

Calico Gateway API

Month Year

Presented by *Solutions Architect*



TIGERA

Agenda



- Recapitulative
- Traffic Control & Routing
- Security & Compliance
- Monitoring & Debugging

01

Recapitulative

Kubernetes Native vs Ingress Gateway

- Kubernetes provides three core service types for exposing apps:
 - ClusterIP - Internally only
 - NodePort - Inflexible, manual, conflictual
 - LoadBalancer - Costly (1 per service)
- Native Kubernetes services handle L4 exposure but lack L7 flexibility, security, and portability. Ingress/Gateway API fills these gaps with standardised, feature-rich traffic management.
- Each solves a piece of the puzzle, but gaps remain.

Feature	Native Kubernetes (ClusterIP/NodePort/LoadBalancer)	Ingress/Gateway API
External Access	Limited (NodePort: manual port management, LoadBalancer: cloud-specific).	Standardized L7 (HTTP/HTTPS) routing.
Traffic Routing	L4 only (no host/path rules).	Path-based (/api), host-based (app.example.com), headers, traffic splitting.
TLS Termination	Manual cert management (e.g., Service Mesh needed).	Built-in TLS termination (e.g., cert-manager integration).
API Gateway Features	None (requires third-party tools).	Rate limiting, auth (OIDC/JWT), request rewriting, retries.
Multi-Tenancy	No namespace isolation for routing.	HTTPRoute scoped to namespaces (Gateway API).
Cloud Portability	Tied to cloud LoadBalancers (AWS ALB, GCP LB).	Vendor-neutral configuration.
Security	Basic NetworkPolicy (L3/L4).	mTLS, WAF integration, L7 policies.
Observability	Limited (service metrics only).	Envoy access logs, Prometheus metrics, distributed tracing.

02

Traffic Control & Routing

Critical Gaps in Kubernetes Traffic Control

"How do you roll out new features without downtime or risk?"

- *What happens if the new version has a critical bug?*
- *How can we minimize user impact during deployments?*
- *Is there a way to automatically roll back if errors occur?*

"How do we handle failures gracefully without manual intervention?"

- *What if a backend becomes unhealthy—does traffic automatically shift away?*
- *Can we define SLA-based routing (e.g., route away if latency > 500ms)?*
- *Is there circuit-breaking to prevent cascading failures?*



"How do we support multi-cluster and hybrid-cloud routing?"

- *Can we split traffic across clusters (e.g., 80% in AWS, 20% in GCP)?*
- *How do we handle failover if a cluster goes down?*
- *Is there geo-aware routing to direct users to the nearest region?*

"How do we monitor and troubleshoot traffic routing in real-time?"

- *Can we see live traffic distribution between versions?*
- *Are there detailed metrics (success rate, latency, errors) per route?*
- *How do we trace a request path for debugging?*

Where Legacy Routing Falls Short



- **No native Kubernetes support for canary/blue-green deployments**
 - Manual scripting or external tools required.
 - No fine-grained control over traffic shifting (e.g., 5% → 10% → 50%).
- **Hard-coded routing rules in legacy Ingress**
 - Static configurations require redeployment for changes.
 - No dynamic traffic splitting based on conditions.
- **Lack of observability in traffic routing**
 - Difficulty tracking which requests go to which version.
 - No built-in metrics for success/failure rates per backend.
- **Security and compliance risks**
 - No built-in way to enforce zero-trust policies during traffic shifts.
 - Risk of exposing unfinished features to unintended users.

2.1

Traffic Routing

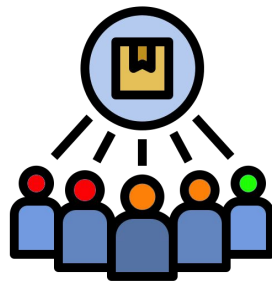
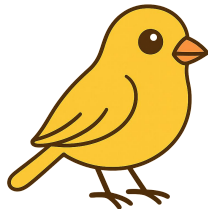
Solution: Traffic Splitting



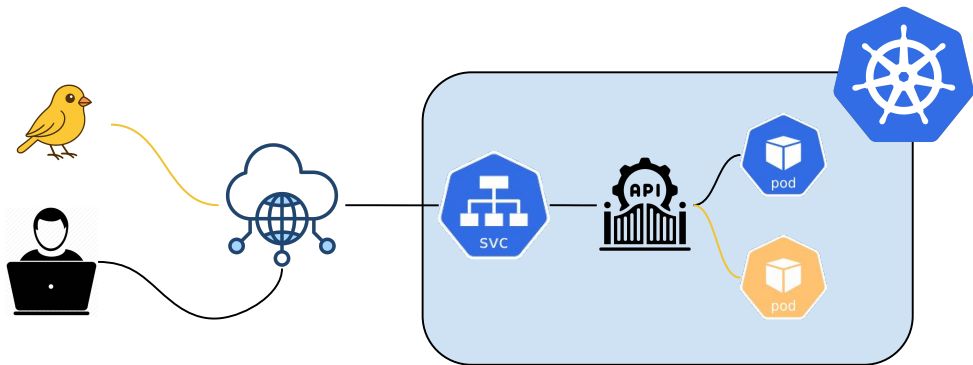
1. **Weighted** Splitting
 - Percentage-based distribution (e.g., 70/30)
2. **Header-Based** Routing
 - Split based on HTTP headers (e.g., `x-user-type: premium`)
3. **Path-Based** Routing
 - Route by URL path (e.g., `/v1/` → `v1-service`)
4. **Query Parameter** Routing
 - Split by URL query params (e.g., `?version=beta`)
5. **Cookie-Based** Routing
 - Direct traffic using session cookies
6. **Geolocation** Routing
 - Route by client IP/country (e.g., EU → `europe-service`)
7. Traffic **Mirroring (Shadowing)**
 - Copy traffic to another cluster without affecting responses
8. **Canary** Deployments
 - Subset-based splitting (e.g., 5% to new version)
9. **Runtime Fractional** Routing
 - Dynamic splits controlled by runtime configuration
10. **Priority-Based** Routing
 - Route to clusters based on priority levels
11. **Load Balancer Subsets**
 - Split traffic to endpoint subsets (e.g., by zone/version)

Each type can be combined (e.g., **weighted + header-based****) for advanced use cases.**

Common Deployment Strategies with Calico Gateway



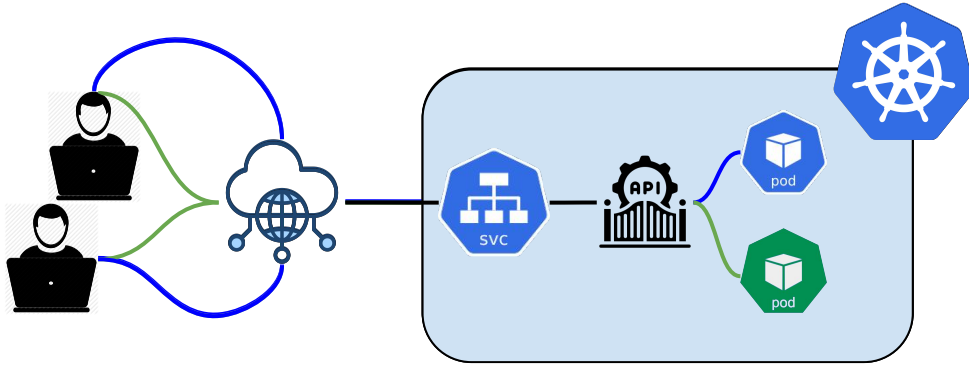
Canary Deployments



- Gradually roll out new versions to a subset of users
- Traffic splitting based on weight 90/10
- Often used for testing new features with minimal risk

```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: canary-demo
  namespace: prodapps
spec:
  parentRefs:
    - name: calico-gateway
      kind: Gateway
  hostnames:
    - "prodapps.com"
  rules:
    - matches:
        - path:
            type: PathPrefix
            value: /prodapps
      backendRefs:
        - name: canary-service
          port: 80
          weight: 10
        - name: primary-service
          port: 80
          weight: 90
```

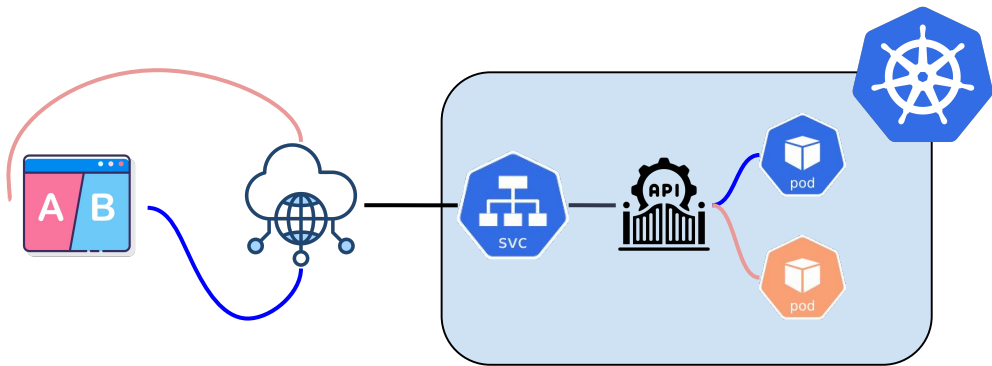
Blue Green Deployments



- Maintain two identical production environments
- Switch all traffic from "blue" (old) to "green" (new) at once
- Enables instant rollback if issues occur



A/B Testing

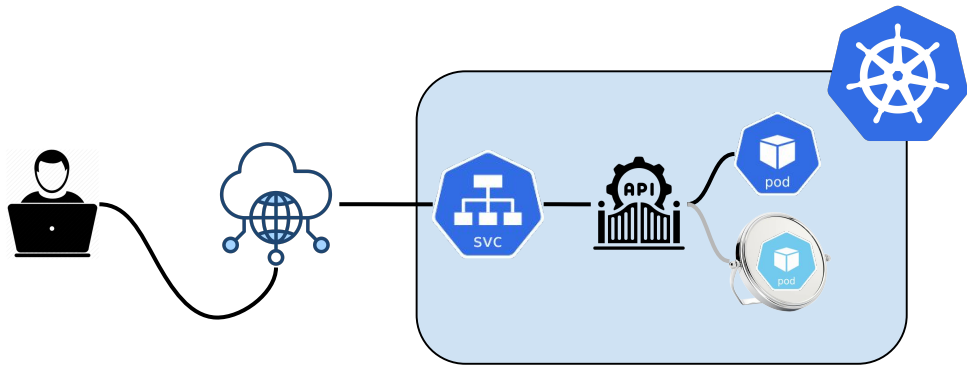


- Route traffic based on request attributes (headers, cookies, etc.)
- Useful for testing different versions with specific user segments (example: Mobile vs PC browsers)
- More sophisticated than simple percentage-based split

```
rules:
#B group: if header "x-ab-group: B" is
present
- matches:
- path:
  type: PathPrefix
  value: /prodapps
  headers:
  - name: x-ab-group
    value: B
  backendRefs:
  - name: ab-version-b
    port: 80
    weight: 100

#Default group A — all other traffic
- matches:
- path:
  type: PathPrefix
  value: /prodapps
  backendRefs:
  - name: ab-version-a
    port: 80
    weight: 100
```

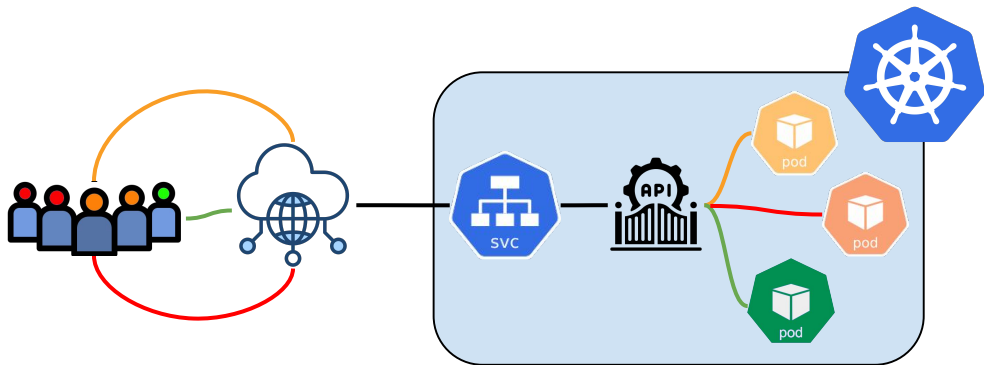
Traffic Mirroring (Shadowing)



- Send a copy of production traffic to new version
- Doesn't affect live users while testing real traffic patterns
- Helps validate performance before actual cutover

```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: mirroring-example
  namespace: default
spec:
  parentRefs:
    - name: calico-envoy-gateway
      kind: Gateway
      namespace: tigera-gateway
  hostnames:
    - "app.example.com"
  rules:
    - matches:
        - path:
            type: PathPrefix
            value: /myapp
      filters:
        - type: RequestMirror
          requestMirror:
            backendRef:
              name: shadow-service
              port: 8080
              group: ""
              kind: Service
            backendRefs:
              - name: primary-service
                port: 8080
```

Header-Based Routing



- Route traffic based on HTTP headers
- Useful for internal testing or feature flags
- Example: Route employees to new version while customers stay on stable

Rule 1: **Green users** (new features)

- matches:

- headers:

- type: **Exact**

name: **x-user-color**

value: **green**

filters:

- type: *RequestHeaderModifier*

requestHeaderModifier:

set:

- name: *x-routed-color*

value: *green*

backendRefs:

- name: **green-service**

port: 8080

weight: 100

Rule 2: **Yellow users** (beta features)

- matches:

- headers:

- type: **Exact**

name: **x-user-color**

value: **yellow**

backendRefs:

- name: **yellow-service**

port: 8080

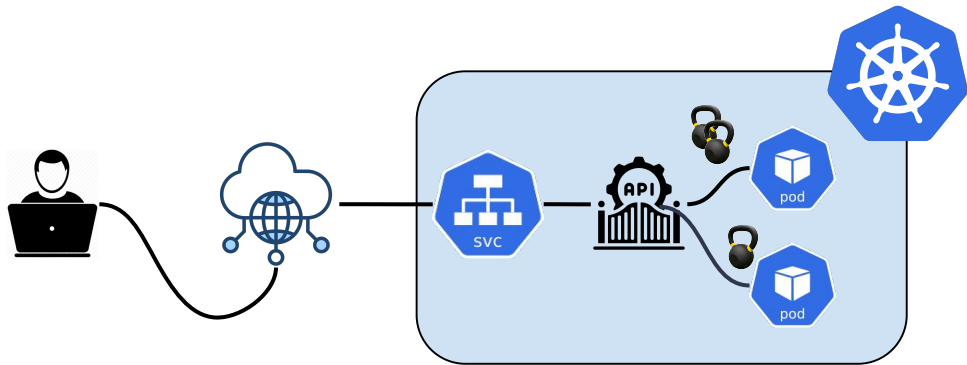
Rule 3: **Red users** (default stable)

- backendRefs:

- name: **red-service**

port: 8080

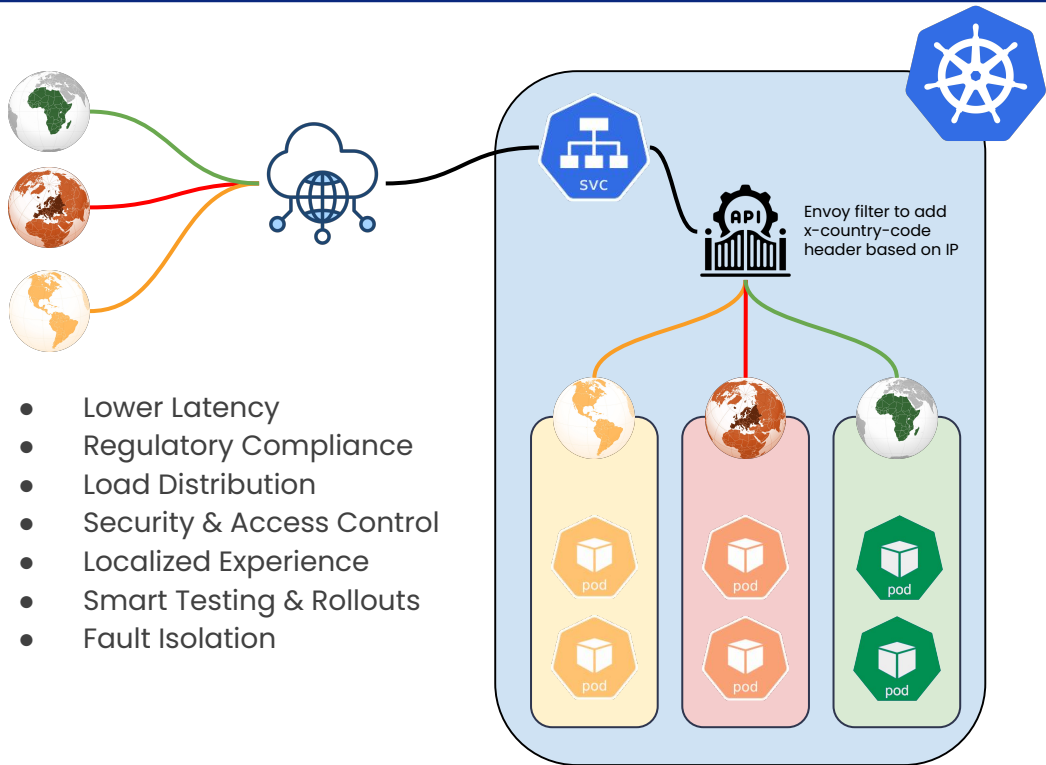
Weighted Traffic Splitting



- Distribute traffic across multiple service versions
- More flexible than simple canary (can split across multiple versions)
- Example: 70% v1 and 30% v2

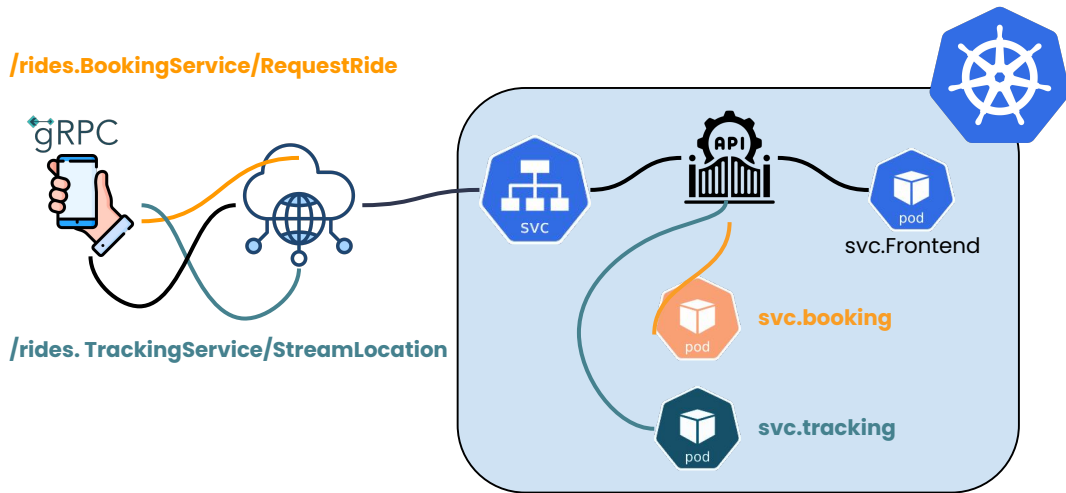
```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: basic-weighted-split
spec:
  parentRefs:
    - name: envoy-gateway
  rules:
    - backendRefs:
        - name: service-v1
          port: 80
          weight: 70
        - name: service-v2
          port: 80
          weight: 30
```


Geographic Routing



```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: geo-routing
spec:
  parentRefs:
  - name: envoy-gateway
  rules:
  - matches:
    - headers:
      - name: "x-geo-region"
        value: "africa"
    backendRefs:
      - name: africa-service
        port: 80
  - matches:
    - headers:
      - name: "x-geo-region"
        value: "europe"
    backendRefs:
      - name: europe-service
        port: 80
  - matches:
    - headers:
      - name: "x-geo-region"
        value: "america"
    backendRefs:
      - name: america-service
        port: 80
  - backendRefs: # Default route
    - name: default-service
      port: 80
```

gRPC Routing



- Your mobile app sends gRPC requests directly to Envoy Gateway.
- Envoy routes each method to the right microservice (no frontend involved).
- Result: Faster, more efficient, and scalable than traditional REST-through-frontend approaches.
- gRPC also supports all the previous http deployments we have covered and many more.

```
hostnames:  
- "api.rideshare.com"  
rules:  
- matches:  
  - method:  
    service: "rides.BookingService"  
    method: "RequestRide"  
  backendRefs:  
    - name: booking-service  
      port: 50051  
      weight: 100 # 100% to  
        booking-service  
  
- matches:  
  - method:  
    service: "rides.TrackingService"  
    method: "StreamLocation"  
  backendRefs:  
    - name: tracking-service  
      port: 50052  
      weight: 100 # 100% to  
        tracking-service
```

2.2

Traffic Shaping

Traffic Shaping

Rate Limiting

Global (external service) and local (in-process) request throttling.
Supports requests per second (RPS) or connection limits.

Circuit Breaking

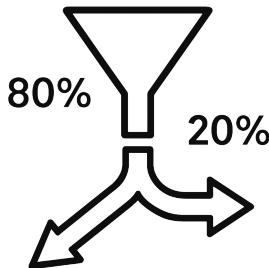
Automatically blocks traffic to overwhelmed services (based on errors/timeouts).
Configurable thresholds for max connections, pending requests, etc.

Retry Policies

Controls retry attempts for failed requests (with backoff strategies).
Can filter retries based on status codes/gRPC codes.

Timeouts

Sets deadlines for requests (global, per-route, or per-cluster).



Load Shedding

Drops or queues requests when upstream services are overloaded.

Request Buffering

Delays or buffers requests (e.g., for streaming or batch processing).

Bandwidth Limits

Throttles bandwidth for HTTP/TCP streams (bytes per second).

Fault Injection

Simulates failures (aborts/delays) to test resilience.

Priority-Based Routing

Directs traffic to high/low-priority clusters based on load.

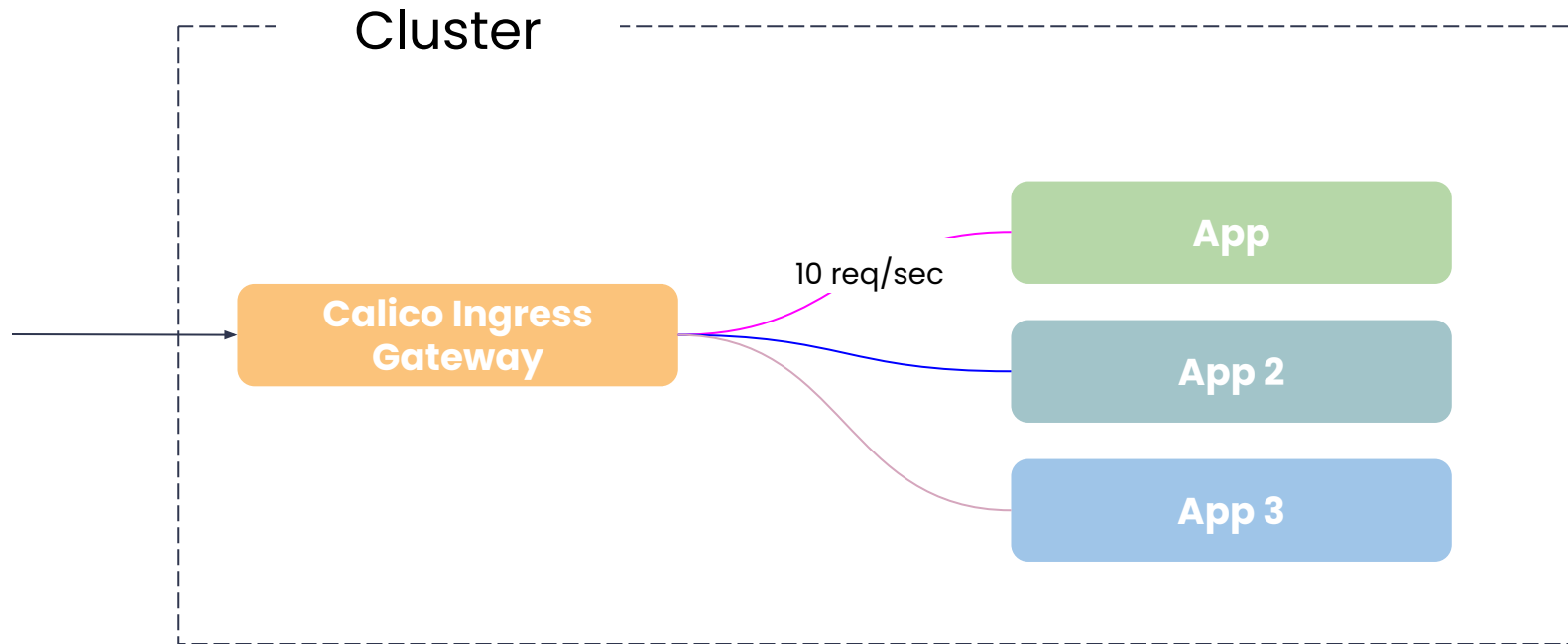
Adaptive Concurrency

Dynamically adjusts request limits based on latency metrics.

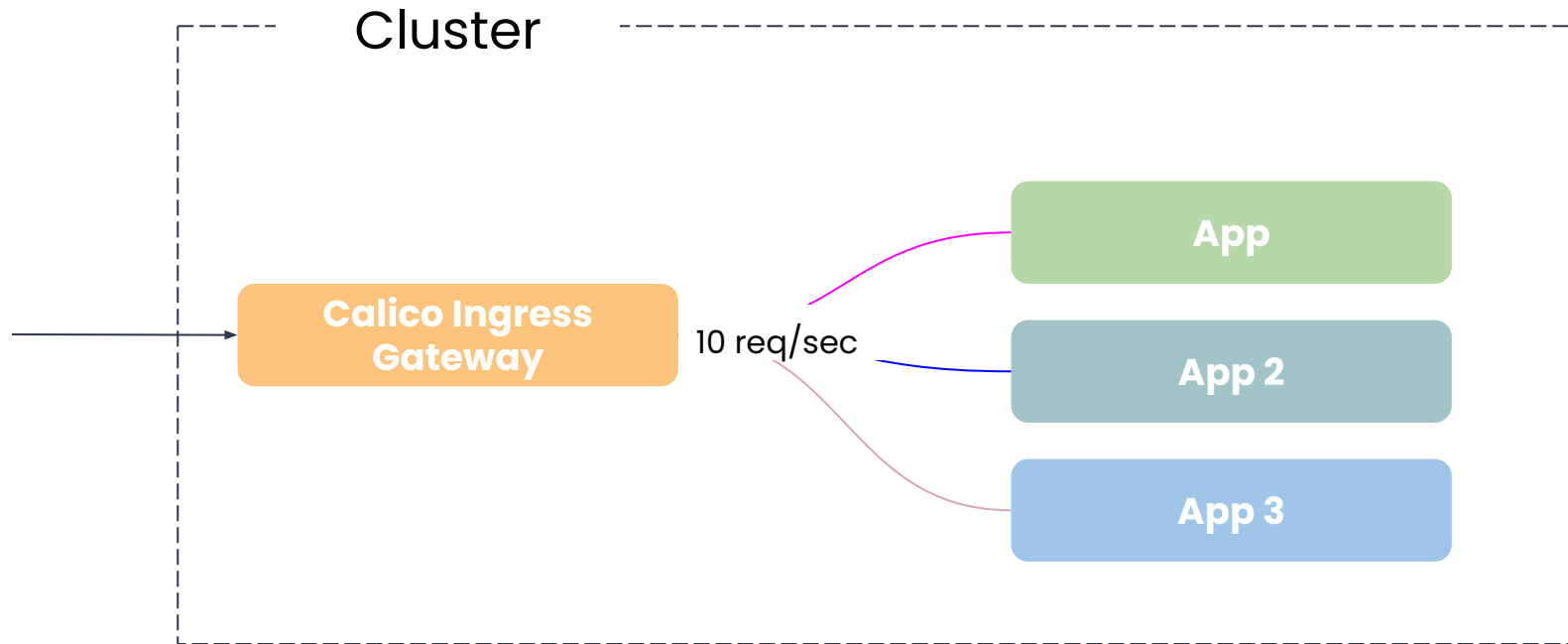
Quota Management

Enforces usage quotas (e.g., API calls/user).

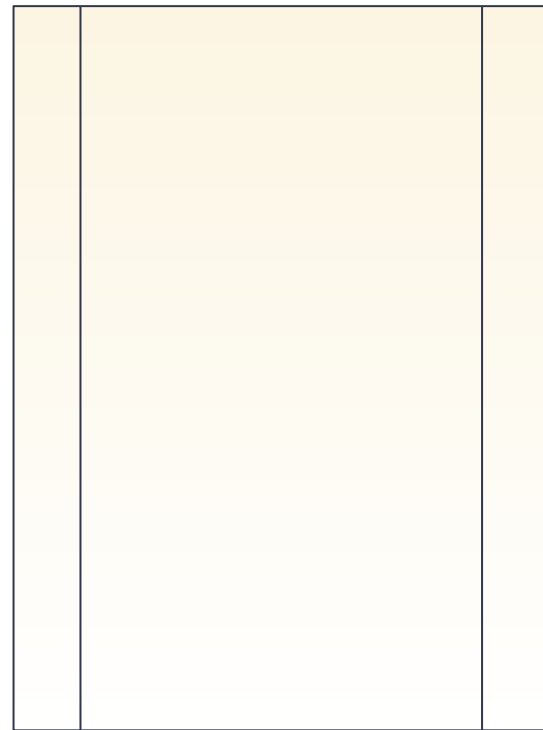
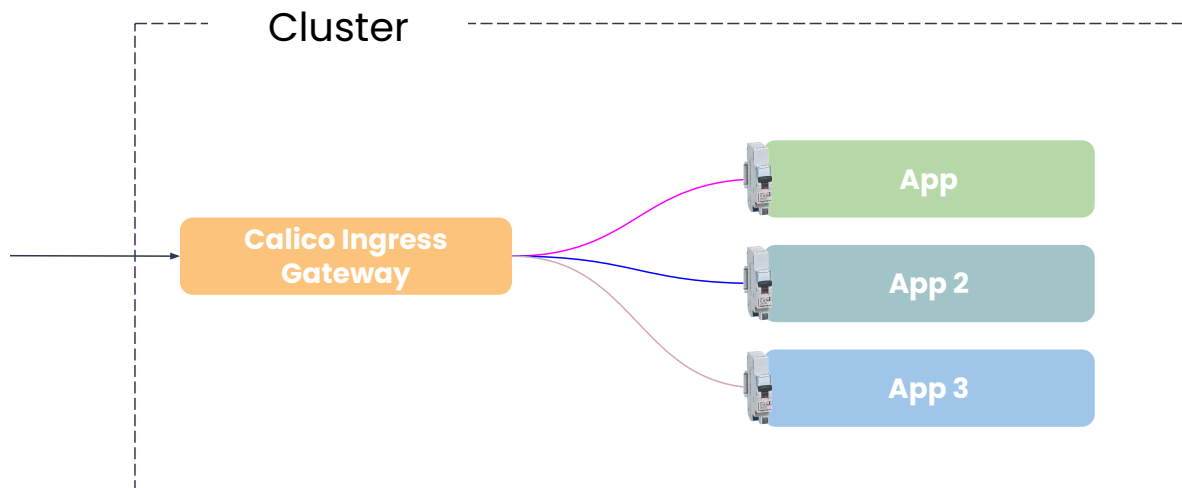
Local Rate Limiting



Global Rate Limiting



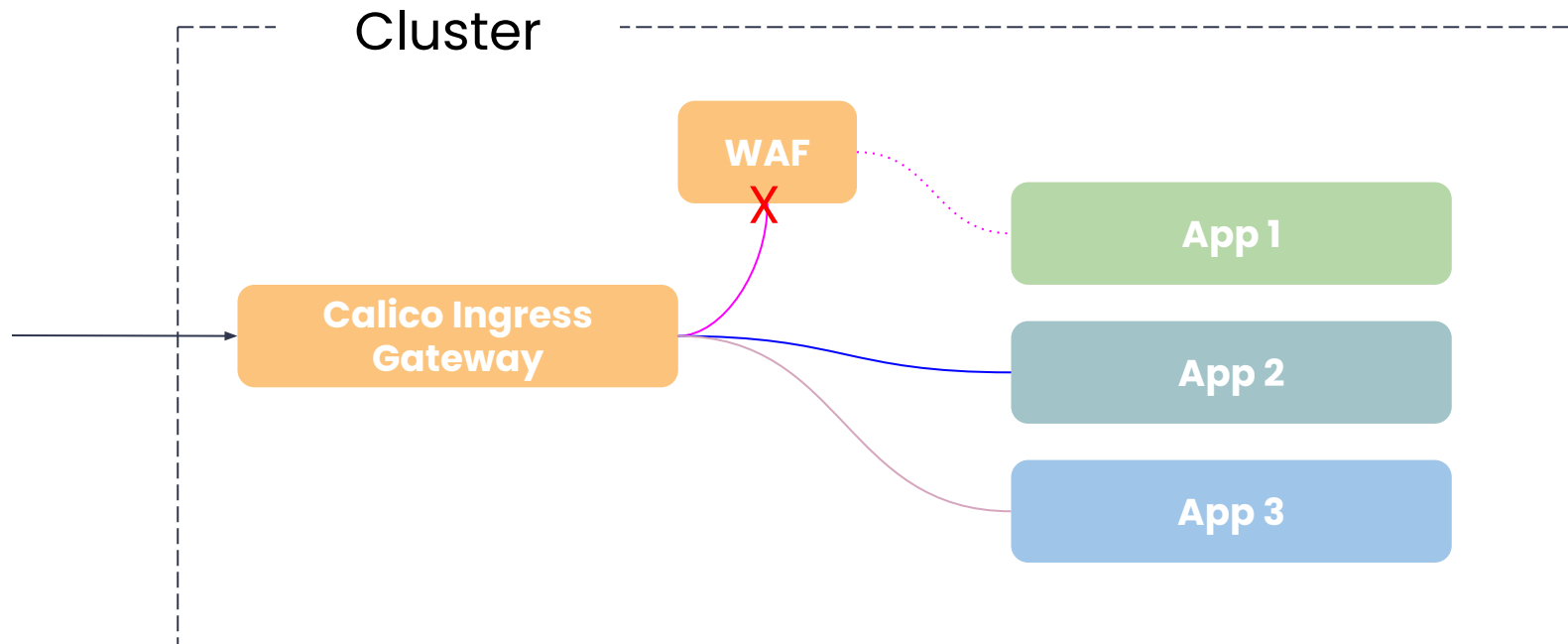
Circuit Breaking



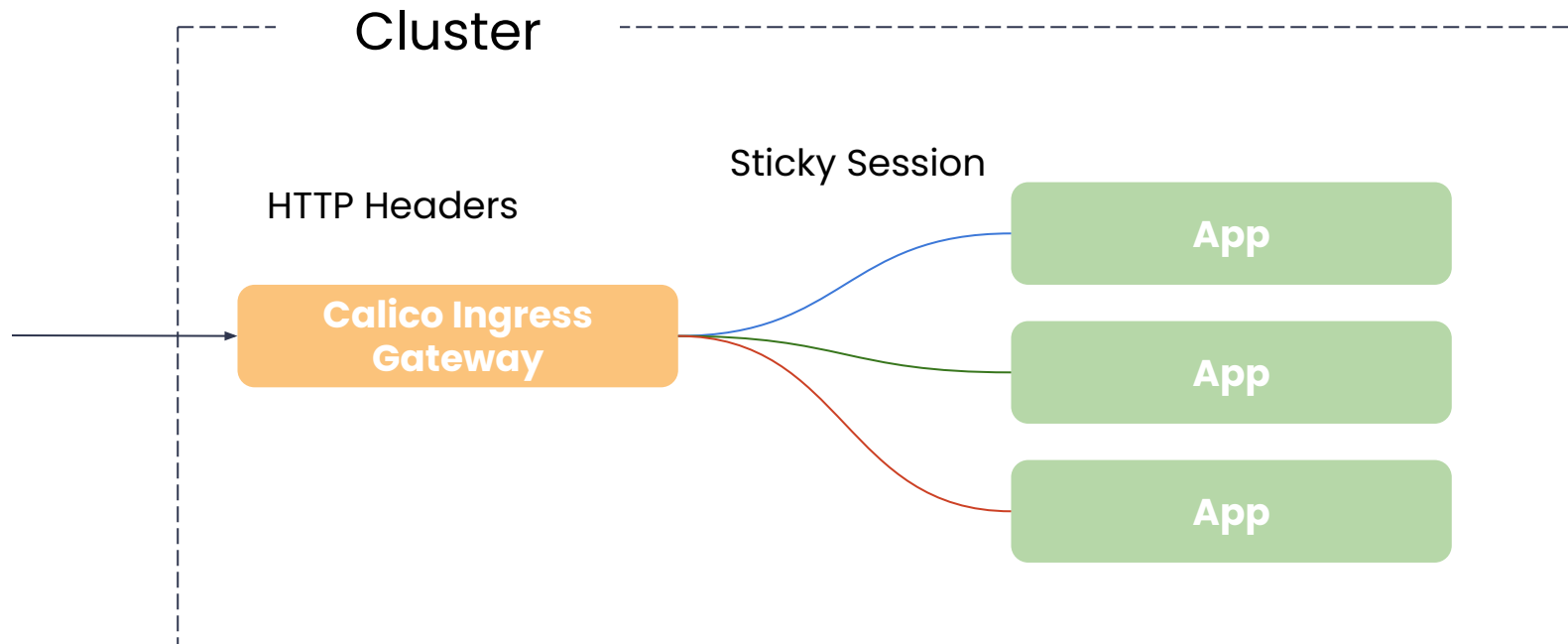
2.3

Other Traffic Functions

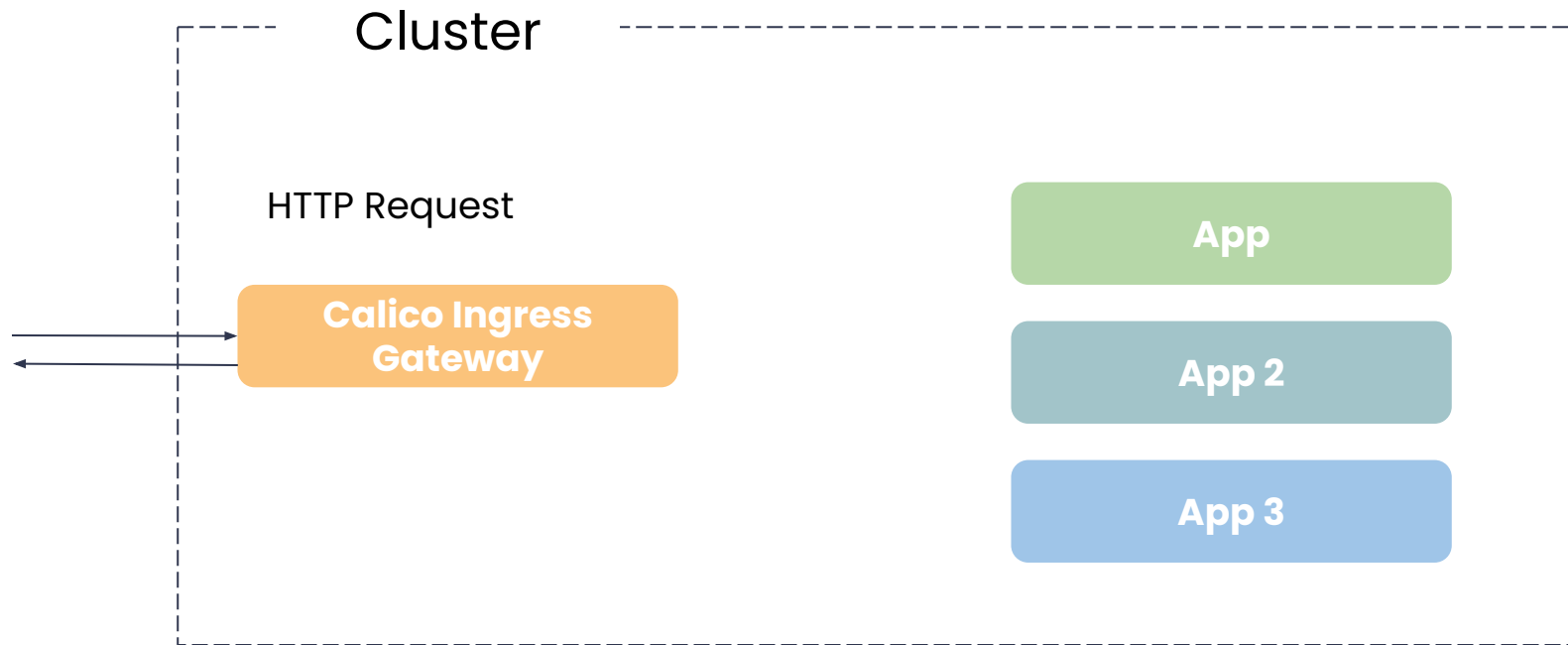
Coming Soon – WAF



Advanced Load Balancing – Consistent Hash



Direct Response



And lots and lots more...

- Circuit Breaker
- Backend Routing
- Client Traffic Policy
- Connection Limit
- Direct Response
- Failover
- Fault Injection
- GRPC Routing
- HTTP Redirects
- HTTP Request Headers
- HTTP Response Headers
- HTTP Timeouts
- HTTP URL Rewrite
- HTTP Request Mirroring
- Multicluster Service Routing
- Response Compression
- Response Override
- Retry
- Accelerated TLS Handshakes
- API Key Authentication
- Backend Mutual TLS: Gateway to Backend
- Backend TLS: Gateway to Backend
- Basic Authentication
- CORS
- External Authorization
- IP Allowlist/Denylist
- JWT Authentication
- JWT Claim-Based Authorization
- Mutual TLS: External Clients to the Gateway
- OIDC Authentication
- Threat Model
- TLS Passthrough
- TLS Termination for TCP

03

Security & Compliance

04

Monitoring & Debugging

Thank you



TIGERA

Follow us on:

