RANCHER

Understanding and Implementing Service Mesh

6 April 2020

Eamon Baumon
Field Engineer
Rancher

Matthew Scheer
Marketing Manager
Rancher
matthew@rancher.com
User Slack: @matthew

Rancher Master Class Series:

- 60 – 75 Minutes
- Questions are always welcome
- Use the questions tab to write your questions
- We may respond to all, so mark your question as private if needed.

#Rancherk8s

This session is being recorded!

Introducing Rancher's New Multi-Tenant Prometheus Support

RANCHER #RancherMeetup • December 19, 2018

0:05 / 1:39:01

Online Meetups and Webinars

Rancher Labs - 1 / 37

- December 2018 Online Meetup Introducing Rancher's New Multi — Rancher Labs
- 2 November 2018 Online Meetup: Building an Application Catalog with — Rancher Labs
- 3 October 2018 Online Meetup: Migrating from Rancher 1.6 to Rancher — Rancher Labs
- 4 September 2018 Online Meetup: Understanding Storage Options for — Rancher Labs
- 5 August 2018 Online Meetup: Building a CI/CD Pipeline with Kubernetes and — Rancher Labs
- 6 June 2018 Online Meetup: Kubernetes Networking Master Class — Rancher Labs

http://youtube.com/c/rancher

RANCHER

#Rancherk8s

RANCHER

Understanding & Implementing Service Mesh

Eamon Bauman
Field Engineer

04/06/2020

Understanding Service Mesh

Basic Service Mesh Concepts

Istio Concepts

Demo: Setup & Configure Istio Using Rancher

Wrap-up

Monolithic Applications

`washington_monument.exe`

- “In software engineering, a **monolithic application** describes a single-tiered software application in which the user interface and data access code are combined into a single program from a single platform.” – Wikipedia

- It’s an all-in-one piece of software

- One code base for everything
  - User interface
  - Data access layer
  - Data store

- See https://blog.heptio.com/what-is-a-monolithic-application-e375f5ad5ecb for a great diagram of this (thanks Kris Nova)

So what’s the problem?

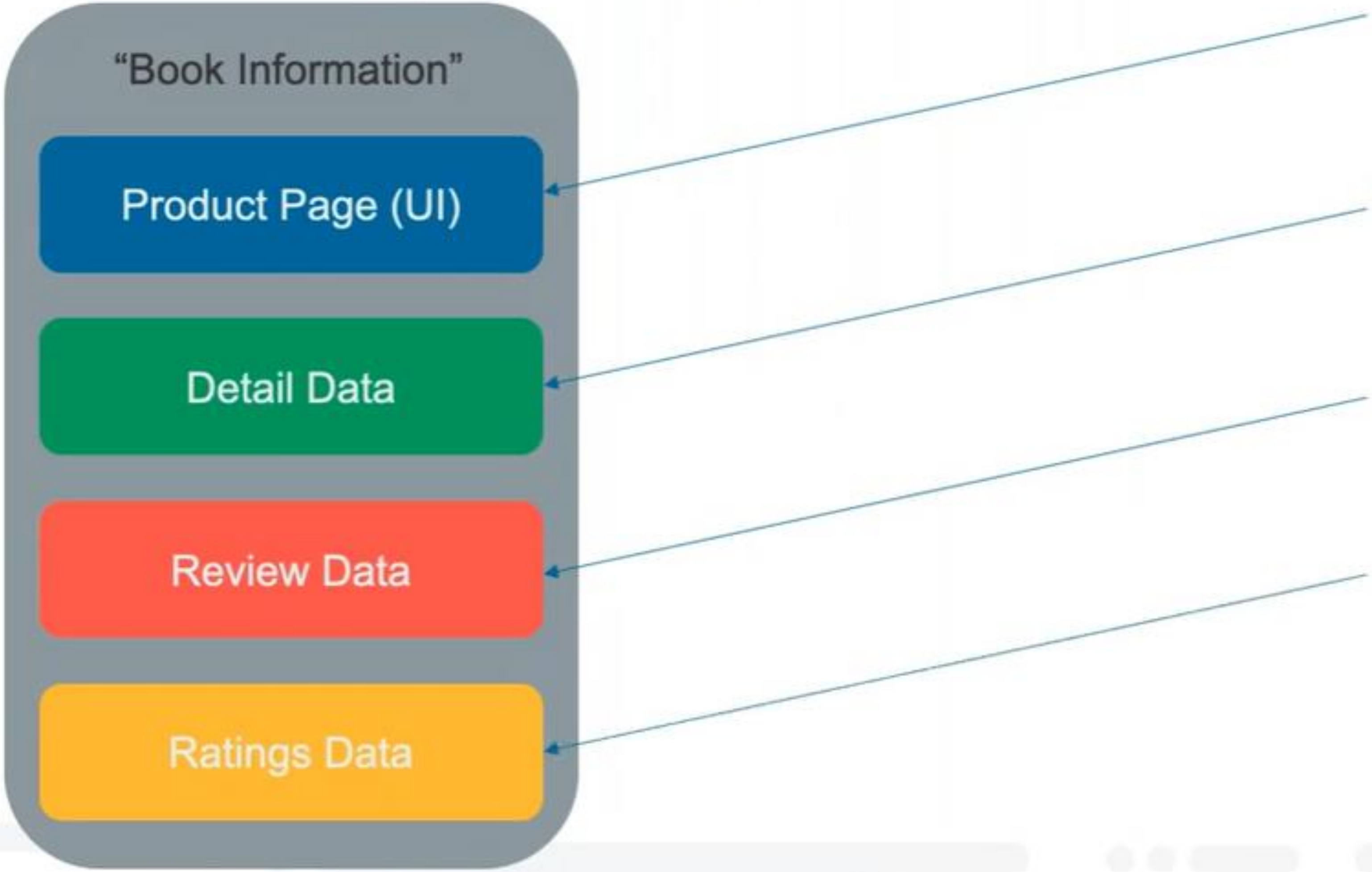
Well, that depends on who you ask

- Issues with monolithic applications become apparent as the application *scales*

- Vertical vs horizontal scaling
- Deployment velocity
- Choice of tooling
- Developer understanding

Sample Monolithic Application

…which will be important later on.



Displays information about books (reviews, ratings, etc.)

Data access & logic layer for getting details about books (authors, publishers, etc.)

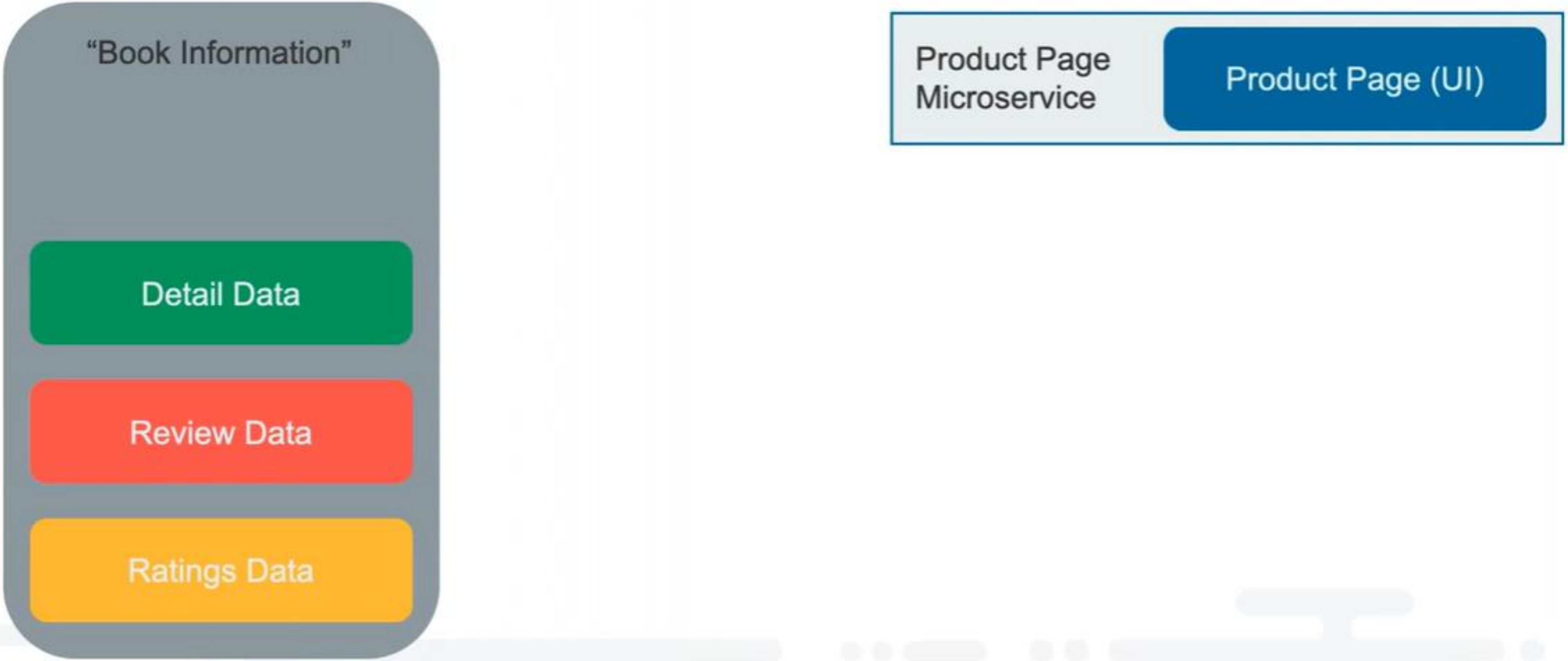
Data access & logic layer for getting reviews about books

Data access & logic layer for getting ratings about books (stars)

Microservices

- Goal: Build an application as a suite of *services*
- Services are independently deployable and scalable
- Each service, since separate, forms a boundary with other services
- Typically (though not always) services are accompanied by independent code bases
- The process of converting a monolith to a set of microservices is called “breaking up” the monolith

Splitting Up the Monolith



"Book Information"

Detail Data

Review Data

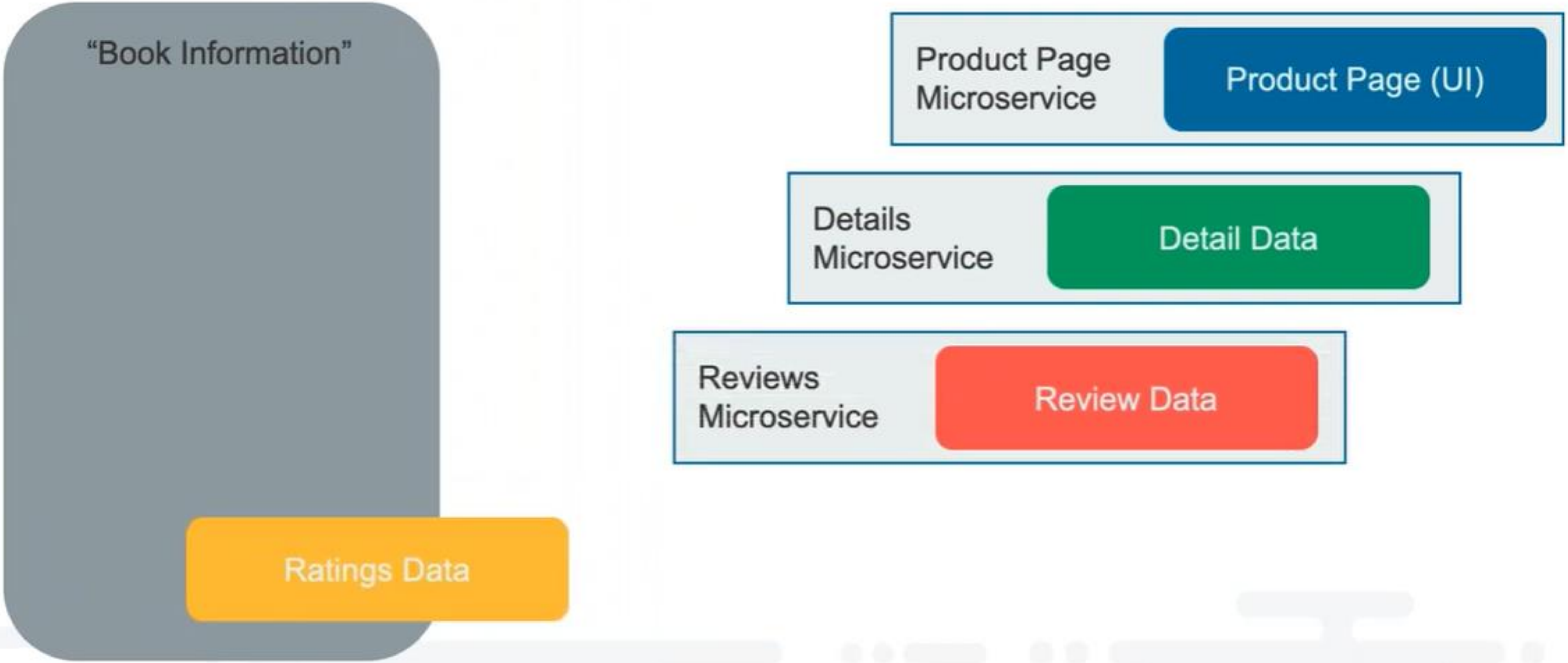
Ratings Data

Product Page Microservice

Product Page (UI)

RANCHER

Splitting Up the Monolith



"Book Information"

Ratings Data

Product Page Microservice

Product Page (UI)

Details Microservice

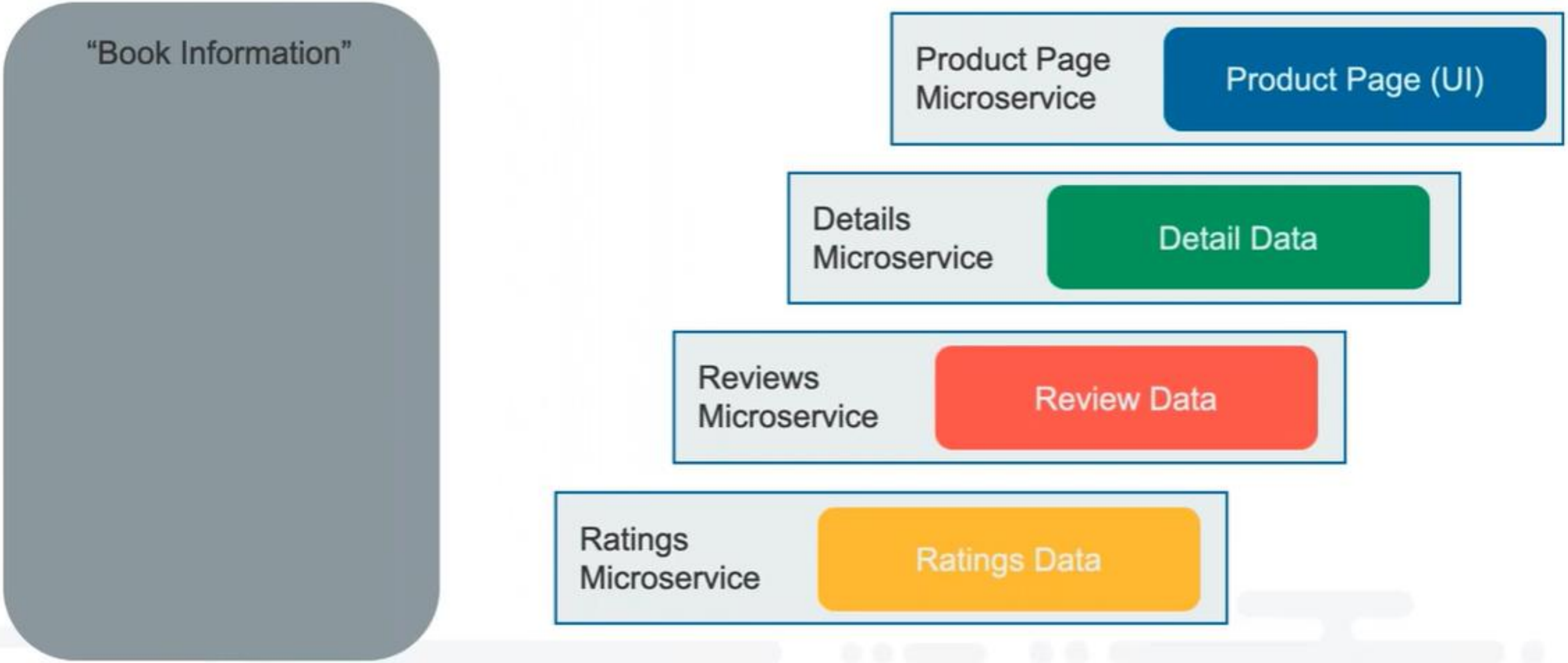
Detail Data

Reviews Microservice

Review Data

RANCHER

Splitting Up the Monolith



"Book Information"

Product Page Microservice

Product Page (UI)

Details Microservice

Detail Data

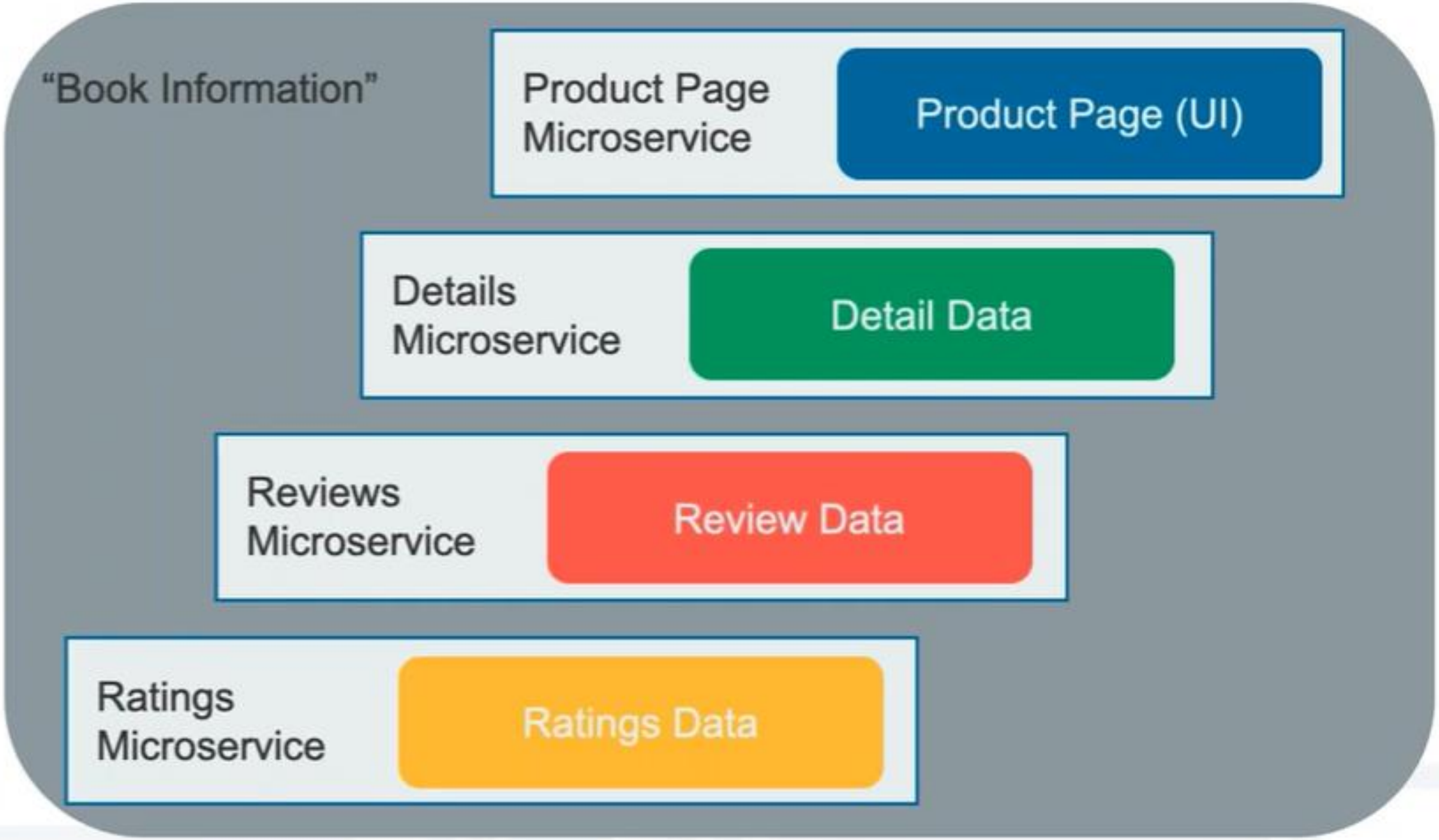
Reviews Microservice

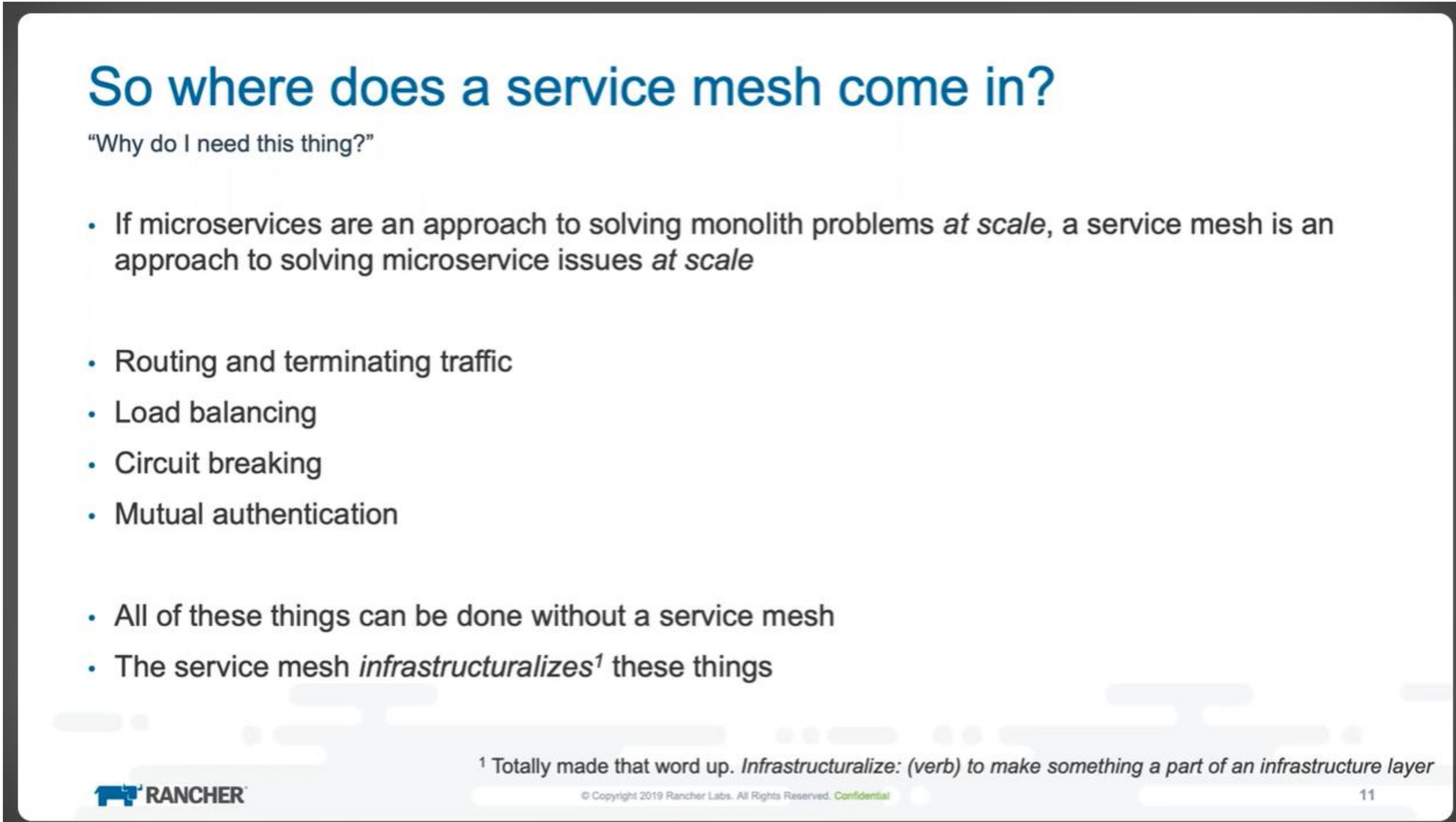
Review Data

Ratings Microservice

Ratings Data

Splitting Up the Monolith





So where does a service mesh come in?

"Why do I need this thing?"

- If microservices are an approach to solving monolith problems *at scale*, a service mesh is an approach to solving microservice issues *at scale*

- Routing and terminating traffic
- Load balancing
- Circuit breaking
- Mutual authentication

- All of these things can be done without a service mesh
- The service mesh *infrastructuralizes*[1] these things

[1] Totally made that word up. *Infrastructuralize: (verb) to make something a part of an infrastructure layer*

RANCHER

Caveat!

That “who you ask” part

- There are exceptions to all rules
- Do not believe that just because your application is getting bigger that it needs to be broken up
- Monoliths have existed for years (decades, really) without issue
- Microservices are great *if you meet the use case both technically and for your organization*

Service Meshes

- There are a lot out there

Service Mesh: Common Concepts

Most implement these concepts.

- **Traffic Management**
  - Routing
  - Load Balancing
  - Ingress
  - Sometimes: Egress
- **Security**
  - AuthN & AuthZ
  - Mutual Security (mTLS)
- **Observability**
  - Monitoring
  - Logging
  - Instrumentation

Traffic Management

- **Routing**
  - Getting traffic from Service A to Service B (intra-mesh)
  - Making determinations on where to send traffic
- **Load Balancing**
  - Round-robin, weighted, least-traffic
- **Ingress**
  - Bringing non-mesh traffic into the mesh
- **Sometimes: Egress**
  - Sending traffic out from the mesh through predetermined nodes

Security

- **AuthN**
  - “Who?”
  - Usually service-based identity, e.g. “I am ServiceA”
- **AuthZ**
  - “Why?”
  - Again, usually service-based, e.g. “Okay, ServiceA, you are allowed to do [x,y,z]”
- **mTLS**
  - Services mutually verifying each other via common CA

Observability

With apologies to the whole observability industry

- **Monitoring**
  - Monitor metrics about traffic flows
- **Logging**
  - Capture traffic logs as a sampling of the traffic flow instead of app-based logging
- **Instrumentation**
  - Performing distributed tracing that can be linked together by the service mesh

Istio Concepts

Istio: Core Components

- Data Plane
  - Envoy – Proxy server that is controlled by Istio. Typically runs as a “sidecar” to service containers in Kubernetes
- Control Plane
  - Pilot – Coordinates service discovery, traffic management, and resiliency features
  - Citadel – Security component used for mTLS, encryption, credential, and policy management
  - Galley – Configuration translation, processing, and validation layer
- Kiali
  - Not a true Istio component – but a very valuable 3rd party integration
  - Provides UI visibility for management of Istio

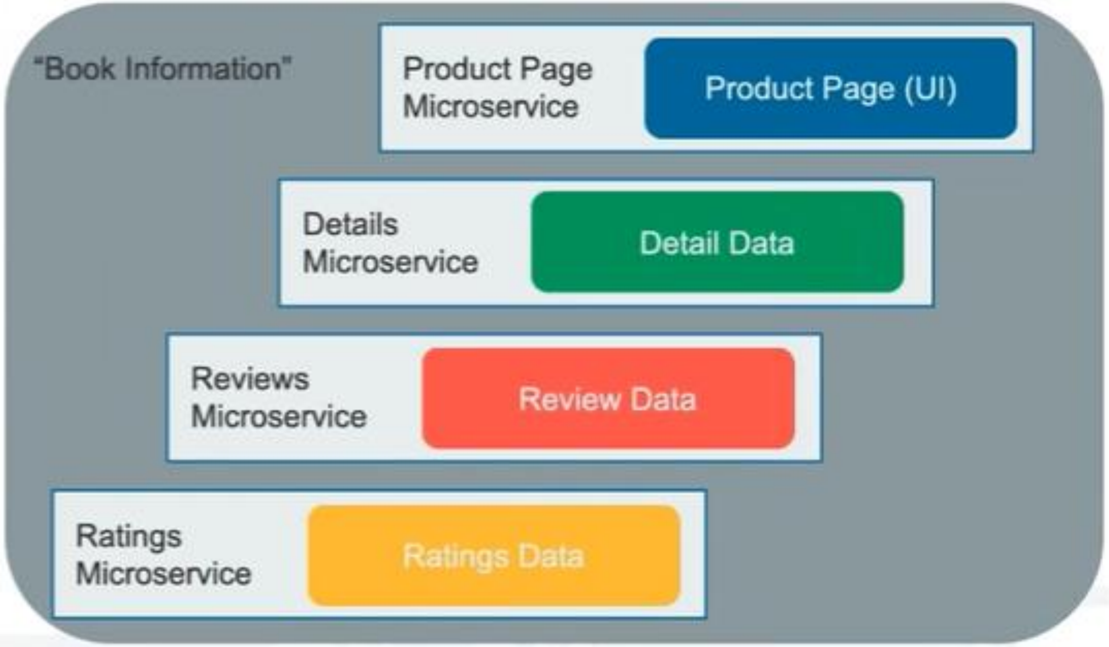
RANCHER

Istio: Building Blocks

- **Virtual Services**
  - Defines how requests are routed to a service (what goes where)
  - Focus on Virtual – an Istio V.S. is not a *real* service (e.g. something that answers requests)
  - A Virtual Service is an abstraction that helps define routing policies
- **Destination Rules**
  - Defines what happens to requests when arriving at a service
  - Compared to Virtual Services, D.R.s are used against *real* service endpoints
  - A destination rule defines behaviors against real endpoints, e.g. load balancing
- **Gateways**
  - Manage how traffic enters (ingress) and leaves (egress) the mesh
- **Service Entries**
  - A method to add endpoints to the service mesh that exist _outside_ the mesh (e.g. databases)
- **AuthN & AuthZ Policies**
  - Defines who can do what, and where, in a service mesh

Book Info: An Istio Example

- ”BookInfo” is the Istio “hello world” example application
- We are going to deploy BookInfo into Istio that has itself been deployed by Rancher
- Rancher makes it easy to deploy Istio and manage its capabilities
- This is not going to be an exhaustive demonstration of BookInfo



Demo Time!

Raise your right hand

… and repeat after me

I, Webinar Attendee, do solemnly swear

not to hold Eamon Bauman responsible

for any mishaps that occur during this live demo.

So Helm me Kubernetes

Demo Review

- We deployed Istio using Rancher.
  - Tools -> Istio at the Cluster level
- Rancher deploys and manages the Istio components for us
  - Installed as a Helm chart under the System project
- We configured some basic Istio resources using Rancher
  - Under Resources -> Istio
  - We can manage Gateways, Destination Rules, and Virtual Services
  - Rancher imports traffic graphs and metrics from Kiali

Advanced Topics

- Circuit Breaking
  - We can stop traffic (or redirect it) when certain situations occur
  - Prevent cascading failures, i.e. “break the circuit”
- Fault Injection
  - We can inject faults into our traffic flow to help troubleshoot and prepare for issues
- Observability
  - Distributed tracing via Jaeger helps uncover complex, service-to-service issues
- Mutual TLS
  - Verify both client and server are who they say they are