Pivoting Your Pipeline for Microservices

What we will cover

Changes in your CI/CD process for a K8s Platform

- When you begin your Cloud Native journey, you’ll quickly find that your current CI/CD pipeline is not going to make the grade. Kubernetes is finally moving us away from a monolithic approach to software development towards a service-based approach. To support this approach, your CI/CD Pipeline will need to pivot. We will cover the problem areas.

- See a use case of how that might look. Meet Nathan Martin of sagecore technologies who will provide us a real-world case study on managing microservice with service mesh routing in a modern continuous delivery pipeline.

Serious Software Simple Solutions
sagecore technologies

DeployHub®

Takeaways

For most organizations, **microservices will have their own repository and workflows**. CD tools will need to support workflow templates.

Configuration management will be lost as large monolithic builds are replaced or non-existence. Link decision making is done at runtime – not by a build manager.

Service Mesh becomes core to routing microservices and changing how we see Dev, Test, and Prod Environments.

Your Presenters

Tracy Ragan, CEO & Co-founder, DeployHub

Microservice Evangelist, Founding Board Member Eclipse Foundation. Founding Board member of the CD Foundation, DevOps Institute Ambassador,
20+ DevOps Experience.

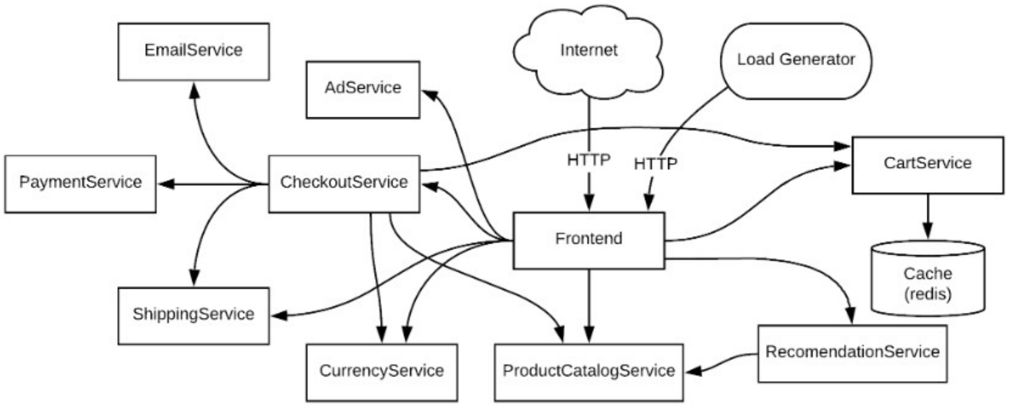
Nathan Martin, CEO & Founder, Sagecore Technologies

Speaker, thought leader, Kubernetes enthusiast. Specialist in enterprise resource planning and cloud native applications.

Think Functions

- The key to understanding microservices is to think ‘functions.’ With a microservice environment the concept of an ‘application’ goes away. It is replaced by a grouping of loosely coupled services connected via APIs at runtime, running inside of containers, nodes and pods.

- Microservices are immutable. You don’t ‘copy over’ the old one, you deploy a new version to the cluster and manage them with *Labels*.



The Result

We have lost the ‘application’

Like taking a wine glass and breaking it into pieces.

Now we have a pile of glass, where is the wine glass?

It is still there, just in pieces.

With Microservices your Landscape Changes

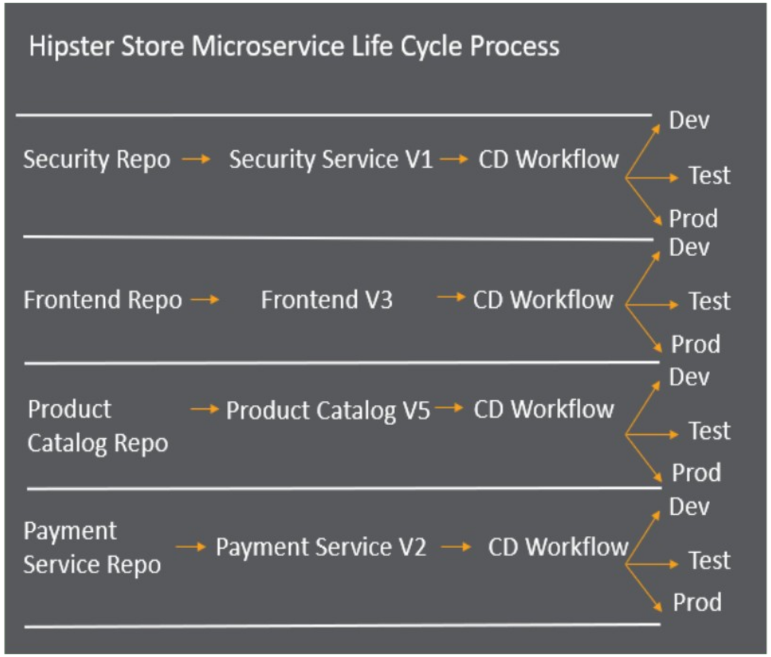
- Shifting to a modern architecture will disrupt our traditional CD pipeline.
- Why is the CD process disrupted?
- Microservices are deployed independently and that change impacts everything.
- Dev, Test and Prod is morphing.

Reality of Builds

- Builds are Different
- Smaller code means smaller builds, if at all. Python is interpreted.
- Linking is done at runtime, not at compile/link time.
- Builds will focus on creating a container.

Reality of Workflows

Multiple Workflows for a Single Application



Hipster Store Microservice Life Cycle Process

Security Repo → Security Service V1 → CD Workflow → Dev, Test, Prod

Frontend Repo → Frontend V3 → CD Workflow → Dev, Test, Prod

Product Catalog Repo → Product Catalog V5 → CD Workflow → Dev, Test, Prod

Payment Service Repo → Payment Service V2 → CD Workflow → Dev, Test, Prod

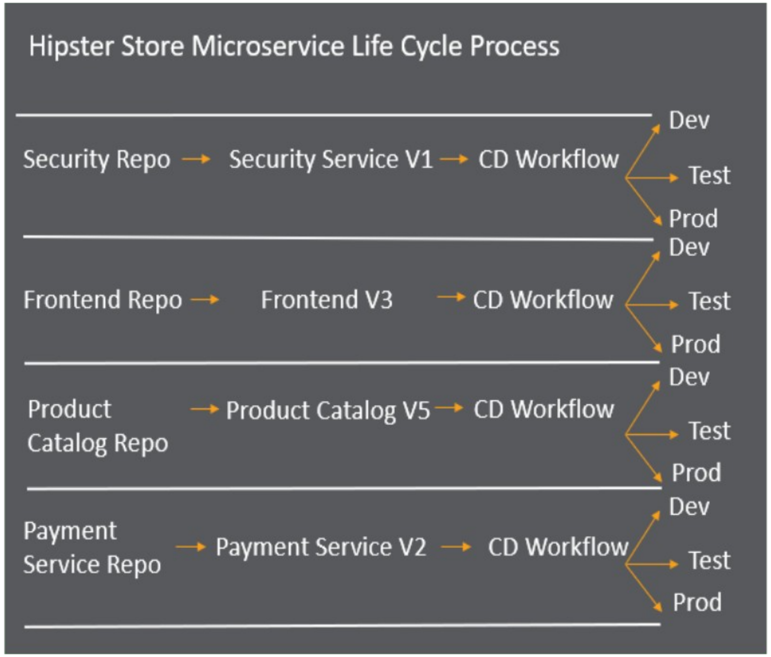
Multiple Workflows

To manage many moving parts, each microservice will have their own repository and CD Workflow. Orchestration of the CD process will become increasingly critical.

Think Templates!

Reality of Deployments

Loss of an Application Version and View



Hipster Store Microservice Life Cycle Process

Security Repo → Security Service V1 → CD Workflow → Dev / Test / Prod

Frontend Repo → Frontend V3 → CD Workflow → Dev / Test / Prod

Product Catalog Repo → Product Catalog V5 → CD Workflow → Dev / Test / Prod

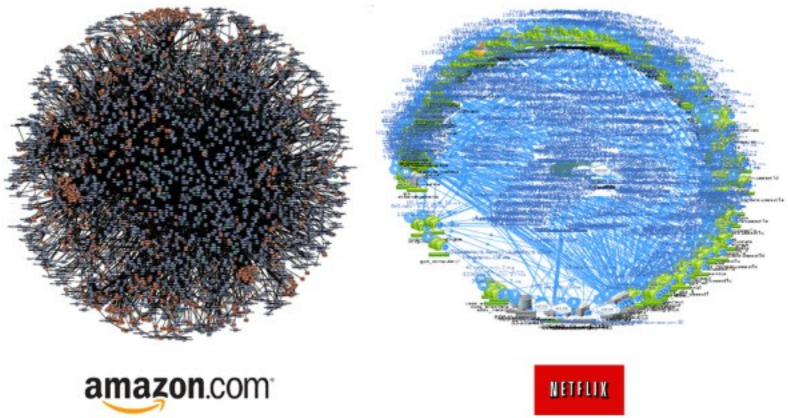
Payment Service Repo → Payment Service V2 → CD Workflow → Dev / Test / Prod

Independently Deployable

Because Microservices are independently deployable - they should have their own Repository and Workflow. Your application workflow will be replaced by many microservice workflows.

Reality of Configuration

Navigating the Deathstar



Haunted Graveyards, Frankenstein Clusters, when do we deprecate?

Configuration Management

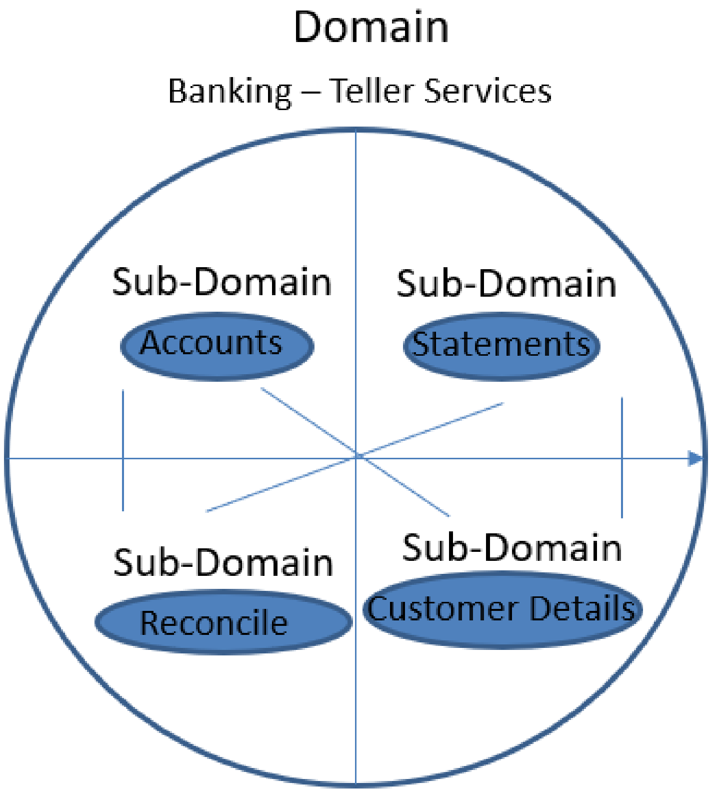
Your application goes away, but a logical view is of its configuration is critical. Mapping (with versions) your service to service and application to service dependencies replaces your traditional software bill of material report.

New - Domain Driven Design

Organizing Your Microservices

- Domain Driven Design is where you are managing an architecture based on the microservice ‘problem space.’

- To find and share microservices they must be organized in a way that meets the needs of your ENTIRE organization, and allows for them to be found and shared.



New - Applying the Container to the Cluster

Tells the Container how to interact with the Cluster

Ortelius
Microservice
Mapping

New – Dependency Mapping and Data Visualization

Tracking your Services

- Tracks the microservice version to the application version that consumes them.
- Supports a Domain Structure for finding and sharing microservices.
- Versions the deployment meta data including the Helm Charts.

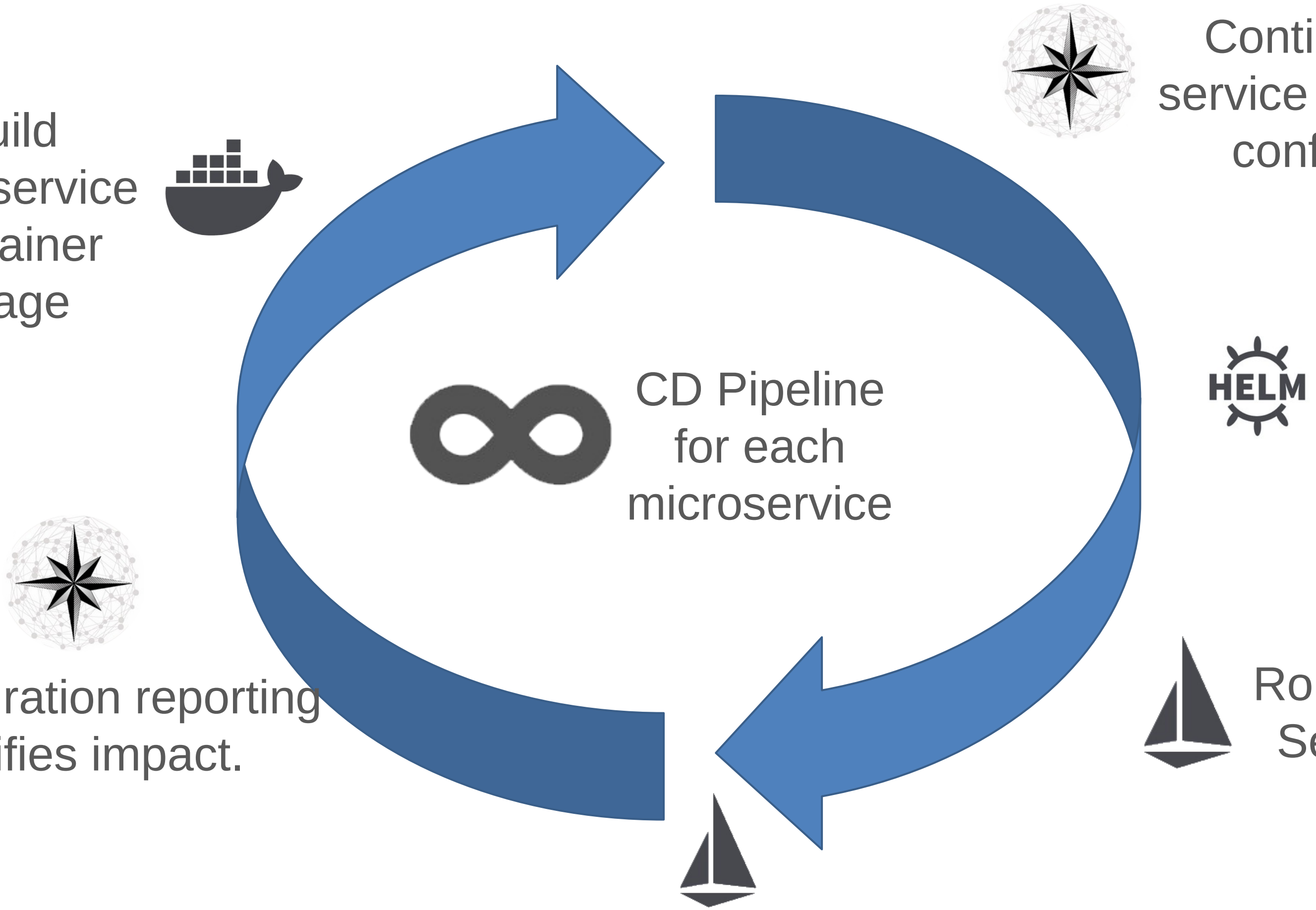
Istio

kiali

New – Service Mesh Routing

- Routes the flow of traffic
- Manage Policies
- Monitoring and feedback

New Microservice Pipeline



Build microservice container image

Continually track service to application configurations

CD Pipeline for each microservice

Push a new single microservice to the cluster (Dev,Test, Prod)

Route the Service

Collect Feedback

Configuration reporting identifies impact.

Results

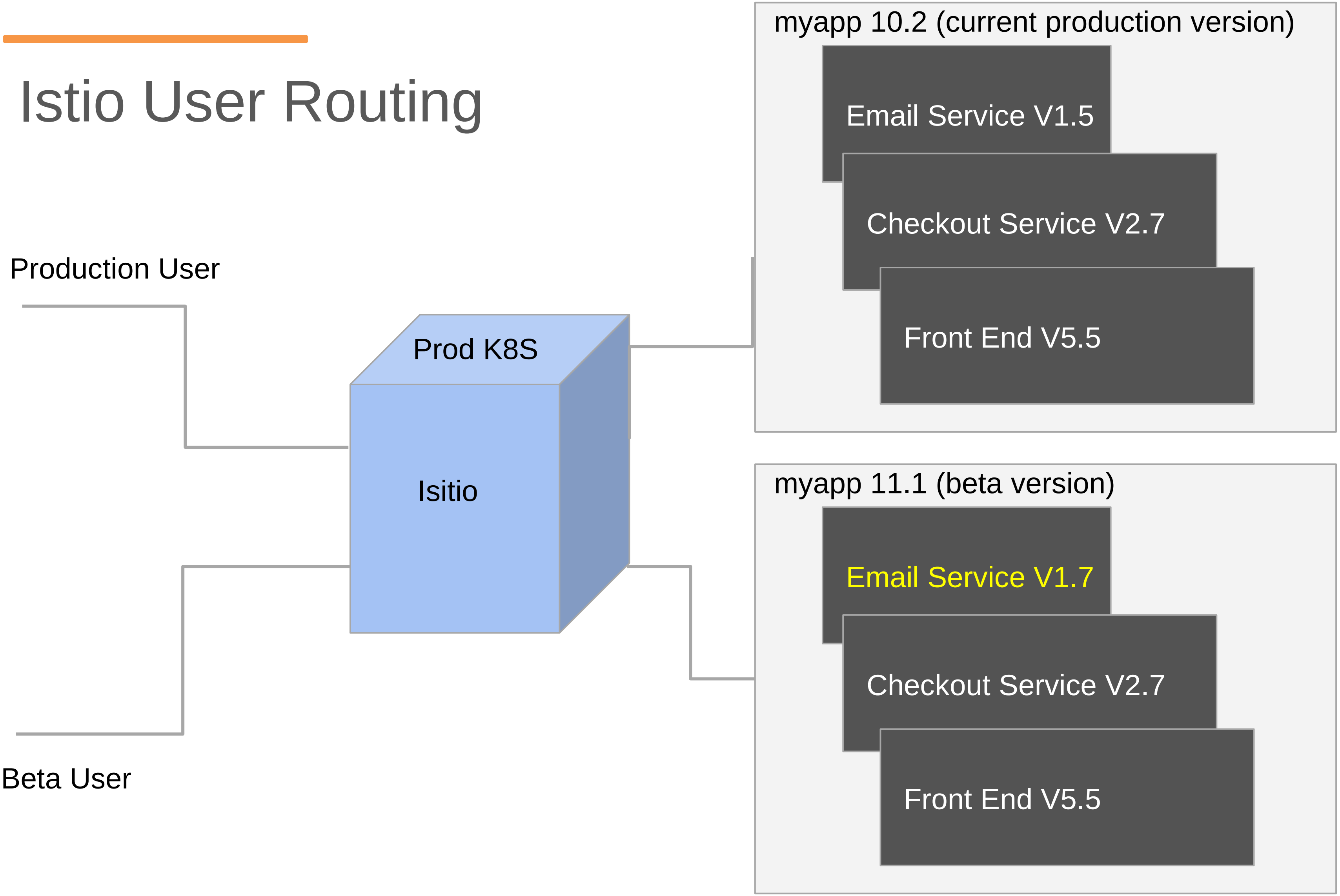
Container Image Build replaces monolithic compile/link scripts

The CD pipeline supports independently deployable microservices with 100s of workflows

Configuration management is critical to provide continuous mapping of service to service and application to service relationships (Your new BOM)

Routing and feedback delivers the microservice with the Helm meta-data.

Istio User Routing



Production User

Prod K8S

Isitio

Beta User

myapp 10.2 (current production version)

Email Service V1.5

Checkout Service V2.7

Front End V5.5

myapp 11.1 (beta version)

Email Service V1.7

Checkout Service V2.7

Front End V5.5

TETRICOR

Real World Use Cases

Tetricor.com

Enterprise Resource Planning Software

Overview

TETRICOR

Tetricor is a dynamic Enterprise Resource Planning software developed by Sagecore Technologies.

Developed on a Kubernetes Platform with Istio.

TETRICOR

Our Journey’s Challenges:

- Deployment Staging
- Application versioning
- Request routing
- Mesh (and cluster) visualization

Deployment Staging

TETRICOR

Solution: The Sagecore Technologies team recognized the need to create a dynamic lifecycle with testing as close to production as possible.

This required automation driven by a CD pipeline with rollout strategies for continuous configuration management, continuous test and continuous deployment.

Getting There

Setup Istio DestinationRules to easily rollout from beta ‘virtual service’ to other application versions.

- What makes this different than updating routing from beta to production using K8s Ingress controller?

Ingress rules have a lag time to live. This method allows for instant rollout of a specific build, and segregation of build schedules for different users (beta, alpha, pre-prod. etc..)

Request Routing

TETRICOR

Problem: Our SaaS model must cleverly allocate resources within our cluster to specific accounts, and routing of specific requests to their corresponding microservices for consumption.

Kubernetes ingress routing is somewhat limited, and seemingly designed for top level network design (as mentioned before there is a lag time in deployment of service routing.)

Request Routing

TETRICOR

Problem: We had necessity to route based upon

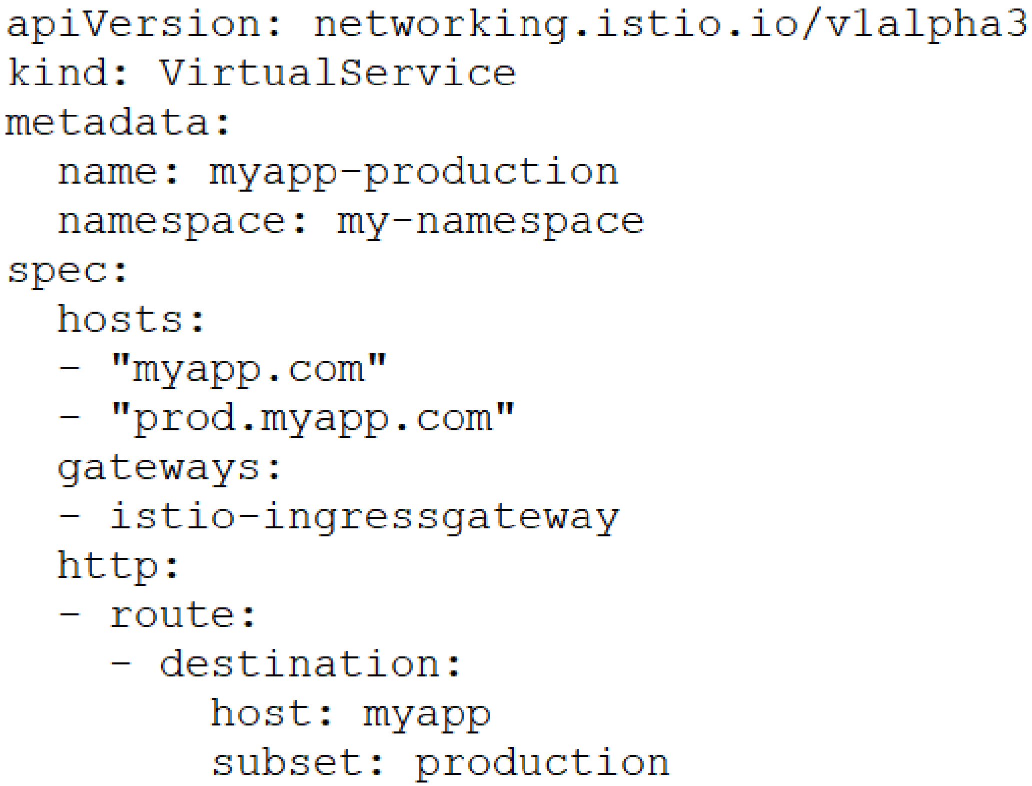
- Headers
  - Domain (Host)
  - Cookie
  - Custom Header
- Path

Without Istio we had no way to attach various SaaS tenants to different application versions, dynamically.

Request Routing

Host Based Routing Example:

This example routes all traffic from ‘myapp.com’ and ‘prod.myapp.com’ to the production subset.



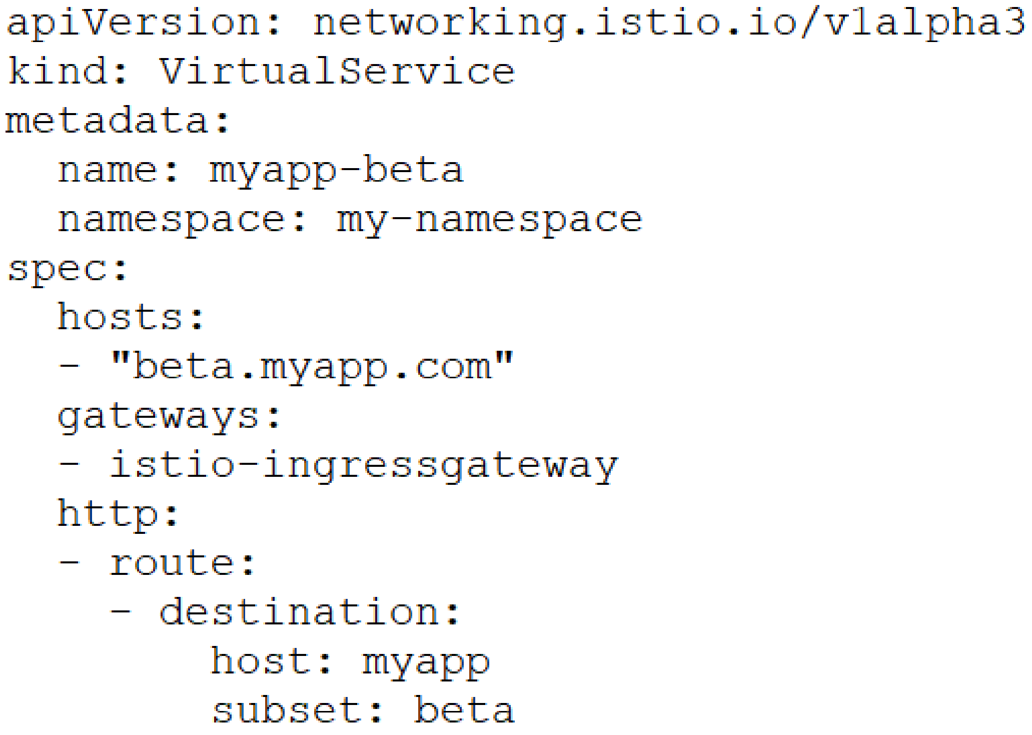
```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: myapp-production
  namespace: my-namespace
spec:
  hosts:
  - "myapp.com"
  - "prod.myapp.com"
  gateways:
  - istio-ingressgateway
  http:
  - route:
    - destination:
        host: myapp
        subset: production
```

Request Routing

TETRICOR

Host Based Routing Example:

This example routes all traffic from ‘beta.myapp.com’ to the beta subset.



```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: myapp-beta
  namespace: my-namespace
spec:
  hosts:
  - "beta.myapp.com"
  gateways:
  - istio-ingressgateway
  http:
  - route:
    - destination:
        host: myapp
        subset: beta
```

Deployment Staging

Example:

Label your deployments with a unique application version each time you deploy a new version.

Deployment 'alpha'

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-2-1
  namespace: my-namespace
  labels:
    app: myapp
    version: v2-1
spec:
  selector:
    matchLabels:
      app: myapp
      version: v2-1
  template:
    metadata:
      labels:
        app: myapp
        version: v2-1

        ...
```

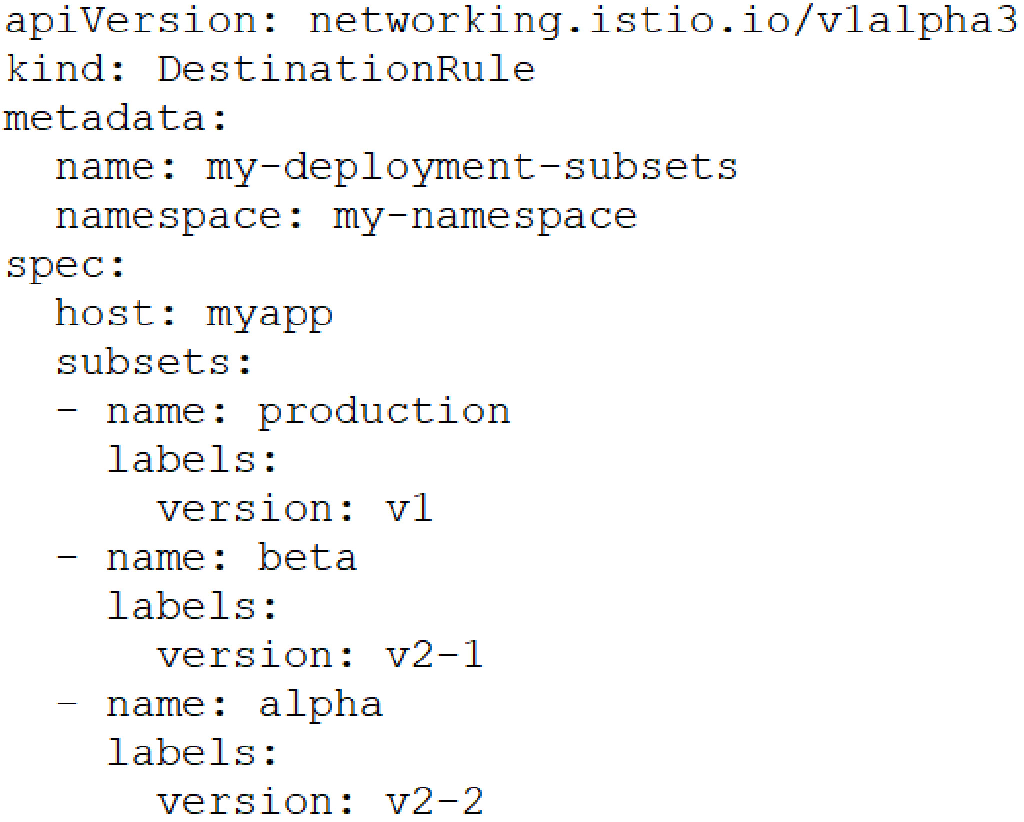
Deployment 'production'

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-1
  namespace: my-namespace
  labels:
    app: myapp
    version: v1
spec:
  selector:
    matchLabels:
      app: myapp
      version: v1
  template:
    metadata:
      labels:
        app: myapp
        version: v1

        ...
```

Deployment Staging

Define a DestinationRule to map your application versions into human-readable application subsets.

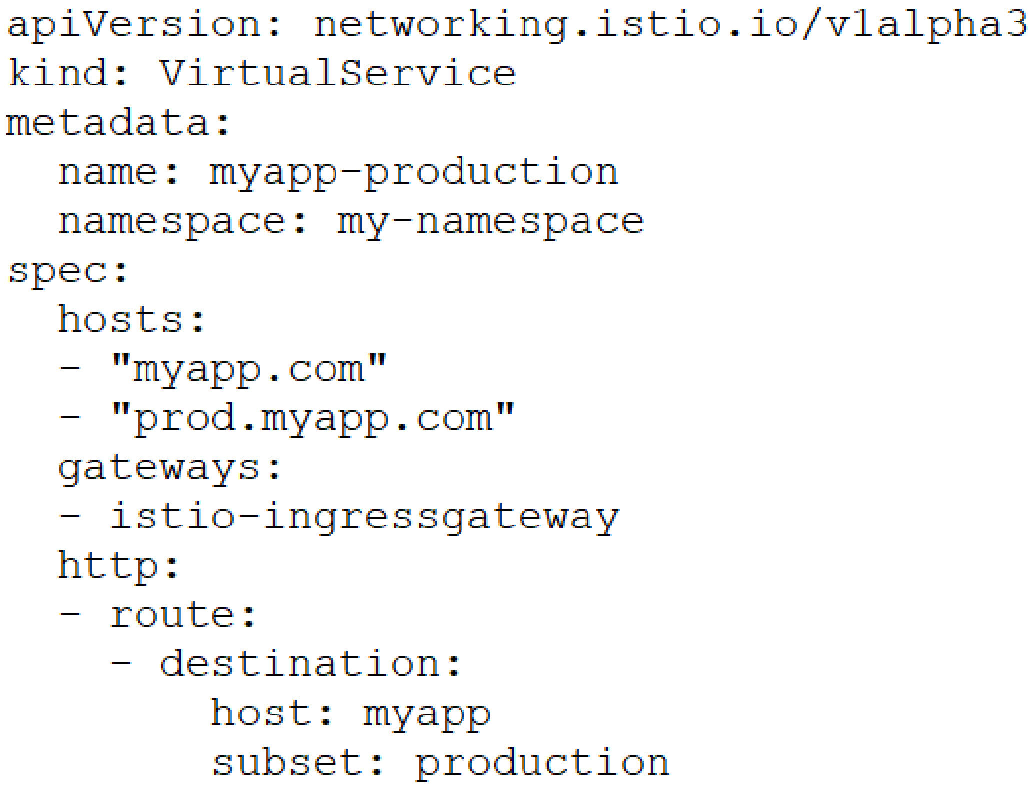


```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: my-deployment-subsets
  namespace: my-namespace
spec:
  host: myapp
  subsets:
  - name: production
    labels:
      version: v1
  - name: beta
    labels:
      version: v2-1
  - name: alpha
    labels:
      version: v2-2
```

Deployment Staging

Define your VirtualServices to route to appropriate subsets.

This example routes all traffic from ‘myapp.com’ and ‘prod.myapp.com’ to the production subset.

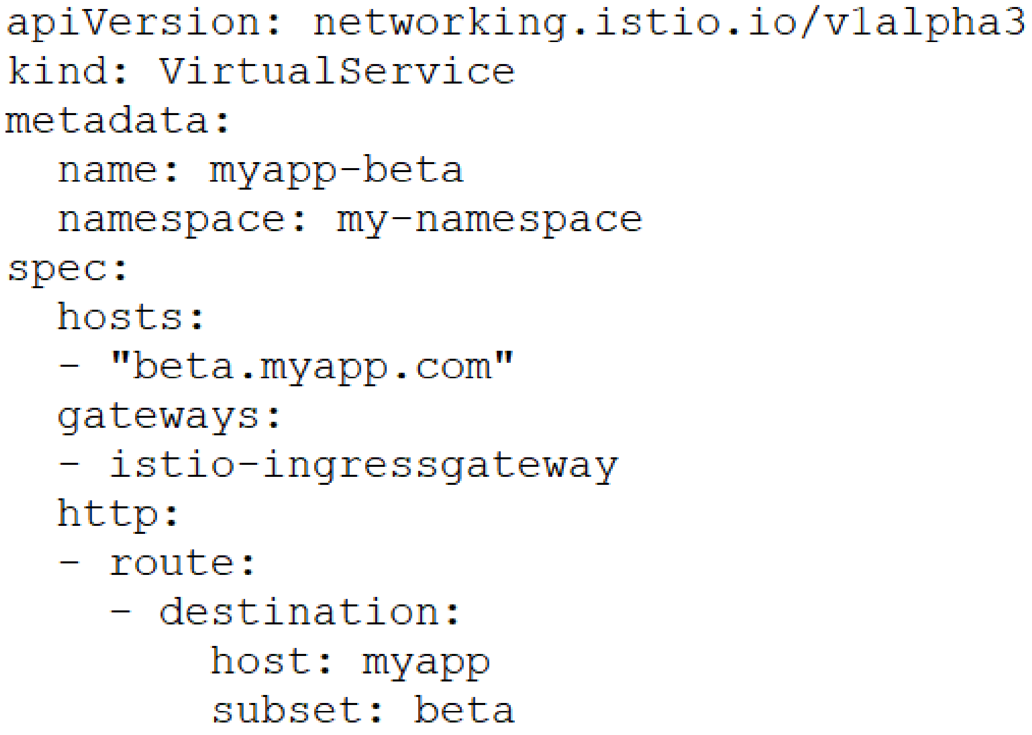


```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: myapp-production
  namespace: my-namespace
spec:
  hosts:
  - "myapp.com"
  - "prod.myapp.com"
  gateways:
  - istio-ingressgateway
  http:
  - route:
    - destination:
        host: myapp
        subset: production
```

Deployment Staging

Define your VirtualServices to route to appropriate subsets.

This example routes all traffic from ‘beta.myapp.com’ to the beta subset.



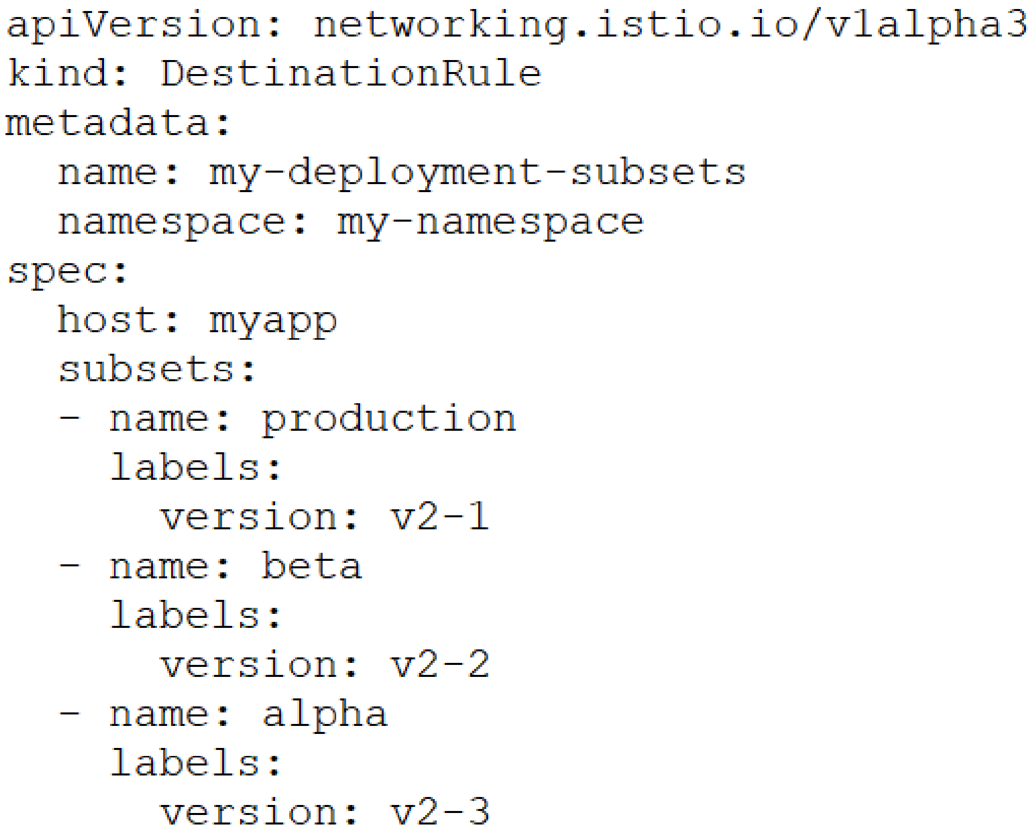
```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: myapp-beta
  namespace: my-namespace
spec:
  hosts:
  - "beta.myapp.com"
  gateways:
  - istio-ingressgateway
  http:
  - route:
    - destination:
        host: myapp
        subset: beta
```

Deployment Staging

TETRICOR

Elevating an application version is easy, simply update your DestinationRule.

In this example, beta (v2-1) was elevated to production.



```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: my-deployment-subsets
  namespace: my-namespace
spec:
  host: myapp
  subsets:
  - name: production
    labels:
      version: v2-1
  - name: beta
    labels:
      version: v2-2
  - name: alpha
    labels:
      version: v2-3
```

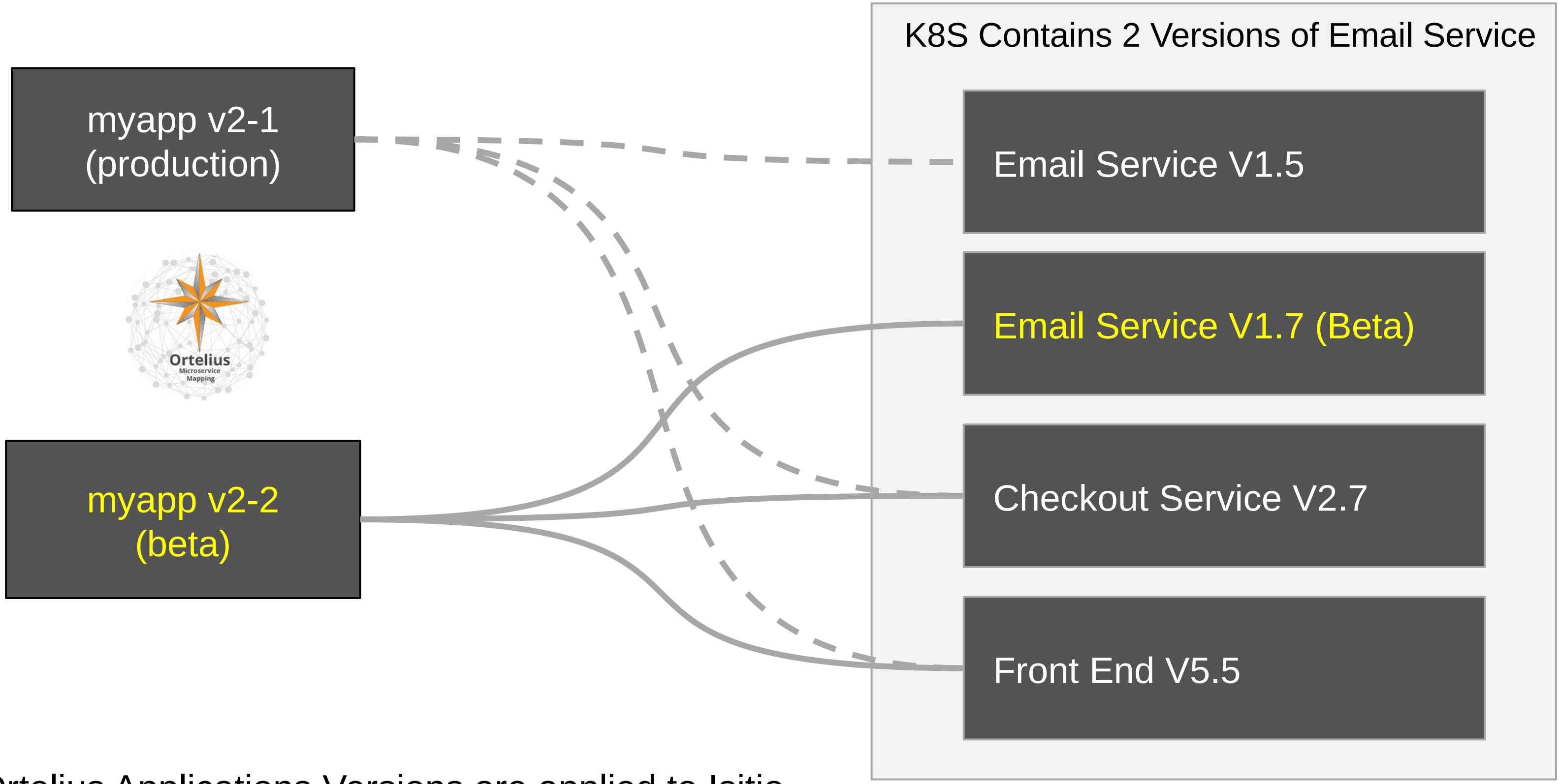
Mesh Visualization

TETRICOR

Problem: Migrating from a monolithic mentality can be difficult.

Visualizing your cluster and understanding how your application is behaving is crucial to deploying and managing an istio/kubernetes powered application and it’s complexity.

Ortelius View of Application Versions


K8S Contains 2 Versions of Email Service
myapp v2-1 (production)
Ortelius Microservice Mapping
myapp v2-2 (beta)
Email Service V1.5
Email Service V1.7 (Beta)
Checkout Service V2.7
Front End V5.5

- Ortelius Applications Versions are applied to Isitio
- Checkout and Front End shared by both versions

Mesh Visualization

TETRICOR

Meet: kiali

https://kiali.io/

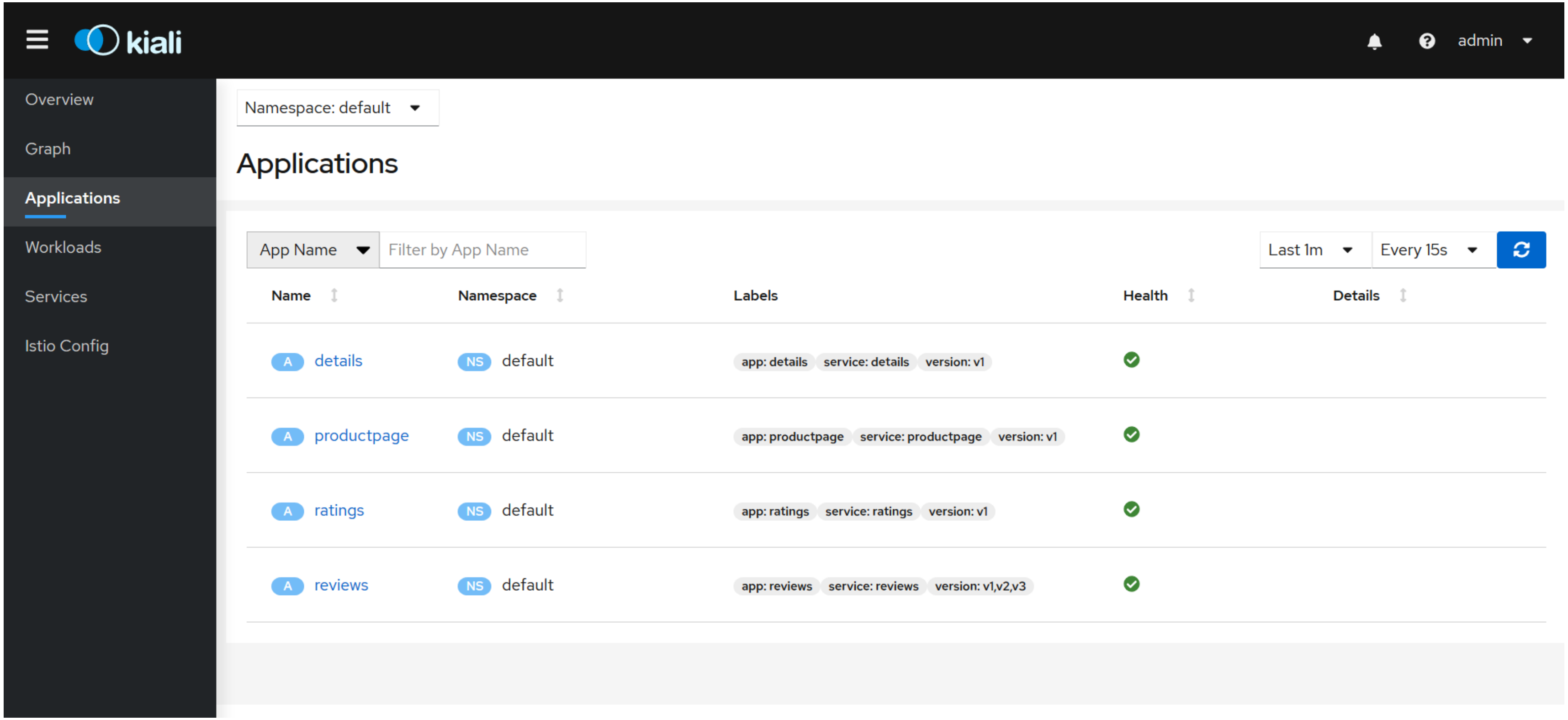
View your service mesh from a bird’s-eye view.

Enabling solutions to problems as they arise.

- Which microservices are part of my service mesh?
- How are they connected?
- How are they performing?
- How can I operate on them?

Mesh Visualization

TETRICOR



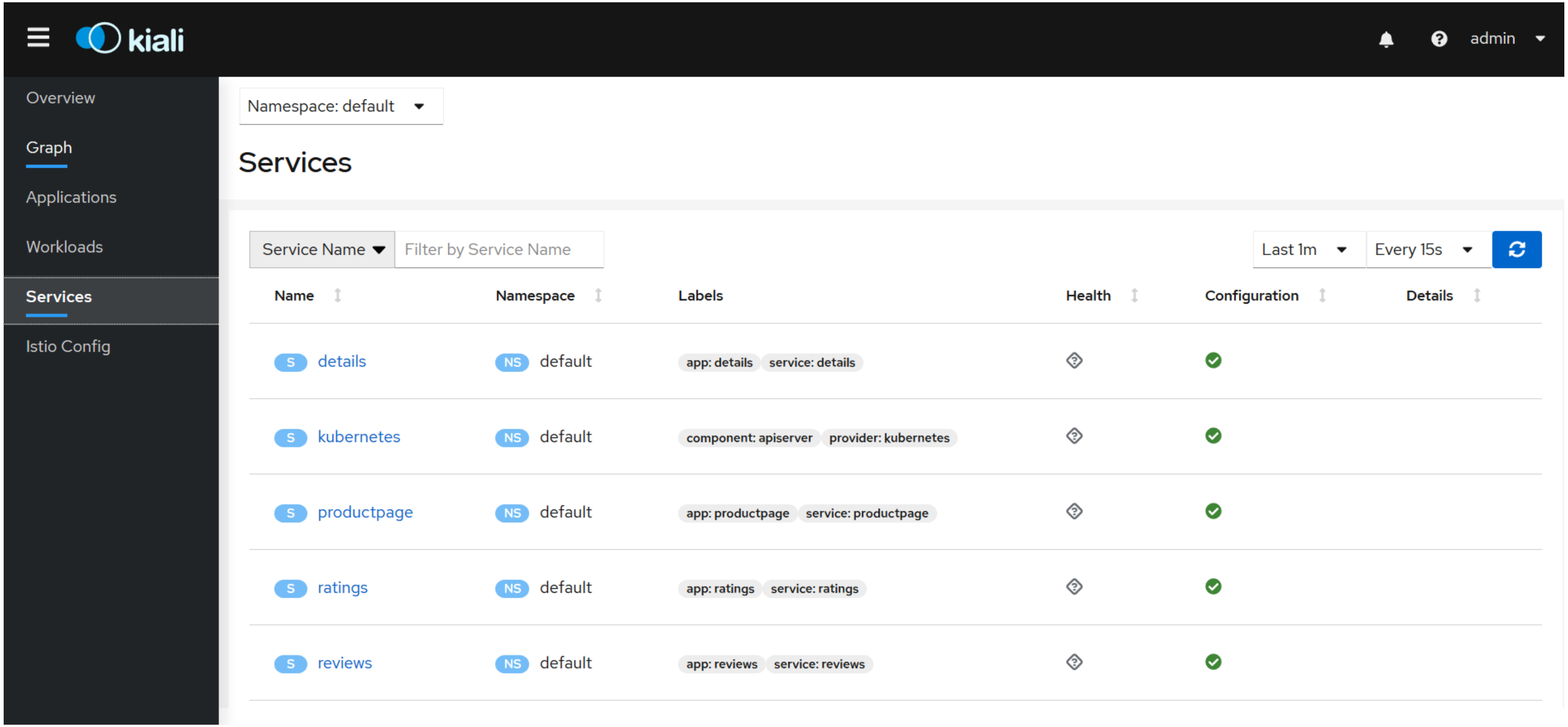
Applications

Name	Namespace	Labels	Health	Details
A details	NS default	app: details service: details version: v1	✓	
A productpage	NS default	app: productpage service: productpage version: v1	✓	
A ratings	NS default	app: ratings service: ratings version: v1	✓	
A reviews	NS default	app: reviews service: reviews version: v1,v2,v3	✓	

Which microservices are part of my service mesh?

Mesh Visualization

TETRICOR



Namespace: default

Services

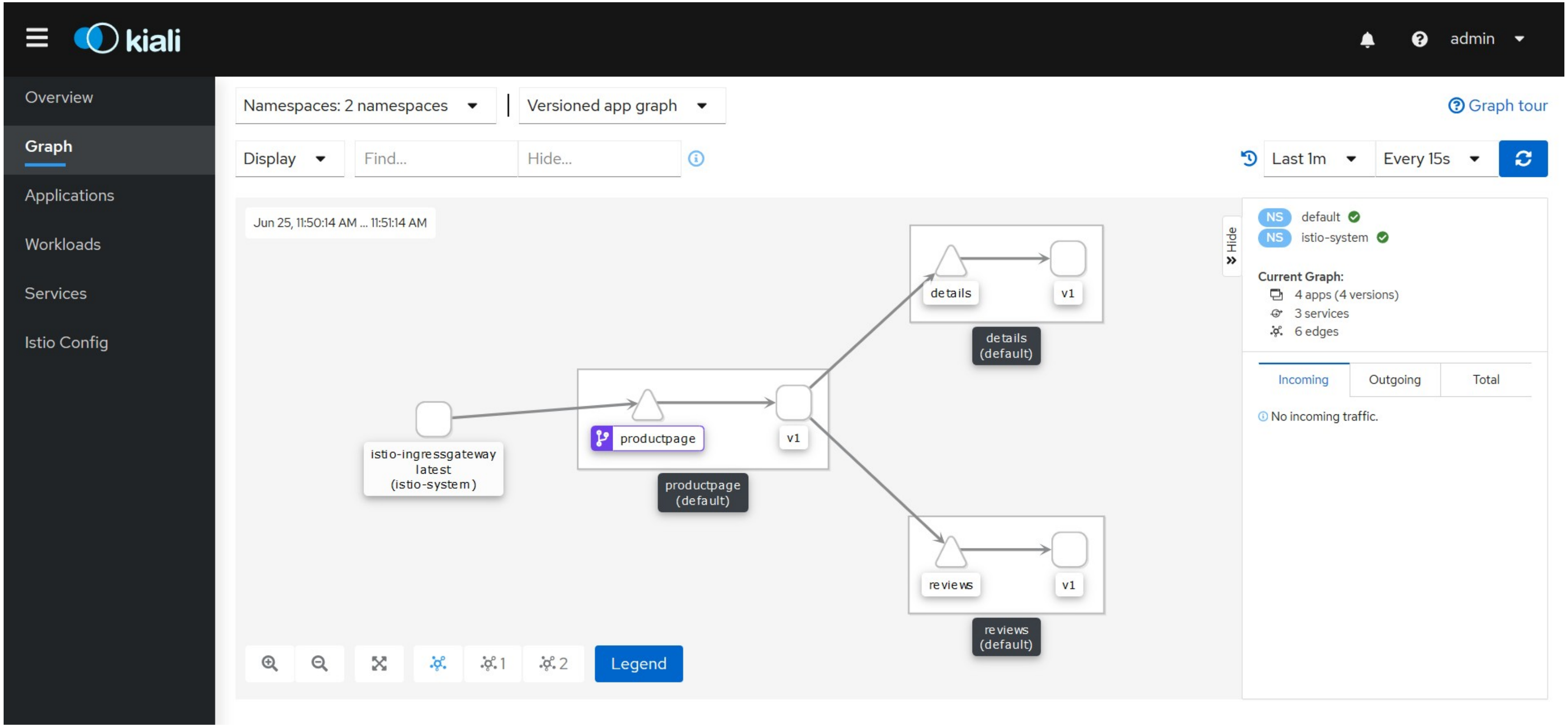
Service Name ▾ Filter by Service Name

Last 1m ▾ Every 15s ▾

Name	Namespace	Labels	Health	Configuration	Details
S details	NS default	app: details service: details	◈	✓	
S kubernetes	NS default	component: apiserver provider: kubernetes	◈	✓	
S productpage	NS default	app: productpage service: productpage	◈	✓	
S ratings	NS default	app: ratings service: ratings	◈	✓	
S reviews	NS default	app: reviews service: reviews	◈	✓	

How are my microservices performing?

Mesh Visualization



Namespaces: 2 namespaces | Versioned app graph

Last 1m | Every 15s

Jun 25, 11:50:14 AM ... 11:51:14 AM

NS default ✓
NS istio-system ✓

Current Graph:
- 4 apps (4 versions)
- 3 services
- 6 edges

No incoming traffic.

How are my microservices connected?

Solution – Ortelius Open Source

Microservice Management for Site Reliability, DevOps Engineers, and Release Teams

- A SaaS based central “Hub” for sharing, managing and releasing microservices
- Tracks and shows the “logical” application map
- Simplifies the complexity of microservices so organizations can achieve business agility
- Supports a Domain Catalog
- Integrates into Continuous Delivery for continuous configuration management.

Ortelius is the Open Source Core of DeployHub.

Get Involved in Ortelius

https://github.com/ortelius/ortelius

Learn more at Ortelius.io

If you think what we are doing is cool:

- Give us a Star on GitHub
- Become an Ambassador
- Become a Committer

We need your help and insight. Everyone sees a different part of the landscape we are mapping.

Abraham Ortelius

Talk to us…

LinkedIn: https://www.linkedin.com/in/tracy-ragan-oms/

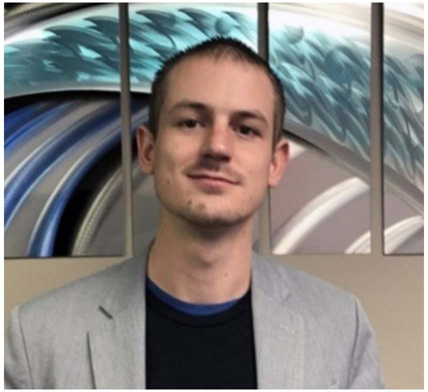
Twitter: https://twitter.com/TracyRagan

Calendar: https://drift.me/tracyragan/meeting/coffeechat

Email: TracyRagan@DeployHub.com

Dig In at: DeployHub.com or Ortelius.io

DeployHub®



LinkedIn:
https://www.linkedin.com/in/nathan-martin-81333211

Email: nathan@sagecoretech.com

Dig In at: Tetricor.com OR SagecoreTech.com

TETRICOR

Serious Software Simple Solutions
sagecore technologies

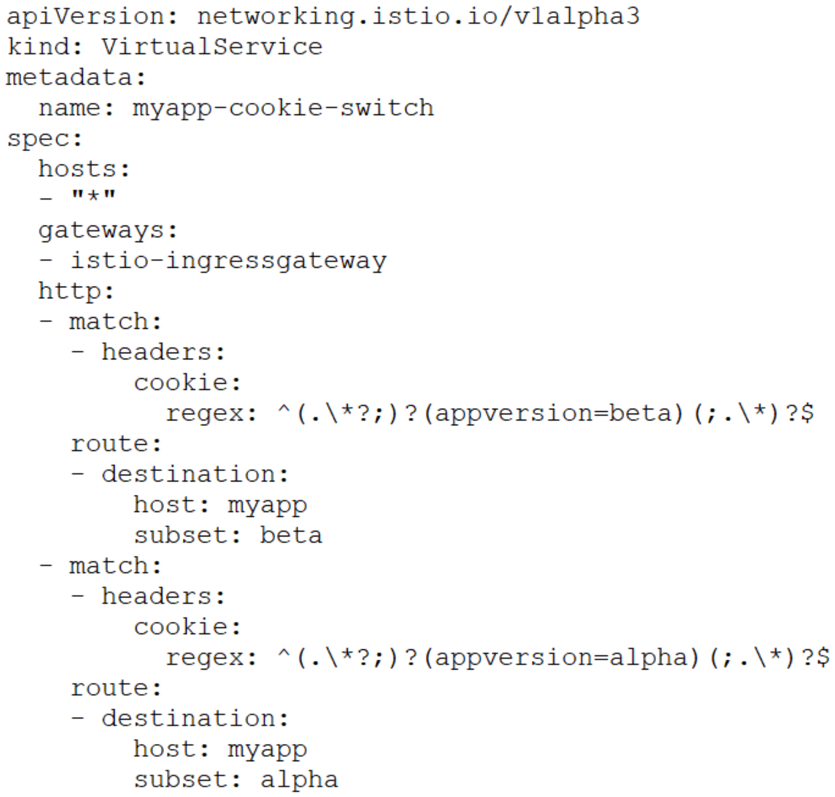
Thank You

Additional Routing

Request Routing

Cookie Routing Example:

This example will route to beta, or alpha application subset depending on the value of the ‘appversion’ cookie (if included in the request)



```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: myapp-cookie-switch
spec:
  hosts:
  - "*"
  gateways:
  - istio-ingressgateway
  http:
  - match:
    - headers:
        cookie:
          regex: ^(.\*?;)?(appversion=beta)(;.\*)?$
    route:
    - destination:
        host: myapp
        subset: beta
  - match:
    - headers:
        cookie:
          regex: ^(.\*?;)?(appversion=alpha)(;.\*)?$
    route:
    - destination:
        host: myapp
        subset: alpha
```

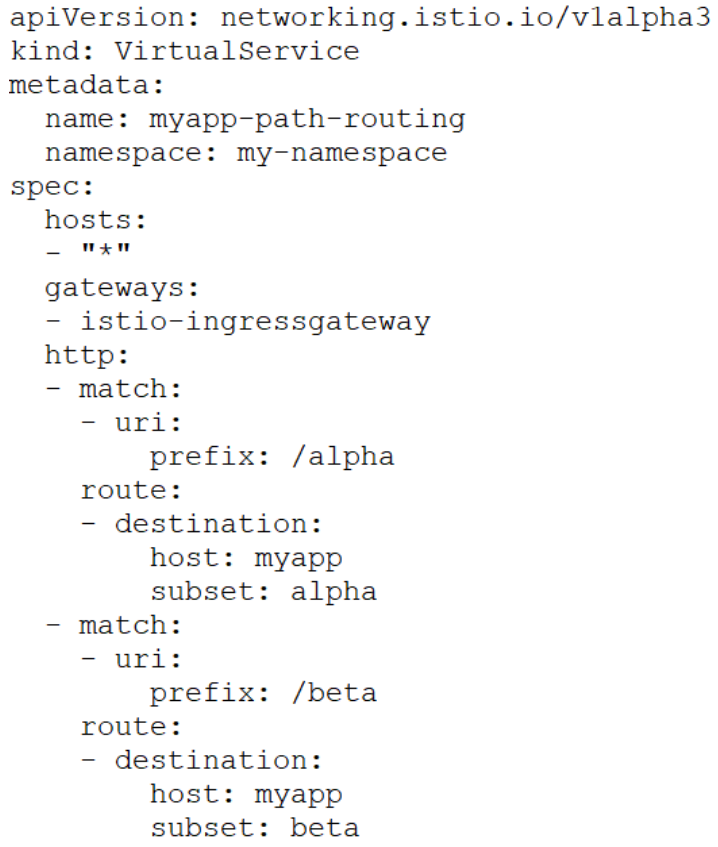
Request Routing

TETRICOR

Path Based Routing Example:

This example will route to beta, or alpha application subset depending on the value of the uri prefix.

In this example:
myapp.com/beta would route to beta subset.

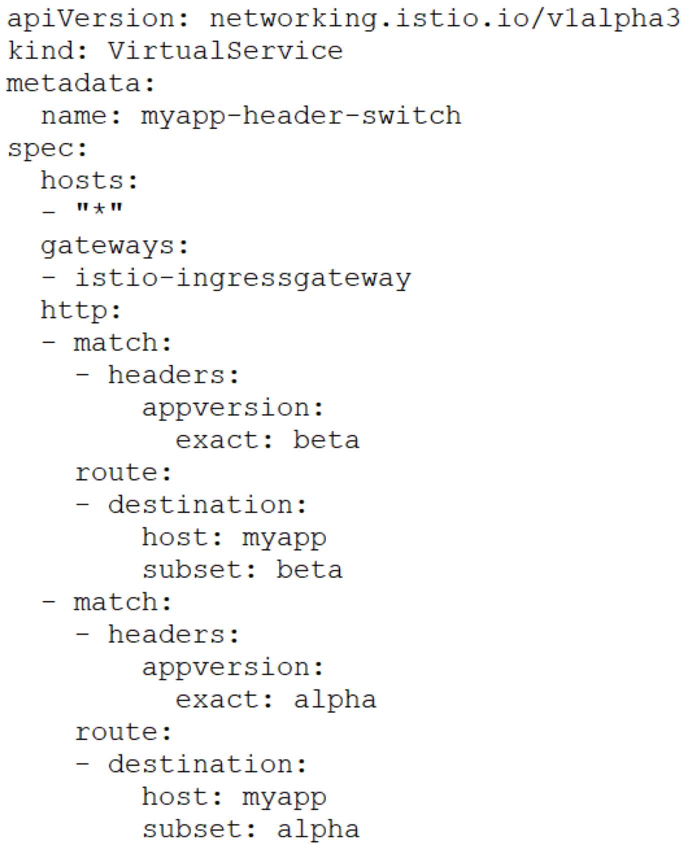


```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: myapp-path-routing
  namespace: my-namespace
spec:
  hosts:
  - "*"
  gateways:
  - istio-ingressgateway
  http:
  - match:
    - uri:
        prefix: /alpha
    route:
    - destination:
        host: myapp
        subset: alpha
  - match:
    - uri:
        prefix: /beta
    route:
    - destination:
        host: myapp
        subset: beta
```

Request Routing

Custom Header Routing Example:

This example will route to beta, or alpha application subset depending on the value of the ‘appversion’ header (if included in the request)



```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: myapp-header-switch
spec:
  hosts:
  - "*"
  gateways:
  - istio-ingressgateway
  http:
  - match:
    - headers:
        appversion:
          exact: beta
    route:
    - destination:
        host: myapp
        subset: beta
  - match:
    - headers:
        appversion:
          exact: alpha
    route:
    - destination:
        host: myapp
        subset: alpha
```