Using Ambassador to Build Cloud-Native Applications

Steve Flanders

CloudNativeCon China, Shanghai 2019

omnition

About Me

Steve Flanders

Head of Product and Experience at Omnition

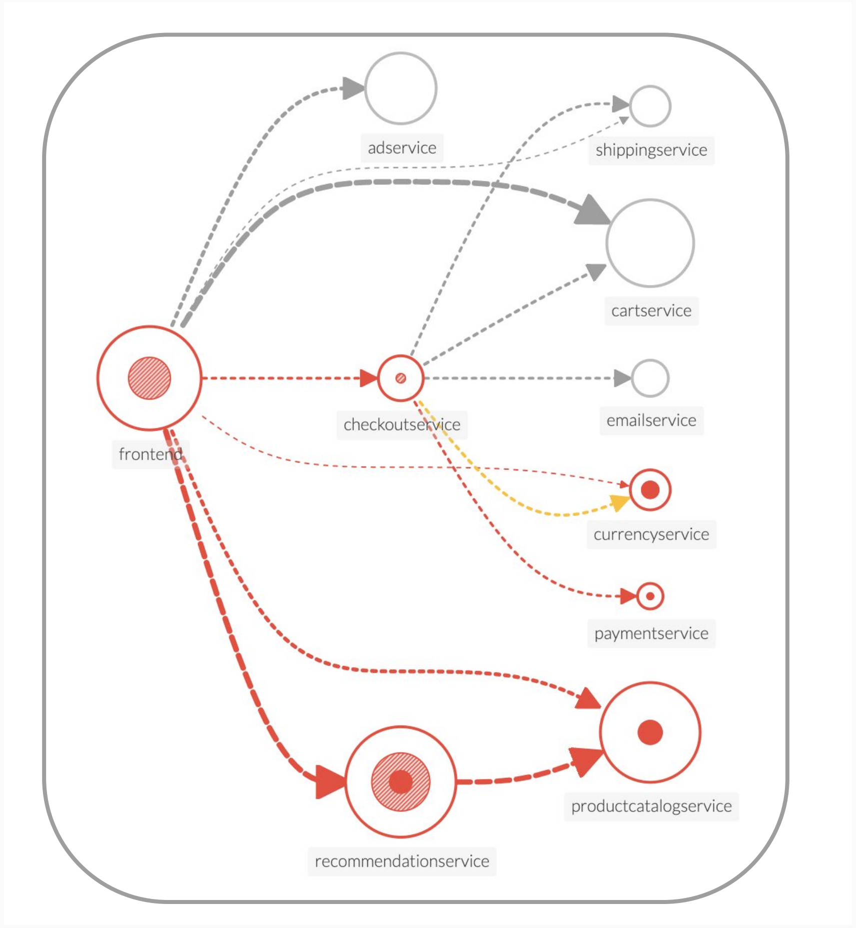
@smflanders | https://sflanders.net

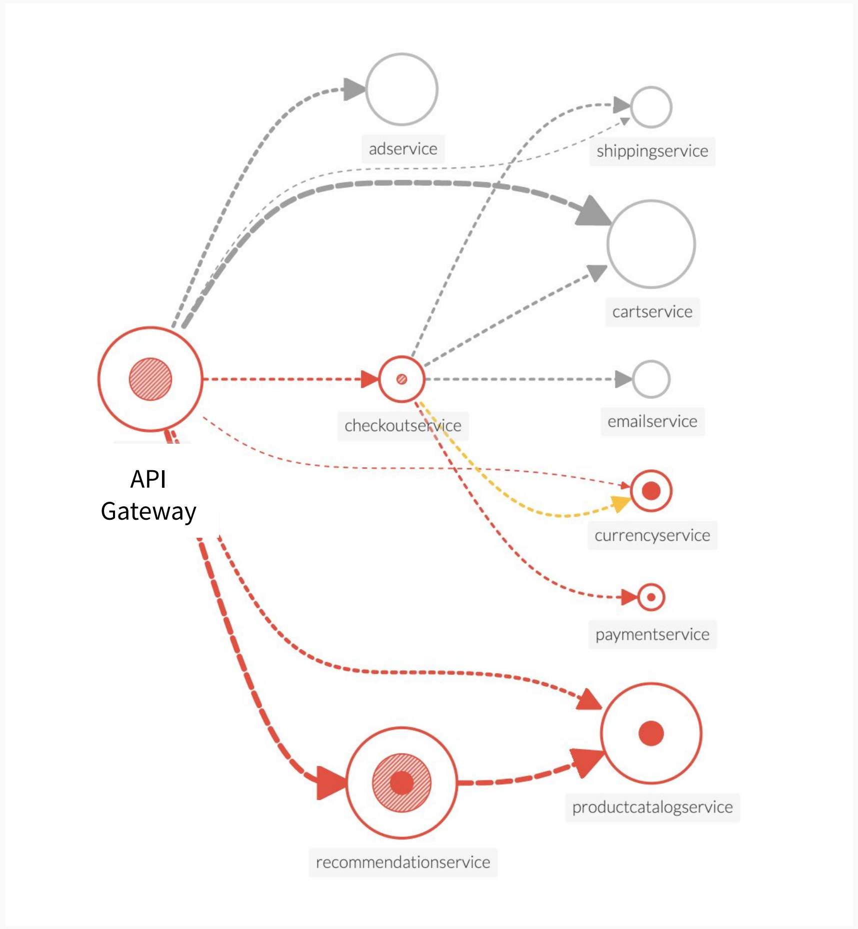
omnition

Agenda

- Background
  - The move from monolith to microservice-based architecture
  - An introduction to API gateways
  - Kubernetes and what it provides
- Ambassador
  - What is it?
  - How does it work?
  - What does it provide?
- Cloud-Native Applications
  - API gateway use-cases
  - Real world examples
  - Configuration examples
  - Lessons learned

Background


adservice
shippingservice
cartservice
frontend
checkoutservice
emailservice
currencyservice
paymentservice
recommendationservice
productcatalogservice


adservice
shippingservice
cartservice
API Gateway
checkoutservice
emailservice
currencyservice
paymentservice
recommendationservice
productcatalogservice

Kubernetes and what it provides

From https://kubernetes.io/: Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications.

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: ambassador
spec:
  type: LoadBalancer
  externalTrafficPolicy: Local
  ports:
   - port: 80
     targetPort: 8080
  selector:
    service: ambassador
```

omnition

Ambassador

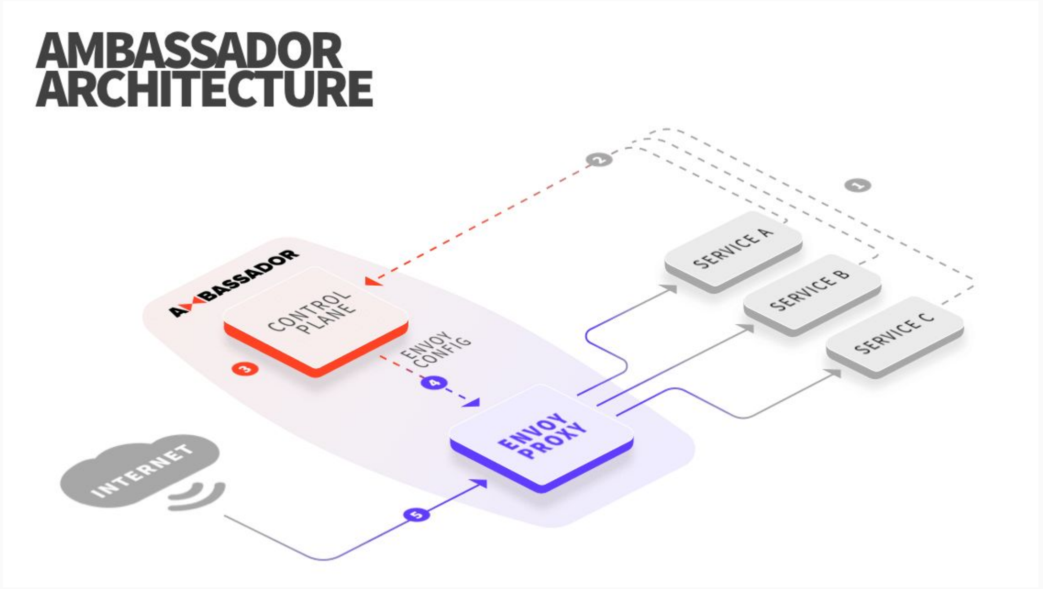
What is it?

Ambassador	● An open source, Kubernetes-native microservices API gateway built on the Envoy Proxy.
Envoy	● An open source edge and service proxy, designed for cloud-native applications.

Put another way: Envoy is a “data plane” and Ambassador is a “control plane”

omnition

How does it work?



1. Service owner defines configuration in Kubernetes manifests.
2. Kubernetes API notifies Ambassador of manifest changes.
3. Ambassador parses the change and transforms the configuration to a semantic intermediate representation. Envoy configuration is generated from this IR.
4. The new configuration is passed to Envoy via the gRPC-based Aggregated Discovery Service (ADS) API.
5. Traffic flows through the reconfigured Envoy, without dropping any connections.

omnition

What does it provide?

Support for:

- Self-Service via Kubernetes Annotations
- Kubernetes-Native Architecture
- Istio Integration
- Flexible Canary Deployments

Features:

- gRPC and HTTP/2 Support
- Authentication
- Rate Limiting
- Integrated Diagnostics

omnition

How is it different?

Alternatives fall in three basic categories:

1. Hosted API gateways
   (e.g. Amazon API gateway)
2. Traditional API gateways
   (e.g. Kong)
3. L7 proxies
   (e.g. Traefik, NGINX, HAProxy, or Envoy, or Ingress controllers built on these proxies)

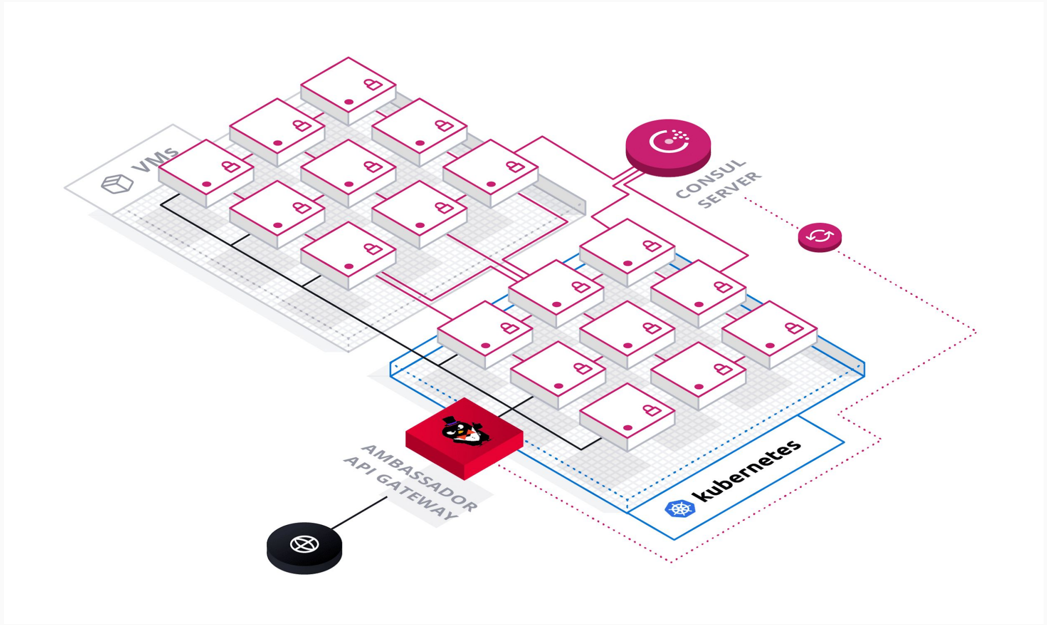
Ambassador differences:

- No vendor lock-in (e.g. hosted gateways)
- No dependency on external database
  (e.g. Kong)
- Self service and Kubernetes native
- Leverages Envoy

omnition

Cloud-Native Applications

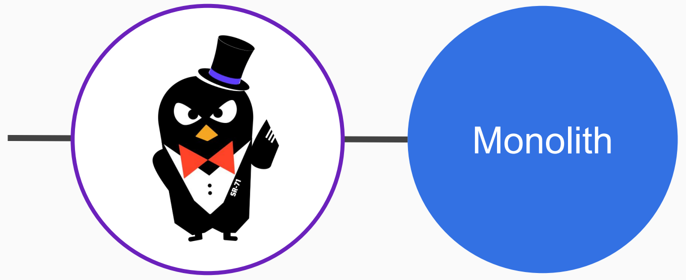
Use-case 1: Edge (North/South) Routing



- Ability to control/route ingress traffic
- Offload requirements such as
  - Authentication (e.g. require all ingress traffic to be authenticated)
  - Encryption (e.g. TLS termination and pass-through)
  - Retries and timeouts

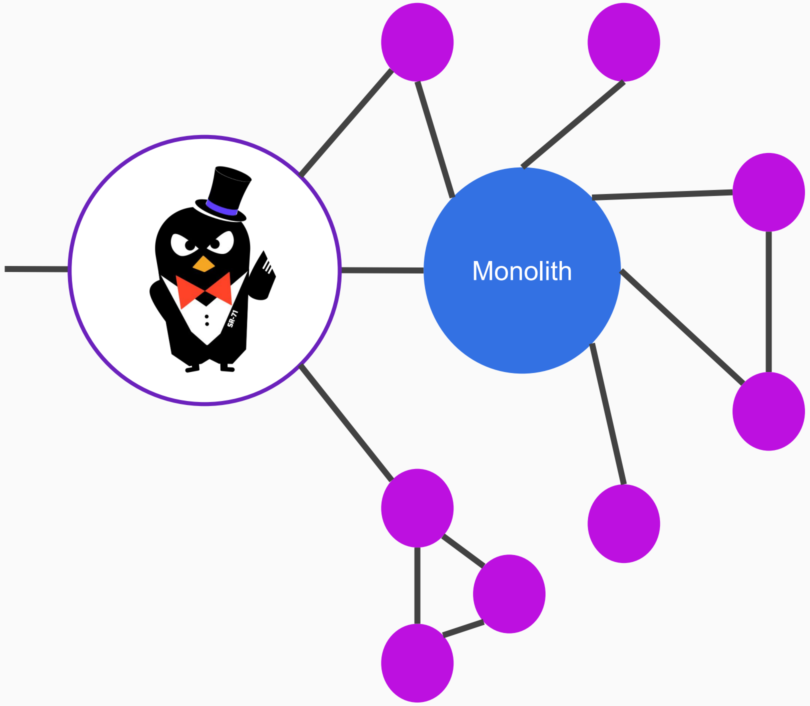
omnition

Real World Example: Monolith to microservices



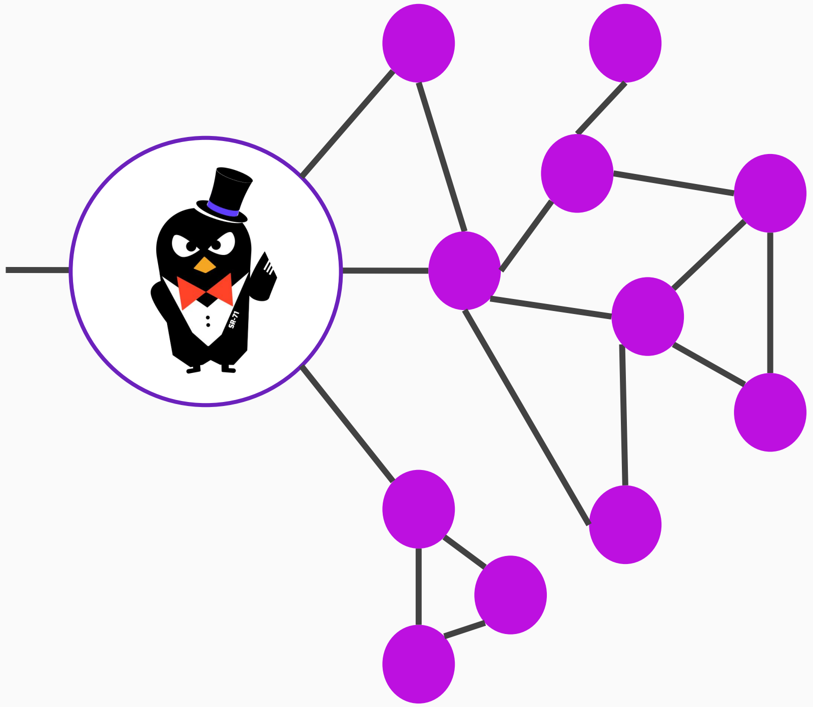
omnition

Real World Example: Monolith to microservices



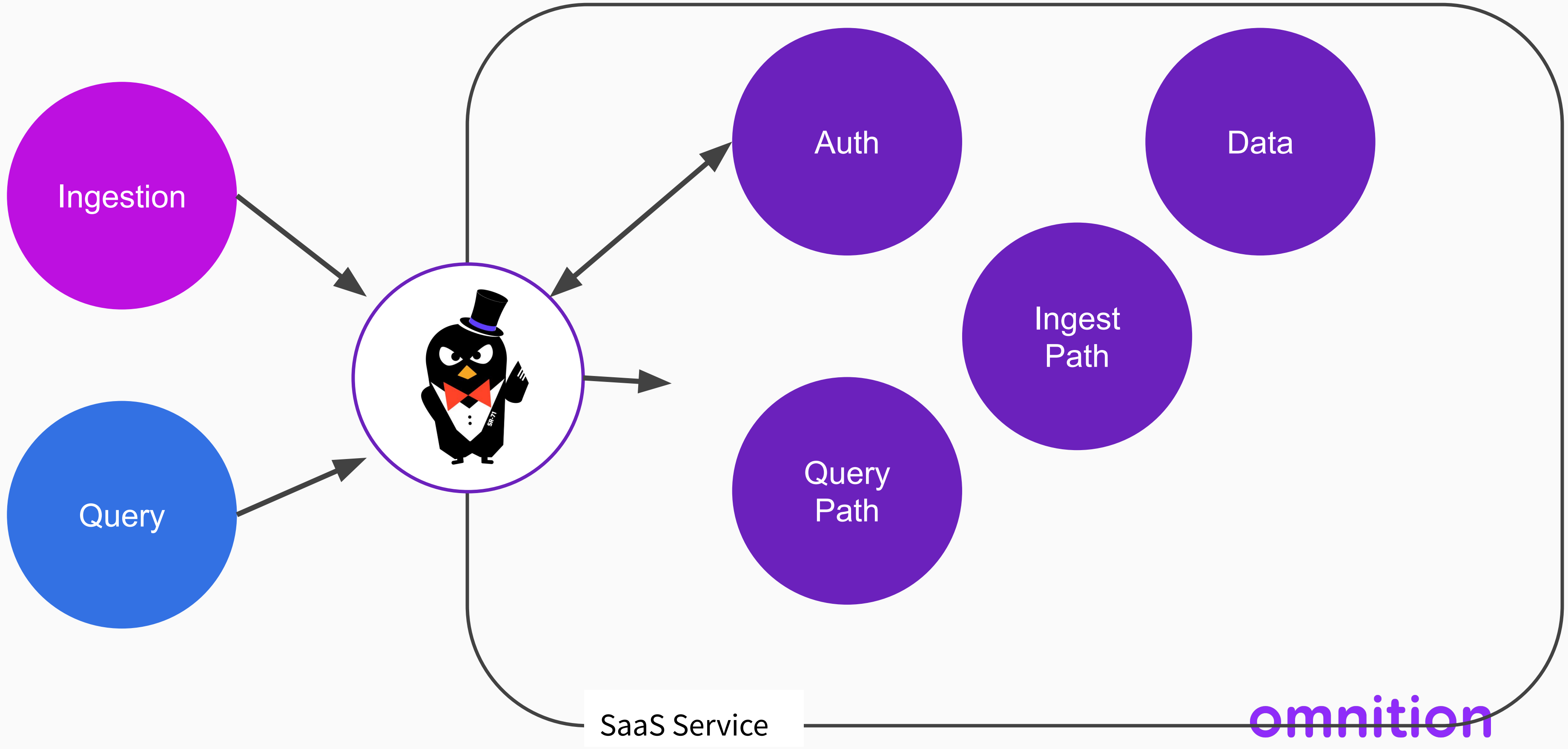
omnition

Real World Example: Monolith to microservices



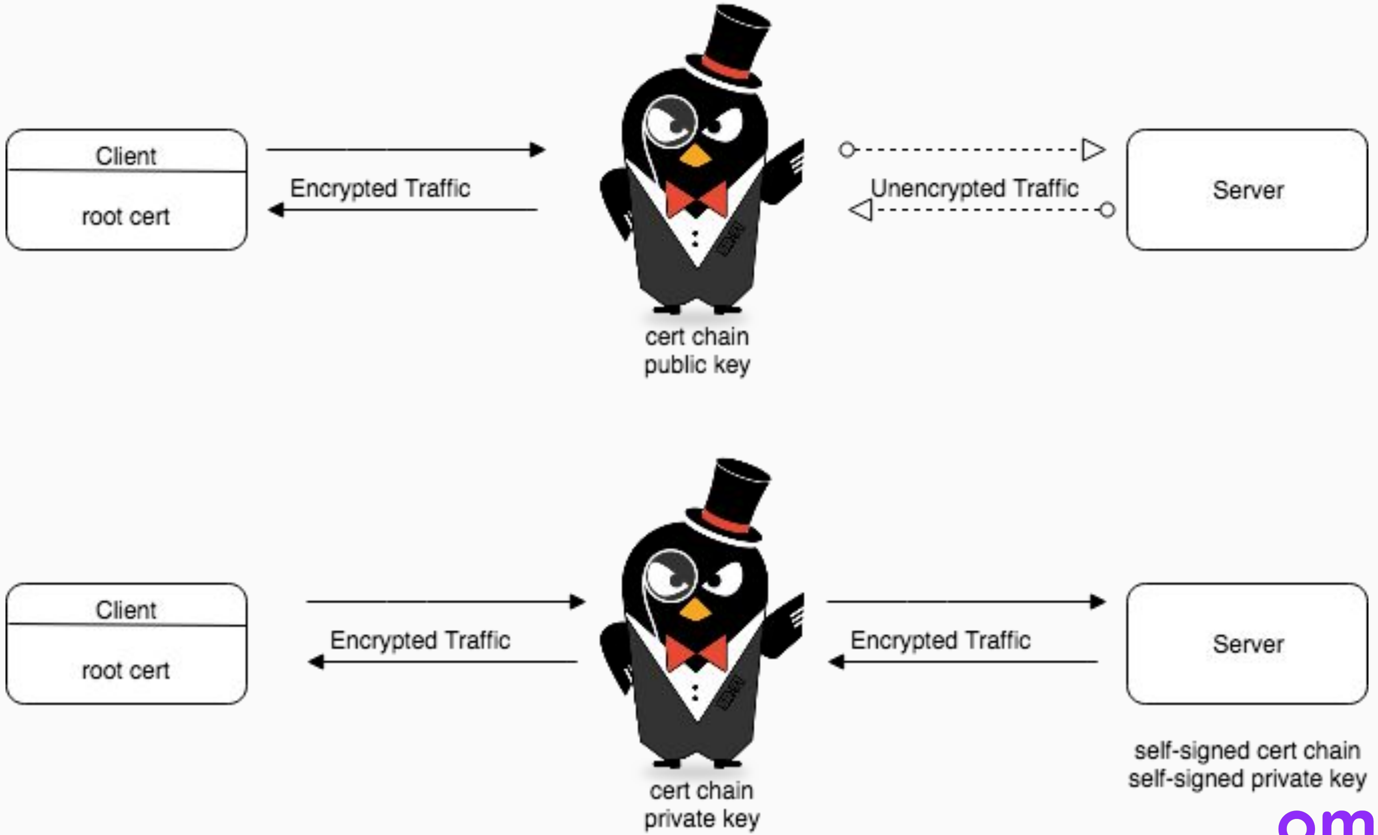
omnition

Real World Example: SaaS Service



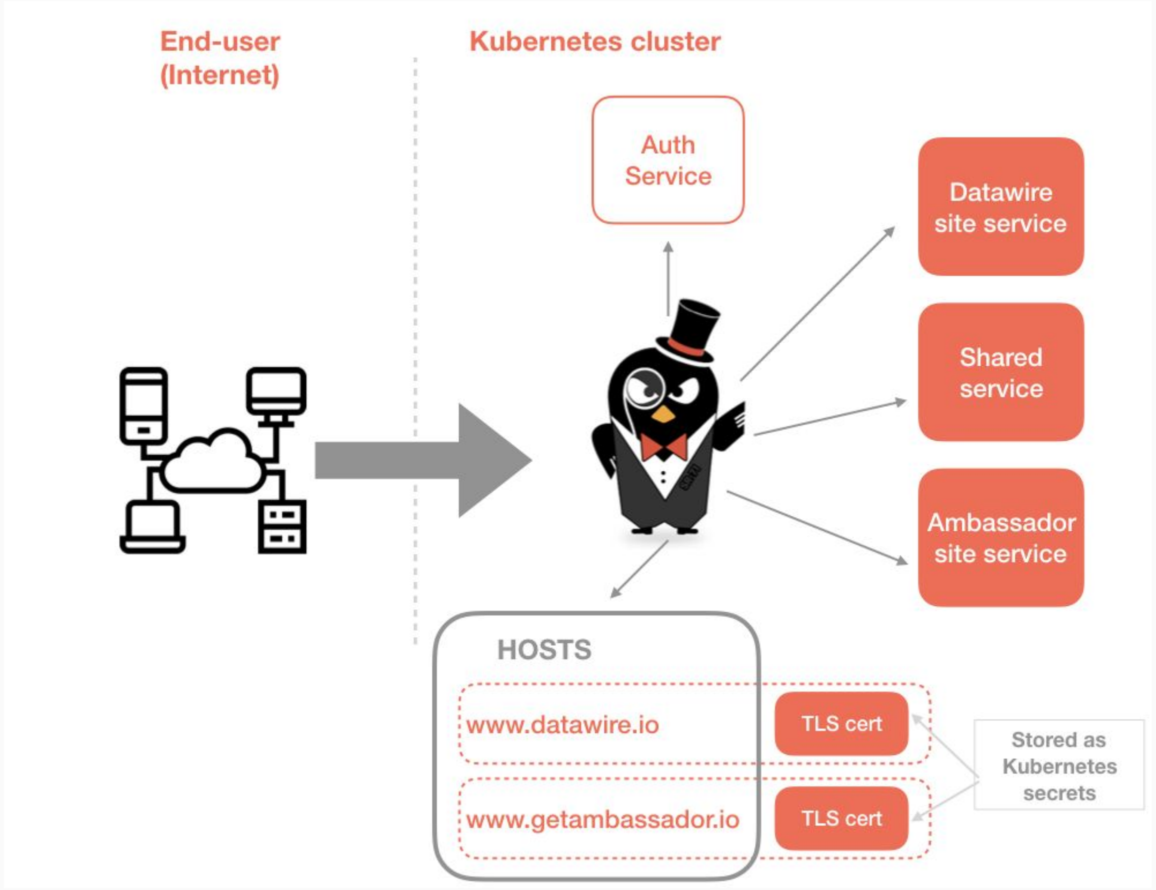
omnition

Understanding TLS



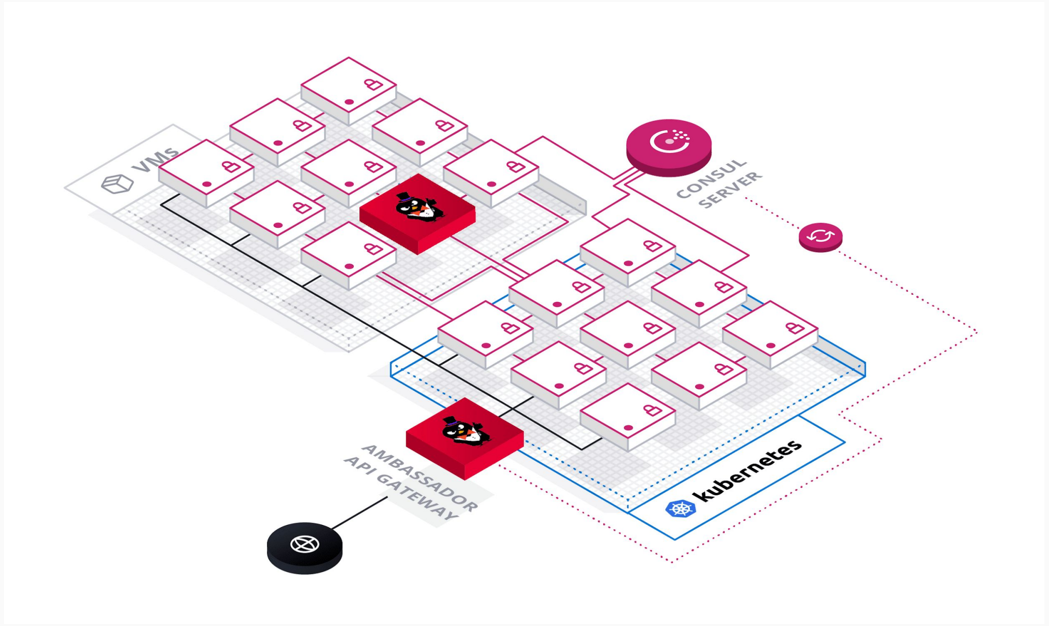
omnition

Understanding TLS and SNI



omnition

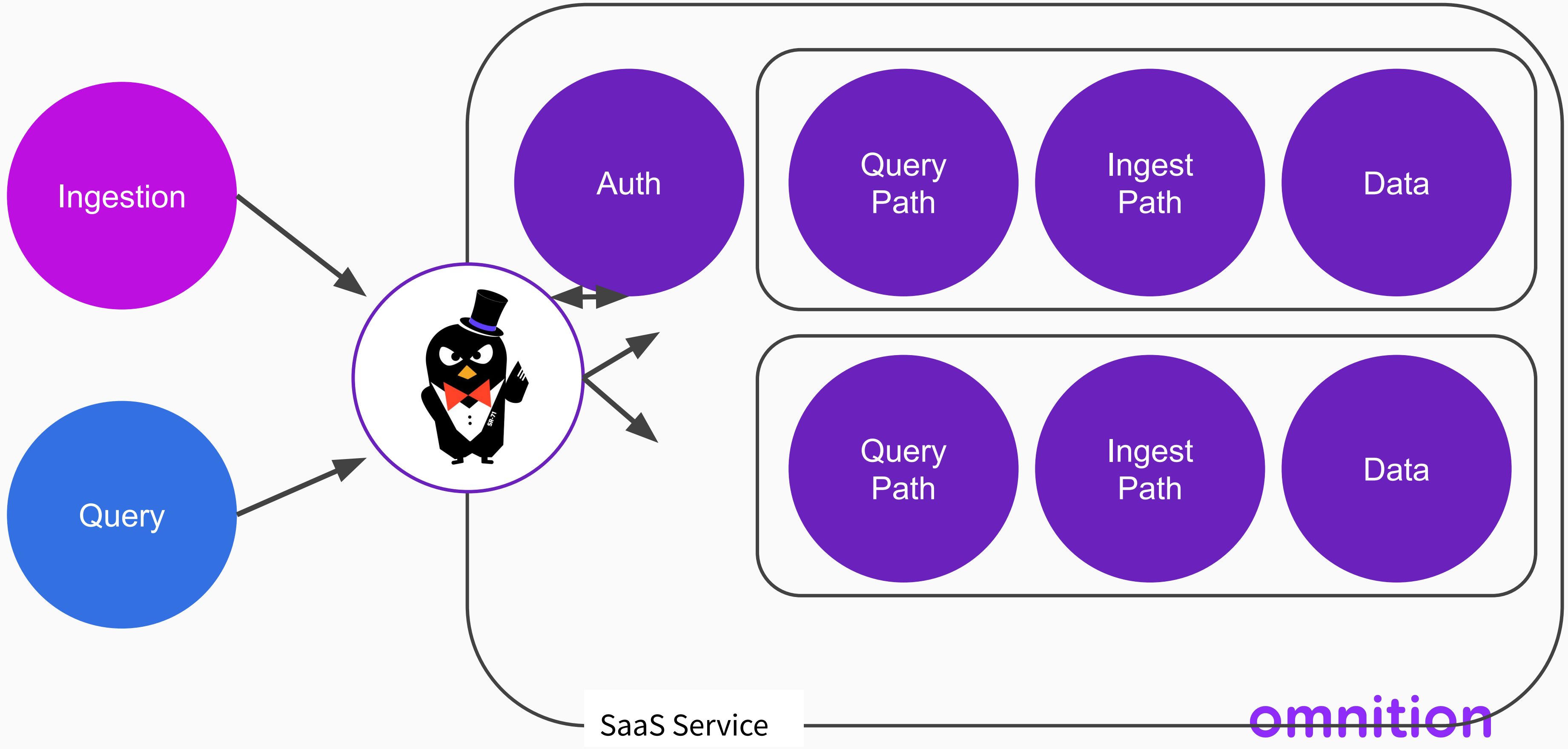
Use-case 2: Internal (North/South) Routing



- Ability to control/route multi-tenant traffic
- Offload requirements such as
    - Mapping (e.g. based on headers)
    - Retries and timeouts

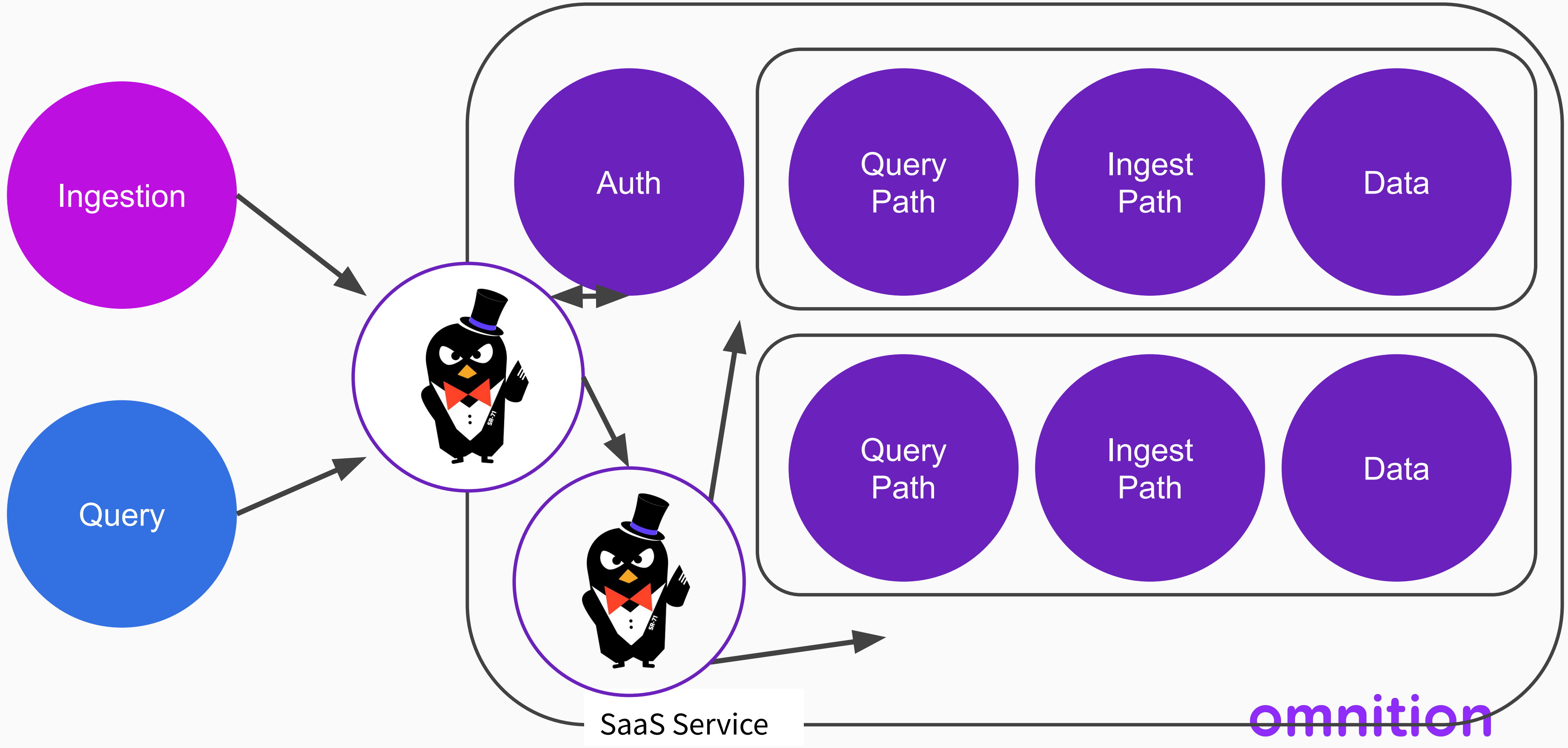
omnition

Real World Example: SaaS Service



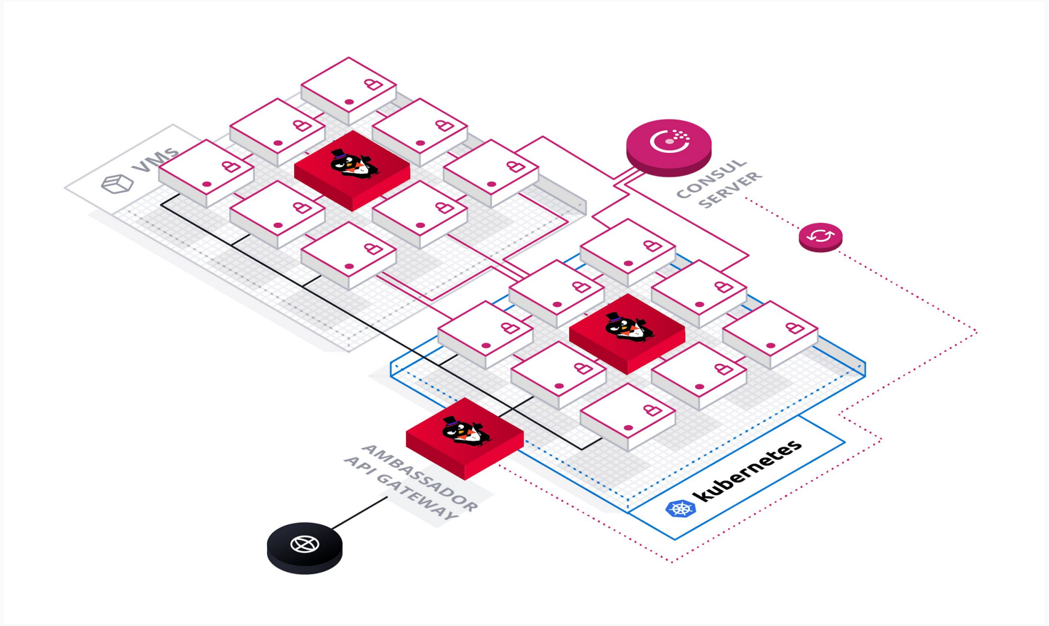
omnition

Real World Example: SaaS Service



omnition

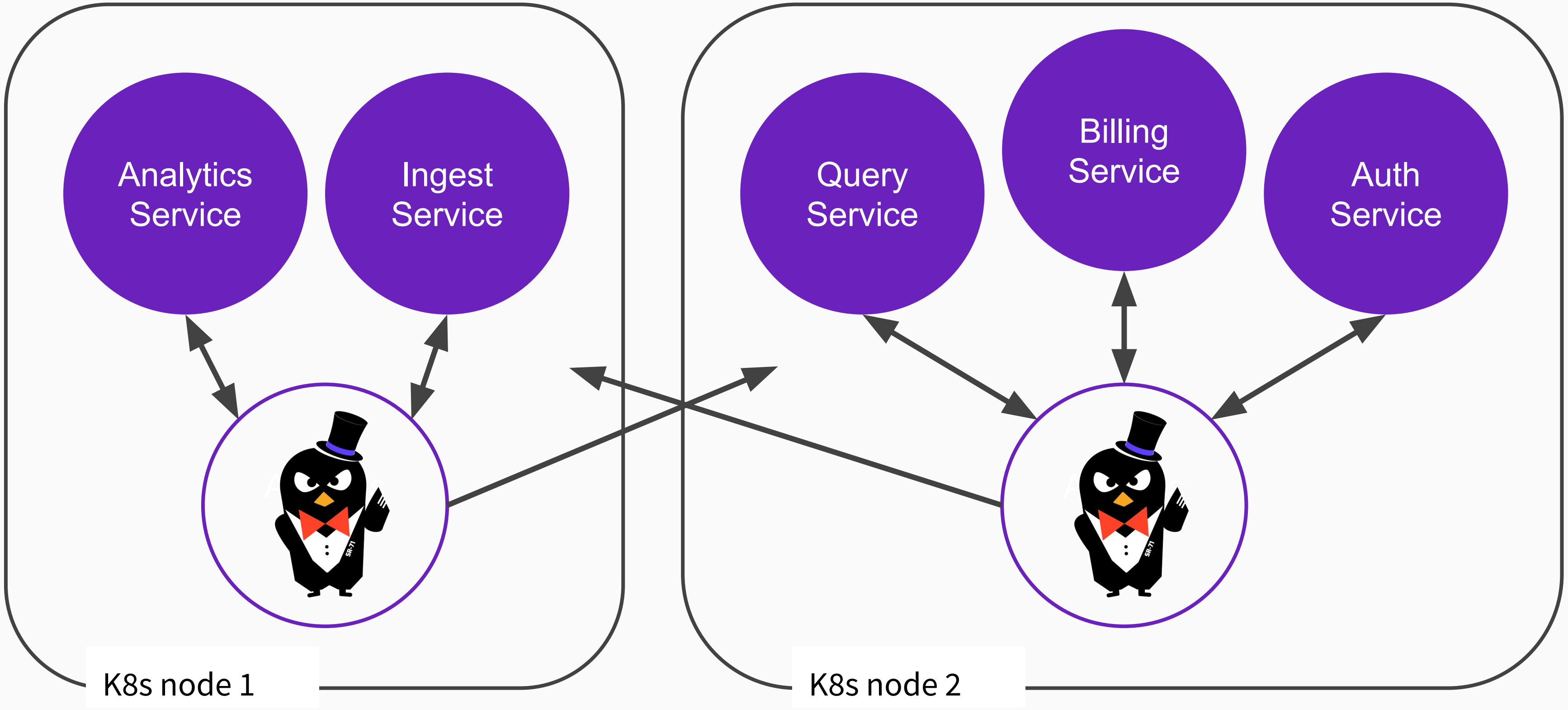
Use-case 3: Internal (East/West) Routing



- Ability to control/route any traffic
- Offload requirements such as
    - Service discovery
    - Load balancing
    - Access control

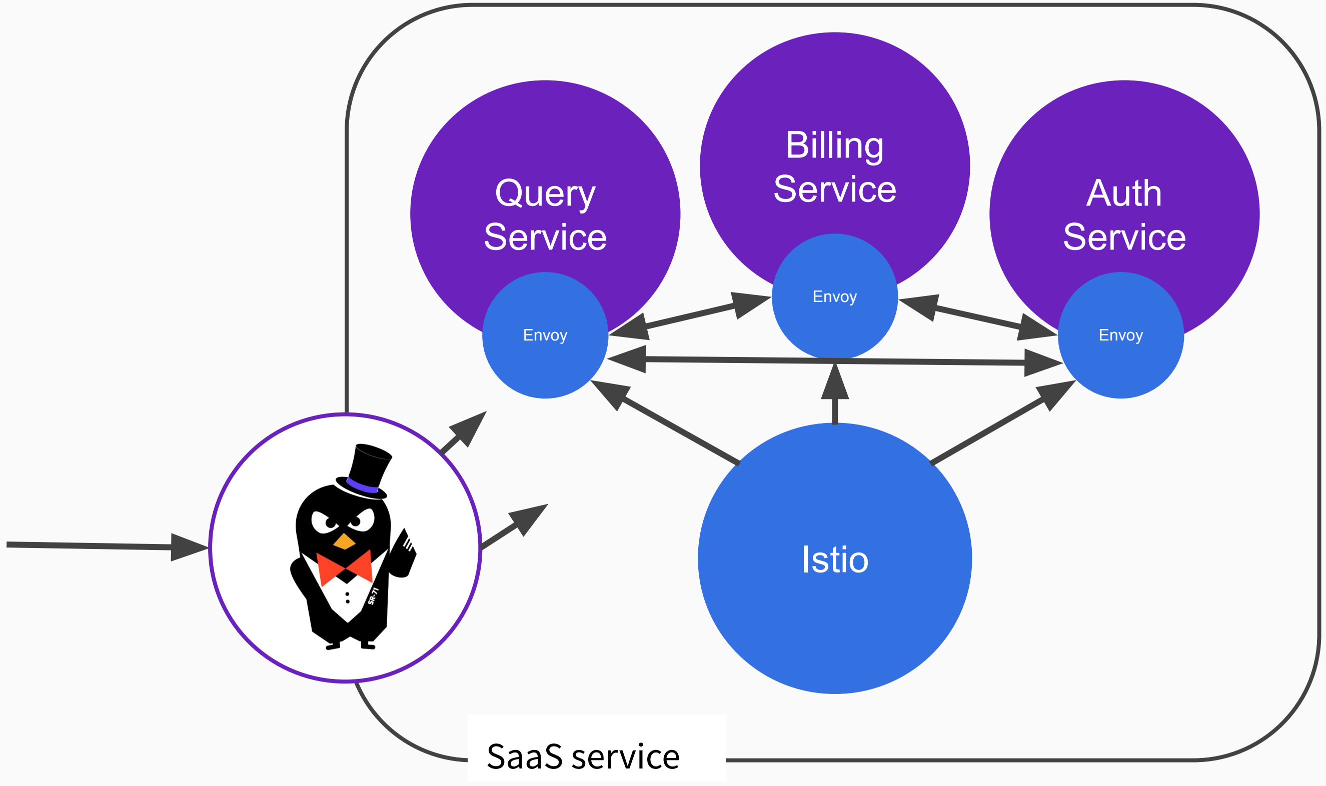
omnition

Real World Example: SaaS Service



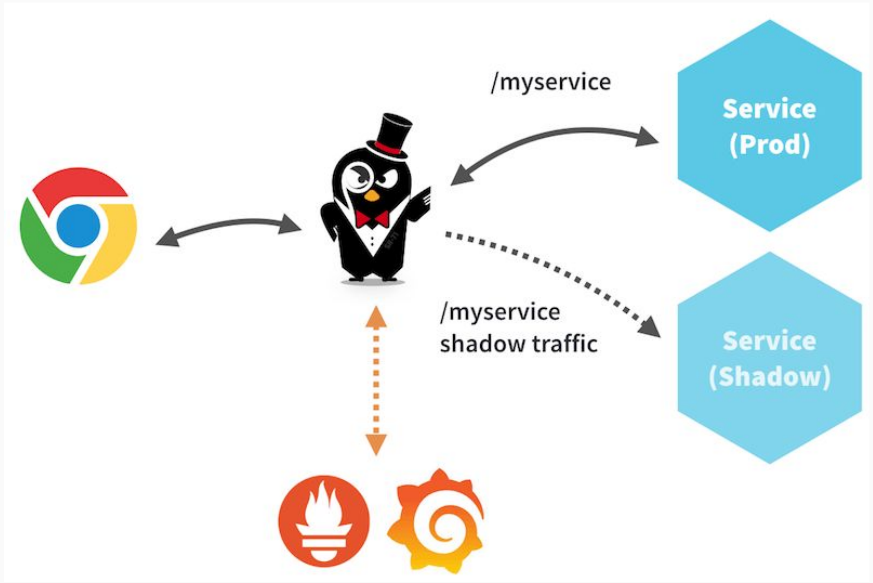
omnition

Real World Example: Service Mesh



omnition

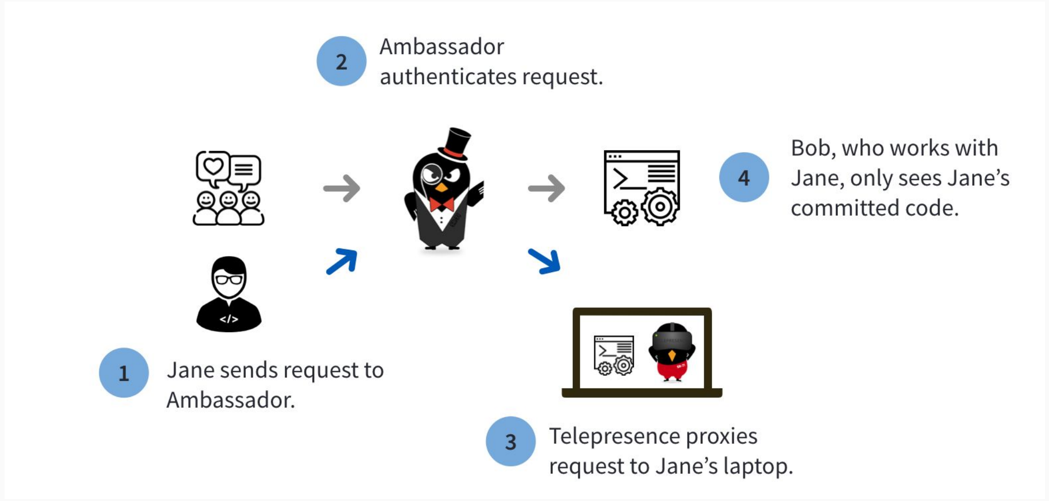
Use-case 4: Traffic Shadowing



- Ability to test code/releases
- Leverage real traffic/load
- Minimize duplicated resources

omnition

Use-case 5: Software Development Testing with Telepresence (paid)



2 Ambassador authenticates request.
4 Bob, who works with Jane, only sees Jane's committed code.
1 Jane sends request to Ambassador.
3 Telepresence proxies request to Jane's laptop.

- Ability to test in production
- Ability to leverage production without impacting production

omnition

Configuration

Options by version

< 0.50.0

- Configmaps
- Annotations

>= 0.50.0

- Annotations

>= 0.70.0

- Annotations
- CRDs

omnition

Encryption

Annotations (0.70.0 and older)

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    getambassador.io/config: |
      ---
      apiVersion: ambassador/v0
      kind:  Module
      name:  tls
      ambassador_id: myID
      config:
        server:
          enabled: True
...
```

CRDs (0.70.0 and NEWER)

```
apiVersion: getambassador.io/v1
kind: Module
metadata:
  name: tls
  namespace: default
spec:
  ambassador_id: myID
  config:
    server:
      enabled: True
...
```

omnition

Authentication

Annotations (0.70.0 and older)

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    getambassador.io/config: |
      ---
      apiVersion: ambassador/v0
      ambassador_id: myID
      kind:  AuthService
      name:  authentication
      auth_service: "auth:3000"
      path_prefix: "/auth/api/check"
      ---
      apiVersion: ambassador/v0
      kind:  Mapping
      ambassador_id: myID
      name: auth
      prefix: /auth/
      rewrite: /auth/
      service: auth:3000
...
```

CRDs (0.70.0 and NEWER)

```
apiVersion: getambassador.io/v1
kind: AuthService
metadata:
  name: authentication
  namespace: default
spec:
  ambassador_id: myID
  auth_service: "auth:3000"
  path_prefix: "/auth/api/check"
---
apiVersion: getambassador.io/v1
kind: Mapping
metadata:
  name: auth-mapping
  namespace: default
spec:
  ambassador_id: myID
  prefix: /auth/
  rewrite: /auth/
  service: auth:3000
```

omnition

Mapping

Annotations (0.70.0 and older)

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    getambassador.io/config: |
      ---
      apiVersion: ambassador/v0
      kind:  Mapping
      name:  omnition-mapping
      ambassador_id: myID
      prefix: /
      host: omnition.io
      service: web.default.svc.cluster.local
...
```

CRDs (0.70.0 and NEWER)

```
apiVersion: getambassador.io/v1
kind: Mapping
metadata:
  name: omnition-mapping
  namespace: default
spec:
  ambassador_id: myID
  prefix: /
  host: omnition.io
  service: web.default.svc.cluster.local
...
```

omnition

Tracing

Annotations (0.70.0 and older)

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    getambassador.io/config: |
      ---
      apiVersion: ambassador/v0
      kind: TracingService
      name: tracing
      ambassador_id: myID
      service: collector.default:9411
      driver: zipkin
...
```

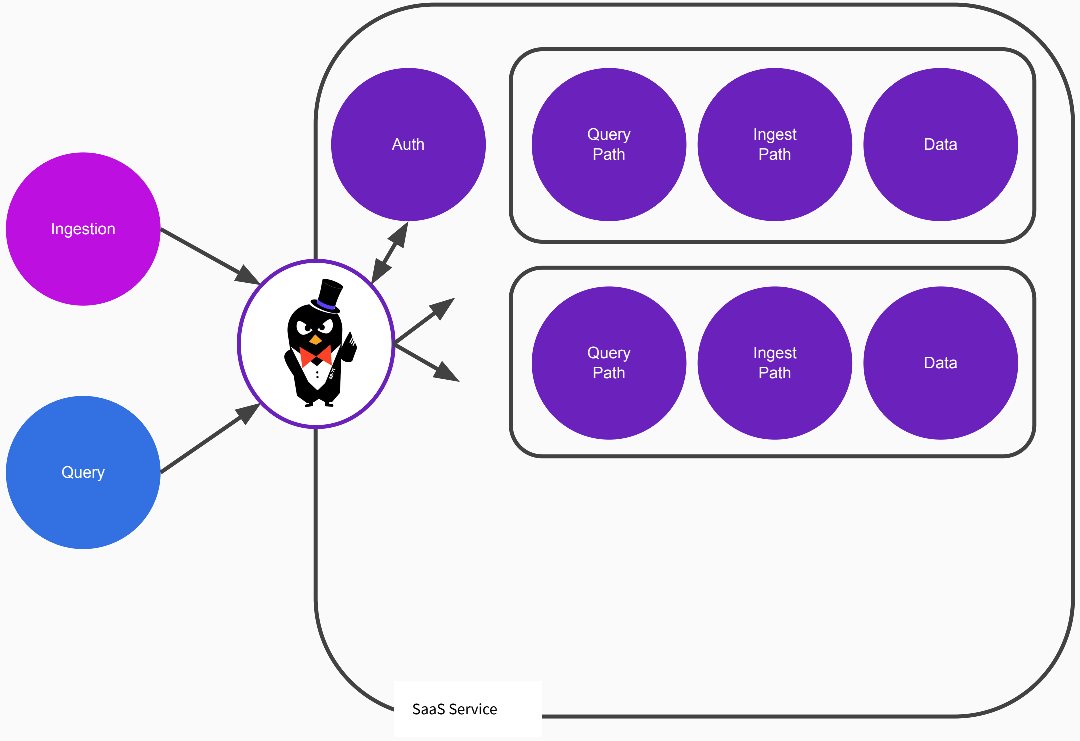
CRDs (0.70.0 and NEWER)

```
apiVersion: getambassador.io/v1
kind: TracingService
metadata:
  name: tracing
  namespace: default
spec:
  ambassador_id: myID
  service: collector.default:9411
  driver: zipkin
...
```

omnition

Lessons Learned

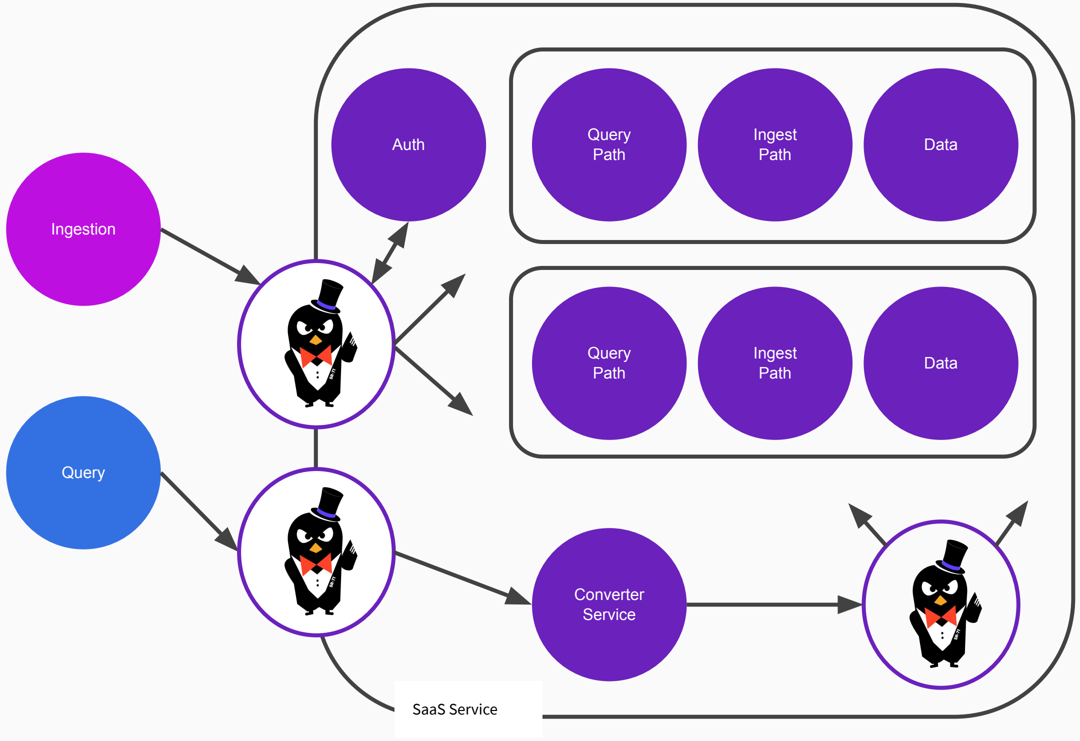
Dedicate Ambassadors per responsibility



- While this is possible and supported it is not ideal (e.g. availability, performance, etc)
- In some circumstances this is not possible (e.g. exposing 80 and 443 from the same Ambassador instance)

omnition

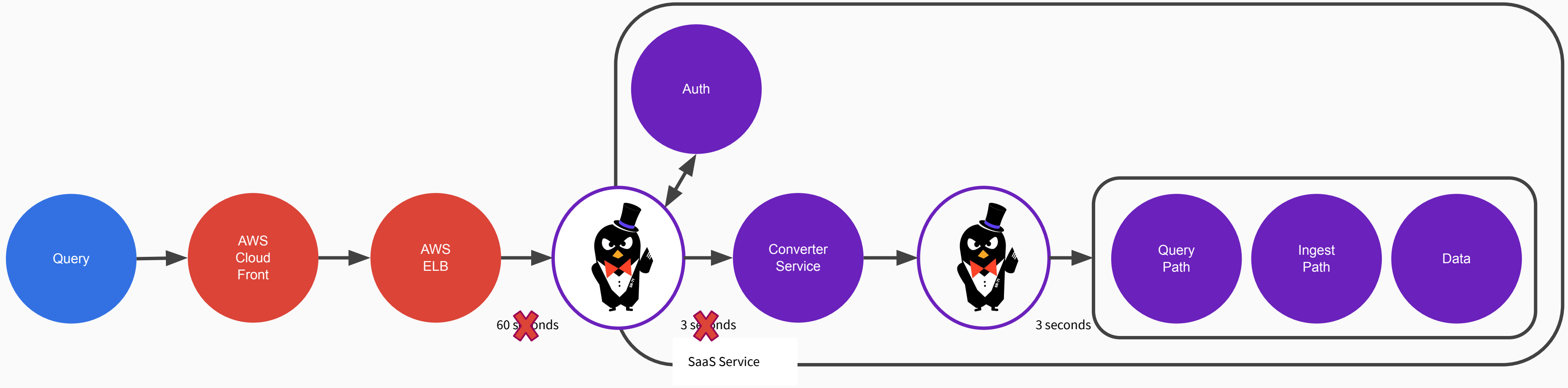
Dedicate Ambassadors per responsibility



- Different Ambassador “clusters” can be created via unique ambassador_id
- Each can be configured and scaled independently

omnition

Understand timeouts



AWS ELB settings:

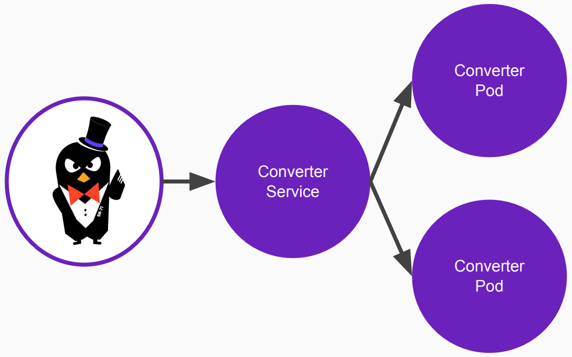
```
service.beta.kubernetes.io/aws-load-balancer-healthcheck-interval: "5"
service.beta.kubernetes.io/aws-load-balancer-healthcheck-timeout: "3"
service.beta.kubernetes.io/aws-load-balancer-healthcheck-unhealthy-threshold: "2"
service.beta.kubernetes.io/aws-load-balancer-healthcheck-healthy-threshold: "2"
```

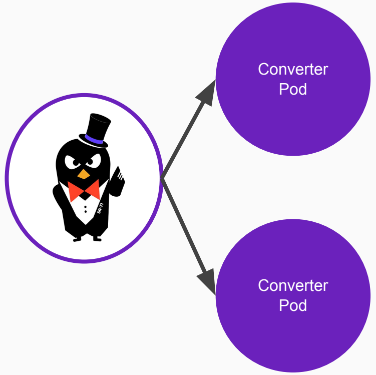
Ambassador timeout settings:

- Request timeout: timeout_ms (default = 3000ms)
- Idle timeout: idle_timeout_ms (default =300000ms)
- Connect timeout: connect_timeout_ms

omnition

Understand service discovery and load balancing



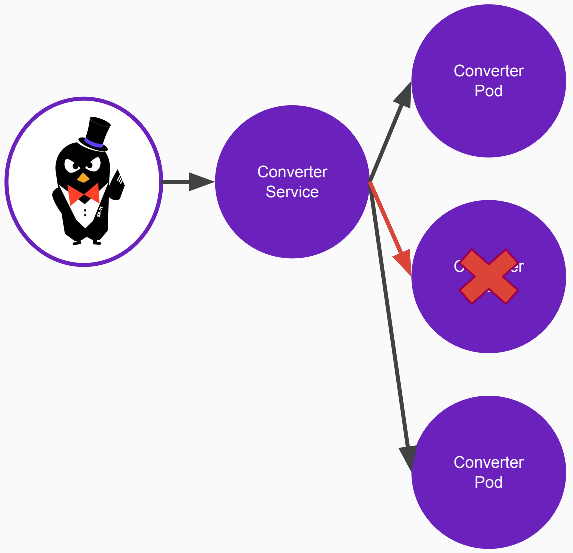


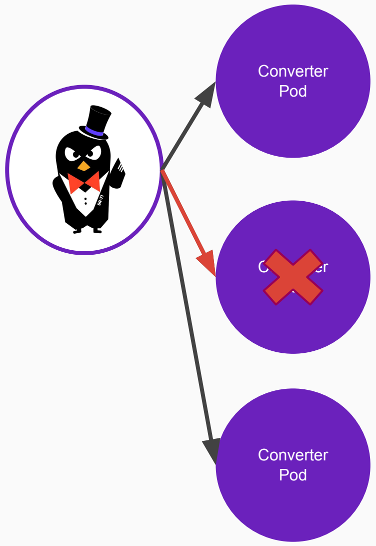
- Typically resolve to k8s service via DNS
  - Use: <service>
  - Use: <service>.<namespace>
  - Not: <service>.<namespace>.svc.cluster.local
- Alternatives
  - K8s endpoint routing
  - Consul routing

- K8s service load balancing
- Alternatives
  - Round robin
  - Ring hash
  - Maglev

omnition

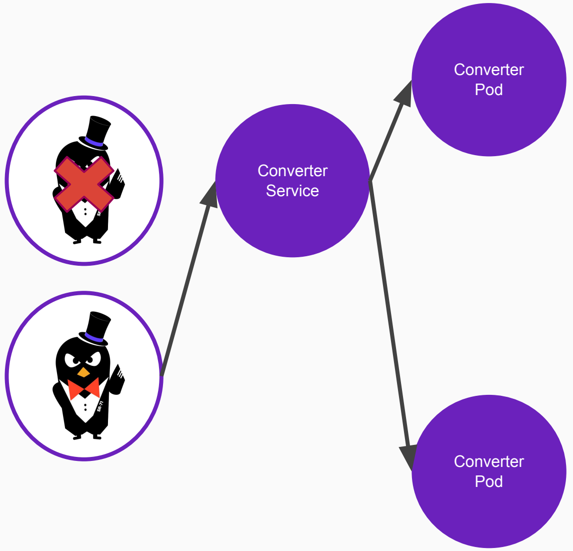
HTTP 503s

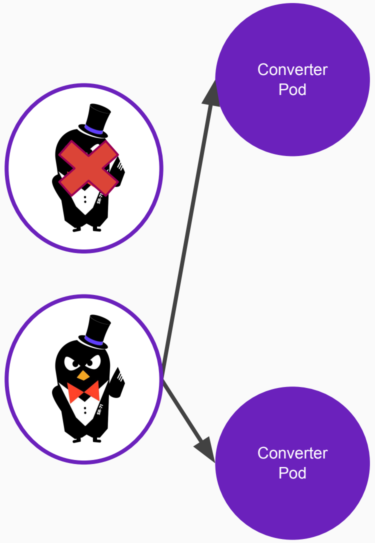




omnition

HTTP 503s





omnition

Others

- Great start, but work/money required
  - How will you handle authentication?
  - How will you handle circuit breaking?
  - How will you handle per-service retries?
- Store configuration with you service
  - Exception for authentication service
  - Everything through CI/CD
- Edge routing is much harder than internal
  - Encryption: end-to-end or terminated?
  - Authentication: which headers?
- Observability - critical, but immature
  - Tracing differs between releases (Envoy)
  - Metrics are not as granular (Envoy)
  - Diagnostic UI is basic and ugly

omnition

Questions?