Cooperators' Journey towards Microservice Architecture & Enterprise Integration Hub

Shahram Jalaliniya

Architecture, IT Strategy & Applied Innovation

Sep 2021

the co-operators®
A Better Place For You®

Agenda

- Who am I?
- The Cooperators Company
- Cooperators Strategies
- Architecture, IT Strategy & Applied Innovation Team
- Cooperators' Microservices Journey
- Enterprise Integration Hub
- Enterprise Service Reference Architecture
- Future Expansion
  - Service Mesh
  - Enterprise Business Process Automation

Shahram Jalaliniya

Now

2018

Enterprise Architect

Part-time Professor

the co-operators

Seneca

CENTENNIAL COLLEGE

Postdoc

2016-6

Internet of Things & People Research Center
Malmo University

Postdoc

2016-3

Gaze IT: Gaze interaction for motor disabled people

DTU

PhD

2016

iCareNet

2014

MSc in Software and Technology
PhD in Computer Science
UX wearable computers @ healthcare

MSc

2011

MSc. Information Systems

LUND UNIVERSITY

MSc

Working as IT Consultant in Haseb System, ISIran, Infotech, etc.

BSc

2002

Industrial Engineering


the co-operators
Insurance/Financial Services

75 years strong

Established in 1945 by a group of farmers to provide financial security for their families and communities

Decision making is guided by co-operative principles

Take a long-term view, balancing profitability with client and community needs

the co-operators®

Offices from coast to coast

- 6,454 employees
- 2,303 licenced insurance representatives
- Serve 238 credit unions; more than 5.2 million members

Source: The Co-operators 2020 Integrated Annual Report

the co-operators®

The Co-operators: A wealth of experience

The Co-operators Group Limited is a leading Canadian co-operative, which offers multi-line financial services and insurance with $47.4 billion in assets under administration. Our group of companies provides financial solutions and security through property and casualty insurance, life insurance, wealth management solutions, institutional asset management, and brokerage operations.

Property & Casualty Insurance

- We insure more than 890,000 homes and more than 1.5 million vehicles
- We provide coverage for 41,000 farms and 265,000 businesses

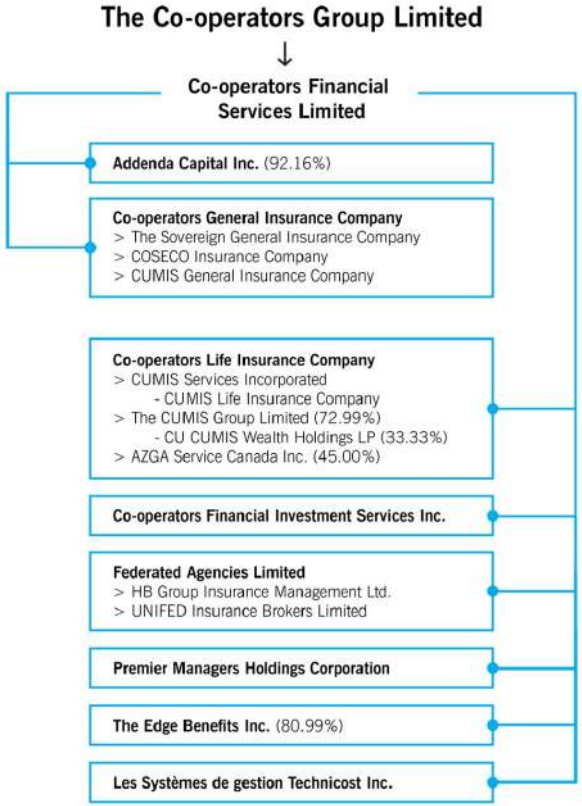
Life insurance

- We protect 522,000 lives
- We insure 230,000 employees through Group Benefits plans
- We offer a wide range of Wealth Management products
- We provide Creditor Life insurance to 545,000 Canadians

Investments

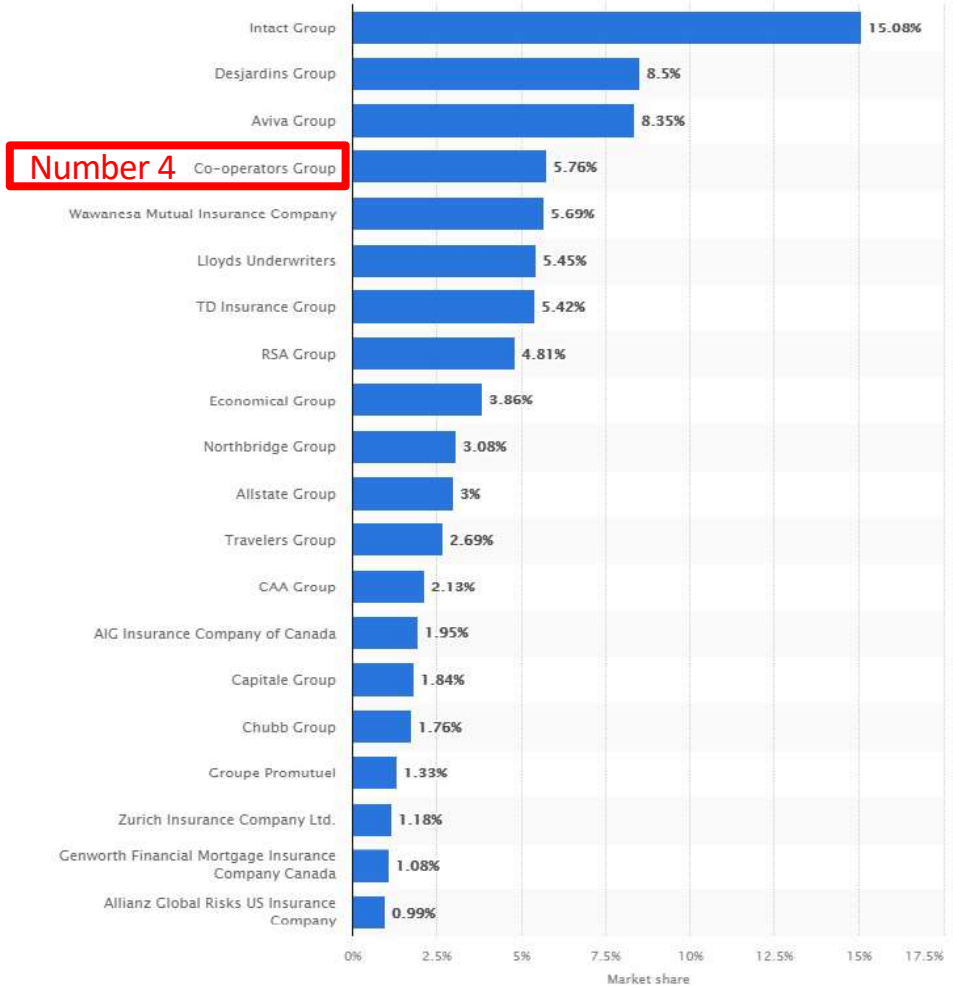
- We manage assets for 171 institutions including pensions, insurance companies, co-operatives, endowments, and foundations.
- Mutual funds are available through Co-operators Financial Investment Services Inc.

Source: The Co-operators 2020 Integrated Annual Report



the co-operators®

Market share of the leading private P&C companies in Canada in 2019



Source: https://www.statista.com/

Our Strategy: A bridge to the future

the co-operators®

Strategic Plan: 2019 to 2022

the co-operators®

COLT (Cooperators Operation Leader Team)

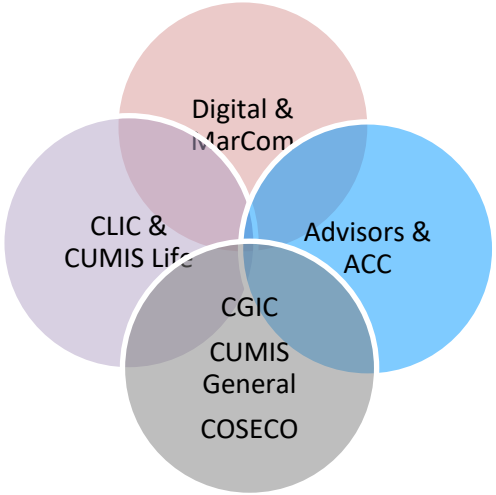
Emmie Fukuchi, **EVP, Chief Digital and Marketing Officer**

Alec Blundell, **EVP, Chief Operating Officer, CLIC; President and COO CUMIS**

Lisa Guglietti**, EVP, COO, P&C Manufacturing**

Kevin Daniel, **EVP, Chief Client Officer**

Carol Poulsen**, EVP, Chief Information Officer**



Mandate:

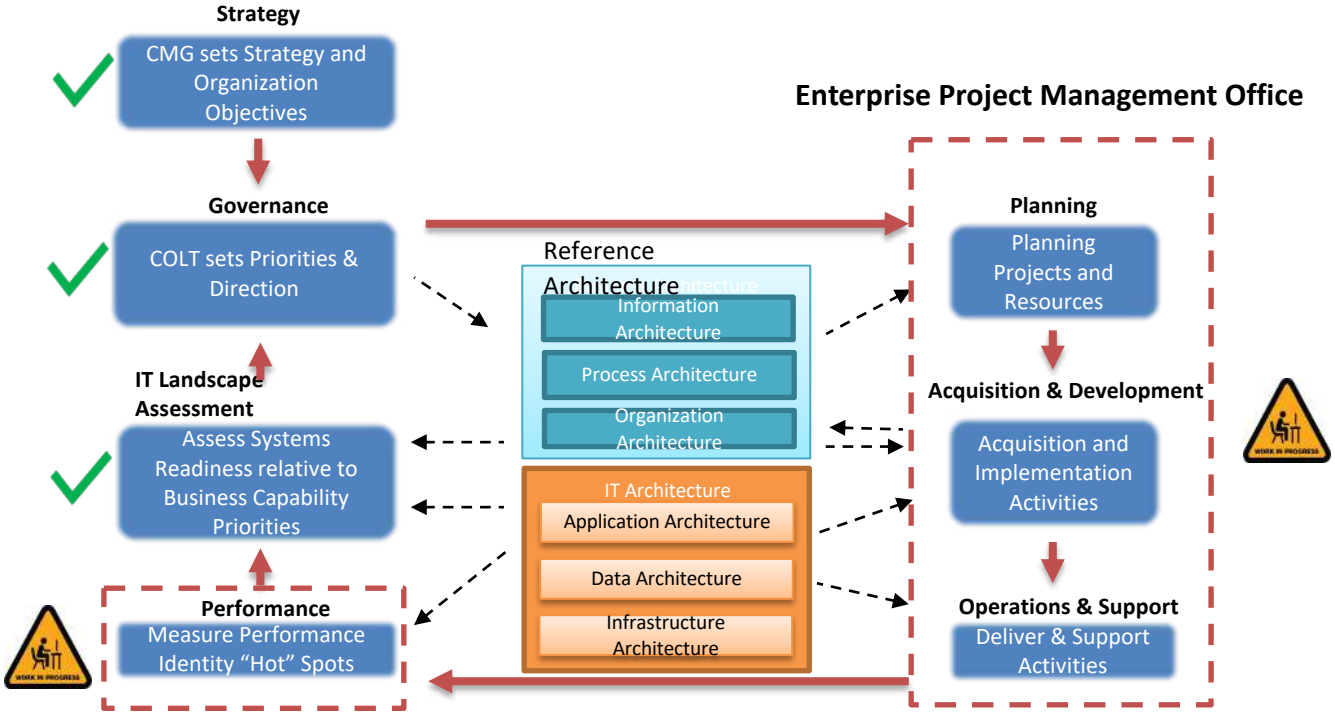
To ensure successful strategic alignment and execution of The Co-operators business strategies and priorities over the next 4 years.

Key Responsibilities

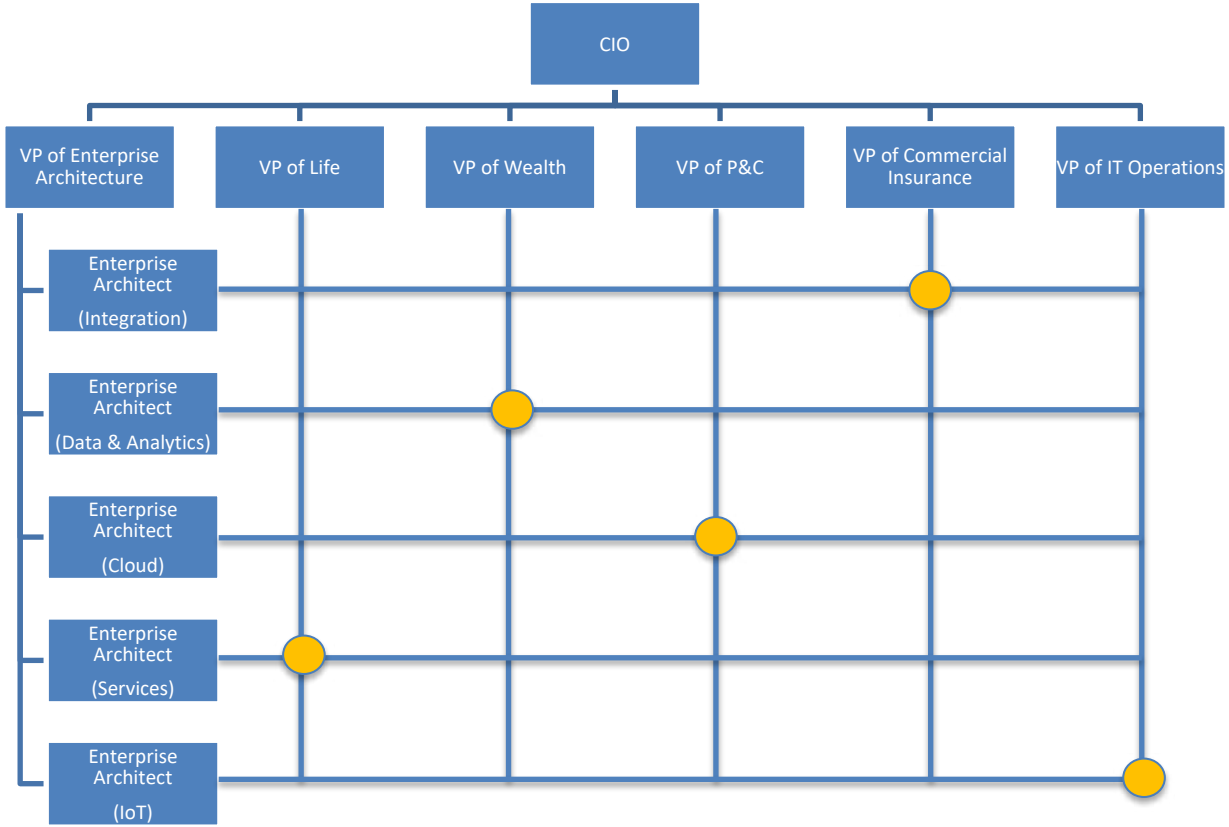
- Align, drive integration, and execute on our enterprise strategies
- Ensure clarity and alignment for operational priorities, timing and expected outcomes
- Surface and resolve emerging issues, concerns, and risks that impact the execution and success of our strategy
- Define and hold the organization accountable to ensure success metrics are achieved
- Share direction and outcomes that clearly articulate action & support required across the enterprise

COLT set priorities to make sure that we build the future

Enterprise Architecture helps with connecting strategies to everyday tasks and projects



A Lean Team of 9 Enterprise Architects are assigned to IT horizontals (data/system/tech/etc.) & business verticals (line of business)



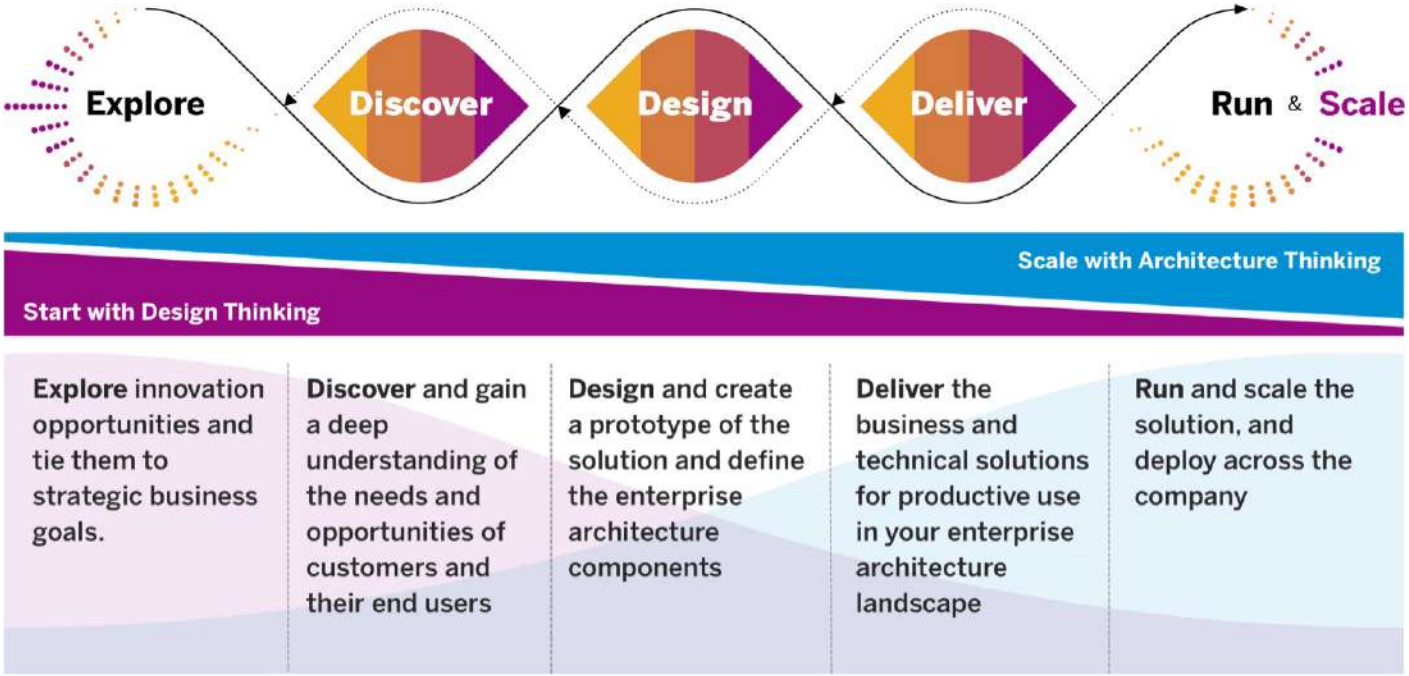
Enterprise Architecture will lead Digital Innovation

Enterprise Architecture Enables Digital Innovation

Gartner

By 2023, 60% of organizations will depend on EA's role to lead the business approach to digital innovation

Enterprise Architecture Driving Innovation

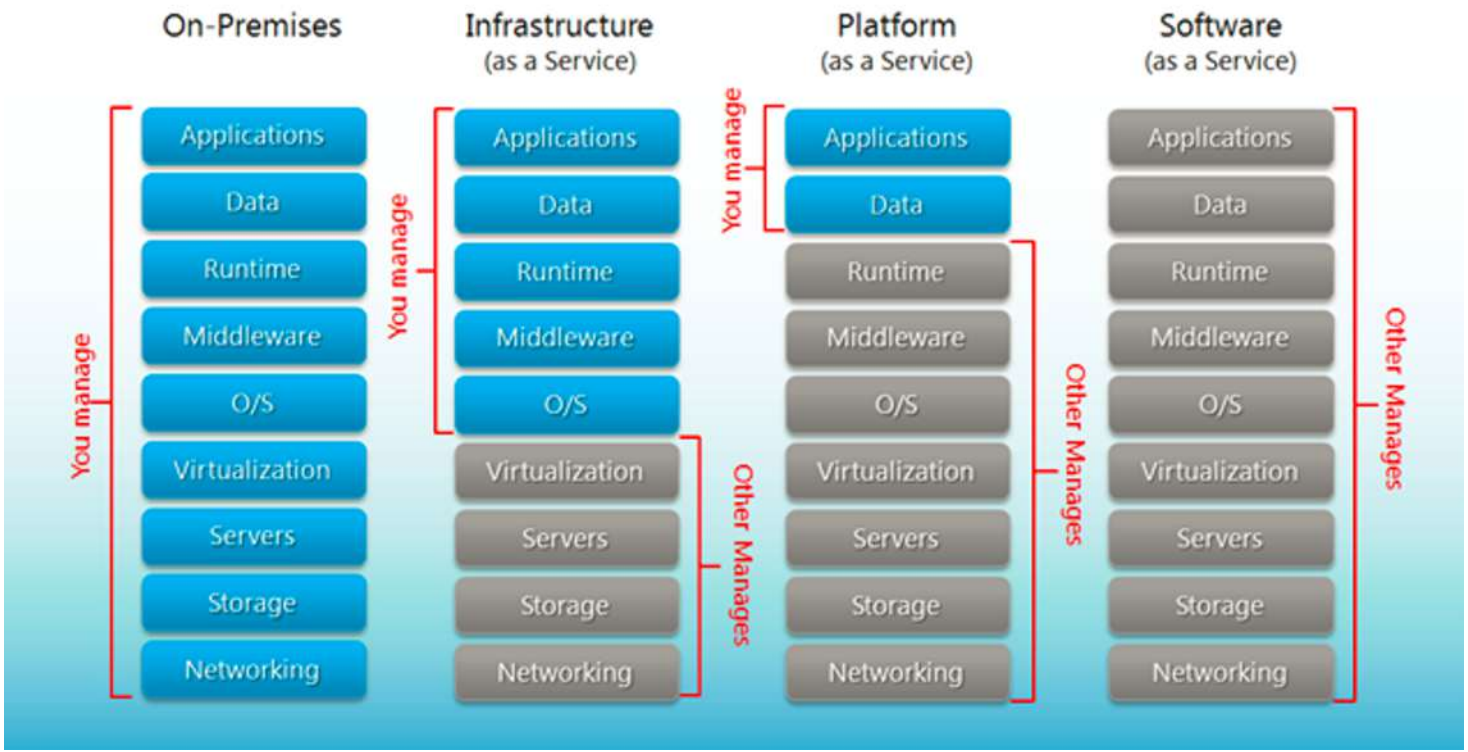


- Internet of Things
- On-demand Insurance
- Microservices
- Enterprise Integration Hub
- …

How do innovative companies leverage technology?

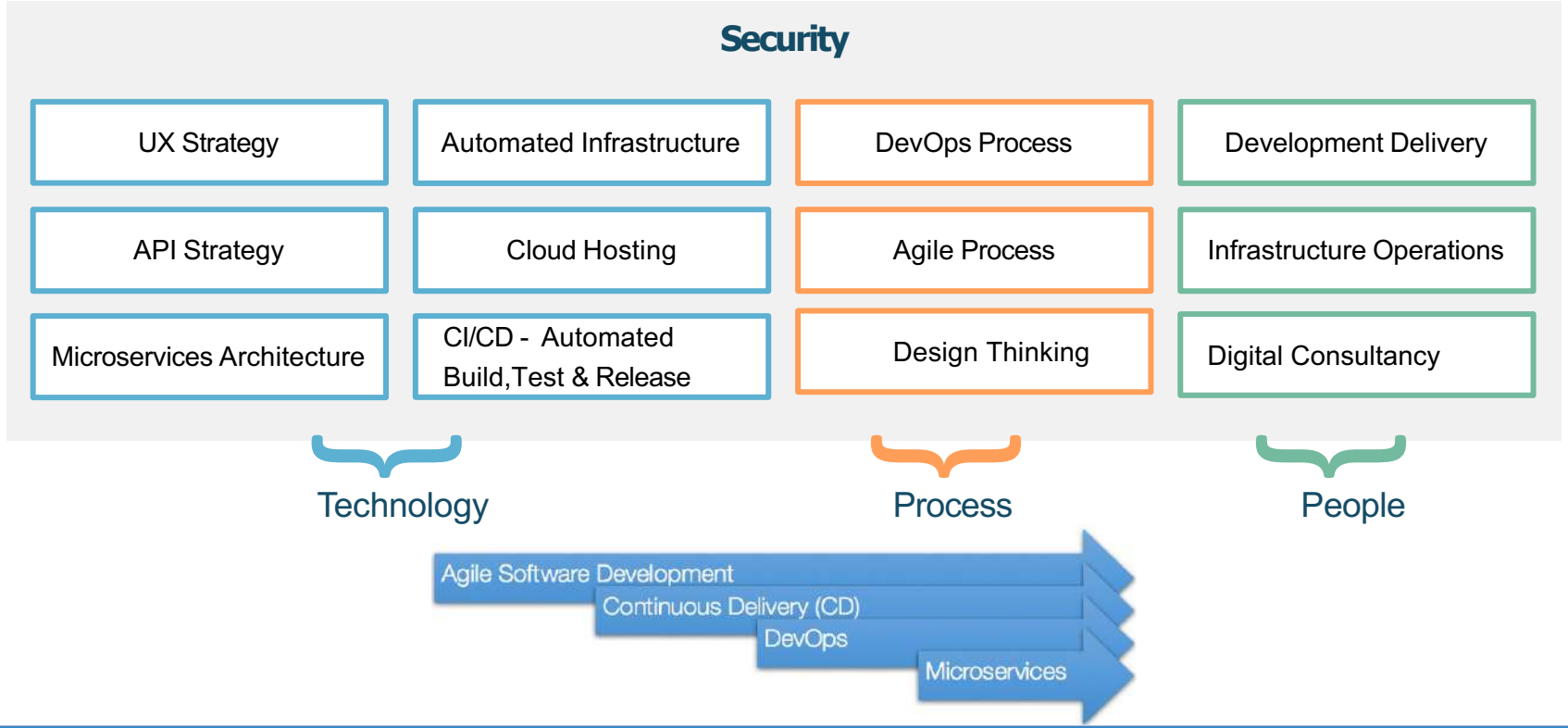
1. Technology is a competitive advantage
2. Treat technology as an investment
3. Use technology to mitigate risk
4. Technology is a core capability
5. Not afraid to change technology

Our Transformation Journey: Moving to Cloud



Our Transformation Journey

Change is all encompassing and far-reaching.

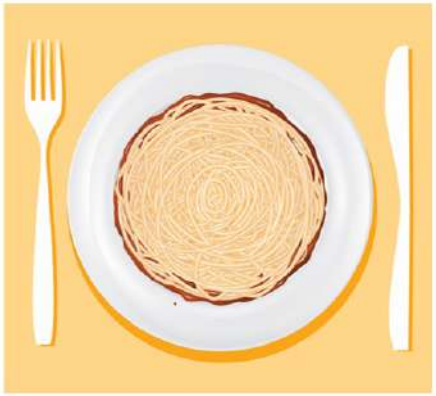


Microservices

"Innovation at the edges will never work if our core systems are locked up."

- Microservices – what are they and why should we care
- When to use and how do we accelerate adoption?
- Cooperators' microservices journey
- How about Microservices for Greenfield vs. Brownfield
- Modernization Strategy & Case Studies
- Discussion

Microservices Introduction



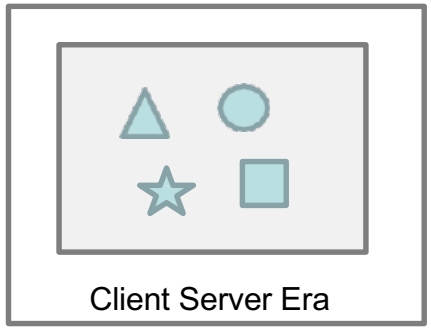
With monolithic, tightly coupled applications, all changes must be pushed at once, making continuous deployment impossible.

With a microservices architecture, developers create, maintain and improve new services independently, linking info through a shared data API.

Microservices architecture involves building applications as small, loosely coupled, reusable, autonomous components

- Monolithic applications are **one large application** that does everything
- Microservices are several **smaller reusable applications** that each does part of the whole
- Microservices are focused on reusable business capabilities, concentrating on **business APIs**
- Enables building and running **cloud native apps** that exploit the advantages of the **cloud computing delivery model.**

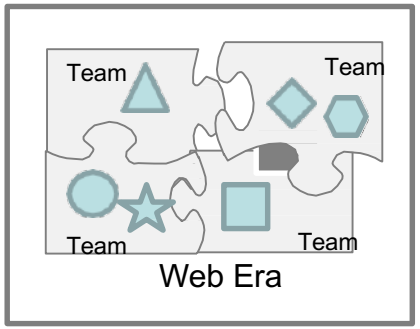
Monolithic



Client Server Era

- Tightly coupled
- Full dependency; each change has unanticipated effects requiring careful testing beforehand
- Large, single code base (e.g., capability)

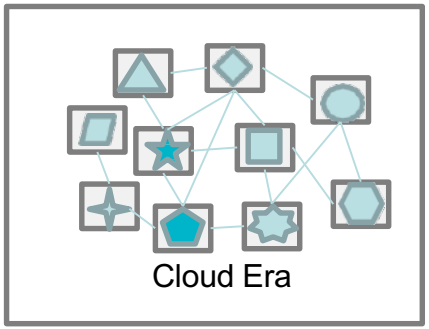
Traditional SOA



Web Era

- Loosely coupled
- Reduced dependencies; elements are developed more autonomously, but must be coordinated to fit overall design
- Greater modularity (e.g., domain)

Microservices



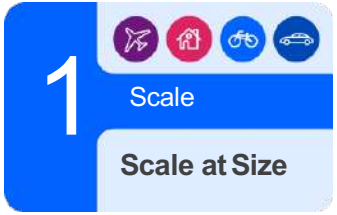
Cloud Era

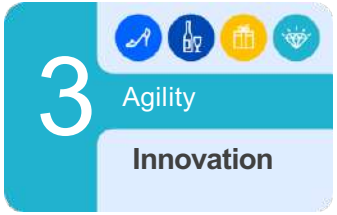
- Decoupled
- Zero dependency; new and/or modified functionality can by independently deployable
- Small components that perform a discrete functions (e.g., feature)

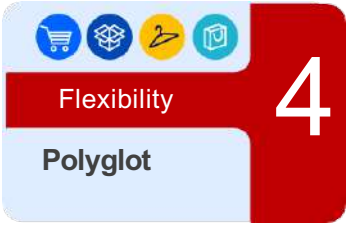
Benefits of the Microservices architectural style

Microservices provide value benefit through:

1. Selective granular scaling provides optimization across all layers of the stack
2. Componentization isolates risks, defects and outages, resulting in greater fault tolerance
3. Designed for fast & frequent change, reusability; unleashed polyglot programming







Lightweight SW code, focused on doing one function well

Functionally & operationally self-contained

Minimal interdependencies with other components

Every Microservice can be developed in a different programming language

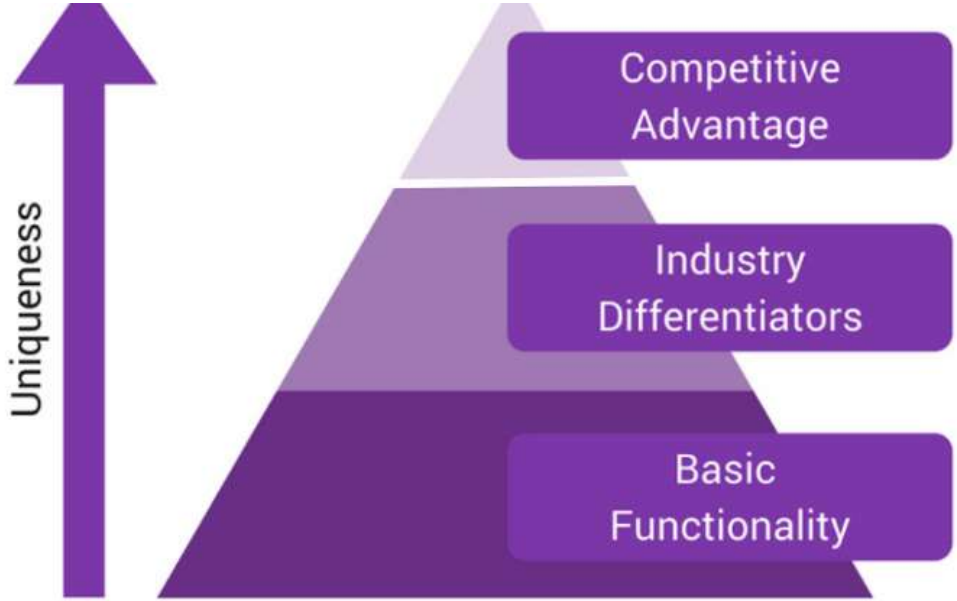
"Open" - can be run on bare OS, Container, Foundry etc

Deployed, Maintained, Managed, Scaled independently

Interoperability – REST API is *typical* interfacing mechanism

With thoughtful implementation of well-accepted design patterns, enables highly resilient systems

Why should I care if I mostly rely on software vendors?



Trying to achieve this section by configuring out of the box software is very expensive and difficult to achieve. Most of the risk in budget blowouts and project overruns originate with this activity.

OFFERED BY MOST VENDORS

Most out of the box vendors will offer basic functionality that any business can use and some will provide a little more customization towards the needs of a specific industry.

The merits of this software are only to take care of business activities that have no opportunity for competitive advantage.

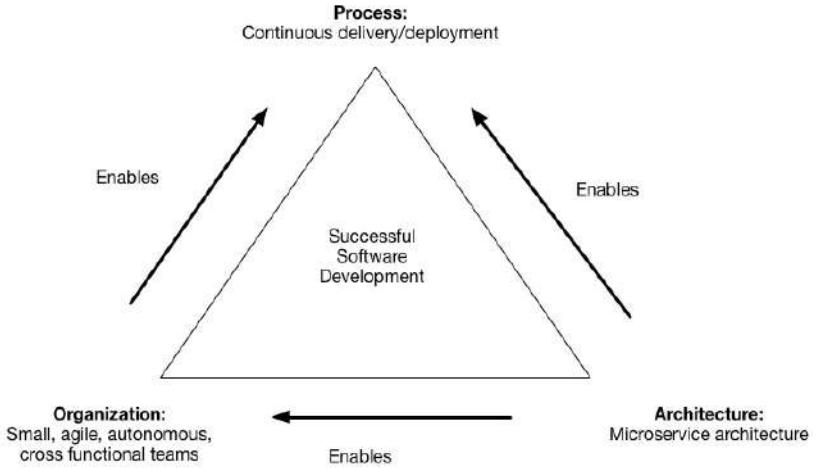
This is the software that most Enterprises will buy from "proven" Vendors. With the hope of becoming more competitive.

When to use Microservices

- Need to deliver **reusable business capabilities**
  - ✓ Improved ROI with reduced TCO
- Services need to be **highly available** and continue to be available through component failure
  - ✓ Increased resilience
  - ✓ Continuous delivery
- Services are likely to **have frequent changes** which need to be made safely with little or no outage
  - ✓ Easier debugging and maintenance
  - ✓ Faster time to market
- Services are likely to have to **scale** for significant peak loads or for future growth
  - ✓ Improved scalability

To accelerate microservices, we need:

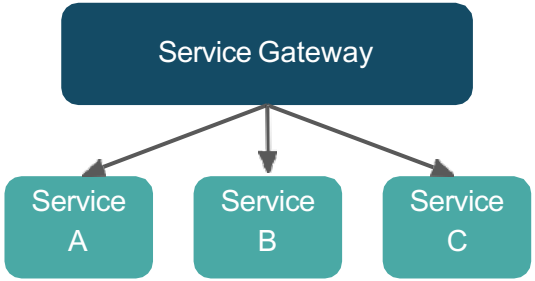
- Agile / MVP approach
- DevOps culture
- Design with failure in mind
- Robust monitoring
- CI/CD
- Rapid provisioning and app deployment



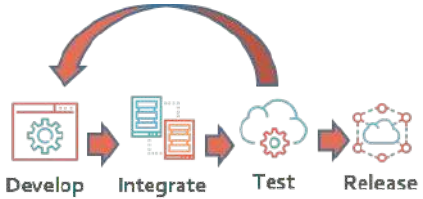
Impact across organization, not just technology

Organizational Design

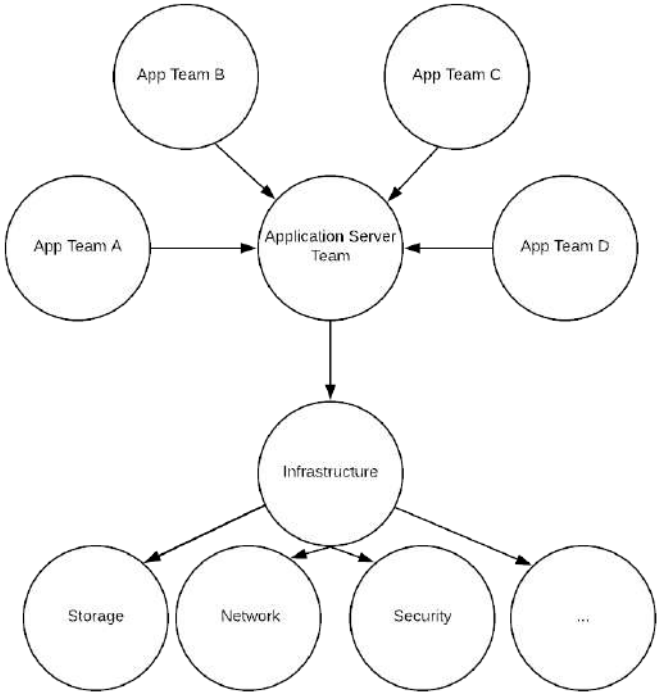
Technology Architecture

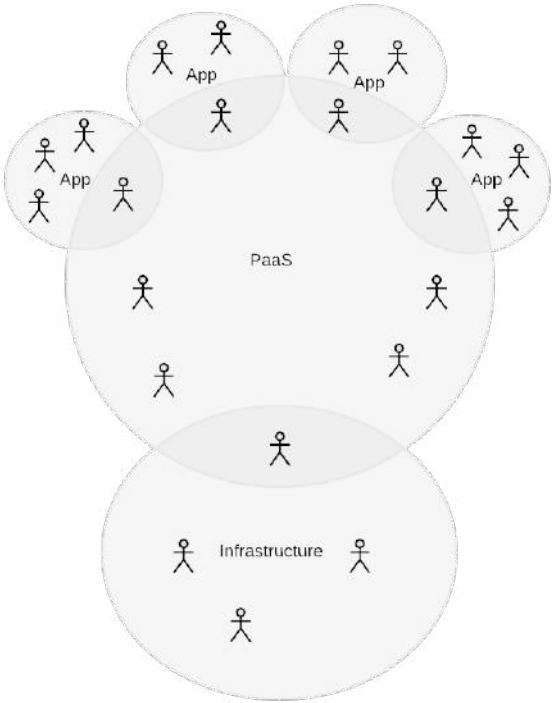


DevOps Processes/Capabilities

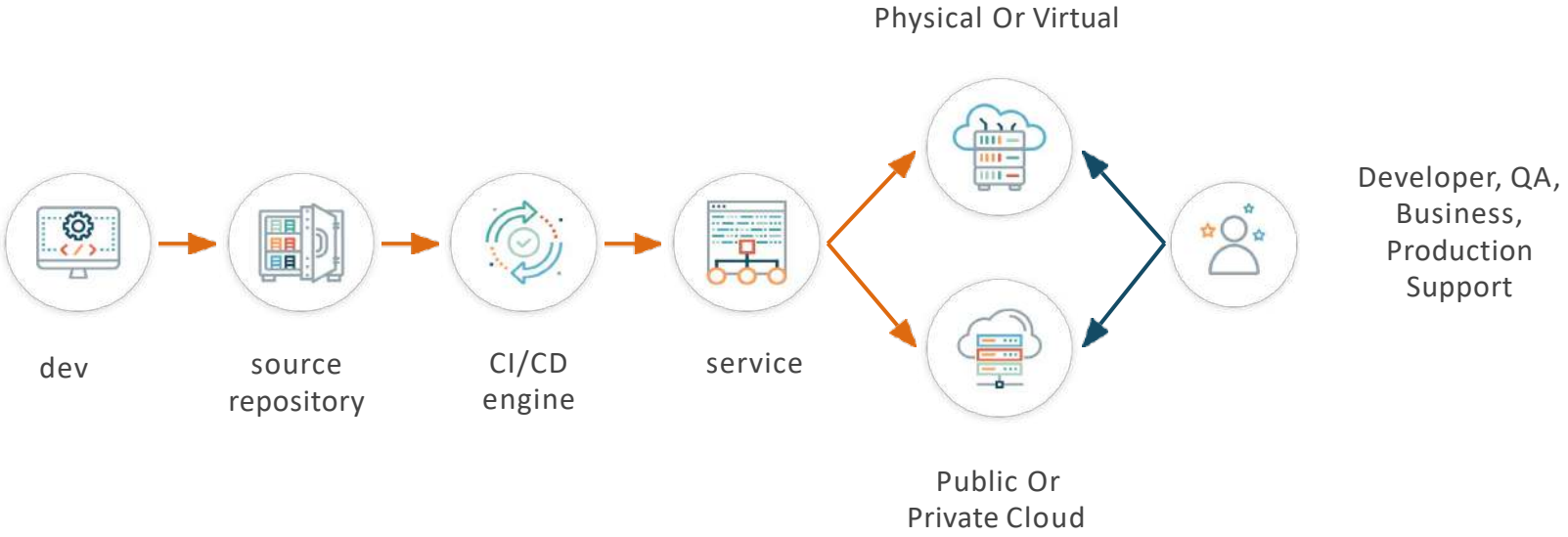


Traditional Organization VS DevOps Organization

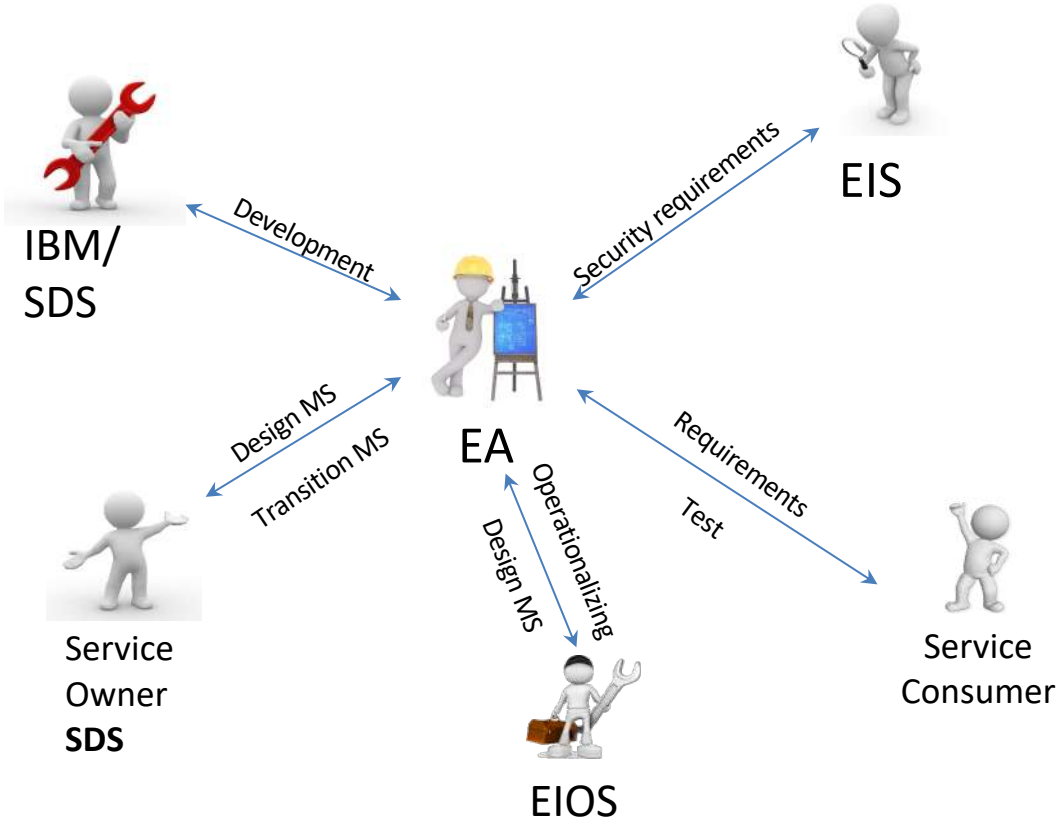




Build and Deployment with Microservices



EA leading the build of initial microservices



How about Microservices for Greenfield vs. Brownfield?

Microservices Considerations for Greenfield vs. Brownfield

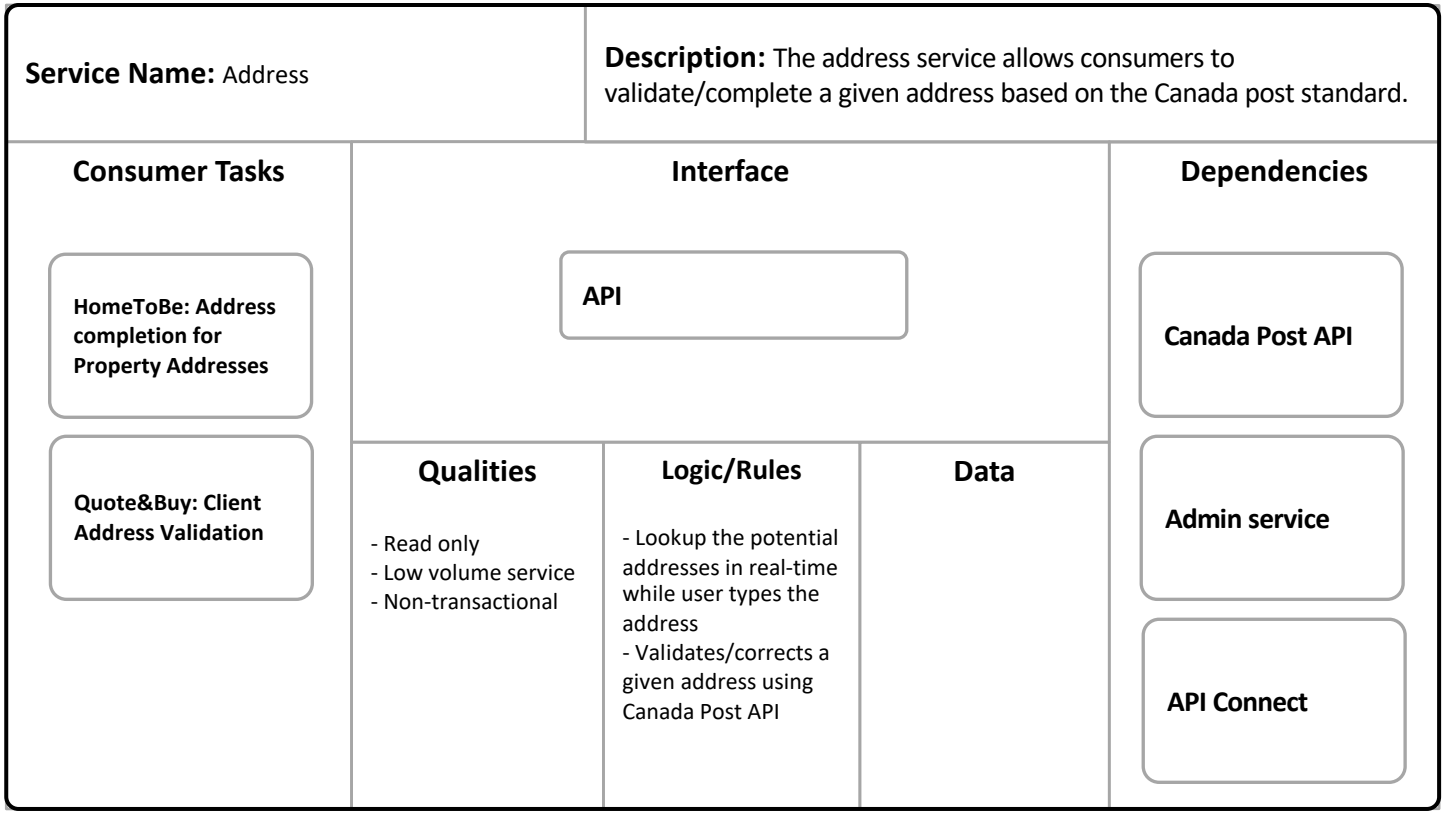
Greenfield (New Application)

- Clean slate in defining Microservice architecture
- Apply Domain Driven Design principles
- Commercial Credit Score
- HomeToBe / Duuo
- Address Validation & Completion
- Admin Service
- Advisor Assist
- … etc.

Brownfield (Existing Application)

- Strangle the Monolith over time with new microservices
- Identify reusable, loosely coupled, autonomous, independent, frequently changing parts of the monolithic and develop a microservice
- Consider data redundancy if microservice has a lot of data exchange with other parts of monolithic

Address Validation Microservice



Personal Credit Score Microservice

Service Name: Personal Credit Score

Description: The personal credit score service returns credit score of a given person from Trans Union API. It also stores the score in a local database for future use.

Consumer Tasks

Real-time Personal Credit Score Request for a single person

Batch Personal Credit Score Request for multiple persons

Interface

API

Query
- Get Personal Credit Score

Qualities

- Read only
- Low volume service
- Non-transactional

Logic/Rules

Searches the credit score of a given person(s) in the local database and returns the score. If the score dost not exist in the database or it is old, the services calls the TransUnion API and returns the score and stores it in the database.

Data

- Personal credit score local database

Dependencies

TransUnion API

Admin service

API Connect

Commercial Credit Score Microservice

Service Name: Commercial Credit Score

Description: Provides a front end to services provided by Dun&Bradstreet to retrieve commercial credit reports for businesses. Incorporates a real-time look up and an optional batch interface. Incorporates a caching mechanism to minimize the reports being retrieved from D&B

Consumer Tasks

- **Batch Commercial Credit Score Request for multiple companies**
- **Real-time Commercial Credit Score Request for a single company**
- **A Sovereign broker uses UM search for a client and retrieve their**
- **Policy Center retrieves a commercial credit score report**

Interface

- **API**
- **Query**
  - Search by company name
  - Retrieve by DUNS #

Qualities

- Read only
- Low volume service
- Non-transactional

Logic/Rules

Searches the credit score of a given company(s) in the local database and returns the score. If the score dost not exist in the database or it is old, the services calls the D&B API and returns the score and stores it in the database.

Data

- Commercial credit score local database

Dependencies

- **D&B Credit Score Service (Custom API developed by D&B for CGL)**
- **Admin service**
- **API Connect**

Admin Microservice

Service Name: Admin Service

Description: Admin service calculate charge back on usage and platform and also provides the email functionality for communication between service providers and consumers.

Consumer Tasks

Calculates charge back on service usage and the platform usage

Provide email service

Interface

Command

Query
- The service lookup the API usage from APIC periodically

Event Subscriptions
- The API is called by users
- User subscribes to the API

Event Publications

Qualities

- Low volume service
- Transactional

Logic/Rules

- When the user subscribes to the APIs, the service owner approves the user and user is authorized for using API. The user receives the invoice for using API and platform periodically.

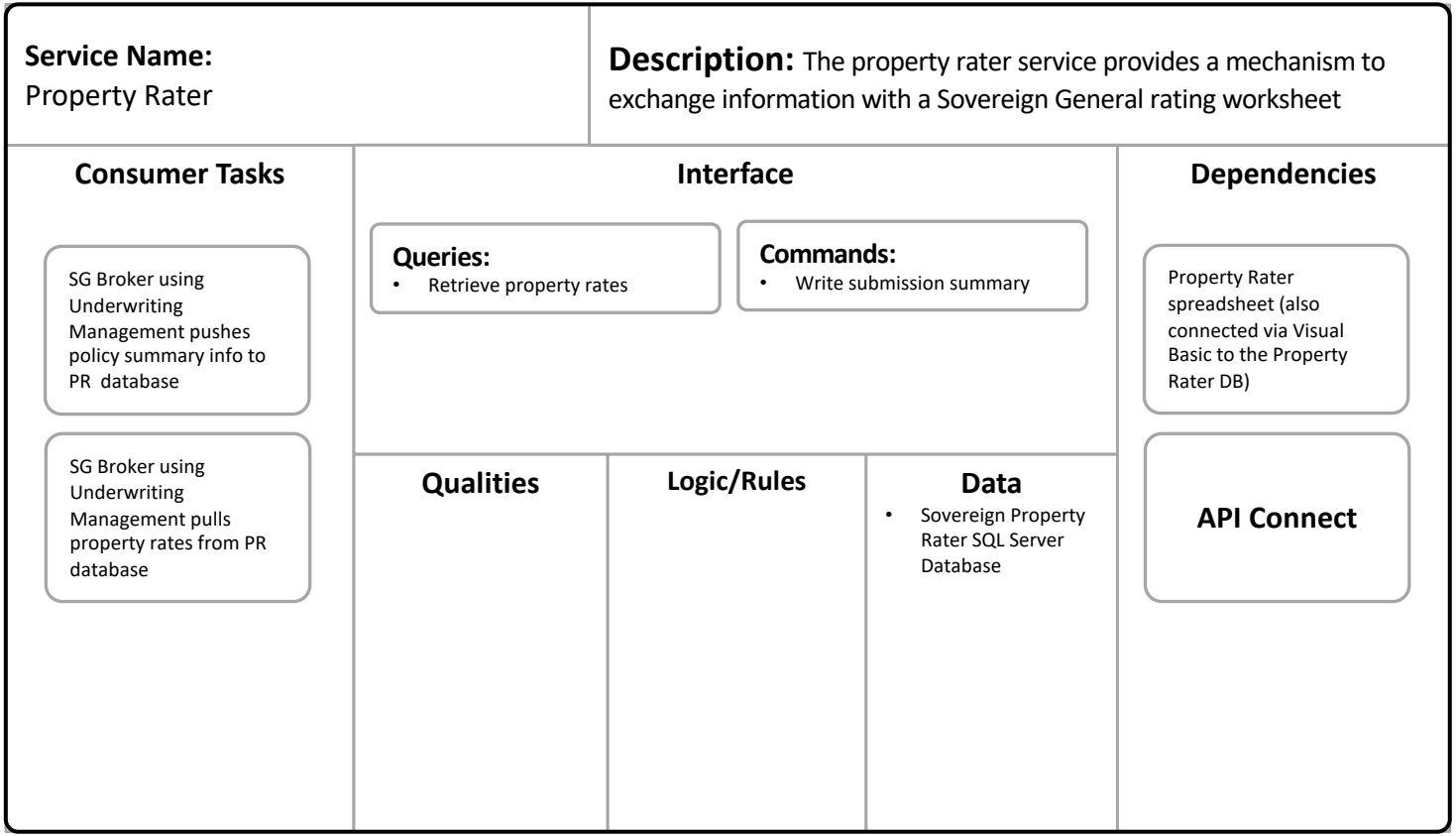
Data

- API usage
- Authorized users

Dependencies

API Connect

Property Rater Microservice



OnBase Microservice

Service Name: OnBase

Description: This service searches, stores and retrieves documents from Co-operators OnBase services

Consumer Tasks

A Sovereign broker using UM to attach or retrieve a document associated with a submission

Interface

Query:
- Search by meta data
- Retrieve by key
- Retrieve data by key

Commands:
- Store document data

Qualities

Logic/Rules

Data

Dependencies

OnBase Implementation is internal “DotNet” application and database

API Connect

Decision Maker Microservice

Service Name: SG Global Decision Maker

Description: Uses a customized algorithm to create a simplified indicator on whether to quote a submission based on information retrieved from commercial credit score (Red – Do not Quote, Yellow – Underwriter assessment required, Green – Proceed to Quote)

Consumer Tasks

A Sovereign broker initiates a GDM request to determine if they should proceed to provide a quote

Interface

Query:
Get GDM (by Submission#)

Qualities

- Read Only

Logic/Rules

Data

Dependencies

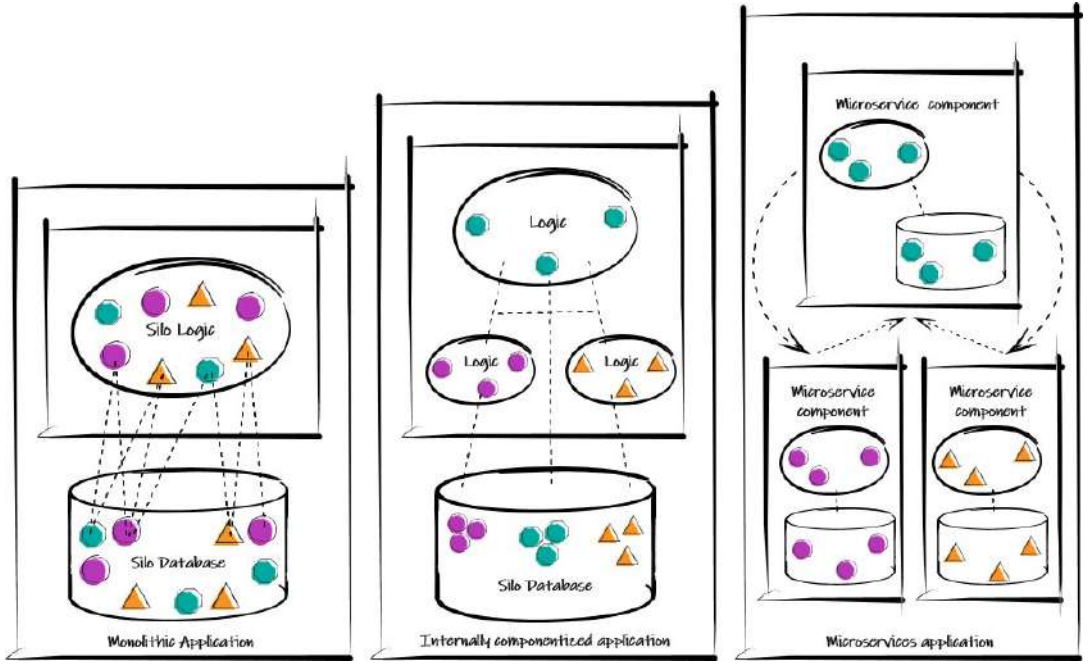
Uses: Commercial Credit Score service

API Connect

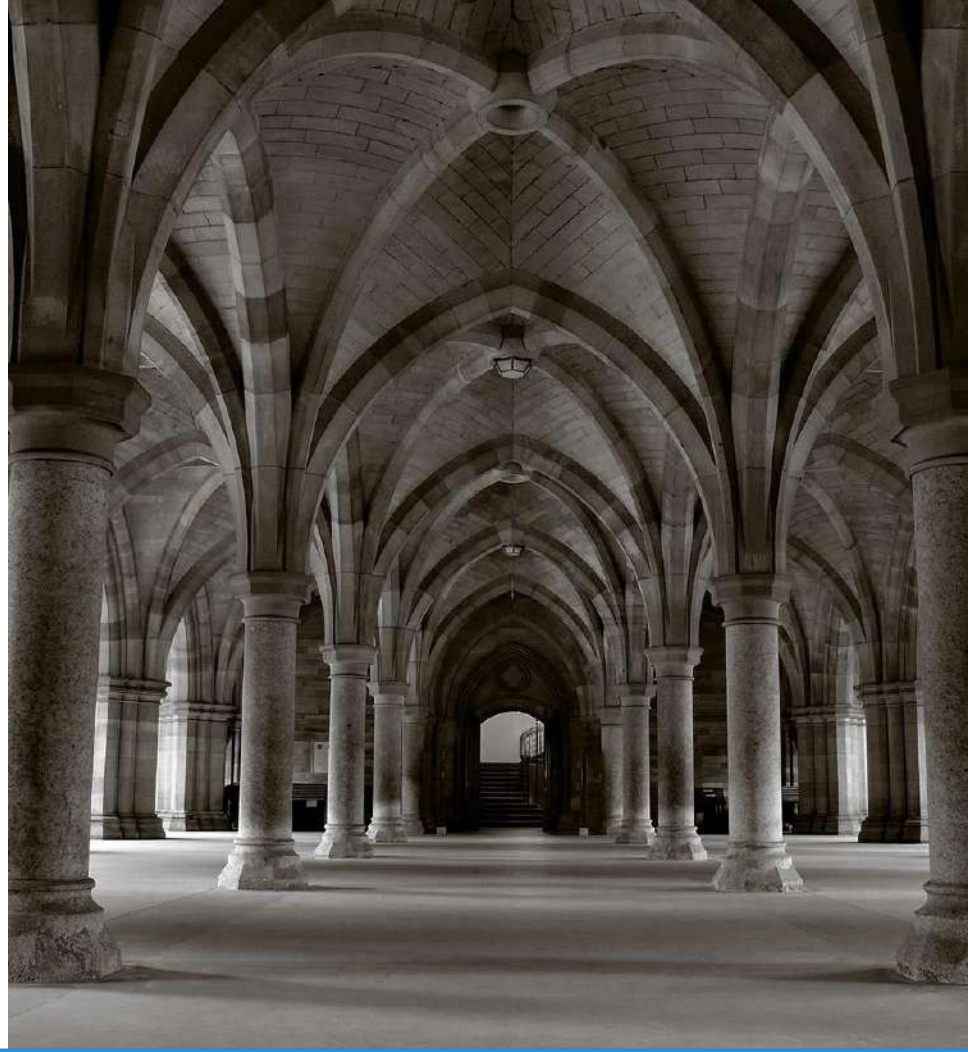
How do we decompose existing monoliths, how do we deal with data, etc.

How do we decompose an existing monolithic

1. Separate databases before separating services.
2. Place existing transactions within a single service or redesign the transaction usage (compensating/eventual consistency).
3. Consider the team composition.
4. Implement new features as microservices around the existing monolith.



Tenets of the Microservices Architectural Style



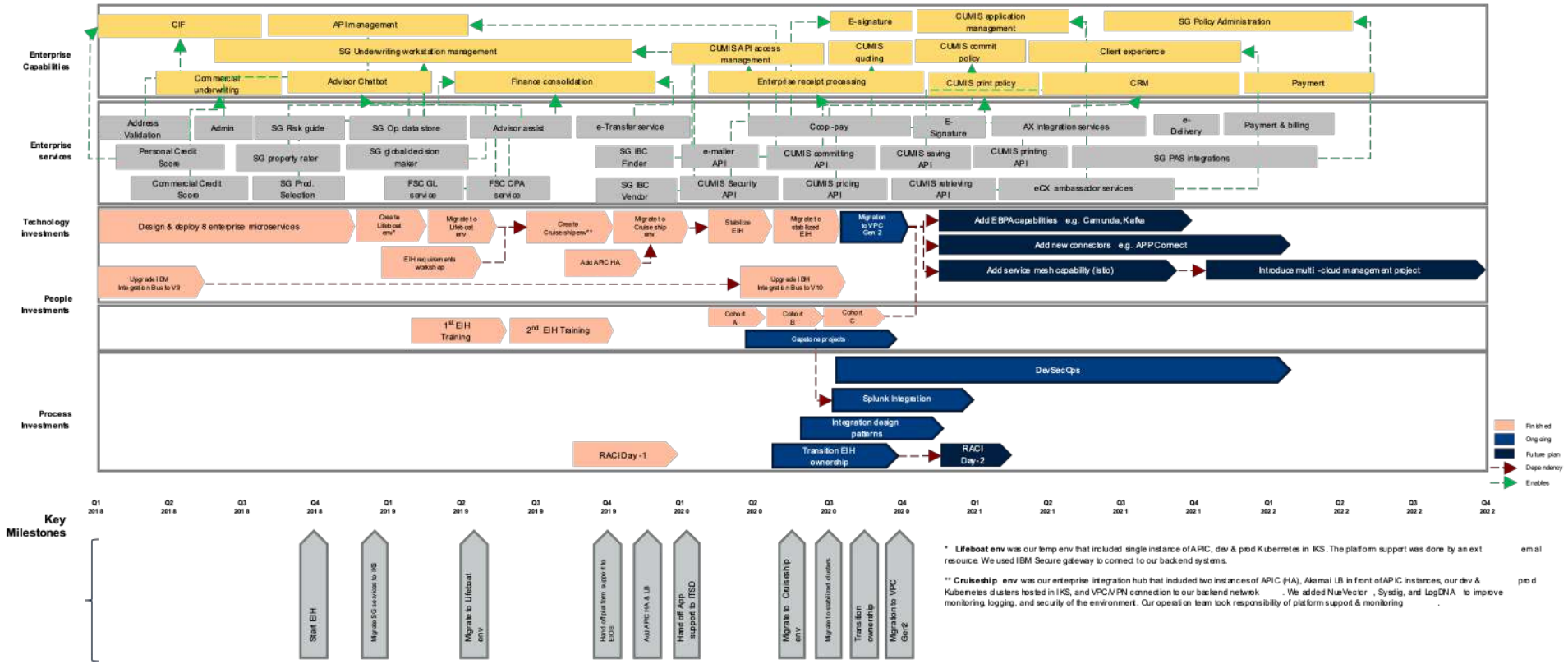
Tenets of the Microservices Architectural style

Large monoliths are broken down into many small services	Services are optimized for a single function	Communication via REST API and message brokers	Per-service continuous integration and continuous deployment (CI/CD)	Per-service high availability (HA) and clustering decisions
Each service runs in its own process One service per container	There is only one business function per service The Single Responsibility Principle (A microservice should have one, and only one, reason to change)	Avoid tight coupling introduced by communication through a database	Services evolve at different rates You let the system evolve but set architectural principles to guide that evolution	One size or scaling policy is not appropriate for all Not all services need to scale; others require autoscaling up to large numbers

When not do Microservice

- To keep a system simple - resist dividing what is cohesive!
- Generally, smaller systems are easier to build and maintain than large ones. But all is trade offs, and given one system to design:
  - reducing the size of its components
  - increases the frequency and complexity of messaging between components.
- Microservices Require Cultural Changes (DevOps)

Our Microservices Journey So Far...



* **Lifeboat env** was our temp env that included single instance of APIC, dev & prod Kubernetes in IKS. The platform support was done by an ext em al resource. We used IBM Secure gateway to connect to our backend systems.

** **Cruiseship env** was our enterprise integration hub that included two instances of APIC (HA), Akamai LB in front of APIC instances, our dev & prod Kubernetes clusters hosted in IKS, and VPC/VPN connection to our backend network . We added NueVector , Sysdig, and LogDNA to improve monitoring, logging, and security of the environment. Our operation team took responsibility of platform support & monitoring .

Enterprise Integration Hub 2019

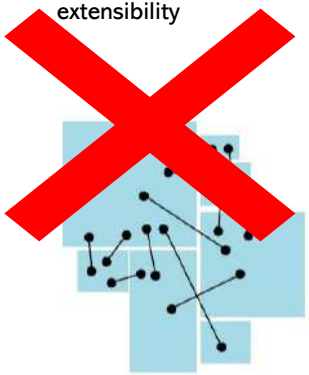
- In 2019, microservices were running in IBM Cloud supported by IBM developers (life boat)
- We needed a cloud environment managed/supported by our operation and application development teams
- We were tired of point-to-point/LoB integrations and needed to move to API-based integration
- Enterprise Integration Hub was born in 2019 to address above needs.

Enterprise Integration Hub 2019

RoadMap to Increasing Integration Agility

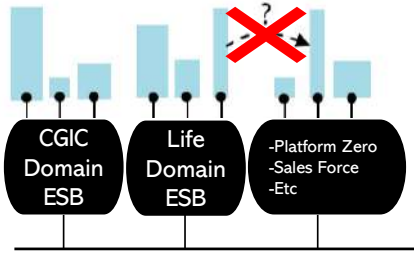
Monolithic Point-to-Point

- Direct Integration
- Point-to-Point patterns
- Limited reusability and extensibility



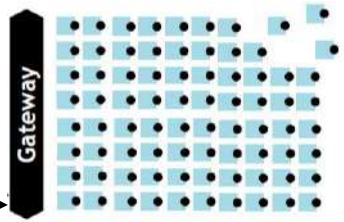
Line of Business Service Bus

- Federating LOB's agility
- Provide LOB's autonomy to continue to leverage heritage technologies and core expertise



Enterprise Integration Hub

- Reusable functionality in small, independent, scalable containers
- Standard, platform independent gateway to connect to legacy



This is our direction in ALL cases.

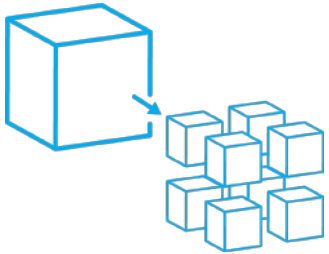
Enterprise Integration Hub Scope 2019

System Integration

A Platform to Integrate different systems

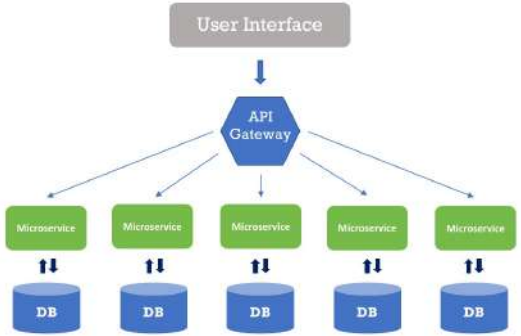
Microservices

A Platform to Develop and Deploy Microservices

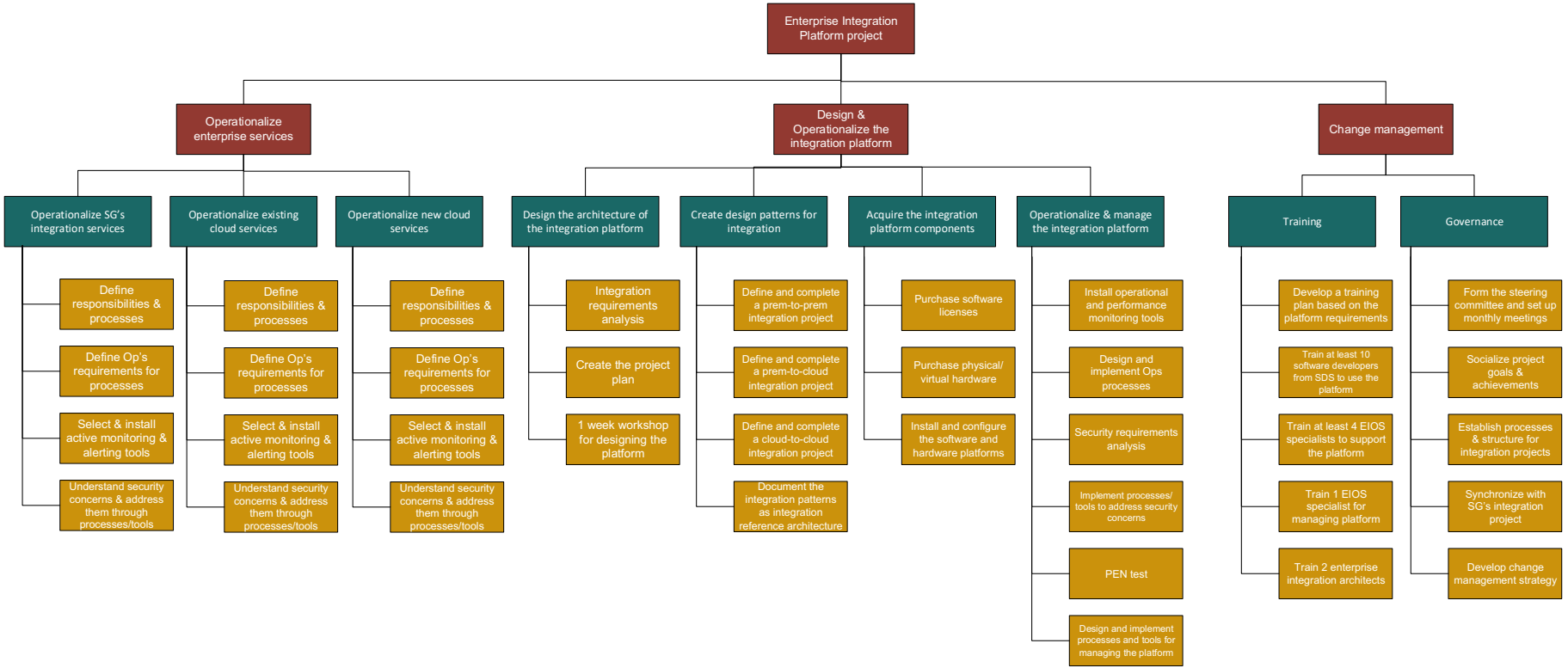


Reusable Services

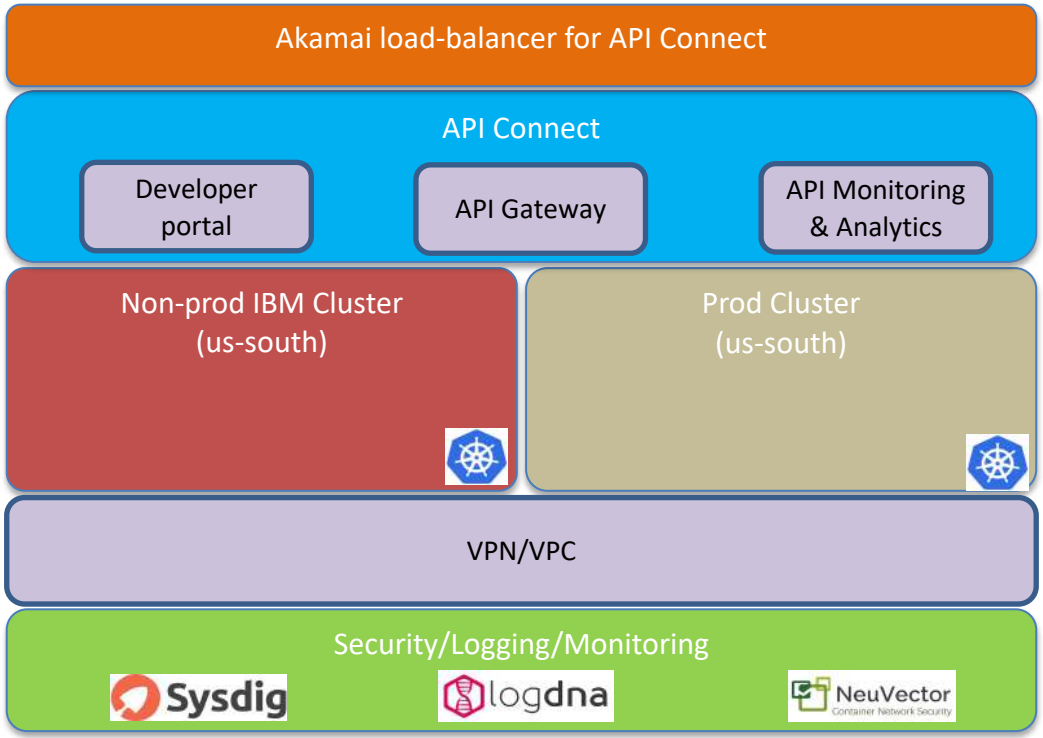
A Platform to Facilitate Reusing Microservices (Security, Monetization, etc.)



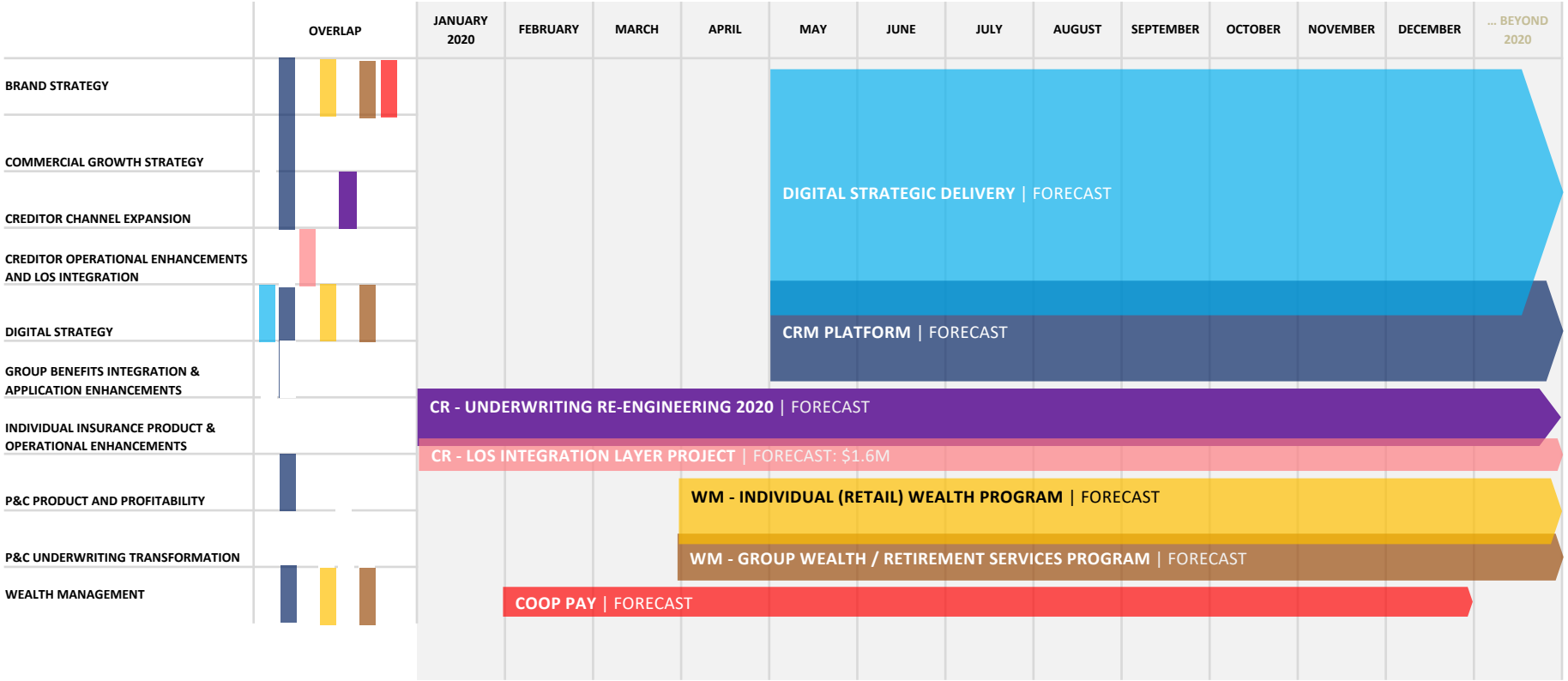
EIH Project Scope (WBS)



Enterprise Integration Platform (EIH) Component Diagram



Several strategic projects relying on EIH - 2020



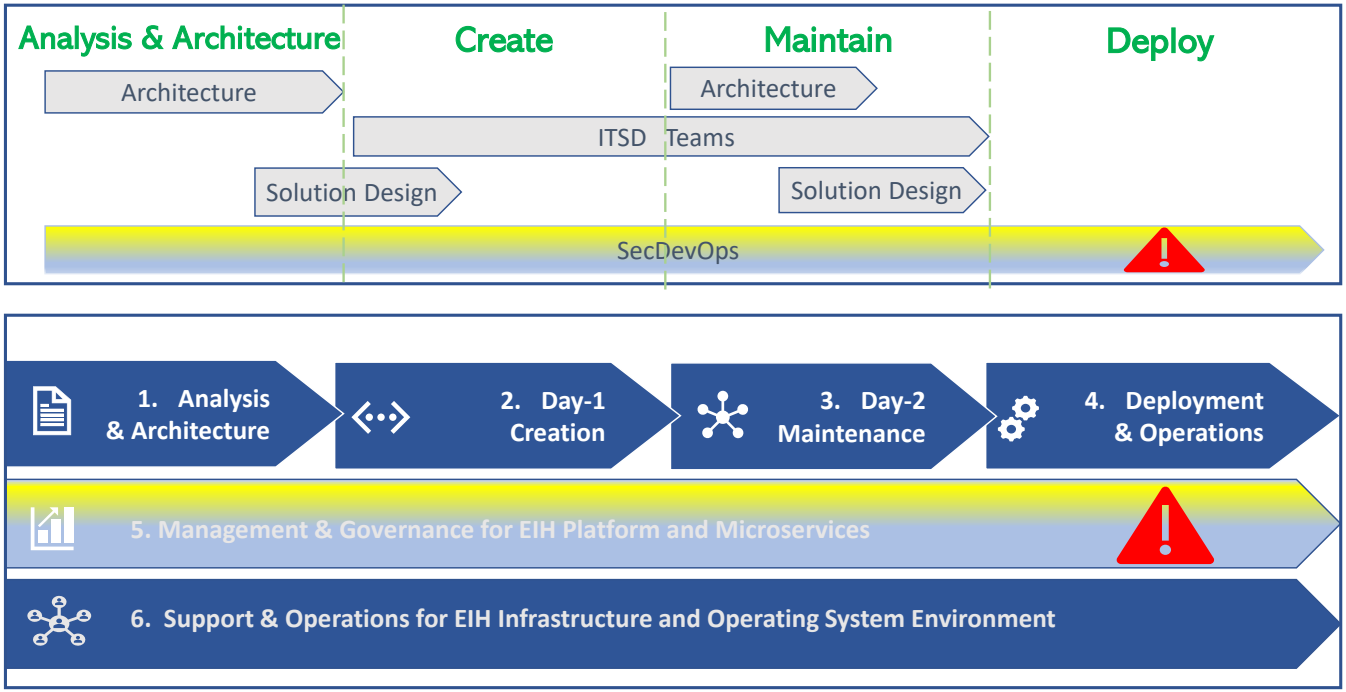
Forecast Cost

Less — More

Key Insights:

- Should deal with risks by Q3 since most overlaps happen in Q3 & Q4
- Big overlap between eCX and CRM for integration services requires synergy between integration teams
- Most overlaps for Brand strategy, Wealth management, & Digital Strategy

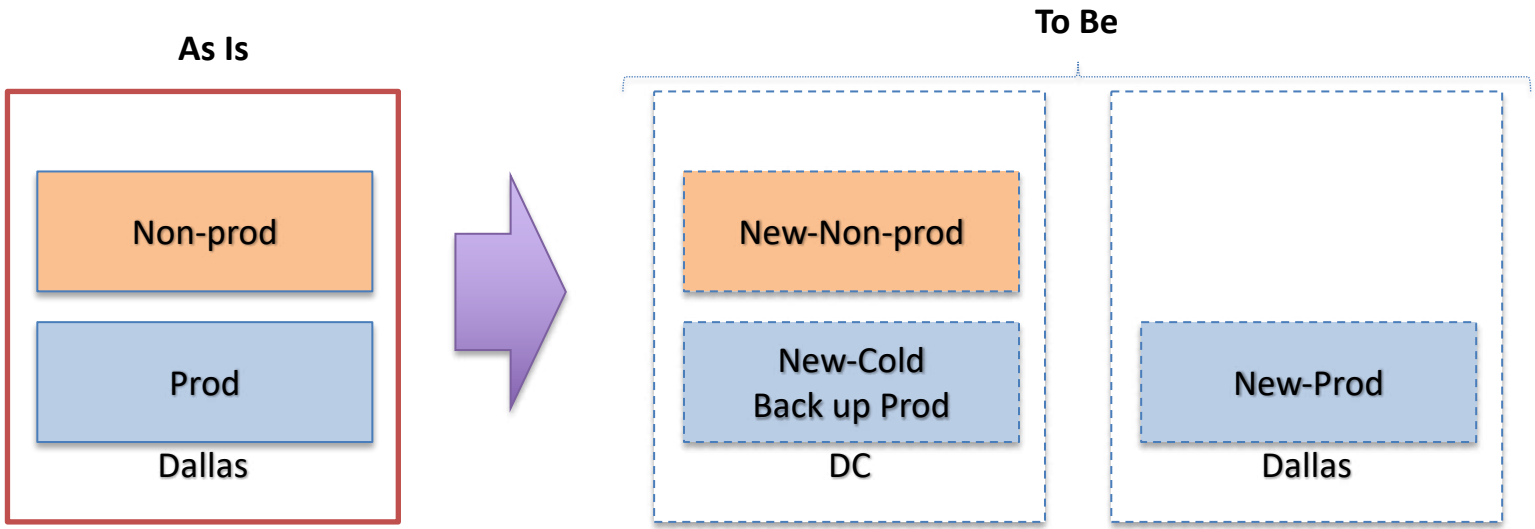
Our processes & capabilities were not fully developed



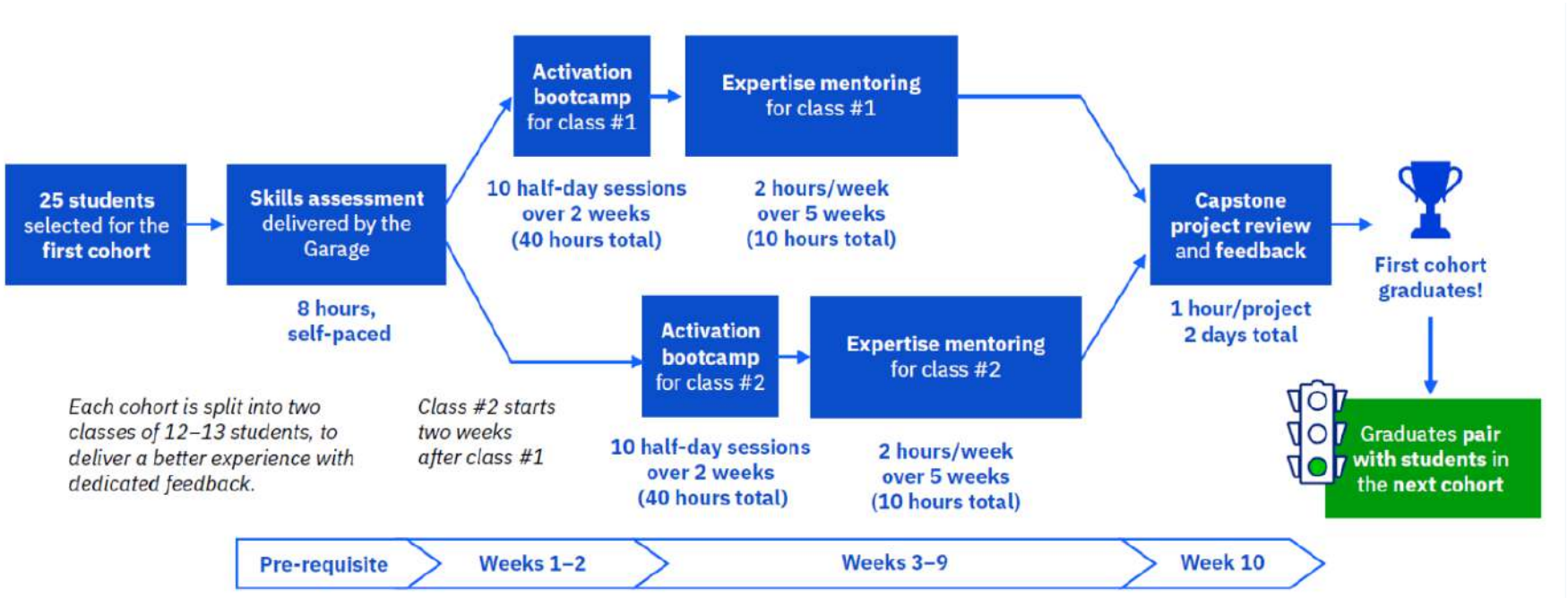
EIH went through couple of migrations to get stable

- We moved from Lifeboat to Cruise ship
- We moved from Cruise ship to stabilized EIH
- We moved from stabilized EIH to EIH-VPC Gen2

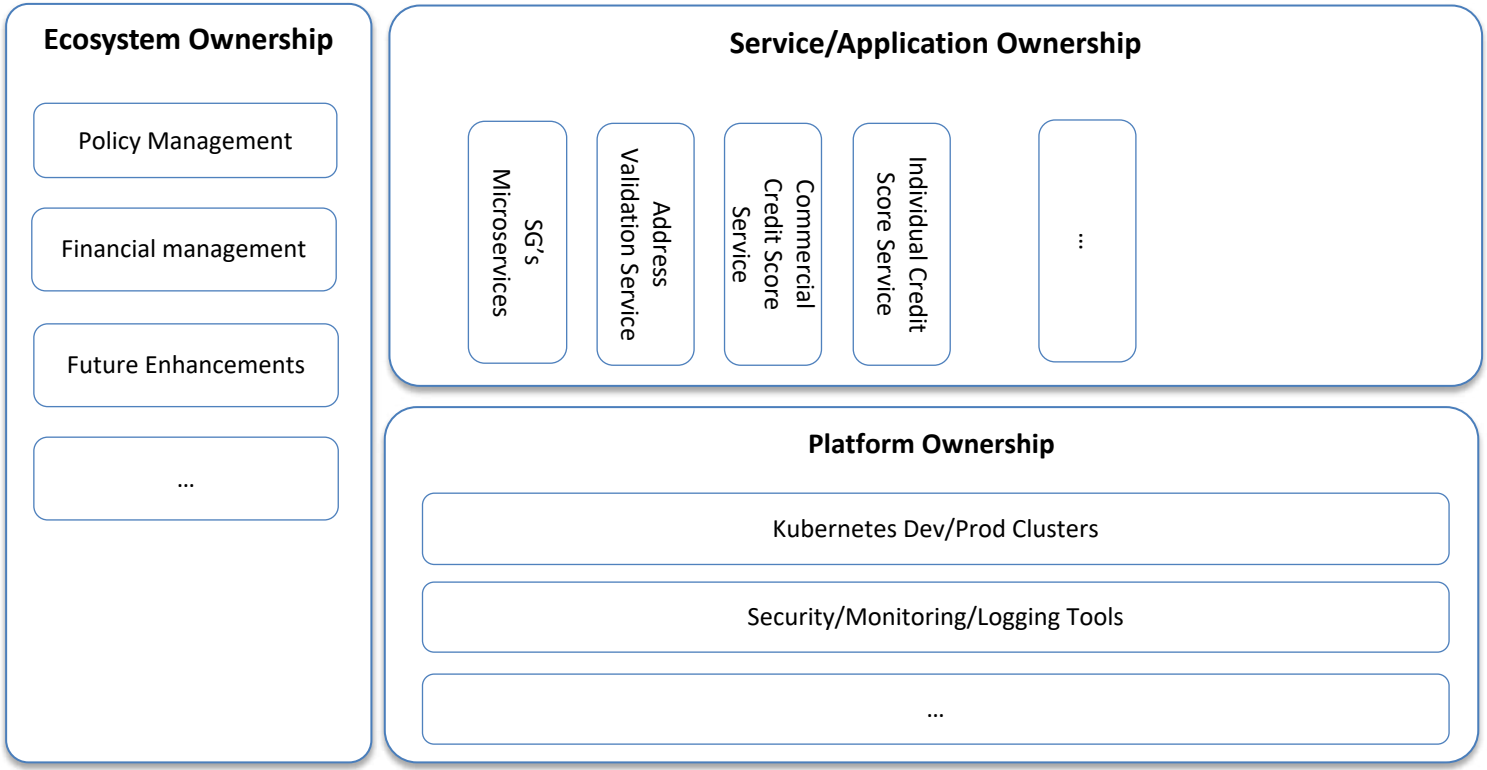
Example moving from Cruise ship to stabilized EIH



100 developers got trained in EIH Activation & Enablement Plan to mitigate the skill risk



We defined Service Ownership Model to address the maturity risk



Initial EIH Service owners (Day 1)

Service or Component	Day 1 Owner
Kubernetes Environment	Midrange
APIC	EA
APIC - akamai LB	EA/WH
VPC	Midrange
NeuVector	SecOps (Dan)
Feature Service *	ITSD
Image Build	ITSD DevOps
Image Deploy	ITSD DevOps
Image Repo (to Image Repo)	Midrange
IBM Code Control and Build Repository	EA
Other Code and Build Repos	ITSD (Various)
Release management (Repo to Non-prod)	ITSD DevOps/New 'DevOps' (Gap)
Release management (Non-prod to Prod)	ITSD DevOps/New 'DevOps' (Gap)
IBM Continuous Integration Continuous Delivery Pipeline (ToolChain)	New 'DevOps' (Gap)
operational resiliency (namespace backups- storage)	Midrange
operational resiliency (namespace backups- restore)	Midrange
Monitoring Service Provisioning	Midrange
Logging Service Provisioning	Midrange
Audit Service Provisioning	Midrange

EA Defined Guiding Principles For Enterprise Services

How might we determine which enterprise services should be containerized/wrapped/exposed through APIC?

MINIMIZE CHANGE

External services that are used across the enterprise should be wrapped to protect consuming applications from the impact of changes to the service provided
External services that require functionality extension beyond the offer should be wrapped as an enterprise service

INCREASE REUSABILITY

Enterprise services should be exposed through APIC if other internal/external developers need to reuse them from external assets (APIC developer portal capability)

REDUCE COMPLEXITY

Containerizing enterprise services reduces the complexity of technology stack to a single plain

MANAGE SERVICE USAGE

All enterprise services should be exposed through APIC when there is need for segregated monitoring, monetization, traffic, and/or provide different service levels

INCREASE SCALABILITY/PORTABILITY

Enterprise services should be containerized specifically when scalability and portability are necessary

IMPROVE SECURITY

Enterprise services exposed to external consumers/networks/clouds or services that require a common security layer should be exposed through APIC because we can control secure access to the services using API key and other security mechanisms in APIC

Use Guiding Principles,

- To decide if an external service needs to be wrapped as an enterprise service
- To decide whether an enterprise service needs to be containerized or not
- To decide whether an enterprise service needs to be exposed through APIC or not

Use Case

How might we determine if the following services should be wrapped/containerized/exposed through APIC?

Adapter/Wrapper Pattern

Problem: An "off the shelf" API offers compelling functionality that we would like to reuse, but its "view of the world" is not compatible with the requirements/architecture of our systems.

Solution: Adapter/wrapper is about creating an intermediary abstraction that translates, or maps, the old component to the new system. Clients call methods on the Adapter object which redirects them into calls to the legacy component. Wrapping a service saves the consumers from future changes (vendor change) and adjusts the services to our needs.

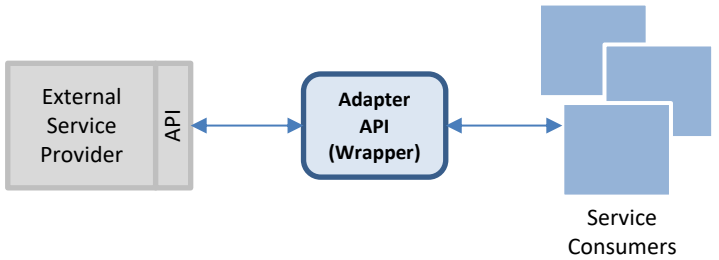
Pattern: When adapter/wrapper pattern is useful

- When multiple application are using the off-the-shelf service (e.g. address validation)
- When we are using only a subset of functionalities of the service (e.g. CCS)
- When there is need to extend the functionality of the service (e.g. e-signature)
- When there is a possibility to switch to a new vendor (e.g. address validation)
- When aggregating API calls provides greater usage visibility and opportunity for volume discounting (e.g. CCS)

Anti pattern: When adapter/wrapper pattern might not be suitable

- When wrapping a service introduces new security risks (e.g. Okta service)

Architecture:



Examples:

- **Address validation service**: this API extends the functionality of Canada post address validation API
- **Personal credit scoring**: this API encapsulates the TransUnion credit score API and decreases total cost of API calls by storing credit scores in a database
- **Commercial credit scoring**: this API encapsulates the D&B commercial credit score API and decreases total cost of API calls by aggregating API calls
- **E-Signature:** this API extends the One Span e-signature service by integrating e-sign service with OnBase which is our document storage system

Ambassador Pattern

Problem: A microservice may need access to shared components that perform common tasks, such as monitoring, logging & auditing. It is not possible to redundantly copy them into the microservice environment because they need to be independently maintained. At the same time, it may be inefficient for the microservices to remotely interact with them.

Solution: A special ambassador container is created to host virtualized copies of the utility components. The ambassador can be developed by security/monitoring specialists and be used as a proxy in multiple services. This saves a lot of effort from service developers since they don't deal with security/monitoring issues.

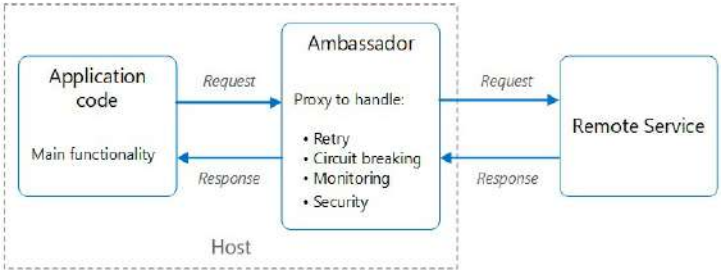
Pattern: When ambassador pattern is useful

- When we need to build a common set of client connectivity features for multiple languages or frameworks.
- When we need to offload client connectivity concerns to infrastructure developers or other specialized teams.
- When we need to support cloud or cluster connectivity requirements in a legacy application or an application that is difficult to modify.

Anti pattern: When ambassador pattern might not be suitable

- When the network request latency is critical for our service.
- When client connectivity features are consumed by a single language.
- When it is impossible/complex to generalize connectivity features.

Architecture:



Examples:

In eCX project, the ambassador pattern was used to provide a consistent way to offload:

- mTLS validation for API Connect and the K8 services
- Client authorization to perform service operations
- Circuit breaking rules

API Gateway Pattern

Problem: In a microservices architecture, the client apps usually need to consume functionality from more than one microservice. If that consumption is performed directly, the client needs to handle multiple calls to microservice endpoints. When the application evolves and new microservices are introduced or existing microservices are updated, handling so many endpoints from the client apps can be a very difficult.

Solution: An API gateway is a service that is the single-entry point for API requests into an application from outside the firewall. API gateway encapsulates the application's internal architecture and provides an API to its clients. It also has other responsibilities such authentication, monitoring, and rate limiting.

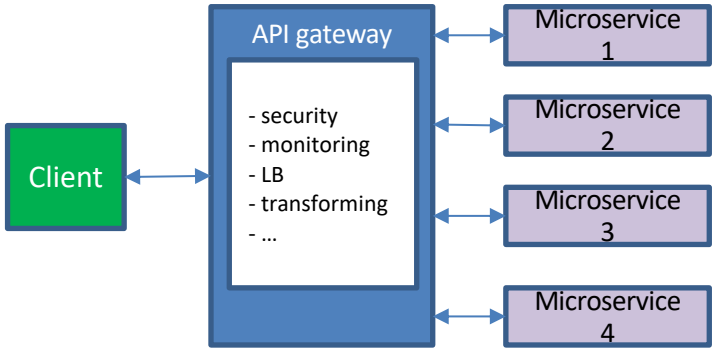
Pattern: When API gateway pattern is useful

- When our enterprise service is used by more than one user group/application
- When we anticipate to change/switch our backend service in the future
- When we need to monitor/segregate/limit traffic to our service for each user
- When we have external service consumers
- When our service is exposed to the internet/other clouds/external networks
- When our service requires a common security layer

Anti pattern: When API gateway pattern might not be suitable

- When all service users are internal and there is no need for monitoring/monetization/a common security layer (e.g. Coop-Pay)

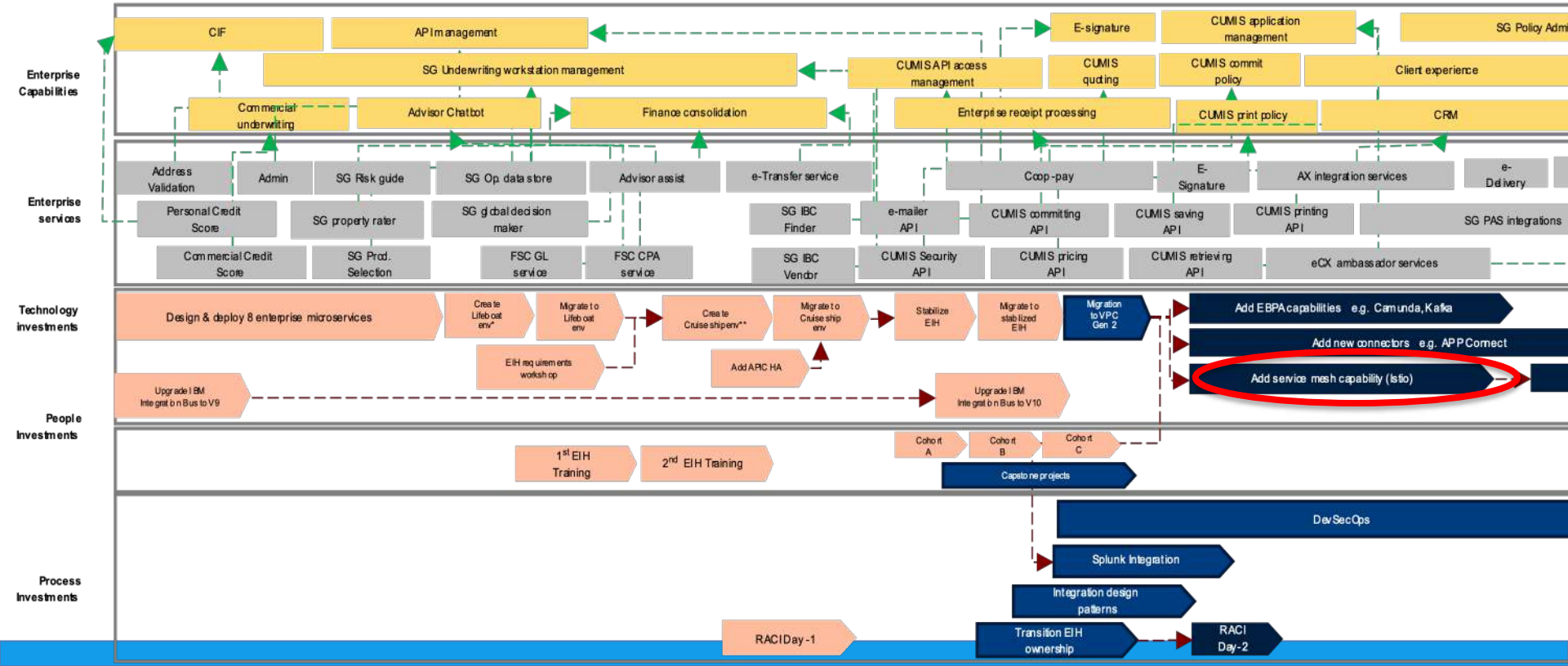
Architecture:



Examples:

All microservices: are exposed through API gateway (APIC) due to the need for monitoring/monetization/common security layer

Our Microservices Journey So Far...



Why Service Mesh?

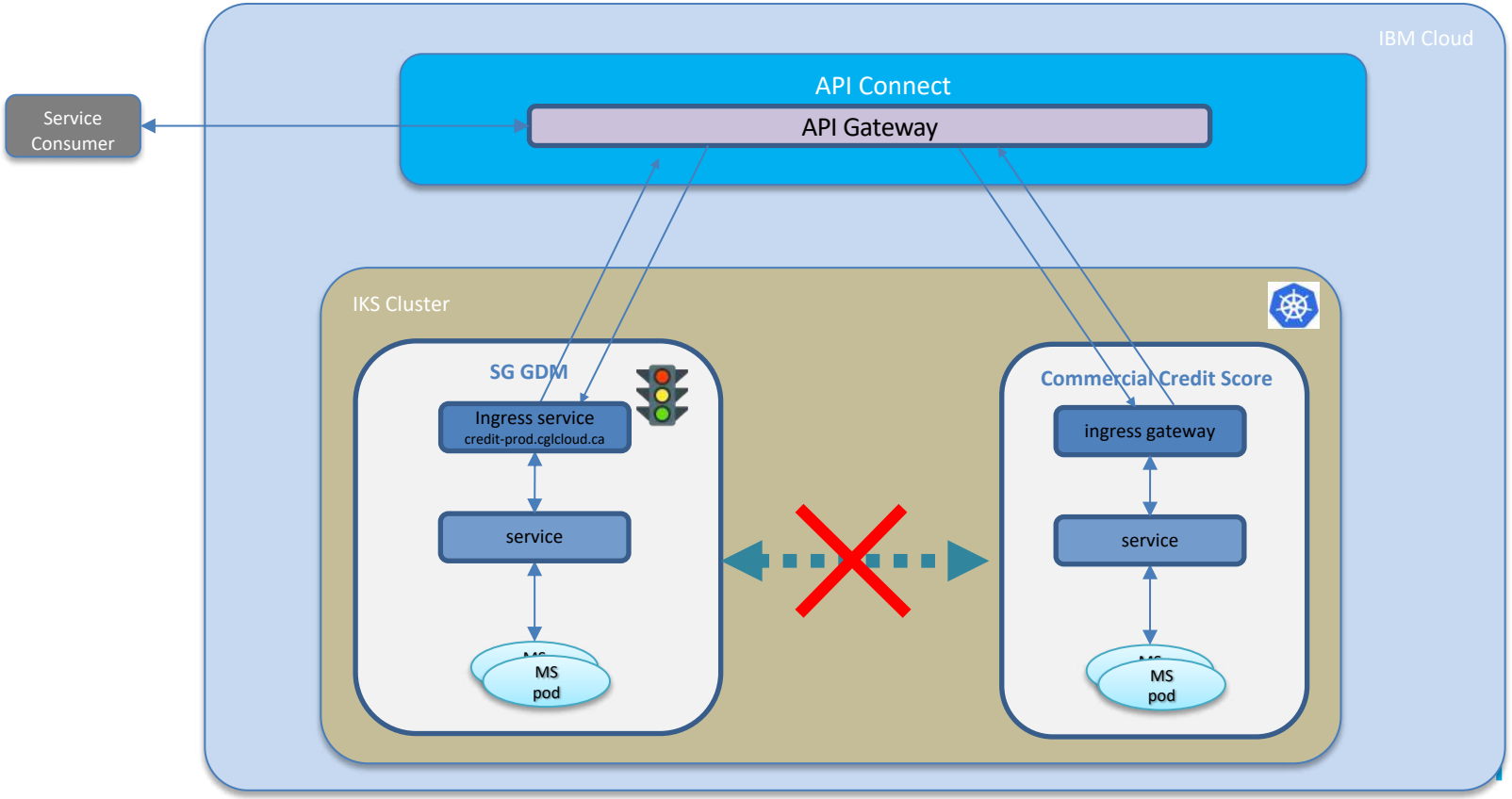
In our current Integration platform, microservices in K8 can only interact with each other through API gateway which is

- Inefficient (requires extra hop)
- Insecure (requires extra manual work to enable mTLS).

As number of services and K8 environments increases this manual process is **cumbersome** and **error-prone**.

How might we facilitate microservices interaction in a secure and efficient way that removes the need for extra manual work?

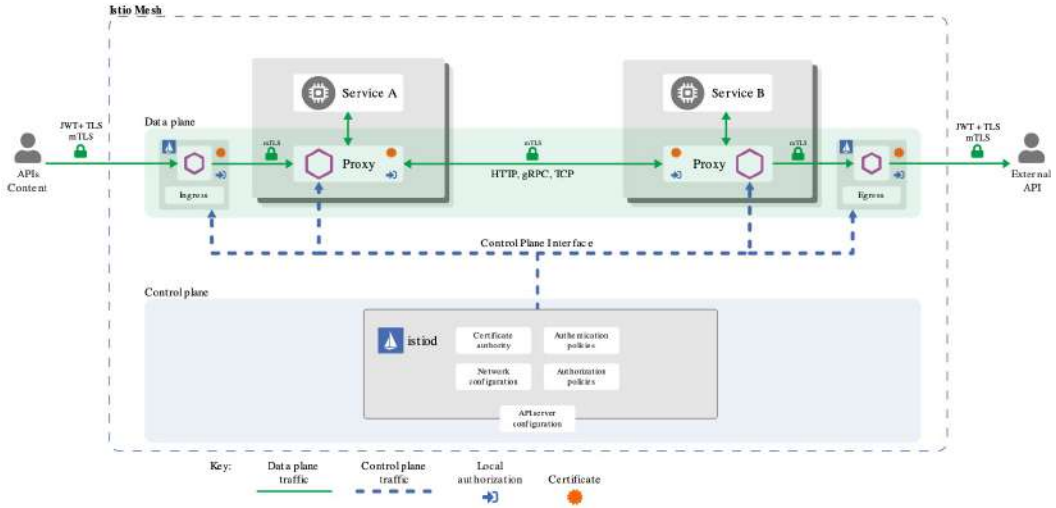
Inter-microservice communication today is indirect which introduces latency and despite mTLS exposes the interaction needlessly to public Internet.



Istio connects monitors and secures inter-microservices communication

Istio is a ***commercial open-source*** service-mesh technology that connects, monitors, and secures the containers in a Kubernetes cluster. It provides a set of security features, namely:

- ✓ Traffic encryption
- ✓ Security audit
- ✓ Mutual TLS
- ✓ Fine-grained access policies



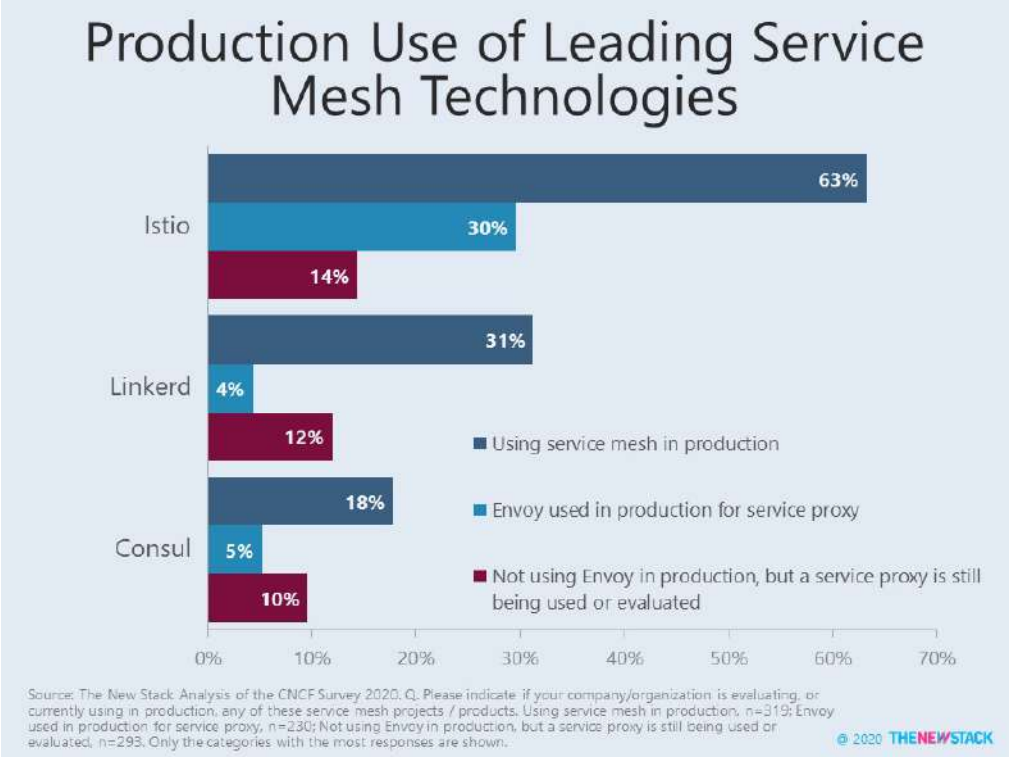
the co-operators
A Better Place For You

IT Enterprise
succeeding together

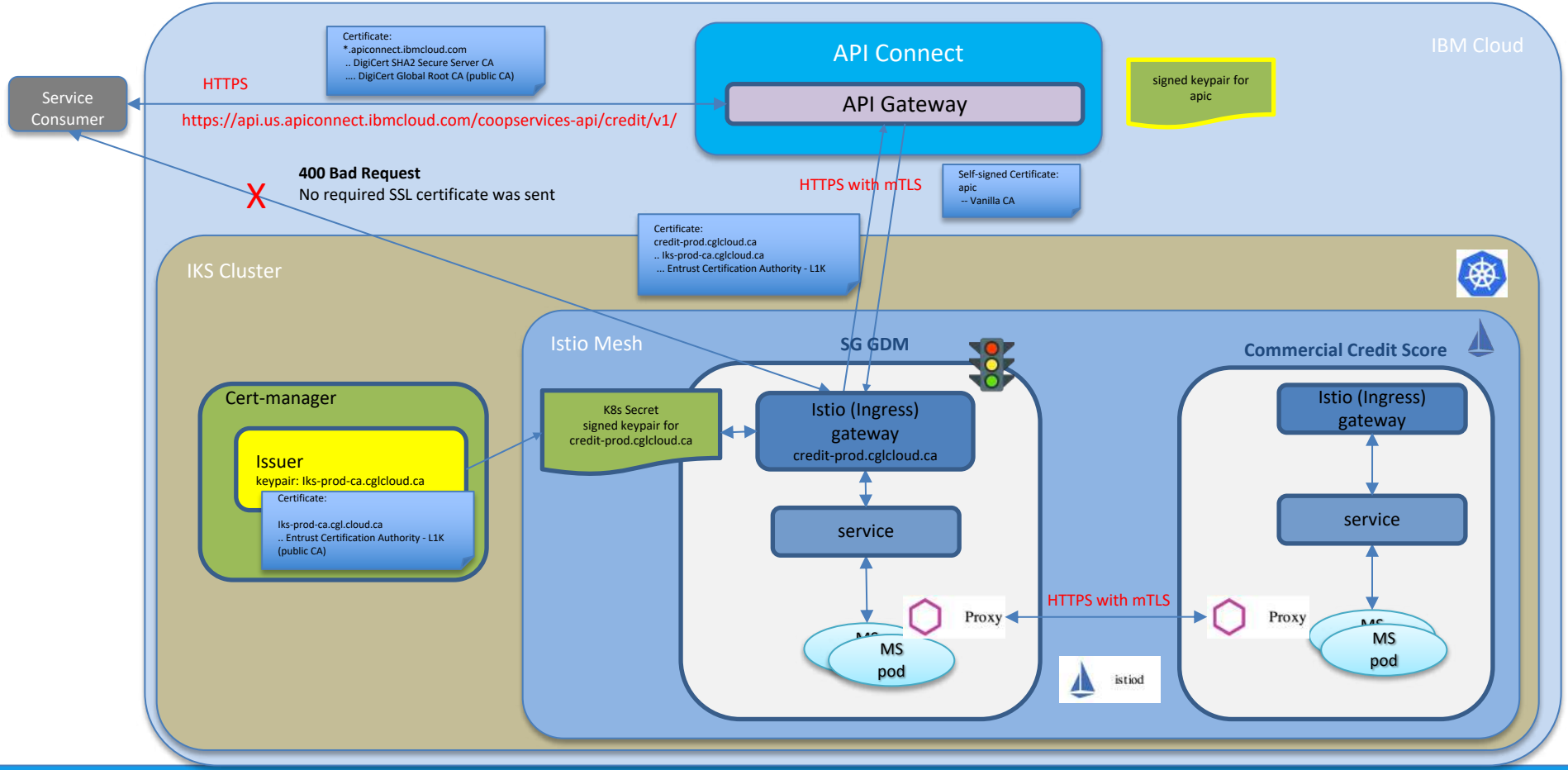
Solution Options: Istio outperforms its competitors

	Istio	LINKERD	Consul
Supported Workloads	Does it support both VMs-based applications and Kubernetes?		
Workloads	Kubernetes + VMs	Kubernetes only	Kubernetes + VMs
Architecture	The solution's architecture has implications on operation overhead.		
Single point of failure	No – uses sidecar per pod	No	No. But added complexity managing HA
Sidecar Proxy	Yes (Envoy)	Yes	Yes (Envoy)
Per-node agent	No	No	Yes
Secure Communication	All services support mutual TLS encryption (mTLS), and native certificate management so that you can rotate certificates or revoke them if they are compromised.		
mTLS	Yes	Yes	Yes
Certificate Management	Yes	Yes	Yes
Authentication and Authorization	Yes	Yes	Yes
Communication Protocols			
TCP	Yes	Yes	Yes
HTTP/1.x	Yes	Yes	Yes
HTTP/2	Yes	Yes	Yes
gRPC	Yes	Yes	Yes
Traffic Management			
Blue/Green Deployments	Yes	Yes	Yes
Circuit Breaking	Yes	No	Yes
Fault Injection	Yes	Yes	Yes
Rate Limiting	Yes	No	Yes
Chaos Monkey-style Testing	Traffic management features allow you to introduce delays or failures to some of the requests in order to improve the resiliency of your system and harden your operations		
Testing	Yes	Limited	No
Observability	In order to identify and troubleshoot incidents, you need distributed monitoring and tracing.		
Monitoring	Yes, with Prometheus	Yes, with Prometheus	Yes, with Prometheus
Distributed Tracing	Yes	Some	Yes
Multicluster Support			
Multicluster	Yes	No	Yes
Installation			
Deployment	Install via Helm and Operator	Helm	Helm
Operations Complexity	How difficult is it to install, configure and operate		
Complexity	High	Low	Medium

Istio adoption is higher for production use



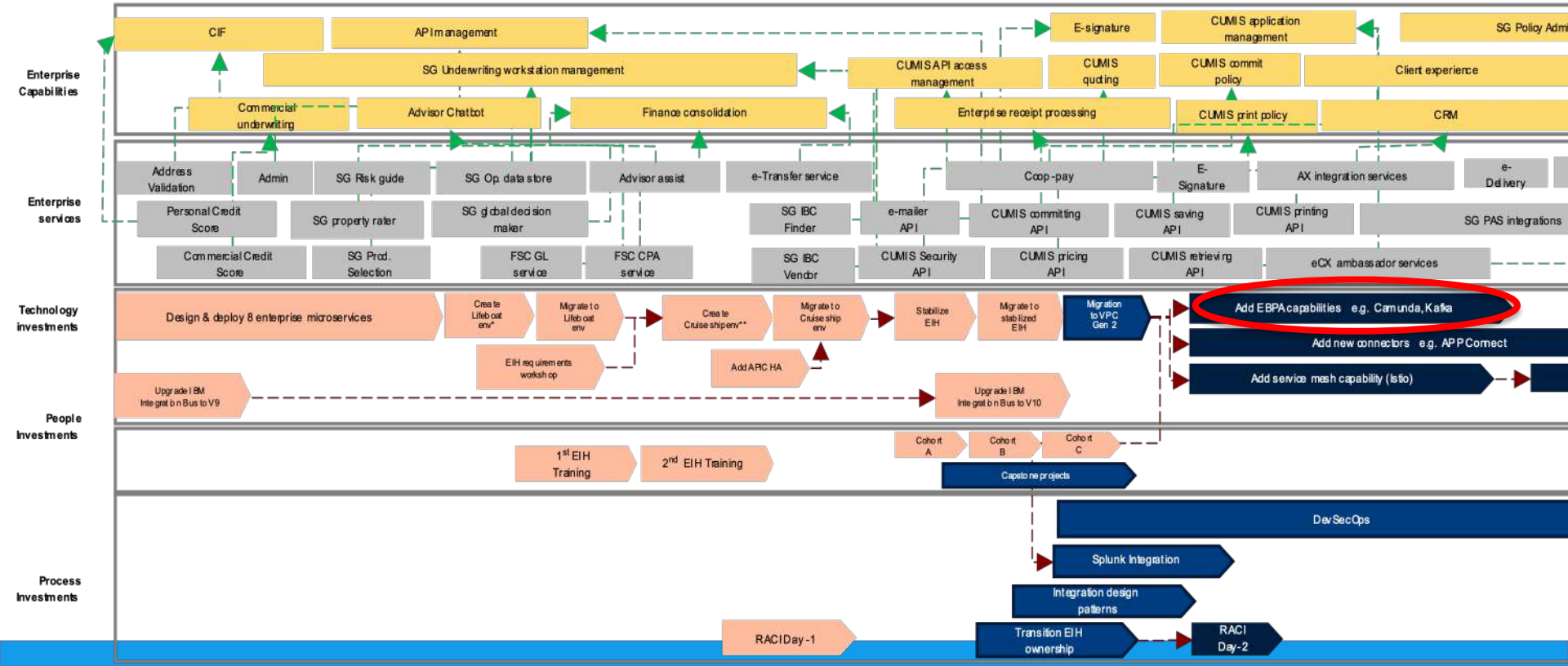
Deploy service mesh (Istio) for direct inter-microservice interactions



Proof of Value Success Criteria

Use Case	Success criteria
Deploy cert manager	• Remove the time developers spend on adding cert management to each microservice since cert manager does it as part of the platform services • Remove the time we spend on managing certs during use of a microservice • Remove the incidents related to expired certs • Minimize impact on existing services due to enabling the service mesh
Deploy Service Mesh	• Reduce security risk of service-to-service connections in IKS because we do not need to go through public internet and API gateway. • Reduces traffic of API gateway for microservice-to-microservice interactions • Improve responsiveness (quality) of the services that are composed of other services by removing IKS←→ APIC ←→ IKS traffic • Minimize impact on existing services due to enabling the service mesh

Our Microservices Journey So Far...

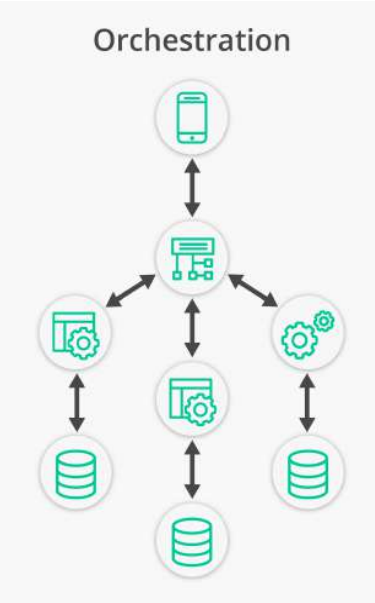


Why Enterprise Business Process Automation (EBPA)?

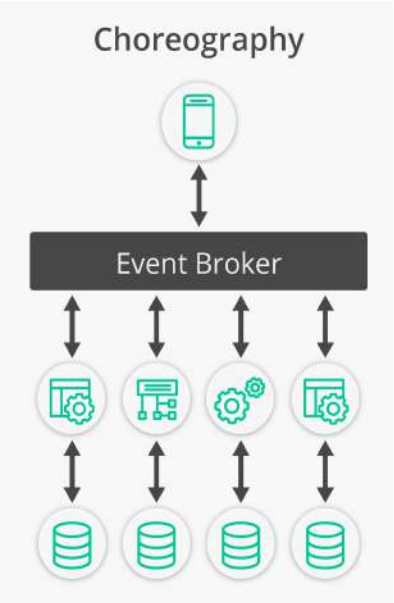
- We have business processes that interact with our employees, advisors, partners and/or clients. These processes glue together several business capabilities and span internally and externally across multiple applications, stakeholders and lines of business.

- How might we integrate our enterprise services to deliver business processes in a consistent manner **across** internal and external consumers while maximizing reuse, change agility and reducing risks?

Microservices Orchestration VS Choreography



- A central service as orchestrator or process flow engine
- Pros:
  - Mature BPM products
  - Easier to introduce human interaction
  - Easy to maintain business processes in one center
- Cons:
  - Services are tightly coupled to the central service
  - Similar to monolith
  - Central service is SPF

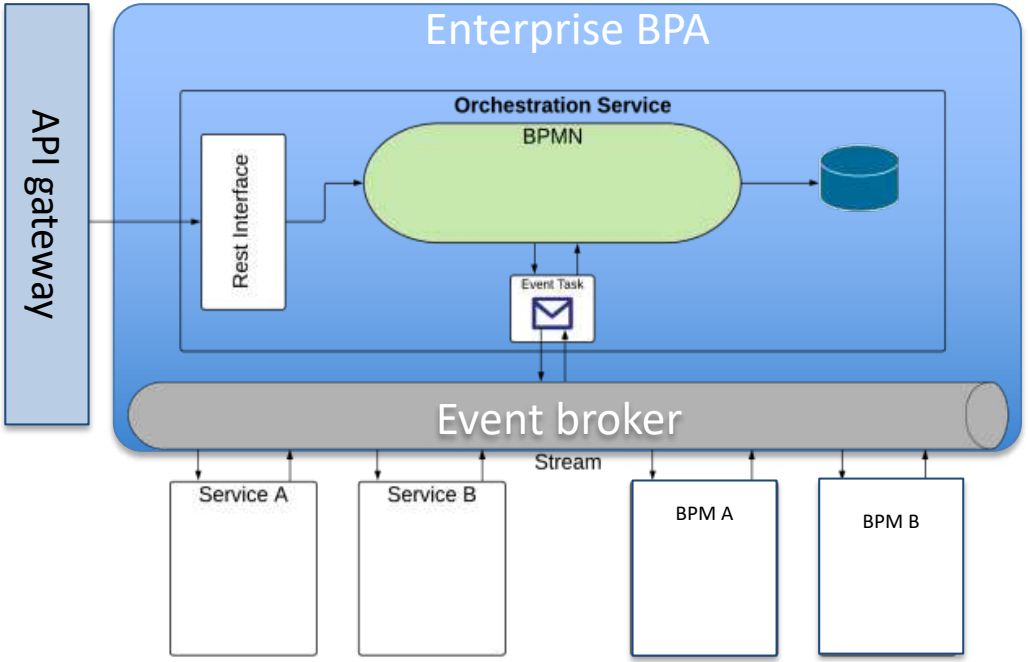


- Decentralizing decisions, logic & interactions between services via *Events* published by an *Event Broker*
- Pros:
  - Low coupling of services
  - Works better with agile delivery
  - Higher performance (faster)
- Cons:
  - Difficult to maintain since business process are spread across multiple services (no notion of process)
  - Managing transactions e.g. error handling is much more difficult
  - Needs custom development for human interaction

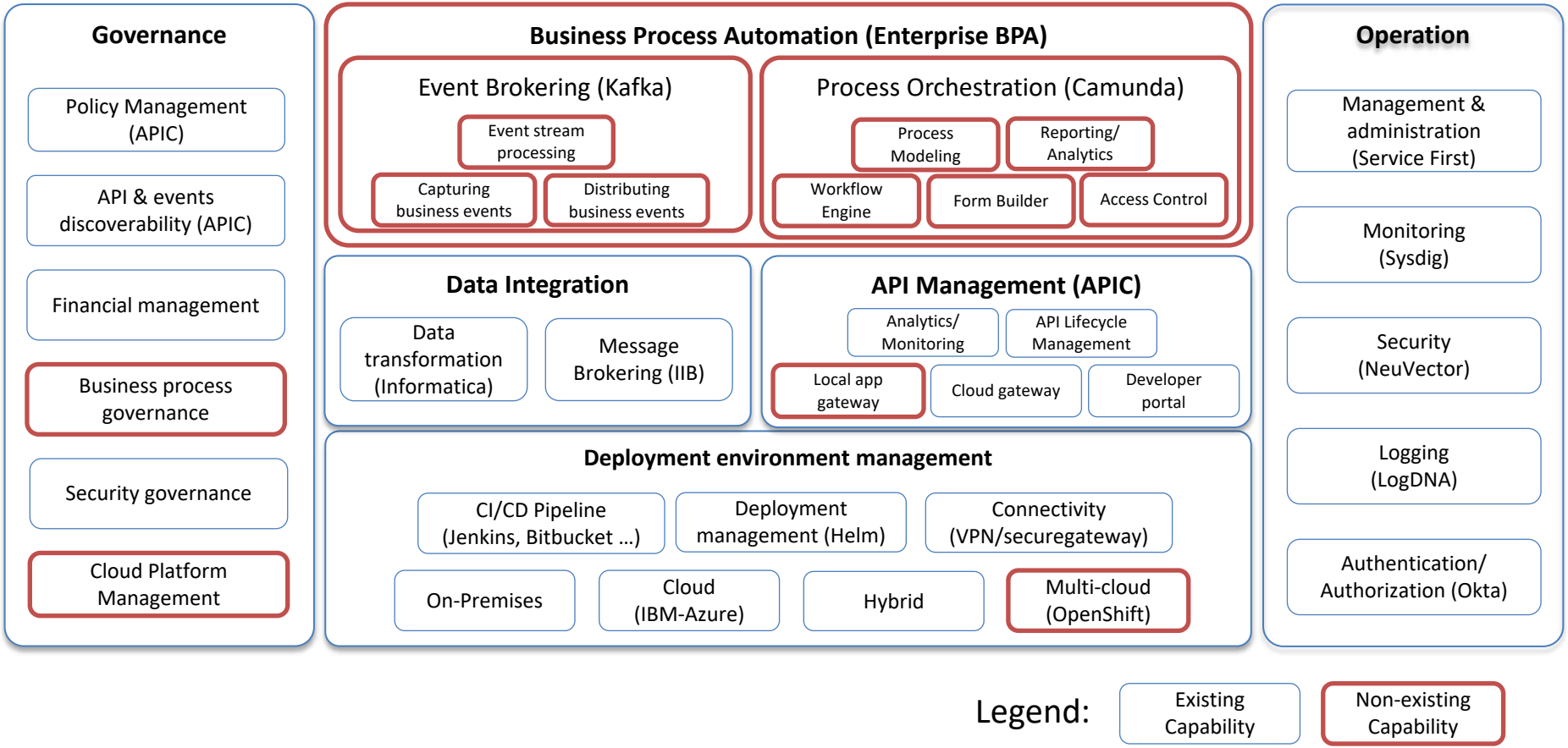
Enterprise BPA: a hybrid approach

Enterprise BPA includes a central orchestration service that manages the business processes through an event broker that communicates to other services and BPMs through events.

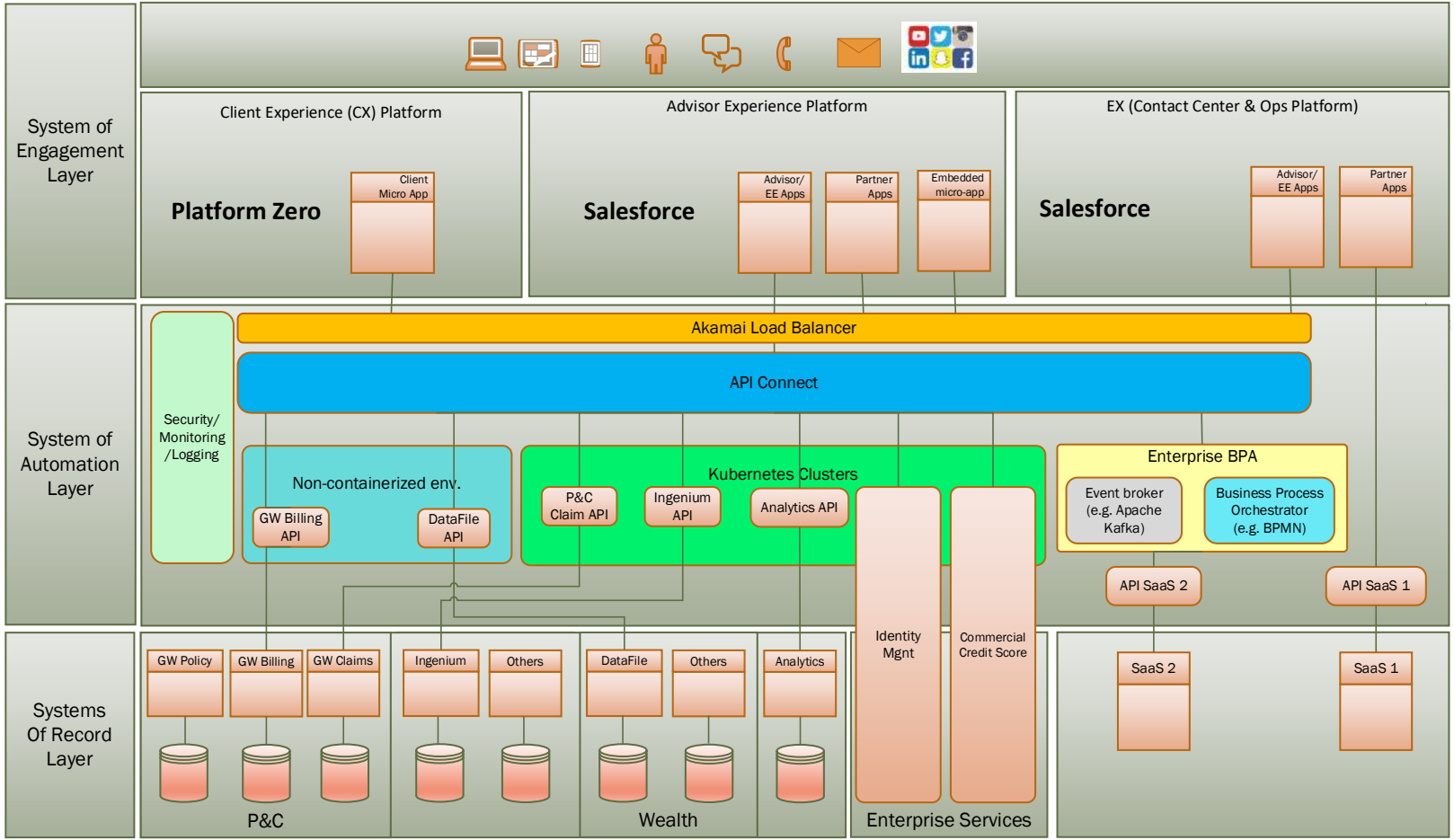
- Easy to manage & monitor complex business processes
- Easy to support processes requiring Human interactions
- Decoupling services from each other
- Reduce blocking
- Scalability (each event processor can be scaled separately)
- Enterprise BPA helps us to not lock in a specific vendor & keep enterprise control of our integration to change/future extension



Our Integration Capability Model



Target Integration Architecture



Engagement

- Client/advisors/partners Interaction Points
- Omni-channel Interaction

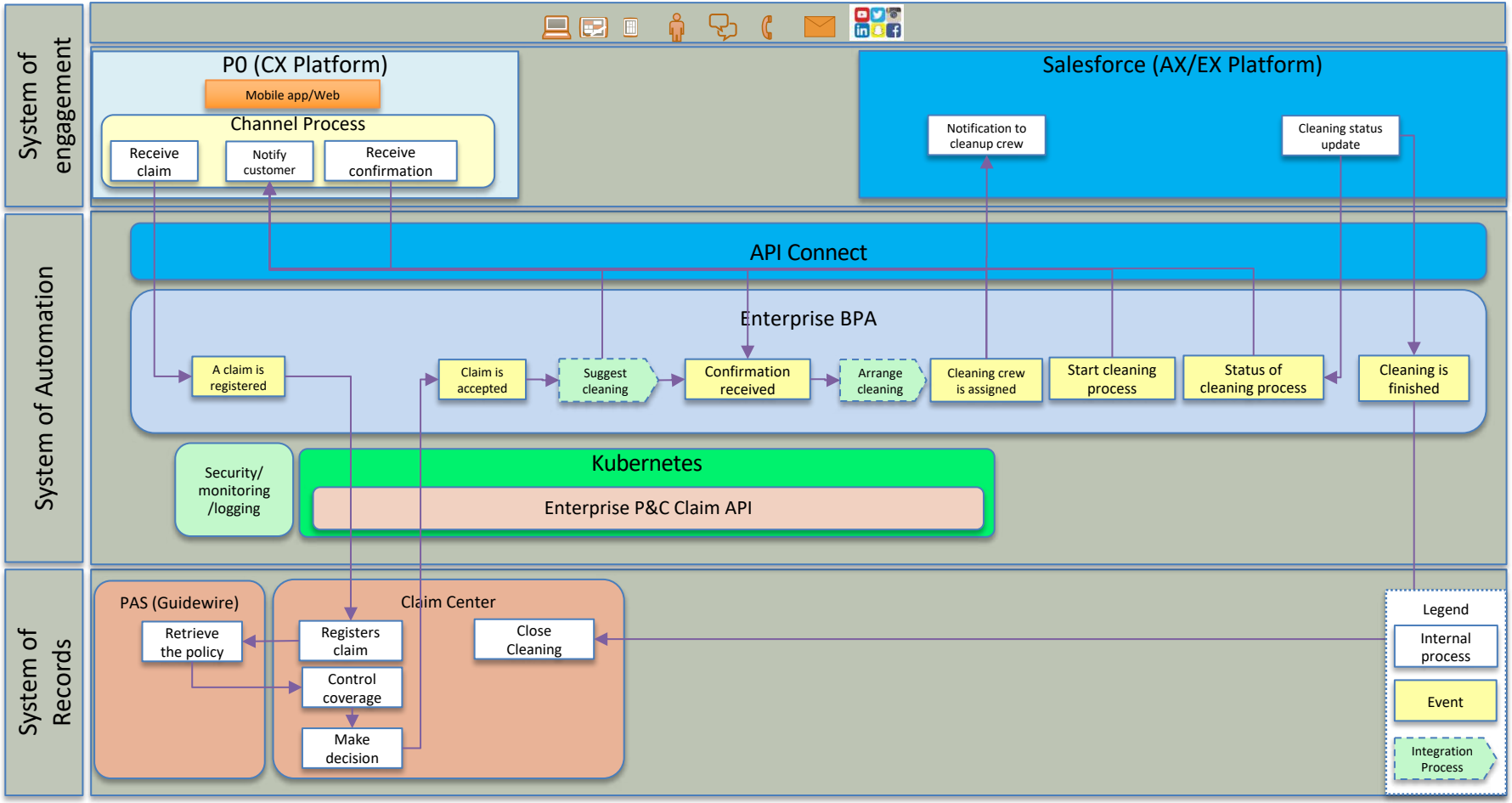
Automation

- Handles interaction between SoE and SoR layers
- Includes Enterprise Integration Hub (API Gateway, K8 clusters) and enterprise process choreographer (Event broker & BPMN)

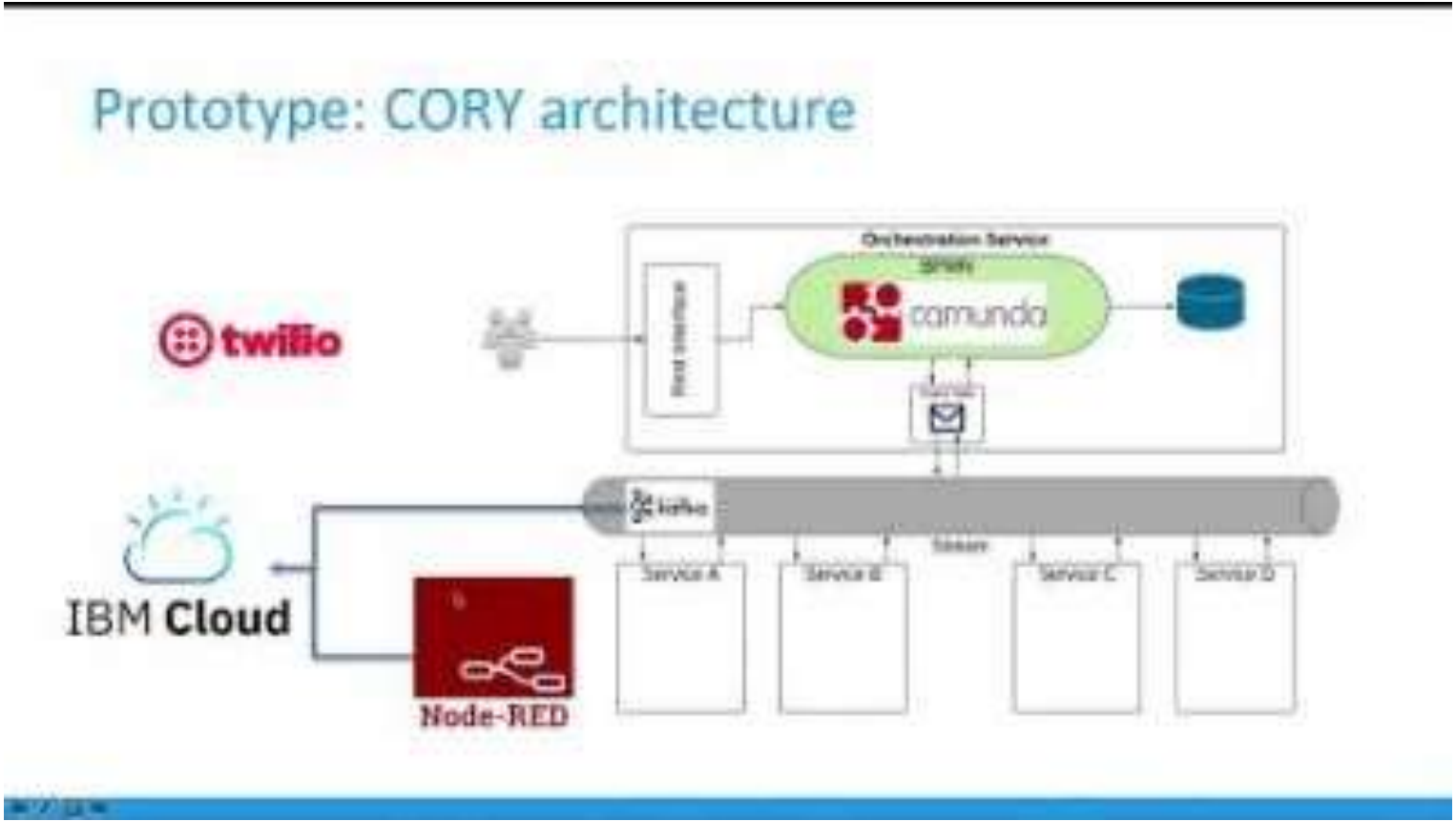
System of Record

- Includes all enterprise systems, enterprise services, APIs and external services/data sources

Example 1 for enterprise process: Flood claims-home cleaning process

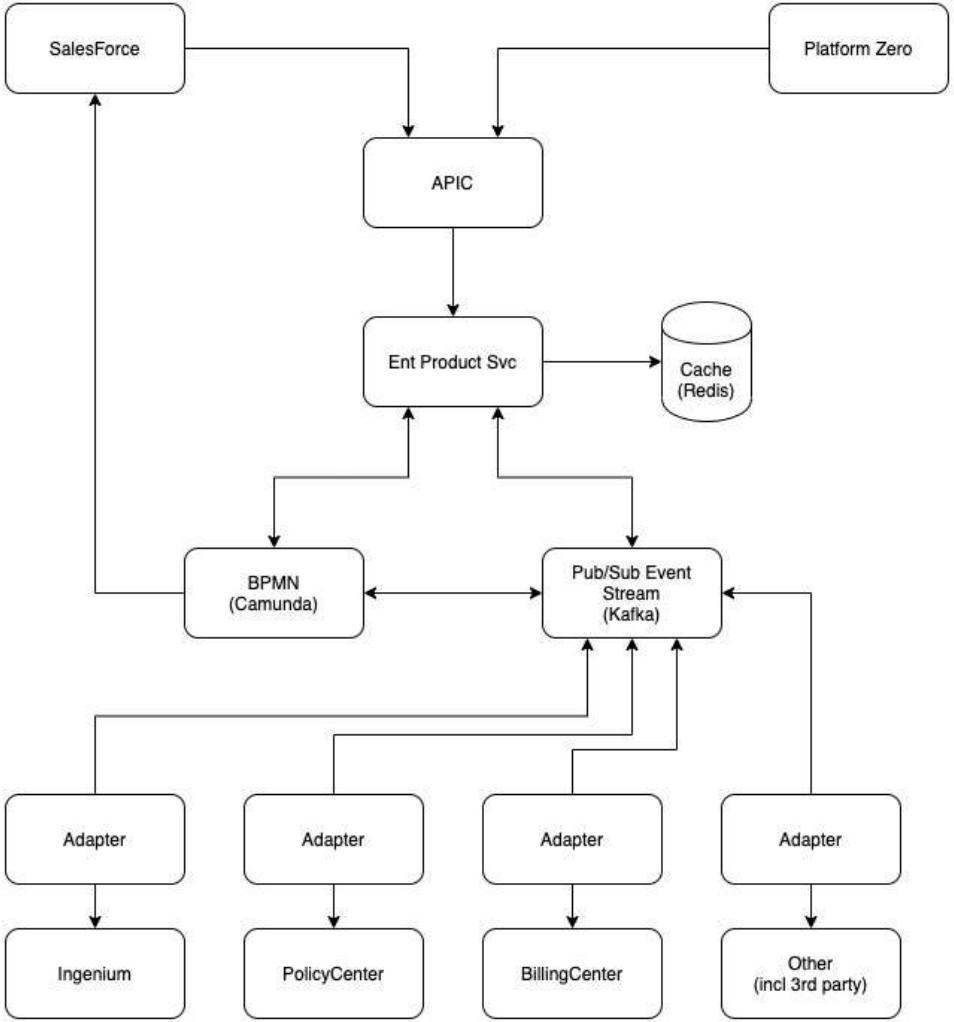


Demo of EBPA prototype flood claims-home cleaning process



Enterprise product service

A pilot use case for EBPA



Summary

- Digital transformation is a vital part of Cooperators strategy
- EIH & Microservices are important enablers for digital transformation
- Success in microservices deployment requires change in culture, skill, processes, and technology (maybe the easiest one)
- A mature, stable, resilient, and easy to use cloud platform (PaaS) is a must for microservices success
- Technology is a changing target (we need to make fast decisions)
- We need to fail faster and not afraid of failing (being more agile)
- Microservices increase complexity and data redundancy. We should be careful when we decompose our monolithic applications to microservices
- Service Mesh and EBPA are important technologies to facilitate microservice-to-microservice interactions
- EA can align the whole enterprise towards a successful migration from monolithic to microservice architecture

Thank you

the co-operators®

A Better Place For You®