Microservices under the microscope

QCon Beijing

Ross Garrett
rgarrett@axway.com
@gssor

April 25th, 2015

“To Improve Is to Change; To Be Perfect Is to Change Often”

Sir Winston Churchill

提高就是要改变,而要达到完美就要不断改变。

温斯顿·丘吉尔

We've already been forced to change…

Digital Business has no Border

- Architecting for Mobile isn't enough
- Omni-channel experiences require a new approach
- Digital products are King

Expectations for IT have changed

- **In the past enterprise architecture was designed for known use cases and integrations**
  - **The consumer cloud & mobile application model has changed the way business users view IT products and processes**

- **Today's enterprise must architect for the unknown**
  - **Enterprise IT systems are no longer an island and must build their capacity to integrate with the outside world**


NETFLIX

Netflix has shown the way…
Reactions Adrian Cockcroft received.

"You guys are crazy! Can't believe it"
-2009

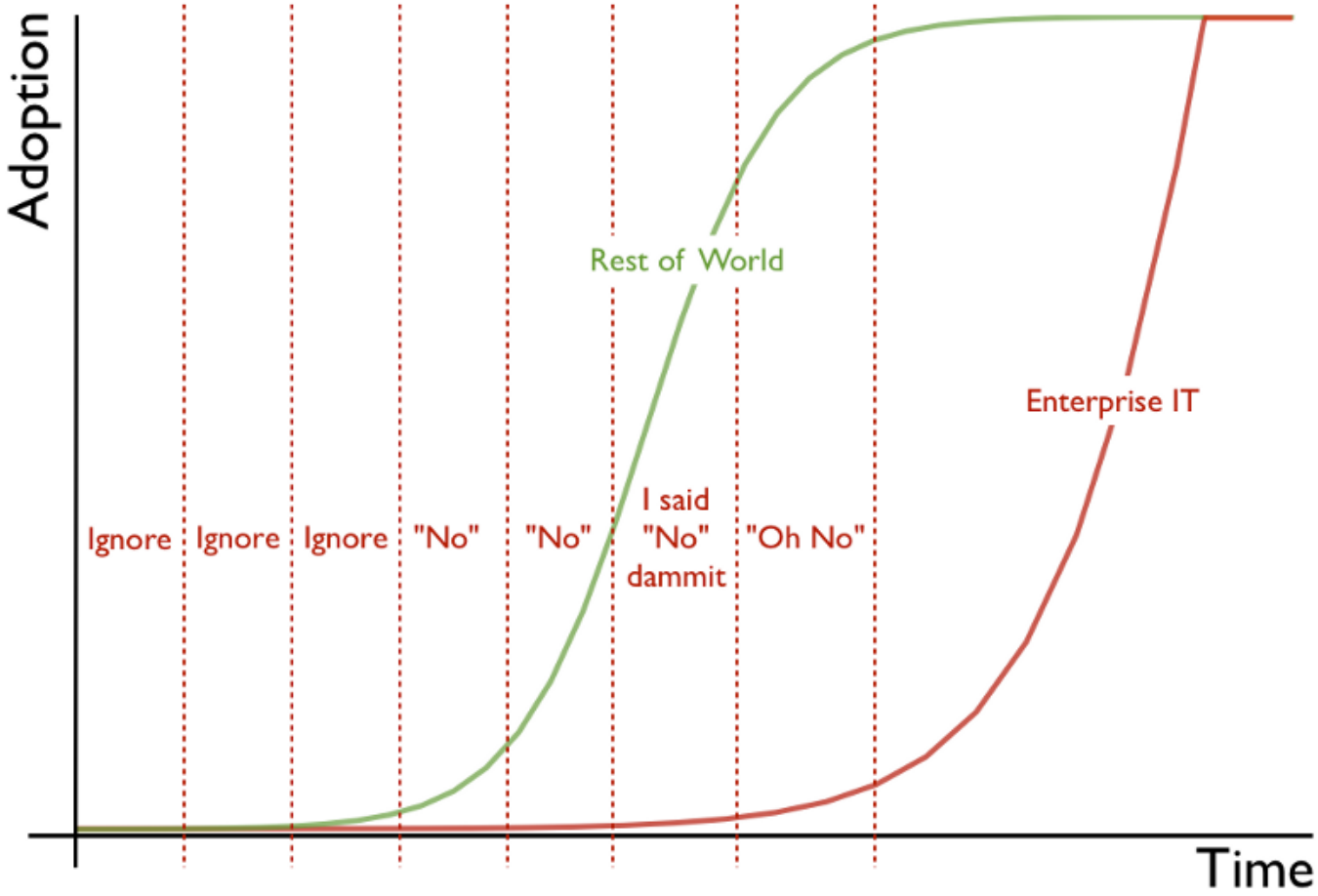
"What Netflix is doing won't work"
-2010

"It only works for Unicorns like Netflix"
-2011

"We'd like to do that, but can't"
-2012

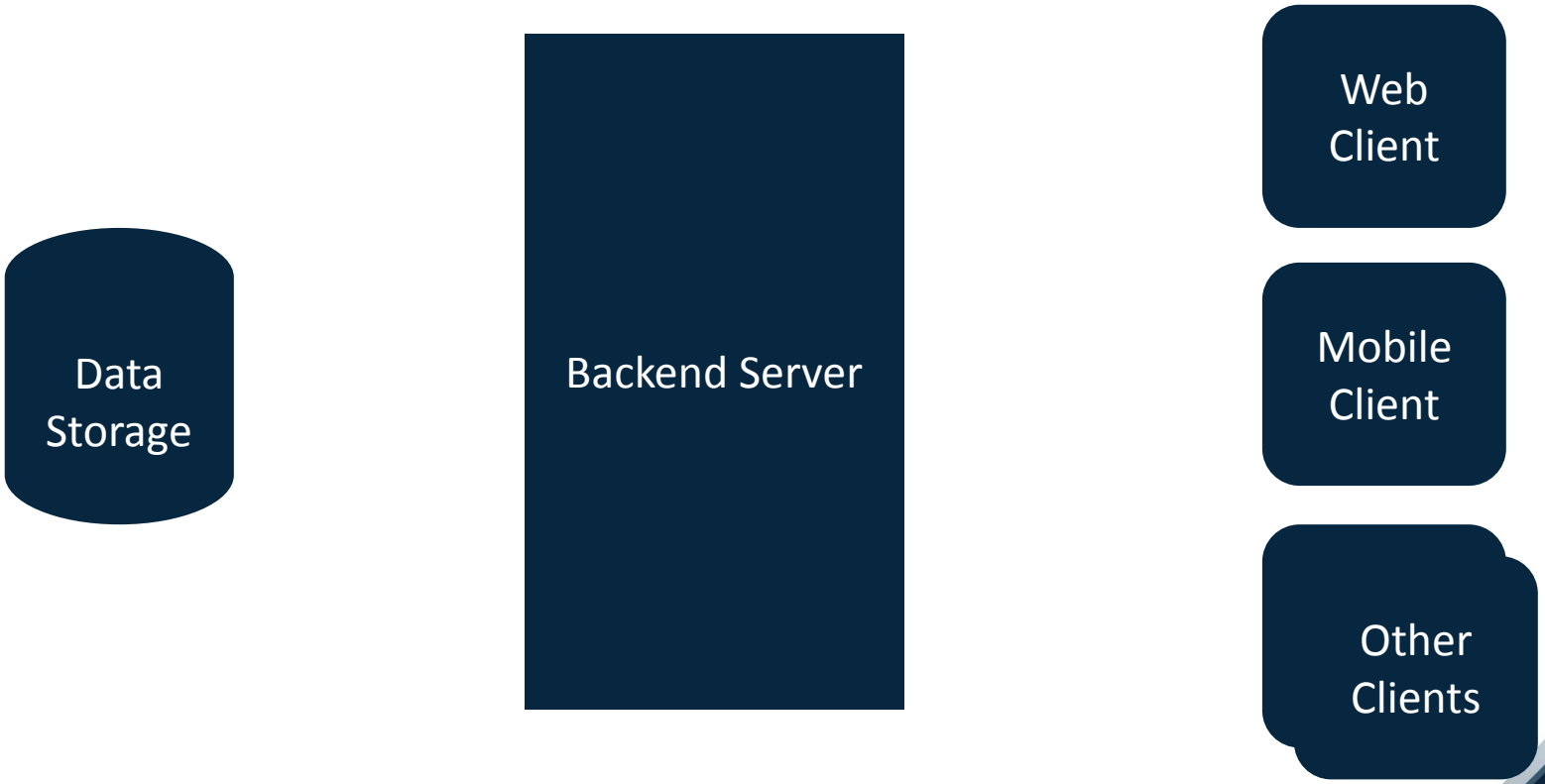
A mandate for change!

Enterprise IT Adoption Cycle



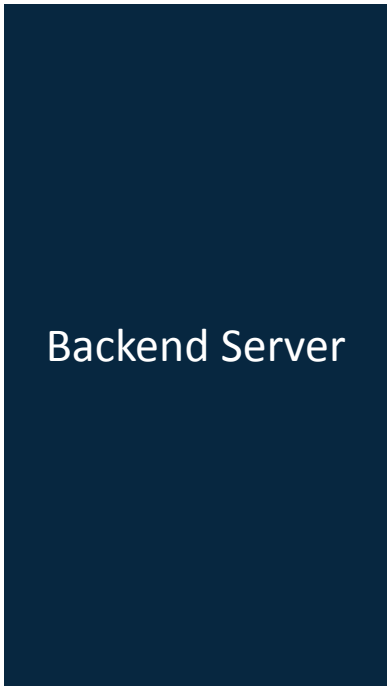
http://blog.gardeviance.org/2012/07/adoption-cycles.html - Simon Wardley

What do Enterprise Applications look like?



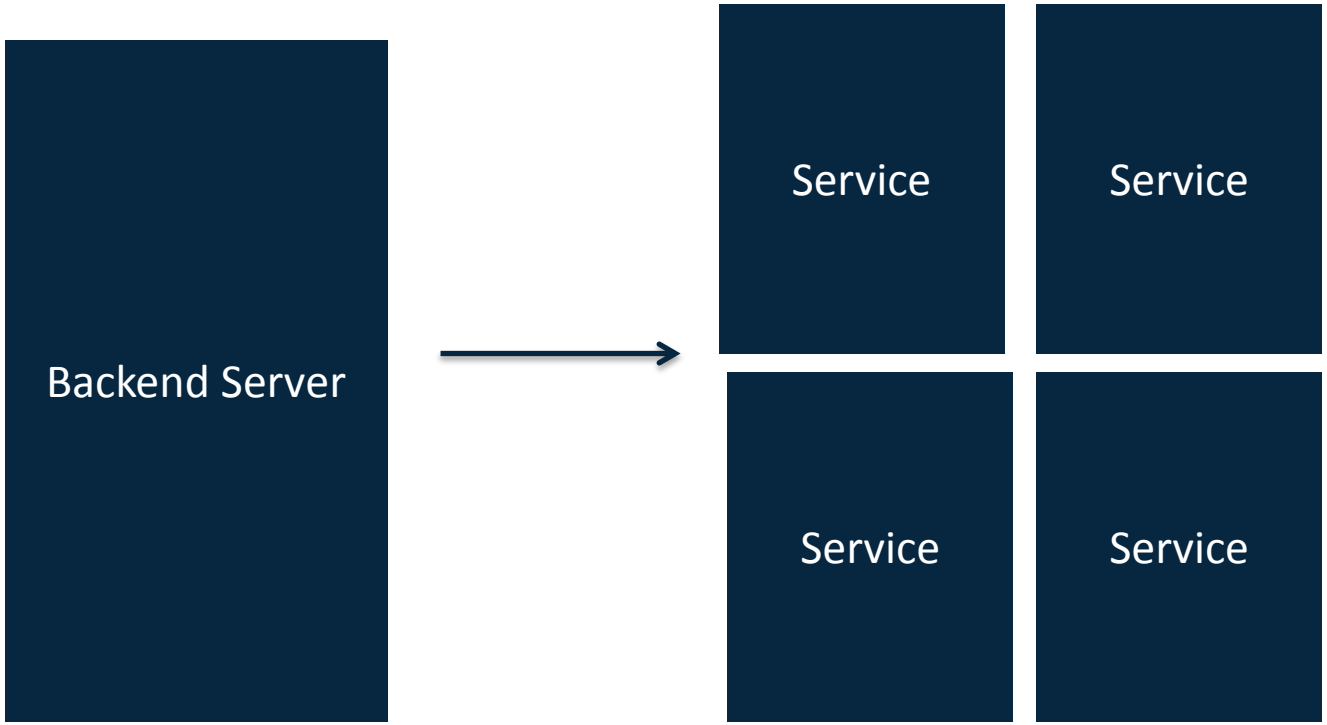
The backend relied on monolithic applications

- All functionality in a single process
- Scale by adding servers



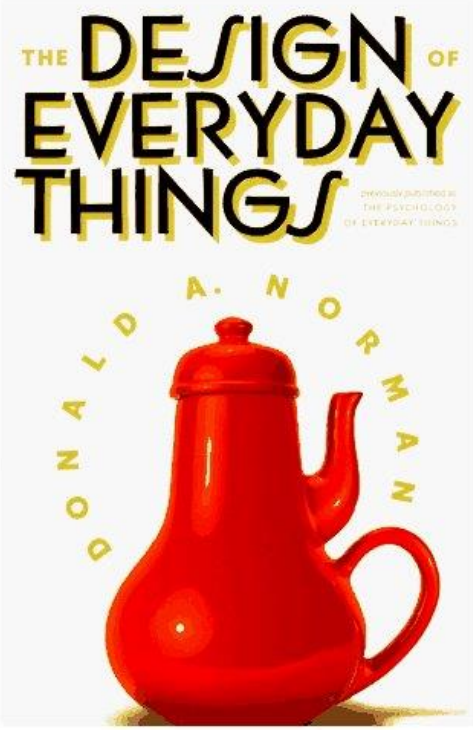
SOA introduced separation

- Separable elements of functionality become services
- Scale & reuse services as needed



Are these services useful?

"The value of a well-designed object is when it has such a rich set of affordances that the people who use it can do things with it that the designer never imagined."

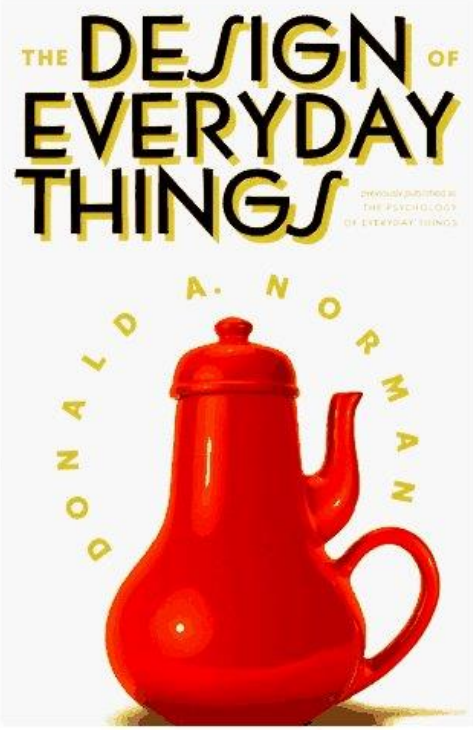


Donald Norman

Are these services useful?

Are these services able to serve today's application needs?

And tomorrow's?



Introducing Microservices

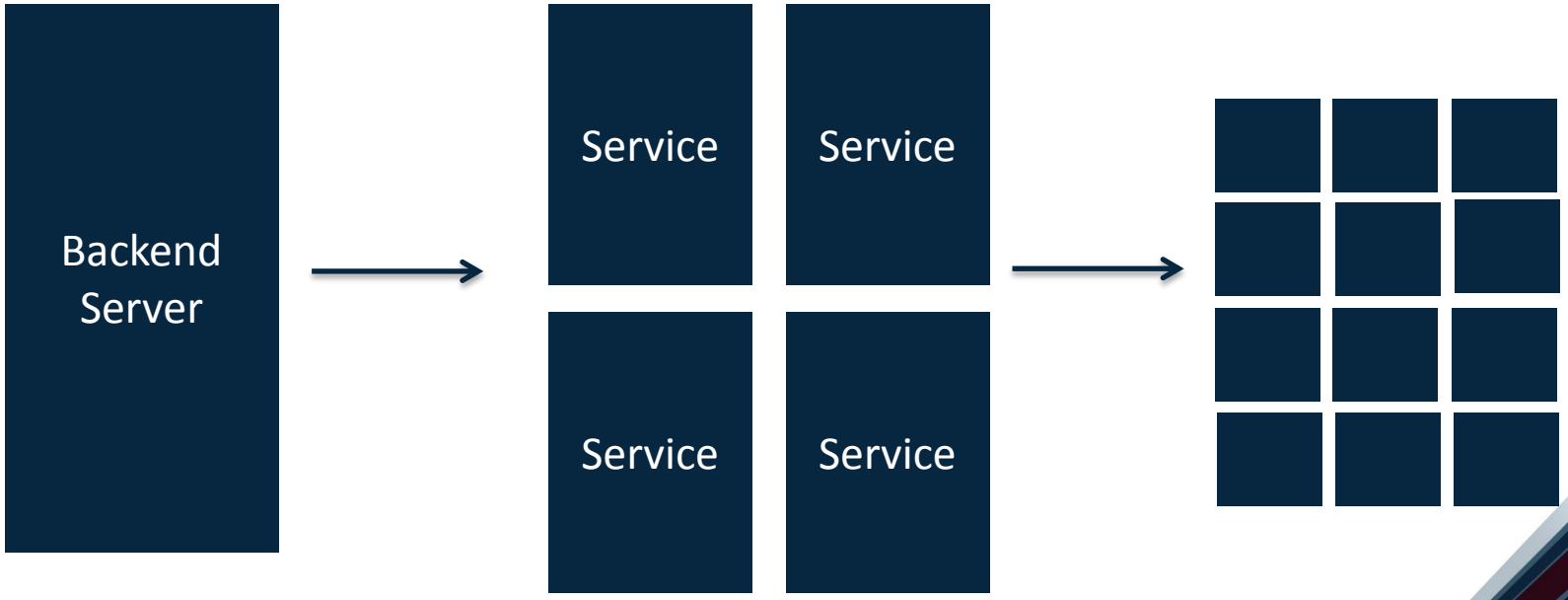
“…the microservice architectural style.. ..is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. …”

Martin Fowler

http://martinfowler.com/articles/microservices.html

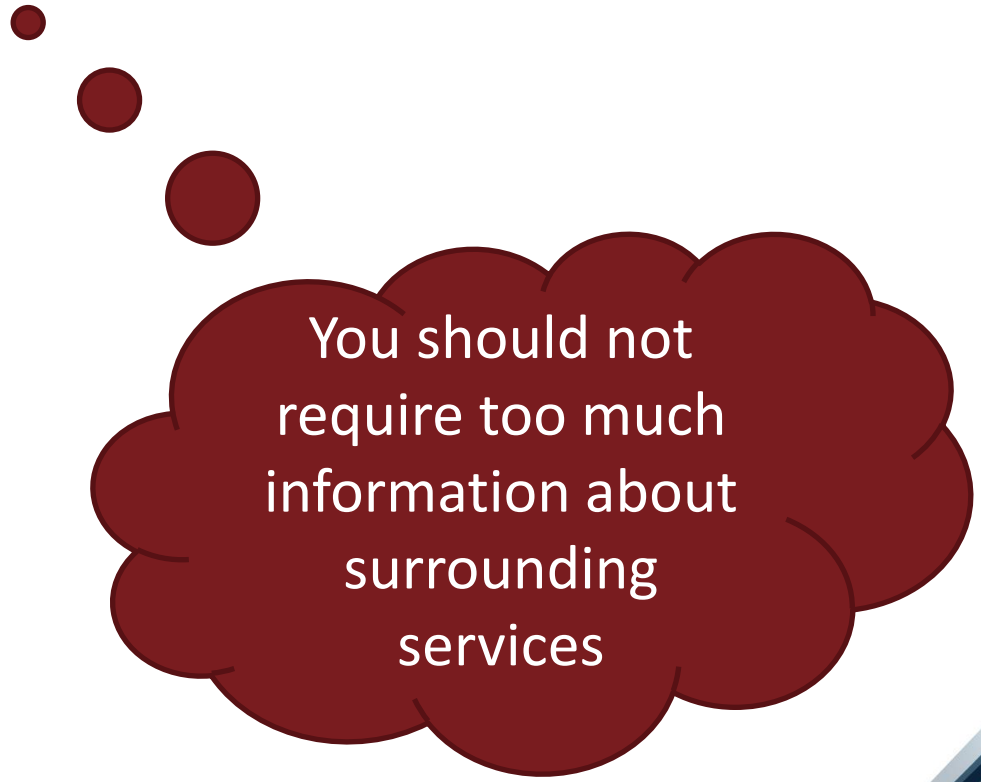
Why Microservices?

- A platform for the business
- Agility
- Not tied to technology



Loosely Coupled

Functionally Bounded

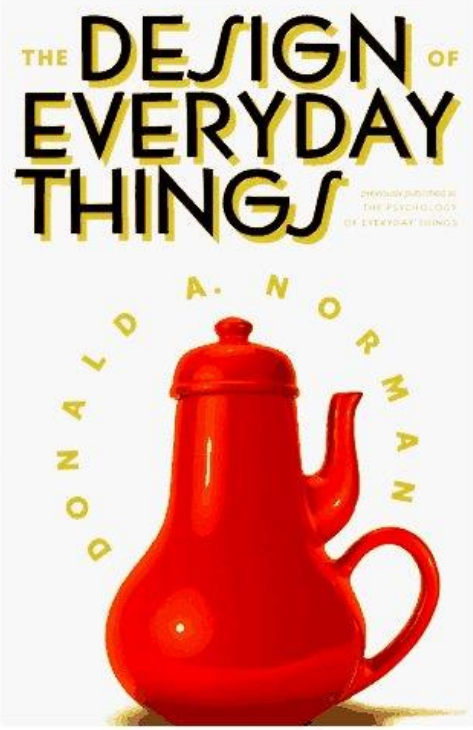


Sounds familiar?

Are these services useful?

How small is microservice?

We want to avoid dumb services that are just CRUD wrappers



SOA vs Microservices

SOA

- Team focused on services
- Services deployed in a shared bus
- Machine readable service registry
- Centralized orchestration
- Centralized data storage

Microservices

- Teams aligned with business
- Services deployed at the edge
- Developer readable catalogue
- Orchestration within each app
- Data storage replicated across atomic instances

Build it, Run it, Own it

- SOA Services were seen as projects
  - The team moves on when the scope of that project is delivered
- Microservices and their APIs must be managed as products
  - Product team owns their service from conception to retirement

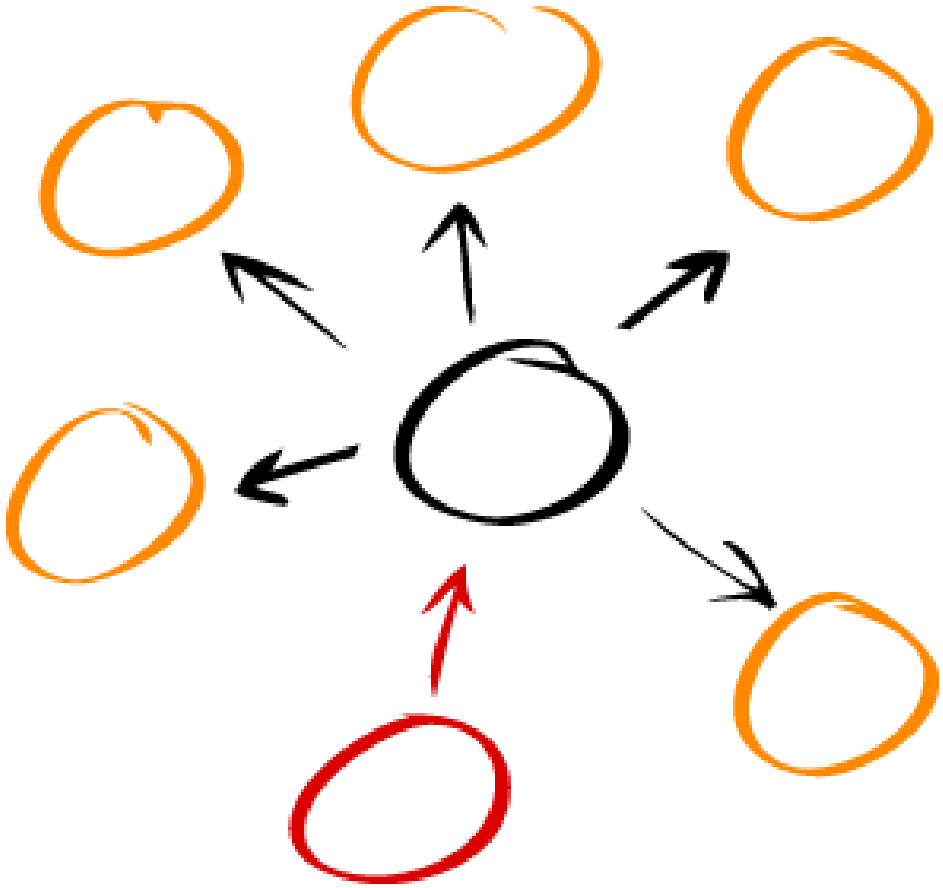
Examining the components of a microservices architecture

Defining Services

Defining Services

- **Build services for business functions not known integration problems**
- **Implement a process to build and deploy each service independently**
- **Existing services can be maintained, while new or partial services are launched**
  - No impact to production until traffic routing is updated

Decentralize Everything



Decentralize Everything

- **Developers can build services on any platform, with any tools**

- **Limit or remove centralized resources**
  - Databases
  - Message Queues
  - Enterprise Service Buses

High Trust, Low Friction

High Trust, Low Friction

- **Developers ~~dislike~~ rebel against strict governance processes**
- **Services should be managed and owned by product teams**
  - Lifecycle management
- **Functional decomposition based on the business**
  - Services should not represent technology or architectural constraints, but rather business requirements

Lightweight Integration Patterns

Lightweight Integration Patterns

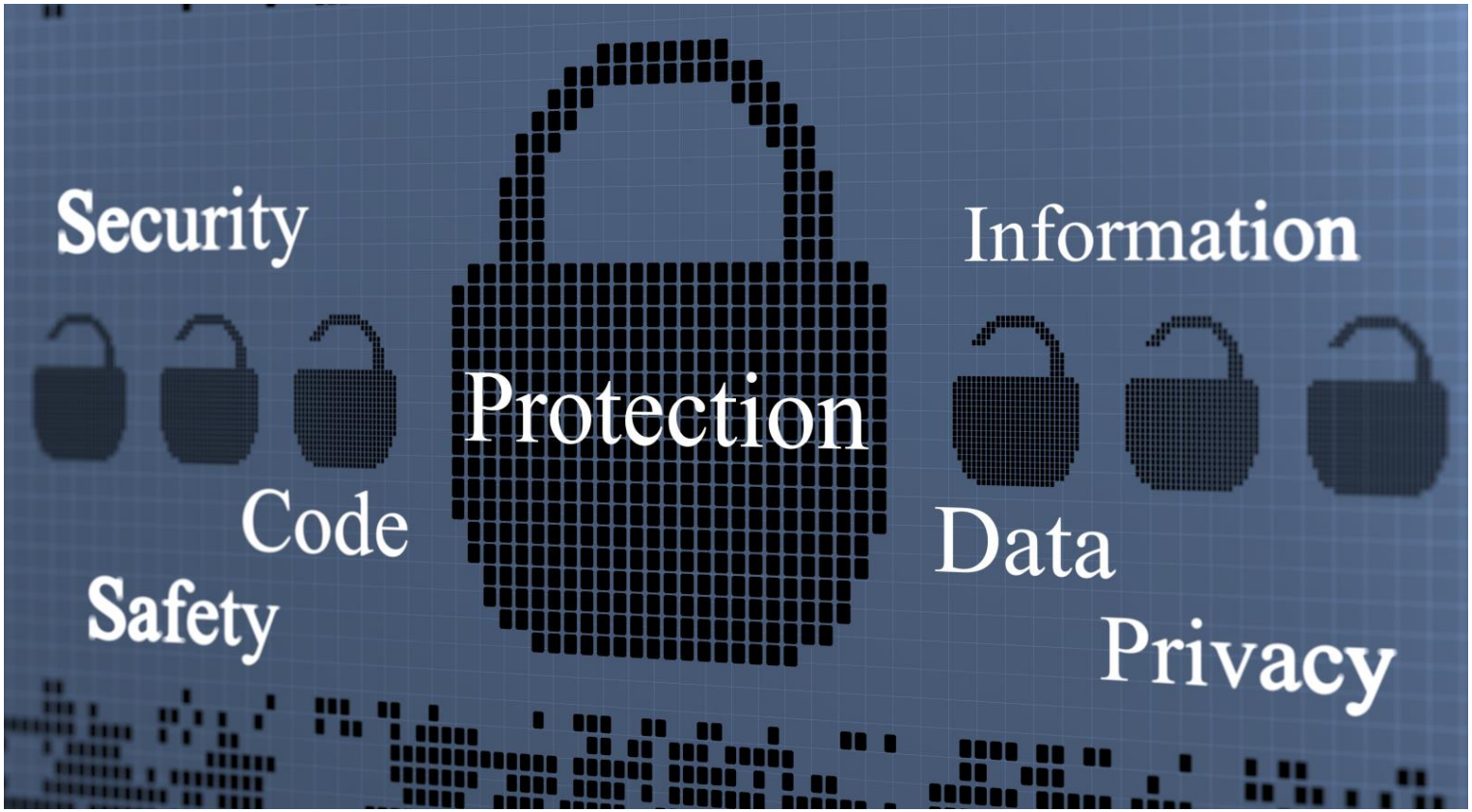
- **Services should be stateless**
- **Hide backend implementation complexity**
- **Document your services and their APIs**
  - Swagger, RAML, Blueprint

Think Differently About API Design

Think Differently About API Design

- **Bring the Web to enterprise integration**
- **Traditional RPC-style APIs can depend on a high degree of context for surrounding services**
- **Design a RESTful representation of your service**
  - This API should look at application needs (outside-in)
- **Hypermedia-Style APIs promote loose-coupling**
  - Use HTTP linking to self-document service capabilities

Design security with public access in mind



Design security with public access in mind

- **Access to services must be managed**
  - Don’t assume internal or private access
- **Define or inherit granular identity and policy rules**
- **Leverage and adhere to security standards**
  - OAuth 2.0
  - API Keys

Architecting for Continuous Change

Holistic approach to digital

The Microservices End Game

The Microservices End Game

- **New versions of services are deployed frequently**
  - Ideally automatically
- **Move away from general purpose orchestration**
  - Apps orchestrate the services they need in the environment they prefer
- **Most architectures will have hundreds of services**
  - But remember useful design

Thank You!

Ross Garrett
rgarrett@axway.com
@gssor