Handling Advanced Traffic Routing Patterns

Nigel Brown

@n_brownuk www.windsock.io

Module Outline

See how and why we might deploy multiple ingress controllers

Learn how to avoid contention between ingress controllers

Apply annotations to influence ingress controller behavior

Look at some advanced traffic routing patterns using ingress

Ingress API Summary

Ingress API

Provides the basics for managing ingress

Proxy Capabilities

Can we exploit reverse proxy features?

Annotations

Mechanism for applying metadata

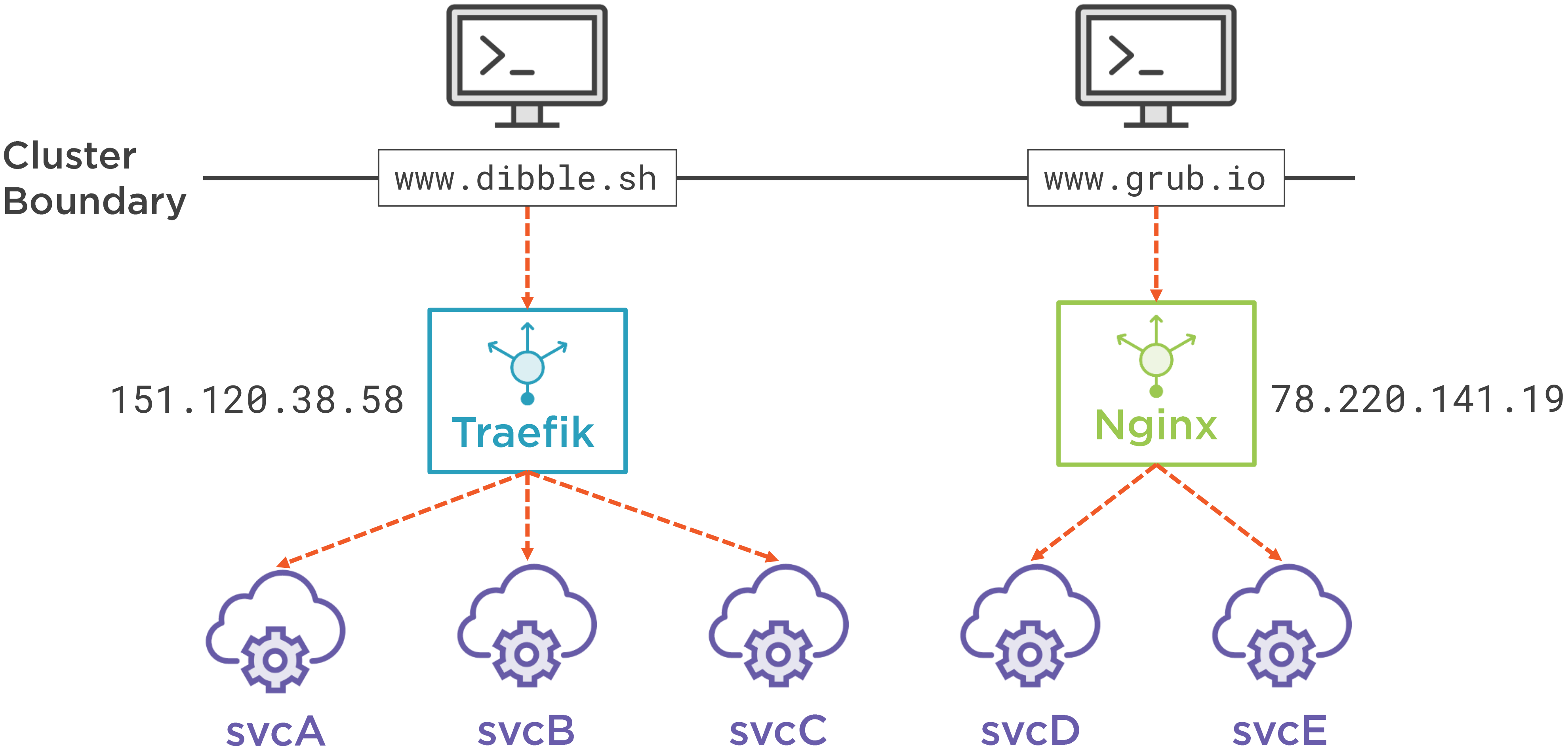
Multiple Ingress Controllers

A cloud provider may already provide a default ingress controller

Perhaps traffic needs to be partitioned for security reasons

Not all ingress controllers provide the same functional capabilities

Deploying Multiple Ingress Controllers



Ingress Controller Contention

Competing ingress controllers will produce unpredictable traffic routing outcomes

Ingress Class

Nginx

Default class: nginx

`--ingress-class`

Traefik

Default class: traefik

`--kubernetes.ingressclass`

Contour

Default class: contour

`--ingress-class-name`

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata
  name: contour-ingress
  annotations:
    kubernetes.io/ingress.class: “contour”
```

Assigning Ingress Objects to Controllers

A missing ingress class annotation may result in controller contention

It’s good practice to annotate ingress definitions with an ingress class

Demo

Deploy two instances of the nginx ingress controller

One handles internal traffic, one handles external traffic

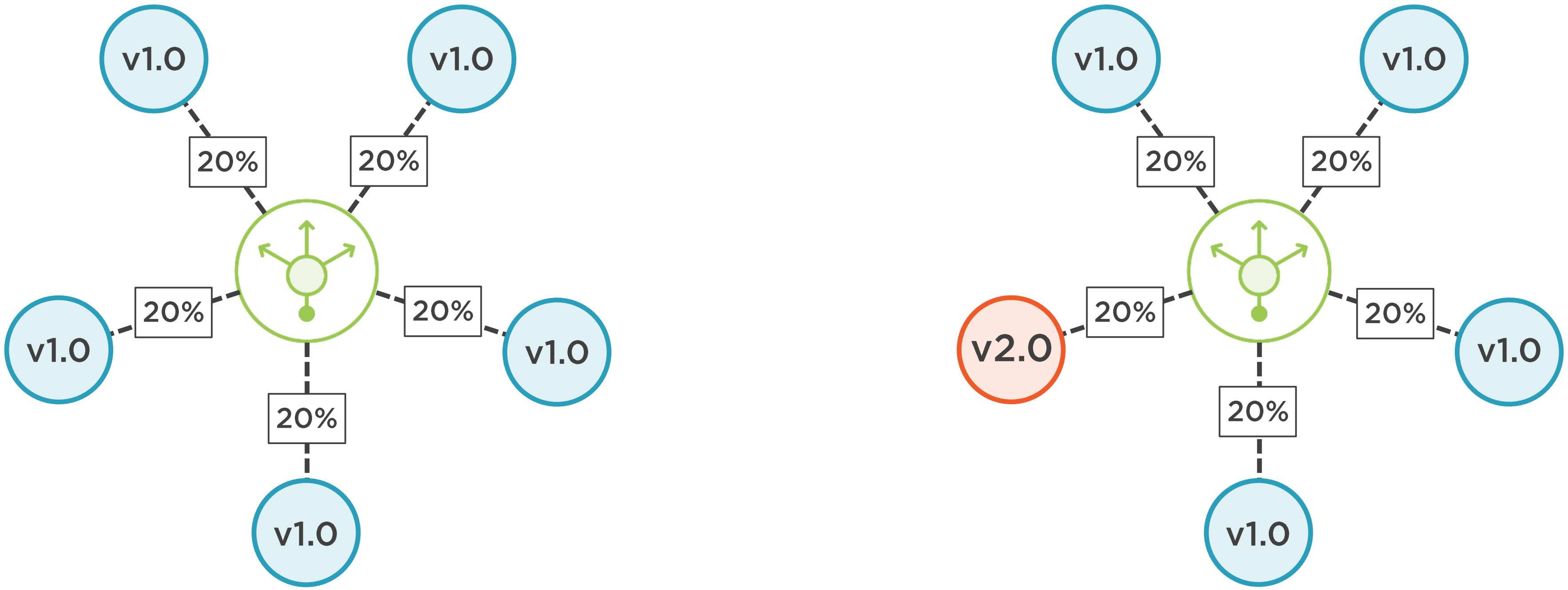
Individual backend services are deployed to serve both traffic types

Each ingress is configured with a class annotation

Canary Release

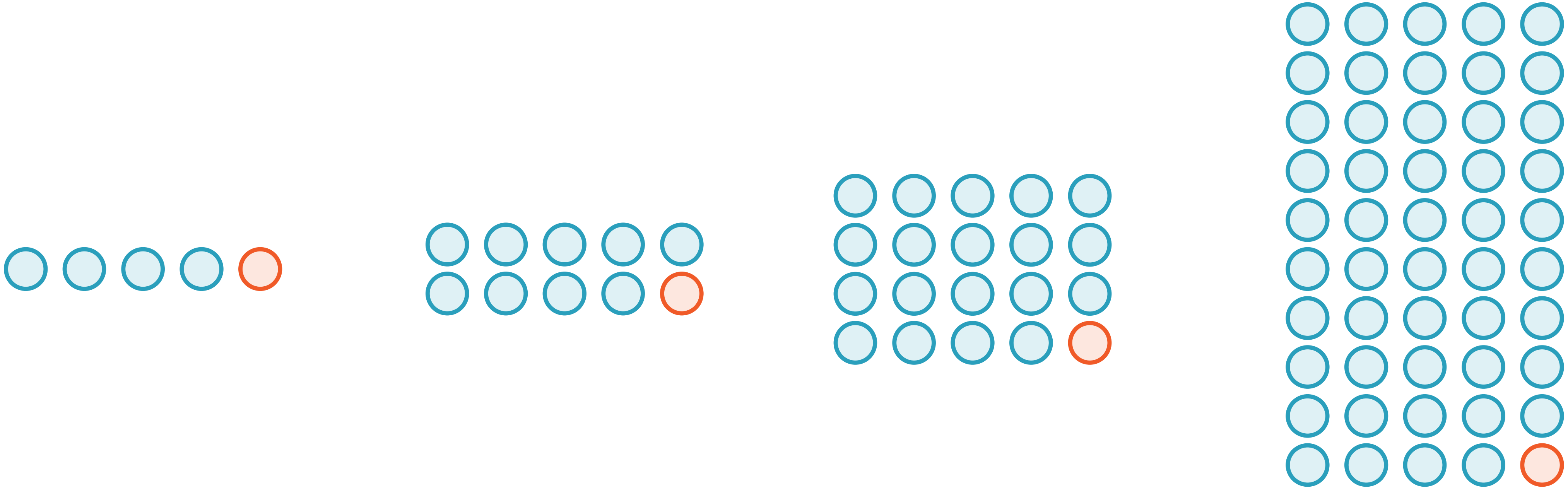
A deployment pattern, where a new release of software is made available to a small subset of users in order to minimize risk.

Canary Deployment



Both deployments share a service using a common label

Deployment Pattern Limitations



20% = 5 replicas

10% = 10 replicas

5% = 20 replicas

2% = 50 replicas

Reverse proxy load balancing techniques can weight traffic to upstream virtual hosts

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata
  name: traefik-ingress
  annotations:
    traefik.ingress.kubernetes.io/service-weights: |
      app: 90%
      app-canary: 10%
```

Weighting Traffic - Traefik Ingress Controller

The backend services must share the same host and path

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata
  name: nginx-ingress
  annotations:
    nginx.ingress.kubernetes.io/canary: "true“
    nginx.ingress.kubernetes.io/canary-weight: “10”
```

Weighting Traffic - Nginx Ingress Controller

An association is made between the two different ingress definitions

Demo

Deploy an application service fronted by an ingress

Introduce a canary release of the application

Configure the ingress to send 5% of traffic to the canary

Test the ingress honors the specified traffic weighting

Module Summary

Limitations of Ingress API for advanced use cases

Annotations provide an effective workaround

Traffic segregation using ingress class annotations

Canary deployment pattern with ingress controller specific annotations