# Splunk Application Lifecycle Management Design

## 1. Introduction

### Purpose

Splunk environments are mission-critical and demand a consistent, auditable, and secure approach to managing custom app lifecycles. This design aims to standardize the build, validation, storage, and deployment of Splunk applications using GitLab, Amazon S3, and a hardened CI/CD pipeline.

### Scope

This design addresses the full lifecycle of internal Splunk applications: from source control to packaging, artifact management, deployment, and monitoring. It excludes marketplace apps and infrastructure-as-code for provisioning Splunk clusters.

### Audience

This document is intended for DevOps engineers, security engineers, platform owners, and developers involved in Splunk app development and deployment.

### Key Terminology

* **Splunk App**: A packaged bundle (.tgz) of configurations, knowledge objects, and optional scripts.
* **S3**: Amazon’s object storage service used as the centralized artifact repository.
* **GitLab**: The authoritative source control and CI/CD orchestrator.
* **CI/CD**: Continuous Integration and Continuous Deployment pipelines.
* **Artifact**: A compiled, versioned Splunk app package ready for deployment.

## 2. Context and Challenges

### Current Gaps

* Manual app deployments increase risk of human error.
* No consistent version tracking or rollback mechanism.
* Artifact sprawl across personal machines, shared drives, and Splunk deployers.

### Foundational Assumptions

* GitLab is hosted internally with runner scalability.
* AWS S3 buckets are accessible via secure endpoints within the organization’s VPC.
* Splunk is deployed in a distributed mode with deployers and search head clusters.

## 3. Proposed Solution Architecture

### Conceptual Overview

Developers commit changes to GitLab repositories. Upon push or merge, CI/CD pipelines validate, package, and store apps in Amazon S3. Once validated, apps are deployed to appropriate Splunk tiers (e.g., deployer, SHC, indexers).

### GitLab

* **Repository Structure**: Favor dedicated repositories per app to improve isolation, permissions, and pipeline granularity.
* **Branch Strategy**:
  + main: production-ready code.
  + develop: integration/testing.
  + Feature branches per enhancement/fix.
* **Runner Model**:
  + Use dedicated runners for security and consistent build environments.

### Amazon S3

* **Bucket Naming**: splunk-app-artifacts-<env>
* **Folder Structure**:
  + /<app\_name>/<version>/<timestamp>.tgz1`
* **Versioning Strategy**:
  + Trust GitLab tags as the source of truth.
  + Enable S3 versioning for additional rollback safeguards.
* **Lifecycle Policy**:
  + Retain last 10 versions; archive or purge older entries beyond 90 days.

### Splunk Integration

* Apps are pulled from S3 and deployed using splunk apply shcluster-bundle (for SHC) or custom Ansible scripts (for indexers).
* Deployments are triggered via GitLab pipeline jobs.

## 4. Detailed Design Specifications

### GitLab CI/CD Pipeline Blueprint

#### Pipeline Stages

1. **Validate**
2. **Build-Package**
3. **Store-Artifact**
4. **Deploy**
5. **Promote**

#### Sample GitLab CI Job Snippet

validate:  
 stage: validate  
 script:  
 - splunk-appinspect inspect . --mode precert  
  
build:  
 stage: build-package  
 script:  
 - tar -czvf myapp.tgz .  
  
upload:  
 stage: store-artifact  
 script:  
 - aws s3 cp myapp.tgz s3://splunk-app-artifacts-dev/myapp/1.2.3/

#### Secure Credential Management

* GitLab CI/CD secrets for AWS credentials (IAM role via OIDC preferred).
* Splunk credentials passed as masked GitLab variables or injected via secure vaults.

#### Pipeline Artifacts

* Lint/test reports
* Packaged .tgz files
* Deployment logs

### Security Architecture

#### IAM and S3

* GitLab runners assume IAM roles with write-only access to specific S3 paths.
* Bucket policy restricts uploads to trusted source IPs and verified GitLab runners.

#### Encryption

* S3 encryption via SSE-KMS.
* GitLab secrets stored encrypted and rotated quarterly.

#### Network Controls

* Use VPC endpoints for S3.
* Isolate GitLab runners in a hardened subnet.

#### Identity & Access

* GitLab access restricted to SSO groups.
* Splunk access controlled via LDAP or SAML integrations.

### Observability and Error Handling

* **Monitoring**: Pipeline status tracked via GitLab dashboards. S3 operations logged to CloudTrail.
* **Logging**: Central log aggregation for all pipeline steps, including stdout/stderr.
* **Rollback**: Maintain versioned artifacts. Previous versions can be redeployed via GitLab tags or manual override.

## 5. Implementation Roadmap

### Phase 1: Foundation Setup

* Provision S3 buckets with policies.
* Harden GitLab runners.
* Baseline Splunk deployment automation (e.g., Ansible roles).

### Phase 2: App Migration

* Onboard critical apps into GitLab.
* Create initial pipelines.

### Phase 3: Pilot and Iterate

* Test with 2–3 apps.
* Refine workflows.

### Prerequisites

* AWS accounts with required IAM roles.
* GitLab admin access.
* S3 bucket access with encryption enabled.
* Network connectivity between runners and Splunk instances.

## 6. Operational Considerations

### Maintenance

* Update pipelines as new Splunk versions require modified packaging or validation rules.
* Rotate credentials and audit access quarterly.

### Troubleshooting Guide

* **Build Failures**: Check appinspect logs.
* **S3 Upload Issues**: Validate IAM permissions and network routes.
* **Deployment Errors**: Inspect Splunk deployer logs and bundle replication status.

### Scalability

* Add more runners as app volume increases.
* S3 scales automatically; monitor cost growth.

### Cost Management

* Use S3 storage class transitions (e.g., to Infrequent Access).
* Monitor GitLab runner usage and optimize job parallelism.

## 7. Future Vision

* **Automated Promotions**: Trigger deployment to staging/production via tag rules.
* **App Catalog**: Internal portal for browsing validated apps.
* **ChatOps Integration**: Trigger deployments from Slack/MS Teams.
* **Drift Detection**: Validate deployed app versions against latest S3 artifacts.
* **Security Scanning**: Integrate static analysis or license compliance checks into pipeline.