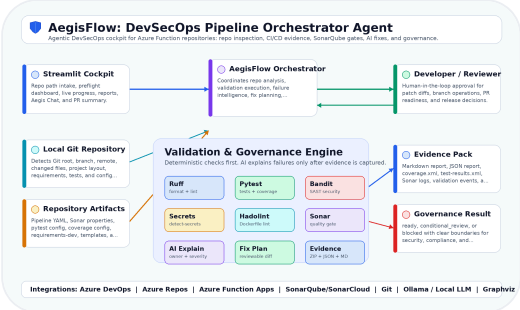
Industrial Project | 2026-05-31

AegisFlow: DevSecOps Pipeline Orchestrator Agent



An agentic DevSecOps cockpit for Azure Function and Python API repositories that validates CI/CD readiness, runs quality and security gates, explains failures, proposes human-approved fixes, and generates downloadable audit-ready evidence packs.

Project Snapshot

DATE
2026-05-31

CATEGORY
Industrial Project

SHORT DESCRIPTION
An agentic DevSecOps cockpit for Azure Function and Python API repositories that validates CI/CD readiness, runs quality and security gates, explains failures, proposes human-approved fixes, and generates downloadable audit-ready evidence packs.

TAGS / TECH STACK
Agentic AI | DevSecOps | CI/CD | Security | MLOps | Evidence Pack

Project Links

Demo Video | Hugging Face Space | GitHub Link | Demo Video | GitHub Link

Project Link | Project Link | GitHub Link

Full Project Content

> Building a DevSecOps orchestration agent that does not only **run checks** — it inspects repositories, maps acceptance criteria, validates CI/CD readiness, explains failures, proposes approved fix plans, and turns every run into a downloadable evidence pack.

GitHub Repository: dranubhaparashar/AegisFlow-DevSecOps-Pipeline-Orchestrator-Agent

---

> 🎥 **Live demo video:** youtube.com/watch?v=l_R8OV8VF8g
>
> 🚀 **Try it live:** Hugging Face Space
>
> 📚 **Wiki documentation:** Architecture · Technical Specification · Security Governance

---

Demo Video

Embedded media: Open video

Vision

Modern software teams often have mature CI/CD pipelines, but the developer experience around pipeline readiness is still fragmented.

A pull request may fail because of missing config files, incorrect import paths, weak test coverage, lint errors, secret-scan violations, Dockerfile problems, SonarQube setup issues, or cloud-only deployment checks. The real problem is not only that the pipeline failed — it is that developers must manually search logs, understand ownership, prepare fixes, rerun checks, and create evidence for reviewers.

AegisFlow turns this process into a guided DevSecOps cockpit.

It takes a local repository path, inspects the Git repo, checks DevSecOps readiness, runs deterministic validation gates, explains failures, proposes reviewable fixes, and produces a governance-ready evidence package.

The project is designed around one practical question:

Can a repository be proven ready for secure CI/CD review before a human approves the pull request?

AegisFlow answers this by combining:

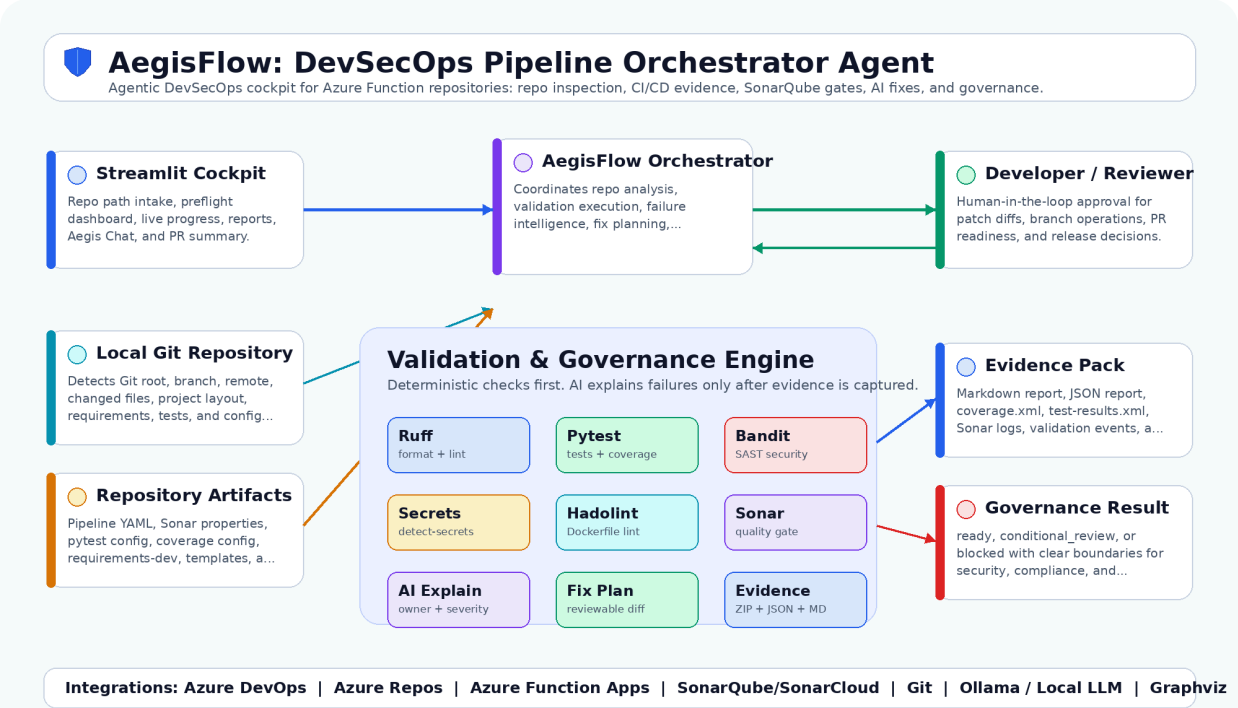
- repository preflight inspection
- acceptance-criteria mapping
- CI/CD file generation and validation
- quality gates using Ruff, Pytest, coverage, and Hadolint
- security gates using secret scanning, detect-secrets, and Bandit
- optional SonarQube or SonarCloud scanning
- AI failure explanation and owner mapping
- human-approved fix plans with exact diffs
- evidence-pack generation for audit, review, and release readiness
- governance status: `ready`, `conditional_review`, or `blocked`

Project Attributes

Attribute	Description
`problem-statement`	DevSecOps pipeline failures are often scattered across logs, quality tools, security scanners, test outputs, cloud checks, and PR comments. Teams need a single cockpit that validates readiness, explains failure causes, and generates auditable evidence.

Attribute	Description
`primary-objective`	Build an agentic DevSecOps orchestrator that validates repository readiness for secure CI/CD, explains pipeline failures, proposes safe fixes, and produces a downloadable evidence pack.
`core-technologies`	Python, Streamlit, Git, Azure DevOps, Azure Functions, Pytest, Ruff, Bandit, detect-secrets, Hadolint, SonarQube/SonarCloud, Ollama/local LLM, Graphviz, Hugging Face Spaces.
`runtime-interface`	Streamlit dashboard with tabs for Repo Preflight, Acceptance Criteria, Live Progress, Reports, AI Error Intelligence, AI Fix Plan, Aegis Chat, SonarQube, Governance, PR Comment, and Industry Use Cases.
`validation-scope`	Repository structure, CI/CD configuration, dependency separation, code quality, static security, tests, coverage XML, Sonar readiness, evidence outputs, and PR readiness.
`deployment-target`	Local Python/WSL execution for full repository validation, Hugging Face Space for public UI demonstration, and Azure DevOps pipeline integration as the enterprise target.
`demo-surface`	Public Hugging Face Space showing the AegisFlow dashboard and workflow. Full local repo validation works best locally because hosted Spaces cannot access private machine paths.
`key-capabilities`	Repo inspection, pipeline template validation, quality gates, security gates, coverage reporting, SonarQube integration, AI failure explanation, approved fix plans, Git automation, and evidence ZIP generation.
`production-focus`	Designed for Azure Function and Python API repositories where teams require traceable PR validation, secure pipeline governance, and auditable evidence before merge or deployment.

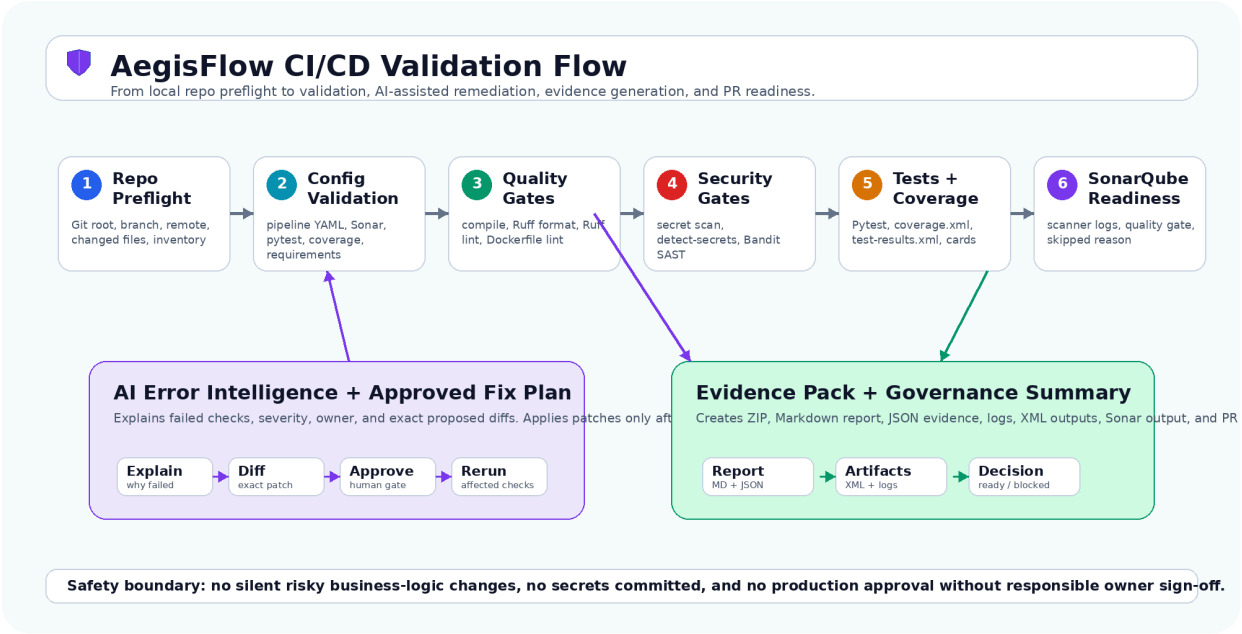
System Architecture



AegisFlow follows a local-first, human-in-the-loop architecture. The user provides a repository path, the orchestrator inspects the repo, deterministic validators capture evidence, AI explains failures only after evidence exists, and safe patches require explicit approval before modification.

```
flowchart LR
    U[Developer / Reviewer] --> UI[Streamlit Cockpit]
    UI --> AG[AegisFlow Orchestrator Agent]
    AG --> R[Local Git Repository]
    AG --> V[Validation & Governance Engine]
    V --> Q[Ruff / Pytest / Bandit / Secrets / Hadolint / Sonar]
    Q --> FI[Failure Intelligence]
    FI --> FP[AI Fix Plan]
    FP --> H[Human Approval]
    H --> RR[Rerun Affected Validation]
    AG --> E[Evidence Pack]
    E --> PR[PR Comment + Governance Summary]
```

CI/CD Validation Flow



The AegisFlow workflow converts a repository into a validation decision through six major stages:

1. **Repo Preflight** — detect Git root, branch, remote, changed files, project layout, and required config files.
2. **Config Validation** — verify pipeline YAML, Sonar properties, Pytest config, coverage settings, `.gitignore`, and dev requirements.
3. **Quality Gates** — run compile checks, Ruff formatting, Ruff linting, and Dockerfile linting where available.
4. **Security Gates** — run secret checks, detect-secrets, and Bandit static application security testing.
5. **Tests + Coverage** — execute Pytest, generate `coverage.xml`, generate `test-results.xml`, and summarize coverage cards.
6. **SonarQube Readiness** — run SonarScanner when credentials are configured, otherwise mark the gate honestly as skipped.

The final output is not a vague pass/fail message. It is a governance summary backed by evidence files, scanner logs, coverage XML, test XML, Markdown reports, and JSON reports.

Why This Matters

> **Note** A failed CI/CD run is not always a coding problem. It can be a missing file, an incorrect pipeline template, a wrong import path, a coverage threshold gap, a

secret-scan violation, a Dockerfile issue, or a missing Sonar credential. AegisFlow centralizes these scattered checks into one explainable workflow.

Important The project separates deterministic validation from AI explanation. AegisFlow first captures real evidence from the repository and tools. Only after that does AI summarize failures, suggest owners, and propose fixes.

Tip The strongest engineering value is the evidence pack. It makes validation portable across reviewers, PR comments, team discussions, and audit trails.

Warning AegisFlow does not silently approve production deployment or compliance sign-off. It gives decision support; final approval remains with the responsible human owner.

Caution The hosted Hugging Face Space is best for demonstrating the dashboard. Full private repository validation, Git automation, local paths, SonarScanner, Hadolint, and Ollama workflows work best when running AegisFlow locally or inside WSL.

Core Capability Map

Capability	What It Does
Repo Preflight	Detects Git root, branch, remote origin, provider, changed files, source folders, test folders, and config inventory.
Acceptance Criteria Cockpit	Converts a DevSecOps user-story checklist into `pass`, `fail`, `partial`, `manual`, or `cloud-only` status.
File Generation & Validation	Generates or checks `azure-pipeline.yml`, `azure-function.config.yml`, `sonar-project.properties`, `pytest.ini`, `.coveragerc`, `.gitignore`, and `requirements-dev.txt`.
Quality Gates	Runs Python compile checks, Ruff format, Ruff lint, Hadolint Dockerfile lint, and structural quality validations.

Capability	What It Does
Security Gates	Runs lightweight secret scanning, detect-secrets, and Bandit SAST checks.
Test & Coverage Engine	Runs Pytest, produces `coverage.xml` and `test-results.xml`, and shows Azure DevOps-style coverage summaries.
SonarQube Support	Runs scanner when `SONAR_HOST_URL` and `SONAR_TOKEN` exist; captures skipped reason when they are missing.
AI Error Intelligence	Explains failed checks, severity, likely owner, likely root cause, and common remediation actions.
AI Fix Plan	Shows exact diffs before applying safe deterministic patches; requires approval before modification.
Aegis Chat	Allows the user to ask questions about failures, Sonar status, evidence, PR readiness, or fix plans.
Git Automation	Supports branch creation/switching, commit, push, and PR preparation after repository confirmation and safety checks.
Evidence Pack	Creates a ZIP containing Markdown/JSON reports, logs, coverage XML, test XML, Sonar output, and selected config files.
Governance Decision	Reports `ready`, `conditional_review`, or `blocked` with clear boundaries for security and production approval.

Dashboard Modules

Tab	Purpose
Repo Preflight	Repository identity, branch, remote, changed files, and file inventory.
Acceptance Criteria	DevSecOps user-story checklist mapped to evidence-backed status.
Live Progress	CI/CD-style execution timeline with heartbeat updates for long-running tasks.
Report & Downloads	Evidence ZIP, Markdown report, JSON report, coverage cards, and test summaries.
AI Error Intelligence	Failure explanations, probable owner, severity, and suggested fixes.

Tab	Purpose
AI Fix Plan	Review exact diffs, approve patch, apply safely, and rerun affected validation.
Aegis Chat	Ask natural-language questions about the current run and evidence.
SonarQube	Scanner status, skipped reason, quality-gate status, and scanner logs.
Governance	Release decision support and responsible sign-off boundaries.
PR Comment	Copyable pull request validation summary.
Industry Use Cases	Application areas across DevOps, DevSecOps, MLOps, compliance, and platform engineering.

Evidence Pack Structure

After a validation run, AegisFlow writes evidence into the target repository:

```
orchestrator_reports/
  devsecops_orchestration_report_<timestamp>.md
  devsecops_orchestration_report_<timestamp>.json
  aegisflow_evidence_pack_<timestamp>.zip
  coverage.xml
  test-results.xml
  sonar-scanner-output.txt
```

The evidence package supports:

- pull request review
- sprint/story closure evidence
- security review documentation
- SonarQube readiness traceability
- release-readiness discussion
- audit and governance handoff
- team debugging when a pipeline fails

Safety and Governance Model

AegisFlow is agentic, but it is not uncontrolled automation.

Area	Safety Behavior
Business logic	Review-only unless the user approves an exact diff.
Secrets	Never generated, printed into code, or committed; recommends Key Vault or secure variables.
Risky files	No silent modifications to production logic or compliance-sensitive content.
Git actions	Branch, commit, and push actions require repository confirmation and safety checks.
Production deployment	Decision support only; final release approval remains human-owned.
Sonar and security	Failing or skipped gates are reported honestly; AegisFlow does not falsely mark them as complete.
Cloud-only tasks	Azure branch policies, deployed endpoint checks, and rollback tests are marked as manual/cloud-only when local evidence is unavailable.

Agentic Workflow

```
flowchart TD
    A[User enters local repo path] --> B[Prefetch repository details]
    B --> C{Correct repo confirmed?}
    C -->|No| D[Stop before risky operations]
    C -->|Yes| E[Run deterministic validation]
    E --> F[Capture logs, XML, reports, and scanner output]
    F --> G{Any failures?}
    G -->|No| H[Generate evidence pack and PR summary]
    G -->|Yes| I[AI Error Intelligence]
    I --> J[Generate fix plan with exact diff]
    J --> K{Human approves?}
    K -->|No| L[Keep review-only recommendation]
    K -->|Yes| M[Apply safe patch]
    M --> N[Rerun affected validation]
    N --> H
```

The key design principle is simple:

> **Evidence first. AI second. Human approval before change.**

Runtime Example

```
# Clone the repo
git clone https://github.com/dranubhaparashar/AegisFlow-DevSecOps-Pipeline-Orchestrator-Agent.git
cd AegisFlow-DevSecOps-Pipeline-Orchestrator-Agent

# Create virtual environment
python -m venv aegisflow_env
source aegisflow_env/bin/activate

# Install dependencies
pip install -r requirements.txt

# Run dashboard
streamlit run app.py --server.port 8501

# Open http://localhost:8501
# Paste your local repository path
# Click Prefetch repo details
# Confirm the repository
# Run orchestrator
# Download evidence pack from Report & Downloads
```

For Conda or WSL:

```
conda create -n aegisflow python=3.11 -y
conda activate aegisflow
pip install -r requirements.txt
python -m streamlit run app.py
```

Optional SonarQube Setup

AegisFlow can run SonarQube or SonarCloud scanning when credentials are present.

```
export SONAR_HOST_URL="https://your-sonarqube-server"
export SONAR_TOKEN="your-token"
export SONAR_PROJECT_KEY="your-project-key"
```

If these values are not available, AegisFlow still runs local validation, test coverage, security scans, and evidence-pack generation. The Sonar stage is marked as skipped with the reason, instead of falsely passing.

Public Demo Surfaces

> **Important** The Hugging Face Space provides browser-based access to the UI and project demonstration. For real local repositories, private paths, and Git actions, run AegisFlow locally or in WSL.

Demo Links

- 🚀 **Live App:** AnubhaParashar / AegisFlow DevSecOps Pipeline Orchestrator Agent
- 🎥 **Demo Video:** AegisFlow YouTube Demo
- 📦 **GitHub Repository:** AegisFlow-DevSecOps-Pipeline-Orchestrator-Agent
- 📚 **Wiki Documentation:** Architecture · Function Reference · Technical Spec

---

Engineering Value

AegisFlow is not just a Streamlit wrapper around command-line tools.

It demonstrates how to transform raw DevSecOps checks into a governed engineering workflow:

- **measurable** — every check creates status, logs, XML, or JSON evidence
- **explainable** — failures are summarized with severity, probable owner, and fix direction
- **reviewable** — proposed fixes are shown as exact diffs before application
- **auditable** — outputs are bundled into a portable evidence pack
- **governed** — the release decision is `ready`, `conditional_review`, or `blocked`
- **human-controlled** — risky changes, Git actions, and production approval remain human-owned
- **platform-aligned** — maps local repo validation to Azure DevOps, SonarQube, and PR review needs

---

Current Strengths

- Local-first validation works before Azure DevOps pipeline execution.
- Acceptance criteria mapping makes story closure evidence clearer.
- Deterministic checks run before AI explanation, reducing hallucinated debugging.
- Evidence pack creates a portable artifact for PR review and governance.
- SonarQube integration supports real quality-gate workflows when credentials are configured.

- AI fix planning supports safe, human-approved remediation instead of uncontrolled automation.
- Git automation is guarded by repository confirmation and safety checks.
- Dashboard gives both developer-level logs and manager-level governance status.

Industry Use Cases

Industry / Team	Use Case
Platform Engineering	Standardize repository readiness checks before teams adopt central CI/CD templates.
DevSecOps Teams	Validate security, quality, coverage, and Sonar gates before PR approval.
MLOps Teams	Check model-serving APIs, Python apps, and deployment repositories before release.
Azure Function Teams	Prepare Python Azure Function repositories for Azure DevOps pipelines.
Compliance Teams	Collect evidence packs for audit, sprint closure, and governance review.
Engineering Managers	Understand whether a PR is ready, conditionally reviewable, or blocked.
Developers	Get failure explanations and safe fix plans without manually reading long logs.

Next Improvements

- Add direct Azure DevOps PR creation using PAT and repository metadata.
- Add pipeline-run polling from Azure DevOps and import cloud logs into the evidence pack.
- Add Azure Function health-check validation against deployed endpoints.
- Add rollback verification and deployment-slot validation.
- Add deeper SonarQube issue parsing and quality-gate trend history.
- Add SARIF export for security and code-scanning integrations.
- Add policy-as-code support using OPA/Rego for enterprise governance rules.
- Add GitHub Actions template support alongside Azure DevOps pipelines.
- Add multi-repository batch validation for platform teams.

- Add LLM provider abstraction for Ollama, Azure OpenAI, OpenAI API, and local models.

Key Innovation

> **Important** AegisFlow connects **DevSecOps evidence capture**, **AI-assisted failure explanation**, and **human-approved remediation** into one workflow. Most teams operate these as disconnected activities: pipeline logs in one place, scanner results in another, code fixes in another, and audit evidence prepared manually.

It turns:

Local repository → deterministic validation → failure intelligence → approved fix plan → rerun validation → evidence pack → PR governance summary

rather than stopping at a failed pipeline log.

The evidence pack is the keystone. It allows the same validation run to support developer debugging, reviewer confidence, story closure, security governance, and release-readiness discussion.

Conclusion

AegisFlow shows how DevSecOps validation can evolve from scattered command-line checks into an agentic orchestration workflow.

It combines:

- repository preflight inspection
- acceptance criteria mapping
- CI/CD file generation and validation
- quality, security, test, coverage, and Sonar gates
- AI-assisted failure explanation
- human-approved fix planning
- evidence ZIP generation
- PR-ready governance summaries
- safe Git automation boundaries

The result is a practical DevSecOps cockpit for teams that need faster validation, better explanations, stronger evidence, and safer remediation before code reaches production.

Final Thought

> From **pipeline failure logs** to **evidence-driven DevSecOps readiness with approved remediation**

> The real value is not only running Ruff, Pytest, Bandit, or SonarQube — it is connecting those tools into a governed workflow where every decision is evidence-backed, every risky change is reviewed, and every PR can be explained clearly.

> Local repo inspection · CI/CD validation · AI failure intelligence · approved fix plan · evidence pack · governance summary.

Source: src/content/posts/aegisflow-devsecops-pipeline-orchestrator-agent/aegisflow-devsecops-pipeline-orchestrator-agent.md