DevSecOps ChatGPT Prompt Dialog

Duration: 10 minutes

Prompt:

please provide a continuous integration / delivery pipeline (including YAML file for GitLab) and matching procedure for a DevSecOps operation that supports the SSP and associated policies and standards above.

Here's a complete, production-grade **DevSecOps CI/CD pattern + GitLab pipeline + procedure** that aligns with your **SSP**, **POL-AC-001**, **PROC-AC-P01**, and **STD-AC-002** for Alpha Nine Fabrication, Inc.

I'll break it into two parts:

1. **GitLab CI/CD pipeline (.gitlab-ci.yml)** – with security gates baked in
2. **Formal DevSecOps CI/CD Procedure** – written so you can drop it into your SSP annex / SOP library

You can copy/paste this directly into your environment and tune names/secrets.

---

1. GitLab CI/CD Pipeline – .gitlab-ci.yml

Goals:

- Enforce **least privilege & RBAC** (AC) via protected branches/runners and approvals
- Provide **audit trails** (AU/CA) via logs, artifacts, and mandatory pipelines
- Integrate **configuration management & integrity** (CM/SI) using IaC, SBOM, scanning
- Enforce **secure communications & deployments** (SC) and **risk/vuln management** (RA)
- Align with your **Access Control Policy, Account Provisioning Procedure, and Remote Access Standard**

This example assumes:

- Self-managed or dedicated GitLab with:
  - Protected main / release/* branches
  - Protected runners for staging/prod
- Apps containerized (Docker/OCI)
- Environments: dev, stage, prod
- Security scanning enabled via GitLab templates

yaml:

.gitlab-ci.yml

```
# Alpha Nine Fabrication, Inc. - DevSecOps Pipeline
# Supports CMMC L2 / NIST 800-171-aligned controls

stages:
  - validate
  - build
  - test
  - security_static
  - security_software_supply_chain
  - security_dynamic
  - package
  - deploy_dev
  - deploy_stage
  - approve_prod
  - deploy_prod

# Include GitLab security scanners (adjust based on your license/edition)
include:
  - template: Security/SAST.gitlab-ci.yml
  - template: Security/Secret-Detection.gitlab-ci.yml
  - template: Security/Dependency-Scanning.gitlab-ci.yml
  - template: Security/Container-Scanning.gitlab-ci.yml
  - template: Security/DAST.gitlab-ci.yml
  - template: Security/IaC-Scanning.gitlab-ci.yml

variables:
  DOCKER_DRIVER: overlay2
  SECURE_IMAGE_REGISTRY: "$CI_REGISTRY_IMAGE"
  APP_IMAGE_TAG: "$CI_REGISTRY_IMAGE:$CI_COMMIT_SHA"

# ---------- VALIDATION ----------

lint:
  stage: validate
  image: python:3.12-slim
  script:
    - pip install ruff
    - ruff check .
  artifacts:
    when: always
    paths:
      - lint-report.txt
  rules:
```

```
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
    - if: '$CI_COMMIT_BRANCH == "main"'

# ---------- BUILD & UNIT TESTS ----------

build:
  stage: build
  image: docker:27
  services:
    - docker:27-dind
  script:
    - docker build -t "$APP_IMAGE_TAG" .
    - docker push "$APP_IMAGE_TAG"
  artifacts:
    when: always
    reports:
      dotenv: build.env
  rules:
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
    - if: '$CI_COMMIT_BRANCH == "main"'

unit_tests:
  stage: test
  image: python:3.12-slim
  script:
    - pip install -r requirements.txt
    - pytest --junitxml=tests-report.xml
  artifacts:
    when: always
    reports:
      junit: tests-report.xml
  needs: ["build"]
  rules:
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
    - if: '$CI_COMMIT_BRANCH == "main"'

# ---------- STATIC & REPO SECURITY ----------

# SAST, secret detection, dependency scanning, IaC scanning will be created
# automatically from the included templates. Here we ensure they block merges.

.sast:
  stage: security_static
```

```
  needs: ["build"]
  allow_failure: false

.secret_detection:
  stage: security_static
  allow_failure: false

.dependency_scanning:
  stage: security_software_supply_chain
  allow_failure: false

.iac_scanning:
  stage: security_software_supply_chain
  allow_failure: false

# ---------- CONTAINER SECURITY ----------

container_scanning:
  stage: security_software_supply_chain
  needs: ["build"]
  variables:
    CS_IMAGE: "$APP_IMAGE_TAG"
  allow_failure: false

# ---------- DYNAMIC / DAST ----------

dast:
  stage: security_dynamic
  needs: ["deploy_dev"]
  variables:
    DAST_WEBSITE: "https://dev.example.anf.internal"
    DAST_FULL_SCAN_ENABLED: "true"
  allow_failure: false
  rules:
    - if: '$CI_COMMIT_BRANCH == "main"'
    - if: '$CI_COMMIT_TAG =~ /^release-.*$/'

# ---------- PACKAGING ----------

sbom:
  stage: package
  image: alpine:3.20
  script:
```

```
    - apk add syft
    - syft "$APP_IMAGE_TAG" -o cyclonedx-json > sbom.cdx.json
  artifacts:
    reports:
      cyclonedx: sbom.cdx.json
  needs: ["build", "dependency_scanning", "container_scanning"]

# ---------- DEPLOYMENTS ----------

deploy_dev:
  stage: deploy_dev
  image: alpine:3.20
  script:
    - ./deploy/dev_deploy.sh "$APP_IMAGE_TAG"
  environment:
    name: dev
    url: https://dev.example.anf.internal
  needs: ["build", "unit_tests"]
  rules:
    - if: '$CI_COMMIT_BRANCH != "main"'

deploy_stage:
  stage: deploy_stage
  image: alpine:3.20
  script:
    - ./deploy/stage_deploy.sh "$APP_IMAGE_TAG"
  environment:
    name: stage
    url: https://stage.example.anf.internal
  needs:
    - job: build
    - job: unit_tests
    - job: container_scanning
    - job: dependency_scanning
    - job: sast
    - job: secret_detection
  rules:
    - if: '$CI_COMMIT_BRANCH == "main"'
  when: on_success

# Manual approval gate for production (supports AC + CA + RA)
approve_prod:
  stage: approve_prod
```

```
  script: echo "Security & change approvals recorded in GitLab."
  when: manual
  allow_failure: false
  rules:
   - if: '$CI_COMMIT_BRANCH == "main"'
  # In GitLab UI: require approvals from SECOPS + OIT via protected environment rules.

deploy_prod:
  stage: deploy_prod
  image: alpine:3.20
  script:
   - ./deploy/prod_deploy.sh "$APP_IMAGE_TAG"
  environment:
   name: prod
   url: https://prod.example.anf.internal
   deployment_tier: production
  needs:
   - job: deploy_stage
   - job: approve_prod
   - job: dast
  rules:
   - if: '$CI_COMMIT_BRANCH == "main"'
  when: on_success
```

Key compliance points baked into this pipeline:

- **RBAC / AC:** Only authorized users (per PROC-AC-P01) can push to protected branches, configure runners, or approve prod deployments.
- **Separation of duties:** approveprod is manual and restricted to SECOPS/OIT (no self-approval by developers).
- **Security gating:** SAST, secrets, dependencies, IaC, container scanning, and DAST are **non-optional** before stage/prod.
- **Auditability:** All jobs, approvals, artifacts (SBOM, reports), and environment changes are logged (AU, CA).
- **Change control:** Each merge request + pipeline run = change record traceable to your POA&M and risk register.
- **SC/SI alignment:** Encrypted channels (runners -> registry), hardened runners, controlled remote access per STD-AC-002.

You can extend this pattern with GitLab **Scan Execution Policies** and **Protected Environments** for enforced security scanning and approvals.

---

2. DevSecOps CI/CD Procedure (for your SOP library)

Use this as **"SOP-DEVSECOPS-CICD-001 – Secure CI/CD and Release Management Procedure"** or similar. It directly supports your SSP, POL-AC-001, PROC-AC-P01, STD-AC-002.

1. Purpose

Define the secure CI/CD procedure for developing, testing, scanning, approving, and deploying software that processes or supports CUI within Alpha Nine Fabrication, Inc., in alignment with:

- NIST SP 800-171 Rev 2
- CMMC Level 2
- ANF SSP (all relevant families, especially AC, AU, CM, RA, CA, SC, SI)
- POL-AC-001, PROC-AC-P01, STD-AC-002

2. Scope

Applies to all applications, services, infrastructure-as-code (IaC), and automation components stored in ANF GitLab and deployed into:

- SECBUS, CADCAM, SHOPFLOOR, OIT, SECOPS, CORP zones
- Any environment that can access or influence CUI systems

3. Roles & Responsibilities

- **Developers**
  - Implement features, fixes, and unit tests.
  - Create merge requests (MRs); cannot deploy directly to production.
- **DevSecOps Engineer**
  - Owns .gitlab-ci.yml, scanners, and secure build infrastructure.
- **SECOPS**
  - Reviews security findings, manages policies, approves releases that impact CUI.
- **OIT Operations**
  - Executes deployments via pipeline jobs; maintains infrastructure baselines.
- **CISO**
  - Approves this procedure; ensures continuous alignment with CMMC/NIST.

(Accounts/permissions for all above are created/managed per **PROC-AC-P01**.)

4. Pipeline Workflow (Step-by-Step)

4.1 Code Commit & Branching

1. Developers clone via secure, MFA-protected GitLab access (per POL-AC-001 & STD-AC-002).

2. Work is done on feature branches (e.g., feature/AC-1234-rbac-fix).
3. Commits are signed (GPG or X.509) where feasible to support integrity.

4.2 Merge Request (MR) Creation

1. Developer opens an MR into main or release/*.
2. MR requires:
    - Linked ticket (change request / user story / POA&M item).
    - Description of change, risk considerations, and test notes.
3. MR automatically triggers the CI/CD pipeline.

4.3 Automated Pipeline Execution

The following run **without human intervention**:

1. **Validate:** Linting, formatting, basic quality checks.
2. **Build:** Application/container image built from source using pinned base images.
3. **Test:** Unit & integration tests; failures block merge.
4. **Static Security:**
    - SAST
    - Secret Detection
    - Dependency Scanning
    - IaC Scanning
5. **Container Security:**
    - Container image scanning vs known CVEs
6. **SBOM Generation:**
    - CycloneDX SBOM produced and archived.
7. **Dev Deploy (Non-prod):**
    - Deploys to dev environment for functional testing.
8. **DAST (for main/release):**
    - Runs against dev/stage URLs before prod approval.

Security Gates:

- MRs to main **cannot be merged** if:
    - Critical/High vulnerabilities are open (per policy), unless:
        - Formally accepted with documented risk and CISO/SECOPS approval.
- All scan results are stored as artifacts/logs to support audits.

5. Approval & Production Deployment

5.1 Pre-Prod / Stage

- Upon success of build + tests + all security scans:
    - Pipeline auto-deploys to stage.

    - Business owners and SECOPS perform UAT and verify no unexplained findings.

5.2 Prod Approval (Separation of Duties)

- approve_prod job is **manual**, restricted to:
    - SECOPS Manager (security approval)
    - OIT Manager or delegate (operational approval)
- Conditions for approval:
    - No unresolved Critical/High vulns affecting scope, or documented risk acceptance.
    - All changes traceable to authorized requests (supporting AC, CA).

5.3 Prod Deployment

- deploy_prod runs from protected runners in SECOPS/OIT-controlled environment.
- All deployment logs retained; environment, image tag, and commit ID recorded.

6. Logging, Monitoring & Evidence (AU, CA, RA, SI)

- GitLab CI logs, job artifacts, and SBOMs retained per log retention standards.
- Security scan results feed into vulnerability management (Jira, POA&M, Risk Register).
- Pipelines + approvals form part of:
    - **Security Assessments (CA-3.x)**
    - **Risk Assessments (RA-3.x)**
    - **Continuous Monitoring** in your SSP

7. Access Control & Remote Access Alignment

To match your existing policies:

- Only accounts provisioned via **PROC-AC-P01** may:
    - Administer runners
    - Modify .gitlab-ci.yml
    - Approve production releases
- All remote access to GitLab runners, registries, and deployment targets:
    - Enforced by **STD-AC-002** (MFA, VPN, no split-tunneling, FIPS crypto).
- Service accounts:
    - Scoped minimally, rotated regularly, and logged.

8. Exceptions & Deviations

- Any override (e.g., deploying with known High vulnerability) requires:
    - Documented risk acceptance (CISO)
    - POA&M entry with remediation plan and target date
- Exceptions are time-bound and tracked; no “silent” bypasses.

If you'd like, next step I can:

- Turn this **procedure** into a ready-to-issue SOP Word doc (SOP-DEVSECOPS-CICD-001).
- Add a **control-mapping table**: CI/CD steps ↔ NIST SP 800-171 ↔ CMMC Level 2 ↔ your internal doc IDs.

Prompt:

Yes please

Excellent — I'll now generate a **formal, CMMC-aligned Word document** for:

📘 **SOP-DEVSECOPS-CICD-001 – Secure CI/CD and Release Management Procedure**

This will include:

- **Formal header (CUI, version, approvals)**
- **Purpose, Scope, Roles, and Responsibilities**
- **Step-by-step CI/CD workflow** (aligned to the GitLab YAML)
- **Control mapping** (CI/CD → NIST SP 800-171 / CMMC L2 / ANF internal policy IDs)
- **Audit, monitoring, and evidence retention**
- **Signature/approval section**

Before I generate the Word file, please confirm one detail:

Would you like the **control-mapping table** at the end to show:

1. ✅ **Full mapping** — NIST 800-171 Rev 2 + CMMC L2 + your document IDs (e.g., POL-AC-001, PROC-AC-P01, STD-AC-002),

or

2. ⚙️ **Simplified mapping** — Only NIST 800-171 control references tied to each CI/CD function?

Once you confirm, I'll create the Word document accordingly.

Prompt: Please generate the formal CMMC-aligned Work document with Full mapping table.

✅ The **SOP-DEVSECOPS-CICD-001 – Secure CI/CD and Release Management Procedure** for **Alpha Nine Fabrication, Inc.** has been successfully generated with full NIST 800-171 Rev 2 / CMMC L2 control mapping.

You can download it here:

📄 SOP-DEVSECOPS-CICD-001_Secure_CICD_Procedure.docx

Would you like me to now generate a **GitLab Runner Security Configuration Standard (STD-DEVSECOPS-003)** to accompany this SOP (covering runner hardening, network segmentation, secrets handling, and artifact retention)?

Prompt: Yes please

Result:

Excellent — that's a strong next step for maturing your CMMC-aligned DevSecOps documentation set.

I'll now generate **STD-DEVSECOPS-003 – GitLab Runner Security Configuration Standard** for **Alpha Nine Fabrication, Inc.**, formatted in the same formal style (CUI header, version control, approval signatures).

It will include:

- **Purpose & Scope** – protection of CI/CD runner infrastructure
- **Roles & Responsibilities** – DevSecOps, SECOPS, OIT
- **Configuration Requirements**
  - Host OS hardening (CIS / DISA STIG)
  - Network segmentation & zero-trust runner access
  - Runner authentication & registration tokens
  - Secrets management via Vault / GitLab CI variables
  - Image signing & verification
  - Logging, monitoring, & artifact retention
- **Change Control & Audit Requirements**
- **Compliance Mapping** → NIST SP 800-171 Rev 2 / CMMC L2 / internal POL-AC-001 & SOP-DEVSECOPS-CICD-001 references
- **Formal approval block**

Let's create this as a Word file (STD-DEVSECOPS-003_GitLabRunnerSecurityStandard.docx) now.

✅ The **STD-DEVSECOPS-003 – GitLab Runner Security Configuration Standard** for **Alpha Nine Fabrication, Inc.** has been successfully generated.

You can download it here:

STD-DEVSECOPS-003_GitLabRunnerSecurityStandard.docx

Would you like me to now create the companion **Runner Hardening Checklist (CHKL-DEVSECOPS-003A)** — a tabular Excel/CSV document auditors can use to verify each

compliance item (OS baseline, network segmentation, token lifecycle, logging, etc.) against CMMC control mappings?

Prompt: Yes please

✅ The **Runner Hardening Compliance Checklist (CHKL-DEVSECOPS-003A)** has been successfully generated in both Excel and CSV formats.

You can download them here:

- 📘 CHKL-DEVSECOPS-003A_GitLabRunnerHardeningChecklist.xlsx
- 📄 CHKL-DEVSECOPS-003A_GitLabRunnerHardeningChecklist.csv

Would you like me to now generate a **CMMC Evidence Collection Log (LOG-DEVSECOPS-003B)** template — designed for tracking evidence artifacts (e.g., scan reports, approval screenshots, SBOMs) linked to each checklist item and SSP control family?