaml
# host_vars/hx-ollama1.yml
---
ollama_models:
  - gemma3:27b
  - gpt-oss:20b
  - mistral:7b
```

Evaluation:
- ✅ Appropriate use of host_vars for host-specific model lists
- ✅ Clean, simple structure
- ⚠️ No model version pinning (could cause inconsistencies)

Recommendation:

```yaml
ollama_models:
  - name: gemma3
    version: "27b"
    checksum: "sha256:abc123..."  # For verification
  - name: gpt-oss
    version: "20b"
```

Vault Management

Issue Found:

```
group_vars/all/vault.yml
group_vars/all/vault.yml.broken  # ⚠️ What happened here?
```

Critical Questions:
1. Why does `vault.yml.broken` exist?
2. Was there a vault password issue?
3. Are there backup/recovery procedures for vault files?
4. Is `vault.yml` properly encrypted?

Recommendation:
- Document vault recovery procedures in README.md
- Remove `.broken` file or move to a `backups/` directory
- Implement vault file rotation strategy
- Consider using separate vaults per environment

Playbook Organization

Overview

Location: `playbooks/`
Total Playbooks: 26
Categories:
- Deployment (16 playbooks)
- Validation (3 playbooks)

- Recovery (1 playbook)
- Maintenance (indirect via maintenance/ directory)

Deployment Playbooks

Playbook	Purpose	Target Hosts
`deploy-base.yml`	System baseline configuration	`all`
`deploy-db.yml`	PostgreSQL + Redis setup	`db_nodes`
`deploy-vector.yml`	Qdrant vector database	`vector_nodes`
`deploy-llm.yml`	Ollama model hosting	`llm_nodes`
`deploy-api.yml`	LiteLLM + Prisma API layers	`api_nodes`
`deploy-orchestrator-local.yml`	Local orchestrator setup	`orchestrator_nodes`

Orchestrator Sub-Playbooks

The orchestrator has **8 specialized sub-playbooks:**

1. `deploy-orchestrator-base.yml` - Foundation setup
2. `deploy-orchestrator-fastapi.yml` - FastAPI application
3. `deploy-orchestrator-postgresql.yml` - Database connections
4. `deploy-orchestrator-redis.yml` - Cache layer
5. `deploy-orchestrator-qdrant.yml` - Vector DB integration
6. `deploy-orchestrator-lightrag.yml` - LightRAG framework
7. `deploy-orchestrator-workers.yml` - Background workers
8. `deploy-orchestrator-langgraph.yml` - LangGraph integration

Additional Framework Integrations:
- `deploy-copilotkit.yml` - CopilotKit integration
- `deploy-pydantic-ai.yml` - Pydantic AI framework

Validation Playbooks

Playbook	Purpose	When to Run
`preflight-check.yml`	Pre-deployment validation	Before any deployment
`validate-mcp-prereqs.yml`	MCP prerequisites check	Before MCP deployment
`smoke-tests.yml`	Post-deployment health checks	After deployment
`deploy-with-validation.yml`	Deployment with inline validation	Full stack deployment

Recovery Playbooks

- `recover-mcp-server.yml` (14,858 lines!) - Comprehensive MCP recovery procedures

Example: deploy-db.yml

```yaml
---
- name: Deploy database services
  hosts: db_nodes
  become: yes
  roles:
    - postgresql
    - redis
  post_tasks:
    - name: Verify PostgreSQL connectivity
      community.postgresql.postgresql_ping:
        db: "{{ db_name | default('hx_citadel') }}"
        login_user: "{{ db_owner | default('hx_pg') }}"
        login_password: "{{ db_password | default(omit) }}"
        login_unix_socket: "{{ pg_login_unix_socket | default('/var/run/postgresql') }}"
      become: yes
      become_user: postgres
      changed_when: false
      no_log: true
  tags:
    - db
```

Strengths ✅:
- Uses `post_tasks` for validation
- Properly uses `become_user` for PostgreSQL operations
- `no_log: true` prevents password leakage
- `changed_when: false` for idempotent checks
- Tagged for selective execution

Recommendation:
Add pre_tasks for validation:

```yaml
pre_tasks:
  - name: Verify required variables are defined
    assert:
      that:
        - db_name is defined
        - db_owner is defined
      fail_msg: "Required database variables not defined"
```

Role Architecture

Role Inventory

Total Roles: 24 custom + 2 Galaxy
Main Task Files: 76

```
roles/
├── base-setup                    # Foundation
├── postgresql, redis             # Data layer (custom)
├── postgresql_role, redis_role   # Data layer (Galaxy)
├── qdrant, qdrant_web_ui        # Vector DB
├── ollama, litellm              # LLM services
├── fastapi, prisma              # API layer
├── orchestrator_base_setup      # Orchestrator foundation
├── orchestrator_fastapi         # FastAPI app
├── orchestrator_postgresql      # DB integration
├── orchestrator_redis           # Cache integration
├── orchestrator_qdrant          # Vector DB integration
├── orchestrator_lightrag        # LightRAG framework
├── orchestrator_workers         # Background workers
├── orchestrator_langgraph       # LangGraph framework
├── orchestrator_copilotkit      # CopilotKit integration
├── orchestrator_pydantic_ai     # Pydantic AI framework
├── observability                # Monitoring
└── fastmcp_server              # MCP protocol
```

Role Structure Example: base-setup

```
roles/base-setup/
├── defaults/
│   └── main.yml                  # Default variable values
├── handlers/
│   └── main.yml                  # Event handlers (restart services, etc.)
├── tasks/
│   ├── main.yml                  # Main task entry point
│   ├── 06-logrotate.yml          # Log rotation config
│   ├── 07-sudo.yml               # Sudo permissions
│   ├── 08-convenience-scripts.yml   # Helper scripts
│   └── 09-health-check.yml       # Health monitoring
├── templates/
│   ├── activate-shield.sh.j2     # Environment activation script
│   ├── health-check.sh.j2        # Health check script
│   ├── logrotate.conf.j2         # Log rotation config
│   ├── manage-services.sh.j2     # Service management
│   └── sudoers.j2               # Sudo configuration
└── vars/
    └── main.yml                  # Role-specific variables
```

Design Patterns:
- ✅ Tasks numbered for execution order clarity (06, 07, 08, 09)
- ✅ Jinja2 templates for all configuration files
- ✅ Separate handlers for service management
- ✅ Clear separation of defaults vs. vars

Role Structure Example: ollama

```
roles/ollama/
├── handlers/
│   └── main.yml
├── tasks/
│   └── main.yml
└── vars/
    └── main.yml
```

Observation:
- Much simpler structure than base-setup
- No templates or defaults directory
- Suggests straightforward installation without heavy configuration

Role Documentation Gap

Critical Issue: Most roles lack README.md files

Current State:

````
$ find roles -name "README.md" | wc -l
# Likely 0 or very few
````

Impact:
- New team members can’t understand role purposes quickly
- No documentation of role variables
- No usage examples
- Difficult to maintain

Recommendation:
Create standardized README.md template:

````
# Role: role_name

## Purpose
Brief description of what this role does.

## Requirements
- OS: Ubuntu 24.04
- Dependencies: list here

## Role Variables

### Required Variables
- `var_name`: Description

### Optional Variables
- `var_name`: Description (default: value)

## Dependencies
List other roles that must run first.

## Example Playbook

\`\`\`yaml
- hosts: servers
  roles:
    - role: role_name
      var_name: value
\`\`\`

## Tags
- `tag_name`: Description

## Author
Team name / contact
````

Scripts and Utilities

Script Inventory

Location: `scripts/`
Total Scripts: 9

Script	Purpose	Type
`check-fqdn.sh`	Validates FQDN usage across codebase	Linting
`auto-fix-types.sh`	Automated type correction	Auto-fix
`validate-types.sh`	Type checking validation	Linting
`fix-main-py-indentation.sh`	Python indentation fixer	Auto-fix
`slack-notify.sh`	Slack notifications	Integration
`linear-api-call.sh`	Linear API interactions	Integration
`fetch-linear-issue.sh`	Fetch Linear issues	Integration
`test-linear-issue.sh`	Test Linear API connectivity	Integration
`fix-linear-issue.sh`	Automated issue fixing	Integration

Integration Ecosystem

The scripts reveal **tight integration** with:
- **Slack** - Notifications and alerts
- **Linear** - Issue tracking and automation
- **Type checking** - Code quality enforcement
- **FQDN validation** - Infrastructure consistency

check-fqdn.sh Analysis

This script is **critical** for maintaining FQDN discipline:

Purpose:
- Scans all YAML files for hardcoded IPs/hostnames
- Validates against `.fqdn-allowlist`
- Enforces the use of `hx_hosts_fqdn` variables

Recommendation:
- Integrate into pre-commit hooks
- Run in CI/CD pipeline
- Document exceptions in `.fqdn-allowlist`

Slack Integration

Files:
- `scripts/slack-notify.sh`
- Likely used for deployment notifications

Potential Issues:
- Are Slack webhooks stored in vault?
- Is there documentation on notification triggers?
- Can notifications be disabled for testing?

Linear Integration

Purpose: Automated issue management

Files:
- `linear-api-call.sh` - Base API wrapper
- `fetch-linear-issue.sh` - Retrieve issue details
- `fix-linear-issue.sh` - Automated issue resolution (8,583 lines!)
- `test-linear-issue.sh` - API connectivity verification

Questions:
- How are Linear API tokens managed? (should be in vault)
- What triggers automated issue fixes?
- Are there safeguards against accidental issue modifications?

Strengths and Best Practices

1. FQDN-Based Infrastructure ⭐⭐⭐⭐⭐

Outstanding implementation. The `fqdn_map.yml` file with its four mapping dictionaries is a **gold standard** for infrastructure management.

Benefits:
- Single source of truth for host information
- Easy environment migration
- Prevents hardcoded values
- Facilitates disaster recovery

2. Modular Role Design ⭐⭐⭐⭐

Well-architected role separation:
- Clear single-responsibility principle
- Reusable components
- Orchestrator broken into 9 specialized sub-roles
- External Galaxy roles for common services

3. Variable Hierarchy ⭐⭐⭐⭐

Proper use of group_vars and host_vars:
- Global defaults in `group_vars/all/`
- Group-specific configs in `group_vars/<group>.yml`
- Host-specific overrides in `host_vars/`

`<host>.yml`
- Clean separation of concerns

4. Playbook Organization ⭐⭐⭐⭐

Logical structure:
- Master `site.yml` imports all deployment stages
- Each stage can run independently
- Post-task validation in key playbooks
- Tags for selective execution

5. Testing Framework ⭐⭐⭐⭐⭐

As documented in TESTS_ANALYSIS.md:
- Comprehensive unit tests
- Integration tests
- Load test planning
- Type checking integration
- 5,665 lines of test code

6. Validation Playbooks ⭐⭐⭐⭐

Proactive validation:
- Preflight checks before deployment
- Smoke tests after deployment
- MCP prerequisite validation
- Inline validation playbook option

7. SSH Optimization ⭐⭐⭐⭐

Performance tuning:
- ControlMaster/ControlPersist for connection reuse
- Pipelining enabled
- Smart fact gathering with caching
- 24-hour fact cache timeout

8. Automation Scripts ⭐⭐⭐⭐

Quality enforcement:
- FQDN validation automation
- Type checking automation
- Auto-fix capabilities
- Integration with external tools

9. Security Consciousness ⭐⭐⭐

Security features:
- Vault for secrets management
- `no_log: true` in sensitive tasks
- Unix socket authentication for PostgreSQL
- Separate sudo configuration

10. Clear Naming Conventions ⭐⭐⭐⭐

Consistent patterns:
- All hosts prefixed with `hx-`

- Role names clearly indicate purpose
- Playbook names follow `deploy-<service>.yml` pattern
- Group names match service types

Areas for Improvement

1. Vault File Management ⚠️⚠️⚠️ HIGH PRIORITY

Issue:

```
group_vars/all/vault.yml
group_vars/all/vault.yml.broken  # What happened?
```

Problems:
- Unclear vault status
- No documented recovery procedures
- Possible encryption/password issues
- Risk of losing encrypted secrets

Recommendations:
1. **Immediate Actions:**
- Document what happened with `vault.yml.broken`
- Verify `vault.yml` is properly encrypted
- Test vault password access
- Remove or move `.broken` file

1. **Long-term Solutions:**
   - Implement vault backup strategy
   - Document vault recovery procedures in README
   - Consider separate vaults per environment
   - Set up vault password rotation schedule
   - Use `ansible-vault rekey` for password changes

2. **Documentation Template:**

````
## Vault Management

### Vault Password Location
- Password file: `~/.ansible_vault_pass`
- Backup location: [specify secure location]

### Rotating Vault Password
\`\`\`bash
ansible-vault rekey group_vars/all/vault.yml
\`\`\`

### Viewing Vault Contents
\`\`\`bash
ansible-vault view group_vars/all/vault.yml
\`\`\`

### Emergency Recovery
If vault password is lost:
1. [document recovery steps]
2. [contact information]
````

2. Inconsistent SSH Key Usage ⚠️⚠️

Issue:

```
# Most hosts
ansible_ssh_private_key_file=~/.ssh/id_rsa

# Exceptions
hx-qwebui-server ansible_ssh_private_key_file=~/.ssh/id_ed25519
hx-mcp1-server ansible_ssh_private_key_file=~/.ssh/id_ed25519
```

Problems:

- Confusion about key management
- Potential authentication failures
- Unclear why certain hosts differ

Recommendations:

1. **Standardize on ed25519:**
- More secure than RSA
- Better performance
- Industry best practice

1. **Implementation:**

```
# In group_vars/all/main.yml
ansible_ssh_private_key_file: "~/.ssh/id_ed25519"

# Remove overrides from inventory unless documented
```

1. **If exceptions are needed:**
    - Document WHY in comments
    - Use group_vars for groups of hosts with special needs
    - Don’t scatter overrides across inventory

3. Security Hardening ⚠️⚠️⚠️ HIGH PRIORITY

Critical Issues:

A. Disabled Host Key Checking

```
# ansible.cfg
host_key_checking = False  # ⚠️ SECURITY RISK
```

Risk: Man-in-the-middle attacks

Fix:

```
[defaults]
host_key_checking = True
# Maintain known_hosts file properly
```

B. Disabled StrictHostKeyChecking

```
# inventory/prod.ini
ansible_ssh_common_args='-o StrictHostKeyChecking=no'  # ⚠️ SECURITY RISK
```

Fix:
- Remove this argument
- Accept host keys on first connection
- Maintain `known_hosts` file

C. PostgreSQL Listening on All Interfaces

```
# group_vars/db_nodes.yml
postgresql_listen_address: "0.0.0.0"  # ⚠️ Too permissive
```

Fix:

```
postgresql_listen_address: "127.0.0.1,{{ ansible_default_ipv4.address }}"
# OR use firewall rules to restrict access
```

D. Redis Without Authentication

```
# group_vars/db_nodes.yml
# redis_requirepass: "CHANGEME_STRONG"  # ⚠️ Commented out!
```

Fix:

```
# In vault.yml
redis_requirepass: "{{ vault_redis_password }}"

# In redis role/tasks
- name: Configure Redis authentication
  lineinfile:
    path: /etc/redis/redis.conf
    regexp: '^# requirepass'
    line: "requirepass {{ redis_requirepass }}"
  notify: restart redis
```

E. Hardcoded IP in Variables

```
# group_vars/db_nodes.yml
redis_additional_bind_ip: "192.168.10.48"  # ⚠️ Violates FQDN policy
```

Fix:

```
redis_additional_bind_ip: "{{ hx_hosts_ip[hx_hosts_fqdn['hx-sqldb-server']] }}"
```

4. Missing Role Documentation ⚠️⚠️

Issue: Zero or very few role README.md files

Impact:
- New team members struggle to understand roles
- No variable documentation
- No usage examples
- Maintenance difficulties

Recommendation:

Create standardized README.md for each role:

````
# Role: <role_name>

## Purpose
Brief description (1-2 sentences)

## Supported OS
- Ubuntu 24.04 (tested)
- Other versions: list or "not tested"

## Requirements

### System Requirements
- Minimum RAM: XGB
- Disk space: XGB
- Network: open ports list

### Dependencies
- Must run after: `role_name`
- Requires: list packages/services

## Variables

### Required Variables
| Variable | Description | Example |
|----------|-------------|---------|
| `var_name` | What it does | `value` |

### Optional Variables
| Variable | Description | Default |
|----------|-------------|---------|
| `var_name` | What it does | `value` |

## Usage Example

\`\`\`yaml
- hosts: target_group
  roles:
    - role: role_name
      var_name: custom_value
\`\`\`

## Tags
- `tag_name` - Description of what this tag controls

## Handlers
- `handler_name` - Triggered when: description

## Files & Templates
- `template.j2` - Purpose and usage

## Testing
\`\`\`bash
# How to test this role
ansible-playbook -i inventory/prod.ini playbooks/test-role.yml --check
\`\`\`

## Maintenance Notes
Special considerations for updates/patches

## Author Information
Team name - contact@email.com
````

Priority Roles to Document First:

1. `base-setup` - Used by all hosts
2. `postgresql` - Critical data layer
3. `redis` - Critical cache layer
4. `orchestrator_*` - Core application logic
5. `ollama` - LLM infrastructure

5. Missing Rollback Procedures ⚠️⚠️

Current State:
- Comprehensive recovery for MCP (`recover-mcp-server.yml`)
- **No general rollback playbooks**
- No documented rollback procedures

Risk:
- Failed deployments leave system in broken state
- No easy way to revert to previous working version
- Manual recovery required

Recommendation:

Create rollback playbooks:

```yaml
# playbooks/rollback-orchestrator.yml
---
- name: Rollback orchestrator to previous version
  hosts: orchestrator_nodes
  become: yes
  vars_prompt:
    - name: rollback_version
      prompt: "Enter version to rollback to"
      private: no

  pre_tasks:
    - name: Verify rollback version exists
      stat:
        path: "/opt/hx-citadel-shield/backups/{{ rollback_version }}"
      register: backup_check
      failed_when: not backup_check.stat.exists

  tasks:
    - name: Stop current services
      systemd:
        name: "{{ item }}"
        state: stopped
      loop: "{{ orchestrator_services }}"

    - name: Restore previous version
      copy:
        src: "/opt/hx-citadel-shield/backups/{{ rollback_version }}/"
        dest: "/opt/hx-citadel-shield/current/"
        remote_src: yes

    - name: Start services
      systemd:
        name: "{{ item }}"
        state: started
      loop: "{{ orchestrator_services }}"

  post_tasks:
    - name: Verify services are healthy
      uri:
        url: "http://localhost:8000/health"
        status_code: 200
      retries: 3
      delay: 5
```

Backup Strategy to Implement:
1. Pre-deployment backups
2. Version tagging
3. Backup retention policy
4. Backup verification

6. No CI/CD Workflow for Playbooks ⚠️⚠️

Current State:
- Tests exist (see TESTS_ANALYSIS.md)
- Type checking configured
- **No GitHub Actions workflow for playbook testing**

Missing:
- Syntax validation (`ansible-playbook --syntax-check`)

- Lint checks (`ansible-lint`)
- Dry run testing (`--check` mode)
- Integration tests

Recommendation:

Create `.github/workflows/ansible-ci.yml`:

```yaml
name: Ansible CI

on:
  pull_request:
    paths:
      - '**.yml'
      - '**.yaml'
      - 'roles/**'
      - 'playbooks/**'
  push:
    branches:
      - main

jobs:
  lint:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.12'

      - name: Install dependencies
        run: |
          pip install ansible ansible-lint yamllint
          ansible-galaxy install -r requirements.yml

      - name: Run yamllint
        run: yamllint .

      - name: Run ansible-lint
        run: ansible-lint

      - name: Syntax check all playbooks
        run: |
          for playbook in playbooks/*.yml; do
            ansible-playbook --syntax-check "$playbook"
          done

      - name: Run check-fqdn.sh
        run: ./scripts/check-fqdn.sh

      - name: Type validation
        run: ./scripts/validate-types.sh

  test:
    runs-on: ubuntu-latest
    needs: lint
    steps:
      - uses: actions/checkout@v3

      - name: Run unit tests
        run: |
          pip install -r requirements-dev.txt
          pytest tests/unit/ -v

      - name: Run integration tests (if safe)
        run: |
          # Dry run on test inventory
          ansible-playbook site.yml -i inventory/test.ini --check
```

7. Inventory Environment Separation ⚠️

Current State:
- `inventory/prod.ini` - Production inventory
- `inventory/hx-qwui.ini` - Appears to be a single-host inventory

Missing:
- Development inventory
- Staging inventory
- Testing inventory

Risk:
- Accidental production deployments during testing
- Can't test changes in isolated environment

Recommendation:

Create environment-specific inventories:

```
inventory/
├── prod.ini           # Production
├── staging.ini        # Staging environment
├── dev.ini           # Development
├── test.ini          # Testing/CI
└── group_vars/
    ├── prod/         # Prod-specific vars
    ├── staging/      # Staging-specific vars
    └── dev/          # Dev-specific vars
```

Usage:

```
# Development
ansible-playbook site.yml -i inventory/dev.ini

# Staging (with vault)
ansible-playbook site.yml -i inventory/staging.ini --vault-password-file ~/.vault_pass
_staging

# Production (requires explicit confirmation)
ansible-playbook site.yml -i inventory/prod.ini --vault-password-file ~/.vault_pass_pr
od
```

8. No Monitoring/Alerting Configuration ⚠️

Current State:
- `monitoring_nodes` group exists
- `hx-metrics-server` host defined
- `observability` role exists
- **No visible monitoring playbook in main deployment**

Missing:
- Prometheus/Grafana deployment
- Alert definitions
- Service health checks
- Metrics collection configuration

Recommendation:

1. **Add monitoring playbook to site.yml:**

```
# site.yml
---
- import_playbook: playbooks/deploy-base.yml
- import_playbook: playbooks/deploy-monitoring.yml  # Add this
- import_playbook: playbooks/deploy-db.yml
# ... rest
```

1. **Create comprehensive monitoring:**

```
# playbooks/deploy-monitoring.yml
---
- name: Deploy monitoring stack
  hosts: monitoring_nodes
  become: yes
  roles:
    - prometheus
    - grafana
    - alertmanager
    - node_exporter  # For all hosts
```

1. **Define health checks:**

```
# Integrate with smoke-tests.yml
- name: Register services with health checks
  uri:
    url: "http://{{ monitoring_server }}/api/v1/targets"
    method: POST
    body_format: json
    body:
      targets:
        - "{{ inventory_hostname }}:9090"  # Prometheus
        - "{{ inventory_hostname }}:5432"  # PostgreSQL
        - "{{ inventory_hostname }}:6379"  # Redis
```

9. Template Version Control ⚠️

Issue: Templates in roles don’t have version indicators

Example:

```
roles/base-setup/templates/health-check.sh.j2
roles/base-setup/templates/manage-services.sh.j2
```

Problem:
- Hard to track template changes
- No way to know which version is deployed
- Difficult to debug template-related issues

Recommendation:

Add version headers to templates:

```
#!/bin/bash
# Template: health-check.sh.j2
# Version: 1.2.0
# Last Modified: 2025-10-12
# Description: Health check script for HX-Citadel services
# Changes:
#   1.2.0 - Added circuit breaker status check
#   1.1.0 - Added Redis connectivity test

# Generated by Ansible on {{ ansible_date_time.date }}
# Host: {{ inventory_hostname }}
# Managed: true - DO NOT EDIT MANUALLY
```

10. No Dependency Documentation ⚠️

Issue: Role dependencies not clearly documented

Current: Some roles clearly depend on others, but it’s not documented

Example:
- `orchestrator_postgresql` requires `base-setup` and `postgresql`
- `orchestrator_redis` requires `redis`
- Execution order matters but isn’t enforced

Recommendation:

Use `meta/main.yml` in roles:

```yaml
# roles/orchestrator_postgresql/meta/main.yml
---
dependencies:
  - role: base-setup
  - role: postgresql
    vars:
      postgresql_ensure_running: yes

galaxy_info:
  author: HX-Citadel Team
  description: PostgreSQL integration for orchestrator
  license: MIT
  min_ansible_version: 2.14
  platforms:
    - name: Ubuntu
      versions:
        - jammy
        - noble
```

Critical Recommendations

Priority 1: Security (Immediate Action Required)

Issue	Risk Level	Action
Disabled host key checking	🔴 CRITICAL	Enable immediately or document risk acceptance
Redis without auth	🔴 CRITICAL	Enable `requirepass` via vault
PostgreSQL on 0.0.0.0	🟡 HIGH	Restrict to specific interfaces
StrictHostKeyChecking=no	🔴 CRITICAL	Remove from inventory
Vault file status unclear	🟡 HIGH	Investigate and document

Priority 2: Documentation (High Priority)

Task	Effort	Impact
Create role README files	High	High
Document vault procedures	Low	Critical
Create rollback playbooks	Medium	High
Add inline comments to complex playbooks	Low	Medium

Priority 3: Operational Excellence (Medium Priority)

Task	Effort	Impact
Implement CI/CD for playbooks	Medium	High
Create environment-specific inventories	Low	High
Add monitoring deployment	High	High
Version control for templates	Low	Medium
Standardize SSH keys	Low	Medium

Priority 4: Enhancement (Lower Priority)

Task	Effort	Impact
Pin external role versions	Low	Medium
Add pre-task validation	Medium	Medium
Create change log system	Low	Low
Implement backup automation	High	High

Security Considerations

1. Secrets Management

Current State:

- Vault configured: `~/.ansible_vault_pass`
- `vault.yml` exists but status unclear
- Some passwords commented out (Redis)

Best Practices:

A. Vault Structure

```yaml
# group_vars/all/vault.yml (encrypted)
---
# Database Credentials
vault_db_password: "super_secure_password_here"
vault_db_admin_password: "another_secure_password"

# Redis Credentials
vault_redis_password: "redis_secure_password"

# API Keys
vault_linear_api_token: "lin_api_xxxxx"
vault_slack_webhook: "https://hooks.slack.com/services/xxxxx"

# SSH Keys (if needed)
vault_deploy_ssh_key: |
  -----BEGIN OPENSSH PRIVATE KEY-----
  ...
  -----END OPENSSH PRIVATE KEY-----
```

B. Using Vault Variables

```
# In roles/redis/tasks/main.yml
- name: Configure Redis password
  lineinfile:
    path: /etc/redis/redis.conf
    regexp: '^# requirepass'
    line: "requirepass {{ vault_redis_password }}"
  no_log: true  # Prevent password from appearing in logs
  notify: restart redis
```

C. Vault Password Management

```
# Development
export ANSIBLE_VAULT_PASSWORD_FILE=~/.ansible_vault_pass_dev

# Production
export ANSIBLE_VAULT_PASSWORD_FILE=~/.ansible_vault_pass_prod

# Or use password prompt
ansible-playbook site.yml --ask-vault-pass
```

2. SSH Key Management

Recommendations:

1. **Use ed25519 keys (modern, secure):**

```
ssh-keygen -t ed25519 -C "ansible-deploy@hx-citadel"
```

1. **Separate keys per environment:**

```
~/.ssh/
├── id_ed25519_dev      # Development
├── id_ed25519_staging  # Staging
└── id_ed25519_prod     # Production (restricted access)
```

1. **Key rotation schedule:**
   - Development: Every 6 months
   - Staging: Every 3 months
   - Production: Every 3 months or after personnel changes

3. Network Security

Current Issues:

```
postgresql_listen_address: "0.0.0.0"  # Too permissive
```

Recommendations:

A. Restrict PostgreSQL

```
# group_vars/db_nodes.yml
postgresql_listen_address: "127.0.0.1,{{ ansible_default_ipv4.address }}"

# Or use specific FQDN
postgresql_listen_address: "127.0.0.1,{{ hx_hosts_ip[hx_hosts_fqdn['hx-sqldb-server']] }}"
```

B. Implement Firewall Rules

```
# In base-setup role
- name: Allow PostgreSQL from specific hosts only
  ufw:
    rule: allow
    from_ip: "{{ item }}"
    to_port: '5432'
    proto: tcp
  loop:
    - "{{ hx_hosts_ip[hx_hosts_fqdn['hx-orchestrator-server']] }}"
    - "{{ hx_hosts_ip[hx_hosts_fqdn['hx-api-server']] }}"
```

C. Redis Security

```
# Configure Redis to bind specific interfaces
redis_bind_interface: "127.0.0.1 {{ ansible_default_ipv4.address }}"

# Enable authentication
redis_requirepass: "{{ vault_redis_password }}"

# Disable dangerous commands
redis_rename_commands:
  - { command: "CONFIG", rename: "CONFIG_{{ 99999 | random }}" }
  - { command: "FLUSHALL", rename: "FLUSHALL_{{ 99999 | random }}" }
```

4. Privilege Escalation

Current State:

- `become: yes` used appropriately
- Sudo configuration templated
- Unix socket auth for PostgreSQL

Recommendations:

A. Principle of Least Privilege

```
# Don't do this
become: yes  # Root for everything

# Do this instead
become: yes
become_user: postgres  # Only escalate to needed user
```

B. Sudo Configuration

```
# templates/sudoers.j2
# Allow app user to manage specific services only
{{ app_user }} ALL=(root) NOPASSWD: /bin/systemctl start hx-*
{{ app_user }} ALL=(root) NOPASSWD: /bin/systemctl stop hx-*
{{ app_user }} ALL=(root) NOPASSWD: /bin/systemctl restart hx-*
{{ app_user }} ALL=(root) NOPASSWD: /bin/systemctl status hx-*

# Deny everything else
{{ app_user }} ALL=!/bin/bash, !/bin/sh
```

5. Audit Logging

Recommendation:

Add audit logging to sensitive operations:

```yaml
- name: Modify production database
  postgresql_db:
    name: "{{ db_name }}"
    state: present
  register: db_change
  become: yes
  become_user: postgres

- name: Log database change
  lineinfile:
    path: "/var/log/ansible-audit.log"
    line: "{{ ansible_date_time.iso8601 }} | {{ ansible_user_id }} | {{ inventory_hostname }} | DATABASE_MODIFIED | {{ db_change }}"
    create: yes
  delegate_to: localhost
  when: db_change.changed
```

Operational Excellence

1. Deployment Workflow

Recommended Process:

```
# 1. Pre-flight checks
ansible-playbook playbooks/preflight-check.yml -i inventory/prod.ini

# 2. Dry run
ansible-playbook site.yml -i inventory/prod.ini --check --diff

# 3. Limited deployment (test one host)
ansible-playbook site.yml -i inventory/prod.ini --limit hx-test-server

# 4. Gradual rollout
ansible-playbook site.yml -i inventory/prod.ini --limit db_nodes
ansible-playbook site.yml -i inventory/prod.ini --limit vector_nodes
# ... etc

# 5. Full deployment
ansible-playbook site.yml -i inventory/prod.ini

# 6. Post-deployment validation
ansible-playbook playbooks/smoke-tests.yml -i inventory/prod.ini

# 7. Notify team
./scripts/slack-notify.sh "Deployment completed successfully"
```

2. Change Management

Create deployment runbook:

```
# Deployment Runbook

## Pre-Deployment
- [ ] Review changes in PR
- [ ] Run tests locally
- [ ] Backup current configuration
- [ ] Notify team in Slack #deployments
- [ ] Schedule maintenance window if needed

## Deployment Steps
1. [ ] Run preflight checks
2. [ ] Execute dry run
3. [ ] Deploy to test environment
4. [ ] Verify test deployment
5. [ ] Deploy to production
6. [ ] Run smoke tests
7. [ ] Monitor for 15 minutes

## Post-Deployment
- [ ] Verify all services healthy
- [ ] Check logs for errors
- [ ] Update documentation
- [ ] Close Linear issue
- [ ] Notify team of completion

## Rollback Procedure
If deployment fails:
1. [ ] Run rollback playbook
2. [ ] Verify rollback successful
3. [ ] Investigate failure
4. [ ] Document in Linear
```

3. Monitoring Integration

Recommended additions to roles:

```yaml
# In each service role
- name: Register service with monitoring
  uri:
    url: "http://{{ monitoring_server }}/api/register"
    method: POST
    body_format: json
    body:
      service_name: "{{ service_name }}"
      host: "{{ inventory_hostname }}"
      port: "{{ service_port }}"
      health_check_url: "http://{{ inventory_hostname }}:{{ service_port }}/health"
  delegate_to: localhost
  when: monitoring_enabled | default(true)
```

4. Performance Optimization

Current optimizations:

- ✅ SSH multiplexing
- ✅ Pipelining
- ✅ Fact caching

Additional recommendations:

A. Parallel Execution

```ini
# ansible.cfg
[defaults]
forks = 10  # Increase from default 5 for faster execution
```

B. Selective Fact Gathering

```yaml
# For playbooks that don't need all facts
- name: Quick deployment
  hosts: all
  gather_facts: no  # Skip if not needed
  tasks:
    - name: Gather minimal facts
      setup:
        gather_subset:
          - '!all'
          - 'network'
      when: network_info_needed
```

C. Async Tasks

```yaml
# For long-running operations
- name: Download large model
  get_url:
    url: "{{ model_url }}"
    dest: "/opt/models/"
  async: 3600  # 1 hour timeout
  poll: 0      # Don't wait
  register: download_job

- name: Wait for download
  async_status:
    jid: "{{ download_job.ansible_job_id }}"
  register: job_result
  until: job_result.finished
  retries: 360
  delay: 10
```

5. Disaster Recovery

Create DR playbooks:

```yaml
# playbooks/disaster-recovery.yml
---
- name: Full system recovery
  hosts: all
  become: yes
  serial: 1  # One host at a time

  pre_tasks:
    - name: Verify backup exists
      stat:
        path: "/backups/{{ recovery_date }}"
      register: backup_check
      failed_when: not backup_check.stat.exists

  roles:
    - restore_system
    - restore_database
    - restore_application

  post_tasks:
    - name: Verify services
      systemd:
        name: "{{ item }}"
        state: started
      loop: "{{ critical_services }}"

    - name: Run smoke tests
      import_tasks: ../playbooks/smoke-tests.yml
```

6. Documentation Standards

Maintain these documents:

1. **README.md** (Root) ✅ Already exists
2. **CONTRIBUTING.md** (How to contribute)
3. **ARCHITECTURE.md** (System architecture)

4. **RUNBOOKS/** (Operational procedures)
   - deployment.md
   - rollback.md
   - disaster-recovery.md
   - troubleshooting.md
5. **CHANGELOG.md** (Version history)
6. **Role README files** (Per role documentation)

Conclusion

Overall Assessment: 8.5/10

The HX-Citadel Ansible infrastructure is **well-architected and production-ready** with strong adherence to best practices. The FQDN management system, modular role design, and comprehensive testing framework are exemplary.

Critical Actions Required

1. **Security (Within 1 week):**
   - ✅ Enable host key checking
   - ✅ Configure Redis authentication
   - ✅ Restrict PostgreSQL listening
   - ✅ Investigate vault.yml.broken

2. **Documentation (Within 2 weeks):**
   - Create role README files (start with critical roles)
   - Document vault recovery procedures
   - Create deployment runbooks

3. **Operational (Within 1 month):**
   - Implement CI/CD for playbooks
   - Create rollback procedures
   - Add monitoring deployment
   - Standardize SSH keys

Final Thoughts

This infrastructure demonstrates **strong engineering discipline** and thoughtful design. The identified improvements are primarily about:
- Hardening security (from "good" to "excellent")
- Improving documentation (for team growth)
- Adding operational safeguards (rollback, DR)
- Enforcing consistency (SSH keys, environments)

With the recommended changes, this would easily be a **9.5/10** infrastructure.

Next Steps

1. **Share this analysis** with the team
2. **Prioritize recommendations** based on your context
3. **Create Linear issues** for tracking improvements
4. **Schedule security review** for critical items

5. **Implement incrementally** - don’t try to fix everything at once

---

Appendix: Quick Reference

Common Commands

```
# Syntax check
ansible-playbook playbooks/deploy-base.yml --syntax-check

# Dry run
ansible-playbook site.yml -i inventory/prod.ini --check --diff

# Run specific tags
ansible-playbook site.yml -i inventory/prod.ini --tags "db,vector"

# Limit to specific hosts
ansible-playbook site.yml -i inventory/prod.ini --limit "llm_nodes"

# Vault operations
ansible-vault edit group_vars/all/vault.yml
ansible-vault view group_vars/all/vault.yml
ansible-vault rekey group_vars/all/vault.yml

# List hosts
ansible-inventory -i inventory/prod.ini --list
ansible-inventory -i inventory/prod.ini --graph

# Ad-hoc commands
ansible all -i inventory/prod.ini -m ping
ansible db_nodes -i inventory/prod.ini -m shell -a "systemctl status postgresql"
```

Directory Structure Reference

```
hx-citadel-ansible/
├── ansible.cfg              # Main configuration
├── site.yml                 # Master playbook
├── requirements.yml         # External dependencies
├── inventory/               # Host definitions
│   └── prod.ini
├── group_vars/              # Group variables
│   ├── all/
│   │   ├── main.yml
│   │   ├── fqdn_map.yml     # ⭐ FQDN mappings
│   │   └── vault.yml         # Encrypted secrets
│   ├── db_nodes.yml
│   └── llm_nodes.yml
├── host_vars/               # Host-specific variables
├── roles/                   # Ansible roles (24 custom)
├── playbooks/               # Playbooks (26 total)
├── scripts/                 # Utility scripts (9 scripts)
├── tests/                   # Test suite
└── docs/                    # Documentation
```

Analysis Complete

For questions or clarifications, refer to the project README.md or contact the infrastructure team.