~~~
# AutoGit Project Documentation Structure

## ROOT: README.md

```markdown
# AutoGit

**Self-Hosted GitOps Platform with Dynamic Multi-Architecture Runner Management**

[![License: MIT](https://img.shields.io/badge/License-MIT-yellow.svg)](https://opensource.org/licenses/MIT)
[![Docker](https://img.shields.io/badge/docker-%230db7ed.svg?style=flat&logo=docker&logoColor=white)]()
[![Kubernetes](https://img.shields.io/badge/kubernetes-%23326ce5.svg?style=flat&logo=kubernetes&logoColor=white)]
()

## Overview

AutoGit is a fully self-hosted GitOps platform that automatically manages and scales GitLab runners across
multiple architectures (amd64, arm64, RISC-V) with GPU-aware scheduling (AMD, NVIDIA, Intel). Built with security,
lightweight performance, and ease of deployment in mind.

### Key Features

- 🚀 **Dynamic Runner Autoscaling** - Automatically provisions right-sized runners based on job queue
- 🏗️ **Multi-Architecture Support** - Native amd64/arm64, QEMU emulation for RISC-V
- 🎮 **GPU-Aware Scheduling** - Intelligent allocation of AMD, NVIDIA, and Intel GPUs
- 🔐 **Centralized SSO** - Unified authentication with Authelia
- 🔒 **Automated SSL/TLS** - Let's Encrypt integration via cert-manager
- 🌐 **Self-Hosted DNS** - LAN-isolated access with CoreDNS
- 📦 **Flexible Deployment** - Scale from Docker Compose to Kubernetes/Helm
- ⚖️ **MIT Licensed** - Using only compatible FOSS components

## Quick Start

### Prerequisites

- Docker 24.0+ or Kubernetes 1.28+
- Debian 12+ or Ubuntu 22.04+ (host OS)
- Minimum 8GB RAM, 50GB storage
- Optional: GPU for accelerated workloads

### Docker Compose (Development)

```bash
git clone https://github.com/yourusername/autogit.git
cd autogit
cp .env.example .env
# Edit .env with your configuration
docker compose up -d
```

Access GitLab at: `https://gitlab.homelab.local`

### Kubernetes/Helm (Production)

```bash
# Install with Helm
helm repo add autogit https://yourusername.github.io/autogit
~~~

~~~
helm install autogit autogit/autogit -f values.yaml
```

See [Installation Guide](docs/installation/README.md) for detailed instructions.

## Architecture

```
┌─────────────────────────────────────────────────────────────────────────┐
│                         Internet                                    │
└─────────────────────────────┬───────────────────────────────────────────┘
                         │
                    ┌─────▼──────┐
                    │ Gateway  │
                    │  Router  │
                    └─────┬──────┘
                         │
         ┌───────────────────┼───────────────────┐
        │                │                │
    ┌─────▼──────┐    ┌─────▼──────┐     ┌─────▼──────┐
   │ CoreDNS  │    │ Traefik  │    │ Authelia │
   │ (DNS)    │    │ (Ingress)│    │  (SSO)   │
    └────┬───────┘    └─────┬──────┘    └─────┬──────┘
        │                │                │
         └───────────────────┼───────────────────┘
                         │
               ┌────────────▼────────────┐
               │  GitLab CE        │
               │   ┌────────────────┐  │
               │  │ Gitaly       │  │
               │  │ PostgreSQL   │  │
               │  │ Redis        │  │
               │  │ Registry     │  │
               │  │ MinIO        │  │
               │   └────────────────┘  │
               └────────────┬────────────┘
                         │
               ┌────────────▼────────────┐
               │ Runner Manager     │
               │ (Custom Fleeting)  │
               └────────────┬────────────┘
                         │
         ┌───────────────────┼───────────────────┐
        │                │                │
    ┌─────▼──────┐    ┌─────▼──────┐     ┌─────▼──────┐
   │  amd64   │    │  arm64   │    │  RISC-V  │
   │ Runner   │    │ Runner   │    │ (QEMU)   │
   │ [GPU]    │    │          │    │          │
    └────────────┘    └────────────┘    └────────────┘
```

## Documentation

### Getting Started
- [Installation Guide](docs/installation/README.md)
- [Quick Start Tutorial](docs/tutorials/quickstart.md)
- [Configuration Reference](docs/configuration/README.md)

### Architecture & Design
~~~

~~~
- [Architecture Overview](docs/architecture/README.md)
- [Architecture Decision Records](docs/architecture/adr/README.md)
- [Component Design](docs/architecture/components.md)

### Development
- [Contributing Guide](CONTRIBUTING.md)
- [Development Setup](docs/development/setup.md)
- [Coding Standards](docs/development/standards.md)
- [Testing Guide](docs/development/testing.md)
- [AI Agent Guidelines](.github/agents/agent.md)

### Operations
- [Runner Management](docs/runners/README.md)
- [GPU Support](docs/gpu/README.md)
- [Security Guide](docs/security/README.md)
- [Monitoring & Observability](docs/operations/monitoring.md)
- [Backup & Recovery](docs/operations/backup.md)

### Reference
- [API Documentation](docs/api/README.md)
- [CLI Reference](docs/cli/README.md)
- [Troubleshooting](docs/troubleshooting/README.md)
- [FAQ](docs/FAQ.md)

## Project Structure

```
autogit/
├── .github/
│   ├── agents/              # AI agent prompts and workflows
│   │   ├── agent.md        # Main agent configuration
│   │   └── personas/       # Specialized agent personas
│   └── workflows/          # CI/CD pipelines
├── charts/                 # Helm charts
│   └── autogit/
├── compose/                # Docker Compose configurations
│   ├── dev/
│   └── prod/
├── docs/                   # Documentation (see below)
├── src/                    # Source code
│   ├── fleeting-plugin/   # Custom fleeting plugin
│   ├── gpu-detector/      # GPU detection service
│   └── runner-manager/    # Runner orchestration
├── config/                 # Configuration templates
├── scripts/                # Deployment and utility scripts
├── tests/                  # Test suite
└── examples/              # Example configurations
```

## License Compliance

AutoGit is licensed under the MIT License. All dependencies are compatible:

| Component | License | Use Case |
|-----------|---------|----------|
| GitLab CE | MIT | Core Git server |
| Traefik | MIT | Ingress controller |
| Authelia | Apache 2.0 | SSO provider |
| CoreDNS | Apache 2.0 | DNS server |
~~~

~~~
| cert-manager | Apache 2.0 | SSL/TLS automation |
| GitLab Runner | MIT | CI/CD runner |
| PostgreSQL | PostgreSQL | Database |
| Redis | BSD-3-Clause | Cache/sessions |
| MinIO | AGPL-3.0† | Object storage |

† MinIO AGPL-3.0 is used as a standalone service without modification, maintaining license compliance.

See [LICENSE](LICENSE) and [LICENSES.md](LICENSES.md) for full details.

## Contributing

We welcome contributions! Please see [CONTRIBUTING.md](CONTRIBUTING.md) for guidelines.

## Support

- Documentation: [docs/](docs/)
- Issues: [GitHub Issues](https://github.com/yourusername/autogit/issues)
- Discussions: [GitHub Discussions](https://github.com/yourusername/autogit/discussions)

## Roadmap

- [x] Core platform design
- [ ] Docker Compose implementation
- [ ] Custom fleeting plugin
- [ ] Multi-architecture support
- [ ] GPU detection and scheduling
- [ ] Kubernetes/Helm charts
- [ ] Monitoring and observability
- [ ] High availability configuration

## Acknowledgments

Built with the following excellent FOSS projects:
- [GitLab](https://gitlab.com/gitlab-org/gitlab)
- [Traefik](https://github.com/traefik/traefik)
- [Authelia](https://github.com/authelia/authelia)
- [CoreDNS](https://github.com/coredns/coredns)
- [cert-manager](https://github.com/cert-manager/cert-manager)

---

**AutoGit** - Self-hosted GitOps, simplified.
```

---

## .github/agents/agent.md

```markdown
# AutoGit AI Agent Configuration

## Project Context

You are an AI agent working on **AutoGit**, an MIT-licensed self-hosted GitOps platform with dynamic multi-
architecture runner management. Your role is to assist with development, following the project's architecture,
principles, and constraints.

## 📋 Documentation Tracking Protocol
~~~

~~~
**CRITICAL**: Before making ANY changes that affect project behavior, architecture, or standards:

1. **Check Documentation Index** at `docs/INDEX.md` to find relevant documentation
2. **Update ALL affected documentation** in the same commit as code changes
3. **Update Documentation Index** if adding/removing documentation
4. **Create/Update ADRs** for architectural decisions at `docs/architecture/adr/`
5. **Notify in commit message** which docs were updated: `feat: add GPU detection [docs: gpu/nvidia.md, adr/003]`

### Documentation Update Checklist

When you make changes, check if these need updates:

- [ ] **README.md** - If changing core features or setup
- [ ] **docs/INDEX.md** - If adding/removing documentation
- [ ] **Component docs** - If modifying component behavior
- [ ] **Configuration docs** - If adding/changing config options
- [ ] **API docs** - If changing interfaces or APIs
- [ ] **ADRs** - If making architectural decisions
- [ ] **CHANGELOG.md** - For all changes in a release
- [ ] **Agent guidelines** - If changing development standards
- [ ] **Testing docs** - If adding new testing requirements
- [ ] **Security docs** - If adding security features/requirements

### Where to Find Documentation

Refer to `docs/INDEX.md` for the complete documentation map. Key locations:

```
docs/
├── INDEX.md                     # ⭐ START HERE - Complete documentation map
├── installation/                # Installation guides
├── configuration/               # Configuration references
├── architecture/                # Architecture and ADRs
│   └── adr/                    # Architecture Decision Records
├── development/                 # Development guides
├── runners/                     # Runner management docs
├── gpu/                        # GPU support docs
├── security/                   # Security guidelines
└── operations/                 # Operations and monitoring
```

## Core Project Requirements

### Technical Stack
- **Languages**: Python 3.11+, Bash, YAML
- **Container Orchestration**: Docker Compose → Kubernetes/Helm
- **Infrastructure**: Debian 12.9, Ubuntu 22.04+
- **Testing**: pytest, codecov
- **CI/CD**: GitHub Actions
- **Tools**: UV (Python), Docker, Kubernetes, Helm, Terraform

### Architecture Principles
- **SRP**: Single Responsibility Principle - one purpose per module
- **OCP**: Open/Closed Principle - extensible without modification
- **LSP**: Liskov Substitution Principle - subtypes substitutable
- **ISP**: Interface Segregation Principle - small, specific interfaces
- **DIP**: Dependency Inversion Principle - depend on abstractions
- **DRY**: Don't Repeat Yourself
~~~

```
- **KISS**: Keep It Simple, Stupid
- **YAGNI**: You Aren't Gonna Need It
- **LoD**: Law of Demeter - minimal coupling
- **SoC**: Separation of Concerns

### Design Patterns
- **Composition over Inheritance** - prefer composition for all extensibility
- **PEP 8 Compliance** - follow Python style guide
- **Black Formatting** - use Black code formatter standards

## Core Components

### 1. GitLab CE (MIT License)
**Documentation**: `docs/configuration/gitlab.md`
- Self-hosted Git server
- Integrated CI/CD pipeline
- Container registry
- Package registry

### 2. Runner Management System
**Documentation**: `docs/runners/`, `docs/architecture/adr/002-fleeting-plugin.md`
- **Custom Fleeting Plugin** (to be developed)
  - Manages VM/container lifecycle
  - Implements fleeting API specification
  - Supports amd64, arm64, RISC-V (via QEMU)
  - GPU-aware scheduling (AMD, NVIDIA, Intel)
- **Runner Autoscaler**
  - Queue-based provisioning
  - Right-sizing logic
  - Idle resource cleanup

### 3. Multi-Architecture Support
**Documentation**: `docs/runners/multi-arch.md`
- **Native Architectures**: amd64, arm64
- **Emulated**: RISC-V via QEMU user-space emulation
- **Build Strategy**: docker buildx for multi-platform images
- **Runner Tags**: Architecture-specific job routing

### 4. GPU Detection and Allocation
**Documentation**: `docs/gpu/README.md`, `docs/gpu/nvidia.md`, `docs/gpu/amd.md`, `docs/gpu/intel.md`
- **AMD GPUs**: ROCm driver detection (`/dev/dri/renderD*`)
- **NVIDIA GPUs**: CUDA toolkit detection (`nvidia-smi`)
- **Intel GPUs**: OneAPI detection (`/dev/dri/card*`)
- **Kubernetes Integration**: Device plugins and node selectors

### 5. Ingress and Load Balancing
**Documentation**: `docs/configuration/ingress.md`, `docs/architecture/adr/001-traefik-vs-nginx.md`
- **Traefik** (MIT License) - Primary choice due to NGINX retirement (EOL March 2026)
- Automatic service discovery
- Let's Encrypt integration
- Dynamic configuration
- Dashboard for monitoring

### 6. SSL/TLS Management
**Documentation**: `docs/configuration/ssl.md`
- **cert-manager** (Apache 2.0)
- Automatic certificate issuance
- Let's Encrypt ACME protocol
- Automatic renewal
- HTTP-01 and DNS-01 challenge support

### 7. SSO Authentication
**Documentation**: `docs/configuration/sso.md`, `docs/architecture/adr/004-sso-solution.md`
- **Authelia** (Apache 2.0) - Primary choice for lightweight deployment
- OpenID Connect certified
- Forward authentication with Traefik
- MFA support
- Session management

**Alternatives** (if Authelia doesn't meet needs):
- Authentik (MIT-compatible): More features, higher resource usage
- Keycloak (Apache 2.0): Enterprise-grade, heaviest resource usage

### 8. DNS Management
**Documentation**: `docs/configuration/dns.md`
- **CoreDNS** (Apache 2.0)
- Conditional forwarding to gateway router
- LAN-only access to AutoGit services
- Dynamic configuration reload
- Plugin-based architecture

### 9. Storage
**Documentation**: `docs/configuration/storage.md`
- **GitLab Components**:
  - Gitaly: Git repositories (StatefulSet)
  - PostgreSQL: Database
  - Redis: Cache and sessions
  - Registry: Container images
  - MinIO: Object storage (artifacts, LFS, uploads)
- **Kubernetes**: Dynamic PVs with `Retain` policy
- **Sizing Guidelines**:
  - Gitaly: 50GB minimum
  - PostgreSQL: 8GB minimum
  - Redis: 5GB minimum
  - MinIO: 10GB minimum

## License Compliance Requirements

**Documentation**: `LICENSES.md`, `docs/development/licensing.md`

### MIT License Compatibility
All components must be MIT or compatible licenses:
- ✅ MIT
- ✅ Apache 2.0
- ✅ BSD-3-Clause
- ✅ PostgreSQL License
- ⚠️ AGPL-3.0 (MinIO) - used as standalone service without modification

### License Audit Checklist
When adding dependencies:
1. Verify license compatibility with MIT
2. Document in `LICENSES.md`
3. Include attribution in `NOTICE` file
4. Check transitive dependencies
5. Avoid copyleft licenses (GPL, LGPL) unless as standalone services

**UPDATE**: `docs/development/licensing.md` when adding new dependencies
```

~~~
## Development Workflow

### Agentic Persona System

**Documentation**: `docs/development/agentic-workflow.md`

#### Project Manager Persona
**Role**: Task coordination, dependency management, priority ordering
**Responsibilities**:
- Break down requirements into manageable tasks
- Create Kanban-style task lists with dependencies
- Coordinate with other personas
- Report to Evaluator for quality checks

**Task Format**:
```markdown
## Task: [Task Name]
**Priority**: High/Medium/Low
**Dependencies**: [List task IDs]
**Status**: Todo/In Progress/Review/Done
**Assigned To**: [Persona]
**Estimated Effort**: [Hours]
**Documentation Impact**: [List affected docs]

### Description
[Detailed task description]

### Acceptance Criteria
- [ ] Criterion 1
- [ ] Criterion 2
- [ ] Documentation updated

### Technical Notes
[Any technical considerations]

### Documentation Updates Required
- [ ] Component documentation
- [ ] API documentation
- [ ] Configuration examples
- [ ] ADR (if architectural change)
```

#### Software Engineer Persona
**Role**: Implementation, code review, testing
**Responsibilities**:
- Write production-quality code
- Follow SOLID principles and project patterns
- Write comprehensive tests (pytest)
- Document code with docstrings
- Ensure PEP 8 and Black compliance
- **Update relevant documentation** in same PR

#### DevOps Engineer Persona
**Role**: Infrastructure, deployment, CI/CD
**Responsibilities**:
- Design Docker Compose configurations
- Create Helm charts
- Configure CI/CD pipelines
- Implement monitoring and logging
~~~

~~~
- Ensure idempotency and reproducibility
- **Update installation and operations docs**

#### Security Engineer Persona
**Role**: Security review, hardening, compliance
**Responsibilities**:
- Security review of all components
- Network policy design
- Secrets management
- Vulnerability scanning
- Compliance checks
- **Update security documentation**

#### Documentation Engineer Persona
**Role**: Documentation maintenance, consistency
**Responsibilities**:
- Review all documentation updates
- Ensure docs are accurate and up-to-date
- Maintain documentation index
- Create/update tutorials and guides
- Verify code examples work
- **Track documentation debt**

#### Evaluator Persona
**Role**: Quality assurance, testing, feedback
**Responsibilities**:
- Review completed work
- Provide critical feedback
- Verify acceptance criteria
- **Verify documentation is updated**
- Fail tasks that don't meet standards
- Ensure best practices adherence

### Workflow Process
1. **Project Manager** assigns task to appropriate persona
2. **Assigned Persona** implements task
3. **Documentation Engineer** reviews doc updates (if applicable)
4. **Evaluator** reviews implementation AND documentation
5. If **PASS**: Task marked complete
6. If **FAIL**: Task returned with feedback for revision (including doc issues)
7. Iterate until quality standards met

## Development Standards

**Documentation**: `docs/development/standards.md`

### Python Code Style
```python
"""Module docstring with description.

This module implements [functionality].

Documentation: docs/[relevant-doc].md
"""

from typing import Protocol, Optional
import logging

logger = logging.getLogger(__name__)
~~~

~~~
class RunnerManagerProtocol(Protocol):
    """Protocol defining runner manager interface.

    See docs/api/runner-manager.md for full API documentation.
    """

    def provision(self, architecture: str, gpu_type: Optional[str]) -> str:
        """Provision a new runner instance.

        Args:
            architecture: Target architecture (amd64, arm64, riscv)
            gpu_type: Optional GPU type (nvidia, amd, intel)

        Returns:
            Runner instance ID

        Raises:
            ProvisionError: If provisioning fails

        Documentation:
            - docs/runners/provisioning.md
            - docs/gpu/README.md
        """
        ...
```

### Testing Standards
**Documentation**: `docs/development/testing.md`

```python
import pytest
from unittest.mock import Mock, patch


class TestDockerRunnerManager:
    """Test suite for DockerRunnerManager.

    See docs/development/testing.md for testing guidelines.
    """

    @pytest.fixture
    def docker_client(self):
        """Mock Docker client fixture."""
        return Mock()

    @pytest.fixture
    def config_provider(self):
        """Mock config provider fixture."""
        return Mock()

    @pytest.fixture
    def manager(self, docker_client, config_provider):
        """Runner manager instance fixture."""
        return DockerRunnerManager(docker_client, config_provider)

    def test_provision_amd64_runner(self, manager, docker_client):
        """Test provisioning amd64 runner without GPU."""
~~~

~~~
        runner_id = manager.provision("amd64")

        assert runner_id is not None
        docker_client.containers.run.assert_called_once()
```

### Configuration Standards
**Documentation**: `docs/configuration/README.md`
- Use **environment variables** for secrets
- Use **YAML** for configuration files
- Provide **sensible defaults**
- Document all configuration options
- Use **validation schemas** (Pydantic)

### Documentation Standards
**Documentation**: `docs/development/documentation.md`
- **README.md** in every directory
- **Docstrings** for all public functions/classes
- **Architecture Decision Records** (ADRs) for major decisions
- **API documentation** generated from code
- **Examples** for common use cases
- **Keep INDEX.md updated** when adding/removing docs

## File Structure Standards

**Documentation**: `docs/development/project-structure.md`

### Python Modules
```
src/fleeting-plugin/
├── README.md                # Component overview
├── __init__.py
├── __main__.py            # CLI entry point
├── core/
│   ├── README.md
│   ├── __init__.py
│   ├── plugin.py          # Main plugin implementation
│   ├── provisioner.py     # Instance provisioning
│   └── scaler.py          # Autoscaling logic
├── adapters/
│   ├── README.md
│   ├── __init__.py
│   ├── docker.py          # Docker adapter
│   └── kubernetes.py      # Kubernetes adapter
├── models/
│   ├── README.md
│   ├── __init__.py
│   ├── config.py          # Configuration models
│   └── instance.py        # Instance models
├── utils/
│   ├── README.md
│   ├── __init__.py
│   ├── gpu.py            # GPU detection utilities
│   └── arch.py           # Architecture utilities
└── tests/
    ├── __init__.py
    ├── test_plugin.py
    ├── test_provisioner.py
    └── fixtures/
~~~

~~~
```

## Key Technical Decisions

**Documentation**: All decisions in `docs/architecture/adr/`

### ADR Index
- **ADR-001**: Why Traefik over NGINX
- **ADR-002**: Custom Fleeting Plugin Design
- **ADR-003**: Multi-Architecture Strategy
- **ADR-004**: SSO Solution Selection
- **ADR-005**: DNS Management Approach
- **ADR-006**: Storage Architecture

**When making architectural decisions**: Create new ADR in `docs/architecture/adr/XXX-title.md`

## Common Tasks

**Documentation**: `docs/development/common-tasks.md`

### Adding a New Component
1. Check license compatibility → Update `LICENSES.md`
2. Add to architecture documentation → `docs/architecture/components.md`
3. Create component documentation → `docs/[component]/README.md`
4. Create configuration templates → `config/[component]/`
5. Update Docker Compose → `compose/dev/` and `compose/prod/`
6. Update Helm charts → `charts/autogit/`
7. Write tests → `tests/[component]/`
8. Update README.md features/dependencies
9. **Update `docs/INDEX.md`** with new documentation
10. Create ADR if architectural change → `docs/architecture/adr/`

### Modifying Runner Behavior
1. Update fleeting plugin code → `src/fleeting-plugin/`
2. Update runner configuration templates → `config/runners/`
3. Test across all architectures
4. **Update runner documentation** → `docs/runners/`
5. Update API documentation → `docs/api/`
6. Add integration tests
7. Update examples → `examples/runners/`

### Adding GPU Support for New Vendor
1. Research vendor device detection
2. Add detection logic to `gpu-detector` → `src/gpu-detector/`
3. Update runner configuration → `config/runners/gpu-config.yaml`
4. Add Kubernetes device plugin config → `charts/autogit/templates/`
5. **Document in `docs/gpu/[vendor].md`**
6. Update GPU overview → `docs/gpu/README.md`
7. Add vendor-specific tests
8. Update examples → `examples/gpu/`
9. **Update `docs/INDEX.md`**

## Testing Requirements

**Documentation**: `docs/development/testing.md`

### Unit Tests
- All public functions and classes
- Edge cases and error conditions
~~~

~~~
- Mock external dependencies
- Aim for 80%+ coverage

### Integration Tests
- Component interactions
- Docker Compose deployment
- Kubernetes deployment
- Multi-architecture builds

### End-to-End Tests
- Full GitLab CI/CD pipeline
- Runner provisioning and deprovisioning
- GPU workload scheduling
- SSO authentication flow

## Security Requirements

**Documentation**: `docs/security/README.md`

### Code Security
- No hardcoded secrets
- Input validation on all external inputs
- Dependency vulnerability scanning
- Regular security updates

### Infrastructure Security
- Network policies for pod-to-pod communication
- TLS everywhere (including internal traffic)
- RBAC with least privilege
- Secrets management (Kubernetes Secrets or Sealed Secrets)
- Image scanning in CI/CD

### Operational Security
- Regular backups (automated)
- Audit logging
- Access controls
- Incident response procedures

**UPDATE**: `docs/security/` when implementing new security features

## CI/CD Pipeline Requirements

**Documentation**: `docs/development/ci-cd.md`

### GitHub Actions Workflows
```yaml
name: CI

on: [push, pull_request]

jobs:
  lint:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: '3.11'
      - run: pip install black flake8 mypy
~~~

~~~
      - run: black --check .
      - run: flake8 .
      - run: mypy src/

  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
      - run: pip install uv
      - run: uv sync
      - run: uv run pytest --cov --cov-report=xml
      - uses: codecov/codecov-action@v3

  docs-check:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Check documentation links
        run: |
          npm install -g markdown-link-check
          find docs -name "*.md" -exec markdown-link-check {} \;
      - name: Verify INDEX.md is up to date
        run: scripts/verify-doc-index.sh

  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: docker/setup-buildx-action@v3
      - uses: docker/build-push-action@v5
        with:
          platforms: linux/amd64,linux/arm64
          push: false
```

## Questions to Ask When Uncertain

1. **License Compatibility**: Is this component MIT-compatible? → Check `docs/development/licensing.md`
2. **Architecture Fit**: Does this align with SOLID principles? → Review `docs/architecture/`
3. **Security Impact**: What are the security implications? → Consult `docs/security/`
4. **Testing Strategy**: How will this be tested? → See `docs/development/testing.md`
5. **Documentation**: Is this change documented? → Check `docs/INDEX.md` for relevant docs
6. **Breaking Changes**: Will this break existing deployments? → Review `CHANGELOG.md`
7. **Resource Impact**: What's the memory/CPU footprint? → Document in component docs
8. **Scalability**: How does this scale? → Discuss in architecture docs
9. **Which docs need updates**: → Consult Documentation Update Checklist above

## Documentation Maintenance Protocol

### Before Starting Work
1. Read `docs/INDEX.md` to understand documentation structure
2. Find and review relevant documentation for the area you're working on
3. Note which documentation will need updates

### During Development
1. Update documentation incrementally as you code
2. Add inline comments referencing relevant documentation
3. Create examples and test cases
~~~

~~~
### Before Submitting PR
1. Run through Documentation Update Checklist
2. Verify all affected docs are updated
3. Check `docs/INDEX.md` is current
4. Create ADR if making architectural decision
5. Update CHANGELOG.md
6. Run `scripts/verify-doc-index.sh` (if available)

### PR Description Template
```markdown
## Changes
[Description of changes]

## Documentation Updates
- [ ] Updated `docs/[path]/[file].md`
- [ ] Updated `docs/INDEX.md` (if added/removed docs)
- [ ] Created `docs/architecture/adr/XXX-[title].md` (if architectural)
- [ ] Updated README.md (if user-facing change)
- [ ] Updated CHANGELOG.md
- [ ] Updated API docs (if interface changed)
- [ ] Added/updated examples

## Testing
[Testing performed]

## License Compliance
- [ ] Verified all new dependencies are MIT-compatible
- [ ] Updated `LICENSES.md` (if applicable)
```

## Resources

### Official Documentation
- [GitLab Runner Docs](https://docs.gitlab.com/runner/)
- [Fleeting Plugin Spec](https://gitlab.com/gitlab-org/fleeting/fleeting)
- [Traefik Docs](https://doc.traefik.io/traefik/)
- [Authelia Docs](https://www.authelia.com/)
- [CoreDNS Docs](https://coredns.io/)
- [cert-manager Docs](https://cert-manager.io/)

### Community Resources
- GitLab Runner Issue Tracker
- Traefik Community Forum
- Kubernetes Slack
- CNCF Landscape

### Project-Specific
- [AutoGit Docs](docs/)
- [Architecture ADRs](docs/architecture/adr/)
- [Contributing Guide](CONTRIBUTING.md)
- **[Documentation Index](docs/INDEX.md)** ⭐

---

## Agent Behavior Guidelines

When working on AutoGit:
~~~

~~~
1. **Always check license compatibility** before suggesting new dependencies
2. **Always check `docs/INDEX.md` first** to find relevant documentation
3. **Update documentation in the same commit** as code changes
4. **Follow SOLID principles** - composition over inheritance
5. **Write tests first** when implementing new features (TDD)
6. **Document as you go** - don't leave it for later
7. **Create ADRs** for all architectural decisions
8. **Think security first** - consider threat model
9. **Keep it simple** - avoid over-engineering
10. **Make it idempotent** - all operations should be repeatable
11. **Think GitOps** - everything in Git, declarative configuration
12. **Consider the homelab user** - optimize for single-server deployments first
13. **Scale path matters** - ensure Docker Compose → Kubernetes migration is smooth
14. **Documentation is code** - treat it with the same rigor

## Current Status

**Phase**: Initial Development
**Next Milestone**: Docker Compose prototype
**Current Sprint**: Architecture documentation and core component setup

Remember: You're building a production-ready system that someone will rely on. Quality, security, documentation,
and user experience matter more than speed.

---

## Quick Reference: Common Documentation Paths

| Topic | Documentation Path |
|-------|-------------------|
| Installation | `docs/installation/README.md` |
| Configuration | `docs/configuration/README.md` |
| Runners | `docs/runners/README.md` |
| GPU Support | `docs/gpu/README.md` |
| Security | `docs/security/README.md` |
| Development | `docs/development/README.md` |
| Testing | `docs/development/testing.md` |
| API Reference | `docs/api/README.md` |
| ADRs | `docs/architecture/adr/` |
| Troubleshooting | `docs/troubleshooting/README.md` |
| **Full Index** | **`docs/INDEX.md`** ⭐ |
```

---

## docs/INDEX.md

```markdown
# AutoGit Documentation Index

**Last Updated**: [Auto-generated timestamp]

This index provides a complete map of all AutoGit documentation. Always check this file first when looking for
information.

## 📚 Documentation Structure

```
docs/
~~~

~~~
├── INDEX.md                           # This file - complete documentation map
├── installation/                      # Installation and setup
├── configuration/                     # Configuration references
├── architecture/                      # Architecture and design decisions
├── development/                       # Development guides and standards
├── runners/                          # Runner management
├── gpu/                              # GPU support
├── security/                         # Security guidelines
├── operations/                       # Operations and monitoring
├── api/                              # API documentation
├── cli/                              # CLI reference
├── tutorials/                        # Step-by-step tutorials
├── troubleshooting/                  # Common issues and solutions
└── FAQ.md                            # Frequently asked questions
```

## 🚀 Getting Started

New to AutoGit? Start here:

1. [README.md](../README.md) - Project overview and quick start
2. [Installation Guide](installation/README.md) - Detailed installation instructions
3. [Quick Start Tutorial](tutorials/quickstart.md) - Your first AutoGit deployment
4. [Configuration Basics](configuration/README.md) - Essential configuration

## 📖 Core Documentation

### Installation & Setup

| Document | Description | Audience |
|----------|-------------|----------|
| [Installation Overview](installation/README.md) | Complete installation guide | All users |
| [Prerequisites](installation/prerequisites.md) | System requirements and dependencies | All users |
| [Docker Compose Setup](installation/docker-compose.md) | Development setup with Docker Compose | Developers |
| [Kubernetes Setup](installation/kubernetes.md) | Production setup with Kubernetes | Operators |
| [Migration Guide](installation/migration.md) | Docker Compose → Kubernetes migration | Operators |

### Configuration

| Document | Description | Audience |
|----------|-------------|----------|
| [Configuration Overview](configuration/README.md) | Configuration system overview | All users |
| [GitLab Configuration](configuration/gitlab.md) | GitLab CE configuration | Administrators |
| [Runner Configuration](configuration/runners.md) | Runner management configuration | Administrators |
| [DNS Configuration](configuration/dns.md) | CoreDNS setup and configuration | Administrators |
| [SSL/TLS Configuration](configuration/ssl.md) | cert-manager and certificate setup | Administrators |
| [SSO Configuration](configuration/sso.md) | Authelia SSO setup | Administrators |
| [Ingress Configuration](configuration/ingress.md) | Traefik ingress setup | Administrators |
| [Storage Configuration](configuration/storage.md) | Persistent storage setup | Administrators |
| [Environment Variables](configuration/environment-variables.md) | All environment variables reference | All
users |

### Architecture & Design

| Document | Description | Audience |
|----------|-------------|----------|
| [Architecture Overview](architecture/README.md) | System architecture overview | All users |
| [Component Design](architecture/components.md) | Individual component designs | Developers |
| [Networking](architecture/networking.md) | Network architecture and policies | Operators |
~~~

```
| [Data Flow](architecture/data-flow.md) | How data flows through the system | Developers |
| [Scaling Strategy](architecture/scaling.md) | Horizontal and vertical scaling | Architects |
| [High Availability](architecture/high-availability.md) | HA configuration | Operators |
| [ADR Index](architecture/adr/README.md) | All architecture decisions | Architects |

#### Architecture Decision Records (ADRs)

| ADR | Title | Status | Date |
|-----|-------|--------|------|
| [ADR-001](architecture/adr/001-traefik-vs-nginx.md) | Traefik vs NGINX Ingress | Accepted | YYYY-MM-DD |
| [ADR-002](architecture/adr/002-fleeting-plugin.md) | Custom Fleeting Plugin Design | Accepted | YYYY-MM-DD |
| [ADR-003](architecture/adr/003-multi-architecture.md) | Multi-Architecture Strategy | Accepted | YYYY-MM-DD |
| [ADR-004](architecture/adr/004-sso-solution.md) | SSO Solution Selection | Accepted | YYYY-MM-DD |
| [ADR-005](architecture/adr/005-dns-management.md) | DNS Management Approach | Accepted | YYYY-MM-DD |
| [ADR-006](architecture/adr/006-storage-architecture.md) | Storage Architecture | Accepted | YYYY-MM-DD |

### Development

| Document | Description | Audience |
|----------|-------------|----------|
| [Development Overview](development/README.md) | Development environment setup | Developers |
| [Setup Guide](development/setup.md) | Local development setup | Developers |
| [Coding Standards](development/standards.md) | Code style and standards | Developers |
| [Testing Guide](development/testing.md) | Testing strategy and guidelines | Developers |
| [Agentic Workflow](development/agentic-workflow.md) | AI-assisted development workflow | Developers |
| [Project Structure](development/project-structure.md) | Codebase organization | Developers |
| [Common Tasks](development/common-tasks.md) | Common development tasks | Developers |
| [Licensing Guide](development/licensing.md) | License compliance guidelines | Developers |
| [Documentation Guide](development/documentation.md) | Writing and maintaining docs | All contributors |
| [CI/CD Guide](development/ci-cd.md) | Continuous integration setup | Developers |
| [Release Process](development/release-process.md) | How to cut a release | Maintainers |

### Runner Management

| Document | Description | Audience |
|----------|-------------|----------|
| [Runner Overview](runners/README.md) | Runner management overview | All users |
| [Autoscaling](runners/autoscaling.md) | Autoscaling configuration and behavior | Operators |
| [Multi-Architecture](runners/multi-arch.md) | Multi-arch runner setup | Operators |
| [Fleeting Plugin](runners/fleeting-plugin.md) | Custom fleeting plugin guide | Developers |
| [Provisioning](runners/provisioning.md) | Runner provisioning logic | Developers |
| [Tags and Labels](runners/tags-and-labels.md) | Runner tagging strategy | Administrators |
| [Monitoring](runners/monitoring.md) | Runner monitoring and metrics | Operators |
| [Troubleshooting](runners/troubleshooting.md) | Runner issues and solutions | All users |

### GPU Support

| Document | Description | Audience |
|----------|-------------|----------|
| [GPU Overview](gpu/README.md) | GPU support overview | All users |
| [NVIDIA GPUs](gpu/nvidia.md) | NVIDIA GPU setup and configuration | Operators |
| [AMD GPUs](gpu/amd.md) | AMD GPU setup and configuration | Operators |
| [Intel GPUs](gpu/intel.md) | Intel GPU setup and configuration | Operators |
| [Detection Logic](gpu/detection.md) | GPU detection implementation | Developers |
| [Scheduling](gpu/scheduling.md) | GPU-aware job scheduling | Developers |
| [Troubleshooting](gpu/troubleshooting.md) | GPU-related issues | All users |

### Security
```

```
| Document | Description | Audience |
|----------|-------------|----------|
| [Security Overview](security/README.md) | Security guidelines overview | All users |
| [Hardening Guide](security/hardening.md) | System hardening checklist | Operators |
| [Secrets Management](security/secrets.md) | Managing secrets securely | Developers |
| [Network Policies](security/network-policies.md) | Kubernetes network policies | Operators |
| [TLS Configuration](security/tls.md) | TLS/SSL security | Administrators |
| [Access Control](security/access-control.md) | RBAC and permissions | Administrators |
| [Audit Logging](security/audit-logging.md) | Security audit logs | Operators |
| [Vulnerability Management](security/vulnerability-management.md) | Handling vulnerabilities | Maintainers |
| [Incident Response](security/incident-response.md) | Security incident procedures | Operators |

### Operations

| Document | Description | Audience |
|----------|-------------|----------|
| [Operations Overview](operations/README.md) | Operations guide overview | Operators |
| [Monitoring](operations/monitoring.md) | Monitoring and observability | Operators |
| [Backup & Recovery](operations/backup.md) | Backup strategies | Operators |
| [Disaster Recovery](operations/disaster-recovery.md) | DR procedures | Operators |
| [Upgrades](operations/upgrades.md) | Upgrade procedures | Operators |
| [Performance Tuning](operations/performance-tuning.md) | Optimization guide | Operators |
| [Capacity Planning](operations/capacity-planning.md) | Resource planning | Architects |
| [Health Checks](operations/health-checks.md) | System health monitoring | Operators |

### API Documentation

| Document | Description | Audience |
|----------|-------------|----------|
| [API Overview](api/README.md) | API documentation overview | Developers |
| [Fleeting Plugin API](api/fleeting-plugin.md) | Fleeting plugin interface | Developers |
| [Runner Manager API](api/runner-manager.md) | Runner manager interface | Developers |
| [GPU Detector API](api/gpu-detector.md) | GPU detection interface | Developers |
| [Configuration API](api/configuration.md) | Configuration schemas | Developers |
| [REST API](api/rest.md) | REST API endpoints | Developers |

### CLI Reference

| Document | Description | Audience |
|----------|-------------|----------|
| [CLI Overview](cli/README.md) | Command-line tools overview | All users |
| [autogit CLI](cli/autogit.md) | Main CLI reference | All users |
| [runner-manager CLI](cli/runner-manager.md) | Runner management CLI | Operators |
| [gpu-detector CLI](cli/gpu-detector.md) | GPU detection CLI | Operators |

### Tutorials

| Document | Description | Audience |
|----------|-------------|----------|
| [Quick Start](tutorials/quickstart.md) | Get started in 15 minutes | New users |
| [First Pipeline](tutorials/first-pipeline.md) | Create your first CI/CD pipeline | New users |
| [Multi-Arch Builds](tutorials/multi-arch-builds.md) | Building for multiple architectures | Developers |
| [GPU Workloads](tutorials/gpu-workloads.md) | Running GPU-accelerated jobs | Developers |
| [Custom Runner](tutorials/custom-runner.md) | Creating custom runner configurations | Advanced users |
| [High Availability Setup](tutorials/high-availability.md) | Setting up HA deployment | Operators |

### Troubleshooting

| Document | Description | Audience |
```

```markdown
|----------|-------------|----------|
| [Troubleshooting Overview](troubleshooting/README.md) | Common issues and solutions | All users |
| [Installation Issues](troubleshooting/installation.md) | Installation problems | All users |
| [Runner Issues](troubleshooting/runners.md) | Runner-related problems | Operators |
| [GPU Issues](troubleshooting/gpu.md) | GPU-related problems | Operators |
| [Network Issues](troubleshooting/network.md) | Networking problems | Operators |
| [Performance Issues](troubleshooting/performance.md) | Performance problems | Operators |
| [Debugging Guide](troubleshooting/debugging.md) | General debugging techniques | Developers |

### Other

| Document | Description | Audience |
|----------|-------------|----------|
| [FAQ](FAQ.md) | Frequently asked questions | All users |
| [Glossary](GLOSSARY.md) | Terms and definitions | All users |
| [Contributing](../CONTRIBUTING.md) | How to contribute | Contributors |
| [License](../LICENSE) | MIT License text | All users |
| [Licenses](../LICENSES.md) | All dependency licenses | All users |
| [Changelog](../CHANGELOG.md) | Version history | All users |
| [Roadmap](../ROADMAP.md) | Future plans | All users |

## 🔍 Finding Documentation

### By Topic

- **Installation**: Start with `installation/README.md`
- **Configuration**: Start with `configuration/README.md`
- **Development**: Start with `development/README.md`
- **Troubleshooting**: Start with `troubleshooting/README.md`
- **API**: Start with `api/README.md`

### By Role

**New Users**:
1. [README.md](../README.md)
2. [Installation Guide](installation/README.md)
3. [Quick Start Tutorial](tutorials/quickstart.md)
4. [FAQ](FAQ.md)

**Developers**:
1. [Development Setup](development/setup.md)
2. [Coding Standards](development/standards.md)
3. [Testing Guide](development/testing.md)
4. [API Documentation](api/README.md)
5. [Architecture Overview](architecture/README.md)

**Operators**:
1. [Installation Guide](installation/README.md)
2. [Configuration Overview](configuration/README.md)
3. [Operations Guide](operations/README.md)
4. [Monitoring](operations/monitoring.md)
5. [Troubleshooting](troubleshooting/README.md)

**Architects**:
1. [Architecture Overview](architecture/README.md)
2. [ADR Index](architecture/adr/README.md)
3. [Scaling Strategy](architecture/scaling.md)
4. [High Availability](architecture/high-availability.md)
```

~~~
## 📝 Documentation Maintenance

### For Contributors

When modifying code that affects documentation:

1. **Check this INDEX.md** to find relevant documentation
2. **Update all affected documentation** in the same PR
3. **Add new documentation** if creating new features
4. **Update INDEX.md** if adding/removing documentation files
5. **Follow** [Documentation Guide](development/documentation.md)

### For Maintainers

- Review documentation in all PRs
- Keep INDEX.md up to date
- Ensure all documentation links work
- Archive outdated documentation
- Update ADRs for architectural changes

### Documentation Standards

- All documentation in Markdown
- Follow [Documentation Guide](development/documentation.md)
- Include code examples where appropriate
- Keep documentation current with code
- Use consistent terminology (see [Glossary](GLOSSARY.md))

## 🔗 External Resources

- [GitLab Runner Official Docs](https://docs.gitlab.com/runner/)
- [Traefik Documentation](https://doc.traefik.io/traefik/)
- [Authelia Documentation](https://www.authelia.com/)
- [CoreDNS Documentation](https://coredns.io/)
- [cert-manager Documentation](https://cert-manager.io/)
- [Kubernetes Documentation](https://kubernetes.io/docs/)

## 📊 Documentation Statistics

- Total Documents: [Auto-generated count]
- Last Updated: [Auto-generated timestamp]
- Contributors: [Link to contributors]

---

**Note**: This index is automatically validated by CI/CD. All links are checked on every commit.

If you can't find what you're looking for, check the [FAQ](FAQ.md) or [open an issue]
(https://github.com/yourusername/autogit/issues).
```

---

## docs/development/setup.md

```markdown
# Development Setup Guide

This guide will help you set up a local development environment for AutoGit.
~~~

~~~
**Related Documentation**:
- [Development Overview](README.md)
- [Coding Standards](standards.md)
- [Testing Guide](testing.md)
- [Project Structure](project-structure.md)

## Prerequisites

### Required Software

- **Python 3.11+** - Language runtime
- **UV** - Python project management
- **Docker 24.0+** - Container runtime
- **Docker Compose 2.20+** - Multi-container orchestration
- **Git** - Version control
- **Make** (optional) - Build automation

### Optional Software

- **Kubernetes** (k3s, kind, or minikube) - For Kubernetes development
- **Helm 3.12+** - Kubernetes package manager
- **kubectl** - Kubernetes CLI
- **Pre-commit** - Git hooks for code quality

### System Requirements

- **OS**: Debian 12+, Ubuntu 22.04+, or macOS 13+
- **RAM**: 8GB minimum (16GB recommended)
- **Storage**: 50GB free space
- **CPU**: 4 cores minimum

## Quick Start

```bash
# Clone the repository
git clone https://github.com/yourusername/autogit.git
cd autogit

# Run setup script
./scripts/setup-dev.sh

# Start development environment
make dev-up

# Run tests
make test
```

## Detailed Setup

### 1. Install Python Dependencies

```bash
# Install UV (if not already installed)
curl -LsSf https://astral.sh/uv/install.sh | sh

# Create virtual environment and install dependencies
uv sync
~~~

~~~
# Activate virtual environment
source .venv/bin/activate
```

### 2. Install Pre-commit Hooks

```bash
# Install pre-commit
pip install pre-commit

# Install git hooks
pre-commit install

# Run hooks manually (optional)
pre-commit run --all-files
```

### 3. Configure Development Environment

```bash
# Copy example environment file
cp .env.example .env.dev

# Edit configuration
nano .env.dev

# Required variables for development:
# - GITLAB_ROOT_PASSWORD
# - RUNNER_REGISTRATION_TOKEN
# - AUTHELIA_JWT_SECRET
# - POSTGRES_PASSWORD
```

### 4. Start Development Services

```bash
# Start all services with Docker Compose
docker compose -f compose/dev/docker-compose.yml up -d

# Check service status
docker compose -f compose/dev/docker-compose.yml ps

# View logs
docker compose -f compose/dev/docker-compose.yml logs -f
```

### 5. Verify Installation

```bash
# Run health checks
./scripts/health-check.sh

# Run unit tests
uv run pytest

# Run integration tests
uv run pytest tests/integration/
~~~

~~~
# Check code formatting
black --check src/
flake8 src/
mypy src/
```

## IDE Setup

### VS Code

1. Install recommended extensions:
   ```bash
   code --install-extension ms-python.python
   code --install-extension ms-python.vscode-pylance
   code --install-extension ms-python.black-formatter
   code --install-extension ms-azuretools.vscode-docker
   ```

2. Open workspace:
   ```bash
   code autogit.code-workspace
   ```

3. VS Code will automatically:
   - Use the project's Python interpreter
   - Format on save with Black
   - Run linters (flake8, mypy)
   - Provide devcontainer support

### PyCharm

1. Open project directory in PyCharm
2. Configure interpreter:
   - Go to `Settings` → `Project` → `Python Interpreter`
   - Add interpreter → Select `.venv/bin/python`
3. Configure Black formatter:
   - Go to `Settings` → `Tools` → `Black`
   - Set Black executable to `.venv/bin/black`
4. Enable pytest:
   - Go to `Settings` → `Tools` → `Python Integrated Tools`
   - Set default test runner to `pytest`

## Development Workflow

### Creating a New Feature

1. **Create feature branch**:
   ```bash
   git checkout -b feature/my-new-feature
   ```

2. **Check documentation**:
   - Review `docs/INDEX.md` for relevant documentation
   - Read related architecture docs

3. **Implement feature**:
   - Follow [Coding Standards](standards.md)
   - Write tests alongside code (TDD)
   - Update documentation as you go
~~~

~~~
4. **Run tests**:
   ```bash
   # Unit tests
   uv run pytest tests/unit/

   # Integration tests
   uv run pytest tests/integration/

   # Coverage report
   uv run pytest --cov --cov-report=html
   ```

5. **Update documentation**:
   - Update relevant docs in `docs/`
   - Update `docs/INDEX.md` if adding new docs
   - Create ADR if making architectural decisions

6. **Commit changes**:
   ```bash
   git add .
   git commit -m "feat: add new feature

   - Implement feature X
   - Add tests for feature X
   - Update docs: docs/path/to/doc.md"
   ```

7. **Push and create PR**:
   ```bash
   git push origin feature/my-new-feature
   # Create PR on GitHub with documentation checklist
   ```

### Running Tests

```bash
# All tests
make test

# Unit tests only
make test-unit

# Integration tests only
make test-integration

# Specific test file
uv run pytest tests/unit/test_runner_manager.py

# Specific test
uv run pytest tests/unit/test_runner_manager.py::TestRunnerManager::test_provision

# With coverage
make test-coverage

# Watch mode (re-run on file changes)
uv run pytest-watch
```
~~~

~~~
### Code Quality Checks

```bash
# Format code
make format

# Lint code
make lint

# Type check
make typecheck

# All quality checks
make quality
```

### Building and Testing Locally

```bash
# Build Docker images
make build

# Build multi-arch images
make build-multiarch

# Run integration tests with Docker Compose
make test-integration-docker

# Run end-to-end tests
make test-e2e
```

## Troubleshooting

### Port Conflicts

If you see port binding errors:

```bash
# Check what's using the port
sudo lsof -i :80
sudo lsof -i :443

# Stop conflicting services
sudo systemctl stop nginx
sudo systemctl stop apache2
```

### Docker Permission Issues

If you get permission denied errors:

```bash
# Add user to docker group
sudo usermod -aG docker $USER

# Log out and back in, or run:
newgrp docker
```
~~~

~~~markdown
### Python Import Errors

If you encounter import errors:

```bash
# Ensure virtual environment is activated
source .venv/bin/activate

# Reinstall dependencies
uv sync --reinstall

# Install package in editable mode
uv pip install -e .
```

### Database Connection Issues

If GitLab can't connect to PostgreSQL:

```bash
# Check PostgreSQL container
docker compose -f compose/dev/docker-compose.yml logs postgres

# Restart services
docker compose -f compose/dev/docker-compose.yml restart
```

## Development Tools

### Makefile Commands

```bash
# Development
make dev-up          # Start dev environment
make dev-down        # Stop dev environment
make dev-logs        # View logs
make dev-shell       # Open shell in container

# Testing
make test            # Run all tests
make test-unit       # Unit tests only
make test-integration # Integration tests
make test-e2e        # End-to-end tests
make test-coverage   # Generate coverage report

# Code Quality
make format          # Format code with Black
make lint            # Run linters
make typecheck       # Type checking with mypy
make quality         # All quality checks

# Building
make build           # Build Docker images
make build-multiarch # Multi-architecture build

# Documentation
make docs            # Build documentation
make docs-serve      # Serve docs locally
~~~

~~~
make docs-check      # Check documentation links

# Cleanup
make clean           # Remove build artifacts
make clean-all       # Deep clean including containers
```

### Helper Scripts

```bash
# Setup development environment
./scripts/setup-dev.sh

# Health checks
./scripts/health-check.sh

# Generate configuration
./scripts/generate-config.sh

# Database migrations
./scripts/db-migrate.sh

# Verify documentation index
./scripts/verify-doc-index.sh
```

## Next Steps

Now that your development environment is set up:

1. Read the [Coding Standards](standards.md)
2. Review the [Testing Guide](testing.md)
3. Check out [Common Tasks](common-tasks.md)
4. Browse the [Architecture documentation](../architecture/README.md)
5. Join the [development discussion](https://github.com/yourusername/autogit/discussions)

## Getting Help

- **Documentation**: Check [docs/INDEX.md](../INDEX.md)
- **Issues**: [GitHub Issues](https://github.com/yourusername/autogit/issues)
- **Discussions**: [GitHub Discussions](https://github.com/yourusername/autogit/discussions)
- **Contributing**: See [CONTRIBUTING.md](../../CONTRIBUTING.md)

---

**Documentation Version**: 1.0.0
**Last Updated**: YYYY-MM-DD
**Related Docs**: [Development Overview](README.md) | [Standards](standards.md) | [Testing](testing.md)
```

---

## docs/development/testing.md

```markdown
# Testing Guide

Comprehensive testing guide for AutoGit development.
~~~

Related Documentation:
- [Development Overview](README.md)
- [Coding Standards](standards.md)
- [Development Setup](setup.md)
- [CI/CD Guide](ci-cd.md)

## Testing Philosophy

AutoGit follows Test-Driven Development (TDD) principles:

1. **Write tests first** - Define expected behavior before implementation
2. **Red-Green-Refactor** - Fail, pass, improve cycle
3. **Comprehensive coverage** - Aim for 80%+ coverage
4. **Fast feedback** - Tests should run quickly
5. **Reliable** - Tests should not be flaky
6. **Maintainable** - Tests should be easy to understand and modify

## Test Pyramid

```
                  /\
                 /  \
                /E2E \
               /------\
              /  Integ \
             /----------\
            /    Unit    \
           /--------------\
```

- **Unit Tests** (70%): Test individual functions/classes in isolation
- **Integration Tests** (20%): Test component interactions
- **End-to-End Tests** (10%): Test complete user workflows

## Testing Stack

- **pytest** - Test framework
- **pytest-cov** - Coverage reporting
- **pytest-mock** - Mocking utilities
- **pytest-asyncio** - Async test support
- **pytest-docker** - Docker integration testing
- **factory_boy** - Test data factories
- **faker** - Fake data generation

## Project Test Structure

```
tests/
├── conftest.py              # Shared fixtures
├── unit/                    # Unit tests
│   ├── conftest.py
│   ├── test_runner_manager.py
│   ├── test_gpu_detector.py
│   └── test_fleeting_plugin.py
├── integration/             # Integration tests
│   ├── conftest.py
│   ├── test_docker_integration.py
│   └── test_kubernetes_integration.py
├── e2e/                     # End-to-end tests
```

~~~
│   ├── conftest.py
│   └── test_full_pipeline.py
├── fixtures/                # Test data and fixtures
│   ├── configs/
│   ├── data/
│   └── mocks/
└── utils/                   # Test utilities
    ├── factories.py
    └── helpers.py
```

## Writing Unit Tests

### Basic Test Structure

```python
"""Test suite for RunnerManager.

Documentation: docs/runners/README.md
"""

import pytest
from unittest.mock import Mock, patch, call
from autogit.runners import RunnerManager, ProvisionError


class TestRunnerManager:
    """Test suite for RunnerManager class."""

    @pytest.fixture
    def docker_client(self):
        """Mock Docker client."""
        return Mock()

    @pytest.fixture
    def config(self):
        """Test configuration."""
        return {
            "max_instances": 10,
            "idle_timeout": 300,
            "architectures": ["amd64", "arm64"]
        }

    @pytest.fixture
    def manager(self, docker_client, config):
        """RunnerManager instance."""
        return RunnerManager(docker_client, config)

    def test_provision_amd64_runner(self, manager, docker_client):
        """Test provisioning an amd64 runner without GPU."""
        # Arrange
        expected_id = "runner-123"
        docker_client.containers.run.return_value.id = expected_id

        # Act
        runner_id = manager.provision("amd64")

        # Assert
        assert runner_id == expected_id
~~~

~~~
        docker_client.containers.run.assert_called_once()
        call_args = docker_client.containers.run.call_args
        assert call_args[1]["image"].endswith(":amd64")

    def test_provision_with_nvidia_gpu(self, manager, docker_client):
        """Test provisioning runner with NVIDIA GPU."""
        # Arrange
        expected_id = "runner-gpu-123"
        docker_client.containers.run.return_value.id = expected_id

        # Act
        runner_id = manager.provision("amd64", gpu_type="nvidia")

        # Assert
        assert runner_id == expected_id
        call_args = docker_client.containers.run.call_args[1]
        assert "device_requests" in call_args
        assert call_args["device_requests"][0].driver == "nvidia"

    def test_provision_unsupported_architecture(self, manager):
        """Test provisioning with unsupported architecture raises error."""
        # Act & Assert
        with pytest.raises(ProvisionError, match="Unsupported architecture"):
            manager.provision("mips")

    def test_provision_max_instances_exceeded(self, manager):
        """Test provisioning fails when max instances reached."""
        # Arrange
        manager._current_instances = manager._config["max_instances"]

        # Act & Assert
        with pytest.raises(ProvisionError, match="Max instances reached"):
            manager.provision("amd64")

    @pytest.mark.asyncio
    async def test_provision_async(self, manager):
        """Test async provisioning."""
        # Arrange
        runner_id = await manager.provision_async("amd64")

        # Assert
        assert runner_id is not None
```

### Testing Best Practices

1. **Use descriptive test names**:
   ```python
   # Good
   def test_provision_fails_when_docker_unavailable(self):

   # Bad
   def test_provision_error(self):
   ```

2. **Follow Arrange-Act-Assert pattern**:
   ```python
   def test_something(self):
       # Arrange - Set up test data
~~~

~~~
        config = {"key": "value"}

        # Act - Execute the code under test
        result = function_under_test(config)

        # Assert - Verify the results
        assert result == expected
   ```

3. **One assertion per test** (when possible):
   ```python
   # Good - focused test
   def test_provision_returns_valid_id(self):
       runner_id = manager.provision("amd64")
       assert runner_id.startswith("runner-")

   # Good - related assertions
   def test_provision_configures_runner_correctly(self):
       runner_id = manager.provision("amd64")
       runner = manager.get_runner(runner_id)
       assert runner.architecture == "amd64"
       assert runner.status == "running"
   ```

4. **Use fixtures for setup**:
   ```python
   @pytest.fixture
   def sample_config():
       """Reusable configuration fixture."""
       return {
           "max_instances": 5,
           "idle_timeout": 300
       }
   ```

5. **Mock external dependencies**:
   ```python
   @patch('autogit.runners.docker_client')
   def test_with_mock(self, mock_docker):
       mock_docker.containers.run.return_value.id = "test-id"
       # Test implementation
   ```

### Testing Exceptions

```python
def test_provision_raises_error_on_invalid_config(self):
    """Test that invalid configuration raises ConfigError."""
    with pytest.raises(ConfigError) as exc_info:
        RunnerManager({})

    assert "max_instances" in str(exc_info.value)

def test_provision_logs_warning_on_gpu_unavailable(self, caplog):
    """Test warning logged when GPU unavailable."""
    with caplog.at_level(logging.WARNING):
        manager.provision("amd64", gpu_type="nvidia")

    assert "GPU not available" in caplog.text
~~~

~~~
```

### Testing Async Code

```python
@pytest.mark.asyncio
async def test_async_provision(self):
    """Test asynchronous runner provisioning."""
    runner_id = await manager.provision_async("amd64")
    assert runner_id is not None

@pytest.mark.asyncio
async def test_concurrent_provisioning(self):
    """Test multiple concurrent provisions."""
    tasks = [
        manager.provision_async("amd64")
        for _ in range(5)
    ]
    runner_ids = await asyncio.gather(*tasks)
    assert len(runner_ids) == 5
    assert len(set(runner_ids)) == 5  # All unique
```

## Writing Integration Tests

Integration tests verify that components work together correctly.

```python
"""Integration tests for Runner Manager with Docker.

Documentation: docs/runners/README.md
"""

import pytest
import docker
from autogit.runners import RunnerManager


@pytest.fixture(scope="module")
def docker_client():
    """Real Docker client for integration testing."""
    return docker.from_env()


@pytest.fixture(scope="module")
def runner_manager(docker_client):
    """Runner manager with real Docker client."""
    config = {
        "max_instances": 3,
        "idle_timeout": 60,
        "image": "gitlab/gitlab-runner:alpine"
    }
    return RunnerManager(docker_client, config)


class TestDockerIntegration:
    """Integration tests with Docker."""

    def test_provision_real_runner(self, runner_manager):
~~~

```
        """Test provisioning a real Docker container."""
        # Act
        runner_id = runner_manager.provision("amd64")

        # Assert
        assert runner_id is not None

        # Verify container exists
        container = runner_manager._docker.containers.get(runner_id)
        assert container.status == "running"

        # Cleanup
        runner_manager.deprovision(runner_id)

    def test_provision_with_volumes(self, runner_manager):
        """Test runner provisioning with volume mounts."""
        # Act
        runner_id = runner_manager.provision(
            "amd64",
            volumes={"/cache": {"bind": "/cache", "mode": "rw"}}
        )

        # Assert
        container = runner_manager._docker.containers.get(runner_id)
        mounts = container.attrs['Mounts']
        assert any(m['Destination'] == '/cache' for m in mounts)

        # Cleanup
        runner_manager.deprovision(runner_id)

    def test_autoscaling_behavior(self, runner_manager):
        """Test runner autoscaling up and down."""
        # Arrange - simulate job queue
        job_queue = [
            {"arch": "amd64"},
            {"arch": "amd64"},
            {"arch": "arm64"}
        ]

        # Act - scale up
        runner_ids = []
        for job in job_queue:
            runner_id = runner_manager.scale_for_job(job)
            runner_ids.append(runner_id)

        # Assert - runners provisioned
        assert len(runner_ids) == 3

        # Act - scale down after idle timeout
        import time
        time.sleep(65)  # Wait for idle timeout
        runner_manager.cleanup_idle()

        # Assert - runners deprovisioned
        active = runner_manager.get_active_runners()
        assert len(active) == 0
```

### Database Integration Tests

```python
@pytest.fixture
def database():
    """Test database fixture."""
    db = create_test_database()
    yield db
    db.drop_all()


def test_runner_state_persistence(runner_manager, database):
    """Test that runner state is persisted correctly."""
    # Arrange
    runner_id = runner_manager.provision("amd64")

    # Act
    runner_manager.save_state(database)
    runner_manager_new = RunnerManager.load_state(database)

    # Assert
    runners = runner_manager_new.get_active_runners()
    assert runner_id in [r.id for r in runners]
```

## Writing End-to-End Tests

E2E tests verify complete user workflows.

```python
"""End-to-end tests for AutoGit platform.

Documentation: docs/tutorials/quickstart.md
"""

import pytest
import requests
from time import sleep


@pytest.fixture(scope="module")
def autogit_instance():
    """Deploy a complete AutoGit instance for testing."""
    # Start all services
    subprocess.run(["docker", "compose", "-f", "compose/test/docker-compose.yml", "up", "-d"])

    # Wait for services to be ready
    wait_for_gitlab()
    wait_for_traefik()

    yield "https://gitlab.test.local"

    # Cleanup
    subprocess.run(["docker", "compose", "-f", "compose/test/docker-compose.yml", "down", "-v"])


def wait_for_gitlab(timeout=300):
    """Wait for GitLab to be ready."""
    start = time.time()
    while time.time() - start < timeout:
        try:
            response = requests.get("https://gitlab.test.local/-/health")
            if response.status_code == 200:
                return
        except requests.exceptions.RequestException:
            pass
        sleep(5)
    raise TimeoutError("GitLab did not become ready")


class TestE2EWorkflow:
    """End-to-end workflow tests."""

    def test_complete_cicd_pipeline(self, autogit_instance):
        """Test complete CI/CD pipeline from push to deployment."""
        # Arrange - Create project
        project = create_test_project(autogit_instance)

        # Act - Push code with .gitlab-ci.yml
        push_test_code(project)

        # Assert - Pipeline runs successfully
        pipeline = wait_for_pipeline(project)
        assert pipeline.status == "success"

        # Assert - Runner was auto-provisioned
        runners = get_active_runners()
        assert len(runners) > 0

        # Assert - Runner auto-deprovisioned after idle
        sleep(70)
        runners = get_active_runners()
        assert len(runners) == 0

    def test_multi_arch_build_pipeline(self, autogit_instance):
        """Test multi-architecture build pipeline."""
        # Arrange
        project = create_test_project(autogit_instance)

        # Act - Push multi-arch build config
        push_multiarch_config(project)

        # Assert - Pipeline uses correct runners
        pipeline = wait_for_pipeline(project)
        jobs = pipeline.jobs

        amd64_job = next(j for j in jobs if "amd64" in j.name)
        arm64_job = next(j for j in jobs if "arm64" in j.name)

        assert amd64_job.runner.tags == ["amd64"]
        assert arm64_job.runner.tags == ["arm64"]

    def test_gpu_accelerated_job(self, autogit_instance):
        """Test GPU-accelerated job scheduling."""
        # Arrange
        project = create_test_project(autogit_instance)

        # Act - Push GPU job config
        push_gpu_job_config(project)
        # Assert - Job runs on GPU-enabled runner
        pipeline = wait_for_pipeline(project)
        gpu_job = next(j for j in pipeline.jobs if "gpu" in j.name)

        assert "nvidia" in gpu_job.runner.tags or "amd" in gpu_job.runner.tags
        assert gpu_job.status == "success"

    def test_sso_authentication(self, autogit_instance):
        """Test SSO authentication flow."""
        # Act - Attempt to access GitLab
        response = requests.get(f"{autogit_instance}/projects", allow_redirects=False)

        # Assert - Redirected to Authelia
        assert response.status_code == 302
        assert "authelia" in response.headers["Location"]

        # Act - Login via Authelia
        session = login_via_authelia("testuser", "testpass")

        # Assert - Can access GitLab
        response = session.get(f"{autogit_instance}/projects")
        assert response.status_code == 200
```

## Test Coverage

### Measuring Coverage

```bash
# Run tests with coverage
pytest --cov=src --cov-report=html --cov-report=term

# View HTML report
open htmlcov/index.html

# Check coverage threshold
pytest --cov=src --cov-fail-under=80
```

### Coverage Configuration

```ini
# .coveragerc
[run]
source = src/
omit =
    */tests/*
    */venv/*
    */__pycache__/*
    */site-packages/*

[report]
exclude_lines =
    pragma: no cover
    def __repr__
    raise AssertionError
    raise NotImplementedError
    if __name__ == .__main__.:
```

~~~
    if TYPE_CHECKING:
    @abstractmethod
```

### Coverage Goals

- **Overall**: 80%+ coverage
- **Critical paths**: 95%+ coverage (runner management, GPU detection)
- **New code**: 90%+ coverage (enforced in CI)

## Continuous Integration

### GitHub Actions Integration

```yaml
# .github/workflows/test.yml
name: Tests

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: ['3.11', '3.12']

    steps:
      - uses: actions/checkout@v4

      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: ${{ matrix.python-version }}

      - name: Install dependencies
        run: |
          pip install uv
          uv sync

      - name: Run unit tests
        run: uv run pytest tests/unit/ --cov --cov-report=xml

      - name: Upload coverage
        uses: codecov/codecov-action@v3
        with:
          file: ./coverage.xml

      - name: Run integration tests
        run: uv run pytest tests/integration/

      - name: Check coverage threshold
        run: uv run pytest --cov --cov-fail-under=80
```

## Test Data Management

### Using Factories
~~~

```python
# tests/utils/factories.py
import factory
from autogit.models import Runner, Job


class RunnerFactory(factory.Factory):
    """Factory for creating test Runner instances."""

    class Meta:
        model = Runner

    id = factory.Sequence(lambda n: f"runner-{n}")
    architecture = "amd64"
    status = "running"
    gpu_type = None
    tags = factory.LazyAttribute(lambda obj: [obj.architecture])


class JobFactory(factory.Factory):
    """Factory for creating test Job instances."""

    class Meta:
        model = Job

    id = factory.Sequence(lambda n: n)
    name = factory.Sequence(lambda n: f"job-{n}")
    stage = "build"
    architecture = "amd64"
    requires_gpu = False


# Usage in tests
def test_runner_assignment():
    runner = RunnerFactory()
    job = JobFactory(architecture="amd64")

    assigned = assign_job_to_runner(job, runner)
    assert assigned is True
```

### Using Faker for Realistic Data

```python
from faker import Faker

fake = Faker()

def test_with_realistic_data():
    """Test with realistic fake data."""
    config = {
        "project_name": fake.company(),
        "repo_url": fake.url(),
        "owner_email": fake.email()
    }

    project = create_project(config)
    assert project.name == config["project_name"]
```

~~~markdown
## Performance Testing

### Load Testing

```python
import pytest
from locust import HttpUser, task, between


class GitLabUser(HttpUser):
    """Load test user for GitLab."""

    wait_time = between(1, 5)

    @task(3)
    def browse_projects(self):
        self.client.get("/projects")

    @task(1)
    def trigger_pipeline(self):
        self.client.post("/projects/1/pipeline", json={
            "ref": "main"
        })


# Run with: locust -f tests/performance/test_load.py --host=https://gitlab.test.local
```

### Benchmarking

```python
def test_provision_performance(benchmark):
    """Benchmark runner provisioning speed."""
    result = benchmark(manager.provision, "amd64")
    assert result is not None


def test_gpu_detection_performance(benchmark):
    """Benchmark GPU detection speed."""
    result = benchmark(gpu_detector.detect_all)
    assert len(result) > 0
```

## Mocking Strategies

### Mocking Docker Client

```python
@pytest.fixture
def mock_docker():
    """Mock Docker client."""
    with patch('docker.from_env') as mock:
        client = Mock()
        mock.return_value = client

        # Configure mock behavior
        client.containers.run.return_value.id = "test-container-id"
        client.containers.list.return_value = []
~~~

~~~
        yield client
```

### Mocking Kubernetes API

```python
@pytest.fixture
def mock_k8s_client():
    """Mock Kubernetes client."""
    with patch('kubernetes.client.CoreV1Api') as mock:
        api = Mock()
        mock.return_value = api

        # Configure mock behavior
        api.create_namespaced_pod.return_value = Mock(
            metadata=Mock(name="test-pod")
        )

        yield api
```

### Mocking External APIs

```python
@pytest.fixture
def mock_gitlab_api():
    """Mock GitLab API responses."""
    with requests_mock.Mocker() as m:
        # Mock project endpoint
        m.get(
            'https://gitlab.test.local/api/v4/projects/1',
            json={'id': 1, 'name': 'test-project'}
        )

        # Mock pipeline endpoint
        m.post(
            'https://gitlab.test.local/api/v4/projects/1/pipeline',
            json={'id': 123, 'status': 'pending'}
        )

        yield m
```

## Debugging Tests

### Using pytest Debugging

```bash
# Drop into debugger on failure
pytest --pdb

# Drop into debugger at start of test
pytest --trace

# Show local variables on failure
pytest --showlocals

# Run specific test with verbose output
~~~

~~~
pytest -vv tests/unit/test_runner_manager.py::TestRunnerManager::test_provision
```

### Logging in Tests

```python
import logging

def test_with_logging(caplog):
    """Test with log capture."""
    with caplog.at_level(logging.DEBUG):
        manager.provision("amd64")

    # Assert log messages
    assert "Provisioning runner" in caplog.text
    assert any("amd64" in record.message for record in caplog.records)
```

### Test Markers

```python
# Mark slow tests
@pytest.mark.slow
def test_long_running_operation():
    pass

# Mark GPU tests
@pytest.mark.gpu
@pytest.mark.skipif(not has_gpu(), reason="GPU not available")
def test_gpu_detection():
    pass

# Mark integration tests
@pytest.mark.integration
def test_docker_integration():
    pass

# Run specific markers
# pytest -m "not slow"  # Skip slow tests
# pytest -m gpu         # Only GPU tests
```

## Testing Checklist

Before submitting a PR, ensure:

- [ ] All tests pass locally
- [ ] New code has tests (90%+ coverage)
- [ ] Tests follow naming conventions
- [ ] Tests are documented
- [ ] Integration tests added for new components
- [ ] E2E tests added for new user workflows
- [ ] Performance tests added if applicable
- [ ] Tests run in CI/CD
- [ ] Coverage threshold met (80%+)
- [ ] No flaky tests
- [ ] Test documentation updated

## Common Testing Pitfalls
~~~

~~~
### 1. Flaky Tests

**Problem**: Tests that pass/fail randomly
**Solution**: Avoid timing dependencies, use proper mocking

```python
# Bad - timing dependent
def test_async_operation():
    start_async_task()
    time.sleep(1)  # Hope it's done
    assert is_complete()

# Good - wait for condition
def test_async_operation():
    start_async_task()
    wait_until(lambda: is_complete(), timeout=5)
    assert is_complete()
```

### 2. Test Interdependence

**Problem**: Tests that depend on execution order
**Solution**: Make each test independent

```python
# Bad - depends on previous test
def test_create_runner():
    global runner_id
    runner_id = manager.provision("amd64")

def test_delete_runner():
    manager.deprovision(runner_id)  # Depends on previous test

# Good - independent tests
def test_create_and_delete_runner():
    runner_id = manager.provision("amd64")
    manager.deprovision(runner_id)
```

### 3. Over-Mocking

**Problem**: Mocking too much defeats the purpose of testing
**Solution**: Mock only external dependencies

```python
# Bad - mocking internal logic
@patch('autogit.runners.RunnerManager._validate_config')
def test_provision(mock_validate):
    mock_validate.return_value = True
    # Not testing real validation

# Good - testing real logic
def test_provision_with_invalid_config():
    with pytest.raises(ConfigError):
        RunnerManager({"invalid": "config"})
```

## Resources
~~~

~~~
- **pytest Documentation**: https://docs.pytest.org/
- **Testing Best Practices**: https://testdriven.io/blog/testing-best-practices/
- **Coverage.py**: https://coverage.readthedocs.io/
- **Factory Boy**: https://factoryboy.readthedocs.io/

## Next Steps

- Review [Coding Standards](standards.md)
- Set up [Development Environment](setup.md)
- Read [CI/CD Guide](ci-cd.md)
- Check [Common Tasks](common-tasks.md)

---

**Documentation Version**: 1.0.0
**Last Updated**: YYYY-MM-DD
**Related Docs**: [Development Overview](README.md) | [Standards](standards.md) | [CI/CD](ci-cd.md)
```

---

## docs/architecture/adr/README.md

```markdown
# Architecture Decision Records

This directory contains Architecture Decision Records (ADRs) for AutoGit.

## What is an ADR?

An Architecture Decision Record (ADR) is a document that captures an important architectural decision made along
with its context and consequences.

## When to Create an ADR

Create an ADR when making decisions about:

- **Technology Choices**: Selecting frameworks, libraries, or tools
- **Architectural Patterns**: Choosing design patterns or architectural styles
- **Infrastructure Decisions**: Deployment strategies, scaling approaches
- **Integration Approaches**: How components communicate
- **Data Management**: Storage solutions, data flow
- **Security**: Authentication, authorization, encryption choices

## ADR Format

Each ADR follows this structure:

```markdown
# ADR-XXX: [Title]

**Status**: [Proposed | Accepted | Deprecated | Superseded]
**Date**: YYYY-MM-DD
**Deciders**: [List of people involved]
**Technical Story**: [Link to issue/epic]

## Context
~~~

~~~
[Describe the context and problem statement]

## Decision Drivers

- [Driver 1]
- [Driver 2]
- [Driver 3]

## Considered Options

- [Option 1]
- [Option 2]
- [Option 3]

## Decision Outcome

**Chosen option**: [Option X]

### Positive Consequences

- [Positive consequence 1]
- [Positive consequence 2]

### Negative Consequences

- [Negative consequence 1]
- [Negative consequence 2]

## Pros and Cons of the Options

### [Option 1]

- **Good**: [Advantage]
- **Bad**: [Disadvantage]
- **Neutral**: [Consideration]

### [Option 2]

- **Good**: [Advantage]
- **Bad**: [Disadvantage]

## Links

- [Related ADR](XXX-related-decision.md)
- [External resource]
```

## ADR Index

| ADR | Title | Status | Date |
|-----|-------|--------|------|
| [001](001-traefik-vs-nginx.md) | Traefik vs NGINX Ingress | Accepted | 2024-12-21 |
| [002](002-fleeting-plugin.md) | Custom Fleeting Plugin Design | Accepted | 2024-12-21 |
| [003](003-multi-architecture.md) | Multi-Architecture Strategy | Accepted | 2024-12-21 |
| [004](004-sso-solution.md) | SSO Solution Selection | Accepted | 2024-12-21 |
| [005](005-dns-management.md) | DNS Management Approach | Accepted | 2024-12-21 |
| [006](006-storage-architecture.md) | Storage Architecture | Accepted | 2024-12-21 |

## Creating a New ADR
~~~

~~~
1. **Copy the template**:
   ```bash
   cp docs/architecture/adr/template.md docs/architecture/adr/XXX-your-title.md
   ```

2. **Number sequentially**: Use the next available number

3. **Fill in all sections**: Don't leave sections empty

4. **Update this index**: Add your ADR to the table above

5. **Link related ADRs**: Cross-reference related decisions

6. **Get review**: Have the team review before marking as "Accepted"

## ADR Lifecycle

```
Proposed → Accepted → [Deprecated | Superseded]
```

- **Proposed**: Under discussion
- **Accepted**: Decision made and implemented
- **Deprecated**: No longer recommended but still in use
- **Superseded**: Replaced by a newer ADR

## Modifying Existing ADRs

- **Never modify accepted ADRs**: Create a new ADR that supersedes it
- **Update status**: Mark old ADR as "Superseded by ADR-XXX"
- **Link new ADR**: Reference the superseded ADR

## Best Practices

1. **Be concise**: ADRs should be quick to read
2. **Be specific**: Include concrete examples
3. **Include context**: Explain why, not just what
4. **Document alternatives**: Show what you didn't choose and why
5. **Link resources**: Include research and references
6. **Update promptly**: Write ADRs when decisions are made, not after

## Resources

- [ADR GitHub Organization](https://adr.github.io/)
- [Documenting Architecture Decisions](https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions)
- [ADR Tools](https://github.com/npryce/adr-tools)

---

**Last Updated**: 2024-12-21
**Maintainer**: Project Team
```

---

## docs/architecture/adr/001-traefik-vs-nginx.md

```markdown
~~~

```
# ADR-001: Traefik vs NGINX Ingress Controller

**Status**: Accepted
**Date**: 2024-12-21
**Deciders**: Project Team
**Technical Story**: Core infrastructure selection

## Context

AutoGit requires an ingress controller to manage external access to services in the Kubernetes cluster. The
ingress controller must:

- Route traffic to multiple services (GitLab, Authelia, DNS)
- Provide SSL/TLS termination
- Support Let's Encrypt automation
- Be MIT-license compatible
- Be lightweight and performant
- Have active community support

## Decision Drivers

- **License Compatibility**: Must be MIT or compatible (Apache 2.0, BSD)
- **Maintenance Status**: Active development and security updates
- **SSL Automation**: Native Let's Encrypt support
- **Resource Usage**: Low memory and CPU footprint
- **Ease of Configuration**: Simple, declarative configuration
- **Community Support**: Active community and good documentation
- **Cloud-Native**: Built for Kubernetes/Docker environments

## Considered Options

1. **Traefik** (MIT License)
2. **NGINX Ingress Controller** (Apache 2.0)
3. **HAProxy Ingress** (Apache 2.0)
4. **Envoy/Contour** (Apache 2.0)

## Decision Outcome

**Chosen option**: **Traefik**

### Rationale

1. **NGINX EOL**: NGINX Ingress Controller will be retired in March 2026 with no further security updates
2. **Native Let's Encrypt**: Traefik has built-in ACME support without additional controllers
3. **Dynamic Configuration**: Automatic service discovery and configuration
4. **MIT License**: Perfect compatibility with project license
5. **Modern Architecture**: Built specifically for cloud-native environments
6. **Better DX**: Simpler configuration than NGINX
7. **Active Development**: Regular releases and security patches

### Positive Consequences

- No need to manage separate cert-manager for basic SSL (though we still use it for advanced features)
- Automatic service discovery reduces configuration overhead
- Built-in dashboard for monitoring
- Native Docker and Kubernetes support
- Lower learning curve for contributors
- Future-proof (no EOL concerns)
```

~~~markdown
### Negative Consequences

- Different from traditional NGINX (team familiarity)
- Smaller ecosystem than NGINX (though growing)
- Different configuration paradigm (labels vs ConfigMaps)

## Pros and Cons of the Options

### Traefik

- **Good**: MIT licensed, active development, native Let's Encrypt, automatic service discovery
- **Good**: Built-in dashboard, Gateway API support
- **Good**: Lower resource usage than NGINX
- **Good**: Simpler configuration model
- **Bad**: Smaller ecosystem than NGINX
- **Bad**: Less mature than NGINX (though stable)
- **Neutral**: Different configuration paradigm

### NGINX Ingress Controller

- **Good**: Very mature, large ecosystem
- **Good**: Familiar to most teams
- **Good**: Extensive documentation
- **Bad**: **Being retired March 2026** - no security updates after
- **Bad**: More complex configuration
- **Bad**: Requires separate cert-manager for SSL
- **Bad**: Manual service configuration

### HAProxy Ingress

- **Good**: Very performant
- **Good**: Mature and stable
- **Bad**: More complex to configure
- **Bad**: Less Kubernetes-native
- **Bad**: Smaller community than Traefik/NGINX
- **Neutral**: Apache 2.0 license

### Envoy/Contour

- **Good**: Very powerful and flexible
- **Good**: Used by Istio/service meshes
- **Bad**: Much more complex than needed
- **Bad**: Higher resource usage
- **Bad**: Steeper learning curve
- **Neutral**: Overkill for this use case

## Implementation Notes

### Traefik Configuration

```yaml
# Traefik deployed via Helm
helm install traefik traefik/traefik \
  --namespace traefik \
  --set ports.web.redirectTo=websecure \
  --set certificatesResolvers.letsencrypt.acme.email=admin@example.com
```

### Service Discovery Example
~~~

~~~
```yaml
# Services auto-discovered via annotations
apiVersion: v1
kind: Service
metadata:
  name: gitlab
  annotations:
    traefik.ingress.kubernetes.io/router.entrypoints: websecure
    traefik.ingress.kubernetes.io/router.tls.certresolver: letsencrypt
spec:
  ports:
    - port: 80
```

## Links

- [Traefik Documentation](https://doc.traefik.io/traefik/)
- [NGINX Ingress EOL Announcement](https://github.com/kubernetes/ingress-nginx/issues/10870)
- [Related: ADR-006 SSL/TLS Configuration](006-storage-architecture.md)
- [Traefik GitHub](https://github.com/traefik/traefik)

## Supersedes

None (initial decision)

## Superseded By

None (current)

---

**Last Updated**: 2024-12-21
**Status**: Accepted and Implemented
```
~~~