## What is Canary Deployment?

Canary deployments allow gradual traffic exposure to new application versions. In our architecture, we use Apigee headers and app-level flags to enable partners to opt-in to newer versions without impacting other clients. This layered approach provides rollback flexibility, minimizes blast radius, and improves testing confidence.

Partner-Driven Version Deployment: Canary Strategy Justification

# **1. Why This Proposal Exists**

A key partner has requested the ability to test new versions of our API code in production without impacting existing traffic. They need the ability to:  
- Gradually roll out new changes to a portion of their traffic.  
- Retain control over which version receives traffic (e.g., via headers).  
- Roll back instantly without relying on Synchrony's release schedule.  
  
While most stakeholders are aware of this partner requirement, this document includes context for any new participants in the review.

# **2. Problem Summary**

Our current deployment process uses a single app route per foundation. Any change to this app replaces code for all consumers. This approach does not allow partners to test newer builds in isolation or roll back safely. It also puts production stability at risk during releases.

# **3. What We Need to Support**

To meet the partner’s expectations, we need:  
- A version-aware deployment strategy.  
- Safe rollback capability controlled by the partner.  
- Routing logic controlled by header value and Apigee Developer App attribute.

# **4. Proposed Solution: Canary Routing via Header + App Attribute**

We recommend using canary deployments alongside Apigee routing logic. This allows:  
- Multiple app versions deployed in parallel (e.g., v1 main, v2 canary).  
- Partner controls exposure by sending `featureFlag=new` header.  
- Synchrony controls exposure by setting `dialup=true` in Apigee Developer App config.

# **5. Key Benefits**

- Enables partner-driven A/B testing or rollout without full release.  
- No impact on other partners or clients.  
- Seamless rollback by removing `dialup` flag or header.  
- Minimal changes to existing app and proxy setup.

# **6. Deployment Strategy Context (for Review)**

While our PCF-based platform currently supports Blue-Green deployments by default, it's important to recognize the broader landscape of deployment strategies to appreciate why a Canary layer is being proposed.

|  |  |  |  |
| --- | --- | --- | --- |
| **Strategy** | **Summary** | **Rollback Ease** | **Use at Synchrony** |
| All-at-once | Replace old version with new at once | ❌ High Risk | ❌ Not preferred |
| Rolling | Gradual replacement of instances | ⚠️ Moderate | ❌ Not in PCF |
| Blue-Green | Deploy new version separately, then switch | ✅ Easy | ✅ Current model |
| Canary | Route small traffic portion to new version | ✅ Easiest | ✅ Proposed layer |

Why Canary on Top of Blue-Green:  
Even though PCF supports Blue-Green deployments, switching versions affects all clients. Canary routing allows us to gradually expose changes to selected partners, based on header and app-level flags—without needing a full production switch. It adds a fine-grained layer of control with minimal risk.

# **7. Future-Readiness for v3, v4, and Beyond**

The current model supports easy extensibility. Future partners or advanced rollout scenarios (v3, v4, etc.) can be handled by deploying additional app versions (e.g., `my-app-v3`) and using the same conditional routing via Apigee.  
By expanding the values of the `x-feature-flag` header (e.g., `v2`, `v3`) and deploying companion apps, we maintain a consistent and scalable routing pattern without major structural changes.

# **8. Operational Checklist: Canary Rollout & Rollback Playbook**

Use the following as a playbook for controlled version rollout via Apigee and PCF:

* - ✅ Deploy canary app (e.g., `my-app-v2`) with a new app name using updated manifest.
* - ✅ Configure Apigee proxy to evaluate routing based on the following conditions:
* • `x-dialup: true` (configured as custom attribute on Developer App)
* • `x-feature-flag: v2` (HTTP header set by partner client)
* - ✅ Validate canary version using limited traffic routed via Apigee.
* - 🔁 For rollback: either unset `x-feature-flag` OR toggle `x-dialup` to `false` on Dev App.
* - 📦 After validation, deploy approved v2 code to main app (`my-app`) and remove canary app.
* - 🧪 Optional: Extend pattern for future versions (`x-feature-flag: v3`, etc.).

# **9. Field Naming Standards**

To maintain clarity and consistency across teams, the following field names will be standardized:  
- `x-feature-flag`: Header used by partners to opt into a specific version (e.g., `v2`, `v3`).  
- `x-dialup`: Developer App custom attribute in Apigee used to enable/disable version routing logic per partner.  
Both names follow standard HTTP header naming practices and clearly convey their purpose.

# Appendix: Whimsical Diagrams

## 📊 End-to-End Traffic Flow with Canary Routing

Depict how traffic flows from the client/partner → Apigee Proxy → PCF foundations. Show that Apigee routing logic (based on x-dialup and x-feature-flag) decides whether traffic goes to `my-app` (v1) or `my-app-v2` (v2). This helps explain the full control path.

[Insert Whimsical diagram link here]

## 📊 Routing Decision Matrix Based on Header and Attribute Flags

Create a decision tree: First check if `x-dialup=true`. If yes, check `x-feature-flag` (old/new). `true` or `new` → Canary (v2), else → Main (v1). If dialup is false or absent → always route to v1.

[Insert Whimsical diagram link here]

## 📊 Application Distribution Across PCF Foundations

Show how both main and canary apps (`my-app`, `my-app-v2`) are deployed in all 4 PCF foundations (DAL1, DAL2, PHX1, PHX2). Add Global Traffic Manager (GTM) in the center pointing to all regions.

[Insert Whimsical diagram link here]

## 📊 Step-by-Step Canary Rollout & Rollback Flow

Build a sequential flow: Deploy canary → Set dialup true for partner → Test with header `new` → If success: promote to main app → Else: rollback by disabling dialup. Add arrows for each step.

[Insert Whimsical diagram link here]

## 📊 Extending the Canary Model to v3, v4, and Beyond

Diagram a scalable model: `x-feature-flag: v3`, `v4`, etc. → route to `my-app-v3`, `my-app-v4`. Show multiple canary app instances and logic remaining backward-compatible.

[Insert Whimsical diagram link here]