



Cloud Native Solution Design

OVERVIEW OF CLOUD COMPUTING

Dr Venkat Ramanathan

rvenkat@nus.edu.sg

Institute of Systems Science

National University of Singapore

© 2009-23 NUS. The contents contained in this document may not be reproduced in any form or by any means, without the written permission of ISS, NUS, other than for the purpose for which it has been supplied.

Total Slides: 44



Objective

- Upon completion of this module, you should be able to
 - Define what is Cloud Computing
 - Understand the benefits & motivations for Cloud Computing
 - Understand the essential characteristics of Cloud Computing
 - Understand the service delivery models and deployment models of Cloud Computing
 - Be aware of leading Cloud Computing service providers
 - Appreciate current industry developments in the Cloud Computing domain



Outline

- Cloud Computing Overview
- Definitions
- Motivations & Scenario
- Cloud Service Models
- Cloud Deployment models
- Industry developments

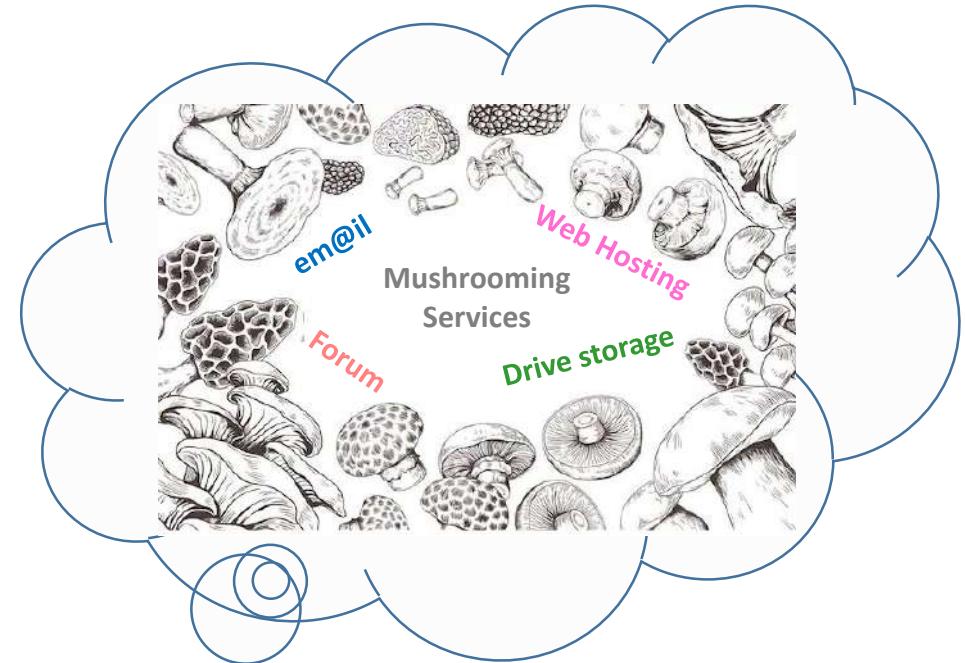


OVERVIEW



Evolution

- Genesis
 - Internet
 - SOA
 - Virtualisation
 - Distributed Computing (Grid)
- Philosophy
 - Rent rather than own the resources
 - Rent what you need
 - Rent when needed





Essential Characteristics: NIST

5 Essential Characteristics of Cloud Computing

Ref: The NIST Definition of Cloud Computing

<http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>



On-demand
self-service

Ubiquitous
network
access

Location
transparent
resource
pooling

Rapid
elasticity

Measured
service with
pay per use

Source: [NIST](#)



What we achieve through this?

- Economy of Scale, of course!
 - Standardized workloads (eg: VM or Containers) can be executed on highly integrated massively replicable infrastructure stack. No need to support wide workloads and heterogeneous stack of hardware, OS and middleware etc.
 - Hardware can be bought in bulk and configured to allow expansion.
 - Software can be stripped down, so that only what is required is loaded.
 - The software itself is written in cloud optimized way hence efficient.
 - There is no need to build your own virtual machines.



DEFINITIONS



Definition

Cloud Computing is:

- **Internet-based** computing, whereby **shared** resources, software and information are provided to computers and other devices **on-demand**, like a **public utility**

Source: http://en.wikipedia.org/wiki/Cloud_computing

- Cloud computing, often referred to as simply “**the cloud**,” is the delivery of **on-demand** computing resources - everything from applications to data centres - over the **internet** on a **pay-for-use** basis.

- **Elastic** resources - Scale up or down quickly and easily to meet demand
- **Metered** service so you only pay for what you use
- **Self service** - All the IT resources you need with self-service access

Source: IBM



More definitions...

Cloud Computing is

- The storing, processing and use of **data** on remotely located computers accessed over the internet.

Source: European Commission

- A paradigm in which information is permanently stored in servers on the **internet** and cached temporarily on clients that include desktops, entertainment centres, table computers, notebooks, wall computers, hand-helds, sensors, monitors, etc.

Source: IEEE Internet Computing

- A model for enabling convenient, **on-demand** network access to a **shared** pool of **configurable** computing resources (e.g., networks, servers, storage, applications, and services) that can be **rapidly provisioned** and **released** with minimal management effort or service provider interaction

Source: NIST



Still more definitions...

Cloud Computing is

- Cloud computing - the dynamic provisioning of IT capabilities, whether hardware, software (SaaS), or services from a third party over a network - is a delivery model that can provide increased operational and financial flexibility and reduced maintenance and support.

Source: Accenture

- A style of computing where massively scalable IT-related capabilities are provided 'as a service' across the Internet to multiple external customers

Source: Gartner



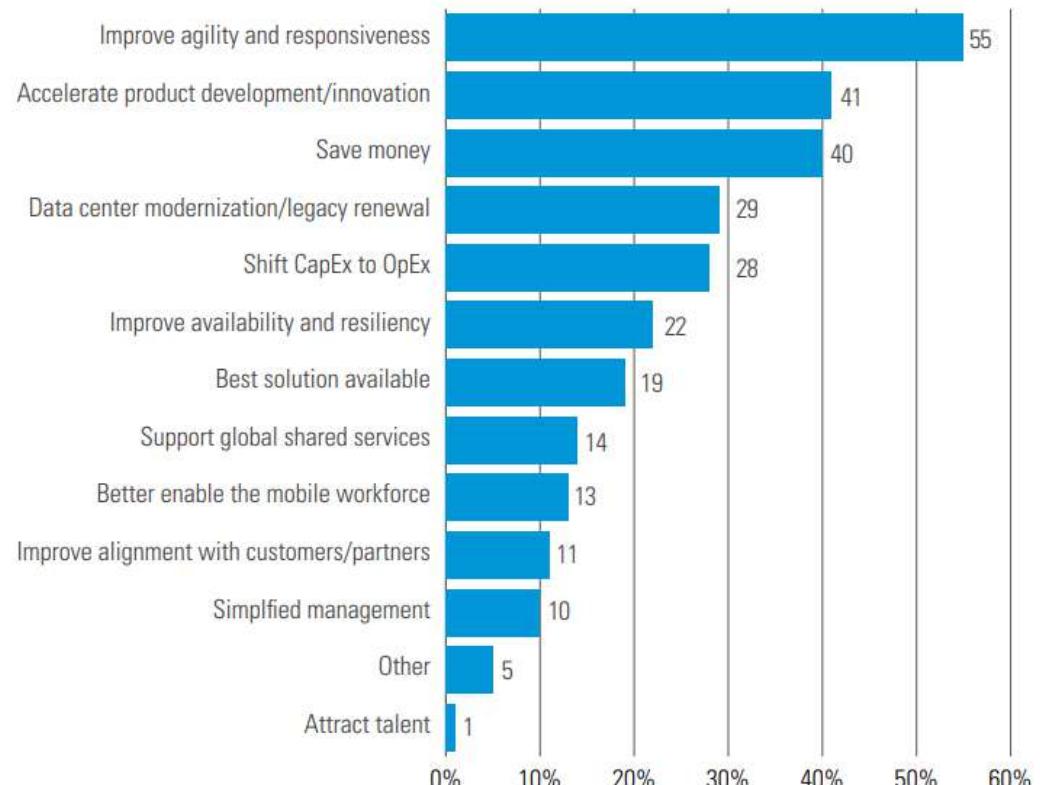
MOTIVATIONS & SCENARIO



Reasons for adopting Cloud Computing

- Cloud computing is being adopted for many reasons
 - Business agility: quickly respond to new opportunities
 - Accelerate innovation
 - Reduce costs
 - Gain efficiency and flexibility
 - etc.

Figure 3: Top reasons for adopting cloud computing



Source: Harvey Nash / KPMG 2016 CIO Survey

Source: [KPMG](#)



Benefits of Cloud Computing

Economic Benefits

- Lower entry costs
- Lower operating costs – licenses, IT resources, maintenance, support, etc.
- Reduce CapEx
- Convert CapEx to OpEx
- Pay only as you use

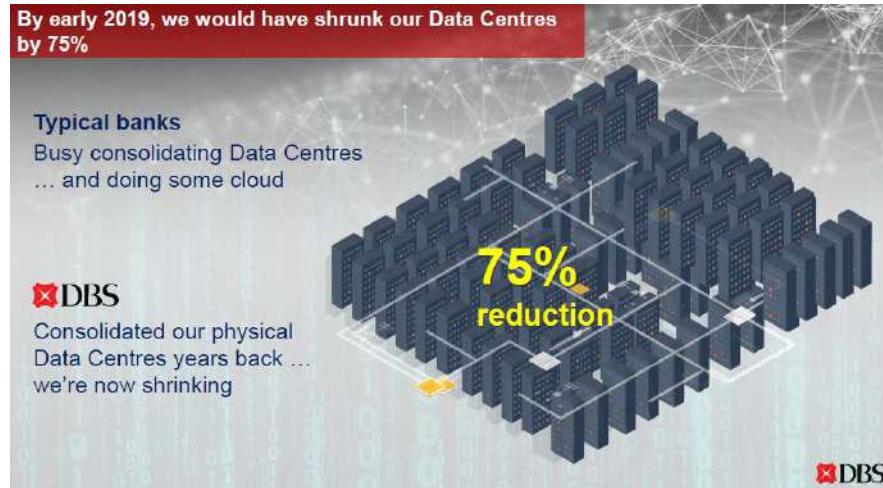
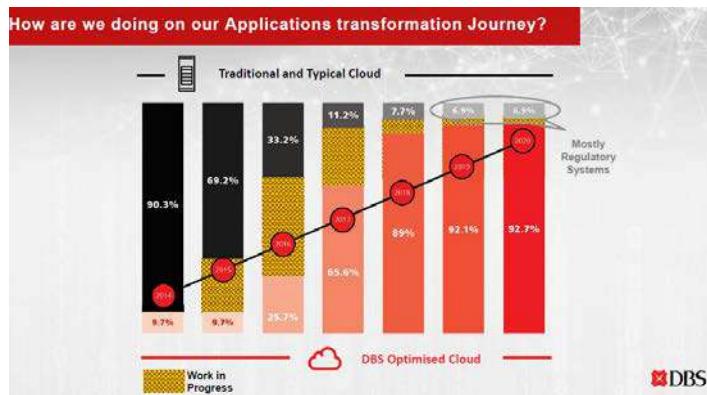
Operational Benefits

- Faster time to market - *scalability, testing, prototyping, innovation, self-service*
- No need for ‘peak’ provisioning
- Automated and standardized interfaces
- Better flexibility and mobility
- Better use of IT resources
- Better service resiliency
- Access to the latest features, updates and patches
- Availability of knowledge in the community



Enterprise adoption is accelerating

- DBS Bank in Singapore is rapidly moving to the Cloud



Source: [DBS Bank](#)



Cloud Computing has become mainstream

- Cloud Computing has become mainstream:
 - Most digital disruption uses cloud services in some form
 - Startups adopt a cloud-first strategy
- Startups:
 - Cloud is the default platform
 - Focus is on creating innovative software products
- Enterprises:
 - Cloud is an enabler of strategic capabilities
 - Focus is on migration, integration and co-existence



CLOUD SERVICE MODELS



Cloud Service

- Cloud Computing SERVICE is
 - *any service made available to users on demand via the Internet from a cloud computing provider's servers as opposed to being provided from a company's own on-premises servers.*

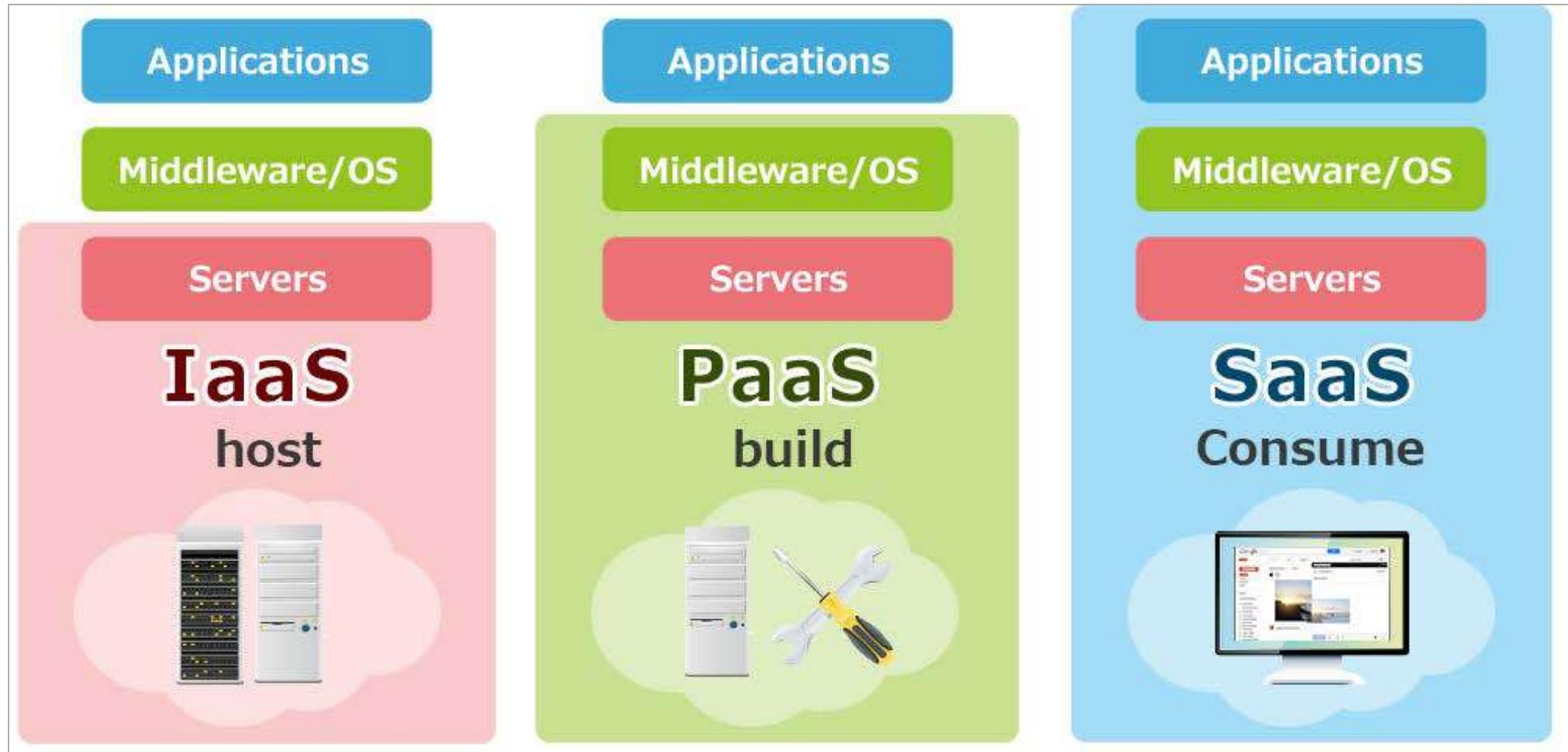
Source: IBM

- A service in this context can provide hardware or software like...
 - Server (*i.e., Hardware consisting of Processors, Memory, Disks etc*)
 - Systems (*e.g. Operating Systems, Utilities, Middleware etc*)
 - Software (*e.g. Email Applications, ERPs etc*)

Key Phrase/Philosophy is: ***as a Service***



Cloud Service Models



Source: medium.com



Cloud Service Models: IaaS

- Infrastructure as a Service (IaaS)
 - Rents computing resource instead of buying them.
 - Service is typically paid for on a usage basis.
 - The service may include Dynamic Scaling so resources can be increased or decreased based on needs.
 - A service level is agreed upon eg:99.999 and more resources will be dynamically provided if usage is > 80%
 - Amazon's Elastic Compute Cloud. EC2
 - Provides a Web interface to access VM
 - Scalability under user's control
 - Provides multiple OS such as Linux, Solaris and Windows



Cloud Service Models: IaaS

- Infrastructure as a Service (IaaS) – How it operates
 - Applications are processed and data is stored on the cloud provider's infrastructure
 - Users use the cloud infrastructure services for compute processing, storage, networking and other IaaS services for their needs – to deploy and run software applications, middleware and tools, to store data, etc.
 - Users do not manage or control the underlying cloud infrastructure
 - Users have control over the operating systems on their compute instances (cloud-based computers), storage and network configuring



Cloud Service Models: PaaS

- Platform as a Service (PaaS)
 - PaaS provides more than just Infrastructure
 - It provides a Solution Stack.
 - Software development and deployment is entirely in the cloud.
 - Similar to Web hosting of internet era. To build a website providers provide development tools.
 - Includes development, source code management, testing and deployment
 - Inherently multi-tenant; means same software instance can be used by multiple customers.
 - Dynamic scaling in this context means software can easily be scaled up or down.
 - Risk is possibility of Vendor tie up if your software solution uses vendor specific frameworks and switching to another vendor may be expensive. Consider Open PaaS.



Cloud Service Models: PaaS

- Platform as a Service (PaaS) – How it operates
 - Software Developers develop and run software applications on the cloud provider's platform
 - Users use self-built or acquired applications created using programming languages and tools supported by the cloud provider
 - Users do not manage or control the underlying cloud infrastructure (e.g., servers, operating systems or storage)
 - Users have control of the deployed applications and possibly the application hosting environment configurations



Service Models: SaaS

- Software as a Service (SaaS)
 - Concept evolves from Application Service Providers of old times (ASP) which grew up when internet commenced.
 - Common example is email.
 - Supply chain and CRM most common business software offered initially
 - Product Example: Sales force.
 - Initially offered sales automation like sales tracking and forecasting
 - Then expanded it to CRM operations.
 - Subsequently added AppExchange providing APIs for third party software providers to integrate their applications
 - Then added Apex which facilitates users to build applications and manage data and processes.



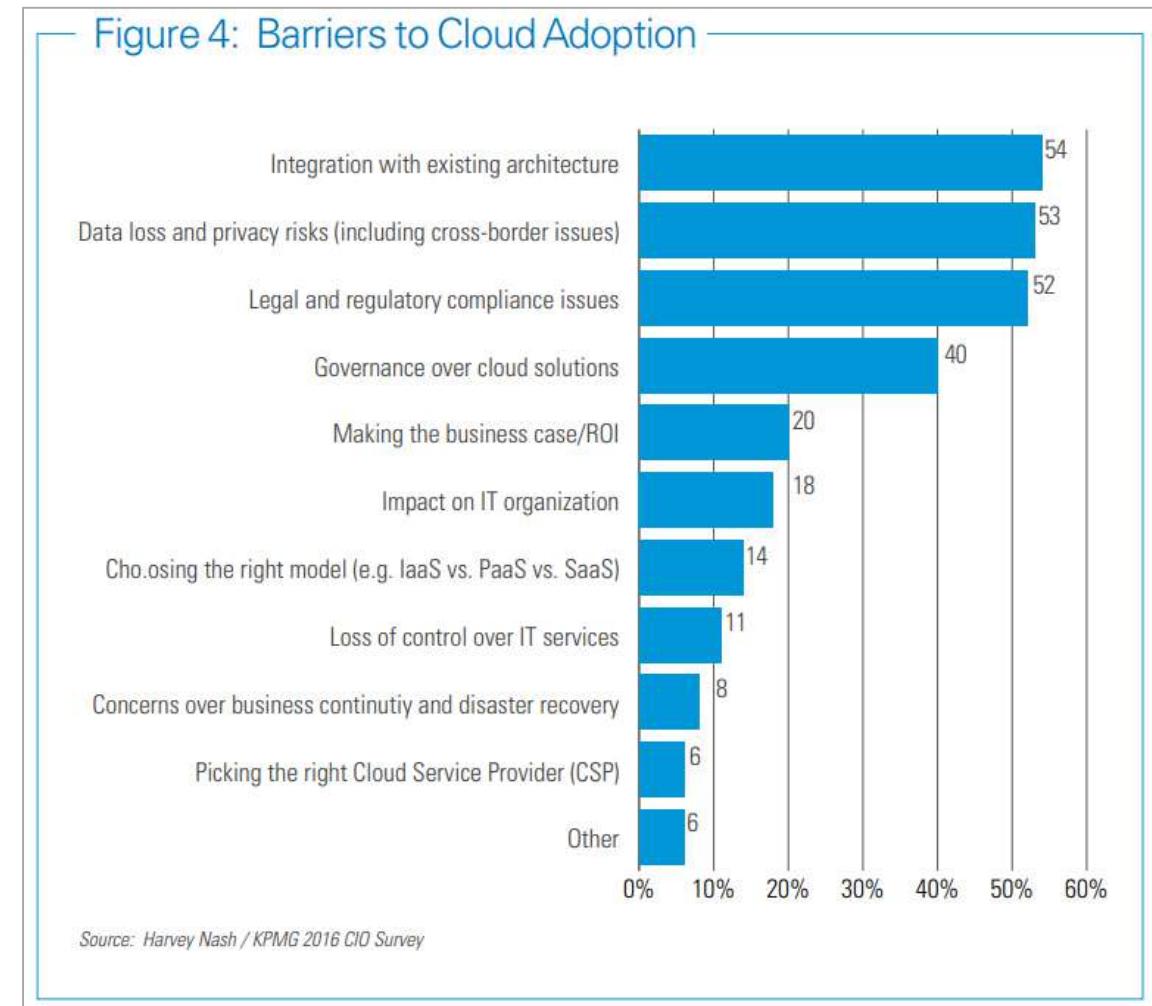
Service Models: SaaS

- Software as a Service (SaaS) – How it operates:
 - Cloud-based applications are offered by the provider
 - Users use applications that runs on the cloud by using a web browser
 - Users do not manage or control the underlying cloud infrastructure (e.g., servers, operating systems, storage or application)
 - Limited user-specific settings may be provided to users to configure the application



Barriers to adopting Cloud Computing for enterprises

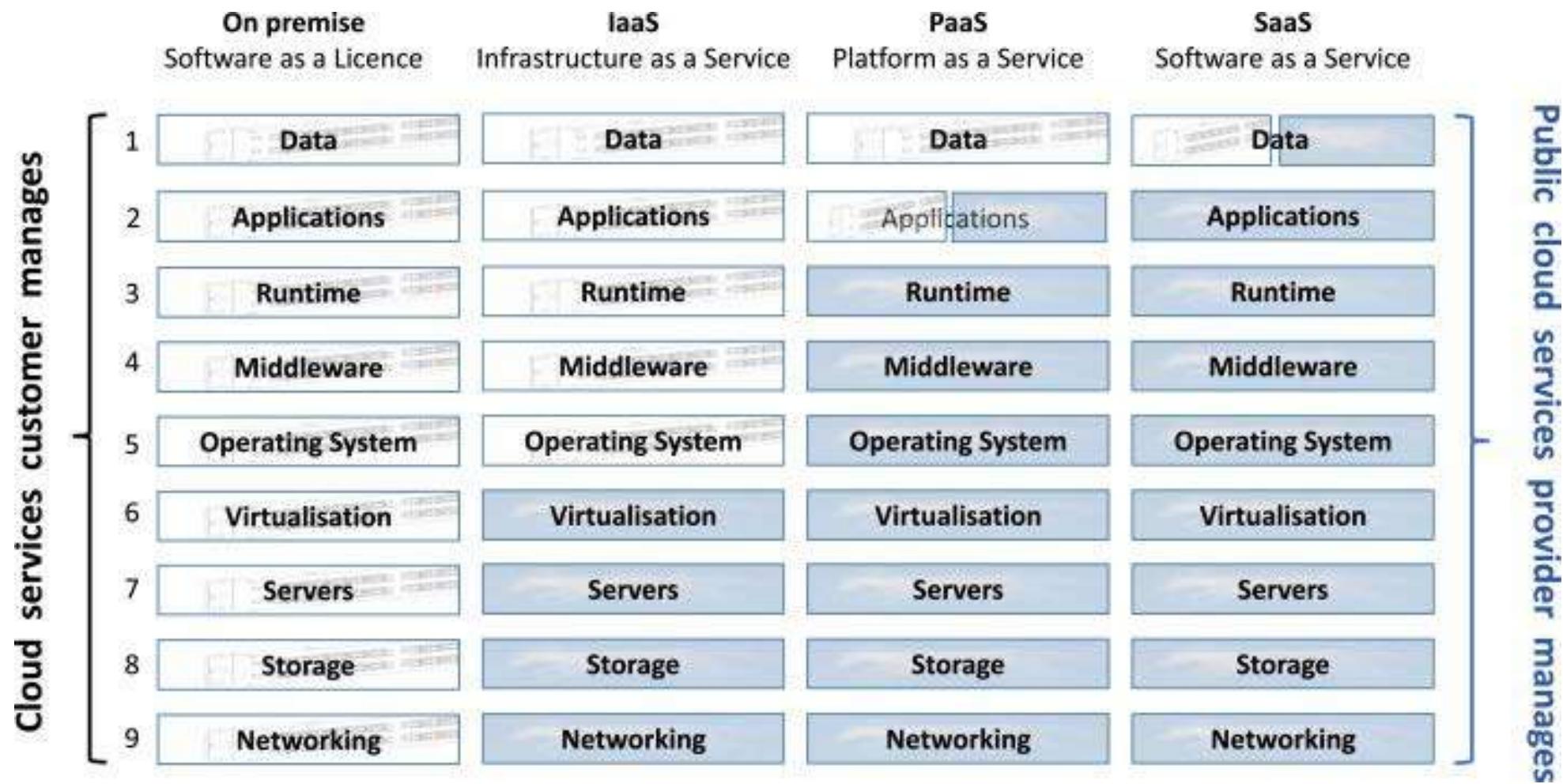
- Established enterprises face more challenges than startups in adopting cloud:
 - Regulatory
 - Governance
 - Security
 - Privacy
 - Migration
 - Integration
 - Co-existence
 - Etc.



Source: [KPMG](#)



Who manages what on the Cloud?



Source: ISO 27018



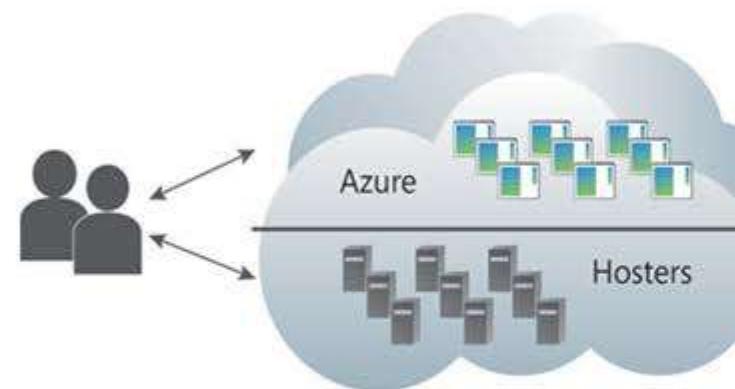
CLOUD DEPLOYMENT MODELS



Cloud Deployment Models

■ Public Cloud

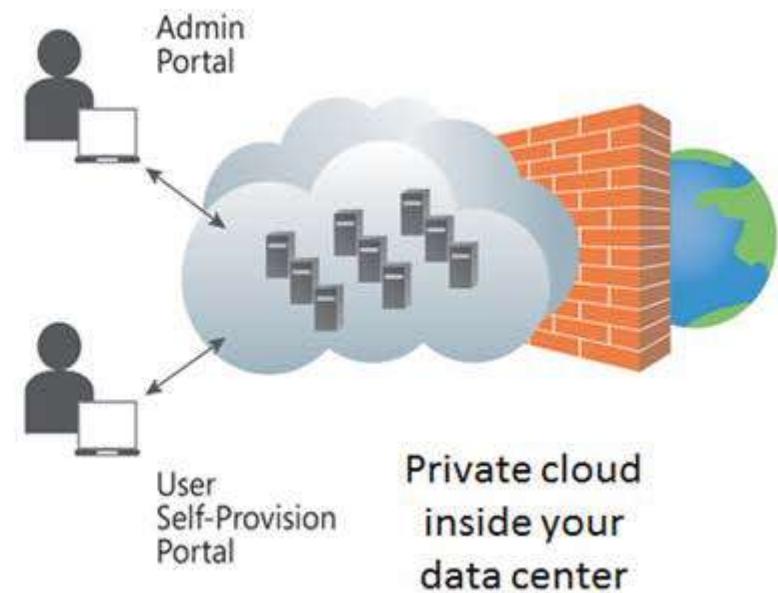
- Available to any paying organisations or the public
- Infrastructure is owned by provider



Public cloud by service providers

■ Private Cloud

- Operated solely for an organisation
- May be managed by organisation or third party
- May exist on-premise or off-premise



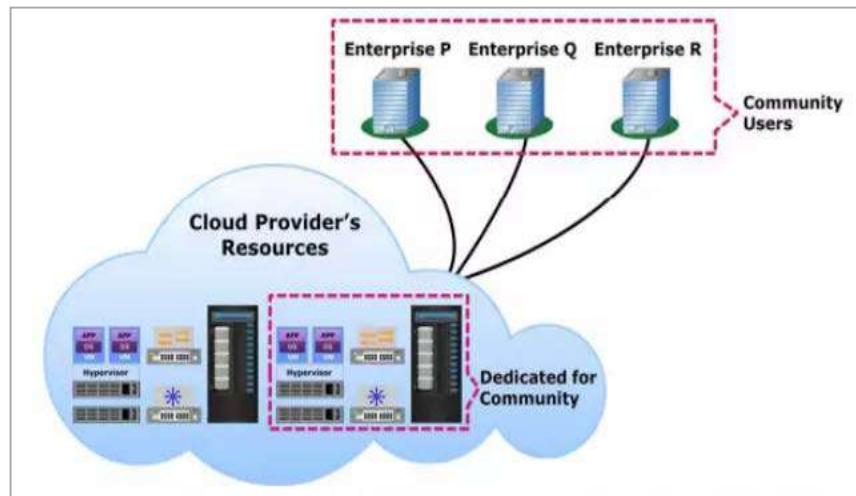
Private cloud inside your data center



Cloud Deployment Models

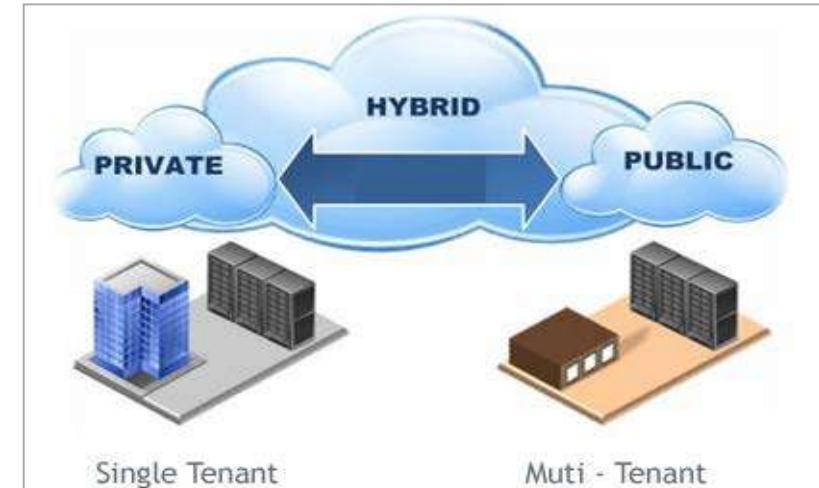
■ Hybrid Cloud

- Made up of two or more cloud types that remain unique entities but are inter-connected to enable sharing of resources



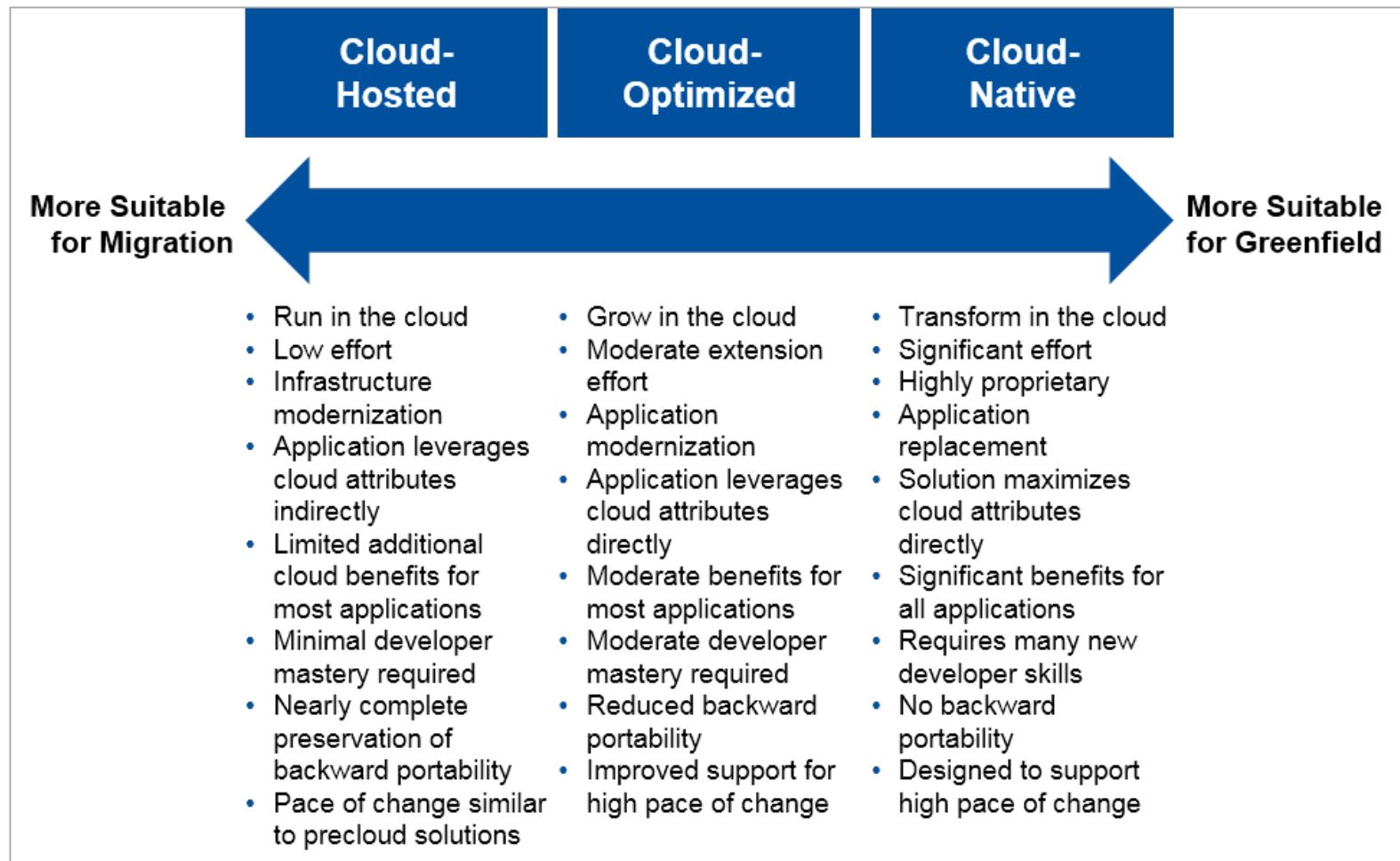
■ Community Cloud

- Infrastructure is shared by several organisations
- Supports a specific community
- May be managed by organisation or third party
- May exist on or off premise





Application Development Options on the Cloud



Source: Gartner, Nov 2015

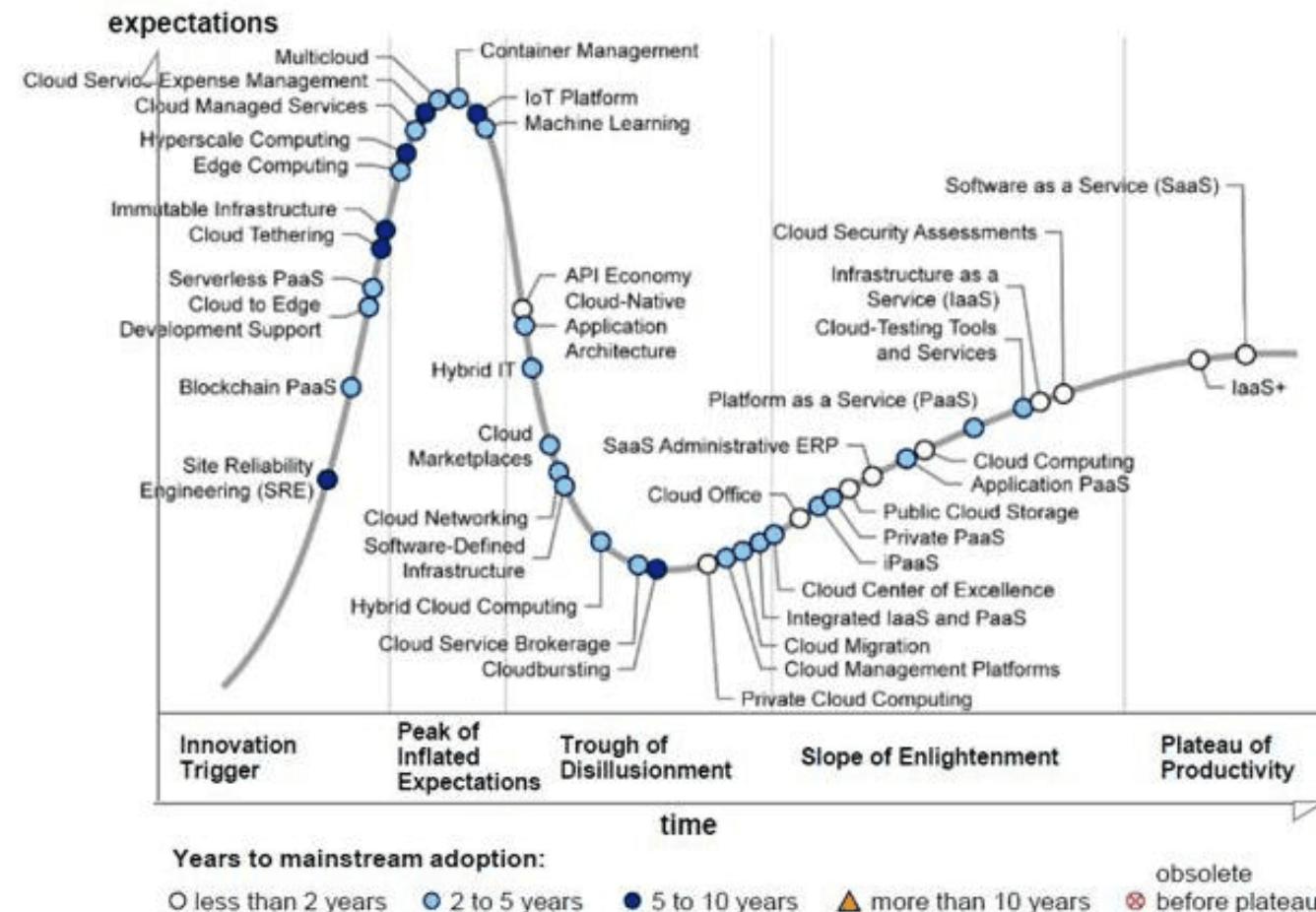


INDUSTRY DEVELOPMENTS



Gartner's Hype Cycle

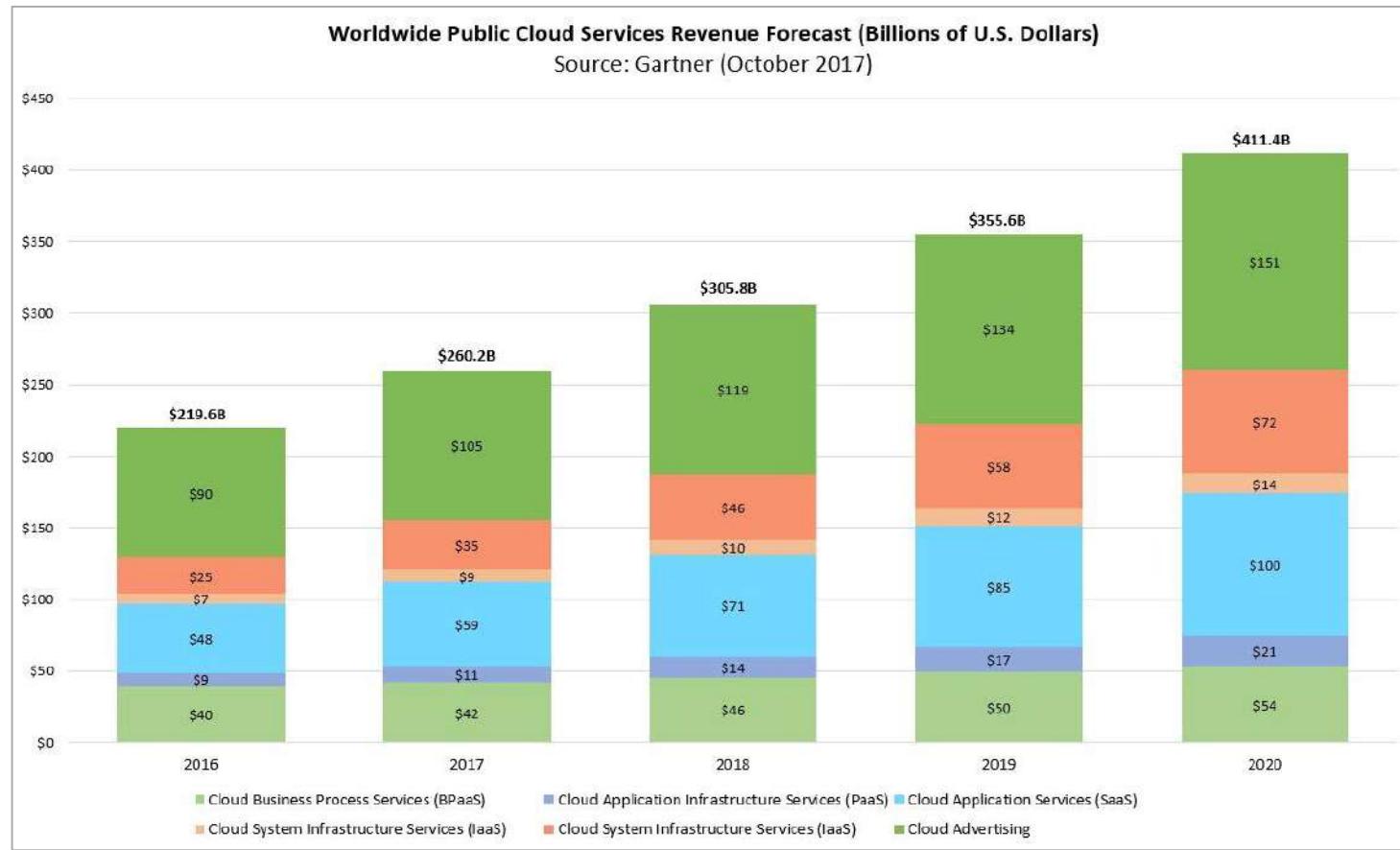
Hype Cycle for Cloud Computing, 2018





Cloud Computing is growing rapidly

- Spending on Cloud Computing is growing at 6 times the rate for overall IT spending from 2015 to 2020





Gartner's Magic Quadrant for Cloud IaaS

- Gartner's Magic Quadrant for Cloud **IaaS** is widely referenced
- AWS and Microsoft Azure were leaders for Cloud **IaaS** in July 2019

Magic Quadrant for Cloud Infrastructure as a Service, Worldwide

Published 16 July 2019 - ID G00365830 - 43 min read

The distinctions among providers are apparent in the market for cloud IaaS in terms of worldwide enterprise adoption, capabilities and service availability. Infrastructure and operations leaders should evaluate providers with broad capabilities and a positive track record for customer success.

Source: [Gartner](#)



Source: [Gartner](#)



Gartner's Magic Quadrant for Cloud PaaS

- Gartner's Magic Quadrant for Cloud **PaaS** is new
- AWS and Microsoft Azure were leaders for Cloud **IaaS** in June 2017
- Various types of PaaS (xPaaS) are available
- High-productivity aPaaS (hpaPaaS) provides rapid application development (RAD) features for development, deployment and execution

Gartner

Magic Quadrant for Enterprise High-Productivity Application Platform as a Service

Published: 27 April 2017 **ID:** G00304071
Analyst(s): Paul Vincent, Van L. Baker, Yefim V. Natis, Kimihiko Iijima, Mark Driver, Rob Dunne

Summary
 High-productivity, or rapid application development and deployment, platform technology in the cloud is growing swiftly as enterprise IT increasingly exploits cloud-based tooling with reduced skill requirements. We examine the leading enterprise vendors for these high-productivity aPaaS platforms.

Source: [Gartner](#)

Magic Quadrant

Figure 1. Magic Quadrant for Enterprise High-Productivity Application Platform as a Service



Source: Gartner (April 2017)

Source: [Gartner](#)



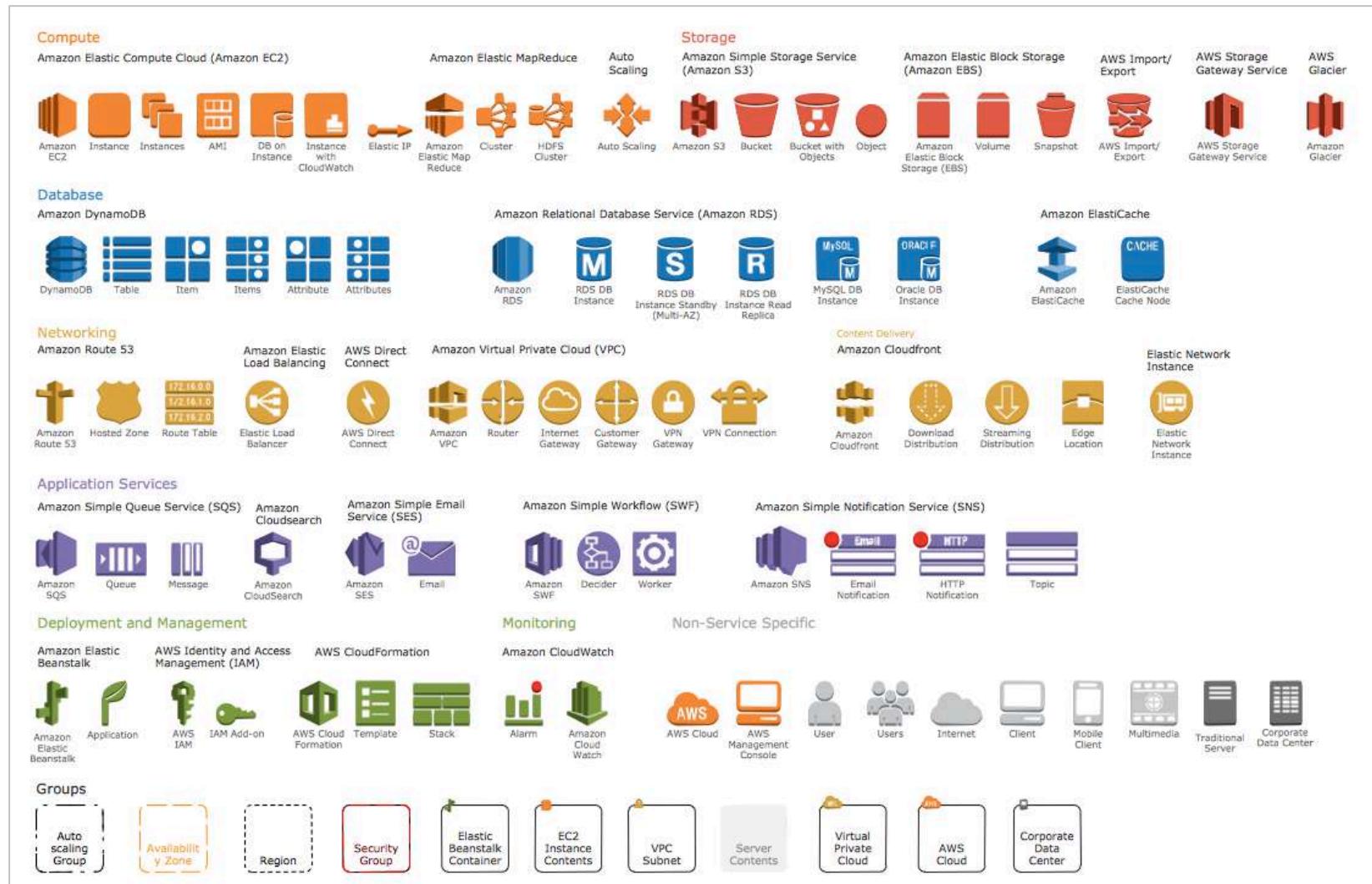
Service provider: Amazon Web Services (AWS)

- Portfolio of cloud services:
 - IaaS
 - PaaS
 - SaaS

Compute	Networking & Content Delivery	Analytics	Application Services
Amazon EC2	Amazon VPC	Amazon Athena	AWS Step Functions
Amazon EC2 Container Registry	Amazon CloudFront	Amazon EMR	Amazon API Gateway
Amazon EC2 Container Service	Amazon Route 53	Amazon CloudSearch	Amazon Elastic Transcoder
Amazon Lightsail	AWS Direct Connect	Amazon Elasticsearch Service	Amazon AppStream
Amazon VPC	Elastic Load Balancing	Amazon Kinesis	
AWS Batch		Amazon Redshift	
AWS Elastic Beanstalk		Amazon QuickSight	
AWS Lambda		AWS Data Pipeline	
Auto Scaling		AWS Glue	
Elastic Load Balancing			
Storage			
Amazon Simple Storage Service (S3)			
Amazon Elastic Block Storage (EBS)			
Amazon Elastic File System (EFS)			
Amazon Glacier			
AWS Storage Gateway			
AWS Snowball			
AWS Snowball Edge			
AWS Snowmobile			
Database			
Amazon Aurora			
Amazon RDS			
Amazon DynamoDB			
Amazon DynamoDB Accelerator (DAX)			
Amazon ElastiCache			
Amazon Redshift			
AWS Database Migration Service			
Migration			
AWS Application Discovery Service			
AWS Database Migration Service			
AWS Server Migration Service			
AWS Snowball			
AWS Snowball Edge			
AWS Snowmobile			
Developer Tools			
AWS CodeStar			
AWS CodeCommit			
AWS CodeBuild			
AWS CodeDeploy			
AWS CodePipeline			
AWS X-Ray			
AWS Command Line Interface			
Management Tools			
Amazon CloudWatch			
Amazon EC2 Systems Manager			
AWS CloudFormation			
AWS CloudTrail			
AWS Config			
AWS OpsWorks			
AWS Service Catalog			
AWS Trusted Advisor			
AWS Personal Health Dashboard			
AWS Command Line Interface			
AWS Management Console			
AWS Managed Services			
Artificial Intelligence			
Amazon Lex			
Amazon Polly			
Amazon Rekognition			
Amazon Machine Learning			
Mobile Services			
AWS Mobile Hub			
Amazon API Gateway			
Amazon Cognito			
Amazon Pinpoint			
AWS Device Farm			
AWS Mobile SDK			
Internet of Things			
AWS IoT Platform			
AWS Greengrass			
AWS IoT Button			
Contact Center			
Amazon Connect			
Game Development			
Amazon GameLift			
Amazon Lumberyard			



Amazon Web Services (selected services)





Service provider: Microsoft Azure

What is Microsoft Azure?

Virtual Machines
Virtual machines are local cloud building blocks. Get full control over a virtual machine with virtual hard disks, install and run software yourself. Virtual machines can be used for a wide variety of workloads, including: Web and mobile services, desktops, databases, and more.

Cloud Services
Easily access and manage three general business VMs. We monitor and update each VM as needed with system updates, patches, and security fixes. We also provide a management interface for monitoring and managing your VMs and their resources. Worker roles are used for processing and running services. The web role is used for serving static content and integrating with its already installed and configured.

App Service
Azure App Service is a high productivity service for developers to build, manage, and scale web applications. It has built-in support for mobile, web, and API services that enables you to easily distribute your applications to the cloud and seamlessly integrate them with on-premises resources and local-based applications.

Catalog of Services

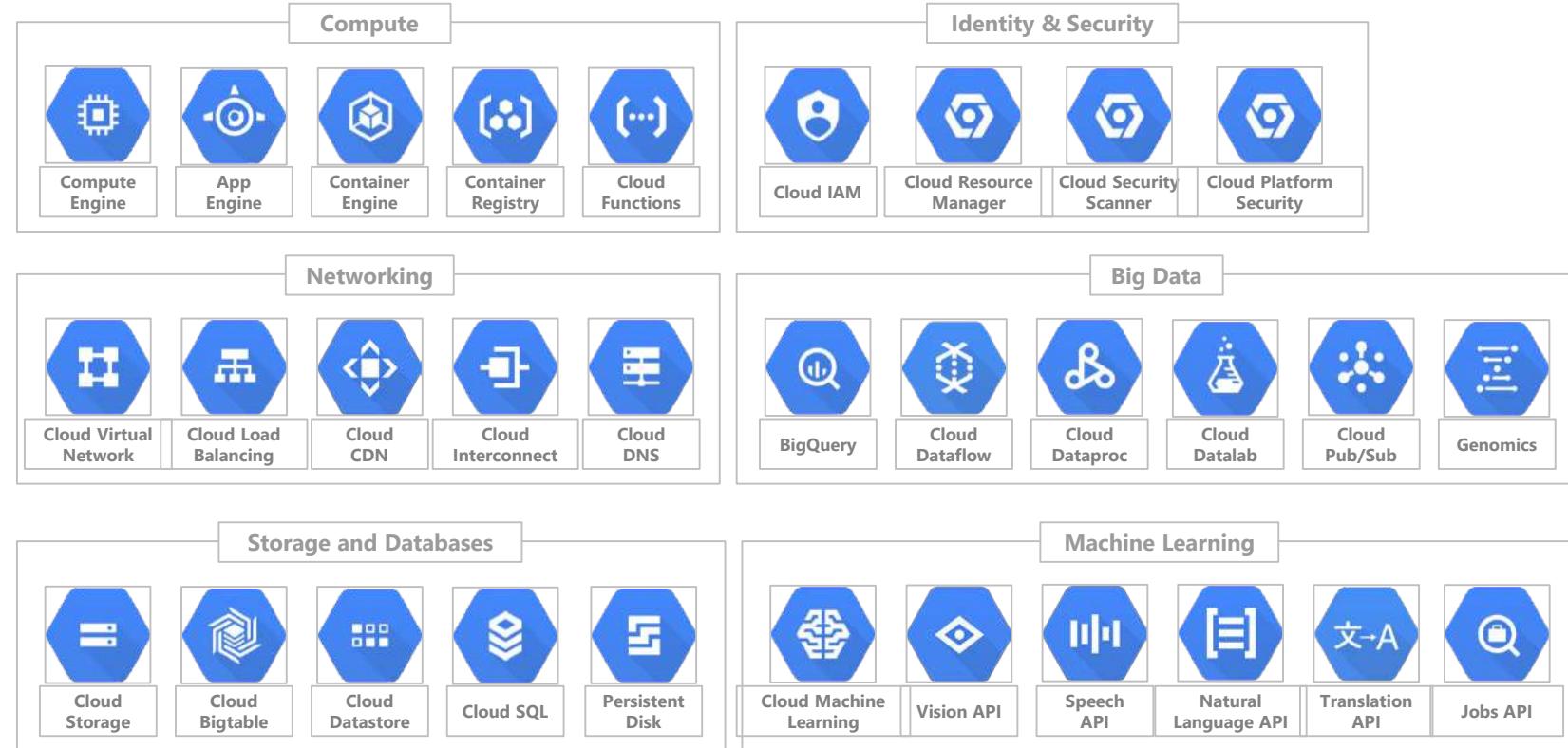
COMPUTE	WEB & MOBILE	STORAGE & BACKUP	HYBRID INTEGRATION	NETWORKING	IDENTITY & ACCESS	MEDIA & CDN
Virtual Machines Get full control over a virtual machine with virtual hard disks, install and run software yourself.	Web Apps Managed web platform, get started with Java and Node.js by going many more languages.	Storage Blobs & Files Store binary application data and media content - move for deduced and storage virtual disk for VMs.	Storage Queues Simple queue system for application-to-application communication.	Virtual Network Manage infrastructure, connect with other clouds.	Active Directory Identify and manage users, groups, and devices for cloud and hybrid scenarios.	Media Services Create content for your app in Content Delivery Network (CDN) at 100+ edge locations to support low latency requests.
Cloud Services Manage infrastructure, connect with other clouds.	Mobile Apps Build hybrid experiences for mobile apps, native clients support on most device platforms.	Backup Managed service for backing up on-premises Server, machine, and file shares.	BizTalk Services Build EDI and Enterprise Application Integration (EAI) solutions in the cloud.	Express Route Provide direct connectivity between your on-premises infrastructure and Azure.	Multi-Factor Authentication Identify and verify users with additional factors for cloud and hybrid scenarios.	CDN Create content for your app in Content Delivery Network (CDN) at 100+ edge locations to support low latency requests.
Service Fabric Build distributed systems and stateful applications composed of microservices.	API Apps Create and manage your API logic as APIs for other services and apps to consume.	Import / Export Full managed data loader - import/export data to remote data stores or back-end storage.	Hybrid Connections Connect apps in Azure with on-premises resources without a proxy or virtual IP.	Traffic Manager Load balance traffic across multiple services running on different locations.	Active Directory Identify and manage users, groups, and devices for cloud and hybrid scenarios.	Media Services Create content for your app in Content Delivery Network (CDN) at 100+ edge locations to support low latency requests.
Batch Handle large-scale computing and high performance computing (HPC) applications.	Logic Apps Build serverless workflows by linking your custom APIs with an API Gallery Marketplace.	Site Recovery Continuity and disaster recovery of on-premises data centers and cloud services.	Service Bus Message queuing and reliable communication between cloud and on-premises applications.	Express Route Provide direct connectivity between your on-premises infrastructure and Azure.	Multi-Factor Authentication Identify and verify users with additional factors for cloud and hybrid scenarios.	CDN Create content for your app in Content Delivery Network (CDN) at 100+ edge locations to support low latency requests.
Scheduler Create jobs for recurring or one-time tasks to invoke any type of service.	Notification Hubs Deliver real-time notifications to mobile devices from any application backend, anywhere.	StorSimple Automated, policy-driven solution to extend on-premises primary storage for backup and disaster recovery.	HDInsight Big Data cluster for Apache Hadoop environments that integrate easily with Microsoft Office.	Machine Learning Allow users to train complex models to predict future events based on historical data.	Data Factory Process data assets in real time to move data from source to sink.	Event Hubs Support event-driven delivery of data from millions of devices.
Remote App Access Windows, Linux, Mac, and other desktops from anywhere.	API Management Publish and manage APIs for downstream consumers to use.	DocumentDB Reliable and consistent document database service with high availability and predictable performance metrics.	Stream Analytics Process real-time data streams in near time to detect events and trends.	Search Managed search service for search results and ranking results.	Mobile Engagement Build cross-platform mobile experiences on the go.	Mobile Engagement Build cross-platform mobile experiences on the go.
APP SERVICES	APP SERVICES	DATA	ANALYTICS	IDENTITY & ACCESS	DEVELOPER SERVICES	DEVELOPER SERVICES
ASP.NET Python Node.js PHP	ASP.NET Python Node.js PHP	SQL Database Reliable and consistent document database service with high availability and predictable performance metrics.	HDInsight Big Data cluster for Apache Hadoop environments that integrate easily with Microsoft Office.	Active Directory Identify and manage users, groups, and devices for cloud and hybrid scenarios.	Visual Studio Online Work code, plan and track projects, build, deploy and test apps in the cloud collaboratively.	Application Insights Analyze app usage, availability and performance to detect bugs, and solve problems proactively.
APP SERVICES	APP SERVICES	DATA	ANALYTICS	IDENTITY & ACCESS	DEVELOPER SERVICES	DEVELOPER SERVICES
API APPS LOGIC APPS WEB APPS MOBILE APPS	API APPS LOGIC APPS WEB APPS MOBILE APPS	Redis Cache Memory-optimized NoSQL cache service with high availability and predictable performance metrics.	Machine Learning Allow users to train complex models to predict future events based on historical data.	Multi-Factor Authentication Identify and verify users with additional factors for cloud and hybrid scenarios.	Visual Studio Online Work code, plan and track projects, build, deploy and test apps in the cloud collaboratively.	Application Insights Analyze app usage, availability and performance to detect bugs, and solve problems proactively.
APP SERVICES	APP SERVICES	DATA	ANALYTICS	IDENTITY & ACCESS	DEVELOPER SERVICES	DEVELOPER SERVICES
APP SERVICES	APP SERVICES	DATA	ANALYTICS	IDENTITY & ACCESS	DEVELOPER SERVICES	DEVELOPER SERVICES

Like it? Get it.

© 2015 Microsoft Corporation. All rights reserved. Version 2.5. Created by the Azure poster team. Email: AzurePoster@Microsoft.com

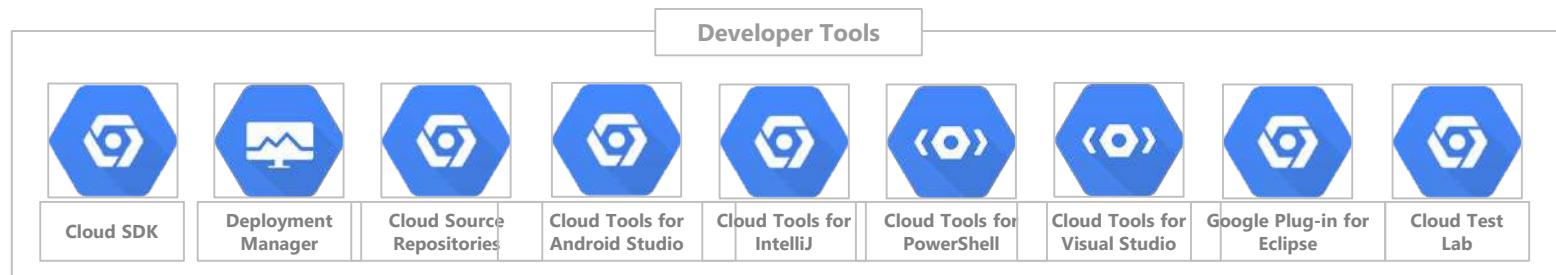
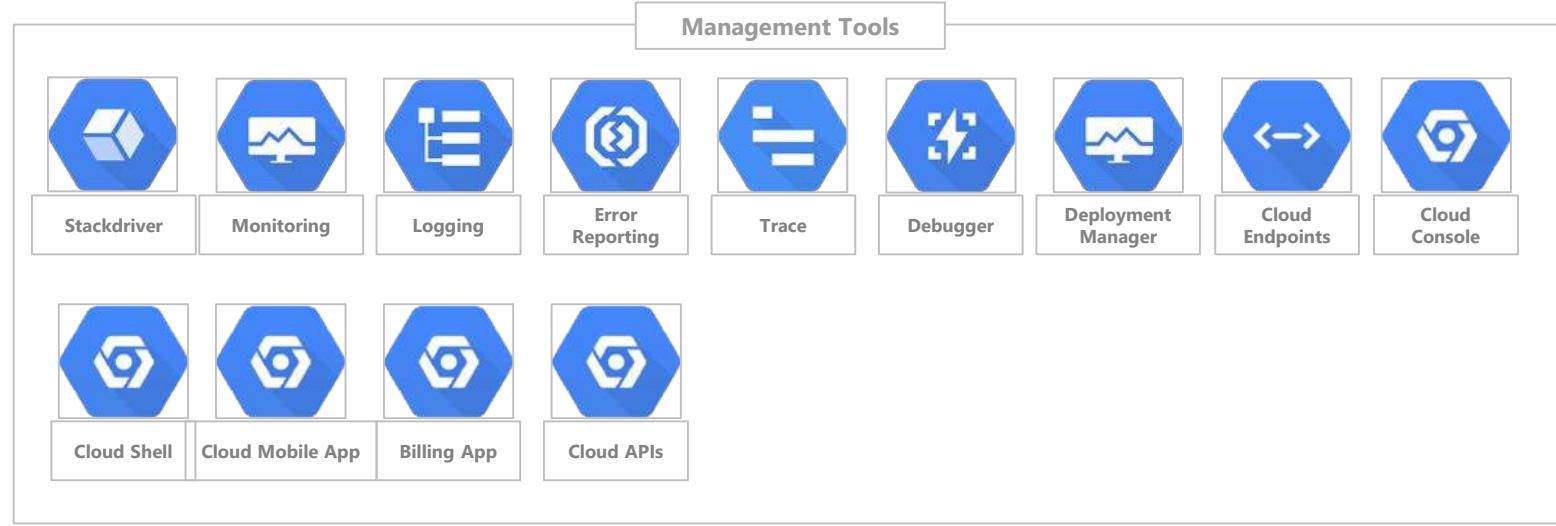


Service provider: Google Cloud Platform (GCP)





Service provider: Google Cloud Platform (GCP)





Service provider: IBM Bluemix

Compute Infrastructure	Compute Services	Storage	Network
<p>Power your most demanding workloads with high performance cloud servers</p> <p>Bare metal servers</p> <p>Virtual servers</p> <p>GPU computing</p> <p>POWER servers</p> <p>Server software</p>	<p>Speed up development and deployment with flexible compute services</p> <p>OpenWhisk</p> <p>Containers</p> <p>Cloud Foundry runtimes</p>	<p>Scale out a flexible storage environment for your data</p> <p>Object storage</p> <p>Block storage</p> <p>File storage</p> <p>Server-backup</p>	<p>Improve the way your systems work together in the cloud</p> <p>Load Balancing</p> <p>Direct link</p> <p>Network appliances</p> <p>Content delivery network</p> <p>Domain services</p>
Mobile	Watson	Data and analytics	Internet of Things
<p>Create dynamic mobile applications with easy-to-use tools</p> <p>Swift</p> <p>Mobile first services starter</p> <p>Mobile foundation</p>	<p>Turn unstructured data into intelligence and competitive advantage</p> <p>Discovery</p> <p>Conversation</p> <p>Natural Language Understanding</p> <p>Speech to text</p>	<p>Integrate powerful data and analytics layers into your systems</p> <p>Data services</p> <p>Analytics services</p> <p>Big data hosting</p> <p>Cloudera hosting</p> <p>MongoDB hosting</p> <p>Rialt hosting</p>	<p>Simplify and derive value from your IoT devices</p> <p>IoT platform</p> <p>IoT platform starter</p> <p>IoT starter for electronics</p>
Security	DevOps	Application services	Integration
<p>Secure your cloud environment and the information stored in it</p> <p>Firewalls</p> <p>Hardware Security Module (HSM)</p> <p>Intel Trusted Execution Technology</p> <p>Security software</p> <p>SSL certificate</p>	<p>Create, automate, deploy, and manage your applications in the cloud</p> <p>Eclipse</p> <p>Continuous delivery</p> <p>Availability Monitoring</p>	<p>Connect your existing apps and backend data to the cloud with scalable APIs</p> <p>Blockchain</p> <p>Message hub</p> <p>Business rules</p>	<p>Build virtual bridges to between your systems, applications and infrastructure</p> <p>API Connect</p> <p>Secure Gateway</p>
View all services		View all services	
Open the full catalog			



Summary: 10 Myths of the Cloud

1. Cloud is always about saving money
2. You have to be on the Cloud to be good
3. Cloud should be used for everything
4. “*The CEO Said So*” is a Cloud strategy
5. We need one Cloud provider
6. Cloud is less secure than on-premises capabilities
7. Cloud is not for mission-critical use
8. Cloud = Data Centre
9. Migrating to the Cloud means that you automatically get all benefits
10. Virtualization = Private Cloud

Source: [The Top 10 Cloud Myths, Gartner, July 2015](#)



THANK YOU!



Cloud Native Solution Design

CLOUD REFERENCE ARCHITECTURE

Suria R Asai

suria@nus.edu.sg

Institute of Systems Science
National University of Singapore

© 2009-23 NUS. The contents contained in this document may not be reproduced in any form or by any means, without the written permission of ISS, NUS, other than for the purpose for which it has been supplied.

Total Slides: 30



Objective

- Upon completion of this module, you should be able to
 - Understand the difference between Cloud Native Application and Cloud Enabled Application
 - Understanding the nature of business requirements that are suitable for Cloud Native Applications
 - Choose an appropriate Cloud Reference Architecture based on the technical and business use case of the problem



Outline

- Cloud Native vs Cloud Enabled Application
- Benefits of Cloud Native
- Reference Architectures



Types of software architects - 1

- **Enterprise architects** are responsible for the technical solutions and strategic direction of an organization.
 - They must work with a variety of stakeholders to understand an organization's market, customers, products, business domain, requirements, and technology.
 - Enterprise architects provide guidance, mentorship, advice, and technical leadership for other architects and developers.
- A **solution architect** converts requirements into an **architecture for a solution**.
 - They work closely with business analysts and product owners to understand the requirements so that they can design a solution that will satisfy those requirements.
 - The designs created by solution architects may be reused in multiple projects.



Types of software architects - 2

- **Application architects** focus on one or more applications and their architecture.
 - They ensure that the requirements for their application are satisfied by the design of that application.
 - They may serve as a liaison between the technical and non-technical staff working on an application.
 - They provide guidance and leadership for team members.
- **Data architects** are responsible for designing, deploying, and managing an organization's data architecture.
 - While data architects focus their attention on databases and data structures, information architects place their focus on users.
 - Information architects want to provide a positive user experience and ensure that users can easily interact with the data.

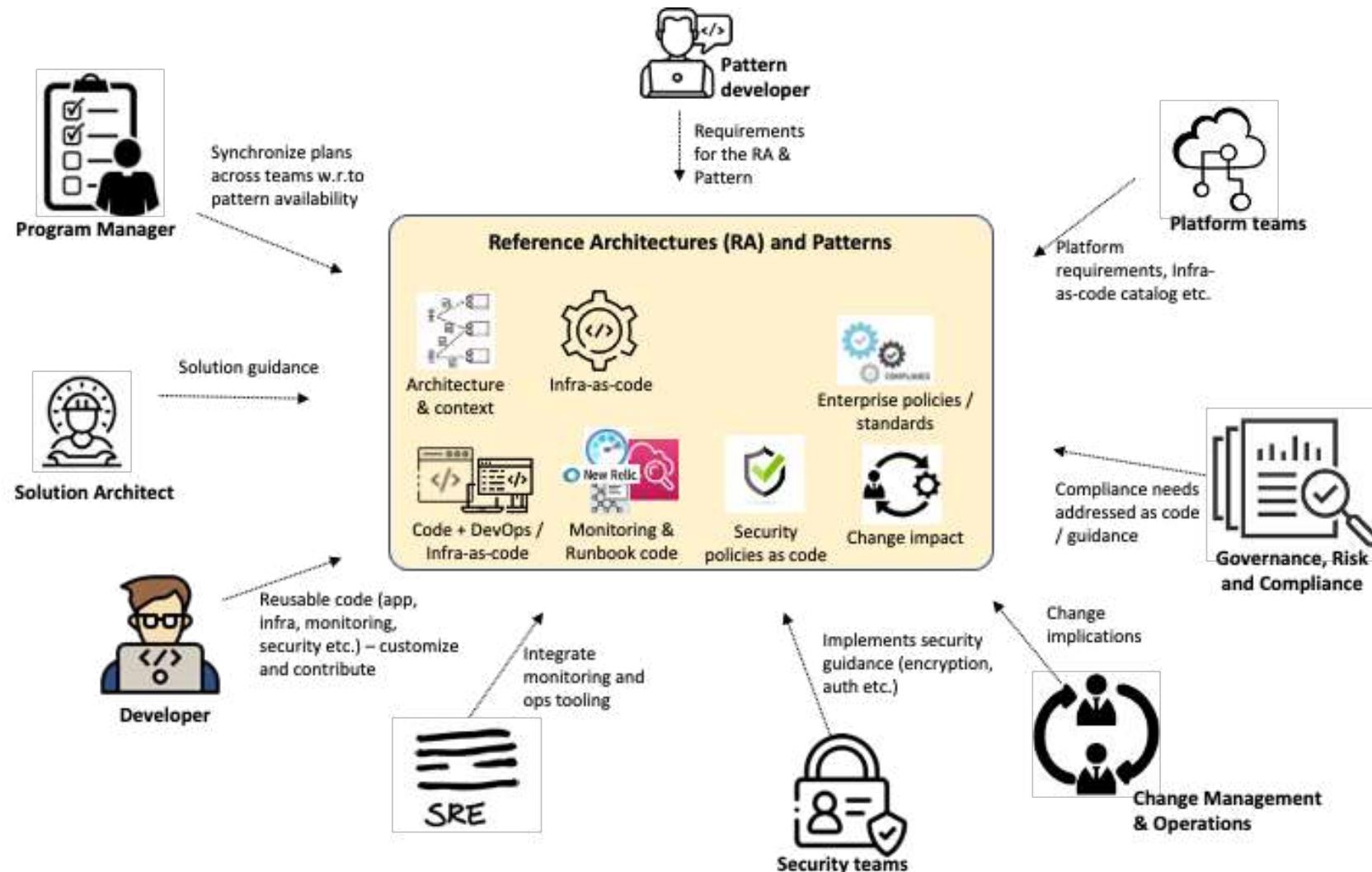


Types of software architects – 3.

- **Infrastructure architects** focus on the design and implementation of an organization's enterprise infrastructure.
 - These architects design servers, network elements, storage systems and facilities such as equipment, power, cooling units and security.
- A **security architect** is responsible for an organization's computer and network security.
 - They build, oversee, and maintain an organization's security implementations.
- A **cloud architect** is responsible for an organization's cloud computing strategy and initiatives.
 - They create cloud migration plans for existing applications.
 - They design new cloud-native applications.
 - They oversee cloud management, and create policies and procedures for governance.



Modern Cloud Reference Architecture



Reference <https://medium.com/@balasreeni/reference-architecture-patterns-for-cloud-transformation-accelerating-target-operating-model-dc9e99b02631>



Cloud Native vs. Cloud Enabled

▪ Cloud Native Application

- Application that was developed with **cloud principles** of multi-tenancy, elastic scaling and easy integration and administration in mind

An approach that builds software applications as microservices and runs them on a containerized and dynamically orchestrated platform to utilize the advantages of the cloud computing model.

▪ Cloud Enabled Application

- Application that was originally developed for deployment in a traditional data center which is **moved to the cloud**

<https://www.linkedin.com/pulse/cloud-native-cloud-enabled-evolution-revolution-dany-shapiro/>

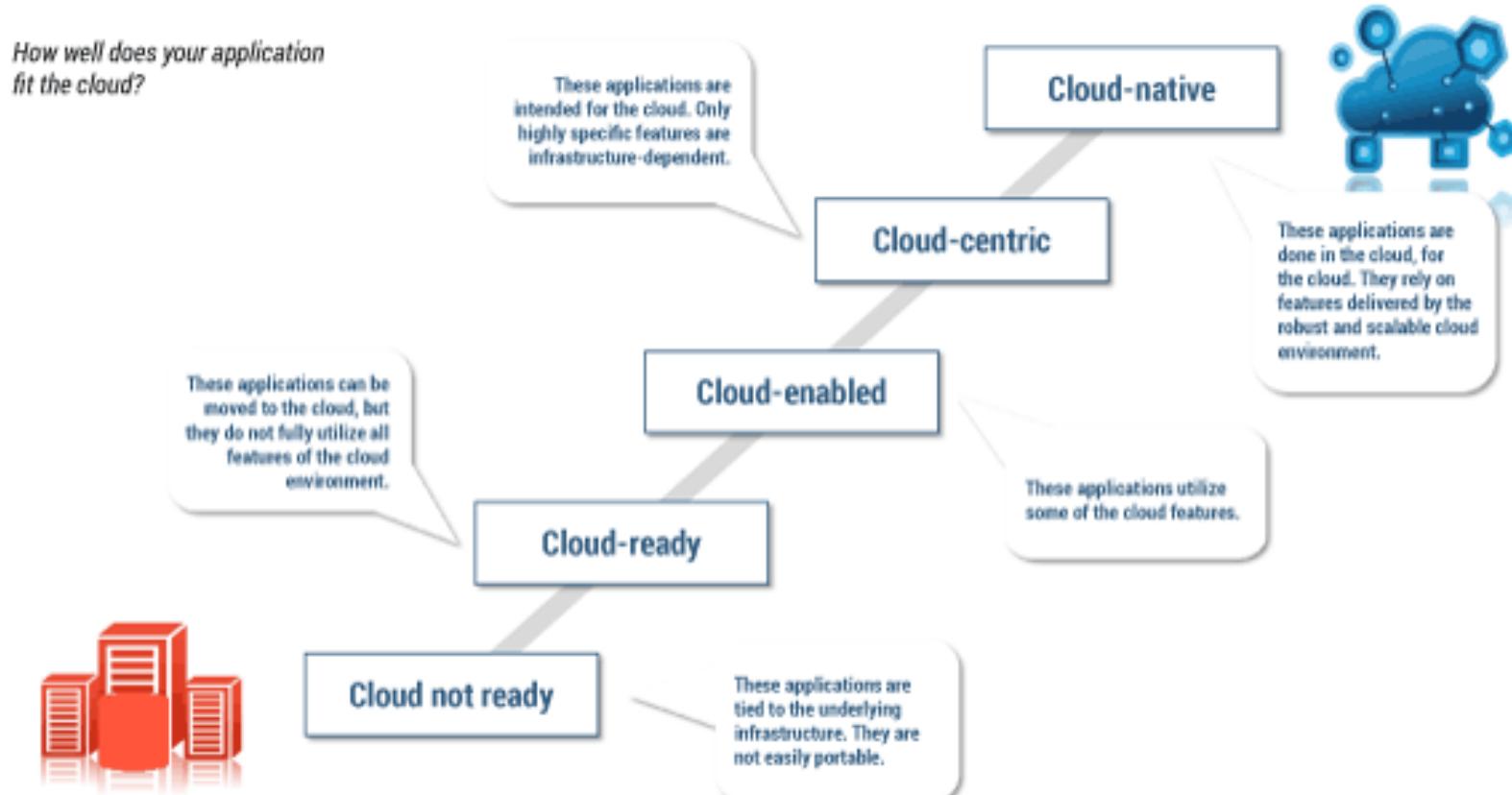


Why go Cloud Native?

- **Infrastructure as a Service (IaaS) and Platform as a Service (PaaS)** services that are utilized in building applications that leverage on **elasticity** of the cloud in a **failsafe** manner
- Adopting **greenfield design** and development of microservices for Internet of Things (IoT), **mobile** devices, **SaaS** integration, and **online** business models.
- These modern digital solutions **use APIs** from various providers, such as Google Maps for location, Facebook/Google for authentication, and Facebook/Twitter for social collaborations.
- **Scale up** the resources to do the heavy lifting in terms of **service/environment provisioning** as the load goes up or in case of failures



How well does your application fit the cloud?



Reference: <https://www.cutter.com/article/cloud-native-design-what-has-changed-488531>



Lift and Shift Model

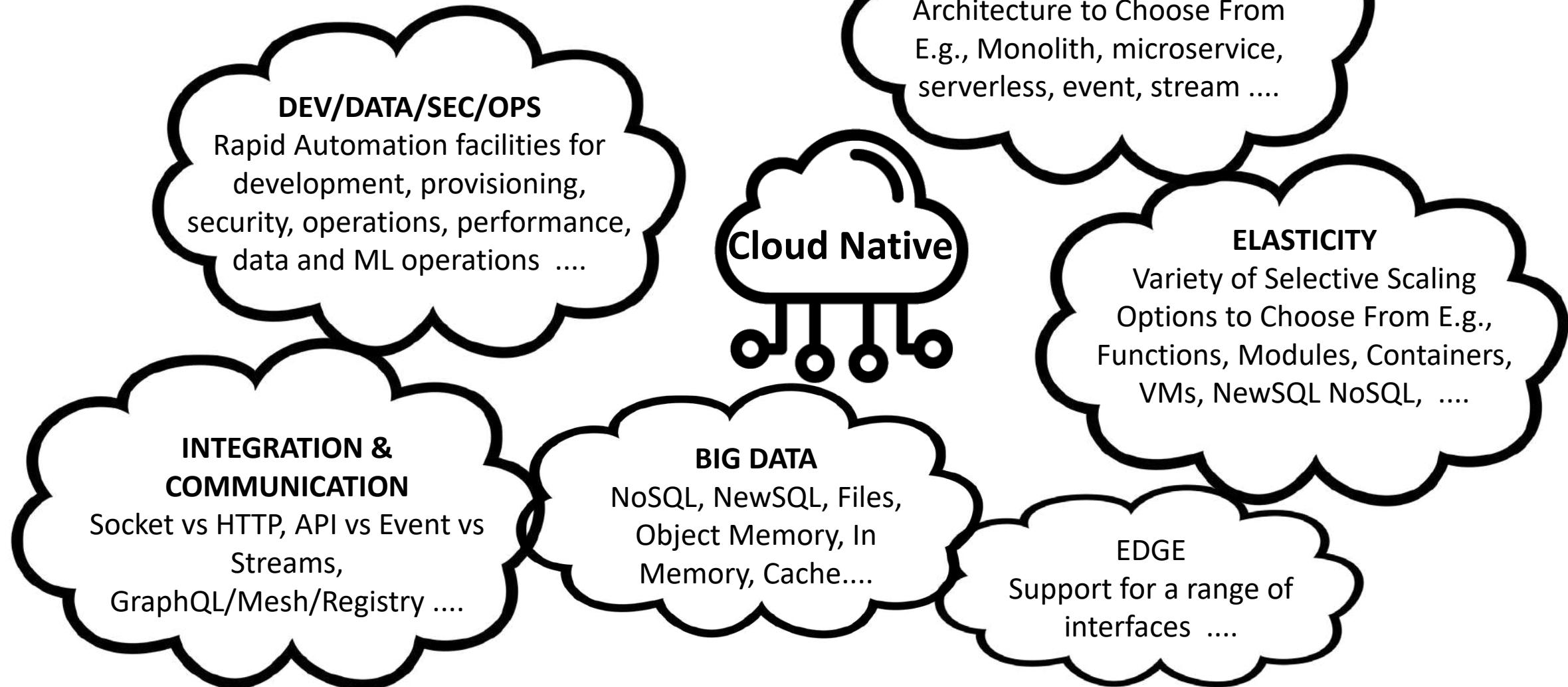
Enterprises typically make use of IaaS in the first wave because of the following:

- **Variability of resources:** The ability to add/remove resources at will, which in turn allows more business agility
- **Utility model:** IaaS provides basic resources that are rented out on an hourly basis, allowing more predictability and an opex model

Progressive Web App (PWA) i



Cloud Native Application

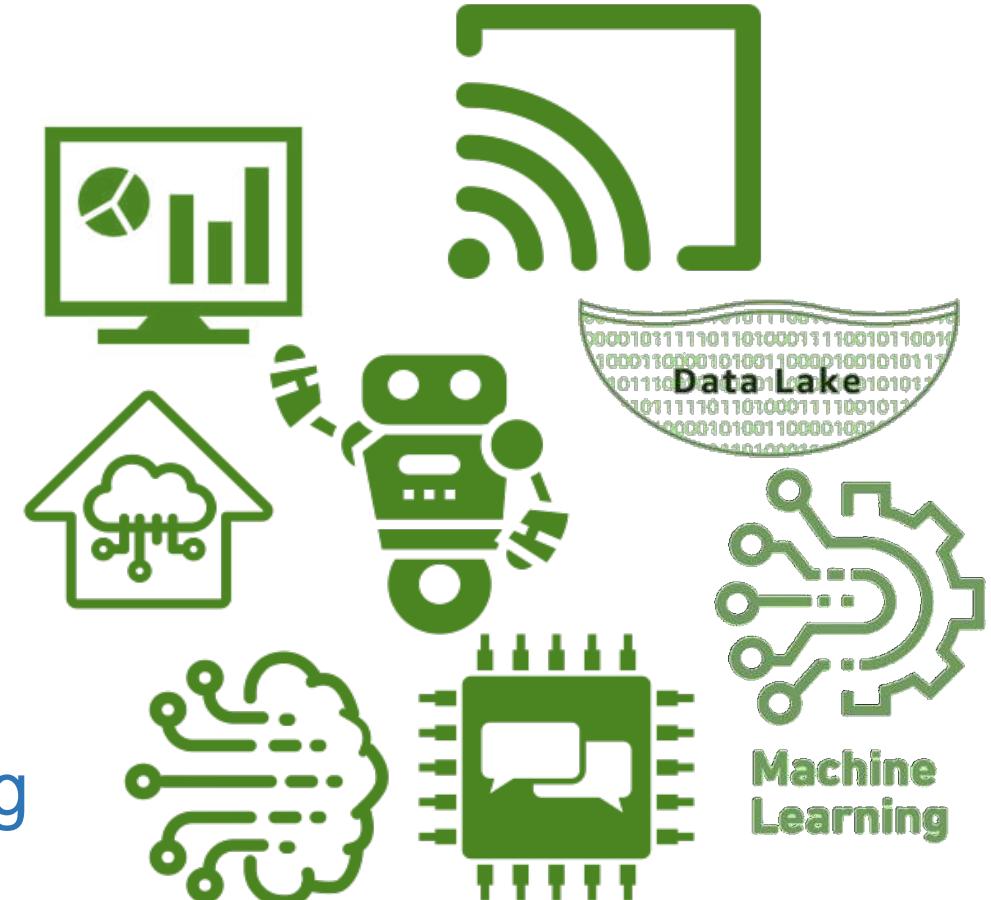




And the list does not stop there . . .

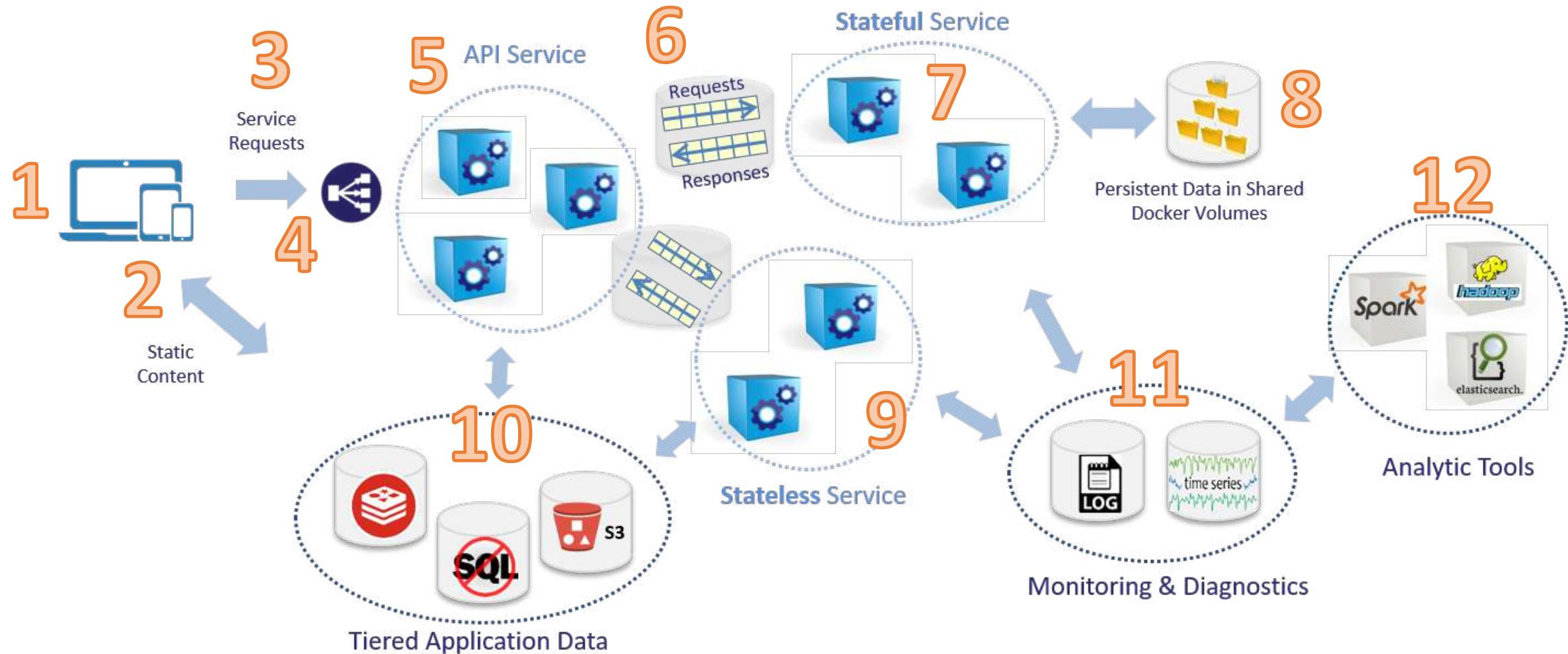
■ Green Field Technologies . . .

- Big Data Analytics
- Stream Processing
- Data Lakes and Meshes
- Machine Learning
- NLP & Text Analytics
- Deep Learning
- Robotics, IoT and Edge Computing
- Many more





Cloud Native Application Components - Example



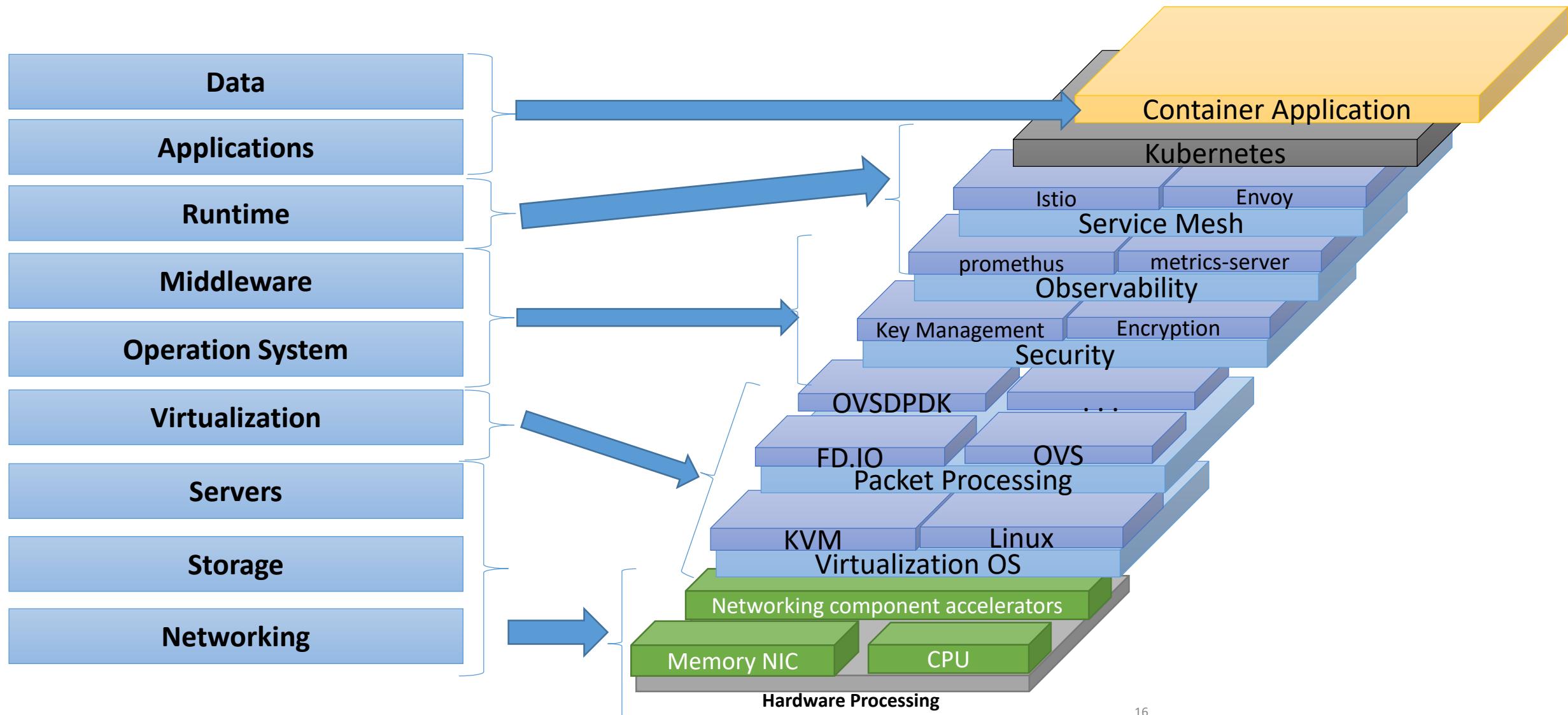


Making the Case . . .

	Pros	Cons
Lift and Shift	<ul style="list-style-type: none"> • Minimal work required to move application • Faster migration and deployment 	<ul style="list-style-type: none"> • Typically does not take advantage of native features of the cloud platform • May cost more to operate in a cloud
Partial Refactor	<ul style="list-style-type: none"> • Only parts of the application are modified • Faster migration and deployment than complete refactoring 	<ul style="list-style-type: none"> • Only takes advantage of some features of the cloud • May cost more to operate in a cloud
Complete Refactor	<ul style="list-style-type: none"> • Applications typically offer higher performance • Applications can be optimized to operate at lower costs 	<ul style="list-style-type: none"> • Much higher cost since much of the application must change • Slower time to deployment



Reference Architectures and Tech Stacks for Cloud Native





Reference Architecture – Vendor Catalogue

- Reference architecture is a template solution for an architecture for a particular domain
- Some resources for technical architecture of cloud solutions:
 - Amazon [AWS](#) Architecture Center
 - [Microsoft](#) Architecture Documentation
 - [Google](#) Cloud Solution Catalogue
 - [IBM](#) Cloud Architecture
 - [Oracle](#) Solution Center
 - [Alibaba](#) Cloud Reference Architecture
 - [Huawei](#) Cloud Stack
 - [Telekom](#) Cloud Data Center



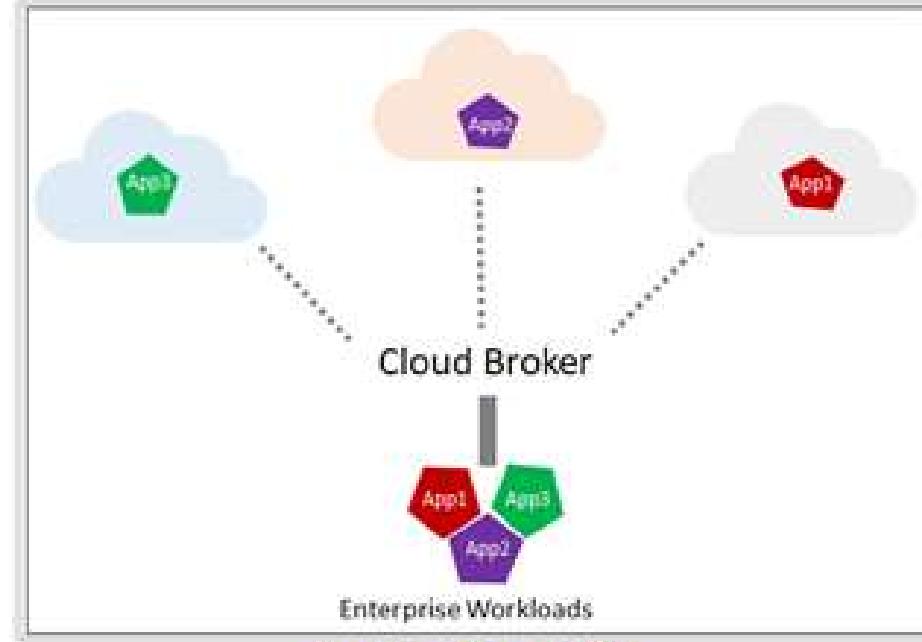


Other reference architectures

- NIST Cloud Computing Reference Architecture
 - NIST 500-292
 - <https://www.nist.gov/publications/nist-cloud-computing-reference-architecture>
 - Describe the actors in cloud computing
- CCRA: Cloud Computing Reference Architecture
 - <https://www.computer.org/csdl/proceedings/scc/2012/3049/00/06274203.pdf>
 - Describe typical architectural building blocks that make up cloud computing services
- IBM Multicloud Architecture
 - <https://www.ibm.com/cloud/garage/architectures/multicloudManagementArchitecture/reference-architecture>

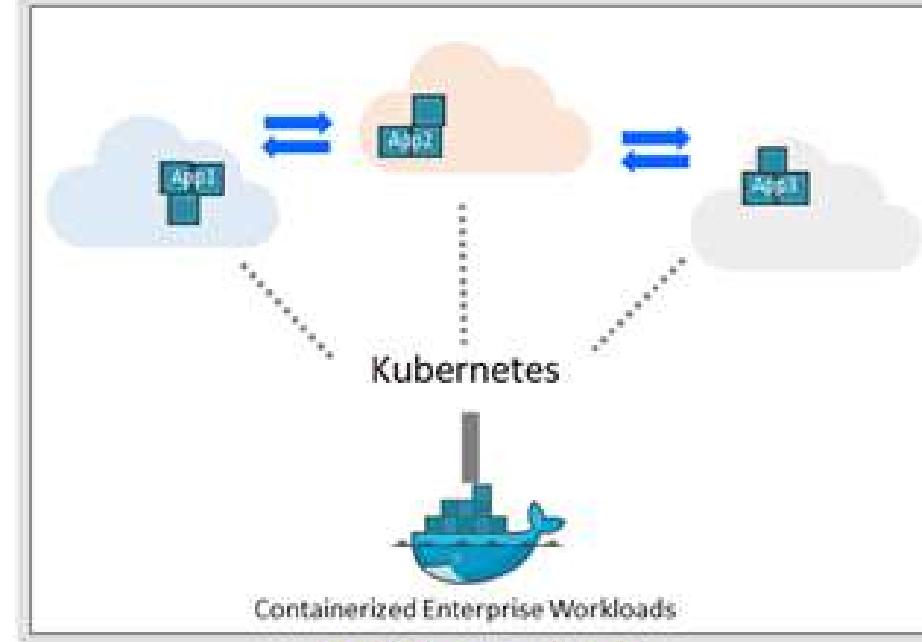


Multi-cloud Strategies



Multi-cloud 1.0

Ability to **distribute** workloads across multiple clouds



Multi-cloud 2.0

Ability to **transfer / recreate** workloads across multiple clouds

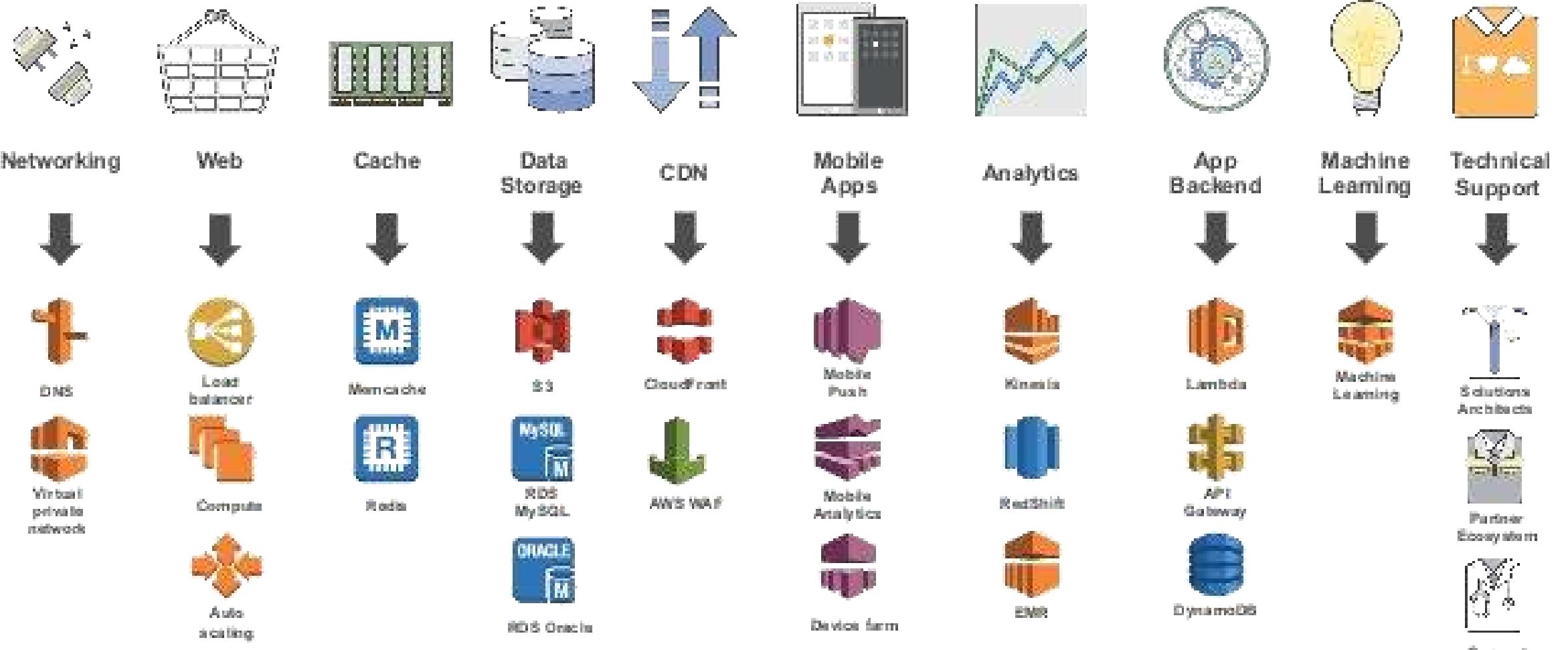


Architecture for different applications

- We will look at how cloud is used for some these use cases. The following reference architectures is taken from AWS website
 - E-commerce Web Site
 - On Line Gaming
 - Big Data Processing
 - Disaster Recovery



AWS Platform For eCommerce

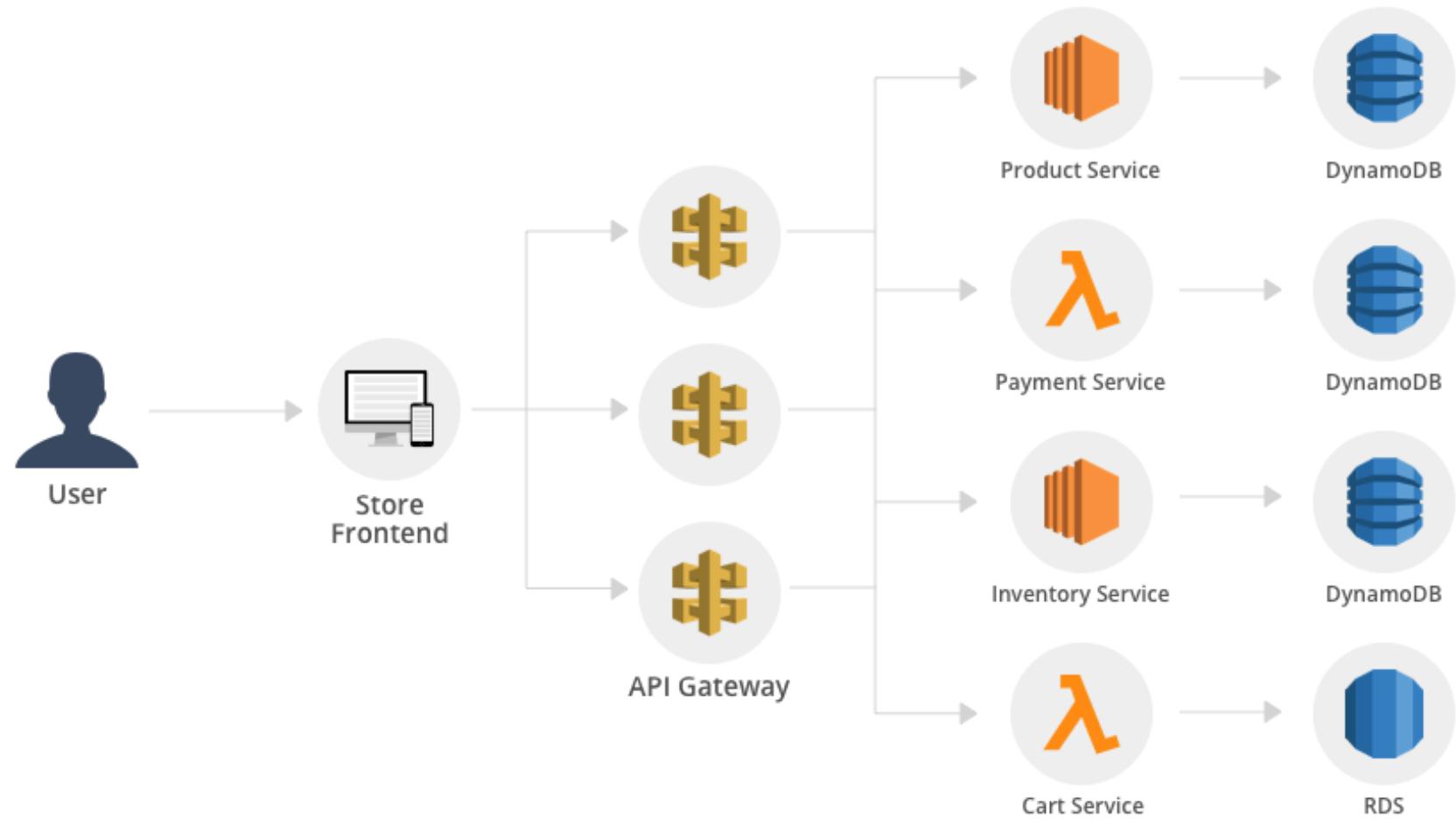


AWS Global Infrastructure

APN Partner Solutions



How to Build an Ecommerce App using Serverless?

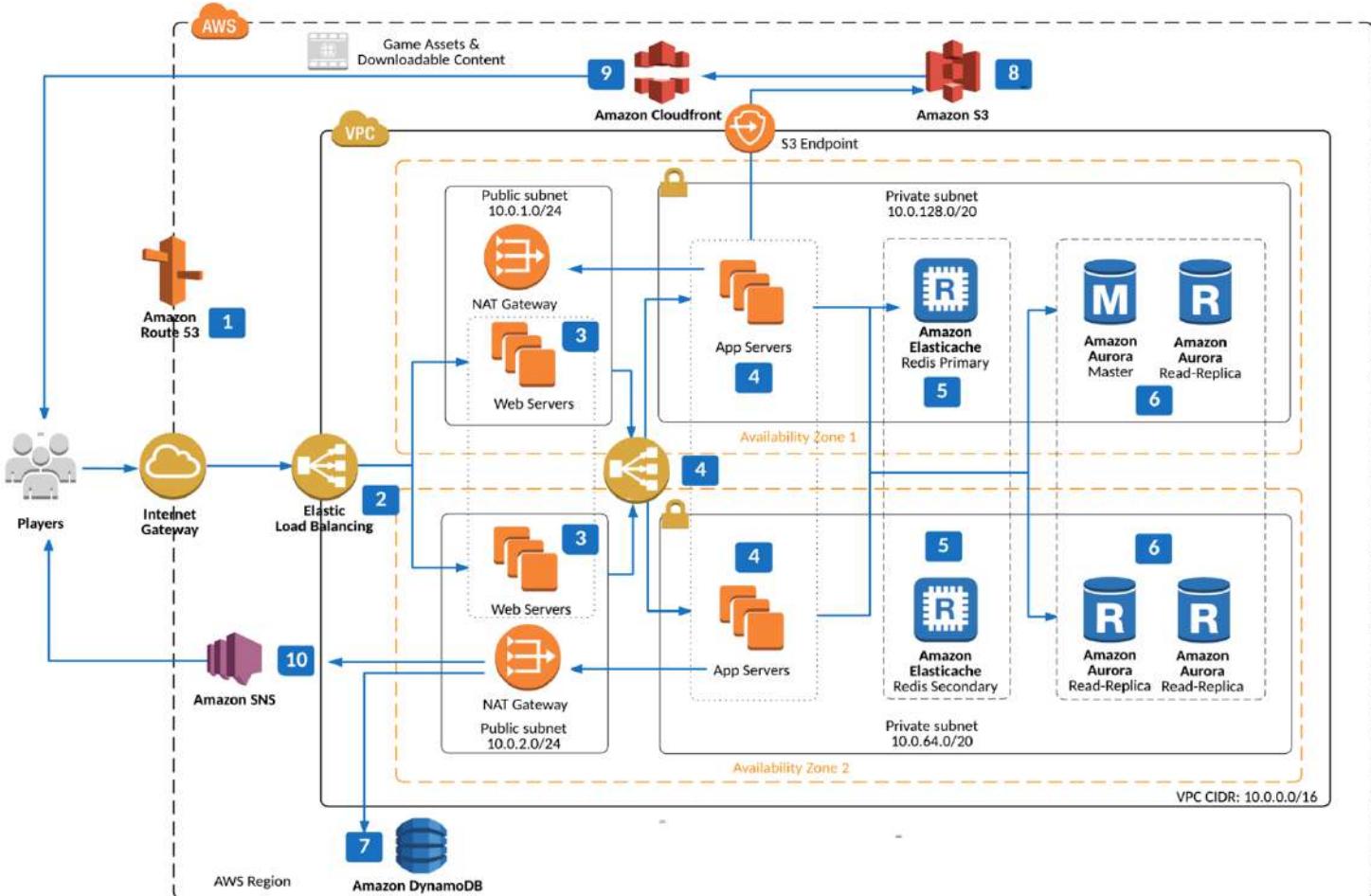




Asynchronous Online Gaming

Highly Available, Scalable & Elastic to Support Millions of Players

This architecture is intended for Mobile & Online Games. These workloads are a natural fit for running on Amazon Web Services, due to unexpected traffic patterns & highly demanding request rates. AWS provides the flexibility to start small & power up your architecture in response to your players. Scale up & scale down your architecture to make sure you are only paying for resources that are driving the best experience for your game. Use our managed services for popular caching & database technologies, & leverage this architecture that captures the best practices of some of the largest games running on AWS today.



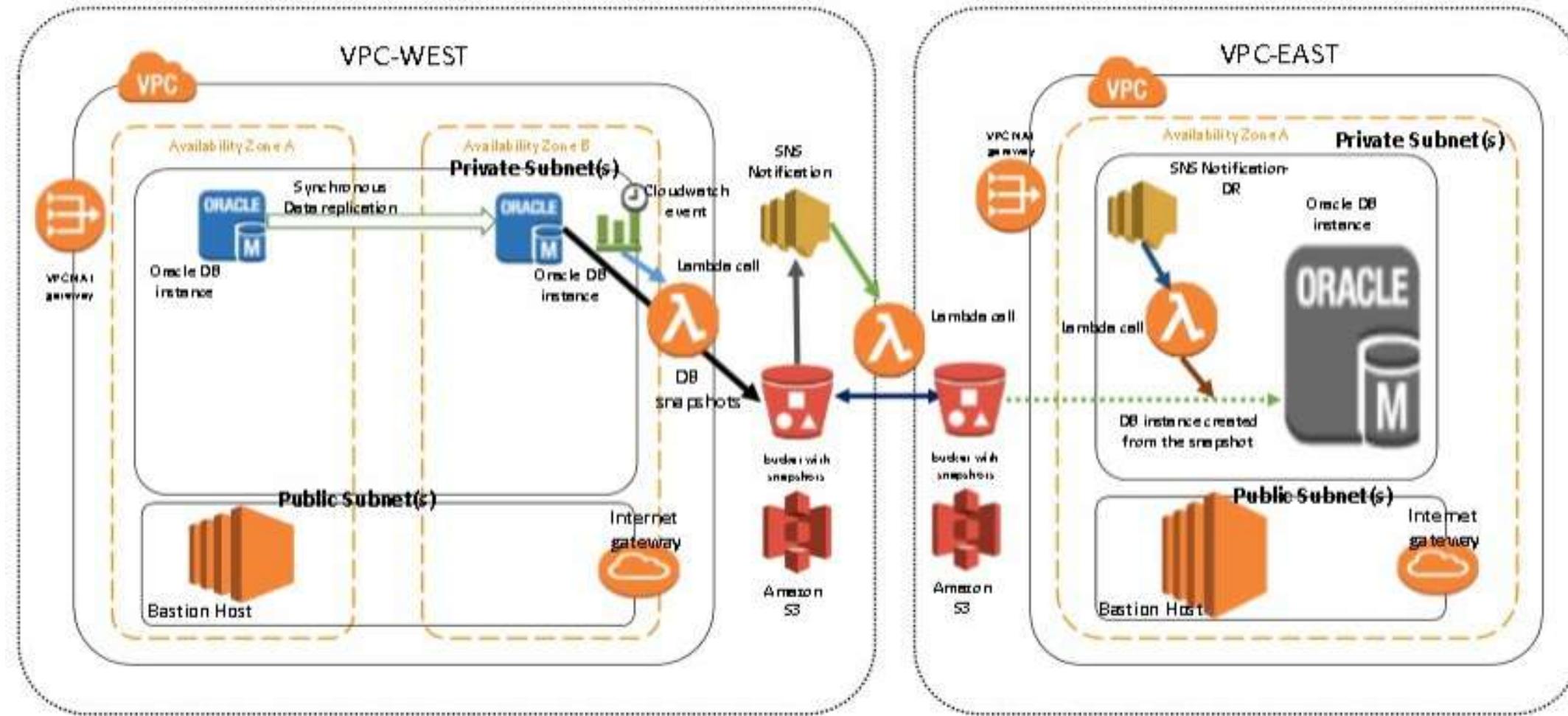
AWS Reference Architectures

- 1 Leverage our 100% SLA for Amazon Route53 to make sure your players are always able to discover your service endpoints. Use the in-built routing policies to route users based on latency or geography.
- 2 Route users to your backend using an **Elastic Load Balancing** that scales automatically for incoming traffic. Keep your players data secure in transit via HTTPS & by leveraging the SSL termination capabilities of the ELB.
- 3 Launch your Web Servers running on **Amazon Elastic Compute Cloud** (EC2) in an Auto Scaling Group that spans Multiple Availability Zones. We recommend the M4 instance type with Enhanced Networking & EBS Optimized enabled.
- 4 If you separate your app tier from your web tier, leverage an internal ELB. This ELB provides additional benefits of added security by residing in a private subnet & making sure no external traffic overwhelms your app tier.
- 5 **Amazon ElastiCache for Redis** provides a fully managed solution that enhances robustness & reduces the cost of installing, operating & maintaining a highly available & scalable Redis cluster. Leverage Multi-AZ ElastiCache in your game to provide automated disaster recovery & a scalable tier with read replicas if needed.
- 6 **Amazon Aurora**, a MySQL compatible database provides very high read & write throughput, up to 64TB 6-way replicated storage & up to 15 low latency read replicas in Multi-AZ. We recommend Aurora as a fast, scalable robust database if you are using a relational database like MySQL. Gaming customers have seen 2-3x reduction in cost after migration to Aurora.
- 7 Your game will also benefit from our high speed, low latency managed No-SQL database, **Amazon DynamoDB** that provides predictable performance & scalability.
- 8 Use **Amazon Simple Storage Service (S3)** to store your game assets, DLC & log files generated by your servers. As your user base grows geographically use Amazon Cloudfront as a globally distributed cache for content.
- 9 For Push Notifications, use our managed **Amazon Simple Notification Service (SNS)** service with out-of-the-box support for Apple, Google, Amazon & Windows platforms.
- 10





Oracle DB on RDS with DR using snapshot restore





Workshop – Deep dive into reference architecture

- Purpose: understand the intent behind the reference architecture
- Instruction:
 - Form Teams
 - Read the Case Study
 - Discuss the architecture within your groups to understand the purpose of different components
 - Use the AWS Products and Services Lingo to come up with a Reference Architecture
 - Present your findings
 - You can also share your experience if you've applied similar architecture in your work.

(Hint: You can choose to use online tools such as [Visual Paradigm](#))



SUMMARY



Concluding Remarks

- **Cloud native** is working across a broad swath of business types and organizations because it underlies application development that is tightly coupled to business goals.
 - With exceptional level of operations automation, cloud native can free highly trained developers and coders from the drudgery of operations management.
 - Cloud Native allows developing high-performance, highly innovative customer-facing applications capable of driving revenue and profit while building customer loyalty.
- **Kubernetes** (often abbreviated as k8s) is an open source project for running and managing containers.
 - Kubernetes is often viewed as a container platform, microservices platform, and/or a cloud portability layer. All of the major cloud vendors have a managed Kubernetes offering today.



APPENDIX



NIST Cloud Computing Reference Architecture

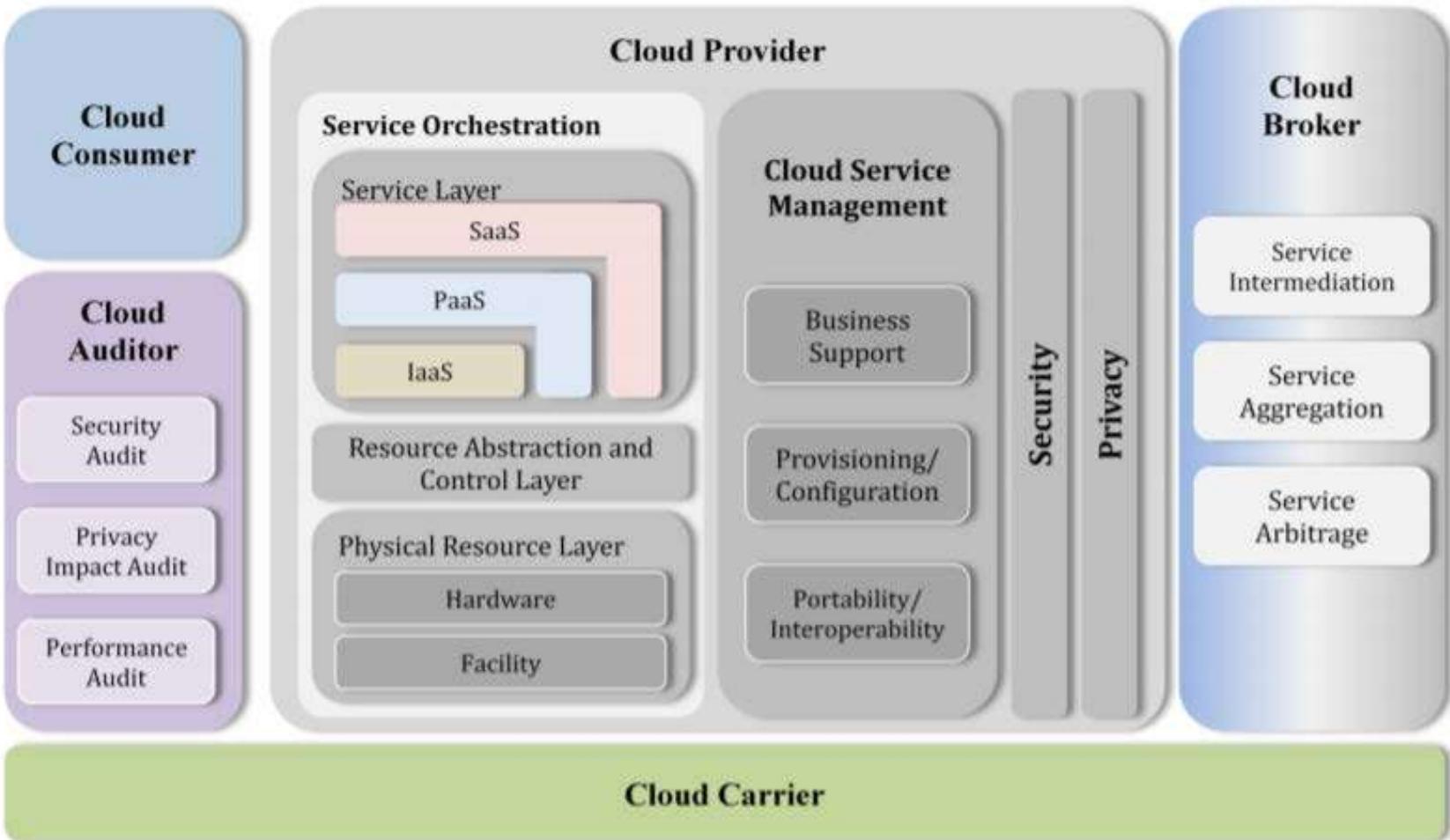
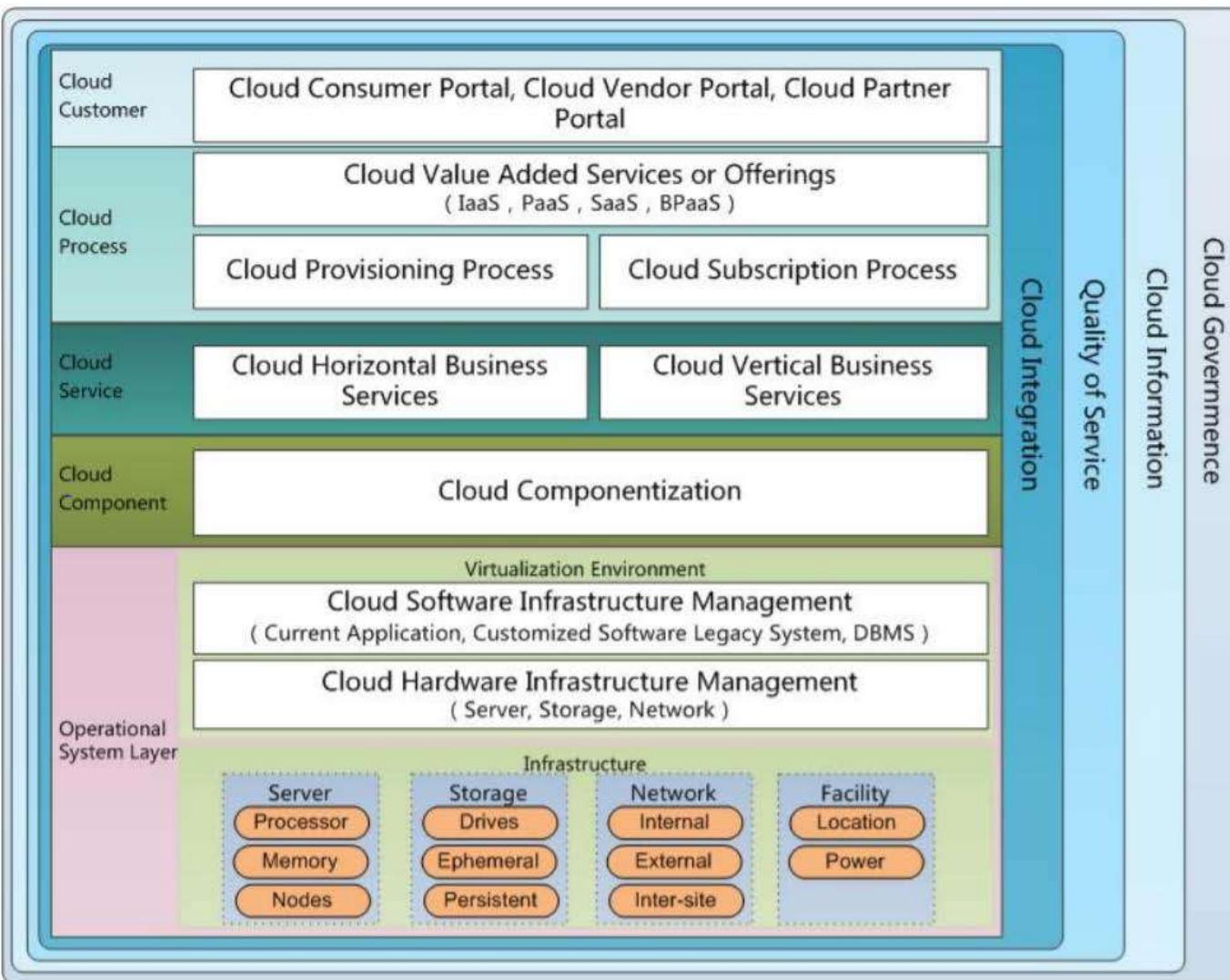


Figure 1: The Conceptual Reference Model



CCRA: Cloud Computing Reference Architecture





Cloud Native Solution Design

CLOUD ADOPTION AND MIGRATION

Suria R Asai

suria@nus.edu.sg

Institute of Systems Science

National University of Singapore

© 2009-23 NUS. The contents contained in this document may not be reproduced in any form or by any means, without the written permission of ISS, NUS, other than for the purpose for which it has been supplied.

Total Slides: 40



Agenda

- Cloud Migration Framework
- Cloud Migration Options
- Cloud Assessment Phase
- Application Concerns
- Cost and Benefit Analysis



Learning Objectives

- On completion of this module, the participant will be able to
 - Understand the consideration needed for cloud adoption
 - Understand the various options for cloud migration
 - Evaluate cloud readiness of an existing application systematically

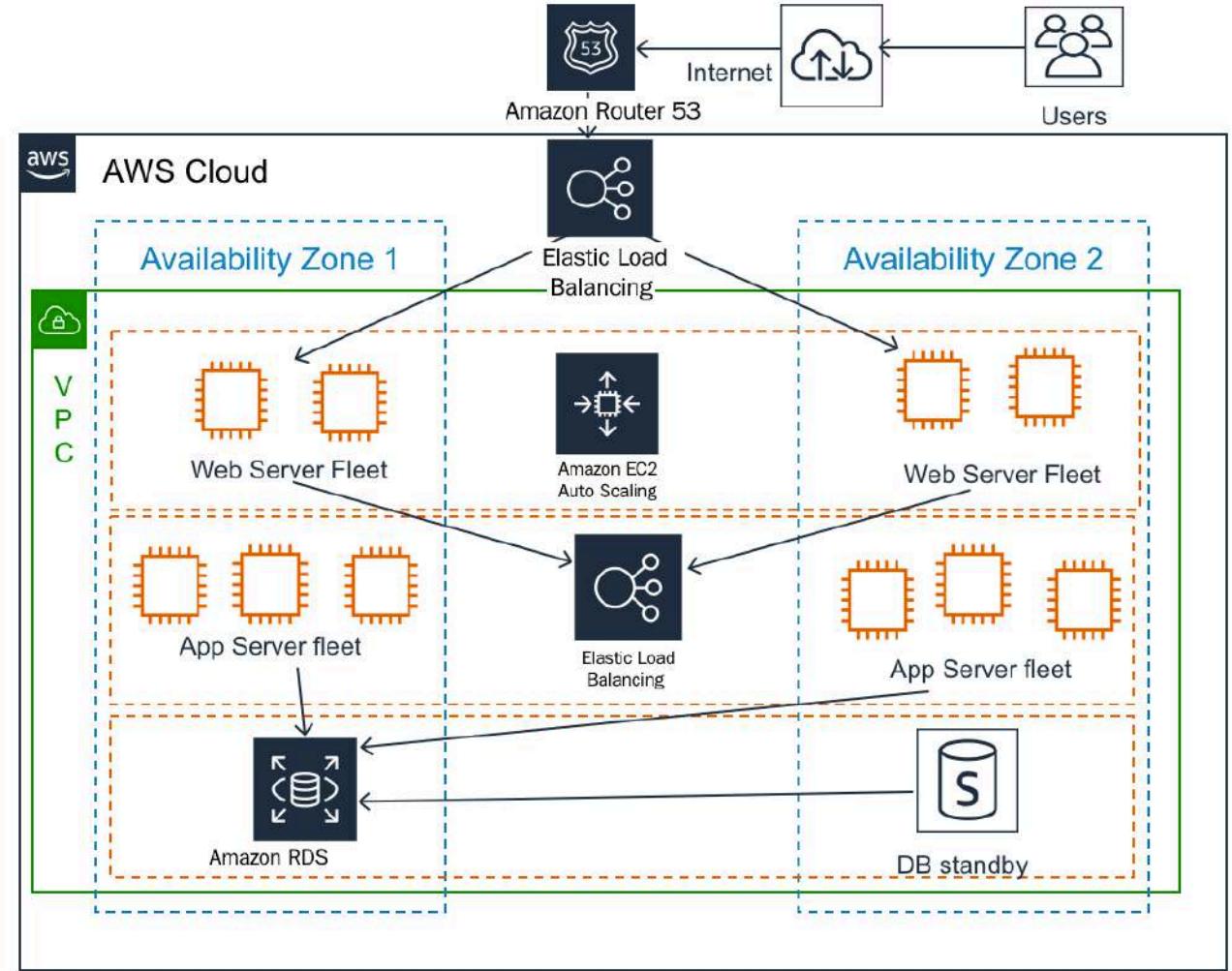
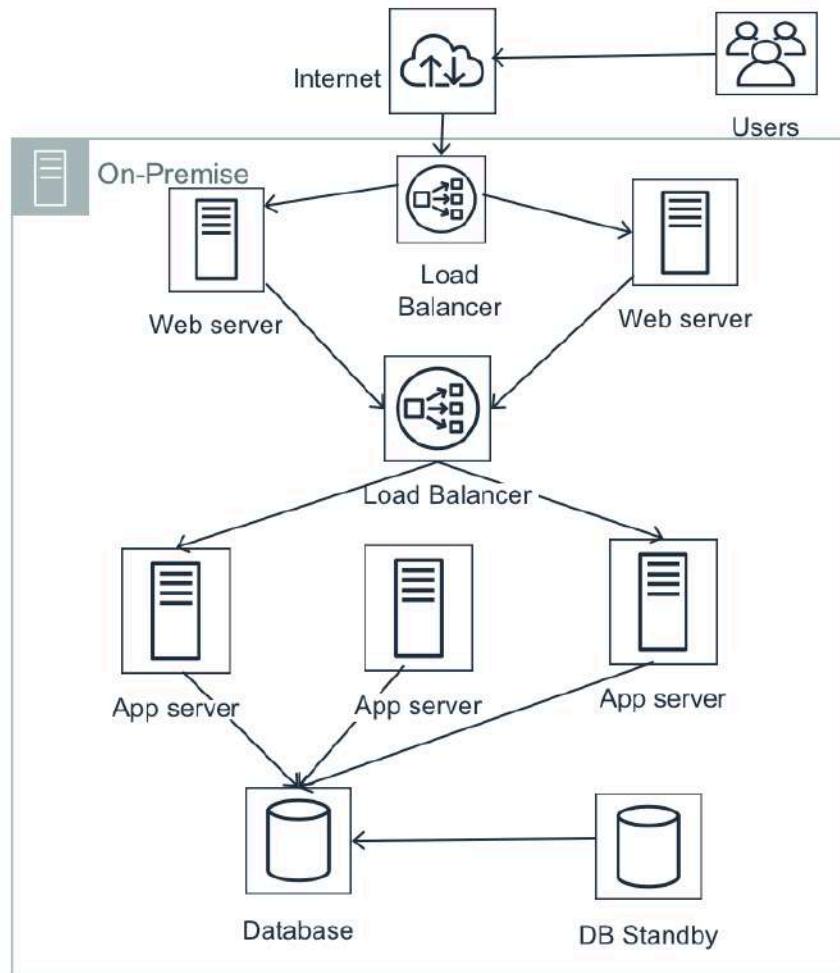


CLOUD MIGRATION FRAMEWORK

HOW DO WE THINK ABOUT AND APPROACH CLOUD MIGRATION?

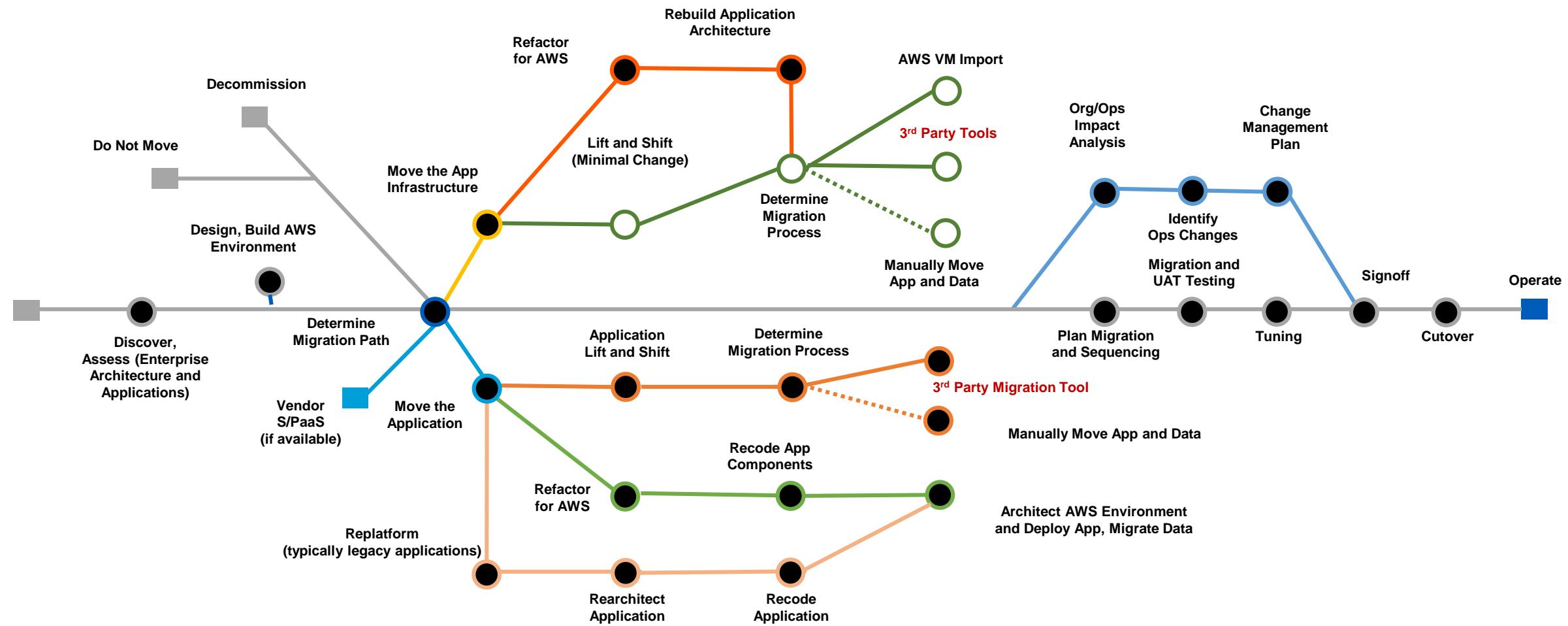


Example





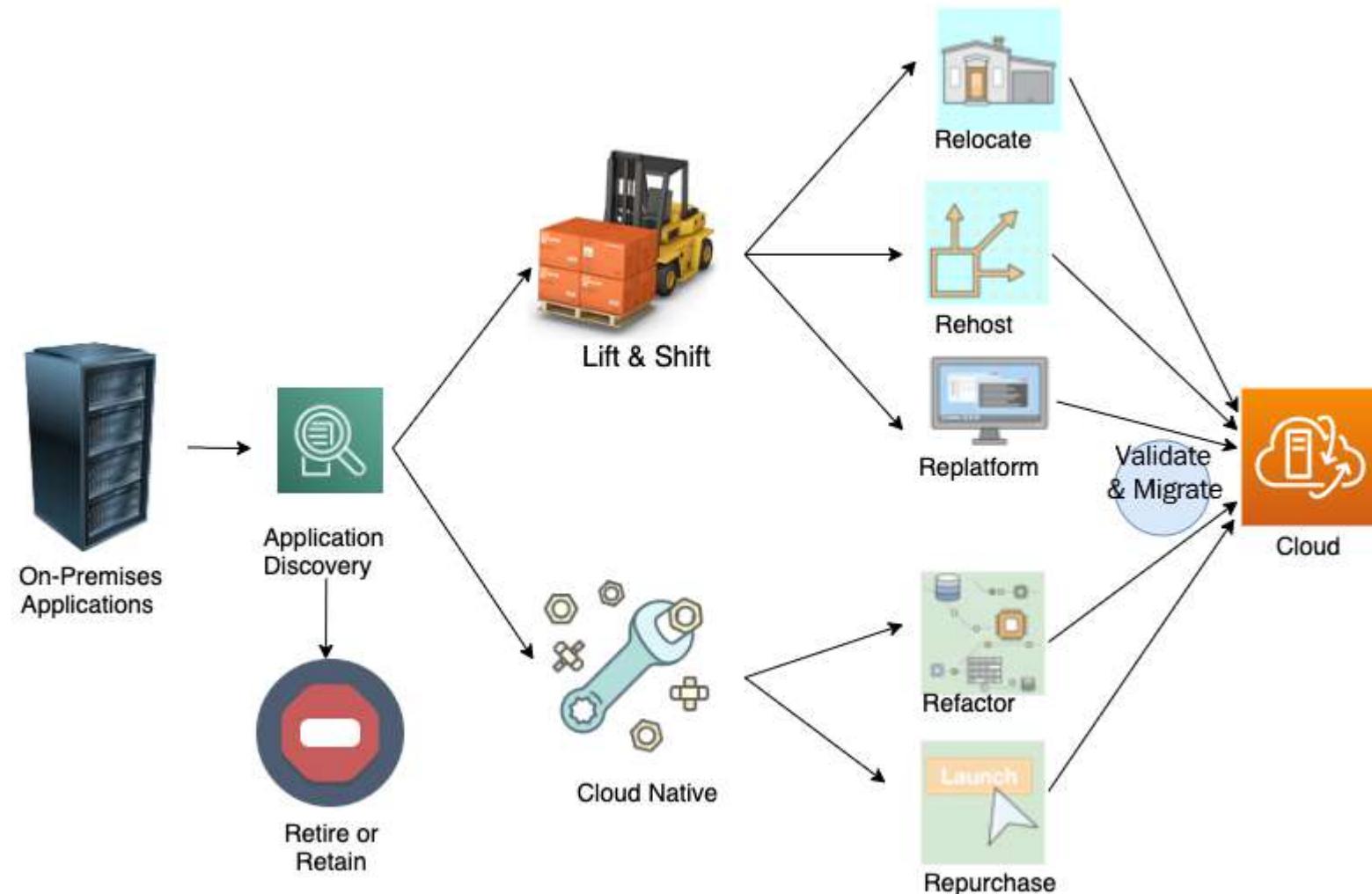
Cloud Migration Options



Source: AWS Migration Planning Roadmap – December 2015



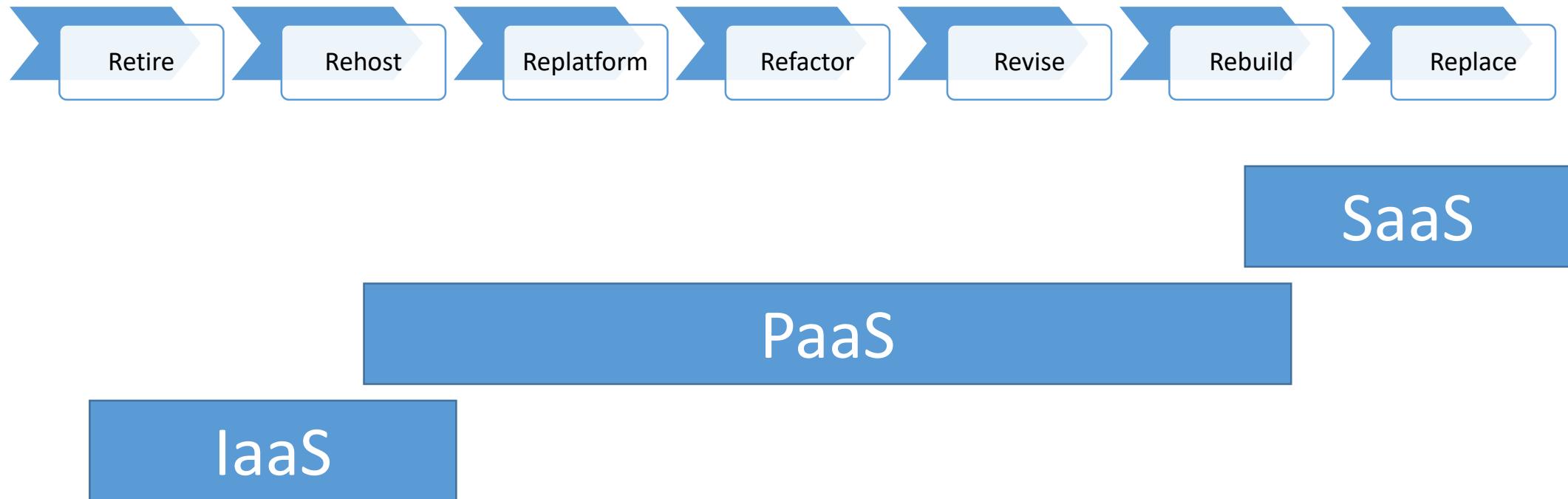
Cloud Migration Strategy





Cloud Model and Migration Options

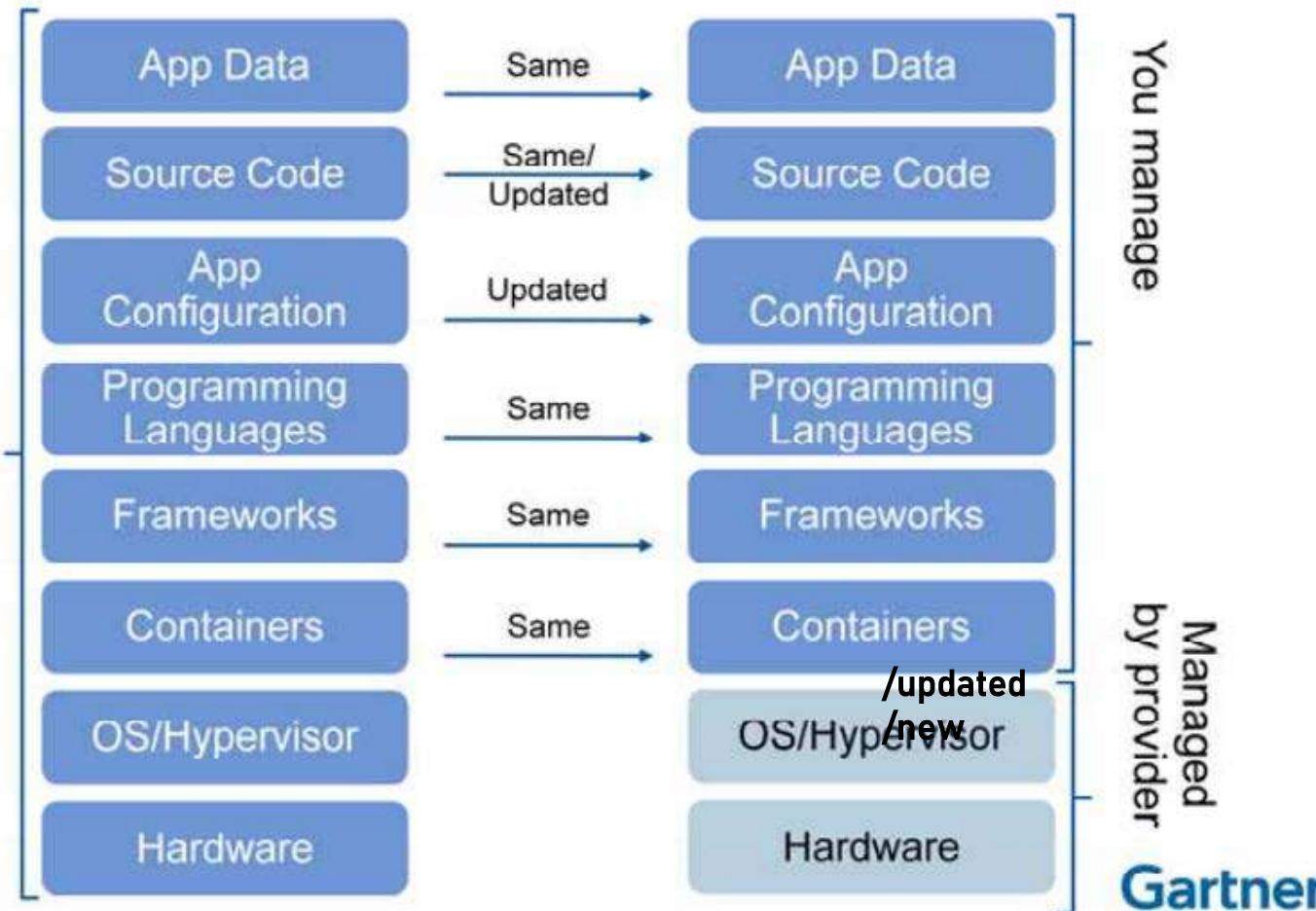
7Rs





Rehost/Replatform

Migrating Application



Source: Five Options for Migrating Applications to the Cloud – 2011

Customers typically use the **rehost** technique for the following reasons:

- The temporary development and testing environment
- For when servers are running packaged software, such as SAP and Microsoft SharePoint
- An application doesn't have an active roadmap

The following common reasons warrant the use of the **replatform** technique:

- Changing the operating system from 32-bit to 64-bit OS
- Changing the database engine
- Updating the latest release of the application
- Upgrading an operating system from Windows 2008 to Windows 2012 or 2016
- Upgrading the Oracle database engine from Oracle 8 to Oracle 11
- To get the benefits of managed services that are available from cloud vendors such as managed storage, databases, application deployment, and monitoring tools

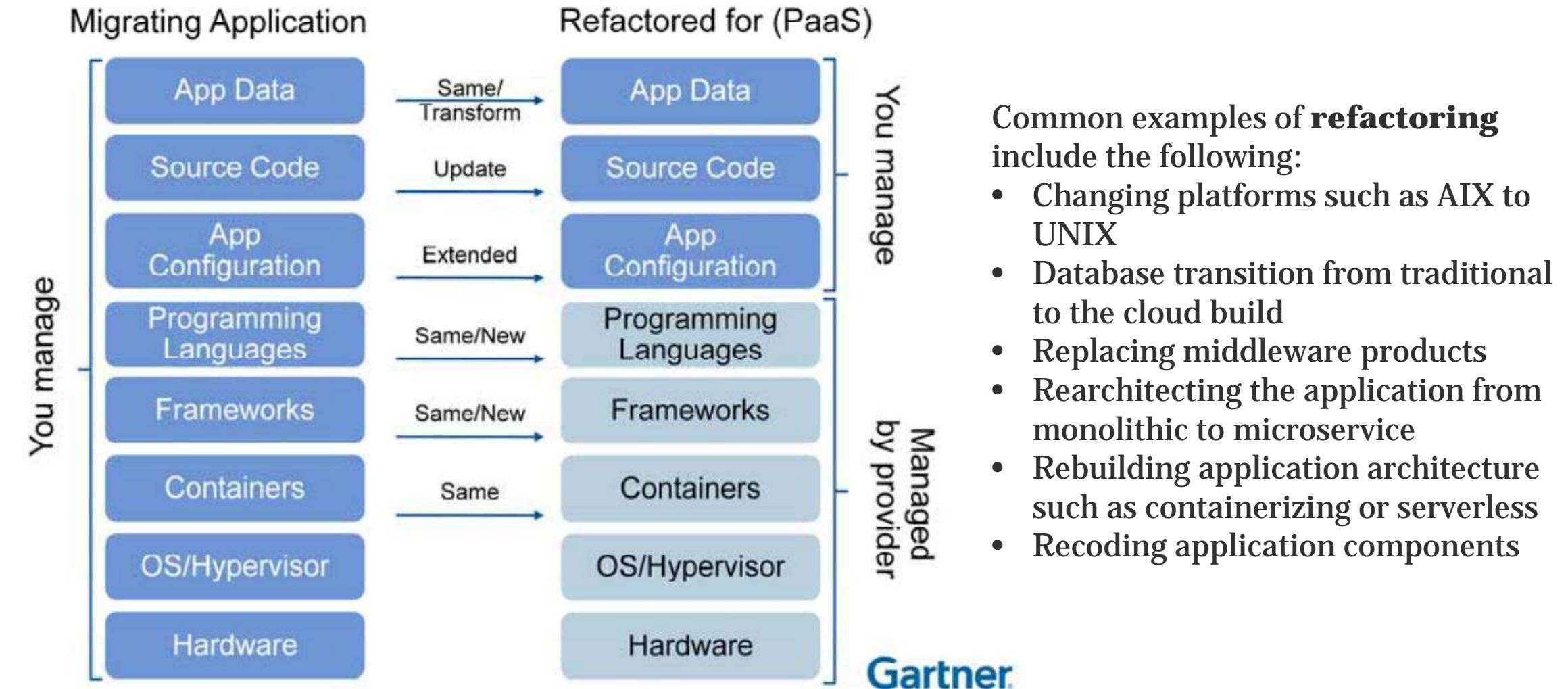


Rehost / Replatform

- Advantages:
 - Can work with systems where code modifications are impossible
- Disadvantages:
 - If done without changes to application architecture scalability benefits may be lost.
 - Assembling a complete application stack is necessary, choose from one off the peg, or engage a third party provider to assemble one. VM image format and management API lock-in is a risk
- Example
 - Deploying an existing Java EE container with a Java EE application on EC2 Linux instances from Amazon Web Services (AWS), backed by Elastic Block Store (EBS) for persistent VM images.



Refactor



Source: Five Options for Migrating Applications to the Cloud – 2011



Refactor

- Advantages:

- Reuses languages, frameworks and containers that developers have invested in, thus leveraging code the organization considers strategic

- Disadvantages:

- Immature PaaS offerings: Missing platform capabilities, transitive risk when PaaS providers build on IaaS providers, and framework lock-in

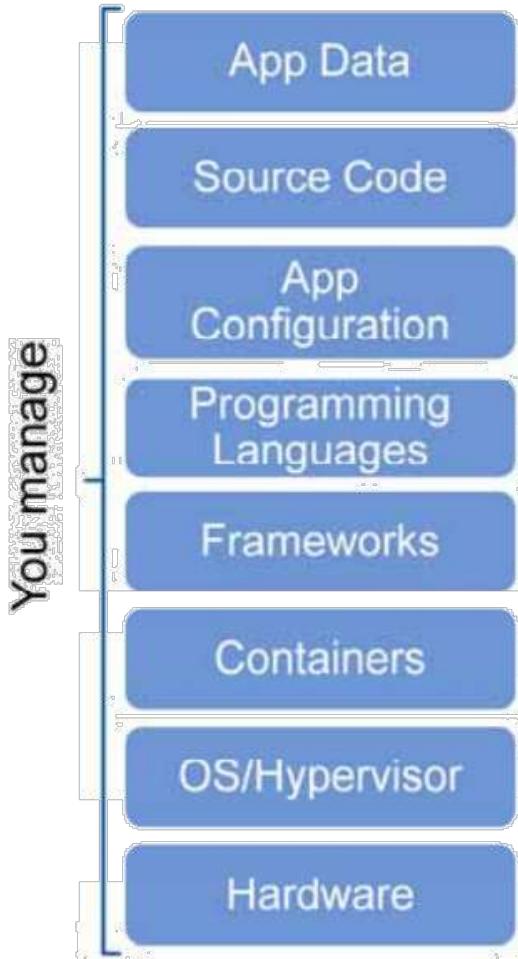
- Example

- Deploying Ruby on Rails Web application to Heroku (including linking a monitoring service based on New Relic) deploying Active Server Pages for .NET (ASP.NET) application to Windows Azure.

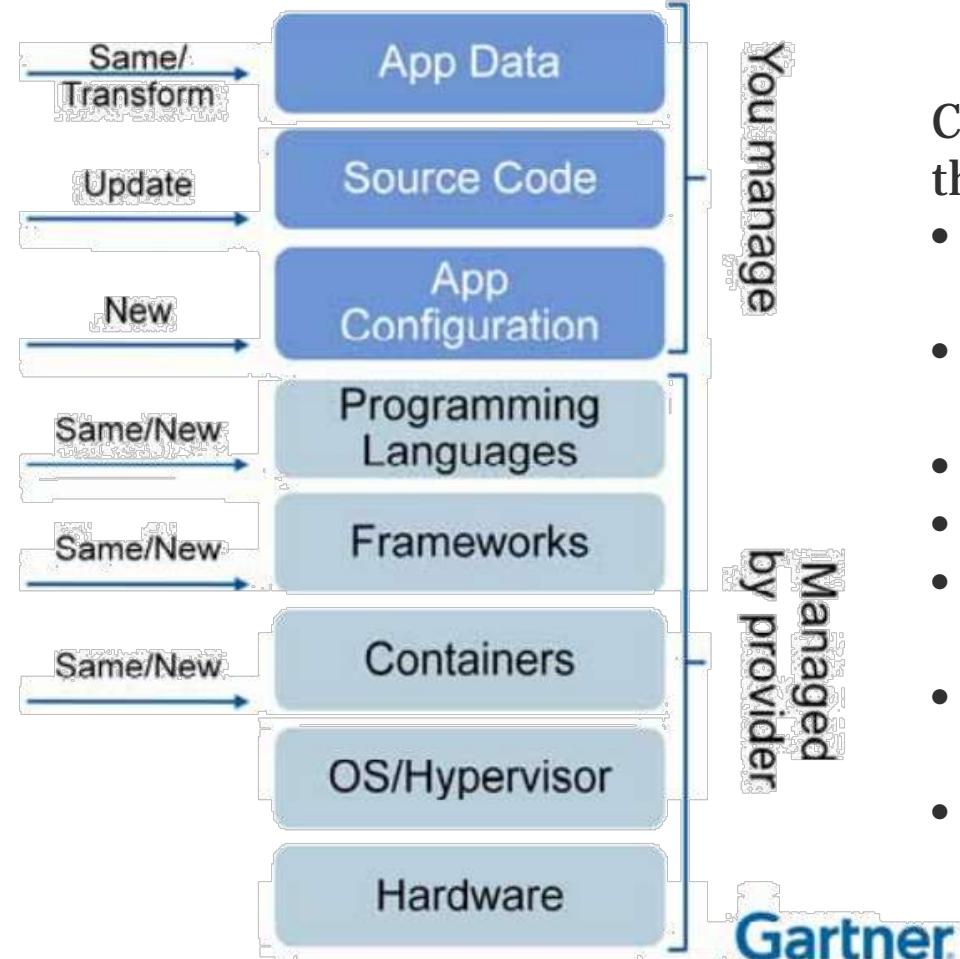


Revise

Migrating Application



Rebuilt on (PaaS)



Gartner

Source: Five Options for Migrating Applications to the Cloud – 2011

Common examples of **revising** include the following:

- Framework from MVC to Reactive Stack
- Database transition from traditional to the cloud build
- Change in programming language
- Replacing middleware products
- Replacing the application configuration
- Rebuilding application architecture such as containerizing or serverless
- Recoding application components



Revise

▪ Advantages:

- Taking advantage of a valuable codebase by enhancing it to meet other cloud adoption or legacy modernization goals. Compared with the other migration alternatives, revise puts the organization in a position to optimize the application to leverage the cloud characteristics of providers' infrastructure

▪ Disadvantages:

- The downside of revising an application is that kicking off a (possibly major) development project will require upfront expenses to mobilize a development team.
- Depending on the scale of the revision, revise is the option likely to take most time to deliver its capabilities. Immature PaaS offerings will also slow down progress

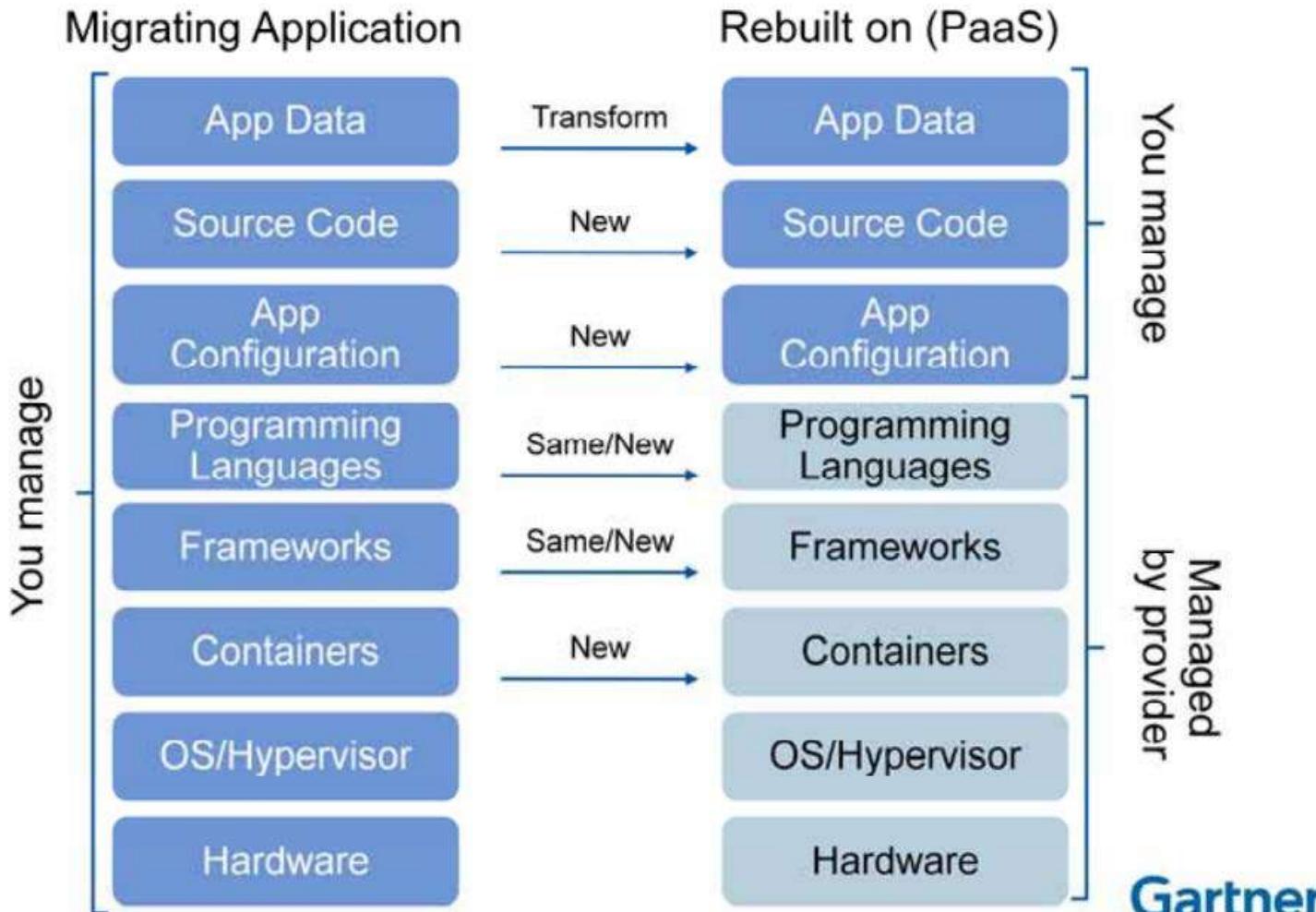
▪ Example

- Redesigning a monolithic Java application decomposing the functions into smaller parallel chunks. Redesigning a monolithic Java application, decomposing the functions into smaller parallel chunks, and then deploying on Rackspace Cloud Servers.



Rebuild

Migrating Application



Source: Five Options for Migrating Applications to the Cloud – 2011

Common examples of **revising** include the following:

- Framework from MVC to Reactive Stack
- Database transition from traditional to the cloud build
- Change in programming language
- Replacing middleware products
- Complete new the application configuration
- Rebuilding application architecture such as containerizing or serverless
- New application components



Rebuild

▪ Advantages:

- Developer productivity is improved with tools that allow application templates and data models to be customized, metadata-driven engines, and communities that supply prebuilt components.
- Transparent automatic scalability.
- Allows non-professional developers the opportunity to develop and deploy simple applications into production.
- Multitenancy means the provider manages upgrades and patches.

▪ Disadvantages:

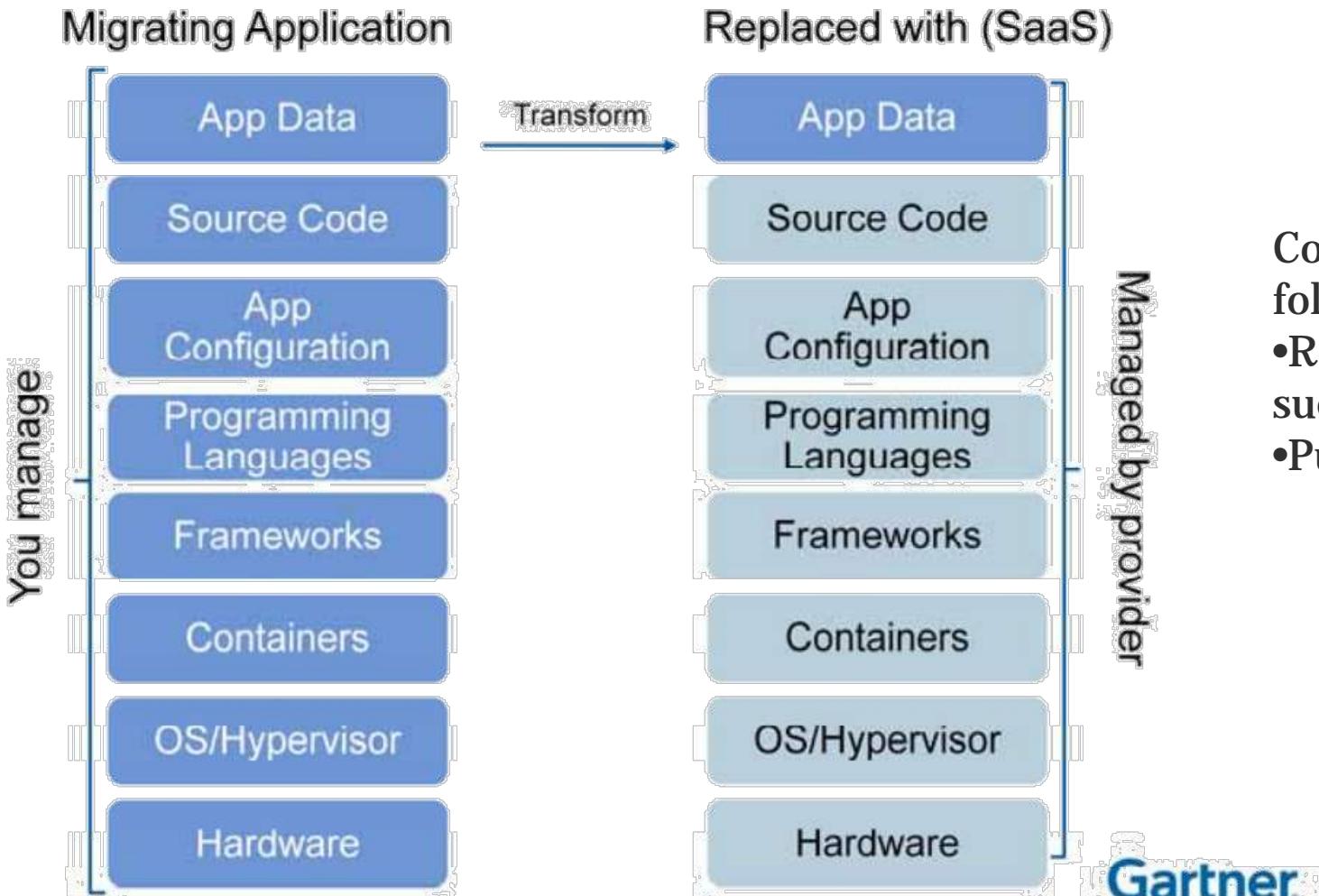
- Lock-in and PaaS immaturity are the primary disadvantages.
- Abandoning familiar programming languages and frameworks means that second sourcing strategies to mitigate lock-in risk may not work.

▪ Example

- Building a force.com application for order management
- Modernizing a C and FORTRAN financial risk calculation application by redesigning it in C#, then using Windows Azure platform libraries and tools to deploy it to Microsoft's cloud.



Replace



Source: Five Options for Migrating Applications to the Cloud – 2011



Replace

- Advantages:

- Unlike custom-built software or single-tenant COTS, SaaS providers can monitor and leverage group behavior within a single tenant (one department or company), or across many tenants

- Disadvantages:

- Possible data lock-in;
- Unless the plan includes a firm schedule for retiring replaced applications, adding more complexity to the applications portfolio landscape;
- Incompatible process, policy or data models.
- Can be difficult to customize or reconfigure;
- Can be difficult to integrate with existing systems and processes, but no more and no less than integrating on premises applications

- Example

- Salesforce CRM, SugarCRM for customer records management, Workday for HR processes, Omniture for Web analytics, LiveMeeting for Web conferencing.



Possible Decision Outcomes

Decision	Definition
Rehost (platform)	Rehost implies redeployment of the application to a different hardware (platform) environment and implies redeployment of the application to a different hardware (platform) environment and changing the application's infrastructure configuration.
Refactor	<p>"Refactor" describes running your applications (usually Web applications) on the cloud provider's infrastructure. You must make application code or configuration changes to connect the application to new infrastructure services.</p> <p>Similar to linking in a new database driver, identity management system, or authentication module. Also similar to moving a Java EE application from IBM WebSphere to Red Hat JBoss (the same type of container, some different frameworks and configuration).</p> <p>Necessary changes vary from none to widespread code changes to invoke new APIs. Refactor is "backward-compatible" PaaS. Existing programming models, languages and frameworks can be used and extended</p>



Possible Decision Outcomes

Decision	Definition
Revise	<p>"Revise" means to modify or extend the existing codebase to support legacy modernization means to modify or extend the existing codebase to support legacy modernization requirements, then use rehost or refactor options to deploy to the cloud.</p> <p>The scale of changes encompasses major revisions to add new functionality or to re-architect the application for the cloud</p>
Rebuild	<p>"Rebuild" your solution on a provider's application platform while discarding code for an existing application platform. Rebuild requires rearchitecting the application for a new container (e.g., from Java to .NET).</p> <p>Tends to result in "Forward-compatible" or incompatible PaaS.</p> <p>Not all existing programming models, frameworks, and languages will be retained.</p>



Possible Decision Outcomes

Decision	Definition
Replace	<p>"Replace" means to discard an existing application (or set of applications) and use commercial means to discard an existing application (or set of applications) and use commercial software delivered as a service to satisfy those business requirements. Typically, existing data requires migration to the SaaS environment. Application data import/export is achieved with an API or configuration/admin console.</p>
Retain	<p>No changes made to application</p>
Retire	<p>The functionality of the application is no longer needed by the business.</p>



Workshop

- Discuss within your group on the following points:
 - What is the **most suited migration** strategy for Tick – it? Discuss which of the 7R approaches you consider to be the appropriate choice – see if you can pick different application that is suitable for different approaches
 - Share the applications that you discussed, the challenges, and the chosen approach to the class



Simple Seven Steps – Migration Plan - 1

important

▪ **Step 1: Identify desired outcomes for moving to the cloud.**

- Before we can hope to succeed at something, we need a goal. What is it that we are trying to achieve? Be as specific as possible.

▪ **Step 2: Classify candidates for the cloud.**

- We need to decide which applications or workloads are the best candidates for the cloud. Which migrations will help us meet our stated outcomes from Step 1?

▪ **Step 3: Define deployment model criteria.**

- What's the best type of cloud for each workload? What is best suited for IaaS? PaaS? SaaS? Private cloud? Figure it out now, before the move, to avoid costly mistakes.

▪ **Step 4: Devise an iterative, agile process for transformation.**

- Now it's time to look internally. How can we organize our teams so that they effectively mirror Agile software development methodologies during the migration? So that they both fail fast and learn fast?



Simple Seven Steps – Migration Plan – 2.

▪ Step 5: Get buy-in to the plan.

- This is where our top-down or bottom-up strategy has to prove that we've got the support we need throughout the organization for a successful migration.

▪ Step 6: Start simply and prioritize learning over Return on Investment (ROI).

- We'll hear advice to "go for the project with the biggest bang for the buck," or the highest ROI. No. Pick the project that is relatively risk-free, and, most important, from which we'll learn the most.

▪ Step 7: Rinse and repeat.

- Success breeds success. After we've triumphed with one workload or application, we move onto the next. And the next.



Classification – Eight Criteria

Step 2: Classify candidates for the cloud

- Architecture
- Security
- Availability
- Performance
- Scalability
- Strategic importance
- Future plans
- Enthusiasm of team



Application Concerns

Performance	Architecture	Financial	Risk	Operations	Security and Compliance
Elasticity Scalability Resource Intensiveness Latency Throughput	User Interface	Operating Cost	Organizational	Business Continuity	Jurisdiction
	Access Points (Mobile or Offline)	Business Value	Business Criticality	Tools/ Integration	Regulation
	Application	Complexity Size Application Life Expectancy	Technical Resource	Deployment	Privacy
			Contractual	Audit	Encryption
			Audit		
	Data				
	Structured Magnitude				
	Unstructured Requirements				
	Complexity				
	Infrastructure				
	Hardware Life Expectancy				



Container VS Virtual Machine (and the in between)

Step 3: Define deployment model criteria

Virtual Machines

Hypervisor (Type 1 & 2)

Hardware

Machines M1 M2 M3 ...

Server

- Dedicated Hardware
- Heavyweight (GBs)
- Runs its own OS and requires more memory
- Example VMWare VSphere Virtual Box Hyper-V
- Starts in minutes
- Provides all OS resources to applications with established management and security tools. Fully isolated and secured.

KEY POINTS

1. The level at which virtualization happens - virtualization happens at hardware level vs. OS level
2. The type of isolation achieved - isolation of machines vs. isolation of processes
3. How resources are accessed - via hypervisor vs. via kernel features such as namespace and cgroups
4. Flexibility of hardware vs. portability

Containers

Containers C1, C2,C3,C4 ...

OS

Kernel (namespace c groups)

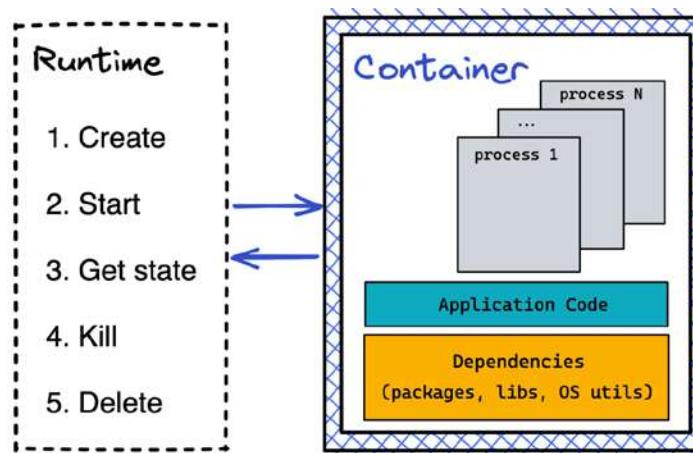
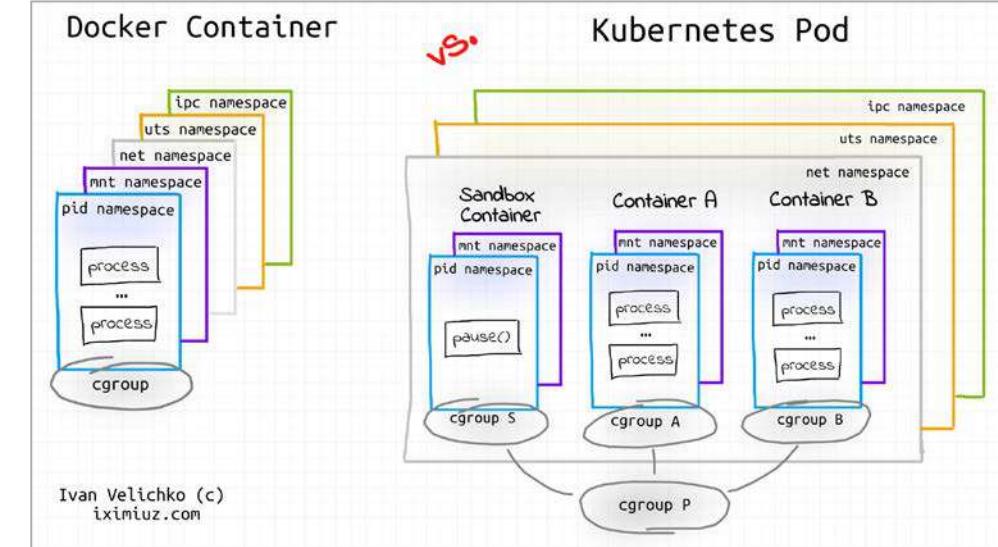
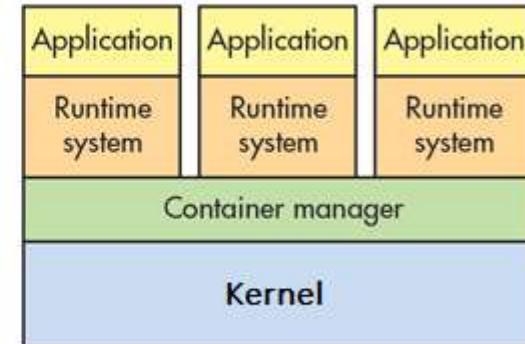
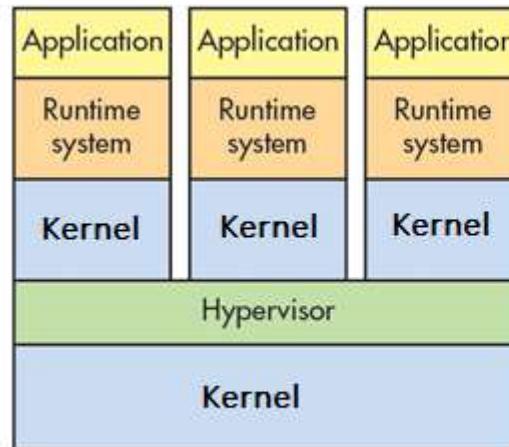
Hardware

- Encapsulated application in own operating environment
- Lightweight (MBs)
- Provides OS virtualization and requires less memory
- Example LXC, LXD, CGManager, Docker
- Starts in milliseconds
- Reduces IP management, simplified security and management, requires minimum code to upload workloads. Process level isolation with less security.

VM is isolation of machines, while Containers is isolation of processes.



Container vs Type 2 VM

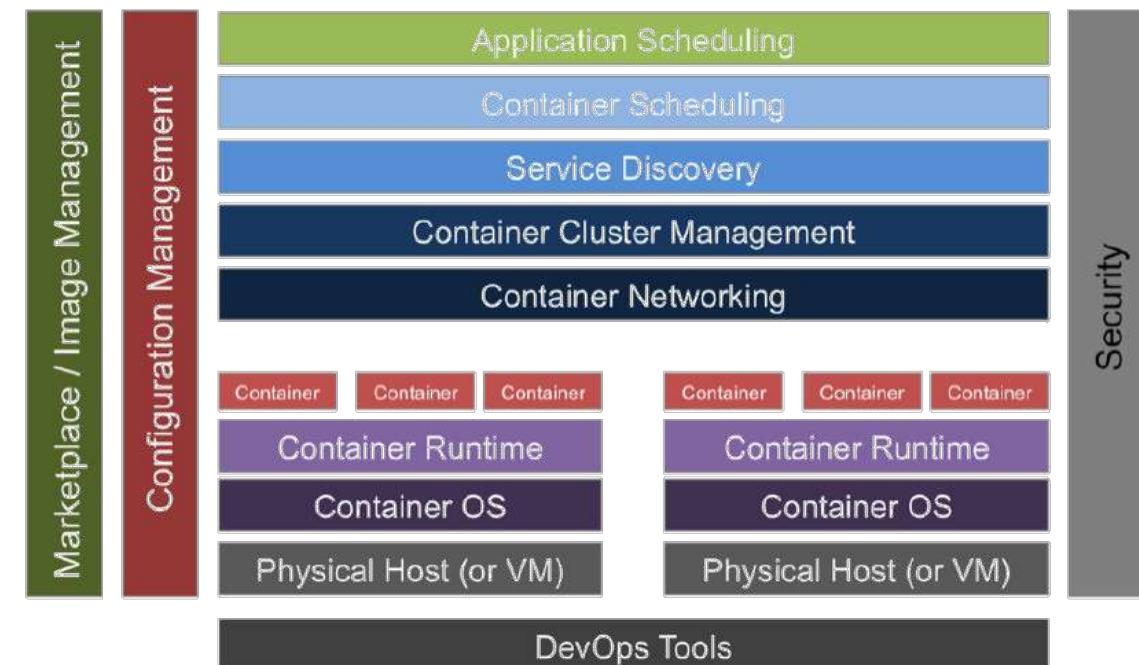


Containers work the same way on

laptop server cloud

with

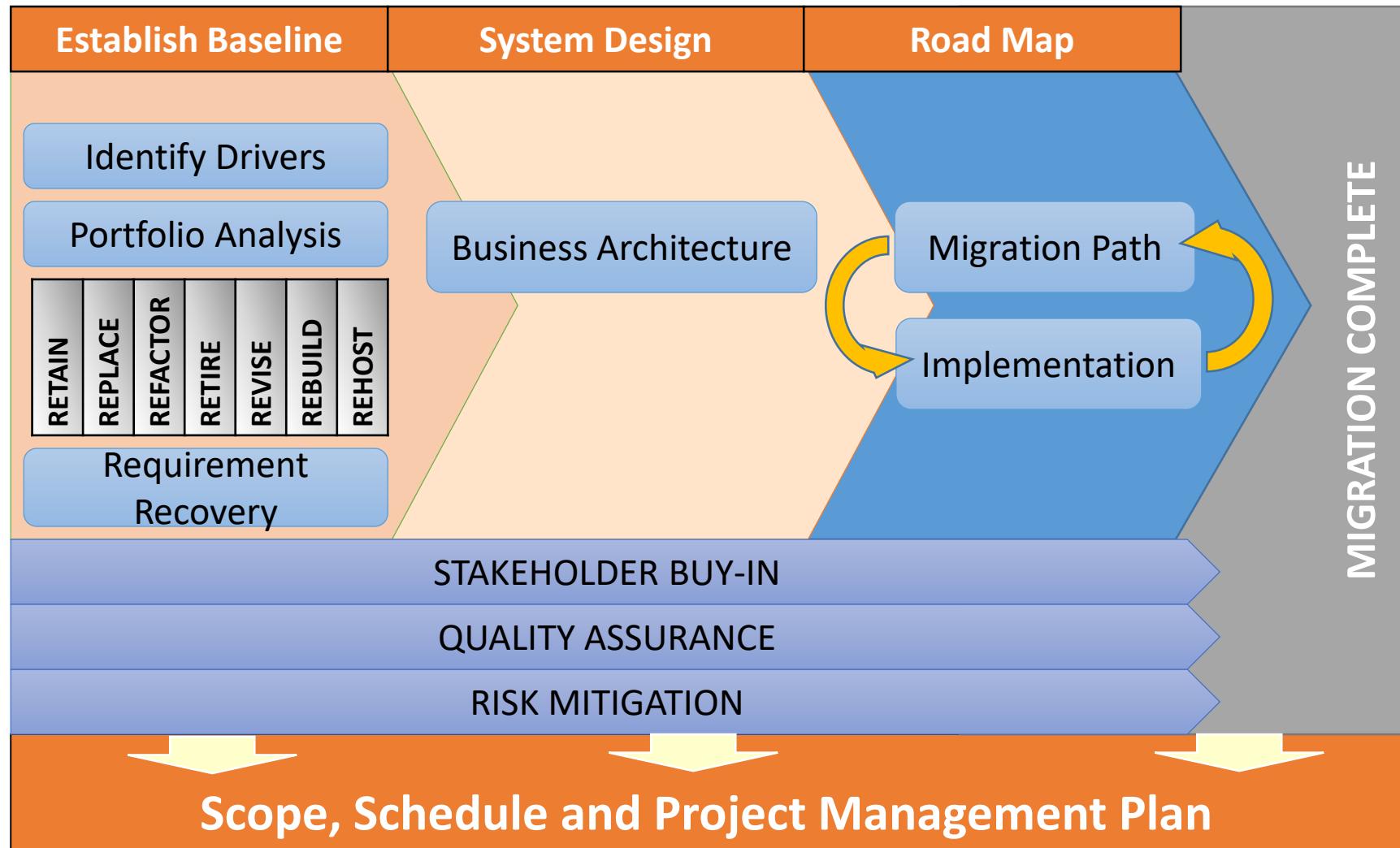
Docker/Kubernetes/Podman etc.





Migration Planning Framework

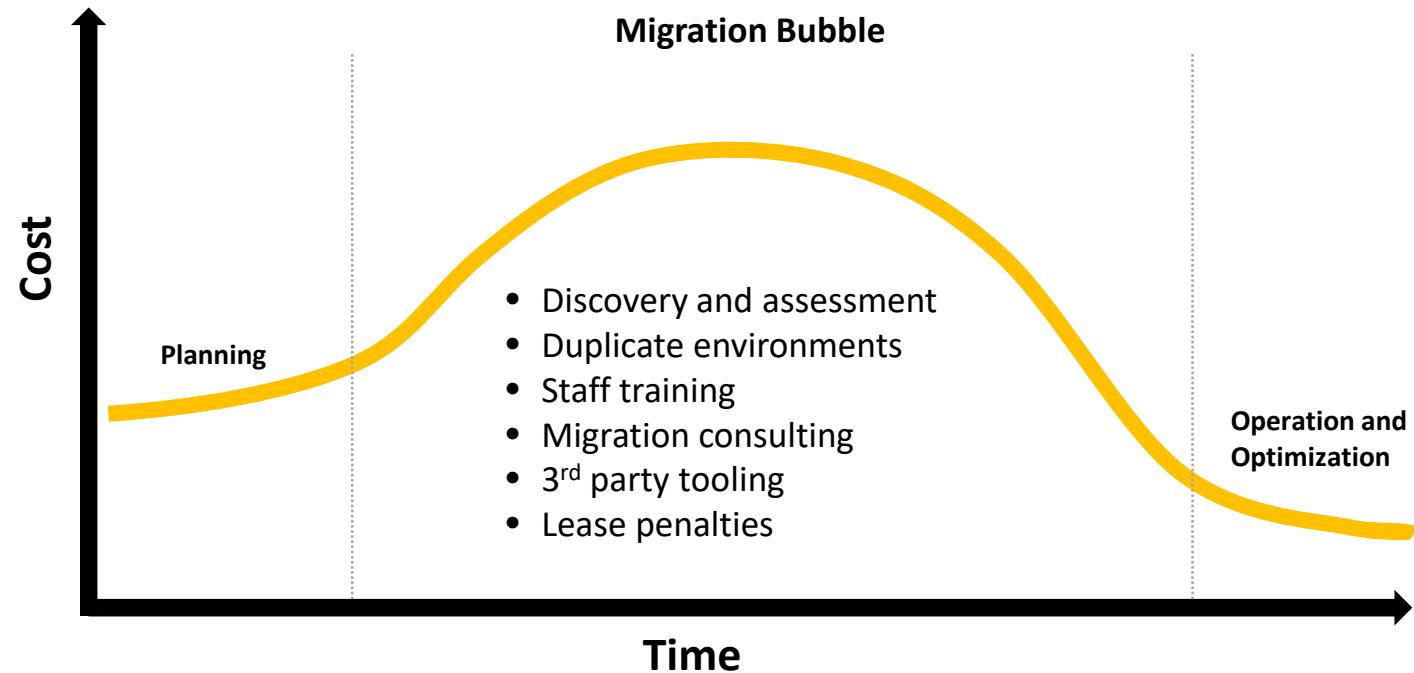
Target Architecture



Source: Project Planning for System Migrations – Feb 2015

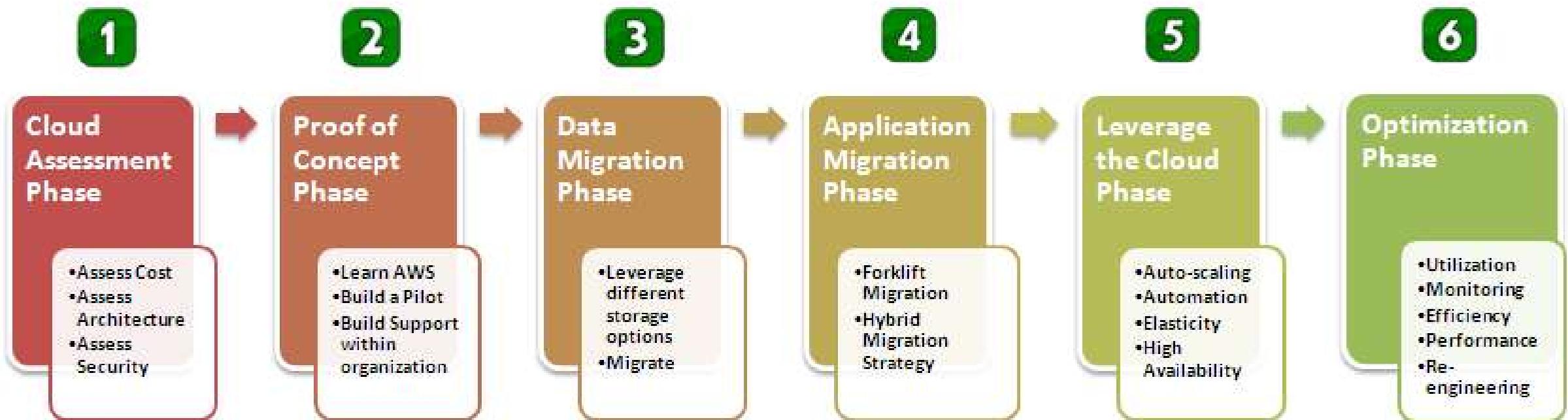


The Migration Bubble





Cloud Migration Stages



Source: Migrating your Existing Application to the AWS Cloud – October 2010



Cloud Assessment Phase

- Financial Assessment
 - Total Cost of Ownership (TCO) calculation
- TCO Calculators
 - AWS: <https://awstcocalculator.com/>
 - Azure: <https://www.tco.microsoft.com/>
 - Google: <https://cloud.google.com/pricing/tco/>
 - IBM Cloud SoftLayer: <http://www.softlayer.com/tco/>

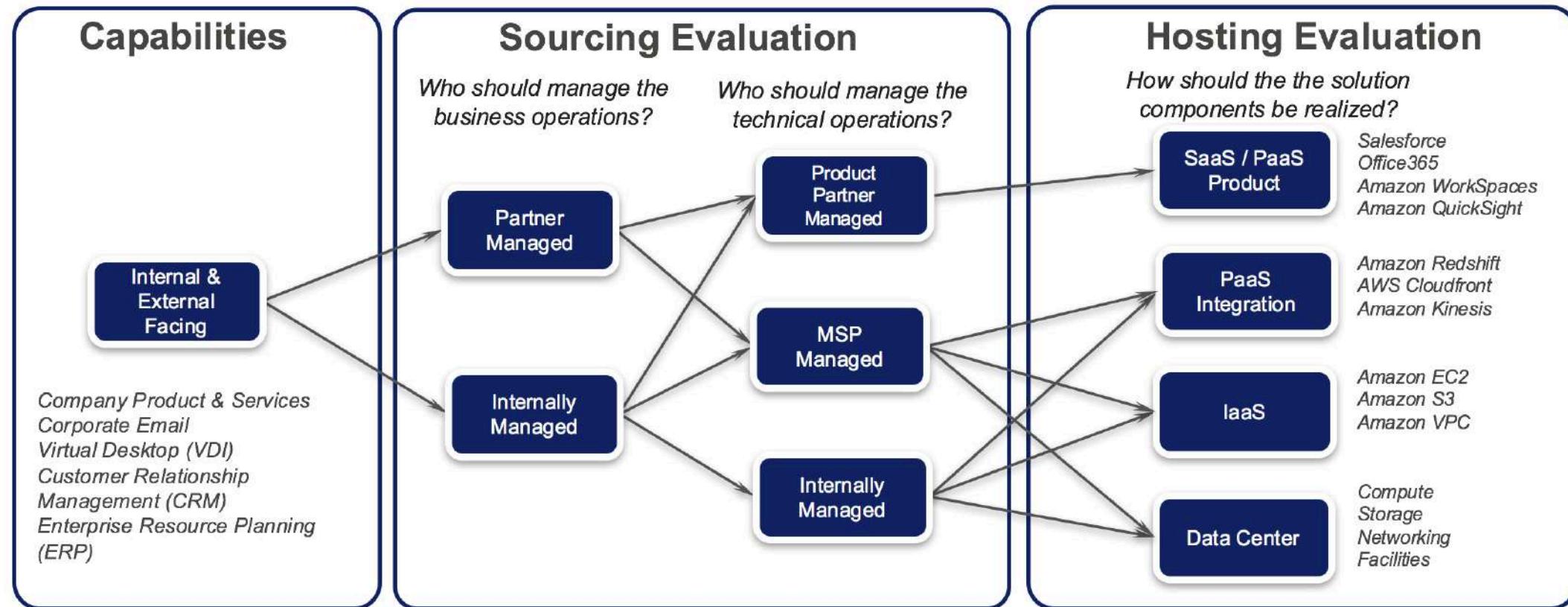


Sourcing Model and Hosting Model

- Sourcing Model
 - Deciding who should manage business and technical operation of a system/capability
 - Two models:
 - Internally managed
 - Partner managed
 - ✓ Managed Service Provider
 - ✓ Cloud Service Provider (Product Partner)
- Hosting Model (a.k.a. Cloud Service Model)
 - Data Center
 - Infrastructure as a Service
 - Platform as a Service
 - Software as a Service



Define the Sourcing Model and Hosting Model



Source: Migration Planning – June 2016



Defining Sourcing Strategy and Hosting Strategy

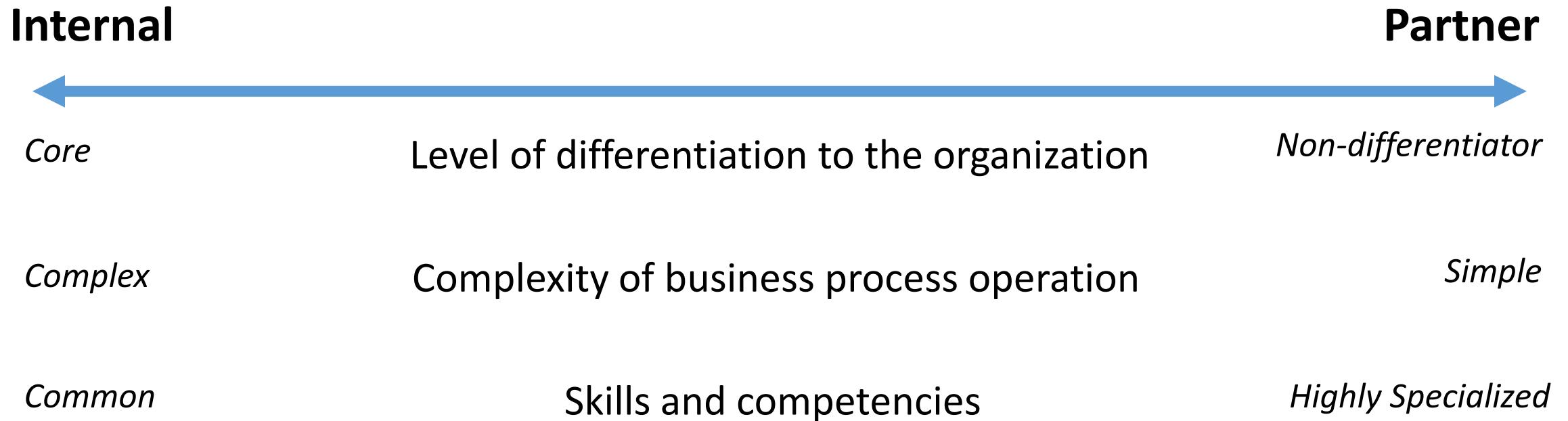
- Define the factors that may influence the decision on choosing the sourcing and hosting strategy.

Example:

- Level of differentiation to the organization
 - Core differentiator should be managed internally
 - Non differentiator should be outsourced to partner
- Complexity of business process operation
 - Complex processes should be managed internally
- Skills and competencies
 - Common skills can be retained in-house
 - Highly specialized skills should be outsourced



Arrange the decision spectrum





Define Sourcing Values and Input Methodology (example)

Organizational applications are classified in accordance with several sourcing criteria provided as part of the portfolio analysis and subsequently validated by application stakeholders. The criteria and narrative describing the assigned values are provided below. Note that in most cases, a composite of multiple values from the inventory is used to arrive at a single classification value.

Assessment Criterion	Input	Classification Justification (1-Low / 3-High)	Weight
Level of Differentiation to the Organization	- Impact on achieving business goal	1 – Non differentiator 2 – Low differentiator 3 – Core differentiator	60%
Complexity of Operation	- Number and complexity of business rules/processes	1 – Low complexity of operation 2 – Moderate complexity of operation 3 – High complexity of operation	20%
Skills and competencies	- Specialized skills required to support and operate	1 – Highly specialized skills that are a challenge to hire and retain 2 – Moderate specialized IT/product skills 3 – Common IT/Admin/Programming skills	20%

Source: Migration Planning – June 2016



SUMMARY & REFERENCES



Summary

- During application assessment, we assess the portfolio of enterprise application from application and data dependencies to identify potential application to be migrated to the cloud
- Assessment includes type of data handled, coupling with other type of applications, compliance requirement and audience
- Another framework is to look at possible sourcing source (outsource - inhouse) and hosting options for the applications (on-premise – cloud).
 - We identify the classification factors and use the weighted average to come up with a likely decision.
- Once we have an direction, propose alternative approaches for an application and compare the pros and cons of various approaches before implementing
 - Proof-Of-Concept development can be applied to reduce risks.



References

- How the Cloud Changes Disaster Recovery - July 2011
 - <http://www.datacenterknowledge.com/archives/2011/07/26/how-the-cloud-changes-disaster-recovery/>
- Migrating your Existing Applications to AWS Cloud – October 2010
 - <https://d0.awsstatic.com/whitepapers/cloud-migration-main.pdf>
- An Overview of the AWS Cloud Adoption Framework version 2 – February 2017
 - https://d0.awsstatic.com/whitepapers/aws_cloud_adoption_framework.pdf
- AWS Cloud Adoption Framework
 - <https://aws.amazon.com/professional-services/CAF/>
- Microsoft Azure – Moving Your Applications to Microsoft Azure – March 2013
 - <https://msdn.microsoft.com/en-us/magazine/jj991979.aspx>
- Moving Applications to the Cloud, 3rd Edition
 - <https://msdn.microsoft.com/library/ff728592>
- Migration Planning – June 2016
 - <https://www.slideshare.net/AmazonWebServices/migration-planning>
- UA-IT Transformation Blueprint – May 2016 (good example)
 - <https://www.alaska.edu/oit/cloud/news-and-events/AWS-IT-Transformation-Kick-Off-Presentation-v1.0-UA.final.pptx>



Cloud Native Solution Design

CLOUD PERSISTENCE

HISTORY - NOSQL – PERSISTENCE DESIGN

Dr Venkat Ramanathan

rvenkat@nus.edu.sg

Institute of Systems Science

National University of Singapore

Total slides: 67

© 2009-23 NUS. The contents contained in this document may not be reproduced in any form or by any means, without the written permission of ISS, NUS, other than for the purpose for which it has been supplied.



Learning Objectives

- On completion of this module, the participant will be able to
 - Understand the various **persistence options** for cloud native application which can be applied to general application as well
 - Evaluates the various persistence options and recommend the appropriate choice
 - Design persistence strategy based on the requirement



Agenda

- History and Concepts
- Cloud Persistence Options
- AWS Example of cloud persistence options
- Designing a cloud persistence strategy
- Summary

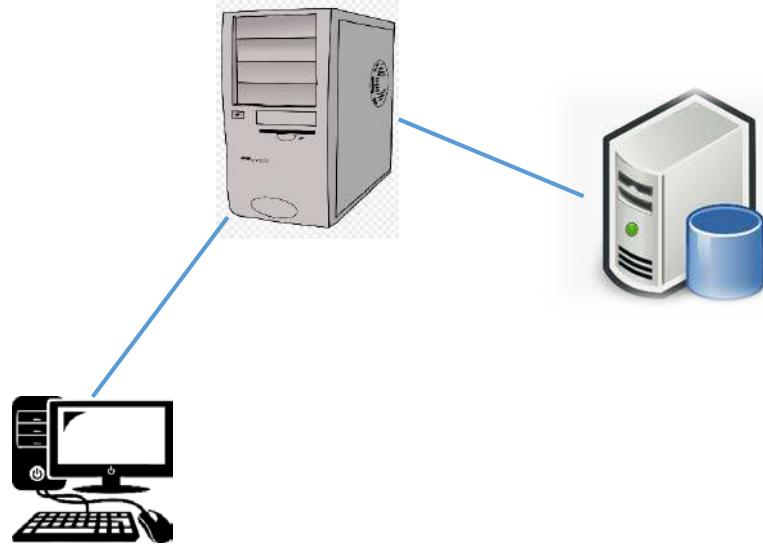


PERSISTENCE OVERVIEW FOR A CLOUD BASED APPLICATION

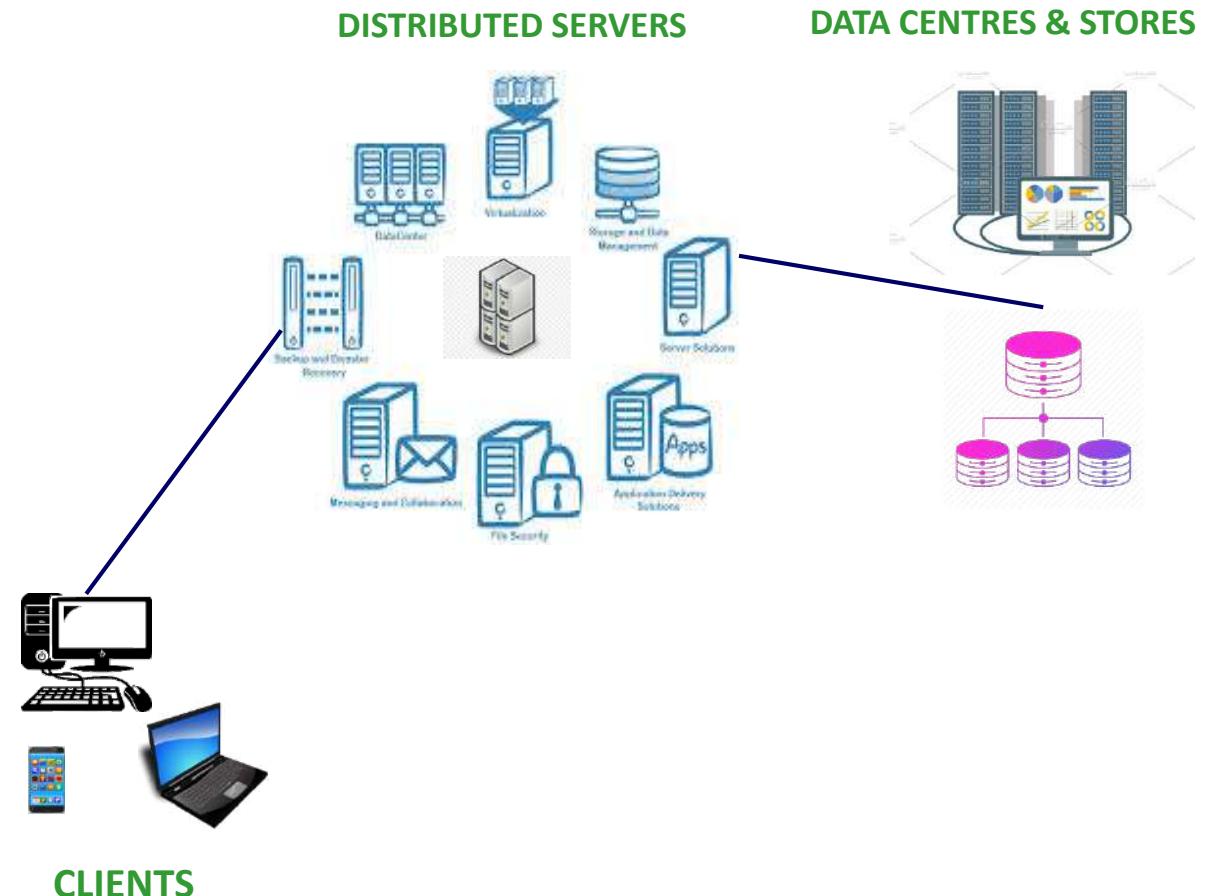


Applications in Context

- Traditional Applications



- Cloud Deployed Applications





The fundamental problem

- We need something to help us store and process data so that we can get some useful information from the data
- Current technology provides some tools to help us deal with that to some extent
- Technology will evolve according to the nature of data that we need to handle



HISTORY AND CONCEPTS

STRUCTURED VS. UNSTRUCTURED

RDBMS AND NOSQL



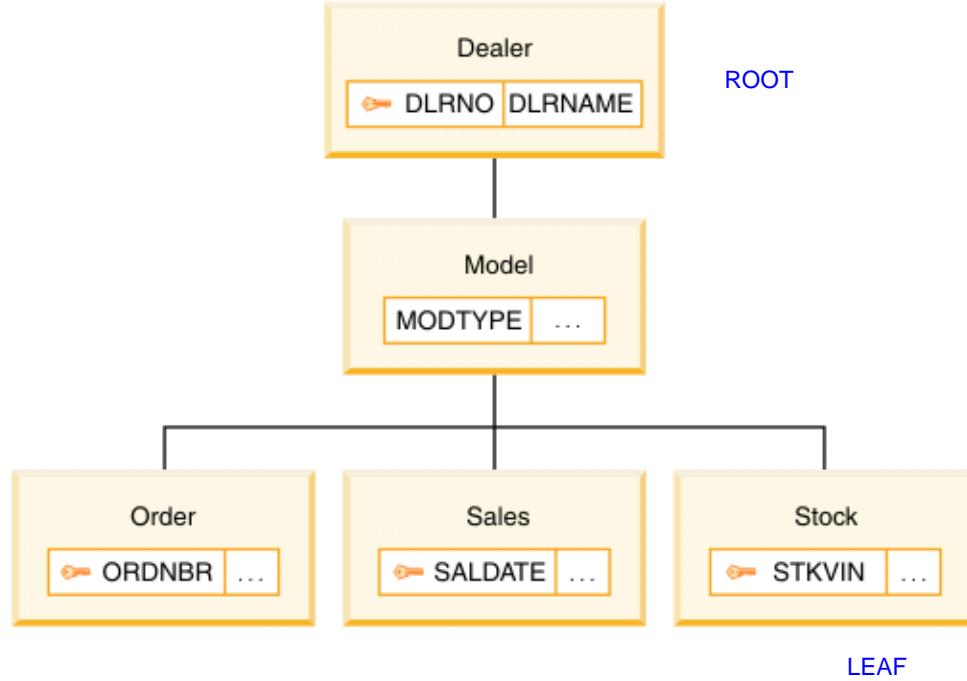
Navigational DBMS (1960s)

- Sometimes called as pre-relational database
- Hierarchical Model
 - data is organized into a **tree-like structure**
 - mandates that each child record has only one parent, whereas each parent record can have one or more child records. In order to retrieve data from a hierarchical database the whole tree needs to be traversed starting from the root node.
 - E.g.: IBM IMS (Information Management System)
- Network Model
 - **schema, viewed as a graph** in which object types are nodes and relationship types are arcs, is not restricted to being a hierarchy
 - allows each record to have multiple parent and child records
 - E.g.: IDMS (Integration Database Management System)



Hierarchical Model

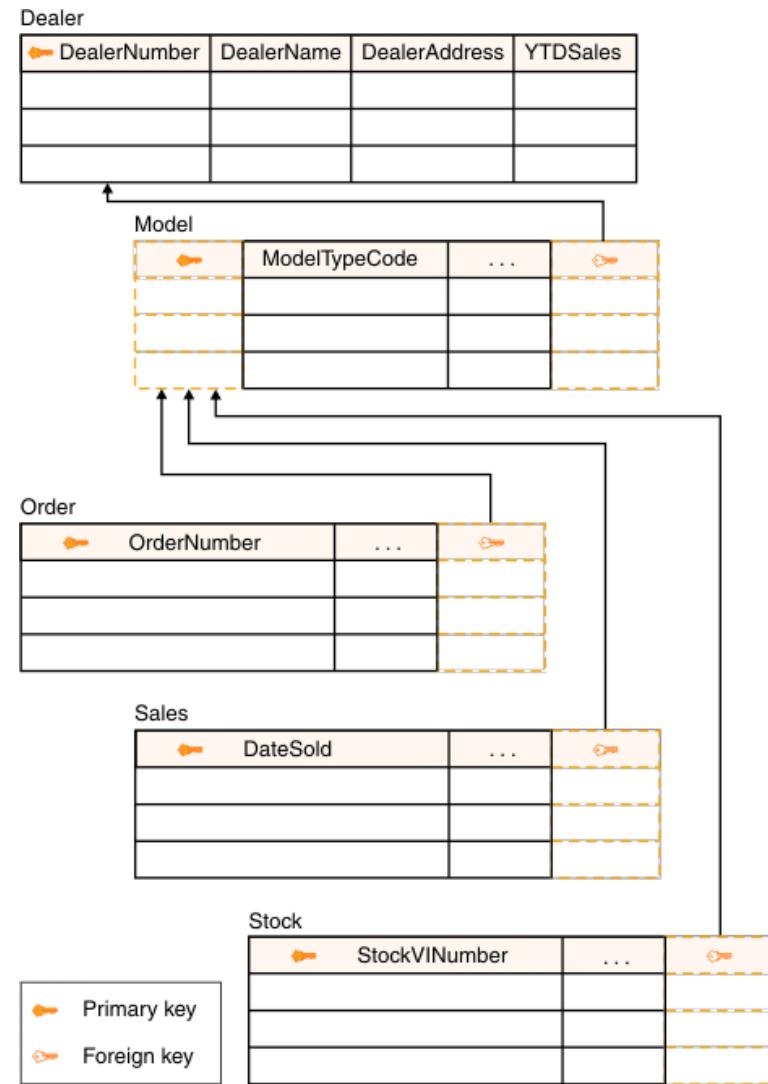
Hierarchical Model



Source:

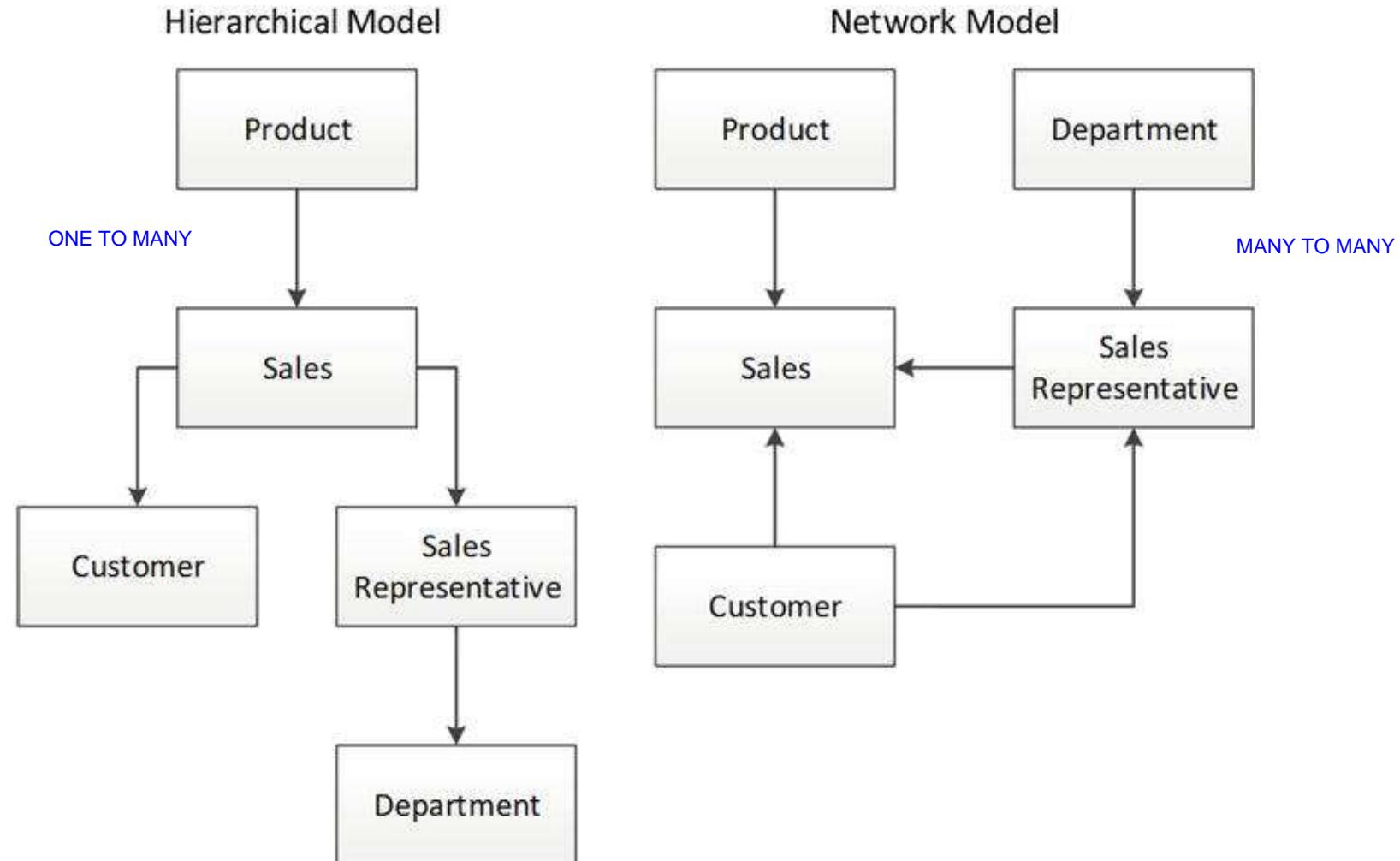
http://www.ibm.com/support/knowledgecenter/SSEPH2_13.1.0/com.ibm.ims13.doc.apg/ims_comparehierandrel dbs.htm

RDBMS Equivalent





Network Model



Source: Next Generation Databases, NoSQL, NewSQL and Big Data



Relational DBMS (1970s)

- Based on a paper from Edgar Codd: “A Relational Model of Data for Large Shared Data Banks”
- Essentially describes how a given set of data should be presented to the user, rather than how it should be stored on disk or in memory
- Levels of conformance to the relational model are described in the various “normal forms.”
- Jim Gray: “A transaction is a transformation of state which has the properties of atomicity (all or nothing), durability (effects survive failures) and consistency (a correct transformation).”
- ACID transactions is strongly associated with relational databases



Normalisation & RDBMS

why normalisation

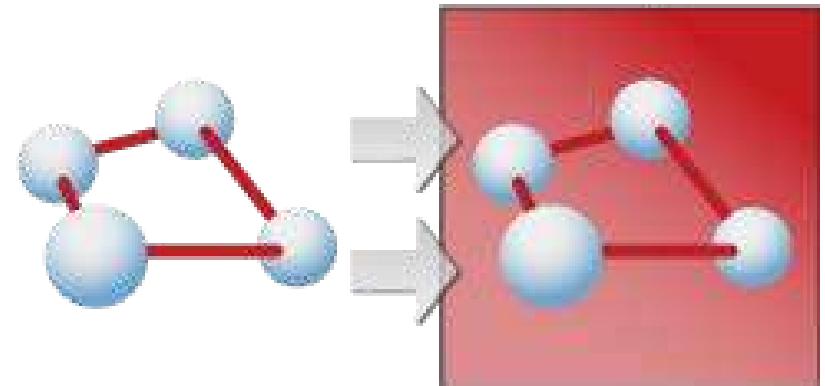
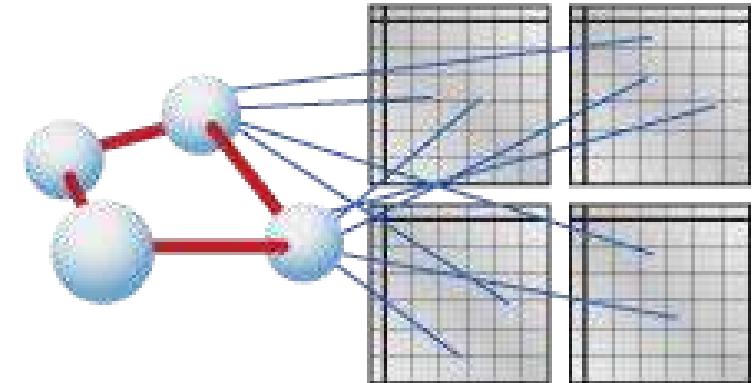
- RDBMS design strongly focuses on Normalisation
 - **Normalization** is a technique used to help reduce data duplication when designing data structures, also resulting in an improvement in data integrity.
 - This is achieved by placing data in multiple smaller tables which can be jointly queried when required.
- What is the rationale for normalization?
 - To free the collection of relations from undesirable insertion, update and deletion dependencies;
 - To **reduce the need for restructuring** the collection of relations, as new types of data are introduced, and thus increase the life span of application programs;
 - To make the relational model **more informative** to users;
 - To make the collection of relations neutral to the query statistics, where these statistics are liable to change as time goes by.

E.F. Codd, "Further Normalization of the Data Base Relational Model"



OODBMS -1990s

- The **OODBMS Manifesto**
(Atkinson/Bancilhon/DeWitt/Dittrich/Maier/Zdonik, '90)
- "A relational database is like a garage that forces you to take your car apart and store the pieces in little drawers"
 - Also SQL is ugly when working on object based design
- "A Object database is like a closet which requires that you hang up your suit with tie, underwear, belt socks and shoes all attached" (Dave Ensor)



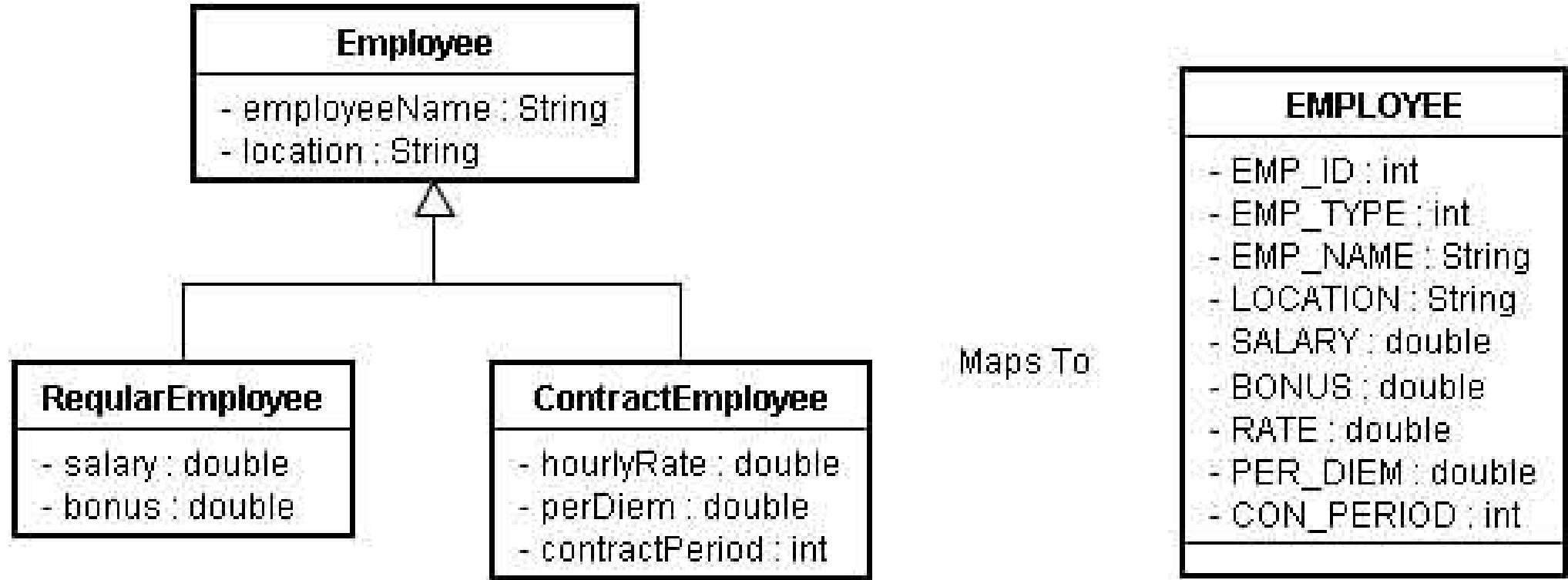


OODBMS (1990s)

- Due to success of Object Oriented Programming
 - Storing object into RDBMS is not simple
- Store objects without normalization
- Support complex objects, object identity, encapsulation, types or classes, inheritance, overriding combined with late binding, extensibility and computational completeness.
<https://www.cs.cmu.edu/~clamen/OODBMS/Manifesto/htManifesto/Manifesto.html>
- Failed to get market share
- Object-Relational Mapping (ORM) helps to solve part of the problems OODBMS tried to solve



Object vs RDBMS Table

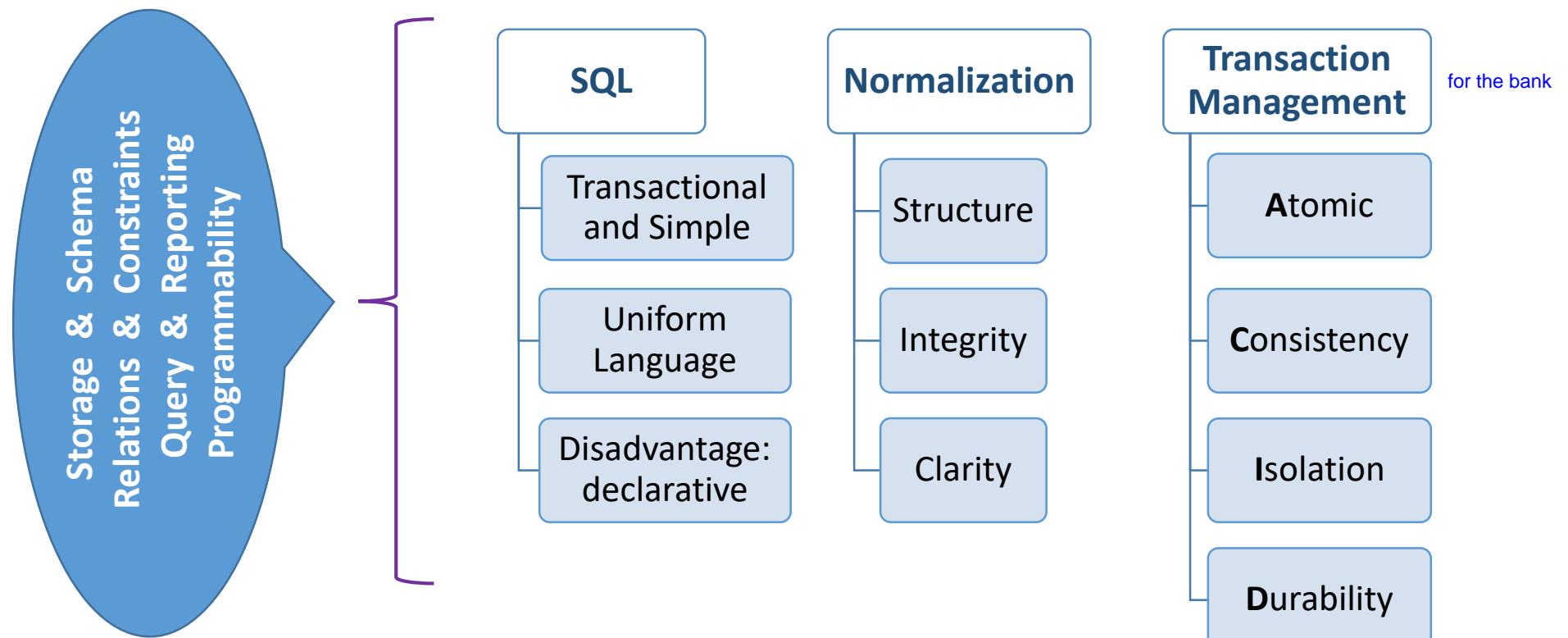


Source: <https://simsonlive.wordpress.com/2008/03/09/how-inheritance-works-in-hibernate/>

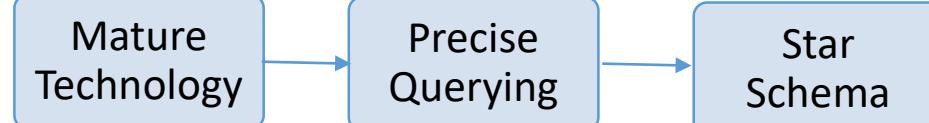


In Summary, RDBMS Features ...

why RDBMS?



And In Effect Offers...



But is...

Expensive when scaling



Atomicity: Transactions are treated as atomic units, meaning they are either completed in their entirety or not at all. In the context of a bank, this ensures that when a customer transfers money from one account to another, either the entire transfer is successful, or it fails completely, avoiding partial updates that could lead to inconsistencies. Consistency: Transactions must leave the database in a consistent state. In a banking system, this ensures that debit and credit operations maintain the balance and account relationships as expected. Isolation: Transactions are isolated from each other to prevent interference. This ensures that concurrent transactions do not impact each other's results. For example, one customer's balance inquiry should not be affected by another customer's deposit. Durability: Once a transaction is committed, its changes are permanent and survive system crashes. This ensures that customers' account information and transaction history are safe and recoverable.

DATA TRENDS

THE EVOLUTION OF BIG DATA IN ENTERPRISES



What is happening to DATA of late?

- Effectively BIG DATA now.
- The SIX V's:
 - **Volume** refers to the vast amount of data generated every second.
 - **Velocity** refers to the speed at which new data is generated and the speed at which data moves around.
 - **Variety** refers to the different types of data we can now use.
 - **Variability** refers to the change in data meaning/models over time.
 - **Veracity** refers to the uncertainty or trustworthiness of the data.
 - **Value** refers to our ability turn our data into value.
- IMPACT: Application changes dramatically due to these megatrends.
 - The growing number of internet users that applications must support (along with elevated user expectations for how applications should perform).

The Need: Better Performance, Scalability, & Flexibility



Challenges to current Persistence Models

- *In Other words the concerning factors were mainly the limitations of RDBMS ...*
- *Variety: The need for Non-static Schema,*
 - Developers are working with applications that create massive volumes of new, rapidly changing data types — structured, semi-structured, unstructured and polymorphic data.
- *Variability: The need for Agility and Heterogeneity*
 - Applications that once served a finite audience are now delivered as services that must be always-on, accessible from many different devices and scaled globally to millions of users. Further development is now carried out in agile sprints, iterating quickly and pushing code every week or two, some even multiple times every day.
- *Volume of Data: The need for Scalability*
 - Organizations are now turning to scale-out architectures using open source software, commodity servers & cloud computing instead of large monolithic DB servers.
- *Velocity: The need for higher performance*
 - With real-time processing needs, the need for performance enhancing needs strong distributed server persistence.

Can the RDBMS (aka loosely called SQL DB) handle the above?
No (say the gurus and industry).



These (guys) who faced the problem

- Major Contributors

- Trigger many innovations
 - Google: Google File System (2003) → MapReduce (2004) → BigTable (2006)
 - Yahoo: Google Map Reduce → Hadoop (2007)
 - Amazon: DynamoDB (2007) → Project Voldemort (Linked in)
 - Facebook: Sharding with MySQL → Cassandra (2008)
 - Any many many others



Google's Requirement

Question	Response
What was their focus?	Search Engine (<i>i.e., in 2003 when the requirement came</i>)
What they needed from Database?	<p>Input: Key word Output: List of web pages</p>
What was their challenge?	<p>My God, the web is exploding, pages are increasing, users are demanding, etc... I need to store lots of records. I don't have too many diverse entities and not many fields in entities. I need to retrieve data fast and display to users. But, hey, my users do not edit data.</p>
Why not live with RDBMS?	<p>I don't need transactions, I don't need multiple entities, I don't need integrity constraints, etc. etc. So do you think I am getting a product that offers too much? Ok but I have to put data in multiple servers due to volume and response time needs – RDMS pains...</p>



Google's Decision

- Discard RDBMS
- Create my own
 - Sigh,
 - who in business is talking of not wasting resources re-inventing wheels now?
 - who in IT is not advocating reusability mantra of software engineering etc?
- Ok what model do I use:
 - Key -> Value (stated already)
 - Key – is a list of key words
 - Value – is a list of web page urls
- Ok lets get some Computer Science into it (not in detail though)
 - We may need in this: Only one Index, Sorted Lexicographically on row key, etc...



Expectations of Database

- Some current day business demand these characteristics from IT Solutions & Systems
 - **Volume of data**
 - Accommodate large volume of data (which may require storing across multiple servers)
 - **Low latency – fast response time**
 - Always a need, all the more in global situation and 24x7 operations and rich customer needs.
 - **Availability**
 - Due to Global operations and real time operations very high availability is required.
 - Incremental **scalability with no downtime**
 - With rapid growth in data and adding storages may need to be done on needs basis.
 - Tolerant to network partition
 - Consequence of distributed storage
 - Run on commodity hardware
 - Low cost and avoid proprietary lockin

Cloud Technology is the Answer



Birth of NoSQL

- Alternates are needed:
 - Guys who got stuck viz., Google, Amazon started doing their own Databases...
- Carlo Strozzi (1998)
 - Named NoSQL for small isolated non-relational databases
- Johan Oskarsson (2009)
 - Used this term as # tag for a conference to discuss the mushrooming databases from Google, Amazon and other players.
- We live with this
 - But strong view exists it is typically not Non-SQL, but Not Only SQL
 - Essentially a family of various types of databases.



How does NoSQL address the motivations?

- **NoSQL accommodates schema-less data storage**, which is one of the main advantages.
 - It will allow the storing of all types of data in different formats and in different schemas, thereby providing more robust and agile development.
- **NoSQL driven platform accommodates servers to scale horizontally**, which means it is very easy to scale the capacity up or down.
 - We can simply add new servers or remove servers to increase or decrease its capacity, storage, and computation power.
- **NoSQL structure facilitates working in a clustered environment** which is much less expensive than a highly reliable server.
 - Server clusters are mainly built on commodity hardware, without affecting the performance or reliability.
- **NoSQL databases spread across multiple nodes** which eases replication to multiple servers.
 - Some of NoSQL database frameworks even work without the master slave concept, which makes them highly available with no single point of failure.



The Era of Polyglot Persistence

- Polyglot persistence is about using different data storage technologies to handle varying data storage needs
- Polyglot persistence can apply across an enterprise or within a single application
- Encapsulating data access into services reduces the impact of data storage choices on other parts of a system
- Adding more data storage technologies increases complexity in programming and operations, so the advantages of a good data storage fit need to be weighed against this complexity

Polyglot Persistence is a fancy term to mean that when storing data, it is best to use multiple data storage technologies, chosen based upon the way data is being used by individual applications or components of a single application.



Structured vs Unstructured Data vs Semi-structured

▪ Structured Data

- Information with a high degree of organization.
- The structure facilitates data processing.
- Typically has a data model.
- Easily entered, stored, queried, analyse.
- The storage understand the structure.

List of students
Employee details

▪ Unstructured Data

- Data that does not have a schema.
- Cannot be queried or processed in typical sense
- For inference may require NLP, Image Processing, ML etc

PDF files
Audio
Video

▪ Semi Structured

- Has a schema but is not rigid.
- Easy to adapt to changes
- Easy to split the dataset and distribute

Employee data stored in XML
JSON



NoSQL Data Store Patterns

Pattern name	Description	Typical uses
Key-value store	A simple way to associate a large data file with a simple text string	Dictionary, image store, document/file store, query cache, lookup tables
Column family store	A way to store sparse matrix data using a row and a column as the key	Web crawling, large sparsely populated tables, highly-adaptable systems, systems that have high variance
Document store	A way to store tree-structured hierarchical information in a single unit	Any data that has a natural container structure including office documents, sales orders, invoices, product descriptions, forms, and web pages; popular in publishing, document exchange, and document search
Graph store	A way to store nodes and arcs of a graph	Social network queries, friend-of-friends queries, inference, rules system, and pattern matching



Key Value Store

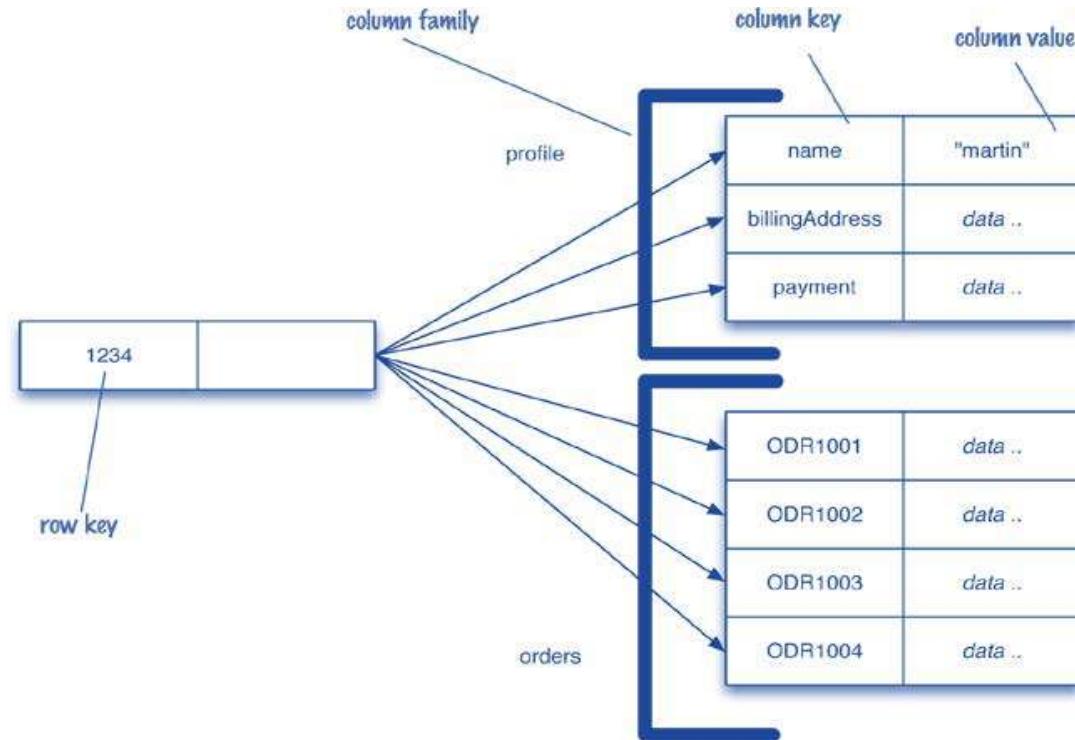
A key-value store, or key-value database, is a data storage paradigm designed for storing, retrieving, and managing associative arrays, a data structure more commonly known today as a dictionary or hash. Dictionaries contain a collection of objects, or records, which in turn have many different fields within them, each containing data.

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623



Column Family Data Store

Column-family databases organize their columns into column families. Each column has to be part of a single column family, and the column acts as unit for access, with the assumption that data for a particular column family will be usually accessed together.



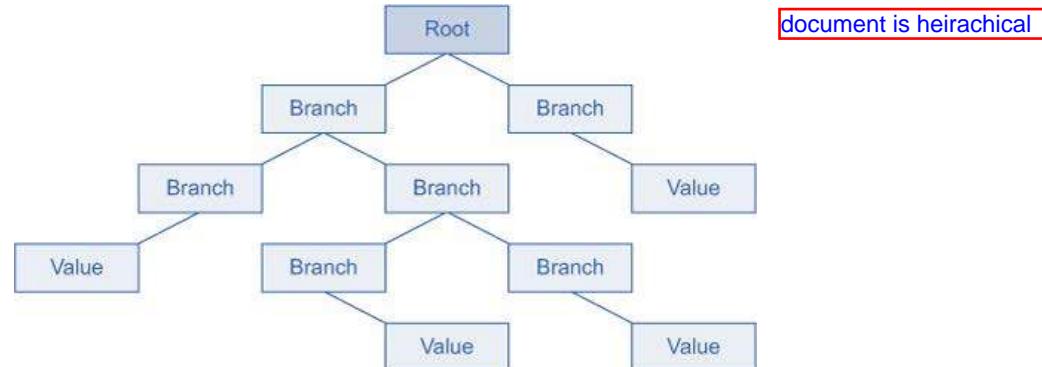
Row-oriented: Each row is an aggregate (for example, customer with the ID of 1234) with column families representing useful chunks of data (profile, order history) within that aggregate.
Column-oriented: Each column family defines a record type (e.g., customer profiles) with rows for each of the records. You then think of a row as the join of records in all column families.



Document Store

In a document data store everything related to a database object is encapsulated together. Thus:

- **Documents are independent units** with better performance and easy distribution of data across multiple servers.
- **Application logic is easier to write.** We can turn the object model directly into a document.
- **Unstructured data can be stored easily** and additionally the information schema structure is known in advance.



```

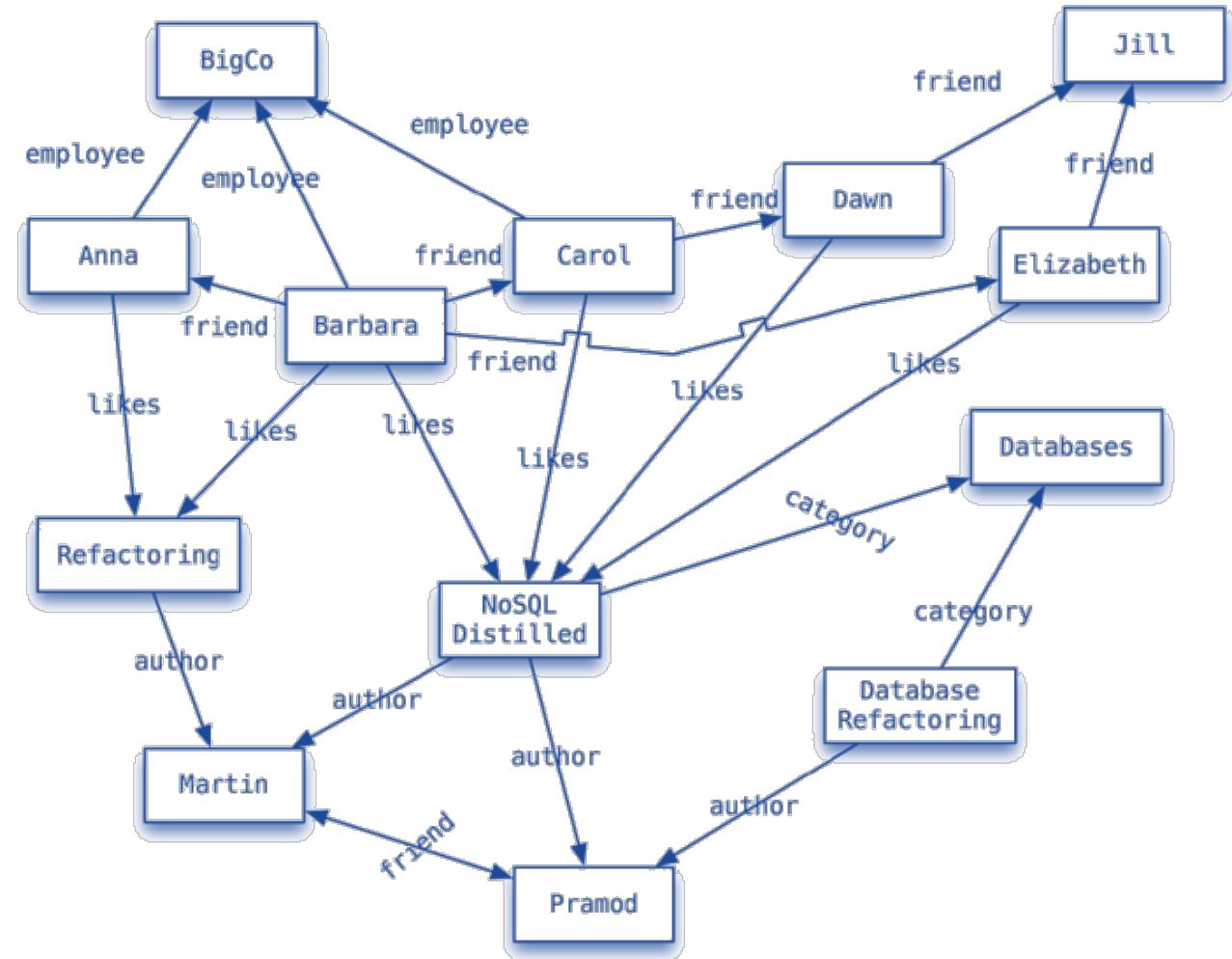
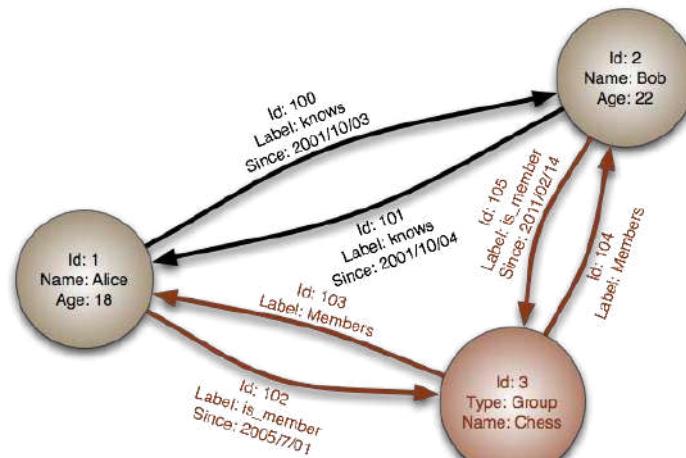
// in customers
{
  "id":1,
  "name":"Martin",
  "billingAddress":[{"city":"Chicago"}]
}

// in orders
{
  "id":99,
  "customerId":1,
  "orderItems":[
    {
      "productId":27,
      "price": 32.45,
      "productName": "NoSQL Distilled"
    }
  ],
  "shippingAddress":[{"city":"Chicago"}]
  "orderPayment": [
    {
      "ccinfo":"1000-1000-1000-1000",
      "txnId":"abelif879rf",
      "billingAddress": {"city": "Chicago"}
    }
  ],
}
  
```



Graph Data Store

Graph data store uses graph structures for semantic queries with nodes, edges and properties to represent and store data.





ACTIVITY



ACTIVITY: Modelling a Data Store (database)

- Consider the following shopping cart application on web that sort of yields (requires the following – derive your own data elements as exercise and arrive at database schema). Refer also to details given in class/handout. Activity: **Group Task 30 min plus presentation.**

The screenshot shows a shopping cart page from iHerb.com. The cart contains two items:

Product(s)	Price	Qty.	Total
Nature's Way, Umcka, Cold + Flu, Berry, 20 Chewable Tablets	SG\$19.22	1	SG\$19.22
Life Extension, Buffered Vitamin C Powder, 16 oz (454 g)	SG\$29.13	1	SG\$29.13

Savings: Enter Promo / Rewards Code and Apply.

Order Summary:

- Subtotal: SG\$48.35
- Shipping: SG\$5.49 (via Singapore Express) - Duties & Taxes may be collected at delivery.
- Order Total:** SG\$53.84

Accepted Payment Methods: PayPal, VISA, MasterCard, American Express, SGS.

Buttons: Proceed to Checkout, Move to Lists, Remove.



When to choose KV? Eg: Redis

- User-defined indexing schemes.
- Indexing of unstructured data files (*including other stores such as Document or Graph*)
- In memory cache of small data sets
- Message queues with real-time new element notification.
- Real-time publish/subscribe notification systems.
- Task queues and job systems.
- High score leaderboards.
- User ranking systems.
- Hierarchical/tree structured storage systems.
- Individual personalized news or data feeds for your users.



Key Value Types and Notable Examples

KV - eventually consistent models

- Apache Cassandra [* popular]
- Dynamo
- Oracle NoSQL Database
- Project Voldemort
- Riak [* popular]
- OpenLink Virtuoso

KV – random consistency models

- Aerospike
- Coherence
- FairCom c-treeACE
- Hazelcast
- memcached
- OpenLink Virtuoso
- Redis
- XAP
- Gemfire

KV – serially ordered consistency models

- Berkeley DB
- FairCom c-treeACE/c-treeRTG
- FoundationDB
- HyperDex
- IBM Informix C-ISAM
- InfinityDB
- LMDB
- MemcacheDB
- NDBM



When to choose Column Family Eg: Cassandra, DynamoDB

- There exists relatively clear structure and static nature of the schema
- When **data is large**
 - Need to distribute across servers (scalability)
 - Need to perform compression
- When the data attributes have a small group of collective data that are more often queried.
- When there is significant aggregation type queries such as in Analytics



When to use document store? Eg: Mongo

- **Data insert consistency:** If there is a requirement of writing huge amount of data without losing some data, then Document Store is the best suited because of its high data insertion rate.
- **Data corruption recovery:** In Document Store, data repair can be done on a database level and there is an automatic command to do so. However, the command reads all the data and rewrites it to a new set of files which may be time consuming if the database is huge but it is preferred over losing the entire dataset.
- **Load balancing:** Document Store supports faster replication and automatic load balancing configuration because of data placed in shards.
- **Avoid joins:** If the developer wants to avoid normalization and joins, Document Store is the best suited.
- **Changing schema:** Document Store is schema-less, hence adding new fields will not result in any issues.
- **Not relational data:** If the data to be stored need not to be a relational one, then Document Store should be selected.
- **Mapping:** If the mapping of application data objects is to be done directly into the document-based storage, Document Store is the best suited.
- **Creating database cluster:** If the database is geographically distributed and the user wishes to create cluster and speed up data queries among remotely located databases, Document Store is suitable.



Notable Document Examples

Name	Notes
Cloudant	Distributed database service based on BigCouch , the company's open source fork of the Apache -backed CouchDB project. Uses JSON model.
Couchbase Server	Distributed NoSQL Document Database, JSON model and SQL based Query Language.
DocumentDB	Platform-as-a-Service offering, part of the Microsoft Azure platform
Informix	RDBMS with JSON, replication, sharding and ACID compliance
Jackrabbit	Java Content Repository implementation
Lotus Notes (IBM Lotus Domino)	MultiValue
Document Store	Document database with replication and sharding, BSON store (binary format JSON)
PostgreSQL	HStore, JSON store
RavenDB	2nd generation document database, JSON format with replication and sharding.
Solr	Search engine
TokuMX	Document Store with Fractal Tree indexing
OpenLink Virtuoso	Middleware and database engine hybrid



When to choose Graph? Eg: Neo4j

- When entities are to be linked in a networking fashion
 - Social networking such as friends in LinkedIn.
 - Recommender systems such as recommending similar products in ecommerce site.
- Data Relationships
 - Data that exhibits Many-to-Many relationship; *viz., Low Latency at Large Scale*
 - Data that has unstructured relationships *viz., Visualisation such as time series and demographics with AI algorithms*
 - Data relationships are prone to change or evolve *viz., intricately structured high value relationships*
- Graphs are useful when navigations between entities require to be fast to improve user experience.
- The size of data is limited in comparison to other data stores.
 - Most graphs are derived from larger data sets in other forms of data stores.



SQL vs NoSQL vs NewSQL

NewSQL is a term coined by 451 Group analyst Matt Aslett to describe a new group of databases that share much of the functionality of traditional SQL relational databases, while offering some of the benefits of NoSQL technologies.

- The term NoSQL was used by Carlo Strozzi in 1998 to name his lightweight, Strozzi NoSQL open-source relational database that did not expose the standard SQL interface, but was still relational.
- Johan Oskarsson reintroduced the term NoSQL in early 2009 when he organized an event to discuss "open source distributed, non relational databases".

Source: wiki

Characteristic	SQL	NoSQL	NewSQL
Relational	Yes	No	Yes
SQL language support	Yes	No*	Yes
Fully ACID Compliant	Yes	No	Yes
Horizontal Scalability (Scale-out vs. Scale-up)	No	Yes	Yes
High Performance	No	Yes	Yes
Schema-on-Write	Yes	No	Yes



SCALING AND CONSISTENCY

SHOW ME YOUR FLOWCHARTS AND CONCEAL YOUR TABLES, AND I SHALL
CONTINUE TO BE MYSTIFIED. SHOW ME YOUR TABLES, AND I WON'T USUALLY
NEED YOUR FLOWCHARTS; THEY'LL BE OBVIOUS.

FRED BROOKS

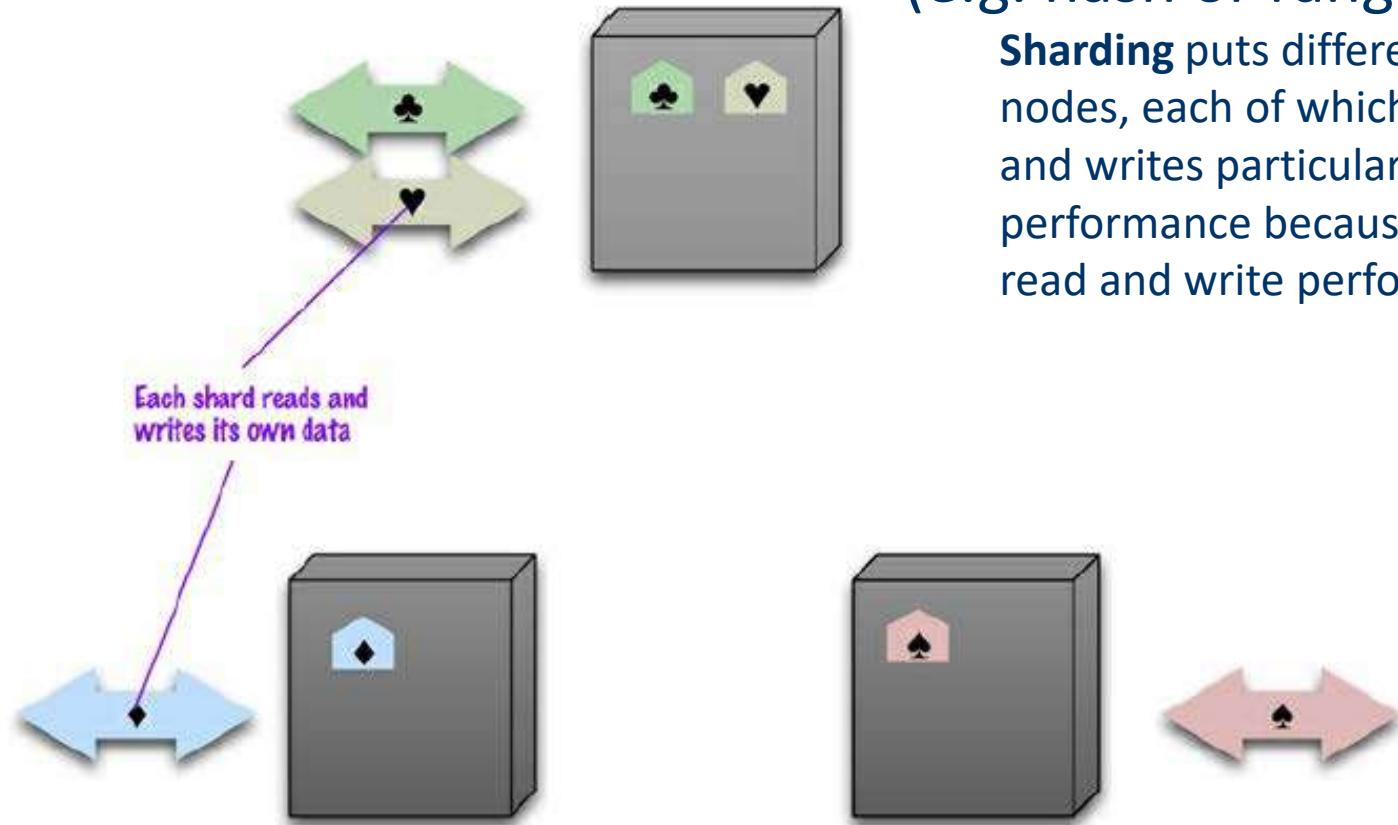


Sharding & Replication

- There are two styles of distributing data:
 - **Sharding** distributes different data across multiple servers, so each server acts as the single source for a subset of data
 - **Replication** copies data across multiple servers, so each bit of data can be found in multiple places.
 - A system may use either or both techniques.
- Replication comes in two forms:
 - **Master-slave** replication makes one node the authoritative copy that handles writes while slaves synchronize with the master and may handle reads
 - **Peer-to-peer** replication allows writes to any node; the nodes coordinate to synchronize their copies of the data.
 - Master-slave replication reduces the chance of update conflicts but peer to- peer replication avoids loading all writes onto a single point of failure



Sharding Illustrated



Horizontal partitioning of data
(e.g. hash or range partitioning)

Sharding puts different data on separate nodes, each of which does its own reads and writes particularly valuable for performance because it can improve both read and write performance

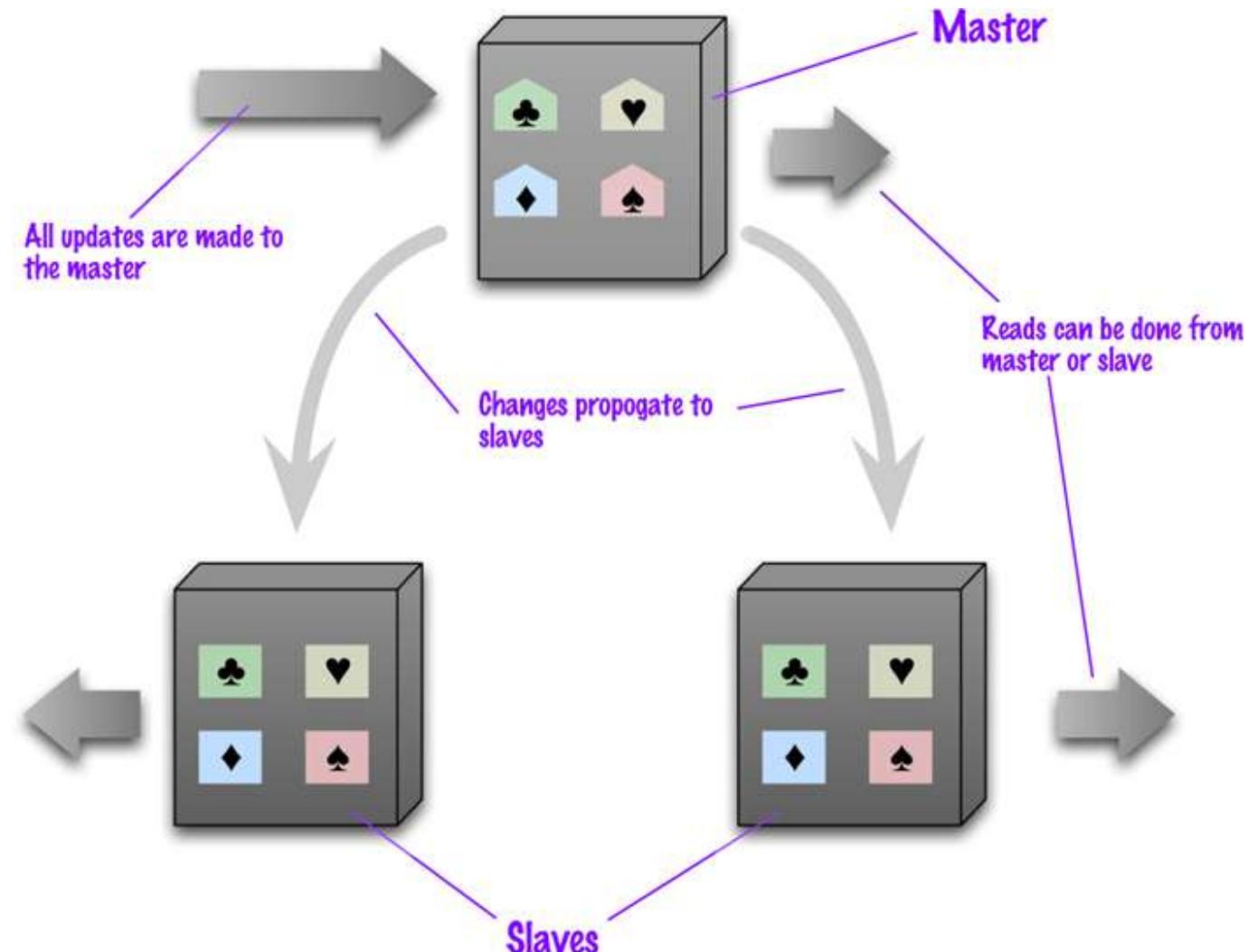


What is the advantage of “Data Sharding”?

- Many NoSQL databases offer **auto-sharding**, where the database takes on the responsibility of allocating data to shards and ensuring that data access goes to the right shard. Consequences:
 - manage parallel access in the application
 - scales well for both reads and writes
 - not transparent, application needs to be partition-aware
- Sharding is valuable for performance since it can improve both read and write performance.
- Using replication, particularly with caching, can greatly improve read performance but does little for applications that have a lot of writes.

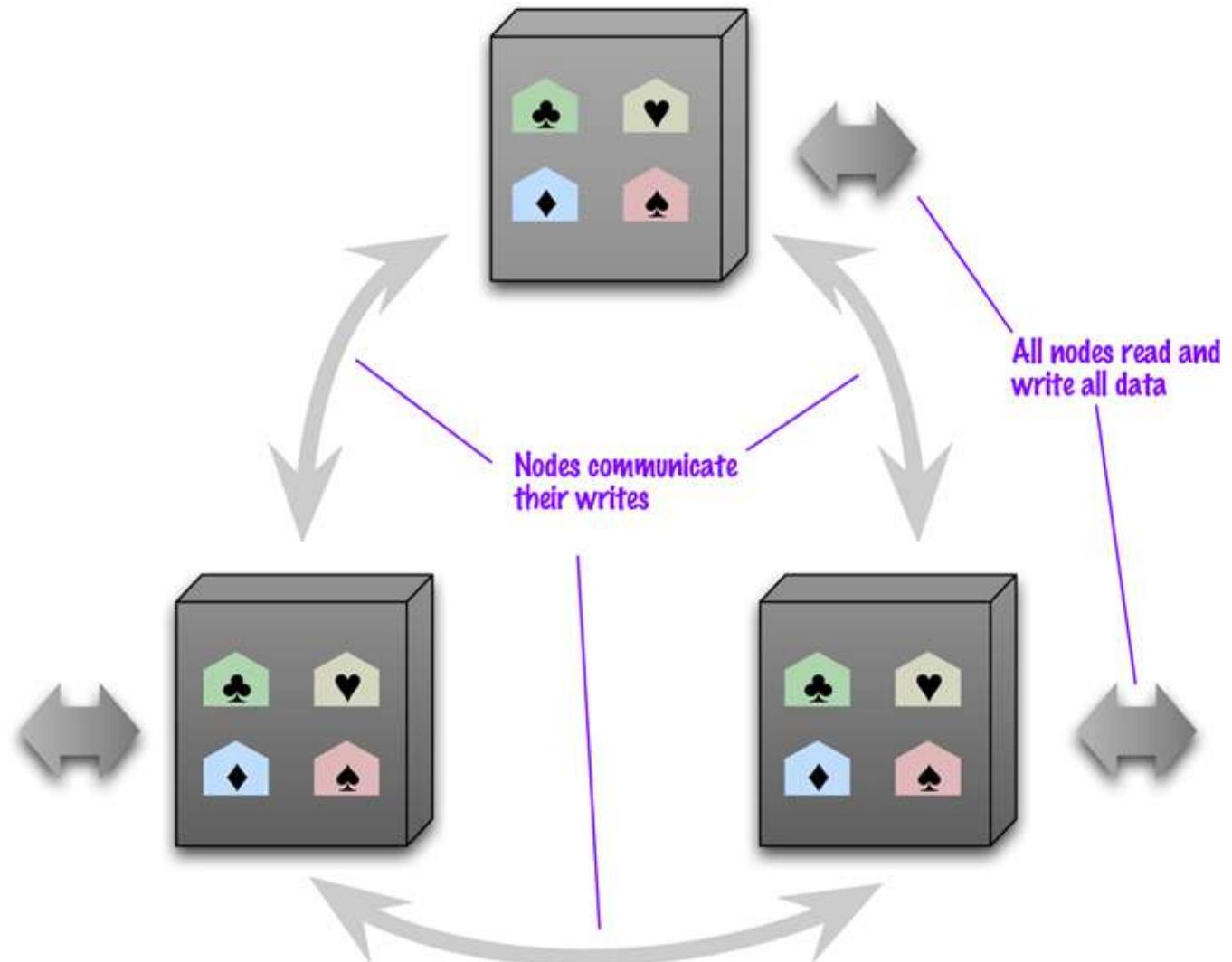


Master Slave Replication





Peer to Peer Replication





Types of Consistency

- Consistency with other users: If two users query the database at the same time, will they see the same data? Traditional relational systems would generally try to ensure that they do, while non-relational databases often take a more relaxed stance.
- Consistency within a single session: Does the data maintain some logical consistency within the context of a single database session? For instance, if we modify a row and then read it again, do we see our own update?
- Consistency within a single request: Does an individual request return data that is internally coherent? For instance, when we read all the rows in a relational table, we are generally guaranteed to see the state of the table as it was at a moment in time. Modifications to the table that occurred after we began our query are not included.
- Consistency with reality: Does the data correspond with the reality that the database is trying to reflect? For example, it's not enough for a banking transaction to simply be consistent at the end of the transaction; it also has to correctly represent the actual account balances. Consistency at the expense of accuracy is not usually acceptable.



Relaxing ACID properties

- **ACID** is hard to achieve in distributed databases
 - not always required
 - E.g. blogs, status updates, product listings, etc.
- **Availability**
 - Traditionally, thought of as the server/process available 99.999 % of time
 - For a large-scale node system, there is a high probability that a node is either down or that there is a network partitioning
- **Partition tolerance**
 - ensures that write and read operations are redirected to available replicas when segments of the network become disconnected



The Idea of Eventual Consistency

- RDBMS was engineered to strongly support ACID consistency
 - This was required for transactional processes
 - It also caters to situations where data is accessed concurrently by multiple users.
 - Difficult to achieve if data is replicated and distributed.
- Eventually Consistency – copies becomes consistent at some later time if there are no more updates to that data item
 - When no updates occur for a long period of time, eventually all updates will propagate through the system and all the nodes will be consistent
 - For a given accepted update and a given node, eventually either the update reaches the node or the node is removed from service
- BASE (Basically Available, Soft State, Eventual consistency) properties, as opposed to ACID
 - Soft state: copies of a data item may be inconsistent
 - Basically Available – possibilities of faults but not a fault of the whole system
 - Data becomes eventually consistent



CLOUD PERSISTENCE OPTIONS



Cloud Persistence Options in the Cloud

- File
 - Local file system
 - Network file system
 - Mounted Volume
 - Object Storage

- Databases
 - RDBMS
 - Document
 - Column
 - Key Value / In-memory cache
 - Graph



File Storage – Local file system

- The local file system of your VM instance
- This storage is located on disks that are physically attached to the host computer.
- Data is not durable in the implementation of many cloud providers
- Applicable only for IaaS
- Data will be lost if
 - The underlying drive fails
 - Instance stops
 - Instance terminates
- Ideal for temporary storage of information that changes frequently, such as buffers, caches, scratch data, and other temporary content, or for data that is replicated across a fleet of instances, such as a load-balanced pool of web servers.
- Analogy: local harddisk of your notebook

Example: Amazon EC2 / FSx for Windows File Server



File Storage – Network File System

- Network file system that can be accessed by multiple instances
- Some cloud providers like AWS provides a managed network file system service
- Allow simultaneous read-write access from multiple virtual machine instances – just like a typical network file system
- Ideal for sharing files across multiple instances
- Applicable for IaaS and probably PaaS (that allows for file system access)
- Analogy: shared folder over network

Example: Amazon Elastic File System (EFS)



File Storage – Mounted Volume

- Block level storage volume that can be mounted to any running VM instances
 - like mounting a virtual hard disk.
- Persists independently from the life of the VM instances
- Can only be attached to one VM instance for read/write access, for obvious reason.
- Some cloud providers may allow the volume to be attached to multiple VM instances for read-only access
- Ideal when the data must be quickly accessible and requires long-term persistence
- Analogy: portable USB hard disk

Example: Amazon Elastic Block Store (EBS)



File Storage – Object Storage

- Storage for object (data/file + metadata) that is accessible through web service (normally REST) or API
- Consistent access method from various type of applications and cloud service model
 - One of the most flexible and ubiquitous storage option
- Ideal for data that need to be accessed from various different channels
- Analogy: Dropbox

Example: Amazon Simple Storage Service (Amazon S3)



STORAGE OPTIONS EXAMPLE

WITH AMAZON WEB SERVICES



Find AWS Storage Services!

Compute

- EC2
- Lightsail
- Elastic Container Service
- Lambda
- Batch
- Elastic Beanstalk

Storage

- S3
- EFS
- Glacier
- Storage Gateway

Database

- RDS
- DynamoDB
- ElastiCache
- Amazon Redshift

Migration

- AWS Migration Hub
- Application Discovery Service
- Database Migration Service
- Server Migration Service

Management Tools

- CloudWatch
- AWS Auto Scaling
- CloudFormation
- CloudTrail
- Config
- OpsWorks
- Service Catalog
- Systems Manager
- Trusted Advisor
- Managed Services

Media Services

- Elastic Transcoder
- Kinesis Video Streams
- MediaConvert
- MediaLive
- MediaPackage
- MediaStore
- MediaTailor

Machine Learning

- Amazon SageMaker
- Amazon Comprehend
- AWS DeepLens
- Amazon Lex

Analytics

- Athena
- EMR
- CloudSearch
- Elasticsearch Service
- Kinesis
- QuickSight
- Data Pipeline
- AWS Glue

Security, Identity & Compliance

- IAM
- Cognito
- Secrets Manager
- GuardDuty
- Inspector
- Amazon Macie
- AWS Single Sign-On
- Certificate Manager
- CloudHSM
- Directory Service
- WAF & Shield
- Artifact

Mobile Services

- Mobile Hub

Customer Engagement

- Amazon Connect
- Pinpoint
- Simple Email Service

Business Productivity

- Alexa for Business
- Amazon Chime
- WorkDocs
- WorkMail

Desktop & App Streaming

- WorkSpaces
- AppStream 2.0

Internet Of Things

- IoT Core
- IoT 1-Click
- IoT Device Management
- IoT Analytics
- Greengrass
- Amazon FreeRTOS

Game Development



Managed RDBMS Service



Amazon Relational Database Service (RDS)

Managed relational database service with a choice of six popular database engines.
Set up, operate, and scale a relational database in the cloud with just a few clicks.



Amazon Aurora

MySQL and PostgreSQL compatible relational database built for the cloud.
Performance and availability of commercial-grade databases at 1/10th the cost.

Amazon RDS Database Engines





Amazon own products



Amazon DynamoDB

Fast and flexible NoSQL database service for any scale.
Pay only for the throughput and storage you need.



Amazon Aurora

MySQL and PostgreSQL compatible relational database built for the cloud.
Performance and availability of commercial-grade databases at 1/10th the cost.



Amazon Redshift

Fast, simple, cost-effective data warehousing.

Amazon Neptune

Fast, reliable graph database built for the cloud





In-memory cache

Amazon ElastiCache

Managed, in-memory data store services. Choose Redis or Memcached to power real-time applications.

Amazon ElastiCache Engines



Amazon ElastiCache for Redis

Manage and analyze fast moving data with a versatile in-memory data store.



Amazon ElastiCache for Memcached

Build a scalable Caching Tier for data-intensive apps.



Storage related AWS Managed Services

Storage Option	AWS Services
Local File System	EC2
Network File System	EFS
Mounted Volume	EBS
Object Storage	S3
RDBMS	Aurora, RDS for PostgreSQL, RDS for MySQL, RDS for MariaDB, RDS for Oracle, RDS for SQL Server
Document	DynamoDB
Column	Redshift
Key-value	ElastiCache for Redis, ElastiCache for Memcached, DynamoDB (according to some documentation)
Graph	Neptune

* The table above exclude the option of installing your own database of choice on EC2 instances – which can be done if you want to use any other database that is not offered by AWS as a managed service



SUMMARY



Tangible benefits of NoSQL

- ***Continuously available***—This is available all the time even in the face of the most devastating infrastructure outages.
- ***Geographically distributed***—Database instances are distributed and available everywhere you need it.
- ***Operationally low latency***—The response time is very less and is suitable for intense operational applications.
- ***Linearly scalable***—Database servers could be added in order to tackle higher user as well as date nodes. The system performance grows linearly with the addition of new servers in the cluster.
- ***Lower total cost of ownership***—The leverage of commodity servers and open source software for NoSQL database management systems results in lower TCO and higher ROI.
- ***Multiple types***—There are specific NoSQL types for specific application requirements and data models.



Summary

- Various storage options in the cloud
 - Unstructured (File)
 - Local File System, Network File System, Volume, Object Storage
 - Structured (Database)
 - RDBMS, Document, Column, Key-Value, Graph
- Polyglot persistence means that you use the right tool for the right use case
 - More to learn but help to produce better system



Reference

AWS Storage Service Whitepaper

- <https://d1.awsstatic.com/whitepapers/Storage/AWS%20Storage%20Services%20Whitepaper-v9.pdf>

AWS Elastic File System

- <https://docs.aws.amazon.com/efs/latest/ug/whatisefs.html>

AWS Elastic Block Storage

- <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonEBS.html>

AWS S3 Documentation

- <https://aws.amazon.com/documentation/s3/>



Reference

AWS Elastic File System

- <https://docs.aws.amazon.com/efs/latest/ug/whatisefs.html>

AWS Elastic Block Storage

- <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonEBS.html>

AWS S3 Documentation

- <https://aws.amazon.com/documentation/s3/>

AWS Storage Service Whitepaper

- <https://d1.awsstatic.com/whitepapers/Storage/AWS%20Storage%20Services%20Whitepaper-v9.pdf>



Cloud Native Solution Design

DEVOPS IN THE CLOUD

Suria R Asai

suria@nus.edu.sg

Institute of Systems Science
National University of Singapore



Objectives

- Upon completion of this module, you will be able to:
 - Understand what is DevOps and why it has become the norm for software production
 - Define what is Cloud Computing, including its characteristics, service delivery models and deployment models
 - Understand the motivations for doing DevOps in the Cloud
 - Become familiar with DevOps tools provided by a leading Cloud platform, AWS



Topics

- Understand what is DevOps and why it has become the norm for software production
- Define what is Cloud Computing, including its characteristics, service delivery models and deployment models
- Become familiar with DevOps tools provided by a leading Cloud platform, AWS



Innovation is essential for success

- Today's environment: dynamic business and social landscape, increasing competition, constraints on budgets, etc.
- New technologies are accelerating disruption
- Transformation is vital for success
- Innovation creates new digital products and services
- Ability to develop and release software quickly and efficiently is key to disrupting and transforming





Definitions of DevOps

- DevOps ('Development' + 'Operations') is a software engineering practice for unifying software development and software operation
- Characteristics of DevOps:
 - Close alignment with business objectives through Agile processes
 - Automation and monitoring across integration, testing, deployment and management
 - Shorter development cycles
 - Increased deployment frequency
 - More dependable releases

Source: Wikipedia, <https://en.wikipedia.org/wiki/DevOps>

DevOps represents a change in IT culture, focusing on rapid IT service delivery through the adoption of agile, lean practices in the context of a system-oriented approach.

Emphasizes people (and culture), and improves collaboration between operations and development teams. Utilizes technology, especially automation tools that can leverage an increasingly programmable and dynamic infrastructure from a life cycle perspective.

Source: Gartner, <https://www.gartner.com/it-glossary/devops>



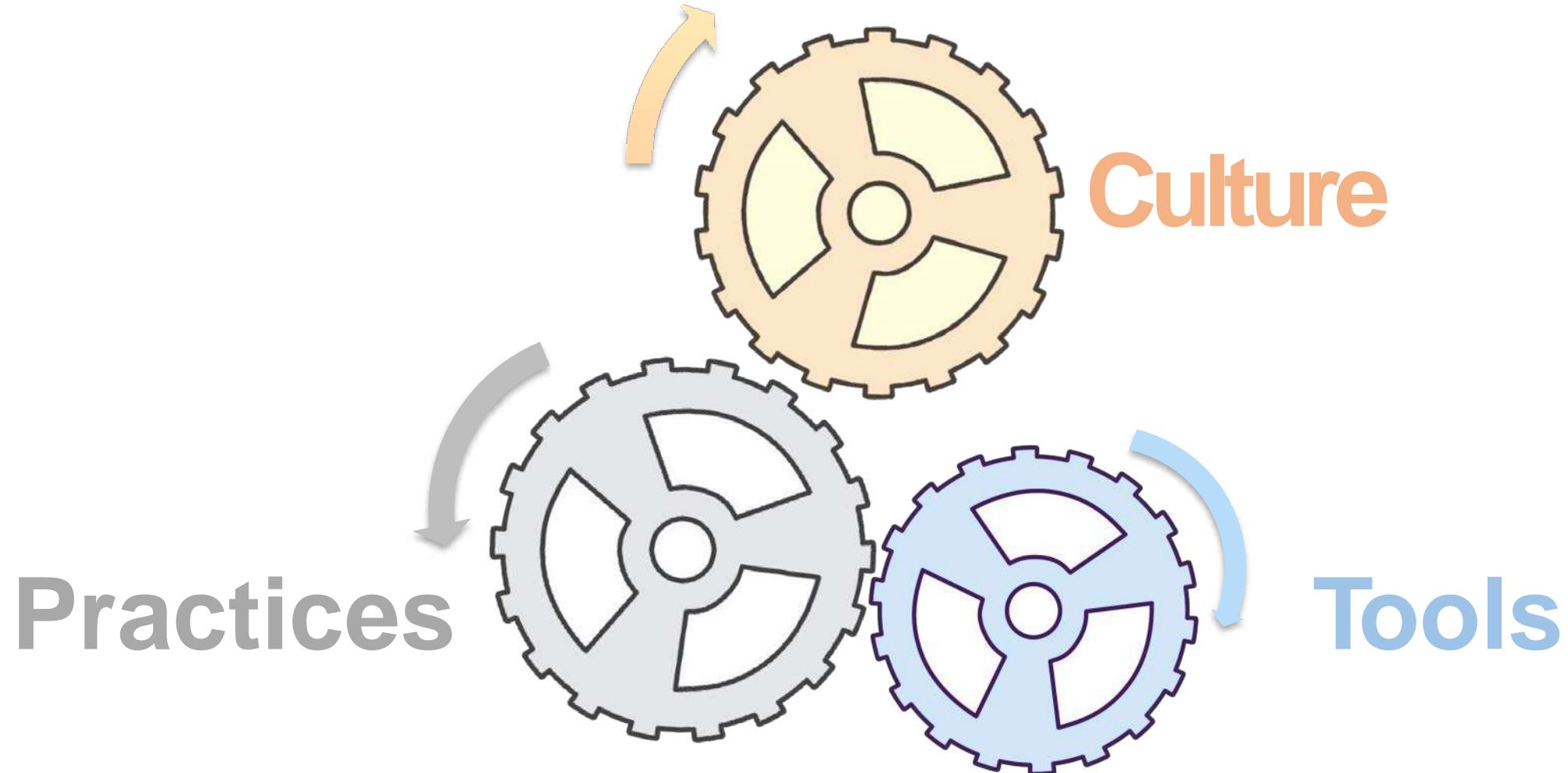
Goals of DevOps

- Improved ability to support the organisation's needs with software
- Faster development and deployment cycles
- High performance culture in software creation and operation
- Improved deployment frequency
- Lower failure rate of new releases and faster time to enhance
- Improved predictability, efficiency and maintainability
- Use of automation to improve efficiency
- Collaboration and sharing of knowledge

Source: <https://en.wikipedia.org/wiki/DevOps>



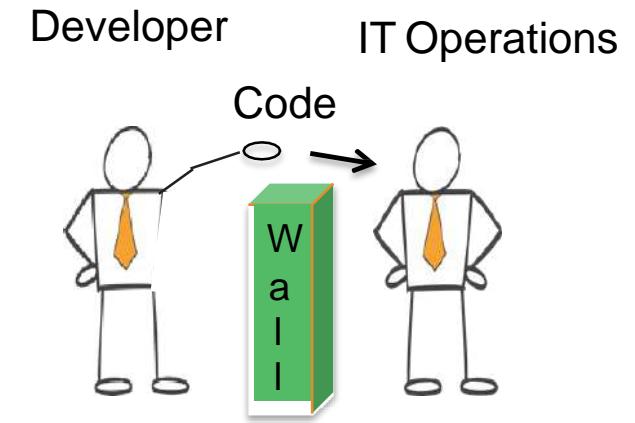
Aspects of DevOps





What is DevOps?

- A philosophy? Cultural change? Paradigm shift ?
- Alignment of development and IT operations with better communication and collaboration?
- Improvement in software deployment ?
- Breaking down the barriers between development and IT operations?
- Akin to Agile software development applied to infrastructure and IT operations?
- Set of tools and processes?
- ***It's all of the above!***

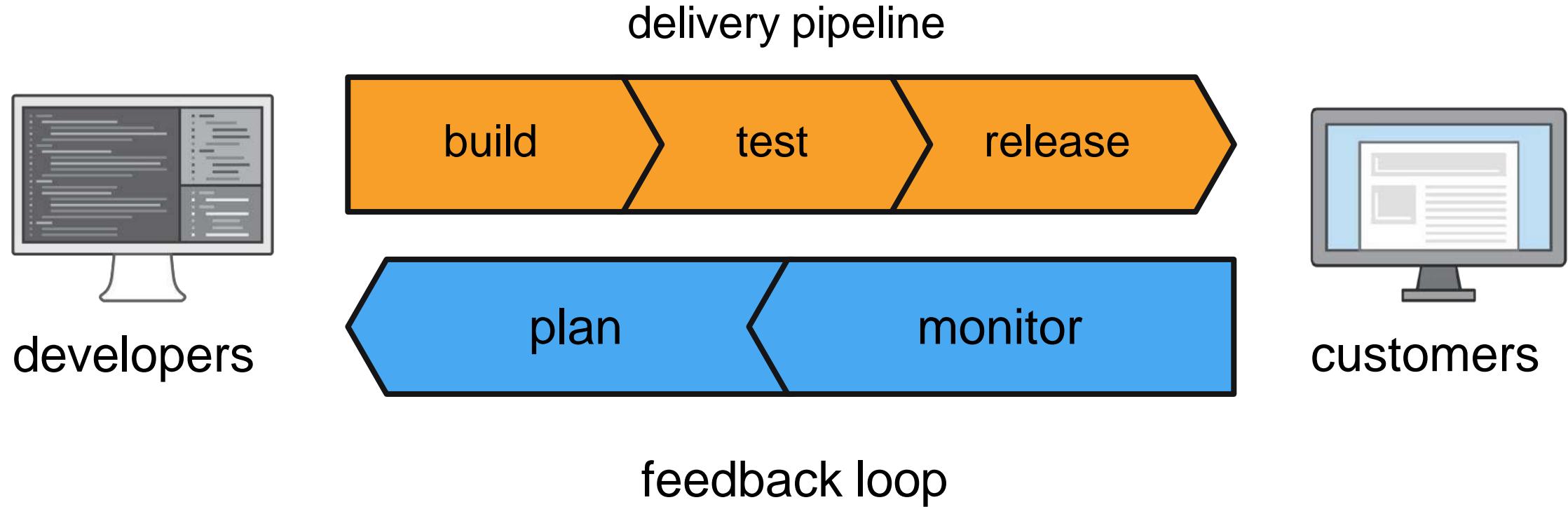


*DevOps breaks the wall
that existed between
Development and
Operations*



What is DevOps?

Software development lifecycle

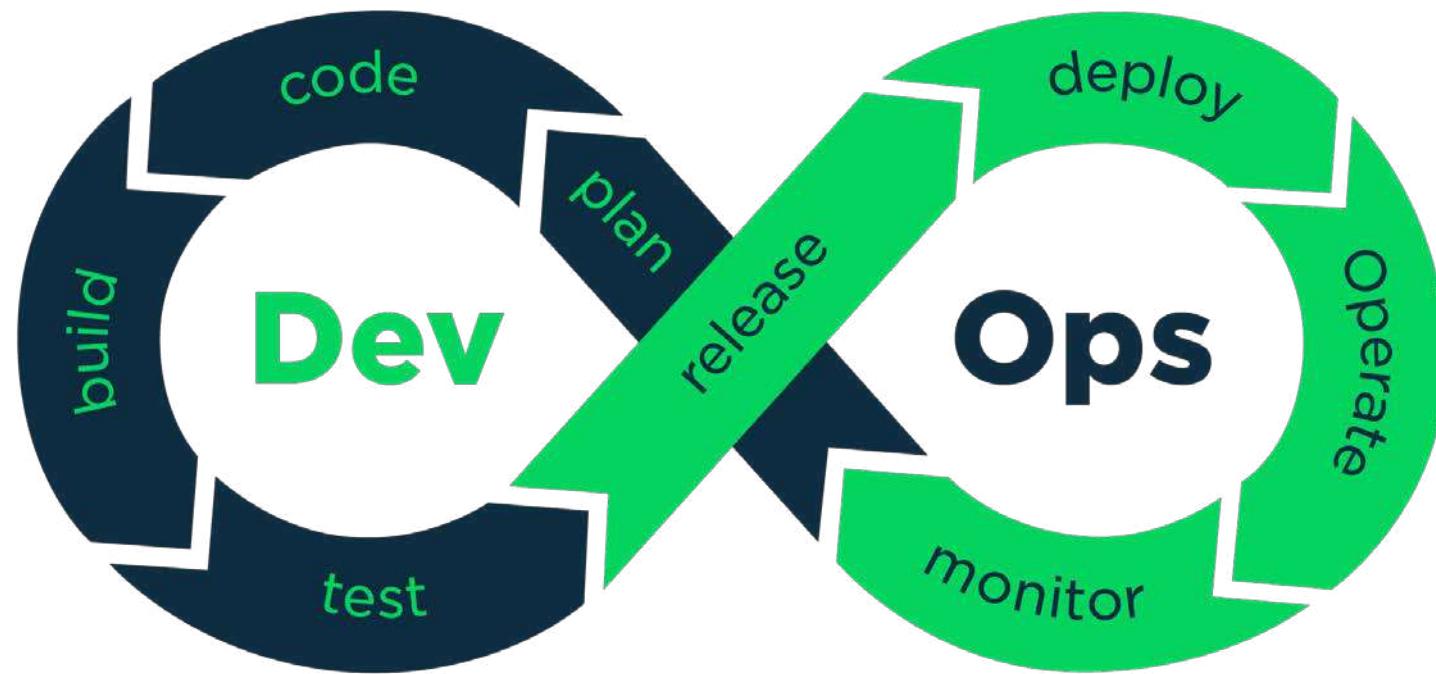


DevOps = efficiencies that speed up this lifecycle



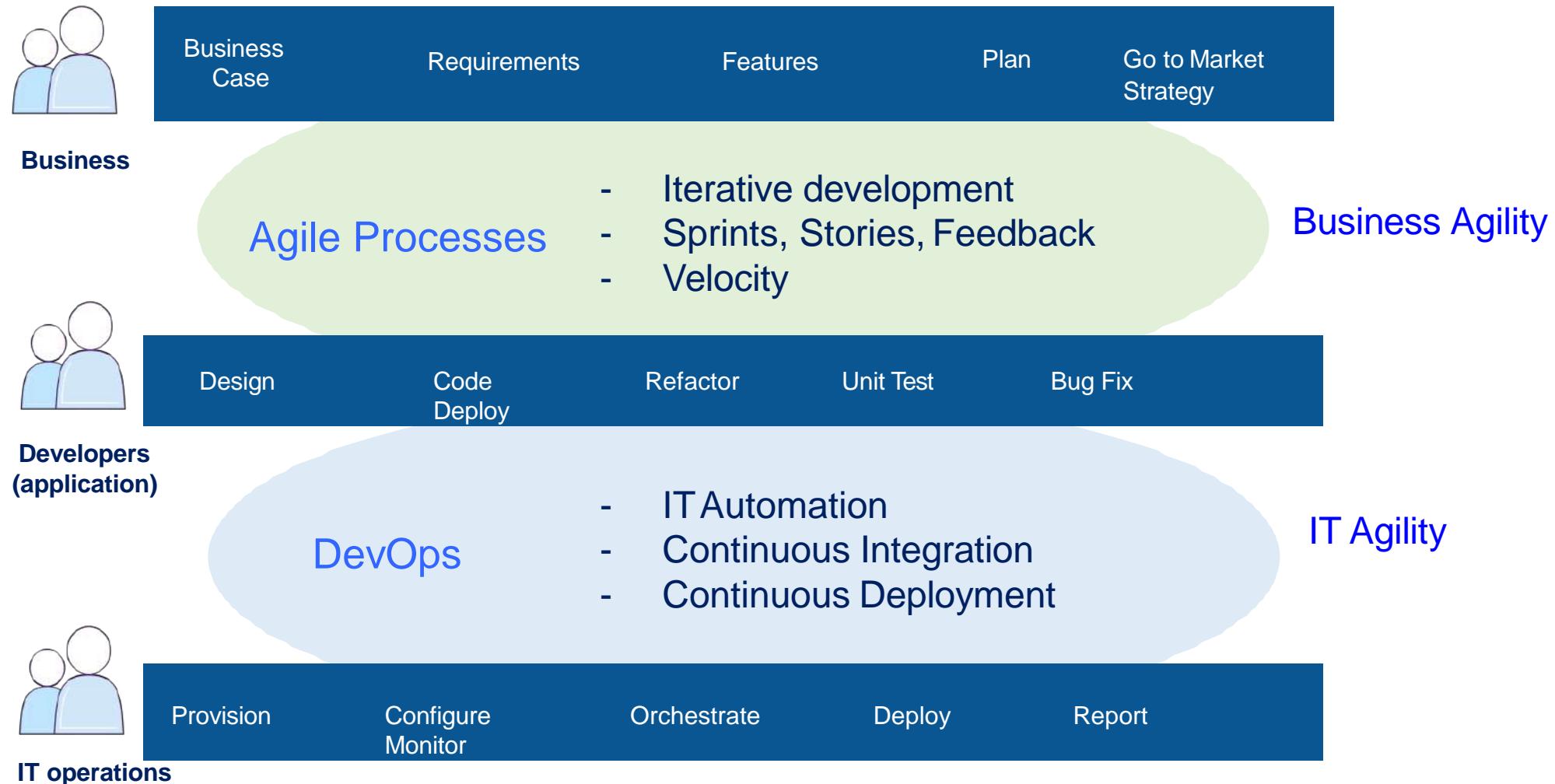
DevOps - unifying Development and Operations

- Unification and automation of software Development and Operations teams, processes and tools



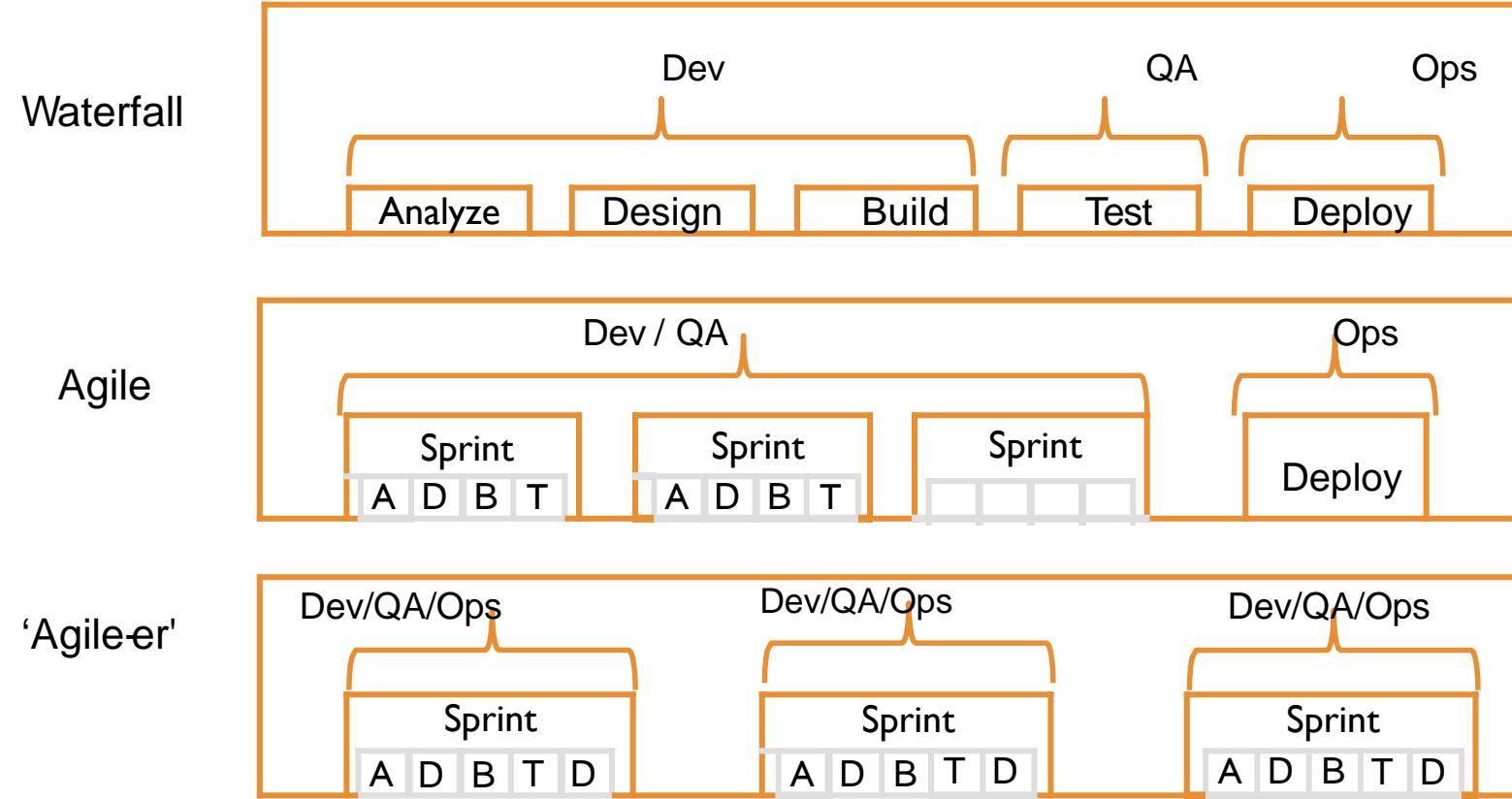


Agile processes and DevOps = Business and IT Agility





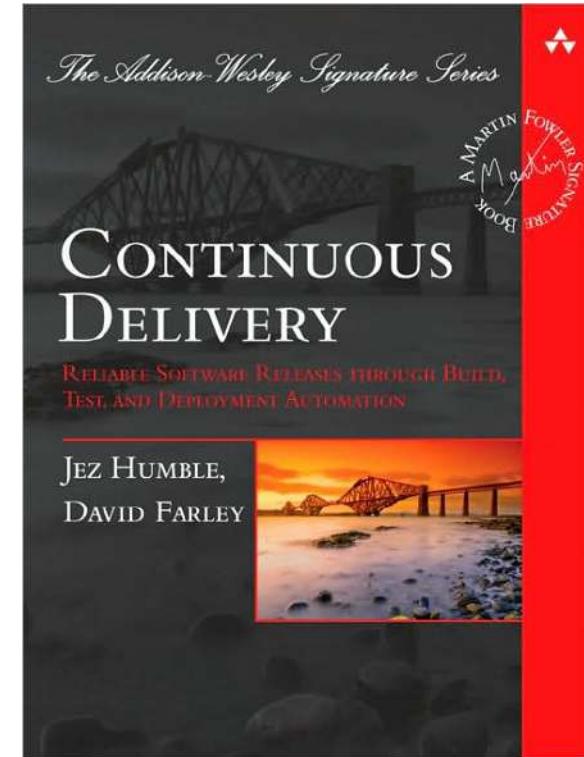
Becoming more Agile with DevOps





Continuous Delivery – the reference guide

- A very good book
- Explains how to set up your continuous delivery process
- Including CI, deployments, etc.
- Explains good software development practices that are necessary for CD



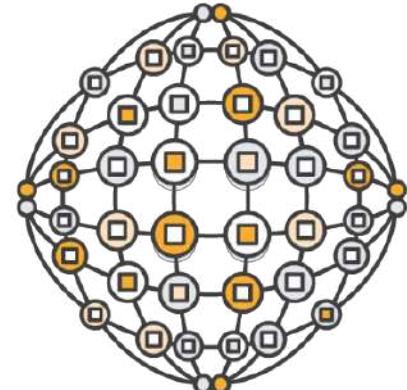


Pillars of effective DevOps

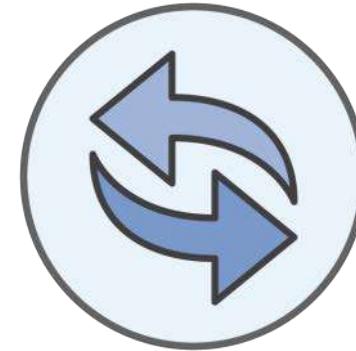
Collaboration	Affinity	Tools	Scaling
<ul style="list-style-type: none"> • Asynchronous code review • Documentation • Updating issues and bug reports • Demonstrating weekly progress • Regular status updates • Pairing 	<ul style="list-style-type: none"> • Interpersonal ties in teams • Team culture • Team cohesion • Diversity • Maintaining an inclusive environment • Finding common ground in teams • Improving team communication 	<ul style="list-style-type: none"> • Software development • Artefact management • Automation • Infrastructure automation • Test and build automation • Monitoring • Evolution of the ecosystem 	<ul style="list-style-type: none"> • Scaling in organizational context • Organizational structure • Growing and scaling teams • Team flexibility • Organizational lifecycle • Complexity and change



DevOps Practices



Microservices



Continuous Integration



Continuous Delivery



Infrastructure as Code



Monitoring and Logging



Communication and
Collaboration



DevOps: Culture + Practices + Tools

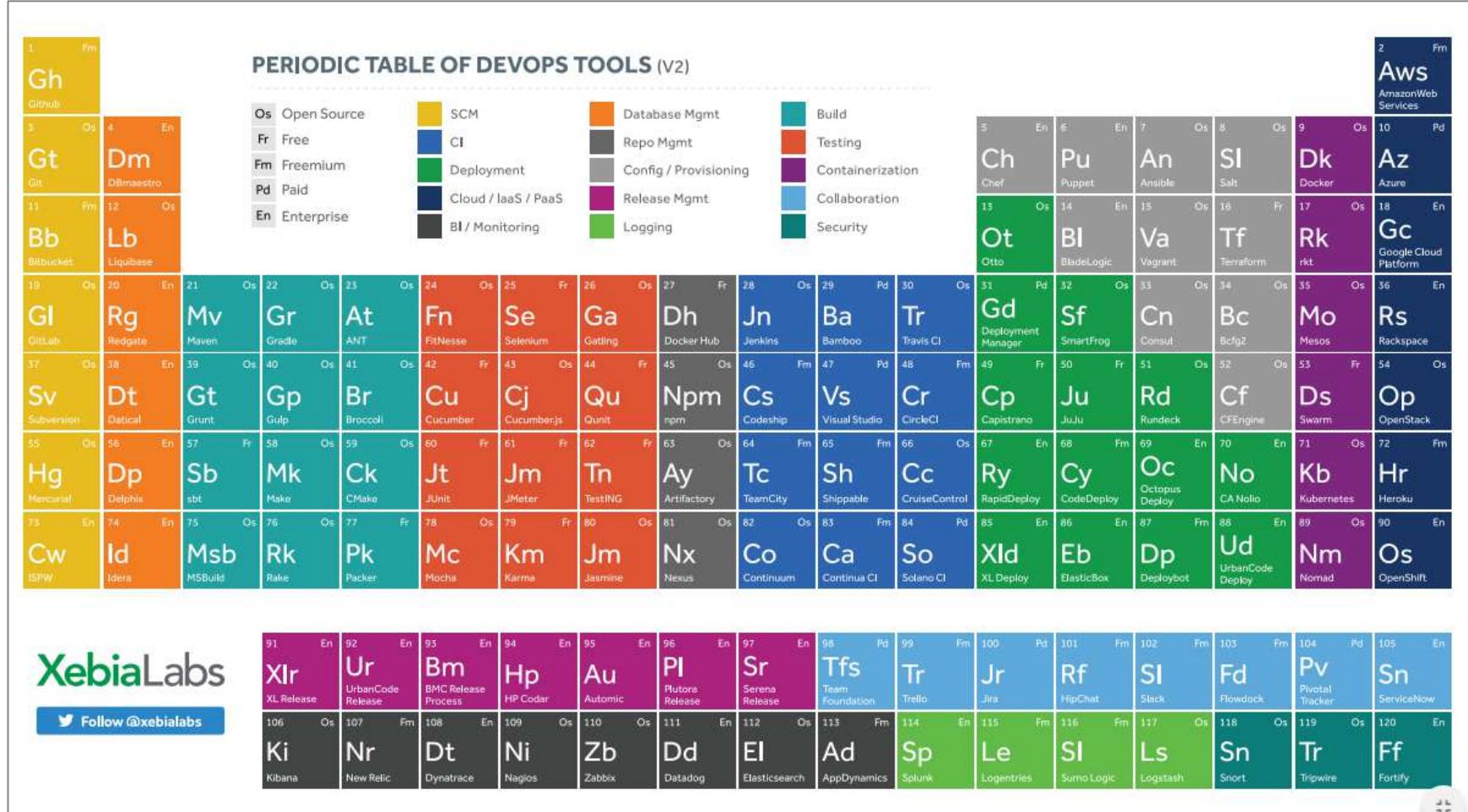
- Each DevOps feature team 'owns' their product:
 - Creates software product
 - Does Q/A of that product
 - “You build it, you run it”: supports product
- Teams generally practice Agile processes
 - Scrum is popular
- Automation using tools
 - Help to enforce best practices



<https://secure.flickr.com/photos/lox/9408028555>



Lots of DevOps tools!

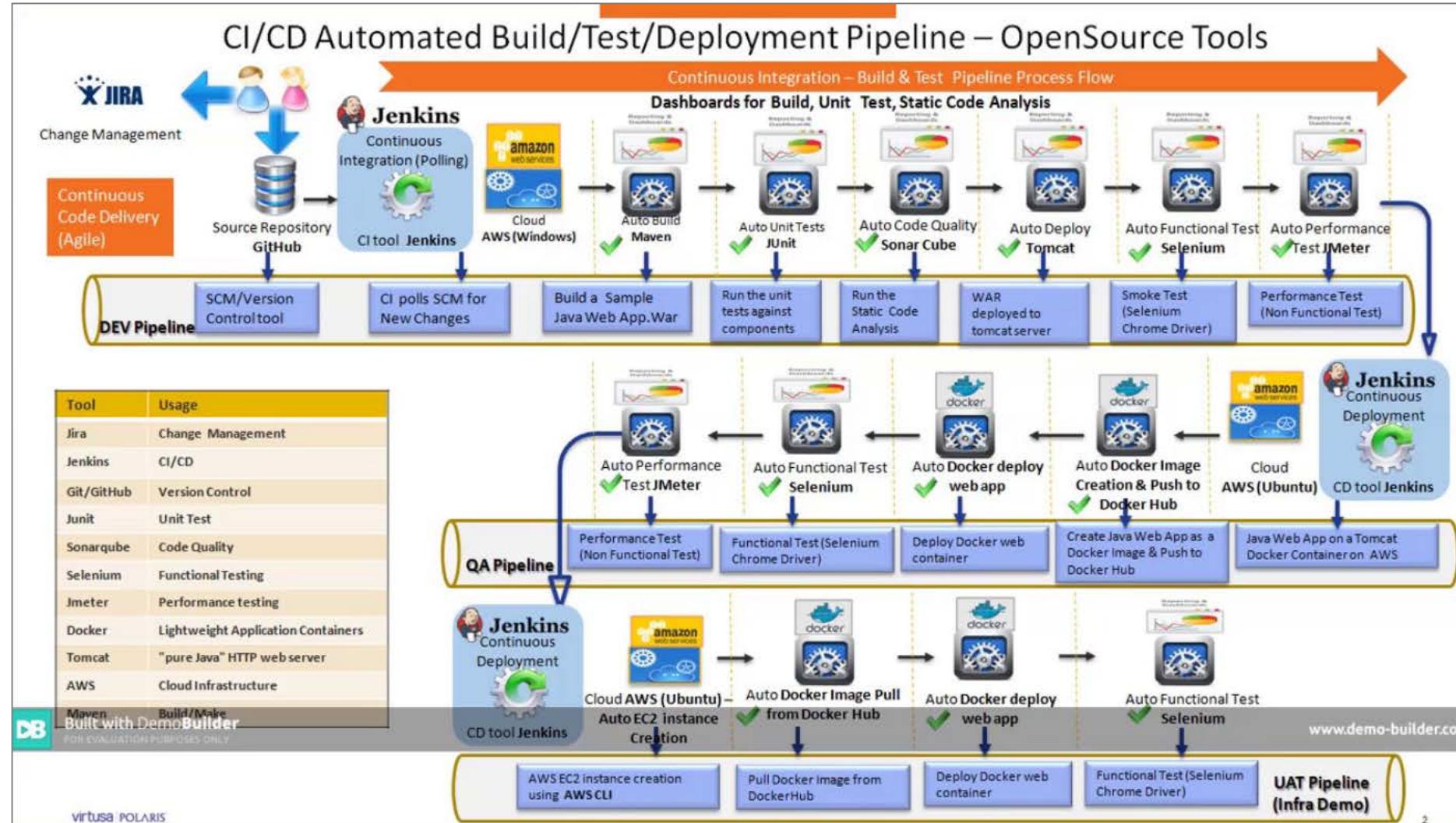


Source: <https://xebialabs.com/periodic-table-of-devops-tools/>



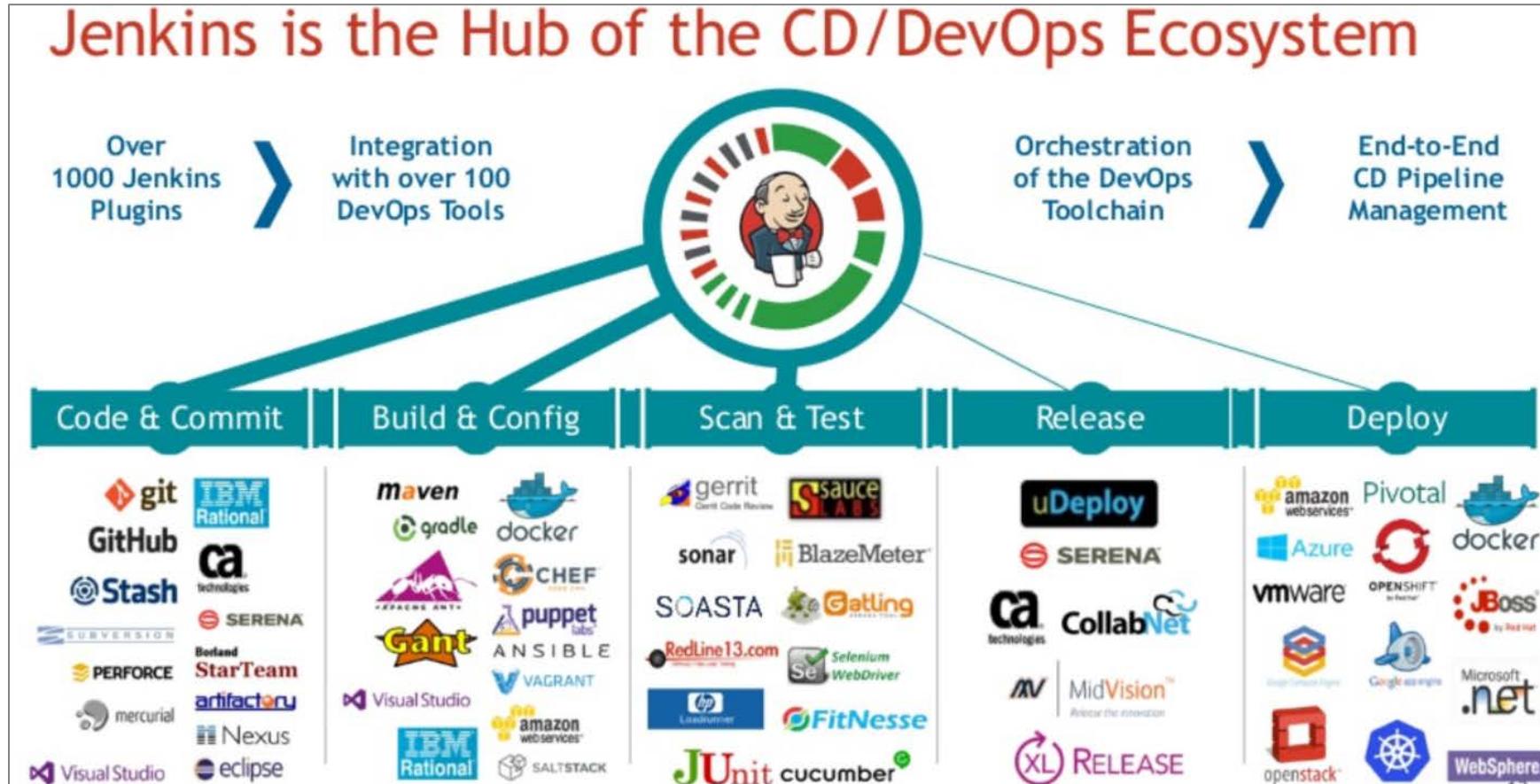
Many open source DevOps tools also!

Source: https://i.ytimg.com/vi/L5oesoeZ_iY/maxresdefault.jpg





Jenkins - a popular open source CI CD automation tool



- An open source automation server
- Automates CI CD process
- Supports many version control tools
- Can execute Apache Ant and Apache Maven projects
- Has plug-ins to integrate with many popular tools
- Can deploy to many platforms – traditional and cloud

Source:
[https://en.wikipedia.org/wiki/Jenkins_\(software\)](https://en.wikipedia.org/wiki/Jenkins_(software))

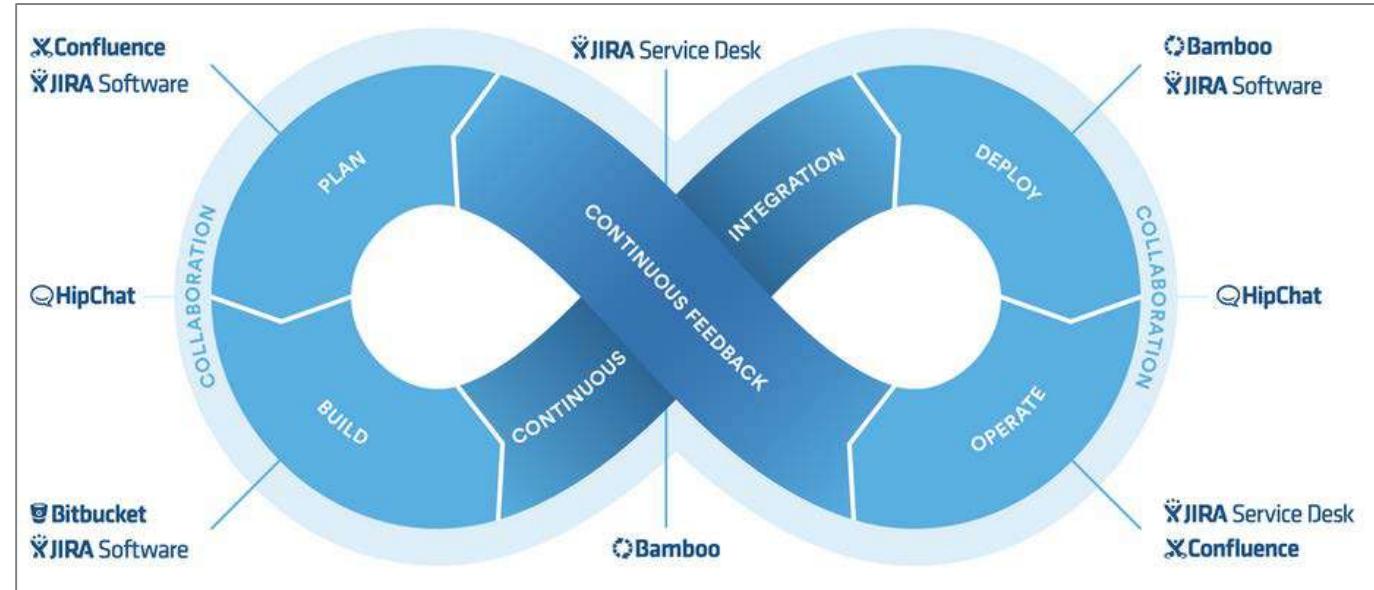
Source: <https://haritibcoblog.com/2017/03/03/jenkins-overview/>



DevOps tools from Atlassian

- Atlassian's DevOps tools are popular with individuals, startups and enterprises
- For more information: <https://www.atlassian.com/>

PLAN, TRACK, & SUPPORT	CODE, BUILD, & SHIP	COLLABORATE & CHAT
Jira Software Plan, track, and release world-class software with the #1 software development tool used by agile teams. Learn more →	Bitbucket Collaborate on code and manage your Git repositories to build and ship software, as a team. Learn more →	Confluence Spend more time getting things done. Organize your work, create documents, and discuss everything in one place. Learn more →
Jira Service Desk Give your customers an easy way to ask for help and your agents a fast way to resolve incidents. Learn more →	Sourcetree Harness the full power of Git and Mercurial in a beautifully simple application. Learn more →	Trello Collaborate and get more done. Trello boards enable your team to organize projects in a fun, flexible, and visual way. Learn more →
Statuspage Incidents happen. Keep your users informed and ditch the flood of support emails during downtime. Learn more →	Bamboo Continuous integration, deployment, and release management. Learn more →	Stride The complete communication solution that empowers teams to talk less and do more. Group messaging, video meetings, and collaboration tools. Learn more →



Source: <https://www.atlassian.com>

Source: <https://www.atlassian.com/blog/devops/how-to-choose-devops-tools>



DevOps Toolchain

- DevOps is a set of tool chain that will help organizations achieve their goals. The tools fit into several categories that are:
 - **Code** - SCM, workflow, development, review and merge.
 - **Build** - Continuous integration and builds.
 - **Test** - Continuous testing that provide feedback.
 - **Package** - Packaging code and release to artifact repository.
 - **Release** - Release, promotion and change management.
 - **Configure** - Automated Infrastructure and configuration management.
 - **Monitor** - Application performance and log monitoring



Assumptions

- Basic understanding of the SDLC lifecycle.
- Basic understanding of the source code management.
- Ability to install tools as part of the course.
- Ability to configure tools as part of the course.
- Basic understanding of Java programming language and build tools like Maven.
- Ability to hands on development in Java using Spring Boot.
- Please note that though the focus here is using Java programming the tool chain can be applied to any technology.



SUMMARY & REFERENCES



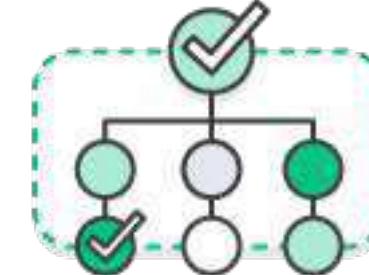
Summary: DevOps in the cloud - easily get benefits



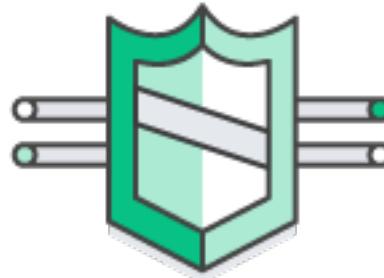
Improved Collaboration



Rapid Delivery



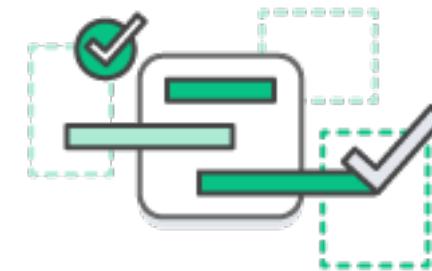
Reliability



Security



Scale



Speed



Resource and Tools Prerequisite

Basic:

- GitHub Account:

<https://github.com/join>

- Install GIT:

<https://www.git-scm.com/downloads>

- Install Atlassian SourceTree:

<https://www.sourcetreeapp.com>

Optional:

- Install GIT FLOW for executing via Command Line

<https://github.com/nvie/gitflow/wiki/Installation>

- Collection of DevOps related tools

• <http://www.devopsbookmarks.com>

- DevOps site for AWS

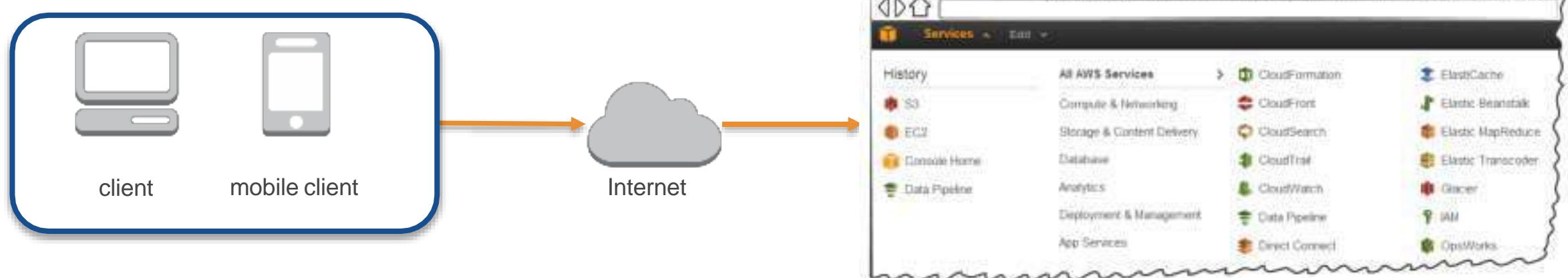
• <https://aws.amazon.com/devops/>

- Download and Install STS: <https://spring.io/tools/sts/all>
- Download Editor like ATOM and Install: <https://atom.io>
- Download Maven: <http://maven.apache.org>
- Download Tomcat: <https://tomcat.apache.org/download-80.cgi>
- Download Artifactory: <https://www.jfrog.com/open-source/>
- Download Jenkins: <https://jenkins.io/download/>
- Download ElasticSearch: <https://www.elastic.co/downloads/elasticsearch>
- Download Logstash: <https://www.elastic.co/downloads/logstash>
- Download Kibana: <https://www.elastic.co/downloads/kibana>



On-Demand Self Service

- User provisions computing resources as needed
- User interacts with cloud service through an online 'console'
- Access is available from a variety of network-connected devices





AWS Global Infrastructure



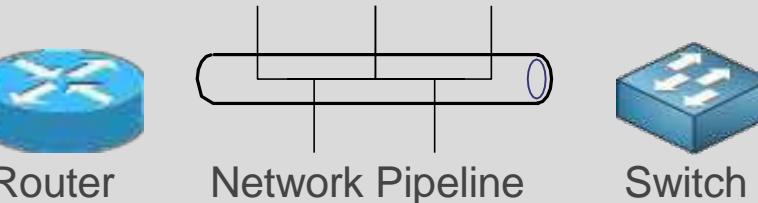


AWS Core Infrastructure and Services

Traditional Infrastructure



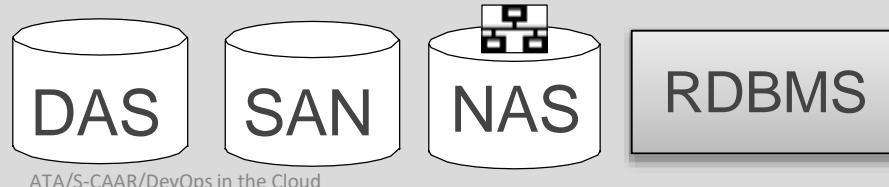
Security



Networking

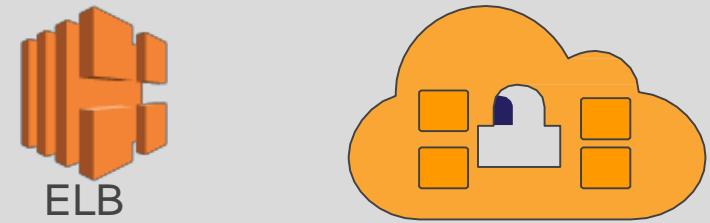


Servers



**Storage
and
Database**

Amazon Web Services





DevOps with AWS

DevOps and AWS
Tooling and infrastructure resources for DevOps practitioners
[Get Started with AWS](#)

What is DevOps? DevOps Blog Partner Solutions Resources

AWS provides a set of flexible services designed to enable companies to more rapidly and reliably build and deliver products using AWS and DevOps practices. These services simplify provisioning and managing infrastructure, deploying application code, automating software release processes, and monitoring your application and infrastructure performance.

DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market. [Learn more about DevOps »](#)

Why AWS for DevOps?

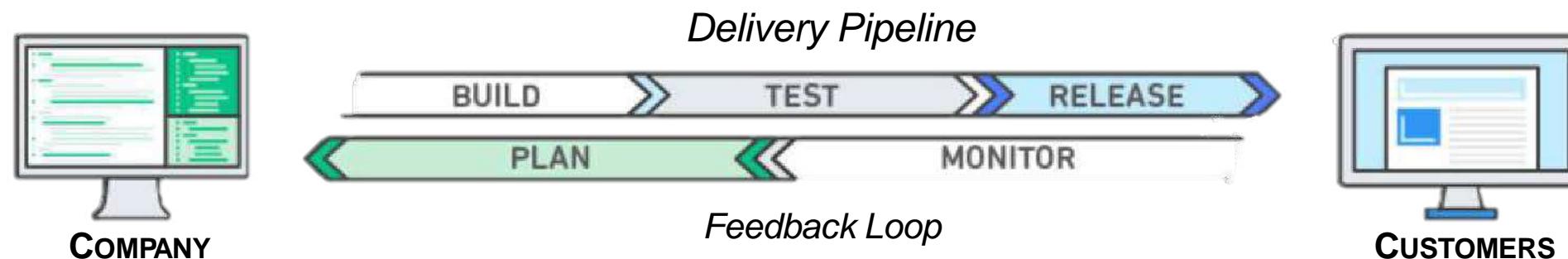
- Get Started Fast**: Each AWS service is ready to use if you have an AWS account. There is no setup required or software to install.
- Fully Managed Services**: These services can help you take advantage of AWS resources quicker. You can worry less about setting up, installing, and operating infrastructure on your own. This lets you focus on your core product.
- Built for Scale**: You can manage a single instance or scale to thousands using AWS services. These services help you make the most of flexible compute resources by simplifying provisioning, configuration, and scaling.
- Programmable**: You have the option to use each service via the AWS Command Line Interface or through APIs and SDKs. You can write, model, and provision AWS resources and your entire AWS infrastructure using declarative AWS CloudFormation templates.
- Automation**: AWS helps you use automation so you can build faster and more efficiently. Using AWS services, you can automate manual tasks or processes such as deployments, development & test workflows, container management, and configuration management.
- Secure**: Use AWS Identity and Access Management (IAM) to set user permissions and policies. This gives you granular control over who can access your resources and how they access these resources.
- Large Partner Ecosystem**: AWS supports a large ecosystem of partners which integrate with and extend AWS services. Use your preferred third-party and open-source tools with AWS to build an end-to-end solution. Visit [here](#) to learn more about our DevOps Partner Solutions.
- Pay-As-You-Go**: With AWS purchase services as you need them and only for the period when you plan to use them. AWS pricing has no upfront fees, termination penalties, or long term contracts. The AWS Free Tier helps you get started with AWS. Visit the pricing pages of each service to learn more.

- AWS provides a rich set of services for DevOps
- Home page: <https://aws.amazon.com/devops/>
- Getting started guide:
<http://docs.aws.amazon.com/devops/latest/gsg/welcome.html>
- Tutorials: <https://aws.amazon.com/getting-started/use-cases/devops/>
- Whitepapers: <https://aws.amazon.com/whitepapers/>
 - Introduction to DevOps on AWS [PDF](#)
 - Development and Test on AWS [PDF](#)
 - Practicing Continuous Integration and Continuous Delivery on AWS [PDF](#)
 - Jenkins on AWS [PDF](#)
 - Infrastructure as Code [PDF](#)
 - Blue/Green Deployments on AWS [PDF](#)
 - Microservices on AWS [PDF](#)



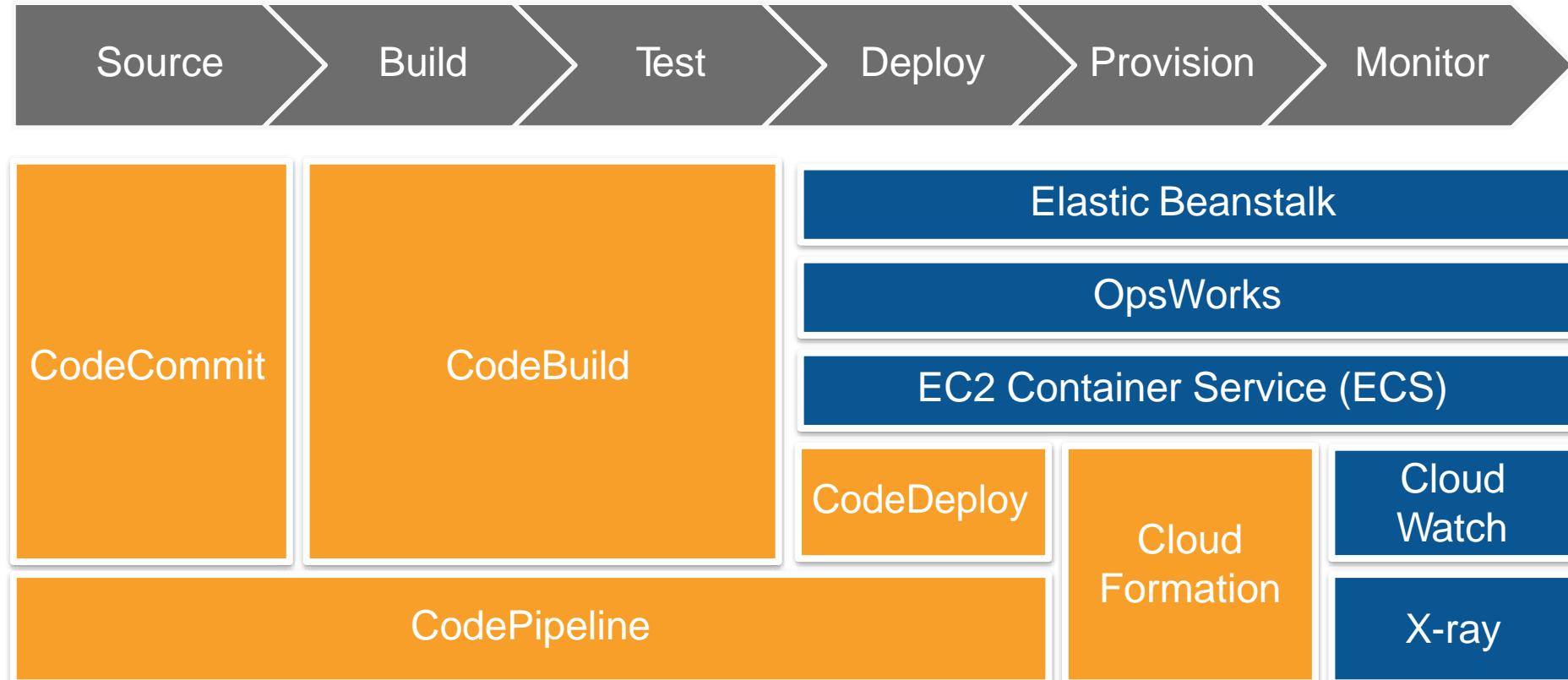
DevOps according to AWS

- Unification of software development and operations
- Migration of Agile continuous development into continuous integration and continuous delivery
- DevOps model:
 - **No silos** – emphasis on communication, collaboration, and cohesion between disciplines
 - Best practices for change, configuration, and deployment automation
 - Deliver apps/services at faster pace
 - High speed product updates





DevOps tools stack on AWS





DevOps Release Processes: major phases

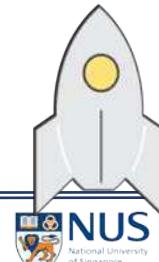
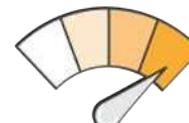
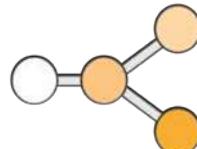
Source

Build

Test

Production

- Check-in source code such as .java files.
- Peer review new code
- Compile code
- Unit tests
- Style checkers
- Code metrics
- Create container images
- Integration tests with other systems
- Load testing
- UI tests
- Penetration testing
- Deployment to production environments



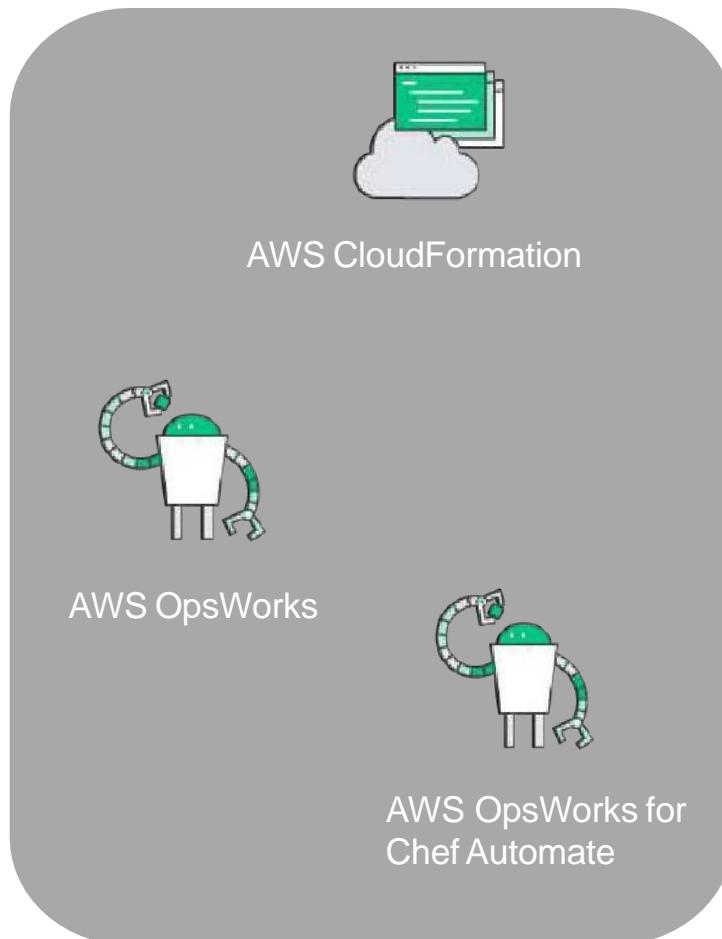


AWS DevOps Portfolio

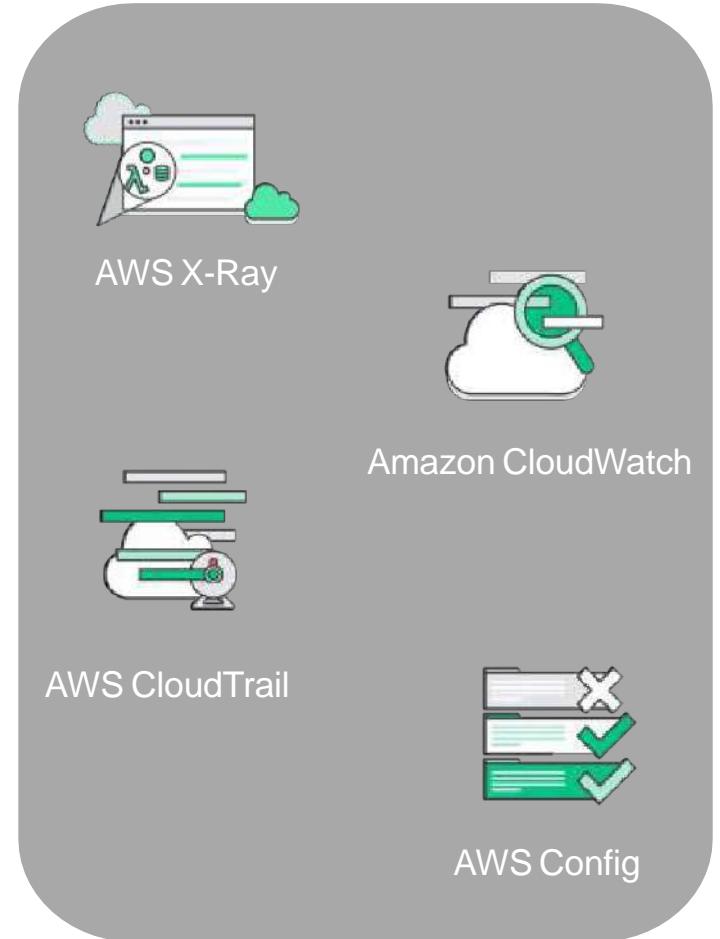
Software Development and Continuous Delivery



Infrastructure as Code



Monitoring & Logging





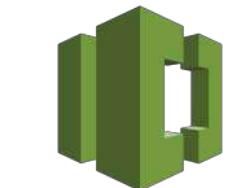
AWS CodePipeline - Integrations

Source

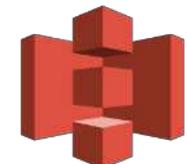
Build

Test

Deploy

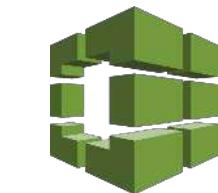


AWS CodeCommit



Amazon S3

GitHub



AWS CodeBuild



CloudBees



Jenkins



Solano Labs

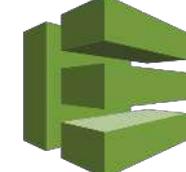


BlazeMeter



HPE StormRunner

Runscope



AWS CodeDeploy



AWS
CloudFormation



AWS
OpsWorks



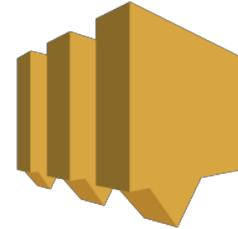
AWS Elastic
Beanstalk

XebiaLabs
Deliver Faster



AWS CodePipeline - integrations (continued)

Approval



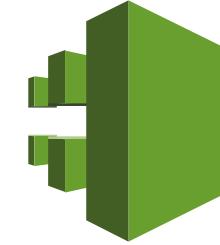
Amazon Simple
Notification Services
(SNS)

Invoke Logic



Amazon
Lambda

Management



AWS CloudTrail

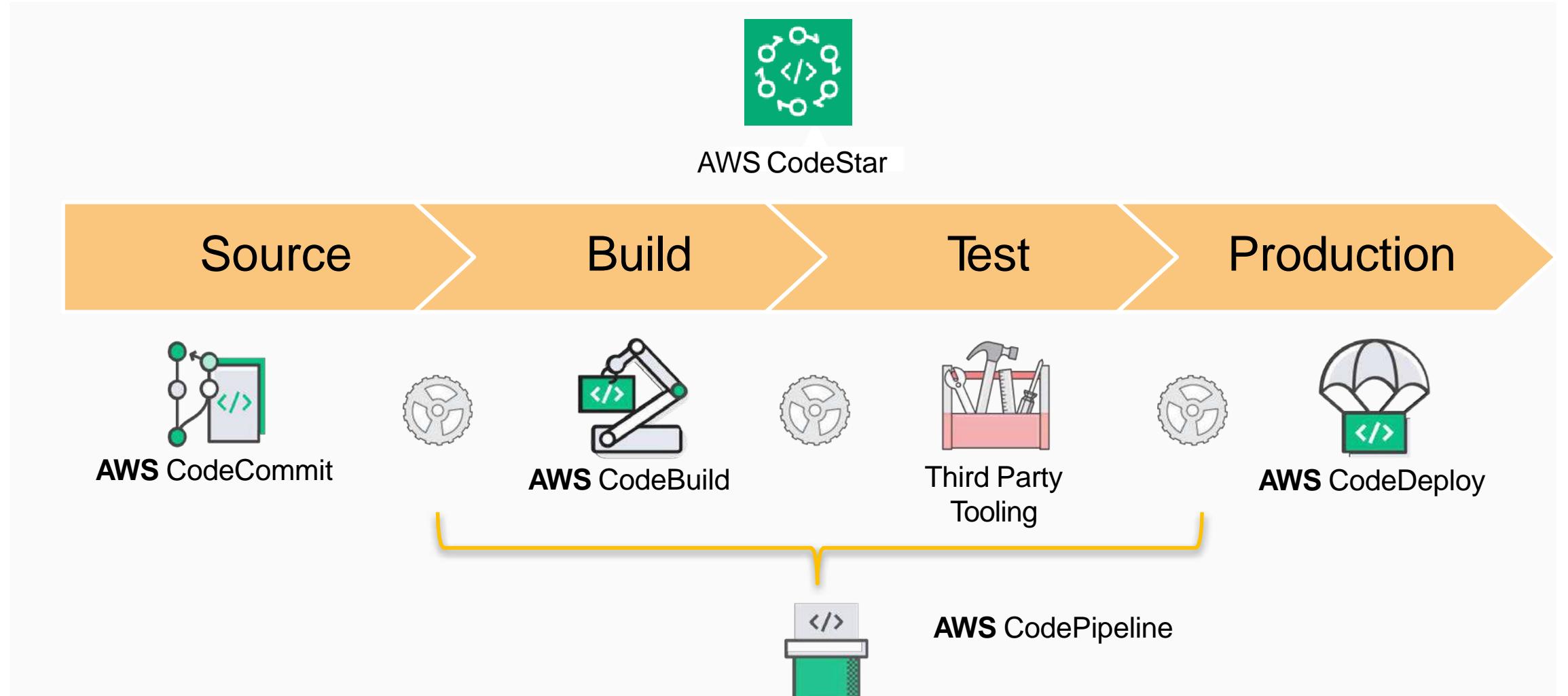


AWS Key
Management Service
(KMS)



AWS CodeStar - Implement AWS DevOps

Software Development and Continuous Integration & Delivery Workflow





Features of AWS CodeStar



- ✓ Project Templates
- ✓ Team Access Management with AWS IAM



- ✓ Managed Build Service with AWS Code Build



- ✓ Unified Project Dashboard using Amazon CloudWatch monitoring service



- ✓ Issue tracking and project management tool in dashboard via integrated Atlassian JIRA Software



AWS CodeCommit for Secure Hosted Git Repository

Automated App Deployments with AWS CodeDeploy and AWS CloudFormation

- ✓ Integration of AWS CodePipeline for Automated Continuous Delivery Pipeline



AWS CodeStar – Project Templates

- CodeStar project templates support popular languages and IDEs

Programming Languages

Java



JavaScript



Python



Ruby



PHP



IDE/Code Editors

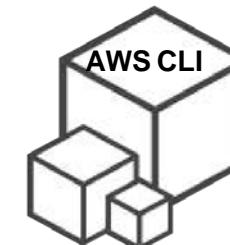
Visual Studio



Eclipse



AWS CLI





Cloud Native Solution Design

DEVOPS IN THE CLOUD

Suria R Asai

suria@nus.edu.sg

Institute of Systems Science

National University of Singapore



Objectives

- Upon completion of this module, you will be able to:
 - Understand what is DevOps and why it has become the norm for software production
 - Define what is Cloud Computing, including its characteristics, service delivery models and deployment models
 - Understand the motivations for doing DevOps in the Cloud
 - Become familiar with DevOps tools provided by a leading Cloud platform, AWS



Topics

- Understand what is DevOps and why it has become the norm for software production
- Define what is Cloud Computing, including its characteristics, service delivery models and deployment models
- Become familiar with DevOps tools provided by a leading Cloud platform, AWS



Innovation is essential for success

- Today's environment: dynamic business and social landscape, increasing competition, constraints on budgets, etc.
- New technologies are accelerating disruption
- Transformation is vital for success
- Innovation creates new digital products and services
- Ability to develop and release software quickly and efficiently is key to disrupting and transforming





Definitions of DevOps

- DevOps ('Development' + 'Operations') is a **software engineering practice** for unifying software development and software operation
- Characteristics of DevOps:
 - Close alignment with business objectives through **Agile processes**
 - Automation and **monitoring** across integration, testing, deployment and management
 - Shorter development **cycles**
 - Increased deployment frequency
 - More dependable releases

DevOps represents a change in **IT culture**, focusing on **rapid IT service delivery** through the adoption of **agile, lean practices** in the context of a system-oriented approach.

Emphasizes **people** (and **culture**), and improves **collaboration** between **operations** and **development teams**. Utilizes technology, especially **automation tools** that can leverage an increasingly **programmable** and **dynamic** infrastructure from a **life cycle** perspective.

Source: Wikipedia, <https://en.wikipedia.org/wiki/DevOps>

Source: Gartner, <https://www.gartner.com/it-glossary/devops>



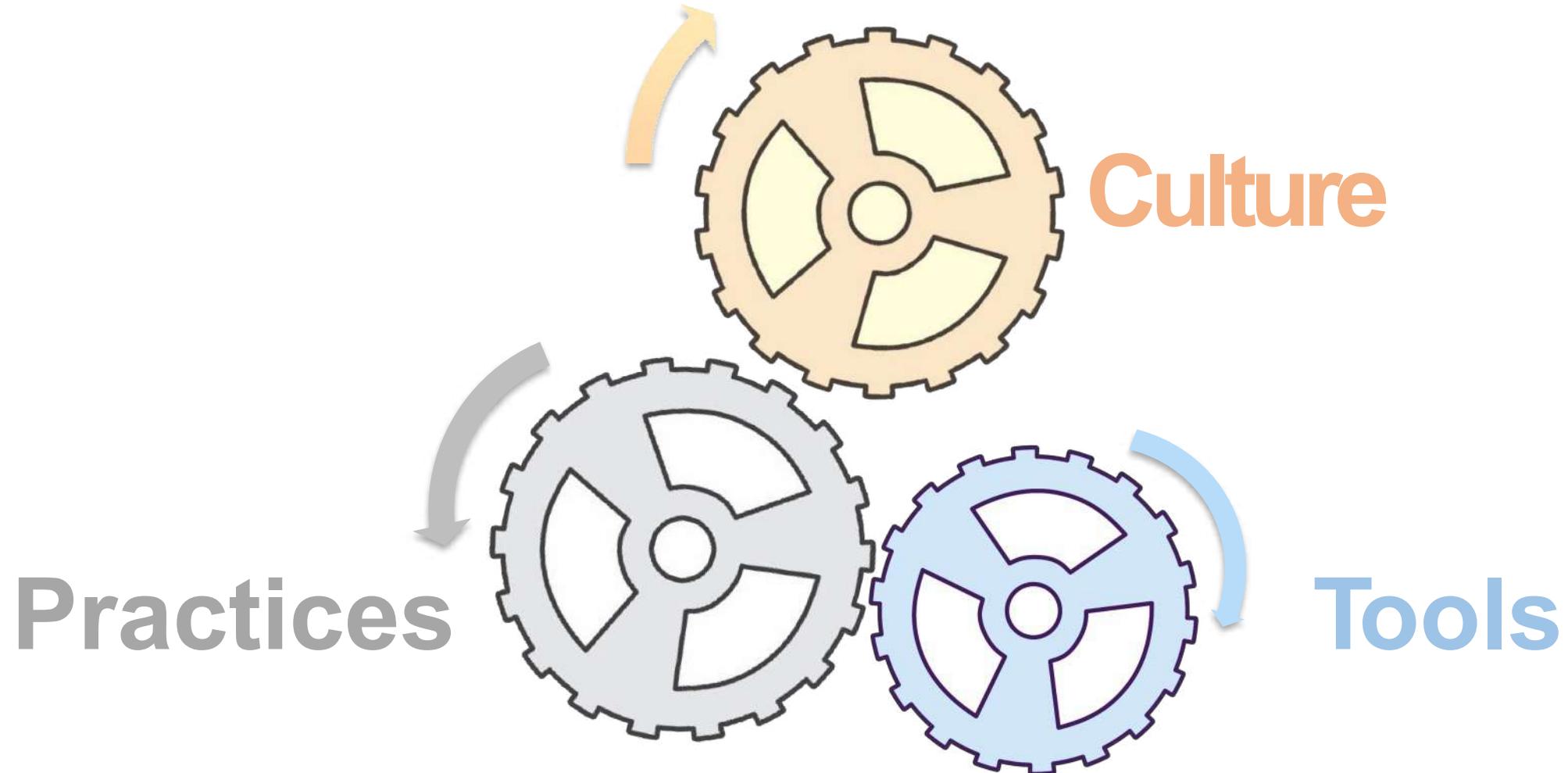
Goals of DevOps

- Improved ability to support the organisation's needs with software
- Faster development and deployment cycles
- High performance culture in software creation and operation
- Improved deployment frequency
- Lower failure rate of new releases and faster time to enhance
- Improved predictability, efficiency and maintainability
- Use of automation to improve efficiency
- Collaboration and sharing of knowledge

Source: <https://en.wikipedia.org/wiki/DevOps>



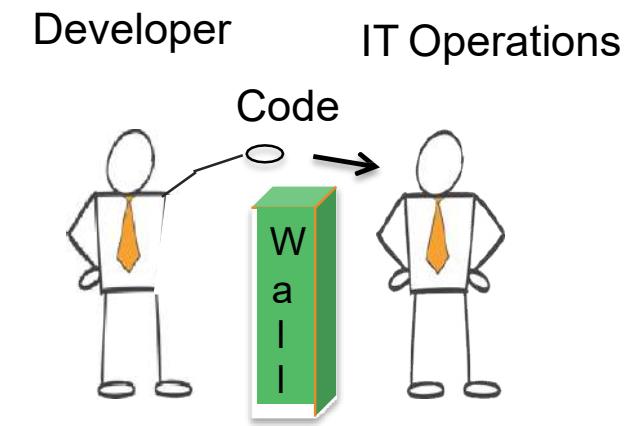
Aspects of DevOps





What is DevOps?

- A philosophy? Cultural change? Paradigm shift ?
- Alignment of development and IT operations with better communication and collaboration?
- Improvement in software deployment ?
- Breaking down the barriers between development and IT operations?
- Akin to Agile software development applied to infrastructure and IT operations?
- Set of tools and processes?
- ***It's all of the above!***

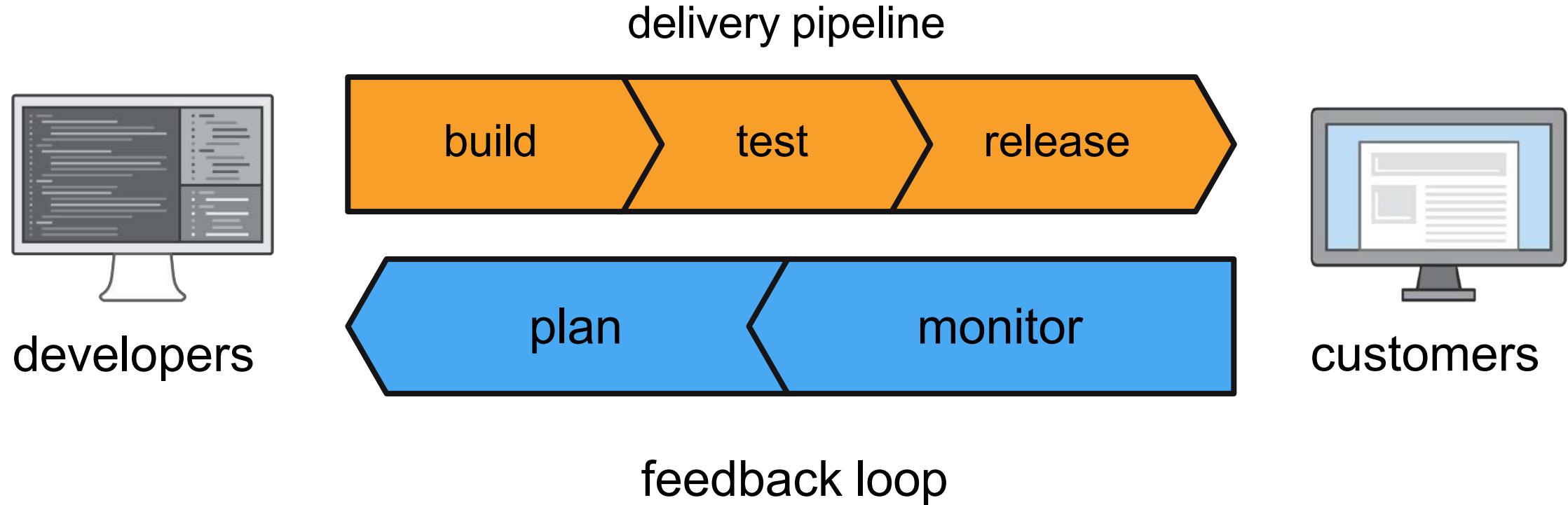


*DevOps breaks the wall
that existed between
Development and
Operations*



What is DevOps?

Software development lifecycle

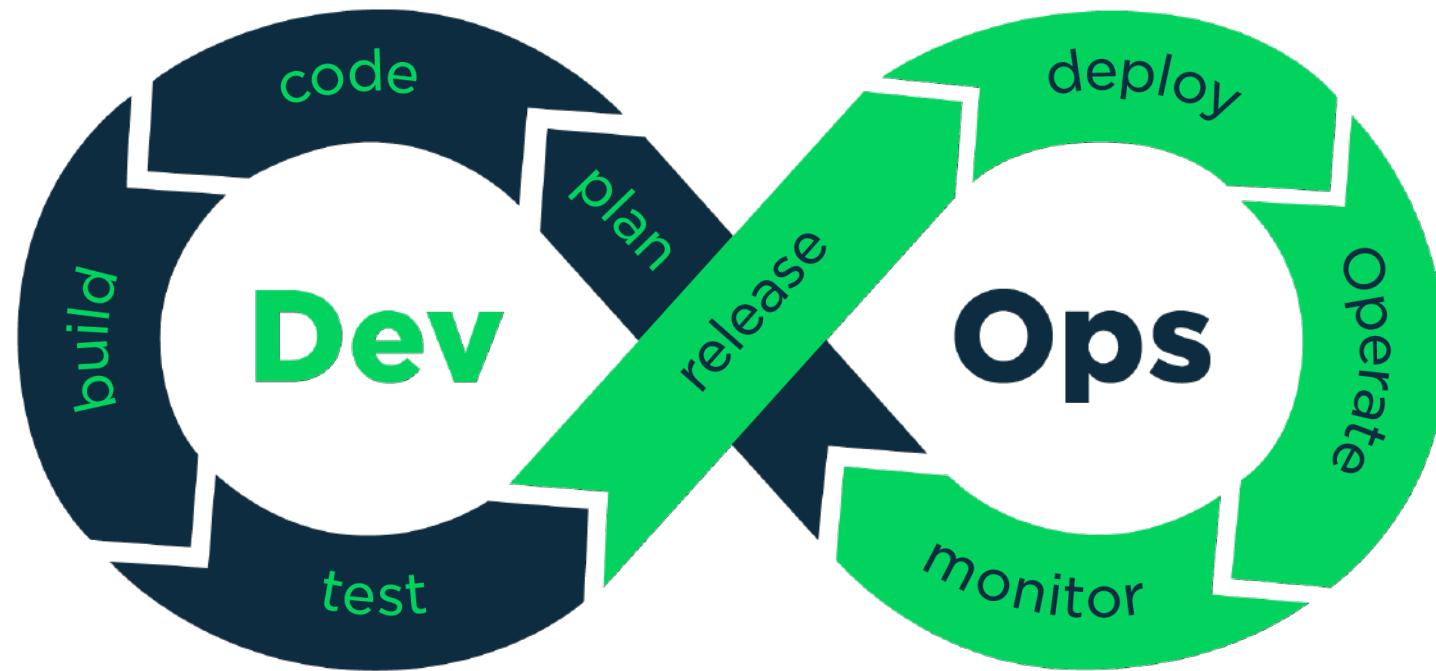


DevOps = efficiencies that speed up this lifecycle



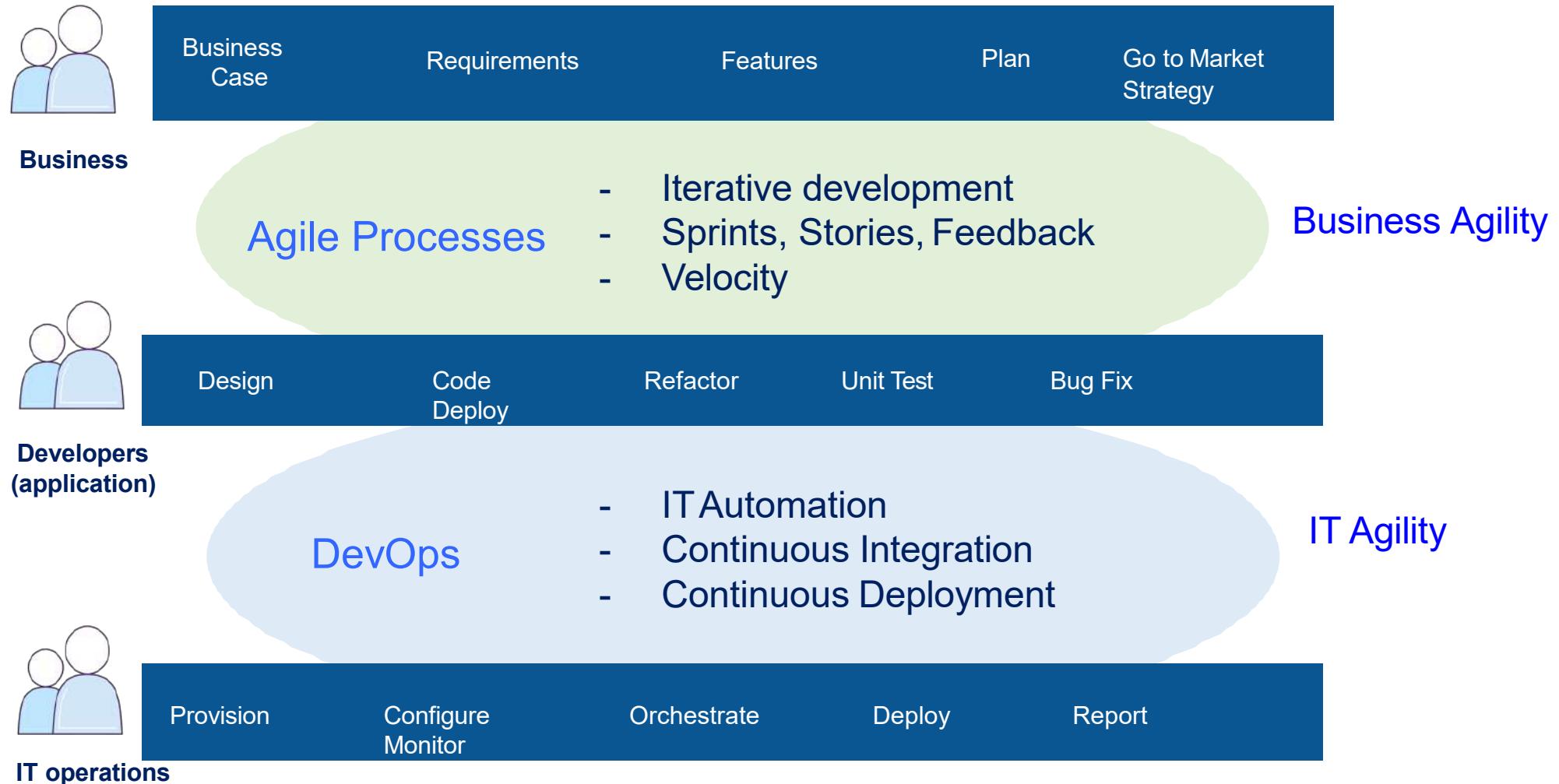
DevOps - unifying Development and Operations

- Unification and automation of software Development and Operations teams, processes and tools



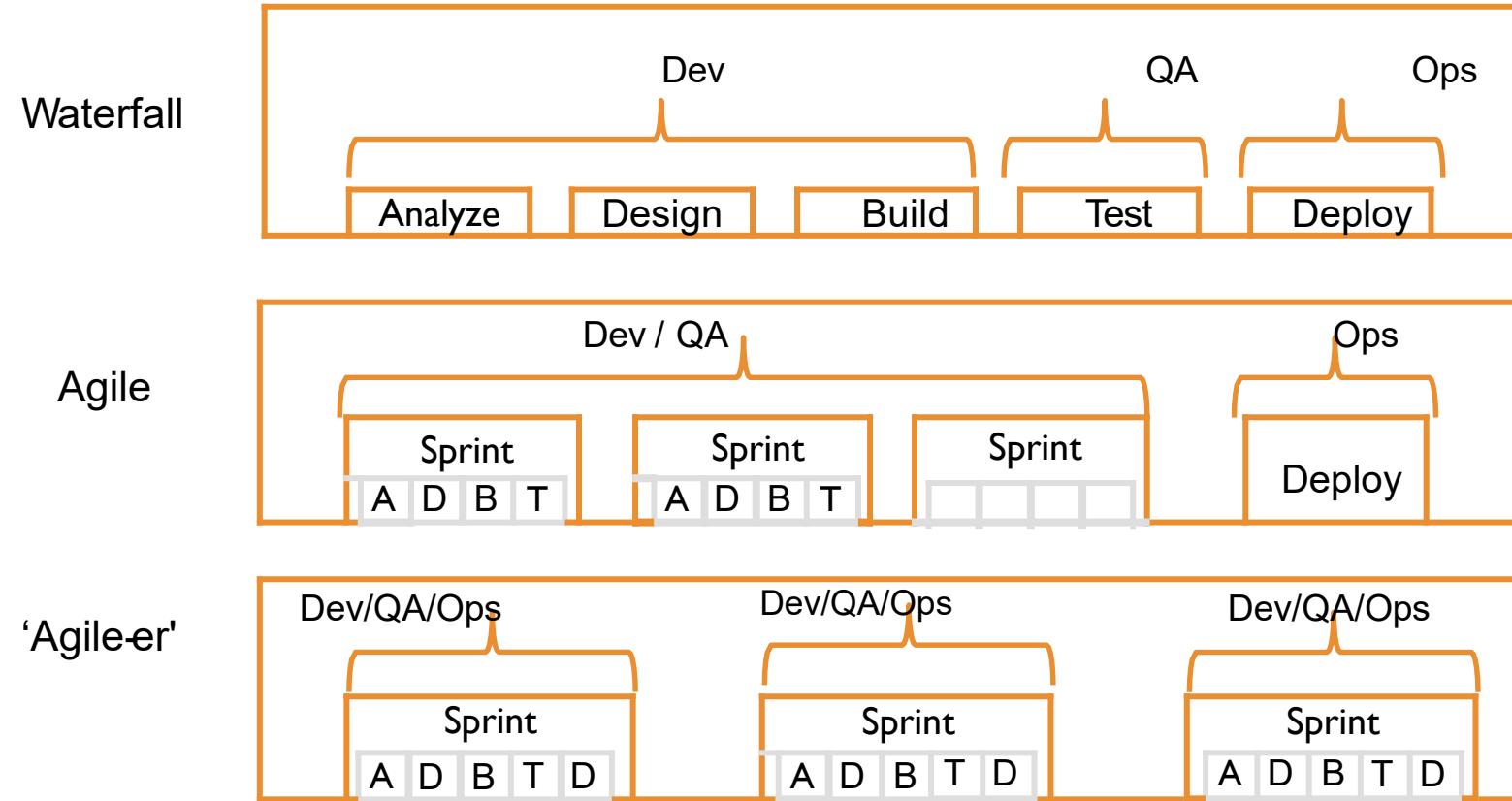


Agile processes and DevOps = Business and IT Agility





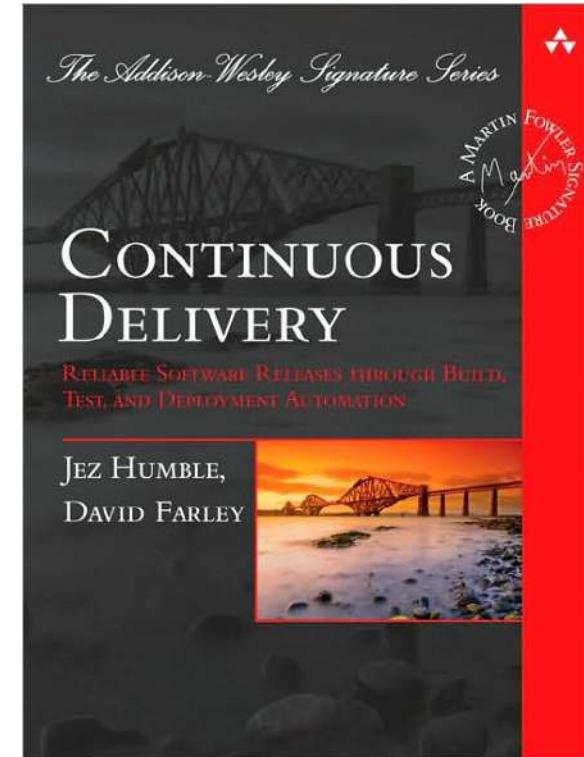
Becoming more Agile with DevOps





Continuous Delivery – the reference guide

- A very good book
- Explains how to set up your continuous delivery process
- Including CI, deployments, etc.
- Explains good software development practices that are necessary for CD



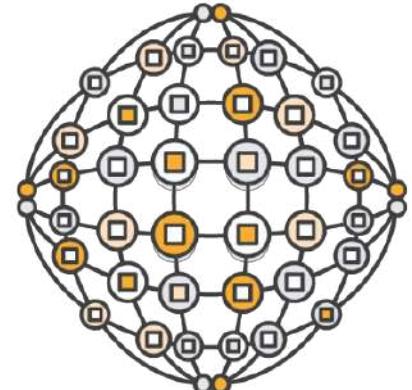


Pillars of effective DevOps

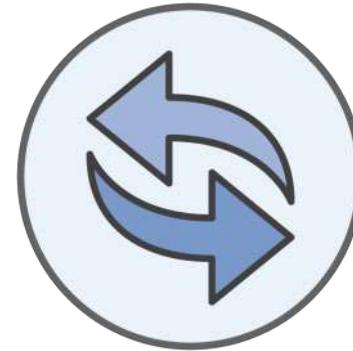
Collaboration	Affinity	Tools	Scaling
<ul style="list-style-type: none"> • Asynchronous code review • Documentation • Updating issues and bug reports • Demonstrating weekly progress • Regular status updates • Pairing 	<ul style="list-style-type: none"> • Interpersonal ties in teams • Team culture • Team cohesion • Diversity • Maintaining an inclusive environment • Finding common ground in teams • Improving team communication 	<ul style="list-style-type: none"> • Software development • Artefact management • Automation • Infrastructure automation • Test and build automation • Monitoring • Evolution of the ecosystem 	<ul style="list-style-type: none"> • Scaling in organizational context • Organizational structure • Growing and scaling teams • Team flexibility • Organizational lifecycle • Complexity and change



DevOps Practices



Microservices



Continuous Integration



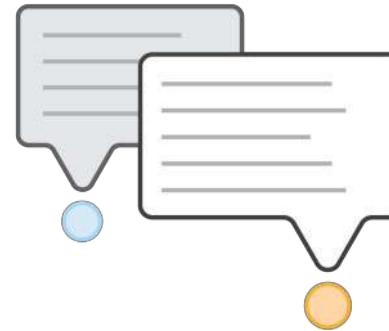
Continuous Delivery



Infrastructure as Code



Monitoring and Logging



Communication and
Collaboration



DevOps: Culture + Practices + Tools

- Each DevOps feature team 'owns' their product:
 - Creates software product
 - Does Q/A of that product
 - “You build it, you run it”: supports product
- Teams generally practice Agile processes
 - Scrum is popular
- Automation using tools
 - Help to enforce best practices



<https://secure.flickr.com/photos/lox/9408028555>

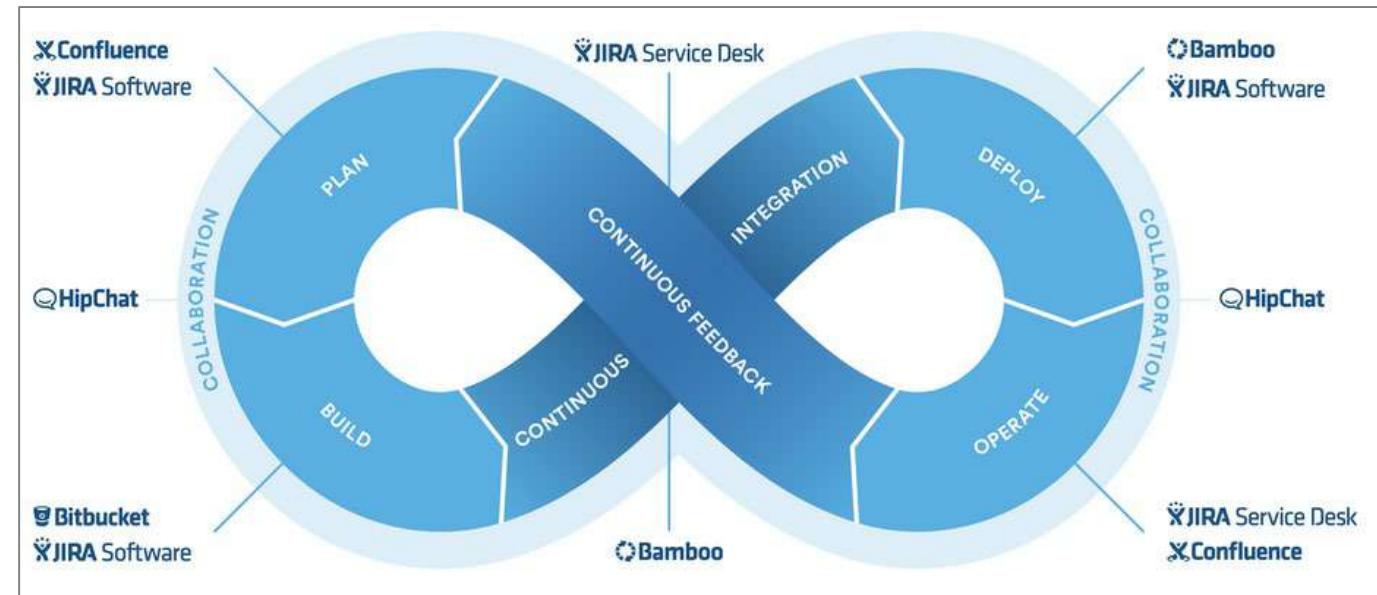


DevOps tools from Atlassian

- Atlassian's DevOps tools are popular with individuals, startups and enterprises
- For more information: <https://www.atlassian.com/>

PLAN, TRACK, & SUPPORT	CODE, BUILD, & SHIP	COLLABORATE & CHAT
Jira Software Plan, track, and release world-class software with the #1 software development tool used by agile teams.	Bitbucket Collaborate on code and manage your Git repositories to build and ship software, as a team.	Confluence Spend more time getting things done. Organize your work, create documents, and discuss everything in one place.
Learn more →	Learn more →	Learn more →
Jira Service Desk Give your customers an easy way to ask for help and your agents a fast way to resolve incidents.	Sourcetree Harness the full power of Git and Mercurial in a beautifully simple application.	Trello Collaborate and get more done. Trello boards enable your team to organize projects in a fun, flexible, and visual way.
Learn more →	Learn more →	Learn more →
Statuspage Incidents happen. Keep your users informed and ditch the flood of support emails during downtime.	Bamboo Continuous integration, deployment, and release management.	Stride The complete communication solution that empowers teams to talk less and do more. Group messaging, video meetings, and collaboration tools.
Learn more →	Learn more →	Learn more →

Source: <https://www.atlassian.com>



Source: <https://www.atlassian.com/blog/devops/how-to-choose-devops-tools>



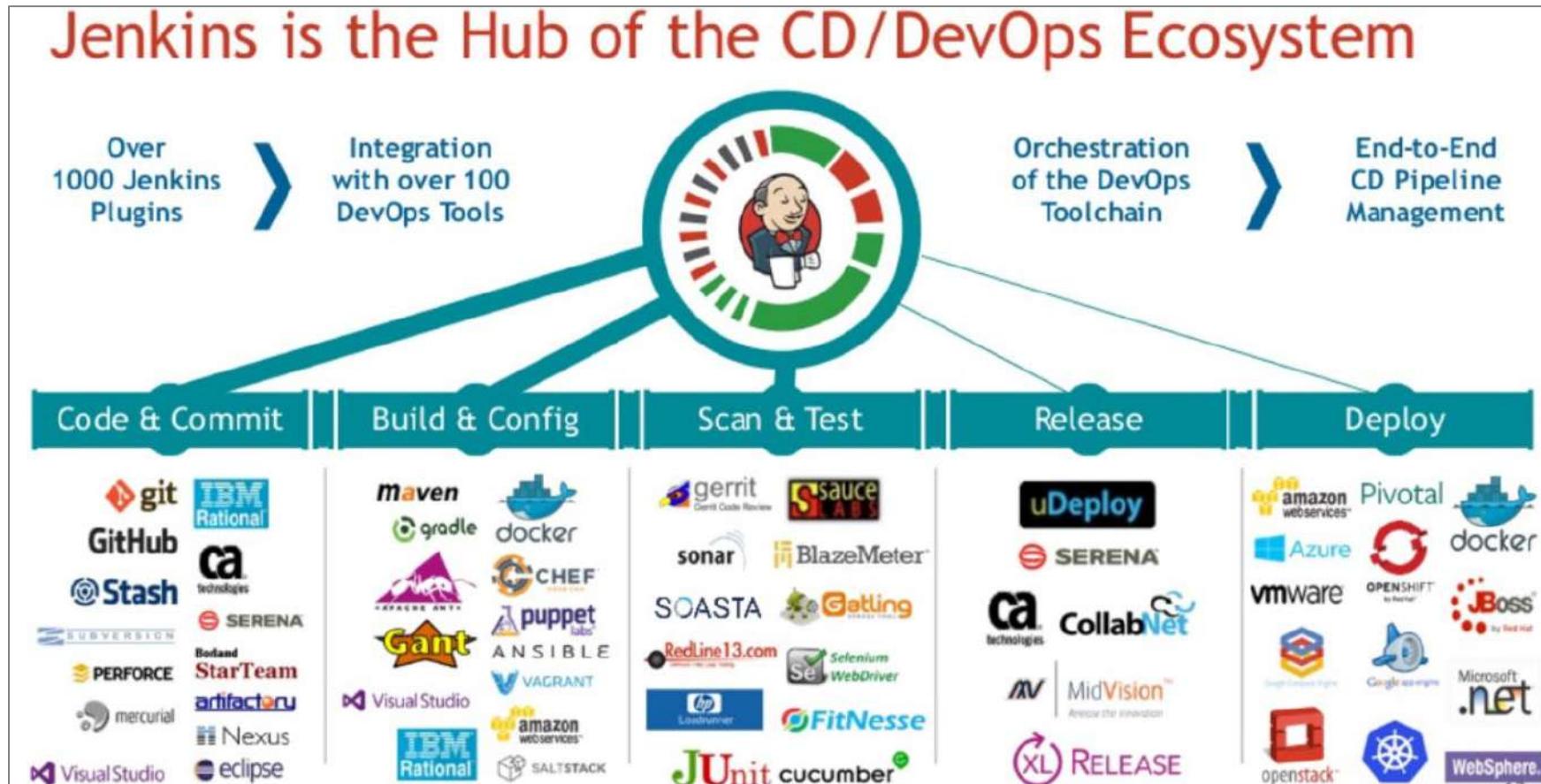
Lots of DevOps tools!

- DevOps is firstly about culture and collaboration
 - However, tools play an important role in automating routine and repeatable activities
 - Improves efficiency, consistency, quality and visibility

Source: <https://xebialabs.com/periodic-table-of-devops-tools/>



Jenkins - a popular open source CI CD automation tool



- An open source automation server
- Automates CI CD process
- Supports many version control tools
- Can execute Apache Ant and Apache Maven projects
- Has plug-ins to integrate with many popular tools
- Can deploy to many platforms – traditional and cloud

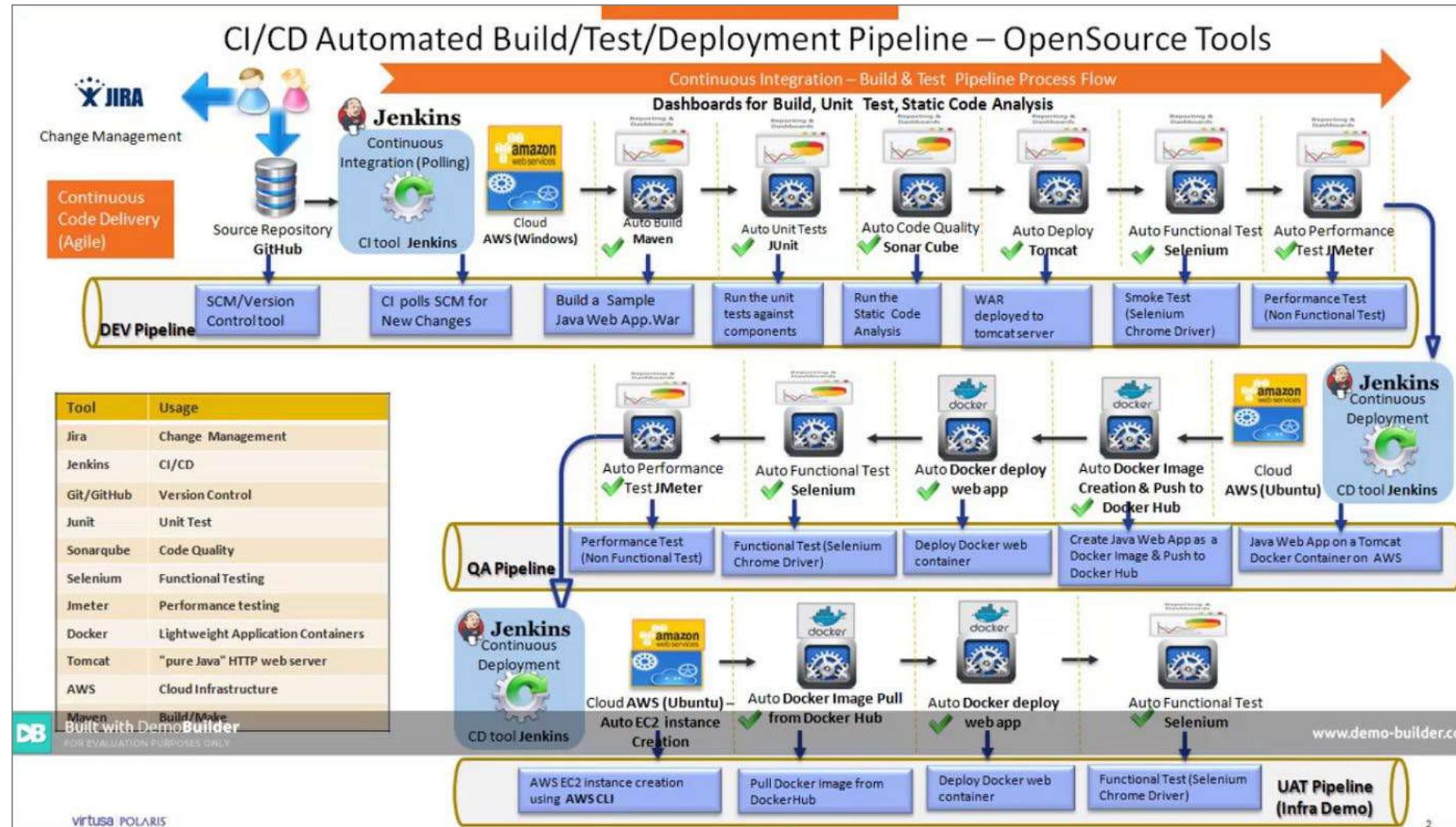
Source:
[https://en.wikipedia.org/wiki/Jenkins_\(software\)](https://en.wikipedia.org/wiki/Jenkins_(software))

Source: <https://haritibcoblog.com/2017/03/03/jenkins-overview/>



Many open source DevOps tools also!

Source: https://i.ytimg.com/vi/L5oesoeZ_iY/maxresdefault.jpg





Assumptions

- Basic understanding of the SDLC lifecycle.
- Basic understanding of the source code management.
- Ability to install tools as part of the course.
- Ability to configure tools as part of the course.
- Basic understanding of Java programming language and build tools like Maven.
- Ability to hands on development in Java using Spring Boot.
- Please note that though the focus here is using Java programming the tool chain can be applied to any technology.



DevOps Toolchain

- DevOps is a set of tool chain that will help organizations achieve their goals. The tools fit into several categories that are:
 - **Code** - SCM, workflow, development, review and merge.
 - **Build** - Continuous integration and builds.
 - **Test** - Continuous testing that provide feedback.
 - **Package** - Packaging code and release to artifact repository.
 - **Release** - Release, promotion and change management.
 - **Configure** - Automated Infrastructure and configuration management.
 - **Monitor** - Application performance and log monitoring

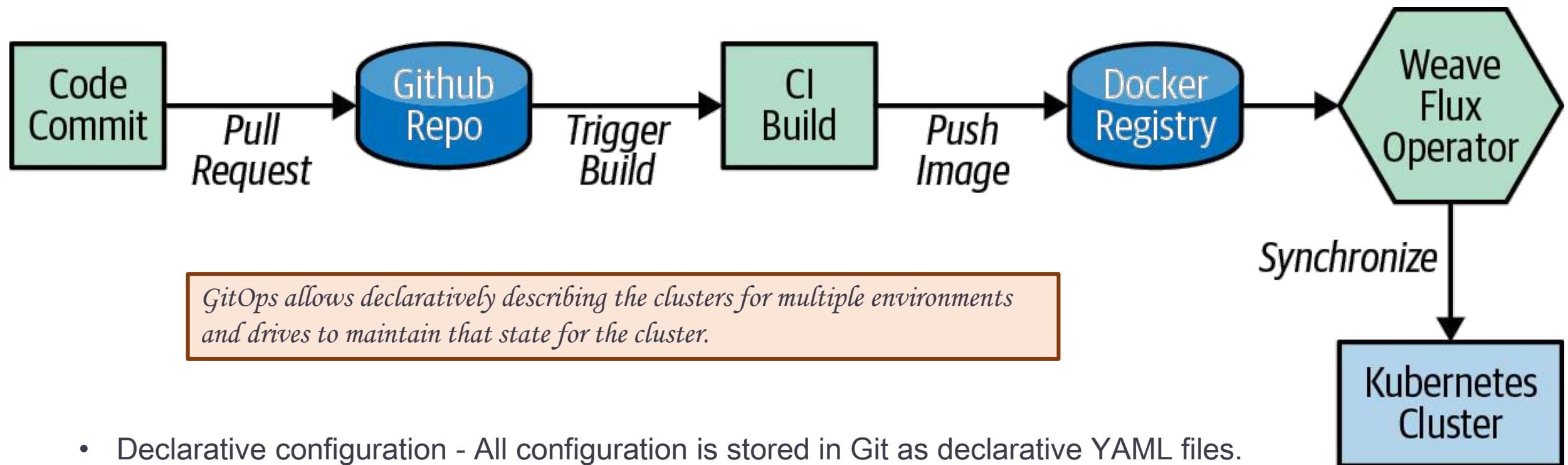


What is GitOps?

- GitOps is a methodology and practice that uses Git repositories as a single source of truth to deliver infrastructure as code.
- The three main pillars of GitOps are:
 - Git is the single source of truth
 - Treat everything as code
 - Operations are performed through Git workflows
- GitOps Principles
 - *Declarative*: A system managed by GitOps must have its desired state expressed declaratively.
 - *Versioned and immutable*: The desired state is stored in a way that enforces immutability and versioning and retains a complete version history.
 - *Pulled automatically*: Software agents automatically pull the desired state declarations from the source.
 - *Continuously reconciled*: Software agents continuously observe the actual system state and attempt to apply the desired state.



GitOps Workflow



- Declarative configuration - All configuration is stored in Git as declarative YAML files.
- Versioned Controlled - Git repository supports immutability and version history.
- Continuous Reconciliation - The cluster state is continuously reconciled with the state defined in Git.
- Security - All changes are made to the git repository and the GitOps agent can automatically reconcile any changes made directly to a Kubernetes resource.

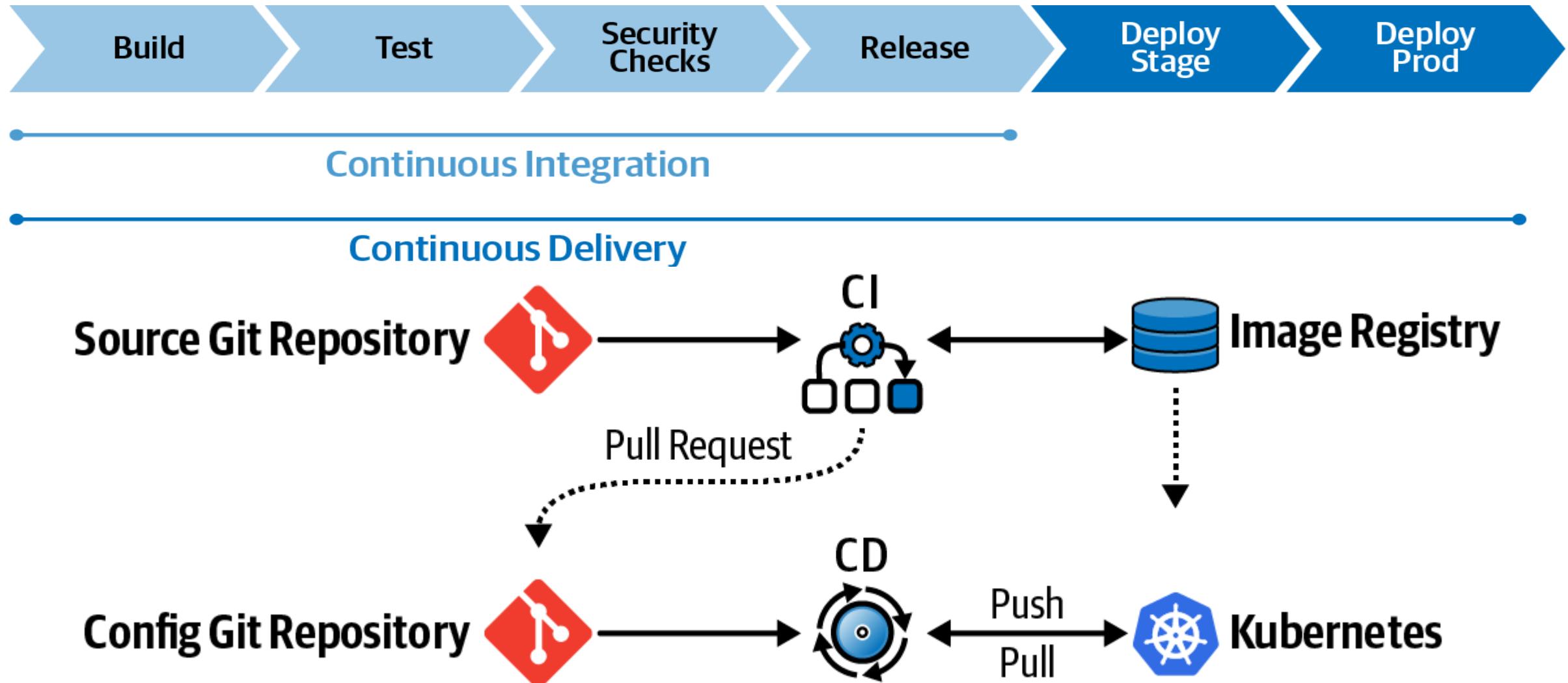


Why GitOps?

- Every change throughout the application lifecycle is traced in the Git repository and is auditable.
- Standard workflow
 - Use familiar tools and Git workflows from application development teams
- Enhanced security
 - Review changes beforehand, detect configuration drifts and take action
- Visibility and audit
 - Capture and trace any change to clusters through Git history
- Multi-cluster consistency
 - Reliably and consistently configure multiple environments and multiple Kubernetes clusters and deployment

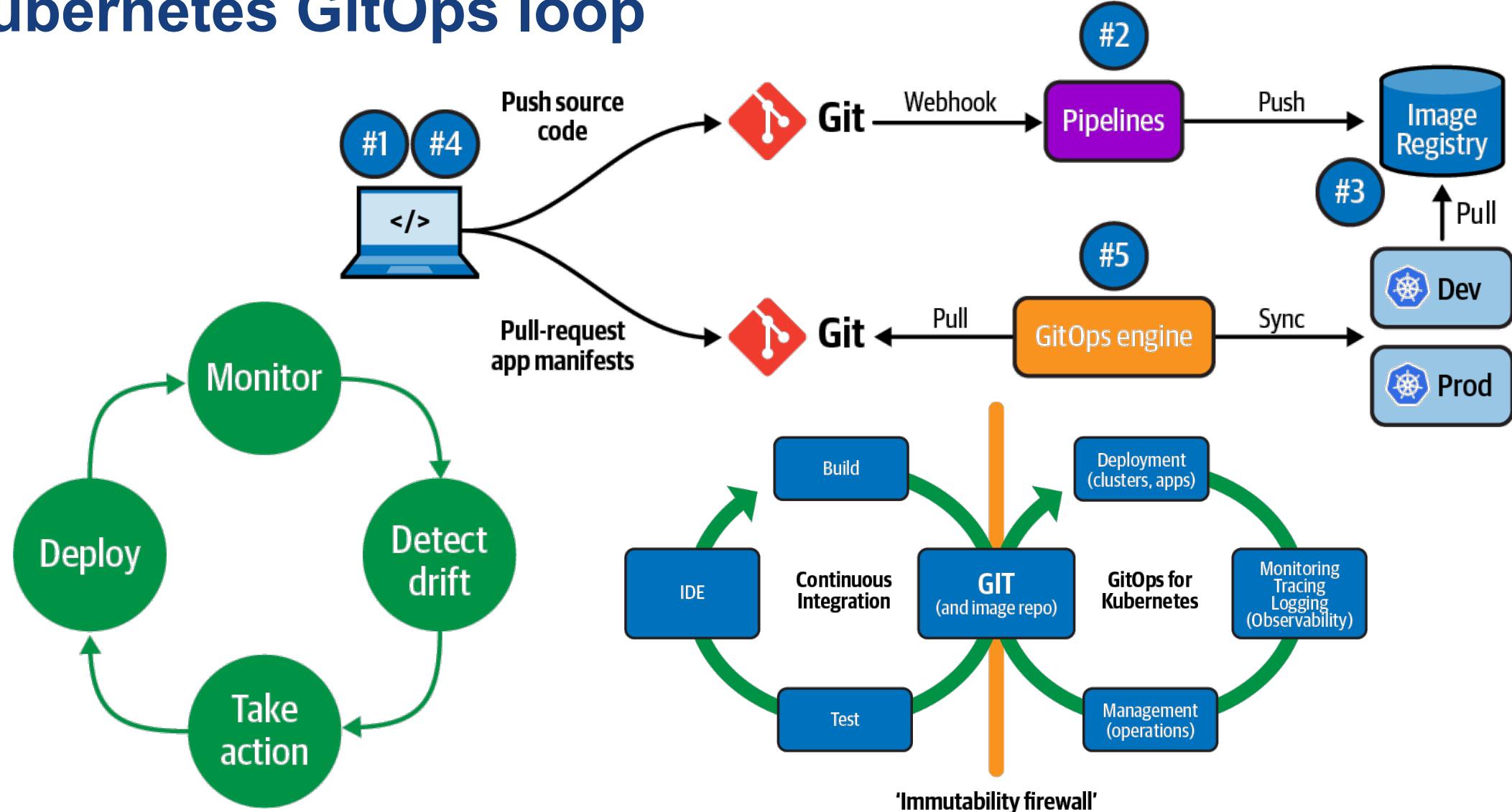


Application deployment model



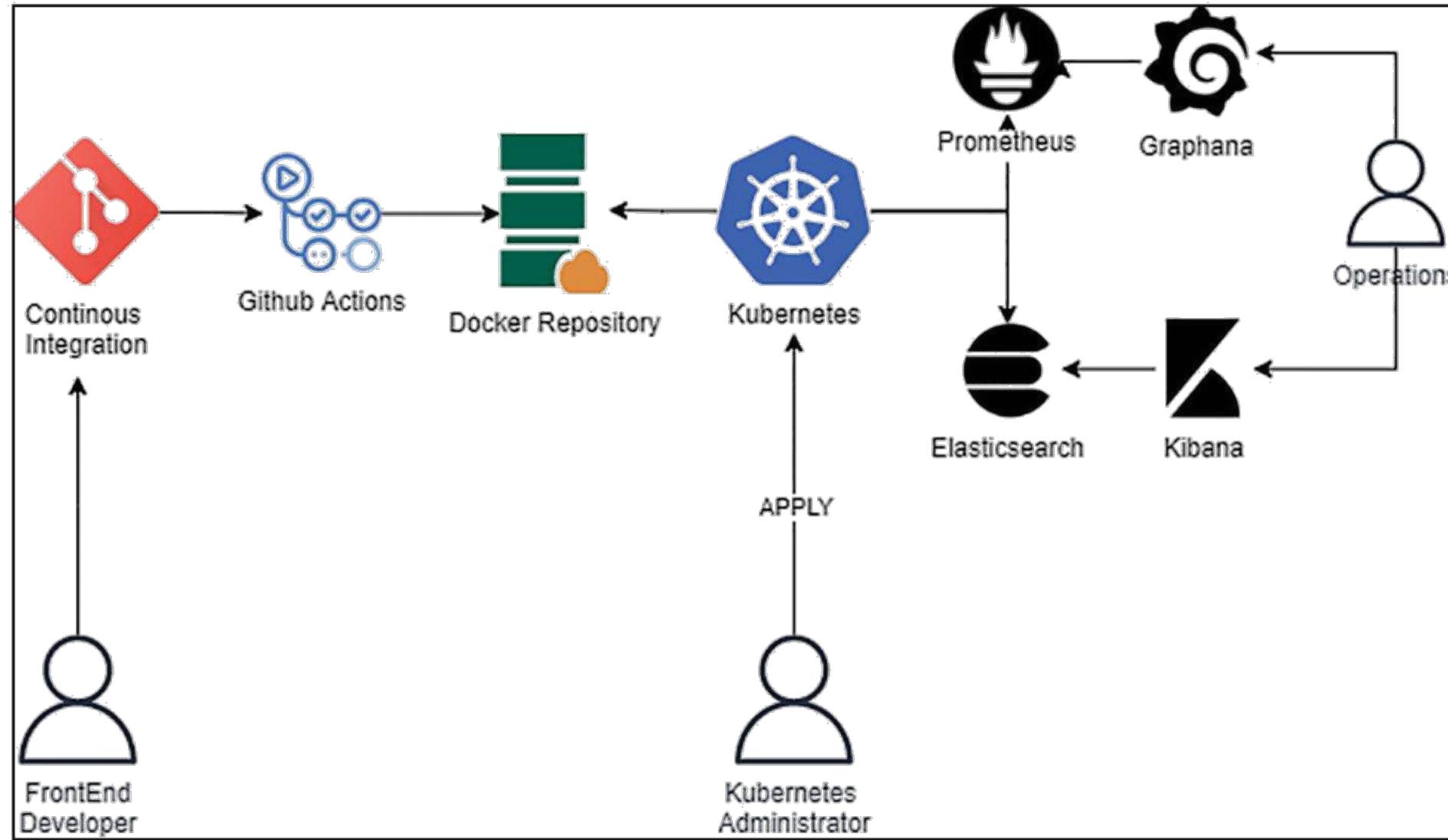


Kubernetes GitOps loop



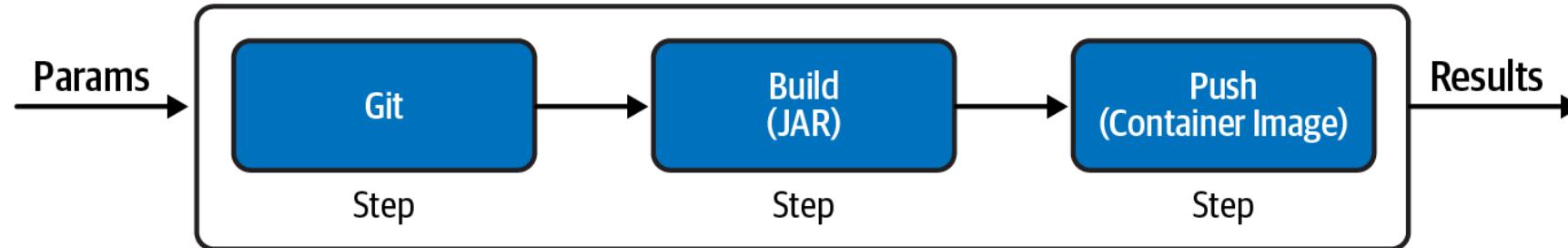


Continuous monitoring using Prometheus, Grafana, and ELK Stack

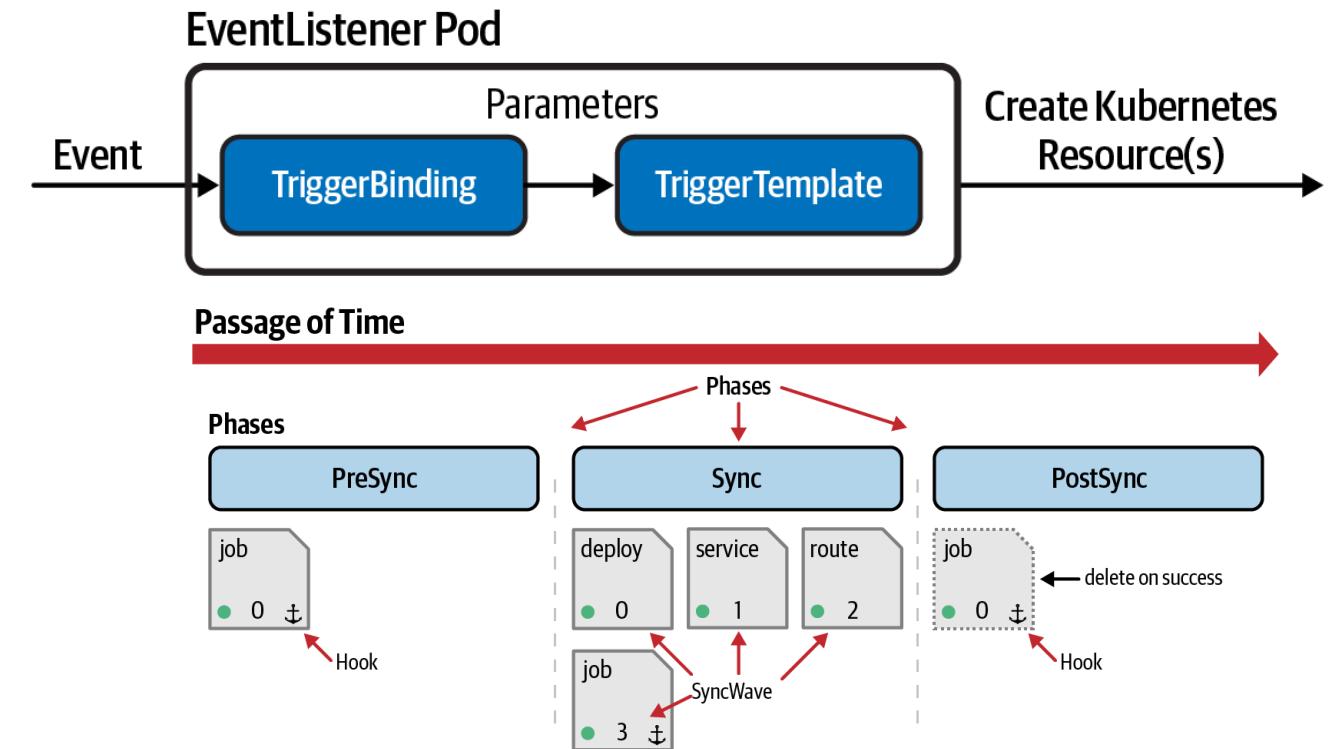
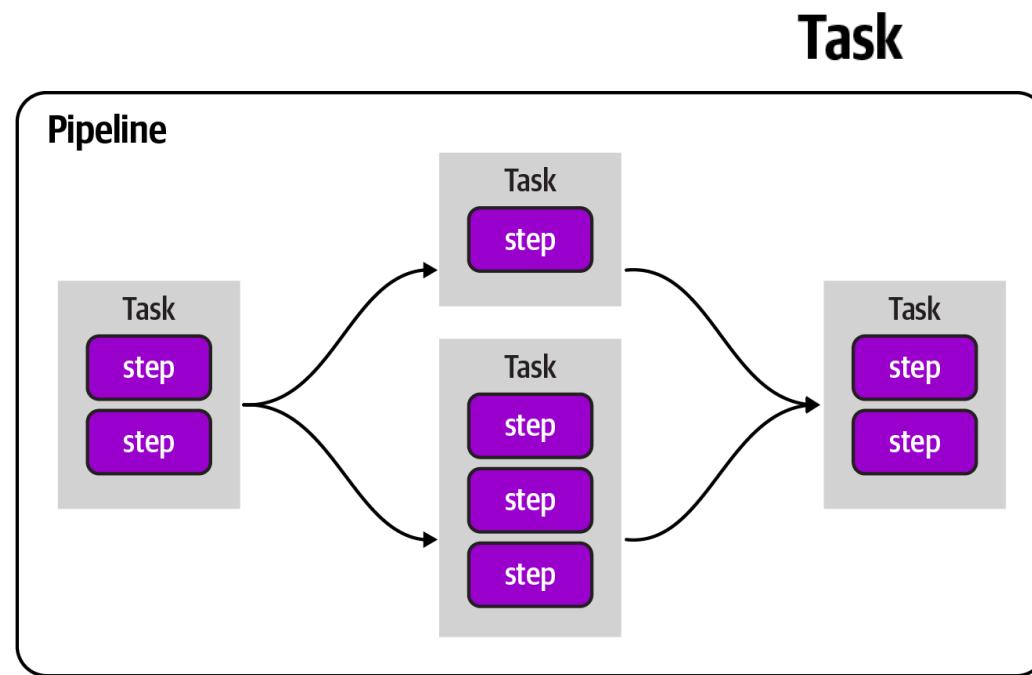




GitOps Summarised



- Task
- Pipeline
- Event
- Phases
- Sync





GitOps Tooling

▪ Flux

- Flux is a Kubernetes operator that watches Git repository for changes and automatically applies those changes to cluster.
- WeaveWorks also provides a hosted version of Flux.

▪ ArgoCD

- Argo CD is an open-source GitOps continuous delivery tool. It monitors cluster and declaratively-defined infrastructure stored in a Git repository and resolves differences.

▪ Codefresh

- Codefresh is a CI/CD platform that can be used to implement GitOps in cluster.
- Codefresh provides a hosted platform that provides ArgoCD as a service.

▪ Harness

- Harness is a CI/CD platform that can be used to implement GitOps in cluster.
- Harness is geared towards enterprise customers and provides a full suite of Continuous Delivery features.



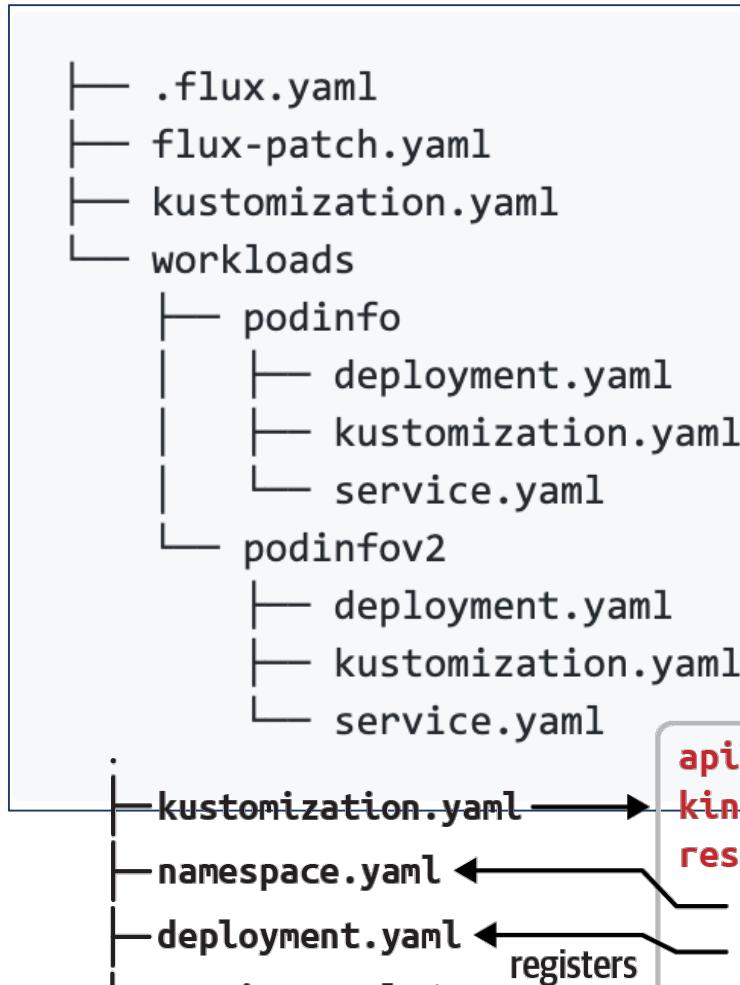


GitOps Best Practices

- Start off with a small application and then scale your efforts to manage everything with a GitOps model. This will allow you to build confidence in your GitOps implementation.
- Evaluate tools that fit your requirements or start with proven OSS tools like Flux or ArgoCD.
- Avoid using branches for your repository layout, as this is probably the most complex and error-prone repository layout.
- Start with a folder per environment, as this provides flexibility and allows you to use tools like Kustomize or Helm for templating.
- Utilize Sealed Secrets or an External secrets provider to manage secrets in your cluster.
- Remember GitOps is a process and not a tool and your existing tool set may fit your needs.



Flux and Kustomize



```

cat <<EOF >./kustomization.yaml
resources:
  - deployment.yaml
configMapGenerator:
  - name: example-configmap-1
    files: - application.properties
EOF

```

```

# Create a application.properties file
cat <<EOF >application.properties
FOO=Bar
EOF

```

```

cat <<EOF >deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  labels: app: my-app
spec:
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
  spec:
    containers:
      - name: app
        image: my-app
        volumeMounts:
          - name: config
            mountPath: /config
    volumes:
      - name: config
        configMap:
          name: example-configmap-1
EOF

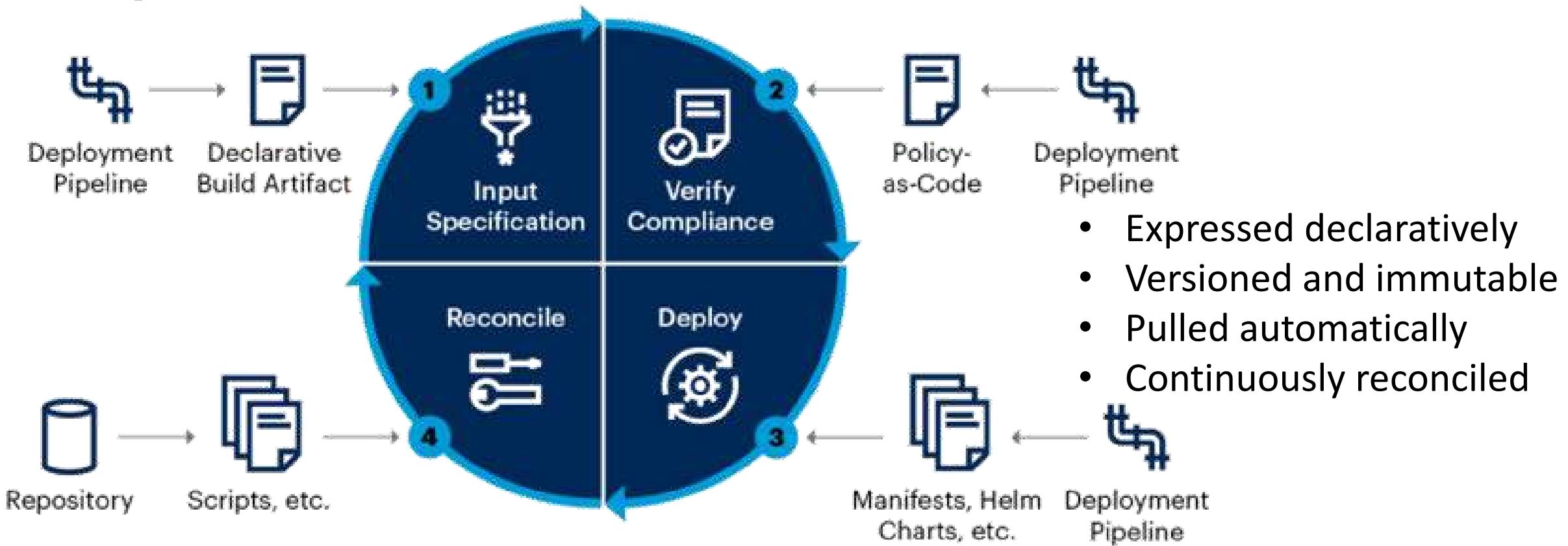
```



SUMMARY & REFERENCES



GitOps Workflow



Source: Gartner
781061_C

Gartner



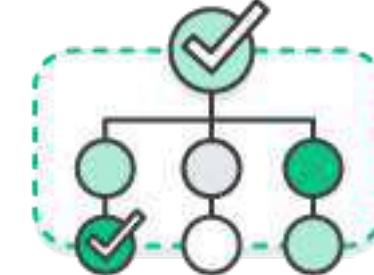
Summary: DevOps in the cloud - easily get benefits



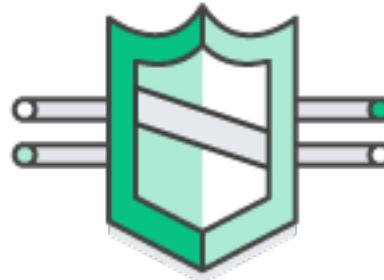
Improved Collaboration



Rapid Delivery



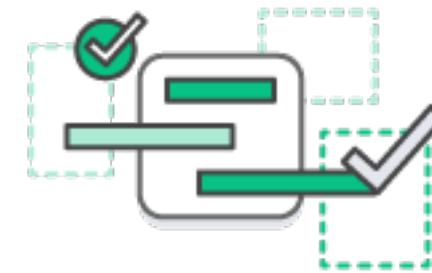
Reliability



Security



Scale



Speed



Resource and Tools Prerequisite

Basic:

- GitHub Account:

<https://github.com/join>

- Install GIT:

<https://www.git-scm.com/downloads>

- Install Atlassian SourceTree:

<https://www.sourcetreeapp.com>

Optional:

- Install GIT FLOW for executing via Command Line

<https://github.com/nvie/gitflow/wiki/Installation>

- Collection of DevOps related tools

• <http://www.devopsbookmarks.com>

- DevOps site for AWS

• <https://aws.amazon.com/devops/>

- Download and Install STS: <https://spring.io/tools/sts/all>
- Download Editor like ATOM and Install: <https://atom.io>
- Download Maven: <http://maven.apache.org>
- Download Tomcat: <https://tomcat.apache.org/download-80.cgi>
- Download Artifactory: <https://www.jfrog.com/open-source/>
- Download Jenkins: <https://jenkins.io/download/>
- Download ElasticSearch: <https://www.elastic.co/downloads/elasticsearch>
- Download Logstash: <https://www.elastic.co/downloads/logstash>
- Download Kibana: <https://www.elastic.co/downloads/kibana>



AWS Global Infrastructure



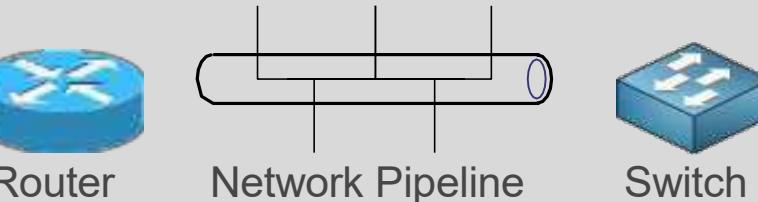


AWS Core Infrastructure and Services

Traditional Infrastructure



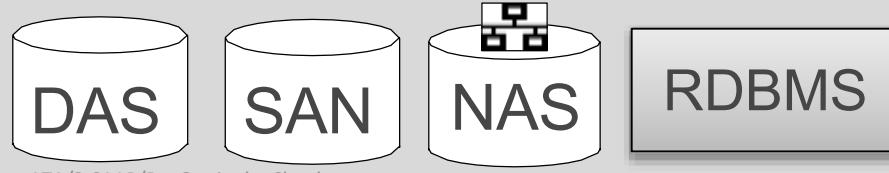
Security



Networking

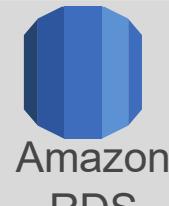
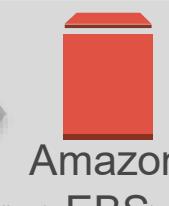


Servers



Storage and Database

Amazon Web Services





DevOps with AWS

DevOps and AWS
Tooling and infrastructure resources for DevOps practitioners
Get Started with AWS

What is DevOps? DevOps Blog Partner Solutions Resources

AWS provides a set of flexible services designed to enable companies to more rapidly and reliably build and deliver products using AWS and DevOps practices. These services simplify provisioning and managing infrastructure, deploying application code, automating software release processes, and monitoring your application and infrastructure performance.

DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market. Learn more about DevOps »

Why AWS for DevOps?

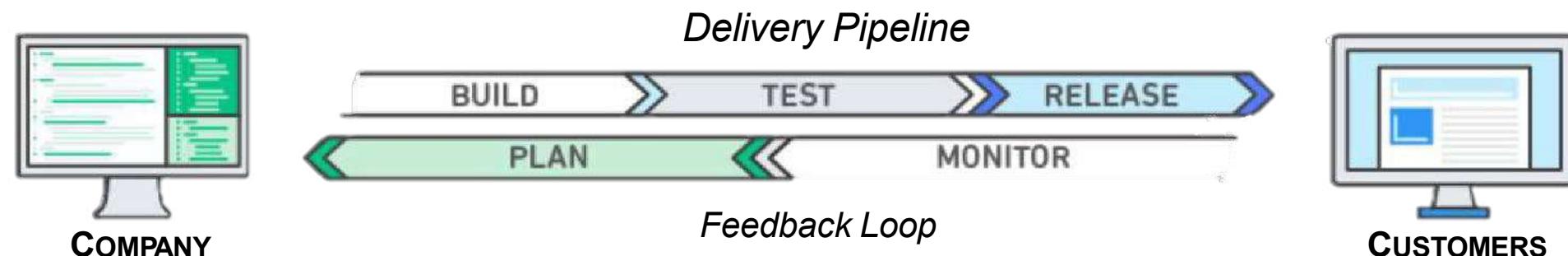
- Get Started Fast**: Each AWS service is ready to use if you have an AWS account. There is no setup required or software to install.
- Fully Managed Services**: These services can help you take advantage of AWS resources quicker. You can worry less about setting up, installing, and operating infrastructure on your own. This lets you focus on your core product.
- Built for Scale**: You can manage a single instance or scale to thousands using AWS services. These services help you make the most of flexible compute resources by simplifying provisioning, configuration, and scaling.
- Programmable**: You have the option to use each service via the AWS Command Line Interface or through APIs and SDKs. You can write, model, and provision AWS resources and your entire AWS infrastructure using declarative AWS CloudFormation templates.
- Automation**: AWS helps you use automation so you can build faster and more efficiently. Using AWS services, you can automate manual tasks or processes such as deployments, development & test workflows, container management, and configuration management.
- Secure**: Use AWS Identity and Access Management (IAM) to set user permissions and policies. This gives you granular control over who can access your resources and how they access these resources.
- Large Partner Ecosystem**: AWS supports a large ecosystem of partners which integrate with and extend AWS services. Use your preferred third-party and open-source tools with AWS to build an end-to-end solution. Visit here to learn more about our DevOps Partner Solutions.
- Pay-As-You-Go**: With AWS purchase services as you need them and only for the period when you plan to use them. AWS pricing has no upfront fees, termination penalties, or long term contracts. The AWS Free Tier helps you get started with AWS. Visit the pricing pages of each service to learn more.

- AWS provides a rich set of services for DevOps
- Home page: <https://aws.amazon.com/devops/>
- Getting started guide:
<http://docs.aws.amazon.com/devops/latest/gsg/welcome.html>
- Tutorials: <https://aws.amazon.com/getting-started/use-cases/devops/>
- Whitepapers: <https://aws.amazon.com/whitepapers/>
 - Introduction to DevOps on AWS [PDF](#)
 - Development and Test on AWS [PDF](#)
 - Practicing Continuous Integration and Continuous Delivery on AWS [PDF](#)
 - Jenkins on AWS [PDF](#)
 - Infrastructure as Code [PDF](#)
 - Blue/Green Deployments on AWS [PDF](#)
 - Microservices on AWS [PDF](#)



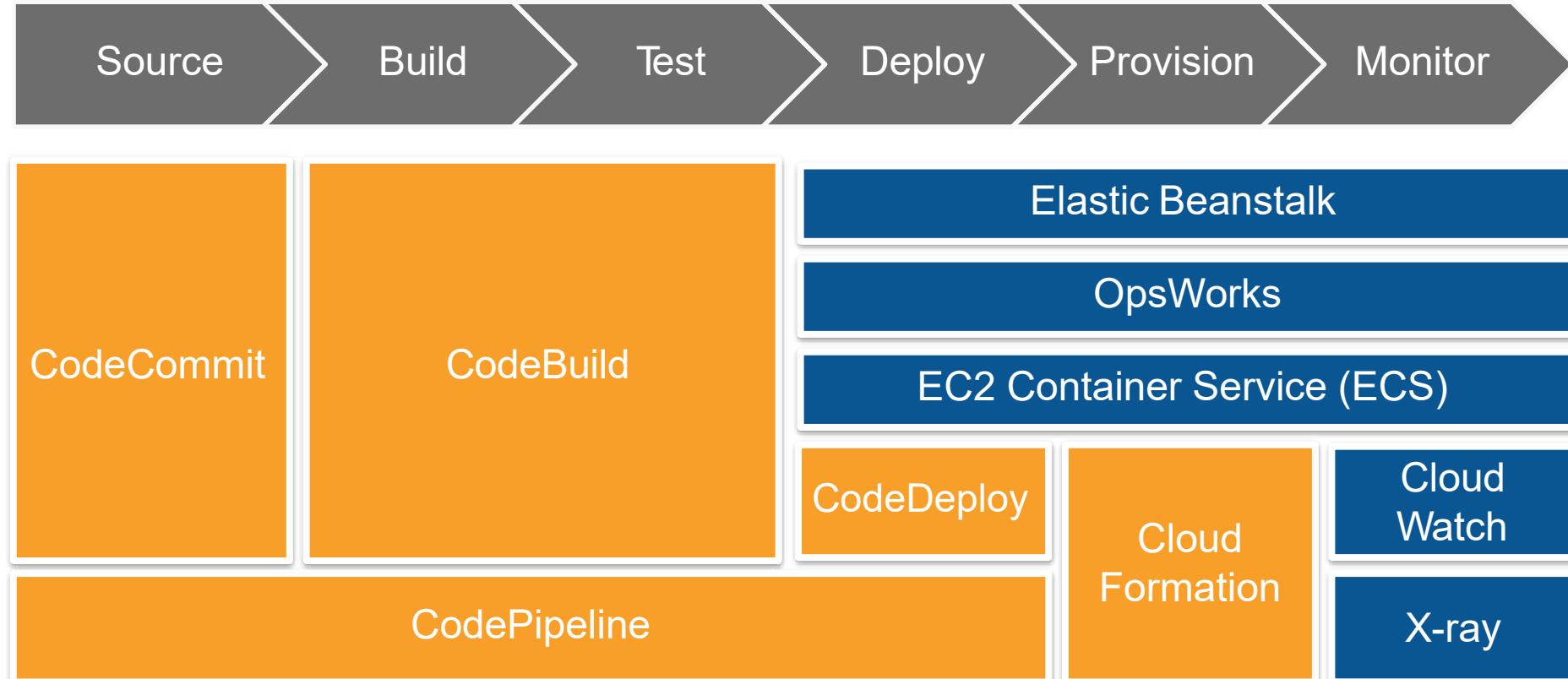
DevOps according to AWS

- Unification of software development and operations
- Migration of Agile continuous development into continuous integration and continuous delivery
- DevOps model:
 - **No silos** – emphasis on communication, collaboration, and cohesion between disciplines
 - Best practices for change, configuration, and deployment automation
 - Deliver apps/services at faster pace
 - High speed product updates





DevOps tools stack on AWS





DevOps Release Processes: major phases

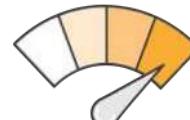
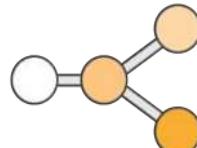
Source

Build

Test

Production

- Check-in source code such as .java files.
- Peer review new code
- Compile code
- Unit tests
- Style checkers
- Code metrics
- Create container images
- Integration tests with other systems
- Load testing
- UI tests
- Penetration testing
- Deployment to production environments



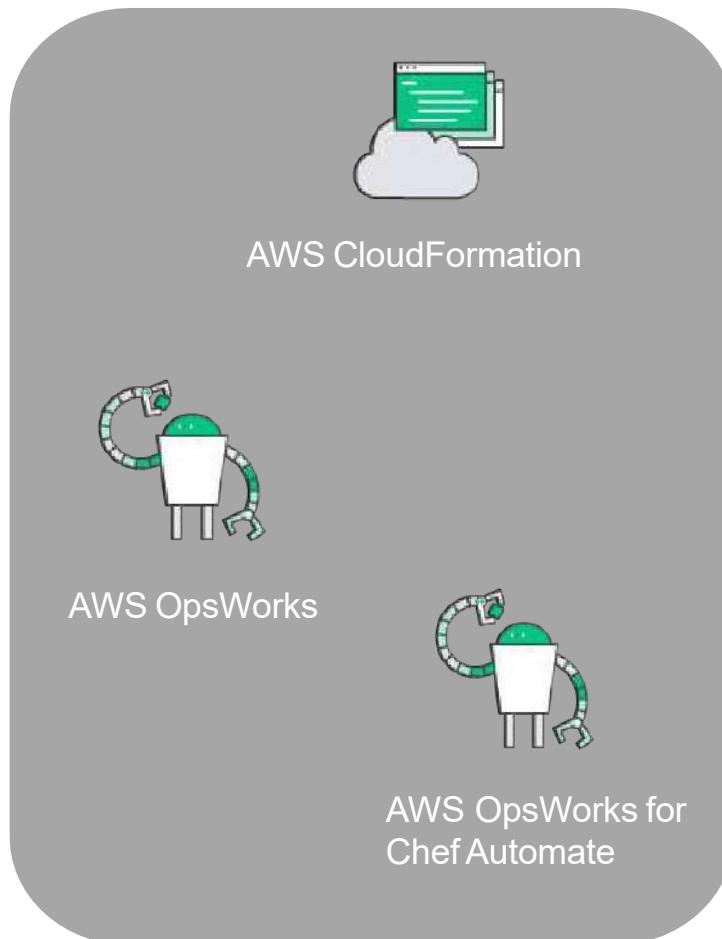


AWS DevOps Portfolio

Software Development and Continuous Delivery



Infrastructure as Code



AWS OpsWorks for
Chef Automate

Monitoring & Logging





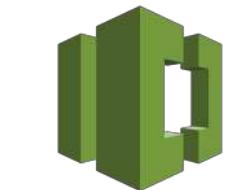
AWS CodePipeline - Integrations

Source

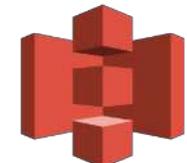
Build

Test

Deploy



AWS CodeCommit



Amazon S3

GitHub



AWS CodeBuild



CloudBees



Jenkins



Solano Labs



BlazeMeter



HPE StormRunner

Runscope



AWS CodeDeploy



AWS
CloudFormation



AWS
OpsWorks



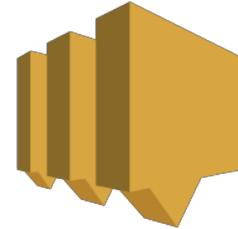
AWS Elastic
Beanstalk

XebiaLabs
Deliver Faster



AWS CodePipeline - integrations (continued)

Approval



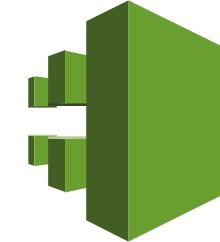
Amazon Simple
Notification Services
(SNS)

Invoke Logic



Amazon
Lambda

Management



AWS CloudTrail

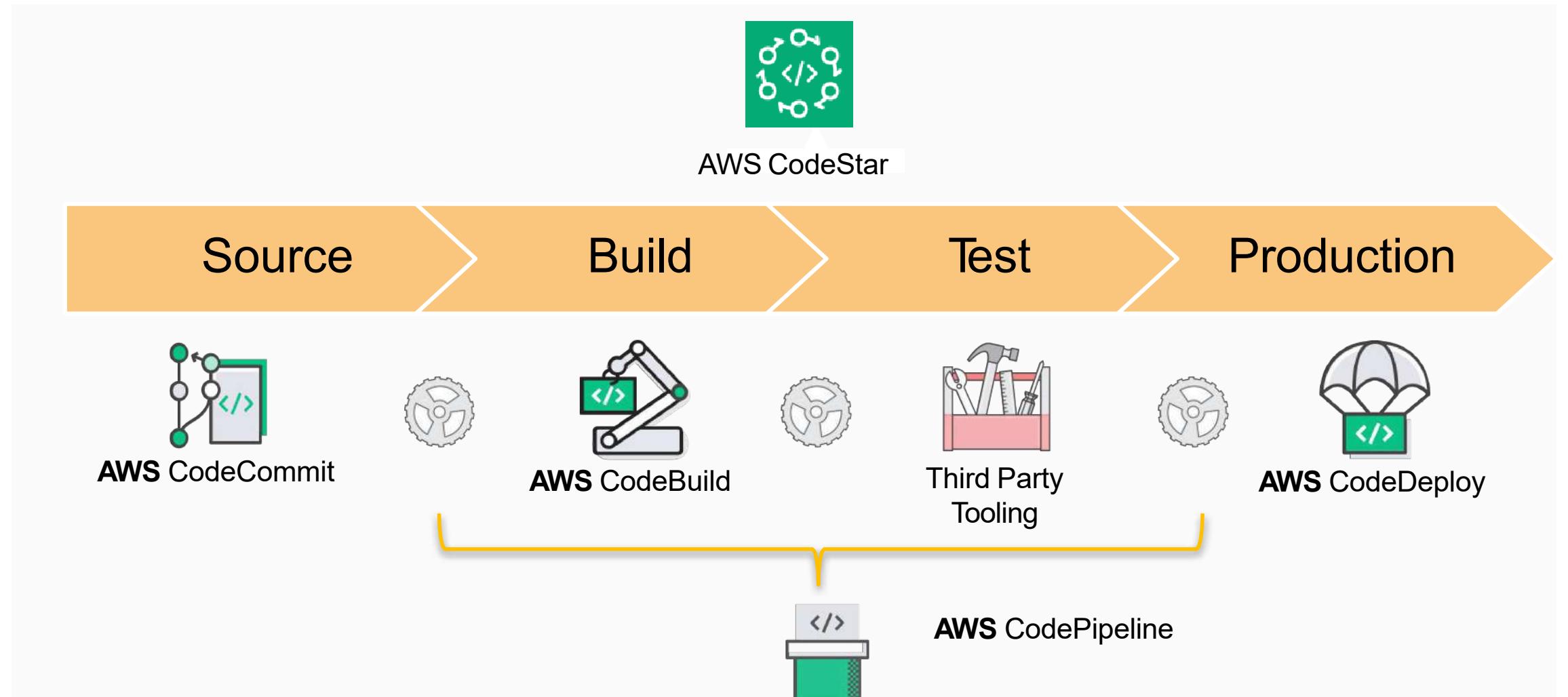


AWS Key
Management Service
(KMS)



AWS CodeStar - Implement AWS DevOps

Software Development and Continuous Integration & Delivery Workflow





Features of AWS CodeStar



- ✓ Project Templates
- ✓ Team Access Management with AWS IAM



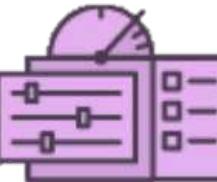
- ✓ Managed Build Service with AWS Code Build



- ✓ Unified Project Dashboard using Amazon CloudWatch monitoring service



- ✓ Issue tracking and project management tool in dashboard via integrated Atlassian JIRA Software



AWS CodeCommit for Secure Hosted Git Repository

Automated App Deployments with AWS CodeDeploy and AWS CloudFormation

- ✓ Integration of AWS CodePipeline for Automated Continuous Delivery Pipeline



AWS CodeStar – Project Templates

- CodeStar project templates support popular languages and IDEs

Programming Languages

Java



JavaScript



Python



Ruby



PHP



IDE/Code Editors

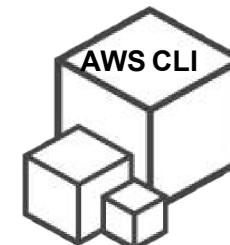
Visual Studio



Eclipse



AWS CLI





Cloud Native Solution Design

ARCHITECTING CLOUD NATIVE APPLICATION

CLOUD NATIVE – VIRTUAL IMAGE VS CONTAINER

Suria R Asai

suria@nus.edu.sg

Institute of Systems Science

National University of Singapore

© 2009-23 NUS. The contents contained in this document may not be reproduced in any form or by any means, without the written permission of ISS, NUS, other than for the purpose for which it has been supplied.

Total Slides 75



Agenda

- Cloud Native Application
- Scaling and Types of Architecture
- Container Vs Virtual Image
- Cloud Native Design Consideration
- Summary



Learning Objectives

- On completion of this module, the participant will
 - Learn about architecting web applications in the cloud.
 - Introduction of the 12-factors for Cloud Native Applications.
 - Design microservices via VM or Container or host.
 - Understand the typical cloud native application design concerns.



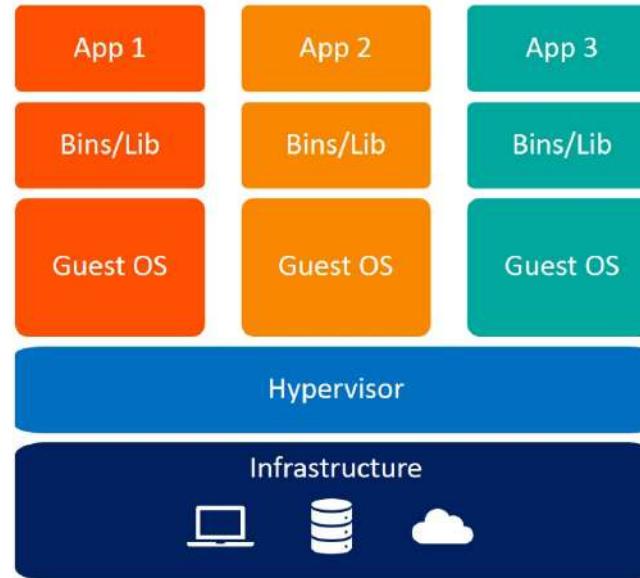
CONTAINERS VS VIRTUAL IMAGES

INFRASTRUCTURE AS A SERVICE PLATFORM AS A SERVICE CHOOSING BETWEEN IAAS AND PAAS

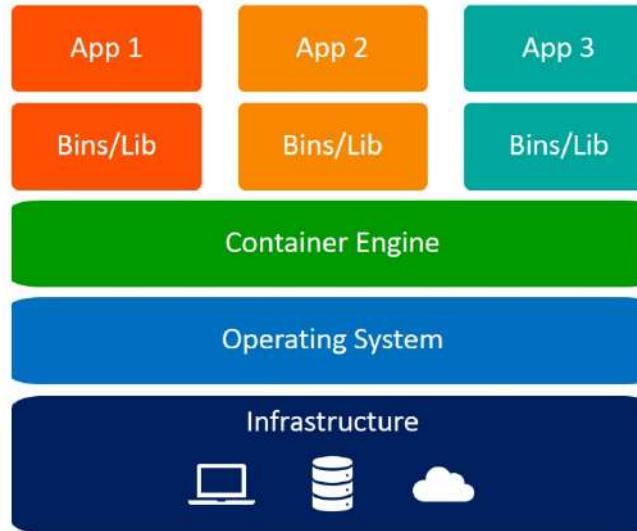


What can be injected?

Recollection Slides



Virtual Machines



Containers

- When using an application on a **virtual machine**, not only the application itself, but also a whole host of other resources are required, including library, OS and other binaries.
- A **container**, only contains the actual application as well as the associated dependencies. The knitted isolation allows multiple containers to access the same kernel resources.

Infrastructure as code (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.



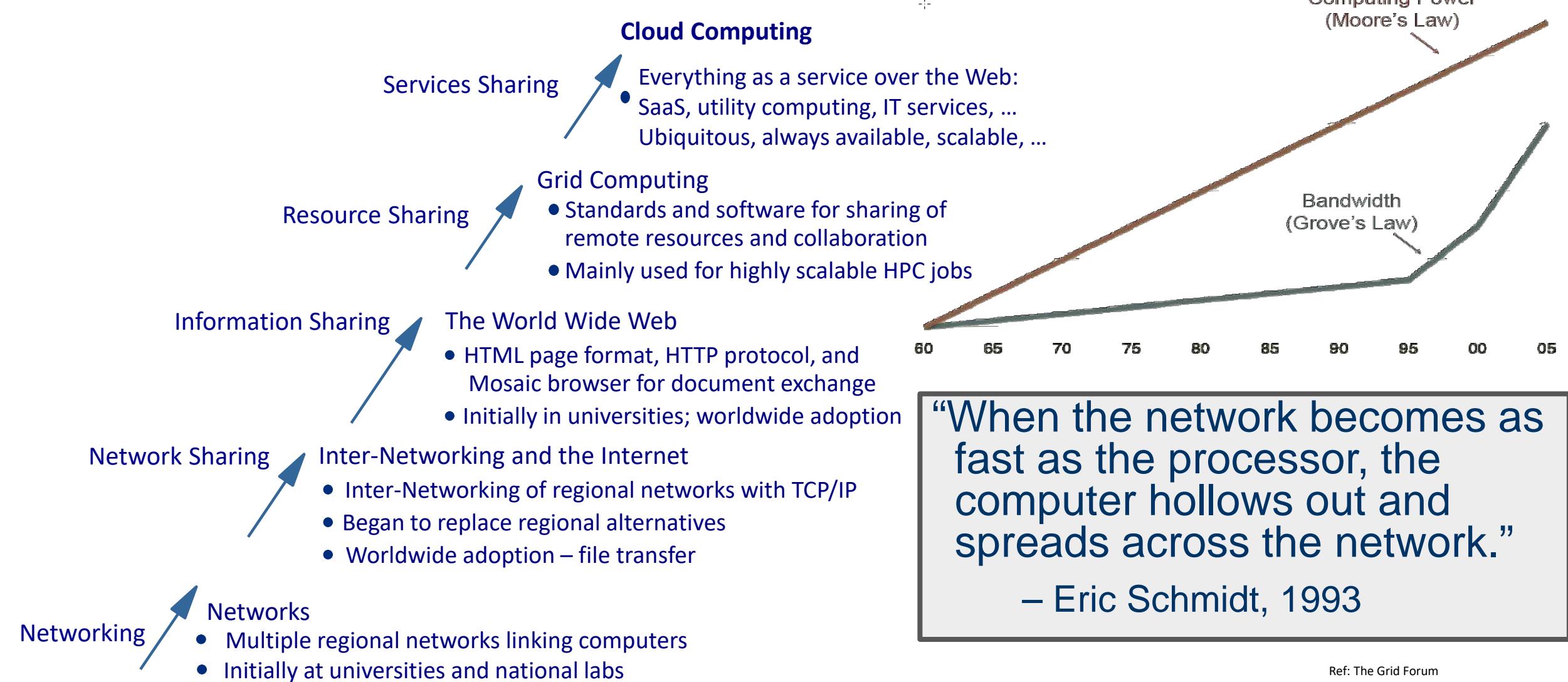
Managed Cloud Services - Definition

“The Managed cloud services are services that offer partial or complete management of a client’s cloud resources or infrastructure.”

- Management responsibilities can include migration, configuration, optimization, security, and maintenance.
- Managed cloud services are designed to enable organizations to maximize benefits from cloud services while minimizing internal time and costs.
- Organizations will contract for services before migration to determine which cloud resources best suit their.
- Managed cloud providers typically offer subscription offerings for a wide range of services.



IaaS – The Evolution of Resource Sharing



"When the network becomes as fast as the processor, the computer hollows out and spreads across the network."

– Eric Schmidt, 1993

Ref: The Grid Forum

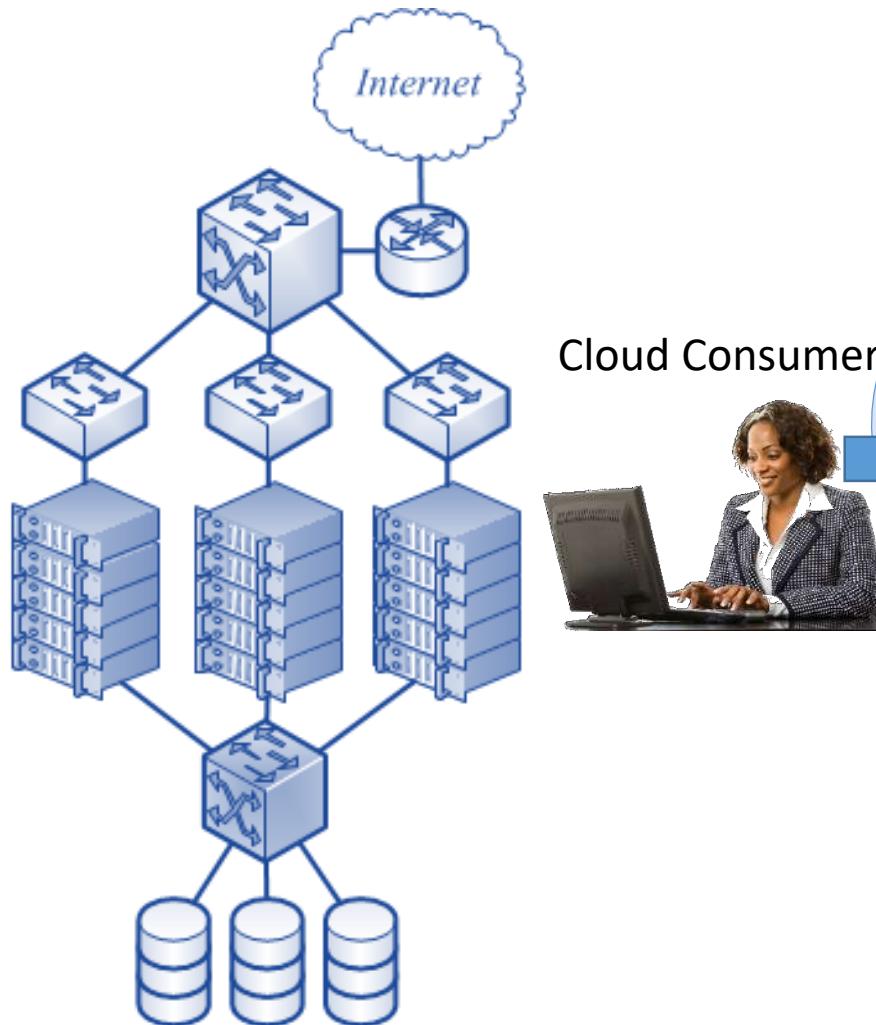


The Need for IaaS

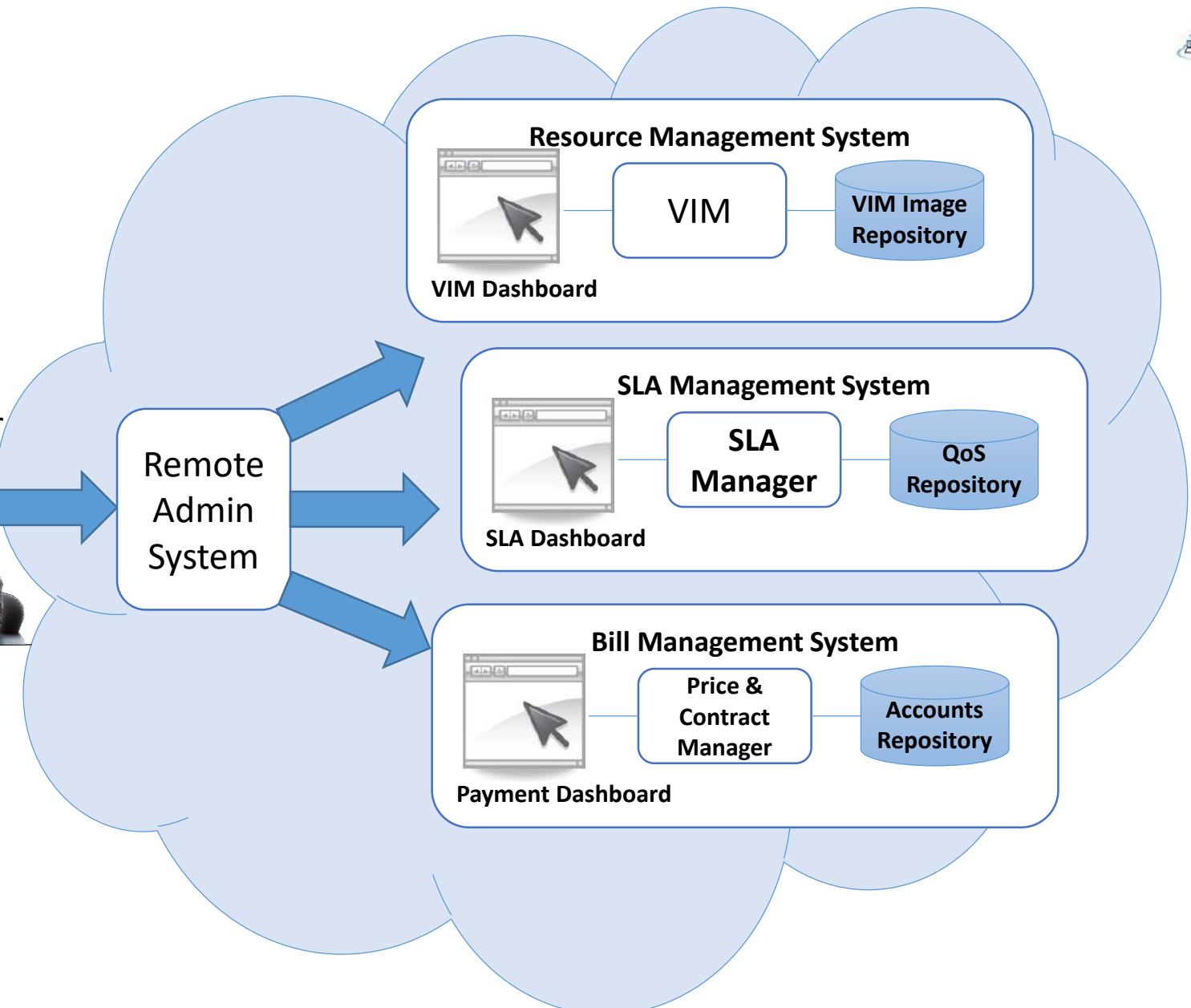
- Optimise IT Efficiency and Reduce Costs
 - Grow and contract efficiently based on needs
- Reduced IT Investment and Expenses
- Ideally Unlimited Capacity
- Intelligent Capacity Planning for Constantly Changing Requirements
- Cost Effective and Latest Disaster Recovery Procedures
- Easy Provisioning for Seasonal Demands
- Business can shift focus to core competencies and innovation
 - Focus on application and services than in infrastructure
 - Provides a competitive edge
- Energy Efficient and lesser resource requirements



IaaS Management



Cloud Consumer





Ready for IAAS cloud applications....

Analytics

- Data mining, text mining or other analytics
- Data warehouses or data marts
- Transactional databases

Business services

- Customer relationship management (CRM)
- Sales force automation
- Enterprise resource planning (ERP) applications
- Industry-specific applications

Collaboration

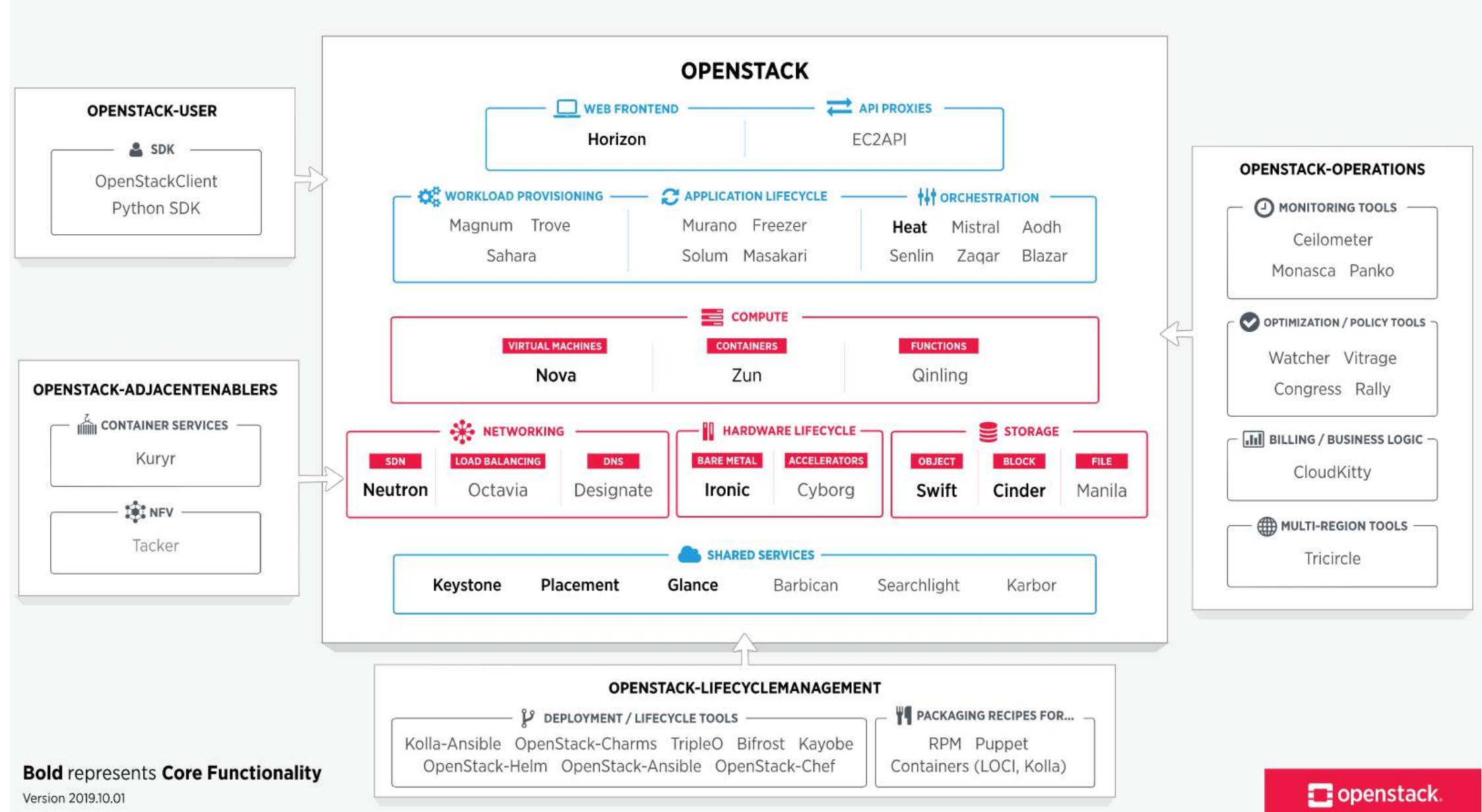
- Audio/video/Web conferencing
- Unified communications
- VoIP infrastructure

Development and test

- Development environment
- Test environment

Infrastructure

- Application servers
- Application streaming
- Business continuity/disaster recovery
- Data archiving
- Data backup
- Data center network capacity
- Security
- Servers
- Storage
- Training infrastructure





Ansible

- **Ansible** is an open-source software provisioning, configuration management, application-deployment and orchestration tool.
 - Can configure both **Unix-like** systems as well as Microsoft **Windows**.
 - Includes its own declarative language to describe system configuration.
 - Written in **Python** and has a fairly minimal learning curve.
 - Simple setup procedure
 - No dependence on any additional software, servers or client daemons.
 - Used orchestration templates (known as **HOT -Heat Orchestration Template**) as a recipe that is written in **YAML (Yet Another Markup Language)**.



Ansible Examples

Create the Ansible playbook for tasks that will launch the instances.

[launch-instance.yml](#)

```
- name: Launch instance on OpenStack
hosts: localhost
gather_facts: false
tasks:
- name: Deploy an instance
os_server:
  state: present
  name: somename
  image: xenial-image
  key_name: demokey
  timeout: 200
  flavor: m1.tiny
  network: private-net
  verify: false
```

```
source openrc ansible-playbook launch-instance.yml
```

```
[WARNING]: provided hosts list is empty, only localhost is available
```

```
PLAY [Launch instance on OpenStack] ****
```

```
TASK [Deploy an instance] ****
changed: [localhost]
```

```
PLAY RECAP ****
localhost : ok=1    changed=1    unreachable=0    failed=0
```

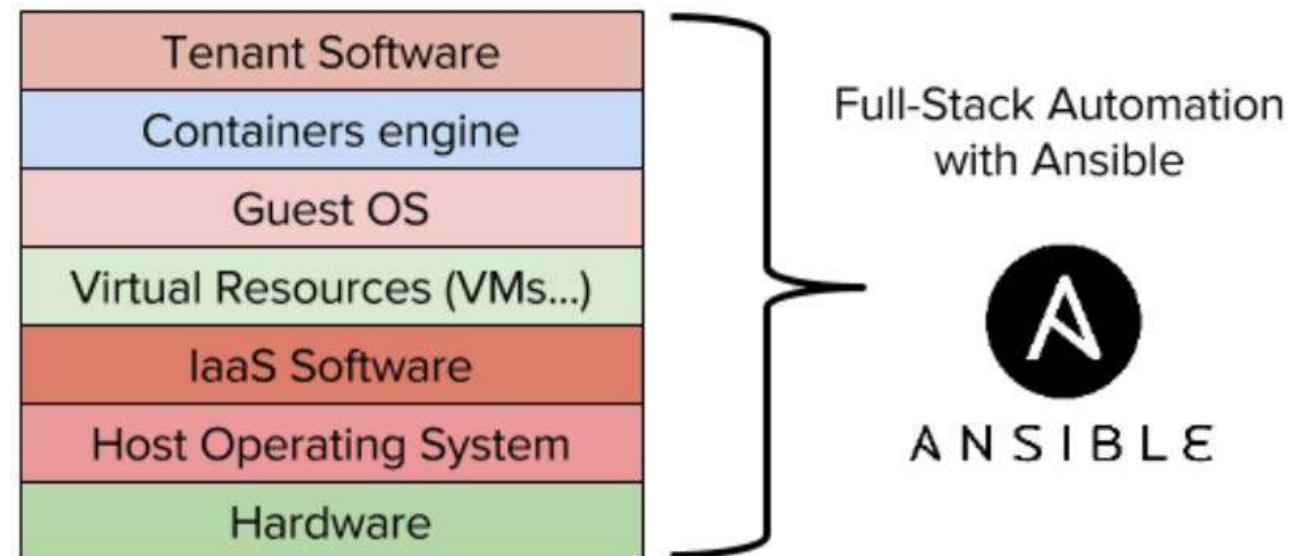
openstack server list

ID	Name	Status	Networks	Image Name
...



Final Open Stack Remarks

- OpenStack is a suite of projects that combine into a software-defined environment to be consumed using cloud friendly tools and techniques.
 - A very popular method for installing OpenStack is the OpenStack-Ansible project (<https://github.com/openstack/openstack-ansible>).





Full Stack Example

full-stack.yml

```
- name: Create OpenStack Cloud Environment
  hosts: localhost
  gather_facts: false
  vars:
    webserver_count: 2

  tasks:
    - name: Download Ubuntu 16.04 Xenial
      get_url:
        url: http://releases.ubuntu.com/16.04/ubuntu-16.04.3-server-amd64.img
        dest: /tmp/ubuntu-16.04.img
    - name: Ensure Ubuntu 16.04 Xenial Image Exists
      os_image:
        name: xenial-image
        container_format: bare
        disk_format: qcow2
        state: present
        filename: /tmp/ubuntu-16.04.img
        verify: false
    . . . Security Groups Sub Net and lot of other configurations
```



PaaS – The Evolution of Compute Abstraction





Platform as a Service (PaaS)



Form Builder



Chat & Video



Report Builder



Visual Workflow



Massively Scalable Data Service



ACID Transactions & Triggers



Social Data Model



Mobile



Self-tuning Query Optimizer



Multi-tenant code execution



Social collaboration framework



REST & SOAP API's



Row Level Security



Identity & Authentication



Real-time Upgrades



Enterprise Search



The need for PaaS

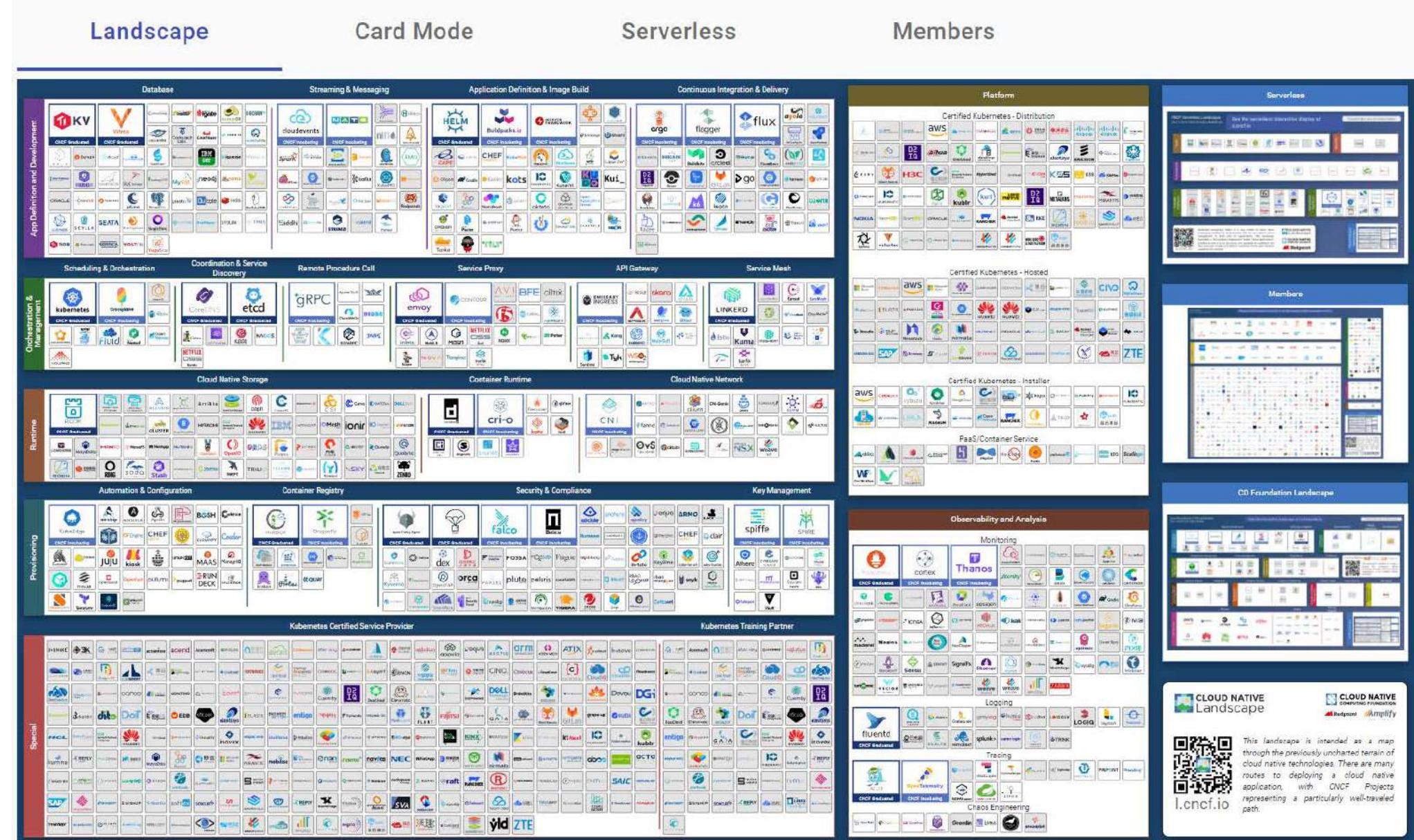
- **Faster Time to Market** – automating many previous manual steps reduces time to market dramatically (months to hours in some cases)
- **Lower Cost** – helping to resource share applications saves infrastructure costs, automation reduces labour (at least 50% reduction)
- **Lower Capital Commitment** – start with small deployments and grow automatically as demand builds (90% reduction)
- **Manage many applications easier** – Application, Tenant, user management, security, load balancing common helps systematize otherwise duplicative functions.
- **More Responsive** – automated deployment means being able to deploy changes faster, automated scaling means responding to demand faster
- **Best Practices** – PaaS incorporates best practices for application management that systematizes and professionalizes the operation of many applications
- **Increase Reuse** – PaaS facilitates reusing services through various kinds of multi-tenancy, load balancing and resource sharing. Reuse facilitates cost reduction as well as faster innovation. Changing a reused service results in improving all those applications using it.
- **Team Collaboration** – PaaS service support developers team collaboration using plethora of tools.



Ready for PaaS cloud applications

- Application Environment
- Mashups of Services
- Configuration Platform
- Scripting Engines
- Hosted Environment
- Application Infrastructure Capabilities
- Databases Data Stores
- Message Queues
- APIs for PaaS Integrations
- Rule Engines
- Mobile Data Hosting Apps
- Analytics Acceleration
- Business Process Automation
- Dynamic pay per use
- Self Service Consumption
- Log Analysis
- Cloud Integration
- Web Server & Application server
- Message queue
- Batch/ETL Jobs
- Continuous Integration and Continuous Deployment
- Application Isolation and Scalability

The Cloud Native Computing Foundation (CNCF) hosts critical components of the global technology infrastructure. CNCF brings together the world's top developers, end users, and vendors and runs the largest open source developer conferences. CNCF is part of the nonprofit Linux Foundation.



Reference: <https://landscape.cncf.io/>



CLOUD NATIVE APPLICATIONS

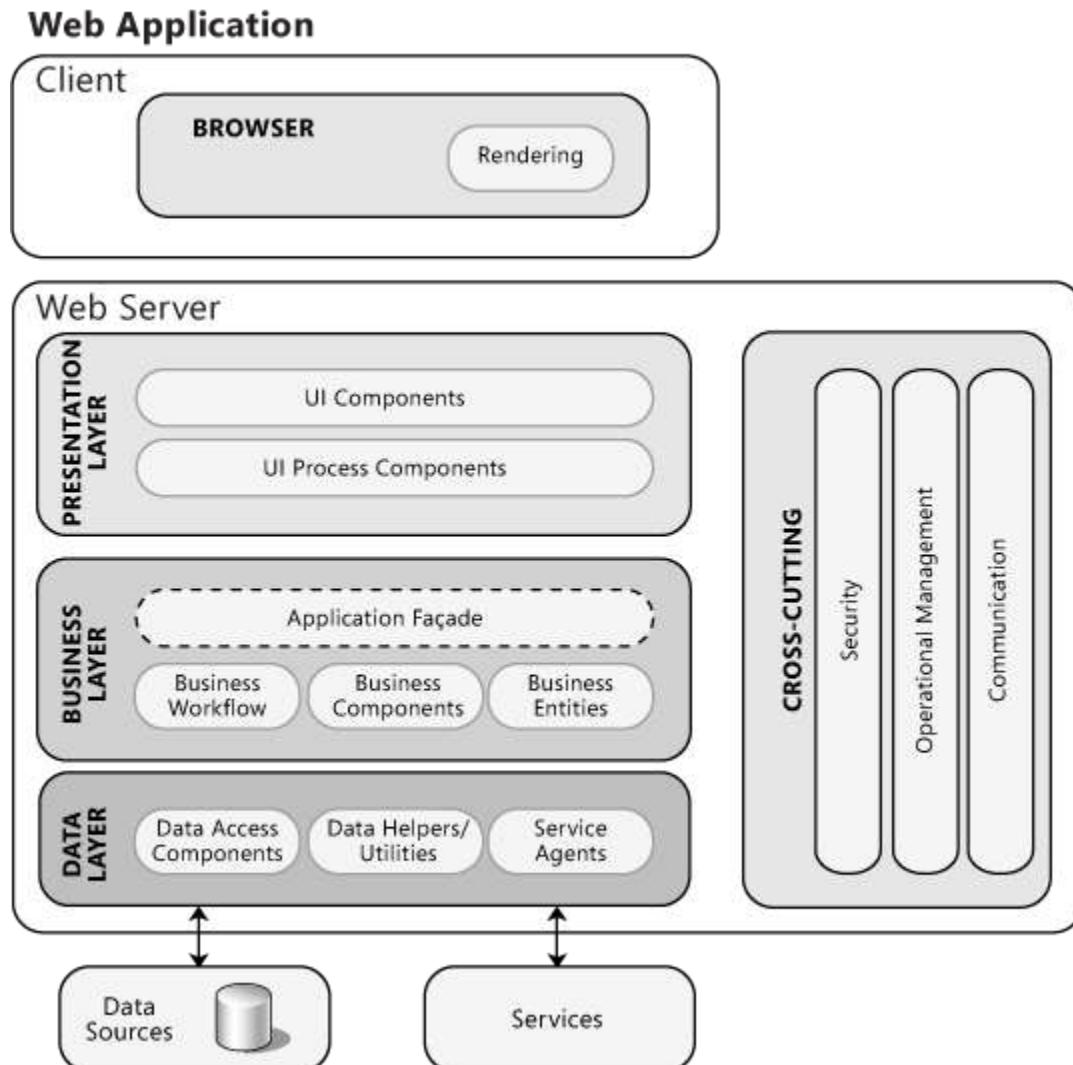
TRADITIONAL WEB APPLICATIONS

CLOUD NATIVE APPLICATION EXAMPLE

12 FACTOR DESIGN PRINCIPLE



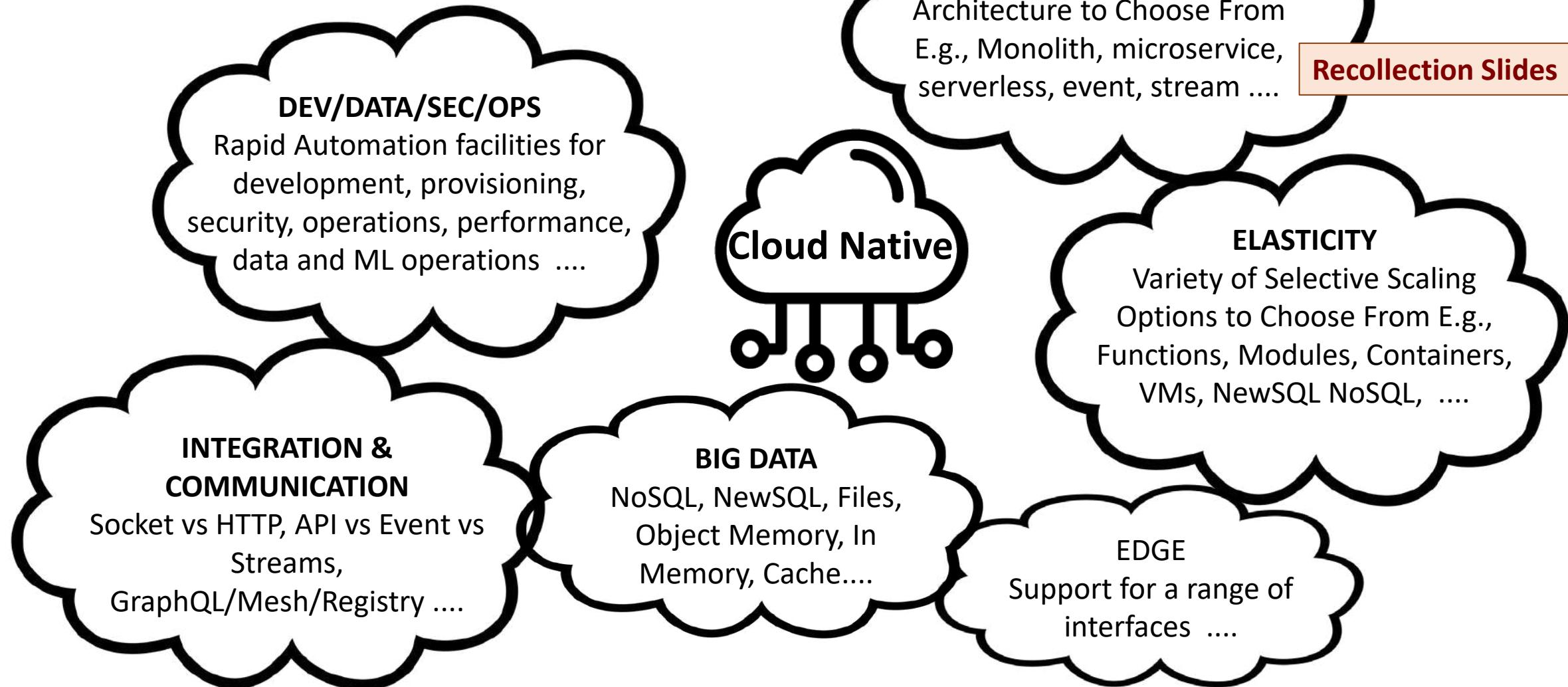
Traditional Web Application



- Application Request Processing
- Navigation, Page Layout & Page Rendering
- Security, Authentication & Authorization
- Caching and Value Bean Management
- Session Management
- Validation and Form Processing
- Exception Management, Errors & Feedback
- Logging and Instrumentation
- Internationalization of Resources
- Wizards, Workflows & Rule Engines



Cloud Native Application

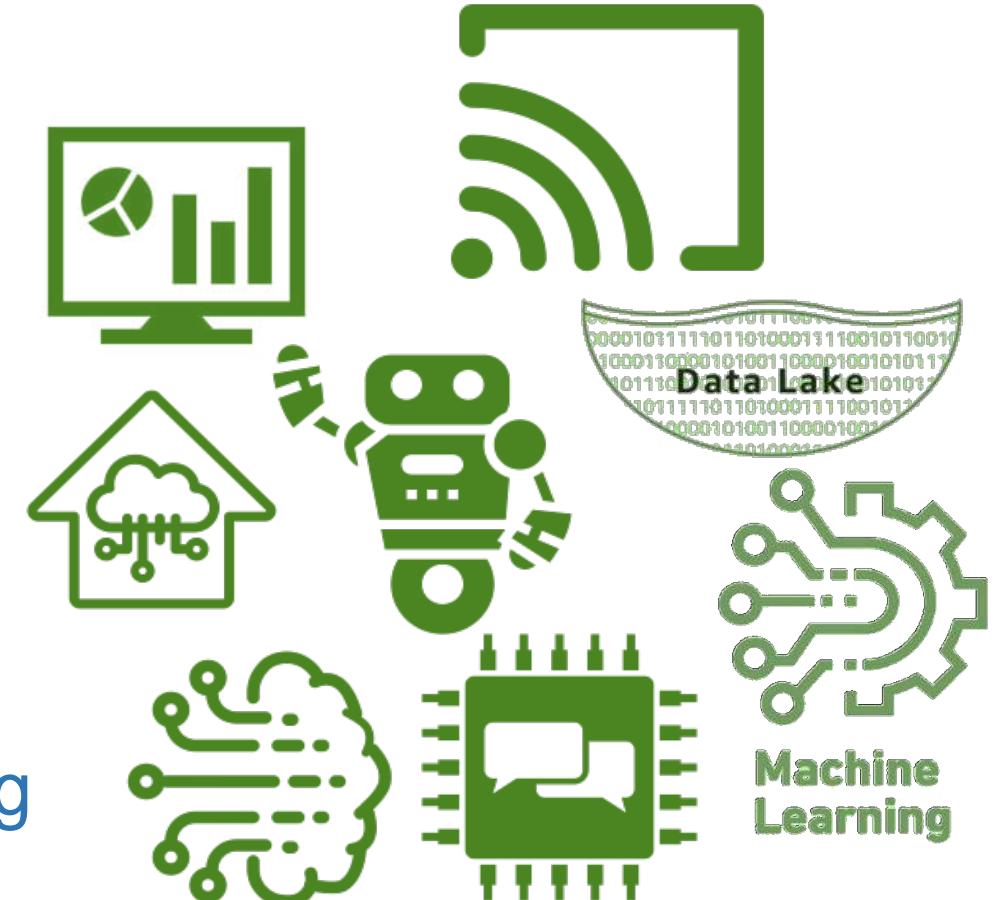




And the list does not stop there . . .

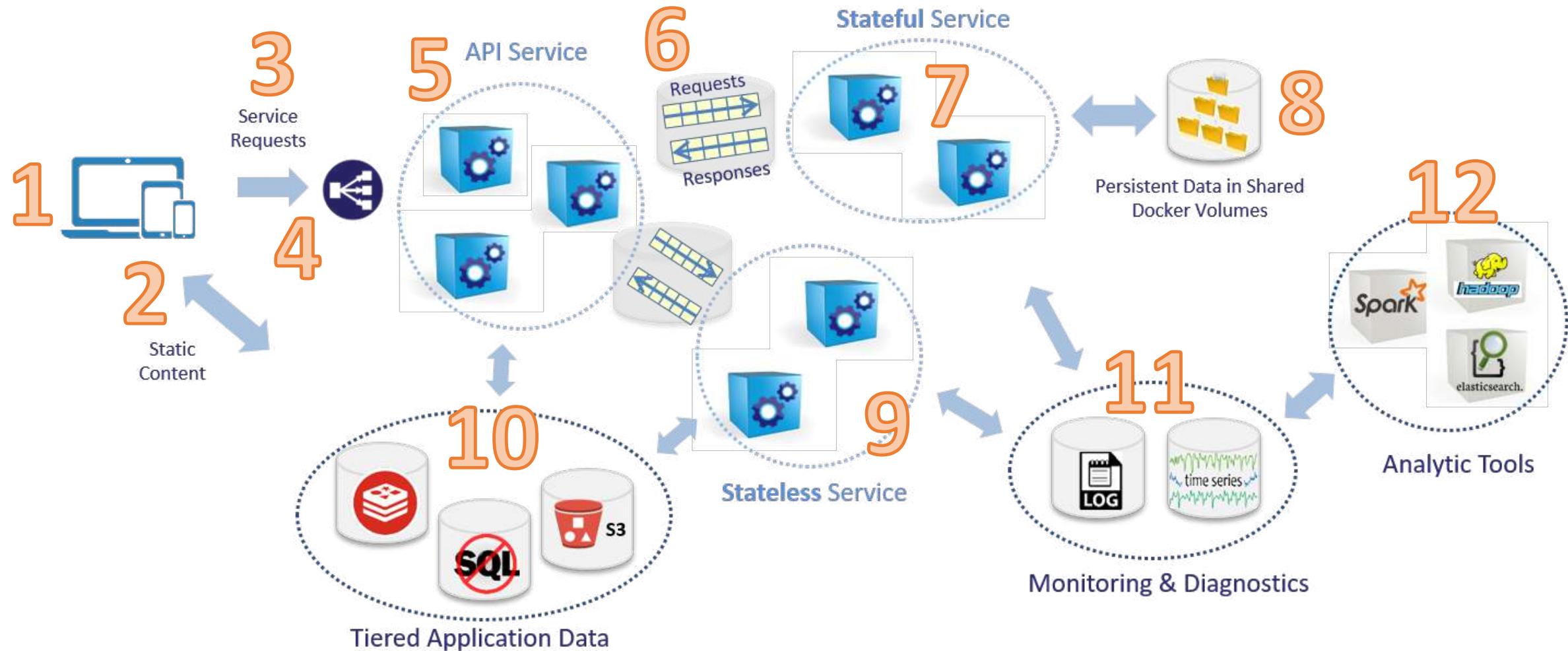
Recollection Slides

- Green Field Technologies . . .
 - Big Data Analytics
 - Stream Processing
 - Data Lakes and Meshes
 - Machine Learning
 - NLP & Text Analytics
 - Deep Learning
 - Robotics, IoT and Edge Computing
 - Many more



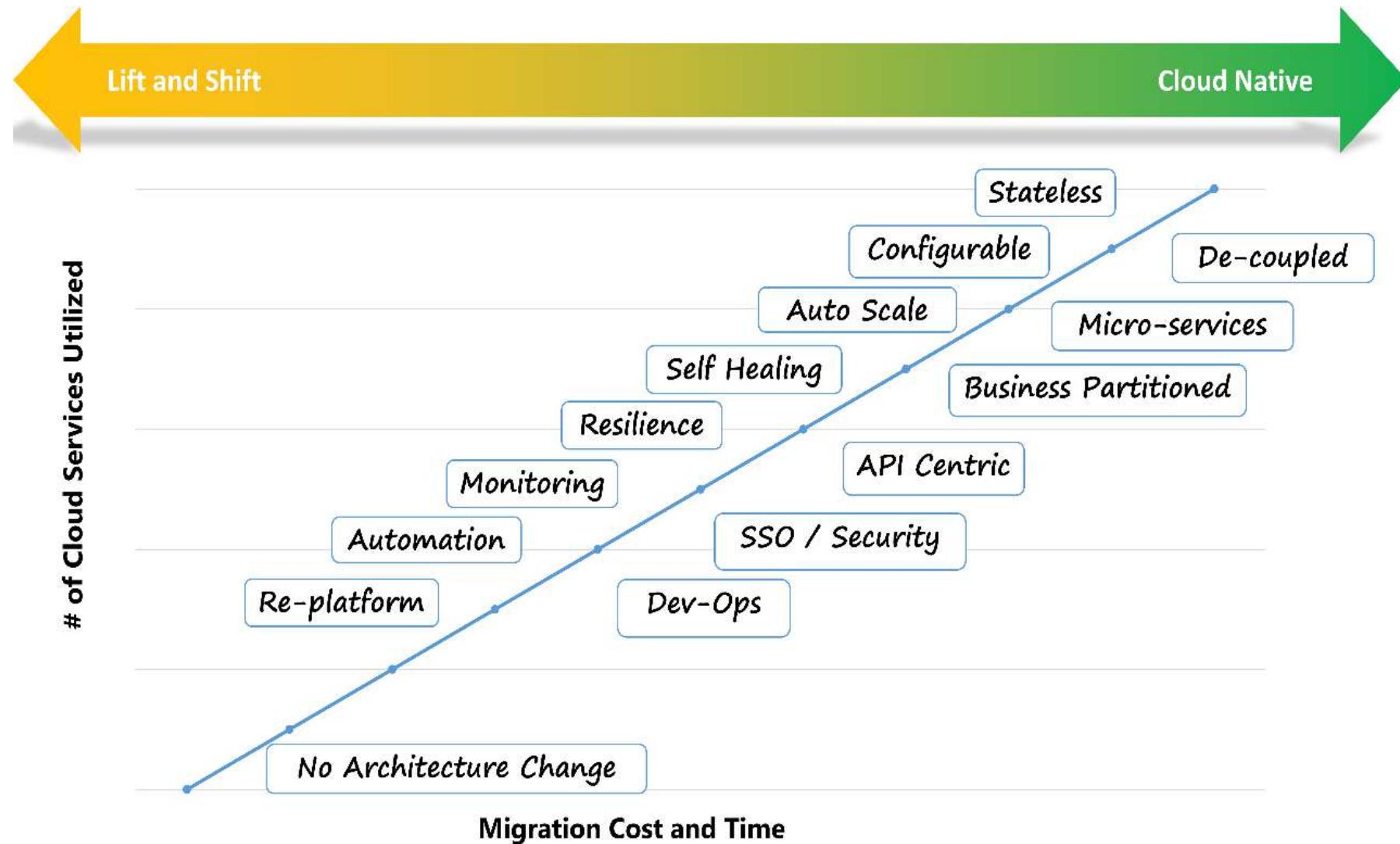


Cloud Native Application Components - Example





Lift & Shift Vs Cloud Native

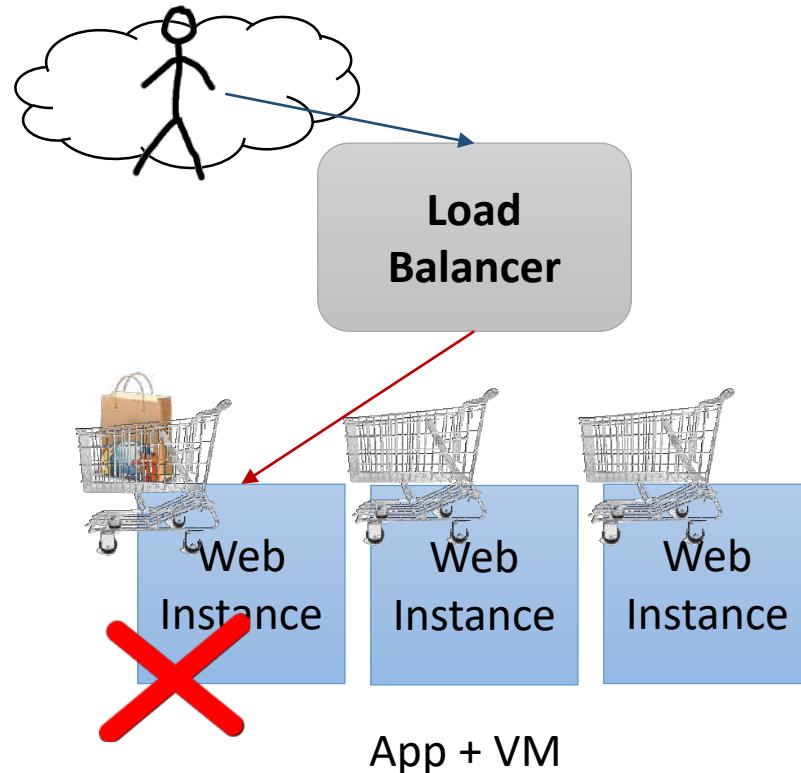




Traditional Vs Cloud Native Apps

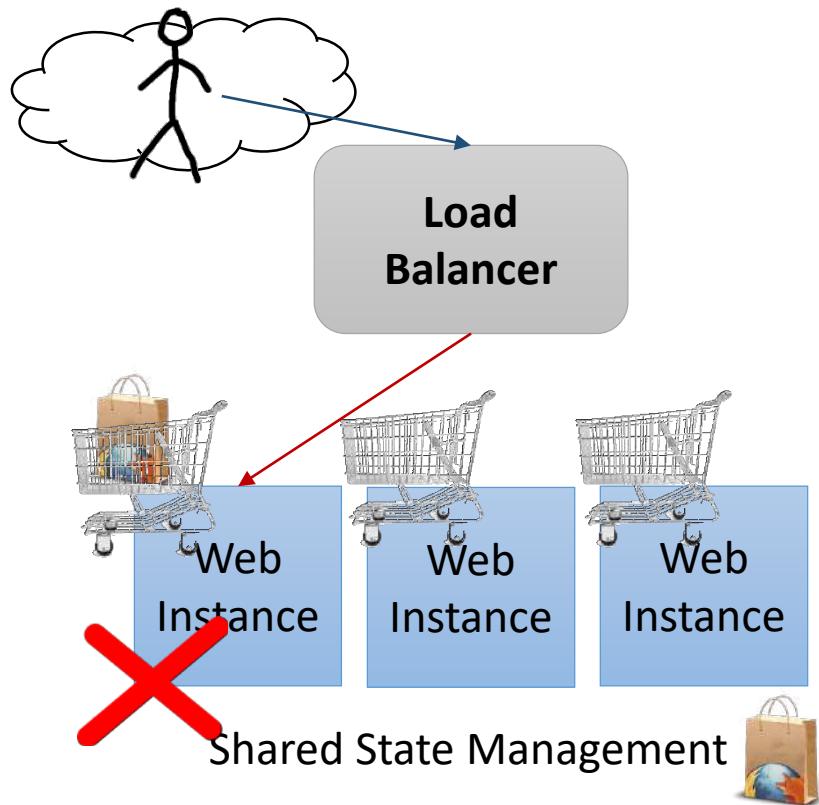
Traditional Applications (Pets)

▪ Sticky Session



Cloud Native Apps (Cattle)

▪ Shared State

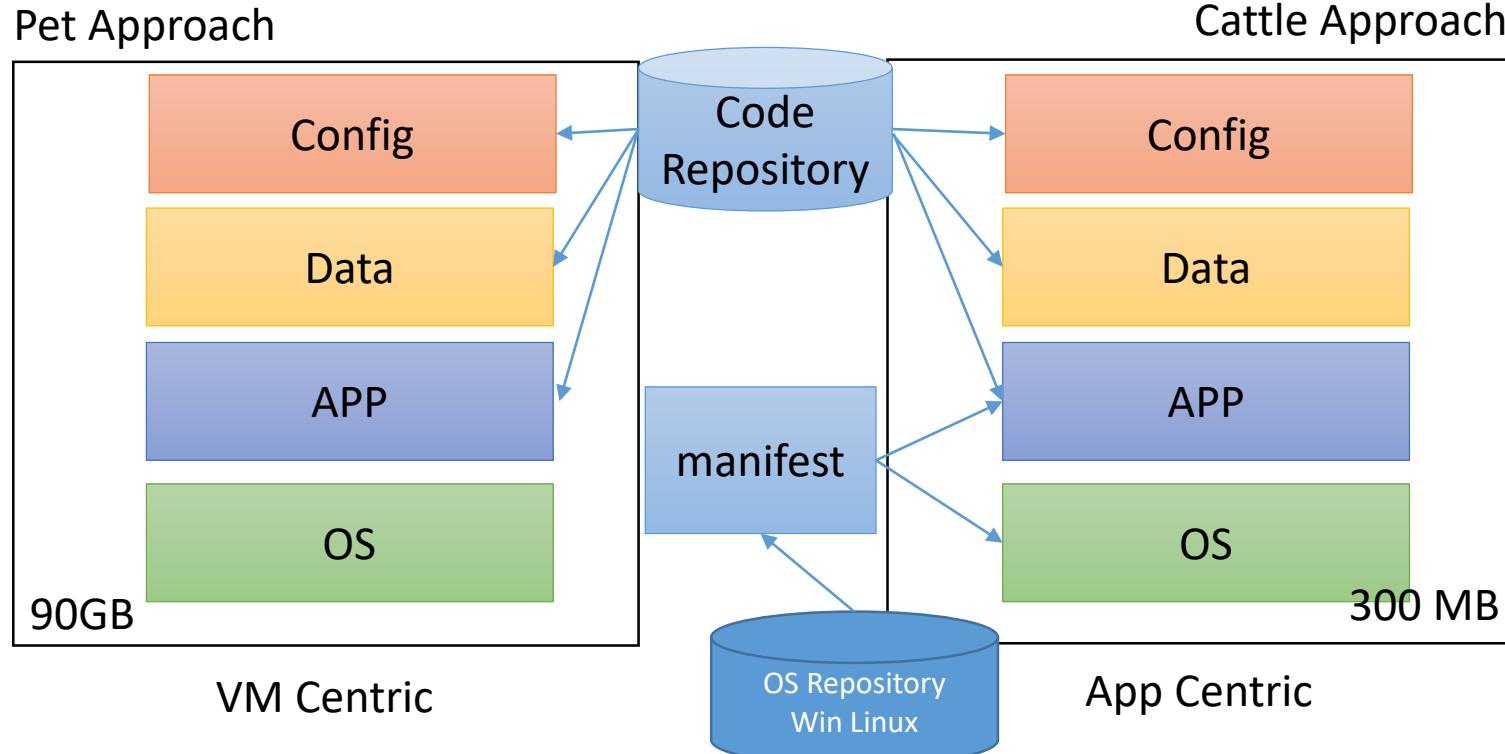




Deeper Mechanics

Traditional Applications (Pets)

- VM Centric (instructive)



Cloud Native Apps (Cattle)

- App Centric (deterministic automation)

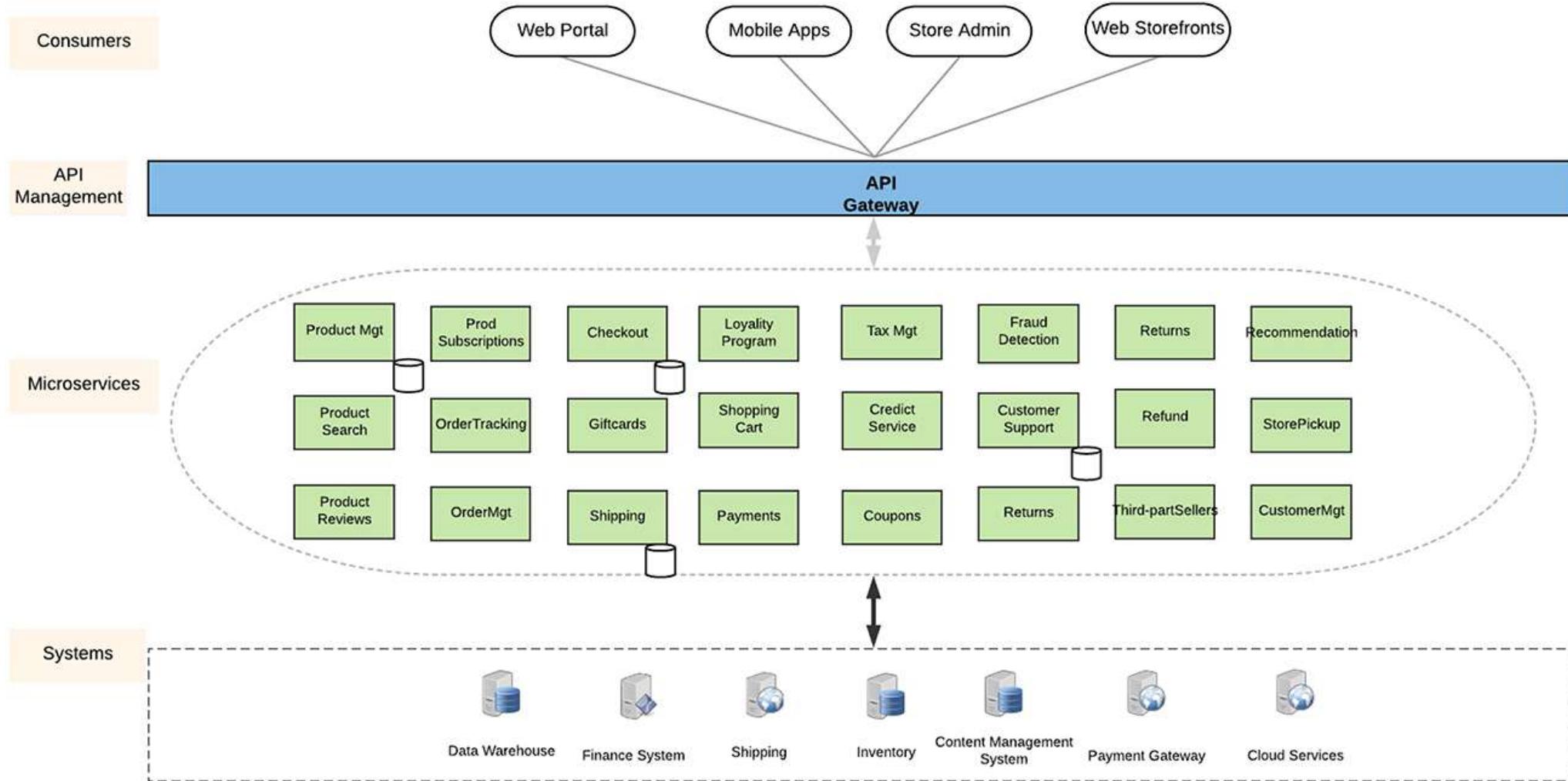


Traditional vs. Cloud Native

	Traditional	Cloud – Native
Focus	Longevity and stability	Speed to market
Development Methodology	Waterfall, semi-agile development	Agile development, DevOps
Teams	Isolated development, operations, QA, and security teams	Collaborative DevOps teams
Delivery Cycles	Long	Short and continuous
Application Architecture	Tightly coupled Monolithic	Loosely coupled Service-based API-based communication
Infrastructure	Server-centric Designed for on-premise Infrastructure-dependent Scales vertically Provisioned for peak capacity	Container-centric Designed for on-premise and cloud Portable across infrastructure Scales horizontally On-demand capacity



Example: An online retail application built using microservices

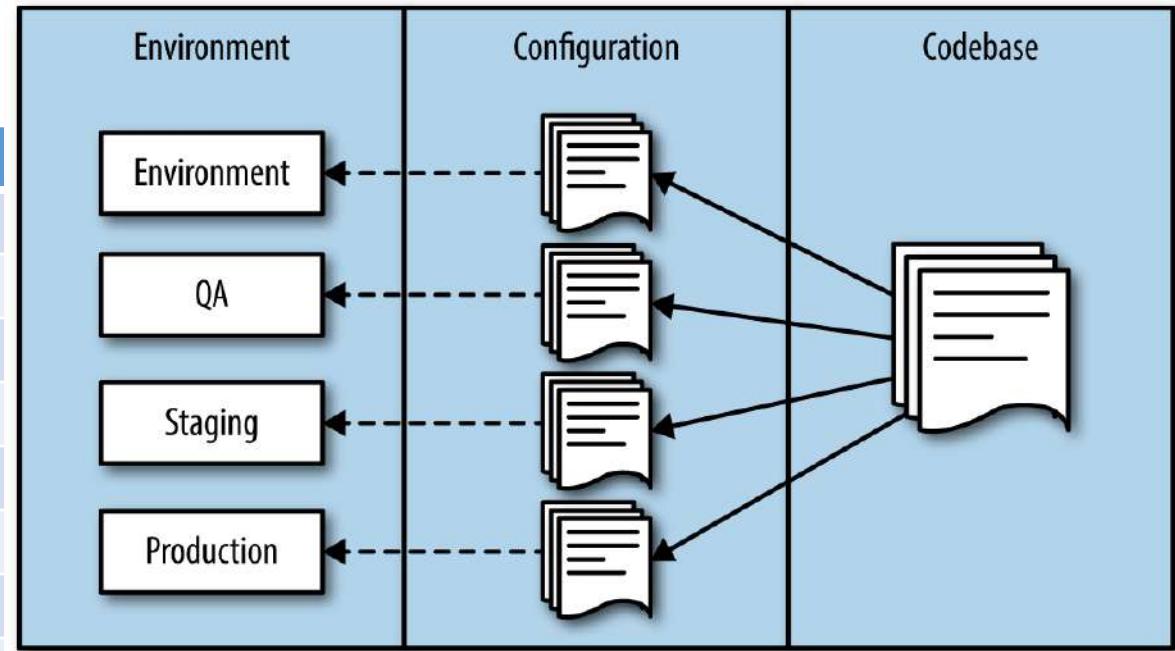




The practices of a twelve-factor application

Attribute	Description
Codebase	One codebase tracked in revision control, many deploys
Dependencies	Explicitly declare and isolate dependencies
Config	Store config in the environment
Backing services	Treat backing services as attached resources
Build, release, run	Strictly separate build and run stages
Processes	Execute the app as one or more stateless processes
Port binding	Export services via port binding
Concurrency	Scale out via the process model
Disposability	Maximize robustness with fast startup and graceful shutdown
Dev/prod parity	Keep development, staging, and production as similar as possible
Logs	Treat logs as event streams
Admin processes	Run admin/management tasks as one-off processes

Twelve Factor Application





The 12-factor app - 1

▪ Single codebase:

- The application must have one codebase, tracked in revision control for every application (read: microservice) that can be deployed multiple times (development, test, staging, and production environments).

▪ Dependencies:

- The application must explicitly declare its code dependencies and add them to the application or microservice. The dependencies are packaged as part of the microservice JAR/WAR file.

▪ Config:

- The application configuration data is moved out of the application or microservice

▪ Backing services:

- All external resources, access should be an addressable URL. For example, SMTP URL, database URL, service HTTP URL, queue URL, and TCP URL.



The 12-factor app - 2

▪ Build, release, and run:

- The entire process of building, releasing, and running is treated as three separate steps.
- This immutable entity will pick the relevant configuration to run the process based on the environment (development, testing, staging, or production).

▪ Processes:

- The microservice is built on and follows the shared-nothing model allowing seamless scalability and load balance.

▪ Port binding:

- The microservice is built within a container.

▪ Concurrency:

- The microservice process is scaled out, meaning that, to handle increased traffic, more microservice processes are added to the environment.
- Within the microservice process, one can make use of the reactive model to optimize the resource utilization.



The 12-factor app – 3.

▪ **Disposability:**

- Immutable with a single responsibility to maximize robustness with faster boot-up times
- Immutability also lends to the service disposability.

▪ **Dev/prod parity:**

- The environments across the application life cycle—DEV, TEST, STAGING, and PROD—are kept as similar as possible to avoid any surprises later.

▪ **Logs:**

- Logs should be treated as event streams and pushed out to a log aggregator infrastructure.

▪ **Admin processes:**

- The microservice instances are long-running processes that continue unless they are killed or replaced with newer versions.



Cloud Native Maturity Model

Cloud Native

- Microservices architecture
- API-first design

Cloud Resilient

- Fault-tolerant and resilient design
- Cloud-agnostic runtime implementation
- Bundled metrics and monitoring
- Proactive failure testing

Cloud Friendly

- 12 Factor App methodology
- Horizontally scalable
- Leverages platform for high availability

Cloud Ready

- No permanent disk access
- Self-contained application
- Platform-managed ports and networking
- Consumes platform-managed backing services



Attributes of Cloud Native Application

Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach. These techniques enable loosely coupled systems that are resilient, manageable, and observable.

- Predictable
 - Predictable behaviour in development, runtime and operation
- OS abstraction
 - Underlying infrastructure dependencies is abstracted away
- Right-sized capacity
 - Dynamically allocate resources based on ongoing needs of the application
- Collaborative
 - Close collaboration between development and operations functions
- Continuous delivery
 - Software are released rapidly to get a tighter feedback loop
- Independent
 - Application is decomposed into small loosely independently operating services
- Automated scalability
 - Infrastructure automation at scale to eliminate downtime due to human error
- Rapid recovery
 - Use containers to allow faster recovery in the event of application failure



In Summary . . . Cloud Native Applications Supports

- Microservices

- Small, independent services that work together to form a complete application

- Loose coupling

- Can be easily divided into smaller parts that can be independently deployed and updated

- Scalability

- Can easily handle increased demand by adding more resources

- Resilience

- Continues to function even if part of the system fails

- Agility

- Applications can be quickly deployed and updated

- Security

- Applications use the security features of the cloud platform they are running on

- Automation

- Applications are deployed and updated without human intervention

- Self-healing

- Can automatically recover from failures

- DevOps culture

- Developers and operations staff work together to create and deploy applications



SCALING & ARCHITECTURE TYPES

SCALING CUBE

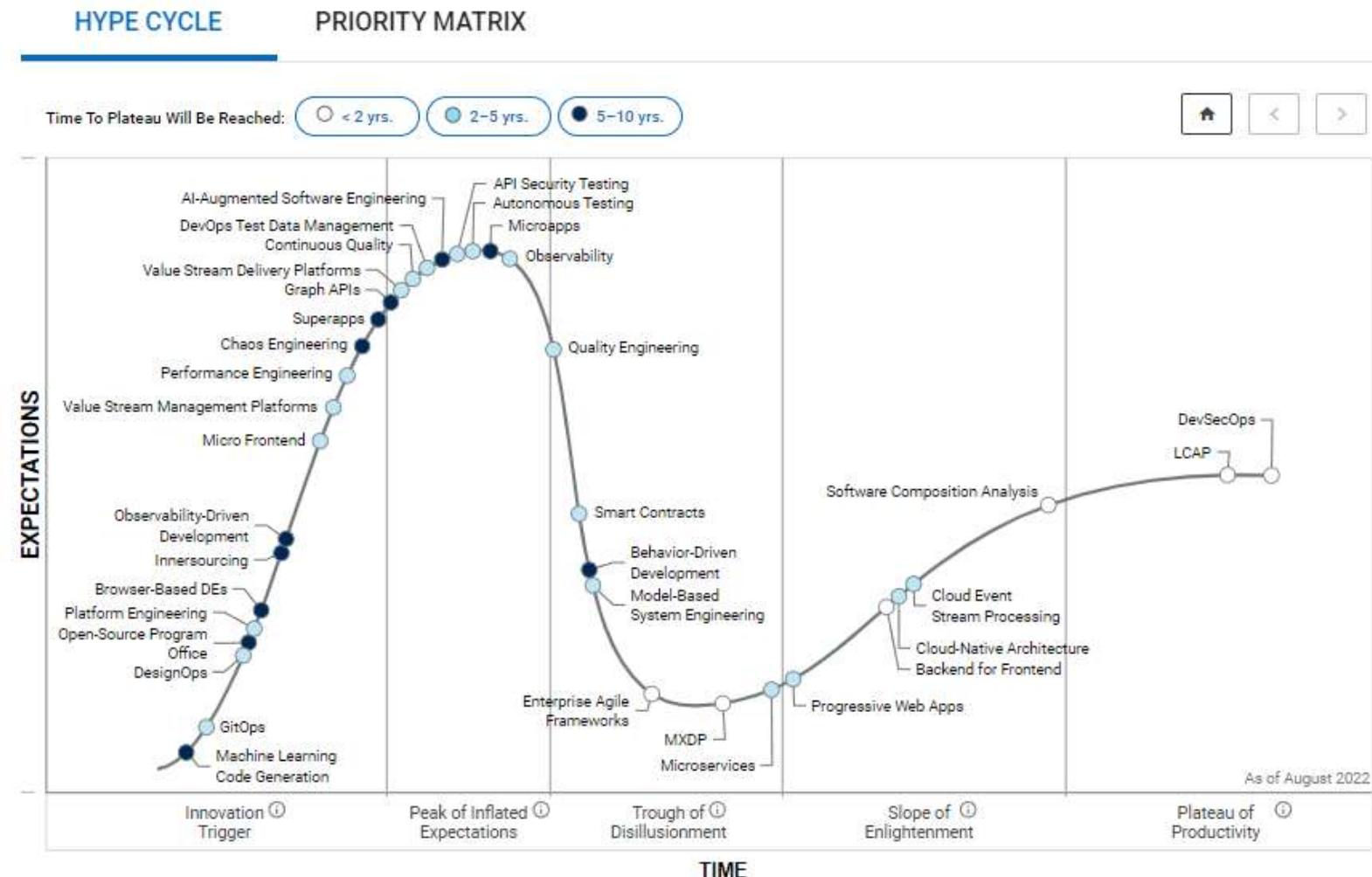
MONOLITHIC AND MICROSERVICES

HEXAGONAL AND SERVERLESS ARCHITECTURE

LAMBDA AND KAPPA ARCHITECTURE



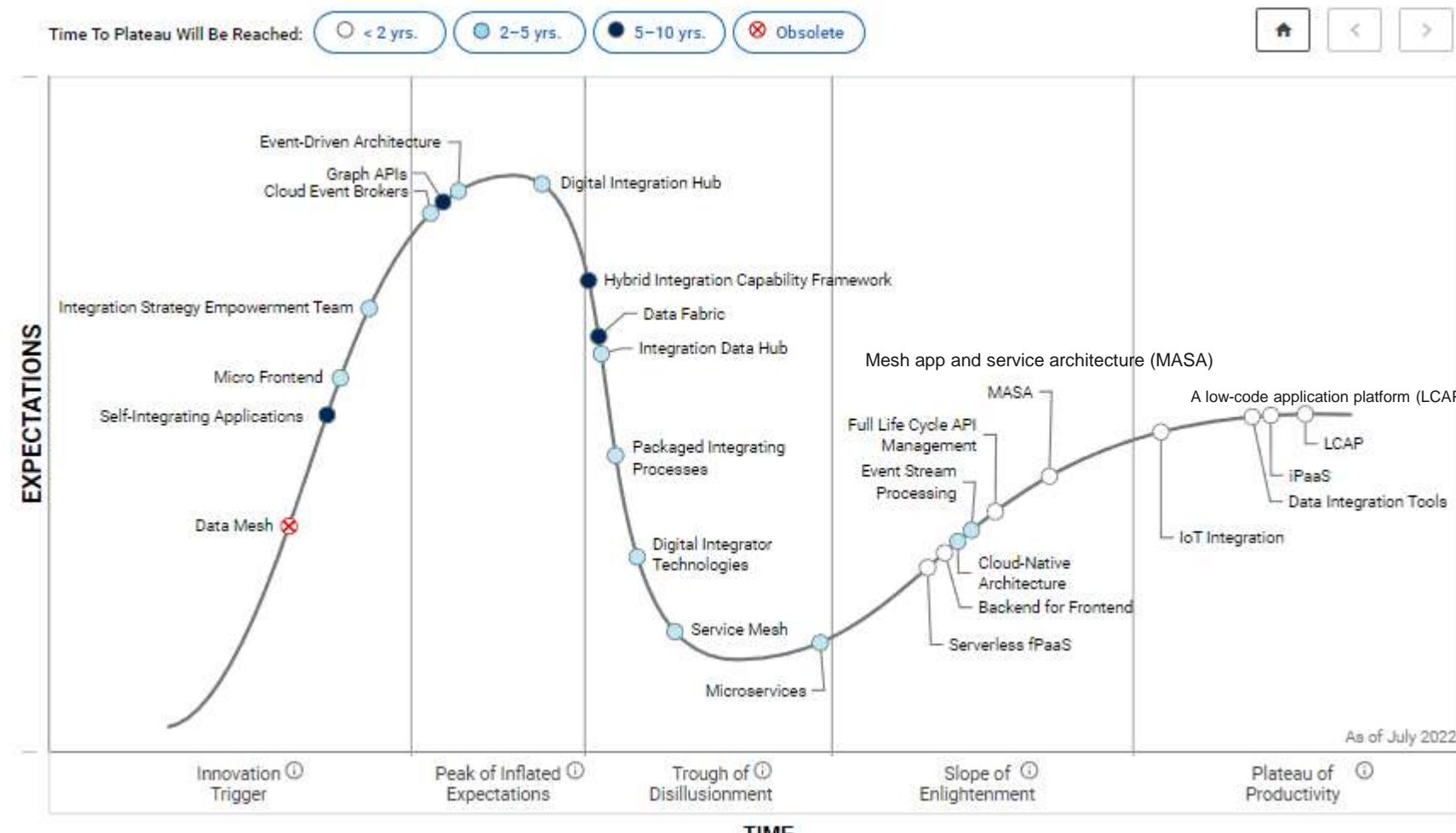
Hype Cycle for Software Engineering, 2022



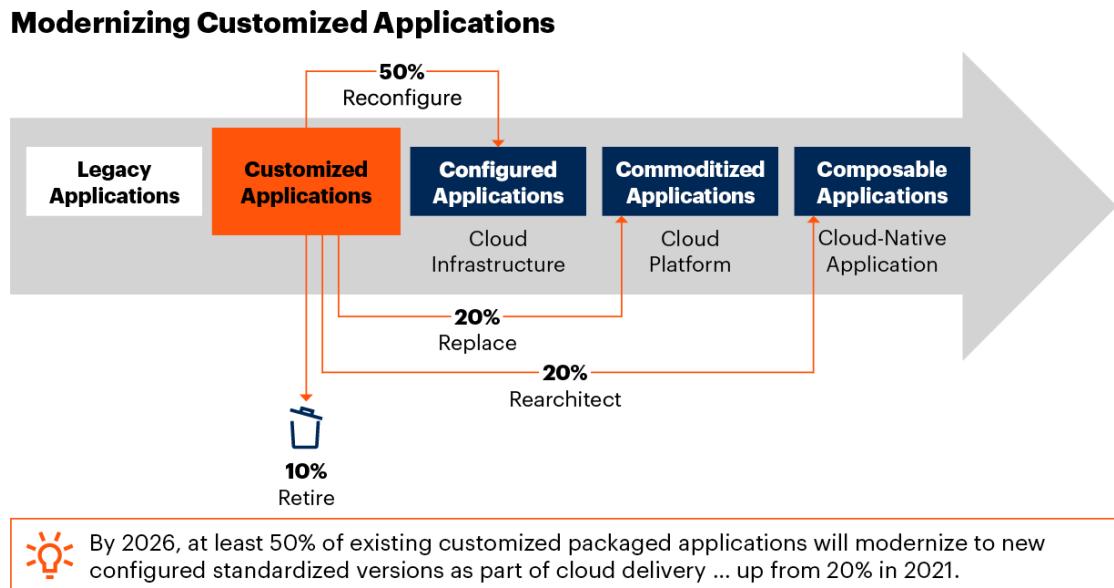
- DevSecOps and software composition analysis has become mainstream.
- Observability, platform engineering, model-based system engineering, and smart contracts will mature in the next two to five years
- AI-augmented software engineering and machine learning code generation tools are on the horizon.



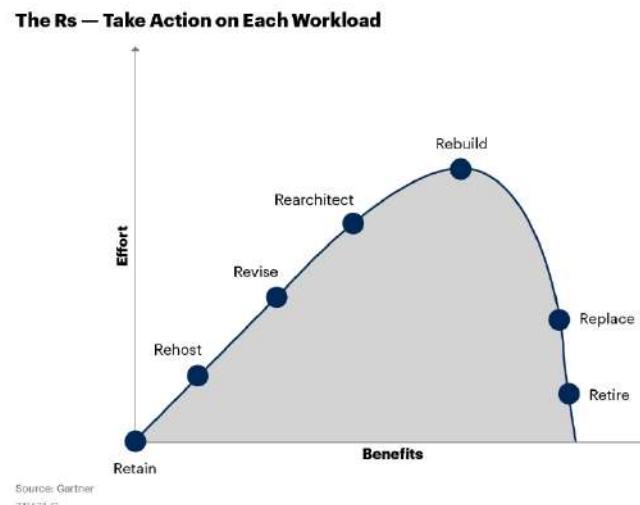
Hype Cycle for Application Architecture and Integration, 2022



- Low code systems and Mesh Apps are here to stay.
- Cloud Native Architecture is well defined.
- Event Stream processing, serverless/microservices and service meshes will mature in two to five years time frame.
- Event driven architecture and self integrating systems are on the horizons.

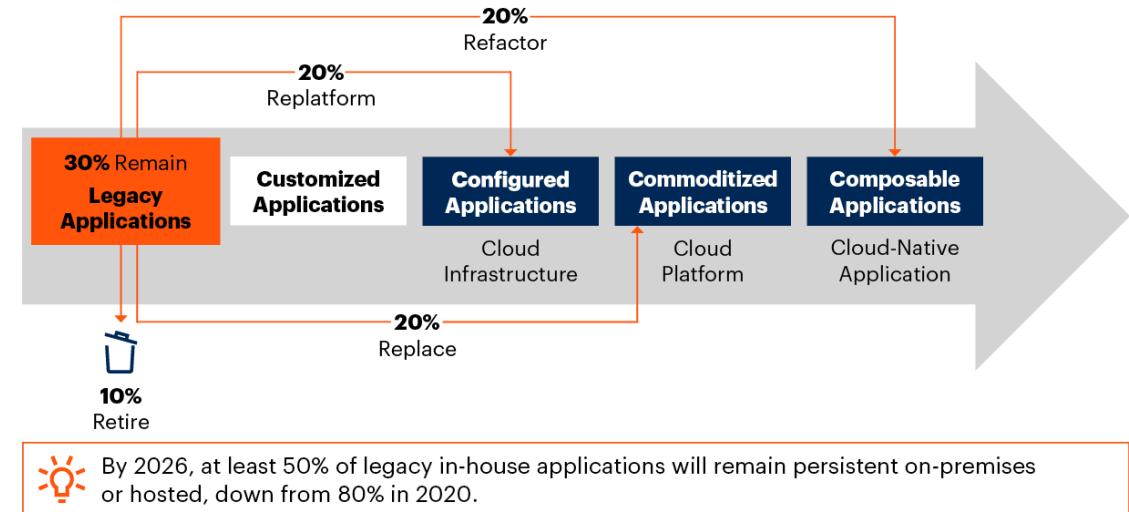


Source: Gartner
740803 C

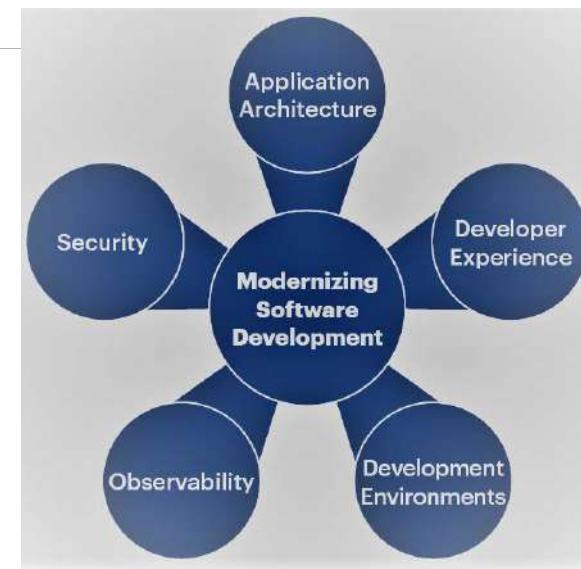


Gartner

Migrating Legacy Applications



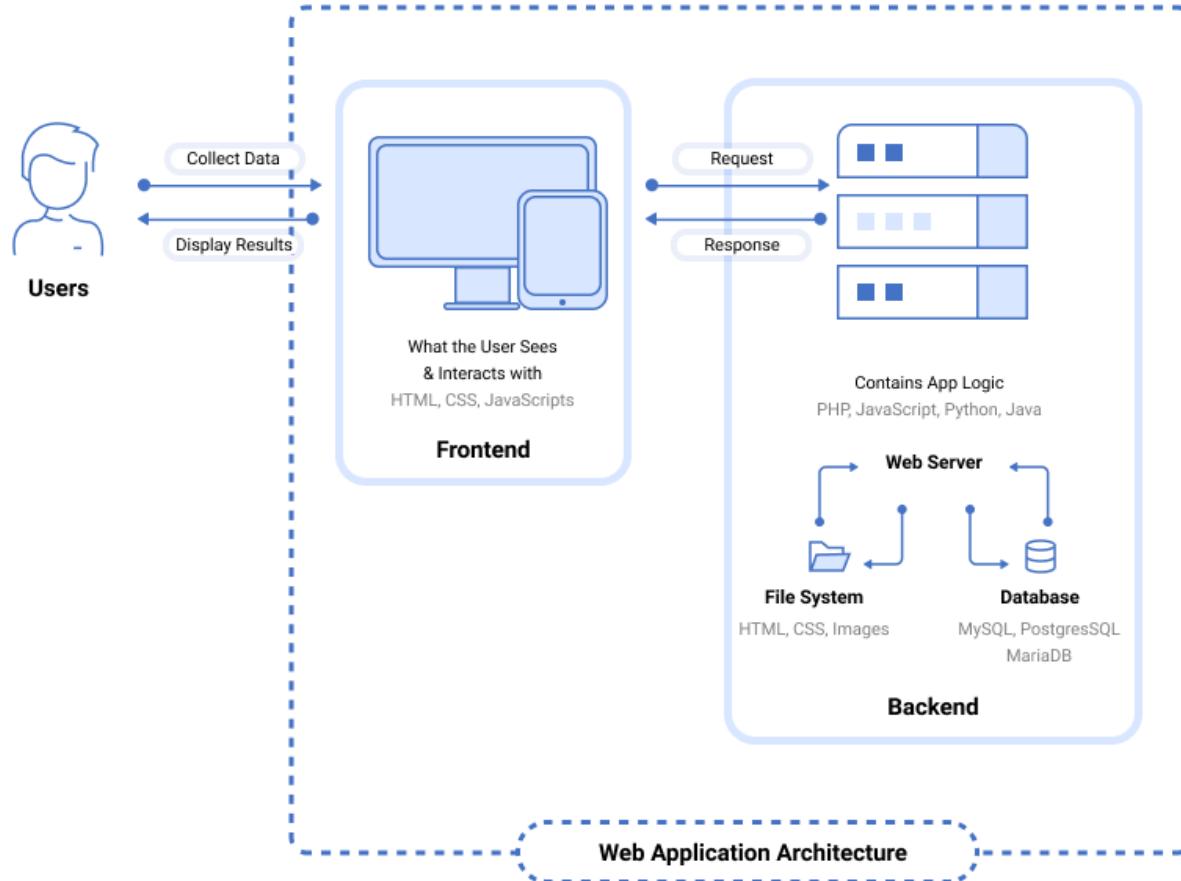
Source: Gartner
740803 C



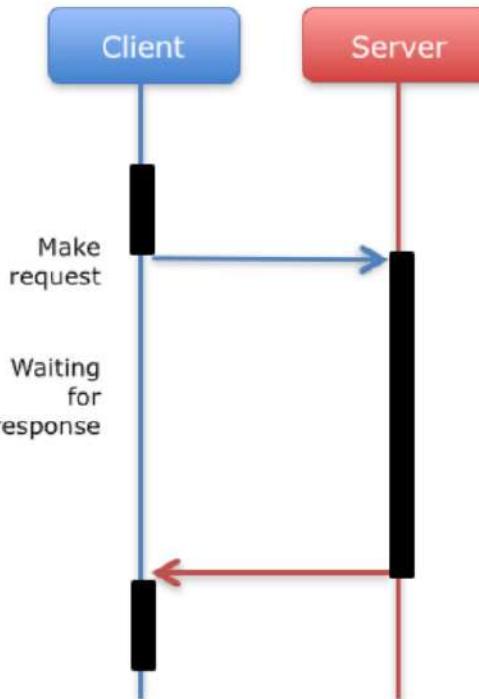
Rationalizing Applications and Infrastructure for Cloud Delivery Published 28 May 2021 - ID G0074080



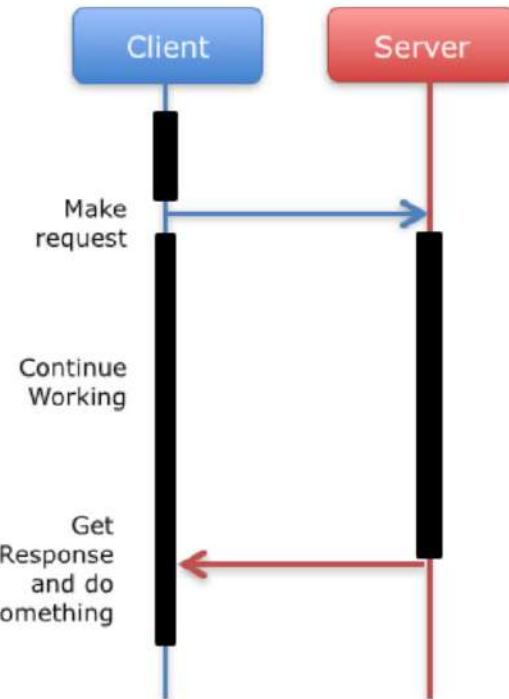
Full Stack Development



Synchronous



Asynchronous





Reactive Manifesto

▪ Responsive

- The system responds in a timely manner if at all possible.

▪ Resilient

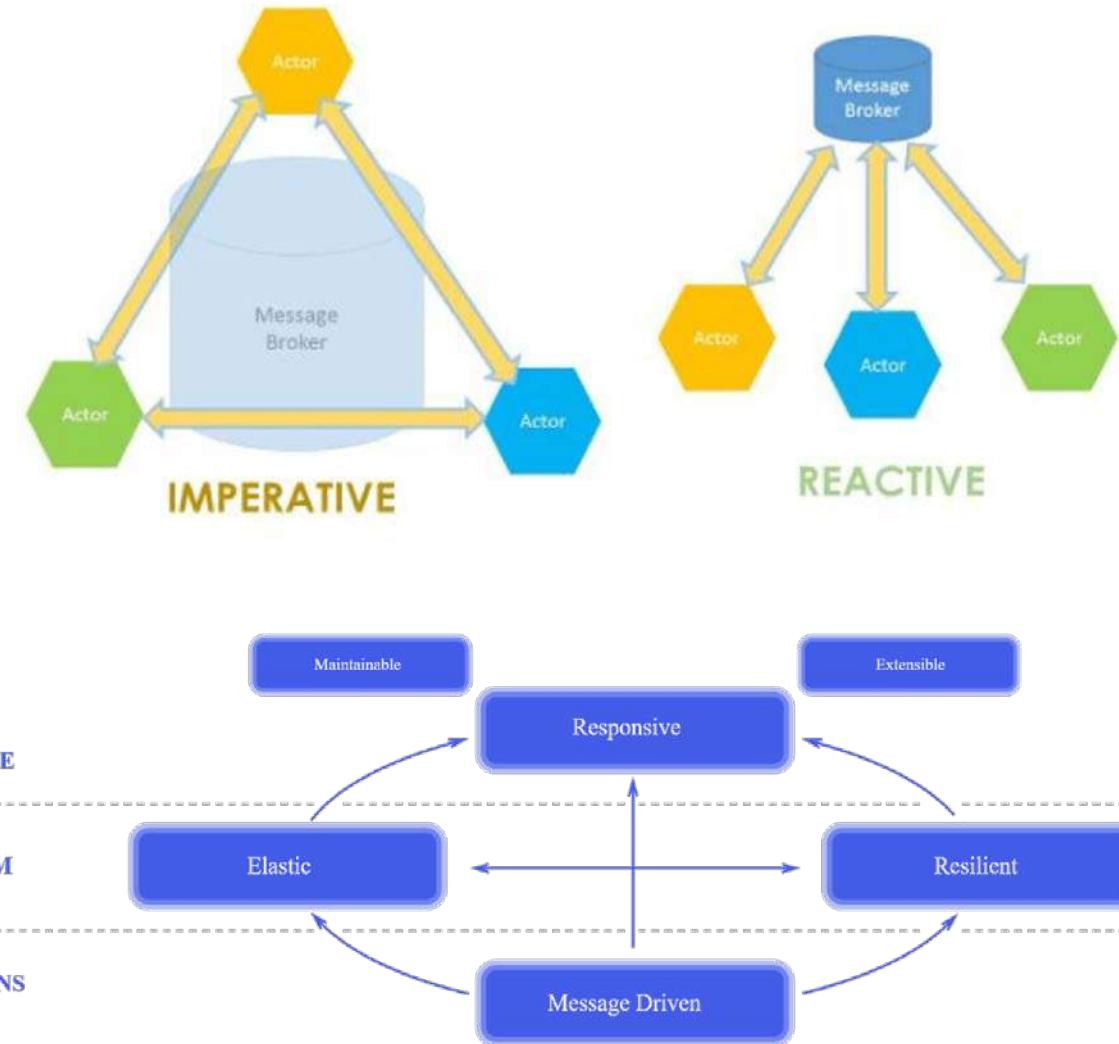
- The system stays responsive in the face of failure. This applies not only to highly-available, mission-critical systems — any system that is not resilient will be unresponsive after a failure.

▪ Elastic

- The system stays responsive under varying workload. Reactive Systems can react to changes in the input rate by increasing or decreasing the resources allocated to service these inputs.

▪ Message Driven

- Reactive Systems rely on asynchronous message-passing to establish a boundary between components that ensures loose coupling, isolation and location transparency.





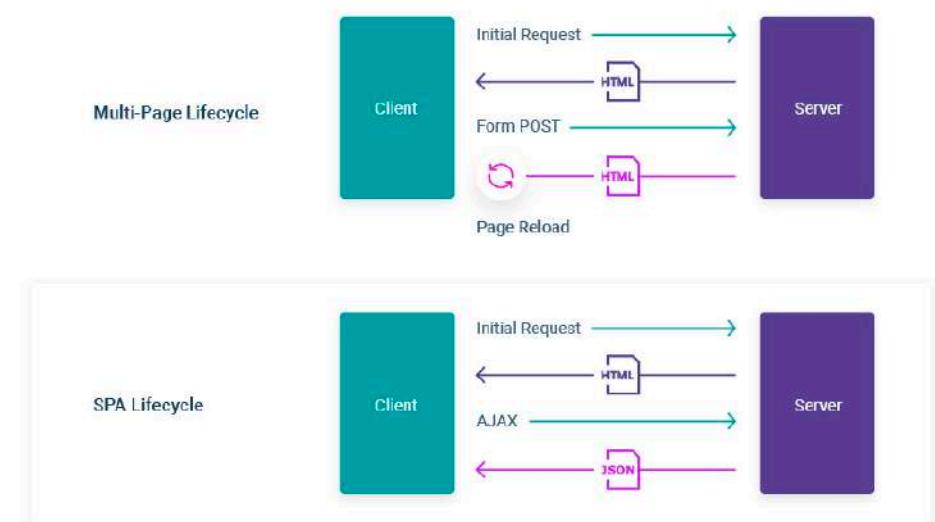
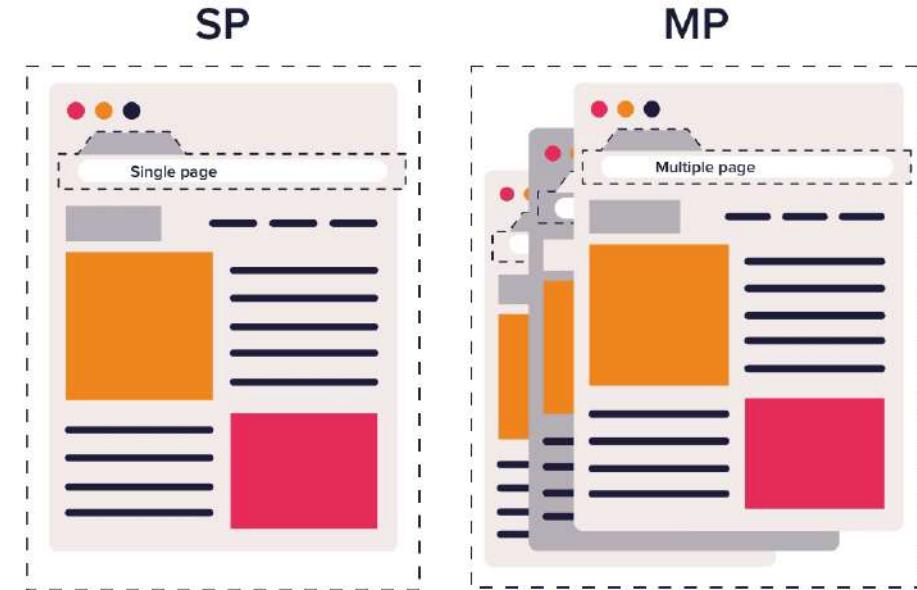
Single Page Vs Multi Page App

▪ Single-Page App or SPA

- SPA stands for a website or a web app that loads all the required information when you enter the page. Single-page apps have one significant benefit — they deliver an amazing user experience since users don't experience web pages reloading.
- It's common to develop single-page web apps using JavaScript frameworks like Angular, React, and others.
- **Well-known SPAs:** Gmail, Facebook, Twitter, Slack

▪ Multi-Page App or MPA

- Multi-page applications are more popular on the Internet since all the websites used to be MPA in the past.
- These days, companies choose MPAs if their website is pretty large (like eBay). Such solutions reload a web page to load or send information from/to a server via the users' browsers.
- **Well-known MPAs:** eBay, Amazon





Progressive Web Apps and Wasm

- Progressive Web Apps (PWA) are built and enhanced with modern APIs to deliver enhanced capabilities, reliability, and installability while reaching *anyone, anywhere, on any device* with a single codebase.



- *WebAssembly (Wasm) is a small and portable VM that runs in modern web browsers, servers, and on a vast array of diverse hardware. Wasm brings new capabilities and additional security features to cloud native developers.*



Progressive Vs Native App

- **Native app** live on the device and are accessible through icons on the home screen of the device. Native apps are installed via an app store.

- Example - Swift, Python, Java, Kotlin, Flutter

- **Progressive web apps (PWA)** offer an alternative approach, improving an app's cross-platform performance across web and mobile.

- Example – Angular, Ionic, ReactJS

App	Description	Installation	Access to the device's features	Internet connection	Updates
Native app	App is written specifically for Android or iOS	Need to be installed on the device via App Store or Google Play Store	Full access	In most cases don't depend on the internet connection	Users must update the app on their devices
Web app	Launches via browser and works on a remote server	Installation on the device isn't required	Limited access to the hardware	Requires constant internet connection	Promptly delivered to users
PWA	Has offline mode available thanks to caching	Not installed on the device	Instant and automated updates	Limited access to the hardware	Website with native-app-like features

Ref: <https://www.pontikis.net/blog/a-comprehensive-guide-on-progressive-web-app>

When To Use

PWA

1. Seeking Easy Distribution
2. On a Tight Budget
3. Unsure of the Business Idea
4. Need Faster Time to Market
5. Wish to Reach Billions

Native

1. App Performance is Priority
2. Want to create a 'Reliable' Image
3. Need Interaction with other Apps
4. Have a focus on Security
5. Need to Fetch Consumer Data



Ref: <https://cute766.info/progressive-web-apps-pwa-vs-native-app-which-is-better-for-your-business/>



Distributed Vs Monolithic

■ Monolithic

- Layered architecture
- Pipeline architecture
- Microkernel architecture

M
λ
K
i

■ Distributed

- Service-based architecture
- Event-driven architecture
- Space-based architecture
- Service-oriented architecture
- Microservices architecture

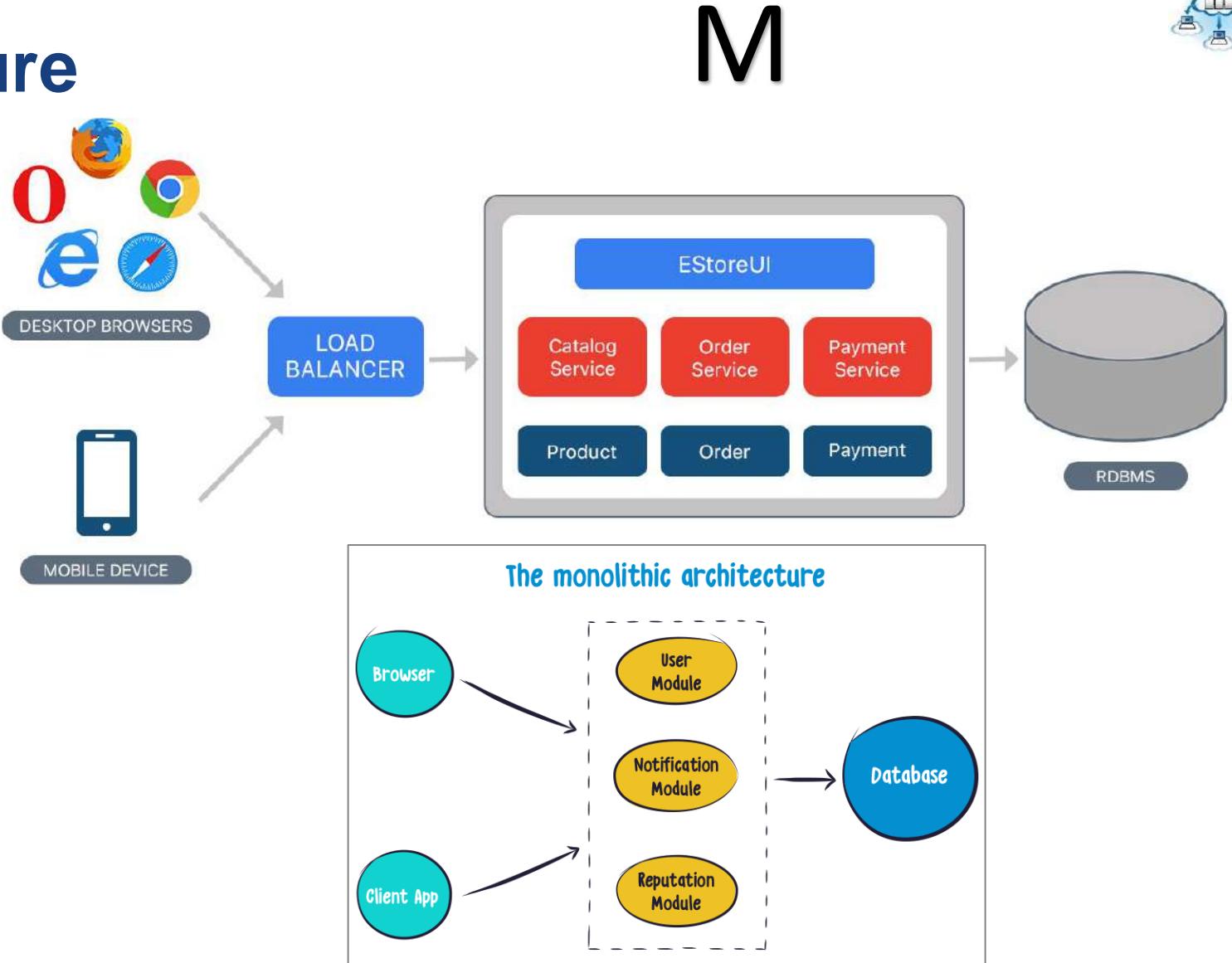
μ

Distributed architecture, while being much more powerful in terms of performance, scalability, and availability than monolithic architecture, have significant trade-offs for this power.



Monolithic Architecture

- Authorization — responsible for authorizing a user
- Presentation — responsible for handling HTTP requests and responding with either HTML or JSON/XML (for web services APIs).
- Business logic — the application's business logic.
- Database layer — data access objects responsible for accessing the database.
- Application integration — integration with other services (e.g. via messaging or REST API). Or integration with any other Data sources.
- Notification module — responsible for sending email notifications whenever needed.



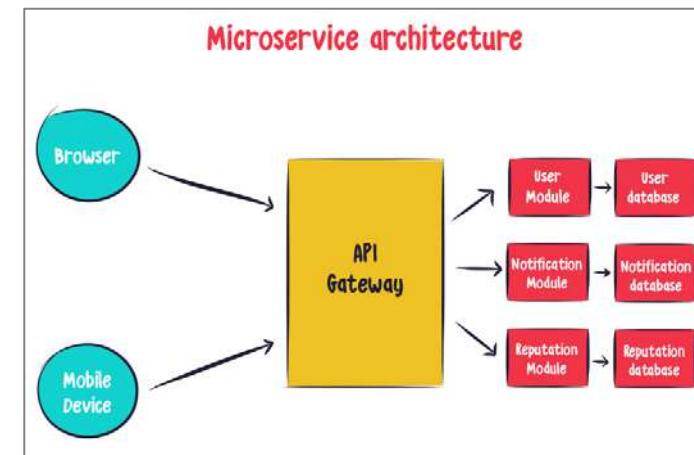
Source: <https://medium.com/koderlabs/introduction-to-monolithic-architecture-and-microservices-architecture-b211a5955c63>



Microservices Architecture

μ

- Authorization Service — Responsible for authorizing customer.
- Order Service — takes an order and process it.
- Catalog Service — Manage products and check products inventory.
- Cart Service — Manage user cart, this service can utilize Catalog service as a data source.
- Payment Service — Manage and Authorize payments.
- Shipping Service — Ships ordered products.



Source: <https://medium.com/koderlabs/introduction-to-monolithic-architecture-and-microservices-architecture-b211a5955c63>



Z

Hexagonal Architecture

- Documented in 2005 by Alistair Cockburn, Hexagonal Architecture is a software architecture that has many advantages and has seen renewed interest since 2015.

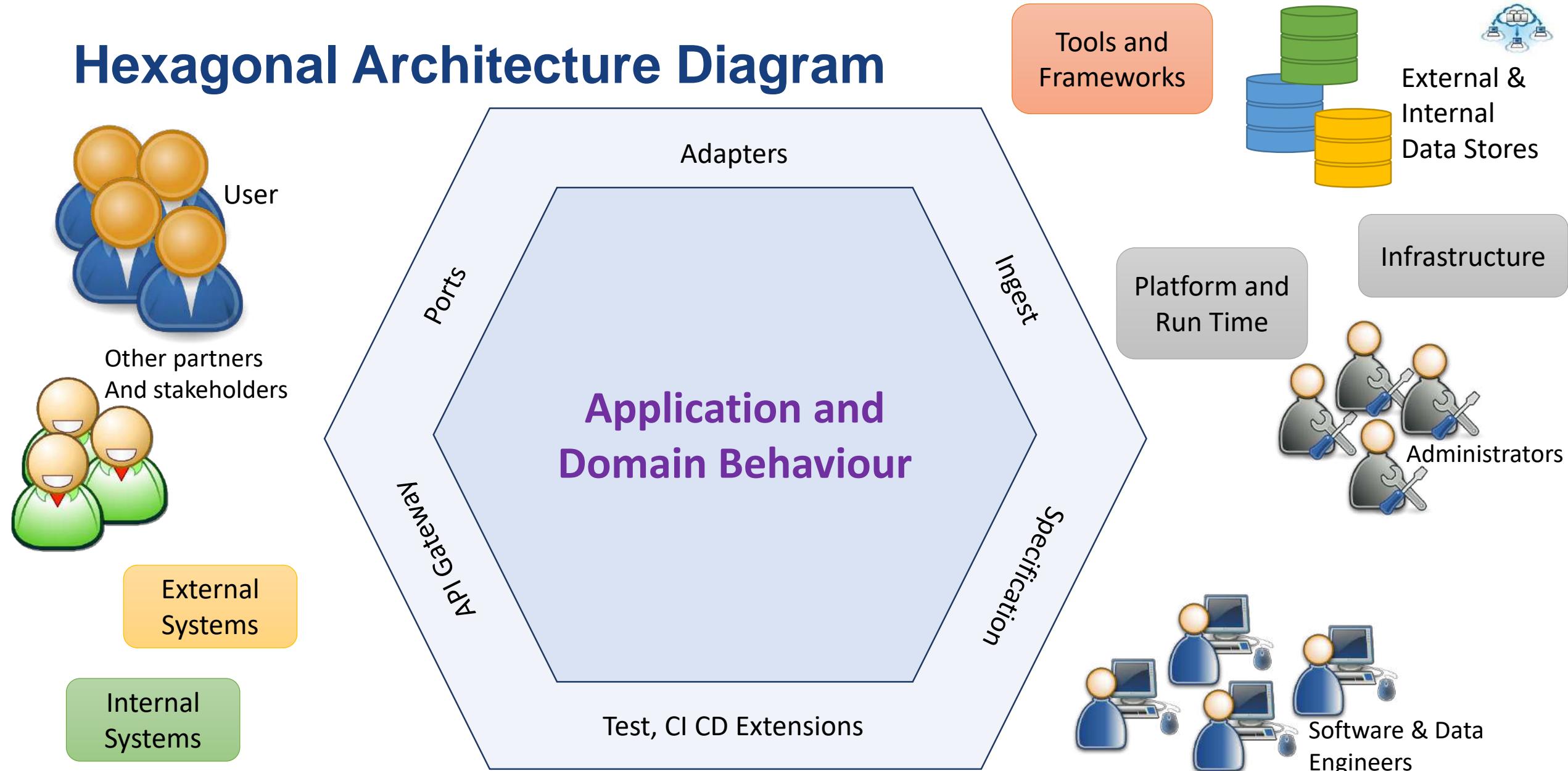
Allow an application to equally be driven by users, programs, automated test or batch scripts, and to be developed and tested in isolation from its eventual run-time devices and databases.

▪ Principles of Hexagonal Architecture

- Explicitly separate Application, Domain, and Infrastructure
- Dependencies are going from Application and Infrastructure to the Domain
- We isolate the boundaries by using Ports and Adapters



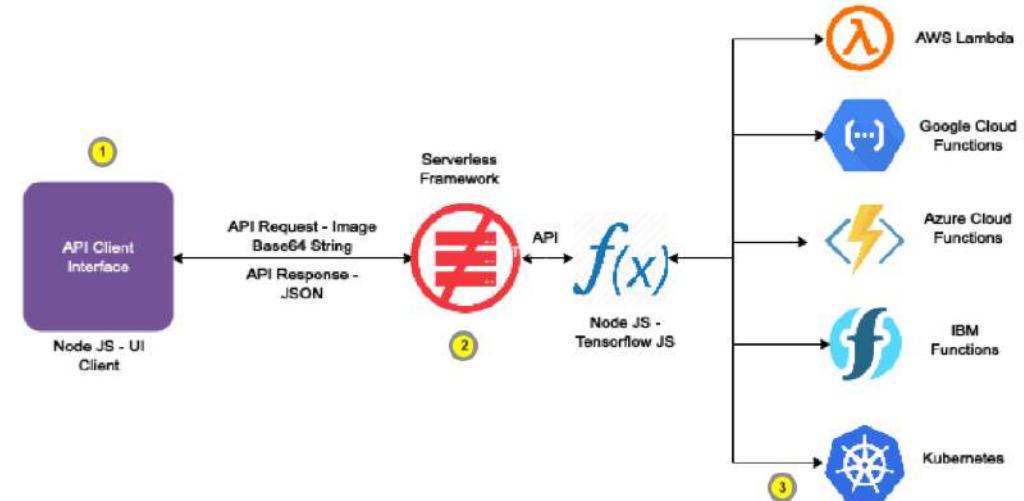
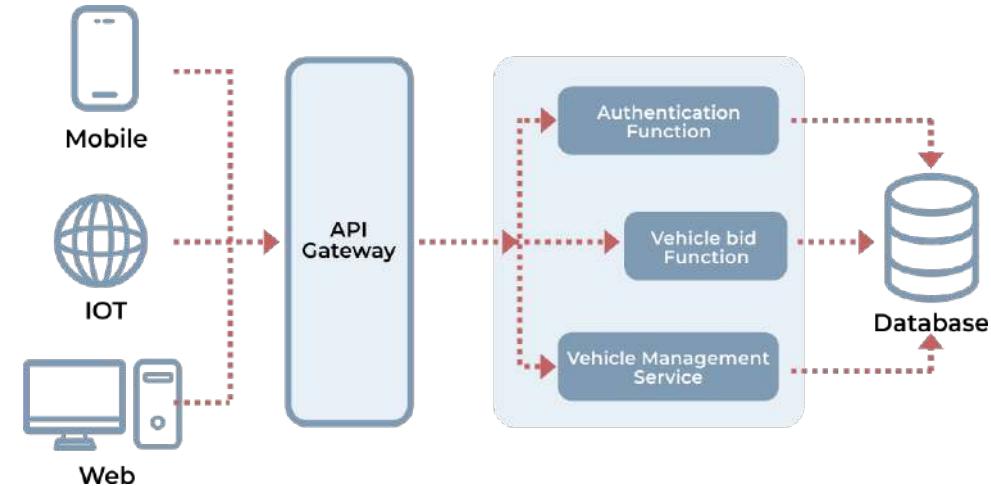
Hexagonal Architecture Diagram





Serverless Architecture

- Build and run applications and services without having to manage infrastructure
- The major advantage is to execute code on need basis, without having to maintain a host environment.
- Rapid provision of resources in real-time, even for unforeseen peak loads and disproportionate growth
- Implementing serverless structures is very labor-intensive

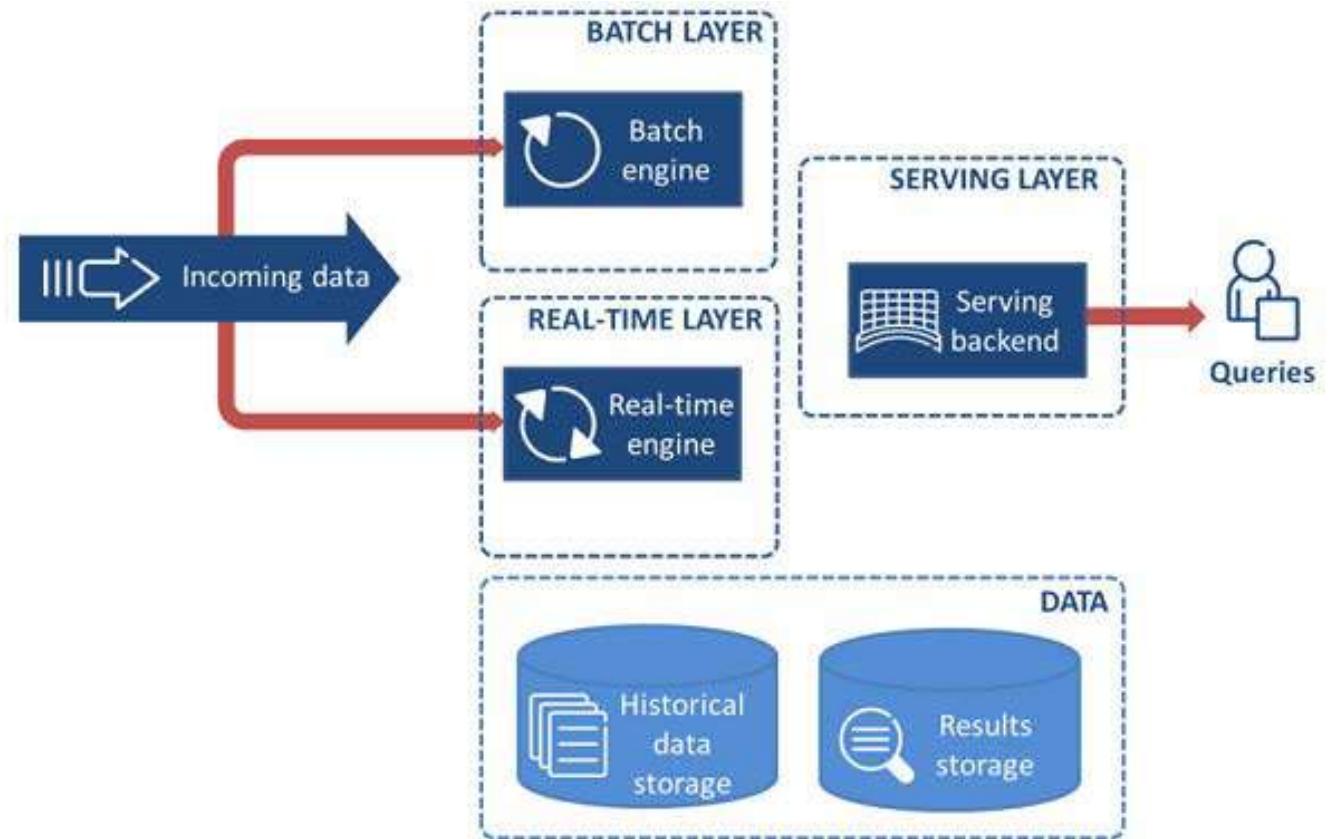




Lambda Architecture

λ

- Lambda is composed of three layers: batch, speed, and serving.
 - **Batch** layer: (a) manage historical data; and (b) re-compute results such as machine learning models.
 - **Speed** layer provides results in a low-latency, near real-time fashion.
 - **Serving** layer enables various queries of the results sent from the batch and speed layers.



Source: <https://www.ericsson.com/en/blog/2015/11/data-processing-architectures--lambda-and-kappa>

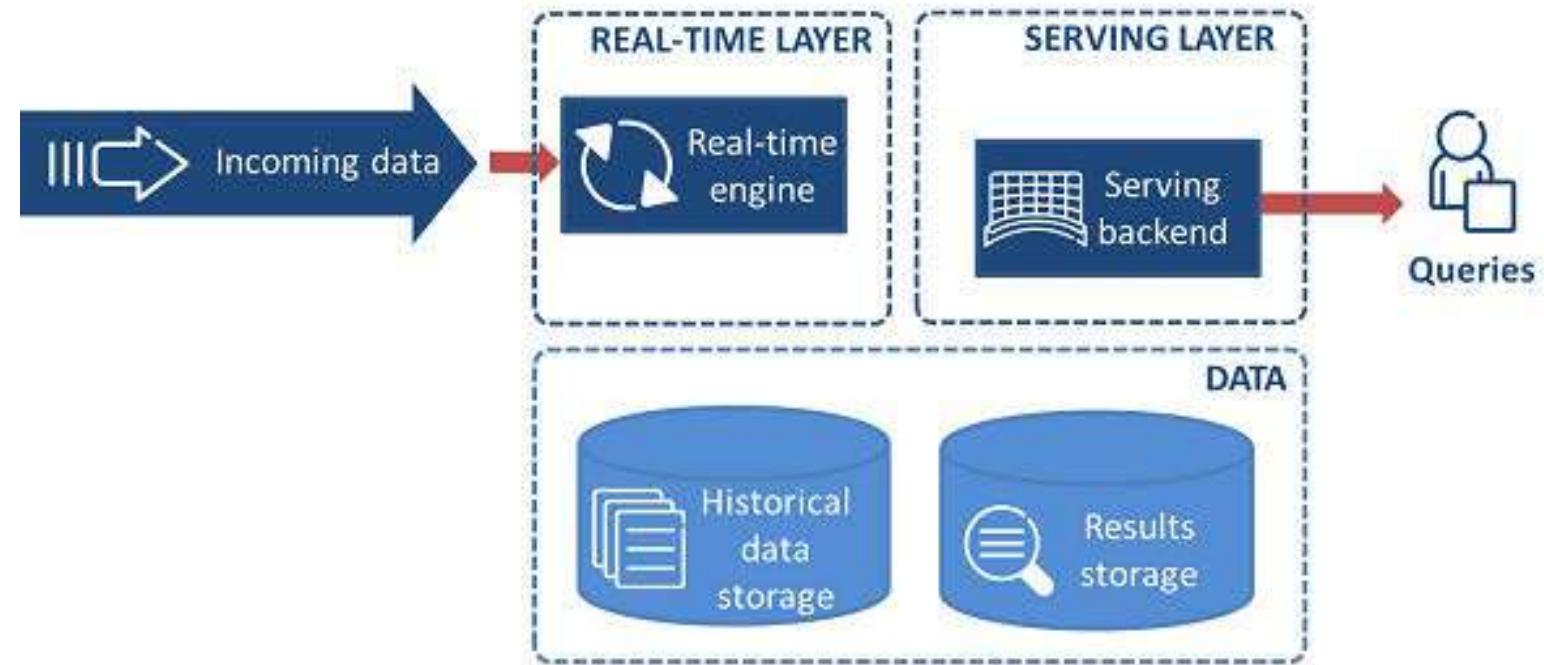


Kappa Architecture

Flink?

K

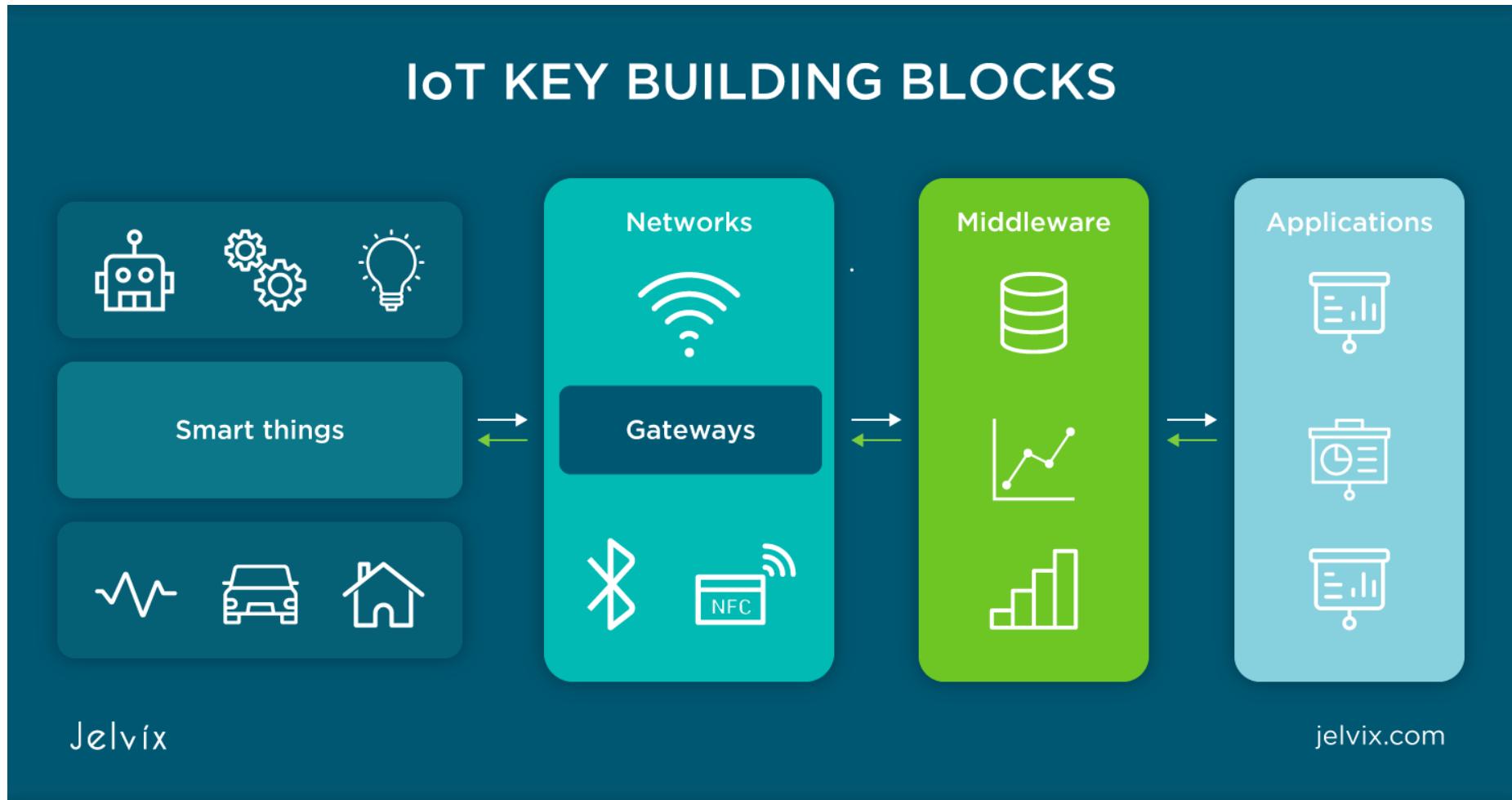
- The key idea is to handle both real-time data processing and continuous data reprocessing using a single stream processing engine.
- Kappa architecture is composed of only two layers: stream processing and serving.



Source: <https://www.ericsson.com/en/blog/2015/11/data-processing-architectures--lambda-and-kappa>



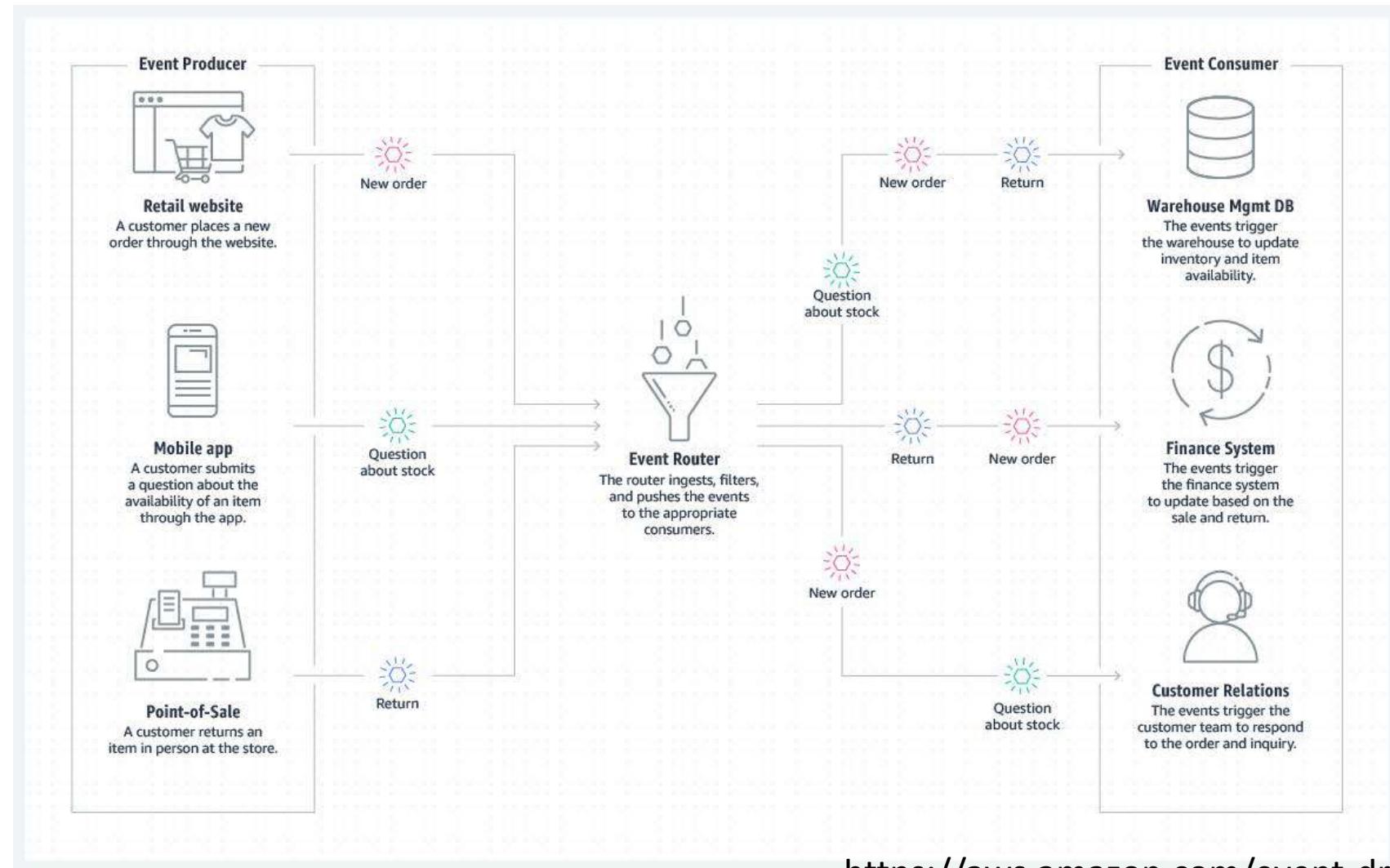
iOTA or IOT Architecture



<https://jelvix.com/blog/iot-architecture-layers>



Event Driven Architecture



ε

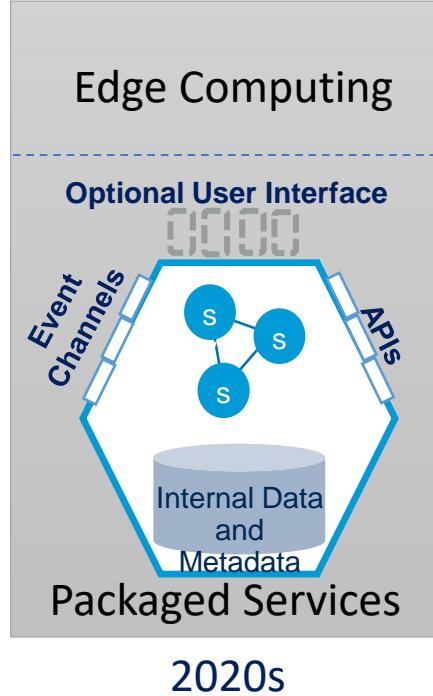
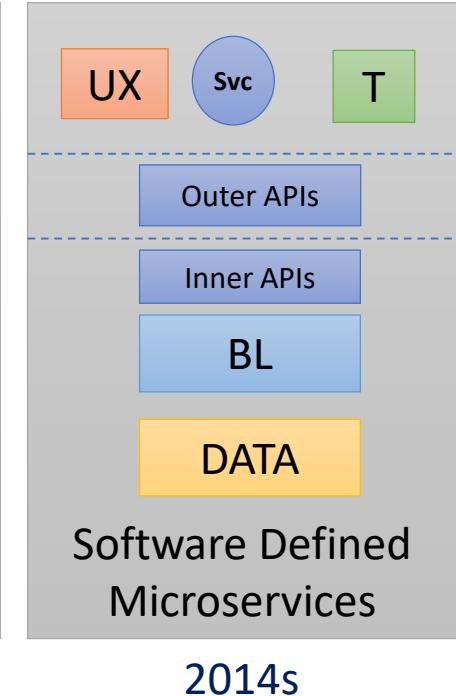
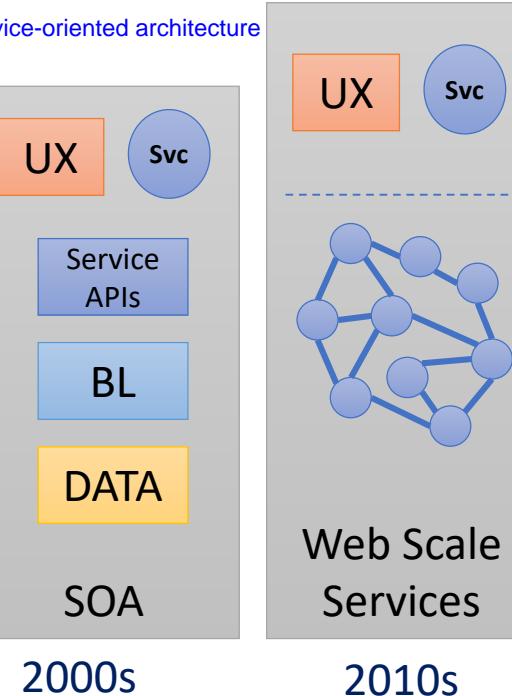
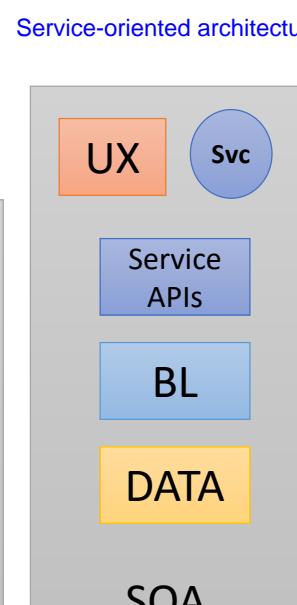
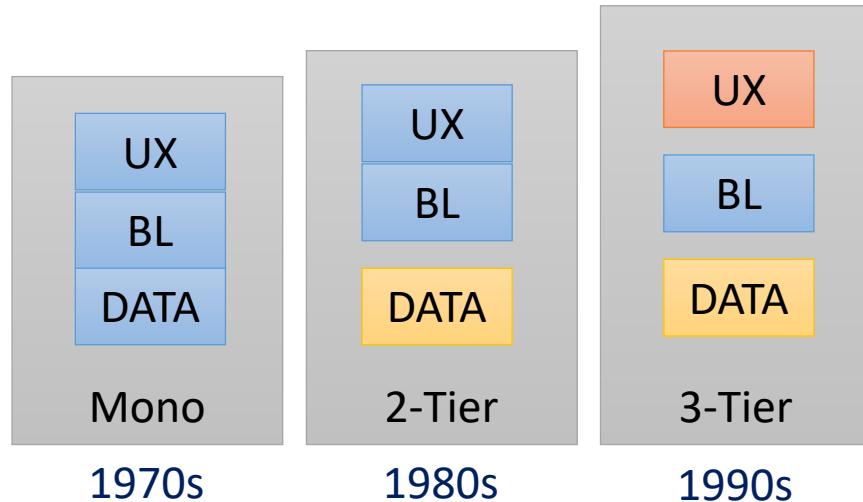
<https://aws.amazon.com/event-driven-architecture/>



From Monoliths to Composable Business Capabilities

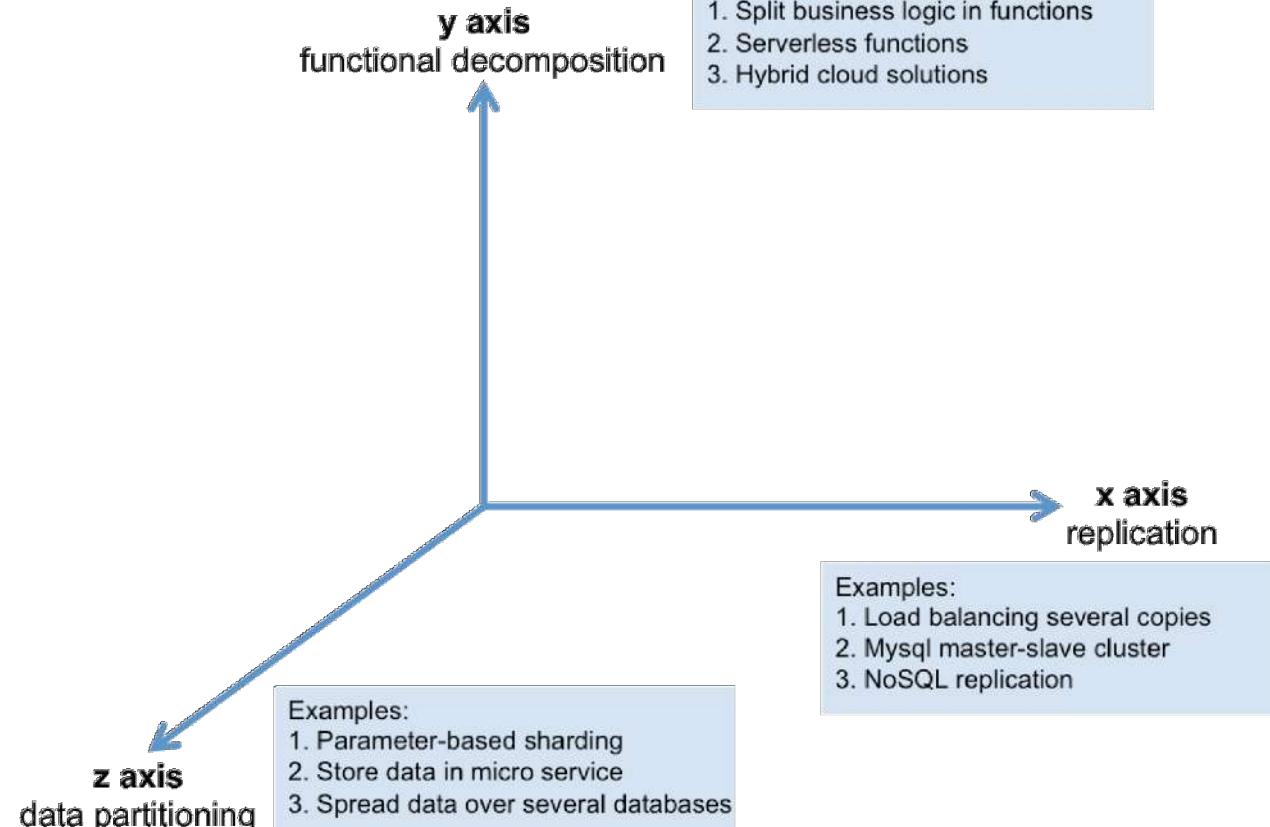
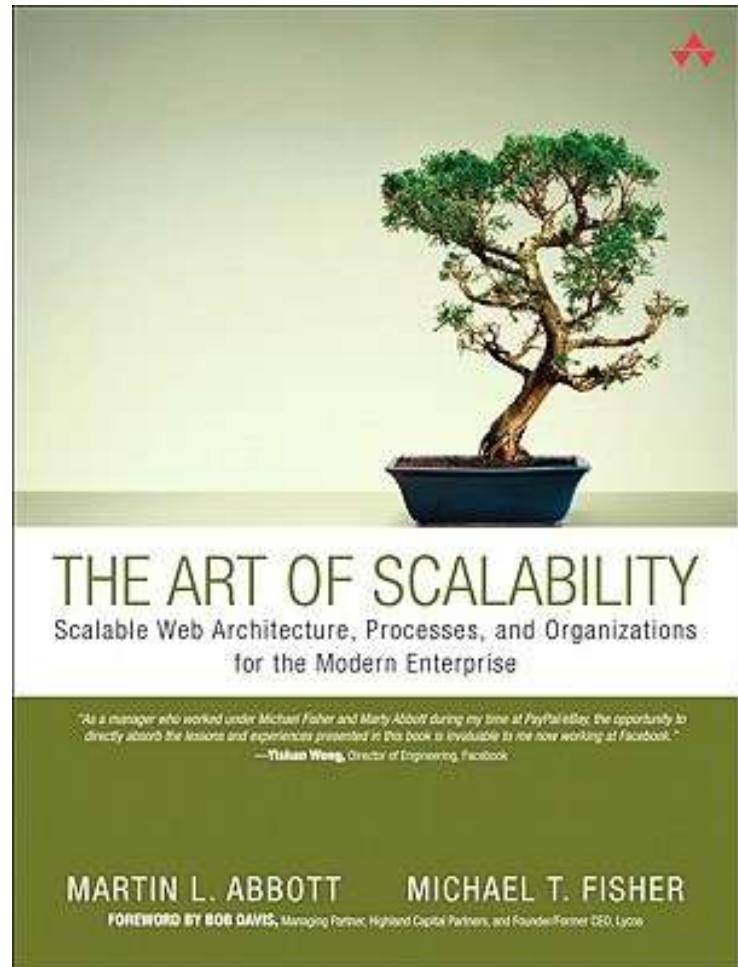
Legend

- UX – User Interaction
- BL – Business Logic
- Svc – Services
- T – Internet of Things
- APIs- Application Programming Interfaces





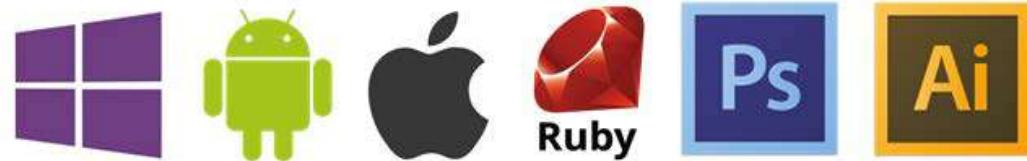
Scalability



<http://akfpartners.com/growth-blog/splitting-databases-for-scale>



Choices???



**Full Stack Project
Using
Spring Boot 2.0 and ReactJS**

@Controller, @RequestMapping

Router Functions

spring-webmvc

spring-webflux

Servlet API

HTTP / Reactive Streams

Servlet Container

Tomcat, Jetty, Netty, Undertow



SUMMARY

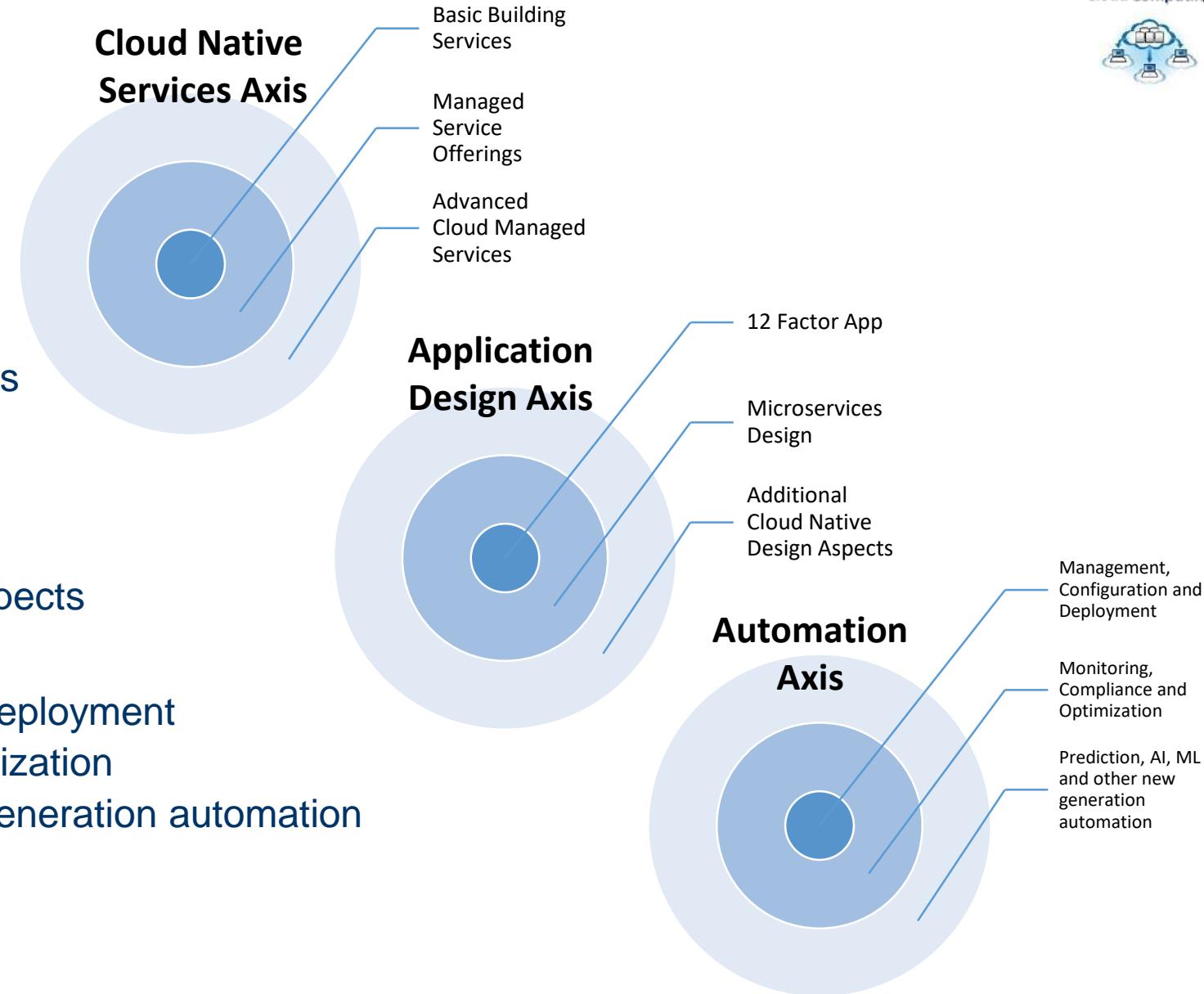


Key Design Principles

1. Cloud Native Services Axis
 1. Basic Building Services
 2. Managed Service Offerings
 3. Advanced Cloud Managed Services

2. Application Centric Design Axis
 1. 12 Factor App
 2. Microservices Design
 3. Additional Cloud Native Design Aspects

3. Automation Axis
 1. Management, Configuration and Deployment
 2. Monitoring, Compliance and Optimization
 3. Prediction, AI, ML and other new generation automation

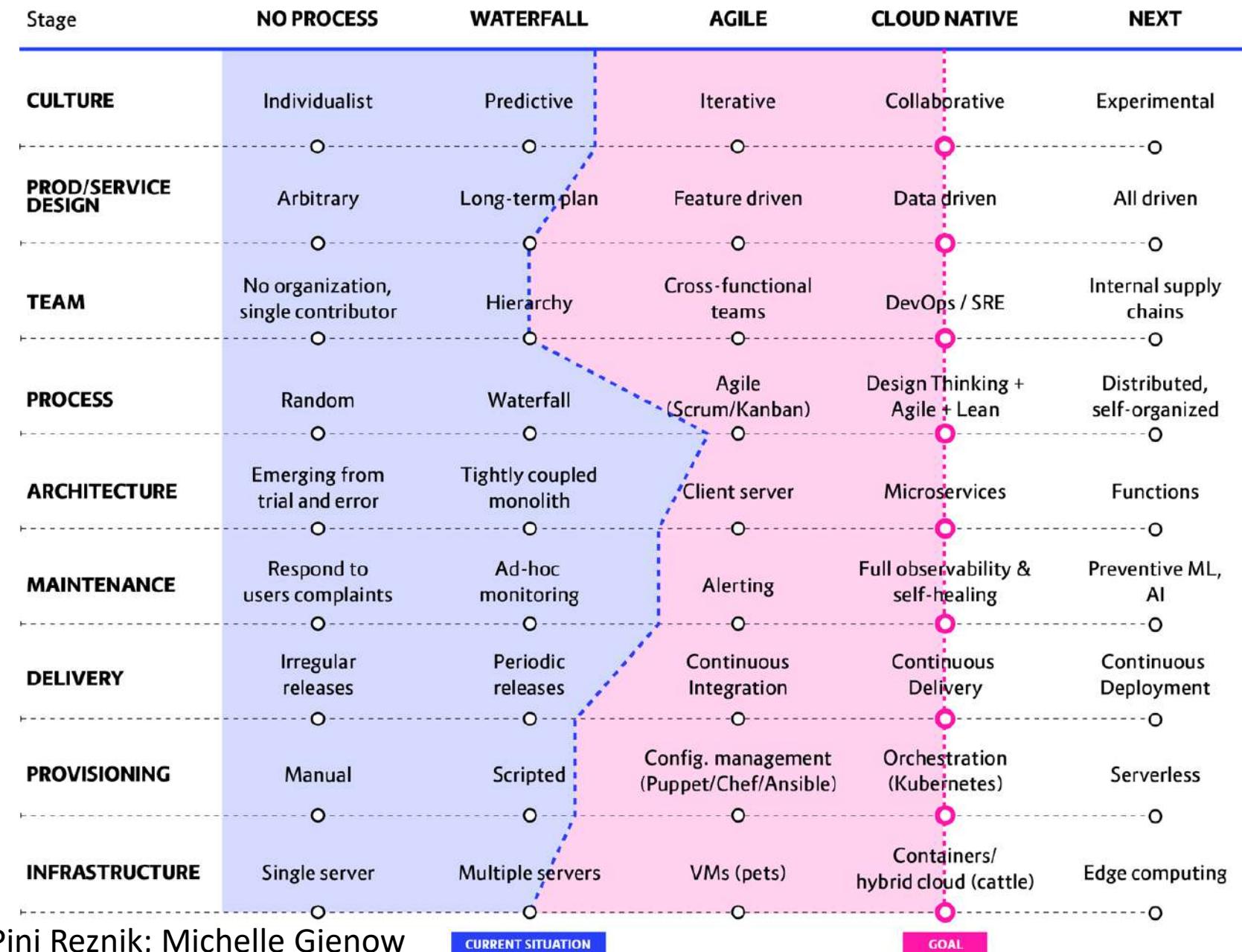


Reference: Cloud Native Architectures by Piyum Zonooz; Kamal Arora; Erik Farr; Tom Laszewski



In Essence Cloud Native

- DevSecOps
- Continuous Delivery, Continuous Upgrades, Automated Scalability
- Containers and Orchestration (Docker/Kubernetes)
- Microservices, Serverless, Functional architecture
- Rapid Recovery
- Resilience
- Multitenancy
- Open Standards
- Multicloud



Authors: Jamie Dobson; Pini Reznik; Michelle Gienow



References

■ Books

- ✓ Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS) by Michael Kavis *Published by John Wiley & Sons, 2014*
- ✓ Cloud Computing: Concepts, Technology & Architecture by Ricardo Puttini; Zaigham Mahmood; Thomas Erl *Published by Prentice Hall, 2013*
- ✓ Cloud Computing Service and Deployment Models by Al Bento; A. Aggarwal *Published by IGI Global, 2012*
- ✓ Cloud Native Java by Josh Long; Kenny Bastani *Published by O'Reilly Media, Inc., 2017*
- ✓ Cloud-Native Applications in Java by Shyam Sundar; Ajay Mahajan; Munish Kumar Gupta, Published by Packt Publishing, 2018

■ Standards

- ✓ <http://cloudcomputing.ieee.org/standards>
- ✓ <http://csc.etsi.org/>
- ✓ <http://cloud-standards.org>
- ✓ <https://www.imda.gov.sg/industry-development/infrastructure/next-gen-national-infocomm-infrastructure/cloud>
- ✓ <http://www.cloud-council.org/>
- ✓ <https://www.nist.gov/>

■ Web Sites, White papers and Vendor Sites

- ✓ Blogs by Art of Scalability, Martin Fowler, & Arun Gupta



APPENDIX 1 - CLOUD NATIVE DESIGN CONSIDERATIONS

DESIGN CHARACTERISTICS

ARCHITECTURAL PRINCIPLES

SCALABILITY, AVAILABILITY, SECURITY

MANAGABILITY

FEASIBILITY



Design Space

Architecture

Pattern

Idiom

Mechanism

Technologies

Example: Security Architecture of Applications

Example: Staged authorization

Example: Separate initialized environments

Example: Closures

Example: Functional Programming



Design Characteristics

▪ Use Pattern

- What is the highest priority in the objective of using cloud:
 - Productivity
 - Ease of migration (backward compatibility)
 - Control over technology

▪ Application Integration

- REST/SOAP
- Messaging
- Data Transformation
- Adapters
- Integration to Internal Network

▪ Data Storage

- Supported data storage technology
- Location of data (built-in, colocated, remote)

▪ Ecosystem

- Third party partners, services and applications

▪ Variety of choice

- Supported languages, type of storage, access method

▪ Portfolio

- Other services provided by the same provider

▪ Pricing

- Fixed vs. use-based
- Contract vs. Ad hoc

▪ Quality of Service

- Availability, Scalability, Disaster Recovery, SLA

▪ Security and Privacy

▪ Self-service

- Administering policy, service management and service provisioning

▪ Type of application

- Two-tier, three-tier, SOA, mobile backend, etc

▪ Elasticity

- Auto scaling capability, multitenant support

▪ Vendor viability

- Commitment to cloud, business execution



Design Guidelines

■ Goal: Separation of Concern

1. Independent of Frameworks. The architecture does not depend on the existence of some library of feature laden software. This allows us to use such frameworks as tools, rather than having to cram our system into their limited constraints.
2. Testable. The business rules can be tested without the UI, Database, Web Server, or any other external element.
3. Independent of UI. The UI can change easily, without changing the rest of the system. A Web UI could be replaced with a console UI, for example, without changing the business rules.
4. Independent of Database. We can swap out Oracle or SQL Server, for Mongo, BigTable, CouchDB, or something else. Our business rules are not bound to the database.
5. Independent of any external agency. In fact our business rules simply don't know anything at all about the outside world.

■ Organize code reflecting the **business** problems that the application solves; rather than seeing only “repositories”, “services”, or other “managers” directories.

- Organizing code: By Components, By Tools, By Layer, and By Kind.



Design Consideration - Scalability

Capacity

- Will we need to scale individual application layers and, if so, how can we achieve this without affecting availability?
- How quickly will we need to scale individual services?
- How do we add additional capacity to the application or any part of it?
- Will the application need to run at scale 24x7, or can we scale-down outside business hours or at weekends for example?

Platform / Data

- Can we work within the constraints of our chosen persistence services while working at scale (database size, transaction throughput, etc.)?
- How can we partition our data to aid scalability within persistence platform constraints (e.g. maximum database sizes, concurrent request limits, etc.)?



Design Consideration – Scalability.

Platform / Data

- How can we ensure we are making efficient and effective use of platform resources? As a rule of thumb, I generally tend towards a design based on many small instances, rather than fewer large ones.
- Can we collapse tiers to minimise internal network traffic and use of resources, whilst maintaining efficient scalability and future code maintainability?

Load

- How can we improve the design to avoid contention issues and bottlenecks? For example, can we use queues or a service bus between services in a co-operating producer, competing consumer pattern?
- Which operations could be handled asynchronously to help balance load at peak times?
- How could we use the platform features for rate-leveling (e.g. Azure Queues, Service Bus, etc.)?
- How could we use the platform features for load-balancing (e.g. Azure Traffic Manager, Load Balancer, etc.)?
[SEP]



Design Considerations - Availability

Uptime Guarantees

- What Service Level Agreements (SLA's) are the products required to meet?
- Can these SLA's be met? Do the different cloud services we are planning to use all conform to the levels required? Remember that SLA's are composite.

Replication and failover

- Which parts of the application are most at risk from failure?
- In which parts of the system would a failure have the most impact?
- Which parts of the application could benefit from redundancy and failover options?
- Will data replication services be required?
- Are we restricted to specific geopolitical areas? If so, are all the services we are planning to use available in those areas?
- How do we prevent corrupt data from being replicated?
- Will recovery from a failure put excess pressure on the system? Do we need to implement retry policies and/or a circuit-breaker?



Design Considerations – Availability.

Disaster recovery

- In the event of a catastrophic failure, how do we rebuild the system?
- How much data, if any, is it acceptable to lose in a disaster recovery scenario?
- How are we handling backups? Do we have a need for backups in addition to data-replication?
- How do we handle “in-flight” messages and queues in the event of a failure?
- Are we idempotent? Can we replay messages?
- Where are we storing our VM images? Do we have a backup?

Performance

- What are the acceptable levels of performance? How can we measure that? What happens if we drop below this level?
- Can we make any parts of the system asynchronous as an aid to performance?
- Which parts of the system are the mostly highly contended, and therefore more likely to cause performance issues?
- Are we likely to hit traffic spikes which may cause performance issues? Can we auto-scale or use queue-centric design to cover for this?



Design Considerations – Security.

- What is the local law and jurisdiction where data is held? Remember to include the countries where failover and metrics data are held too.
- Is there a requirement for federated security (e.g. ADFS with Azure Active Directory)?
- Is this to be a hybrid-cloud application? How are we securing the link between our corporate and cloud networks?
- How do we control access to the administration portal of the cloud provider?
- How do we restrict access to databases, etc. from other services (e.g. IP Address white-lists, etc.)?
- How do we handle regular password changes?
- How does service-decoupling and multi-tenancy affect security?
- How we will deal with operating system and vendor security patches and updates?



Design Considerations – Manageability

Monitoring

- How are we planning to monitor the application?
- Are we going to use off-the-shelf monitoring services or write our own?
- Where will the monitoring/metrics data be physically stored? Is this in line with data protection policies?
- How much data will our plans for monitoring produce?
- How will we access metrics data and logs? Do we have a plan to make this data useable as volumes increase?
- Is there a requirement for auditing as well as logging?
- Can we afford to lose some metrics/logging/audit data (i.e. can we use an asynchronous design to “fire and forget” to help aid performance)?
- Will we need to alter the level of monitoring at runtime?
- Do we need automated exception reporting?



Design Considerations – Manageability.

Deployment

- How do we automate the deployment?
- How do we patch and/or redeploy without disrupting the live system?
Can we still meet the SLA's?
- How do we check that a deployment was successful?
- How do we roll-back an unsuccessful deployment?
- How many environments will we need (e.g. development, test, staging, production) and how will deploy to each of them?
- Will each environment need separate data storage?
- Will each environment need to be available 24x7?



Design Considerations – Feasibility.

- Can the SLA's ever be met (i.e. is there a cloud service provider that can give the uptime guarantees that we need to provide to our customer)?
- Do we have the necessary skills and experience in-house to design and build cloud applications?
- Can we build the application to the design we have within budgetary constraints and a timeframe that makes sense to the business?
- How much will we need to spend on operational costs (cloud providers often have very complex pricing structures)?
- What can we sensibly reduce (scope, SLAs, resilience)?
- What trade-offs are we willing to accept?



Cloud Native Solution Design

DOCKER

A SIMPLE INTRODUCTION

Suria R Asai

suria@nus.edu.sg

Institute of Systems Science

National University of Singapore

© 2009-23 NUS. The contents contained in this document may not be reproduced in any form or by any means, without the written permission of ISS, NUS, other than for the purpose for which it has been supplied.

Total slides: 21.



SCIENCE BEHIND DOCKERS

DOCKER CONTAINERS

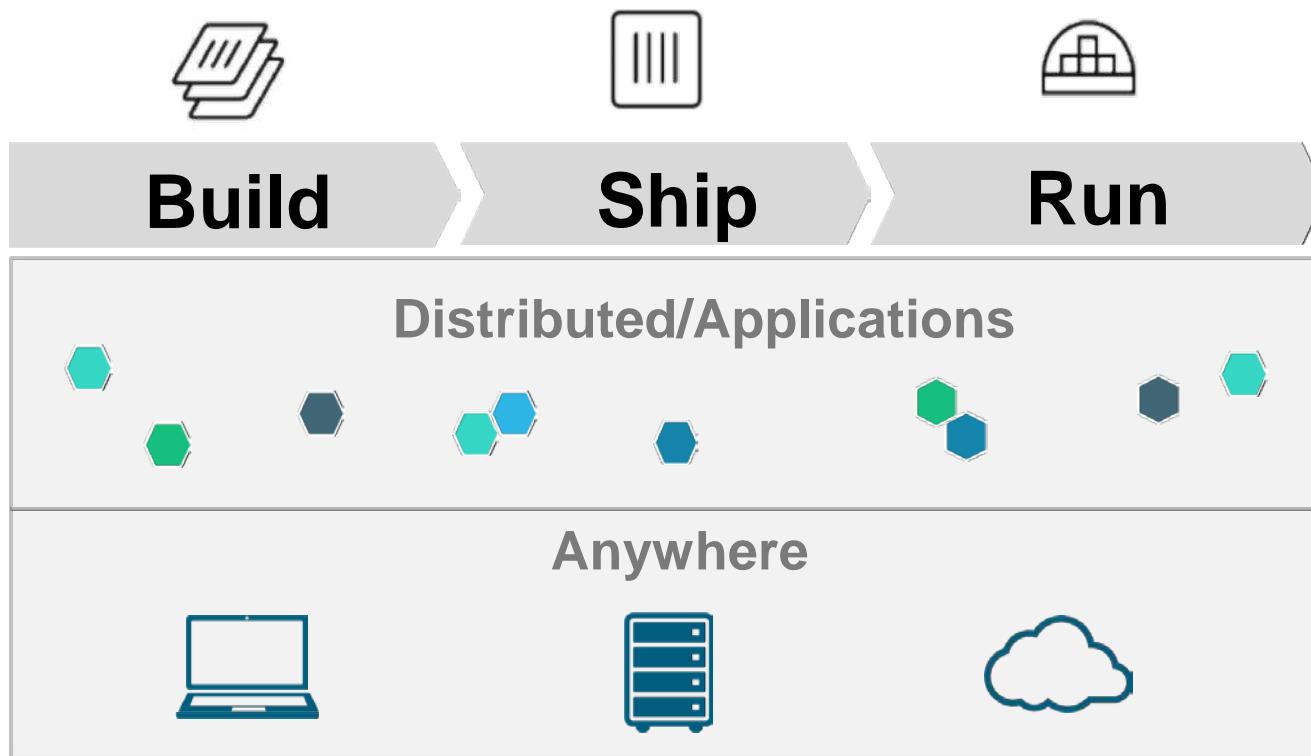
INSTALLING DOCKERS

DOCKER INTERFACE



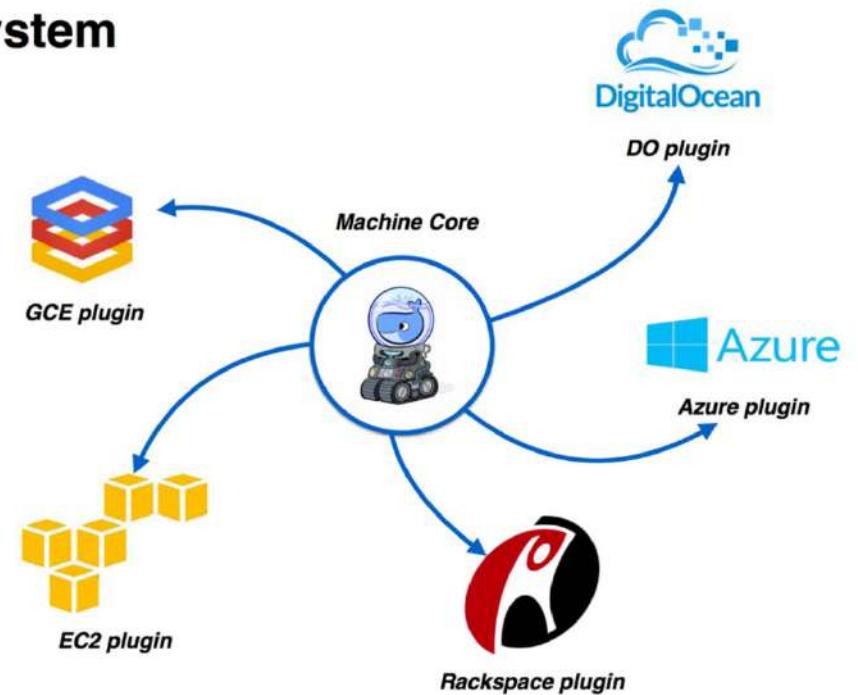
Docker Pattern

Docker Mission



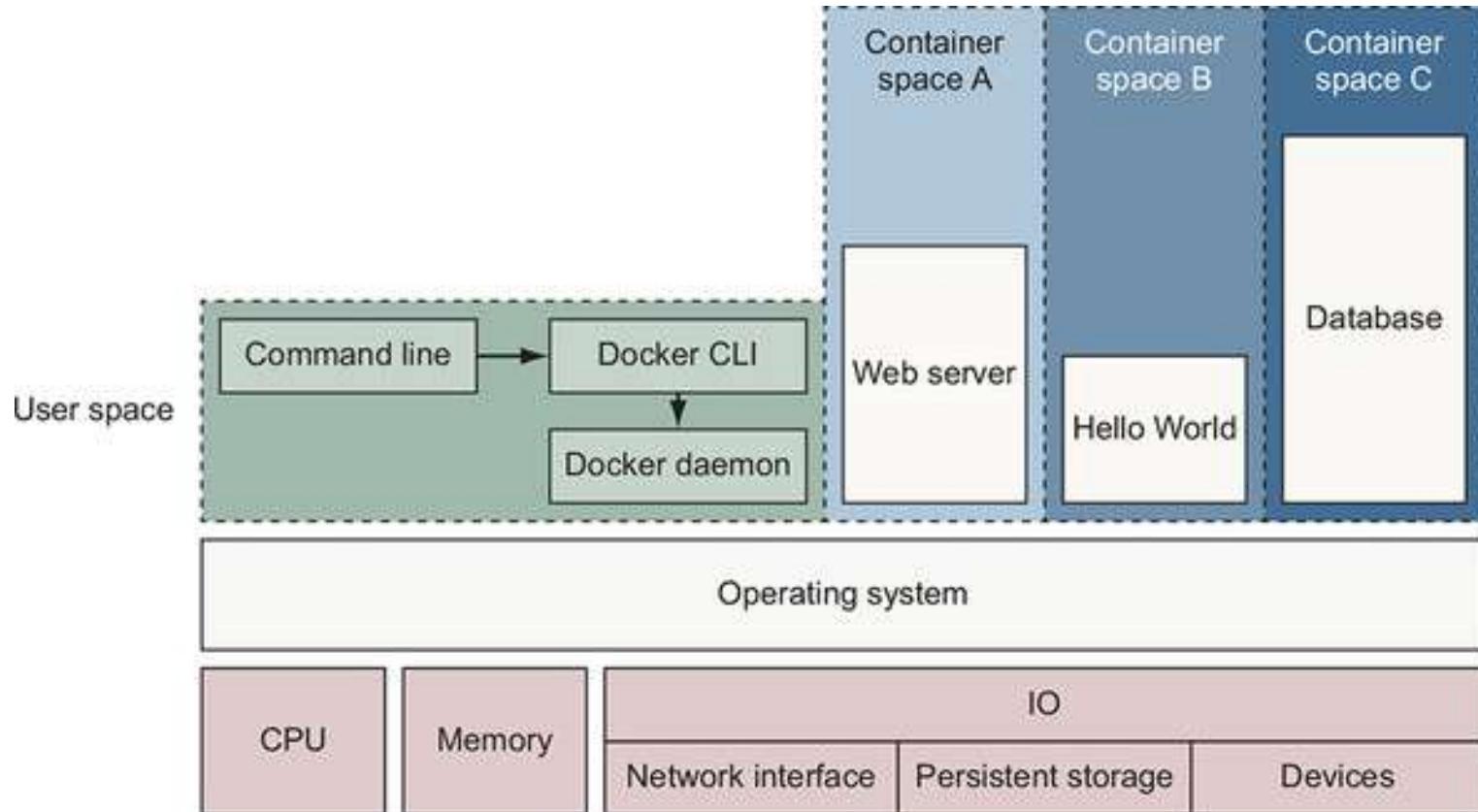
Docker Machine Providers

System





Docker Features



- ✓ **PID namespace**— Process identifiers and capabilities
- ✓ **UTS namespace**— Host and domain name
- ✓ **MNT namespace**— Filesystem access and structure
- ✓ **IPC namespace**— Process communication over shared memory
- ✓ **NET namespace**— Network access and structure
- ✓ **USR namespace**— User names and identifiers
- ✓ `chroot syscall`— Controls the location of the filesystem root
- ✓ **cgroups**— Resource protection
- ✓ **CAP drop**— Operating system feature restrictions
- ✓ **Security modules**— Mandatory access controls



When to use Docker? - 1

■ Small Medium Business

- Reduce responsibility for system administration
- Reduce costs of managing application and platform versions
- Achieve enterprise-class high availability and disaster recovery
- Achieve state-of-the-art data security protection
- Gain productivity and access to modern platform technology
- Reduce IT capital costs
- Obtain access to applications and services that is only available in the cloud
- Focus IT resources on development of business solution

■ Enterprise

- Obtain services that is only available in the cloud
- Assist and manage LOB “shadow-IT” initiatives
- Establish neutral intermediary between multiple divisions of organizations
- Extend and integrate SaaS application services
- Reduce time to start a software project
- Improve development productivity and cost to produce custom applications
- To support time-boxed projects at a lower cost
- To convert capital costs to operational cost
- To manage applications with unpredictable/fluctuating demand



When to use Docker? – 2.

■ ISV

- Porting their on-premises applications as cloud services
- Eliminate costs of supporting multiple hardware and software platforms and manage versioning
- Improve reliability, performance, scalability and availability
- Reduce the cost of maintenance of their applications (version control, distribution and control)
- Achieve high resource utilization using the elasticity capability of the cloud platform
- Improve competitiveness by achieving lower cost
- Reduce financial risk by avoiding big upfront capital investment

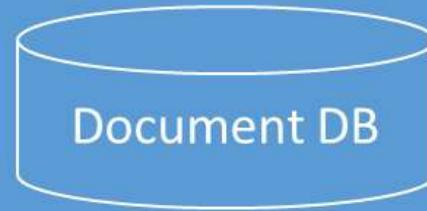


Data architecture for microservices

User Session



Product Info



Shopping Cart



Personalization



Log Data

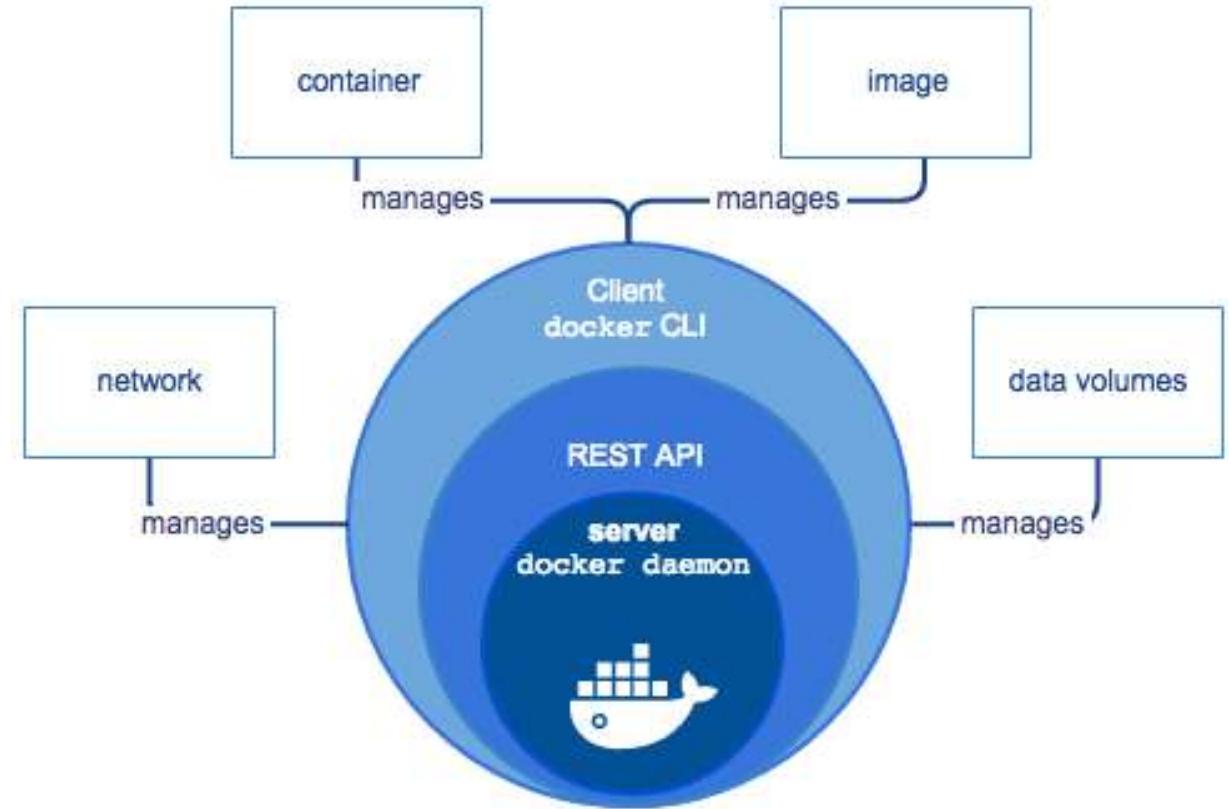
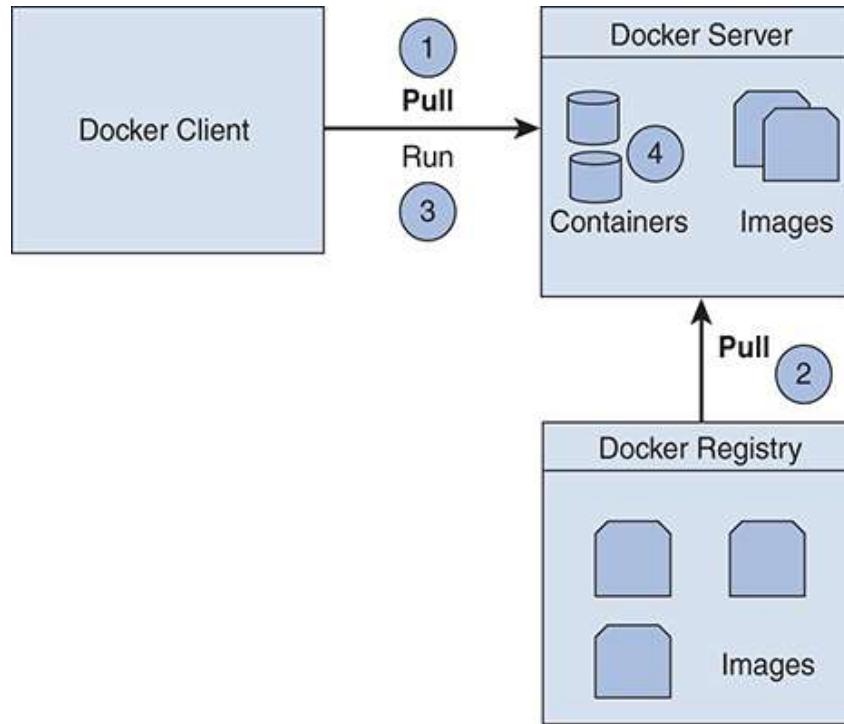


Financial Data



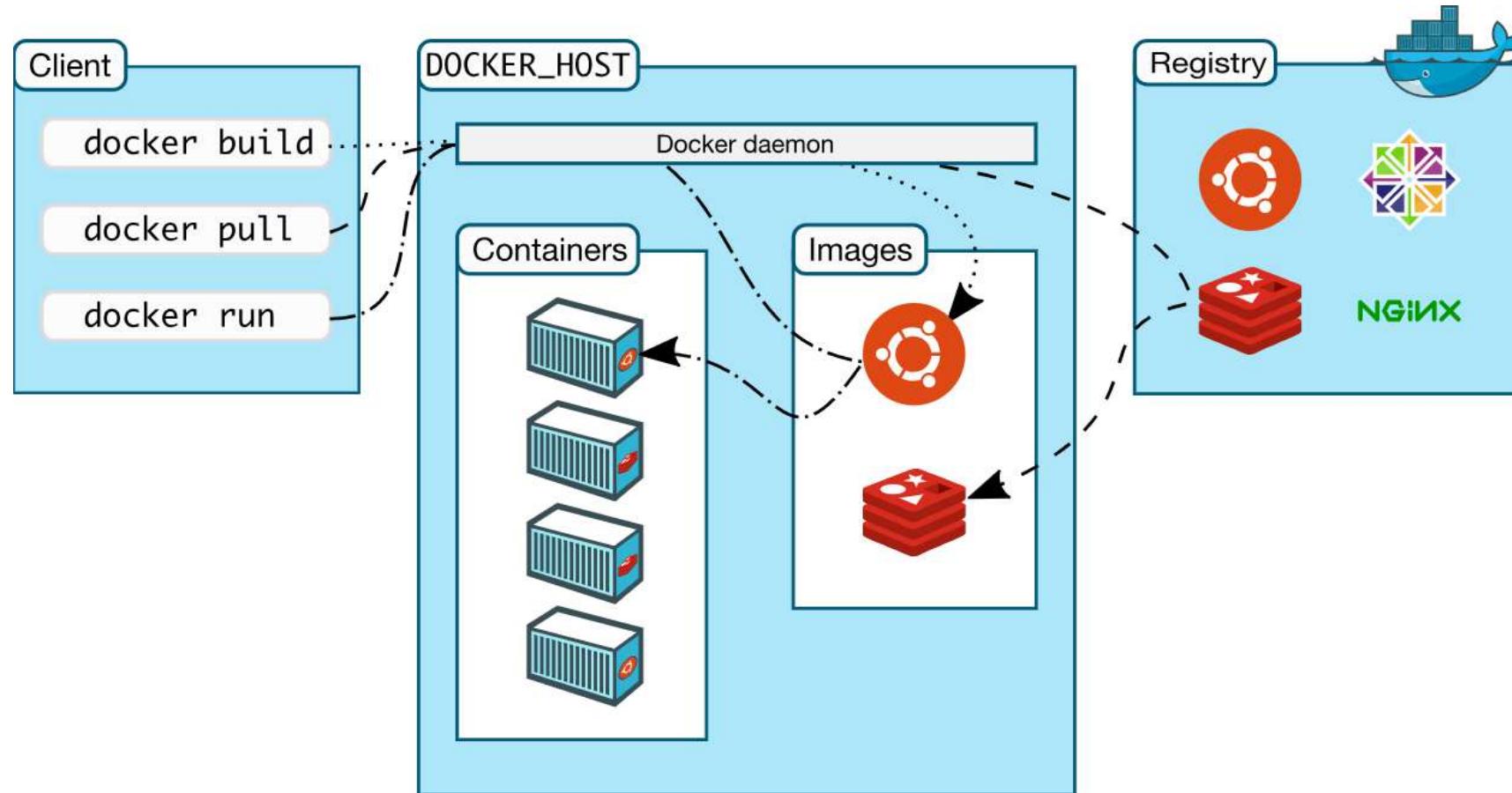


Docker Architecture and Components - 1





Docker Architecture and Components - 2





Docker Architecture and Components – 3

- Docker server or daemon.
 - This resides on the host system and manages all the containers running on the host machine.
- Docker container.
 - This is a standalone virtual system that contains the running process, all the files, dependencies, process space, and ports that are required to run the application.
 - Since every container has all the ports available, we do the mapping at the Docker level.
- Docker client.
 - A user interface or a command-line interface is used to communicate with the Docker daemon.
- Docker images.
 - These are read-only template files of a Docker container that we can move around and distribute.
 - Unlike with virtual machines, these files can be version controlled.

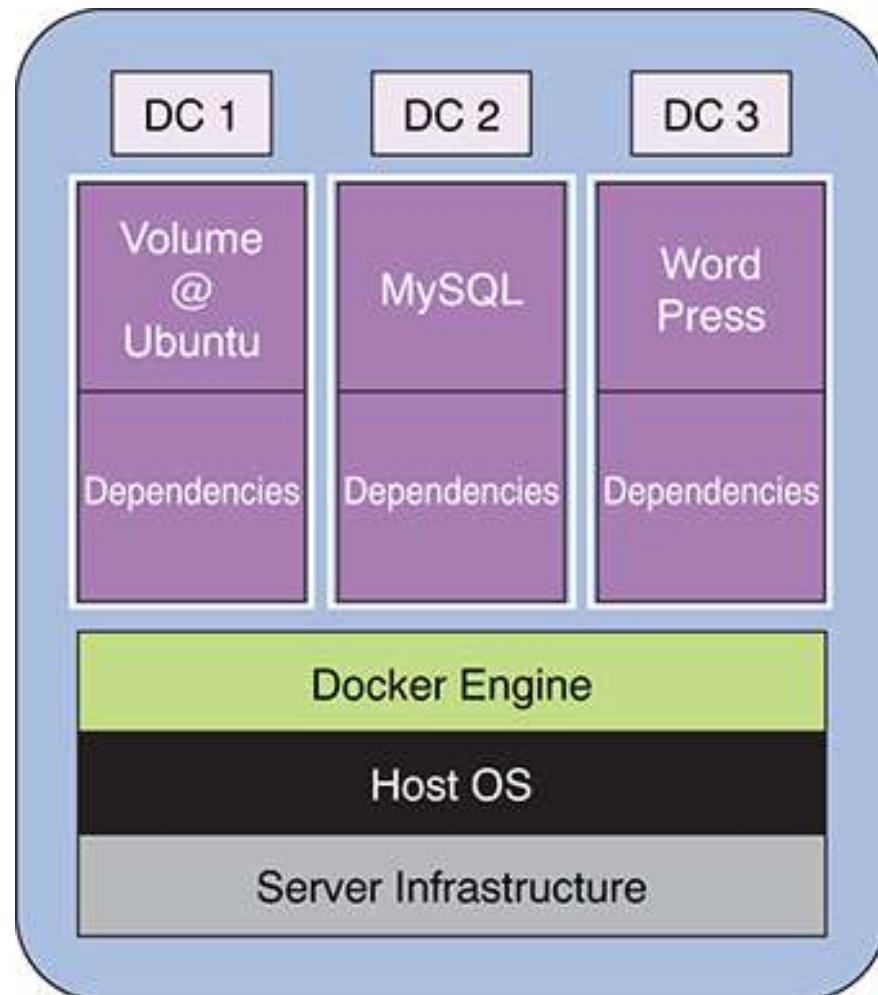


Docker Architecture and Components – 4.

- Docker registry.
 - This is a repository for sharing and storing Docker container images.
 - A well-known registry is Docker Hub (just like GitHub) that allows you to pull or push the container images with public or private access.
- Dockerfile.
 - This is a very simple text file where we specify the commands to build Docker images.
 - It allows us to set up instructions to install software; set up environment variables, working directories, and ENTRYPOINT; and add new code using Docker commands.
- Docker Machine.
 - Docker Machine allows us spin up Docker hosts on local machine or within public or private cloud, including on various service providers such as Amazon and Microsoft Azure.
 - It also provides a way to manage the hosts through Docker Machine commands—start, stop, inspect, and more.
- Docker Swarm/Engine/Compose.
 - **Swarm** provides out-of-the-box clustering capability wherein a pool of Docker nodes act as one large Docker host.
 - **Engine** is a separate tool, which can install using Docker Machine or, manually, by pulling the Swarm image.
 - **Docker Compose** is an application will have multiple components and consequently will be running multiple containers.



Example Docker Codes



■ Code Example

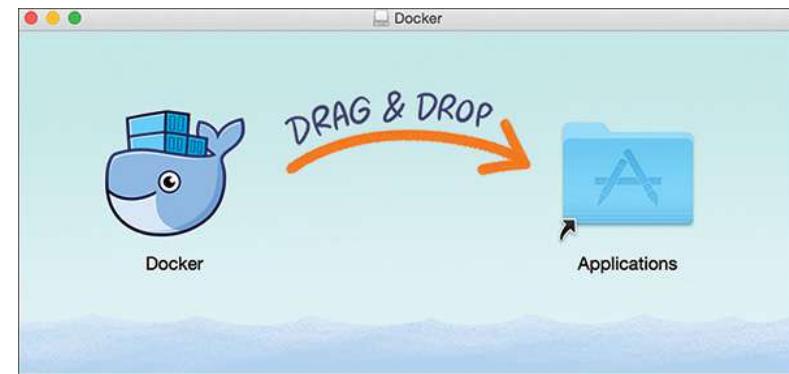
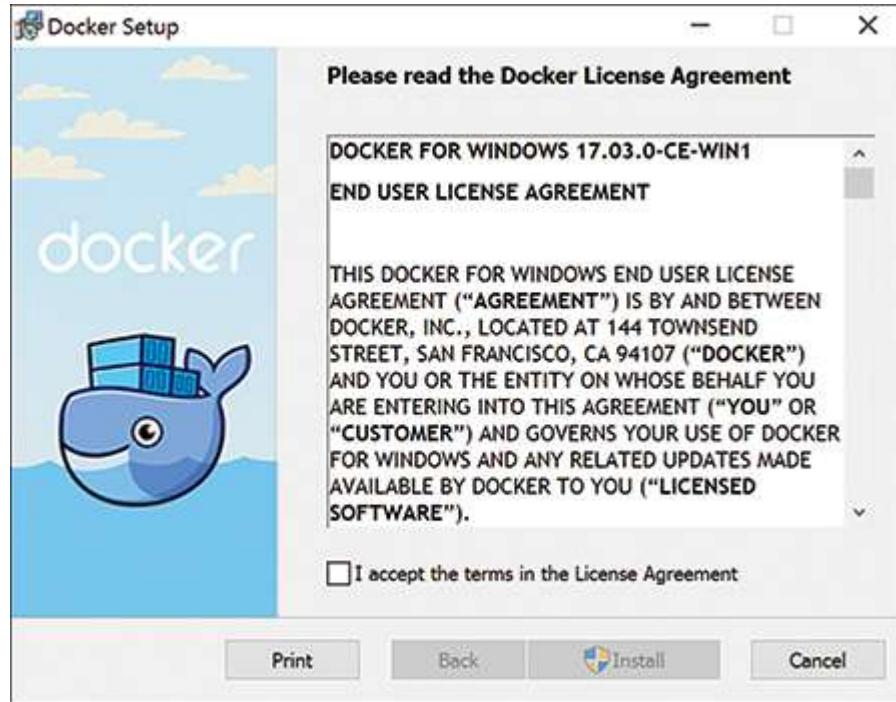
```
docker create --name mysql_data_container
-v /var/lib/mysql Ubuntu
```

```
docker run --volumes-from
mysql_data_container -v
/var/lib/mysql:/var/lib/mysql -
-e MYSQL_USER=mysql -e MYSQL_PASSWORD=mysql
-e MYSQL_DATABASE=test -
-e MYSQL_ROOT_PASSWORD=test -it -p 3306:3306
-d mysql
```

```
docker run -d --name wordpress --link
mysql:mysql wordpress
```



Docker Installation



```
pkocher@pkocher-dev:~$ sudo apt-get update
Ign http://us.archive.ubuntu.com trusty InRelease
Ign http://extras.ubuntu.com trusty InRelease
Get: 1 http://us.archive.ubuntu.com trusty-updates InRelease [65.9 kB]
Get: 2 http://extras.ubuntu.com trusty Release.gpg [72 B]
Ign http://dl.google.com stable InRelease
Get: 3 http://security.ubuntu.com trusty-security InRelease [65.9 kB]
Hit http://extras.ubuntu.com trusty Release
Get: 4 http://dl.google.com stable Release.gpg [916 B]
Get: 5 http://us.archive.ubuntu.com trusty-backports InRelease [65.9 kB]
Hit http://extras.ubuntu.com trusty/main Sources
Get: 6 http://security.ubuntu.com trusty-security/multiverse amd64 Packages [4,139 B]
Get: 7 http://dl.google.com stable Release [1,189 B]
Hit http://extras.ubuntu.com trusty/main amd64 Packages
Hit http://us.archive.ubuntu.com trusty Release.gpg
Hit http://extras.ubuntu.com trusty/main i386 Packages
Get: 8 http://dl.google.com stable/main amd64 Packages [1,427 B]
Get: 9 http://us.archive.ubuntu.com trusty-updates/main Sources [393 kB]
Get: 10 http://security.ubuntu.com trusty-security/universe amd64 Packages [154 kB]
Get: 11 http://us.archive.ubuntu.com trusty-updates/restricted Sources [5,911 B]
Get: 12 http://security.ubuntu.com trusty/main amd64 Packages [593 kB]
```



Docker Interface - 1

▪ Docker Lifecycle

- docker **create** creates a container but does not start it.
- docker **rename** allows the container to be renamed.
- docker **run** creates and starts a container in one operation.
- docker **rm** deletes a container.
- docker **update** updates a container's resource limits.

▪ Docker Starting and Stopping

- docker **start** starts a container so it is running.
- docker **stop** stops a running container.
- docker **restart** stops and starts a container.
- docker **pause** pauses a running container, "freezing" it in place.
- docker **unpause** will unpause a running container.
- docker **wait** blocks until running container stops.
- docker **kill** sends a SIGKILL to a running container.
- docker **attach** will connect to a running container.



Docker Interface - 2

▪ Docker Info

- docker **ps** shows running containers.
- docker **logs** gets logs from container. (You can use a custom log driver, but logs is only available for json-file and journald in 1.10).
- docker **inspect** looks at all the info on a container (including IP address).
- docker **events** gets events from container.
- docker **port** shows public facing port of container.
- docker **top** shows running processes in container.
- docker **stats** shows containers' resource usage statistics.
- docker **diff** shows changed files in the container's FS.

▪ Docker Import / Export

- docker **cp** copies files or folders between a container and the local filesystem.
- docker **export** turns container filesystem into tarball archive stream to STDOUT.

▪ Docker Executing Commands

- docker **exec** to execute a command in container.



Docker Interface - 3

▪ Image Lifecycle

- docker **images** shows all images.
- docker **import** creates an image from a tarball.
- docker **build** creates image from Dockerfile.
- docker **commit** creates image from a container, pausing it temporarily if it is running.
- docker **rmi** removes an image.
- docker **load** loads an image from a tar archive as STDIN, including images and tags (as of 0.7).
- docker **save** saves an image to a tar archive stream to STDOUT with all parent layers, tags & versions (as of 0.7).

▪ Image Info

- docker **history** shows history of image.
- docker **tag** tags an image to a name (local or registry).



Docker Interface – 4.

▪ Network Lifecycle

- docker network **create**
- docker network **rm**

▪ Network Info

- docker network **ls**
- docker network **inspect**

▪ Network Connection

- docker network **connect**
- docker network **disconnect**

▪ Docker Registry and Repository

- docker **login** to login to a registry.
- docker **logout** to logout from a registry.
- docker **search** searches registry for image.
- docker **pull** pulls an image from registry to local machine.
- docker **push** pushes an image to the registry from local machine.



Docker Characteristics

▪ Shared Resources

- Resources are shared between independent application instances while maintaining logical isolation between them

▪ Multitenancy

- Logical application instances are isolated and managed in a shared environment
- Includes tracking, resource allocation, security, SLA enforcement, billing, logging, backup, error recovery

▪ Elasticity

- Allocate an optimal amount of computing resources to an application instance based on changing demand

▪ Self-service

- Most services require some degree of self-service from the service subscriber

▪ Continuous Versioning

- Deploying of version upgrade and patches to the PaaS platform should not disrupt the operation of running application instances

▪ Use based billing framework

- Subscribers are charged based on their usage of the PaaS resources
- More advanced PaaS will be able to track usages by their subtenants (tenants of their ISV who use PaaS to deploy their solutions), and help the ISV to establish subtenant billing processes



Docker Performance Considerations

▪ High Productivity

- Focus on projects looking for ease of use, fast results, high productivity and low cost of entry

▪ High Control

- Focus on projects that is looking for maximum control of its technology context

▪ High Backward Compatibility

- Target migration projects
- Provides minimal disruption to the established code, skill and practices

▪ In-memory computing

- a.k.a. in-memory data grid, distributed caching
- Allow some data to be contained in memory to improve performance of the application

▪ High Availability

- Uninterrupted operation of application instance despite hardware and software failure

▪ Auto-scaling

- Provide optimal resource allocation for the application instance based on its demand and the defined policy

▪ Security

- Application instances and their data are protected from other application sharing the same cloud environment

▪ Data and transaction integrity

- Consistency of application data need to be protected in the case when accessing shared database instance at the same time



SUMMARY



Concluding Remarks

- Docker takes a logistical approach to solving common software problems and **simplifies** your experience with **installing, running, publishing, and removing software**.
 - It's a command-line program, an engine background process, and a set of remote services.
 - It's integrated with community tools provided by Docker Inc.
- The **container abstraction** is at the core of its logistical approach.
 - Working with containers instead of software creates a consistent interface and enables the development of more sophisticated tools.
 - Containers help keep your computers tidy because software inside containers can't interact with anything outside those containers, and no shared dependencies can be formed.
 - Because Docker is available and supported on Linux, macOS, and Windows, most software packaged in Docker images can be used on any computer.
- Docker doesn't provide container technology; it hides the complexity of working directly with the container software and turns best practices into reasonable defaults.
 - Docker works with the greater container ecosystem; that ecosystem is rich with tooling that solves new and higher-level problems.



Cloud Native Solution Design

KUBERNETES

A SIMPLE INTRODUCTION

Suria R Asai

suria@nus.edu.sg

Institute of Systems Science

National University of Singapore

© 2009-23 NUS. The contents contained in this document may not be reproduced in any form or by any means, without the written permission of ISS, NUS, other than for the purpose for which it has been supplied.

Total slides: 33.



Why Kubernetes?

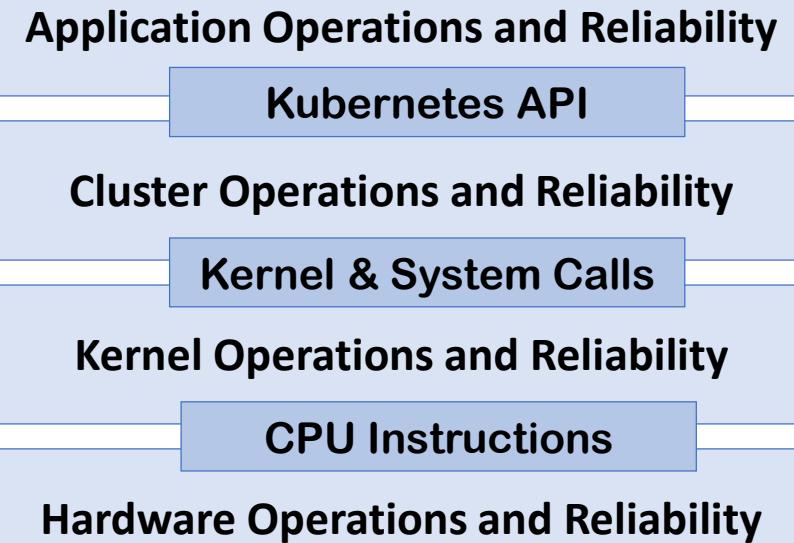
- **Developer Velocity**
 - Architecture Immutability
 - Declarative Instructions
 - Online Self-healing
 - Shared reusable libraries and tools
- **Scaling (Both software and team productivity)**
 - Pods
 - Services
 - Namespaces
 - Ingress
- **Infrastructure Abstraction**
- **Efficiency**
- **Cloud Native Ecosystem**

A Container is a self-contained ready to run application. It needs a runtime to establish communication between kernel and container.

Container Images are binary packages that encapsulates all of the files necessary to run a program inside of a container.

Container Registry stores pre-existing images in standard formats (example Open Container Image OCI).

A **Kubernetes** platform is useful for creating, deploying and managing distributed applications.





Container Deployment Era

- Containers have become popular because they provide extra benefits, such as:
 - **Agile** application creation and deployment: increased ease and efficiency of container image creation compared to VM image use.
 - Continuous development, integration, and deployment: provides for reliable and frequent container image build and deployment with quick and easy rollbacks (due to image immutability).
 - **Dev and Ops** separation of concerns: create application container images at build/release time rather than deployment time, thereby decoupling applications from infrastructure.
 - **Observability** not only surfaces OS-level information and metrics, but also application health and other signals.
 - Environmental consistency across development, testing, and production: Runs the same on a laptop as it does in the cloud.
 - **Cloud** and **OS** distribution portability: Runs on Ubuntu, RHEL, CoreOS, on-premises, on major public clouds, and anywhere else.
 - Application-centric management: Raises the level of abstraction from running an OS on virtual hardware to running an application on an OS using logical resources.
 - Loosely coupled, distributed, elastic, liberated micro-services: applications are broken into smaller, independent pieces and can be deployed and managed dynamically – not a monolithic stack running on one big single-purpose machine.
 - Resource **isolation**: predictable application performance.
 - Resource **utilization**: high efficiency and density.



What is Kubernetes?

- Its core functionality is its ability to **schedule workloads** in containers across your infrastructure
 - Mounting **storage** systems
 - Distributing **secrets**
 - Checking application **health**
 - **Replicating** application instances
 - Using horizontal pod **autoscaling**
 - **Naming and discovering**
 - **Balancing** loads
 - **Rolling** updates
 - **Monitoring** resources
 - Accessing and ingesting **logs**
 - **Debugging** applications
 - Providing **authentication** and **authorization**

What Kubernetes is not? Kubernetes is not a platform as a service (PaaS). It doesn't dictate many of the important aspects of your desired system; instead, it leaves them up to you or to other systems built on top of Kubernetes, such as Deis, OpenShift, and Eldarion.

- *Kubernetes doesn't require a specific application type or framework*
- *Kubernetes doesn't require a specific programming language*
- *Kubernetes doesn't provide databases or message queues*
- *Kubernetes doesn't distinguish apps from services*
- *Kubernetes doesn't have a click-to-deploy service marketplace*
- *Kubernetes allows users to choose their own logging, monitoring, and alerting systems*



Containers

A container is made up of two different pieces, and a group of associated features. A container includes:

- A container image
- A set of operating system concepts that isolates a running process or processes

- Kubernetes' job is to take a group of machines that provide resources, like CPU, memory, and disk, and transform them into a container-oriented API that developers can use to deploy their containers.
 - The orchestration system is about scheduling containers to machines.
 - Kubernetes orchestrator knows how to heal those containers if they fail.
 - Kubernetes can manage liveness checks and load-balanced service (readiness checks).
 - Kubernetes can perform zero-downtime rollouts and that manage configurations, persistent volumes, secrets, and much more.



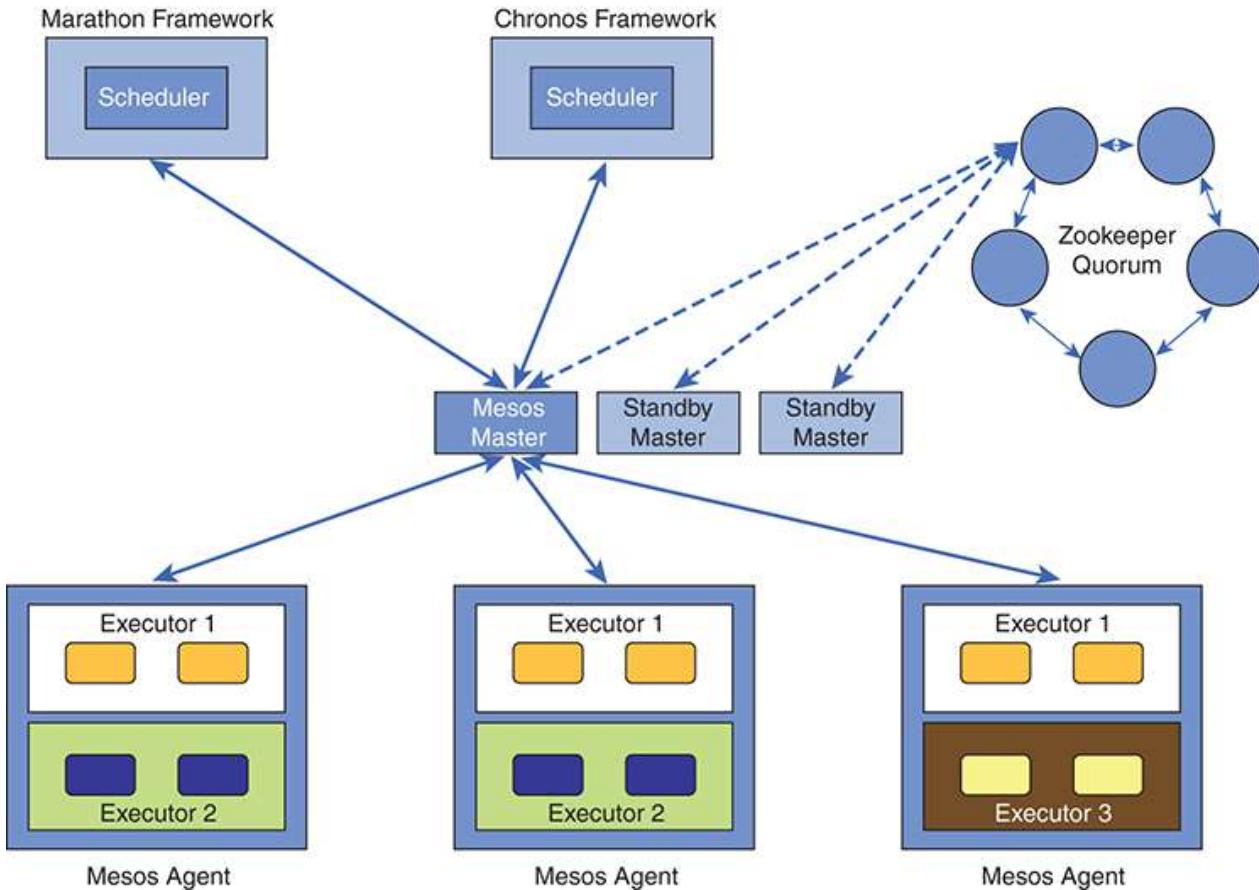
Orchestration Technologies



Kubernetes is Greek for pilot or helmsman (the person holding the ship's steering wheel). Pronounce it more like Koo-ber-netties.



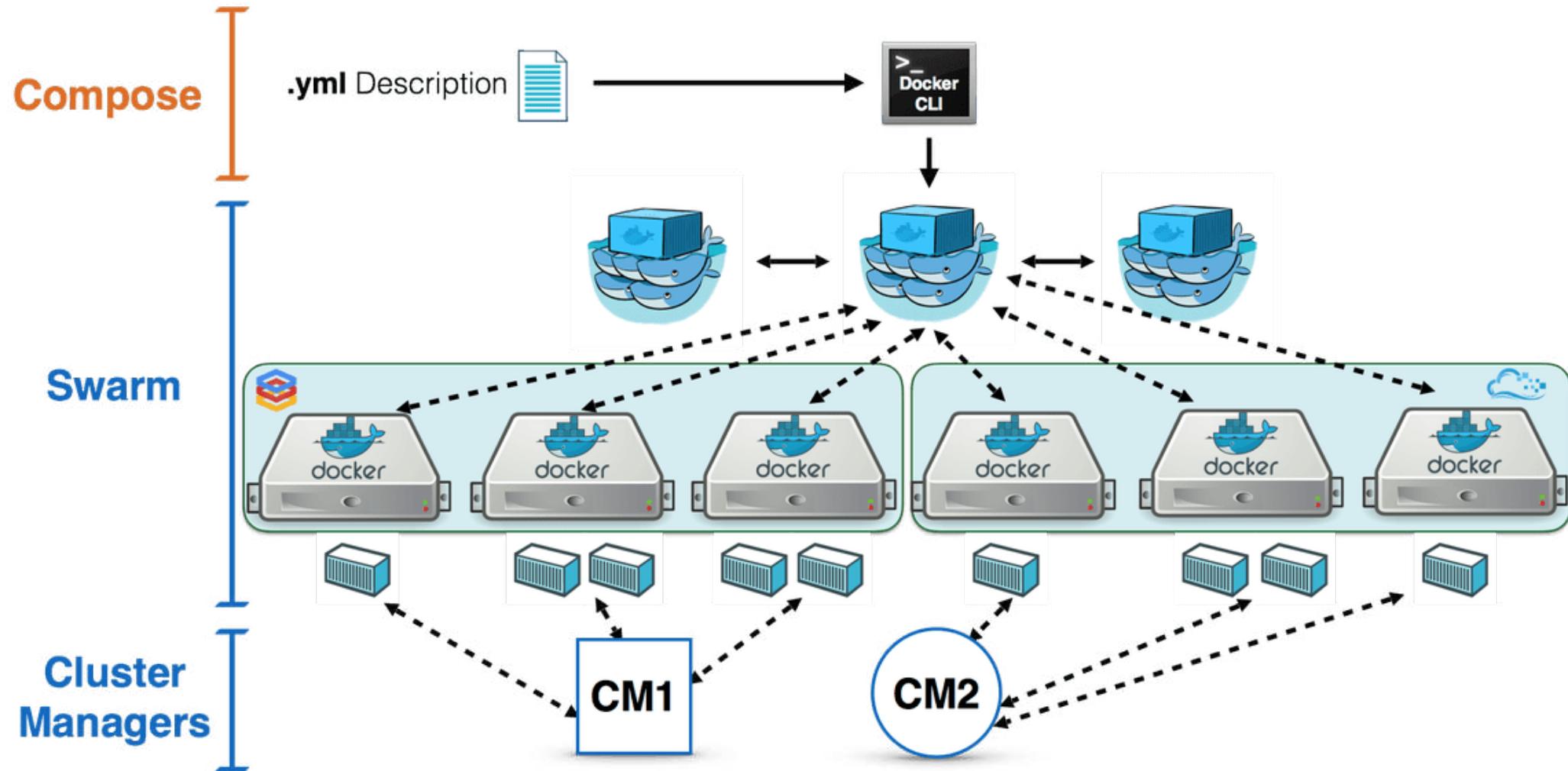
APACHE MESOS AND MARATHON



Apache Mesos is an open source containers orchestration framework that is proven to work well in large-scale production environments. Mesos is like an operating system kernel that manages resources in a cluster of machines. It works in a master/slave-based architecture.

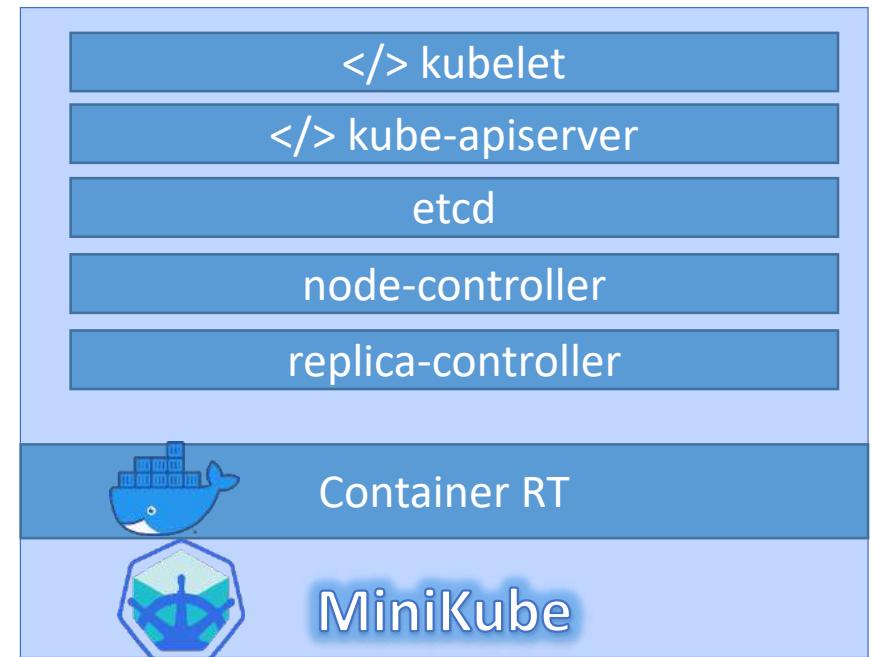
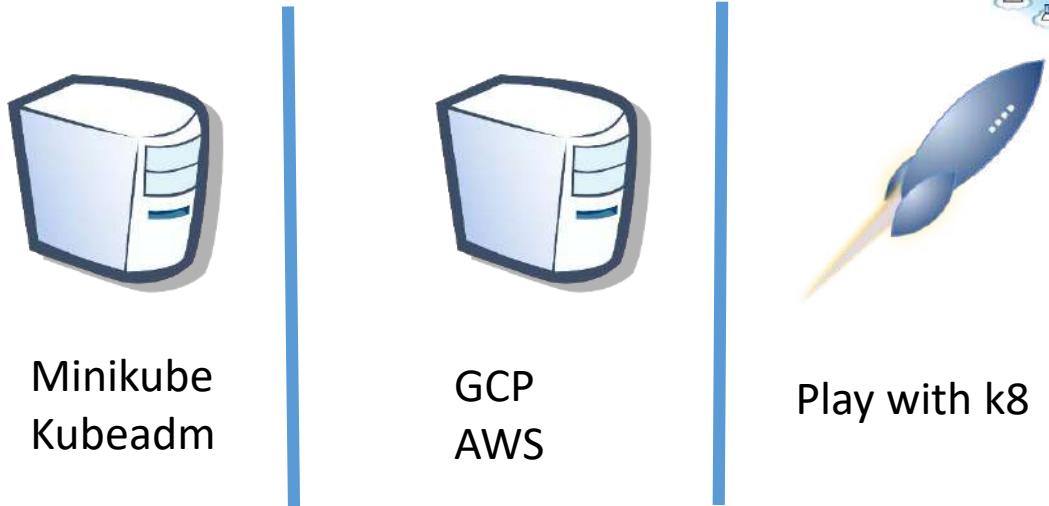
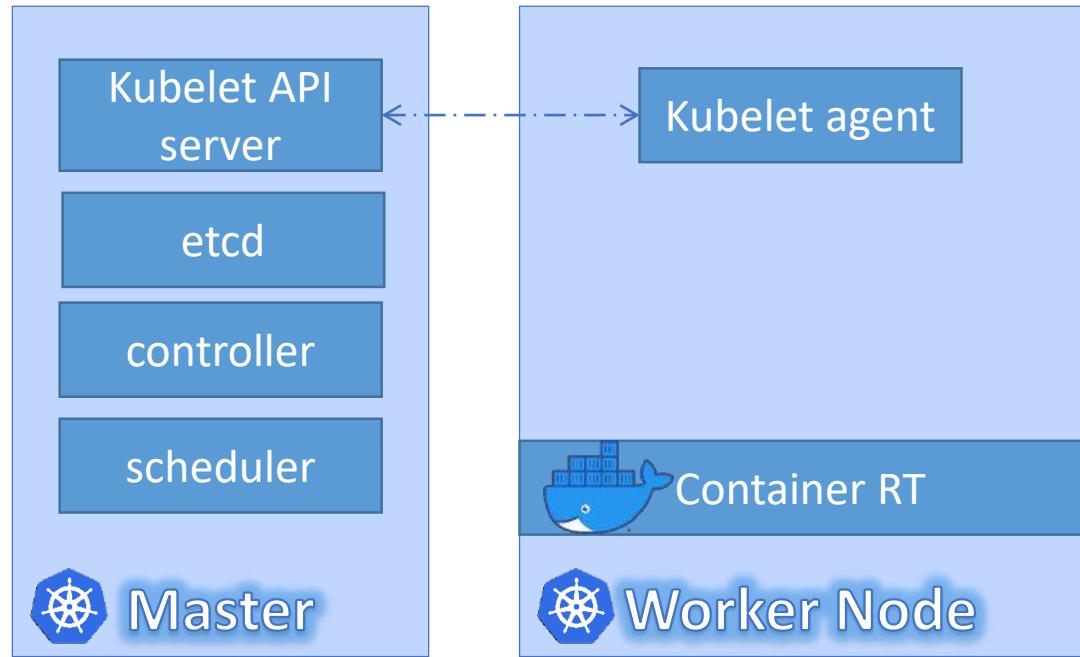


Docker Swarm



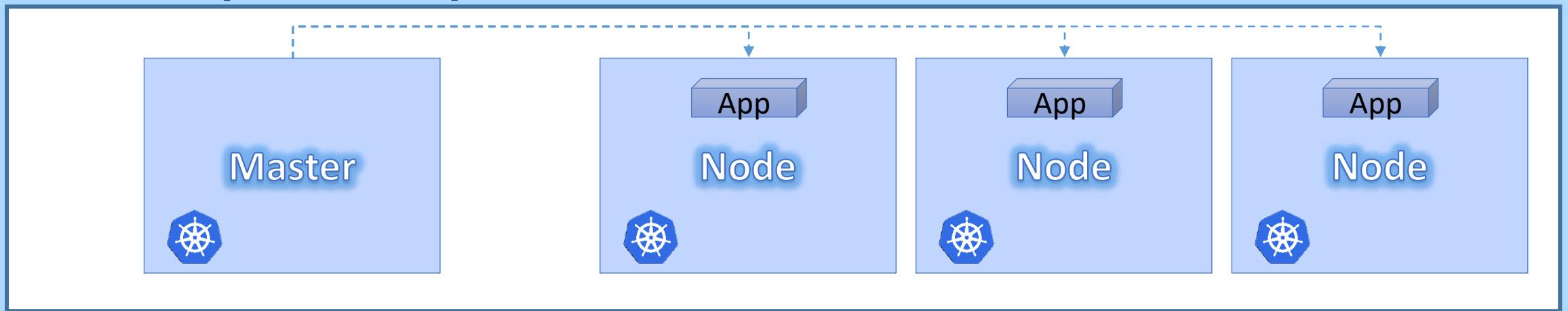


The K8s Cluster





Nodes (Minions), Cluster, Master



Pods



- Container
- Shared IP
- Shared Storage
- Shared resources
- Shared lifecycle

Replicaset/ Deployment



- Desired state
- Replica
- Pod Templates
- Health checks
- Resource Image

Service



- Grouping of Pods
- Stable Virtual IP
- DNS name space
- Acts as one unit

Persistent Volume



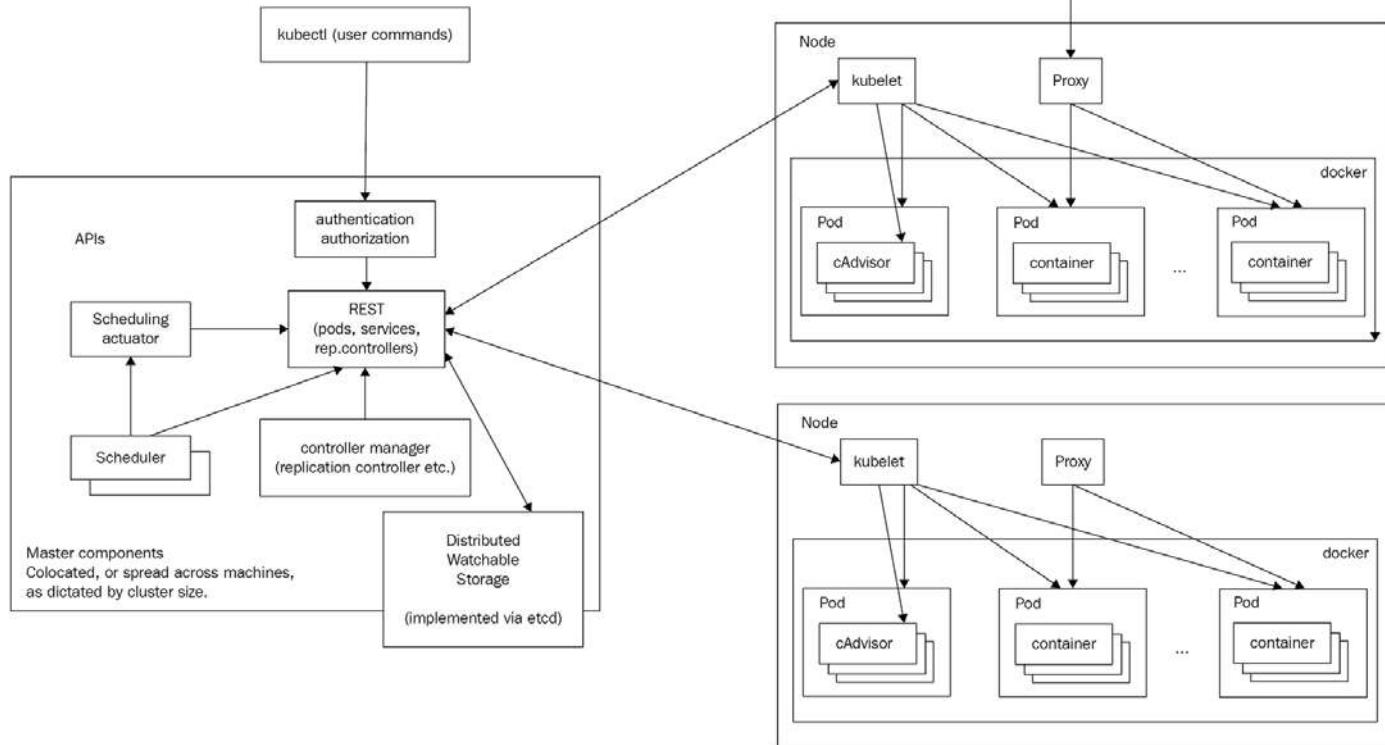
- Network available storage
- PV and PVC

Label

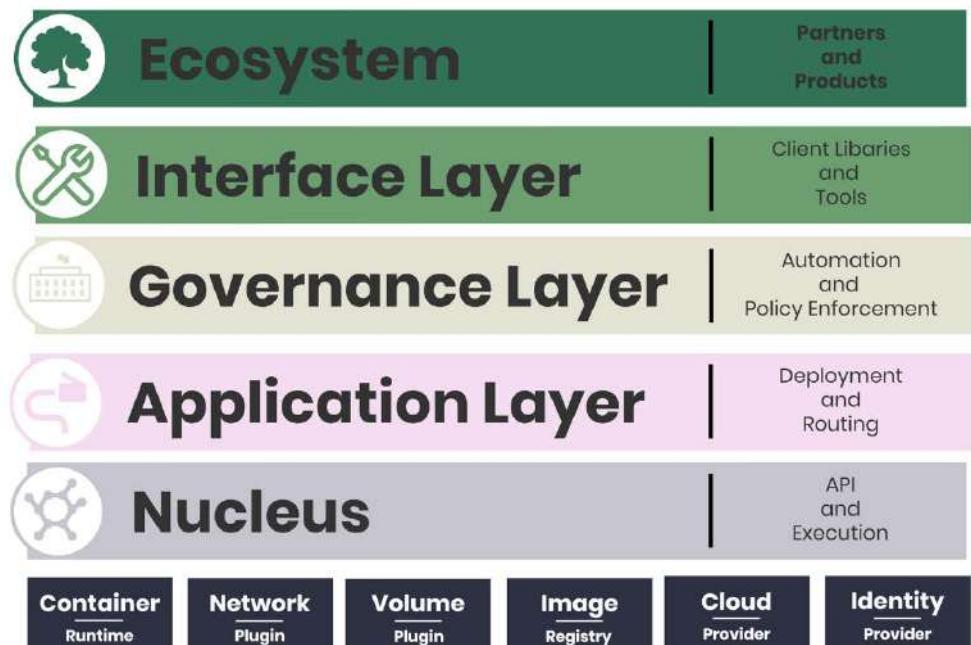
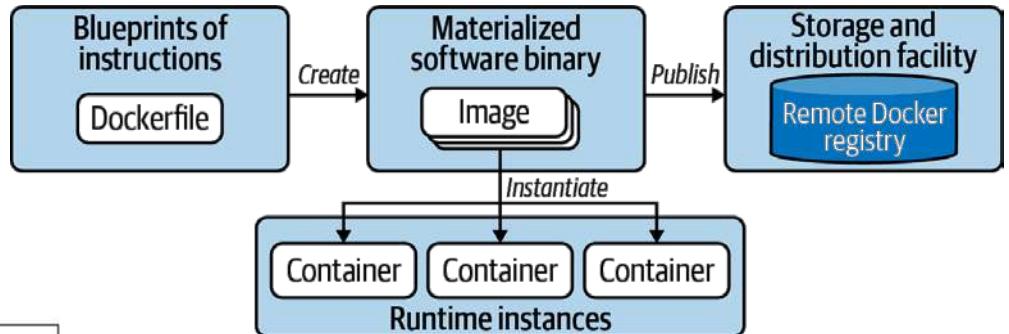
- Key Value Pairs
- Refers to Kubernetes objects



A Kubernetes Cluster



Containerization Process



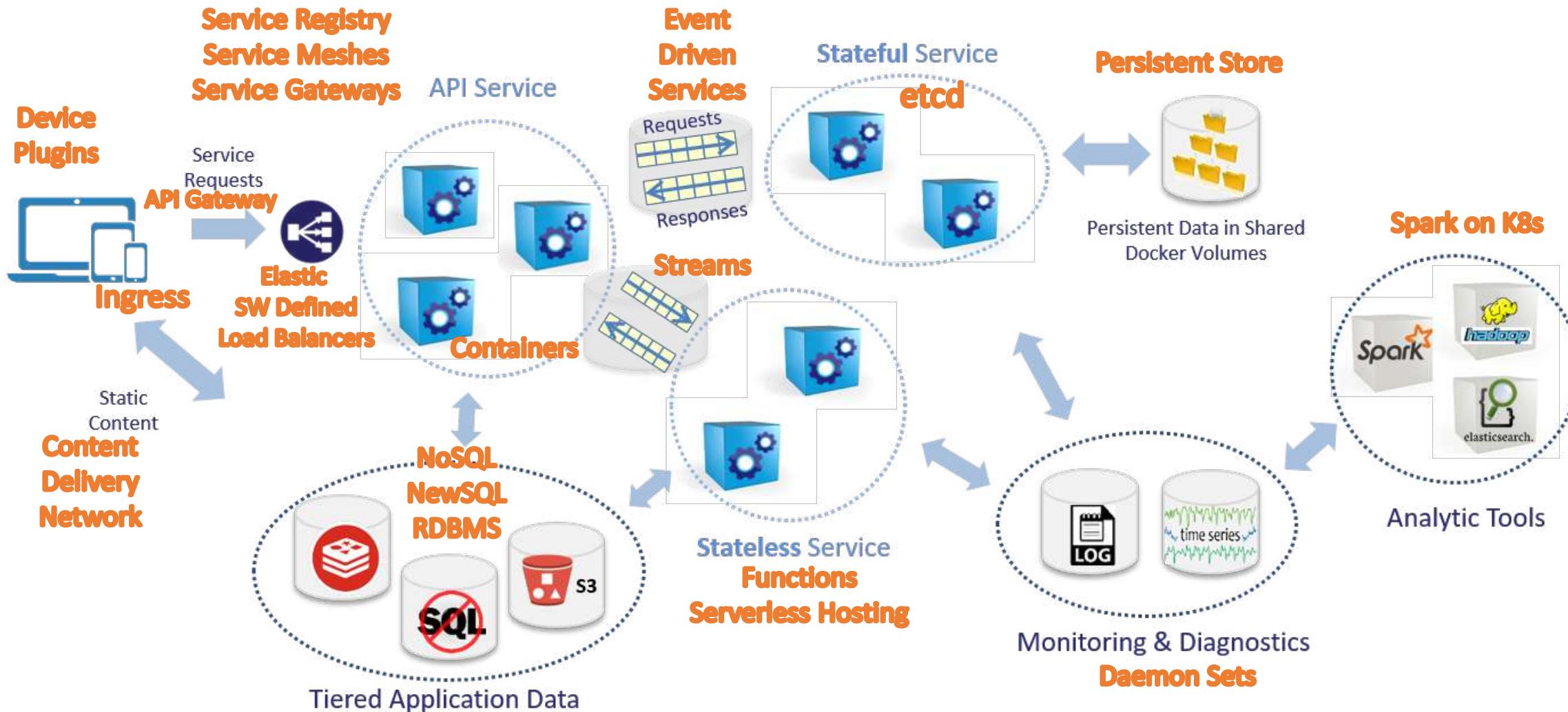


Kubernetes as Distributed System





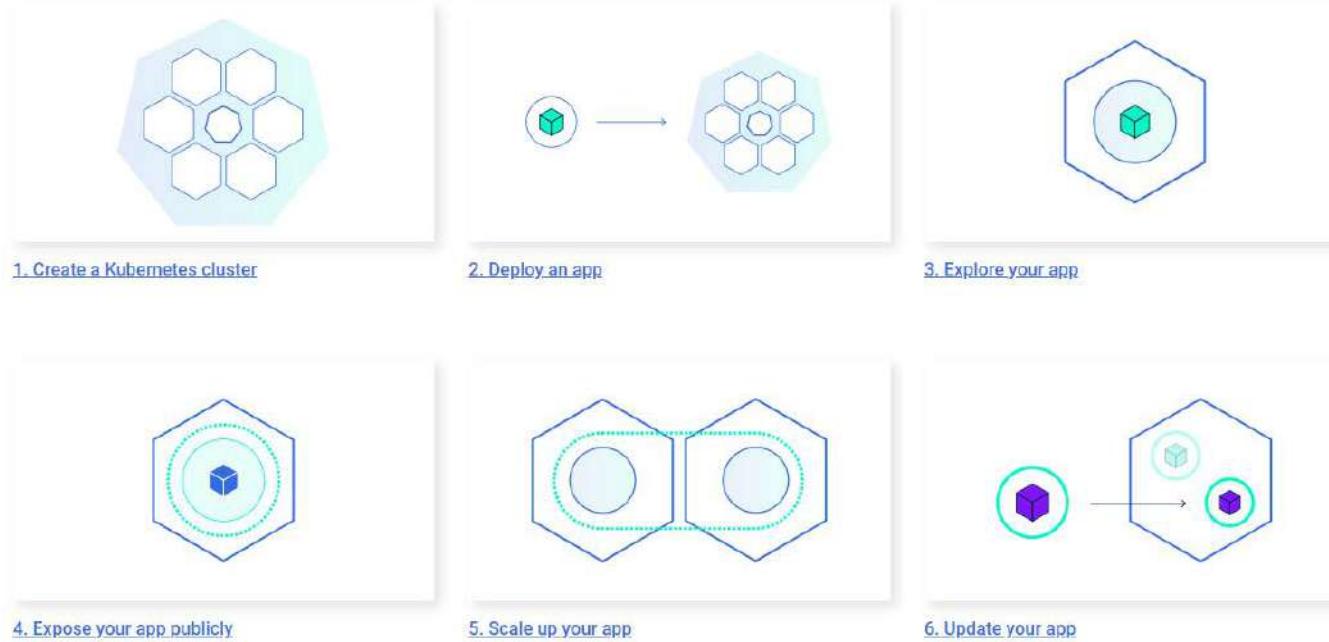
Cloud Native Design on Kubernetes





Capabilities

Kubernetes Basics Modules



▪ Portability

- How do I guarantee maximum portability across diverse network environments while taking advantage of unique network characteristics?

▪ Service Discovery

- How do I know where services are living as they are scaled up and down?

▪ Load Balancing

- How do I share load across services as services themselves are brought up and scaled?

▪ Security

- How do I segment to prevent the wrong containers from accessing each other?
- How do I guarantee that a container with application and cluster control traffic is secure?

▪ Performance

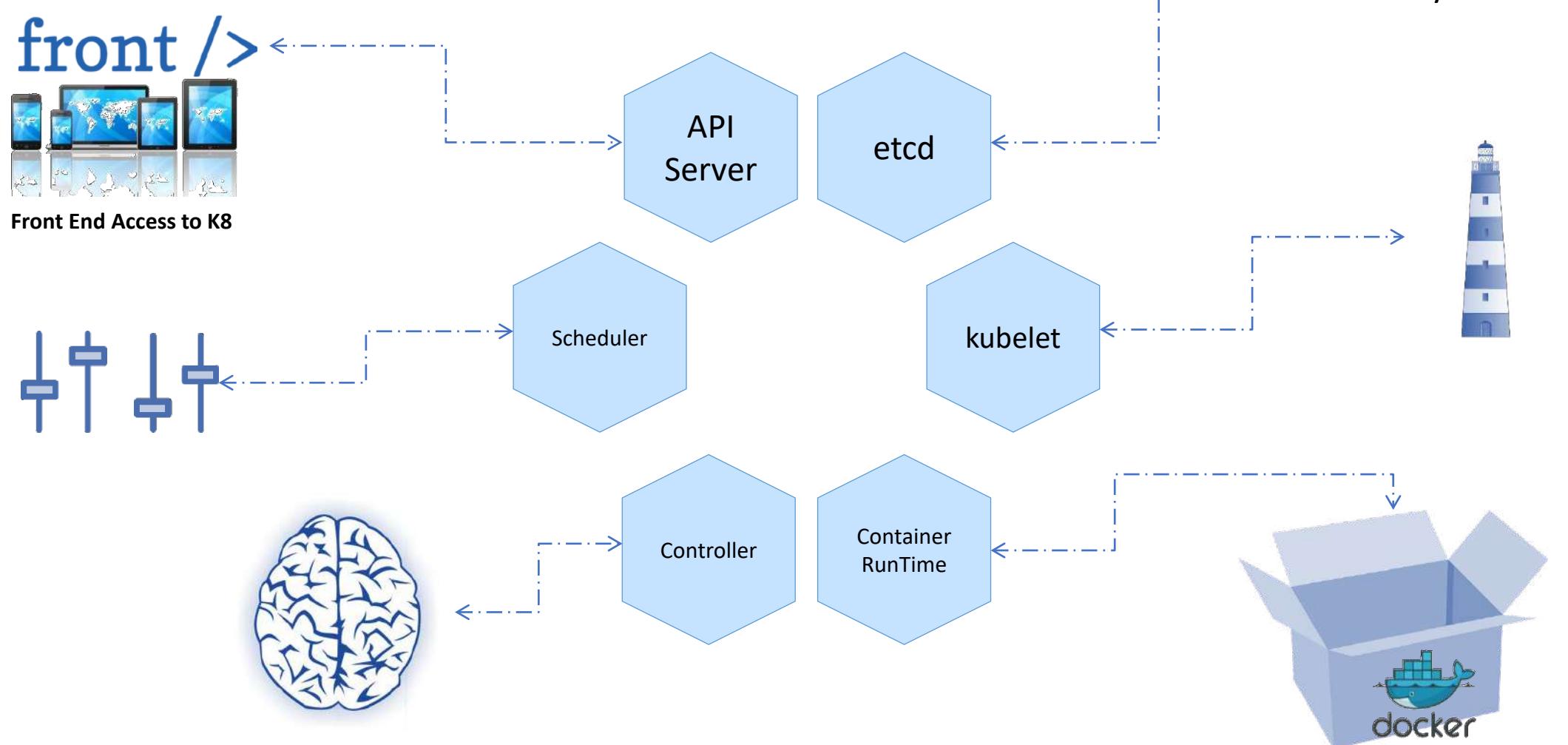
- How do I provide advanced network services while minimizing latency and maximizing bandwidth?

▪ Scalability

- How do I ensure that none of these characteristics are sacrificed when scaling applications across many hosts?



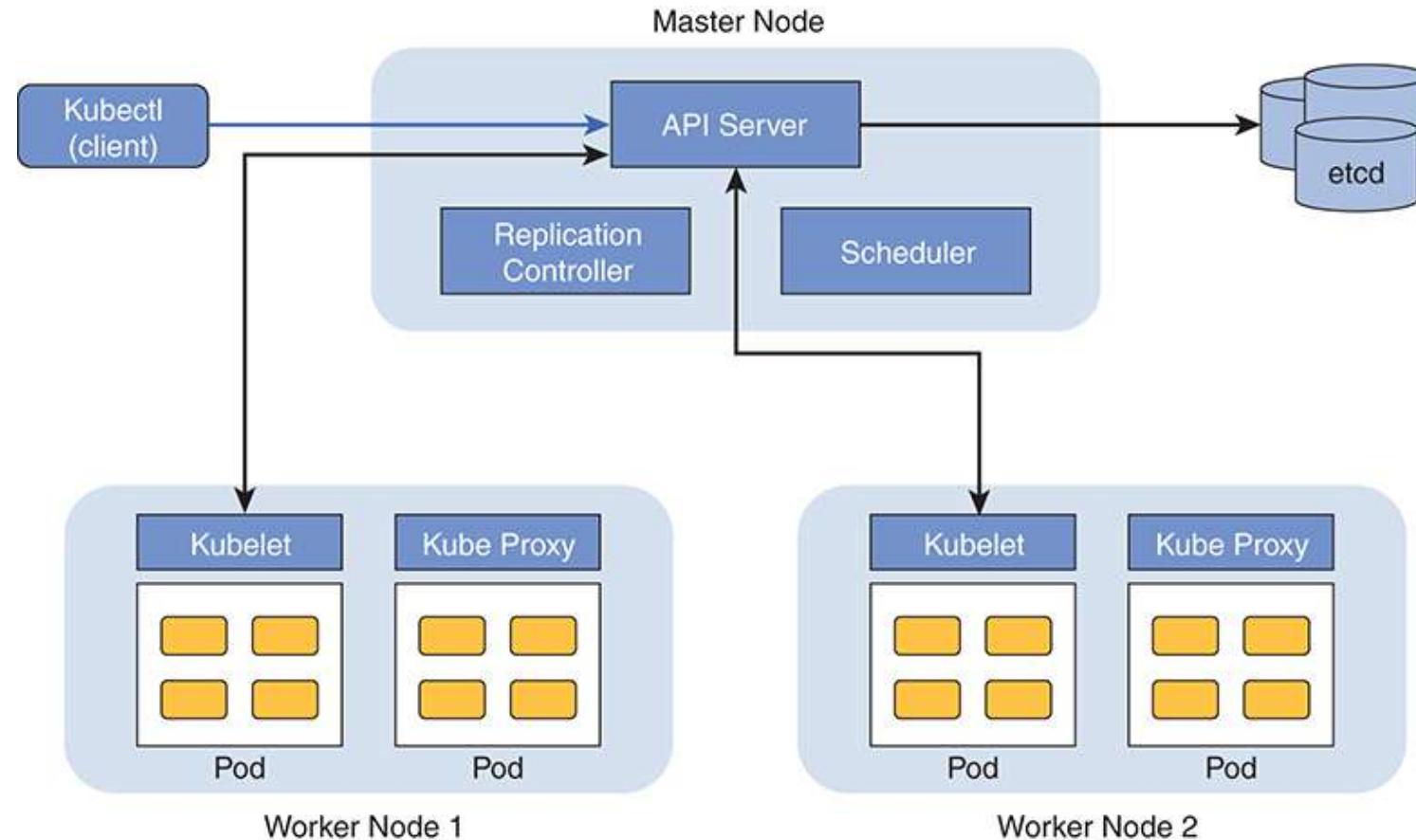
Kubernetes Components



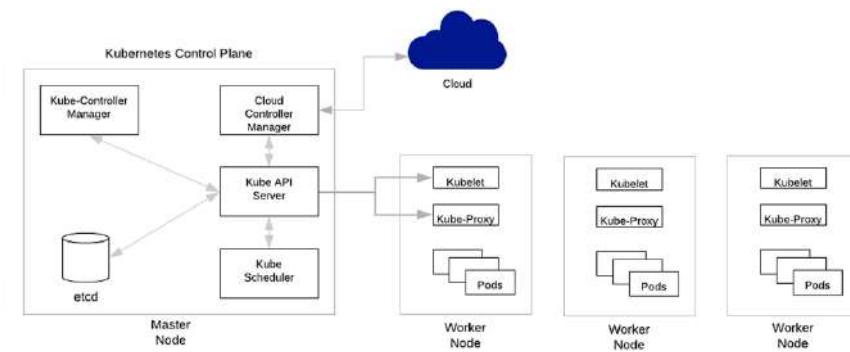


Example: Kubernetes Cluster

Kubernetes is one of the orchestration engines that helps you run your containerized applications where and when you want by providing the resources and capabilities they need.



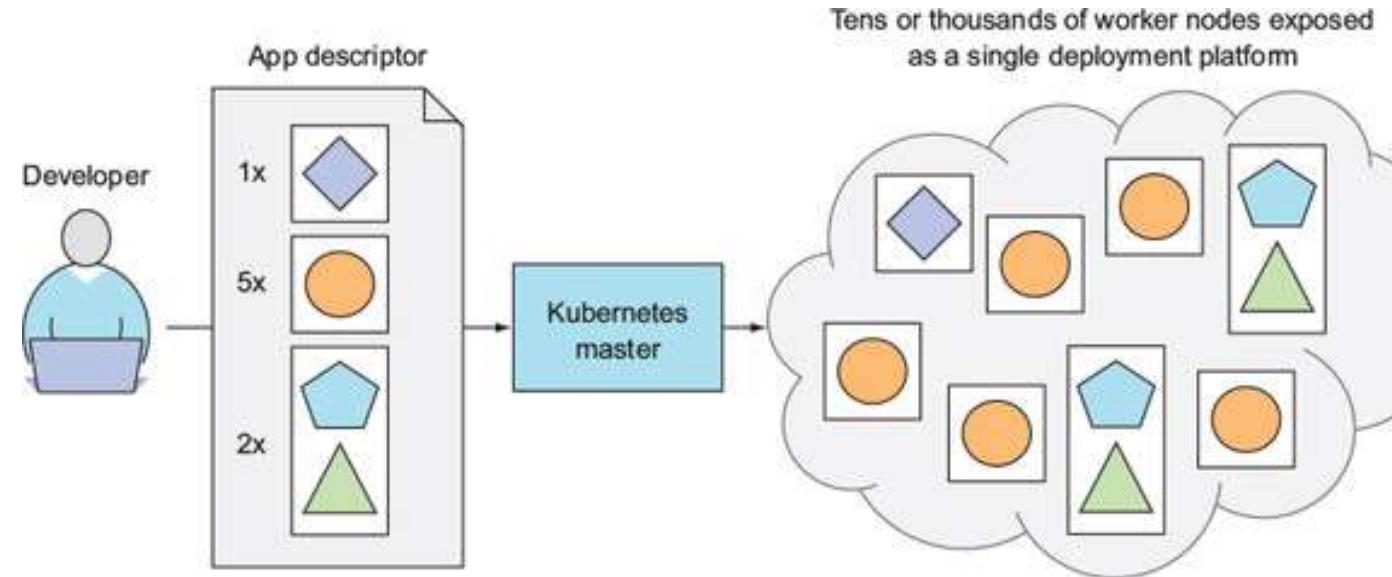
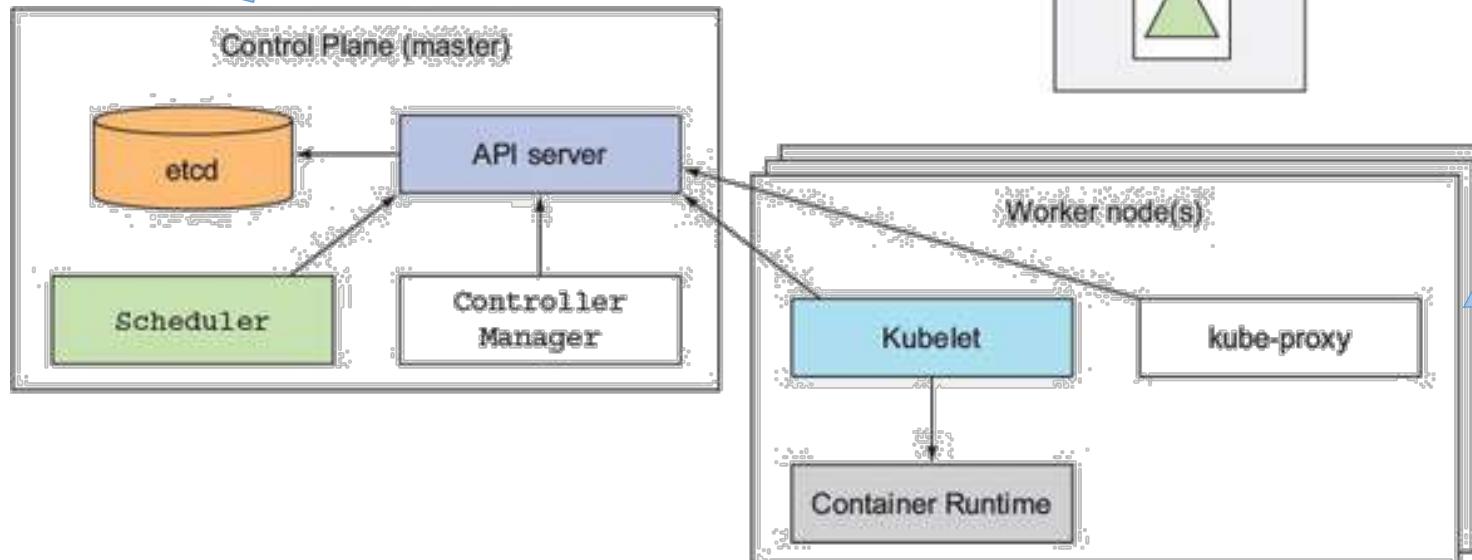
- **Pods**, or groups of containers, can group together container images developed by different teams into a single deployable unit.
- Kubernetes **services** provide load balancing, naming, and discovery to isolate one microservice from another.
- **Namespaces** provide isolation and access control, so that each microservice can control the degree to which other services interact with it.
- **Ingress** objects provide an easy-to-use frontend that can combine multiple microservices into a single externalized API surface area.



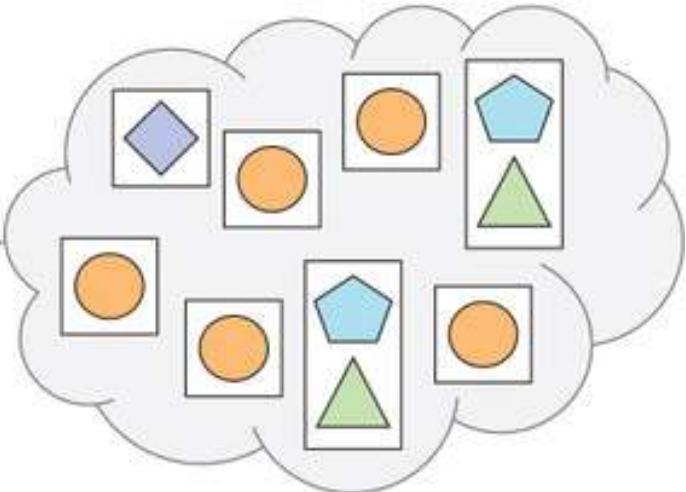


Kubernetes exposes the whole datacenter as a single deployment platform.

- The **Kubernetes API Server** to communicate to on control plane
- The **Scheduler**, which schedules our apps
- The **Controller Manager**, which performs cluster-level functions
- **etcd**, a reliable distributed data store that persistently stores the cluster configuration.



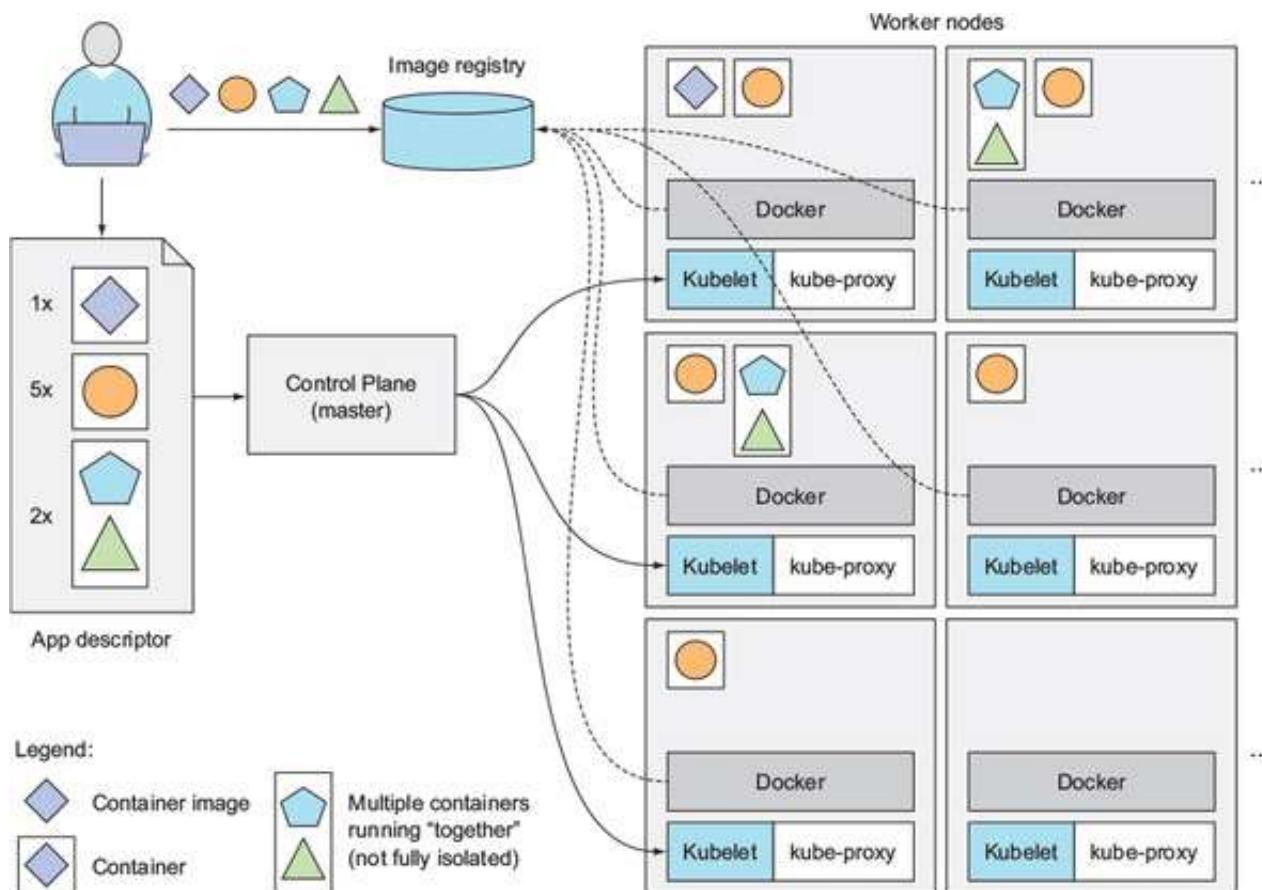
Tens or thousands of worker nodes exposed as a single deployment platform



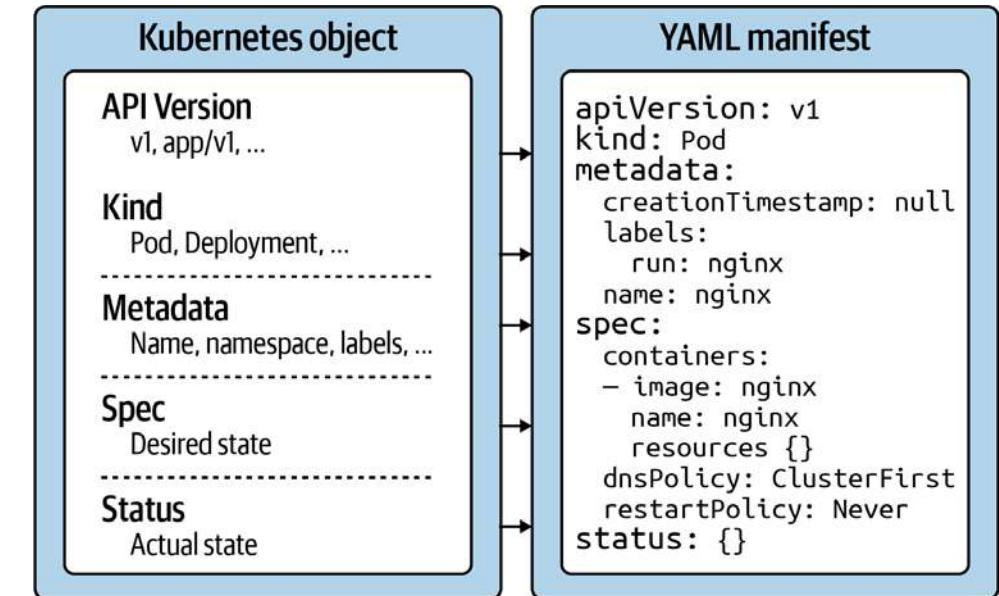
- **Docker, rkt, or another container runtime**, which runs your containers
- The **Kubelet**, which talks to the API server and manages containers on its node
- The **Kubernetes Service Proxy (kube-proxy)**, which load-balances network traffic between application components



Kubernetes architecture and an application running on it



kubia-manual.yaml



- **1 Descriptor conforms to version v1 of Kubernetes API**
- **2 Describing a pod.**
- **3 The name of the pod**
- **4 Container image to create the container from**
- **5 Name of the container**
- **6 The port the app is listening on**



K8s on Minikube that runs on your workstation

- Minikube is a tool that makes it easy to run Kubernetes locally.
 - Minikube runs a single-node Kubernetes cluster inside a Virtual Machine (VM) on your workstation for users looking to try out Kubernetes or develop with it day-to-day.
- Minikube supports the following Kubernetes features:
 - DNS, NodePorts, ConfigMaps and Secrets, Dashboards, Container Runtime: Docker, CRI-O, and containerd, enabling CNI (Container Network Interface) and Ingress
- Not a production grade technology – useful for development and testing purposes



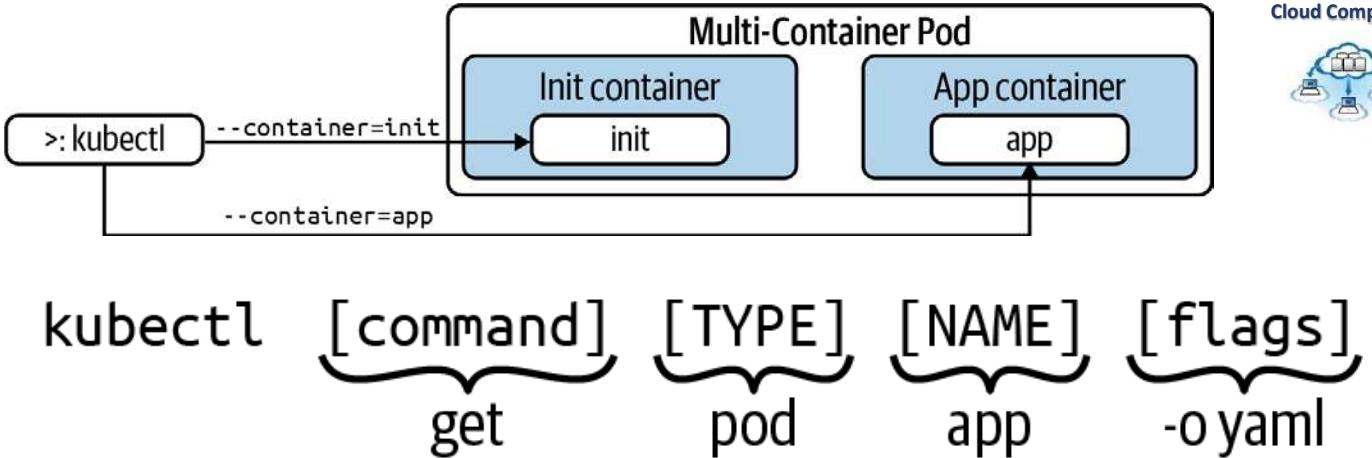
Yaml Example

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: kube-demo-deployment
  labels:
    app: demo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: demo
  template:
    metadata:
      labels:
        app: demo
    spec:
      containers:
        - name: demo
          image: suriarasai/demo:latest
          ports:
            - containerPort: 8080

```

App
Pod
Deployment
API reference
Failure Recovery
Application Architecture
Container Manifest
Communication



```

Administrator: Windows PowerShell
PS E:\ARTS2021\workspace\demo> kubectl apply -f deployment.yaml
deployment.apps/kube-demo-deployment created
PS E:\ARTS2021\workspace\demo>

```

```

PS E:\ARTS2021\workspace\demo> kubectl get deployments
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
kube-demo-deployment   0/1       1           0           47s
PS E:\ARTS2021\workspace\demo>

```

```

PS E:\ARTS2021\workspace\demo> kubectl expose deployment kube-demo-deployment
service/my-service exposed
PS E:\ARTS2021\workspace\demo>

```

```

PS E:\ARTS2021\workspace\demo> kubectl get services
NAME      TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes  ClusterIP  10.96.0.1   <none>        443/TCP   12h
my-service  LoadBalancer  10.105.87.50  <pending>    8080:30546/TCP  20m
PS E:\ARTS2021\workspace\demo>

```



Minikube Highlights and Common Commands

▪ Some minikube commands

```
$ minikube start
```

```
$ minikube stop
```

```
$ minikube delete
```

```
$ minikube service hello-minikube --url
```

```
$ minikube start --docker-env http_proxy=http://$YOURPROXY:PORT \
--docker-env https_proxy=https://$YOURPROXY:PORT
```

▪ Some client commands

```
$ kubectl version
```

```
$ kubectl create deployment hello-minikube --image=k8s.gcr.io/echoserver:1.10
```

```
$ kubectl expose deployment hello-minikube --type=NodePort --port=8080
```

```
$ kubectl get pod
```

```
$ kubectl delete services hello-minikube
```

```
$ kubectl delete deployment hello-minikube
```



K8s on Google Cloud Platform (GCP)

- Naturally, as a birthplace of K8s, GCP has high level of support for Kubernetes through its Google Container Engine (GKE) offerings.
 - No additional fee for clusters <= 5 nodes (User will pay for additional VMs they choose to run)
 - Automated cluster setup, maintenance, monitoring, scaling, health checking
- One can also use kops on manual configuration to bring up the cluster on Google Compute Engine VMs, if needed.

Once you have gcloud installed, first set a default zone:

```
$ gcloud config set compute/zone us-west1-a
```

Then you can create a cluster:

```
$ gcloud container clusters create mycompany-cluster
```

This will take a few minutes. When the cluster is ready you can get credentials for the cluster using:

```
$ gcloud auth application-default login
```



K8s on Amazon Web Services

- AWS's premier container offering, namely, Elastic Container Service (ECS) is a proprietary orchestration system that is not yet k8s compatible
 - AWS has joined CNCF and its support form k8s is offered as EKS
 - AWS also offers EC2 Elastic Compute Cloud instances that can be used with k8s
 - Kops is a tool for production grade k8s installation, upgrades and management – maintained by CNCF

Once you have eksctl installed and in your path, you can run the following command to create a cluster:

```
$ eksctl create cluster --name mycompany-cluster ...
```

For more details on installation options (such as node size and more), view the help using this command:

```
$ eksctl create cluster --help
```



K8s on Azure Kubernetes Service

- Azure Kubernetes Service (AKS) offers serverless Kubernetes, an integrated continuous integration and continuous delivery (CI/CD), security and governance.
 - Easily define, deploy, debug, and upgrade even the most complex Kubernetes applications, and automatically containerize applications.

When you have the shell up and working, you can run:

```
$ az group create --name=mycompany --location=westus
```

Once the resource group is created, you can create a cluster using:

```
$ az aks create --resource-group=mycompany --name=mycompany-cluster
```

This will take a few minutes. Once the cluster is created, you can get credentials for the cluster with:

```
$ az aks get-credentials --resource-group=mycompany --name=mycompany-cluster
```

If you don't already have the kubectl tool installed, you can install it using:

```
$ az aks install-cli
```



yaml, Annotations Examples

```
object1.yaml x
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: hello-world
          image: hello-world:latest
        ports:
          - containerPort: 80
```

```
Welcome demo.bal x
1 import ballerina/http;
2 import ballerinax/kubernetes;
3
4 @kubernetes:Service {
5   serviceType: "NodePort",
6   name: "ballerina-demo"
7 }
8 endpoint http:Listener listener {
9   port: 9090
10 };
11
12 @kubernetes|
13 service<http:kubernetes>:ConfigMap
14 @http:R:kubernetes>:Deployment
15 met:kubernetes>:HPA
16 }>kubernetes>:Ingress
17 hi(end:kubernetes>:PersistentVolumeClaim
18 str:kubernetes>:Secret
19 _=kubernetes>:Service
20 }
21 }
```

POM.xml

```
...
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-kubernetes</artifactId>
</dependency>
...

```

config.yaml

```
apiVersion: v1 by d
kind: ConfigMap
metadata:
  name: client-service
data:
  application.properties: |-
    bean.message=Testing reload!
    Message from backend is: %s <br/> Services : %s
```

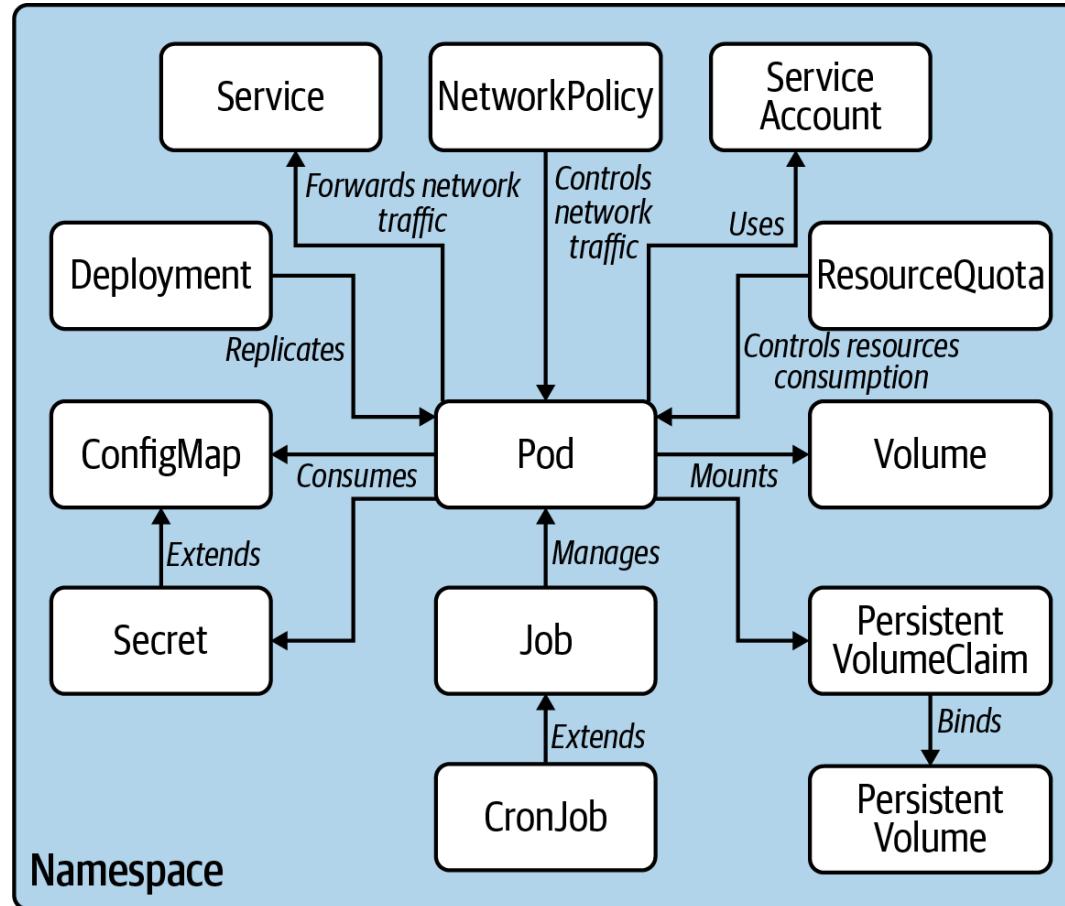
Code Injection

```
@Configuration
@ConfigurationProperties(prefix = "bean")
public class ClientConfig {

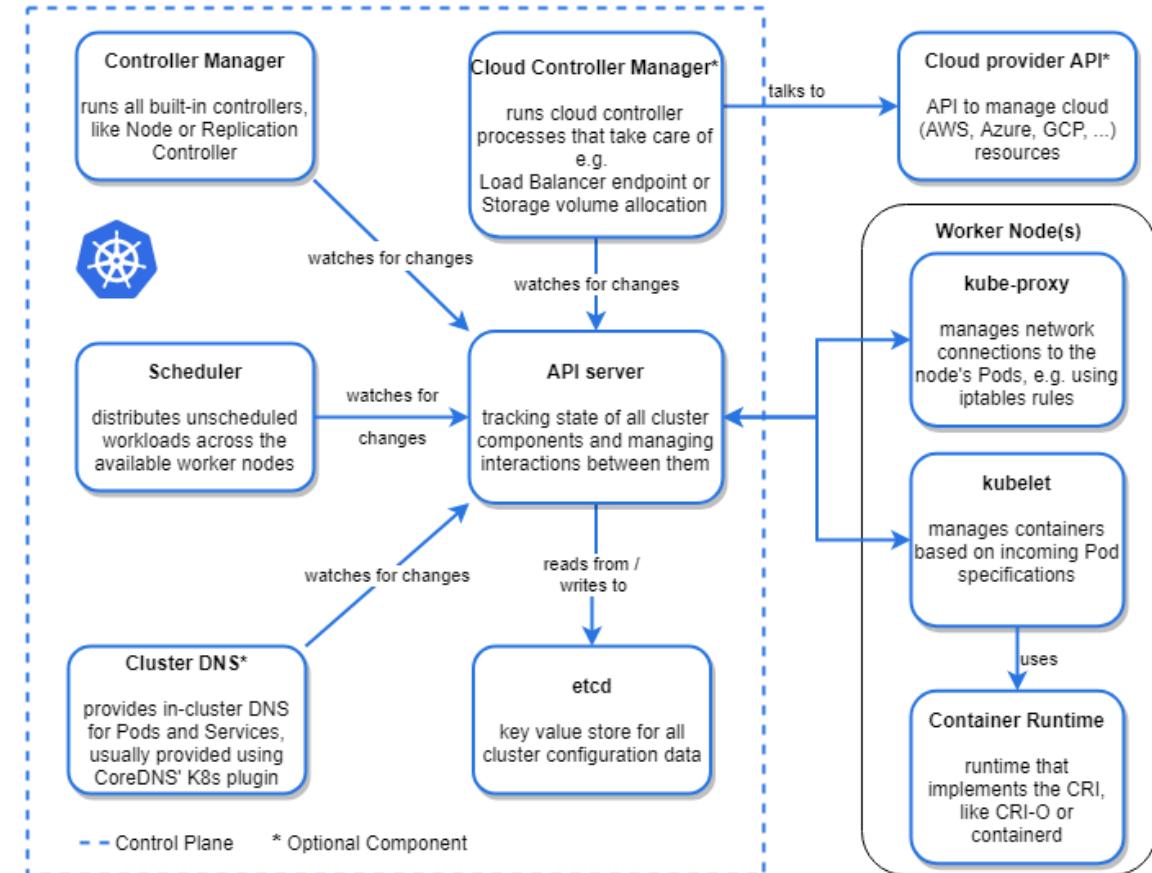
  private String message = "Message from backend is
    <br/> Services : %s";
  // getters and setters
}
```



What else can the configuration do?



Kubernetes Architecture

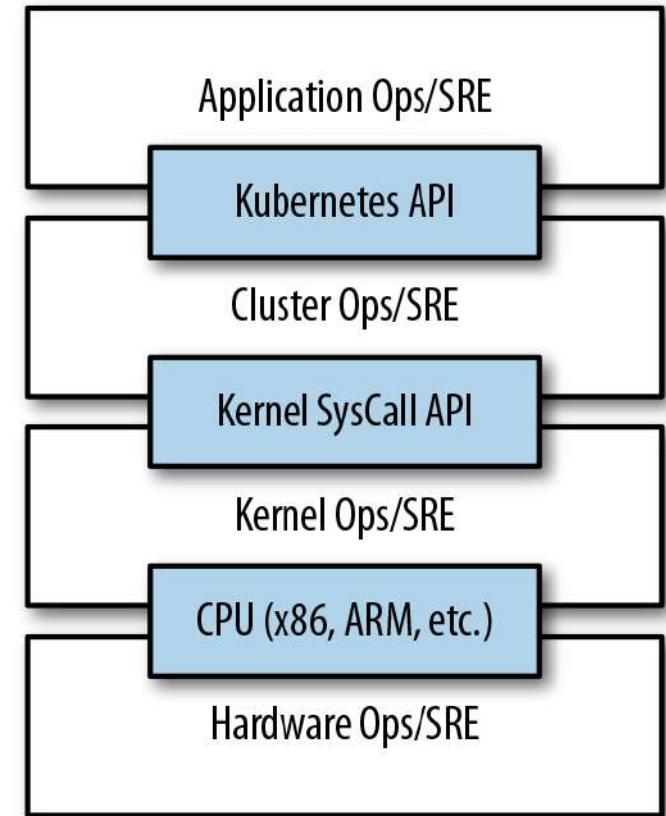


<https://brennerm.github.io/posts/kubernetes-overview-diagrams.html>



Kubernetes Advantages

- Velocity
 - Immutability
 - Declarative configuration
 - Online self-healing systems
- Scaling (of both software and teams)
 - scalability by favoring *decoupled* architectures
 - Easy Scaling for Applications and Clusters
 - Scaling Development Teams with Microservices
 - Separation of Concerns for Consistency and Scaling
- Abstracting your infrastructure
- Efficiency



<https://queue.acm.org/detail.cfm?id=2898444#content-comments>



SUMMARY & APPENDIX



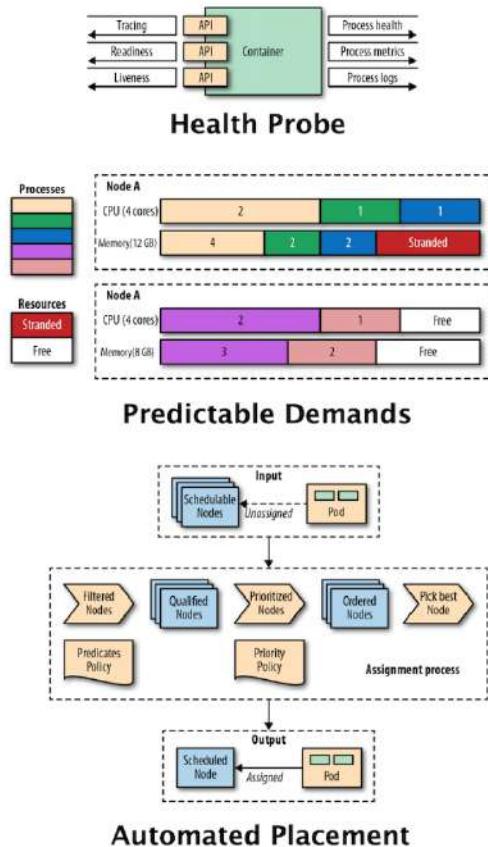
Concluding Remarks

- **Kubernetes** represents its functionality for deploying and operating a cloud-native application with the help of primitives.
 - Each **primitive** follows a general structure: the API version, the kind, the metadata and the desired state of the resources, also called the spec.
 - Upon creation or modification of the object, the Kubernetes **scheduler** automatically tries to ensure that the actual state of the object follows the defined specification.
- **Kubernetes** is a production-ready, open source platform designed with Google's accumulated experience in container orchestration, combined with best-of-breed ideas from the community.
 - Kubernetes is an open source container orchestration engine for automating deployment, scaling, and management of containerized applications.
 - The open source project is hosted by the Cloud Native Computing Foundation (CNCF).
 - Kubernetes services, support, and tools are widely available.

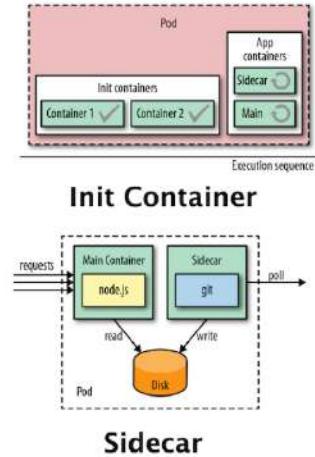


Top 10 Must-Know Design Patterns for Kubernetes Beginners

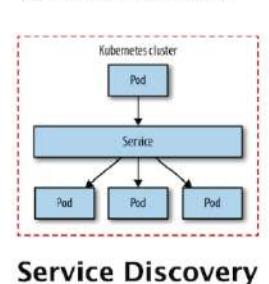
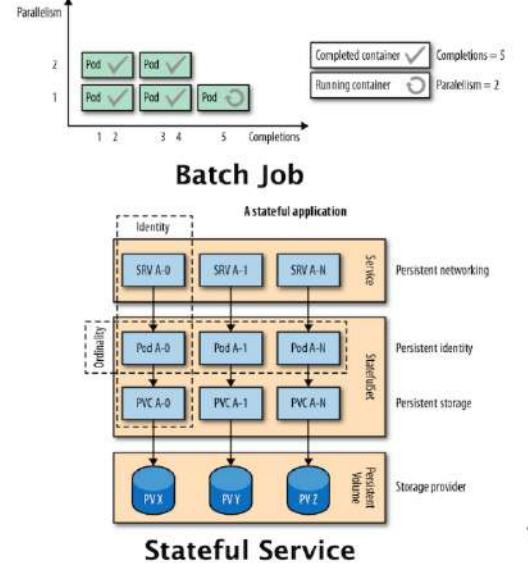
Foundational



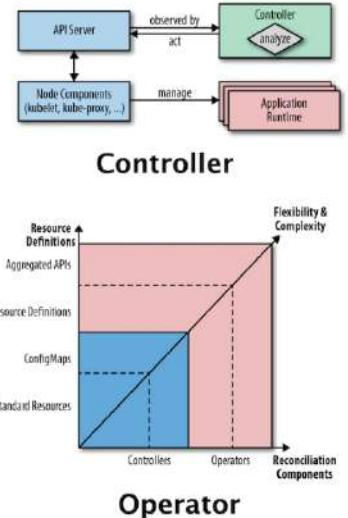
Structural



Behavioural

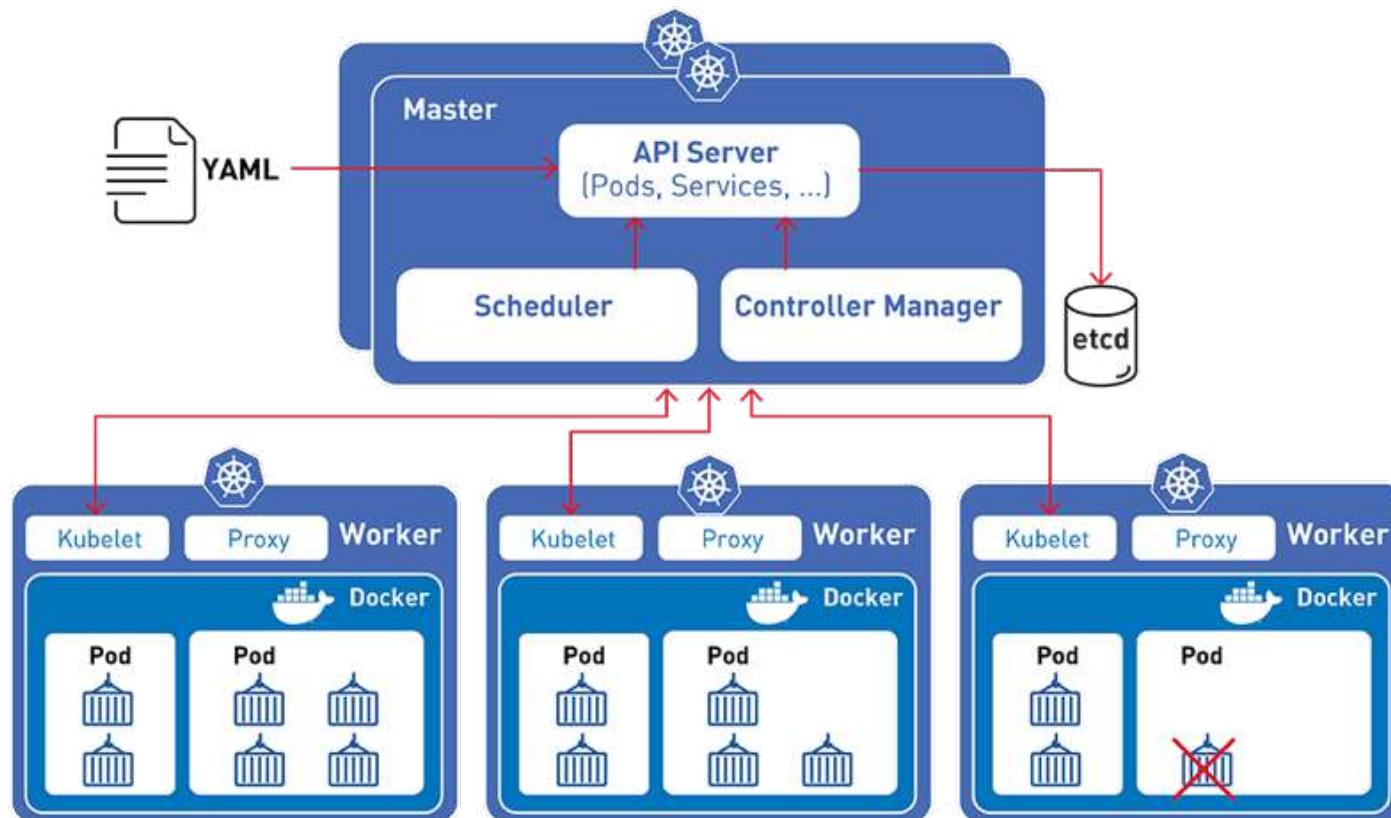
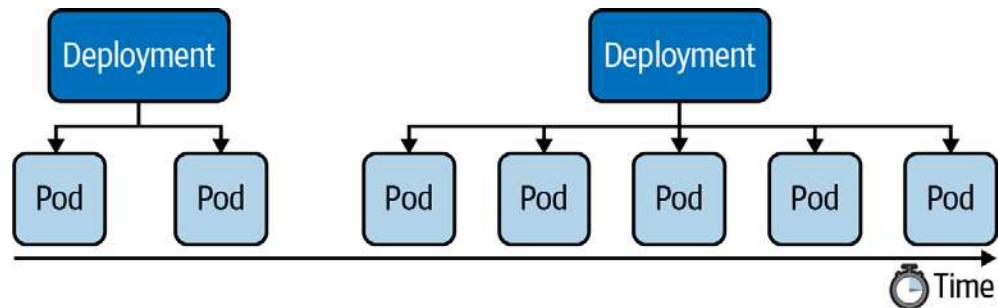


Higher-level





KUBERNETES



```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: ml-model-serving
  labels:
    app: ml-model
spec:
  replicas: 10
  selector:
    matchLabels:
      app: ml-model
  template:
    metadata:
      labels:
        app: ml-model
    spec:
      containers:
        - name: ml-rest-server
          image: ml-serving:1.0
          ports:
            - containerPort: 80

```

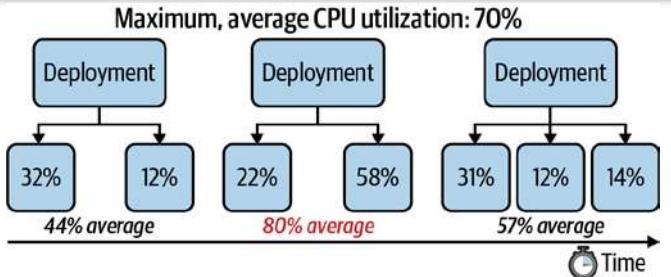
How many Pods should be running?

How do we find Pods that belong to this Deployment?

What should a Pod look like?

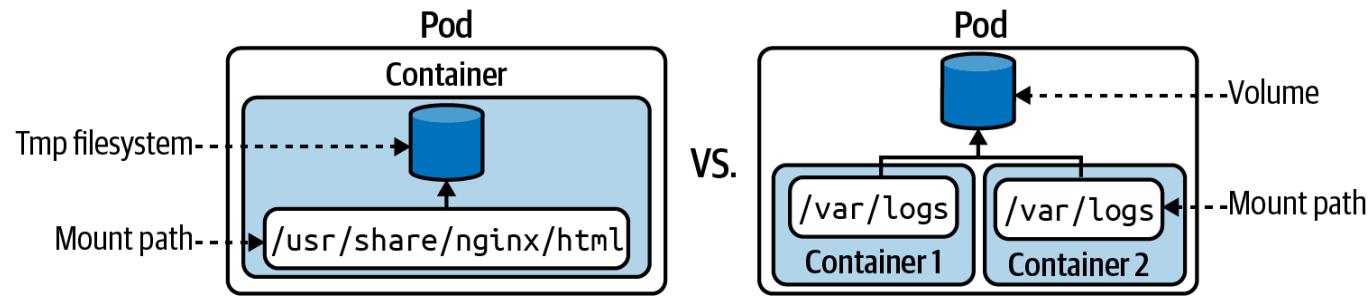
Add a label to the Pods so our Deployment can find the Pods to manage.

What containers should be running in the Pod?





State Management



```

kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: db-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 512m
  
```

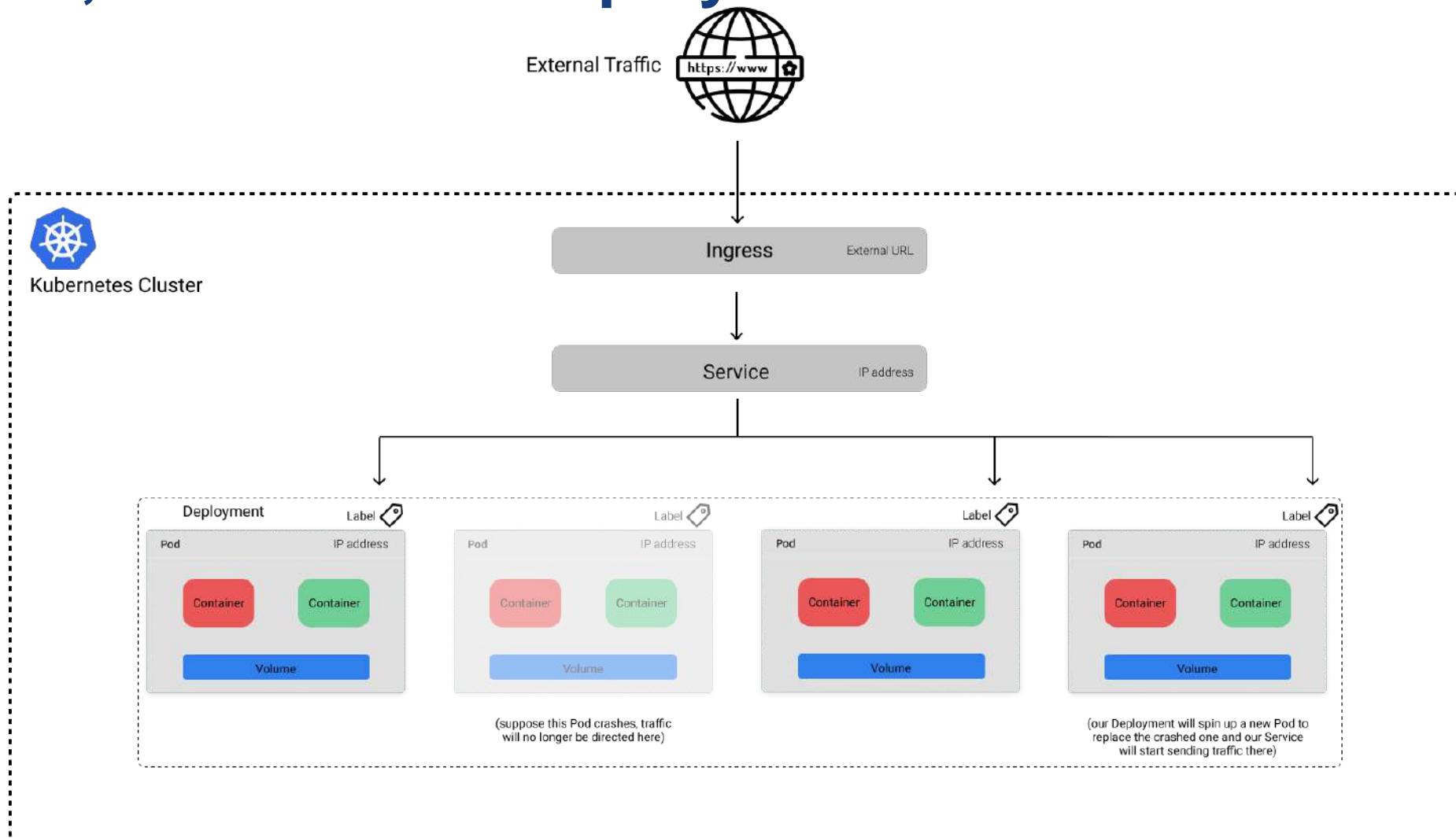


```

apiVersion: v1
kind: Pod
metadata:
  name: app-consuming-pvc
spec:
  volumes:
    - name: app-storage
      persistentVolumeClaim:
        claimName: db-pvc
  containers:
    - image: alpine
      name: app
      command: ["/bin/sh"]
      args: ["-c", "while true; do sleep 60; done;"]
      volumeMounts:
        - mountPath: "/mnt/data"
          name: app-storage
  
```



Ingress, Service and Deployment Labels



Reference: <https://www.jeremyjordan.me/kubernetes/>



Cloud Native Solution Design

OPEN STACK STANDARD FOR VIRTUAL IMAGE

Suria R Asai
suria@nus.edu.sg
Institute of Systems Science
National University of Singapore

© 2009-23 NUS. The contents contained in this document may not be reproduced in any form or by any means, without the written permission of ISS, NUS, other than for the purpose for which it has been supplied.

Total Slides 15



What is open stack?

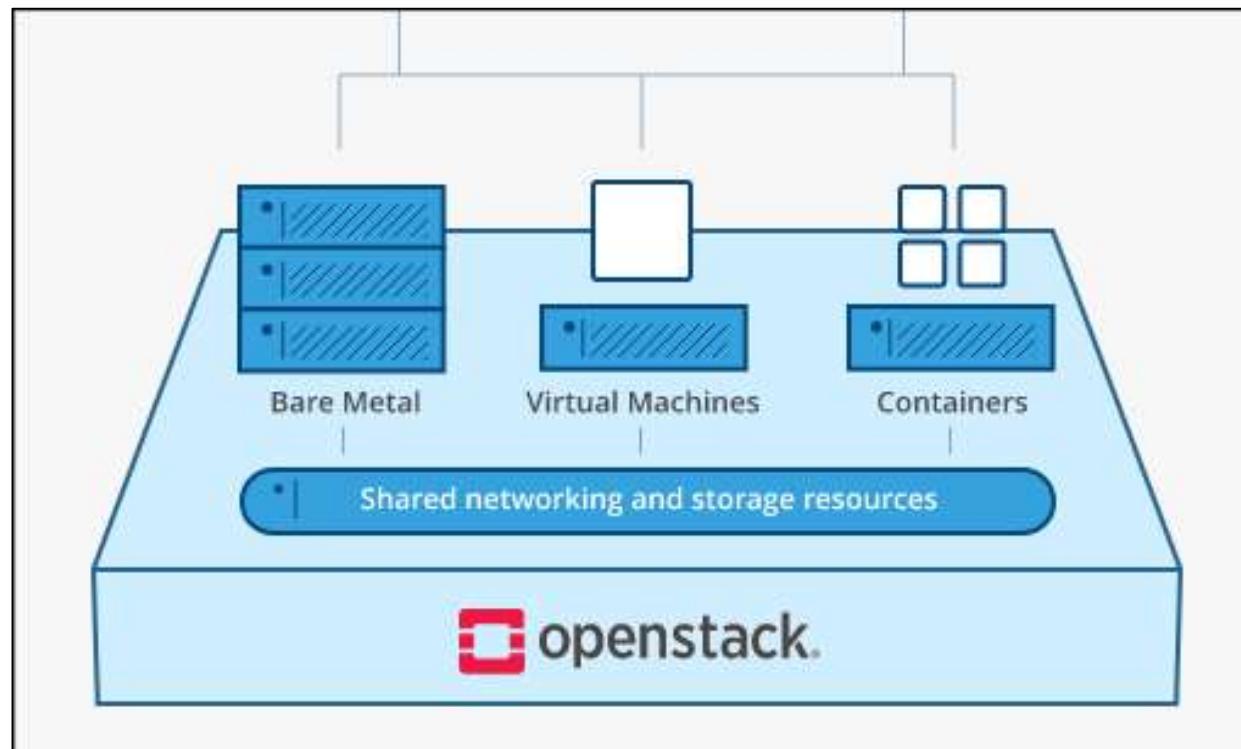
Kubernetes

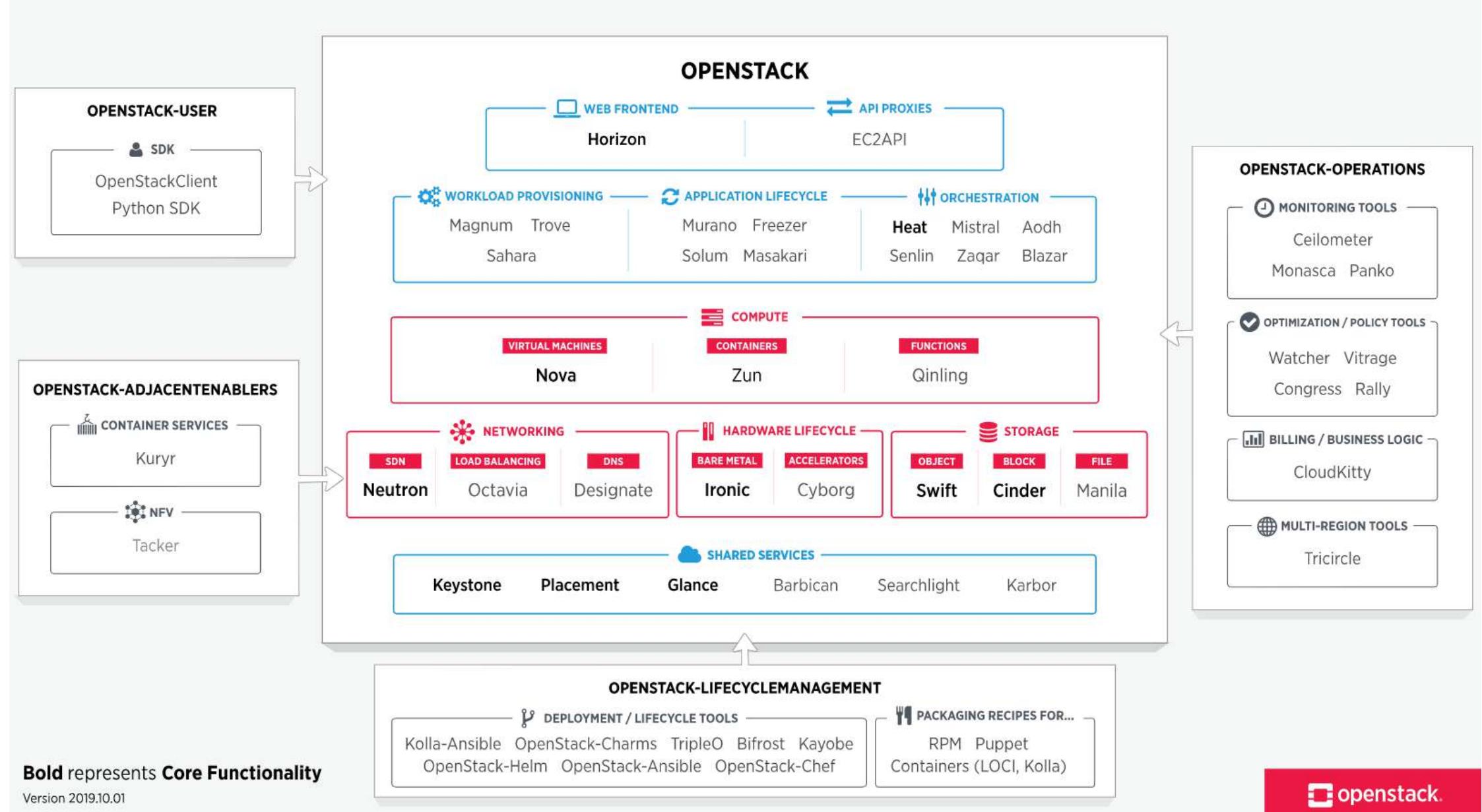
CloudFoundry

TerraForm

OpenStackSDK

HorizonWebUI



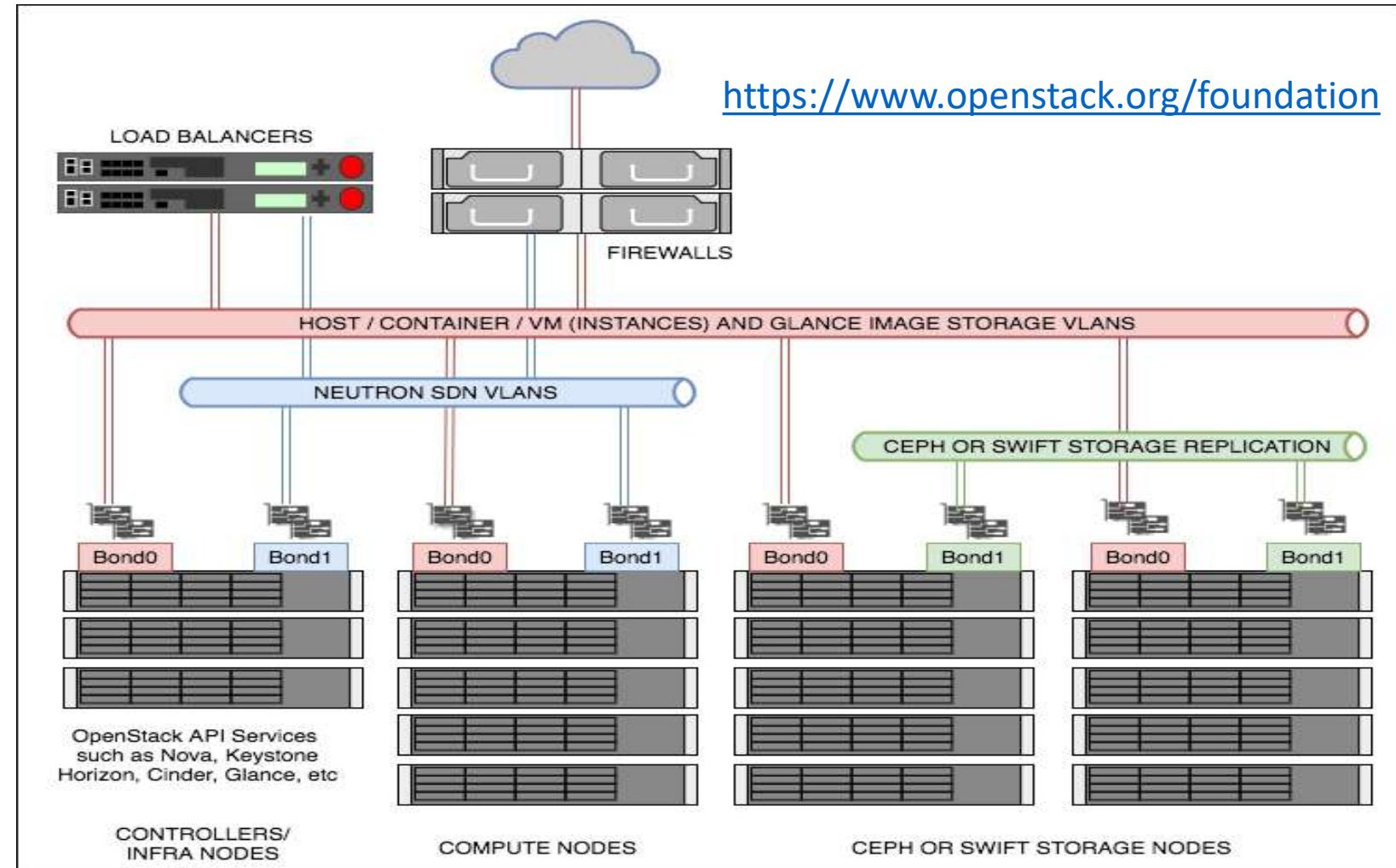




Recommended OpenStack architecture

▪ OpenStack

- Suite of projects that combine into a **software-defined environment** to be consumed using cloud friendly tools and techniques.
- OpenStack can be architected in any number of ways
- Very popular method for installing OpenStack is the **OpenStack-Ansible** project





Core Components - 1

▪ Controllers

- The controllers (infra nodes) run the heart of the OpenStack services and are the only servers exposed (via load balanced pools) to end users.
- The controllers run the API services, such as Nova API, Keystone API, and Neutron API, as well as the core supporting services such as MariaDB for the database required to run OpenStack, and RabbitMQ for messaging.

▪ Computes

- Servers that run the hypervisor or container service that OpenStack schedules workloads to when a user requests a Nova resource (such as a virtual machine).



Core Components - 2

▪ Storage

- Storage in OpenStack refers to block storage and object storage.
 - Block storage (providing LUNs or *hard drives* to virtual machines) is provided by the Cinder service.
 - Object storage (API driven object or blobs of storage) is provided by Swift or Ceph. Swift and Ceph manage each individual drive in a server designated as an object storage node, very much like a RAID card manages individual drives in a typical server.

▪ Load balancing

- The end users of the OpenStack environment expect services to be highly available, and OpenStack provides REST API services to all of its features.



Minimal Data Center Deployment Requirements

- **Controller servers** (also known as infrastructure nodes)

- At least 64 GB RAM
- At least 300 GB disk (RAID)
- 4 Network Interface Cards (for creating two sets of bonded interfaces; one would be used for infrastructure and all API communication, including client, and the other would be dedicated to OpenStack networking: Neutron)
- Shared storage, or object storage service, to provide backend storage for the base OS images used

- **Compute servers**

- At least 64 GB RAM
- At least 600 GB disk (RAID)
- 4 Network Interface Cards (for creating two sets of bonded interfaces, used in the same way as the controller servers)

- **Optional** (if using Ceph for Cinder) 5 Ceph Servers (Ceph OSD nodes)

- At least 64 GB RAM
- 2 x SSD (RAID1) 400 GB for journaling
- 8 x SAS or SSD 300 GB (No RAID) for OSD (size up requirements and adjust accordingly)
- 4 Network Interface Cards

- **Optional** (if using Swift) 5 Swift Servers

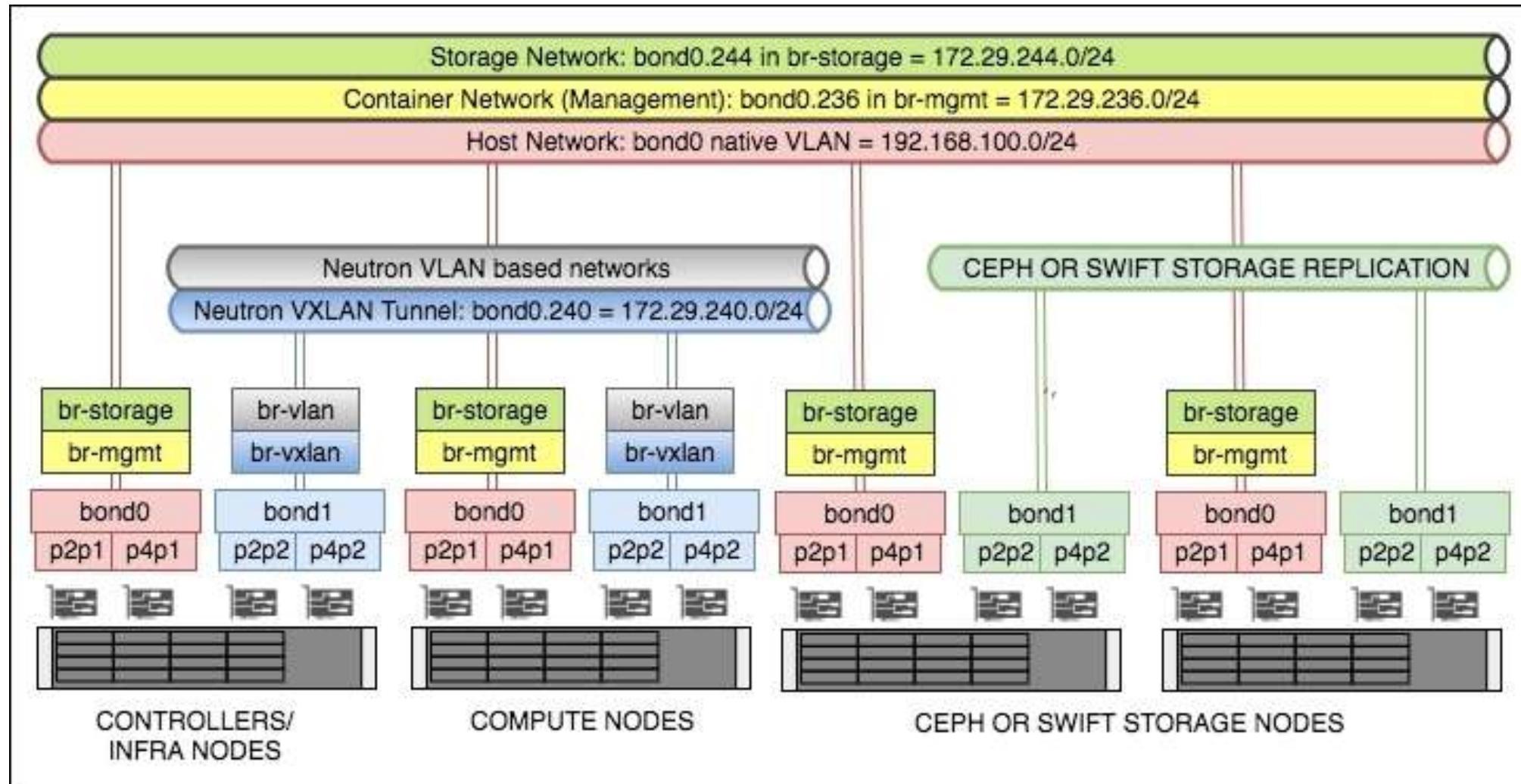
- At least 64 GB RAM
- 8 x SAS 300 GB (No RAID) (size up requirements and adjust accordingly)
- 4 Network Interface Cards

- **Load balancers**

- 2 physical load balancers configured as a pair
- Or 2 servers running HAProxy with a Keepalived VIP to provide as the API endpoint IP address:
- At least 16 GB RAM
- HAProxy + Keepalived
- 2 Network Interface Cards (bonded)



Example Setup

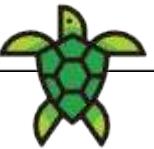
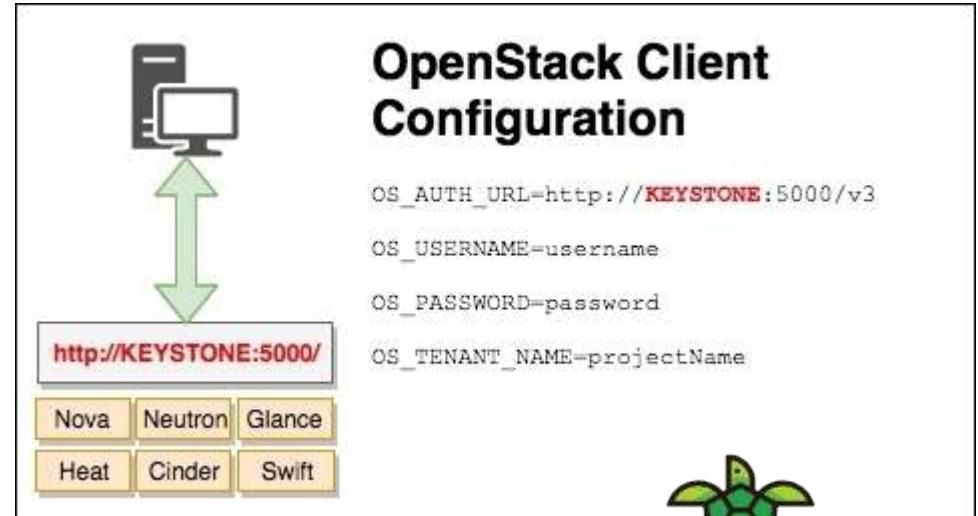




Client

*The **Keystone** service (OpenStack Identity service) essentially performs two functions in OpenStack. It authorizes users to allow them to perform the actions they have requested, as well as provides a catalogue of services back to the user.*

```
openstack network create NETWORK_NAME
openstack subnet create SUBNET_NAME
--network NETWORK_NAME--subnet-range CIDR
openstack security group rule create
--remote-ip 0.0.0.0/0
--dst-port 80:80
--protocol tcp
--ingress
--project development
webserver
openstack image create
--container-type bare
--disk-format qcow2
--public
--file
/path/to/cirros-0.3.5-x86_64-disk.img
```





Ansible

- **Ansible** is an open-source software provisioning, configuration management, application-deployment and orchestration tool.
 - Can configure both **Unix-like** systems as well as Microsoft **Windows**.
 - Includes its own declarative language to describe system configuration.
 - Written in **Python** and has a fairly minimal learning curve.
 - Simple setup procedure
 - No dependence on any additional software, servers or client daemons.
 - Used orchestration templates (known as **HOT -Heat Orchestration Template**) as a recipe that is written in **YAML (Yet Another Markup Language)**.



Ansible Examples

Create the Ansible playbook for tasks that will launch the instances.

[launch-instance.yml](#)

```
- name: Launch instance on OpenStack
hosts: localhost
gather_facts: false
tasks:
- name: Deploy an instance
os_server:
  state: present
  name: somename
  image: xenial-image
  key_name: demokey
  timeout: 200
  flavor: m1.tiny
  network: private-net
  verify: false
```

```
source openrc ansible-playbook launch-instance.yml
```

```
[WARNING]: provided hosts list is empty, only localhost is available
```

```
PLAY [Launch instance on OpenStack] ****
```

```
TASK [Deploy an instance] ****
changed: [localhost]
```

```
PLAY RECAP ****
localhost : ok=1    changed=1    unreachable=0    failed=0
```

openstack server list

ID	Name	Status	Networks	Image Name
...



Full Stack Example

full-stack.yml

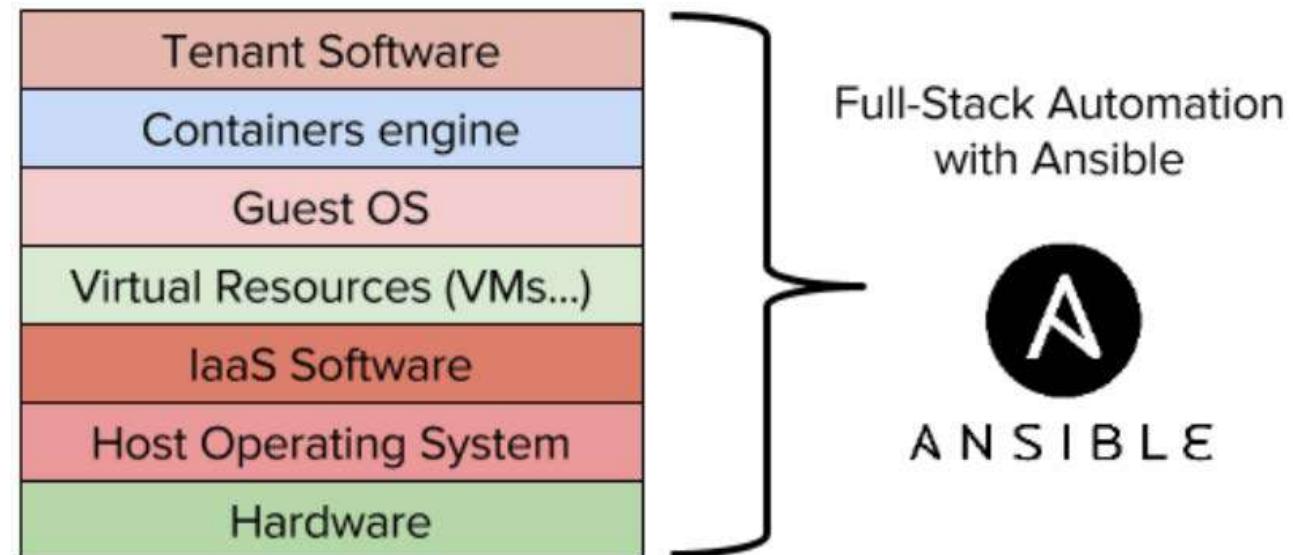
```
- name: Create OpenStack Cloud Environment
  hosts: localhost
  gather_facts: false
  vars:
    webserver_count: 2

  tasks:
    - name: Download Ubuntu 16.04 Xenial
      get_url:
        url: http://releases.ubuntu.com/16.04/ubuntu-16.04.3-server-amd64.img
        dest: /tmp/ubuntu-16.04.img
    - name: Ensure Ubuntu 16.04 Xenial Image Exists
      os_image:
        name: xenial-image
        container_format: bare
        disk_format: qcow2
        state: present
        filename: /tmp/ubuntu-16.04.img
        verify: false
    . . . Security Groups Sub Net and lot of other configurations
```



Final Open Stack Remarks

- OpenStack is a suite of projects that combine into a software-defined environment to be consumed using cloud friendly tools and techniques.
 - A very popular method for installing OpenStack is the OpenStack-Ansible project (<https://github.com/openstack/openstack-ansible>).





SUMMARY



Summary

- Openstack controls large pools of compute, storage, and networking resources, all managed through APIs or a dashboard.
 - Beyond standard infrastructure-as-a-service functionality, additional components provide orchestration, fault management and service management amongst other services to ensure high availability of user applications.
 - Host your cloud infrastructure internally or find an OpenStack partner in the Marketplace.
 - OpenStack is trusted to manage 20 Million+ cores around the world, across dozens of industries.



Cloud Native Solution Design

MICROSERVICES DESIGN

INTERPROCESS COMMUNICATION – MICROSERVICES CHASSIS & DESIGN

Suria R Asai

suria@nus.edu.sg

Institute of Systems Science

National University of Singapore

© 2009-23 NUS. The contents contained in this document may not be reproduced in any form or by any means, without the written permission of ISS, NUS, other than for the purpose for which it has been supplied.

Total slides: 66.



Agenda

- Introduction to Microservices
- Microservices in Detail
- Summary



Learning Objectives

- On completion of this module, the participant will
 - Understand the science of building microservices in detail.
 - Effectively build industrial strength software.
 - Transition into microservices and docker containers.
 - Address the deployment and scalability problems by separating applications from the infrastructure dependencies.



INTRODUCTION TO MICROSERVICES

MONOLITH VS MICROSERVICES

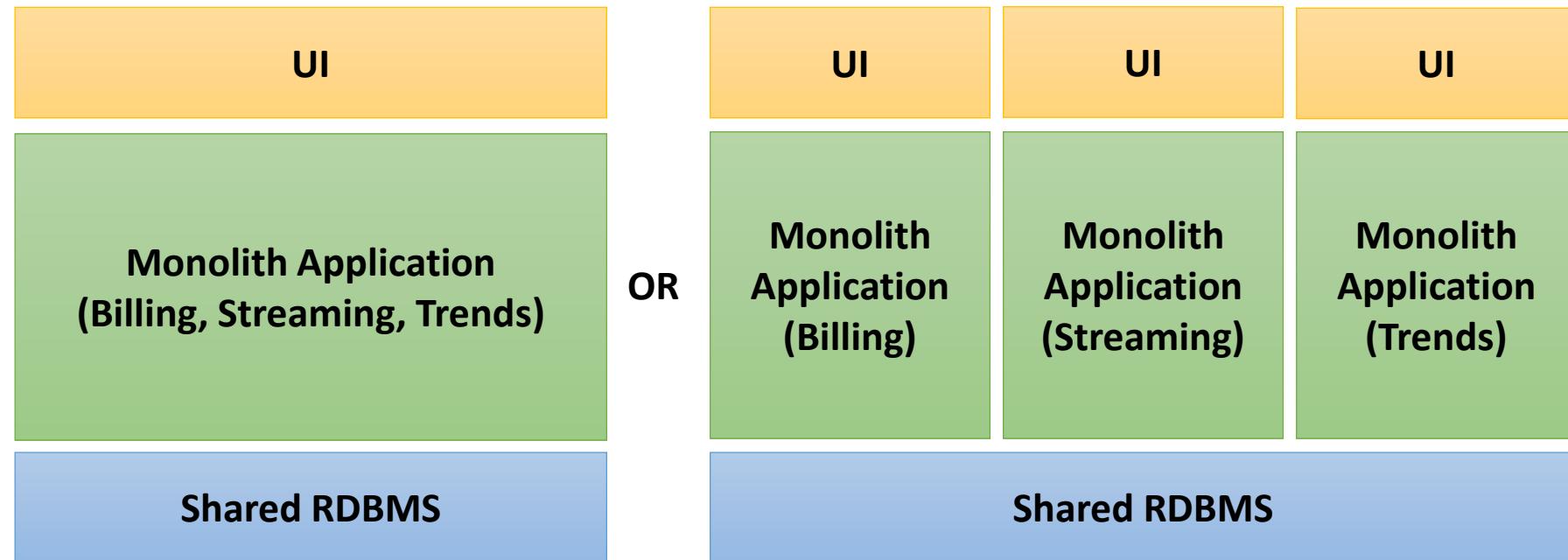
DOCKER DESIGN AND EXAMPLES



Monolithic Application - 1

Monolithic architecture is something that build from single piece of material, historically from rock. Monolith term normally use for object made from single large piece of material." - **Non-Technical Definition**

"Monolithic application has single code base with multiple modules. Modules are divided as either for business features or technical features. It has single build system which build entire application and/or dependency. It also has single executable or deployable binary" – **Technical Definition**

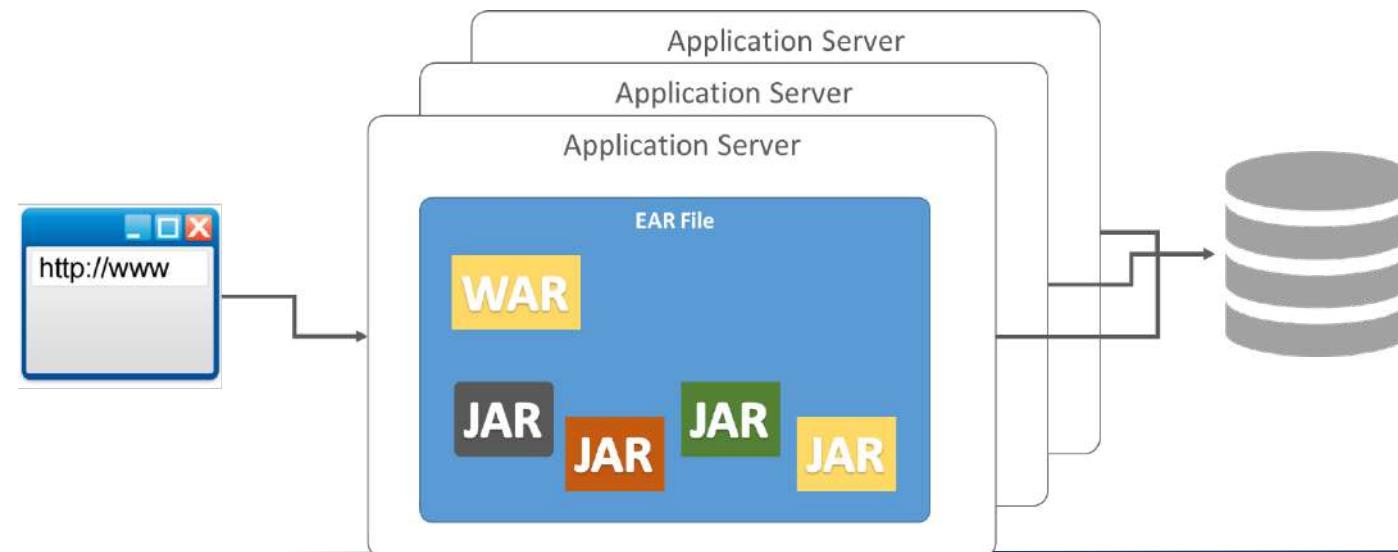




Monolithic Application - 2.

A business application built as a monolithic application is typically characterized by the following factors:

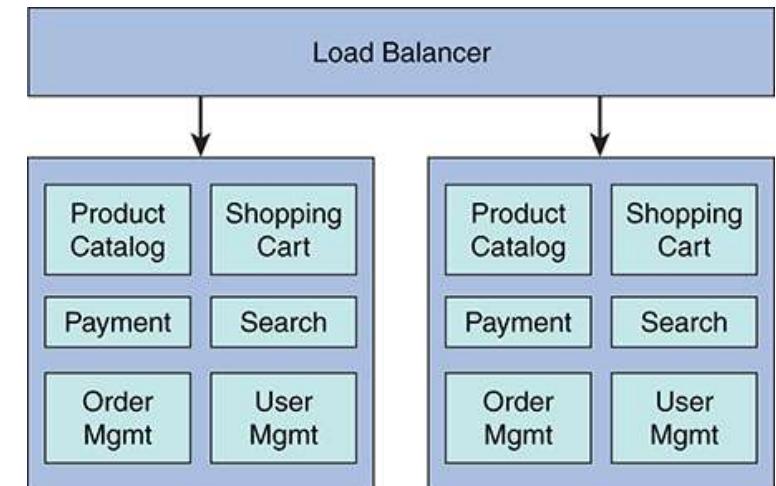
- The entire application logic is packaged into a single EAR file
- The application reuse is derived by sharing JARs
- Application changes are planned months in advance, typically in a big push once a quarter
- There is one database that encompasses the entire schema for the application
- There are thousands of test cases that signify the amount of regression
- The application design, development, and deployment requires coordination among multiple teams and significant management





Challenges in Monolithic Architecture

- Performance issues
- Scalability
- Longer cycles for regression testing
- Longer cycles to upgrade and redeployment, leading to an inability to deploy small fixes and enhancements
- Unscheduled downtime
- Potential downtime during upgrades
- Stuck with the existing technology and programming language
- No way to scale just the required components or functionality





Shortcomings of Monolith Architecture

- Availability Design is Difficult
- Not designed for rapid scalability
- Release Coordination is tricky
- Team Coordination is tricky
- Cost is high
- Branching and Promotion is Hard
- Updating libraries and managing conflicts is difficult
- Bottleneck for innovation
- Expensive Maintenance

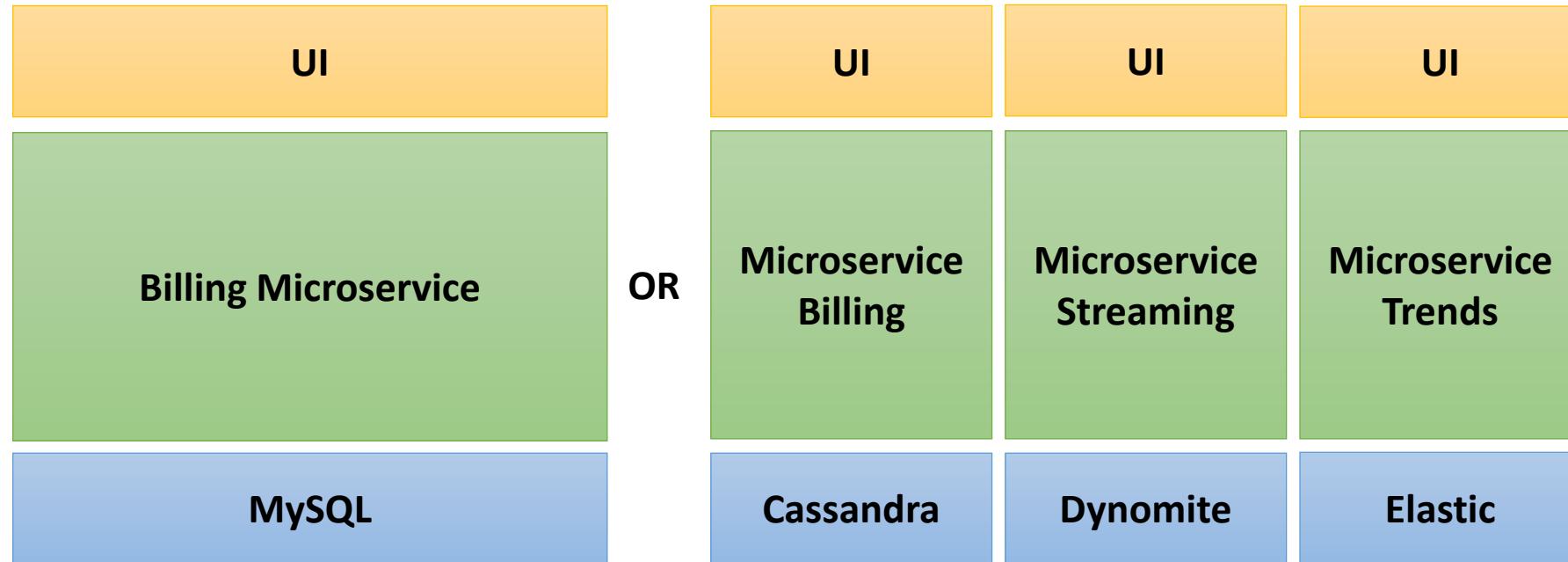


Microservices – Architecture 1

- Microservices is a **software architecture** style where complex applications are made up of **small**, independent processes which communicate with each other using **language-agnostic APIs**.
 - Microservices are a **modularization** concept. Their purpose is to divide large software systems into smaller parts with **single purpose**. Thus they influence the organization and development of software systems.
 - Microservice can be **deployed independently** of one another and are loosely coupled. Changes to one microservice can be taken into production independently of changes to other microservices.
 - Microservices can be implemented in **different technologies**. There is no restriction on the programming language or the platform for each microservice.
 - Microservices **possess their own data storage**: a private database or a completely separate schema in a shared database. Microservices can bring their own support services along, for example a search engine or a specific database.
 - Microservices have to **communicate via the network**. To do so microservices use protocols that support loose coupling, such as REST or messaging.



Microservices Architecture - 2



- If we break down monolithic complexity by using a microservices paradigm, we will end up with a modular architecture that significantly increases the shelf life.
- We can immediately reduce our dependence on multiple standards and bulky software development processes to save time, thus fast-tracking the overall software development lifecycle.



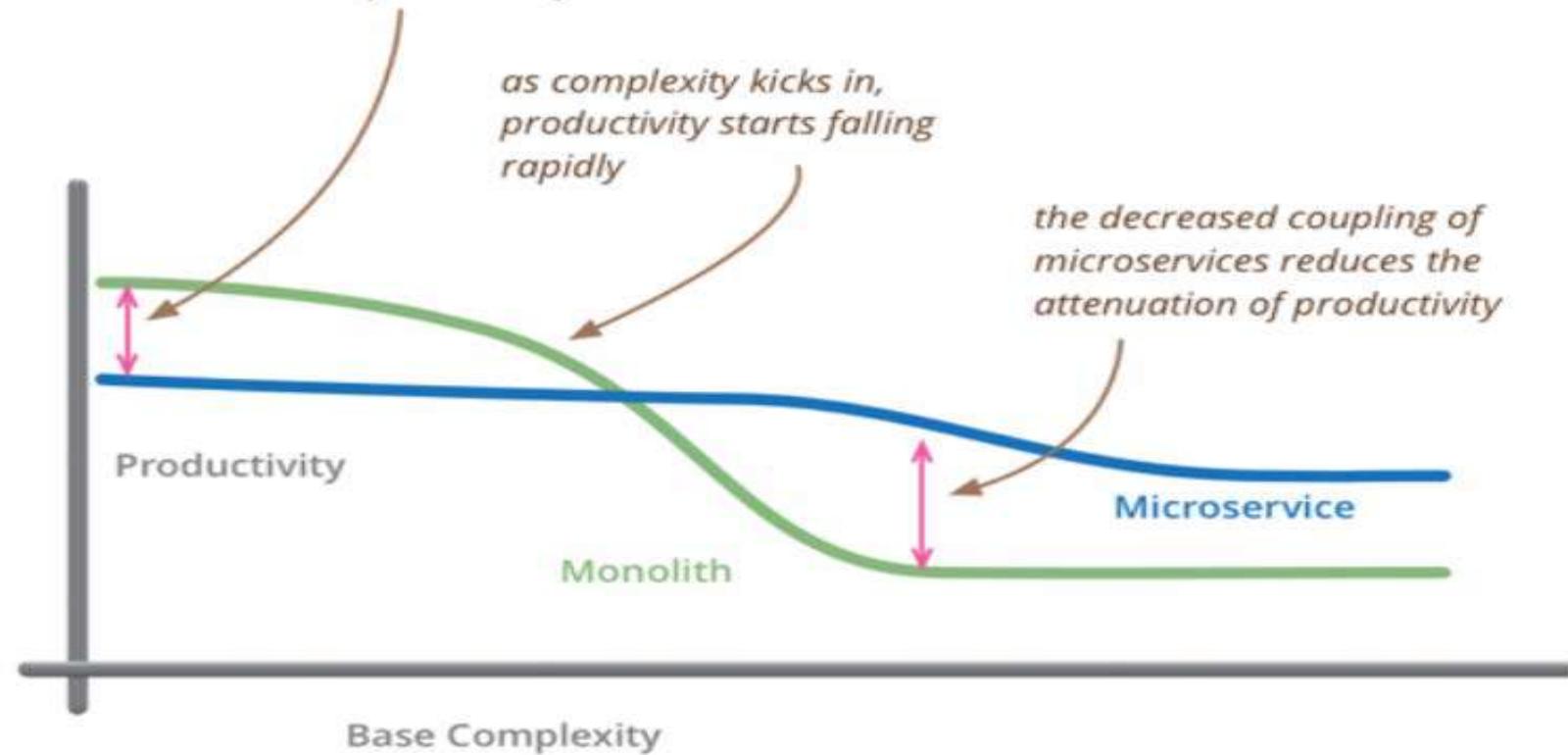
Microservices Architecture – 3.

- Microservices are designed and developed, keeping in mind that a business application can be built by composing these services. The microservices are designed around the following principles:
 - **Single-responsibility principle:** Each microservice implements only one business responsibility from the bounded domain context.
 - **Share nothing:** Microservices are autonomous, self-contained, stateless, and manage the service state (memory/storage) through container-based encapsulation models.
 - **Reactive:** This is applicable for microservices with concurrent loads or longer response times.
 - **Externalized configuration:** This externalizes the configurations in the config server, so that it can be maintained in hierarchical structure, per environment.
 - **Consistent:** Services should be written in a consistent style, as per the coding standards and naming convention guidelines.
 - **Resilient:** Services should handle exceptions arising from technical reasons (connectivity and runtime), and business reasons (invalid inputs) and not crash.
 - **Good citizens:** Microservices should report their usage statistics, the number of times they are accessed, their average response time, and so on through the JMX API or the HTTP API.
 - **Versioned:** Microservices may need to support multiple versions for different clients, until all clients migrate to higher versions.
 - **Independent deployment:** Each of the microservices should be independently deployable, without compromising the integrity of the application.



Microservices is not “Silver Bullet”

for less-complex systems, the extra baggage required to manage microservices reduces productivity



“don’t even consider microservices unless you have a system that’s too complex to manage as a monolith”

~ Martin Fowler

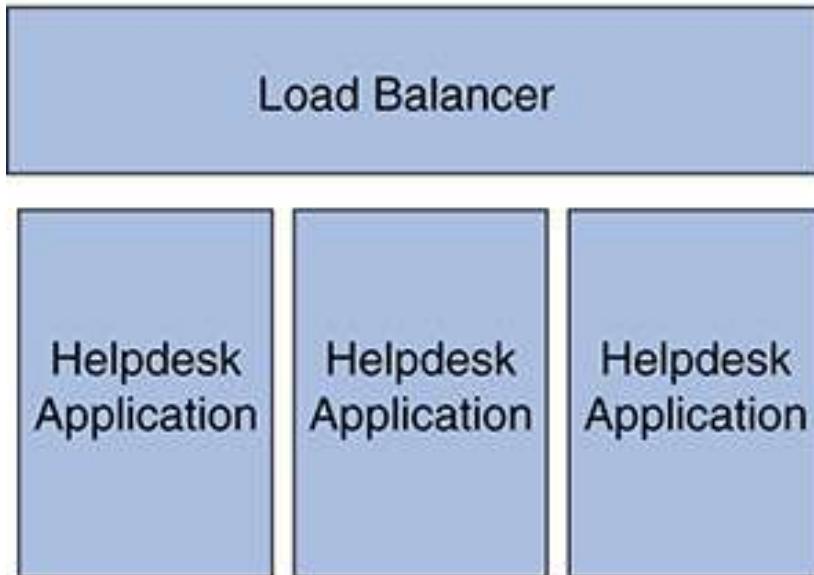
but remember the skill of the team will outweigh any monolith/microservice choice

<http://martinfowler.com/bliki/MicroservicePremium.html>

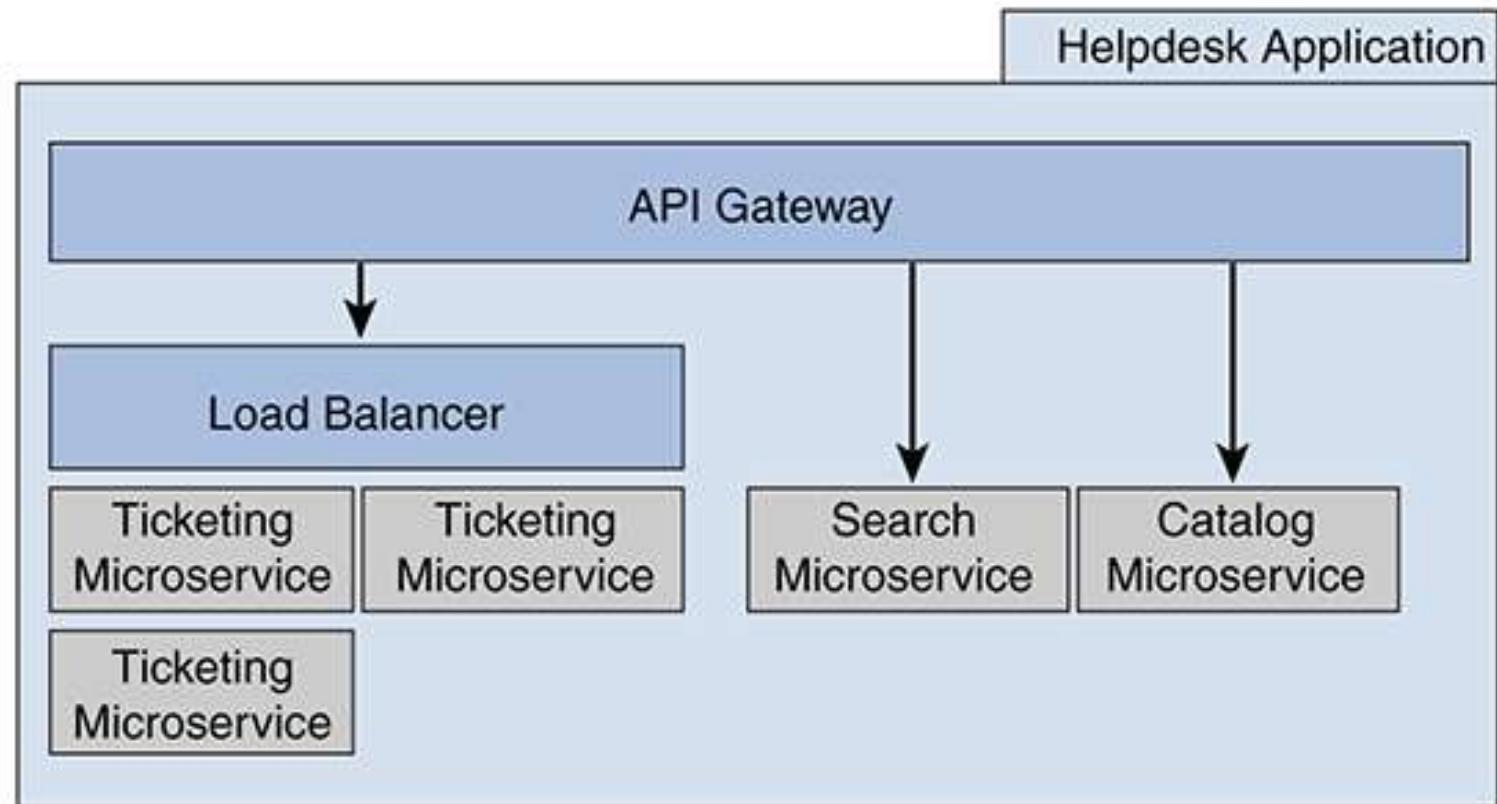


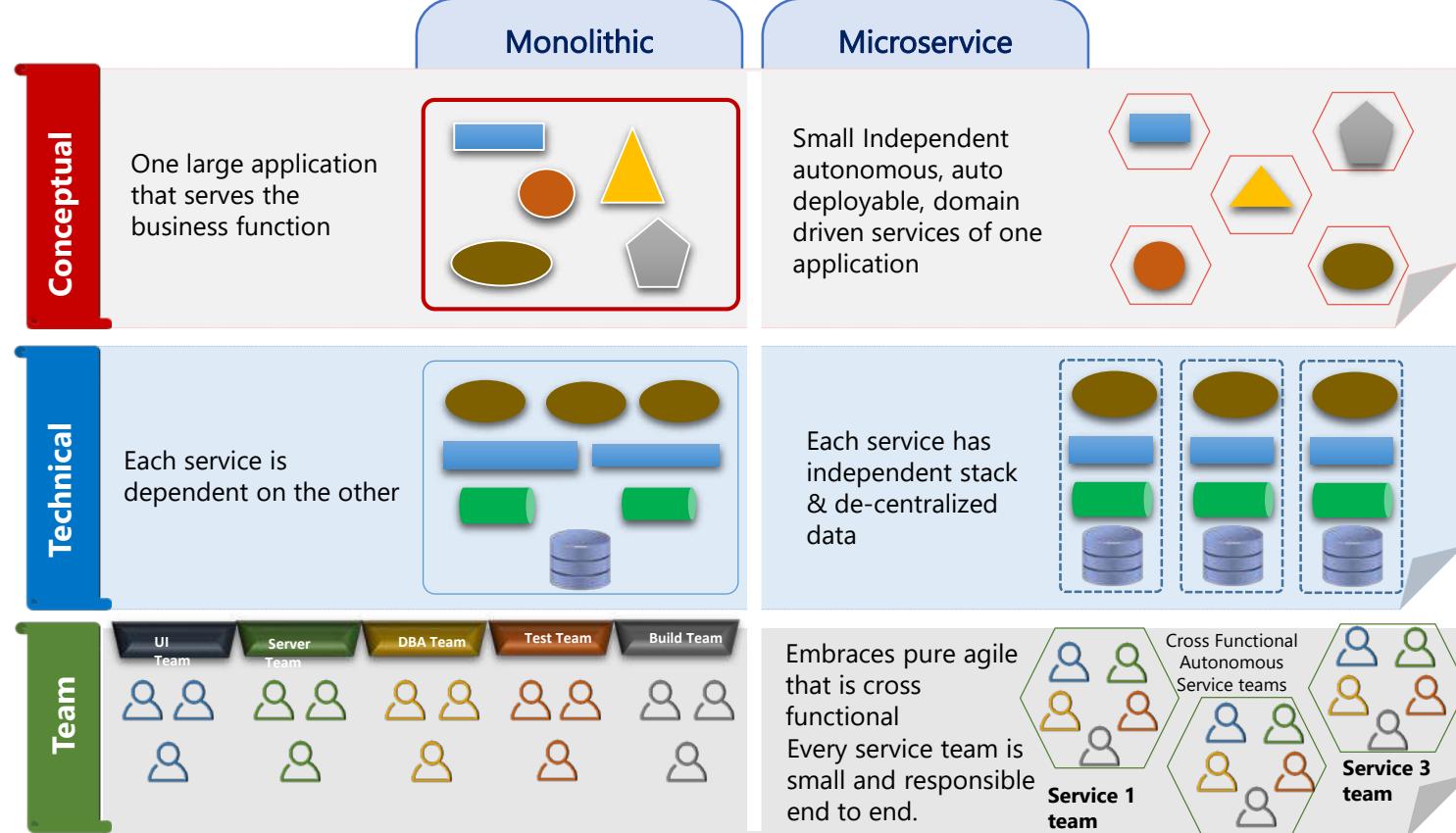
Scaling Comparison

Monolithic



Microservices





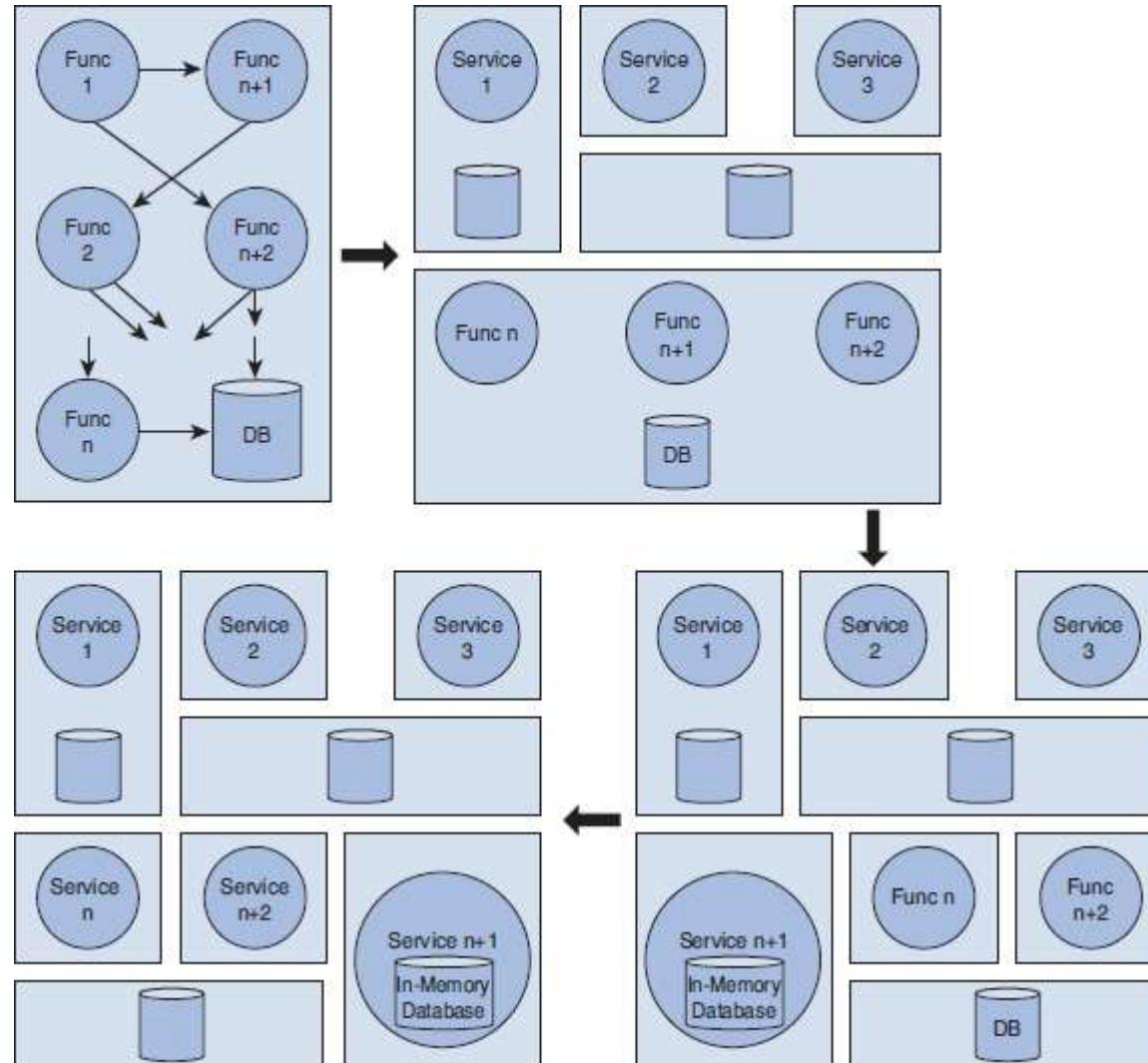
<http://blog.arungupta.me/monolithic-microservices-refactoring-javaee-applications/>

Mono Vs Micro Services (Developer's Perspective)

	Monolith	Microservice
Archives	1	5 (Contracts, Order, User, Catalog, Web)
Web pages	8	8
Config Files	4 (persistence.xml, web.xml, load.sql, template.xhtml)	12 (3 per archive)
Classes	12	26 (Service registration/discovery, Application)
Archive Size	24 KB	~52 KB total



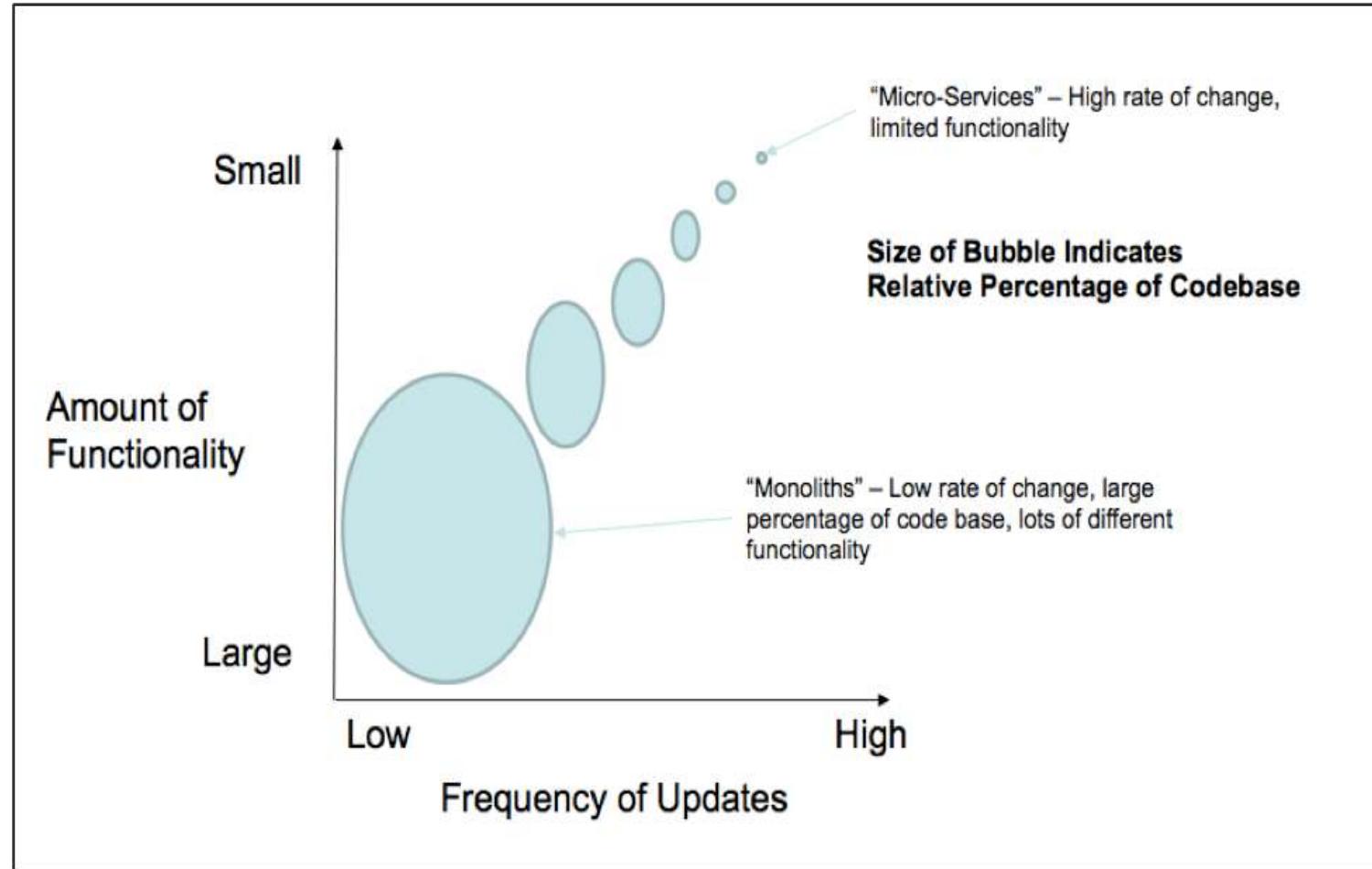
Transition from Monolithic to Microservices



- The transition is a journey.
- Determine which functions are highly used. Convert the highly used services or application functionality as microservices first.
- Design your microservice in such a way that it does one and only one thing well.
- Determining the boundary is often going to be hard.
- Domain-specific programming languages can be employed.
- You can leverage the new language (such as go) for a given microservice while the rest of the application can still be using a different language.
- If any such component in the application may benefit from NoSQL, identify them and migrate appropriately.



When Should You Split Services?



Factors to Consider:

- **Developer Throughput**
 - Frequency of Change
 - Degree of Reuse
 - Team Size
 - Specialized Skills
- **Availability and Fault Tolerance**
 - Desired Reliability
 - Criticality of Business
 - Risk of Failure
- **Scalability Considerations**
 - Scalability of Data
 - Scalability of Services
 - Dependency on other Services
- **Cost Considerations**

<http://akfpartners.com/growth-blog/when-should-you-split-services>



MICROSERVICES IN DETAIL

SWITCHING TO MICROSERVICES

ELEMENTS OF MICROSERVICES ARCHITECTURE

INTER-PROCESS COMMUNICATION

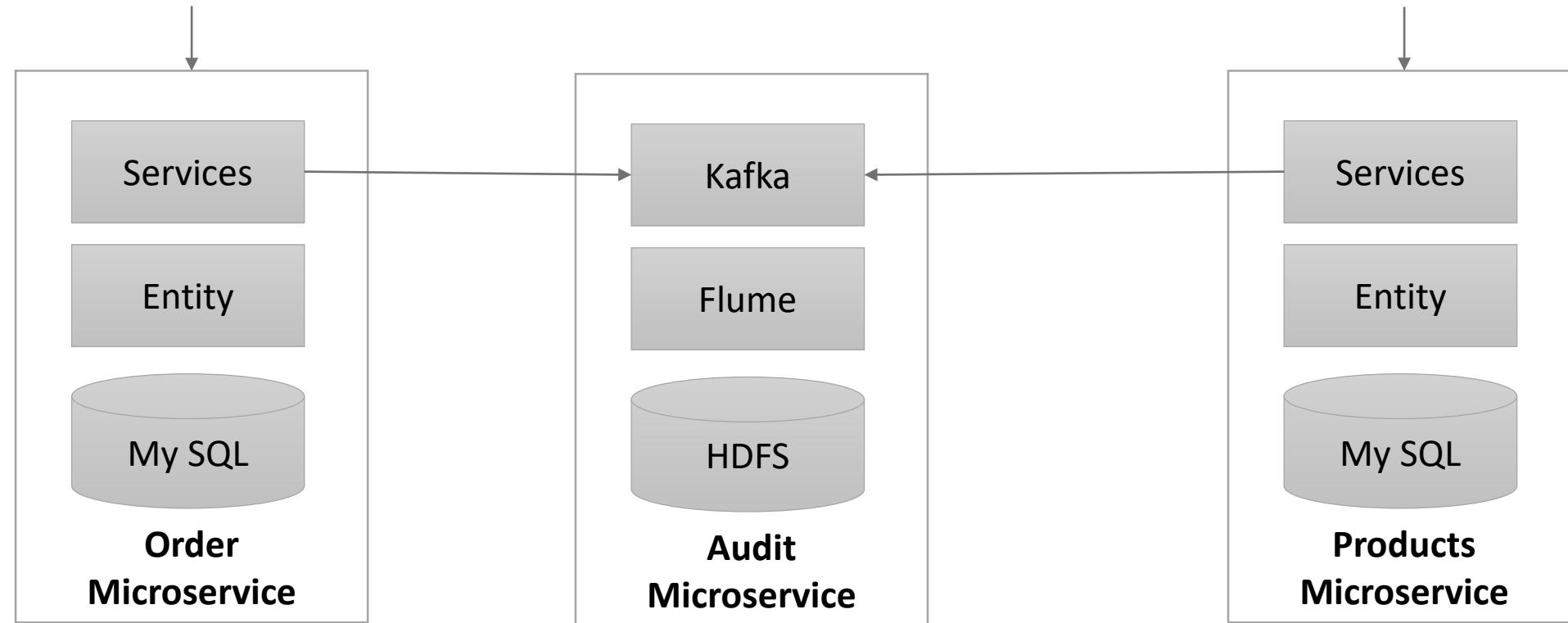
IMPLEMENTATION MECHANISM



Why Microservices? - 1

▪ Supports polyglot architecture

- With microservices, architects and developers can choose fit for purpose architecture/technology for each microservice in a cost effective manner.

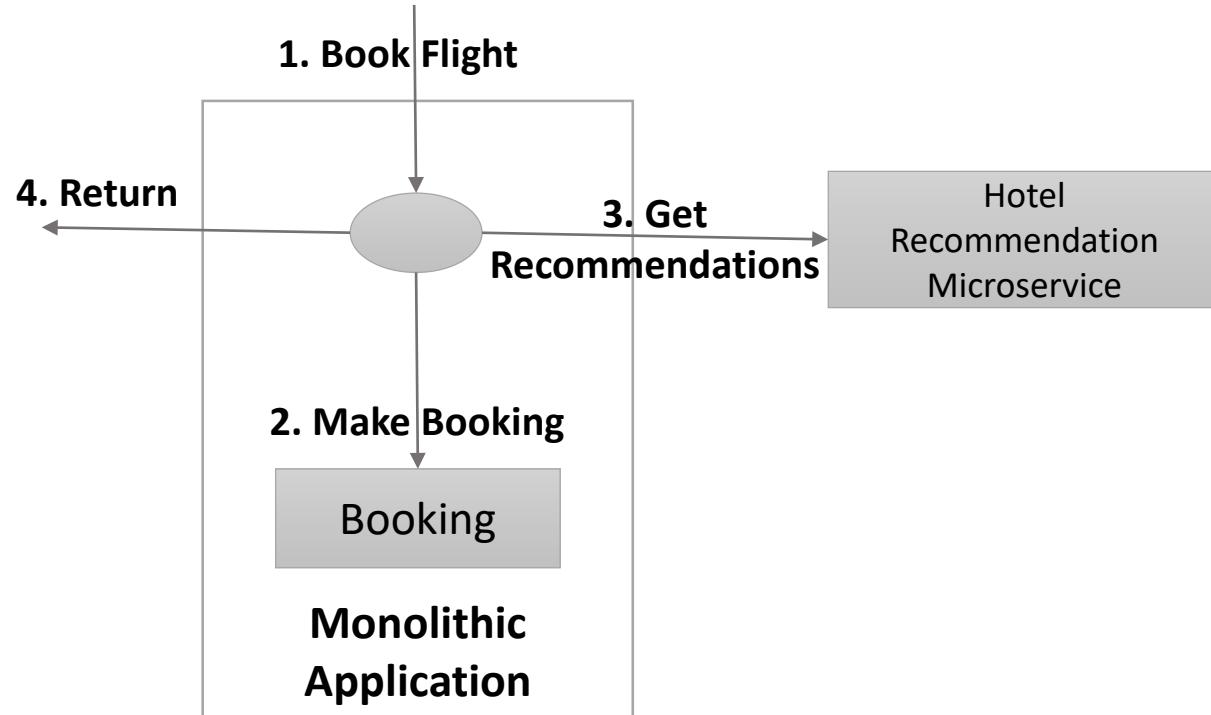




Why Microservices? - 2

▪ Enabling experimentation and innovation

- Microservices are one of the key enablers for enterprises to do disruptive innovation by offering the ability to experiment and fail fast.



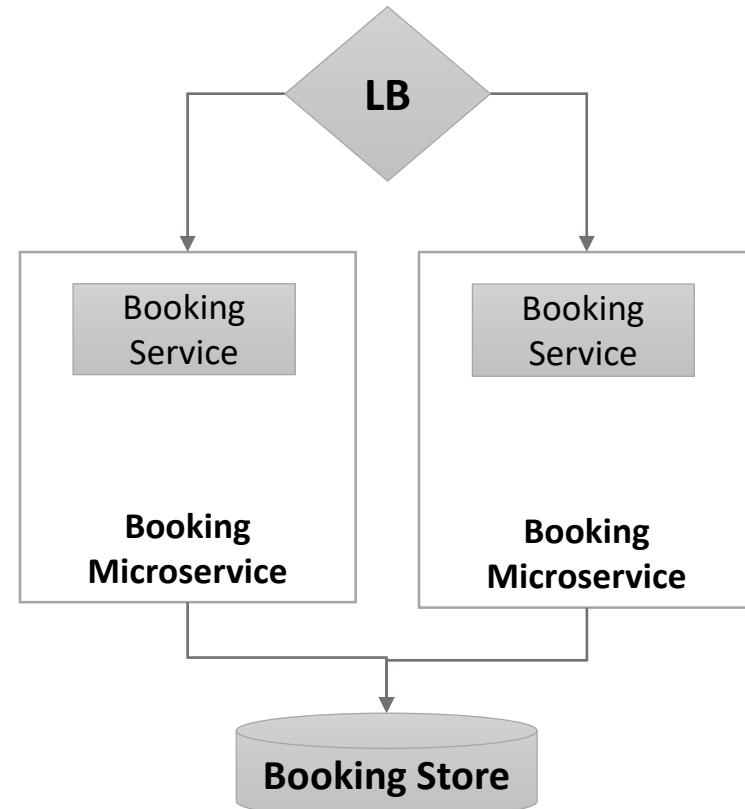
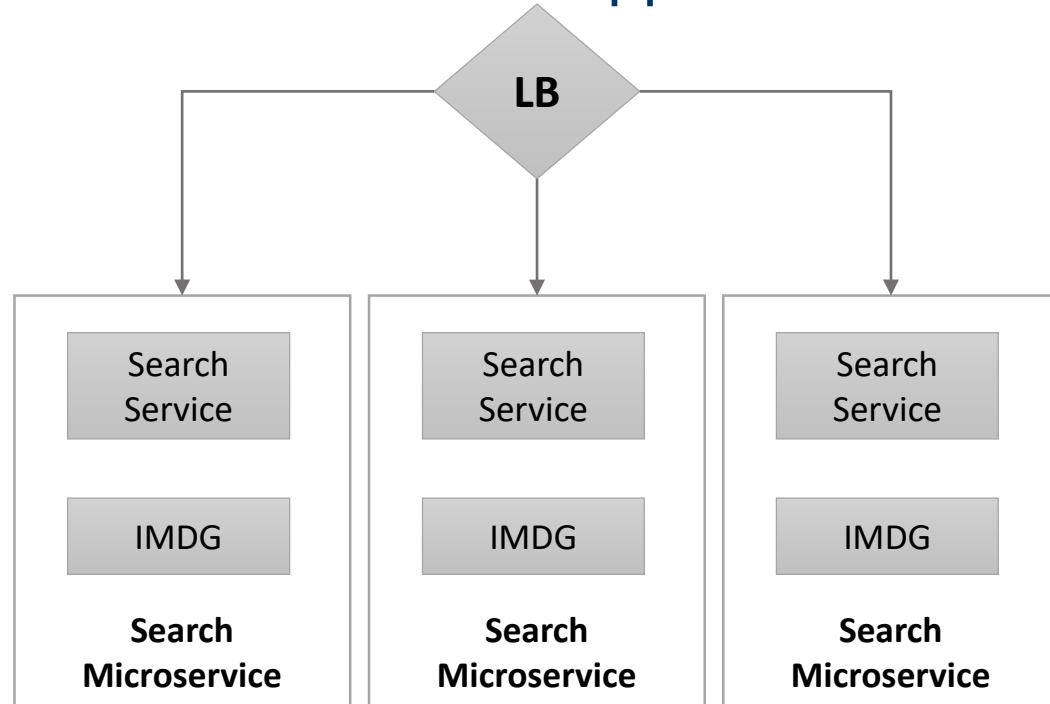
It is convenient to write a microservice that can be plugged into the monolithic applications booking flow rather than incorporating this requirement in the monolithic application itself



Why Microservices? - 3

▪Elastically and selectively scalable

- As microservices are smaller units of work, they enable us to implement selective scalability. Scalability requirements may be different for different functions in an application.

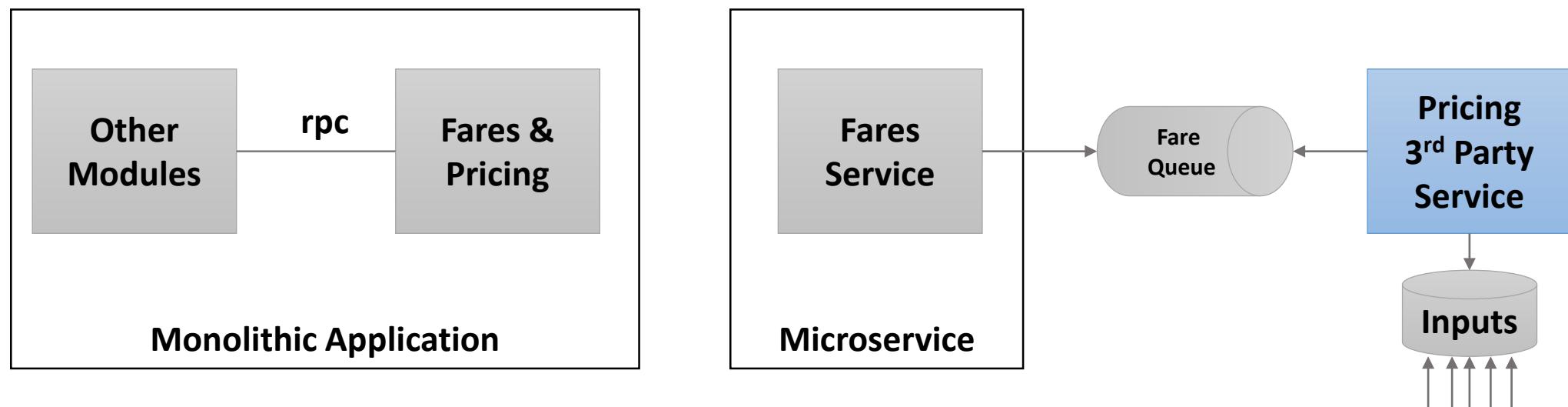




Why Microservices? - 4

▪ Allowing substitution

- Microservices are self-contained, independent deployment modules enabling the substitution of one microservice with another similar microservice.
- Architecturally, a microservice can be easily replaced by another microservice developed either in-house or even extended by a microservice from a third party.

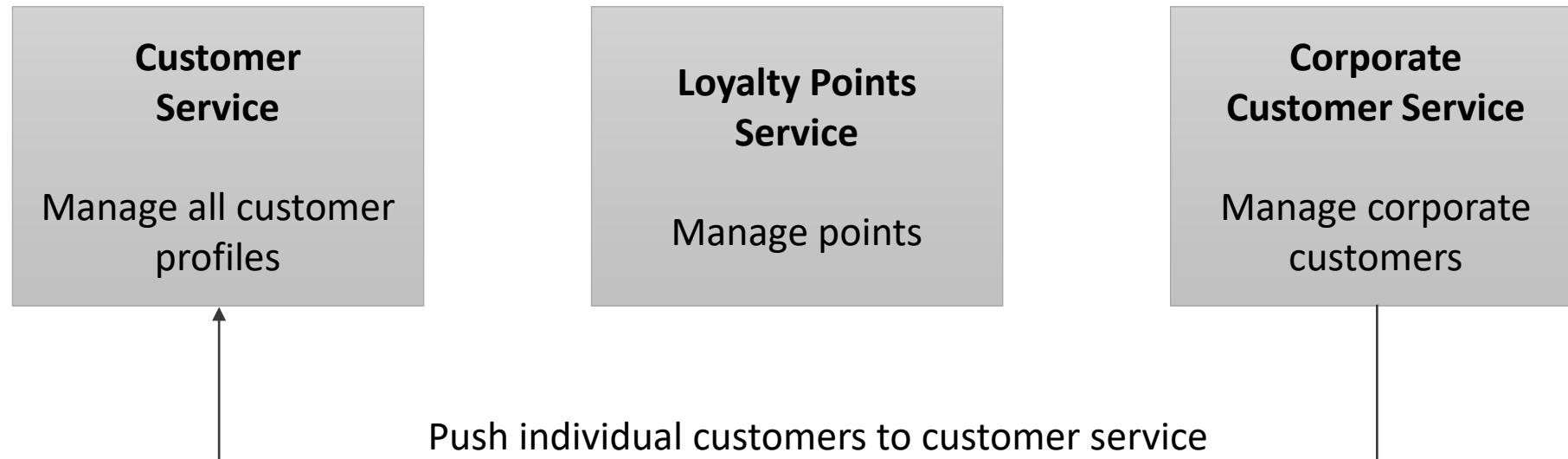




Why Microservices? - 5

▪Enabling to build organic systems

- Microservices help us build systems that are organic in nature. This is significantly important when migrating monolithic systems gradually to microservices.



As shown in the preceding diagram, in a microservices-based architecture, customer information would be managed by the Customer microservice and loyalty by the Loyalty Points microservice.



Why Microservices? - 6

▪Helping reducing technology debt

- While older versions of the services are running on old versions of technologies, new service developments can leverage the latest technologies. The cost of migrating microservices with end-of-life technologies is considerably less compared to enhancing monolithic applications.

Development Timelines

Development of
Microservice 1
(on V1 of technology)

Development of
Microservice 2
(on V2 of technology)

Development of
Microservice 3
(on V3 of technology)

Development of
Microservice 4
(on V4 of technology)

Migration of
Microservice 1
(on V3 of technology)

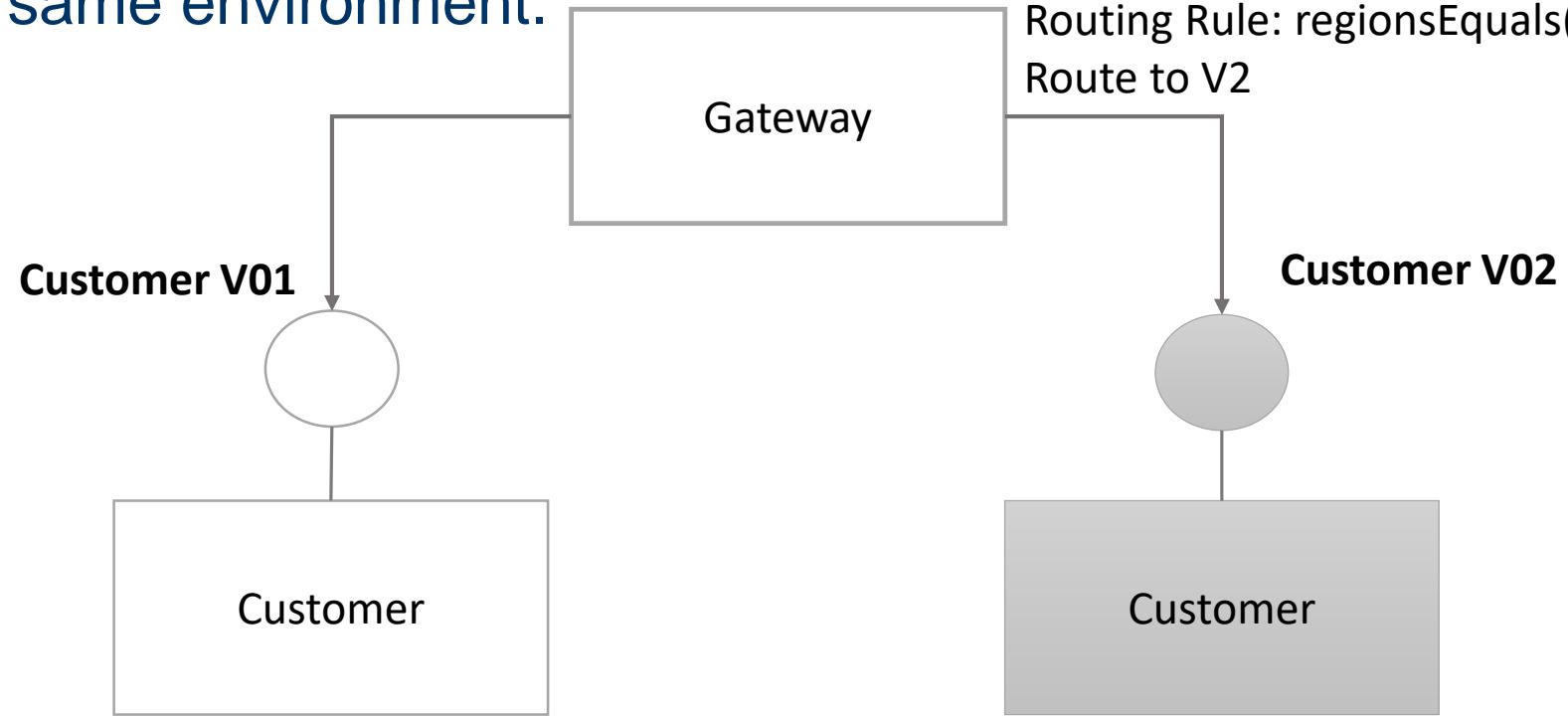
Migration of
Microservice 2
(on V4 of technology)



Why Microservices? - 7

▪ Allowing the coexistence of different versions

- As microservices package the service runtime environment along with the service itself, this enables having multiple versions of the service to coexist in the same environment.

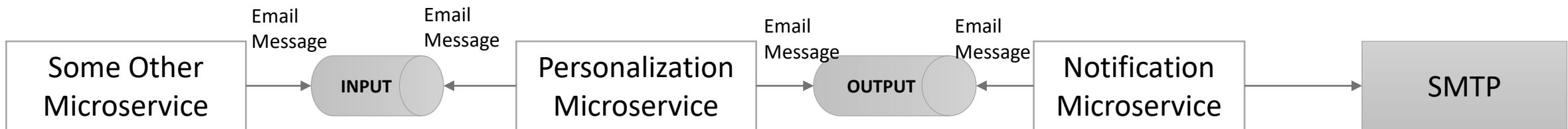




Why Microservices? - 8

▪ Supporting the building of self-organizing systems

- A self-organizing system support will automate deployment, be resilient, and exhibit self-healing and self-learning capabilities.
- In a well-architected microservices system, a service is unaware of other services. It accepts a message from a selected queue and processes it.
- At the end of the process, it may send out another message, which triggers other services. This allows us to drop any service into the ecosystem without analyzing the impact on the overall system.

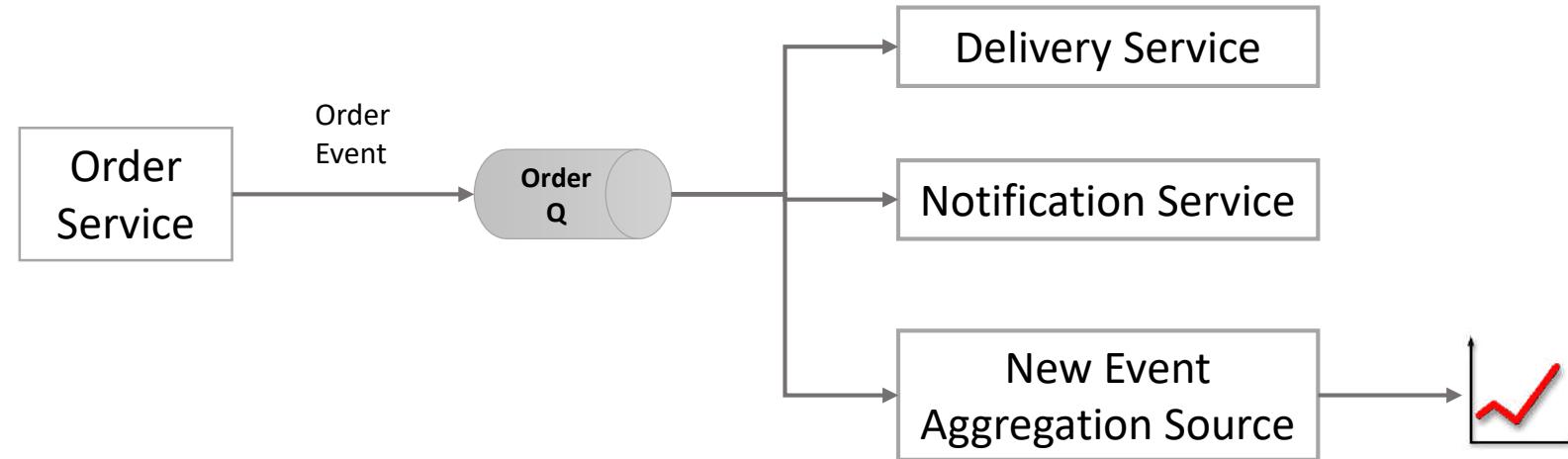




Why Microservices? - 9

▪ Supporting event-driven architecture

- A well-architected microservice always works with events for both input and output. These events can be tapped by any service. Once extracted, events can be used for a variety of use cases.



Order Event is already published whenever an order is created. This means that it is just a matter of adding a new service to subscribe to the same topic, extract the event, perform the requested aggregations, and push another event for the dashboard to consume.



Why Microservices? – 10.

▪ Enabling DevOps

- DevOps is widely adopted as a practice in many enterprises, primarily to increase the **speed of delivery and agility**.
- DevOps advocates to have agile development, high-velocity release cycles, **automatic** testing, automatic infrastructure provisioning, and automated deployment.
- Small footprint microservices are more **automation-friendly** and therefore can more easily support these requirements.
- Microservices also enable smaller, focused **agile teams** for development. Teams will be organized based on the boundaries of microservices.



Microservices early adopters - 1

▪ **Netflix (www.netflix.com):**

- Netflix, an international on-demand media streaming company, is a pioneer in the microservices space.
- Netflix transformed their large pool of developers developing traditional monolithic code to smaller development teams producing microservices.

▪ **Uber (www.uber.com):**

- Uber then moved to SOA-based architecture by breaking the system into smaller independent units.
- Each module was given to different teams and empowered them to choose their language, framework, and database.

▪ **Airbnb (www.airbnb.com):**

- Airbnb developed their own microservices or SOA ecosystem around its legacy monolithic services.

▪ **Orbitz (www.orbitz.com):**

- Orbitz went through continuous architecture changes. Later, Orbitz broke down their monolithic to many smaller applications.



Microservices early adopters – 2.

- **eBay (www.ebay.com):**

- eBay moved to smaller decomposed systems based on Java and web services. They employed database partitions and functional segregation to meet the required scalability.

- **Amazon (www.amazon.com):**

- Amazon separated out the code as independent functional services, wrapped with web services, and eventually advanced to microservices.

- **Gilt (www.gilt.com):**

- Gilt went through an architecture overhaul by introducing Java and polyglot persistence. Later, Gilt moved to many smaller applications using the microservices concept.

- **Twitter (www.twitter.com):**

- Twitter uses Scala and Java to develop microservices with polyglot persistence.

- **Nike (www.nike.com):**

- Nike moved to a microservices-based architecture that brought down the development cycle considerably.

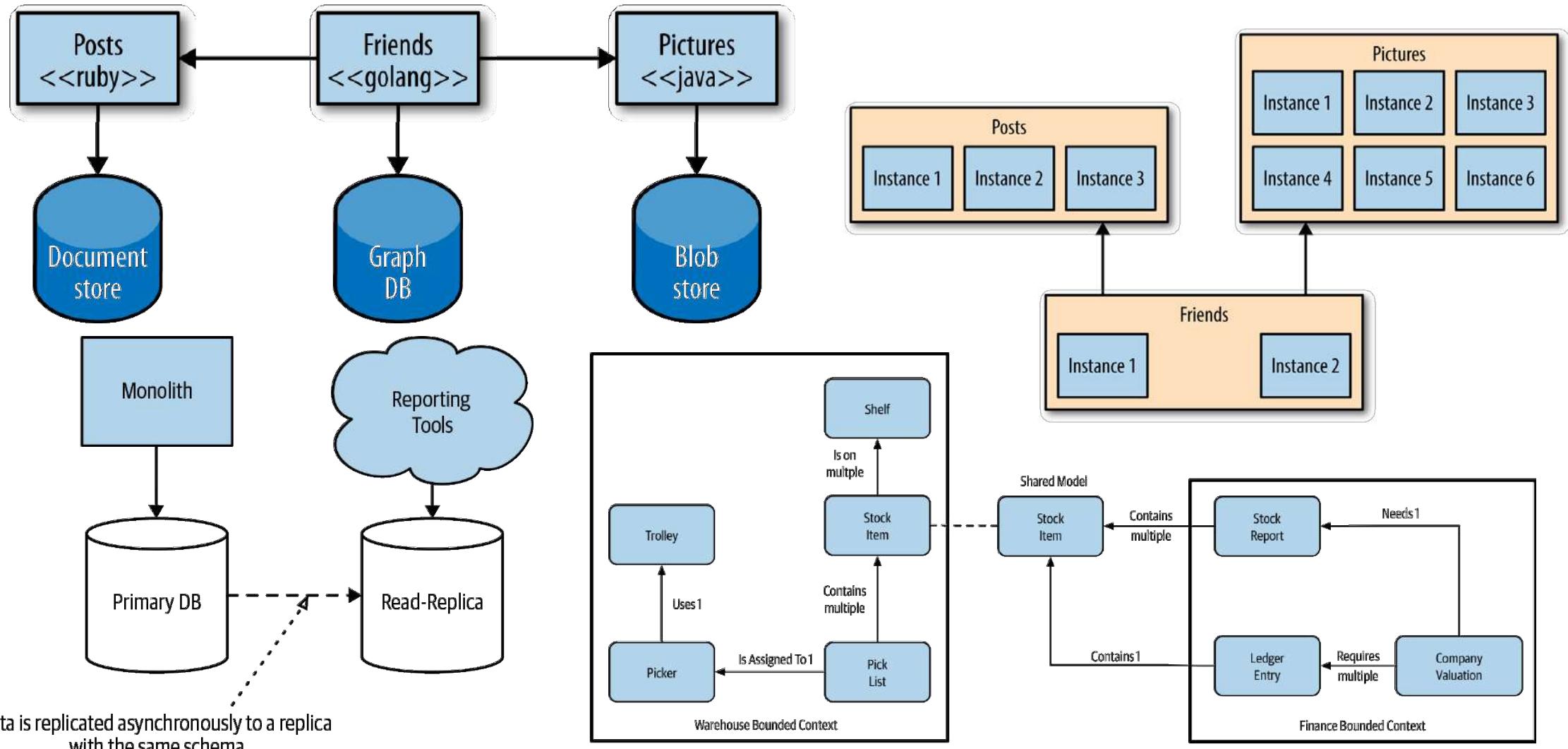


Microservices Decomposition Strategy

- An ***architectural style*** determines the components and connectors, together with a set of constraints on how they can be combined.
 - A **service** is a standalone, independently deployable software component that implements some useful functionality.
 - All interaction with a service happens via its **API**, which encapsulates its implementation details.
- Three step process to define a microservice architecture:
 1. **Step 1: Identify system operations.**
 - Distill the application's requirements into the key requests/operations.
 2. **Step 2: Identify services.**
 - Define services corresponding to business capabilities or around domain-driven design subdomains.
 3. **Step 3: Define service APIs and collaborations.**
 - Assign each system operation identified in the first step to a service.



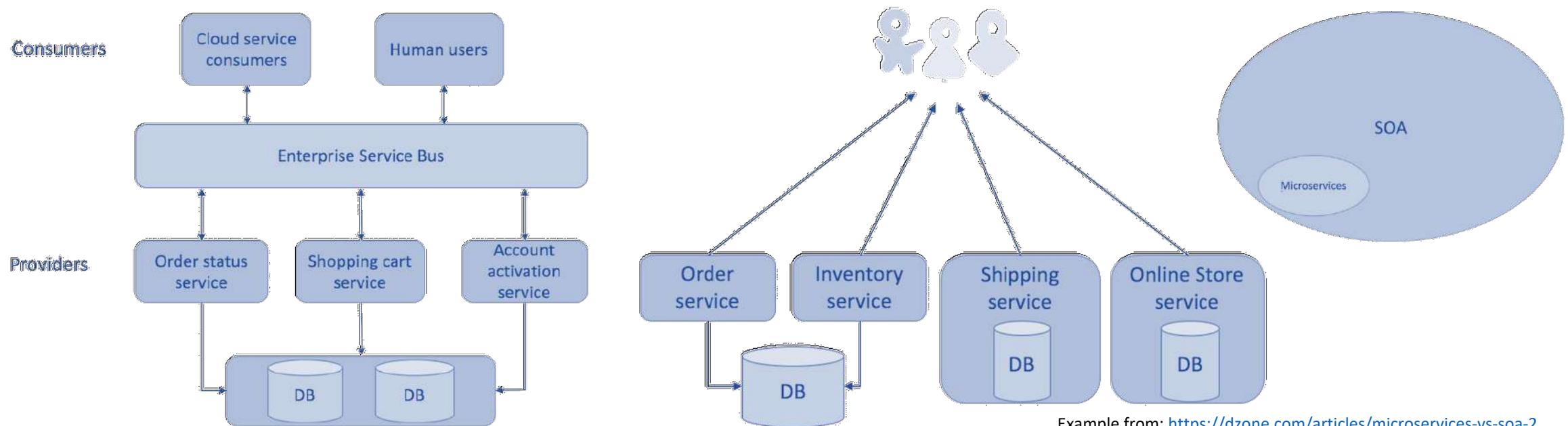
Enabling heterogenous technologies via Microservice





Characteristics of microservices architecture - 1

Business-oriented: It is critical to design microservices with the ultimate business objective in mind. This might require teams across organizational boundaries to come together and design the services jointly, rather than using microservices to route calls between teams. Micro service should be independently deployable, or be able to shut-down a service when it is not required in the system and that should not have any impact on other services.



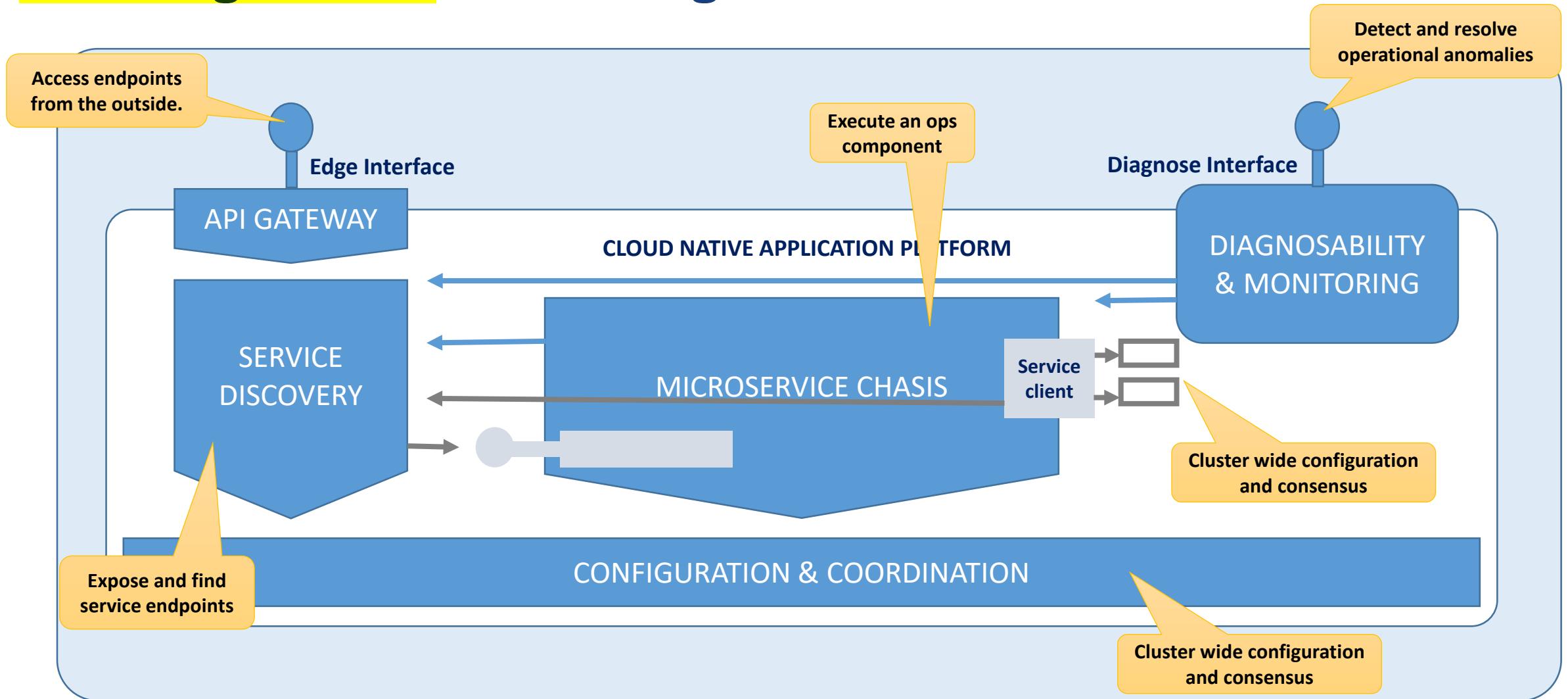


Characteristics of microservices architecture - 2

- **Design for failure:** Isolated failures are inevitable, but the design goal is to keep the system functioning for as long as possible. Engineering practices, such as fault modeling and fault injection, are part of a continual release process for reliability. There are several design patterns to design for failure and stability.
- **Decentralized data management:** Microservices potentially deal with transactions that are spread across multiple services. To avoid distributed transactions where possible microservices manage their own databases enabling polyglot persistence.
- **Discoverability:** Microservices architecture requires making services reliable and fault tolerant. Most service discovery models rely on a registry service to provide service discovery capabilities (example Zookeeper, Consul, and Eureka) and few by advertising their cloud controllers on Cloud Foundry (example BlueMix).
- **Inter-service communication design:** Synchronous HTTP-based mechanisms, such as Representational State Transfer (REST), SOAP, or WebSockets, for event-driven request/response. Asynchronous messaging using a message broker for passive processing.
- **Evolutionary design:** Microservices architecture enables an evolutionary design approach.

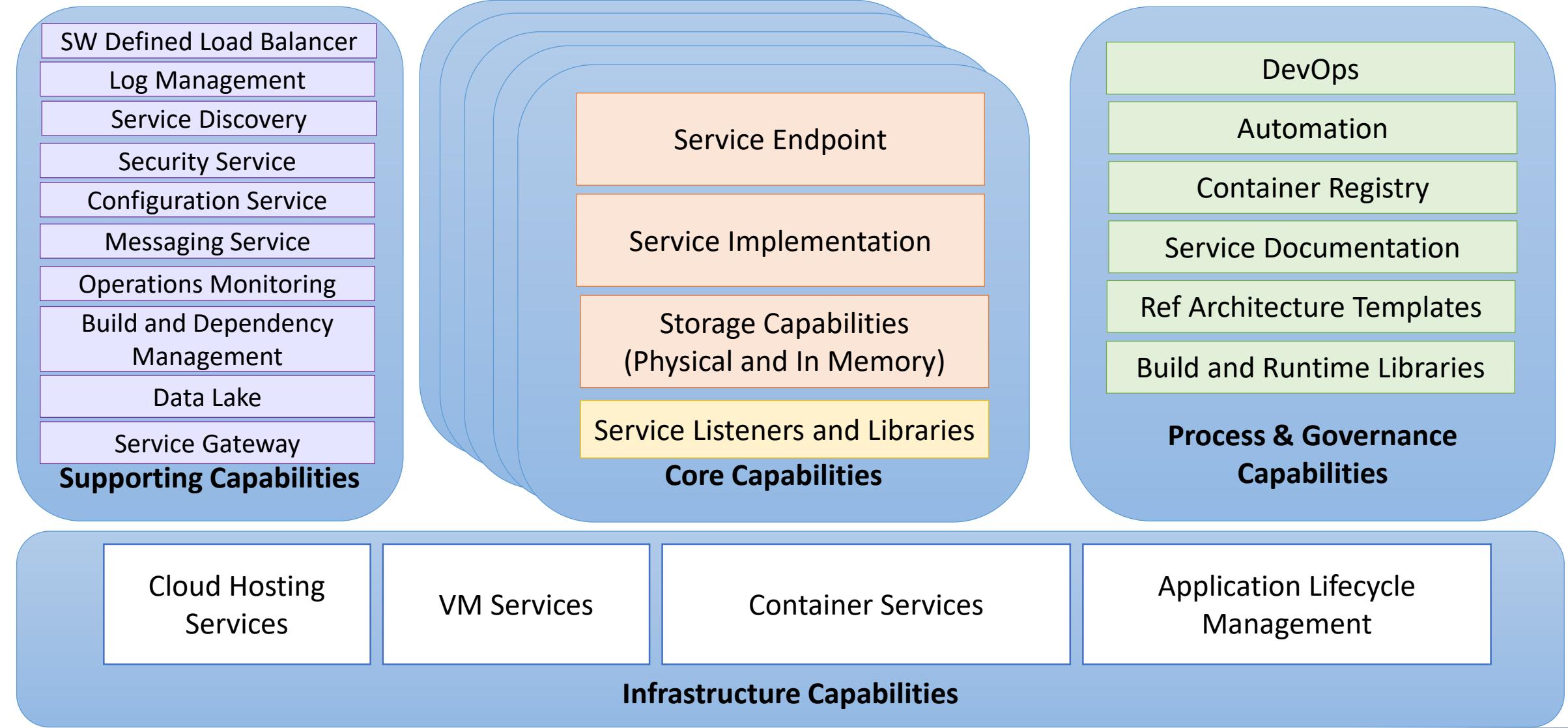


Building blocks of hosting a Microservice

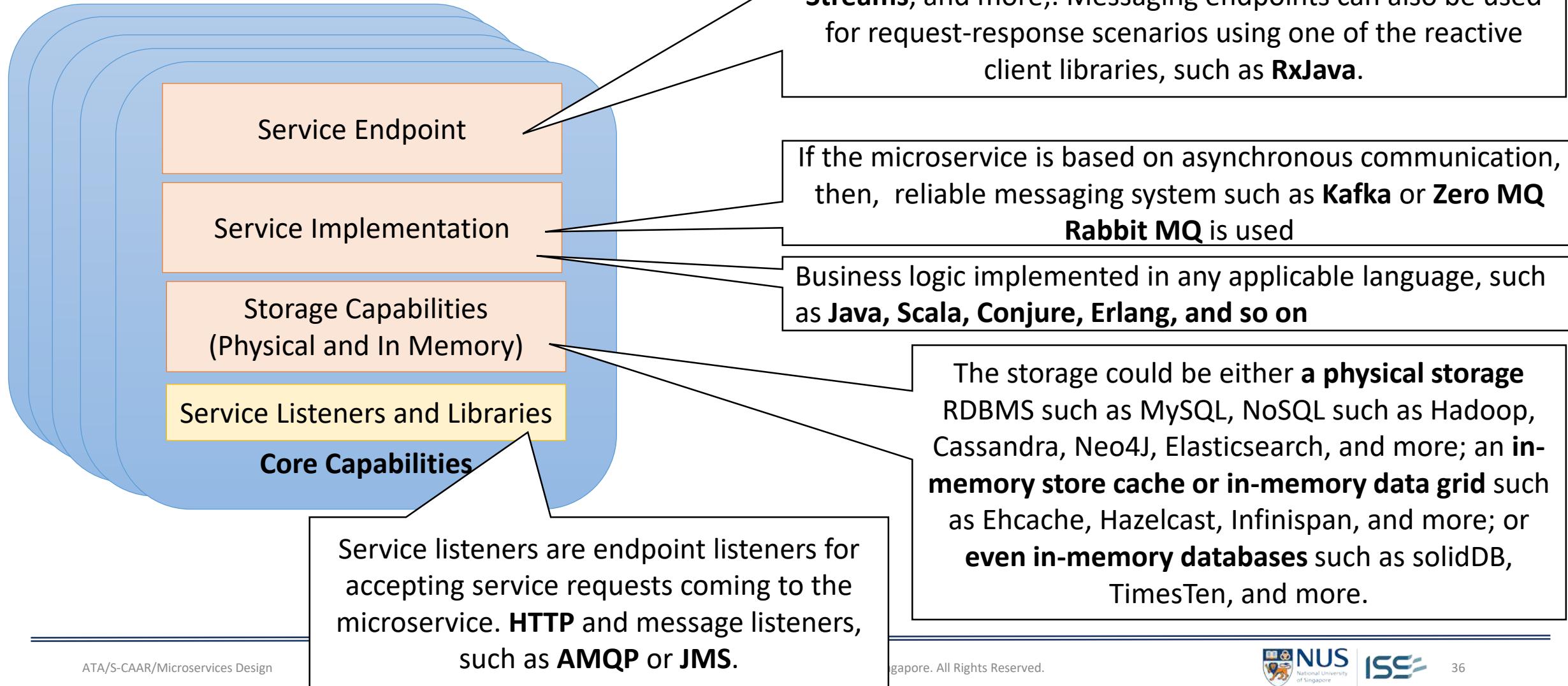




Microservices Capabilities



Microservices Tools

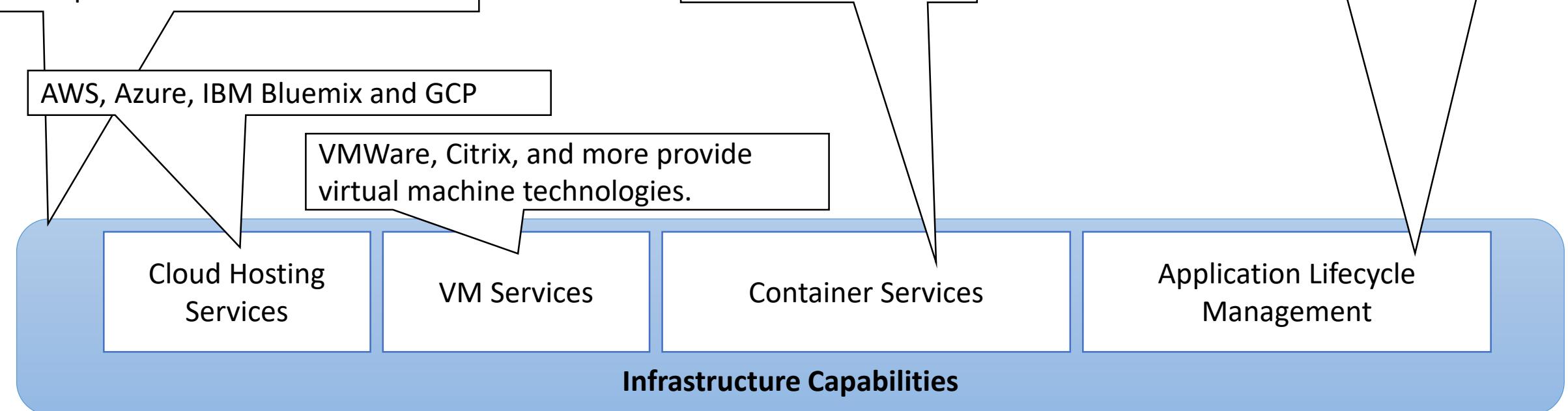


Microservices Tools

Platform as a Service (PaaS) vendors such as Red Hat OpenShift offer all these capabilities out of the box.

Docker, Rocket, and LXD are some of the containerized technologies.

Apache Mesos, Rancher, CoreOS, and Kubernetes are examples of popular container orchestration tools. These tools are also called container scheduler and, sometimes, container as service. Container orchestration tools also helps us manage application life cycle activities, including application availability and constraints-based deployment





Microservices Tools

Development processes, continuous integration, automated QA checks, automated delivery pipelines, automated deployments, and automatic infrastructure provisioning.

Test automation and continuous delivery approaches such as AB Testing, Future Flags, Canary Testing, Blue-Green deployments, and Red-Black deployments

Docker Hub, Google Container Repository, Core OS Quay, and Amazon EC2 container registry are some of the examples. Registry can be either private or public

Swagger, RAML, or API blueprint are helpful in achieving good microservices documentation.

DevOps

Automation

Container Registry

Service Documentation

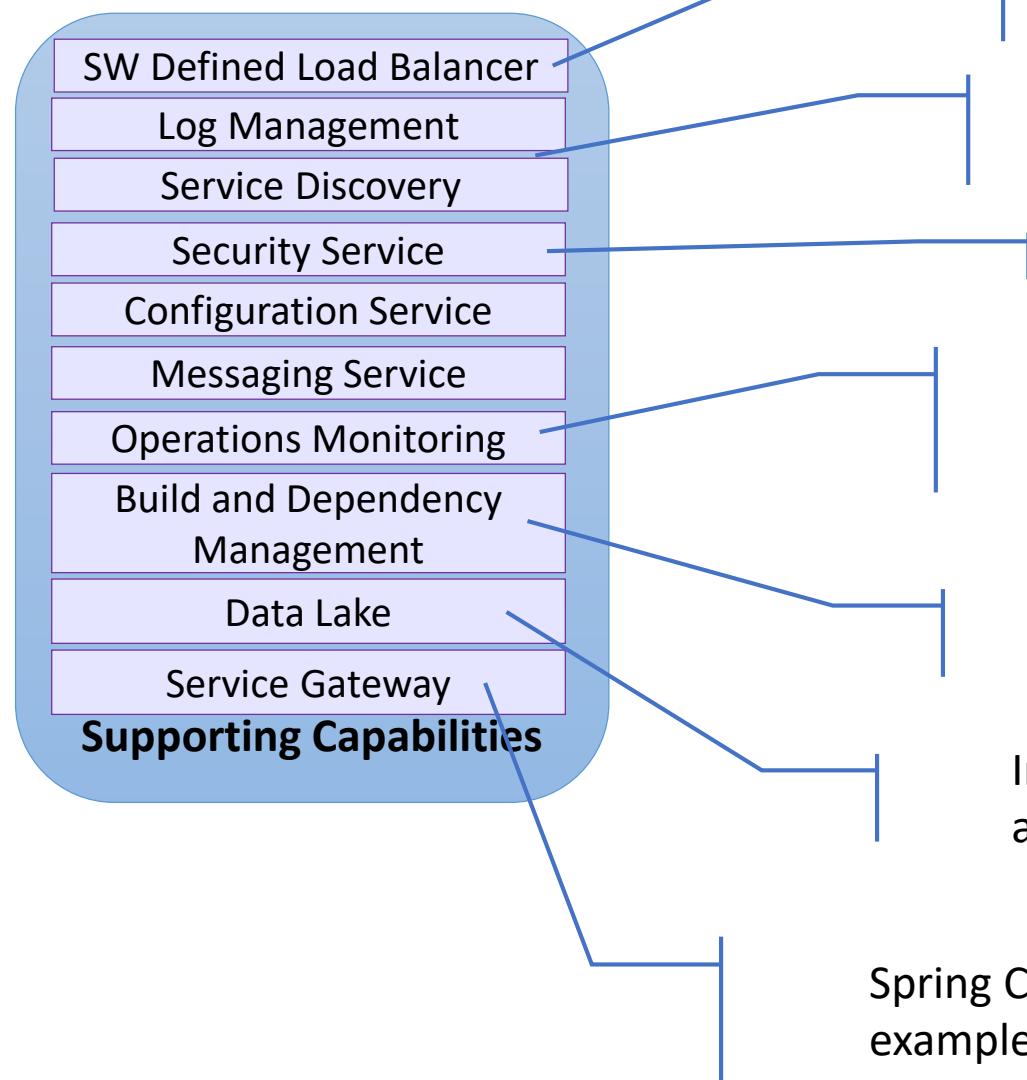
Ref Architecture Templates

Build and Runtime Libraries

Process & Governance Capabilities



Microservices Tools



A combination of **Ribbon**, **Eureka**, and **Zuul** provide this capability in **Spring Cloud Netflix**. Alternately, **Marathon Load Balancer**.

Eureka from Spring Cloud, **Zookeeper**, and **Etcd** are some of the service registry tools available.

Spring Security and Spring Security OAuth

End-to-end monitoring tools, such as **AppDynamic**, **NewRelic**, and **Dynatrace**, and other tools, such as **Statd**, **Sensu**, and **Simian Viz**, could add value in microservices monitoring. Tools such as **Datadog** help us manage infrastructure efficiently.

AppDynamic, **Cloud Craft**, **Light Mesh**, and **Simian Viz** are some of the other tools.

Ingestion tools such as **Spring Cloud Data Flow**, **Kafka**, **Flink**, **Flume**, and so on. Block Storage/File Storage / Object Storage – **HDFS**, **GFS**

Spring Cloud Zuul, Mashery, Apigee, Kong, WSO2, and 3scale are some examples of the API gateway providers.



Key Decisions in Wiring Web Services

1. Protocol

- **HTTP** is the gold standard. HTTP Request / Response will be book marked.
- **AJAX** based XMLHttpRequest / Response would also be considered and managed in responsive web scenarios.
- Sometimes special protocols may also be useful.

2. Web Services Standard

- **RESTful** is widely accepted and recommended.
- **SOAP** is bulky enough that it requires client- and server-side implementation.
- Data is an open protocol used for building and consuming RESTful APIs.

3. Message Format

- There are plenty of commonly used and entirely acceptable message formats to choose from, including **XML**, **RSS**, and **JSON**.



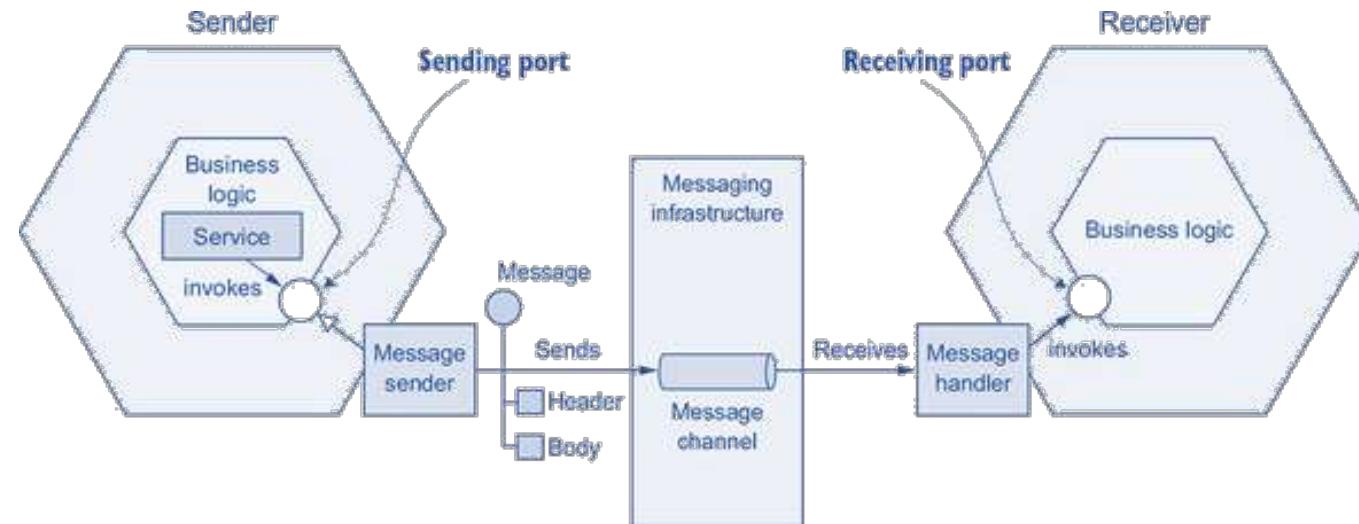
Message Formats

- Text-based message formats
 - ❖ Text-based formats such as JSON and XML are self describing.
 - Structure of XML documents is specified by an XML schema.
 - For JSON the equivalent is JSON Schema standard (<http://json-schema.org>)
 - Messages tend to be verbose and there is overhead of parsing text.
- Binary message formats
 - ❖ There are several different binary formats to choose from.
 - Popular formats include Protocol Buffers
(<https://developers.google.com/protocol-buffers/docs/overview>)
 - Avro (<https://avro.apache.org>).



Message Channels

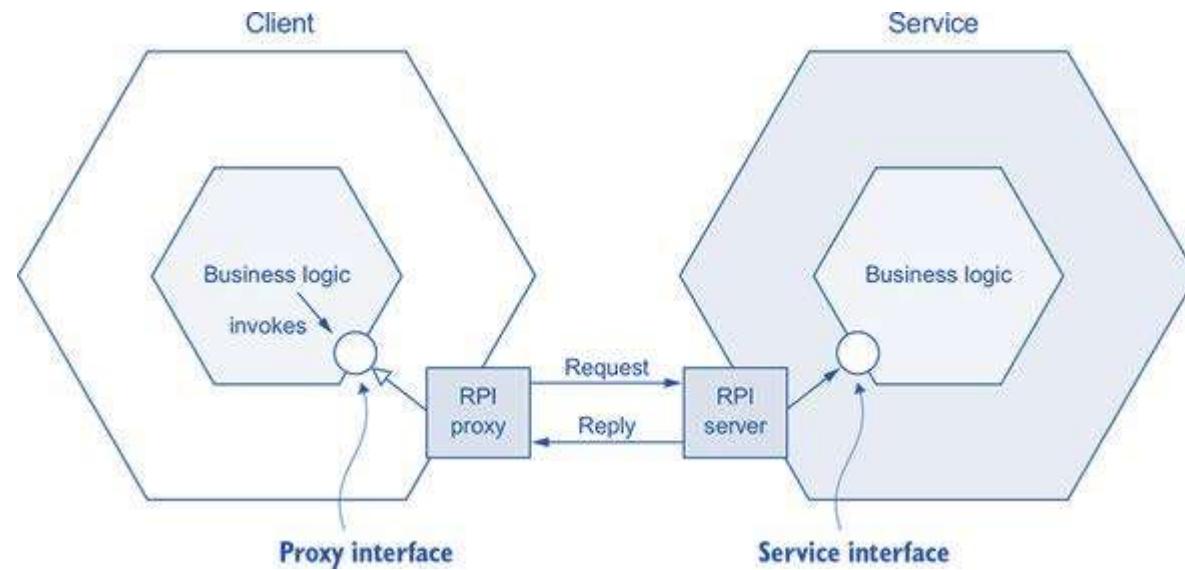
- The business logic in the sender invokes a sending port interface, which is implemented by a message sender adapter.
 - The message sender sends a message to a receiver via a message channel.
 - The message channel is an abstraction of messaging infrastructure.
 - A message handler adapter in the receiver is invoked to handle the message.
 - It invokes the receiving port interface implemented by the receiver's business logic.





Remote procedure invocation

- The client's business logic invokes an interface that is implemented by an RPI proxy adapter class. The RPI proxy class makes a request to the service. The RPI server adapter class handles the request by invoking the service's business logic.





gRPC

- Define RPC APIs using a Protocol Buffers-based IDL
 - One challenge with using REST is that because HTTP only provides a limited number of verbs.
 - An IPC technology that avoids this issue is gRPC (www.grpc.io), a framework for writing cross-language clients and servers.
- gRPC benefits:
 - It's straightforward to design an API that has a rich set of update operations.
 - It has an efficient, compact IPC mechanism, especially when exchanging large messages.
 - Bidirectional streaming enables both RPI and messaging styles of communication.
 - It enables interoperability between clients and services written in a wide range of languages.
- gRPC drawbacks:
 - It takes more work for JavaScript clients to consume gRPC-based API than REST/JSON-based APIs.
 - Older firewalls might not support HTTP/2.



Inter-process communication in Microservices

▪ ***Request/response***

- A service client makes a request to a service and waits for a response.
- The client expects the response to arrive in a timely fashion.
- It might even block while waiting.
- This is an interaction style that generally results in services being tightly coupled.

▪ ***Asynchronous request/response***

- A service client sends a request to a service, which replies asynchronously.
- The client doesn't block while waiting, because the service might not send the response for a long time.

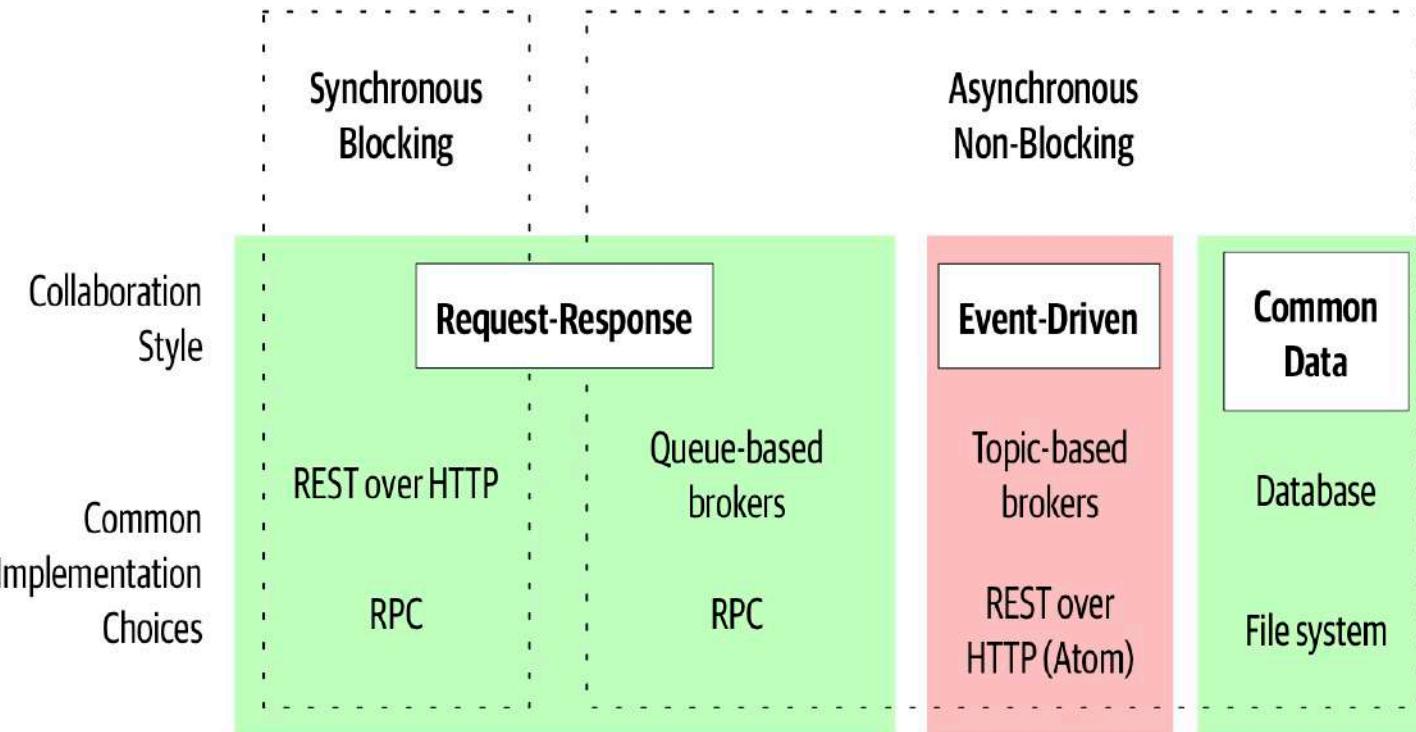
▪ ***One-way notifications***

- A service client sends a request to a service, but no reply is expected or sent.

	one-to-one	one-to-many
Synchronous	Request/response	
Asynchronous	Asynchronous request/response One-way notifications	Publish/subscribe Publish/async responses



Styles of Microservice Communication



- **Synchronous Blocking:** A microservice makes a call to another microservice and blocks operation waiting for the response.
- **Asynchronous Non-Blocking:** The microservice emitting a call is able to carry on processing whether or not the call is received.
 - **Request-response:** A Microservice sends a request to another microservice asking for something to be done. It expects to receive a response to the request informing it of the result.
 - **Event-Driven:** Microservices emit events, which other microservices consume and react to accordingly. The microservice emitting the event is unaware of which microservices, if any, consume the events it emits.
 - **Common Data:** Not often seen as a communication style, microservices collaborate via some shared data source.



How to decide which style to choose?

- Both synchronous and asynchronous approaches have their own merits and constraints. It is not possible to develop a system with just one approach.
 - An effective approach in this case is to start with a synchronous request response, and refactor later to introduce an asynchronous style when there is value in doing that.
- An asynchronous style is always better in the microservices world, but identifying the right pattern should be purely based on merits.
 - Alternatively, use reactive programming frameworks to avoid complexity when modeling user-driven requests, modeled in an asynchronous style.



The REST maturity model

- Leonard Richardson defines a very useful maturity model for REST (<http://martinfowler.com/articles/richardsonMaturityModel.html>) :
 - Level 0— Clients of a level 0 service invoke the service by making HTTP POST requests to its sole URL endpoint.
 - Each request specifies the action to perform, the target of the action (for example, the business object), and any parameters.
 - Level 1— A level 1 service supports the idea of resources.
 - To perform an action on a resource, a client makes a POST request that specifies the action to perform and any parameters.
 - Level 2— A level 2 service uses HTTP verbs to perform actions.
 - GET to retrieve, POST to create, and PUT to update. The request query parameters and body, if any, specify the actions' parameters. This enables services to use web infrastructure such as caching for GET requests.
 - Level 3— The design of a level 3 service is based on the terribly named HATEOAS (Hypertext As The Engine Of Application State) principle.
 - The basic idea is that the representation of a resource returned by a GET request contains links for performing actions on that resource.



Discovery Services

- Two main challenges in micro-service architecture:
 1. Clients have to call multiple services at same time to achieve the same functionality that they previously got with just one call in a monolithic application.
 2. Clients will have to know the location of the services.

▪ API Gateway

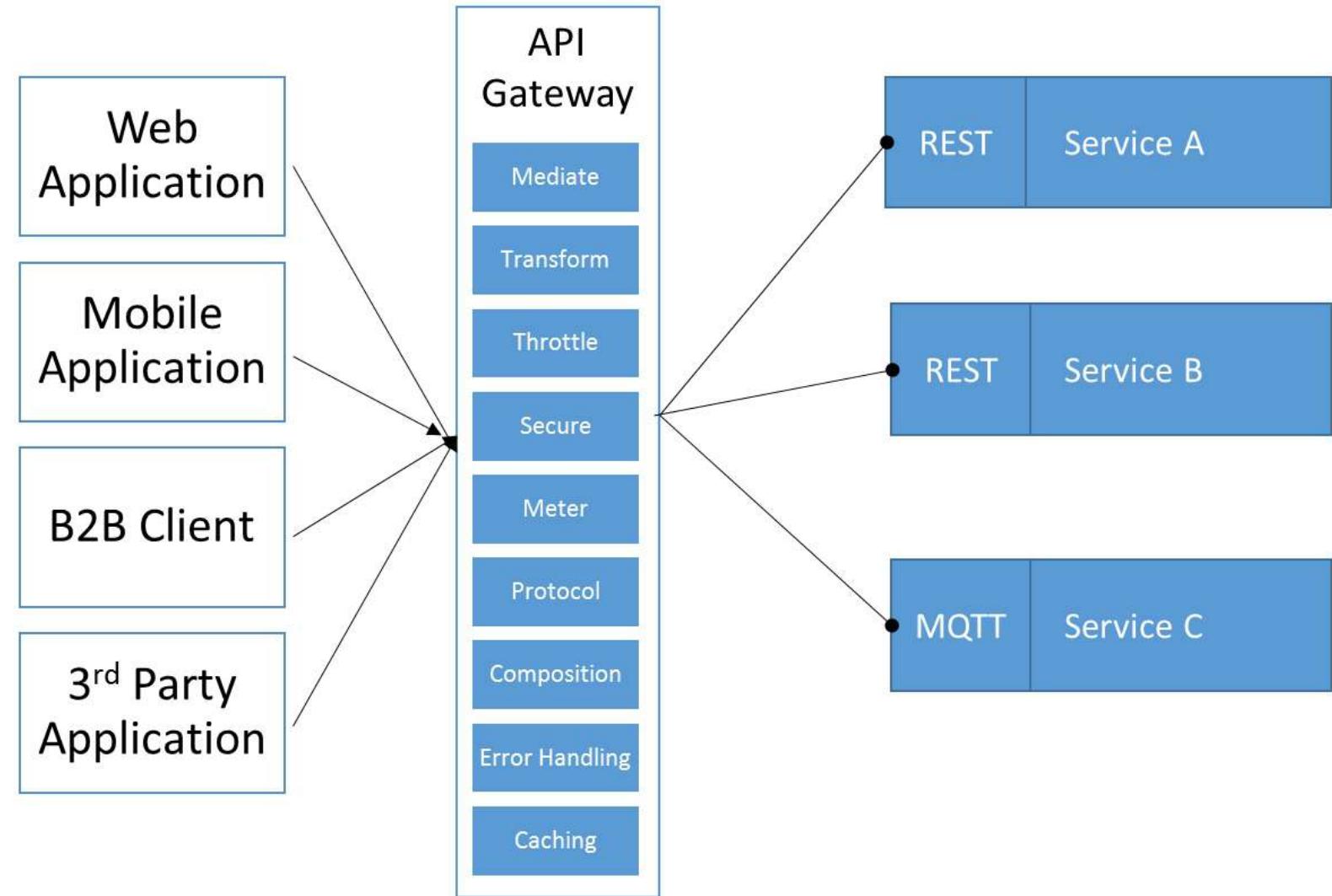
- An API gateway addresses both challenges and acts as an entry point for all calls.
- An API provides more flexibility for changing, combining, dividing, adding, or removing microservices as required. Increases efficiency and reduces internal complexity.
- An API gateway is responsible for receiving the client requests, calling all the required microservices and sending back the aggregated results from microservices to complete the client request.

▪ Service Registry

- With thousands of microservices in place, our API gateway also needs to know locations such as the IP addresses of all the services so that it can do its job.
- The idea behind service registry is that it provides a database of all the microservices and their locations, and the database can be queried when required.
- The microservice developer needs to create and maintain service registry.



API Gateway





Benefits of API gateway

▪ Separation of concerns:

- Insulates the microservice providers from the service consumers on the application side. This allows the separation of the application tier from the service requesting clients.

▪ Consumer oriented:

- API gateways provide a unified hub for a large number of APIs and microservices. This allows the consumer to focus on API utility instead of locating where a service is hosted, managing service request limits, security, and so on.

▪ API oriented:

- Provides an optimum API based on the type of the client and required protocols.

▪ Orchestration:

- Provides the ability to orchestrate multiple services calls into one API call, which in turn simplifies the logic for a client. Now, instead of calling multiple services, it can invoke one API.
- Fewer requests means less invocation overhead and improve the consumer experience overall. An API gateway is essential for mobile applications.

▪ Monitor:

- An API gateway also provides the ability to monitor API invocations, which in turn allows enterprises to evaluate the success of APIs and their usage.



Microservices Maintenance

▪ Supporting existing client implementations.

- Developers take care of backward compatibility of micro-service because chances are that one or more other microservices (consumers) are making use of this published interface for communication.
- Old versions are supported until the consumer microservices team changes its implementation to consume newer interface.

▪ Failsafe design.

- If a called web service is down, the simplest fail-safe design is to add timeout in client code.
- On the provider side, cover the error cases by returning proper error codes or, in some cases, default values. This practice also improves troubleshooting efforts.

▪ Monitoring.

- Proactively monitor microservices by calling each at regular intervals or through other methods.
- Take appropriate action if any of the microservices is down. Monitor calls cause extra traffic.
- Frameworks such as Marathon help to achieve availability and orchestration. Marathon has the heartbeat mechanism to detect failures and spin up another web server instance.

▪ Queue.

- Prefer publish/subscribe method when building asynchronous web services.
- The advantage being even when the service goes down, request can be picked from bus upon service recovery.

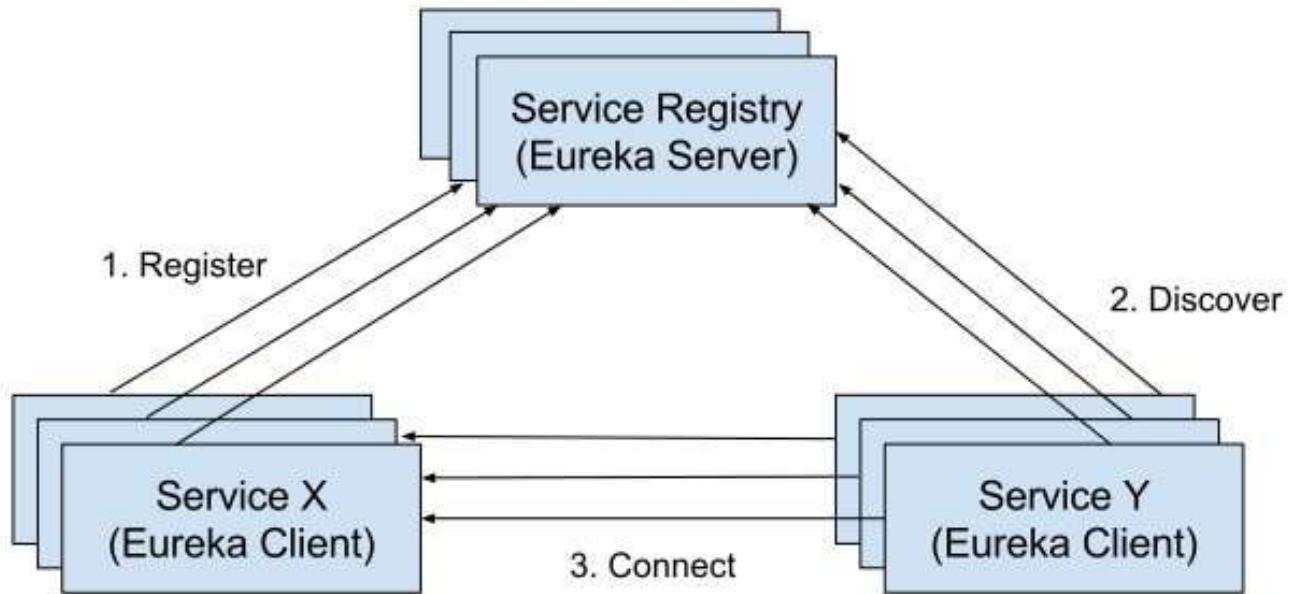


Use Semantic Versioning

- The Semantic Versioning specification (<http://semver.org>) is a useful guide to versioning APIs.
 - The Semantic Versioning specification (Semvers) requires a version number to consist of three parts: MAJOR.MINOR.PATCH.
 - MAJOR— When you make an incompatible change to the API
 - MINOR— When you make backward-compatible enhancements to the API
 - PATCH— When you make a backward-compatible bug fix
 - Backward-compatible changes are additive changes to an API:
 - Adding optional attributes to request
 - Adding attributes to a response
 - Adding new operations



Microservice registry and discovery with Eureka

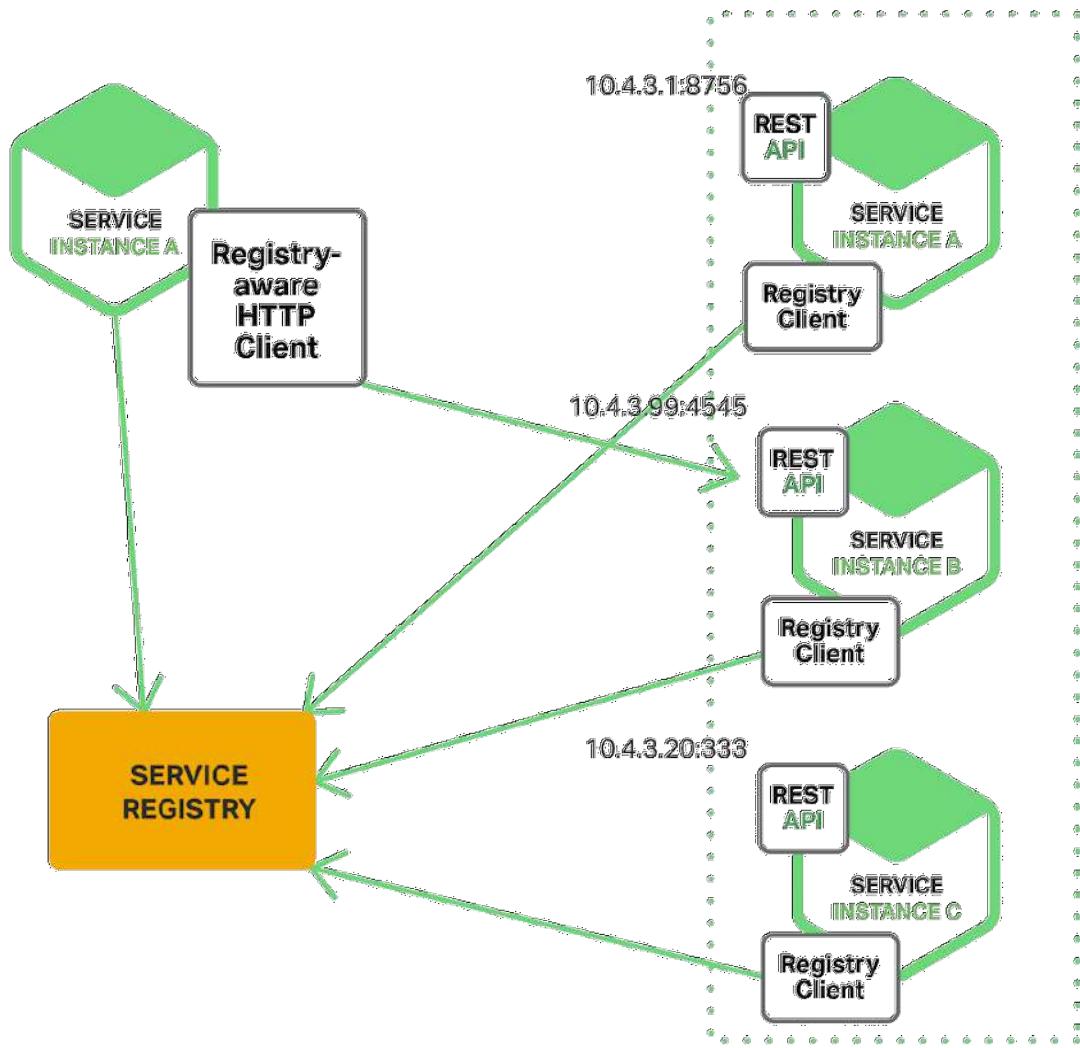


We can implement service discovery using two approaches:

- client-side service discovery
- server-side service discovery.



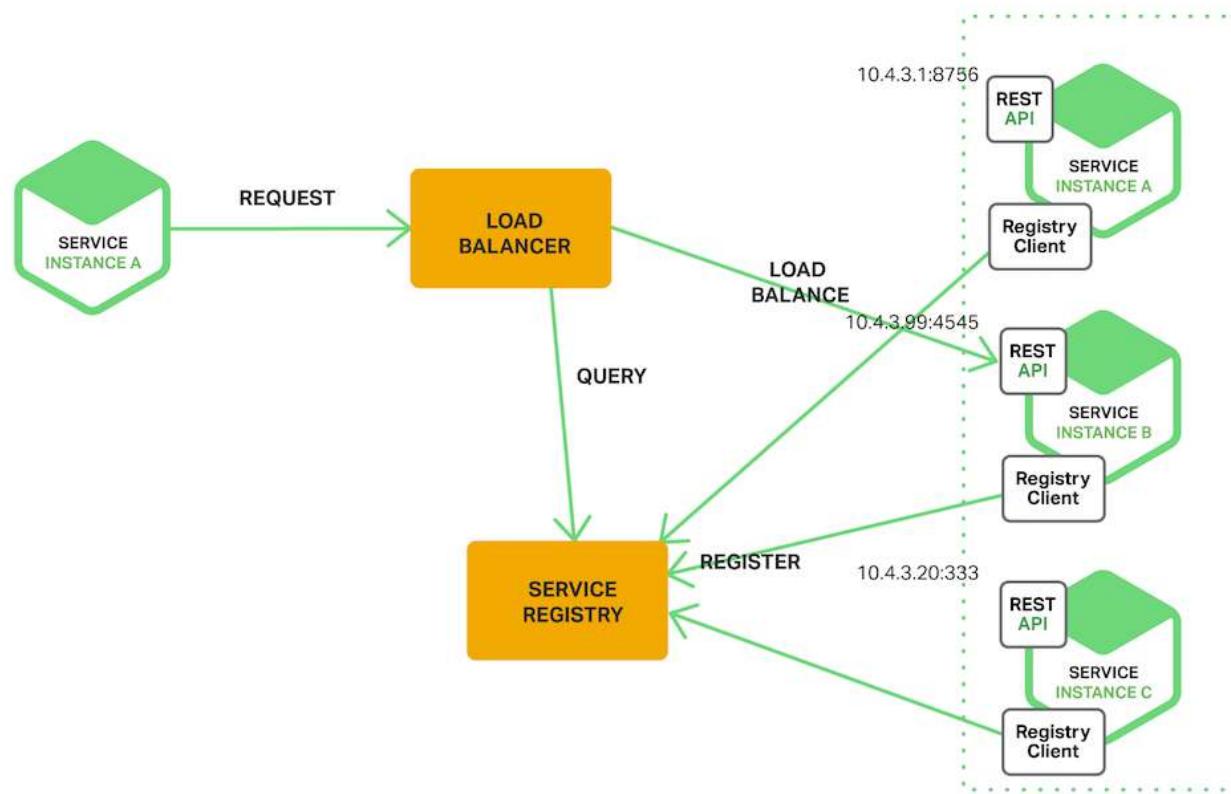
Client-side service discovery



When using client-side discovery, the client is responsible for determining the network locations of available service instances and load balancing requests across them. The client queries a service registry, which is a database of available service instances. The client then uses a load-balancing algorithm to select one of the available service instances and makes a request.



Server side service discovery

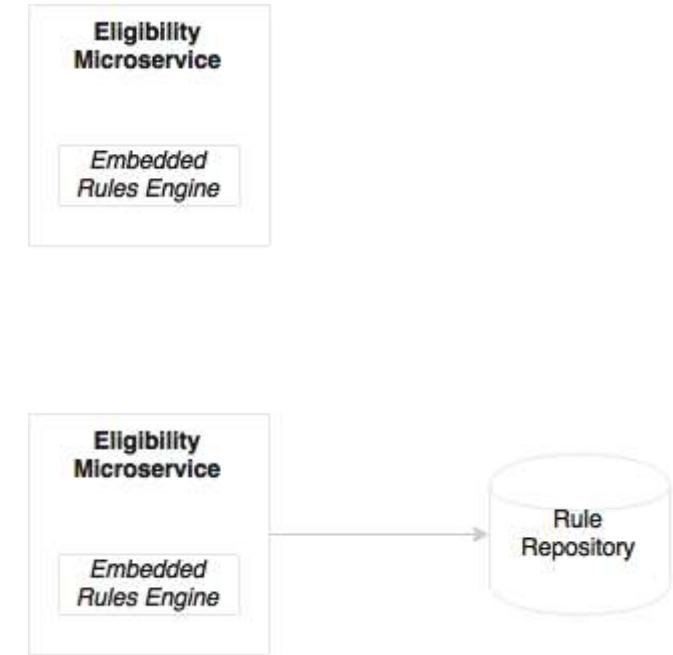


The client makes a request to a service via a load balancer. The load balancer queries the service registry and routes each request to an available service instance. One benefit is that details of discovery are abstracted away from the client. This eliminates the need to implement discovery logic for each programming language and framework used by service clients. Unless the load balancer is provided by the deployment environment, it is yet another highly available system component that we need to set up and manage.



Rule Engines and Workflows

- If the rules are simple enough, few in number, only used within the boundaries of a service – embedded **rule engine**.
- If the rules are complex, or if we are reusing rules from other service domains, then a central authoring **rule repository**.
- **BPM or dashboard** is used for composing multiple microservices in situations where end-to-end, cross-functional business processes are modeled by automated systems and human interactions.





Microservices maturity model

	Level 0 Traditional	Level 1 Basic	Level 2 Intermediate	Level 3 Advanced
Application	Monolithic	Service Oriented Integrations	Service Oriented Applications	API Centric
Database	One Size Fit All Enterprise DB	Enterprise DB + No SQLs and Light databases	Polyglot, DBaaS	Matured Data Lake / Near Realtime Analytics
Infrastructure	Physical Machines	Virtualization	Cloud	Containers
Monitoring	Infrastrucure	App & Infra Monitoring	APMs	APM & Central Log Management
Process	Waterfall	Agile and CI	CI & CD	DevOps



SUMMARY



Characteristics of Microservices

Small and focussed: The “micro” in microservices is about scope, not size and is focussed towards a specific problem.

Bounded Context:
Microservices are independent of the underlying architecture of other microservices.



Language neutral: Different microservices application can be written in a different programming language and communication with microservices is done through REST API (an HTTP-based resource API).

Loosely coupled: This essential characteristics enables organizations to deploy rapidly and frequently

Multiple code base:
Each microservice has its own code base



Advantages of Microservices - 1

▪ Technology Heterogeneity

- composed of multiple, collaborating services, that use different technologies

▪ Modular Architecture

- A modular architecture usually significantly increases the shelf life, reduces dependency on multiple standards, saves time and thus fast tracks development cycle.

▪ Simplicity.

- Each microservice performs only one distinct and well-defined function, so there is less code to take care of, less cohesion and dependency within the code, and a lower probability of bugs.

▪ Scalability.

- Selectively scale out only components that are expected to be highly loaded



Advantages of Microservices – 2

▪ Continuous delivery.

- Smaller code base and hence enables itself to a culture of continuous delivery and DevOps.

▪ More freedom and fewer dependencies.

- Microservices are standalone and independent.
- Team focuses on enhancing functionality ; worry less about interfaces and backward compatible.

▪ Fault isolation.

- A fault in one microservice will not impact the rest of microservices.

▪ Resilience.

- With microservices we can build systems that handle the total failure of services and degrade functionality accordingly.



Advantages of Microservices – 3.

▪ Data segregation and decentralization.

- Microservices provide us an opportunity to segregate data.
- Each microservice usually owns its data and does not share its data directly with other microservices.

▪ Choices.

- Microservices-based applications offer the opportunity to use the best tool for each specific job.
- Long-term commitment to technology stacks is no longer necessary.

▪ Organizational Alignment

- Microservices minimizes the number of people working on any one codebase to hit the sweet spot of team size and productivity.

▪ Composability and Optimization

- Microservices opens up seams for outside parties and owing to small size makes service replacement easier.



Disadvantages of Microservices

- **Troubleshooting complexity.**

- Inter-microservices communication are potential points of failure.

- **Increased latency.**

- Intra-process communication is much faster than the inter-process communication used by microservices.

- **Operational complexity.**

- With several hundreds to thousands of microservices in a real-world application, operations teams have to deal with complex infrastructure, deployment, monitoring, availability, backups, and management.

- **Version control.**

- Because a microservices-based application may have thousands of microservices, the versioning and management becomes little complex. It requires better version control and management systems.



REFERENCE



References

■ Books

- ✓ Cloud Computing Service and Deployment Models by Al Bento; A. Aggarwal Published by IGI Global, 2012
- ✓ Cloud Native Java by Josh Long; Kenny Bastani Published by O'Reilly Media, Inc., 2017
- ✓ Cloud-Native Applications in Java by Shyam Sundar; Ajay Mahajan; Munish Kumar Gupta, Published by Packt Publishing, 2018
- ✓ Microservices and Containers, First edition, by Parminder Singh Kocher , Published by Addison-Wesley Professional, 2018

■ Standards

- ✓ <https://docs.docker.com/>
- ✓ <https://kubernetes.io/>
- ✓ <https://www.cncf.io/>

■ Web Sites, White papers and Vendor Sites

- ✓ Blogs And Web Articles



Cloud Native Solution Design

SERVERLESS ARCHITECTURE

Kubecti

Suria R Asai

suria@nus.edu.sg

Institute of Systems Science
National University of Singapore

© 2009-23 NUS. The contents contained in this document may not be reproduced in any form or by any means, without the written permission of ISS, NUS, other than for the purpose for which it has been supplied.

Total 36 Slides



Agenda

- Server-less Architecture
 - Functions As a Service
 - Cloud Vendors
- Service Management
 - Registry Alternatives
 - API Gateway
 - Connectors
- Summary



Learning Objectives

- On completion of this module, the participant will
 - Understand Key characteristics of serverless architectures and their benefits
 - Select appropriate use cases for serverless architectures
 - Building a simple function as a service with appropriate design considerations when transitioning from server to serverless
 - Discuss an overview of '*Service Management Strategies*'



SERVERLESS ARCHITECTURE

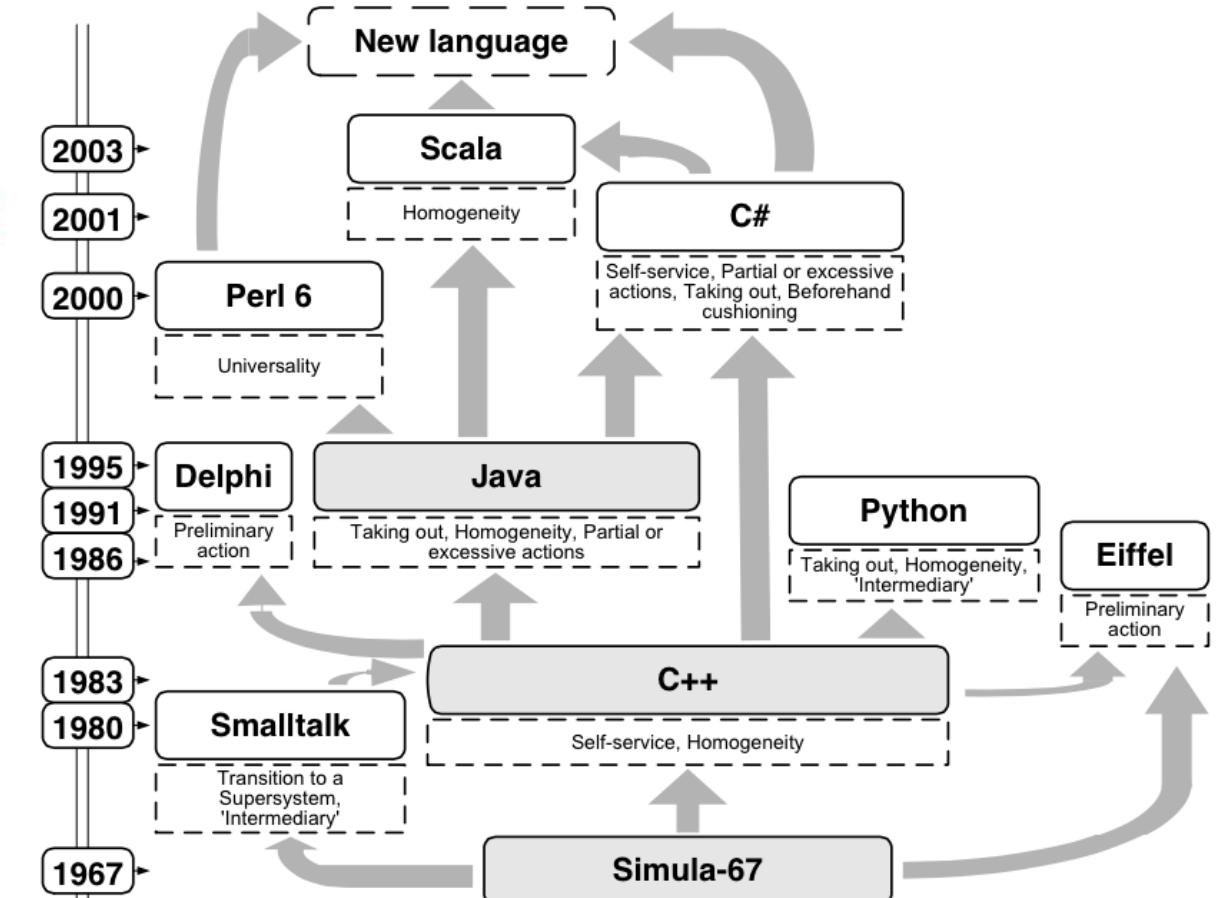
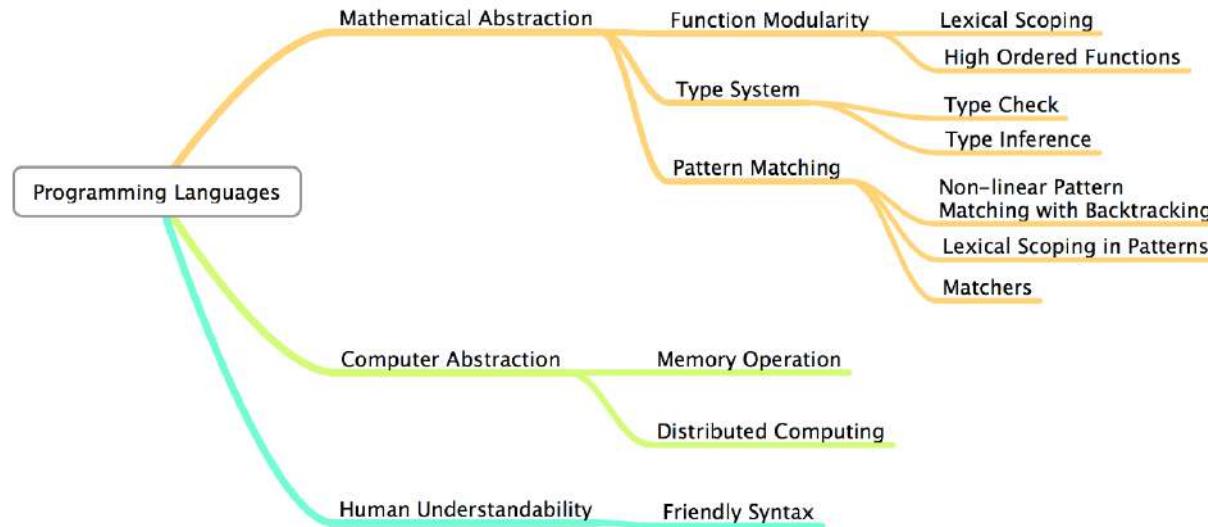
LAMBDA FUNCTIONS

API GATEWAYS

CONNECTORS



The Evolution of Programming Languages



<https://www.egison.org/blog/evolution.html>

<https://www.researchgate.net/publication/285018102 THE EVOLUTION OF THE OBJECT-ORIENTED PROGRAMMING LANGUAGES>



What is serverless computing?

An architectural approach to software solutions that relies on small independent functions running on transient servers in an elastic runtime environment.

- Serverless computing platforms share the following features:
 - No operating systems to configure or manage
 - Pay-per-invocation billing model
 - Ability to automatically scale with usage
 - Built-in availability and fault tolerance
- A well-planned serverless architecture can provide opportunities to reduce, organize, and manage complexity.
- A serverless architecture can make future changes easier, which is an important factor for any long-term application.



A Gentle Comparison

Monolithic Architecture

Products / Orders / Invoice / Payment as Single API

Microservices Architecture

Products API

Orders API

Invoice API

Payment API

Function as a Service

Products List Fn

Cancel Order Fn

Orders List Fn

Email Invoice Fn

Process Payment Fn

Create Order Fn

Process Order Fn

Cancel Payment Fn

Process Invoice Fn

Generate Invoice Fn



Four Features of a Serverless Platform

■ Event-Driven

- Serverless events are useful to broadcast information to one or more other components, with serverless events can publish events to a scalable queue.
- An example workload is web page clickstream ingestion.

■ Streaming Data

- A stream is a sequence of events, messages, or data that can be processed after they have occurred, but in the same sequence as their occurrence, which can be repeated and distributed to multiple consumers or subscribers.

■ Auto-Scaling

- Serverless architecture is composed of individual components have a short run life and can be implemented to work in a stateless way; thus scales seamlessly.

■ Fault Tolerance

- Serverless architecture implements fault-tolerant features like circuit breakers, time-outs, and other patterns.



Principles of Serverless Architecture

- Use a compute service to execute code on demand
 - Custom code is written and executed as isolated, independent, granular functions that are run in a stateless compute service such as AWS Lambda.
- Write single-purpose stateless functions.
 - A function that does just one thing is more testable and robust and leads to fewer bugs and unexpected side effects.
- Design push-based, event-driven pipelines.
 - The most flexible and powerful serverless designs are event-driven.
- Create thicker, more powerful front ends.
 - Fewer hops between online resources and reduced latency will result in a better perception of performance and usability of the application.
- Embrace third-party services.



When to use serverless architecture

- Serverless architecture allows developers to focus on software design and code rather than infrastructure.
- Reduce some of the complexity of the system by minimizing the number of layers and amount of code you need.
- Scalability and high availability are easier to achieve, and the pricing is often fairer.
- The statelessness and scalability of compute can be used to solve problems that benefit from parallel processing.
- Serverless architecture provides an opportunity to reduce some of the complexity and code in comparison.



Use cases for serverless architectures

▪ Application Back End

- appropriate for building scalable back ends for all kinds of web, mobile, and desktop applications.
- Authentication and authorization
- Communications gateway
- Registry (a way to assign a unique identity to each device)
- Device shadowing (persistent device state)
- A rules engine (a service to transform and route device messages to AWS services)

▪ Data Processing and Manipulation

- A common use for serverless technologies is data processing, conversion, manipulation, and transcoding.

▪ Real Time Analytics

- Ingestion of data—such as logs, system events, transactions, or user clicks—can be accomplished using services such as Amazon Kinesis Streams

▪ Legacy API Proxy

- The API Gateway is used to create a RESTful interface, and Lambda functions are used to transpose request/response and marshal data to formats that legacy services understand.

▪ Scheduled Services

▪ Bots and Skills



Layers of increasing abstraction

#	Layer	Examples	
1	Serverless Applications	Website API/RPC services Simple data pipeline	
2	Serverless Infrastructure	Compute Services Functions Events Tasks Scheduling	Data Blob Storage Key-Value Database Big Data Query Engine Big Data Transform Pipeline
3	Virtual Infrastructure	Virtualized hardware providing servers, networking, etc.	
4	Physical infrastructure	Actual servers, networking, load balancers, disks	



Compute Services

▪ Functions

- A function is a small piece of code that can handle an internal cloud event or respond to an HTTP request.

▪ Services

- Serverless services are similar to functions, as they also run on dynamically provisioned infrastructure, and can send HTTP requests to them.

▪ Events

- Serverless events are useful when you need to broadcast information to one or more other components faster.

▪ Tasks

- Serverless tasks are used to reliably send a message to a single other component.

▪ Scheduling

- Scheduling jobs that are triggered by other components for repeating compute jobs.



Google Products

<u>Compute</u>		<u>Data</u>	
Services	Cloud Run	Blob Storage	Cloud Storage
Functions	Cloud Functions	Key-Value Database	Cloud Firestore
Events	Cloud Pub/Sub	Big Data Query Engine	Cloud BigQuery
Tasks	Cloud Tasks	Big Data Transform Pipeline	Cloud Dataflow
Scheduling	Cloud Scheduler		



Amazon Products

<u>Compute</u>		<u>Data</u>	
Services	Amazon Fargate EKS	Blob Storage	Amazon Simple Storage Service
Functions	Amazon Lambda	Key-Value Database	Amazon DynamoDB
Events	Amazon SQS, Amazon MQ	Big Data Query Engine	Amazon Athena
Tasks	Amazon Simple Workflow Service (SWF)	Big Data Transform Pipeline	Amazon Kinesis
Scheduling	Amazon Batch		



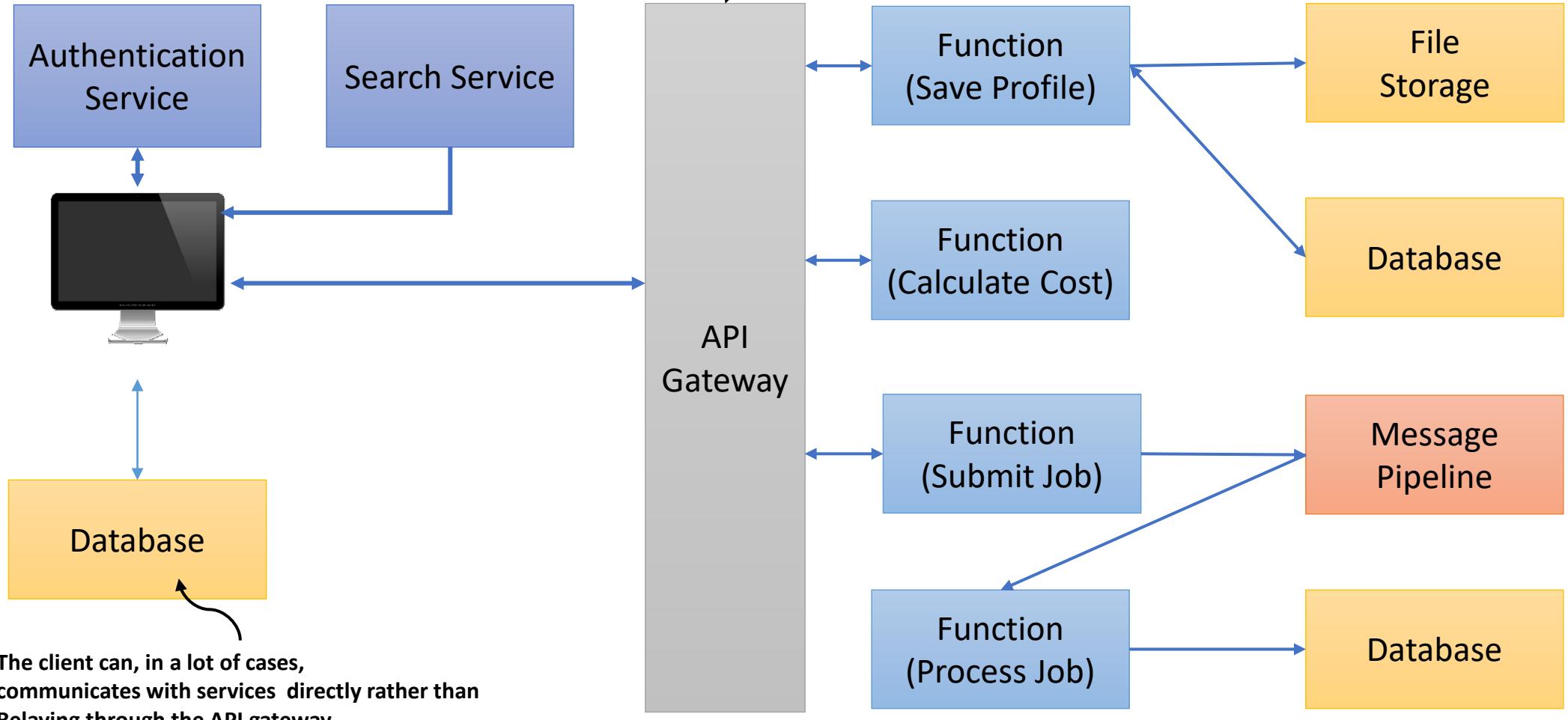
Azure Products

<u>Compute</u>		<u>Data</u>	
Services	Azure Container Instances	Blob Storage	Azure File Storage
Functions	Azure Functions	Key-Value Database	Azure Table Storage
Events	Azure Event Hub	Big Data Query Engine	Azure Cosmos
Tasks	Azure Task	Big Data Transform Pipeline	Azure Data Lake
Scheduling	Azure Batch		



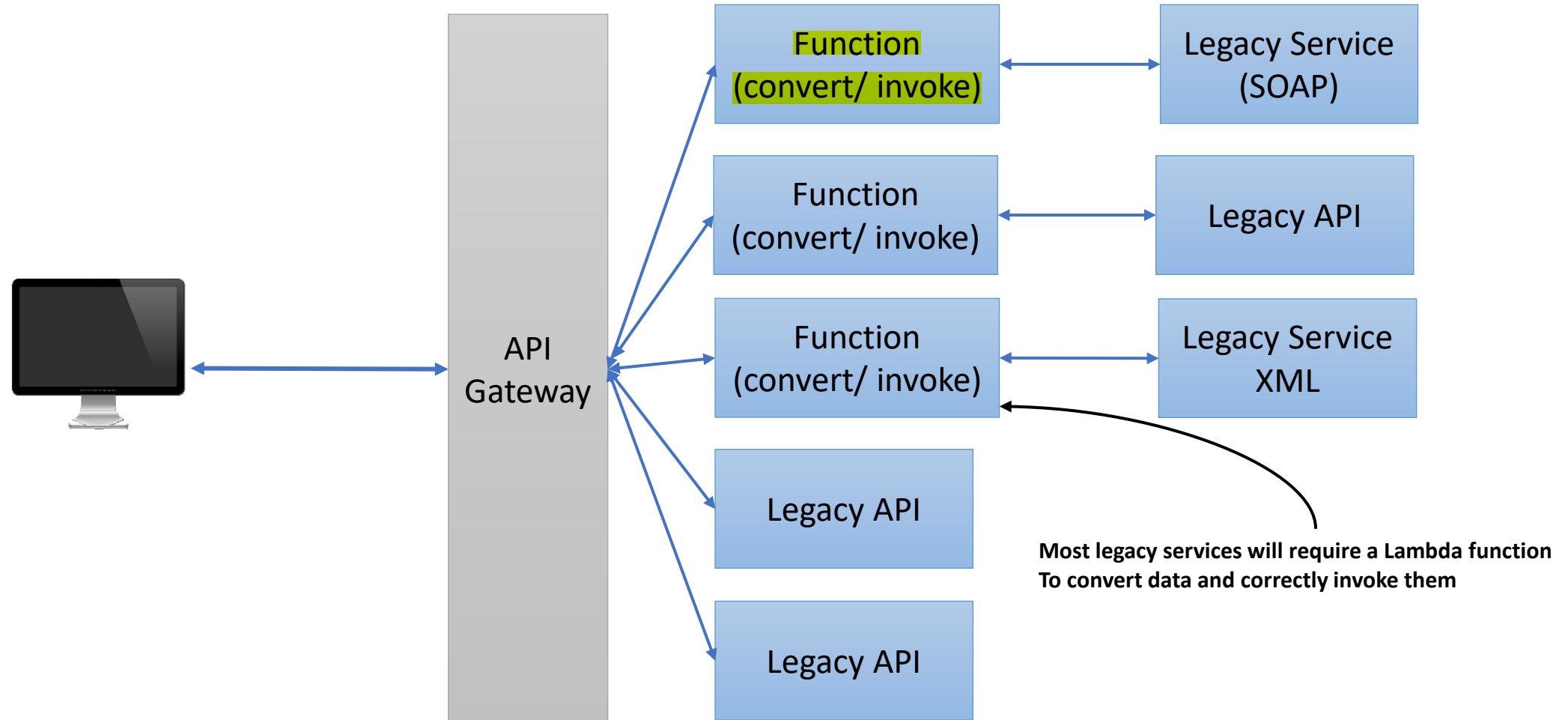
Example – Application Back End

The API Gateway creates a RESTFUL interface and hides Lambda functions and other services behind it.
Lambda functions can carry out custom tasks and Communicate with other services



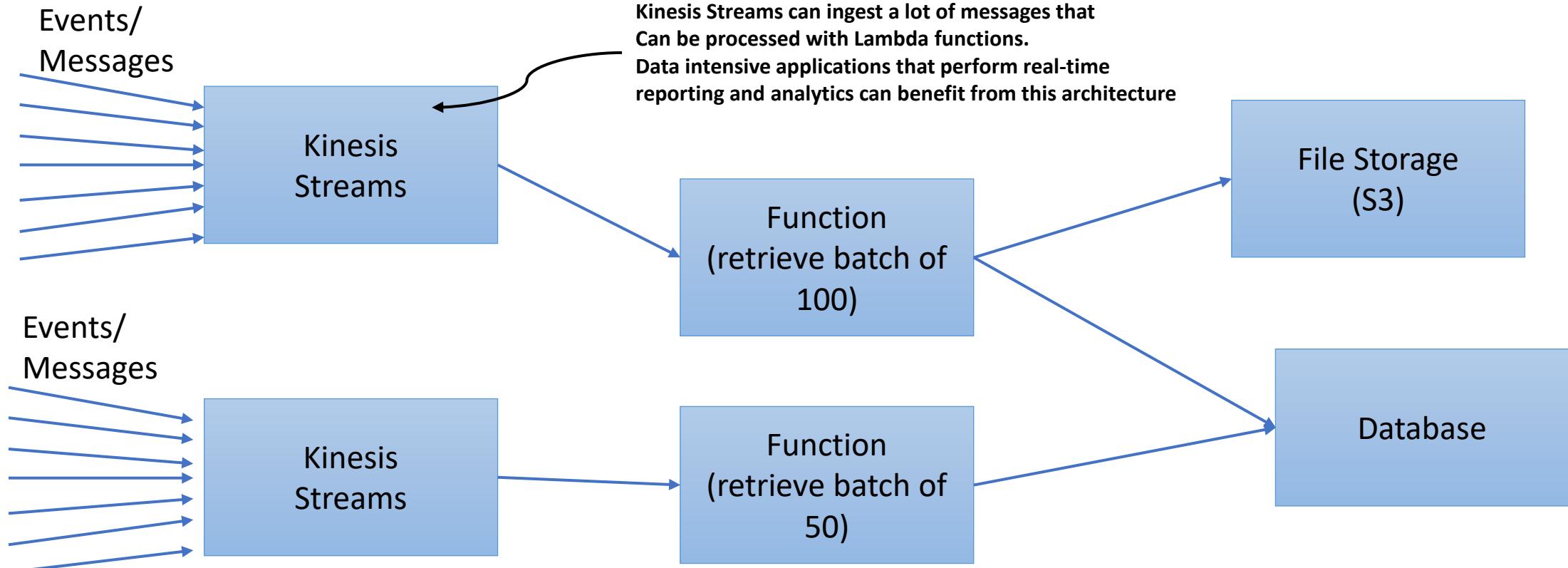


Example – API Proxy Architecture



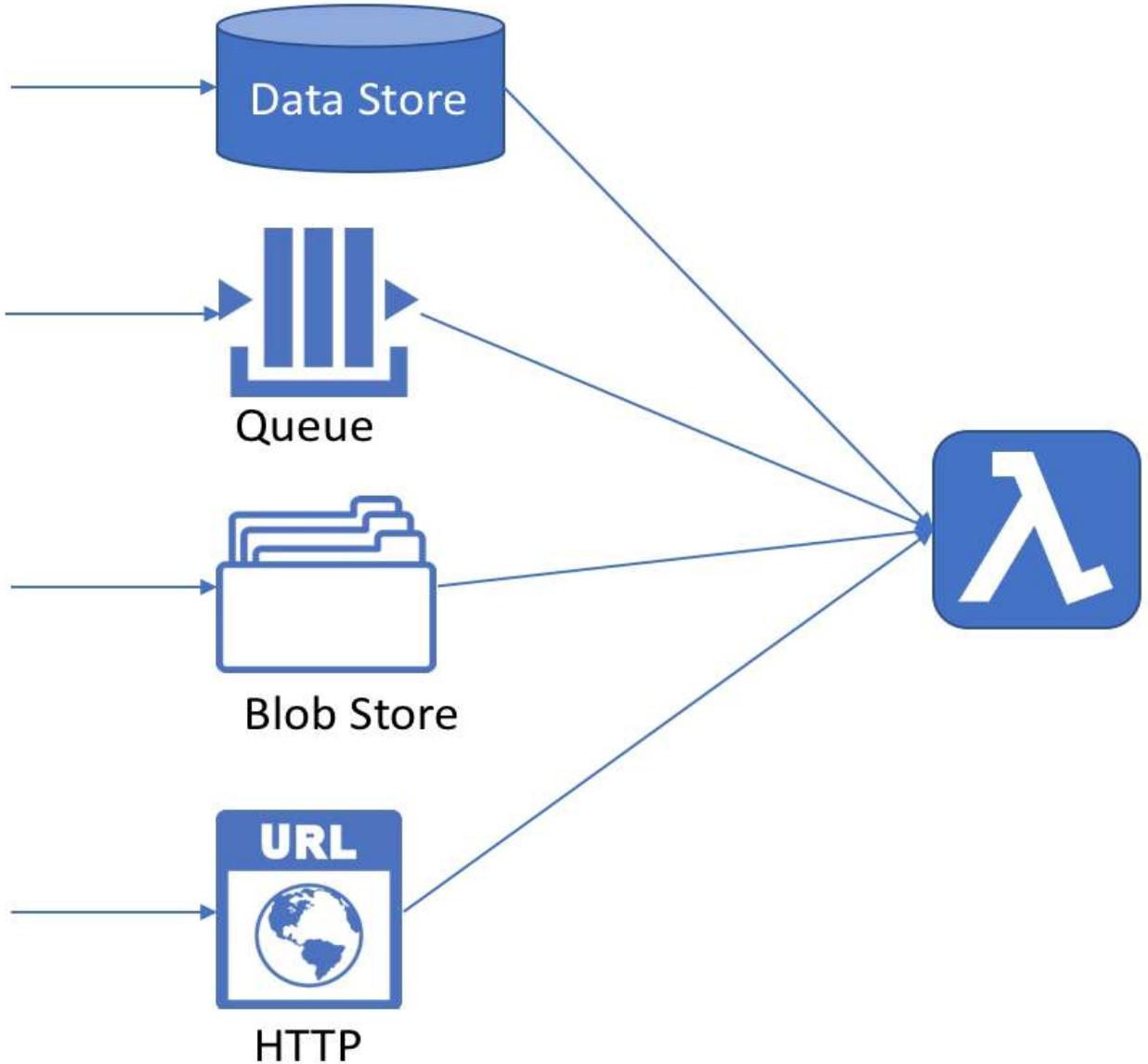


Example – Real Time Streams



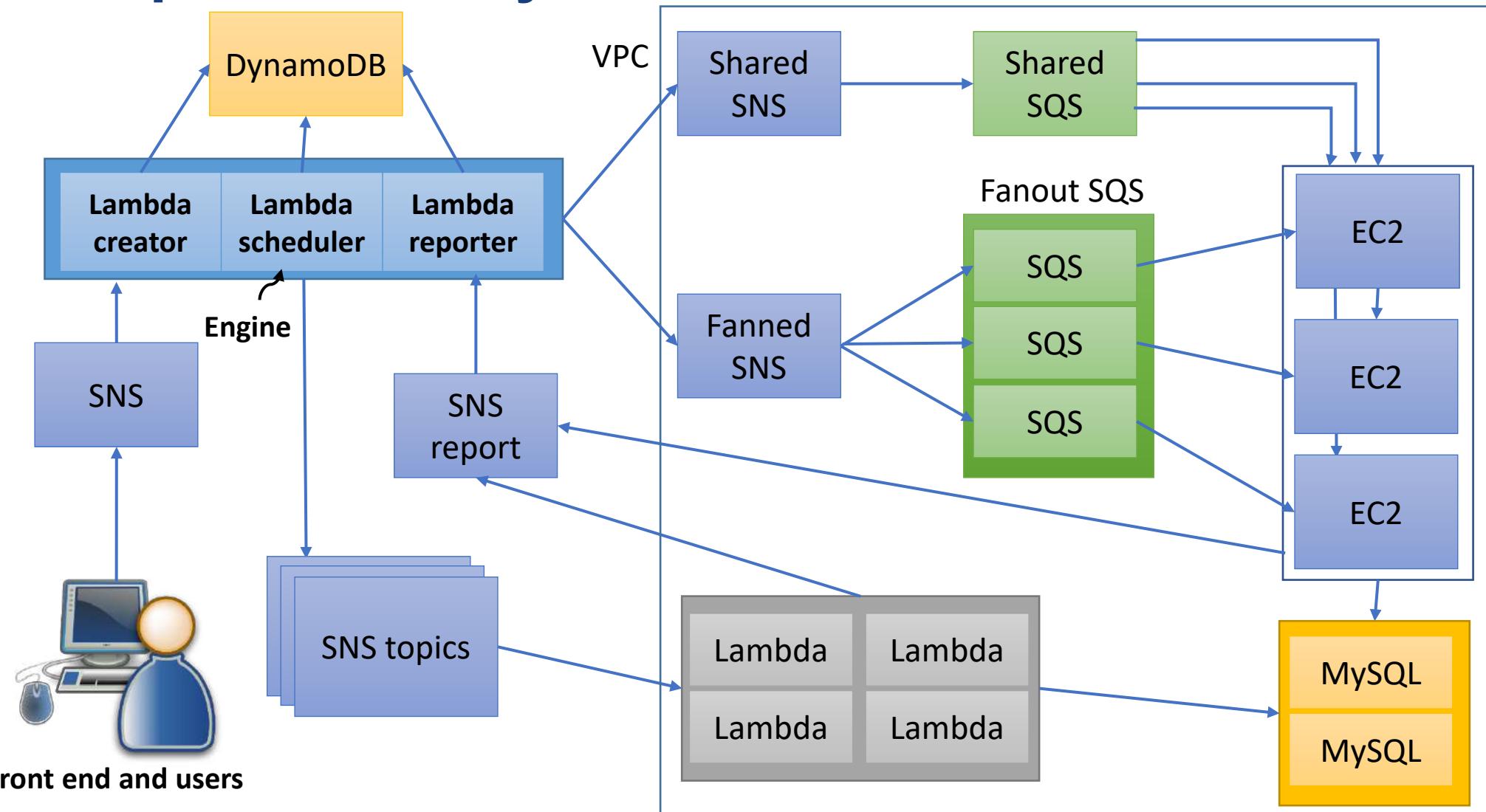


Deploying to Lambda





Example – AWS Hybrid Architecture.





SUMMARY



Concluding Remarks for Serverless Architecture

- Serverless architectures are the latest advance for developers and organizations to think about, study, and adopt.
 - We discussed what serverless architecture is, looked at its principles, and saw how it compares to traditional architectures.
 - We explored the important architectures and patterns.
 - We discussed specific use cases where serverless architectures were used to solve a problem.
- In many circumstances, serverless applications will be cheaper to run and faster to implement.
- The reduction in cost and time spent on infrastructure maintenance and the benefits of scalability are good reasons for consideration



REFERENCE



References

- AWS Lambda in Action: Event-driven serverless applications by Danilo Poccia *Published by Manning Publications, 2016*
- Serverless Architectures on AWS: With examples using AWS Lambda by Peter Sbarski *Published by Manning Publications, 2017*
- Serverless Design Patterns and Best Practices by Brian Zambrano *Published by Packt Publishing, 2018*
- International Journal of Systems and Service-Oriented Engineering (IJSSOE) Volume 6, Issue 1 by Dickson K.W. Chiu Published by IGI Global, 2016
- Managing Trade-offs in Adaptable Software Architectures by Rick Kazman; Ivan Mistrik; Bradley Schmerl; Nour Ali; John Grundy Published by Morgan Kaufmann, 2016
- Cloud Computing Service and Deployment Models by Al Bento, A. Aggarwal Publisher: IGI Global Published: October 2012
- T. Erl, SOA Design Patterns, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2009.



APPENDIX 1 – SERVERLESS IN AWS

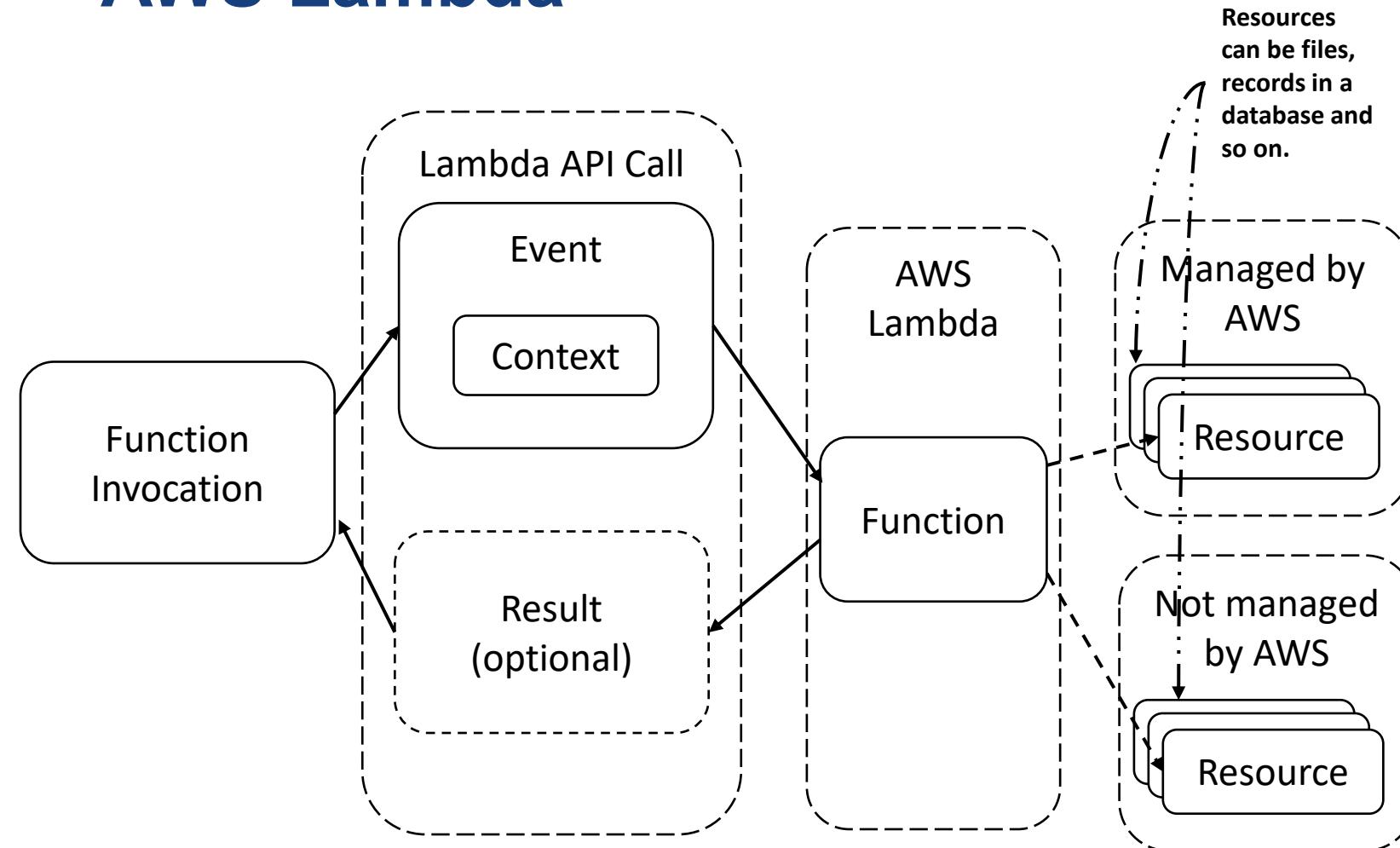


AWS Lambda

- Functions are executed in containers.
- When we create a new function with AWS Lambda, we choose a *function name*, create code, and specify the configuration of the execution environment that will be used to run the function, including the following:
 - The maximum *memory size* that can be used by the function
 - A *timeout* after which the function is terminated, even if it hasn't completed
 - A *role* that describes what the function can do, and on which resources, using AWS Identity and Access Management (IAM)
- With AWS Lambda you pay for
 - The number of invocations
 - The hundreds of milliseconds of execution time of all invocations, depending on the memory given to the functions



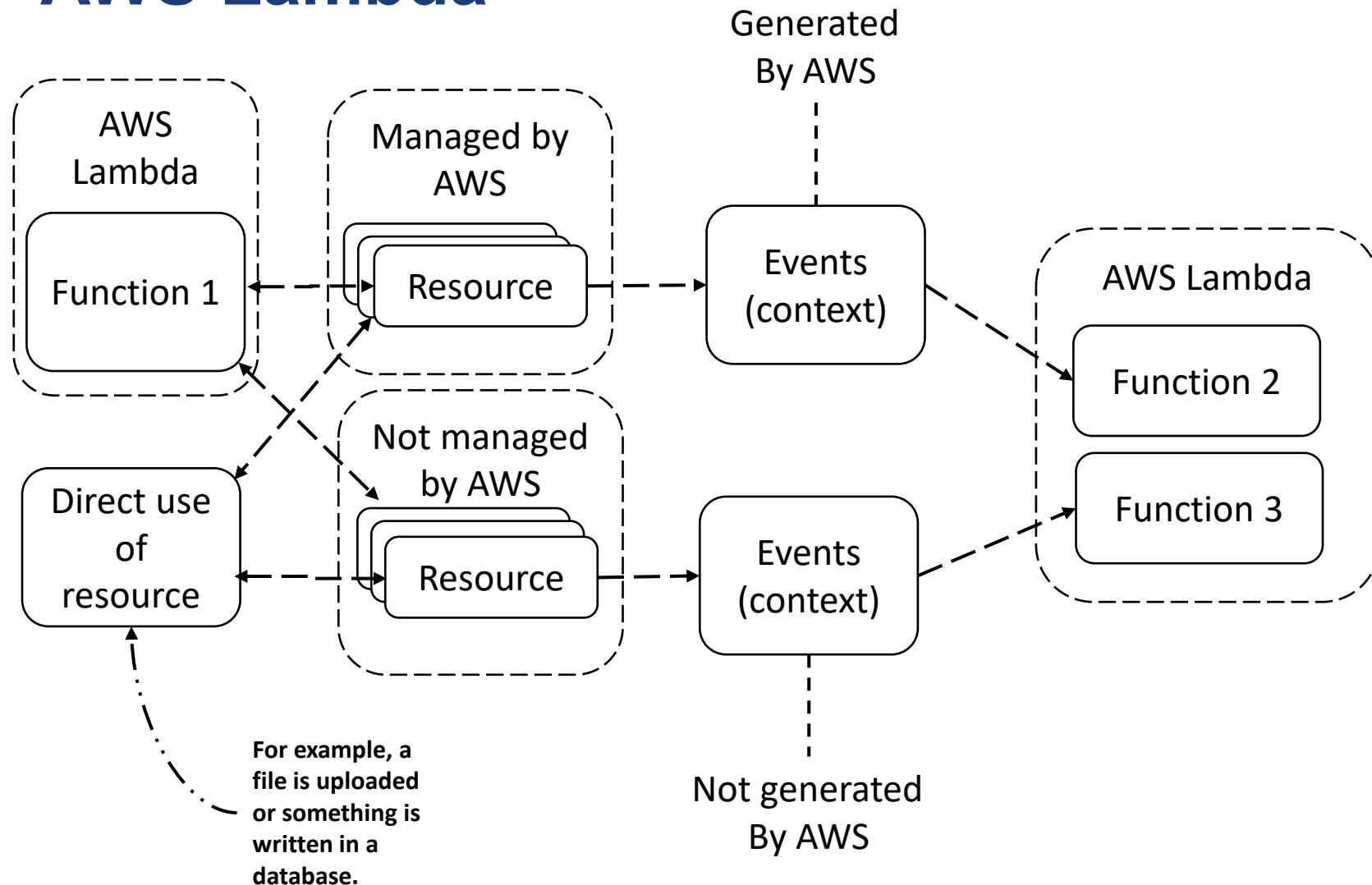
AWS Lambda



- The *event* is the way to send input parameters for your function and is expressed using JSON syntax.
- The *context* is used by the service to describe the execution environment and how the event is received and processed.



AWS Lambda



- Asynchronous calls are useful when functions are subscribed to events generated by other resources, such as Amazon S3, an object store, or Amazon DynamoDB, a fully managed NoSQL database.



AWS Lambda as a Web API

- AWS Lambda function as a web API can be created by:
 - Integrating a Lambda function with a method of the API Gateway
 - Pass parameters over HTTP to the function in different ways
 - Format the result of the function as the response of the web API
 - Quickly test the integration of the API Gateway from the web console
 - Use the context provided by the API Gateway within a Lambda function



AWS Lambda – Define Function

Lambda > New function

- [Select blueprint](#)
- [Configure triggers](#)
- Configure function**
- [Review](#)

Configure function

A Lambda function consists of the custom code you want to execute. [Learn more about Lambda functions.](#)

Name*	greetingsOnDemand	The name of the function
Description	Returns greetings when you ask for them	A description for the function
Runtime*	Node.js 4.3	Runtime to use to execute the function

Lambda function code

Provide the code for your function. Use the editor if your code does not require custom libraries (other than the aws-sdk). If you need custom libraries, you can upload your code and libraries as a ZIP file. [Learn more about deploying Lambda functions.](#)

Code entry type: [Edit code inline](#)

```

1- exports.handler = (event, context, callback) => {
2-   // TODO implement
3-   callback(null, 'Hello from Lambda');
4- };

```

The code for the function goes here.

Instead of editing the code inline, you can choose the option to upload a zip file with custom libraries.

Reference Book: AWS Lambda in Action: Event-driven serverless applications



AWS Lambda – Code a Function

```

import json

print('Loading function')

def lambda_handler(event, context):
    print('Received event: ' +
          json.dumps(event, indent=2))
    if 'name' in event:
        name = event['name']
    else:
        name = 'World'
    greetings = 'Hello ' + name + '!'
    print(greetings)
    return greetings

```

Initialization; in Python you need the “json” module to dump the event.

Function declaration; the input event is a Python built-in type, usually a dict.

Logging to Amazon CloudWatch Logs

End function and return value

Reference Book: AWS Lambda in Action: Event-driven serverless applications



AWS Lambda – Security Settings

Which function in the code should be called by AWS Lambda

Execution role and relative permissions

Maximum memory to use

Timeout after which the function is automatically terminated

Optionally you can access resources within an Amazon VPC, a logically isolated section of the AWS Cloud.

Lambda function handler and role

Handler* IndexHandler

Role* Choose an existing role

Existing role

Advanced settings

These settings allow you to control the code execution performance and costs for your Lambda function. Changing your resource settings (by selecting memory) or changing the timeout may impact your function cost. [Learn more](#) about how Lambda pricing works.

Memory [MB]* 128

Timeout* 0 min 3 sec

All AWS Lambda functions run securely inside a default system-managed VPC. However, you can optionally configure Lambda to access resources, such as databases, within your custom VPC. [Learn more](#) about accessing VPCs within Lambda. Please ensure your role has appropriate permissions to configure VPC.

VPC No VPC

* These fields are required.

Cancel Previous Next

Reference Book: AWS Lambda in Action: Event-driven serverless applications



AWS Lambda – API Gateway

`https://some.domain/stage/resource1/resource2/.../resourceN`

*Unique domain
that you can
customize*

Resources as part of the URL

*API stage; for example
prod, test, dev, or v0, v1, ...*

GET	→ function1
POST	→ function2
PUT	→ function3
DELETE	→ function1
HEAD	→ function2
PATCH	→ function3
OPTIONS	→ function1

HTTP verb
used when
accessing the URL Function
to execute

You can optionally import an API,
providing a Swagger API definition,
or create an example API.

Amazon API Gateway APIs > Create

Show all hints ?

Create new API

In Amazon API Gateway, an API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

New API Import from Swagger Example API

Name and description

Choose a friendly name and description for your API.

API name* My Utilities

Description A set of small utilities

* Required

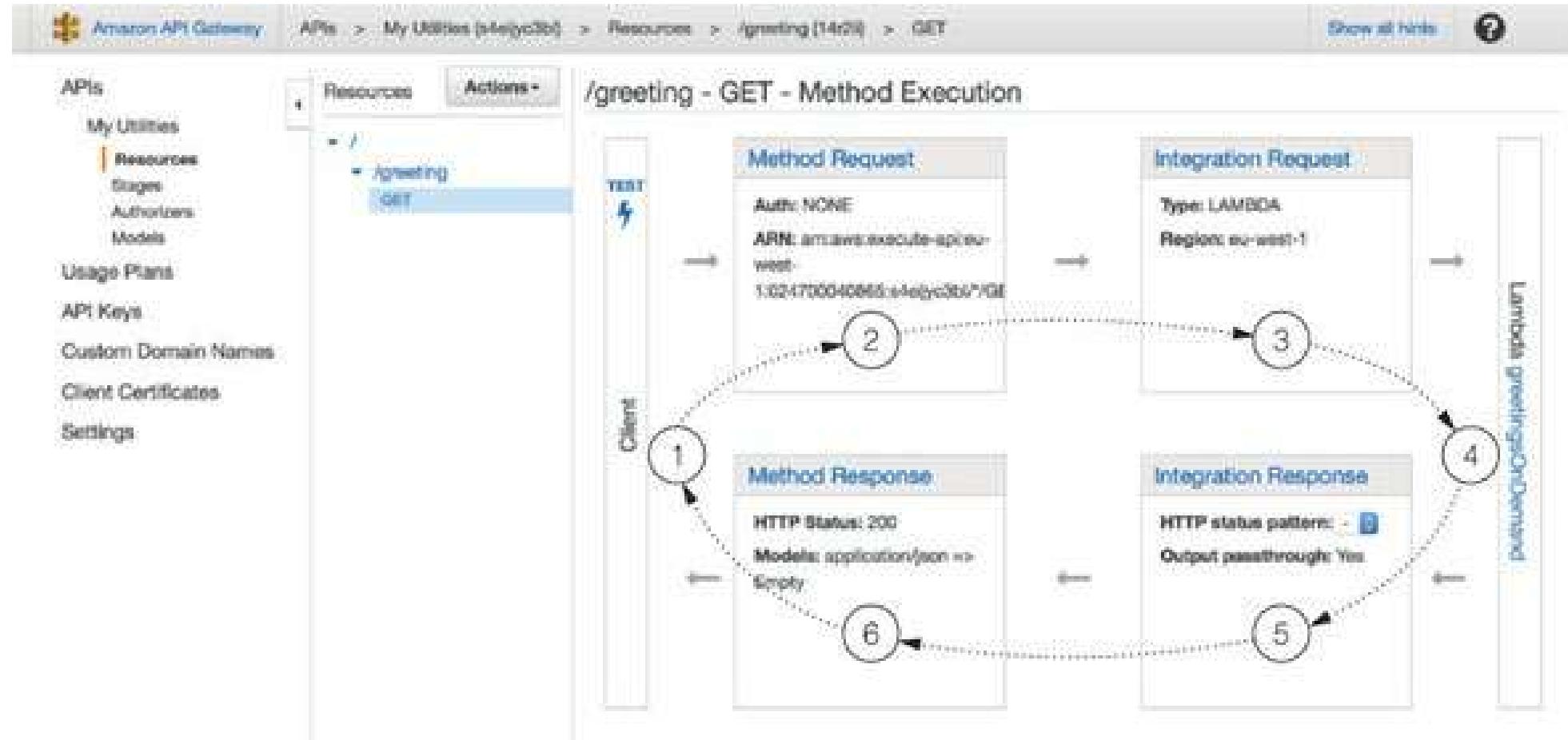
Create API

Creating a new API is straightforward,
you need a name and a description.

Reference Book: AWS Lambda in Action: Event-driven serverless applications



AWS Lambda – Flow Execution



Reference Book: AWS Lambda in Action: Event-driven serverless applications



AWS Lambda –Testing.

Amazon API Gateway APIs > My Utilities (1 resource) > Resources > /greeting (14KB) > GET Show all items ?

APIs

- My Utilities
- Resources**
- Shapes
- Authorizers
- Models

Usage Plans:

- API Keys:
- Custom Domain Names
- Client Certificates
- Settings

Resources

/ /greeting /test

Method Execution /greeting - GET - Method Test

Make a test call to your method with the provided input.

Path
No path parameters exist for this resource. You can define path parameters by using the syntax {myPathParam} in a resource path.

Query Strings

name: John

Headers
No header parameters exist for this method. You can add them via Method Request.

Stage Variables
No stage variables exist for this method.

Request Body
Request Body is not supported for GET methods.

Request

Request: /greeting?name=John
Status: 200
Latency: 1098 ms
Response Body:
"Hello John!"

Response Headers

{"Content-Type": "application/json"}

Logs

```
Execution log for request test-request
Wed Aug 31 14:31:29 UTC 2016 : Starting execution for request: test-Invoke-request
Wed Aug 31 14:31:29 UTC 2016 : HTTP Method: GET, Resource Path: /greeting
Wed Aug 31 14:31:29 UTC 2016 : Method request path: /
Wed Aug 31 14:31:29 UTC 2016 : Method request query string: {name=John}
Wed Aug 31 14:31:29 UTC 2016 : Method request headers: {}
Wed Aug 31 14:31:29 UTC 2016 : Method request body before transformations: null
Wed Aug 31 14:31:29 UTC 2016 : Endpoint request URL: https://lambda.eu-west-1.amazonaws.com/2015-03-31/functions/greetingsOnDemand/invocations
Wed Aug 31 14:31:29 UTC 2016 : Endpoint request headers: {x-amz-lambda-integration-tag=test-request, Authorization=*****}
*****
```

Input parameter for the method, a query string in this case

Response Body returned by calling the method

Response Headers returned by calling the method

Logs to understand the interaction between the Amazon API Gateway and AWS Lambda

Reference Book: AWS Lambda in Action: Event-driven serverless applications



Cloud Native Solution Design

CLOUD PATTERNS AND PRACTICES

Suria R Asai

suria@nus.edu.sg

Institute of Systems Science

National University of Singapore

© 2009-23 NUS. The contents contained in this document may not be reproduced in any form or by any means, without the written permission of ISS, NUS, other than for the purpose for which it has been supplied.

Total Slides 68



Agenda

- Introduction to Pattern School of Thoughts
- Cloud Native and Microservices Patterns
- Data Storage Patterns
- Content Aggregation Patterns
- Tenancy Models
- Summary



Learning Objectives

- On completion of this module, the participant will
 - Understand Infrastructure as a Service (IaaS) and Platform as a Service (PaaS).
 - Learn about architecting web applications in the cloud.
 - Introduction of the 12-factors for Cloud Native Applications.
 - Design microservices via VM or Container or host.
 - Build distributed microservices based application.
 - Understand the typical cloud native application design concerns.



INTRODUCTION

CLOUD COMPUTING PATTERNS - SCHOOL OF THOUGHTS



Cloud Patterns

- **An architectural pattern** is the idea of capturing design ideas as archetypal and reusable descriptions.
 - Addresses performance limitations, high availability and minimization of a business risk.
 - Some architectural patterns have been implemented within software frameworks.
- **A design pattern** is a general, reusable solution to a commonly occurring problem within a given context in software design.
 - A template for how to solve a problem that can be used in many different situations.
 - Design patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system.

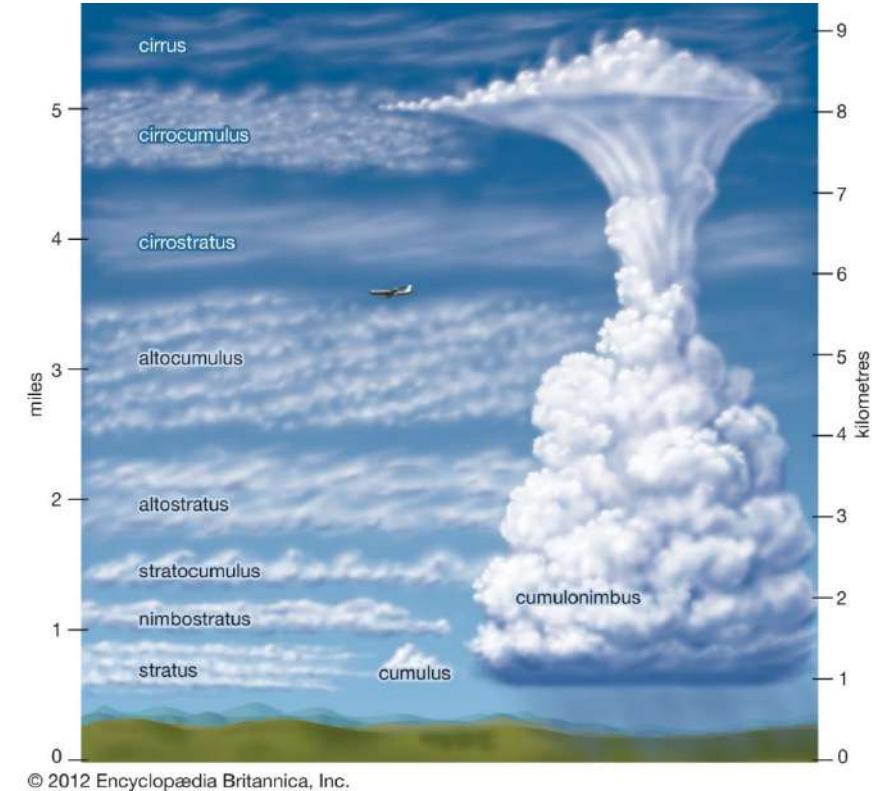
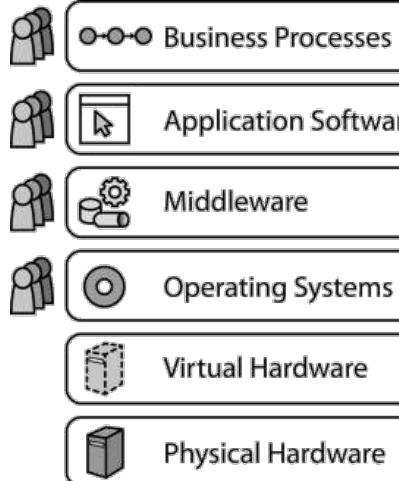
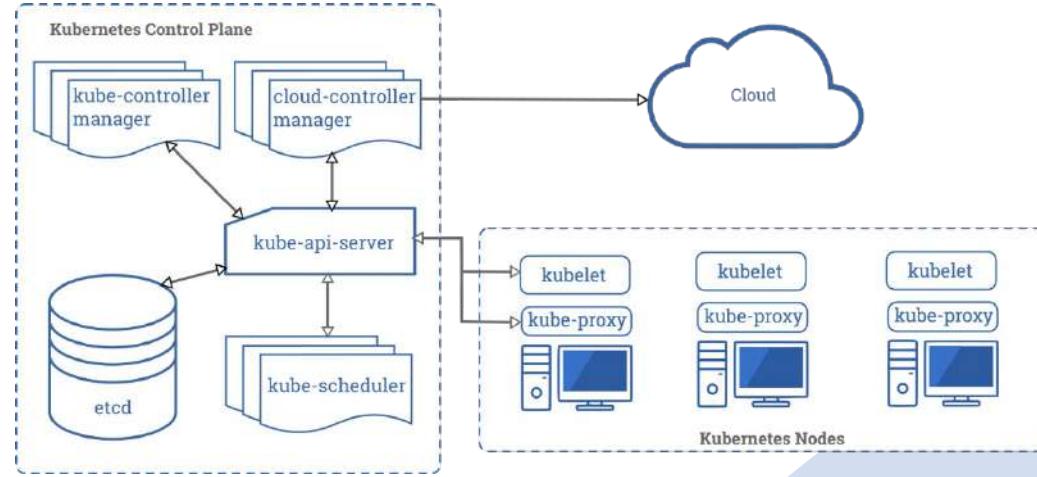


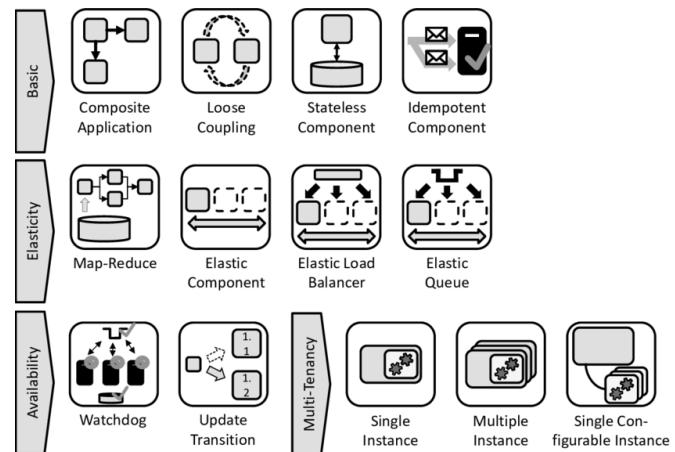
Image Source : <https://www.britannica.com/science/climate-meteorology/Snow-and-sleet>



Cloud Computing Patterns – A long way



VM



Containers



Cloud
Native

Traits of the Cloud Natives



Customer-centric



Learning culture



Agile development



Focus on products,
not projects



Automation of entire
delivery process



Heavy use of open-
source software



Example

- Ambassador
- Anti-Corruption Layer
- Backends for Frontends
- Bulkhead
- Cache-Aside
- Circuit Breaker
- CQRS
- Compensating Transaction
- Competing Consumers
- Compute Resource Consolidation
- Event Sourcing
- External Configuration Store
- Federated Identity
- Gatekeeper
- Gateway Aggregation
- Gateway Offloading
- Gateway Routing
- Health Endpoint Monitoring
- Index Table
- Leader Election
- Materialized View
- Pipes and Filters
- Priority Queue
- Publisher/Subscriber
- Queue-Based Load Leveling
- Retry
- Scheduler Agent Supervisor
- Sharding
- Sidecar
- Static Content Hosting
- Strangler
- Throttling
- Valet Key



Martin Fowler et al

▪ <https://www.martinfowler.com/articles/writingPatterns.html>

Design and Implementation

- External Config Store
- Federated Identity
- Gatekeeper
- Runtime Reconfiguration
- Valet Key
-

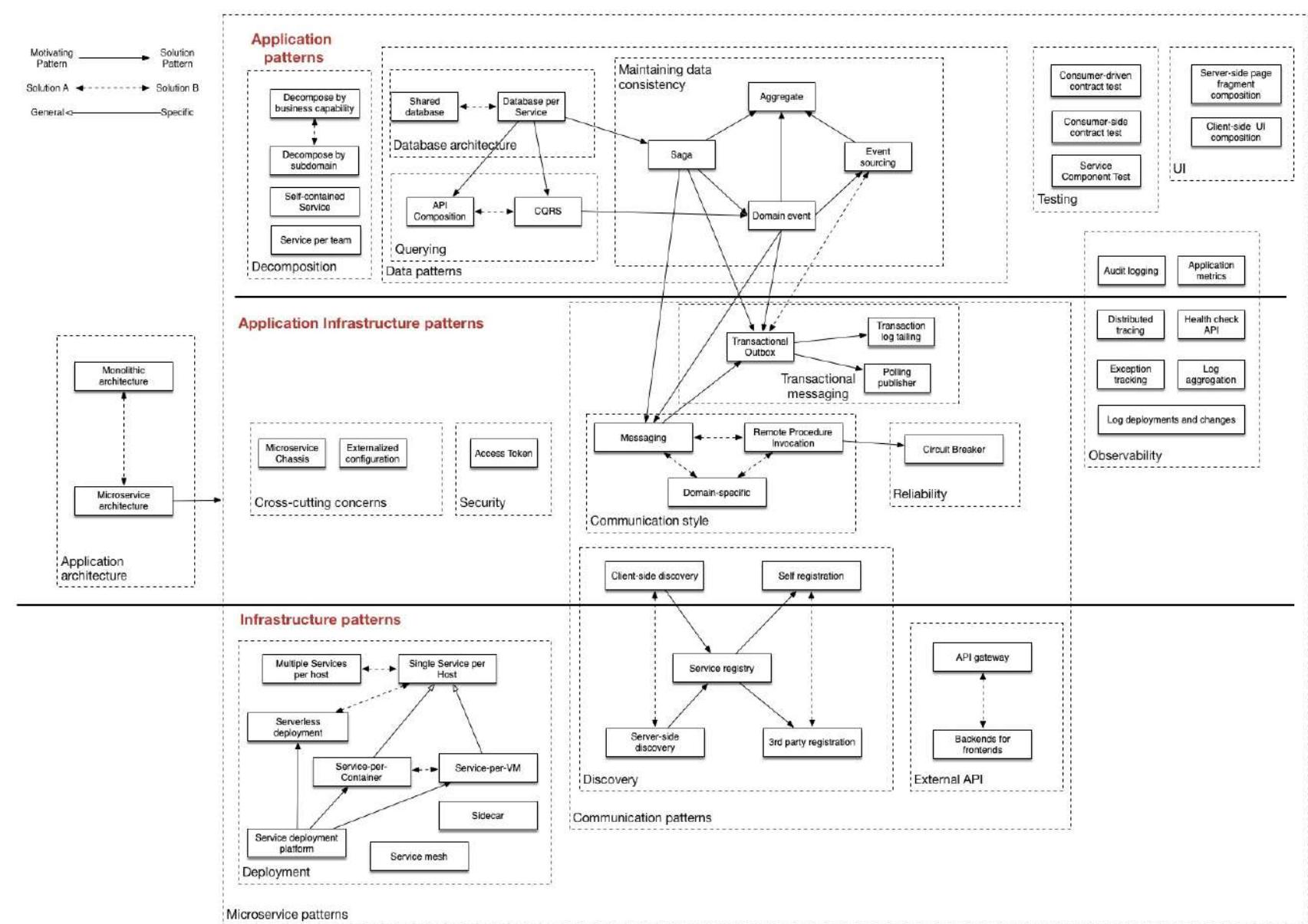
Data Management and Performance

- Automatic Scaling
- Cache-aside
- CQRS
- Event Sourcing
- Sharding
- Static Content Hosting
-

Availability and Resilience

- Circuit Breaker
- Compensating Transaction
- Health Endpoint Monitoring
- Queue-based Load Leveling
- Retry
- Throttling
-

Microservices.io (Chris Richardson et al)



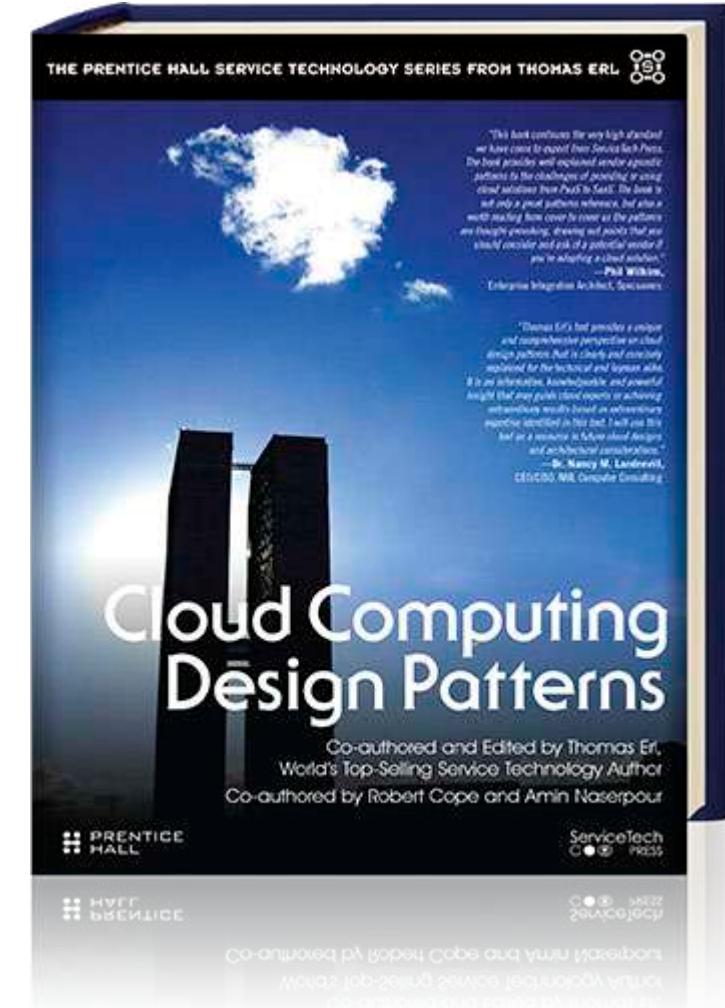
Copyright © 2019, Chris Richardson Consulting, Inc. All rights reserved.

Learn-Build-Assess Microservices <http://adopt.microservices.io>



Architectura Catalogue (Thomas Erl et al)

- Sharing, Scaling and Elasticity Patterns
- Reliability, Resiliency and Recovery Patterns
- Data Management and Storage Device Patterns
- Virtual Server and Hypervisor Connectivity and Management Patterns
- Monitoring, Provisioning and Administration Patterns
- Cloud Service and Storage Security Patterns
- Network Security, Identity & Access Management, and Trust Assurance Patterns
- Common Compound Patterns





Vendor Blogs

- <https://aws.amazon.com/blogs/>
- <https://cloud.google.com/blog/>
- <https://azure.microsoft.com/en-us/blog/>
- <https://resource.alibabacloud.com/webinar/index.htm>
- <https://blog.digitalocean.com/>



CLOUD NATIVE & MICROSERVICES PATTERNS



How inclusive is cloud native architecture?

The Multi-Architectural-Patterns and polyglot microservices world



Kubernetes Patterns (Automatable containers at scale)

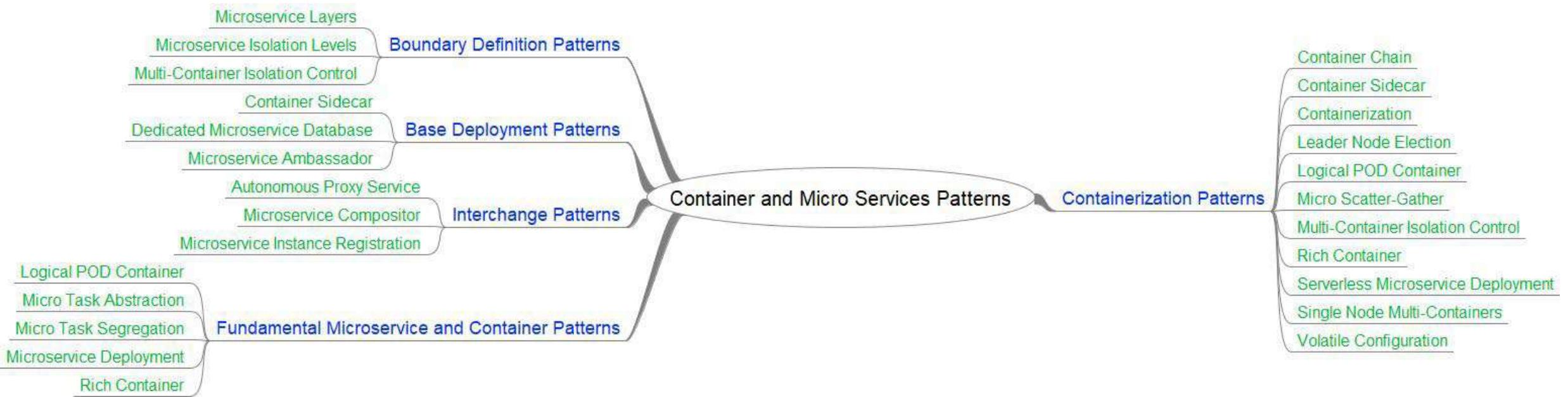
Microservices Principles (Services optimized for change)

Domain-Driven Design (Ubiquitous domain model)

Clean Code (Well-crafted code)



Container and related microservices patterns



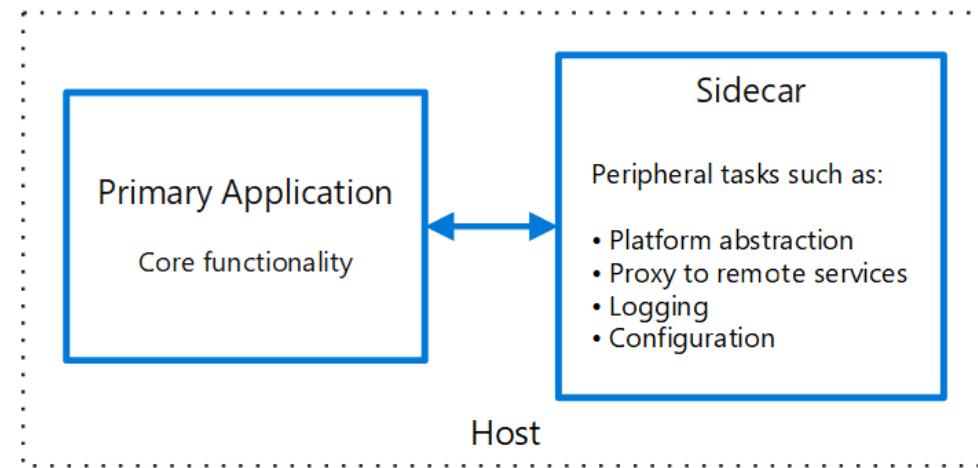
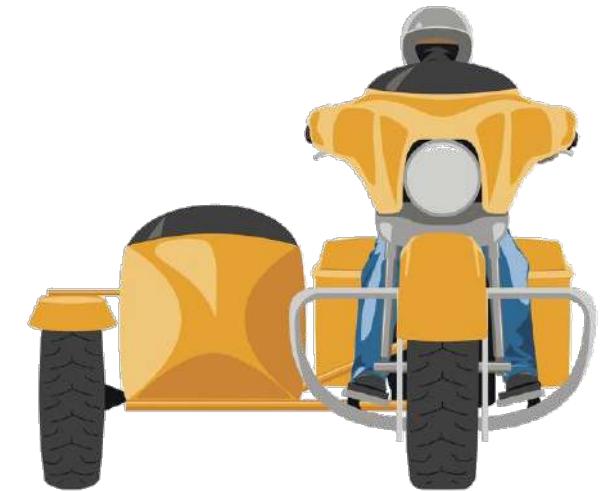
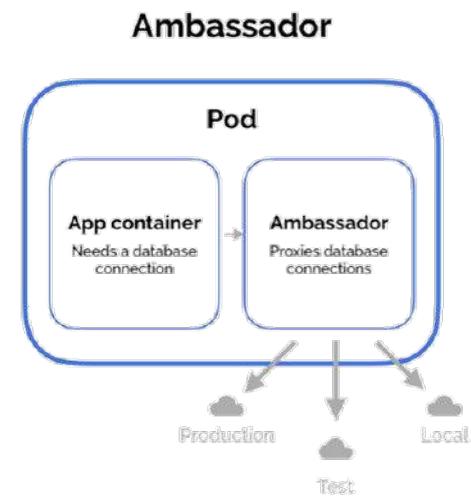
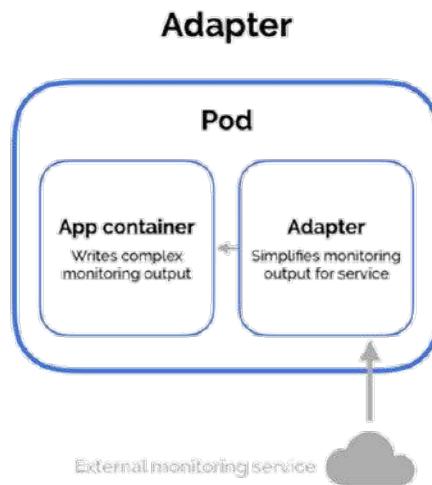
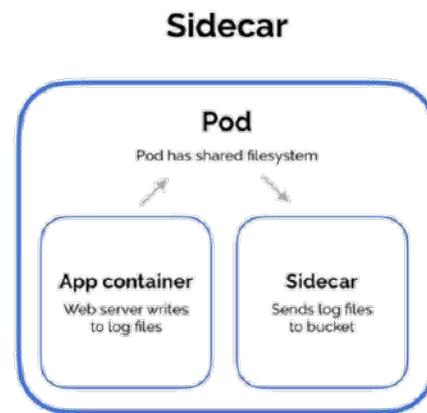
A modularized Container is

1. Linux Process
2. API Driven
3. Descriptive

4. Disposable
5. Immutable
6. Self Contained
7. Small

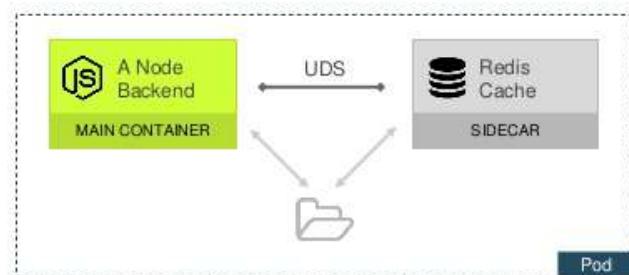


Container Side Car



Pattern: Sidecar / Sidekick

Enhance & extend the main container.
K8S: transparently. Netflix: platform features.

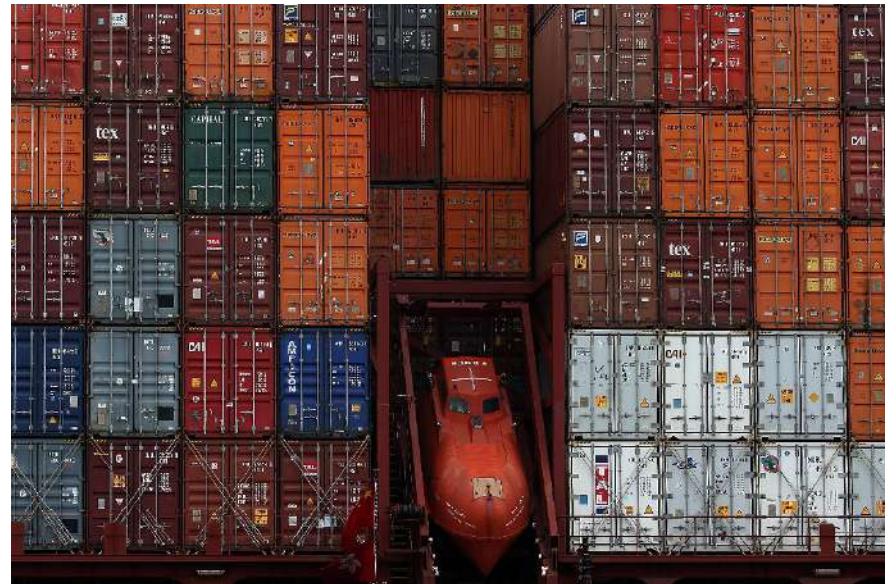
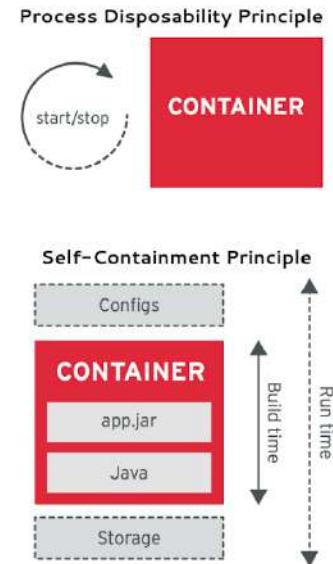
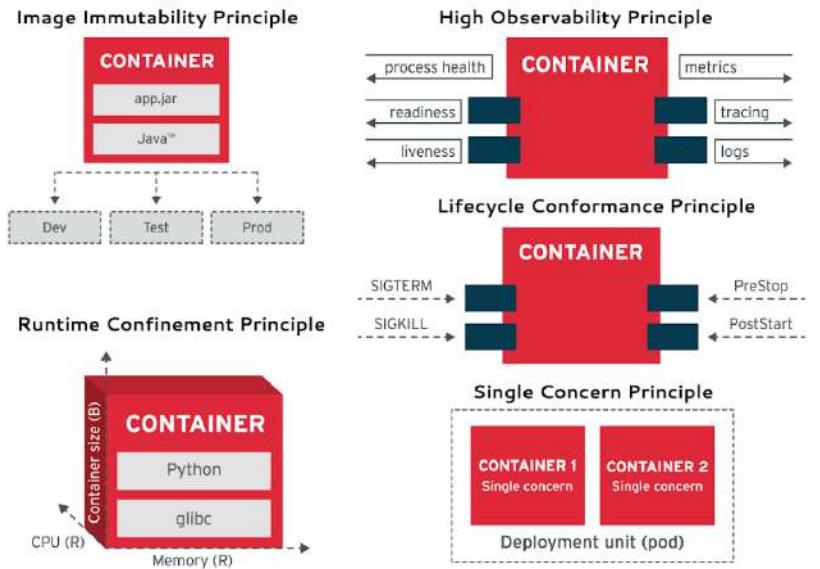
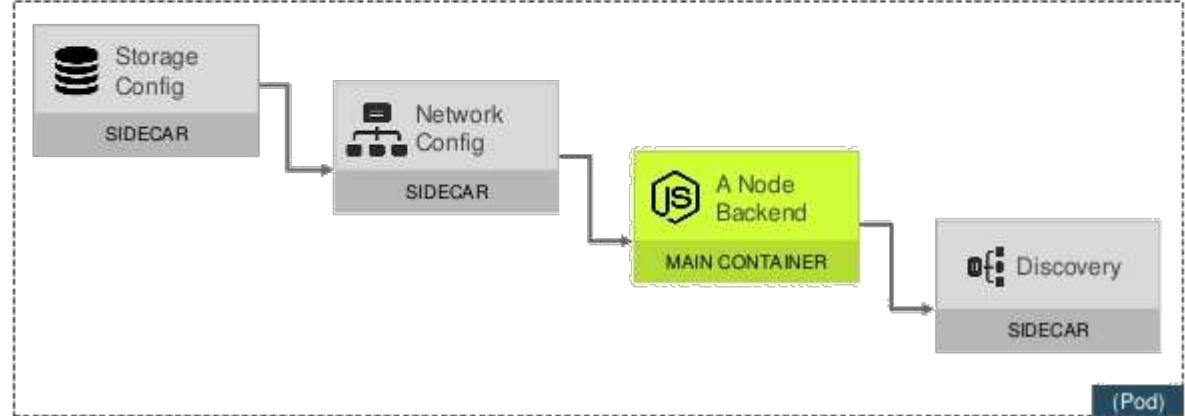


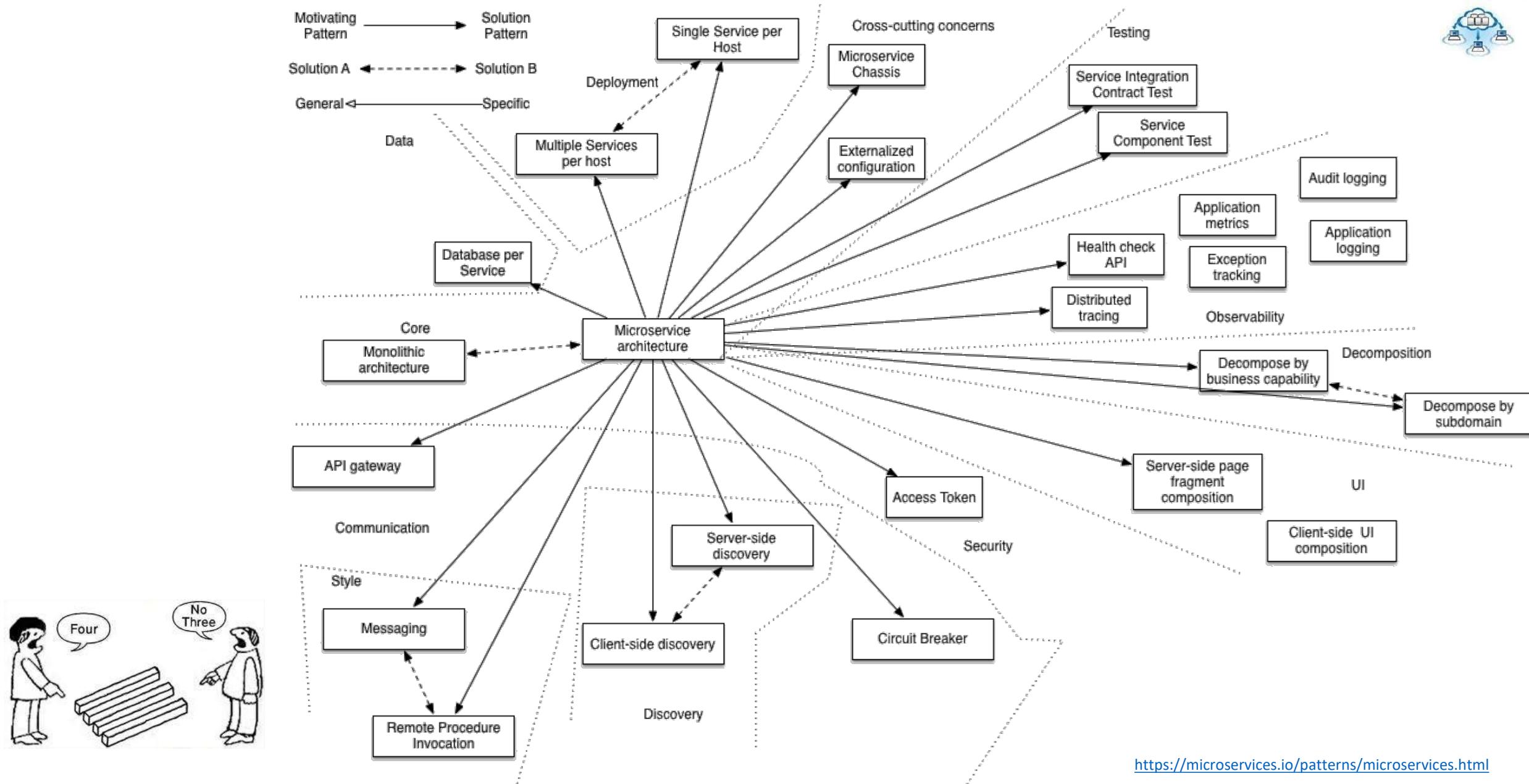


Container Chain

Pattern: Container chains

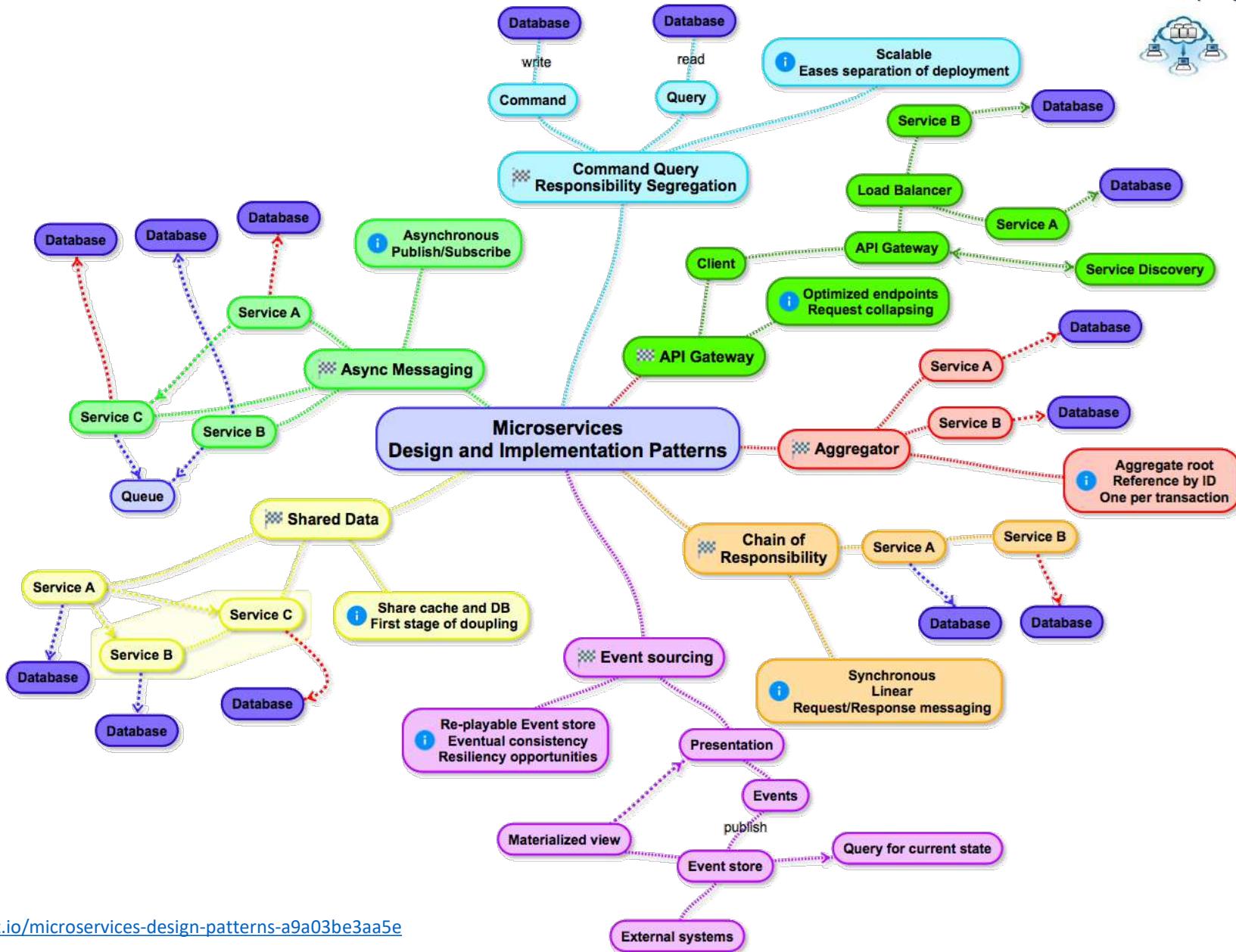
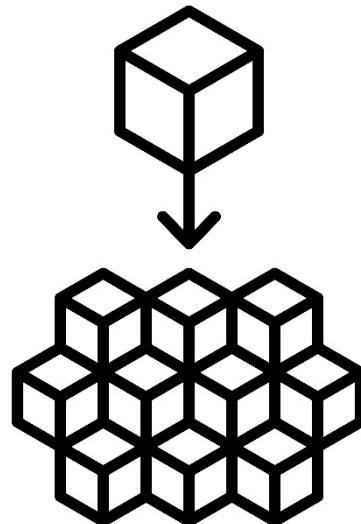
Defined order of starting and stopping sidecar containers.







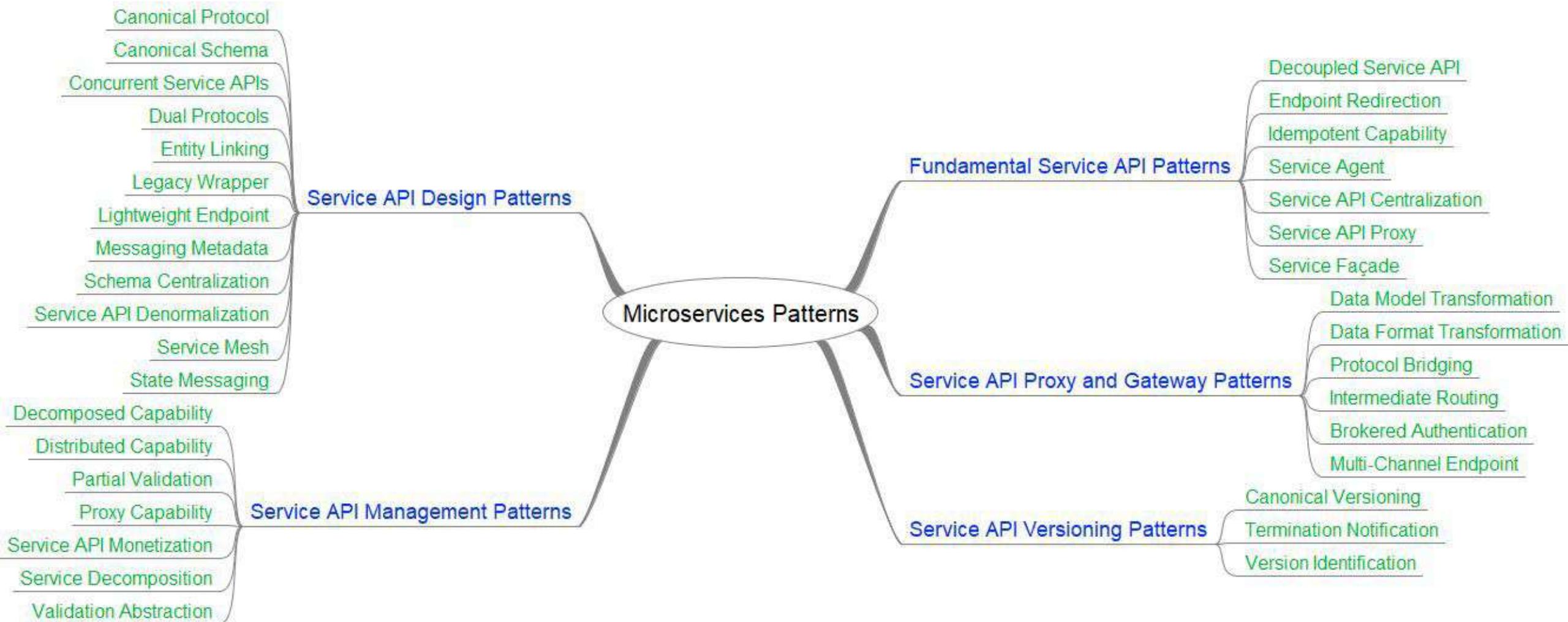
Effective Microservices



<https://techburst.io/microservices-design-patterns-a9a03be3aa5e>

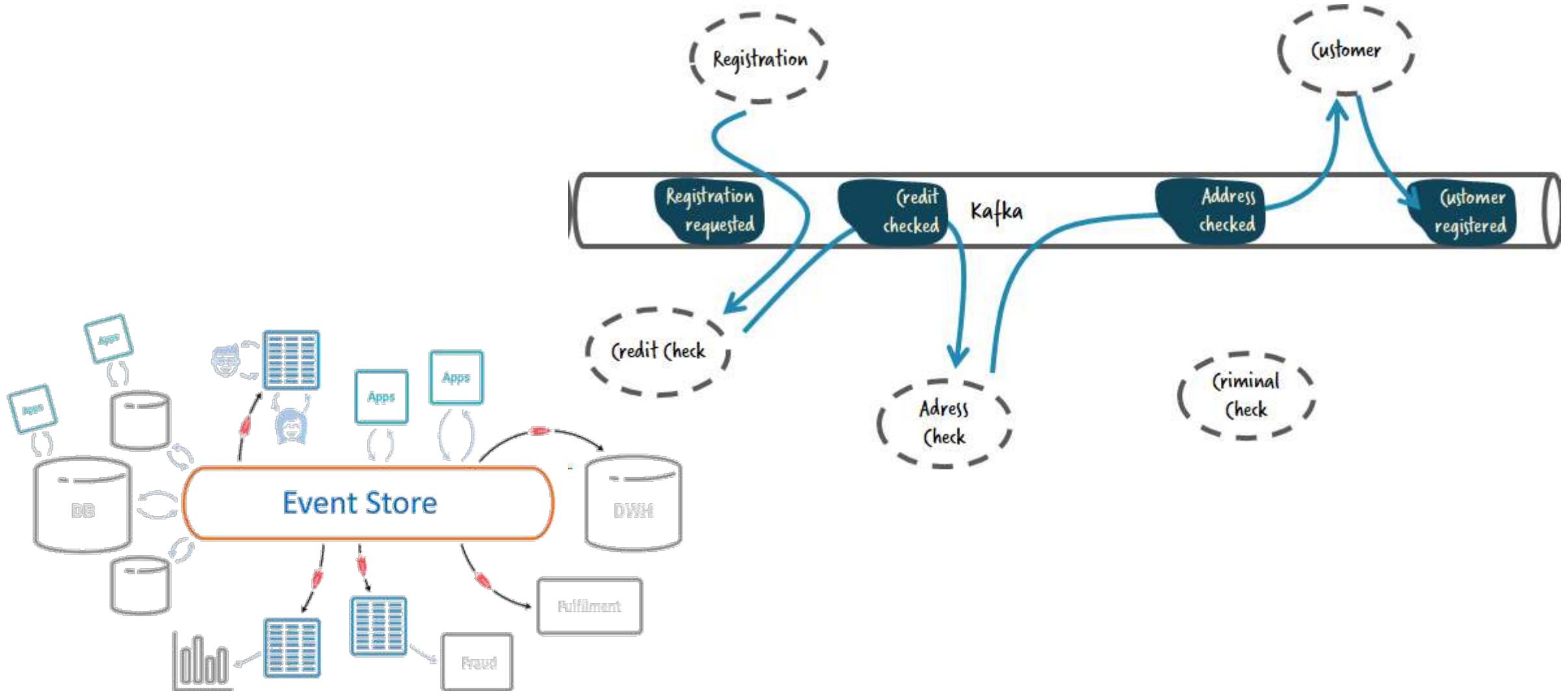


Thomas Erl Classification



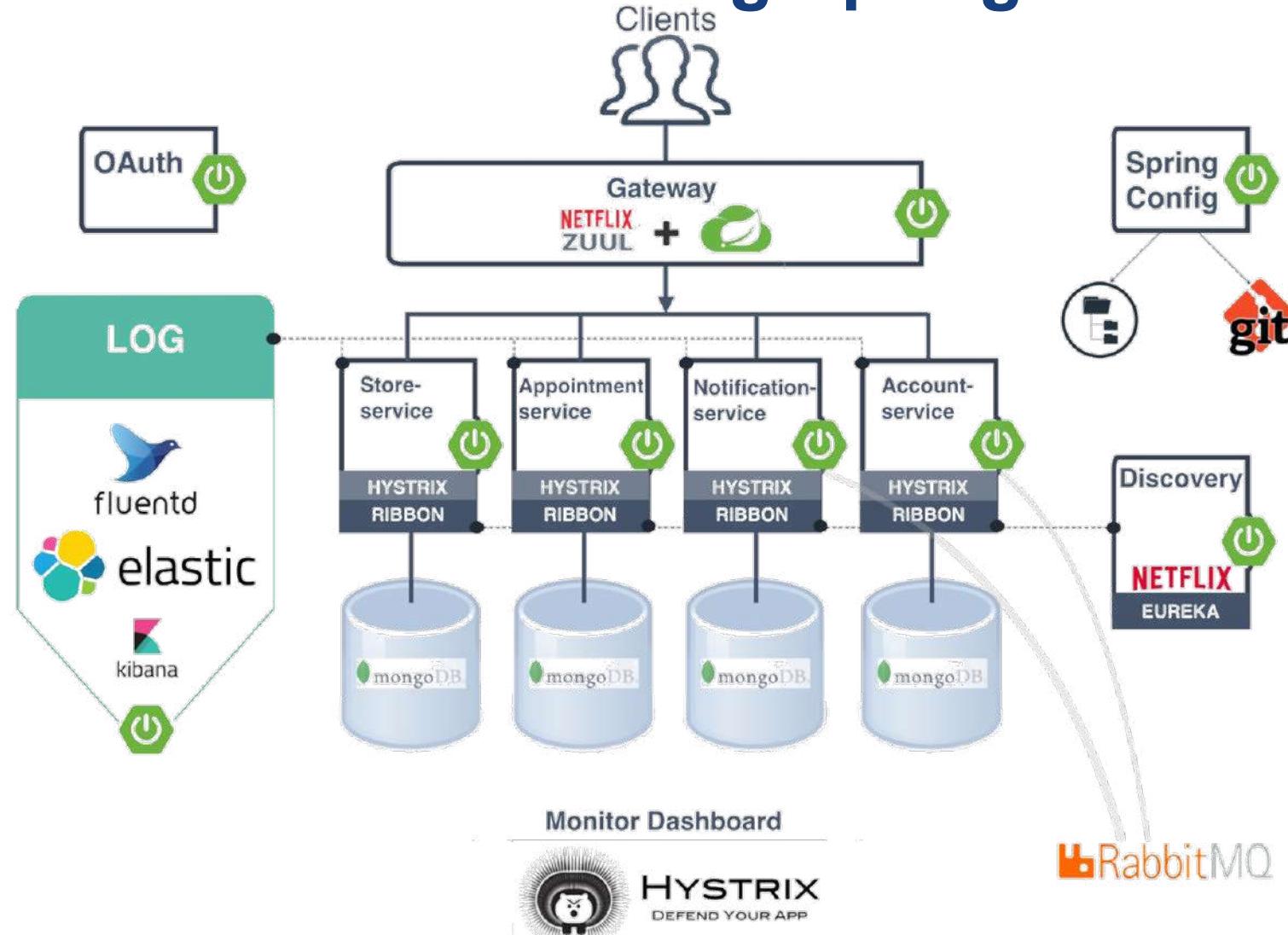


Event-driven Microservices





Event Driven Microservices using Spring Boot



<https://www.linkedin.com/pulse/event-driven-microservices-architecture-using-spring-cloud-bruksha/>

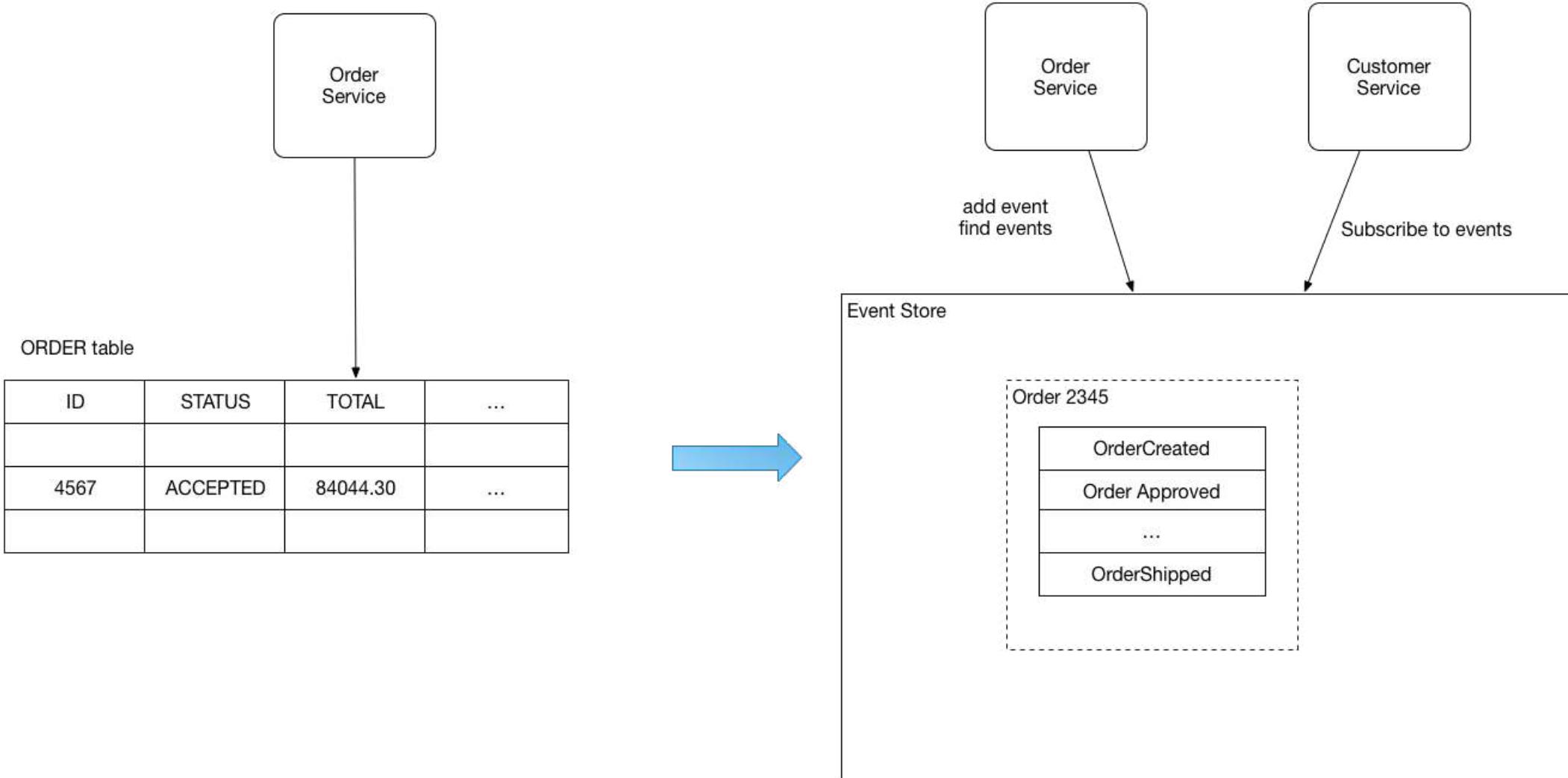


Event Sourcing

- How to reliably/atomically update the database and publish messages/events? Cannot use 2PL
- Event sourcing persists the state of a business entity such an Order or a Customer as a sequence of state-changing events.
 - Whenever the state of a business entity changes, a new event is appended to the list of events. Since saving an event is a single operation, it is inherently atomic. The application reconstructs an entity's current state by replaying the events.
 - Applications persist events in an event store, which is a database of events. The store has an API for adding and retrieving an entity's events.
 - The event store also behaves like a message broker. It provides an API that enables services to subscribe to events. When a service saves an event in the event store, it is delivered to all interested subscribers.
- Some entities, such as a Customer, can have a large number of events.
 - In order to optimize loading, an application can periodically save a snapshot of an entity's current state. To reconstruct the current state, the application finds the most recent snapshot and the events that have occurred since that snapshot. As a result, there are fewer events to replay.



Event Sourcing



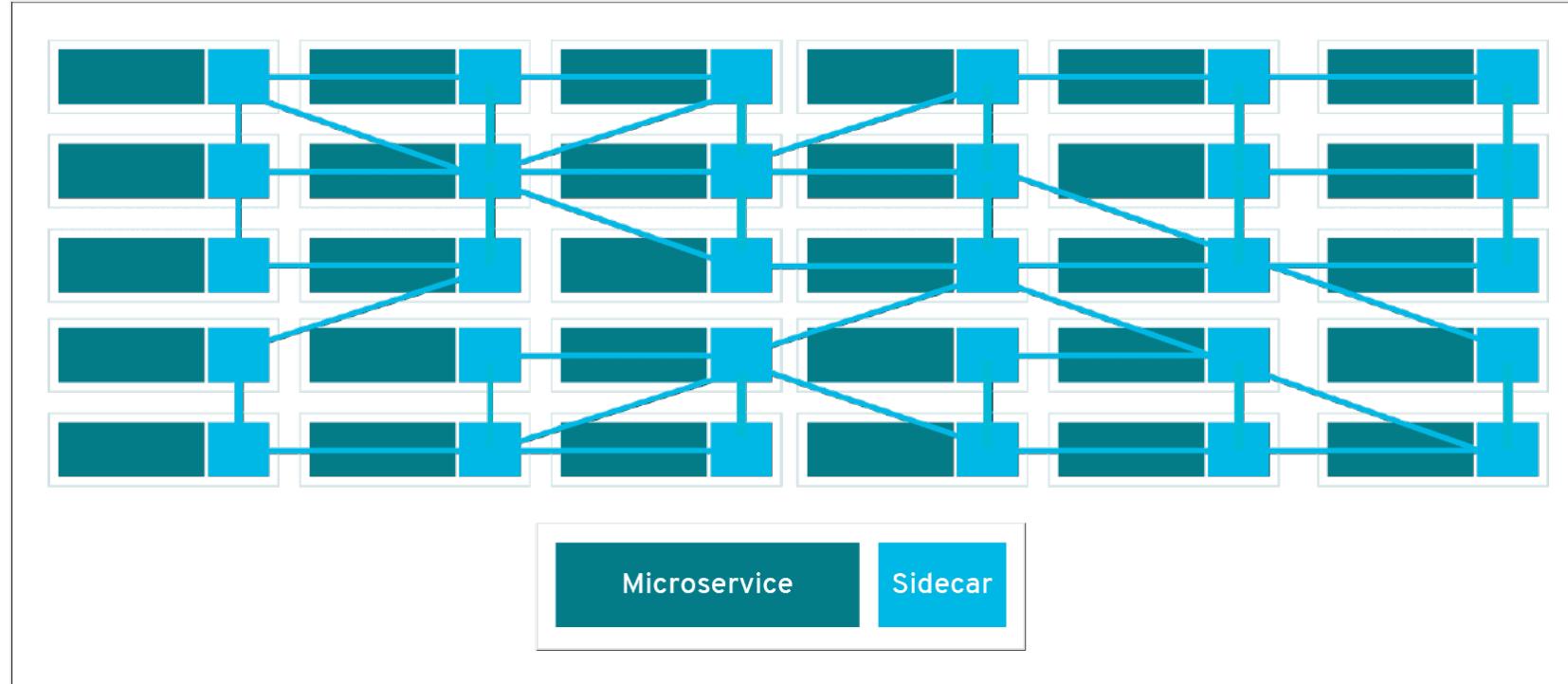


Event Sourcing Analysis

- The advantages of event-sourcing are as follows:
 - Solves atomicity issues.
 - Maintains history and audit of records.
 - Can be integrated with data analytics as historical records are maintained.
- There are a few limitations, which are:
 - Queries on the latest data or a particular piece of data in the event store involve complex handlings.
 - To make the data eventually consistent, this involves asynchronous operations because the data flow integrates with messaging systems.
 - The model that involves inserting and querying the data is the same and might lead to complexity in the model for mapping with the event store.
 - The event store capacity has to be larger in storing all the history of records.



Service Mesh

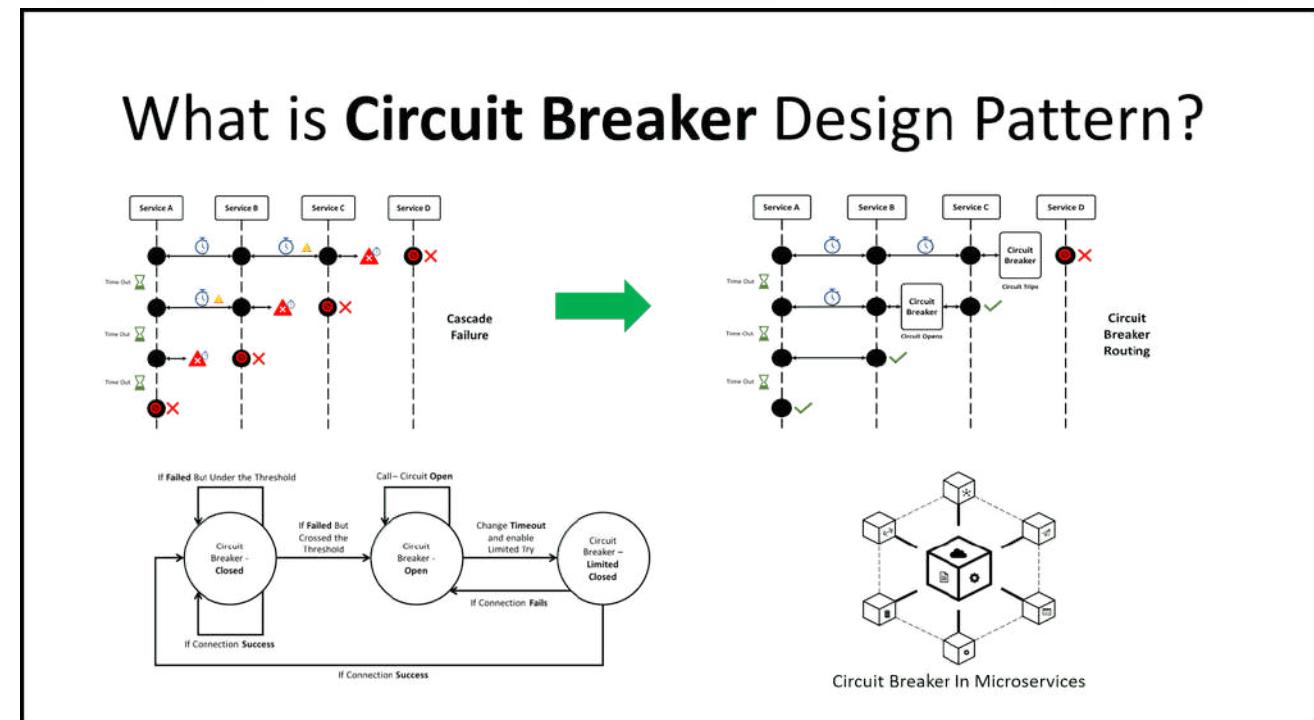
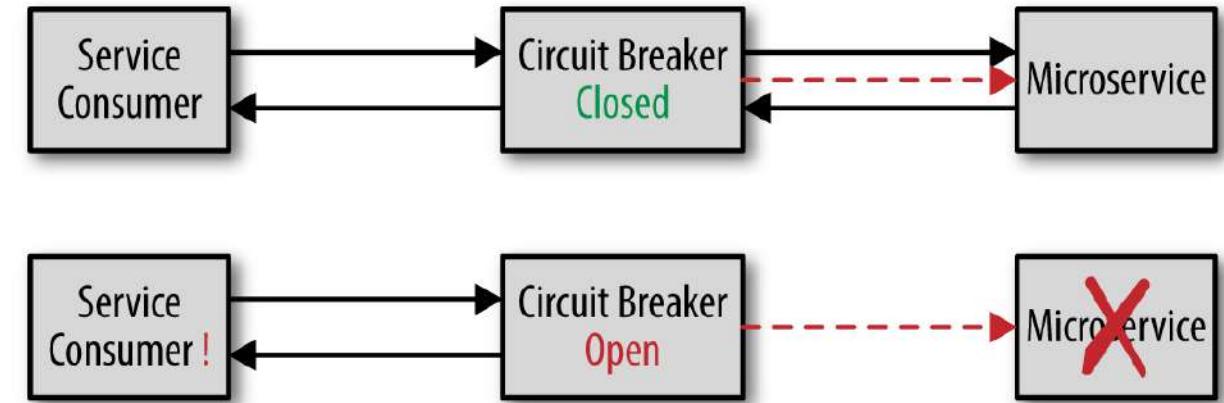


In a service mesh, requests are routed between microservices through proxies in their own infrastructure layer. For this reason, individual proxies that make up a service mesh are sometimes called “sidecars,” since they run *alongside* each service, rather than *within* them. Taken together, these “sidecar” proxies—decoupled from each service—form a mesh network.



Circuit Breaker

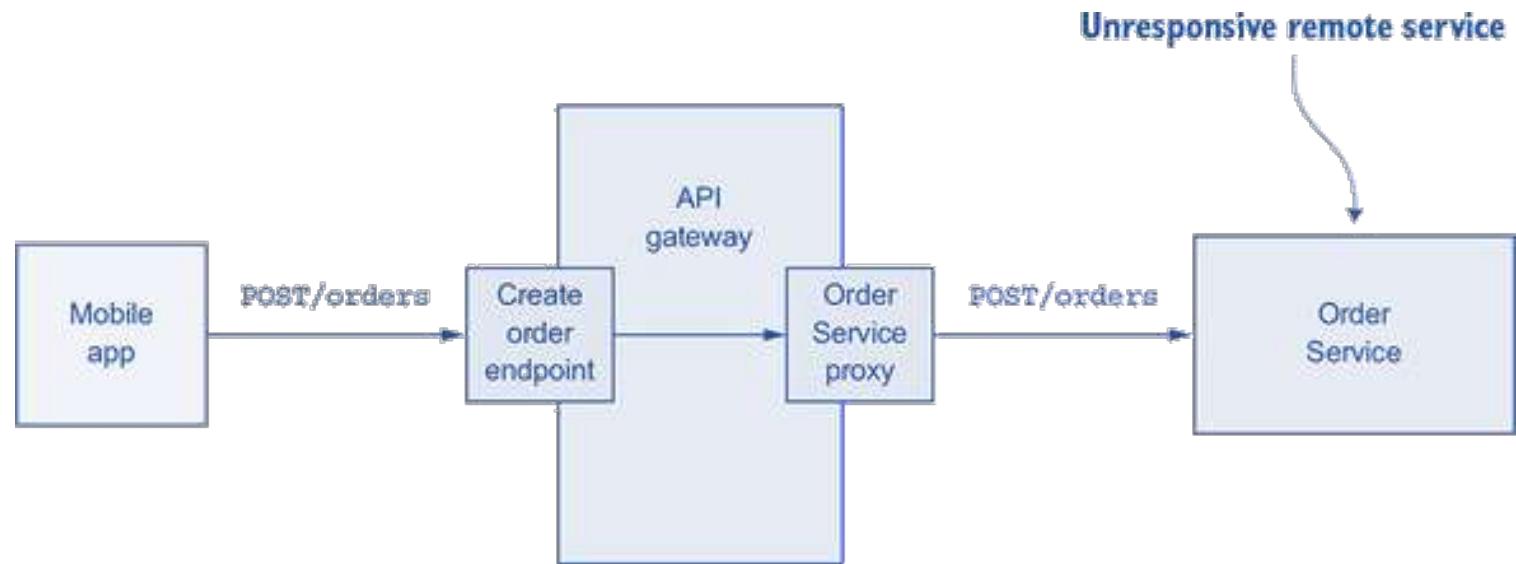
Services sometimes collaborate when handling requests. When one service synchronously invokes another there is always the possibility that the other service is unavailable or is exhibiting such high latency it is essentially unusable. Precious resources such as threads might be consumed in the caller while waiting for the other service to respond. This might lead to resource exhaustion, which would make the calling service unable to handle other requests. The failure of one service can potentially cascade to other services throughout the application.





Handle Failure with Circuit Breaker

- An RPI proxy that tracks the number of successful and failed requests, and if the error rate exceeds some threshold, trip the circuit breaker so that further attempts fail immediately.
 - An API gateway must protect itself from unresponsive services.
 - For example – Order Service as shown below





DATA STORAGE PATTERNS

SHARED AND DEDICATED DATA STORE

CQRS

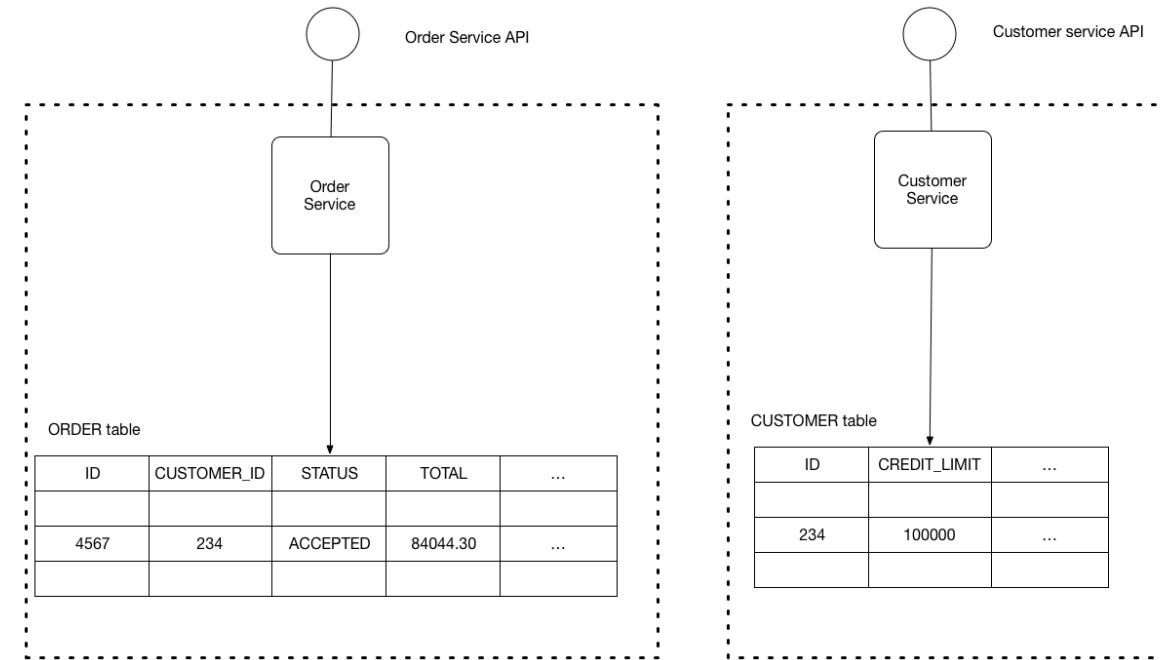
SAGA

SHARDING



Data store per service

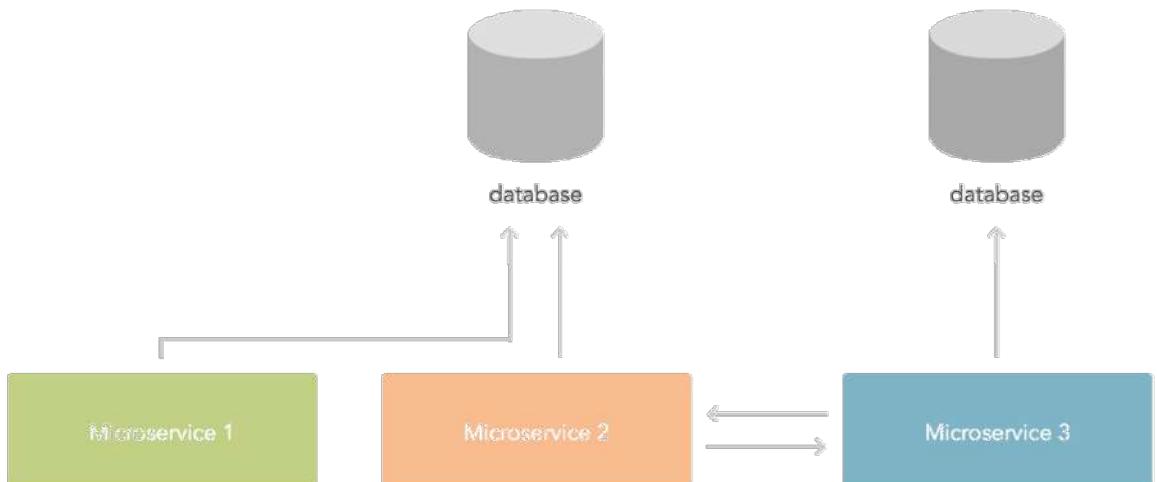
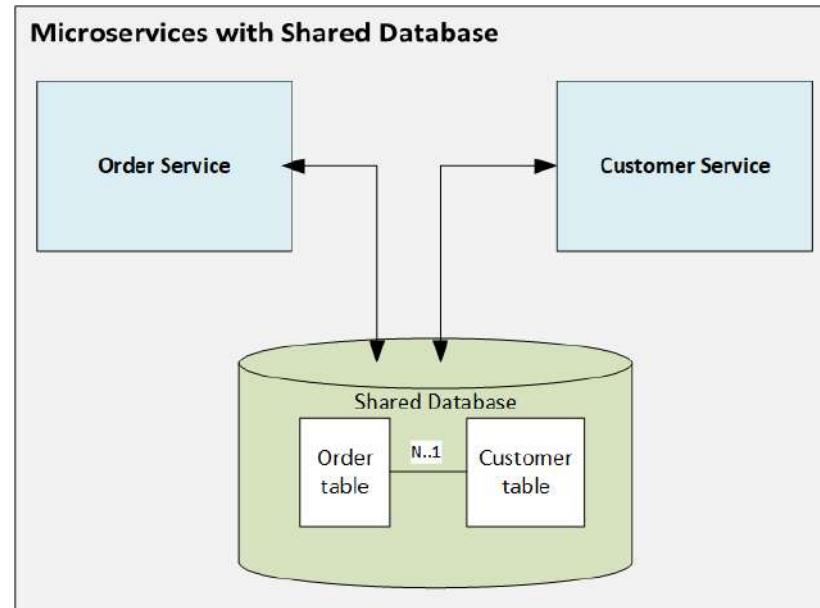
- Most services need to persist data in some kind of database.
 - For example, the Order Service stores information about orders and the Customer Service stores information about customers.
- Keep each microservice's persistent data private to that service and accessible only via its API. A service's transactions only involve its database.





Shared Data store

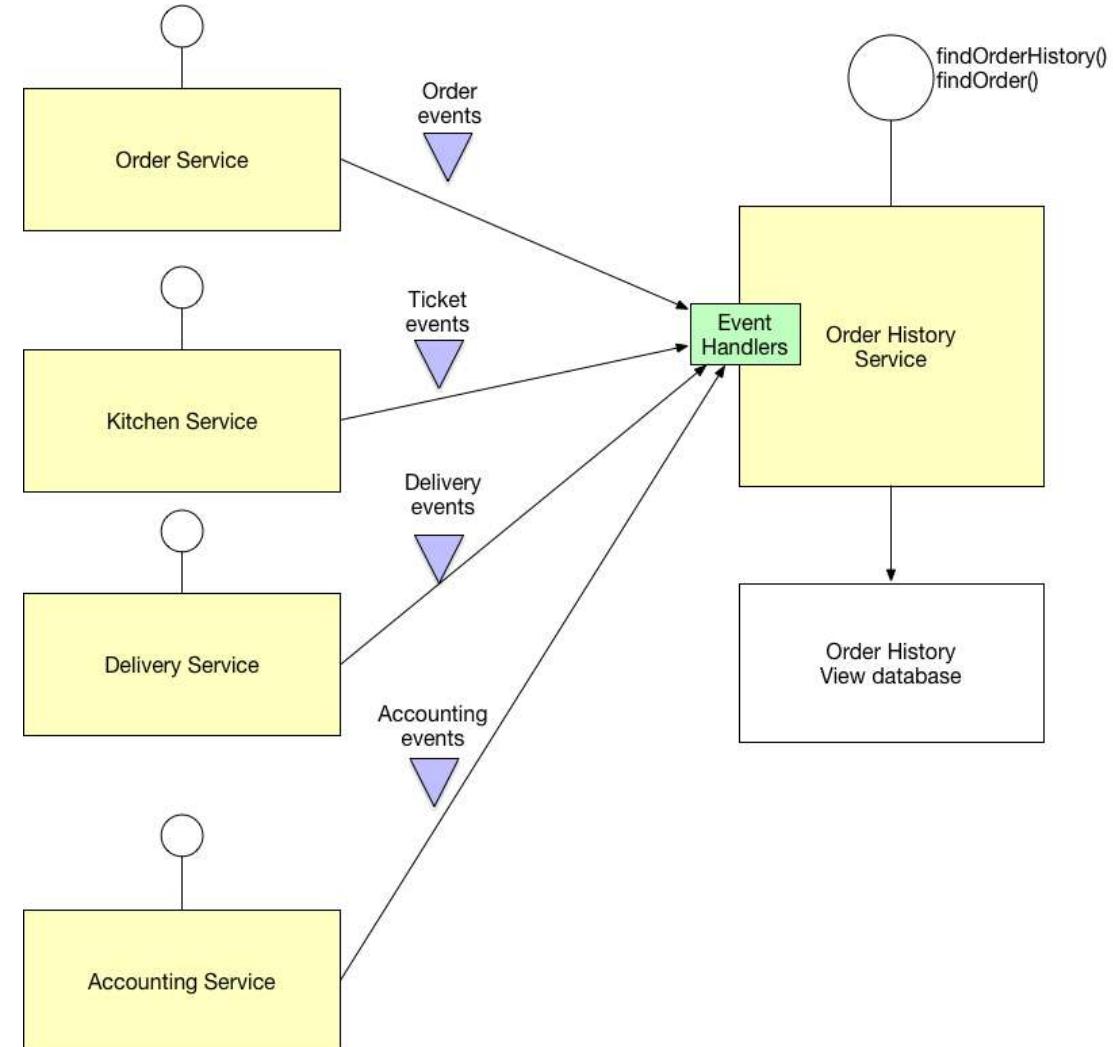
- Use a (single) database that is shared by multiple services.
 - Each service freely accesses data owned by other services using local ACID transactions.
 - It makes much more sense to share data inside a domain boundary if required than share data between unrelated domains.





Command Query Responsibility Segregation (CQRS)

- How to implement a query that retrieves data from multiple services in a microservice architecture? CQRS
 - Define a view database, which is a read-only replica that is designed to support that query.
 - The application keeps the replica up to date by subscribing to Domain events published by the service that own the data.





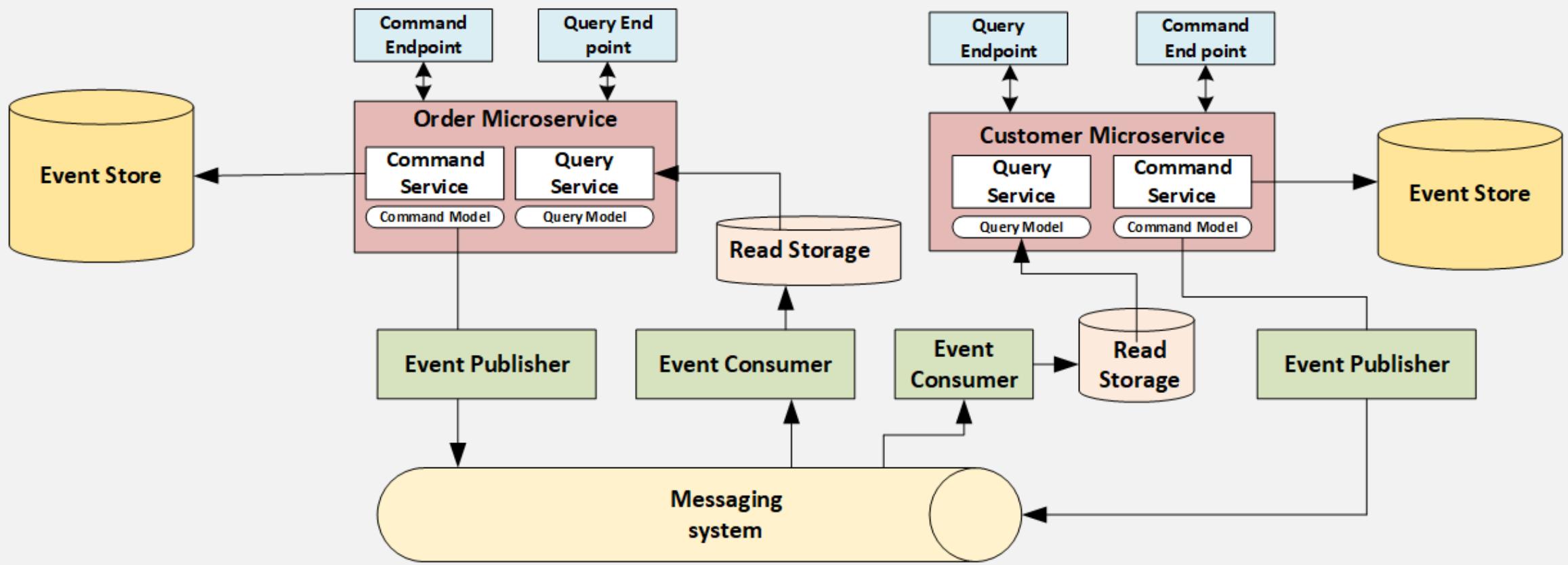
CQRS - Shared Data via Event Sourcing

- The advantages of CQRS integrated with event sourcing and microservices are:
 - Leveraging microservices for modularity with separate databases.
 - Leveraging event sourcing for handling atomic operations.
 - Maintain historical/audit data for analytics with the implementation of event sourcing.
 - CQRS having separate models and services for read and insert operations.
 - Request load can be distributed between read and insert operations.
 - Read operations can be faster as the load is distributed between read and insert services.
 - Read model or DTO need not have all the fields as a command model, and a read model can have required fields by the client view which can save the capacity of the read store.
- The limitations of this approach are:
 - Additional maintenance of infrastructure, like having separate databases for command and query requests.
 - Models should be designed in an optimal way, or this will lead to complexity in handling and troubleshooting.



CQRS and Event Sourcing

Microservices with CQRS and Event Sourcing



<https://dzone.com/articles/microservices-with-cqrs-and-event-sourcing>



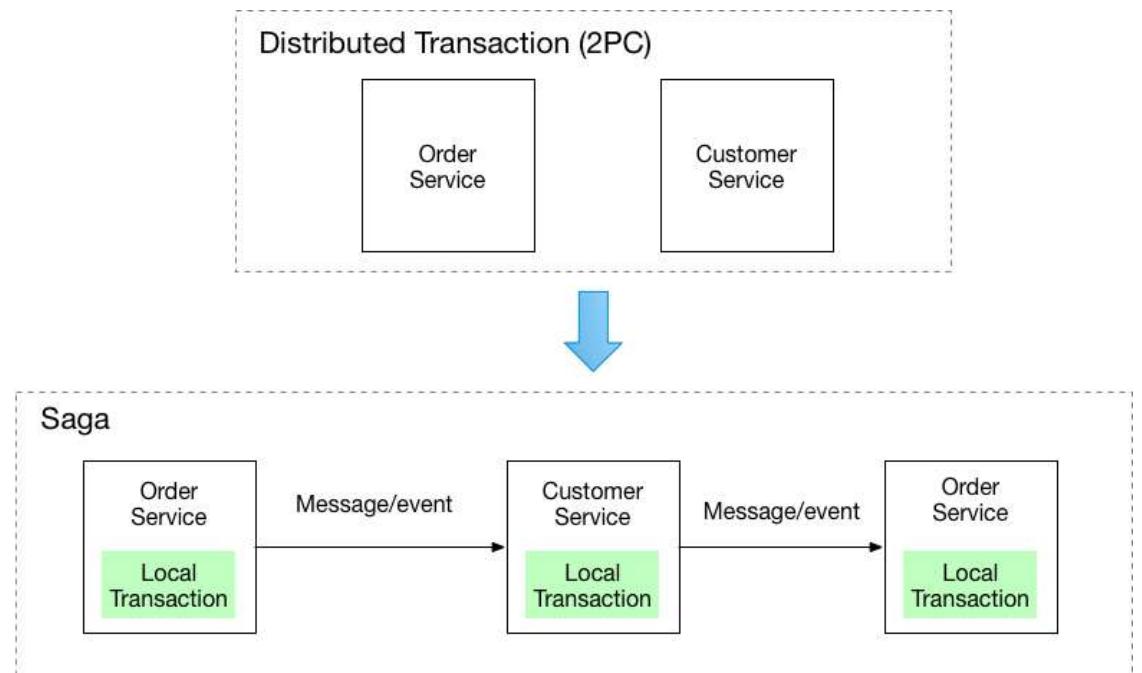
SAGA

- Each service has its own database.
- Some business transactions, however, span multiple service so you need a mechanism to ensure data consistency across services.
- For example, lets imagine that you are building an e-commerce store where customers have a credit limit.

There are two ways of coordination sagas:

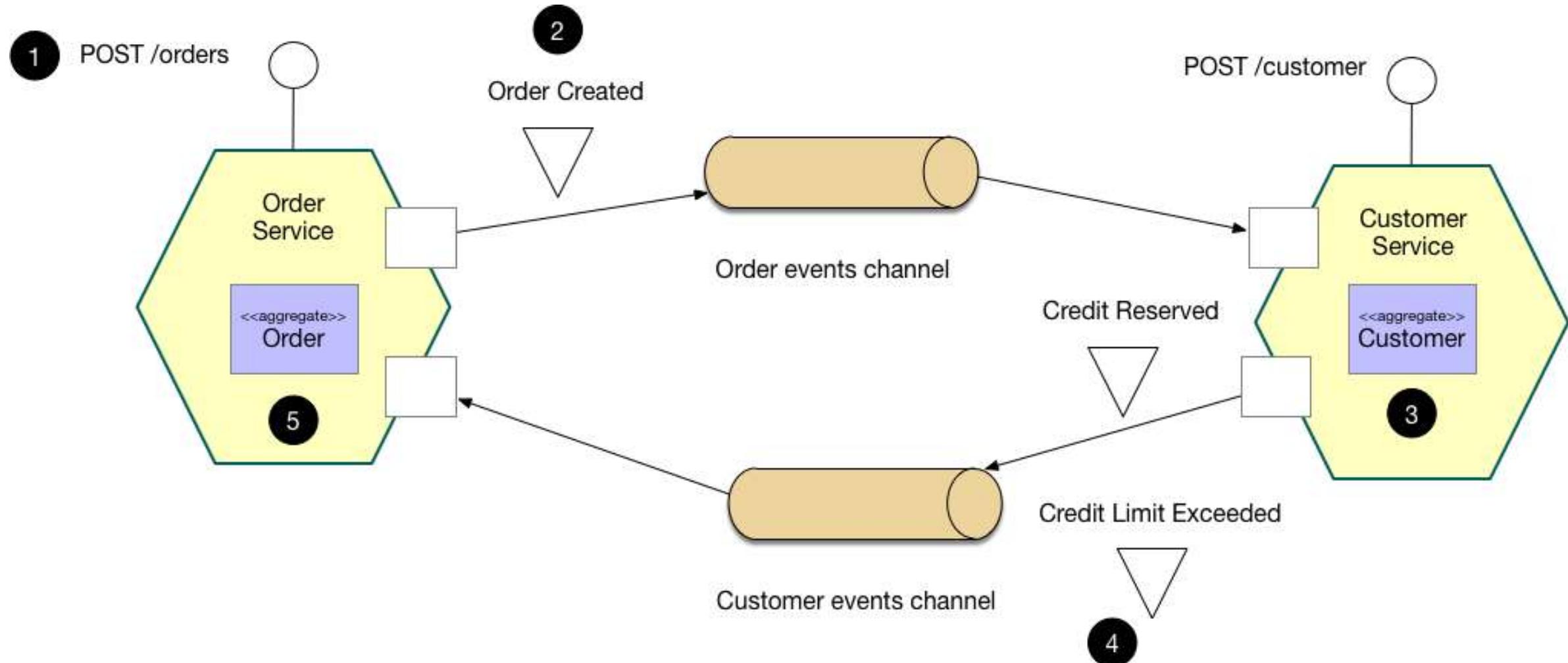
1. **Choreography** - each local transaction publishes domain events that trigger local transactions in other services
2. **Orchestration** - an orchestrator (object) tells the participants what local transactions to execute

<https://microservices.io/patterns/data/saga.html>



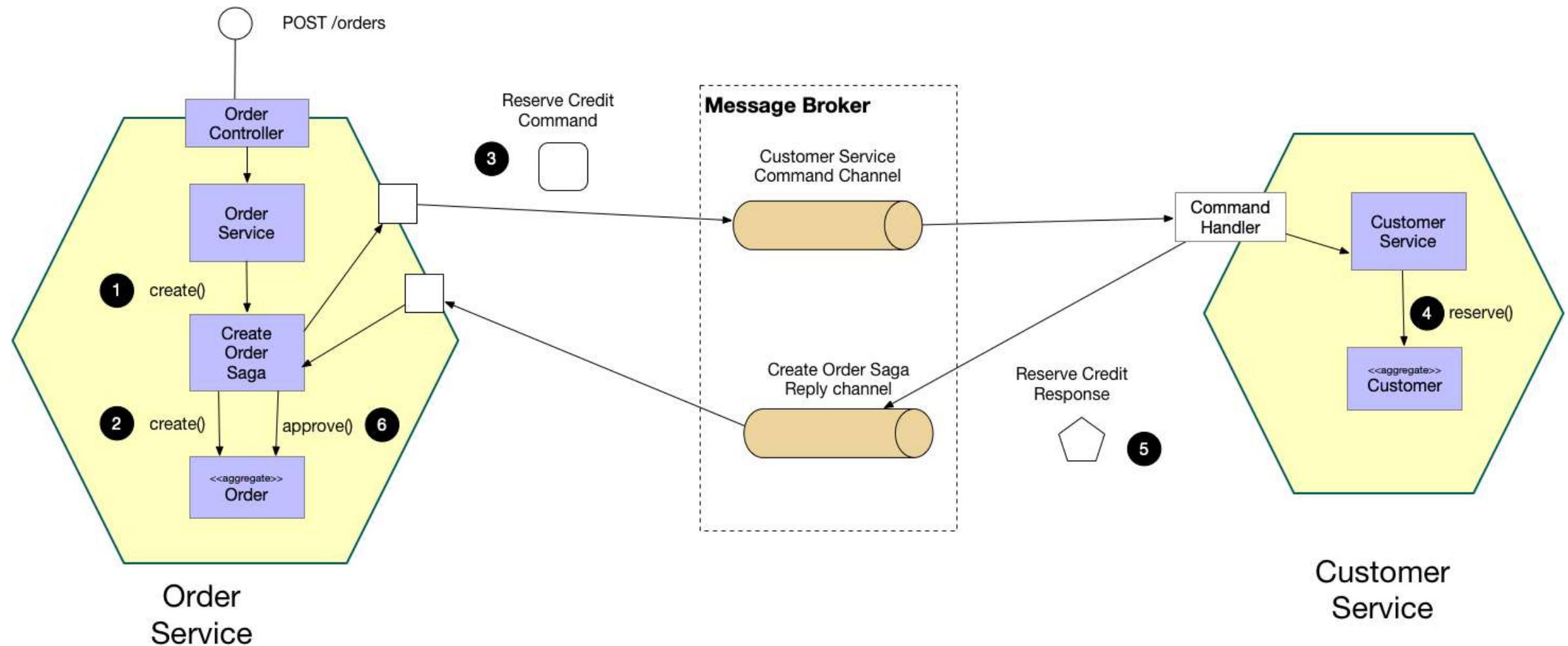


Example: Choreography-based saga





Example: Orchestration-based saga





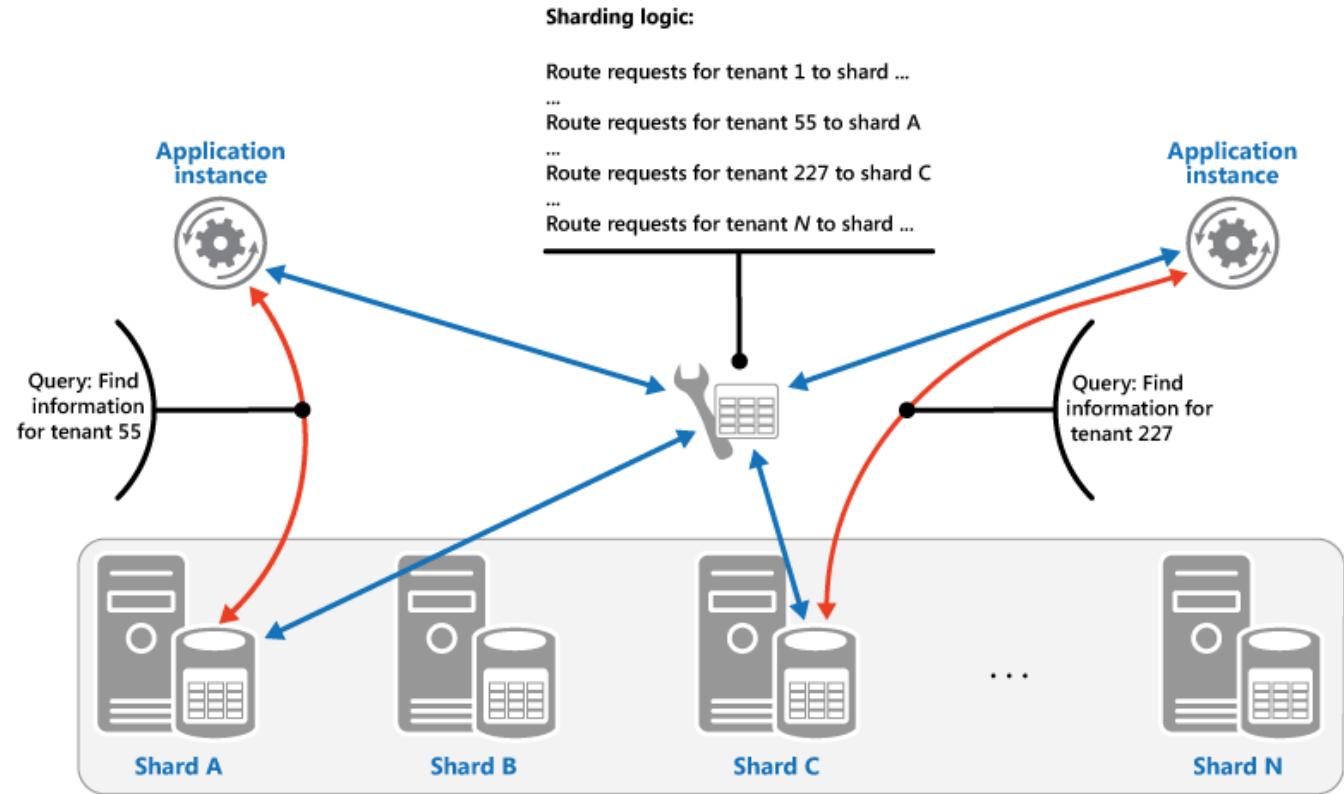
Sharding Patterns

- A data store hosted by a single server might be subject to the following limitations:
 - **Storage space.**
 - A data store for a large-scale cloud application is expected to contain a huge volume of data that could increase significantly over time.
 - **Computing resources.**
 - A cloud application is required to support a large number of concurrent users, each of which run queries that retrieve information from the data store.
 - **Network bandwidth.**
 - The performance of a data store running on a single server is governed by the rate the server can receive requests and send replies.
 - **Geography.**
 - It might be necessary to store data generated by specific users in the same region as those users for legal, compliance, or performance reasons, or to reduce latency of data access.



The Lookup Strategy

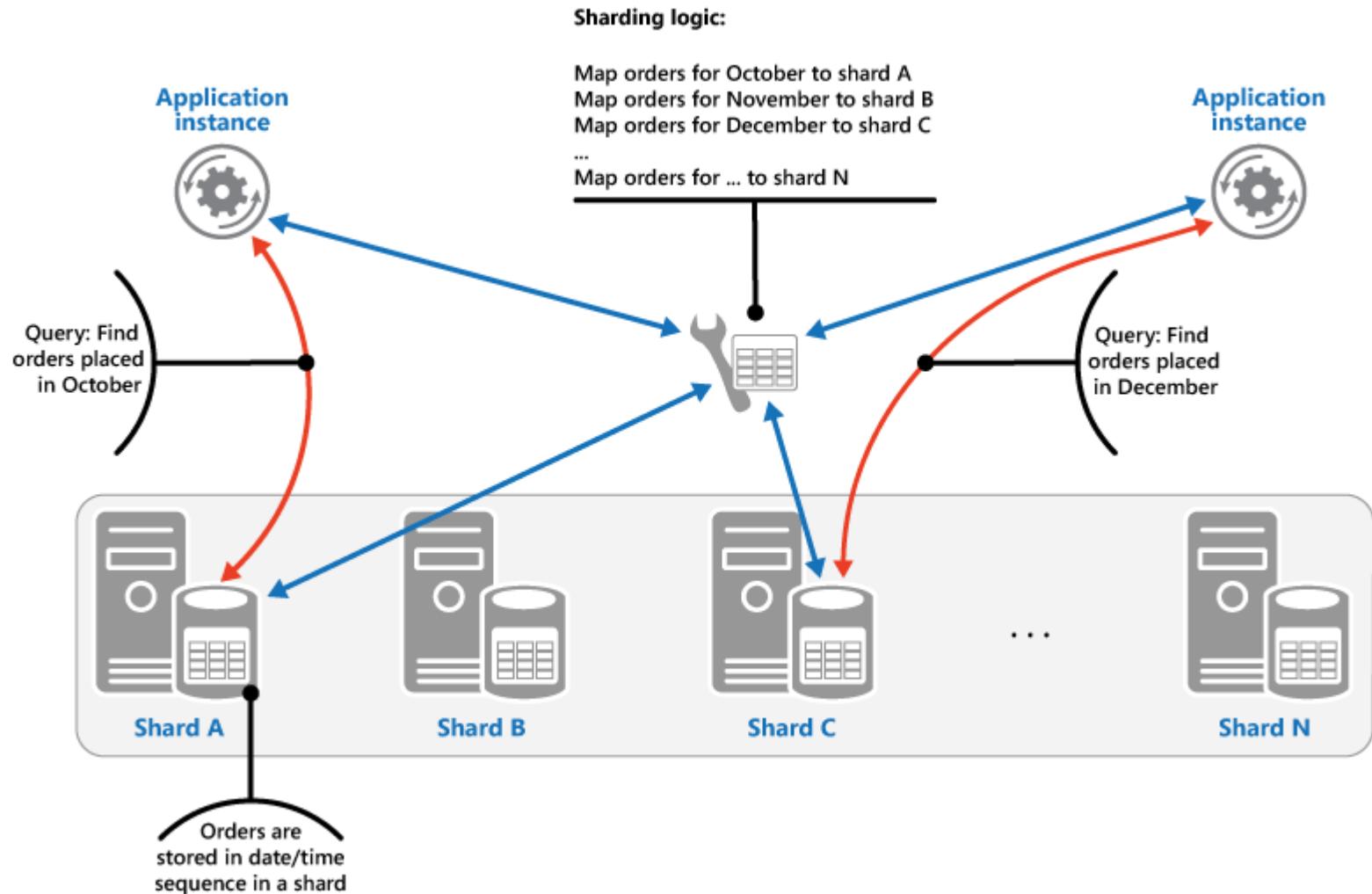
- Implements a map that routes a request for data to the shard that contains that data using the shard key.
 - In a multi-tenant application all the data for a tenant might be stored together in a shard using the tenant ID as the shard key.





The Range Strategy

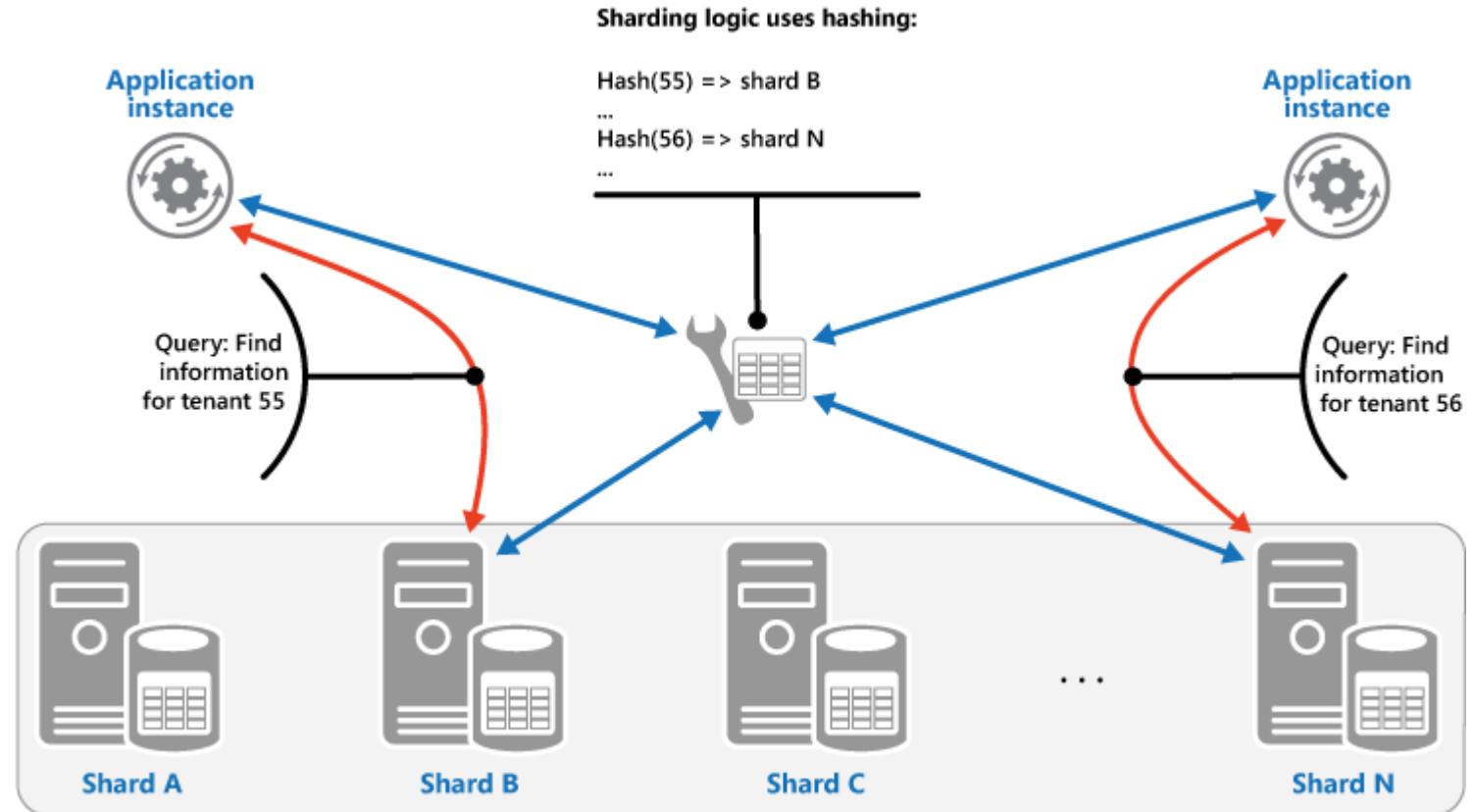
- Groups related items together in the same shard, and orders them by shard key—the shard keys are sequential.





The Hash Strategy

- The purpose of this strategy is to reduce the chance of hotspots
 - shards that receive a disproportionate amount of load
 - distributes the data across the shards in a way that achieves a balance between the size of each shard and the average load that each shard will encounter.





MULTITENANCY MODELS

INTRODUCTION TO TENANCY MODELS



Key Design Considerations

1. **Multi-Tenancy:** With this multi-tenancy model, a single version of the application, with a single configuration (hardware, network, operating system), is used for all customers ("tenants").
2. **Scalability:** Designing and maintaining computing systems for scalability - it affects both the cloud consumers and cloud producers.
 - *Scalability can be defined as the ability of a system, network, or process to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth.*
3. **Data Storage:** Designing normal and scalable data stores in a polyglot persistence environment.



Making Dumplings Analogy





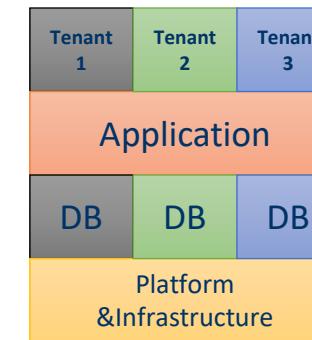
Multi-Tenant Architecture Options Continuum

Tenant 1	Tenant 2	Tenant 3
App	App	App
DB	DB	DB
Plat& Infra	Plat& Infra	Plat& Infra

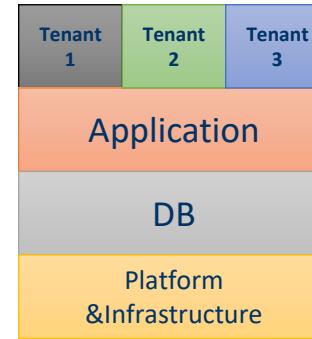
VM Isolation

Tenant 1	Tenant 2	Tenant 3
App	App	App
DB	DB	DB
Plat& Infra	Plat& Infra	Plat& Infra

Application Isolation



Storage Isolation



Sub Tenancy Isolation



What about
mixed stack?

Physical Isolation

Run Time Isolation

Application Isolation

Isolating....

Sharing.....

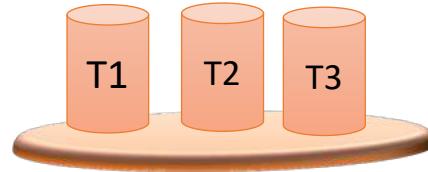
Easier Customization, security & control

Better Scale of Economy, Simpler management



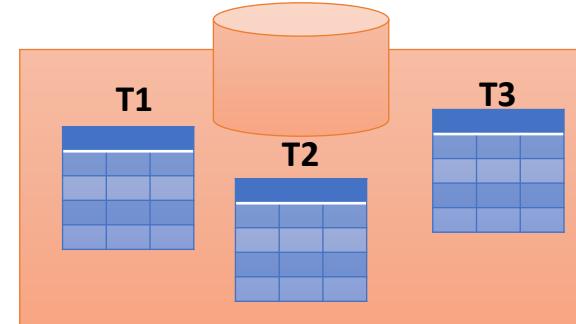
Multi-Tenant Schema Options Continuum - 1

Separated DB



Isolated

Separated Schema



Easier Customization, security & control

Shared Schema

Better Scale of Economy

TenantID	CustName	Address	
4	TenantID	ProductID	ProductName
1	4	TenantID	Shipment
6	1	4711	324965
4	6	132	2006-02-21
4	680	115468	2006-04-08
4711		654109	2006-03-27
		324956	2006-02-23



Multi-Tenant Schema Options Continuum - 2.

Separated DB	Separated Schema	Shared Schema
<p>Approach:</p> <ul style="list-style-type: none"> Separate database for each tenant Database maintains data dictionary <p>Advantages:</p> <ul style="list-style-type: none"> Easy to implement Meta data identifies database instance for each tenant <p>Tradeoff:</p> <ul style="list-style-type: none"> Number of tenants per database server is low Infrastructure cost of providing service rise quickly <p>When to use:</p> <ul style="list-style-type: none"> When tenant has data isolation requirements Able to monetize the data extension/isolation feature 	<p>Approach:</p> <ul style="list-style-type: none"> All tenants data in one database. Pre-defined set of custom fields <p>Advantages:</p> <ul style="list-style-type: none"> Easy to implement Maximize number of tenants per database server <p>Tradeoff:</p> <ul style="list-style-type: none"> Tendency to results in sparse table <p>When to use:</p> <ul style="list-style-type: none"> When data co-mingling is OK Easy to anticipate pre-defined custom fields 	<p>Approach:</p> <ul style="list-style-type: none"> All tenants data in one database. Pre-defined set of custom fields <p>Advantages:</p> <ul style="list-style-type: none"> Easy to implement Maximize number of tenants per database server <p>Tradeoff:</p> <ul style="list-style-type: none"> Tendency to results in sparse table <p>When to use:</p> <ul style="list-style-type: none"> When data co-mingling is OK Easy to anticipate pre-defined custom fields



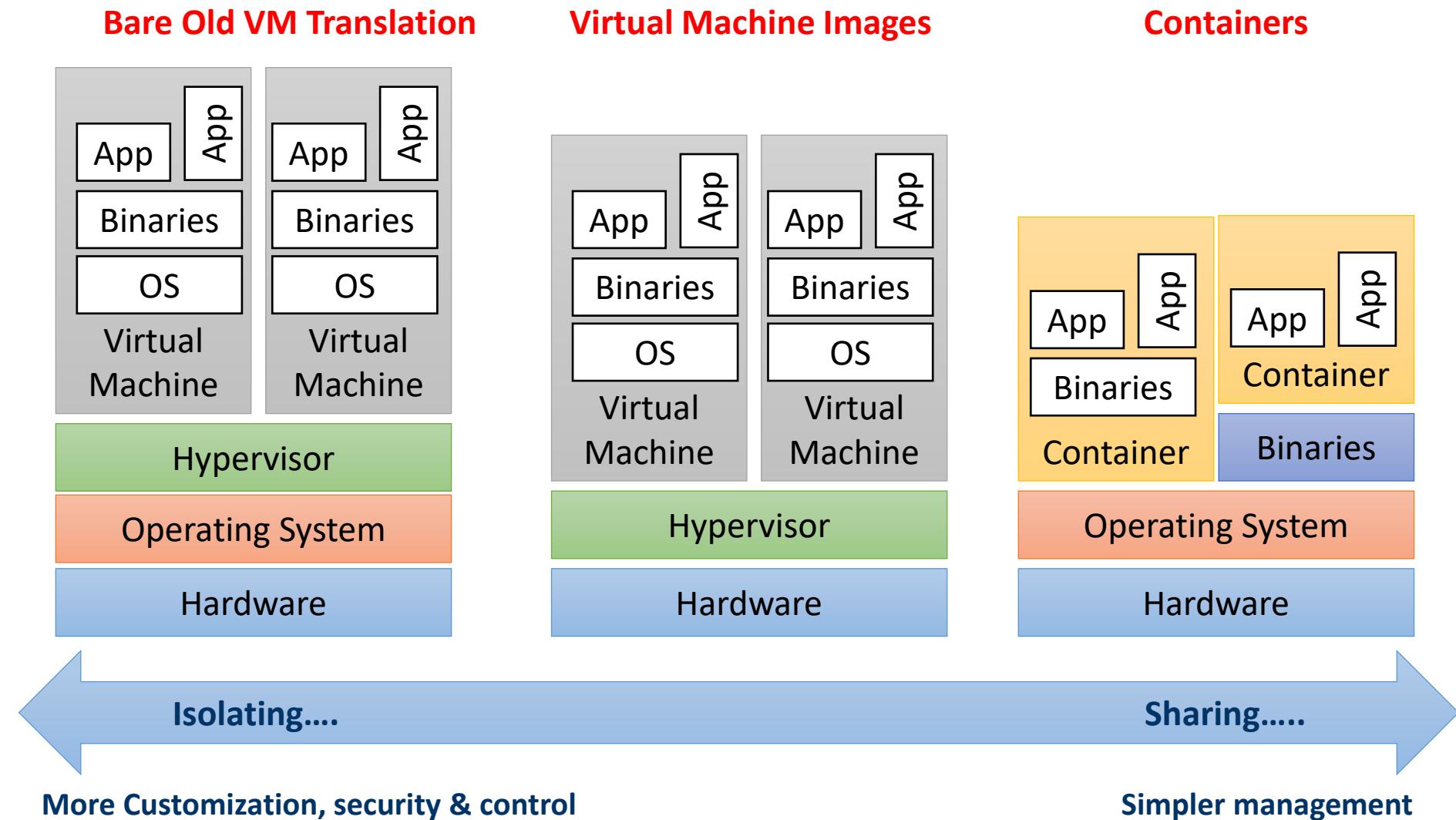
Multi-Tenant Data Storage Options Continuum

- **Are we storing relational (structured) or unstructured data?** Structured data is best suited to be stored on RDBMS and blob storage can be used to store media files etc.,
- **How easy it is to query the data?** Key/Value data stores are good choice to get a single row of data, but are not the best option for complex queries.
- **Can aggregations etc., be executed at server-side?** Cases where data aggregation (group by) needs to be performed.
- **Requirements on data scalability and support for native scale-out? How easy is it to add/remove capacity (size and throughput)?**
- **Ability to instrument, monitor and manage?**
- **Support for Transactions**
- **Backup, restore and disaster recovery**
- **Specialized security needs, data security and audit capabilities?**

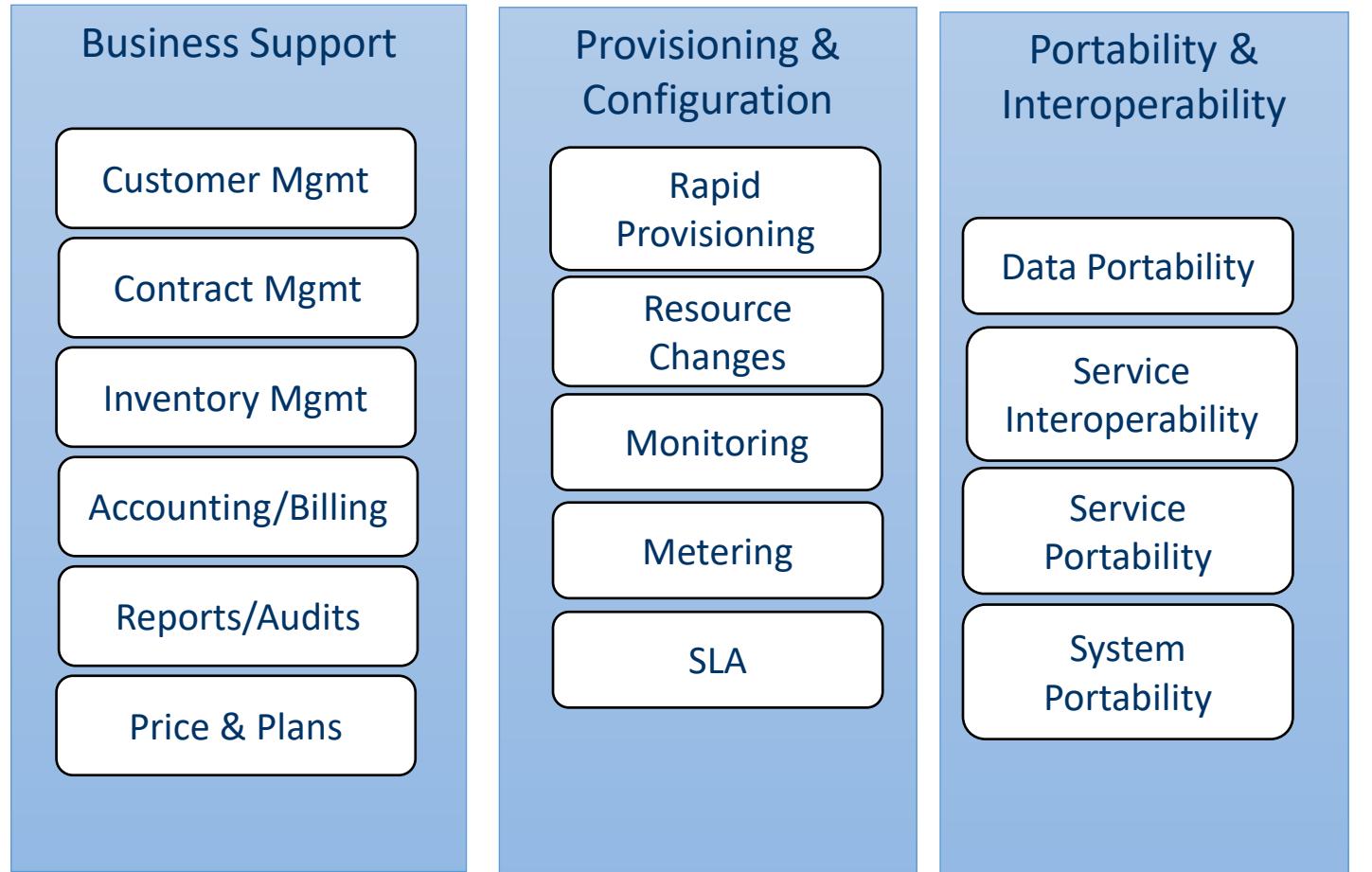
Relational	Key/Value	Column Family	Document	Graph
 <ul style="list-style-type: none"> Windows Azure SQL Database SQL Server Oracle MySQL SQL Compact SQLite Postgres 	 <ul style="list-style-type: none"> Windows Azure Blob Storage Windows Azure Table Storage Windows Azure Cache Redis Memcached Riak 	 <ul style="list-style-type: none"> Cassandra HBase 	 <ul style="list-style-type: none"> MongoDB RavenDB CouchDB 	 <ul style="list-style-type: none"> Neo4j



Multi-Tenant Deployment Options Continuum



Managed Services & Maturity



Levels	Description
Level 0: Outsourcing is not SaaS	With outsourcing, a major application or a unique application landscape for a large enterprise customer is operated by a service provider. The outsourcing company can't leverage this application for a second customer, outsourcing does not qualify as SaaS.
Level 1: Manual ASP business models target midsize companies	A hosting provider runs packaged applications like SAP's ERP 6.0, which require significant IT skills, for multiple midsize enterprises. The client has a dedicated server running its instance of the application and is able to customize the installation in the same way as self-hosted applications.
Level 2: Industrial ASPs cut the operating costs of packaged applications to a minimum	An ASP uses sophisticated IT management software to provide identical software packages with customer-specific configurations to many SMB customers. The software package is the original software created for self-hosted deployment.
Level 3: Single-app SaaS is an alternative to traditional packaged applications	Software vendors create new generations of business applications that have SaaS capabilities built in. Web-based user interface (UI) concepts and the ability to serve a number of tenants with scalable infrastructure are typical characteristics. Customization is restricted to configuration. Single-app SaaS adoption focusing on SMBs. Salesforce.com's (CRM) applications initially entered the market at this level.
Level 4: Business-domain SaaS provides all the applications for an entire business domain	Advanced SaaS vendors provide a well-defined business applications and a platform for additional business logic. This complements the original single application of the previous level with third-party packaged SaaS solutions and custom extensions. The model satisfies the requirements of a large enterprise, which can migrate a complete business domain like "customer care" toward SaaS.
Level 5: Dynamic Business Apps-as-a-service is the visionary target	Forrester's Dynamic Business Application imperative embraces a new paradigm of application development: "design for people, build for change." Advanced SaaS vendors coming from level 4 will provide a comprehensive application and integration platform on demand, which they will pre-populate with business applications or business services. They can compose tenant-specific and user-specific business applications on various levels. The resulting process agility will attract customers, including large enterprise.

Source: Forrester's Research



CONTENT AGGREGATION PATTERNS

AGGREGATION BY CLIENT

API AGGREGATION

MICROSERVICES AGGREGATION

DATABASE AGGREGATION



Microservices and Content Management

1. Content Aggregation Patterns

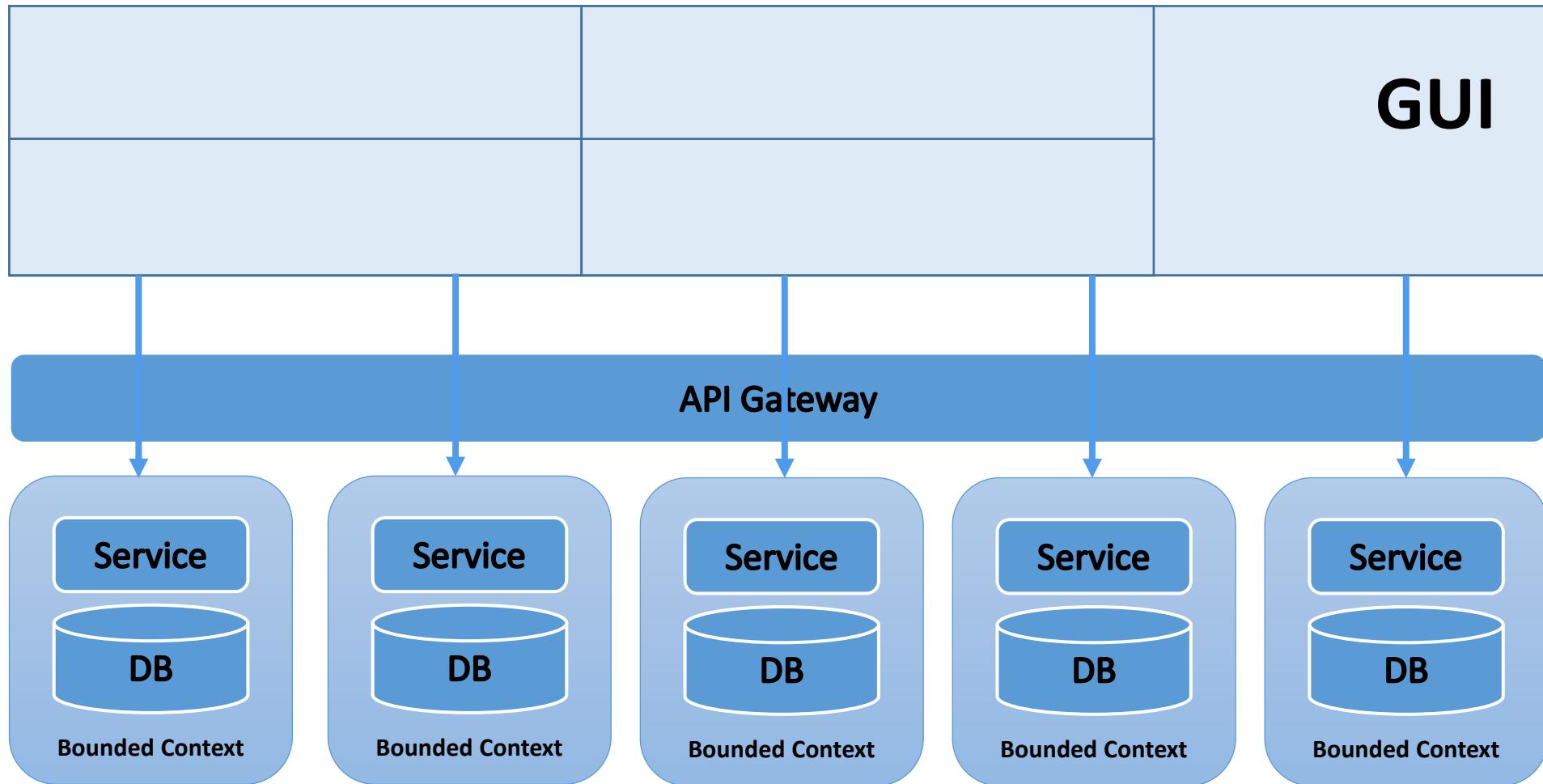
1. Aggregation by Client
2. API Aggregation
3. Microservices Aggregation
4. Database Aggregation

2. Microservices Coordination



Content Aggregation Patterns

Aggregation by client





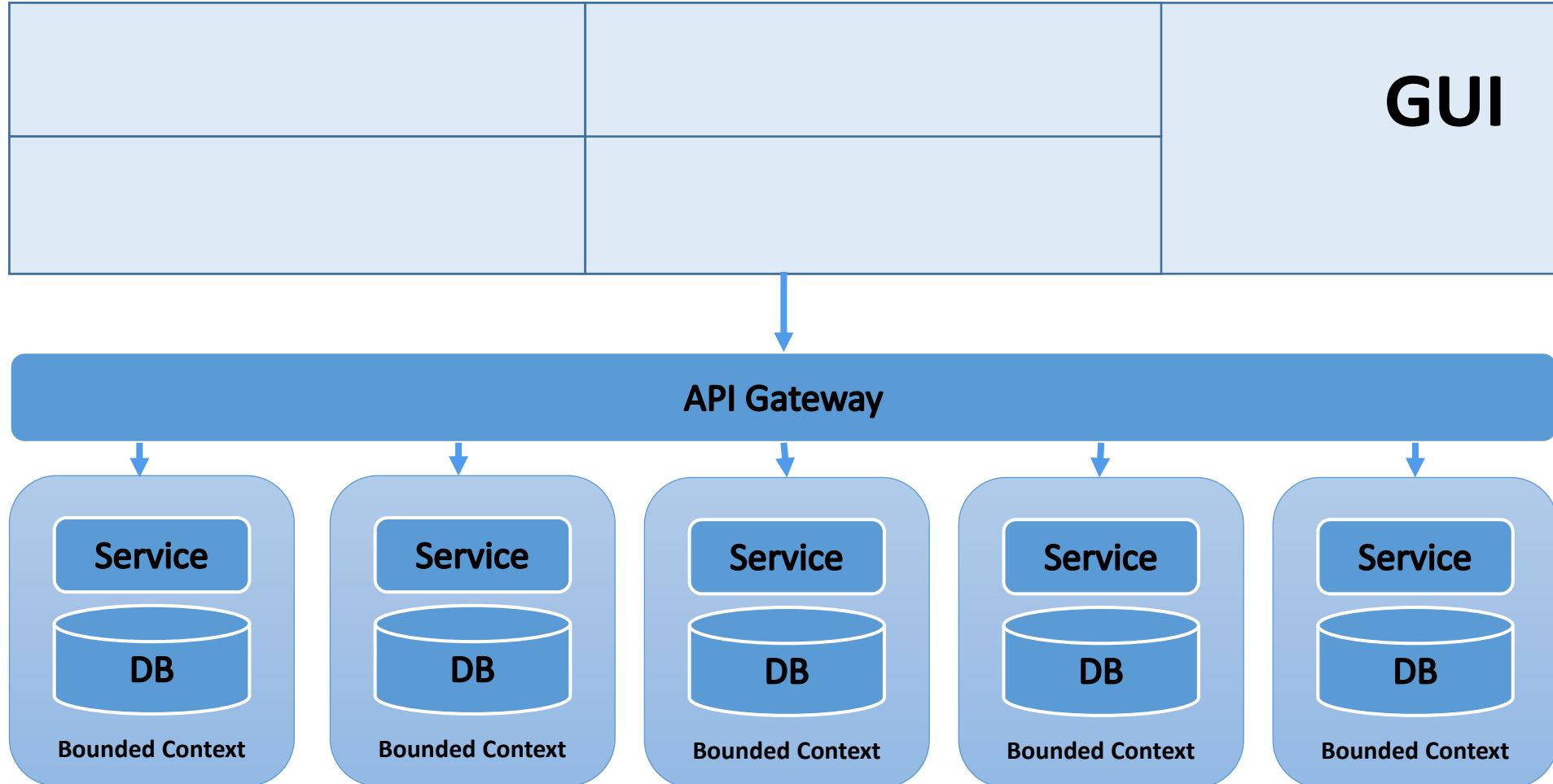
Aggregation by client

- Aggregation at the last mile. This pattern is typically used in the home page that aggregates various subject areas.
- Benefits
 - Decoupled approach at the services layer. Easier for agility and maintainability at each individual service.
 - Faster perceived performance at the UI layer, since the requests, can run in parallel to populate the various areas on the screen. More enhanced when there is a higher bandwidth available to fetch data in parallel.
- Trade-offs
 - Sophisticated user interface processing capabilities, such as Ajax and single-page application required
 - The knowledge of aggregation is exposed at the UI layer, hence if the similar output was given as a dataset to a third-party, aggregation would be required



Content Aggregation Patterns

API Aggregation





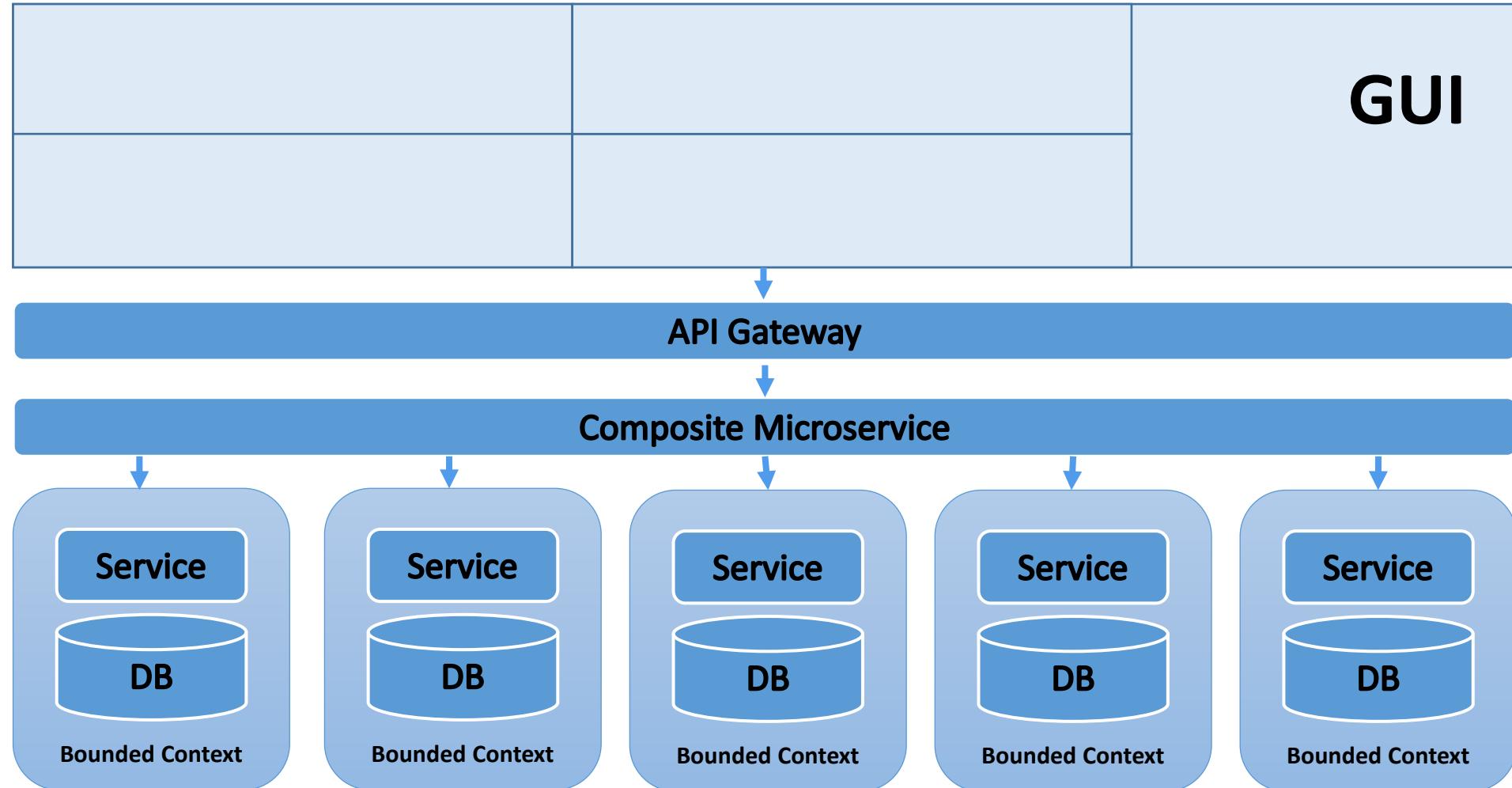
API Aggregation

- Aggregation at the gates. The API gateways are designed to do this aggregation and then expose a unified service to the client.
- Benefits
 - The individual service details are abstracted from the client by the API gateway. Hence it gives the flexibility to change the services internally without affecting the client tier.
 - Better in bandwidth constrained scenarios where running parallel HTTP requests may not be a good idea.
 - Better in UI processing constrained scenarios where processing power might not be enough for concurrent page generation.
- Tradeoffs
 - Where there is sufficient bandwidth, the latency of this option is higher than the aggregation by the client. This is because the API gateway waits for all the content to be aggregated before sending the data out to the client.



Content Aggregation Patterns

Microservice Aggregation





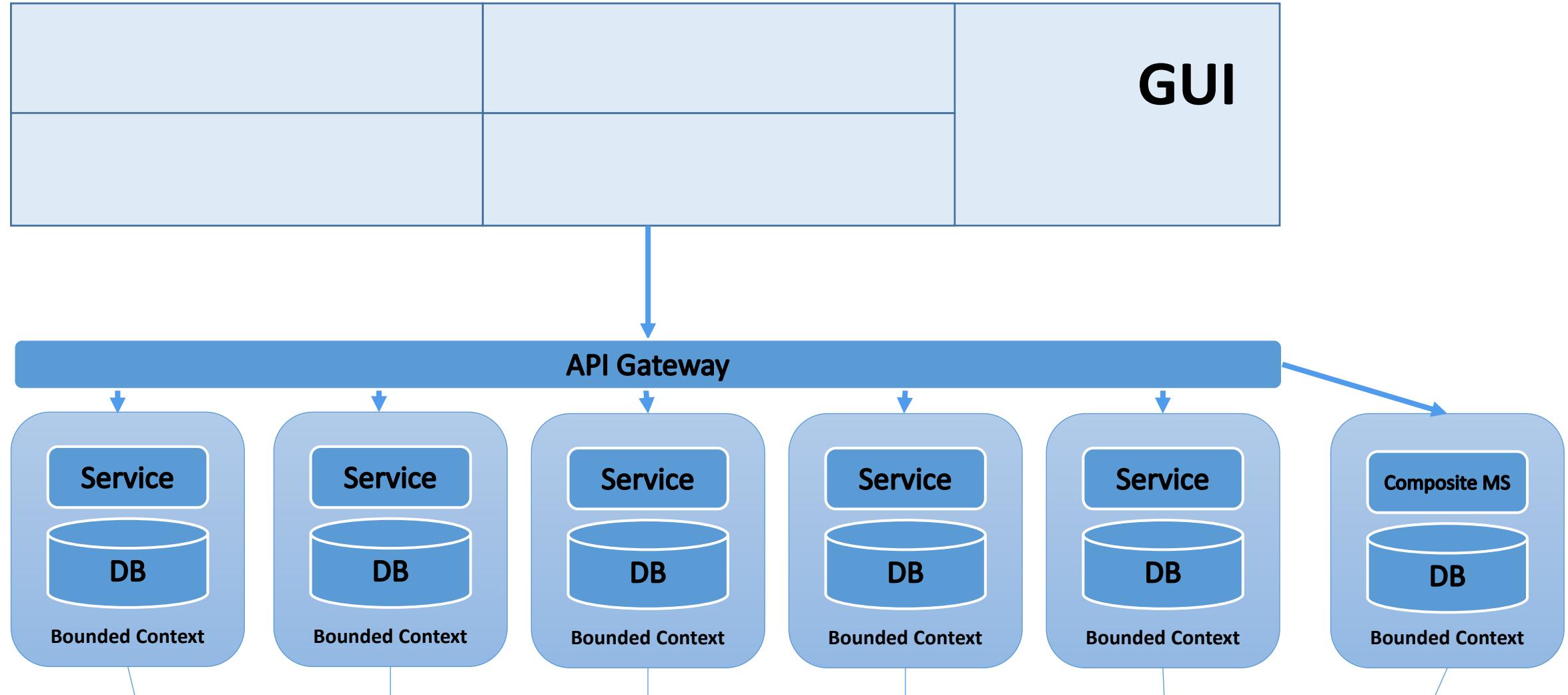
Microservice Aggregation

- Aggregation at the business tier. In this approach, a microservice aggregates the responses from the various constituent microservices. This pattern is useful if there is any real-time business logic to be applied while aggregating data.
- Benefits
 - Finer control on the aggregation. Also, there is a possibility of incorporating the business logic based on aggregated data. Thus, offering richer content aggregation capabilities.
 - Lower dependency on API gateway capabilities.
- Tradeoffs
 - Lower latency and more code, as there is an additional hop introduced due to an additional step.
 - More chances of failure or making mistakes. Parallel aggregation from microservices will need sophisticated code such as reactive or call back mechanisms.



Content Aggregation Patterns

Database Aggregation





Database Aggregation

- Aggregation at the data tier. In this approach, data is pre-aggregated into an operational data store (ODS) typically a document database.
- Benefits
 - Additional enrichment of data by analytical jobs is possible. For example, in a customer 360° view, based on the customer portfolio aggregated in the ODS, additional analytics can be applied for **next-best-action (NBA)** scenarios.
 - More flexible and capable as compared to the earlier approaches, and finer control on the data model can be exercised.
- Tradeoffs
 - Higher complexity
 - Data duplication and more data storage requirements
 - Additional ETL or **change data capture (CDC)** tools required to send the data from the system of a record to a central ODS store



Capabilities for microservices coordination - 1

▪ State management:

- The state manager component is required to manage the output state of the services that it is coordinating.
- This state will need to be held in a persistent store that is immune to **server-side state management (SSM)** failure.

▪ Transaction control:

- Transaction boundaries are affected by microservices. Two separate function calls to two methods in a single transaction now become two separate service calls through a composite service. There are two approaches to handle this scenario.

▪ Distributed transactions:

- These support the two-phase commit protocol. They are not scalable, increase latency and deadlocking scenarios, and need expensive products and infrastructure to support them. They may not be supported over selected protocols, such as REST or messaging. The benefit of this style is that the system is always in a consistent state.

▪ Compensating transactions:

- Where the transaction control is functionally enforced by running functionally reverse transactions instead of trying to roll back to an earlier transaction. This is a more decoupled, and hence scalable, approach. We would recommend compensating transactions over distributed transactions due to simplification in the technical product requirements.



Capabilities for microservices coordination – 2.

▪ Post service call coordination:

- Atomic service calls can result in success, that is, when the constituent services have finished their work successfully; or a failure, when either of the coordination services has either not responded or failed in processing due to a technical or functional error.
- The composite service will need to get the response of the completed services and decide on the next step of action.

▪ Timeout handling:

- Initiate a timer when starting a microflow. If the services do not respond in a particular time from starting the microflow, then raise an event to send to the event bus.

▪ Configurability:

- Multiple instances of the SSM component will run to cater for various micro flows.
- In each of the micro flows, the service coordination, timer, and actions will differ. Hence, it is important to provide a framework that can have parameterized configuration of the timers, compensation transactions, and post-processing actions.



SUMMARY



Cloud Native Design & Architecture Checklist-1

- **App Design & Architecture**
 - Composite, **loosely coupled** layered architecture
 - **Service** oriented architecture
 - Intuitive **user interaction** and RIA
 - Use of Implicit and Explicit **Caching**
 - **Fault Tolerant** Infrastructure
 - **Environment** Centric Design
 - **Plug and play** architecture
 - Reporting, screen mash-ups and **BI** modules

- **Tenancy Model**
 - Multi tenancy instance management
 - Level of Customization
 - Calculation of Monetization according to chosen tenancy Model
- **Scale**
 - Load Balancing
 - Distributed architecture
 - Virtualization & provisioning
 - Automated load testing and benchmarking
- **Upgrades (without Downtime)**
 - Patch Management
 - Version Updates
 - Service Upgrades



Design & Architecture Checklist-2.

▪ Data Store

- Data Redundancy and Recovery
- Data Retrieval Performance
- Data Storage Scaling
- Distributed Database
- Data Restoring and Moveable Table spaces
- Data Partitioning and Caching

▪ Security

- Data Privacy and Encryption
- Multi-Token based Authentication and Authorization
- User Tracking and Logging
- Field level security & roles provisioning
- Automated self service features such as Password Resets and Notifications
- Secure Cross domain Single Sign On
- Security Auditing and Monitoring Practices

▪ Integration

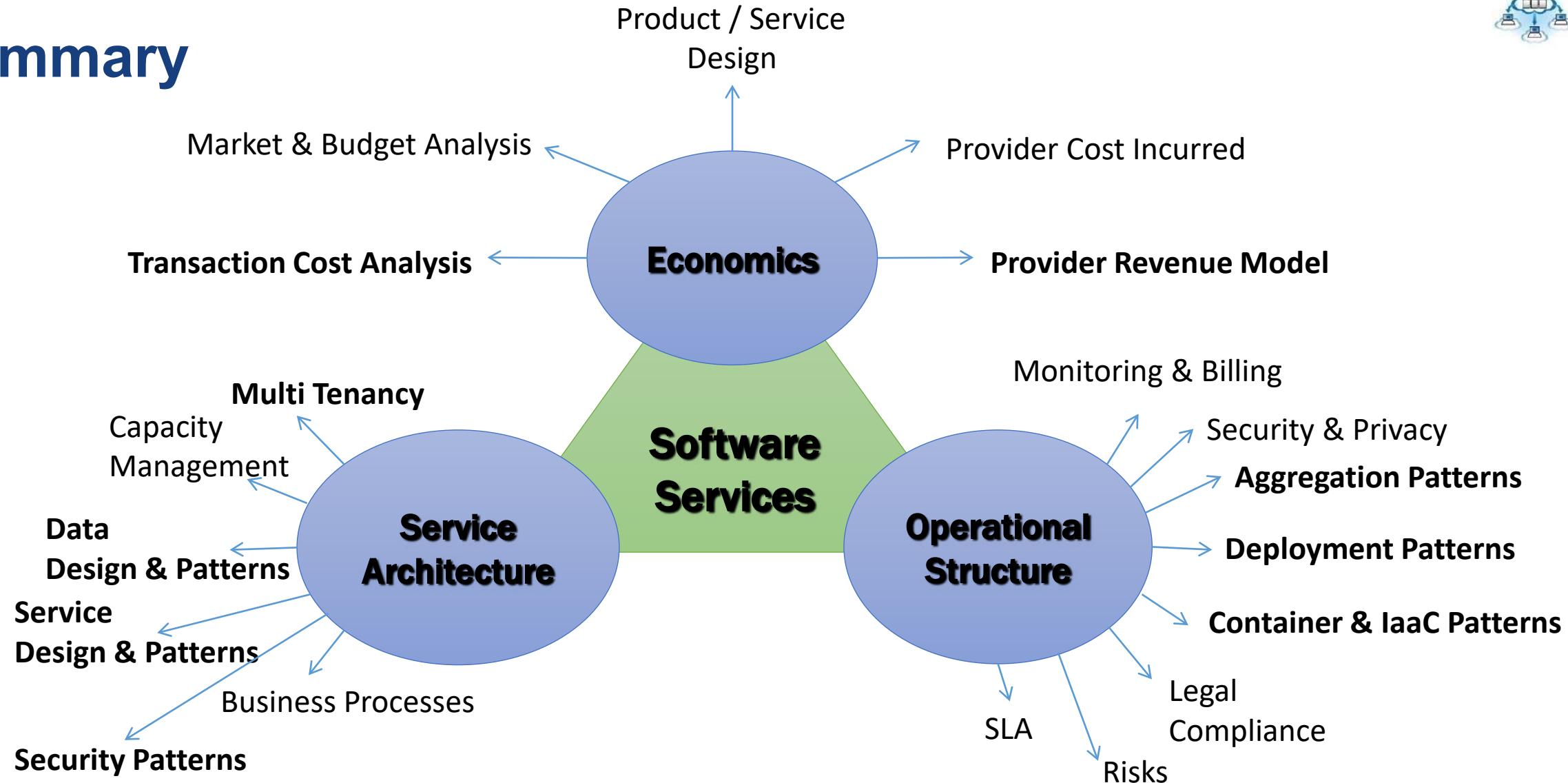
- Interaction & integration patterns
- Ability to respond in multiple formats: JSON, SOAP-XML, RSS etc
- Service-level Agreements (SLAs) and variations per tenant
- Shared APIs for Development

▪ Middleware

- Metadata for Configuration
- Application Logging
- Messaging Infrastructure, Queues
- Service Discovery, Service Directory, Services Management
- Performance Monitoring and Centralized Reporting



Summary





REFERENCE



References

■ Books

- International Journal of Systems and Service-Oriented Engineering (IJSSOE) Volume 6, Issue 1 by Dickson K.W. Chiu Published by IGI Global, 2016
- Managing Trade-offs in Adaptable Software Architectures by Rick Kazman; Ivan Mistrik; Bradley Schmerl; Nour Ali; John Grundy Published by Morgan Kaufmann, 2016
- David Sarna, Implementing and Developing Cloud Computing Applications, Auerbach, Nov 2010
- Cloud Native Patterns, by Cornelia Davis, Manning Publications, 2019
- Cloud Computing Design Patterns, First Edition, by Amin Naserpour, Robert Cope, Thomas Erl, Prentice Hall, 2015
- Serverless Design Patterns and Best Practices by Brian Zambrano *Published by Packt Publishing, 2018*

■ Web Sites, White papers and Vendor Sites

- ✓ [Blogs by Art of Scalability, Richardson, & Erl](#)
- ✓ [Microservices.io](#)
- ✓ [Architectura Blogs](#)
- ✓ [Code Ranch](#)
- ✓ [Quora](#)
- ✓ [Slack](#)



Cloud Native Solution Design

CLOUD SECURITY AN OVERVIEW

Venkat Ramanathan

rvenkat@nus.edu.sg

Institute of Systems Science

National University of Singapore

Total Slides: 53

© 2009-23 NUS. The contents contained in this document may not be reproduced in any form or by any means, without the written permission of ISS, NUS, other than for the purpose for which it has been supplied.



Agenda

- Cloud Security Breach Overview
- Security Threats
- Enterprise Security Concerns
- Cloud Security Model
- Cloud Security Strategies
- Managing Cloud Security
- Summary



SECURITY THREATS



Security Threats - Motivation

- Stealing Information for competitors
 - Example stealing Intellectual Property in Research fields
- Stealing information to make financial gains
 - Example stealing credit card or PIN information for making fraudulent transactions
- Stealing information for espionage
 - Example stealing information that is secret pertaining to military or other government domains
- Creating operational disruption
 - Example introduction of denial of service so that an organisations operations are hampered.

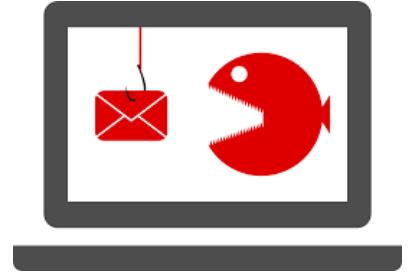




Security Threat - Classifications

- Phishing
 - Sending email impersonating authorised sender with view to obtaining sensitive information.

- Malware
 - Includes spyware, ransomware, viruses, and worms.
 - Once inside the system, Malwares can do the following:
 - Blocks access to key components of the network (ransomware)
 - Installs malware or additional harmful software
 - Covertly obtains information by transmitting data from the hard drive (spyware)
 - Disrupts certain components and renders the system inoperable.





Security Threat - Classifications

- Man-in-the-middle attack
 - Attackers insert themselves into a two-party transaction to filter and steal information.

- Denial-of-service attack
 - Attacker floods systems, servers, or networks with traffic to exhaust resources and bandwidth to cripple the servers from doing legitimate operations.

- SQL injection:
 - Attacker inserts malicious code into a server that uses SQL and forces the server to reveal information it normally would not.

- DNS Tunneling
 - DNS tunneling utilizes the DNS protocol to communicate non-DNS traffic over port 53 to exfiltrate data from a compromised system to the attacker's infrastructure.





Security Threats – Statistics

- Hackers attack every 39 seconds, on average 2,244 times a day.
- *University of Maryland*
- The average cost of a data breach is \$3.92 million as of 2019.
- *Security Intelligence*
- Only 5% of companies' folders are properly protected, on average.
- *Varonis*
- 62% of businesses experienced phishing and social engineering attacks in 2018.
- *Cybint Solutions*
- Data breaches exposed 4.1 billion records in the first half of 2019.
- *RiskBased*
- 71% of breaches were financially motivated and 25% were motivated by espionage.
- *Verizon*



Security Threats – Statistics

- 52% of breaches featured hacking, 28% involved malware and 32–33% included phishing or social engineering, respectively.
- *Verizon*
- The average time to identify a breach in 2019 was 206 days while the average lifecycle of a breach was 314 days (from the breach to containment).
- *IBM*
- In 2016, 3 billion Yahoo accounts were hacked in one of the biggest breaches of all time.
- *NY Times*
- In 2016, Uber reported that hackers stole the information of over 57 million riders and drivers.
- *Uber*



CLOUD SECURITY MODEL



Cloud Security

- What is new?

Security in the cloud is similar to security in your on-premises data centers - only without the costs of maintaining facilities and hardware. In the cloud, you don't have to manage physical servers or storage devices. Instead, you use software-based security tools to monitor and protect the flow of information into and out of your cloud resources.

- Amazon

- The concept of Shared Responsibility



Cloud security is a **Shared Responsibility** between the customer and Cloud Provider, where customers are responsible for “security in the cloud” and Cloud Provider is responsible for “security of the cloud.”



Shared Responsibility Model

- The security responsibility of the cloud consumer and cloud provider depends on the type of cloud service used.
- There are controls which both cloud consumers & providers are responsible for.
Example: while the cloud provider is responsible to ensure the reliability of the firewall service, the cloud consumer is responsible for the secure configuration.

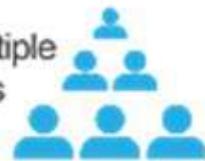
Responsibility	On-Prem	IaaS	PaaS	SaaS
Data classification & accountability	Cloud Customer	Cloud Customer	Cloud Customer	Cloud Customer
Client & end-point protection	Cloud Customer	Cloud Customer	Cloud Customer	Cloud Provider
Identity & access management	Cloud Customer	Cloud Customer	Cloud Provider	Cloud Provider
Application level controls	Cloud Customer	Cloud Customer	Cloud Provider	Cloud Provider
Network controls	Cloud Customer	Cloud Provider	Cloud Provider	Cloud Provider
Host infrastructure	Cloud Customer	Cloud Provider	Cloud Provider	Cloud Provider
Physical security	Cloud Customer	Cloud Provider	Cloud Provider	Cloud Provider

Legend: Cloud Customer (Blue), Cloud Provider (Grey)

Image credit: <https://blogs.msdn.microsoft.com/azuresecurity/2016/04/18/what-does-shared-responsibility-in-the-cloud-mean/>



Cloud Deployment Models vs Security

**VS****VS**Publicly Shared
Virtualised ResourcesSupports multiple
customersSupports connectivity
over the internetSuited for less
confidential informationPrivately Shared
Virtualised ResourcesCluster of dedicated
customersConnectivity over
internet, fibre and private networkSuited for secured
confidential information
& core systemsCombination of
Public
&
Private
Cloud



CLOUD SECURITY MANAGEMENT

**KEY ISSUES AND CASE EXAMPLES
THREAT ANALYSIS & MITIGATION**



Enterprise Concerns

- Security
- Privacy
- Ownership and control
- Trust
- Governance requirements
- Policies, regulations and legislation
- Access to skills



Top Threats Working Group
The Treacherous 12
Cloud Computing Top Threats in 2016

February 2016



CLOUD SECURITY ALLIANCE

- Not for Profit organisation
- Mission:

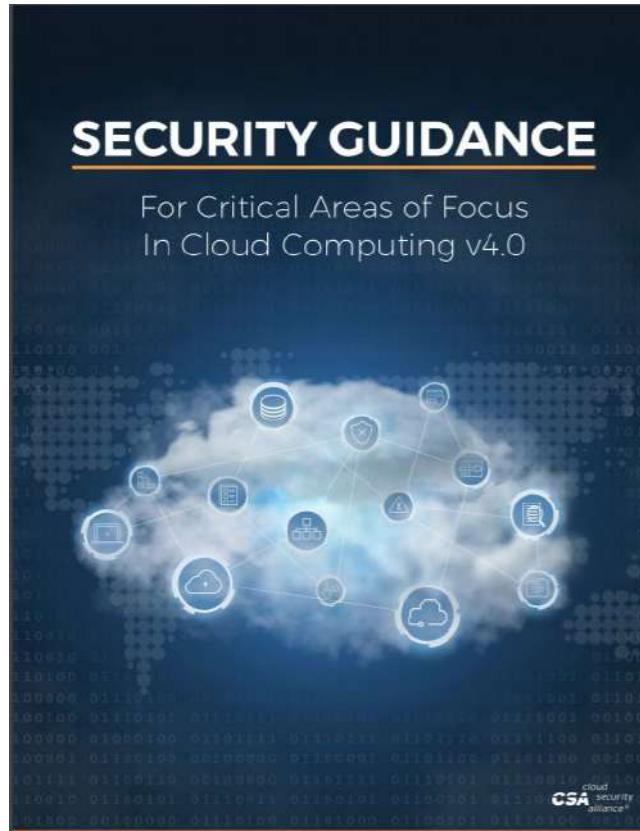
To promote the use of best practices for providing security assurance within Cloud Computing, and to provide education on the uses of Cloud Computing to help secure all other forms of computing.

- Establish standards partnering major cloud providers such as AWS, Google etc.
- Provide cloud certifications

We use CSA for this class discussion. Other standards have evolved through other consortiums such as (ISC)² etc.



Guidelines and Framework

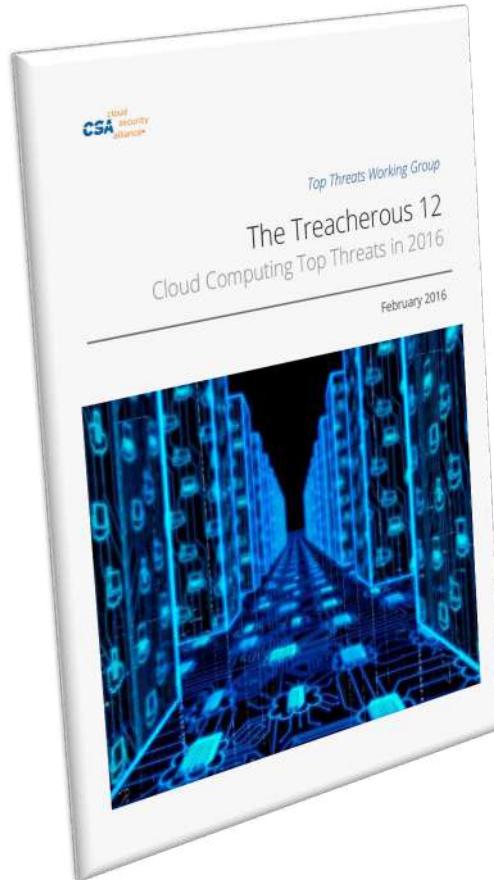


Domains:





The Treacherous Twelve



Case 1:

In mid-2015, BitDefender, an antivirus firm, had an undisclosed number of customer usernames and passwords stolen due to a security vulnerability in its public cloud application hosted on AWS. The hacker responsible demanded a ransom of \$15,000.

The Security Concern: Data Breach

THREAT ANALYSIS

STRIDE:

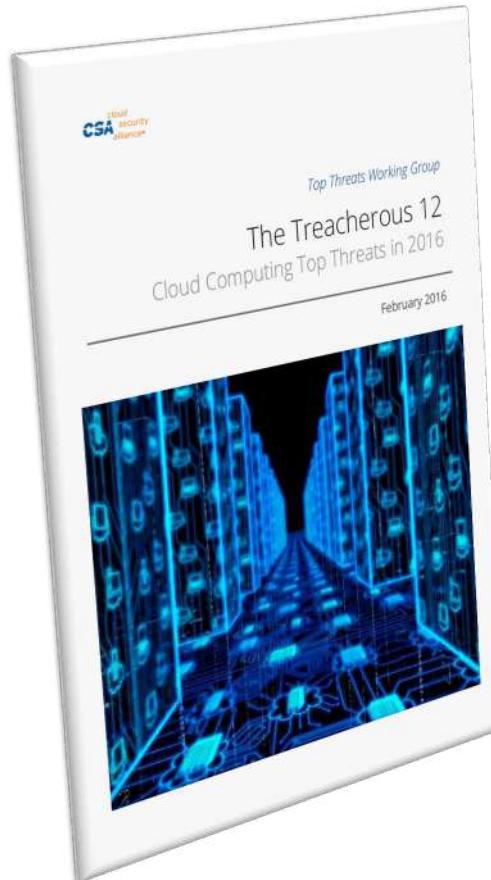
- Spoofing Identity
- Tampering with data
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege

Domain

1		6		11	<input checked="" type="checkbox"/>
2		7		12	<input checked="" type="checkbox"/>
3		8		13	
4		9		14	
5	<input checked="" type="checkbox"/>	10	<input checked="" type="checkbox"/>		



The Treacherous Twelve



Case 2:

Cloud service provider credentials included in a GitHub project were discovered and misused within 36 hours of the project going live

The Security Concern: Insufficient Identity, Credential & Access Management

THREAT ANALYSIS

STRIDE:

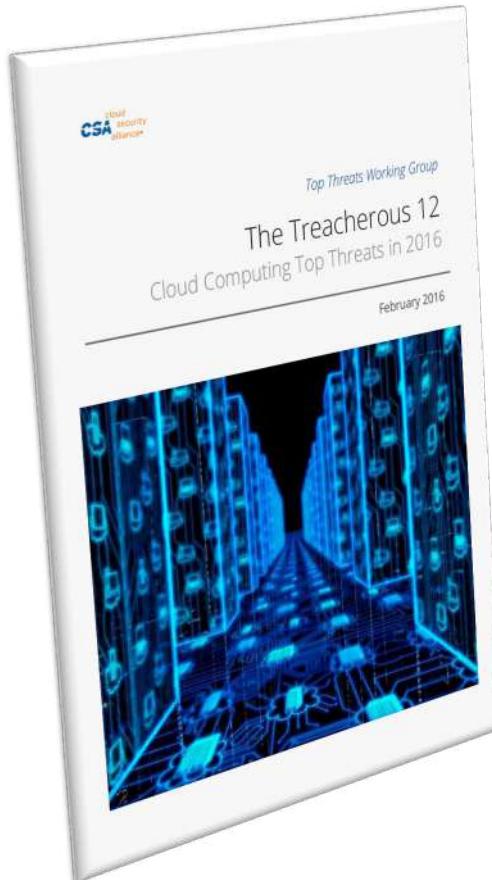
- Spoofing Identity
- Tampering with data
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege

Domain

1		6		11	<input checked="" type="checkbox"/>
2		7		12	<input checked="" type="checkbox"/>
3		8		13	
4		9		14	
5		10			



The Treacherous Twelve



Case 3:

In mid-2015, the US Internal Revenue Service (IRS) exposed over 300,000 records via a vulnerable API (“Get Transcript”)

The Security Concern: Insecure Interface and Application Program Interface

THREAT ANALYSIS

STRIDE:

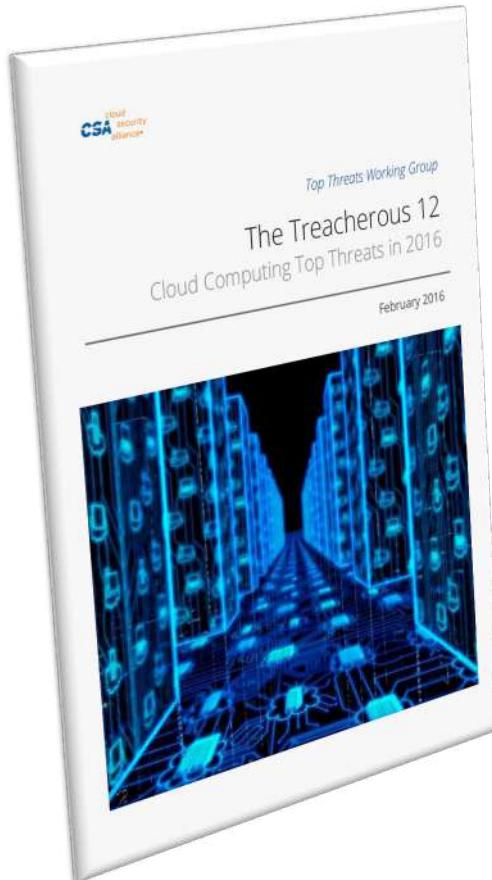
- Spoofing Identity
- Tampering with data
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege

Domain

1		6	✓	11	✓
2		7		12	✓
3		8		13	
4		9	✓	14	
5	✓	10	✓		



The Treacherous Twelve



Case 4:

Heartbleed and Shellshock proved that even open source applications, which were believed more secure than their commercial counterparts were vulnerable to threats. They particularly affected systems running Linux, which is concerning given that 67.7% of websites use UNIX, on which the former (Linux) is based.”

The Security Concern: System Vulnerabilities

THREAT ANALYSIS

STRIDE:

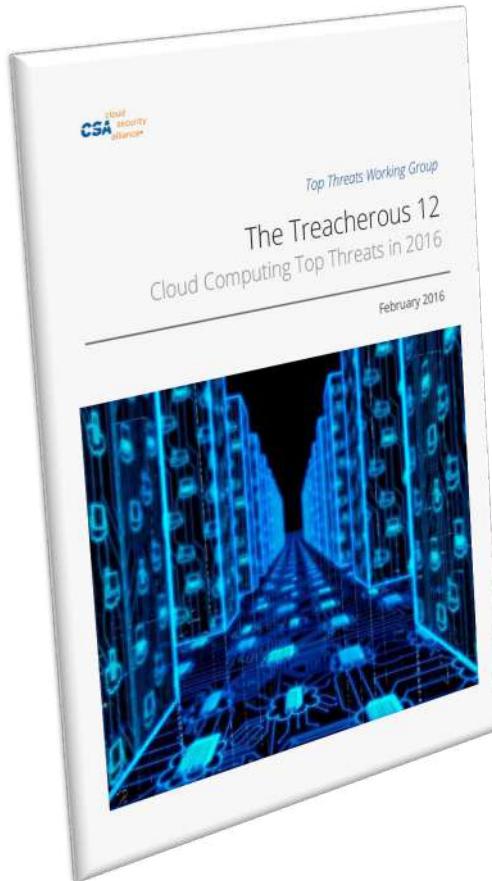
- Spoofing Identity
- Tampering with data
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege

Domain

1	✓	6		11	
2	✓	7	✓	12	
3		8	✓	13	✓
4		9		14	
5		10	✓		



The Treacherous Twelve



Case 5:

In April 2010, Amazon experienced a cross-site scripting (XSS) bug that allowed attackers to hijack credentials from the site. In 2009, numerous Amazon systems were hijacked to run Zeus botnet nodes.

The Security Concern: Account Hijacking

THREAT ANALYSIS

STRIDE:

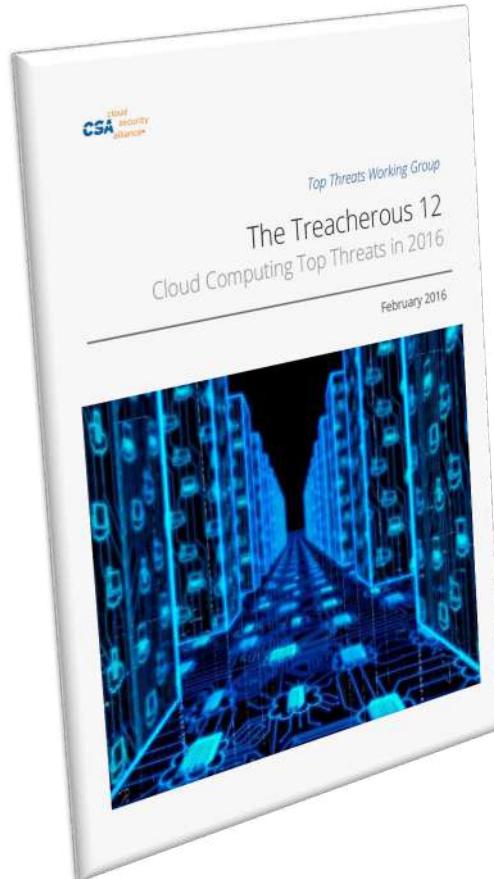
- Spoofing Identity
- Tampering with data
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege

Domain

1		6		11	<input checked="" type="checkbox"/>
2		7		12	<input checked="" type="checkbox"/>
3		8		13	
4		9	<input checked="" type="checkbox"/>	14	
5		10			



The Treacherous Twelve



Case 6:

*Cloud's Privileged Identity Gap Intensifies Insider Threats –
“Organizations need to rein in shared accounts and do a better job
tracking user activity across cloud architectures.*

The Security Concern: Malicious Insiders

THREAT ANALYSIS

STRIDE:

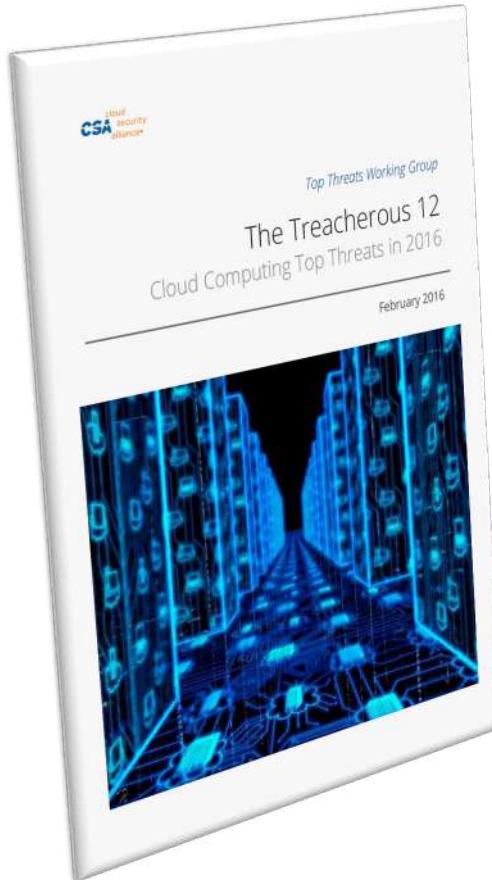
- Spoofing Identity
- Tampering with data
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege

Domain

1		6		11	✓
2	✓	7		12	✓
3		8		13	
4		9		14	
5	✓	10			



The Treacherous Twelve



Case 7:

The alleged Chinese Cyber-Espionage with its APTs caused the theft of 'hundreds of terabytes of data from at least 141 organizations across a diverse set of industries beginning as early as 2006.

The Security Concern: Advanced Persistent Threats

THREAT ANALYSIS

STRIDE:

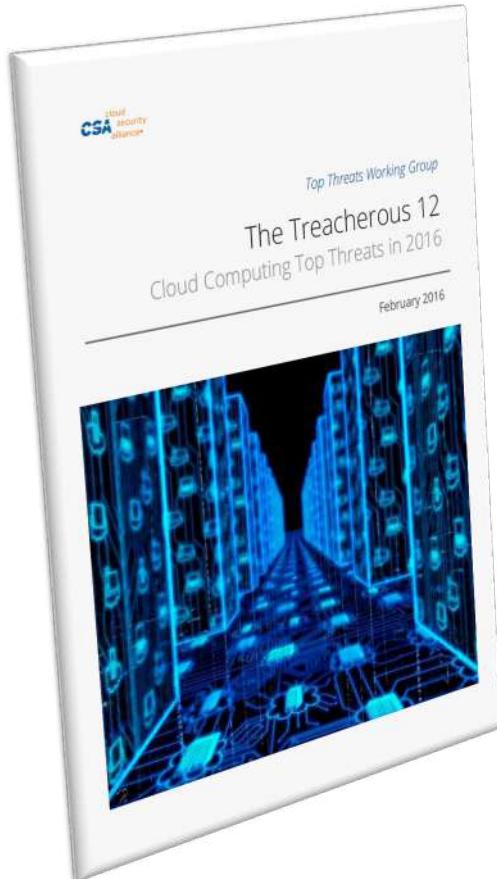
- Spoofing Identity
- Tampering with data
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege

Domain

1	✓	6		11	
2	✓	7	✓	12	
3		8	✓	13	✓
4		9		14	
5		10	✓		



The Treacherous Twelve



Case 8:

In June 2014, Code Spaces, an online hosting and code publishing provider, was hacked, leading to the compromise and complete destruction of most customer data. The company was ultimately unable to recover from this attack and went out of business.

The Security Concern: Data Loss

THREAT ANALYSIS

STRIDE:

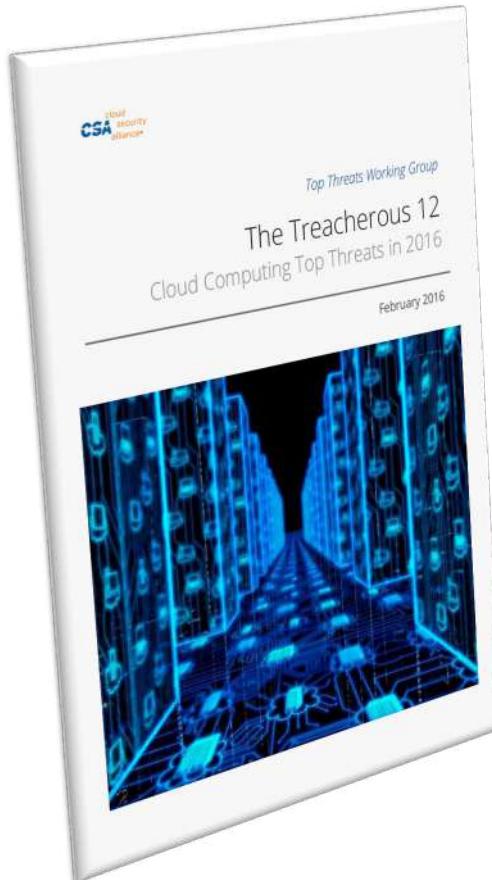
- Spoofing Identity
- Tampering with data
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege

Domain

1		6		11	
2		7		12	✓
3		8		13	✓
4		9		14	
5	✓	10	✓		



The Treacherous Twelve



Case 9:

In 2013, Nirvanix, a cloud storage specialist that hosted data for IBM, Dell and its own customers, filed for Chapter 11 bankruptcy and shuttered its operations. Customers were given less than two weeks to move their data to another service.

The Security Concern: Insufficient Due Diligence

THREAT ANALYSIS

STRIDE:

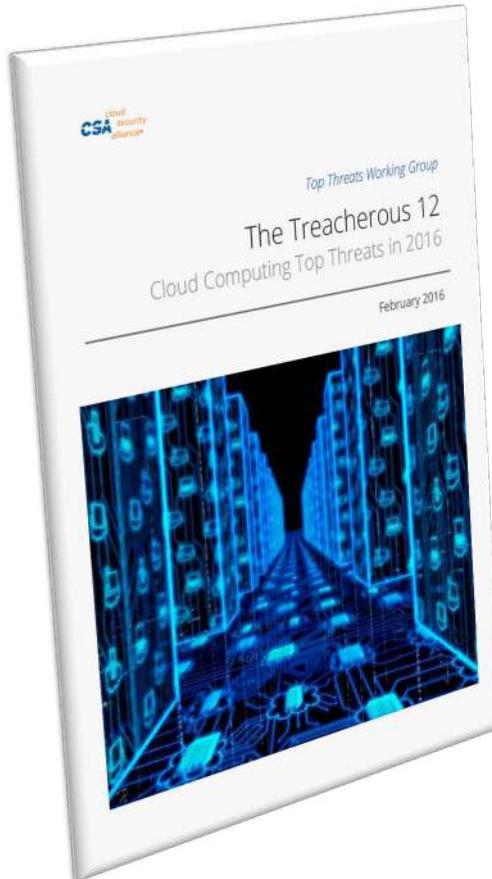
- Spoofing Identity
- Tampering with data
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege

Domain

1	✓	6	✓	11	✓
2	✓	7	✓	12	✓
3	✓	8	✓	13	✓
4	✓	9	✓	14	✓
5	✓	10	✓		



The Treacherous Twelve



Case 10:

Hackers prefer outsourcing to companies like Amazon due to their gargantuan size and relative inability to vet anyone who might want to use the cloud provider for more malicious reasons than most. Their organizations can slip in unnoticed, set up shop, and run DDoS attacks for upwards of two weeks before Amazon catches on, by which time the ring has already made their money back in spades and ditched all the account info that was used to launch the campaign in the first place.

The Security Concern: Abuse and Nefarious Use of Cloud Services

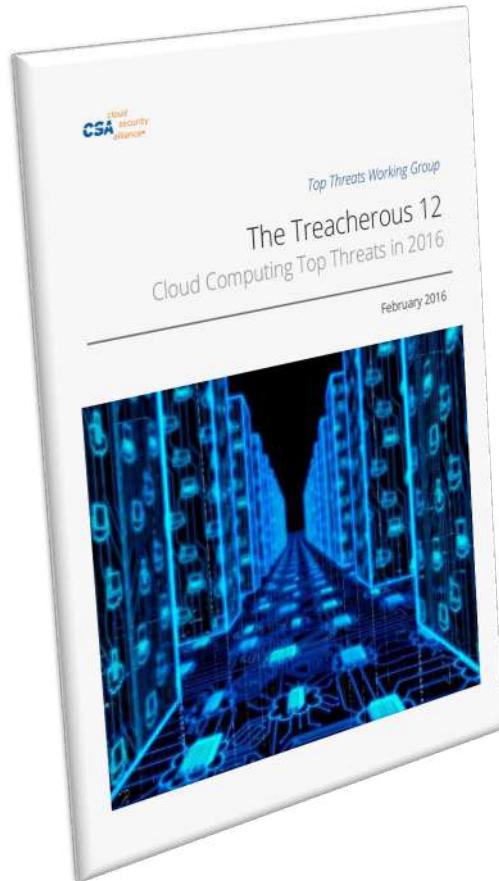
THREAT ANALYSIS					
STRIDE:					
<input type="checkbox"/> Spoofing Identity					
<input type="checkbox"/> Tampering with data					
<input type="checkbox"/> Repudiation					
<input type="checkbox"/> Information Disclosure					
<input checked="" type="checkbox"/> Denial of Service					
<input type="checkbox"/> Elevation of Privilege					

Domain

1		6		11	
2		7	✓	12	
3	✓	8		13	
4		9	✓	14	
5		10			



The Treacherous Twelve



Case 11:

As Cloud Use Grows, So Will Rate of DDoS Attacks – “Cloud providers face increasing number of DDoS attacks, [similar to those that] private data centers already deal with today”

The Security Concern: Denial of Service

THREAT ANALYSIS

STRIDE:

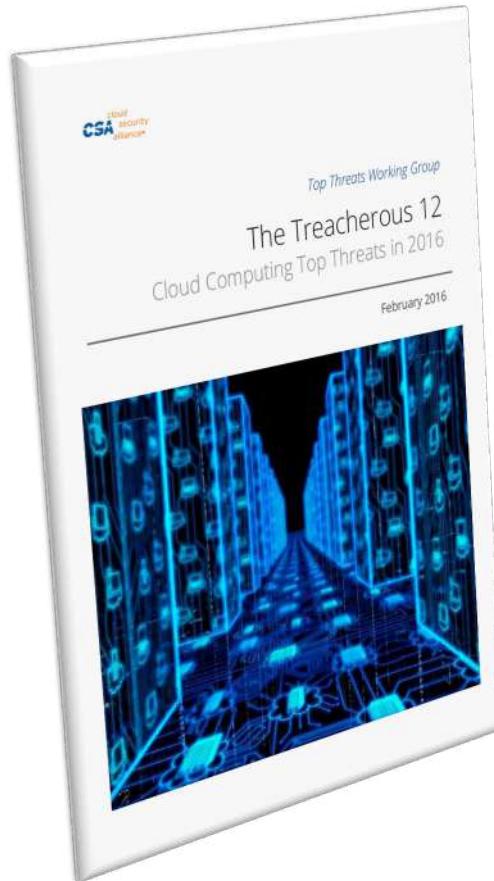
- Spoofing Identity
- Tampering with data
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege

Domain

1		6		11	
2		7		12	
3		8	✓	13	✓
4		9	✓	14	✓
5		10	✓		



The Treacherous Twelve



Case 12:

Understanding the VENOM Vulnerability – “The unchecked buffer vulnerability (CVE-2015-3456) occurs in the code for QEMU’s virtual floppy disk controller. A successful buffer overflow attack exploiting this vulnerability can enable an attacker to execute his or her code in the hypervisor’s security context and escape from the guest operating system to gain control over the entire host.

The Security Concern: Shared Technology Vulnerabilities

THREAT ANALYSIS

STRIDE:

- Spoofing Identity
- Tampering with data
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege

Domain

1	✓	6		11	✓
2		7		12	✓
3		8		13	✓
4		9		14	
5	✓	10			



CLOUD SECURITY TECHNICAL ASPECTS

KEY TECHNOLOGIES, TECHNIQUES AND ARCHITECTURES



Classifying Security Components in Cloud

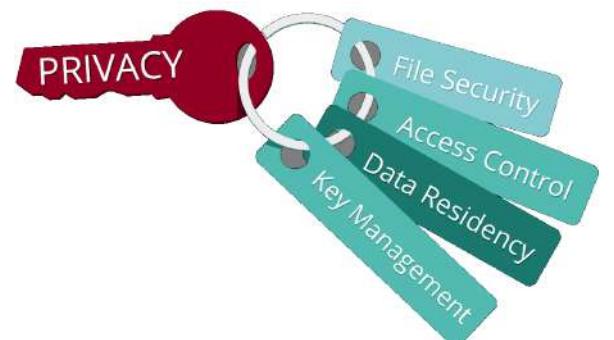
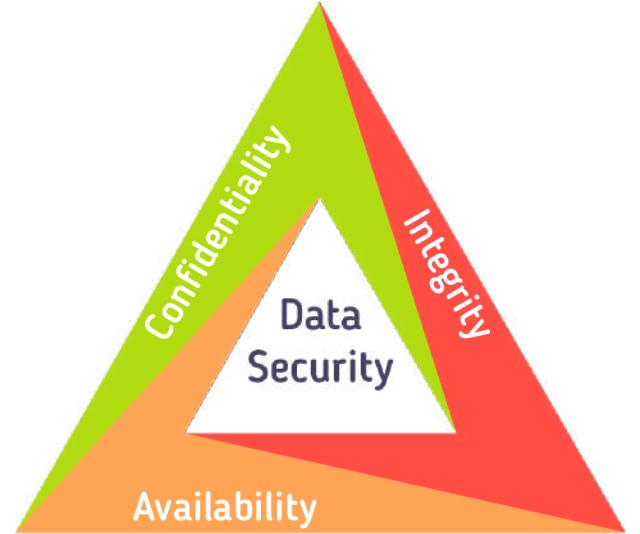
- A typical Cloud Native Application





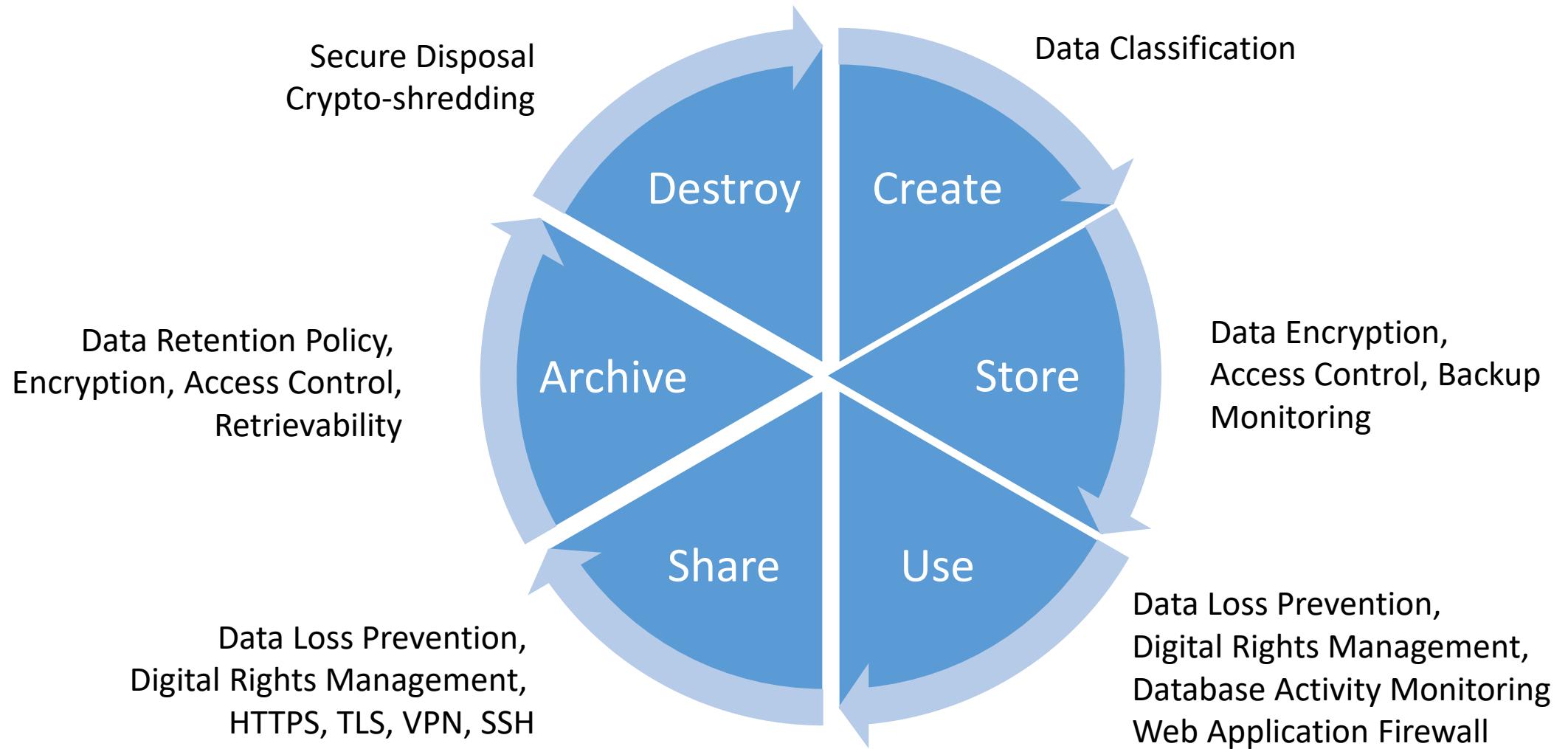
Data Security in Cloud

- Type of Data
 - Is it Confidential Data?
 - Is it Public Data?
 - Is it Sensitive Data?
- Location of the Data
 - Data at rest (*databases*)
 - Data in motion (*over the network*)
 - Data in memory (*data used by apps in memory*)
- Securing the Data
 - Encrypting stored and transmitted data
 - Key Management (BYOK, KYOK)





Principle of Data Security





Data Encryption Challenges

- Who holds the encryption key
 - If cloud provider hold the key, not all data threats are answered
 - If cloud consumer hold the key, cloud consumer is responsible for key protection
- Cloud provider may not be able to process encrypted data
- RAM may be vulnerable due to some weaknesses
- Data can change location, formats => encryption and encryption keys



Typical Data Encryption in Cloud

- Network transfer
 - TLS for HTTPS
 - VPN
- Volume Storage
 - Individual File/Folder encryption
 - Volume Storage Encryption
- Object Storage Encryption
- Database Encryption
 - Transparent Database Encryption
 - Application Level Encryption



Data Masking

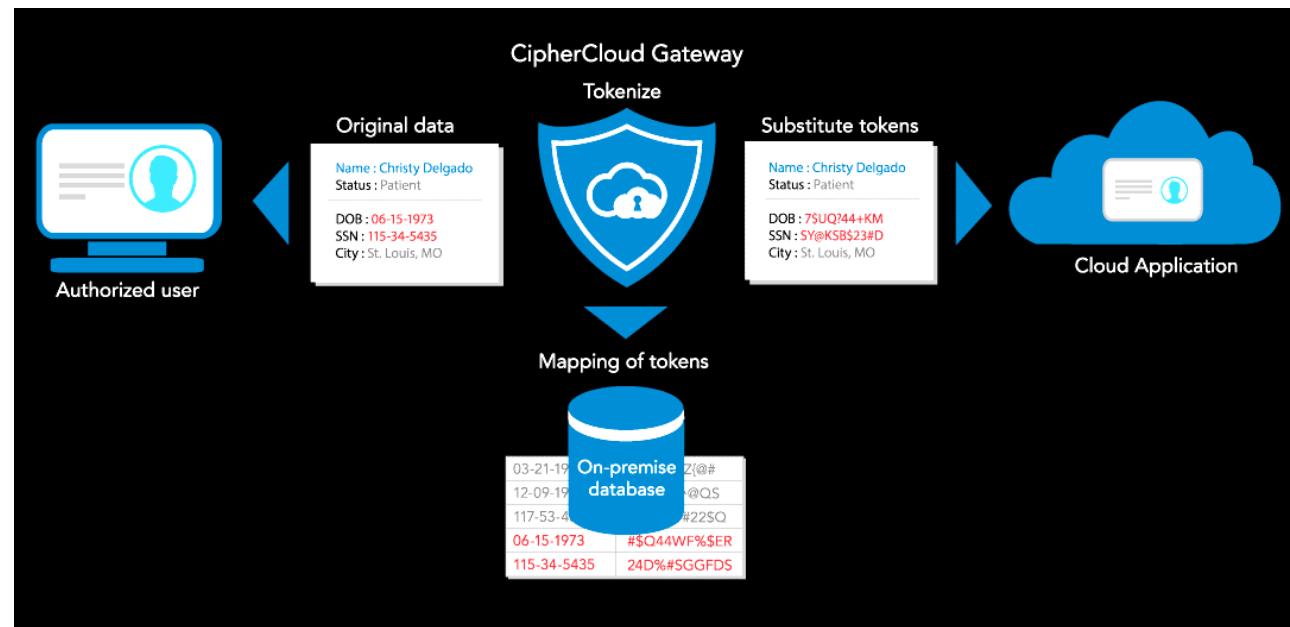
- Hide the original data with modified content
- Protect data that is classified as sensitive information, but the data still need to be usable





Tokenization

- The process of substituting a sensitive data element with a nonsensitive equivalent, referred to as a token.
- Can assist with:
 - Complying with regulations or laws.
 - Reducing the cost of compliance.
 - Mitigating risks of storing sensitive data and reducing attack vectors on that data.





Application Security in Cloud

- Type of Apps
 - Is it a Mission Critical App?
 - Is it a Infrastructure App?
 - Is it a Business Function App?

- Securing the App
 - Scan your Applications
 - Dev Sec Ops Approach
 - Scan the Containers

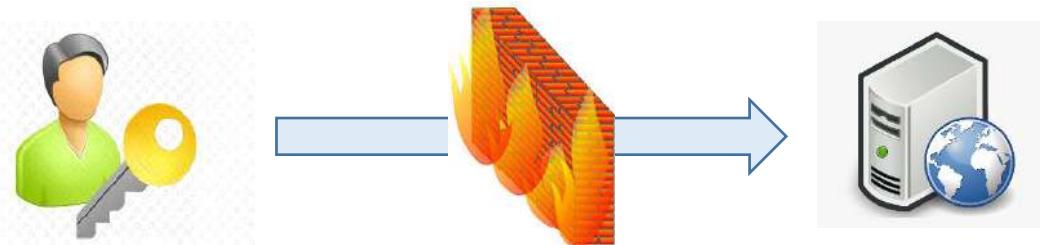




User Security in Cloud

- Types of Users
 - Is she a public user?
 - Is he a registered user?
 - Are they Employees/Developers/Administrators?

- User Access Control
 - Resource Protection
 - Control Access to Applications, Servers & Data.
 - Where are they coming from?
 - Network Protection
 - Web Application Firewalling.
 - Network Access Controls.
 - Distributed Denial of Service Protection.





User Access Control - Network & Hardware Level Security

- Create isolated subnets for network zoning
- Use security groups to manage firewall rules to access the VM instances
- Use host-based firewalls for additional defence to each instances
- Use VPN-based connection to create direct connection between internal network to a particular subnet in the cloud
- Web Application Firewall protects from
 - DDoS protection
 - Common web exploits
- Hardware Security Module
 - Stronger cryptography capability
 - Tamper-proof



Access Control

- Access control decision should be made based on:
 - Actor
 - The identity of the user making the action
 - Function
 - The action that the actor is trying to make
 - Location
 - The location of the actor – which may include
 - ✓ Network location : in office vs. from home vs. VPN
 - ✓ Machine used to access: own machine vs. company issued laptop vs. company desktop
 - ✓ Country : data export restriction
 - Other attributes
 - Group membership of the actor
 - Time of day

Enforce Access Management such as Key Management, Federated Identity & IAM



CLOUD SECURITY STRATEGIES



Strategies for managing risks

- What are the pain points?
 - Security? Data privacy? Costs? *These must be understood & then managed.*
- Strategies
 - Start small: reduce risk exposure
 - Proof-of-concept – answer feasibility question
 - Pilot – answer adoption and scalability questions
 - Protect your data: avoid critical risk
 - Start with non-critical data and non-personal data
 - Ensure data portability and regular backups can be done
 - Explore in a safe environment
 - Start with small scale adoption
 - Use it in non-critical areas, e.g., testing and development
 - Sanctioned environment: pass the risk on
 - Use a policy-endorsed Cloud provider

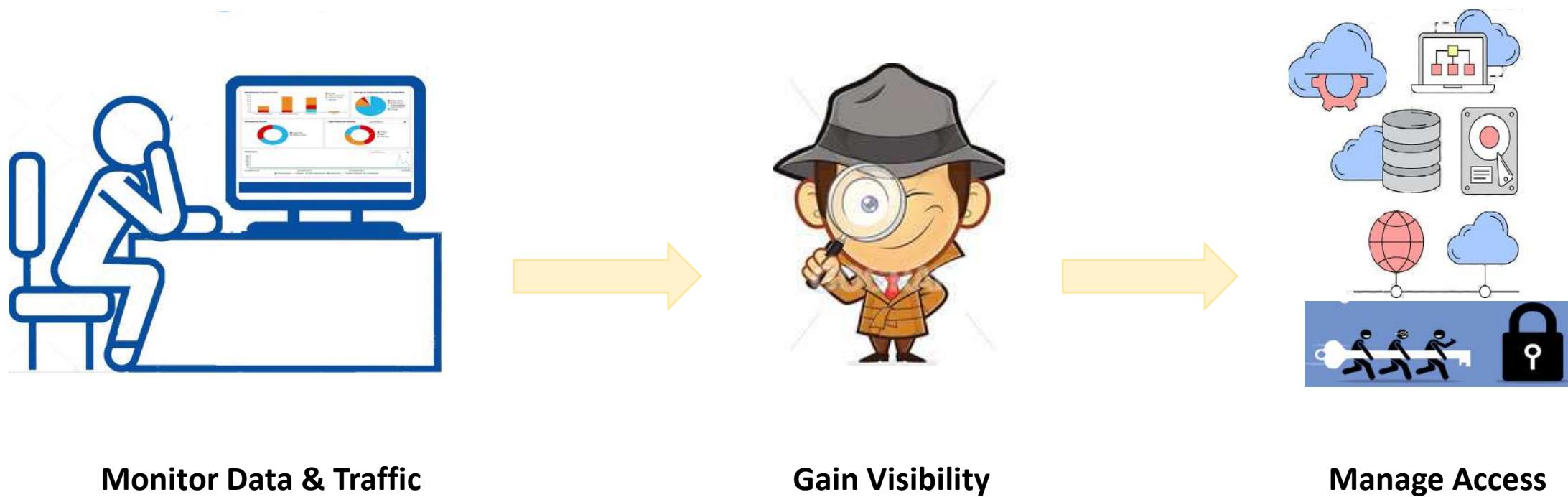




MANAGING CLOUD SECURITY



Threat Monitoring, Identification & Management





Monitoring Data

- Use tools provided by the Cloud Provider
- Tools have a Web Interface
 - Provides intelligent algorithms to track exceptions.
 - Provides scope for alarms to be set
 - Provides dashboard for visual review
- Example AWS Cloud Watch
 - Monitors EC2 and other AWS resources
 - Able to set custom metrics
 - Stores Logs
 - Alarm setting facility
 - Dashboard for Graphs & Statistics
 - Reacts to changes in Resources.





AWS Services

▼ All services



Compute

- EC2
- EC2 Container Service
- Lightsail
- Elastic Beanstalk
- Lambda
- Batch



Developer Tools

- CodeStar
- CodeCommit
- CodeBuild
- CodeDeploy
- CodePipeline
- X-Ray



Internet of Things

- AWS IoT
- AWS Greengrass



Contact Center

- Amazon Connect



Storage

- S3
- EFS
- Glacier
- Storage Gateway



Management Tools

- CloudWatch
- CloudFormation
- CloudTrail
- Config
- OpsWorks
- Service Catalog
- Trusted Advisor
- Managed Services



Game Development

- Amazon GameLift



Database

- RDS
- DynamoDB
- ElastiCache
- Amazon Redshift



Security, Identity & Compliance



Mobile Services

- Mobile Hub
- Cognito
- Device Farm
- Mobile Analytics
- Pinpoint

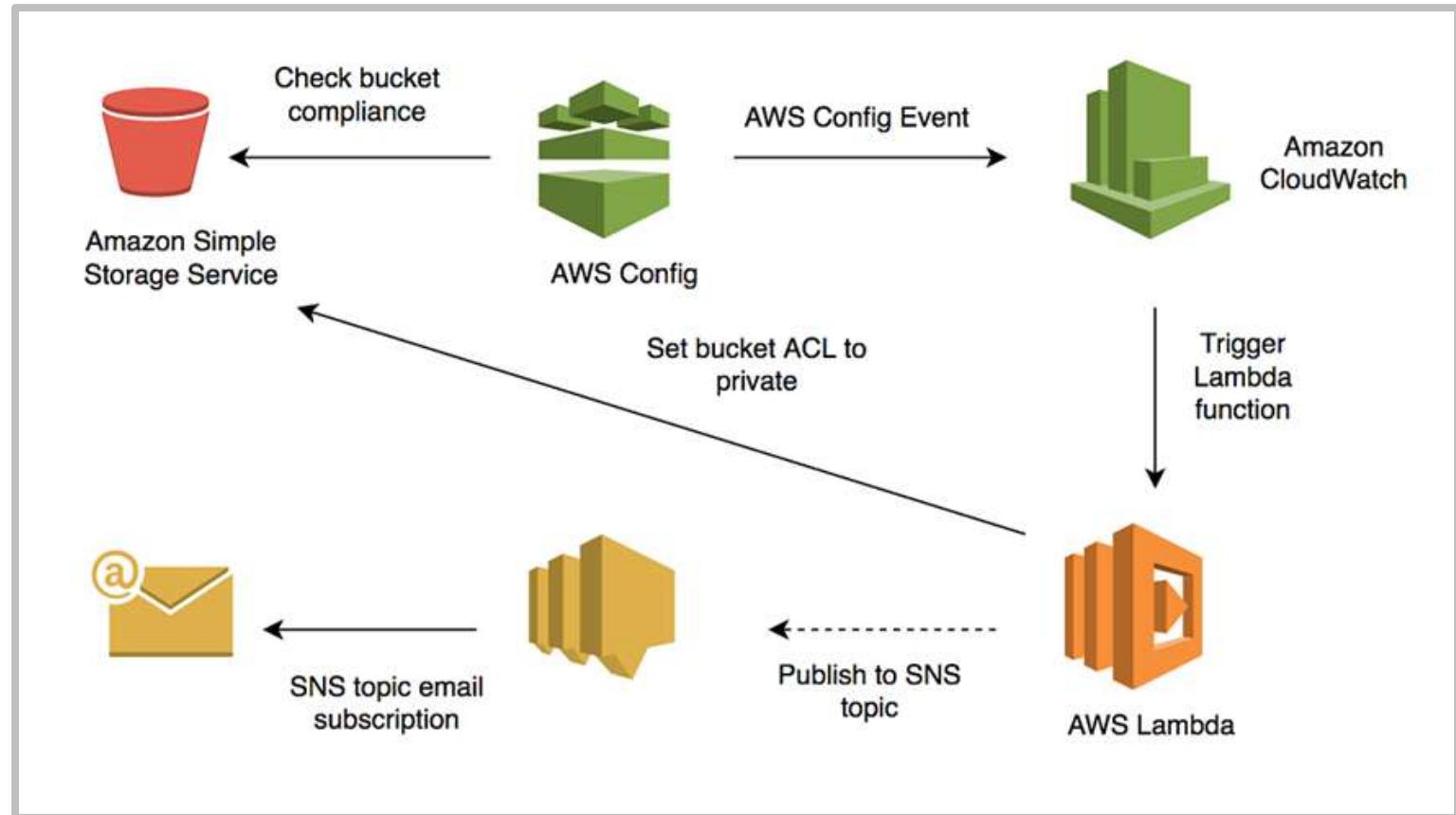


Application Services

- Step Functions



Monitoring Events - Flow



Source:

<https://aws.amazon.com/blogs/security/how-to-use-aws-config-to-monitor-for-and-respond-to-amazon-s3-buckets-allowing-public-access/>



Gaining Visibility

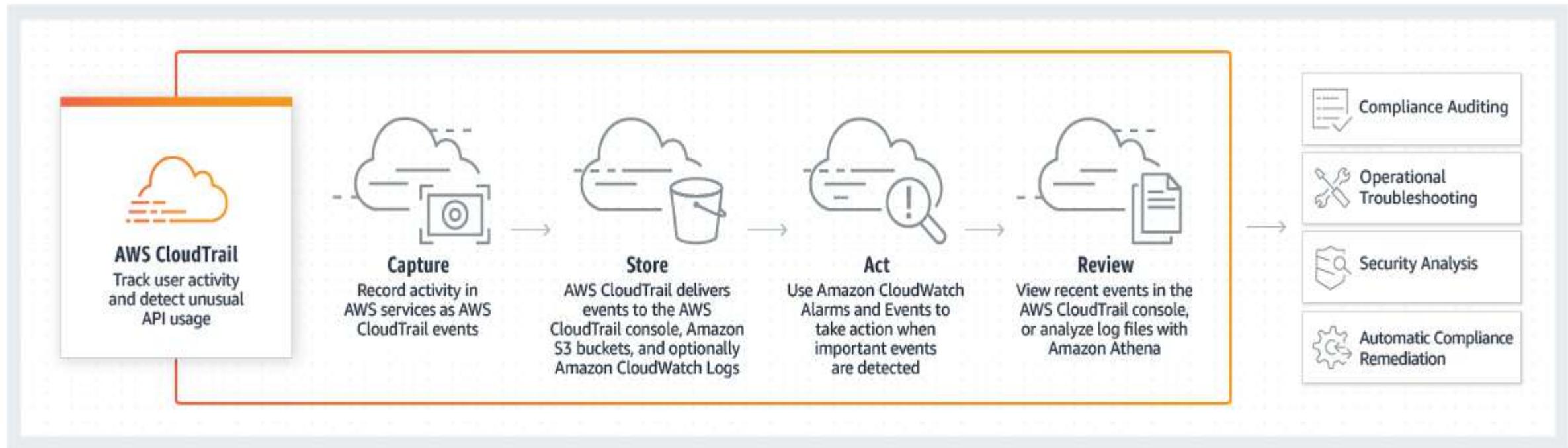
- Tries to address the two main questions
 - What cloud services and resources are we using?
 - Are those services and resources secure?
- Cloud Provider Tools Provide the following:
 - Facilitates creation of logs and audit trails
 - Provides identity of user that requested a service with which you may able to zero in on the “hacker” and his attributes to tighten Security
- AWS Cloud Trail

AWS CloudTrail is an AWS service that helps you enable governance, compliance, and operational and risk auditing of your AWS account. Actions taken by a user, role, or an AWS service are recorded as events in CloudTrail. Events include actions taken in the AWS Management Console, AWS Command Line Interface, and AWS SDKs and APIs.





AWS Cloud Trail



Source: <https://aws.amazon.com/cloudtrail/>



Managing Access

■ Identity Access Management

Identity and Access Management (IAM) is the security discipline that enables the right individuals to access the right resources at the right times for the right reasons. IAM addresses the mission-critical need to ensure appropriate access to resources across increasingly heterogeneous technology environments

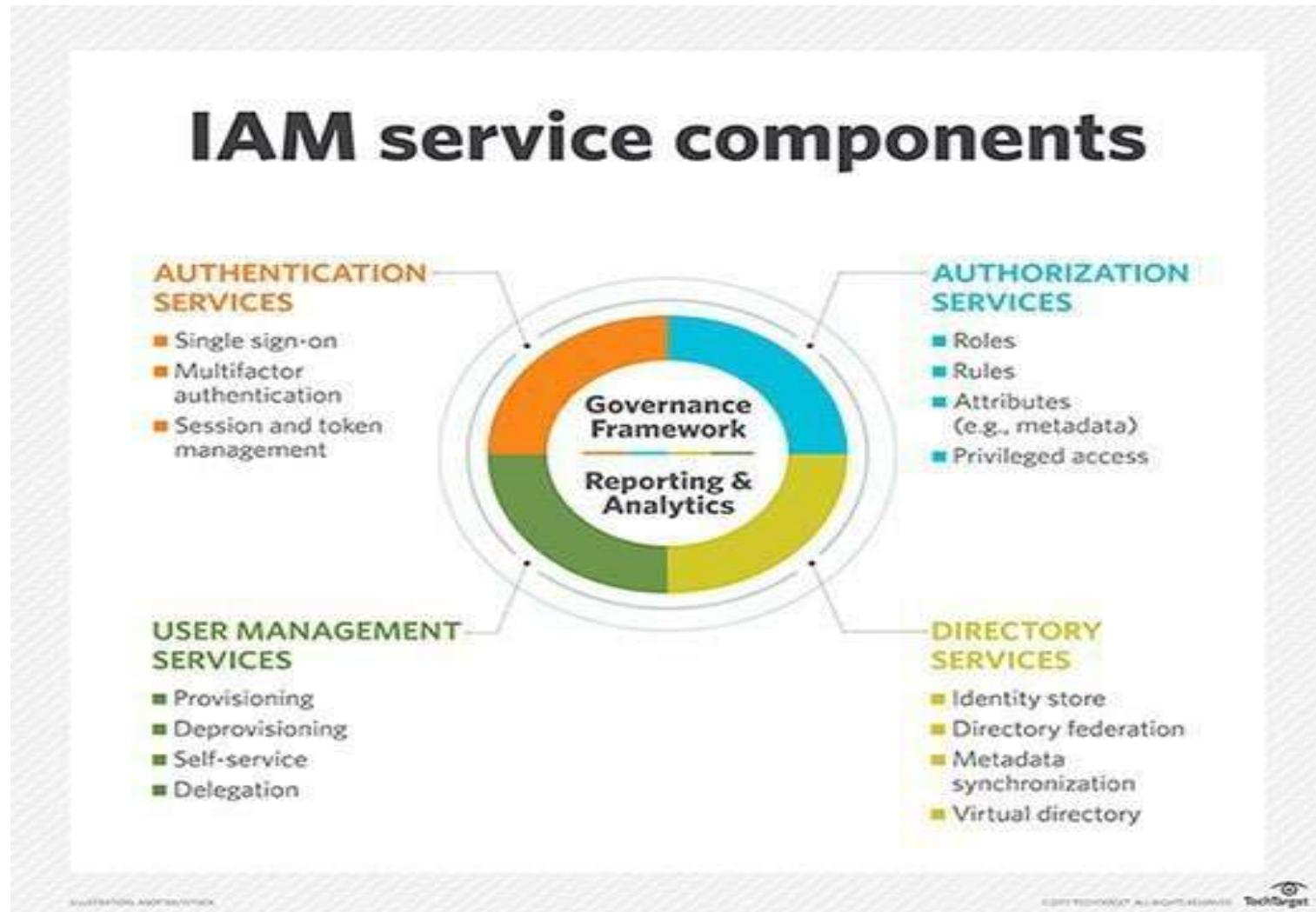
■ Key Functions

- Manage user identities
- Provisioning/deprovisioning users
- Authenticating users
- Authorizing users
- Reporting
- Single Sign On





Identity and Access Management





Typical IAM related cloud services

- Management of privileged users
 - Manage people who can access and manage cloud services or cloud resources
 - Example: AWS IAM
- Manage users and/or roles for applications deployed on the cloud
 - Example: AWS Cognito, Active Directory
- Broker/proxy to a third party authentication provider for federated identity
 - Example: API gateway



Concluding Remarks

- SLA provides the contract between cloud consumers and cloud providers
- Shared responsibility model means that responsibility varies depends on the cloud services used
- Cloud providers provide various security services from which covers network security, security devices, identity management, authentication/authorization and encryption services
- In cloud computing environment, the security focus shift from infrastructure security to data security
- Data should be secured in transit, at rest and in use
- Access control should follow classification and enforced in all lifecycle of data



CLOUD NATIVE SOLUTION DESIGN

SEPTEMBER 2023

Suria (suria@nus.edu.sg)

OVER
6,873

GRADUATE
PROGRAMME
ALUMNI

OFFERING OVER
150 ENTERPRISE IT, INNOVATION
& LEADERSHIP PROGRAMMES

TRAINING OVER
157,000 DIGITAL LEADERS
& PROFESSIONALS

Other courses for your consideration



Digital Products & Platforms

- Architecting Platforms as a Business #
- Certified Scrum Product Owner
- NICF - Certified ScrumMaster (SF) #
- NICF - Digital Product Strategy (SF) #
- NICF - Managing Business Analytics Projects (SF) #
- NICF - Managing Digital Products (SF) #
- NICF - Product Thinking for Organisations (SF)
- NICF - Strategic Product Market Fit (SF)
- PCP for Digital Product Manager*



Digital Agility

- Certified Scrum Product Owner
- NICF - Agile Testing (SF)
- NICF - Business Agility Bootcamp (SF)
- NICF - Business Analysis for Agile Practitioners (SF)
- NICF - Coaching for Agile Teams (SF)
- NICF - DevOps Foundation with BizOps (SF)
- NICF - Essential Practices for Agile Teams (SF)
- NICF - Site Reliability Engineering - Processes and Management (SF)
- NICF - Systems Thinking & Root Cause Analysis (SF)



StackUp – Startup Tech Talent Development

- NICF - Autonomous Decision Making With Reinforcement Learning (SF)
- NICF - Containers for Deploying and Scaling Apps (SF)
- NICF - Data and Feature Engineering for Machine Learning (SF)
- NICF - Feature Extraction and Supervised Modeling with Deep Learning (SF)
- NICF - Python for Data, Ops and Things (SF)
- NICF - RESTful APIs Design (SF)
- NICF - Sequence Modeling with Deep Learning (SF)
- NICF - Supervised and Unsupervised Modeling with Machine Learning (SF)
- PCP for FullStack Software Developers *



Cybersecurity

- AI and Cybersecurity
- NICF - (ISC)² CCSP CBK Training Seminar (SF)
- NICF - (ISC)² CISSP CBK Training Seminar (SF) #
- NICF - (ISC)² CSSLP CBK Training Seminar (SF)
- NICF - (ISC)² SSPC CBK Training Seminar (SF)
- NICF - (ISC)² AISCP Qualified Information Security Professional Course (SF)
- NICF - Cyber Security for ICT Professionals (SF)
- NICF - Cybersecurity Risk Awareness (SF)
- NICF - Design Secure Mobile Architecture (SF) #
- NICF - Developing Cybersecurity Architecture (SF)
- NICF - Managing Cybersecurity Risk (SF)
- NICF - Platform Security (SF) #
- NICF - Secure Software Development Lifecycle for Agile (SF) #
- NICF - Securing IoT (SF)



Software Systems

- Architecting Platforms as a Business #
- NICF - (ISC)² CISSP CBK Training Seminar (SF) #
- NICF - Agile Testing (SF)
- NICF - Architecting IOT Solutions (SF) #
- NICF - Architecting Software Solutions (SF) #
- NICF - Architecting Systems For Real Time Data Processing (SF) #
- NICF - Autonomous Decision Making With Reinforcement Learning (SF)
- NICF - Big Data Engineering for Analytics (SF) #
- NICF - Cloud Native Solution Design (SF) #
- NICF - Design Secure Mobile Architecture (SF) #
- NICF - Designing Intelligent Edge Computing (SF) #
- NICF - DevOps Engineering and Automation (SF) #
- NICF - Digital Product Strategy (SF) #
- NICF - Envisioning Smart Urban IoT Solutions (SF)
- NICF - Essential Practices for Agile Teams (SF)
- NICF - Humanizing Smart Systems (SF) #
- NICF - Information Architecture for Data-Driven Insights (SF) #
- NICF - Object Oriented Analysis & Design (SF)
- NICF - Object Oriented Design Patterns (SF)
- NICF - Platform Engineering (SF) #
- NICF - Platform Security (SF) #
- NICF - Secure Software Development Lifecycle for Agile (SF) #
- NICF - Securing IoT (SF)
- NICF - Service Design (SF) #
- NICF - Strategic Product Manager™ (SF) #
- NUS Certificate in Digital Solutions Development – Design #
- NUS Certificate in Digital Solutions Development – Foundations #
- NUS Certificate in Digital Solutions Development - Mobility Applications #
- NUS Certificate in Digital Solutions Development – Web Application #
- NUS Graduate Diploma in Systems Analysis #
- NUS Graduate Diploma in Systems Analysis - Capstone & Internship #
- NUS-ISS Stackable Certificate Programmes in Digital Solutions Development #
- PCP for Solutions Architect*



Data Science

- NICF - Advanced Customer Analytics (SF) #
- NICF - Big Data Engineering for Analytics (SF) #
- NICF - Campaign Analytics (SF) #
- NICF - Conversational UIs (SF) #
- NICF - Complex Predictive Modelling & Forecasting (SF) #
- NICF - Customer Analytics (SF) #
- NICF - Data Analytics Process and Best Practice (SF) #
- NICF - Data Driven Decision Making (SF)
- NICF - Data Governance & Protection (SF) #
- NICF - Data Storytelling (SF) #
- NICF - Feature Engineering and Analytics using IOT Data (SF) #
- NICF - Graph and Web Mining (SF) #
- NICF - Health Analytics (SF) #
- NICF - Managing Business Analytics Projects (SF) #
- NICF - New Media and Sentiment Mining (SF) #
- NICF - Predictive Analytics - Insights of Trends and Irregularities (SF) #
- NICF - Recommender Systems (SF) #
- NICF - Service Analytics (SF) #
- NICF - Social Media Analytics
- NICF - Statistics for Business II (SF) #
- NICF - Statistics Bootcamp (SF) #
- NICF - Text Analytics (SF) #
- NICF - Text Processing using Machine Learning (SF) #
- NICF - Web Analytics & SEO (SF)
- PCP for Data Analysts*
- PCP for Business Analytics Programme Analyst*



Digital Strategy & Leadership

- AI and Cybersecurity
- Certified Scrum Product Owner
- e-Government Leadership
- NICF - AISCP Qualified Information Security Professional Course (SF)
- NICF - (ISC)² CCSP CBK Training Seminar (SF)
- NICF - (ISC)² CISSP CBK Training Seminar (SF) #
- NICF - (ISC)² CSSLP CBK Training Seminar (SF)
- NICF - (ISC)² SSPC CBK Training Seminar (SF)
- NICF - Business Agility Bootcamp (SF)
- NICF - Business Analysis for Agile Practitioners (SF)
- NICF - Business Process Reengineering (SF)
- NICF - Certified Enterprise Architecture Practitioner Course (SF)
- NICF - Cyber Security for ICT Professionals (SF)
- NICF - Cybersecurity Risk Awareness (SF)
- NICF - Data Driven Decision Making (SF) #
- NICF - Data Governance & Protection (SF) #
- NICF - Design Secure Mobile Architecture (SF) #
- NICF - Developing Cybersecurity Architecture (SF)
- NICF - Digital Business Analysis (SF)
- NICF - Digital Transformation Planning (SF)



Artificial Intelligence

- AI and Cybersecurity
- NICF - Innovation Bootcamp (SF)
- NICF - Managing Cybersecurity Risk (SF)
- NICF - Platform Security (SF) #
- NICF - RPA and IPA - Strategy and Management (SF) #
- NICF - Secure Software Development Lifecycle for Agile (SF) #
- NICF - Securing IoT (SF)
- Professional Certificate in Digital Agility & Change Leadership #
- Professional Certificate in Digital Business Strategy #
- Professional Certificate in Digital Organisation Models #
- Professional Certificate in Innovation by Design #
- Professional Certificate in Mastering Digital Architecture #
- Professional Certificate in Managing Digitalisation Complexity #
- Professional Certificate in Strategic Thinking & Digital Foresight #
- Professional Certificate in Talent & Leadership Pathways #



Digital Innovation & Design

- NICF - Digital & Social Engagement Strategy (SF)
- NICF - Digital User Experience Design (SF)
- NICF - Innovation Bootcamp (SF)
- NICF - Mobile User Experience Design (SF)
- NICF - Service Design (SF) #
- NICF - Social Media Analytics (SF)
- NICF - Web Analytics & SEO (SF)

<https://www.iss.nus.edu.sg/executive-education>

NUS-ISS Short Course Opening/Closing Slides 2023

© Copyright National University of Singapore. All Rights Reserved

2



www.iss.nus.edu.sg



facebook.com/ISS.NUS



twitter.com/ISSNUS



@iss.nus



linkedin.com/company/iss_nus



youtube.com/user/TheISSNUS/