API security in a microservice architecture

Matt McLarty

VP, API Academy, CA Technologies

Feb. 28, 2018

O'REILLY®
Software Architecture

softwarearchitecturecon.com | #OReillySACon

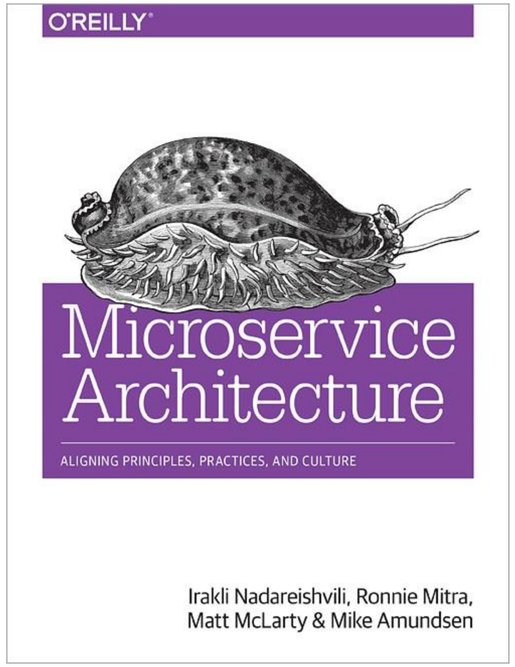
Agenda

- **Purpose and Goals**
- **Background**
- **Current Approaches**
  - Network-level Controls
  - Application-level Controls
  - Emerging Approaches
- **Proposed Approach**
  - Domain Hierarchy Access Regulation for Microservice Architecture (DHARMA)
  - Platform-Independent DHARMA Implementation
- **What Next?**

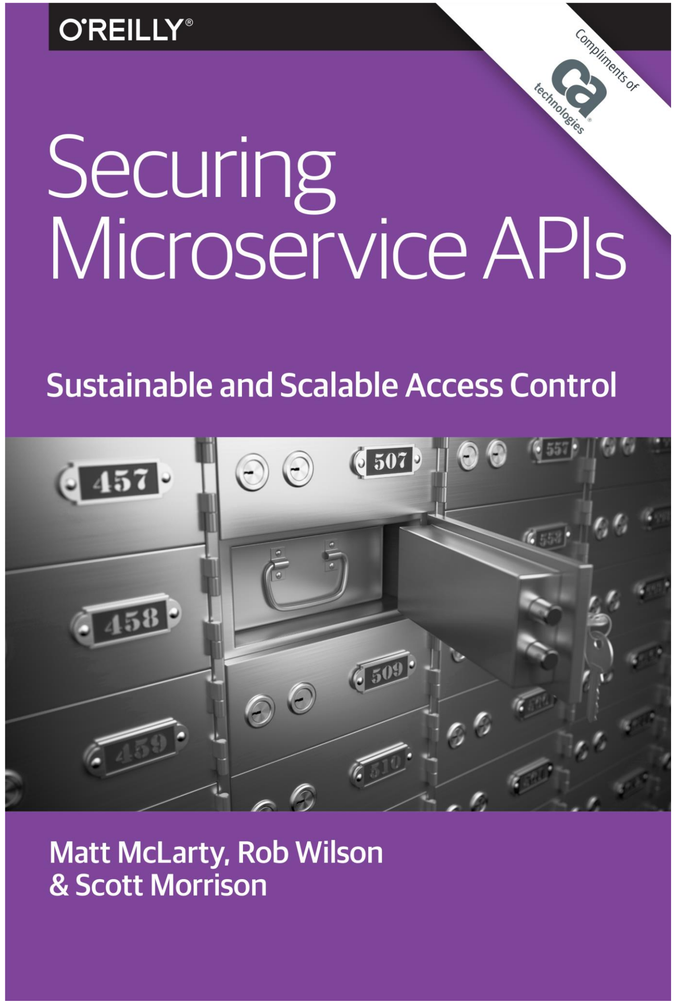
About

Matt McLarty

- Vice President of the API Academy (CA Technologies)
- Co-author of *Microservice Architecture* from O'Reilly
- Instructor for *Microservices for the Enterprise* O'Reilly training
- 20+ years in development, enterprise IT, software architecture
- Architect, writer, speaker
- Live in Vancouver, BC, Canada



O’Reilly Report



https://transform.ca.com/API-securing-microservice-apis-oreilly-ebook.html

Goals

Primary

- Create a comprehensive “literature review” for Microservice API Security
- Define a general model for API access control applicable to microservices
- Refine the general model for practical use in a microservice architecture
- Anticipate the next level of problems and solutions required for microservice API security

Secondary

- Help to develop a common language for microservices and distributed systems in general
- With Fielding as inspiration, try to define a methodology for general solutions like this

Some background…

Microservice Architecture Characteristics

Service orientation

Independent deployability and manageability

Ephemerality and elasticity

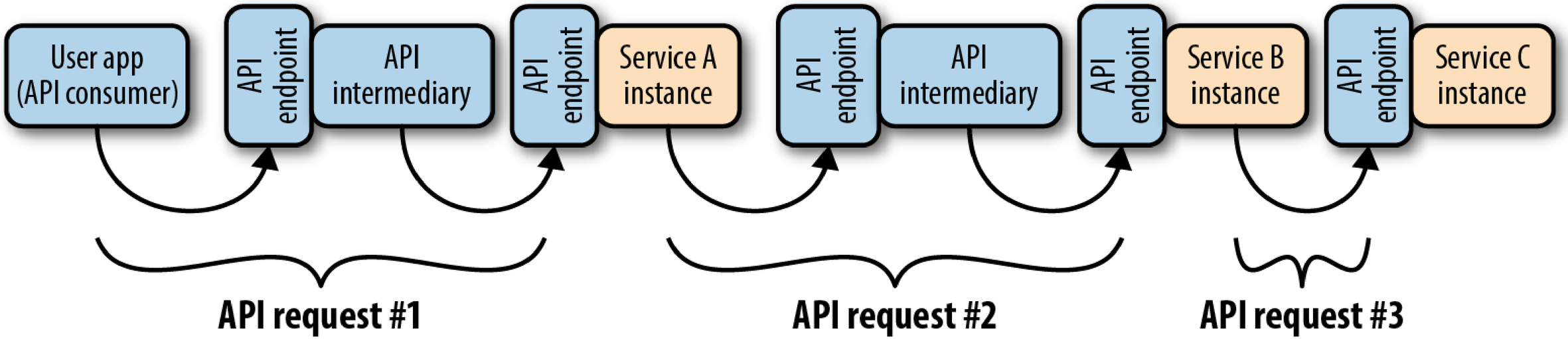
Web API communication

Container-based deployment

Microservice API Terminology

- Service
  - Service Instance
- API
  - API Endpoint
- API Request
- API Response
- API Consumer
- API Provider
- API Intermediary
  - API Gateway
  - Service Proxy

The Microservice API Landscape



IAAA Framework for Microservice APIs

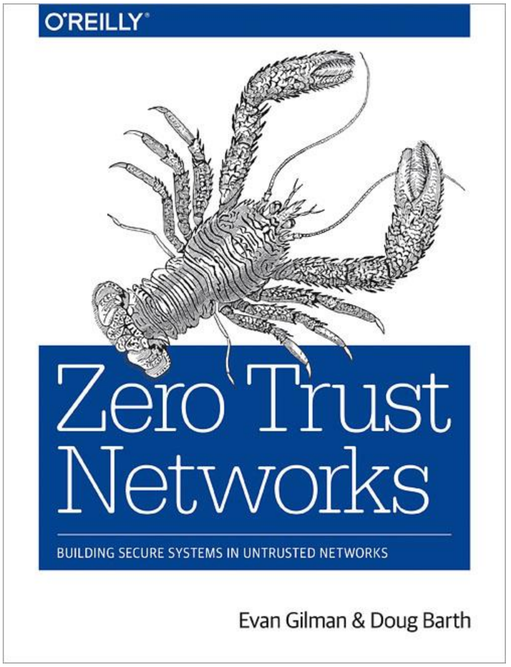
Identification	• Must support multiple identities and attributes (end users, system components, domains)
Authentication	• Must support multiple authentication methods as well as delegated authentication
Authorization	• Authorization for a single request may be decided at multiple points in the request path
Accountability	• Capture of relevant security data or metadata from API messages

Current approaches…

About Trust

- Trust is fundamental in distributed systems
- Implicit trust is everywhere!
  - e.g. network isolation
- Trust is about understanding and compromise

Trusted communication should be more efficient than untrusted



Network-Level Controls

- Localhost isolation
- Network segmentation
- SSL/TLS

When to Use Network Segmentation

1. When you trust the physical security of the server and network infrastructure
2. When you trust the infrastructure isolation mechanism and process
3. When you trust every entity on the network segment

SPIFFE

- “Secure Production Identity Framework for Everyone”
- PKI functions for ephemeral environments
- SVID’s
  - “SPIFFE Verifiable Identity Documents”
  - Identity for services and other components
- SPIRE
  - “SPIFFE Runtime Environment”
  - Agent/Server architecture

Application-Level Controls – Traditional Web Tokens

Cookie-based Sessions

- Can have a role as long as storage is performant and scalable
- Session ID open to hijack
- Sessions do not cross security domains

SAML

- Some concepts useful
- Too centralized and heavy for microservice architectures
- Does not support delegation

Application-Level Controls – API-oriented Tokens

API Keys

- An application identifier, not a security mechanism!

OAuth 2.0

- Framework for API authorization, supports delegation
- Agnostic of token types

OpenID Connect

- Extends Oauth 2.0 with ID Token

JWT

- Packaging format for exchanging claims
- Convenient and popular in practice

Application-Level Controls –Token Types

Opaque (“by-reference”) tokens

- Indecipherable to third parties, but require centralized management

Transparent (“by-value”) tokens

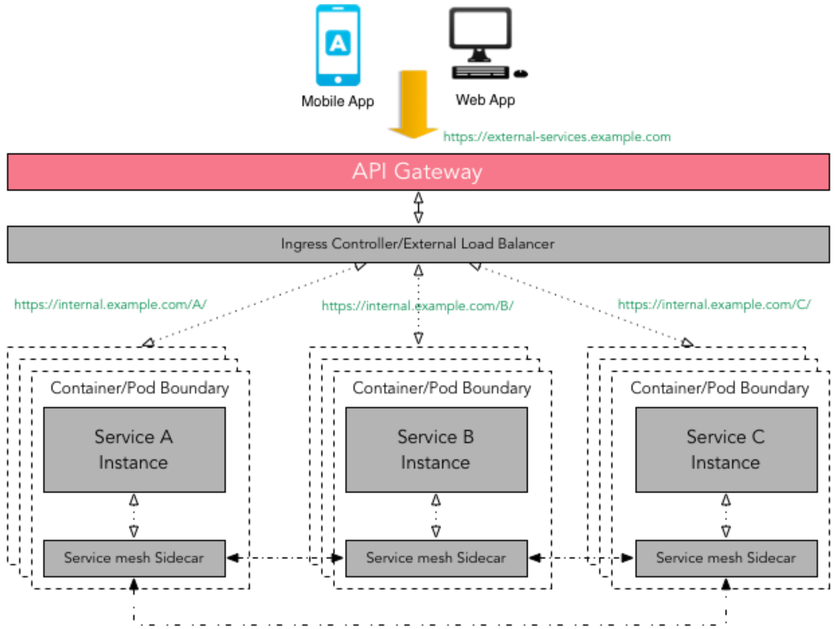
- Management can be decentralized, but accessible to third parties

When to Use Tokens

1. You need to authenticate and authorize users and applications.
2. Your trust needs to cross boundaries, which might be organizational, geographical, application, or virtual.
3. You can tolerate ceremony between applications and users.
4. You have the infrastructure to facilitate the token exchange.

Infrastructure – API Intermediaries

- API Gateway
  - “North-south” (proxies consumer-to-provider)
  - Centralized at the perimeter
  - Fully-featured
- Service Proxy
  - “East-west” (proxies service-to-service)
  - Local to service (sidecar)
  - Streamlined



https://abhishek-tiwari.com/a-sidecar-for-your-service-mesh/

Infrastructure – Network Overlays

- Platform-specific capabilities
- Open source projects
  - OpenContrail, Romana: network overlays
  - Project Calico: native support for Docker, Kubernetes, Mesos
  - Cilium: uses Linux kernel modifications

Infrastructure – Platform Capabilities

Kubernetes

- Network rules restrict communication between various abstractions: clusters, nodes, pods, services
- Authentication ultimately left to application logic

Cloud Foundry

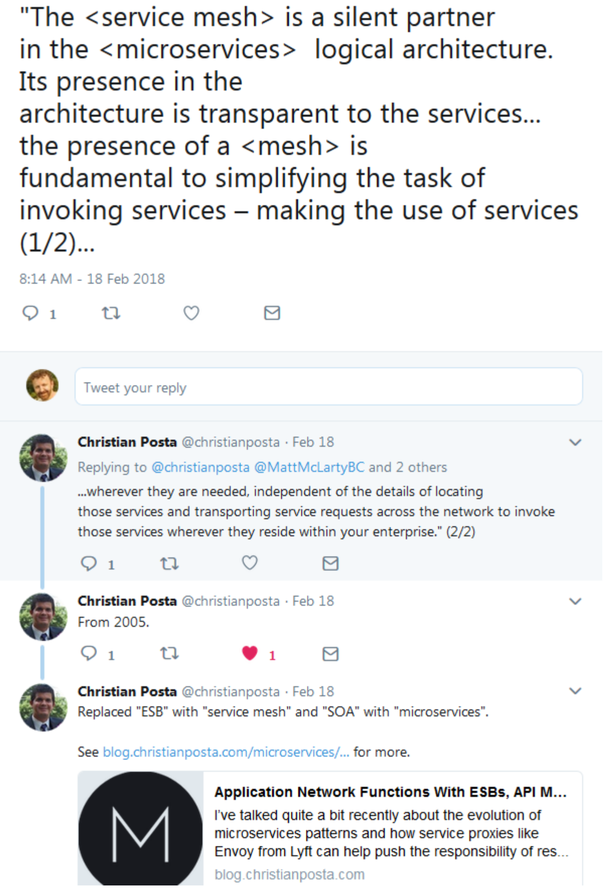
- UAA for user authentication (OAuth 2.0 with JWT's)
- Multiple options for network ACL's

AWS

- Built-in proprietary IAM and certificate management
- API access control generally left to application logic

Emerging Approaches – Service Mesh

- Both an emerging and a time-worn concept →
- In practice, network of service proxies
- In theory, general policy enforcement for “the system”
  - Routing, service level management, security
- Sample implementation: Istio
  - “Control plane” for the service mesh
  - Istio-Auth for authentication, using SPIFFE



Emerging Approaches – Serverless

- Constrained but convenient
  - Less access to infrastructure configuration
  - Distinction between functions and communication
- Access control tied to platform
  - e.g. AWS Lambda tied to AWS IAM + AWS API Gateway

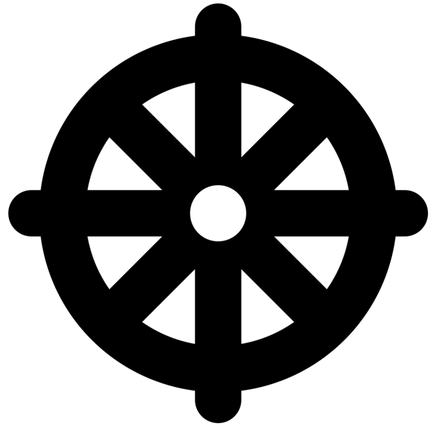
A Proposed Approach…

Common Patterns in Microservice API Security

- “Zero trust” not a common practice due to inefficiency
- Many multi-faceted approaches with heterogeneous parts
- Many platform-specific capabilities
- Binary pattern:
  - “Fast lane” for traffic based on trust
  - “Slow lane” for untrusted traffic requiring authentication

Domain Hierarchy Access Regulation for Microservice Architecture (DHARMA)

- A multi-cloud approach to API security in a microservice architecture
- Applicable at any level of the architecture
- Agnostic of domain methodology



What's in a name?

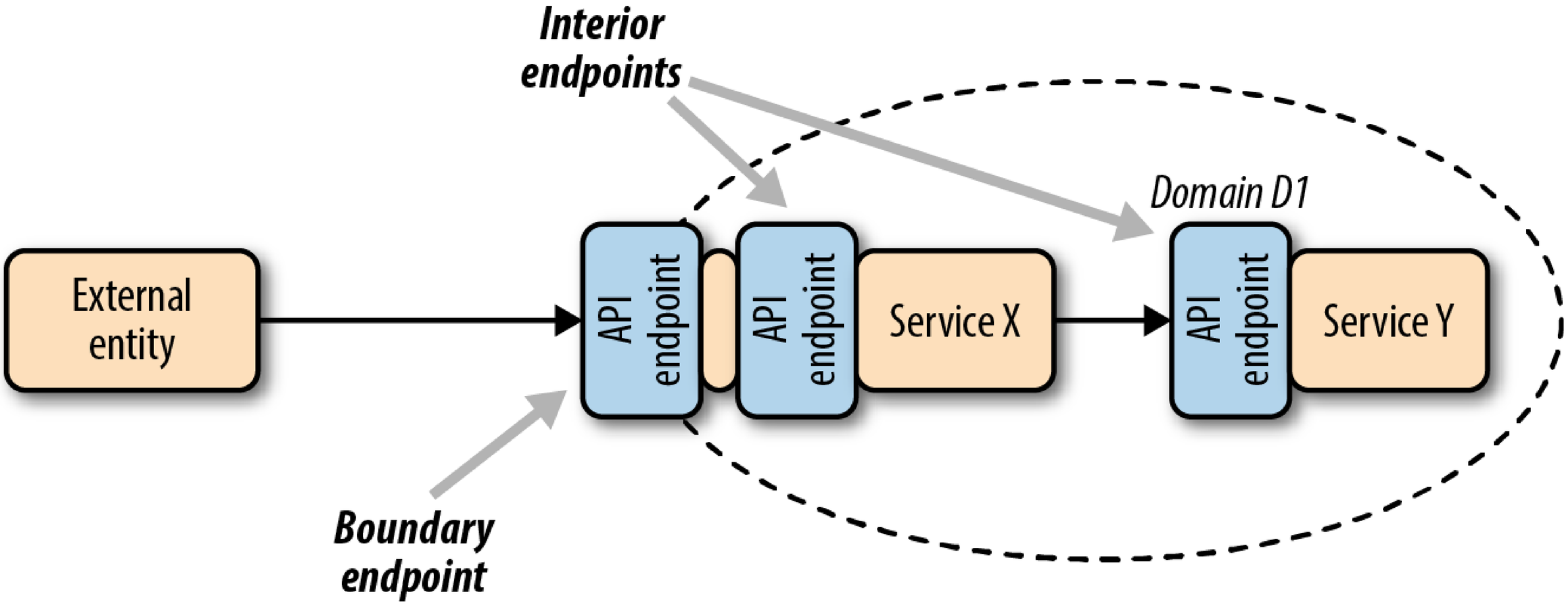
- Dharma *n.* – The principle of cosmic order
  - *We want order in a complex system*
- Significant concept in multiple religions
  - *We want a multi-cloud solution*
- Wheel of Dharma: 
  - *Helm of Kubernetes:* 

(And NO… this has nothing to do with the show "Lost"!)

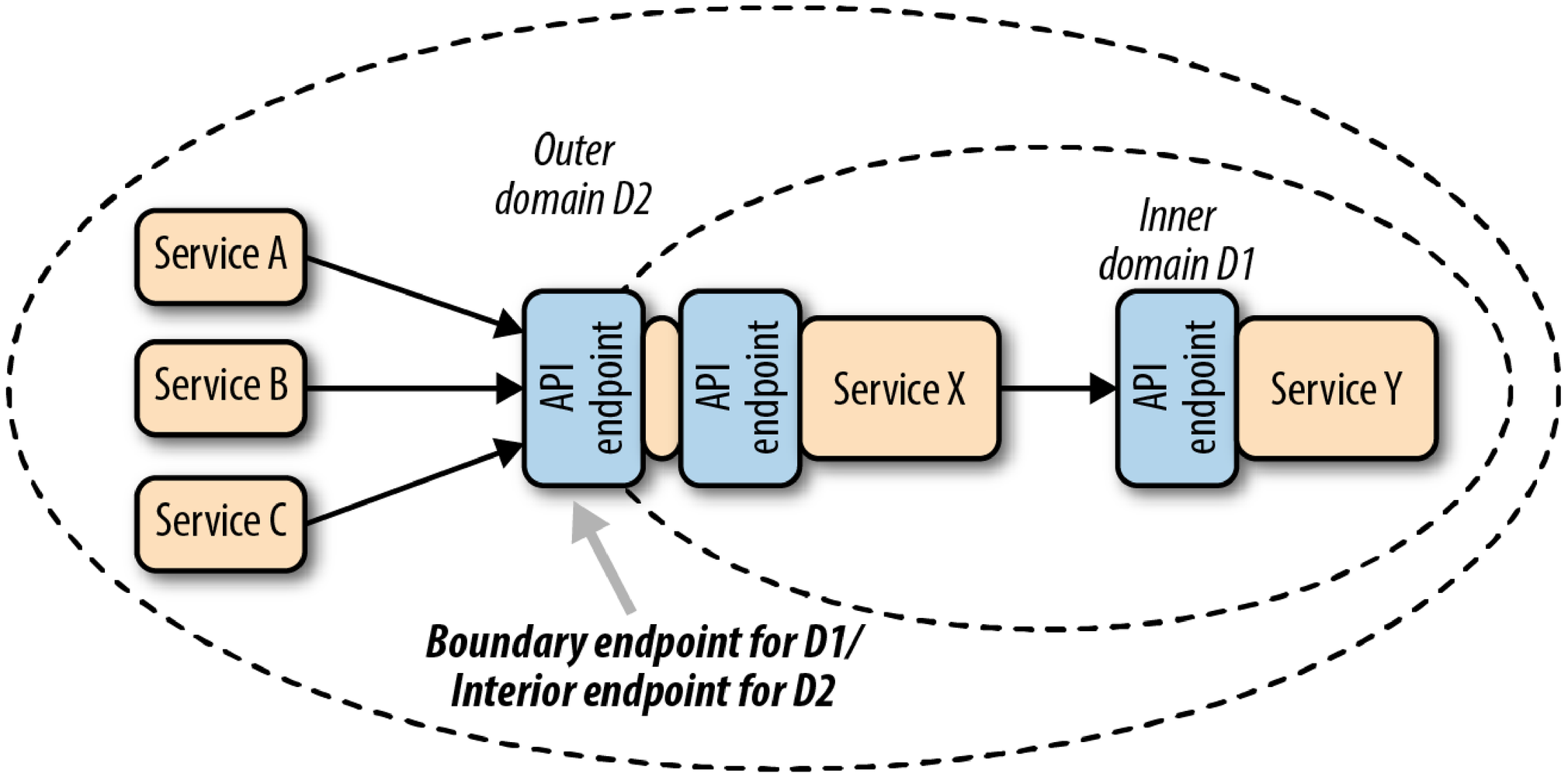
DHARMA Foundational Concepts

Concept	Definition
Trust Domain	A set of services that communicate with each other in a privileged way
Domain Relation	The reason for a domain's services to be grouped together
Trust Mechanism	The method used by services within the domain to verify that an API request is coming from a trusted source
Access Mechanism	The method that allows API requests from outside the domain to be authenticated and authorized
Interior Endpoint	An API endpoint that is accessible to other services within the domain, authorized through the domain's trust mechanism
Boundary Endpoint	An API endpoint that is accessible to services outside the domain, authorized through the domain's access mechanism
Hierarchical Endpoint	An API endpoint that is an interior endpoint for one domain and a boundary endpoint for another

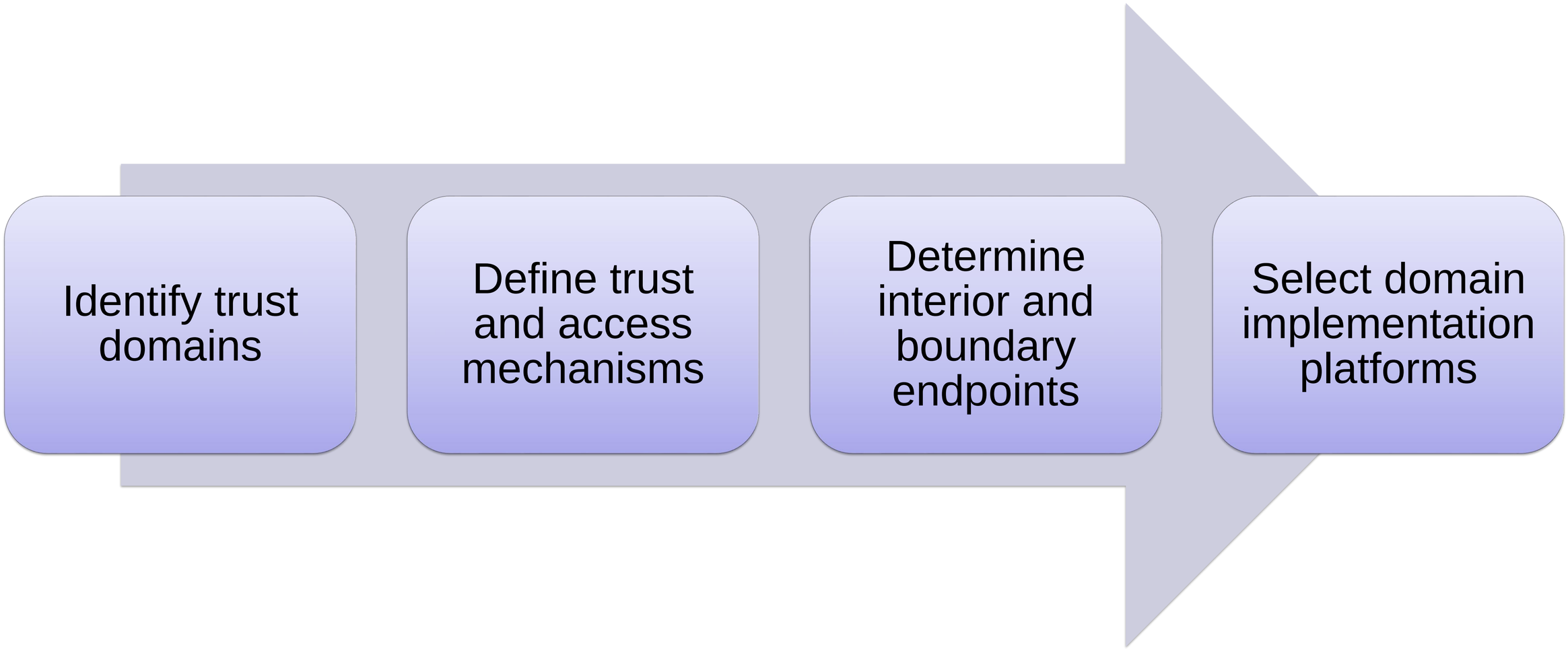
DHARMA Request Flow – Single domain



DHARMA Request Flow – Two domains in a hierarchy

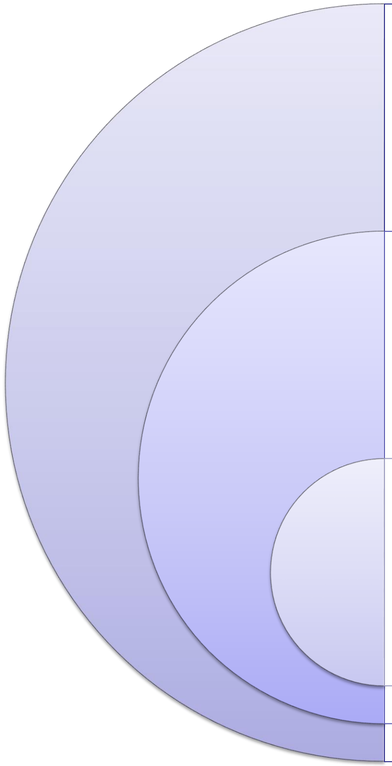


A DHARMA Design Methodology



Platform-Independent DHARMA Implementation

Domain Hierarchy



Unbounded Area	•External consumers •Beyond org's control
Outer Domain	•Public services •Experience-oriented
Inner Domain	•Private services •Logic-, data-oriented

Platform-Independent DHARMA Implementation

Domain	Access Mechanism	Trust Mechanism
Outer Domain	OAuth 2.0, opaque access token	Signed JWT using org-issued certificate
Inner Domain	Signed JWT using org-issued certificate	Network isolation, optionally propagated JWT

Platform-Independent DHARMA Implementation

Implementation considerations

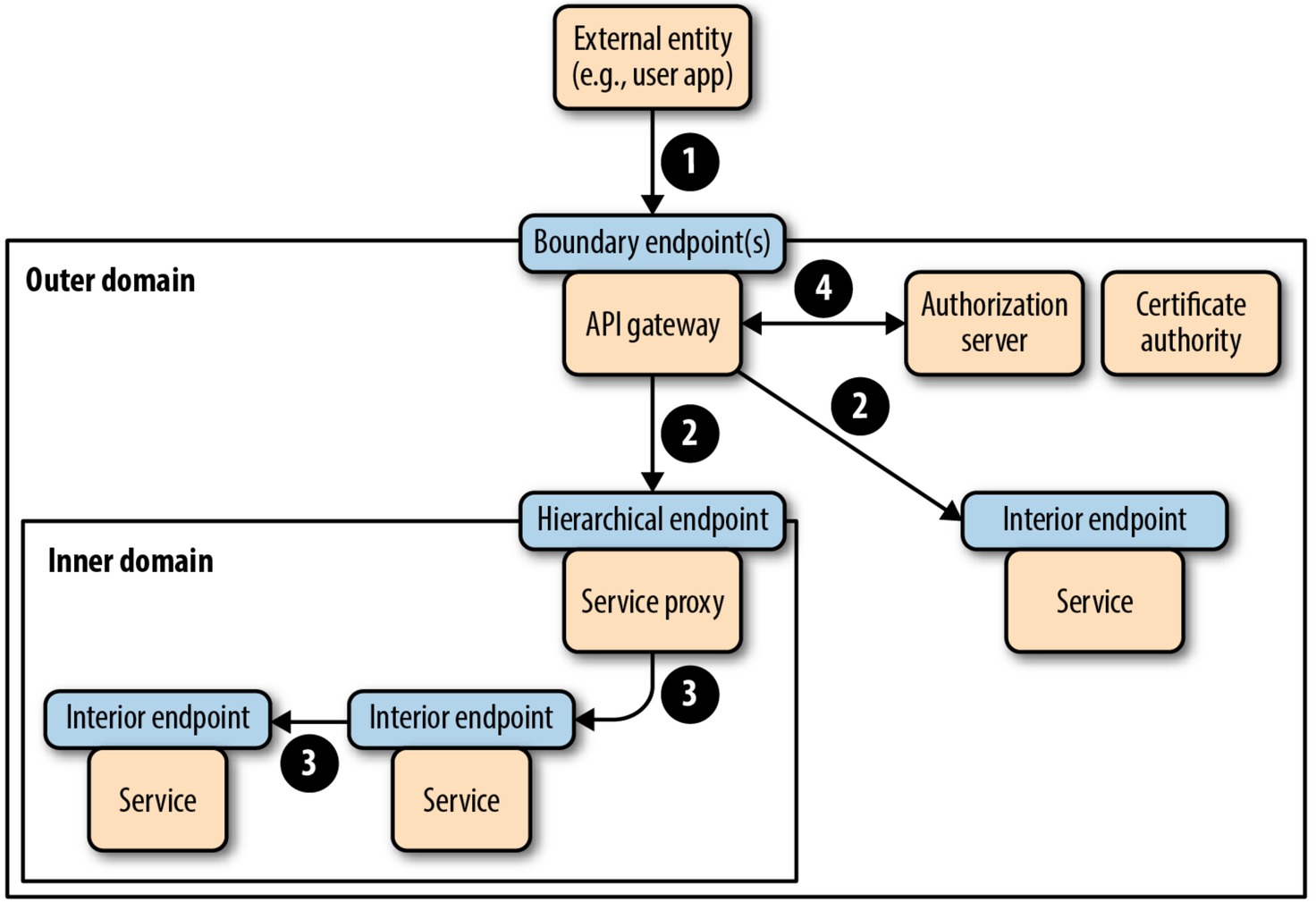
- Certificate management
- Token management
- Component provisioning
- Service and endpoint deployment
- Accountability

Platform-Independent DHARMA Implementation

Interaction	Identification	Authentication	Authorization
External Client Request	External client obtains access token from authorization server, sends on API request to outer domain boundary endpoint	Receiving API Gateway sends access token to authorization server for validation	Authorization server validates access token, exchanges for JWT which is sent back to API Gateway, which forwards request to service's interior endpoint
Outer Domain Service-to-Service Request OR Outer Domain-to-Inner Domain Request	Service consumer either sends previously obtained JWT, or obtains new JWT from Authorization Server and sends on API request to outer domain interior endpoint/inner domain boundary endpoint	Receiving service proxy validates token signature and certificate chain	Service checks JWT claims and processes accordingly
Inner Domain Service-to-Service Request	Service consumer either sends previously obtained JWT, or obtains new JWT from local secure token service and sends on API request	Trusted based on network isolation	Service checks JWT claims and processes accordingly

Platform-Independent DHARMA Implementation

1. API request with valid Oauth 2.0 access token
2. API request with signed JWT (domain CA-issued certificate)
3. API request with JWT for accounting, not authorization
4. Token dereferencing/validation/exchange

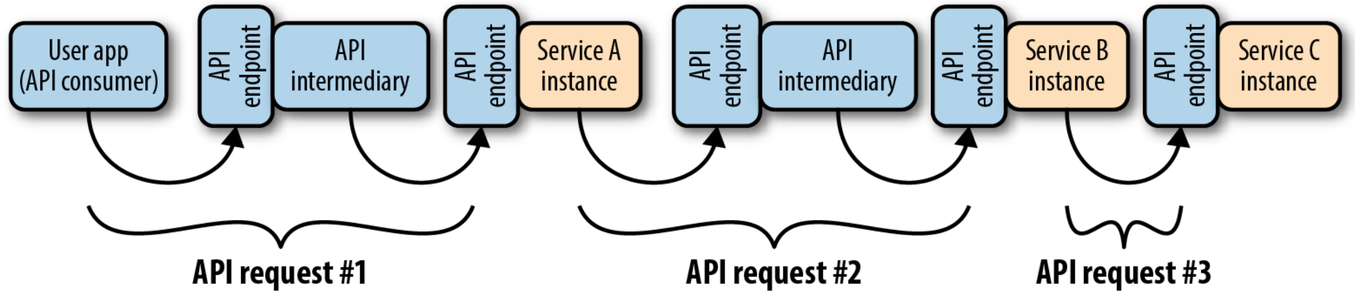


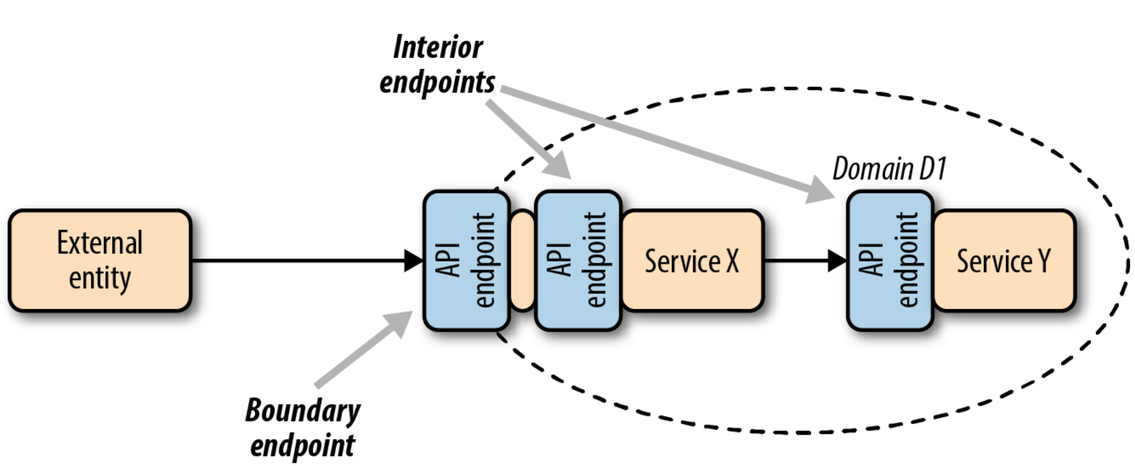
DHARMA Developer Experience

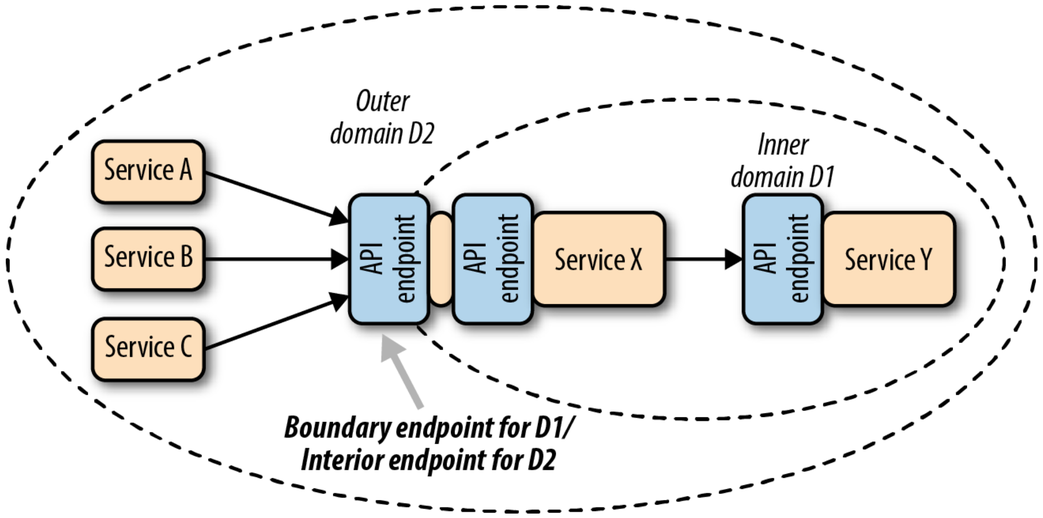
Enabling Access Control for a Service/API	• Service developers should only need to consider deployment domain, claim-related authorization logic, and API message auditing within the service
Publishing and Discovering API Access Control Policies	• Policies should be articulated clearly, platform agnostic (e.g. OpenAPI) • Provide tooling for API consumers
Access Control Policy Change Management	• Organization-wide policies enforced by API intermediaries for ease of change

What next?

Standardizing the Language of Microservices







Refining DHARMA

- Vetting the implementation example
- Platform-specific implementations
- Re-casting existing security approaches

Extending DHARMA

- Metadata for interoperability
- Other synchronous protocols (e.g. gRPC, GraphQL)
- Event-based/reactive systems (e.g. Kafka)

Conclusion

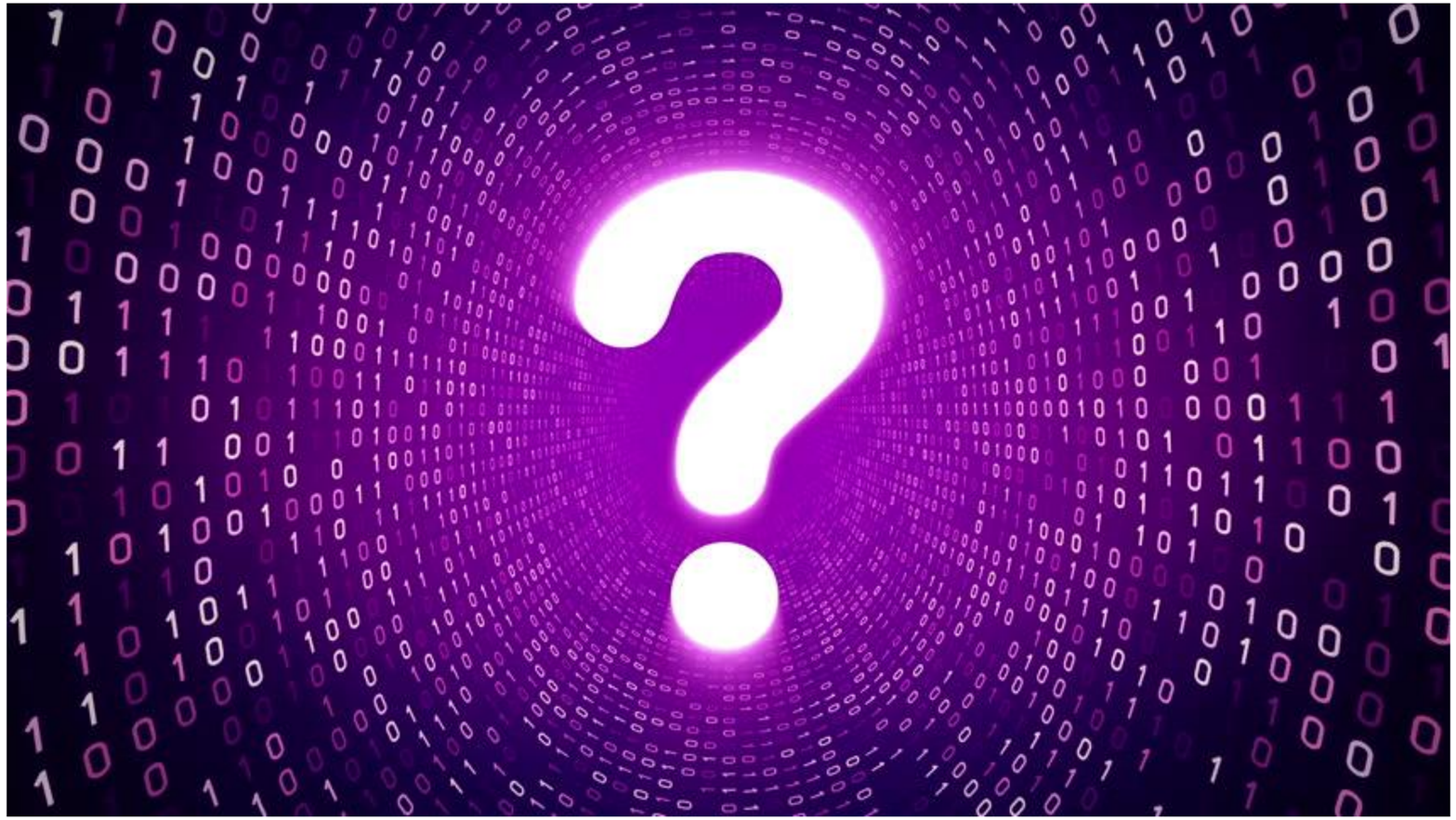
API security is essential in a microservice architecture

A wide variety of current approaches are in use, based on networks, tokens, platforms and solutions

DHARMA offers an adaptable methodology for API access control in a microservice architecture

Lots of room to evolve and refine DHARMA to cover other gaps in the microservice API security landscape

Questions?



Thank You!

Matt McLarty
Vice President, API Academy, CA Technologies
matthew.mclarty@ca.com

@mattmclartybc
www.slideshare.net/MattMcLarty
linkedin.com/in/mattmclartybc

apiacademy.co