

DevOps: The Secret Sauce for Building Forward Looking Software

Jim Nasr
CDC/OPHSS/CSELS

CDC Health Information Innovation Consortium (CHIIC)
August 16, 2016



“in brief...”

- Underlying currents lead us to evaluate architectural approaches for building software at DHIS
 - These include: state of existing DHIS portfolio, interagency interoperability, fed direction, public health trends and emerging technology standards to support innovation and better customer/citizen interaction
- IF (open_data == mission) THEN (APIs, DevOps == king) AND (software_approach != past)
- How? Start small. Use R&D as accelerator. Build Architectural Guidelines and (tactile) Artifacts. Reconceive Project Implementation. Seed Development. Collaborate for DevOps.
- Automate code integration, build, testing and release. Monitor services & use smart deployment techniques.



What is Forward Looking Software?

- Software designed to be business domain-driven
- Software designed to interoperate
- Software designed to be uncoupled (from other applications or DBs, vendors, proprietary software)
- Software designed to abstract implementation, be observable, deploy independently
- Software designed to be test-driven, built automatically, released rapidly
- Software designed to be distributable, scale on demand, be resilient, isolate failure



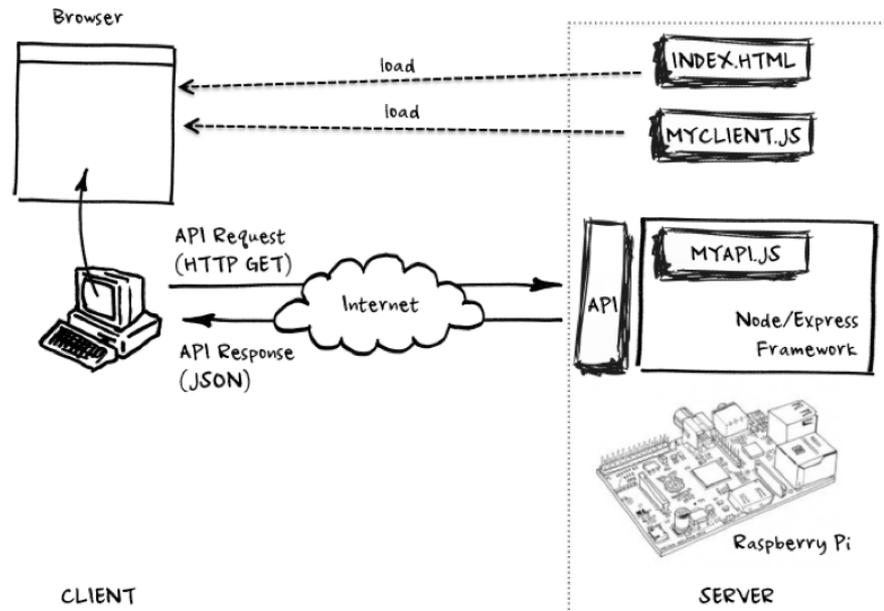
Select Trends in Public Health

- Open Data
- Interoperability
- HL 7
- Open Source
- FHIR



Select Trends in Technology

- Cloud
- Mobile
- Real-time
- Open Source
- IoT (Internet of Things)
- Distributed Computing
- RESTful APIs
- Micro-services



Observations at CDC

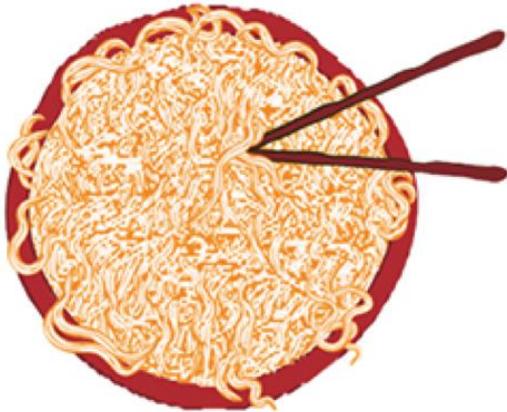
- Traditional Siloed Applications
- “New” Services Initiatives
 - DHIS: Message Transport Service (MTS)
 - EOC: RedSky
 - NHSN: Vocabulary, NHSN2
 - MISO: Global Travel, IRIS
 - Chronic: Extract data and visualization (Socrata)
 - NCHS: Coroner Case Management modernization
 - Cross Agency: Surveillance Data Platform (SDP)
- Innovation with 3rd Parties
 - GA Tech and other academic partners; FHIR, SaaS apps



Evolution to Micro-services

1990s and earlier

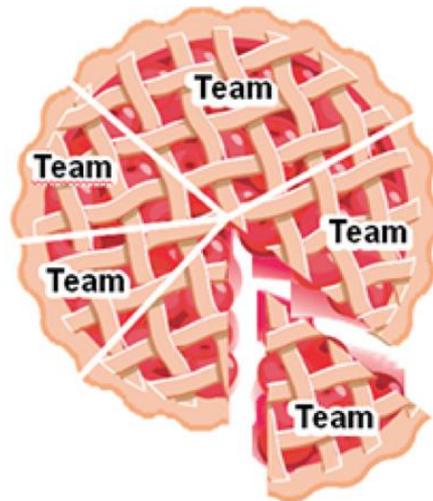
Pre-SOA (monolithic)
Tight coupling



For a monolith to change, all must agree on each change. Each change has unanticipated effects requiring careful testing beforehand.

2000s

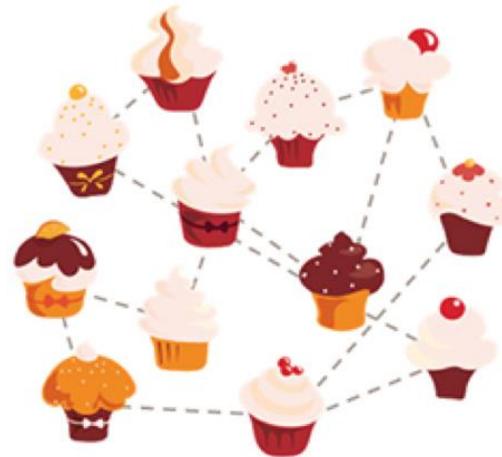
Traditional SOA
Looser coupling



Elements in SOA are developed more autonomously but must be coordinated with others to fit into the overall design.

2010s

Microservices
Decoupled



Developers can create and activate new microservices without prior coordination with others. Their adherence to MSA principles makes continuous delivery of new or modified services possible.

Source: PWC



Micro-services Shifts Complexity to Where it Belongs

Microservices shift complexity to where it belongs

Microservices are no free lunch, but they are smart nutrition.

By Irakli Nadareishvili, April 5, 2016

Editor's note: Register for the free webcast "[Taking a Design-Based Approach to Microservices](#)," hosted by Mike Amundsen and Matt McLarty for the CA Technologies API Academy. You'll learn how to design a microservice architecture that uses best practices and accounts for your organization's culture.

Virtually every talk, discussion or a blog post about microservices architecture makes a point of warning its audience that this novel architectural style, while bringing many benefits, increases operational complexity significantly.

Source: O'Reilly

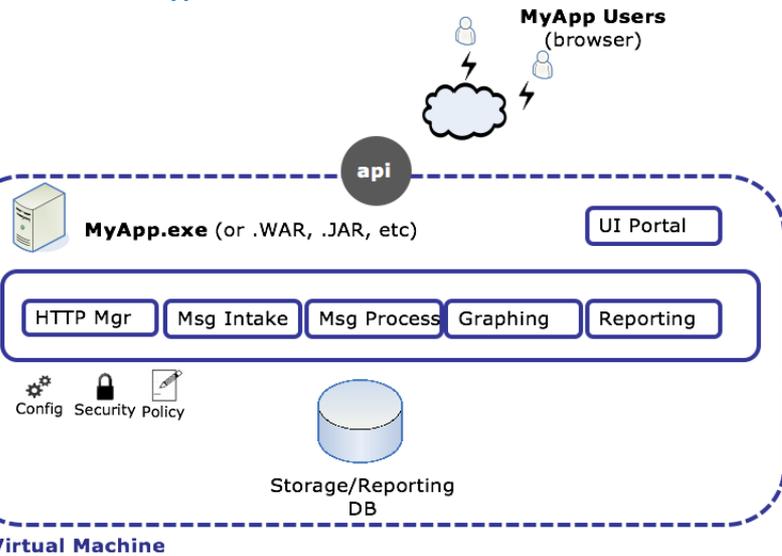


London Eye (source: Pixabay).

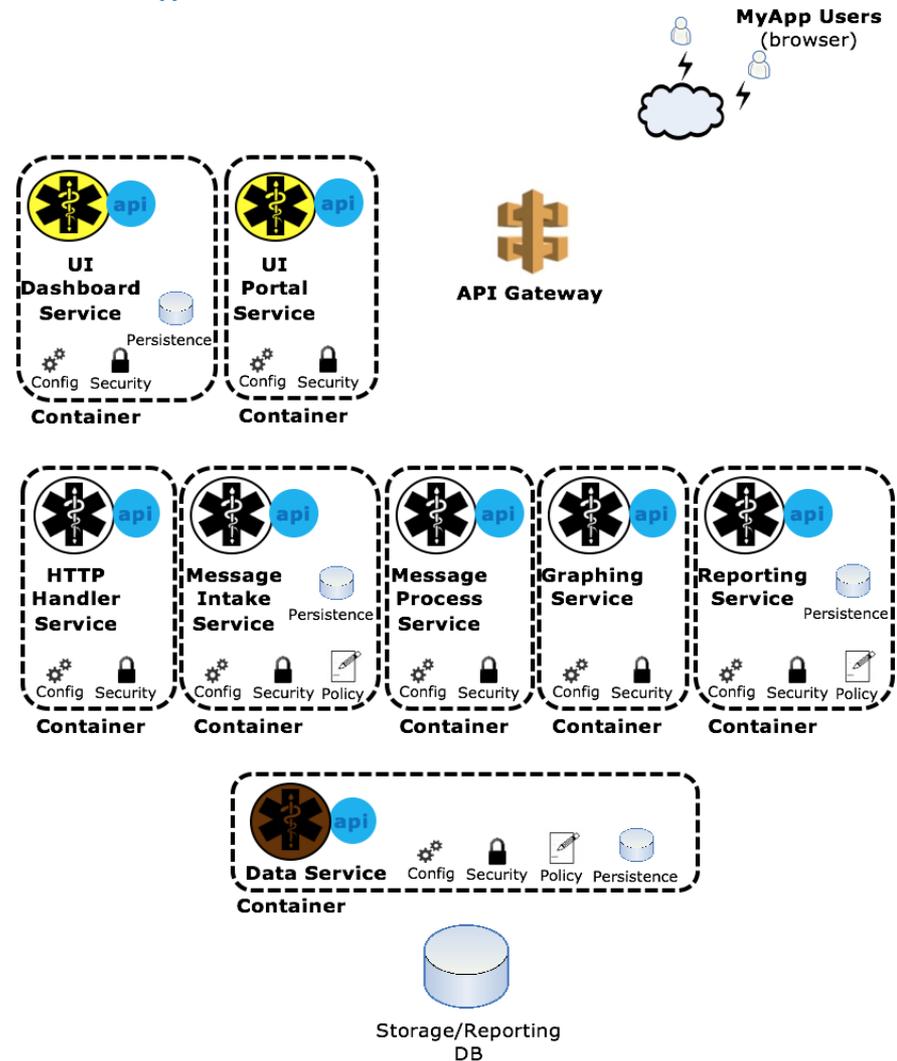


Micro-services Example

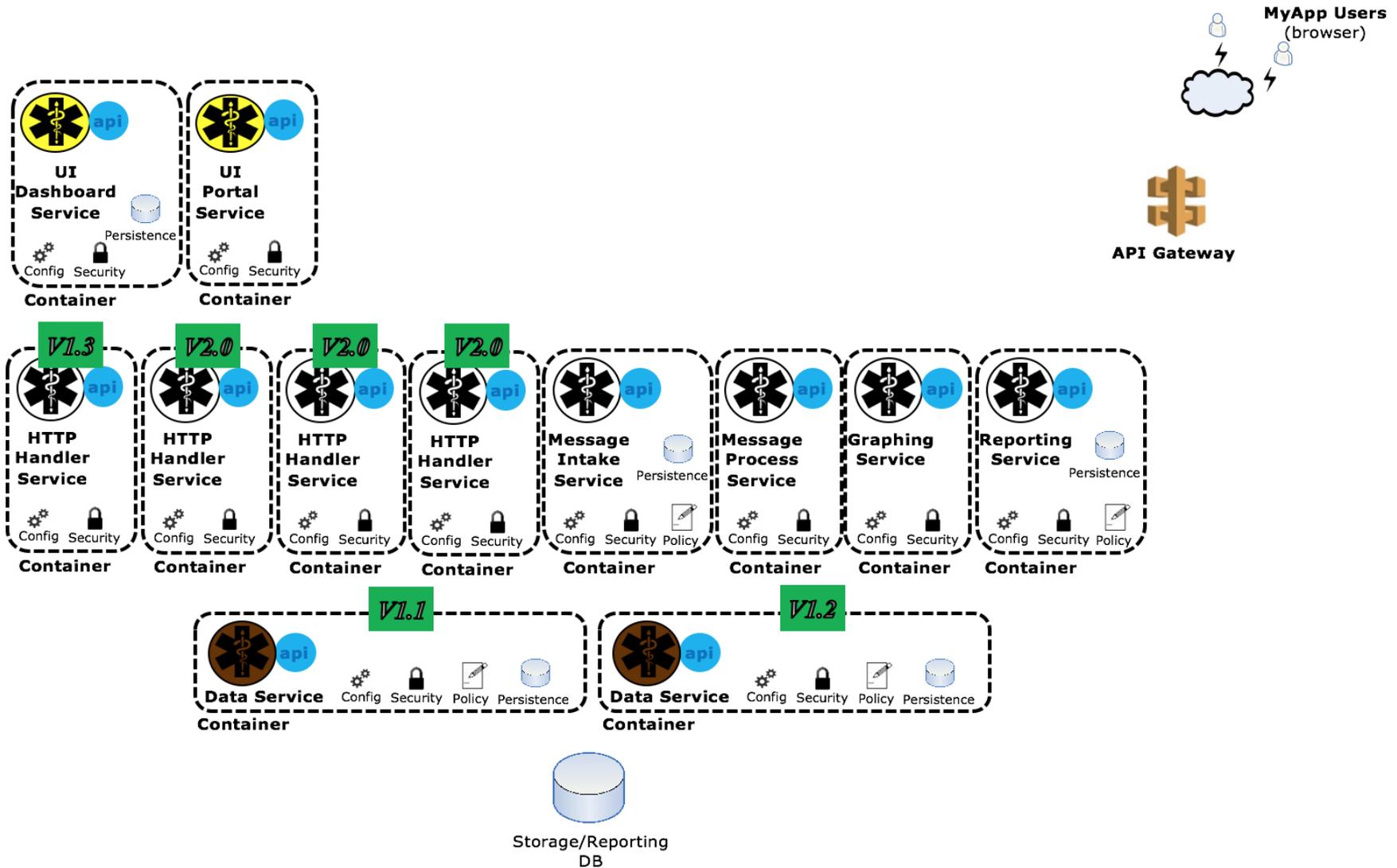
Monolithic App



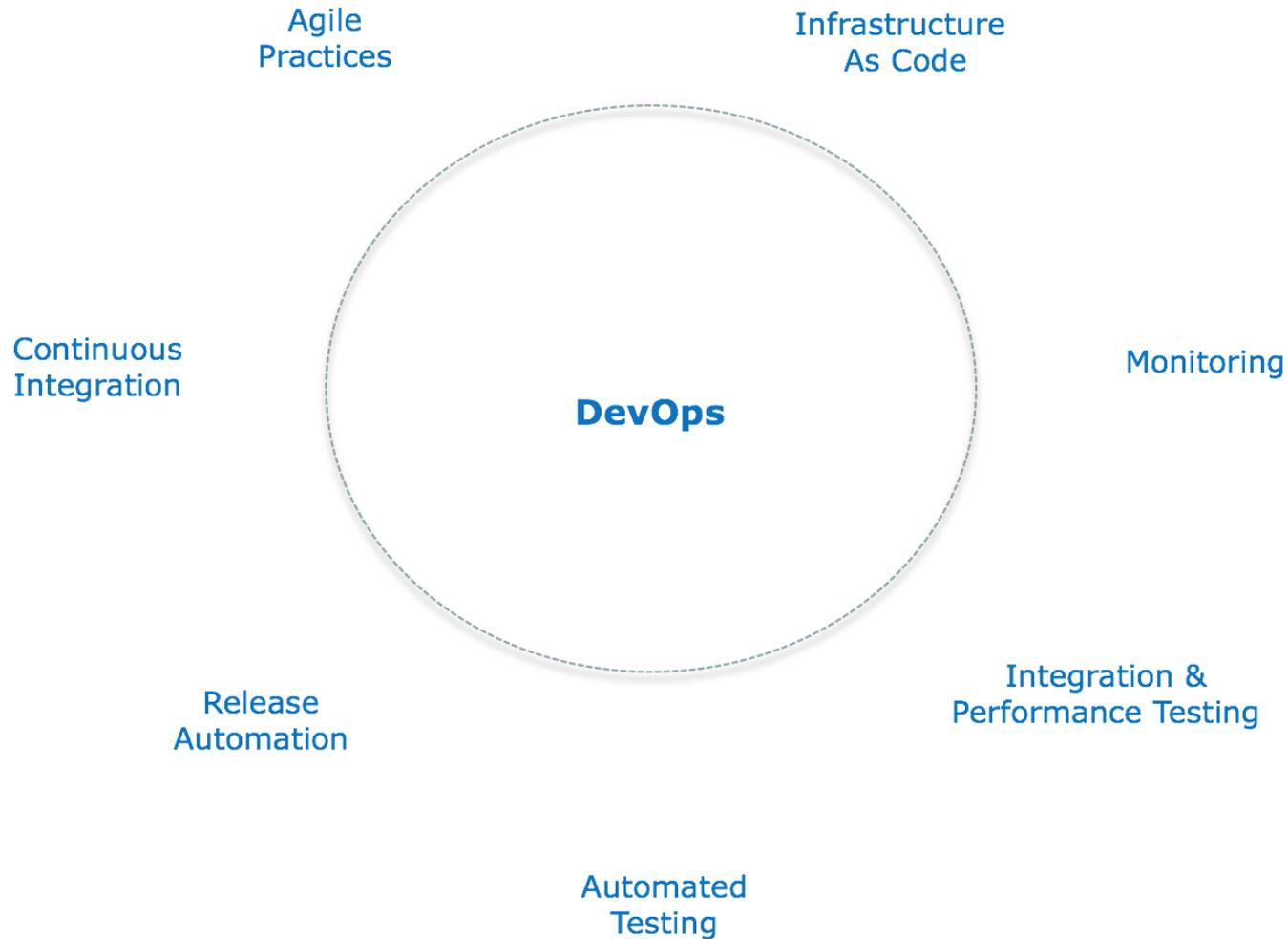
Micro-services App



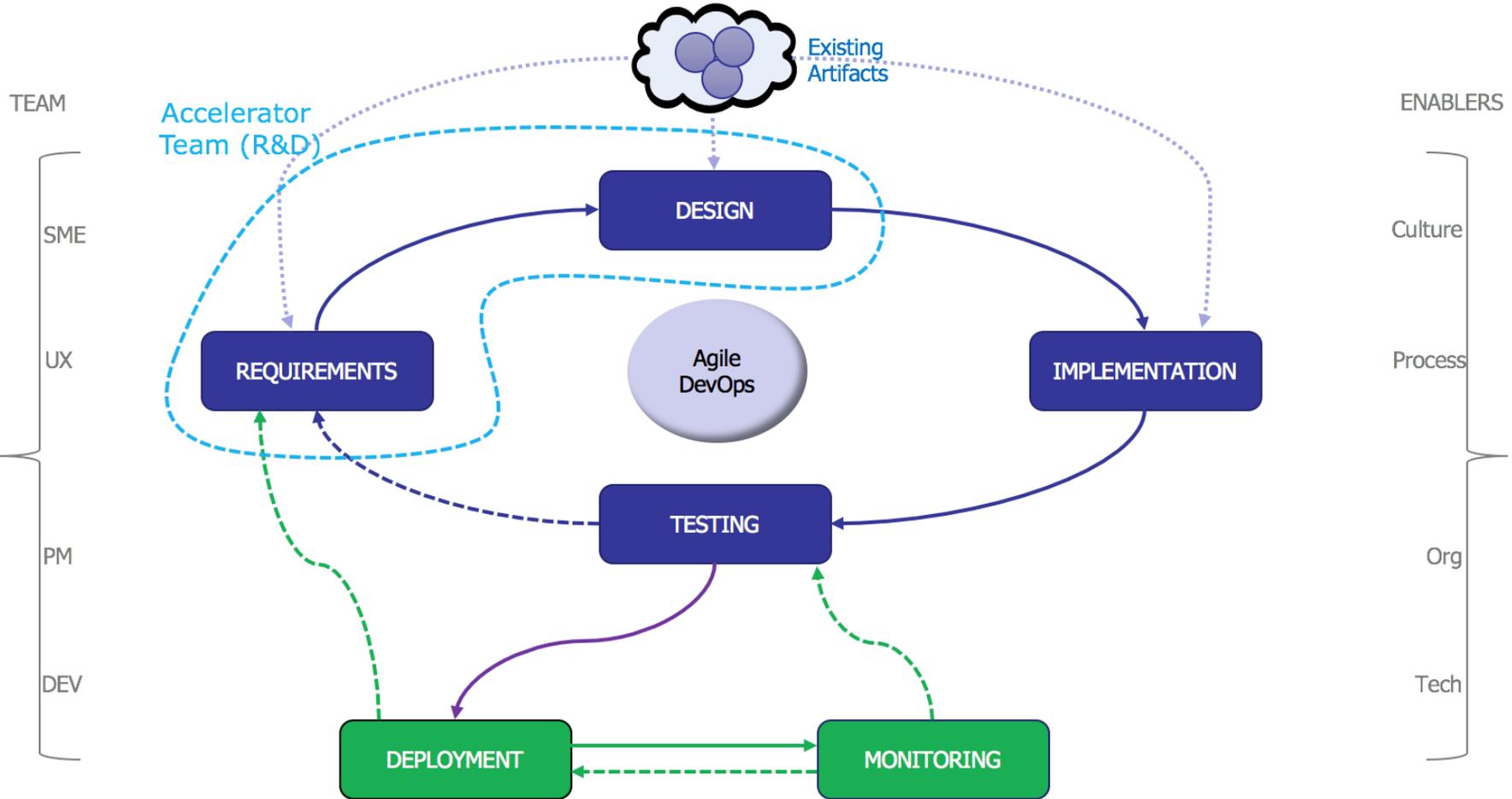
Micro-services in Real Life



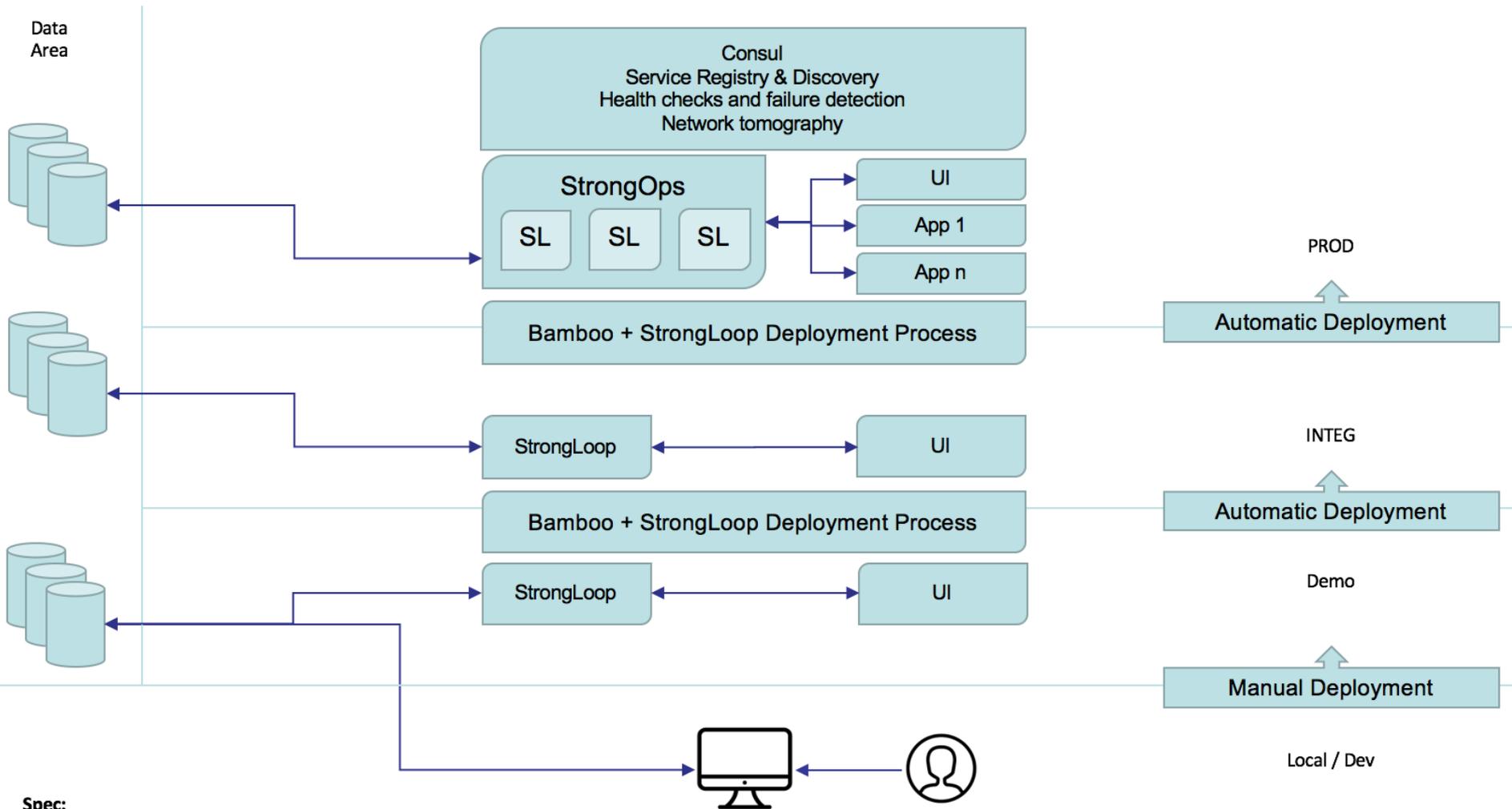
DevOps for Micro-services



Impact of Micro-services and DevOps on Application Development at DHIS



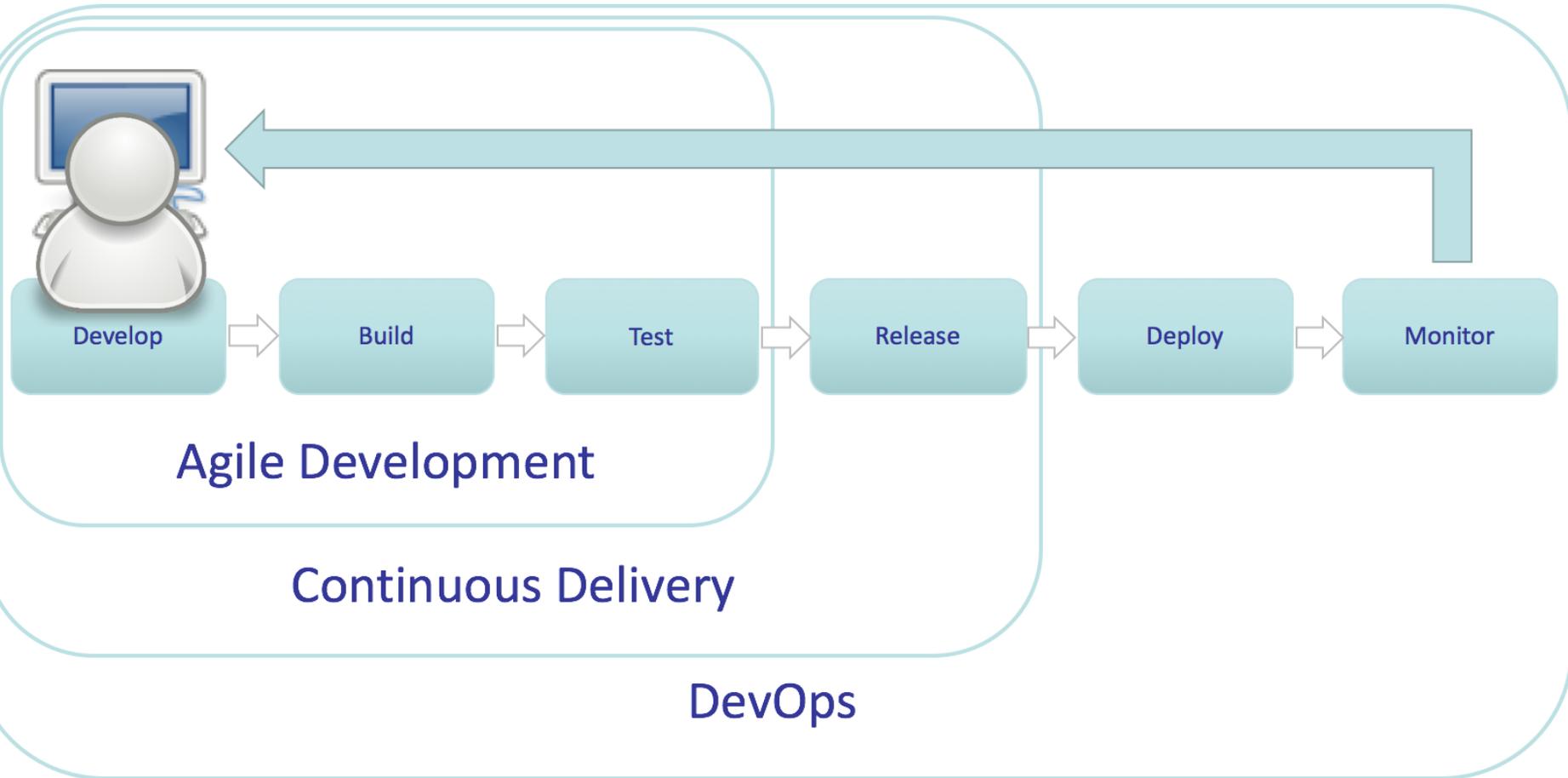
Implementing Full DevOps in Sandbox



Spec:
-3*CentOS
-1*Win7 64bit



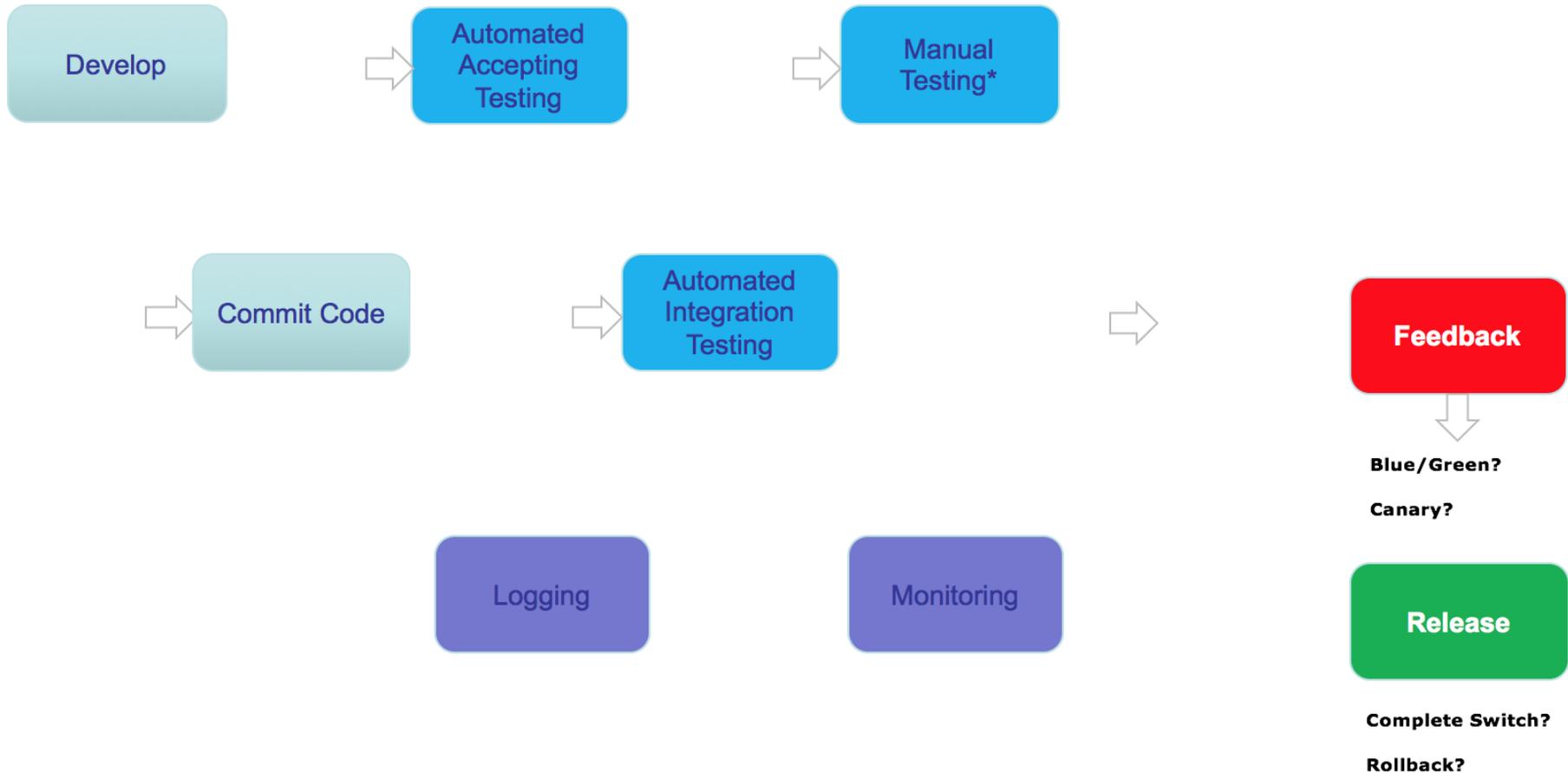
Continuous Delivery at a Glance



Source: GTRI



Continuous Delivery in Detail



Why Continuous Delivery?

- Practically *impossible* to do micro-services otherwise
- Auto provisioning—faster packaging, smaller deployments
- Comprehensive testing
- Feedback from production
 - Risk mitigation through circuit breakers, switch gates and rollback
- Monitoring discrete services
- Logs for discrete services, events & triggers
- Reproducible errors through automation



Continuous Delivery Technologies



Jenkins



Jenkins



Commit Code
To Git



Compile, Test &
Package



GitHub



Local Git Clone
Repository

GitHub



Local Git Repository &
Docker Images

Build components &
containers, perform
integration testing



Deployed to
Host Server

Full lifecycle
backlog/issue
tracking



Landscape of DevOps for Micro-services

Quick Mock-up

Pixate
Sketch

Code

Github
Plunker

Proxy

Varnish
Squid

OAuth2

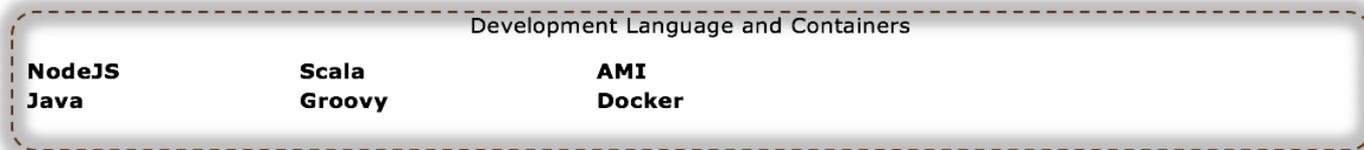
Google
OpenAM
Gluu

Testing

Latency Monkey
Chaos Monkey

Security Testing

ZAP
Brakeman
Nessus

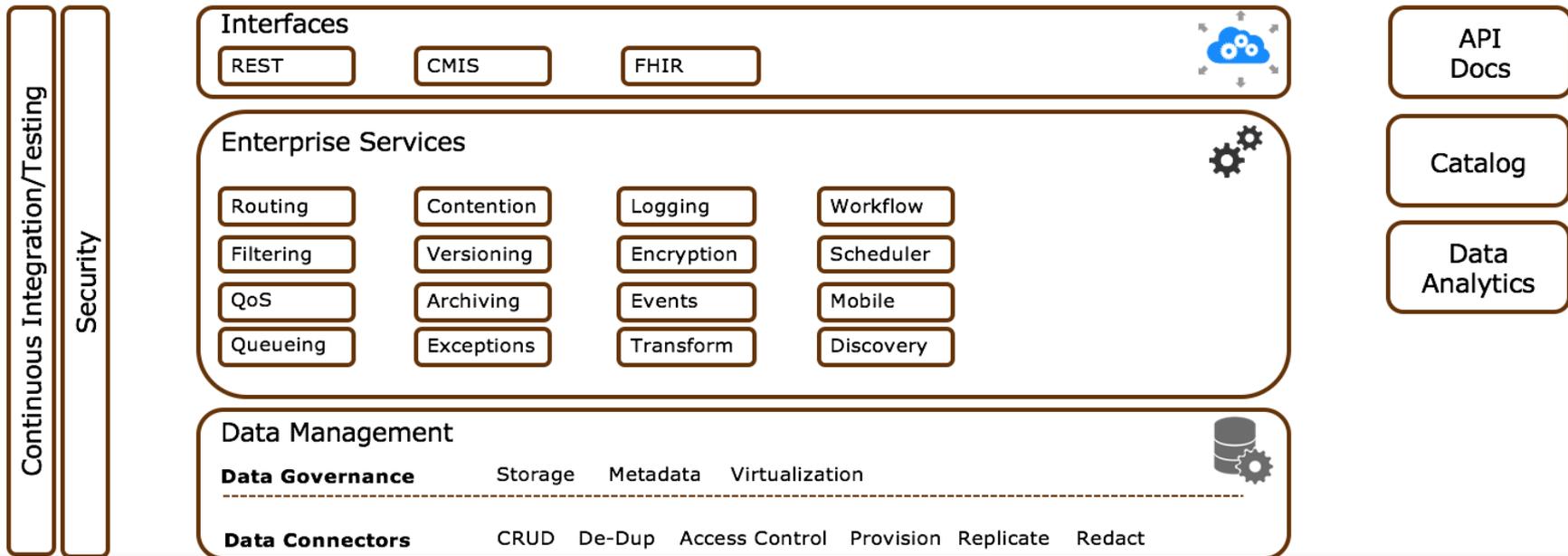


Future Architecture Overview

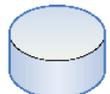
Existing Applications



Future Applications



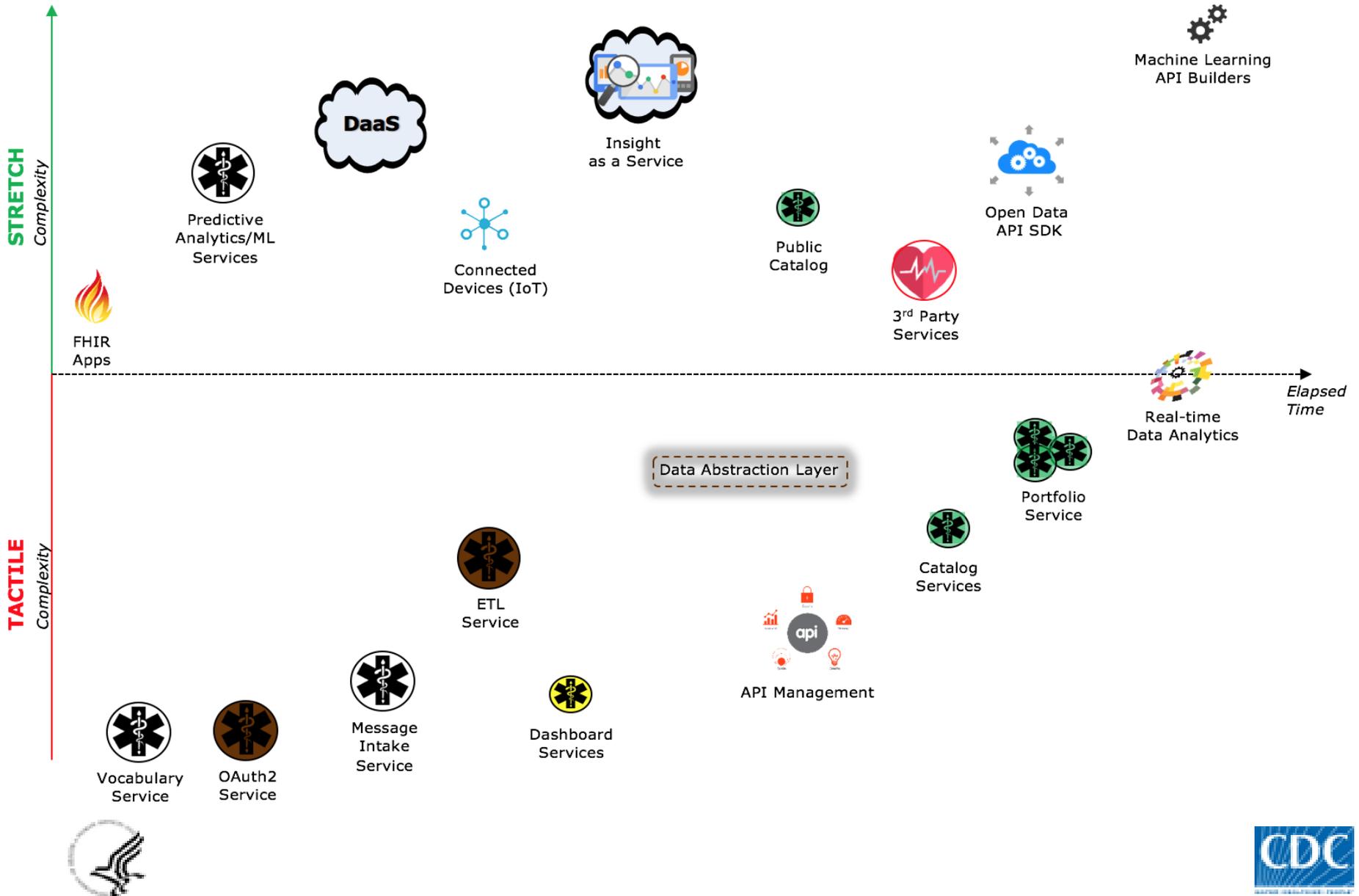
Existing Data Sources



Data Abstraction Layer



Potential Development Roadmap



Close

Questions?

