GSE UK Conference 2018

Better, stronger, faster; The Mainframe….. the Machine!

GUIDE SHARE EUROPE UK REGION

Agile Applications using a Microservices approach.

Ian J Mitchell, IBM DE
Application Agility Architect

5th November 2018

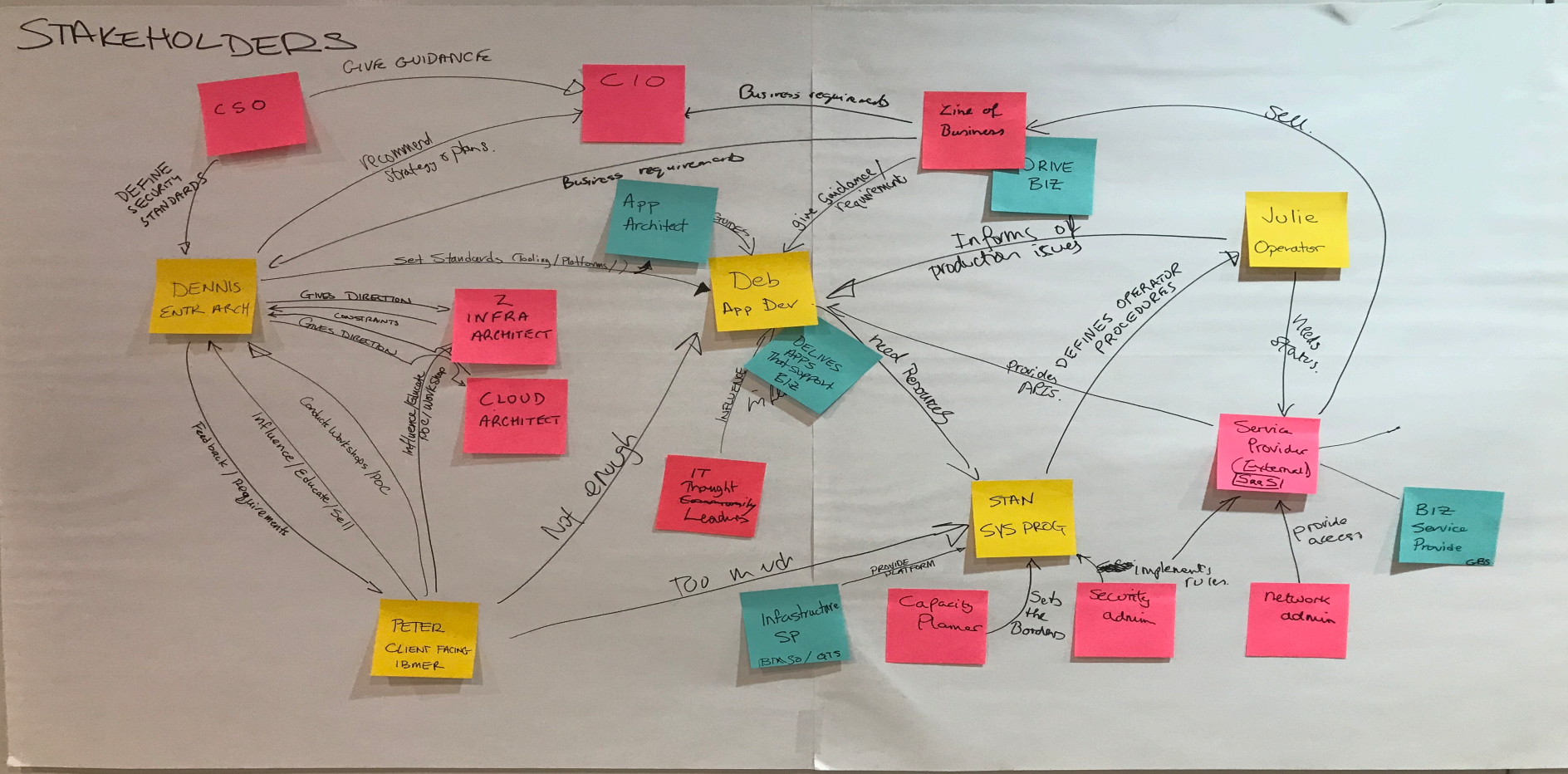
Session: GB

IBM

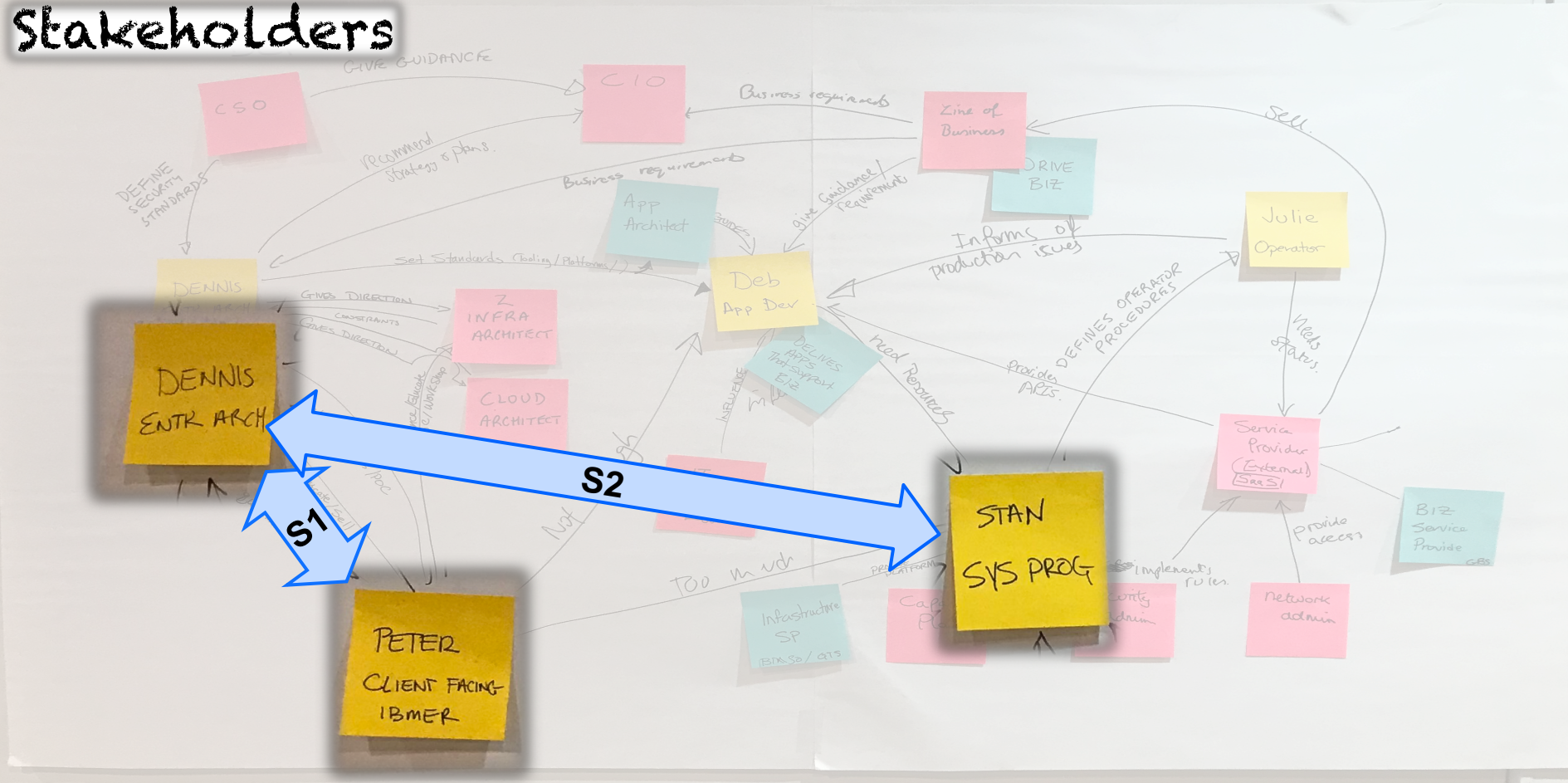
Agenda

- Understanding the Opportunity
- Seeing is Believing
- Learning from Experience

STAKEHOLDERS



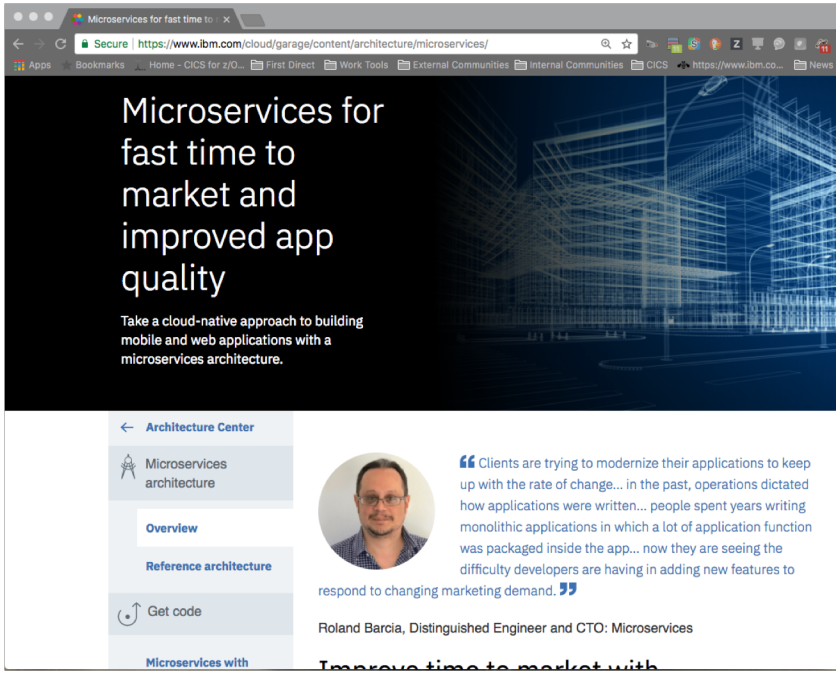
Stakeholders



Understanding the Opportunity

So, you want to respond to business requirements more quickly?

- The services you provide are not seen as transforming to meet new business needs fast enough.
- Your existing application structure means it takes too long and carries too much risk to change fast.
- Your development process imposes unacceptable delays.
- Your teams are not accepting of agile practises and are not aligned or empowered to keep ahead of business needs



You've heard from companies which "need greater agility and scalability" are using Microservices to achieve "fast time to market and improved app quality" by using them to "prioritize the continuous delivery of single-purpose services".

But <u>no one believes</u> that your mainframe **culture** can be agile enough to match this competition.

Definition:

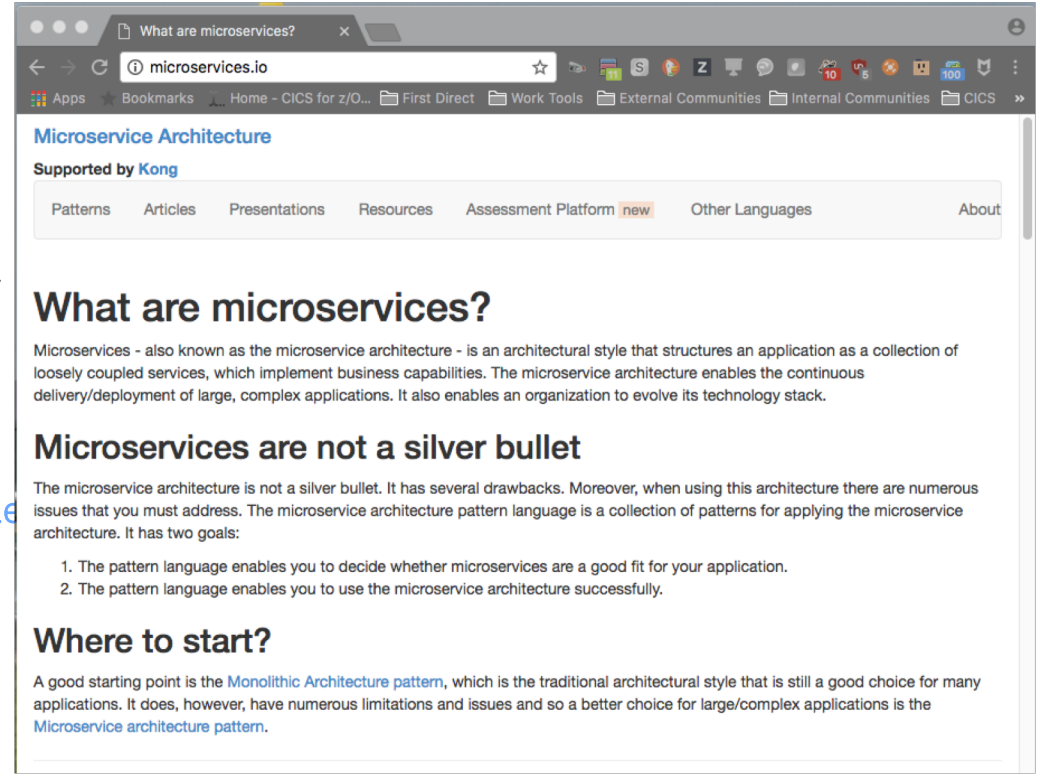
What…

Microservices is an architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities.

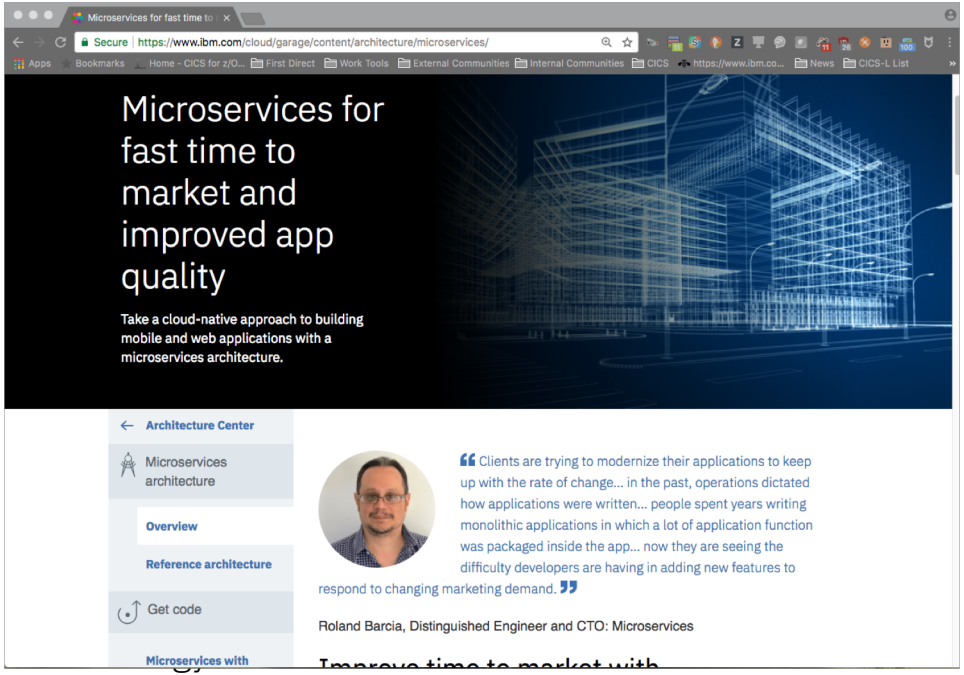
The microservice architecture enables the continuous delivery/deployment of large, comple applications.

It also enables an organization to evolve its technology stack.

From Chris Richardson (Microservice.io)



Definition:



From Chris Richardson (Microservice.io)

Why…

For fast time to market and improved app quality (IBM Cloud Garage Method).

To prioritize the continuous delivery of single-purpose services. Becoming popular with companies that need greater agility and scalability (Pivotal).

"balancing speed and safety at scale."

You need more knowledge about microservices: Thoughtworks

"Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations"
- Conway, 1967

https://www.thoughtworks.com/insights/microservices

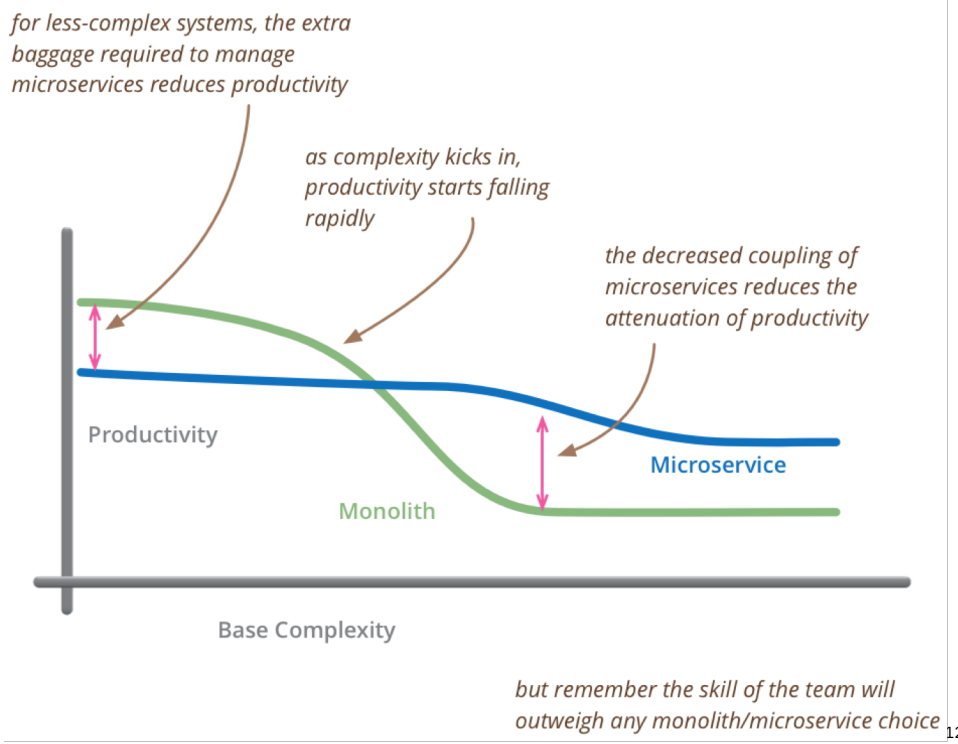
You need to understand when to adopt Microservices: the Microservices Premium

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

Contents

Sidebars

https://martinfowler.com/bliki/MicroservicePremium.html



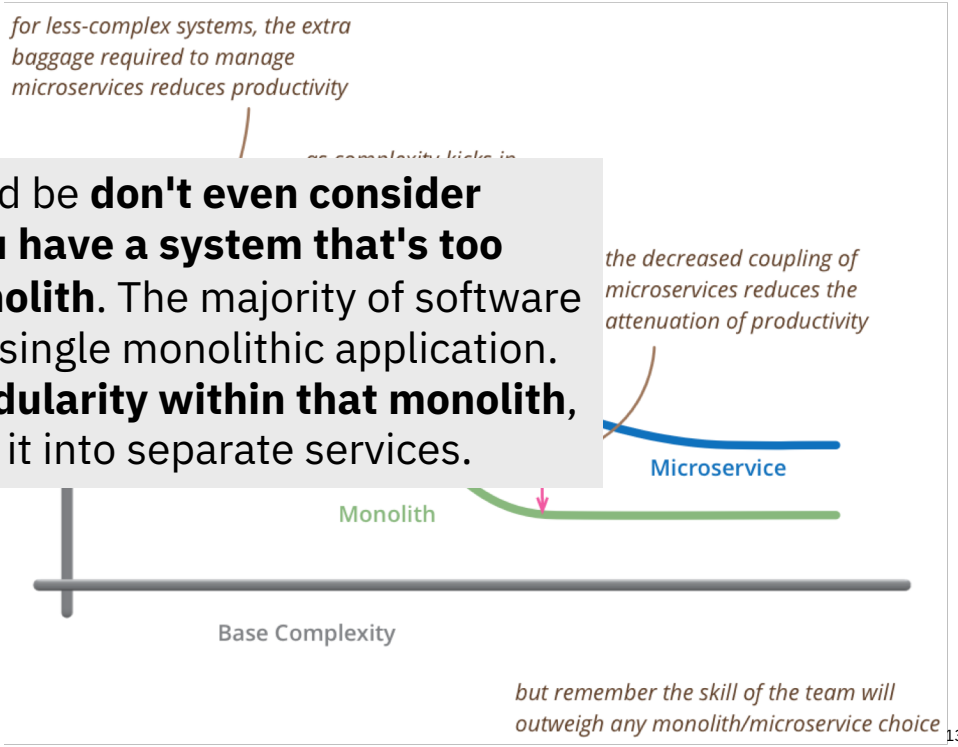
When to adopt Microservices: the Microservices Premium (Martin Fowler)

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

Contents

...primary guideline would be **don't even consider microservices unless you have a system that's too complex to manage as a monolith**. The majority of software systems should be built as a single monolithic application. **Do pay attention to good modularity within that monolith**, but don't try to separate it into separate services.

https://martinfowler.com/bliki/MicroservicePremium.html



Common misconceptions about microservices

~~"microservices is SOA done right"~~

~~"microservices is what we've always done, just smaller components"~~

~~"APIs are microservices"~~

~~"microservices are fine grained web services"~~

~~"microservices is a technical concept"~~

~~"we don't need to review our organizational structure to do microservices"~~

~~"microservices has no downside, it just enables greater agility and scalability"~~

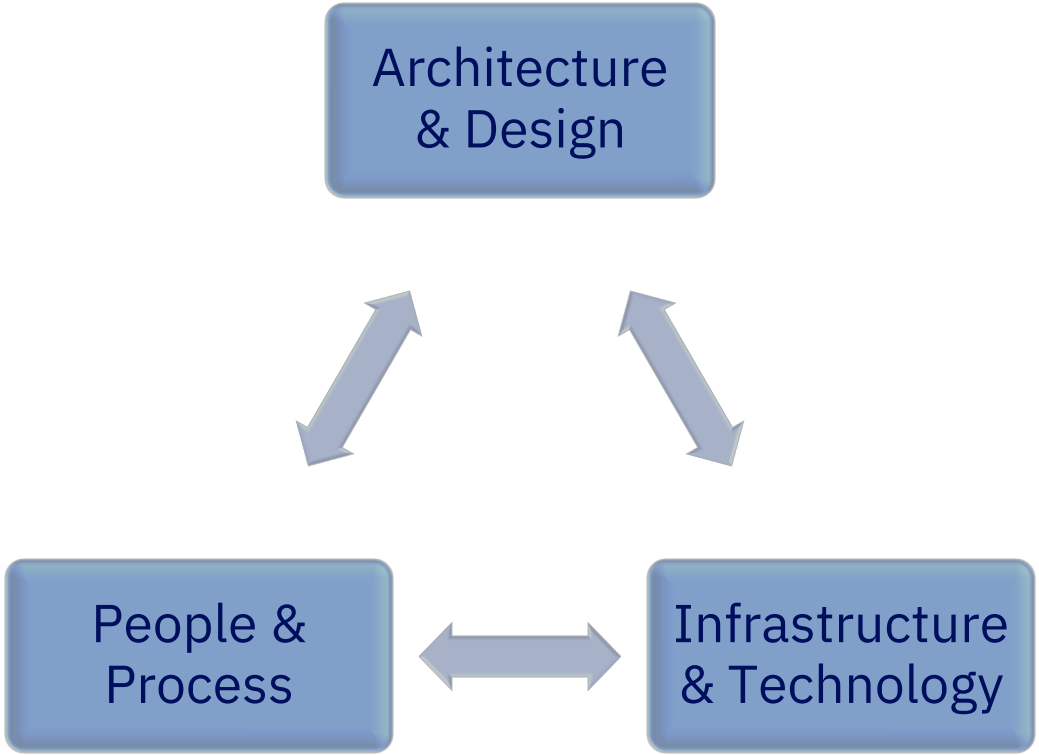
~~"everyone is doing microservices"~~

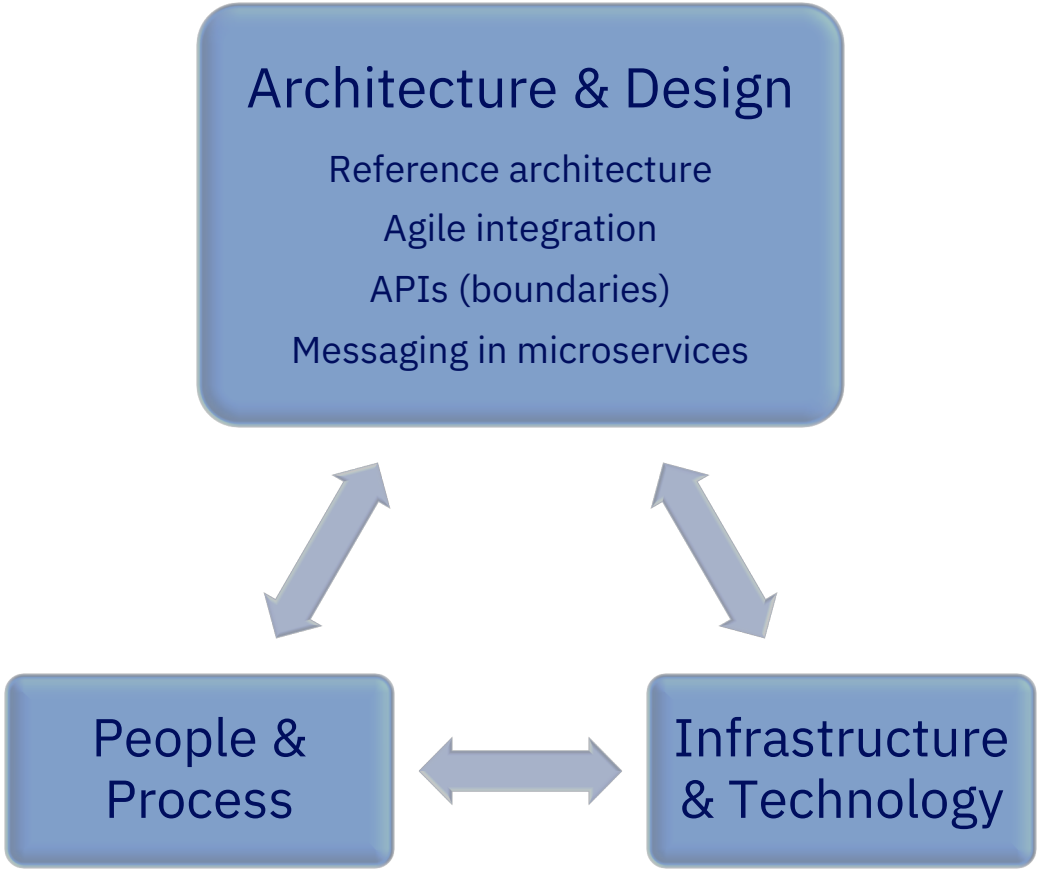
~~"we are going to refactor our 20 year old mainframe system into microservices"~~

~~"microservices is just about scalability"~~

~~"microservices means letting go of governance and control"~~

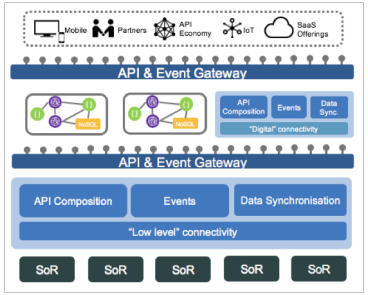
Core pillars (“dimensions of disruption!”)




Architecture & Design
Reference architecture
Agile integration
APIs (boundaries)
Messaging in microservices
People & Process
Infrastructure & Technology

Publically available material (typically publicized via https://developer.ibm.com/integration/blog)

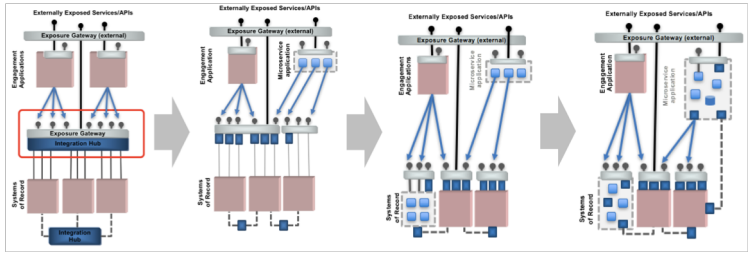
Hybrid Integration Reference Architecture



article (~15 pages) webinar (45 mins)

http://ibm.biz/HybridIntRefArch http://ibm.biz/HybridIntRefArchYouTube

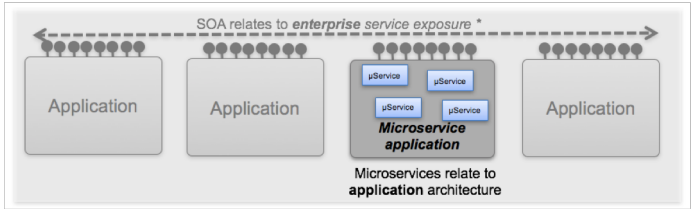
Moving from ESB to agile integration architecture



moving to agile integration architecture
http://ibm.biz/AgileIntegArchPaper

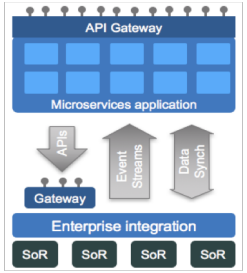
posts and articles
http://ibm.biz/AgileIntegArchLinks

Microservices vs SOA



short blog post http://ibm.biz/MicroservicesVsSoaBlog,
video (10 mins) http://ibm.biz/MicroservicesVsSoaVideoShort
paper (~15 pages) http://ibm.biz/MicroservicesVsSoa
webinar (55 mins) http://ibm.biz/MicroservicesVsSoaFullWebinar

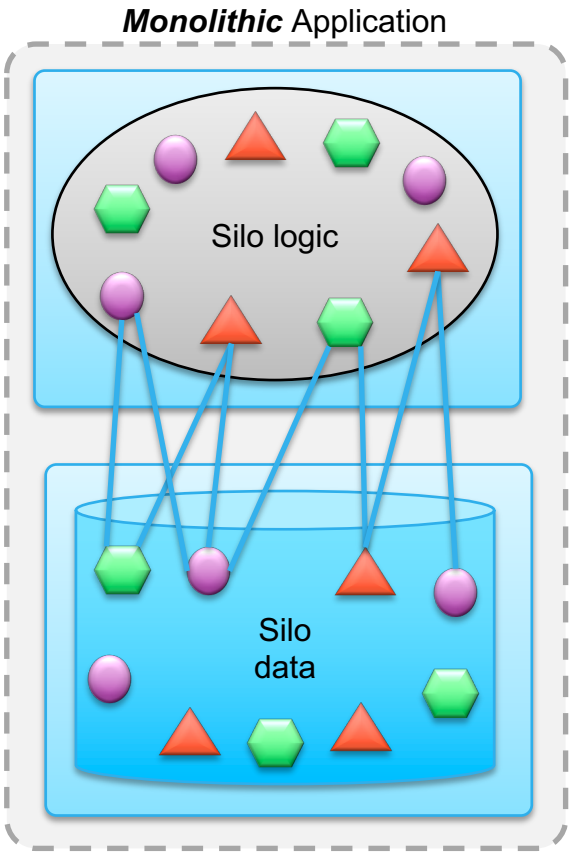
Microservices and messaging



webninar (20 mins)
http://ibm.biz/MicroservicesAndMessagingWebinar

Note: The importance and positioning of **API management** is discussed in **all** of the above

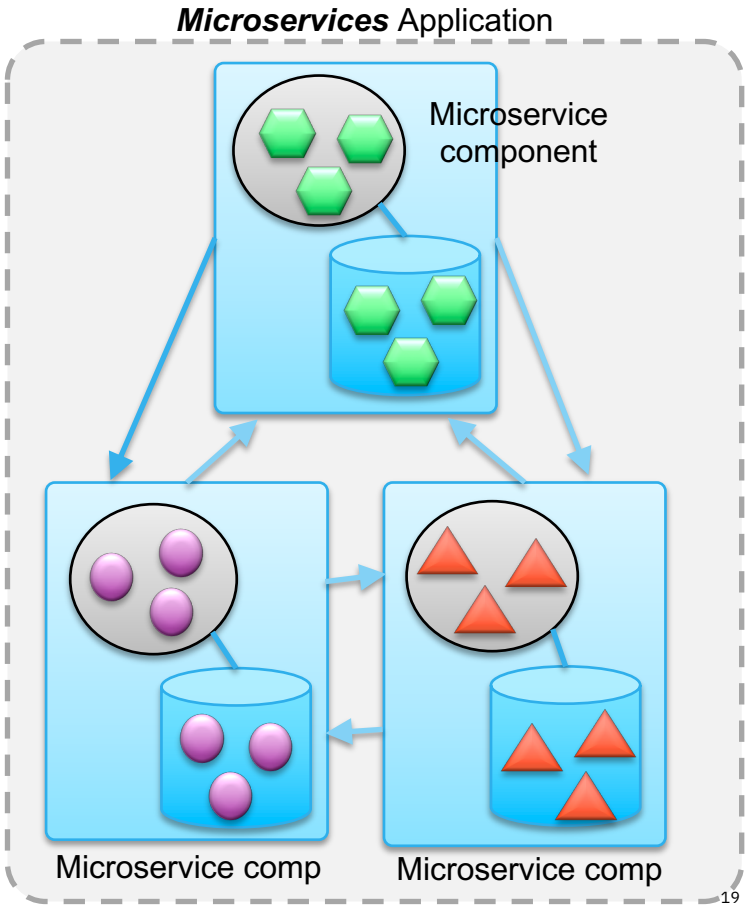
What *is* microservices architecture?



A set of principles that guide the breaking down of large components into smaller, more self contained ones enabling greater agility, more elastic scalability and and more differential resilience.

Fundamental principles include maximizing isolation and decoupling between components (including their data), making components as stateless as possible and avoiding affinities.

Supporting requirements include fully automated DevOps pipelines and good automated test coverage.



Putting Microservices in context

Microservices = APIs

Microservices demand APIs.

An API means the service can be consumed and managed more easily.

Microservices = APIs + DevOps

Microservices demand a combination of APIs and DevOps

A fast and reliable DevOps cycle means the service can continue to meet the business needs whilst maintaining high quality of delivered production instances.

Microservices = (APIs + DevOps) x Innovation

Microservices demand a combination of APIs and DevOps where Innovation is rewarded.

Business need for innovative use of the services drives the demand.

Microservices = (APIs + DevOps) x Innovation x Agile Culture

Microservices demand a combination of APIs and DevOps where Innovation is rewarded within an Agile Culture.

Without an agile culture the services will quickly go stale.

Our chief weapon is surprise...surprise and fear...fear and surprise.... Our two weapons are fear and surprise...and ruthless efficiency.... Our *three* weapons are fear, surprise, and ruthless efficiency...and an almost fanatical devotion to the Pope.... Our *four*...no... *Amongst* our weapons.... Amongst our weaponry...are such elements as fear, surprise....

Questions

Time for questions to refine that objective:

- Does your organisation have APIs?
- Does you organisation have Microservices?
- Do you believe Microservices and APIs are the same?
- Are Microservices a cloud-only architecture?
- Are you creating, services, APIs or both?
- Does this impact your business?
- How does this impact your business? Speed/Scale/Quality/other
- Is this natural for your organisation now? eg Fowler - products not projects.
- What (who?) are the inhibitors which will make this difficult?
- Can you initiate the necessary changes?

Seeing is believing

Organizations Are Adopting a Multi-Cloud Strategy

8 out of 10 committing to Multi-Cloud
71% use 3 or more clouds

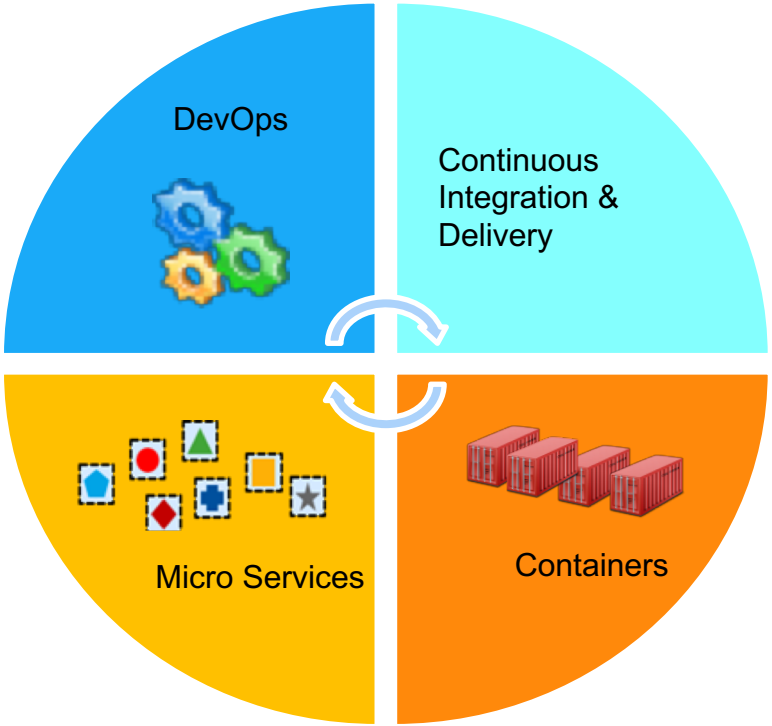
+

Getting new value from third parties

Extracting value from your entire business

Do you have a multi cloud strategy. Who are your vendors?

Evolution of how workloads are built & delivered



By 2018,

Over **60%** of **New Apps** will use cloud-enabled **continuous delivery** and **cloud-native application**

architectures to enable faster innovation and business agility.

(***IDC Prediction***)

Do you have a PAAS strategy?

Does your company use containers and orchestration tech? which? Are you doing cloud native?

Key Use Cases Driving “Private Cloud” Adoption

1

Create new cloud-native applications

2

Modernize and optimize existing applications

3

Opening up enterprise data centers to work with cloud services

Multi-cloud management and orchestration

What are your adoption patterns are you seeing in private cloud?

Solution Overview – IBM Cloud Private

… to enable enterprises to both innovate & optimize

Enterprise Content Catalog
Open Source and IBM Middleware, Data, Analytics, and AI Software

Core Operational Services
Log Management, Monitoring, Security, Alerting

Kubernetes Container Orchestration Platform

kubernetes

docker

Choose your infrastructure:

vmware

intel

openstack.

Power Systems

IBM Spectrum

IBM Z

What workloads in production you run on containers if any?

Are you already using another vendor here? (openshift, docker EE etc)

Strategic Value:

Self-service catalog

Agility, scalability, and elasticity

Self-healing

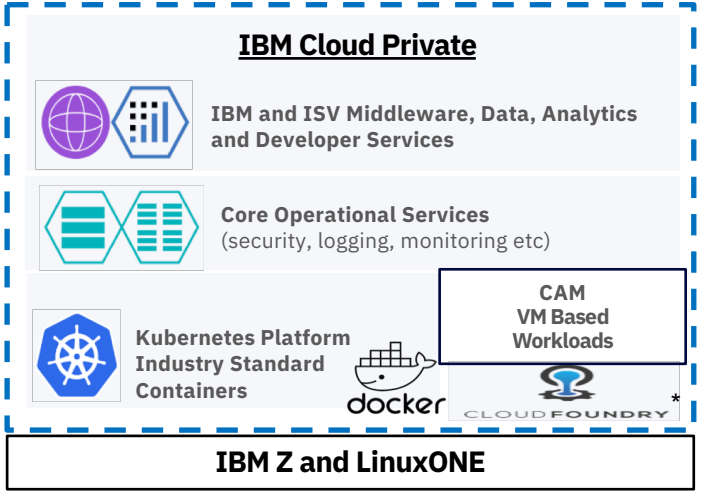
Enterprise security

No vendor lock-in

Introducing IBM Cloud Private on Z

Benefits on Z

- Modernization and **Digital Transformation Speed**.
- Highest levels of **Security**
- **Only private cloud offering that can support IBM Z**



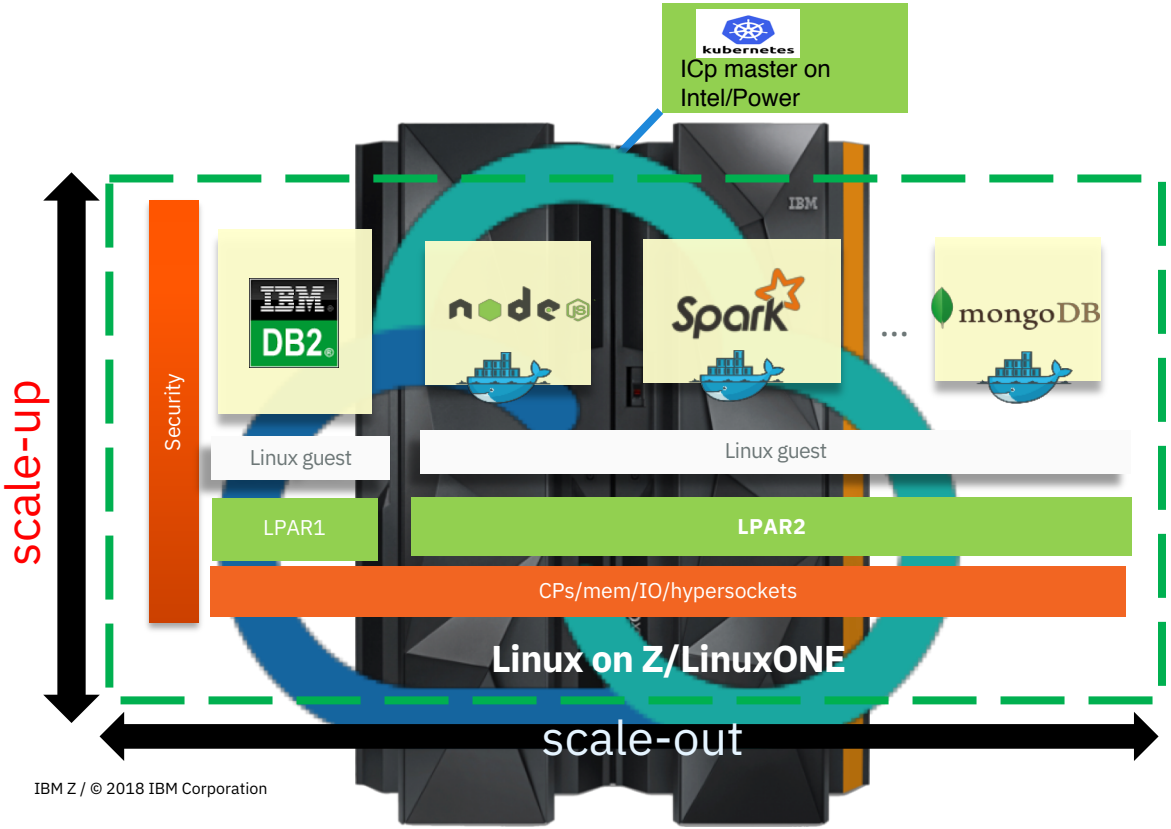
Adoption Patterns

- **Application Modernization** with IBM Middleware and Container content.
- **Digital Transformation** with Z/OS*
- **Cloud Native Services** with Hyper Protect Containers and Runtimes

Enterprise grade **Open by design** **Secured by IBM Z**

LinuxONE: Scalable, highly-available and secure private cloud

Compose high-performance scalable applications. Dynamically and seamlessly re-allocate resources. Provide right-time analytics and powerful engagement



Extreme Virtualization and Scale

- 1k Linux guests/hypervisor
  - 85 EAL5+ zVM/KVM per box
- +2 million docker containers
- 17TB Mongo instance
- Hypervisor communication via fast, in-memory TCP/IP Hipersockets or Shared-OSA
  - 5x less latency than discrete servers
- Massive dedicated I/O
  - 640 Power co-processors
- 3GB combined cache, 5Ghz CPUs, crypto acceleration

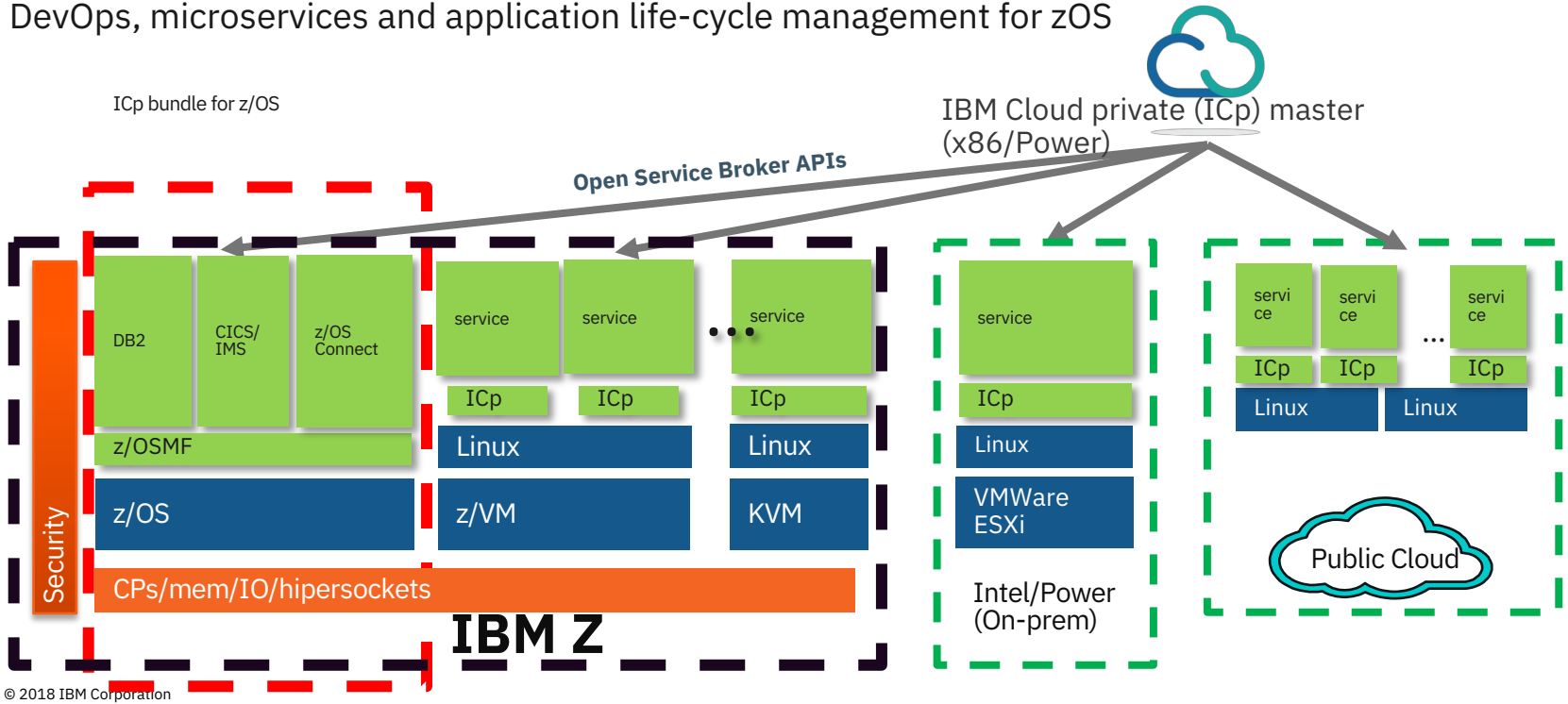
Super Elastic System

- Combine scale-up and scale-out
- Non-disruptively add/remove resources from Linux guests
- Non-disruptively add/remove Linux guests

Digital Transformation Inclusive of z/OS

IBM Z as a differentiating asset in ICP from services that span z/OS, Linux on Z, private and public cloud

- Cloud consumption for z/OS (DBz-aaS, WASz-aaS, MQ-aaS, CICS-aaS etc)
- DevOps, microservices and application life-cycle management for zOS



Learning from Experience

The May GMAC asked for a consolidated view of the current state of agile application projects across **Fiducia /GAD, ADP, and American Express** as sponsor clients.

Workshops were conducted across multiple clients with the aim to understand working approaches to application agility on the Z platform and distill practical guidance which can be published to the client community.

By exploring this topic in this way, IBM will be better able to make recommendations and supply the right tools to improve the developers' user experience and the enterprise's agility on the Z platform.

What we heard

What's working…

"We can do things quickly!"

- Some example of different development models
- Examples of fast production deployments in 2 week sprints (or faster) do exist.
- Agile "whole" team practices assisted by organizational adjustments. "*The Band-Aid was ripped off*"
- Semi-automated deployment pipelines
- Skills are mixing across Z and the enterprise
- Examples of Ops in scrum team
- Teams owning a product for full lifecycle
- Global "production like" test environments exist
- Access to Dev-Test pricing containers barriers to development testing.
- API's are prolific

Pain Points & Gaps

- Tooling inconsistent and unfamiliar across the enterprise
- Parallel development restricted by not using incumbent tooling
- Lack of interlock between development and operations team
- Uncertainty in change quality leading to risk aversion
- Uncertainty on the cost impact of changes
- Limited knowledge of production impact by developers.
- Impact of deployments to operations unclear
- Minimal automated testing, consistent testing environments, access to stable testing environments for both applications and data
- Minimal access to new technology when they want it
- Continuity of support for deployed applications

Insights

Organization

Break down traditional boundaries and unite the teams across platforms

Get everyone involved

Identify a champion to lead the change
Involve all personas in all phases of development

Cross training

Teach Mainframe folks Java and Container skills.

Teach Distributed folks COBOL and Mainframe skills

One team

All squads have both Mainframe and Distributed skills on the team

Align organization to products

Reorganize the organization to align with Products instead of projects with a one team approach

Testing

Code, build, release and deploy with confidence

For developers

We need private environments for Unit Test, with *FAST* turnaround in the edit, compile, and debug cycle.

Fed by data

An efficient supply of high quality, anonymized data is essential for effective automated testing.

Automated

We have too many people running tests when they should be designing and implementing automated tests.

Repeatable

Regression testing is essential to build confidence that cycle-times can be reduced.

Release Pipeline

A common release pipeline using open source on all platforms.

Common pipeline

One pipeline and practices for mainframe and distributed.

Open source

Use open source tools to build that pipeline. Aspiration that release pipeline open source packages run on the mainframe.

Automate everything

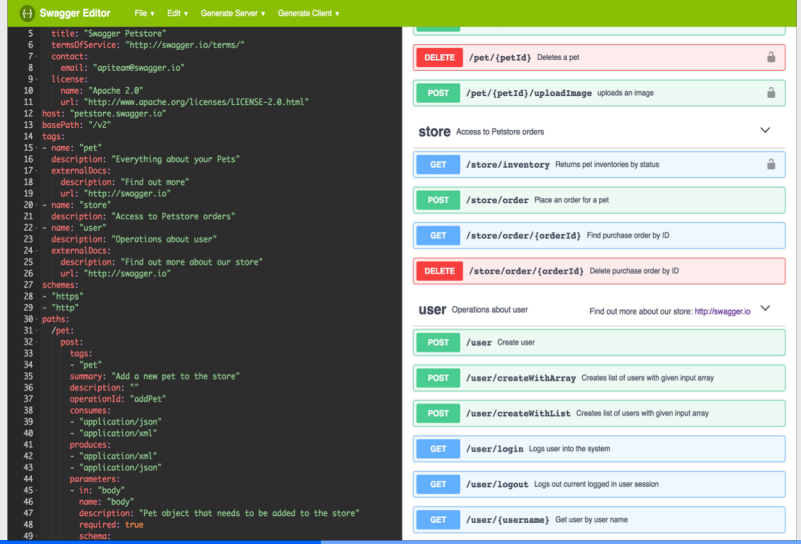
Minimize the need for manual steps through automation. Everyone is using the same automation for build, test, and deployment.

Frequent

Squads need frequent feedback from working code which requires frequent quality builds

APIs and Refactoring

Realizing the value of current assets and ensuring it endures



Consumability

APIs are the face of your service. Designing APIs which delight the developer/consumer unlocks innovation.

Decomposition

Modeling and exposing a monolith via API is the first step to modernize it. Decomposing enables the asset to align with the new organization.

Evolution

A more fine grained set of assets is faster and easier to change/version, test and deploy.

Curating

Today's monoliths are difficult to evolve and tools are needed. Eg. measuring and eliminating dead code and its overheads.

Runtimes

Freedom to put any content on the platform

Don’t replace

Instead of rebuilding everything on mainframe be strategic and rewrite in a platform agnostic way

Exploit platform isolation

Run applications in isolation with middleware, data, and monitoring.

Developer self service

Allow developers to allocate and create containers with middleware and anonymize test data

Freedom to run

Write workloads in a platform agnostic manner and create a governance model to determine run platform

Next steps

Common Practice

Testing	Good enough? • Too manual, cumbersome to provision, labor-intensive, non-repeatable, often neglected, too expensive…
Pipeline	Proprietary and bespoke • Inflexible SCM, bespoke/homegrown automation, manually driven, …
APIs and Refactoring	API exposure of existing applications • Manual identification or historical knowledge • Refactoring of “edges” to API enable
Containers	Some container like characteristics: • Isolation, WLM, Security Multiple languages: COBOL, PLI, Java, Node.js Missing packaging and standardization and deployment

Possible Practice

Testing	Quickly and reliably provision isolated and representative test environments with apps and data
Pipeline	Automated pipeline GIT/RTC, Jenkins, DBB allowing parallel development
APIs and Refactoring	Identification of API candidates, business rule discovery and source identification for refactoring or extending (in new languages) Code coverage, dead code identification
Containers	Full standard experience on Linux on Z Platform agnostic workloads with Java and Node.js (LoZ and zOS)

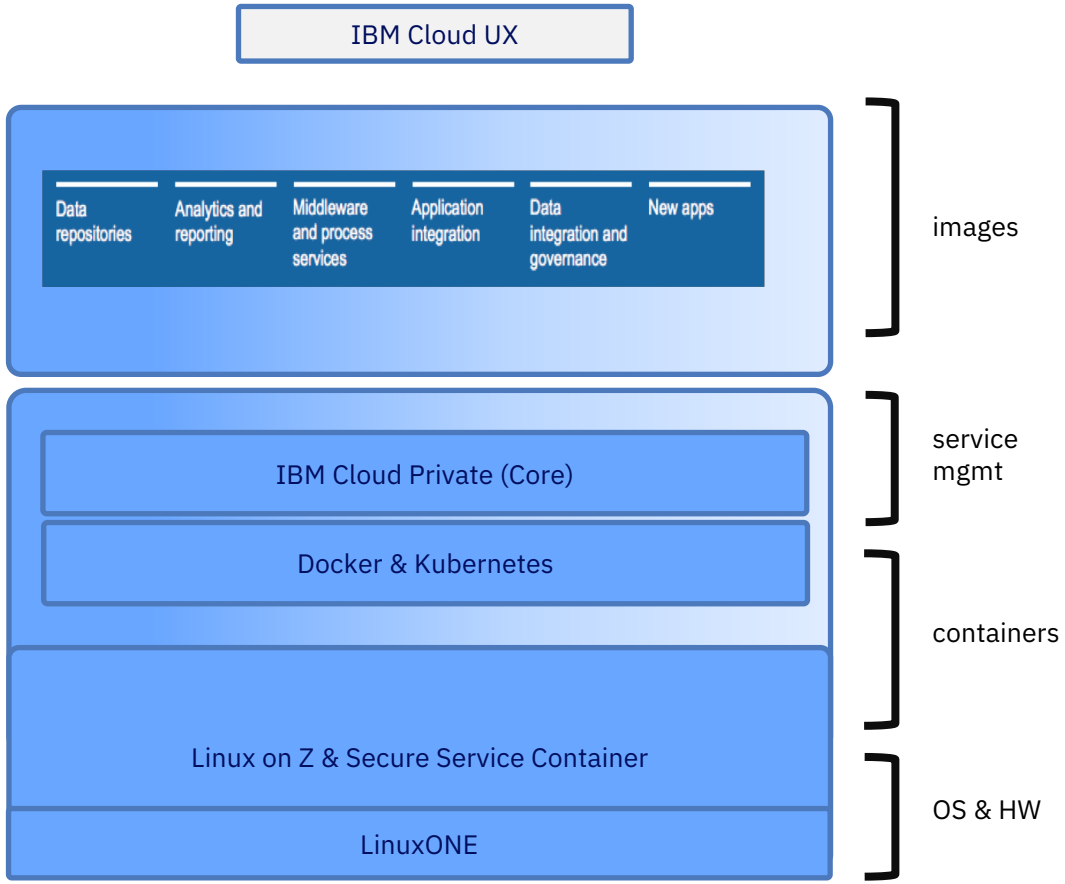
Future Practice

Testing	Auto provisioned, monitored, integrated end to end, validated quality gates governing pipeline.
Pipeline	For developers “one click” provision and deploy Integrate opensource pipeline tools
APIs and Refactoring	Developers understanding the impact of app changes before deployment Developer experience consistent across the whole enterprise
Containers	What’s next for zOS?

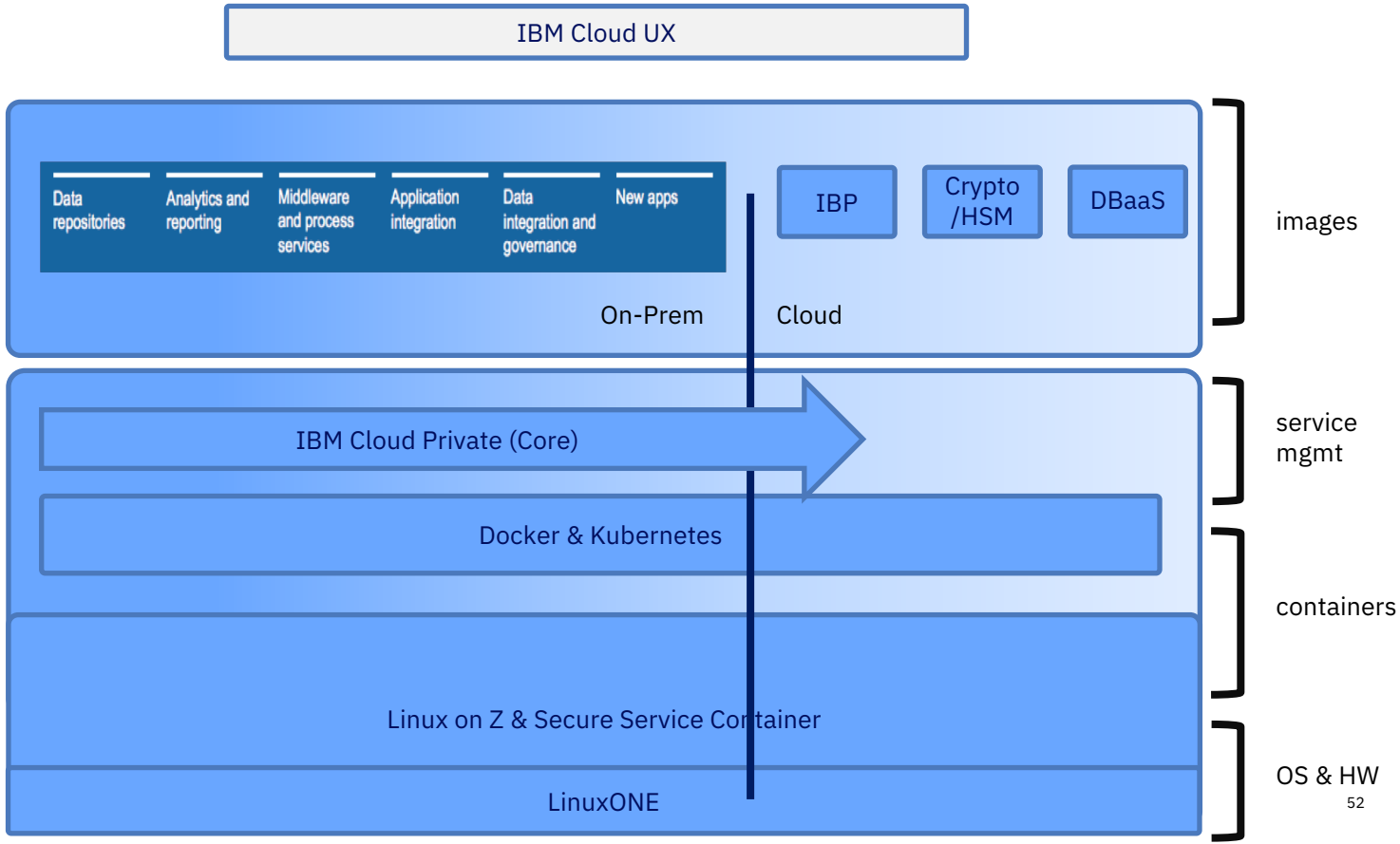
Actions For Us All

- Align teams including cross organization members: development, operations, cloud, mainframe…
- Implement possible practices in your shop today
- Refine future roadmap with sponsor-users across:
  - Testing
  - Release Pipeline
  - APIs and Refactoring
  - Runtimes

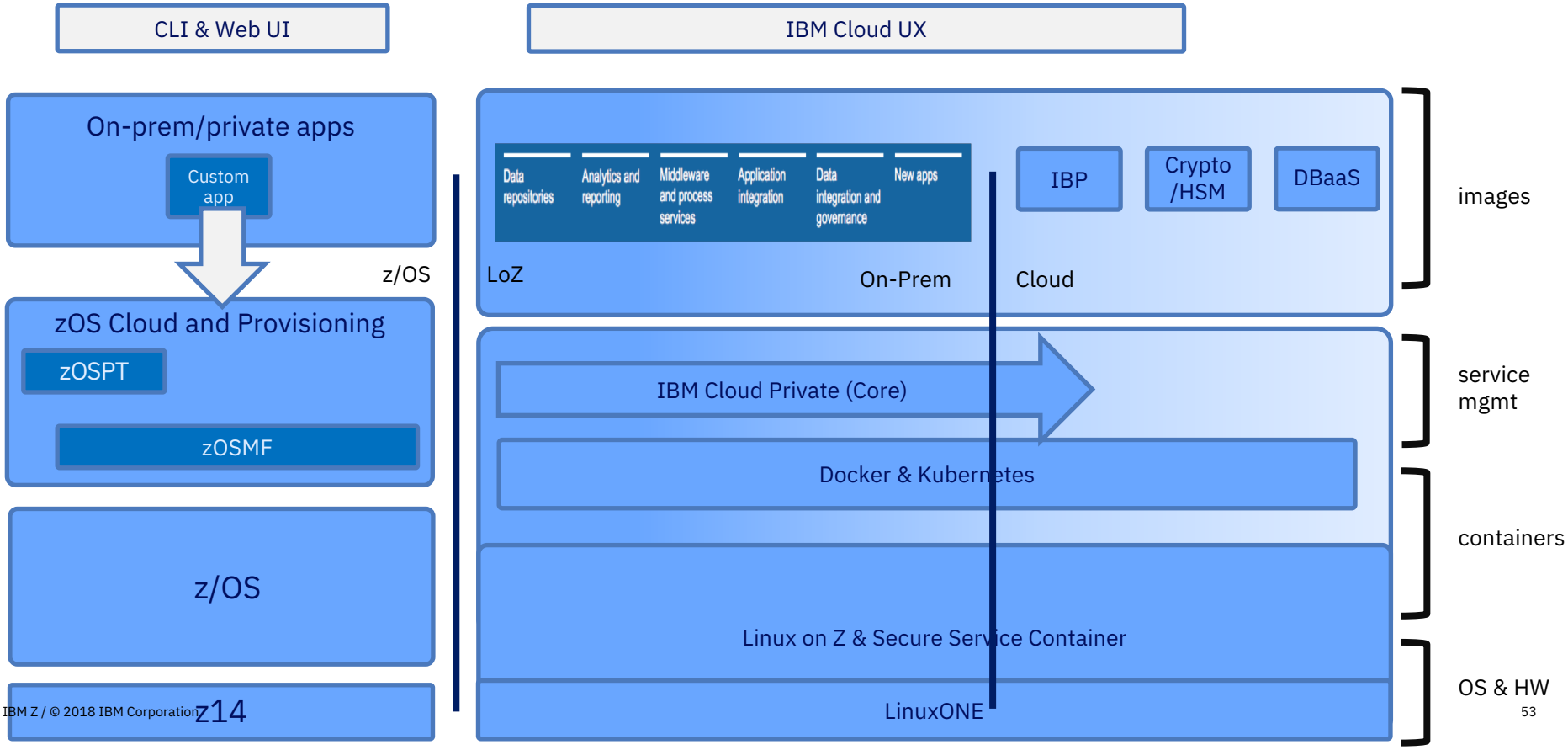
Our landscape:
Coherency of experience



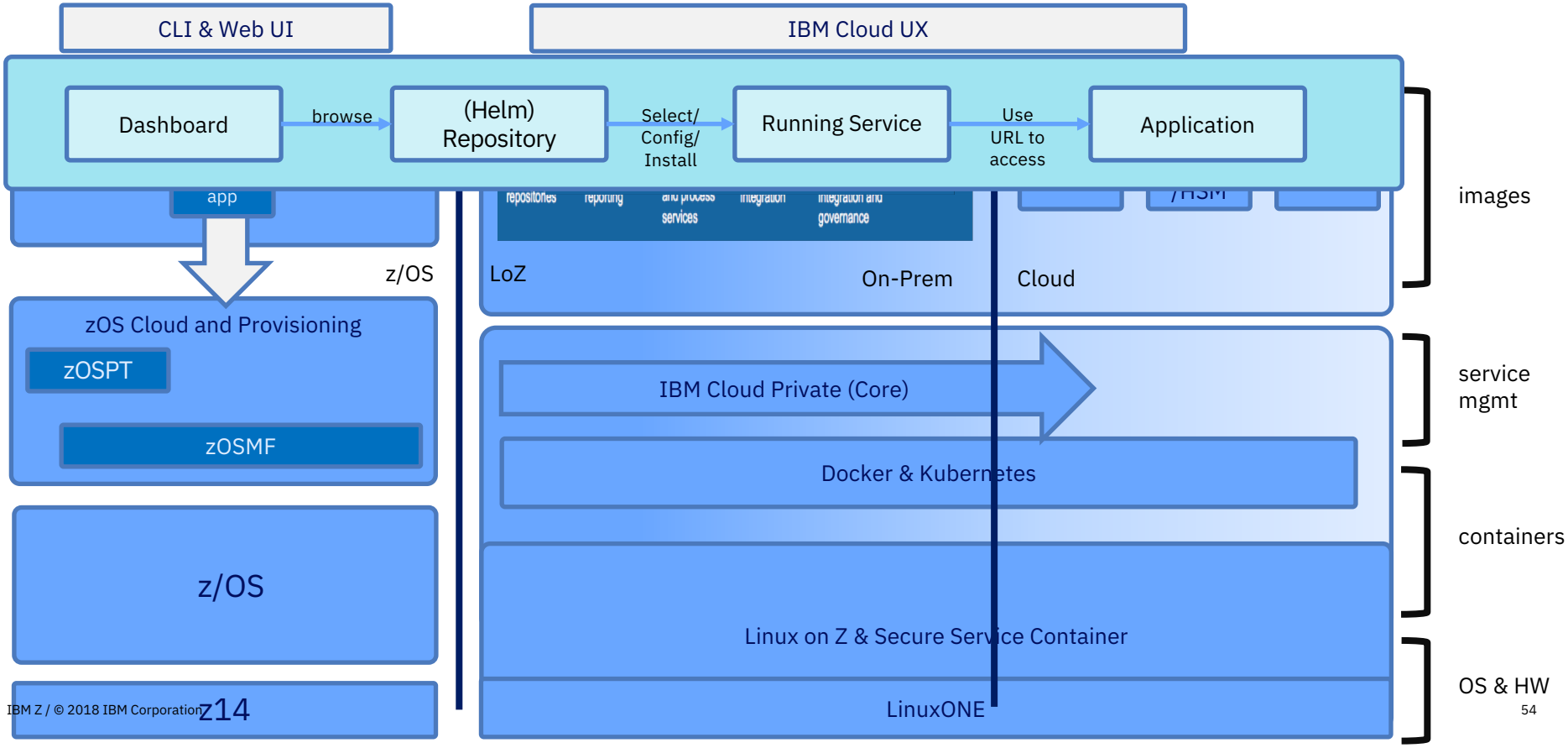
Our landscape:
Coherency of experience



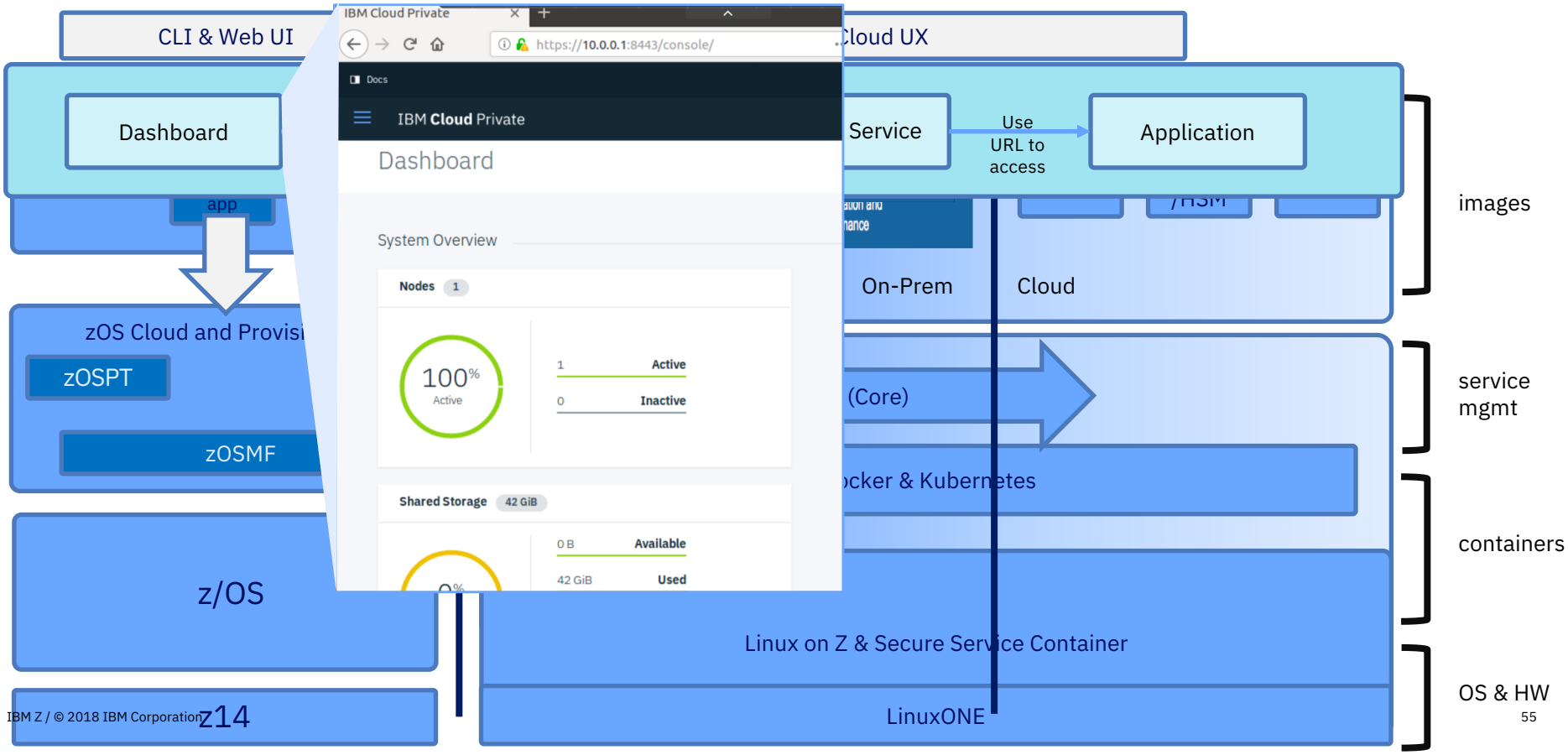
Our landscape:
Coherency of experience



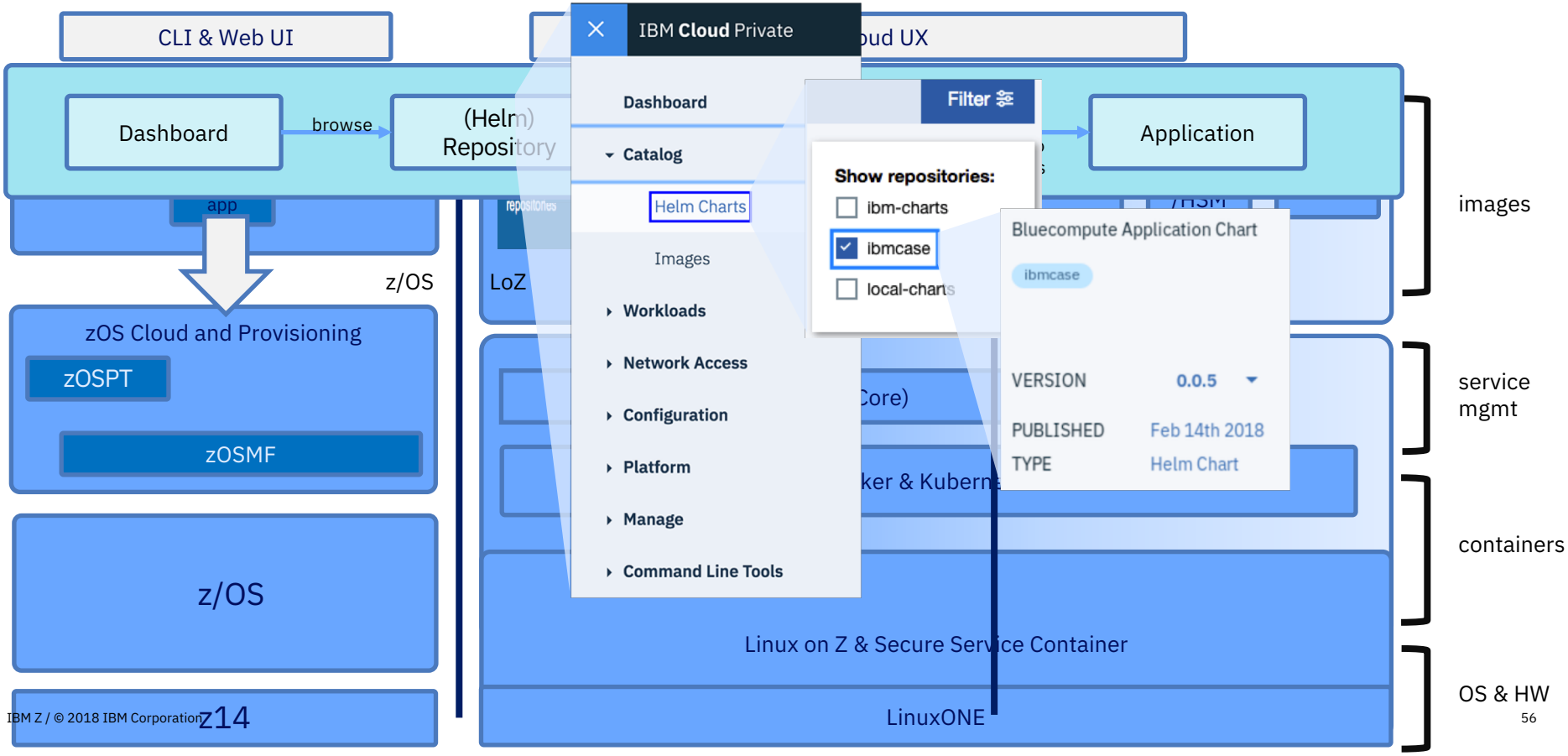
Our landscape:
Coherency of experience



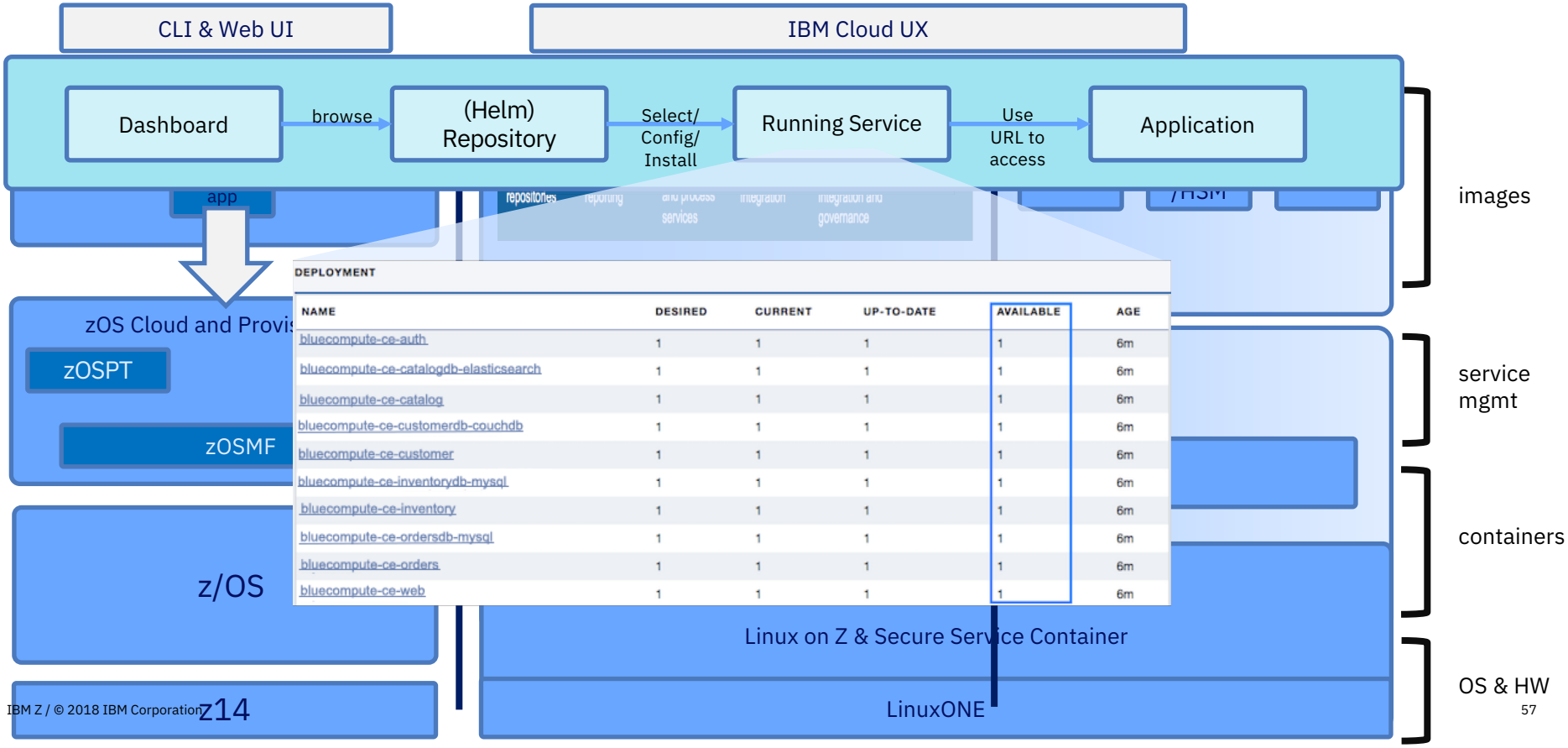
Our landscape:
Coherency of experience



Our landscape:
Coherency of experience



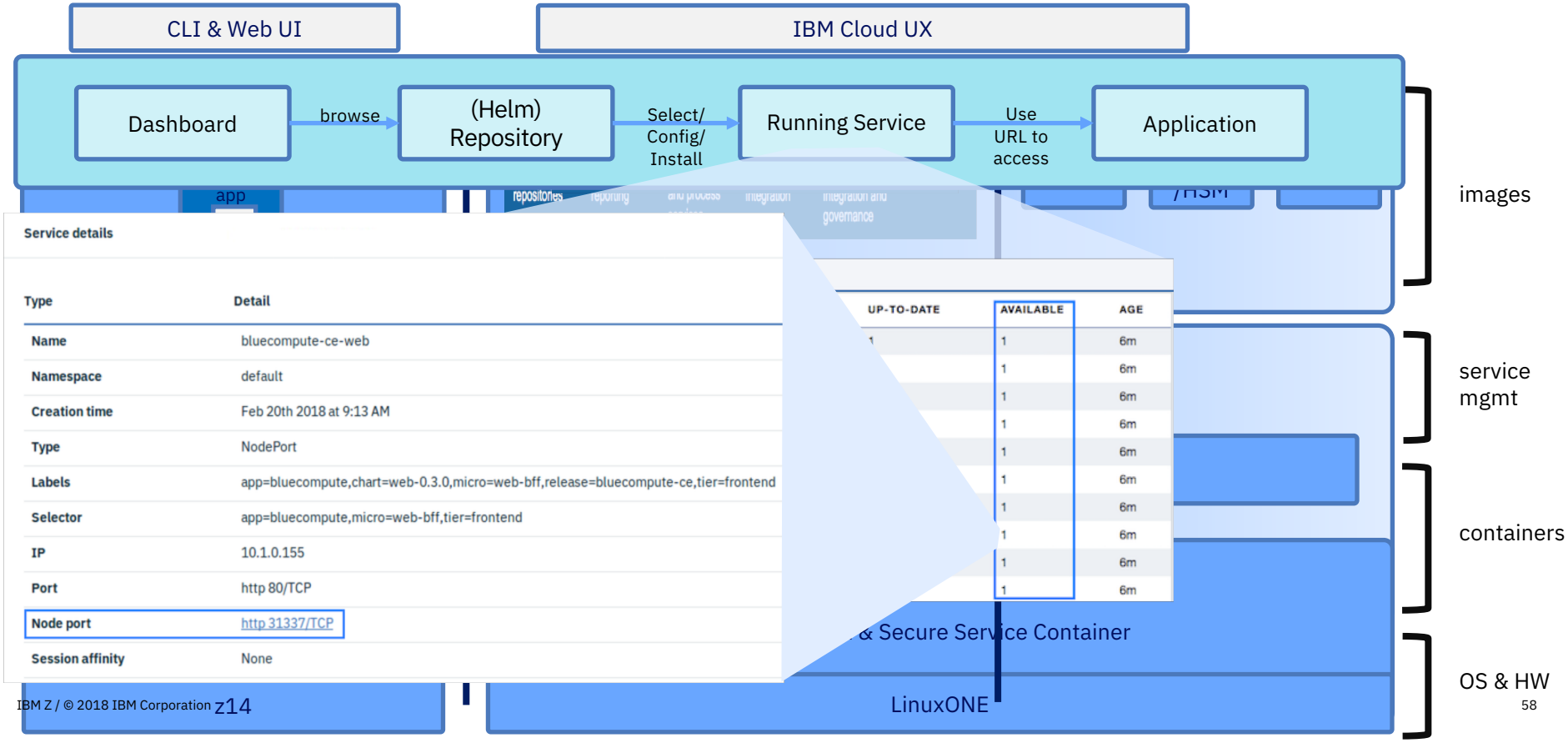
Our landscape:
Coherency of experience



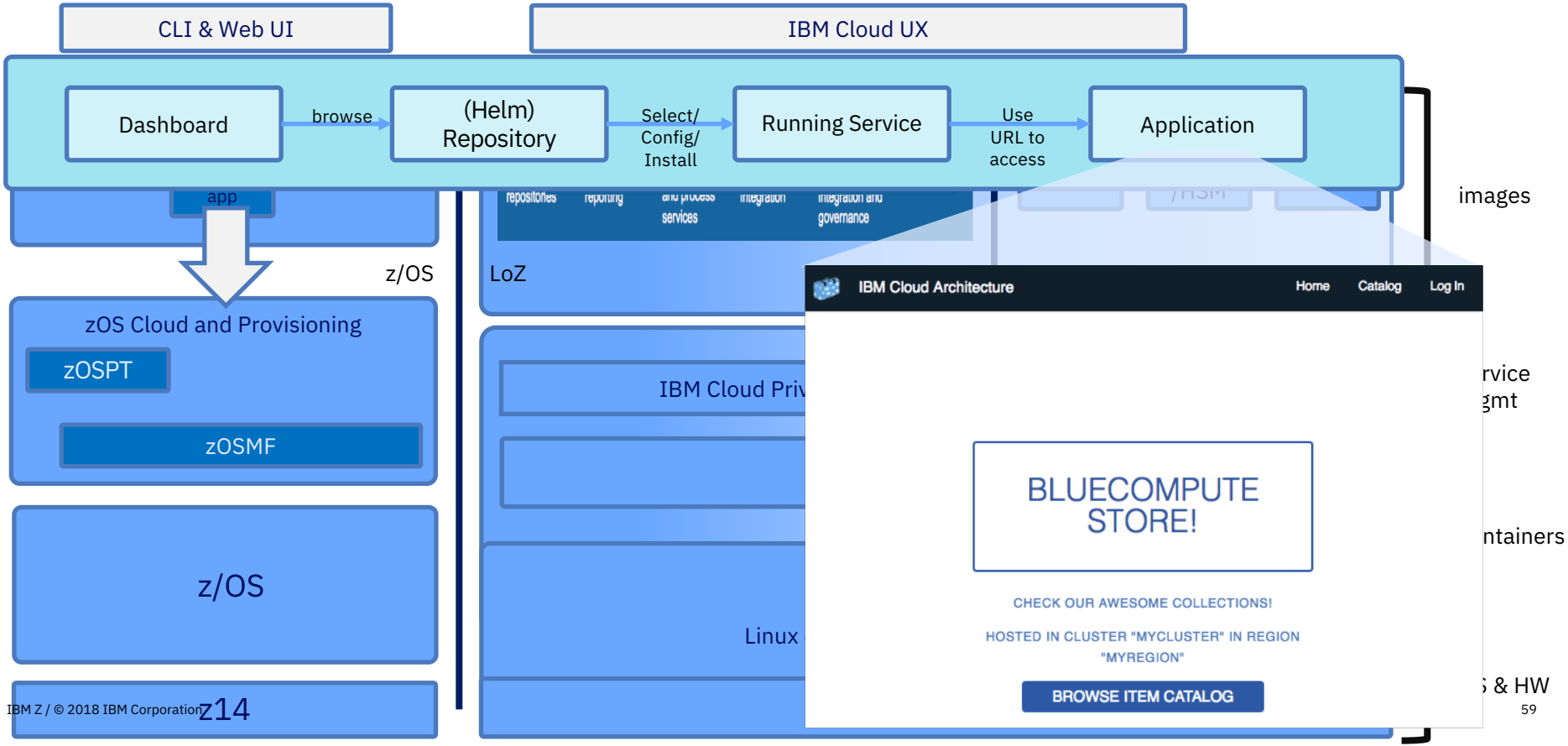
DEPLOYMENT

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
bluecompute-ce-auth	1	1	1	1	6m
bluecompute-ce-catalogdb-elasticsearch	1	1	1	1	6m
bluecompute-ce-catalog	1	1	1	1	6m
bluecompute-ce-customerdb-couchdb	1	1	1	1	6m
bluecompute-ce-customer	1	1	1	1	6m
bluecompute-ce-inventorydb-mysql	1	1	1	1	6m
bluecompute-ce-inventory	1	1	1	1	6m
bluecompute-ce-ordersdb-mysql	1	1	1	1	6m
bluecompute-ce-orders	1	1	1	1	6m
bluecompute-ce-web	1	1	1	1	6m

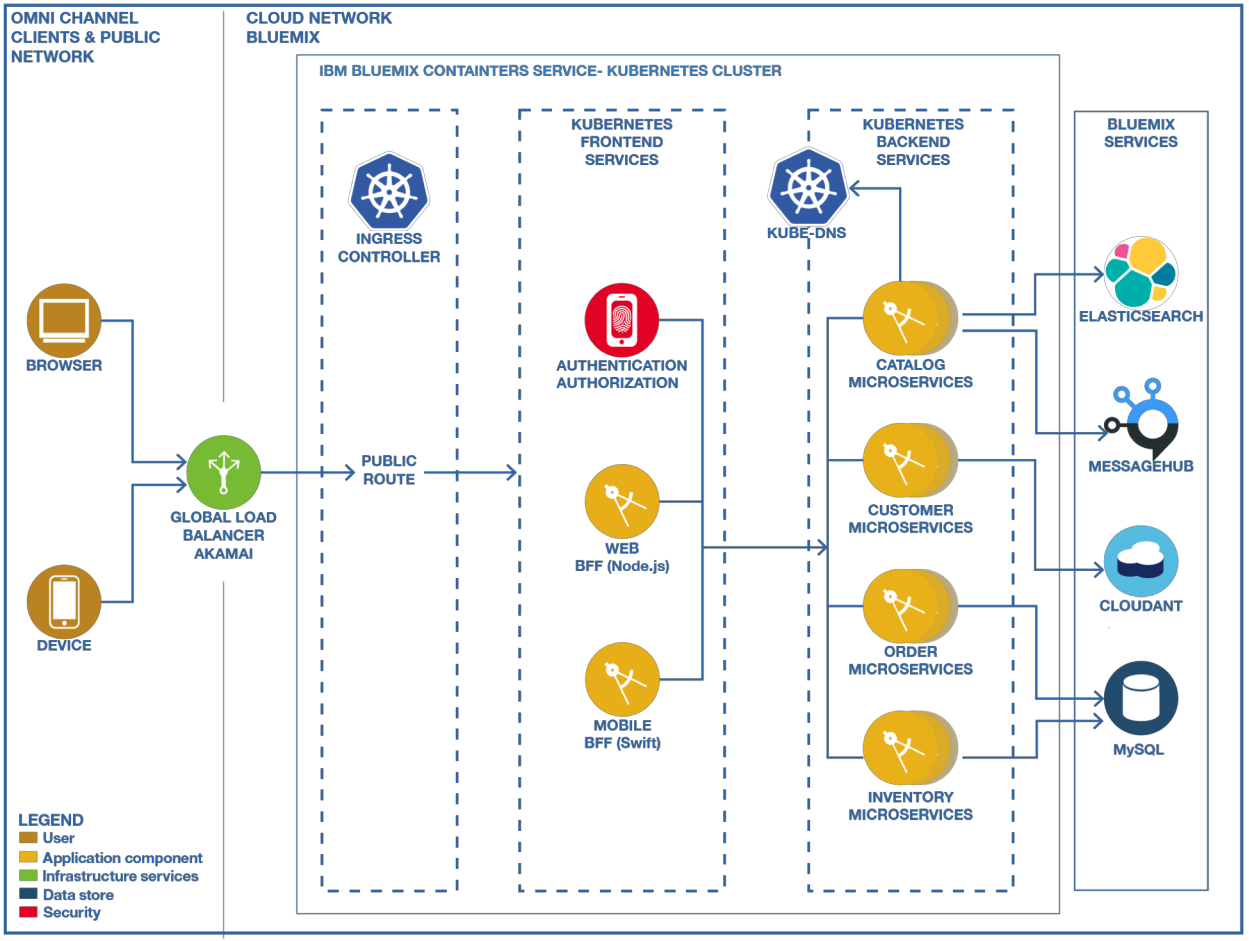
Our landscape:
Coherency of experience



Our landscape:
Coherency of experience



IBM Cloud Tutorial Architecture



Other views of Microservices

The 12 Factor App (https://12factor.net/)

The Twelve-Factor App

Introduction

In the modern era, software is commonly delivered as a service: called *web apps*, or *software-as-a-service*. The twelve-factor app is a methodology for building software-as-a-service apps that:

- Use **declarative** formats for setup automation, to minimize time and cost for new developers joining the project;
- Have a **clean contract** with the underlying operating system, offering **maximum portability** between execution environments;
- Are suitable for **deployment** on modern **cloud platforms**, obviating the need for servers and systems administration;
- **Minimize divergence** between development and production, enabling **continuous deployment** for maximum agility;
- And can **scale up** without significant changes to tooling, architecture, or development practices.

The twelve-factor methodology can be applied to apps written in any programming language, and which use any combination of backing services (database, queue, memory cache, etc).

	Factor	Z PoV
I	**One codebase tracked in revision control, many deploys**	*“If there are multiple codebases, it’s not an app – it’s a distributed system. Each component in a distributed system is an app, and each can individually comply with twelve-factor.”* This is saying 12 Factor apps are unitary.
II	**Explicitly declare and isolate dependencies**	This factor concerns modularity, reuse and dependency expression. The example of the value of this factor is one of reliable build and deploy
III	**Strict separation of config from code**	*”everything that is likely to vary between deploys (staging, production, developer environments, etc)”*

	Factor	Z PoV
IV	**Backing Services**	This is perhaps the most important of the 12 Factors which would need to be understood thoroughly to successfully apply any of these principles to z/OS applications. This factor demands a structural separation between the specific implementation of the apps unique value from common underlying services such as databases, messaging services, email providers or caching services. The principle here is that a 12 factor app will *bind* to specific underlying service providers
V	**Build, release, run**	*“Strictly separate build and run stages”* What’s not to like?

	Factor	Z PoV
VI	**Processes**	A statement about the execution environment of the app and it’s more or less just a version of the sysplex principle that an app should not have any affinities with the multiple servers requests to it might get dispatched to.
VII	**Port Binding**	The assumption here is that the app is providing a service which can be bound to by a client requesting the service, so it had better be able to bind to the externally available transport resources required.
VIII	**Concurrency**	WLM, Threadsafety and elimination of affinities again.

	Factor	Z PoV
IX	**Disposability**	**Resilience** Deployed instances of the app need to tolerate management and unmanaged termination. Managed disposibility enables the scaling down as well as up, and unmanaged disposability means when accidents happen it's not disruptive (either a quick restart is sufficient if the service instance was actually a SPOF, or the workload was maintained on surviving instances).
X	**Dev/Prod Parity**	All about the speed of the DevOps cycle - minimizing the feedback time. There's some appeal to the loose coupling from **IV. Backing Services** to make developers' lives easier
XI	**Logs**	This is actually just all about diagnostics.
XII	**Admin Processes**	Developers and operators need similar abilities to perform admin functions against the service.