Version-Aware Canary Deployment Strategy for External & Internal APIs

# 1. Overview / Executive Summary

To address increasing partner demands for version-aware rollouts and safe rollback mechanisms, we are introducing a Canary Deployment strategy tailored for our PCF-based API ecosystem. This solution empowers both external partners and internal consumers to selectively access new application versions based on headers and configuration flags, ensuring safer deployments with minimal risk.

# 2. Problem Statement

Our current deployment model lacks fine-grained traffic control, making it difficult to support gradual rollouts or partner-specific testing. Internal APIs are directly accessed by consumers, bypassing Apigee routing. Rollback options are manual and risky. Partners have requested a more controlled, self-service-based approach to manage version rollouts.

# 3. Goals

- Enable version-aware routing using headers and Apigee Dev App attributes.

- Support canary rollouts and rollback without full redeployment.

- Decouple partner and internal consumer traffic safely.

- Prepare a scalable solution that aligns with AWS migration in future.

# 4. Current State Architecture

Today, our APIs are hosted in PCF and exposed to partners via Apigee and Avi GTM. Deployments are handled using Bitbucket (manifest files), CloudBees pipelines, and UDeploy. However, there is no mechanism to route traffic conditionally based on versions.

[Placeholder: Insert diagram of current state architecture]

# 5. Proposed Canary Routing Strategy

We introduce two flags to control traffic:   
- featureFlag (from partner request header): values 'new' or 'old'   
- dialup (Apigee Dev App attribute): values 'true' or 'false'  
  
Routing logic:  
- If dialup=true and featureFlag=new → Route to Canary App  
- Else → Route to Main App

[Placeholder: Insert decision flow diagram for Canary Routing]

# 6. Client-Side Changes

- Add `featureFlag` header in request.

- Use `featureFlag=new` when testing new version.

- Remove or revert to `featureFlag=old` to use stable version.

# 7. Synchrony-Side Changes

- Modify Apigee proxy to evaluate flags and route requests accordingly.

- Configure Dev App attribute `dialup` as toggle per client.

- Create new PCF app version (canary) and expose it via unique GTM route if needed.

# 8. Internal Consumer Routing Options

Internal consumers currently call PCF apps directly and may not benefit from Apigee-level logic.  
Options include:  
- Dual GTMs (main + canary)  
- A centralized Routing Microservice  
- Future AWS-native routing (e.g., Route 53 + Lambda)

[Placeholder: Internal routing option diagram]

# 9. Long-Term vs Short-Term Approach

- Short-term: Dual PCF apps, GTMs, and Apigee flag logic.

- Long-term: Route 53-based or App Mesh-based routing in AWS. Canary logic moves out of Apigee.

# 10. Migration to AWS – Impact & Plan

- Avi GTM → Route53

- UDeploy → CodeDeploy / AppRunner

- Canary routing can be integrated with App Mesh or API Gateway with header-based routes.

# 11. Industry Adoption / Benchmarking

Companies like Netflix, Amazon, and Google leverage canary rollouts to test new versions in production without full-scale impact. It’s a proven pattern in enterprise environments for continuous delivery.

# 12. Playbook / Checklist

[Placeholder for detailed deployment checklist to be added separately]

# 13. Appendix: Diagrams

Include all final Whimsical diagrams here once complete.

- Current Architecture

- Canary Routing Logic

- Internal Routing Options

- AWS Equivalent Canary Flow