# Splunk App CI/CD Design Document

## 1. Introduction

### Purpose

This document outlines the design for a centralized, version-controlled, and automated CI/CD solution for managing the lifecycle of Splunk applications. The solution leverages GitLab for source control and CI/CD, Amazon S3 for centralized artifact storage, and integrates with Splunk for streamlined deployment.

### Scope

**Included:**

* GitLab-based version control and CI/CD pipelines
* S3-based centralized storage for Splunk app artifacts
* Secure and automated deployment mechanisms to Splunk

**Excluded:**

* Detailed Splunk app development guidance
* Splunk configuration not related to app deployment

### Target Audience

* DevOps Engineers
* Splunk Administrators
* Security Engineers
* Infrastructure Architects

### Glossary of Terms

* **Splunk App**: A package that extends the capabilities of Splunk.
* **S3**: Amazon Simple Storage Service for object storage.
* **GitLab**: A web-based DevOps lifecycle tool that provides Git repository management.
* **CI/CD**: Continuous Integration/Continuous Deployment.
* **Artifact**: A packaged version of a Splunk app (.tgz or .tar.gz).

## 2. Current State (Assumptions/Context)

### Challenges Without the Proposed Solution

* Manual app deployment across environments.
* Inconsistent version tracking.
* No centralized storage for app packages.
* Difficult rollback and auditability.

### Assumptions

* AWS account and S3 bucket are available.
* GitLab (Self-Managed or GitLab.com) is operational.
* Splunk Enterprise (Deployer, SHC, Indexers) is already configured.
* GitLab Runners can reach AWS and Splunk.

## 3. Proposed Solution Architecture

### High-Level Diagram

Developer -> GitLab Repo -> CI/CD Pipeline -> S3 Storage -> Splunk Deployment

### Component Breakdown

#### GitLab

* **Repo Structure**: One repository per app to promote modularity.
* **Branching Strategy**: GitLab Flow (main, dev, feature/*, release/*).
* **CI/CD Runners**: Dedicated runners in a secure subnet with IAM roles.

#### Amazon S3

* **Bucket Naming**: splunk-app-artifacts-<env>
* **Folder Structure**: /apps/<app\_name>/<version>/<app\_name>-<version>.tgz
* **Versioning**: Rely on GitLab tagging and file versioning in S3.
* **Lifecycle Policies**: Archive/delete objects older than 180 days.

#### Splunk

* **App Deployment**:
  + Deployer fetches app package from S3.
  + Uses splunk apply shcluster-bundle for SHC.
  + Indexers via automation (e.g., Ansible).

## 4. Detailed Design

### GitLab CI/CD Pipeline Definition

#### Stages

1. **Test**: Lint, validate app structure
2. **Build**: Create .tgz package
3. **Deploy**: Upload to S3
4. **Promote**: Trigger deployment to Splunk

#### Jobs per Stage

* **Test**:
* script:  
   - splunk-appinspect inspect . --mode precert
* **Build**:
* script:  
   - tar -czvf $APP\_NAME-$CI\_COMMIT\_TAG.tgz ./app\_source
* **Deploy**:
* script:  
   - aws s3 cp $APP\_NAME-$CI\_COMMIT\_TAG.tgz s3://$S3\_BUCKET/apps/$APP\_NAME/$CI\_COMMIT\_TAG/
* **Promote**:
* script:  
   - curl -X POST https://splunk-deployer.example.com/deploy?app=$APP\_NAME&version=$CI\_COMMIT\_TAG

#### Variables & Secrets Management

* Stored in GitLab CI/CD Variables:
  + AWS\_ACCESS\_KEY\_ID
  + AWS\_SECRET\_ACCESS\_KEY
  + S3\_BUCKET
  + SPLUNK\_DEPLOYER\_API\_KEY

#### Artifacts

* .tgz package files
* AppInspect report
* Logs from pipeline execution

### Security Design

#### IAM Roles & Policies

* Least privilege policy for GitLab runners:
* {  
   "Action": ["s3:PutObject", "s3:GetObject"],  
   "Resource": "arn:aws:s3:::splunk-app-artifacts-\*/apps/\*",  
   "Effect": "Allow"  
  }

#### S3 Bucket Policies

* Restrict access by IP and IAM principal.
* Enable logging and SSE-KMS encryption.

#### Network Security

* Use VPC endpoints for S3.
* Runners and Splunk in private subnet with security group rules.

#### Authentication & Authorization

* GitLab SSO with MFA.
* Role-based access control for repo and pipeline.

### Monitoring & Logging

* GitLab job monitoring for each pipeline stage.
* AWS CloudTrail for S3 access.
* Splunk internal logs for deployment status.

### Error Handling & Rollback

* On failure, alert via Slack/Email.
* Re-deploy previous version using S3 copy.
* Maintain Git tags for stable releases.

## 5. Implementation Plan

### Phased Approach

1. **Phase 1**: Set up GitLab repos and runners.
2. **Phase 2**: Implement packaging and S3 upload.
3. **Phase 3**: Integrate deployment to Splunk.
4. **Phase 4**: Harden security and test end-to-end.

### Prerequisites

* IAM roles and bucket creation
* GitLab runner with AWS and Splunk access
* Splunk API endpoints or orchestration tool

### Testing Strategy

* **Unit Tests**: Linting, appinspect
* **Integration Tests**: End-to-end pipeline
* **E2E Tests**: Validate deployment to test Splunk env

## 6. Operational Considerations

### Maintenance

* Periodic review of IAM policies
* Archive old artifacts
* Rotate secrets regularly

### Troubleshooting

* Check GitLab pipeline logs
* Monitor S3 logs for failed uploads
* Review Splunk logs for deployment errors

### Scalability

* Horizontal scaling of runners
* S3 supports unlimited storage
* Repo-per-app model allows parallel development

### Cost Optimization

* Use S3 IA or Glacier for older artifacts
* Auto-scale GitLab runners to reduce idle time

## 7. Future Enhancements

* Auto-promotion across dev/test/prod
* Integration with ServiceNow or Jira for release management
* App Marketplace for internal discovery
* Integration with Splunk Cloud push APIs
* Usage metrics dashboard via Splunk

**End of Document**