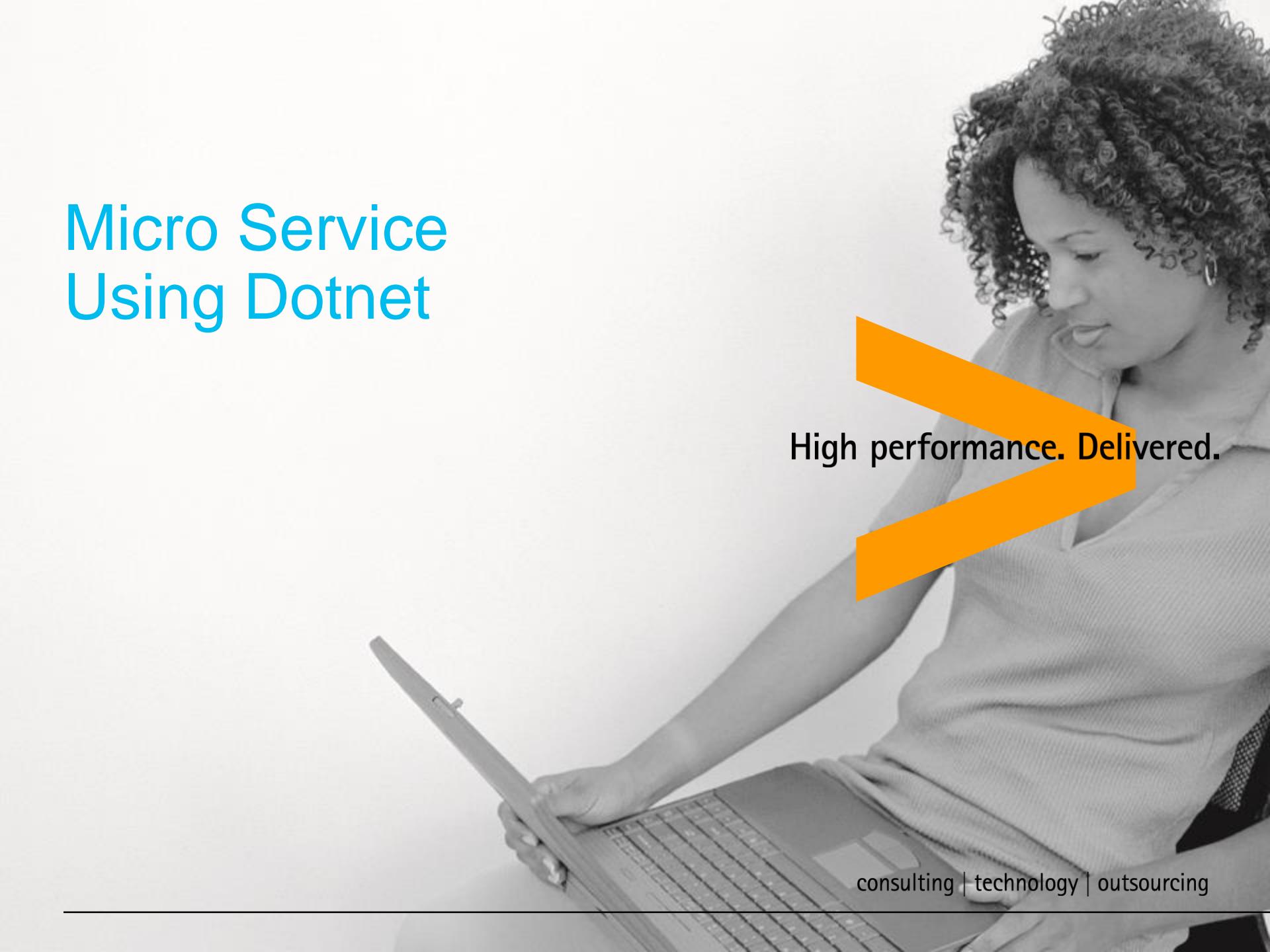


# Micro Service Using Dotnet

A black and white photograph of a woman with curly hair, wearing a light-colored shirt, sitting at a desk and working on a laptop. She is looking down at the screen. A large orange diagonal bar starts from the top right and extends towards the center of the image.

High performance. Delivered.



## Goals

- .NET Microservices: Architecture for Containerized .NET Applications
- Introduction to Containers and Docker
- Architecting Container and Microservice Based Applications
- Easy self-serve deployments of components
- Service-oriented architecture
- Microservices architecture



## Goals

- Identifying domain-model boundaries for each microservice
- Communication between microservices
- Asynchronous message-based communication
- Creating, evolving, and versioning microservice APIs and contracts“
- Microservices addressability and the service registry
- Creating composite UI based on microservices, including visual UI shape and layout generated by multiple microservices



# Micro Service

---

## Goals

- Orchestrating microservices and multi-container applications for high scalability and availability
- Using Azure Service Fabric
- Deploying Single Container Based .NET Core Web Applications on Linux or Windows Nano Server Hosts
- Migrating Legacy Monolithic .NET Framework Applications to Windows Containers
- Designing and Developing Multi Container and Microservice Based .NET Applications
- Implementing an event bus with RabbitMQ for the development or test environment Subscribing to events



## Goals

- Testing ASP.NET Core services and web apps
- Tackling Business Complexity in a Microservice with DDD and CQRS Patterns
- Applying CQRS and CQS approaches in a DDD microservice in eShopOnContainers
- Implementing reads/queries in a CQRS microservice
- Handling partial failure
- Strategies for handling partial failure
- Implementing retries with exponential backoff



# Micro Service

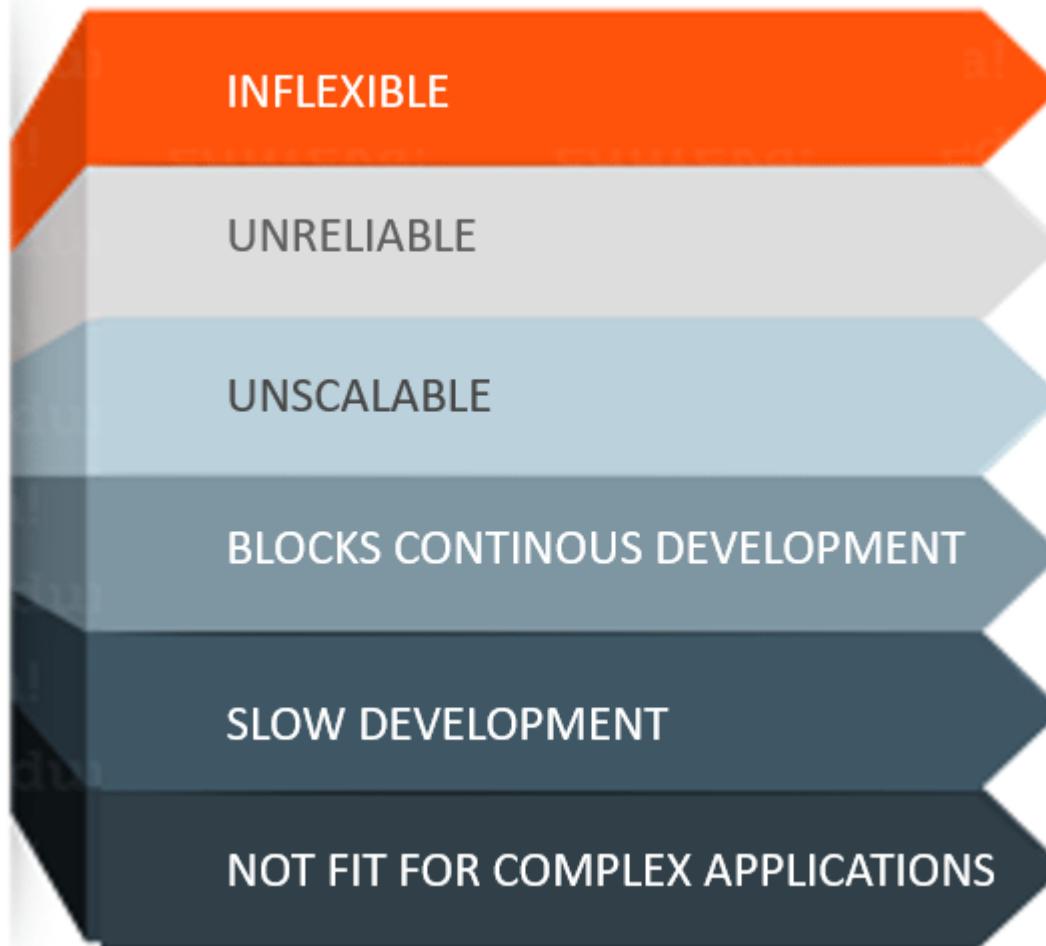
---

## Goals

- Securing .NET Microservices and Web Applications
- About authorization in .NET microservices and web applications
- Storing application secrets safely during development
- Using Azure Key Vault to protect secrets at production time
- Key takeaways
- <https://steeltoe.io/>
- <https://istio.io/>
-



# Monolithic Architecture





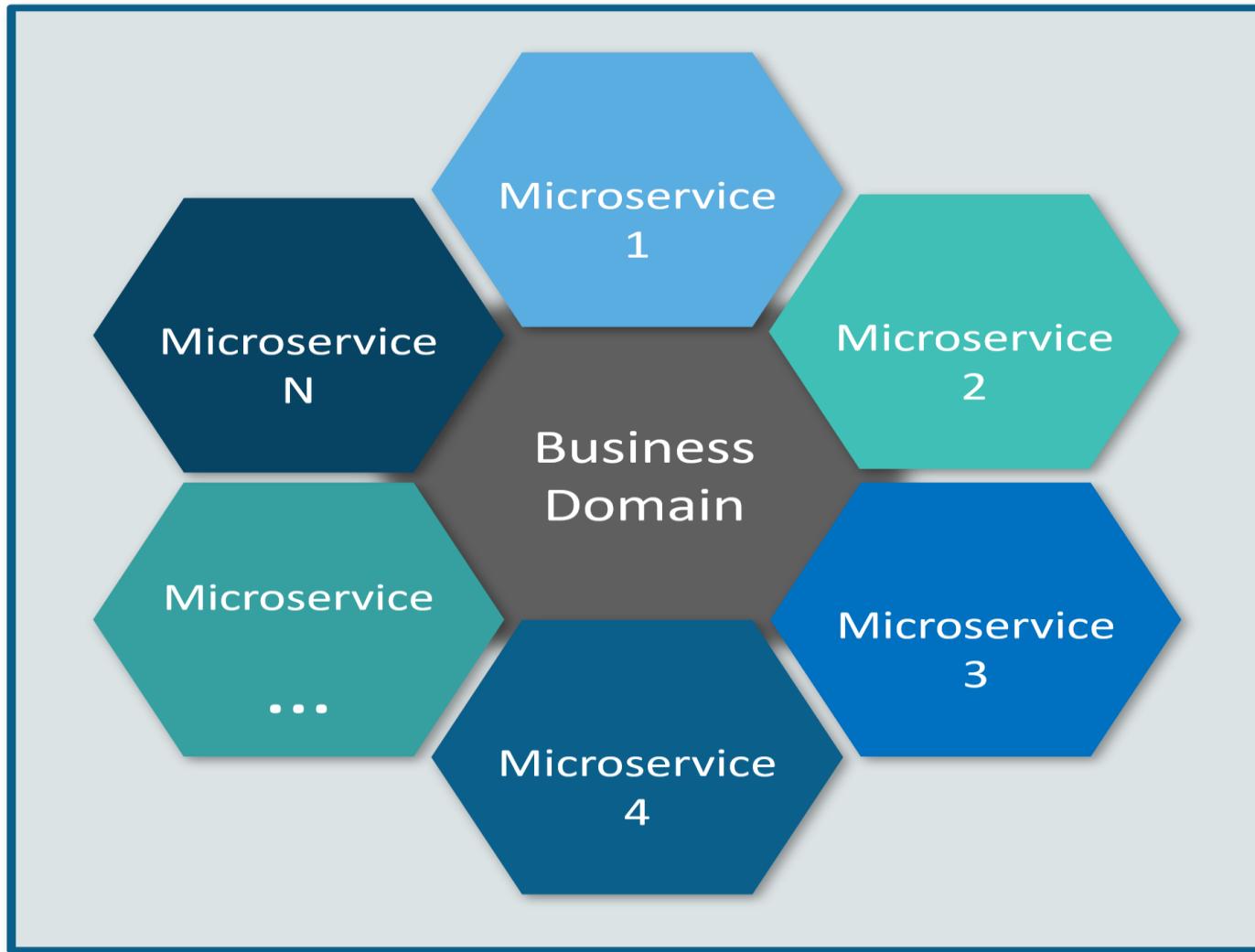
# Micro Service

*Small autonomous services that work together - Sam Newman*

*There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies - James Lewis and Martin Fowler*



# Microservice





*In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API....contd*



# Micro Service

*In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API....contd*



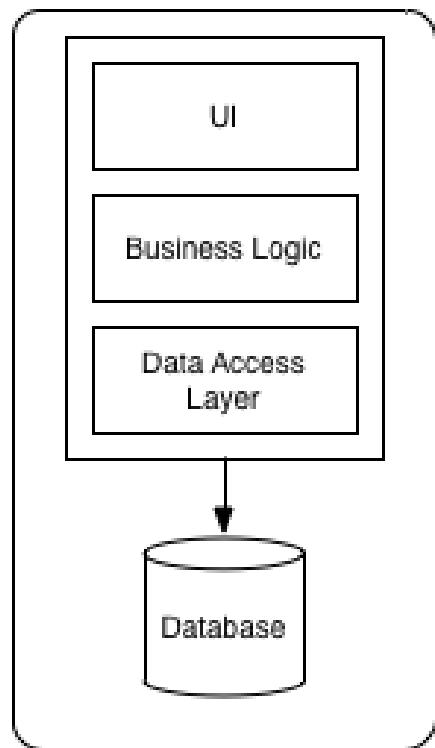
# Micro Service

*These services are built around  
business capabilities and  
independently deployable by fully  
automated deployment  
machinery...contd*

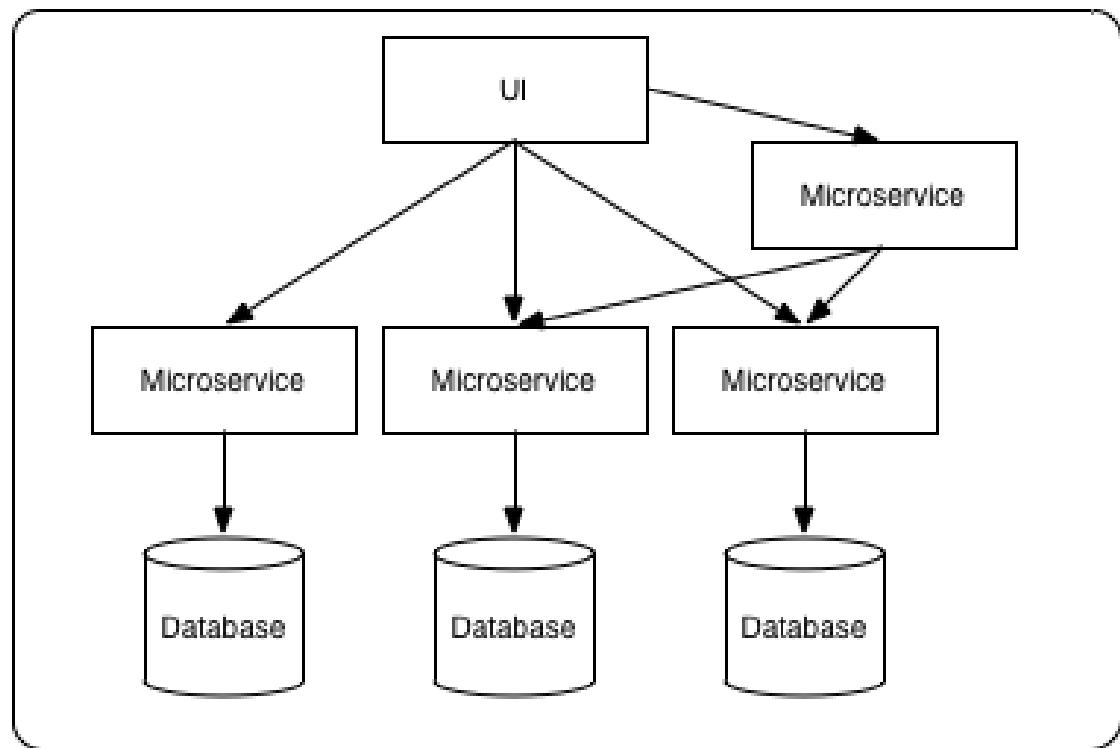


*There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies - James Lewis and Martin Fowler*

# MicroService



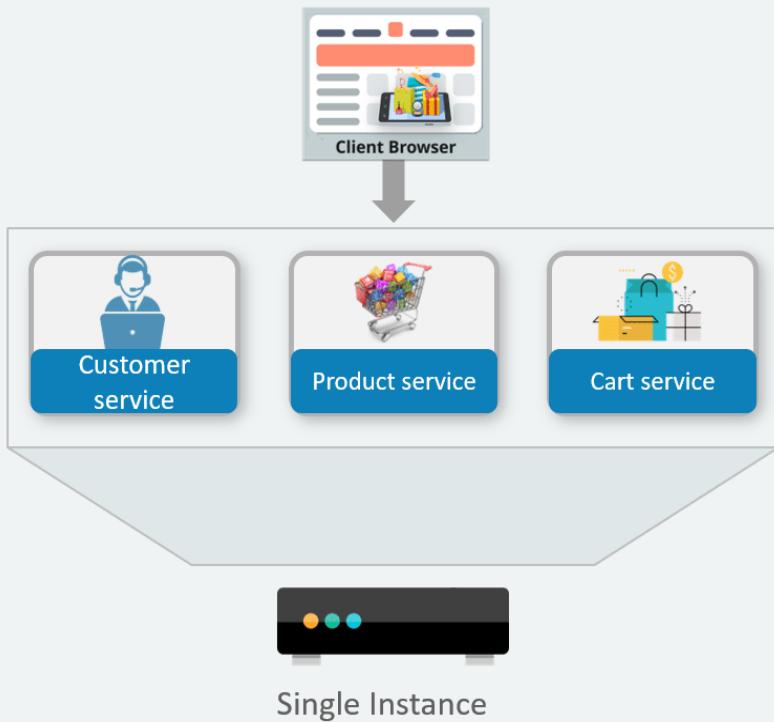
Monolithic Architecture



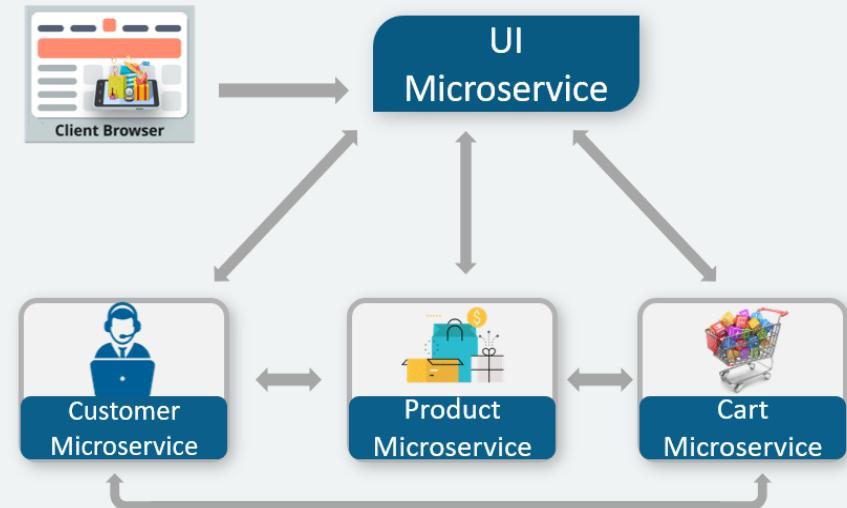
Microservices Architecture



## Monolithic Architecture

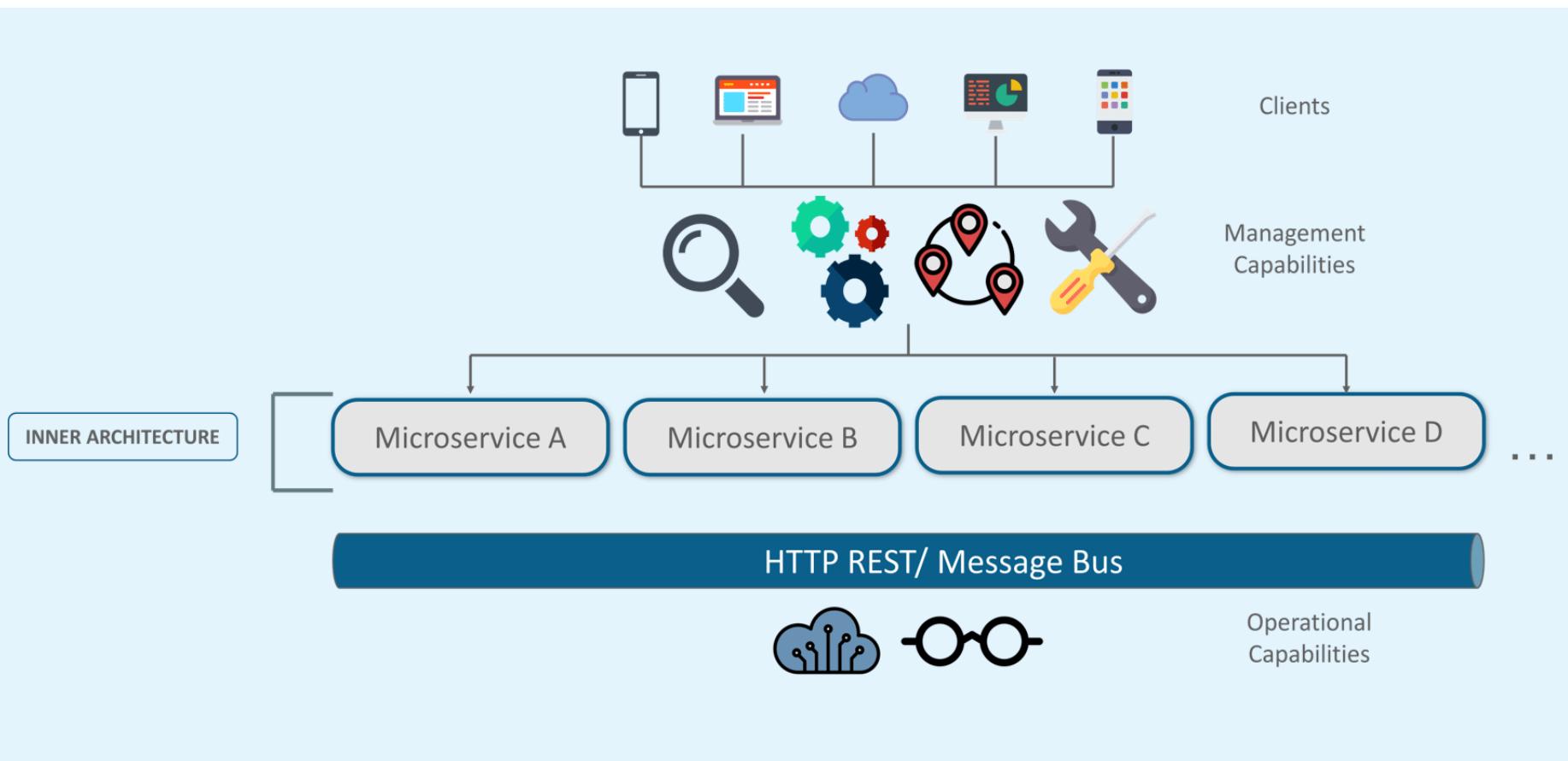


## Microservice Architecture





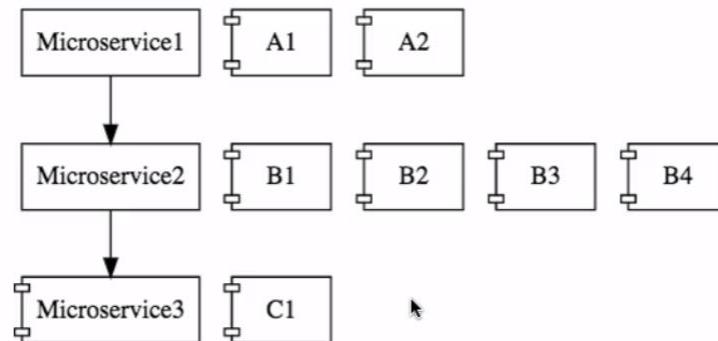
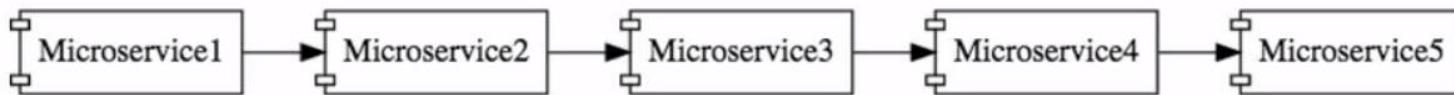
# Microservice Architecture



# Micro Service



- RESTful Web Services
- & Small Well Chosen Deployable Units
- & Cloud Enabled





# What is Microservices

---

- **Microservices** is a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services.



# What is Microservices

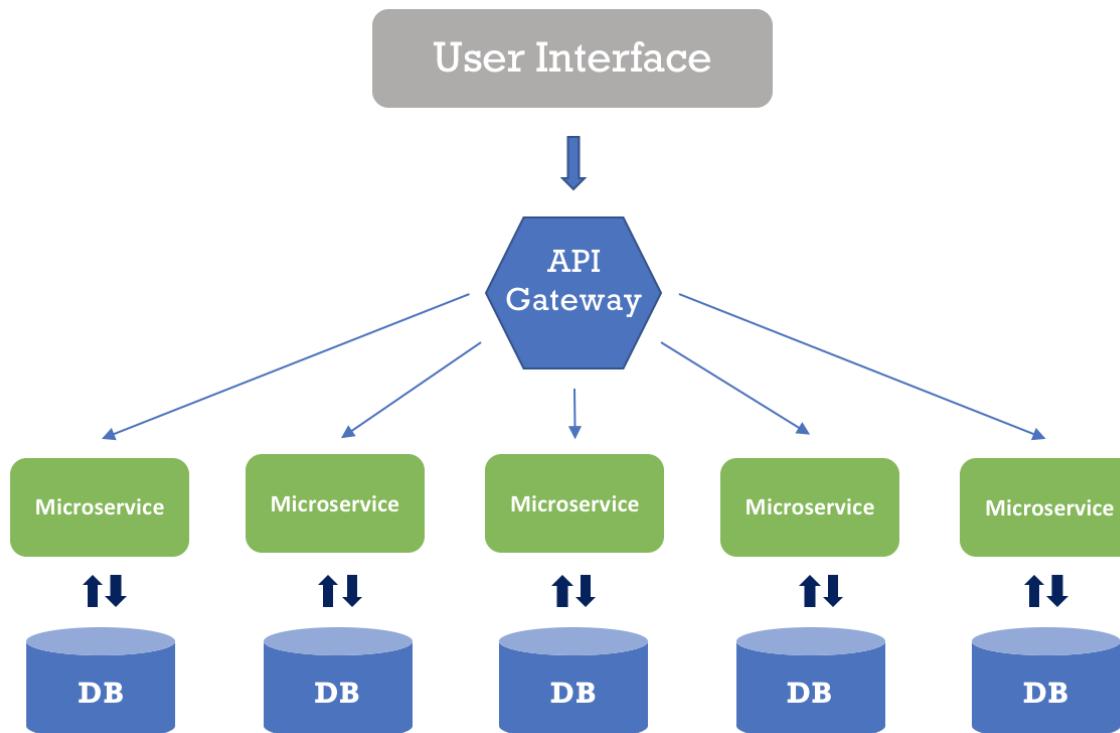
---

- In a microservices architecture, services should be fine-grained and the protocols should be lightweight.
- The benefit of decomposing an application into different smaller services is that it improves modularity and makes the application easier to understand, develop and test.



# What is Microservices

- It also parallelizes development by enabling small autonomous teams to develop, deploy and scale their respective services independently.



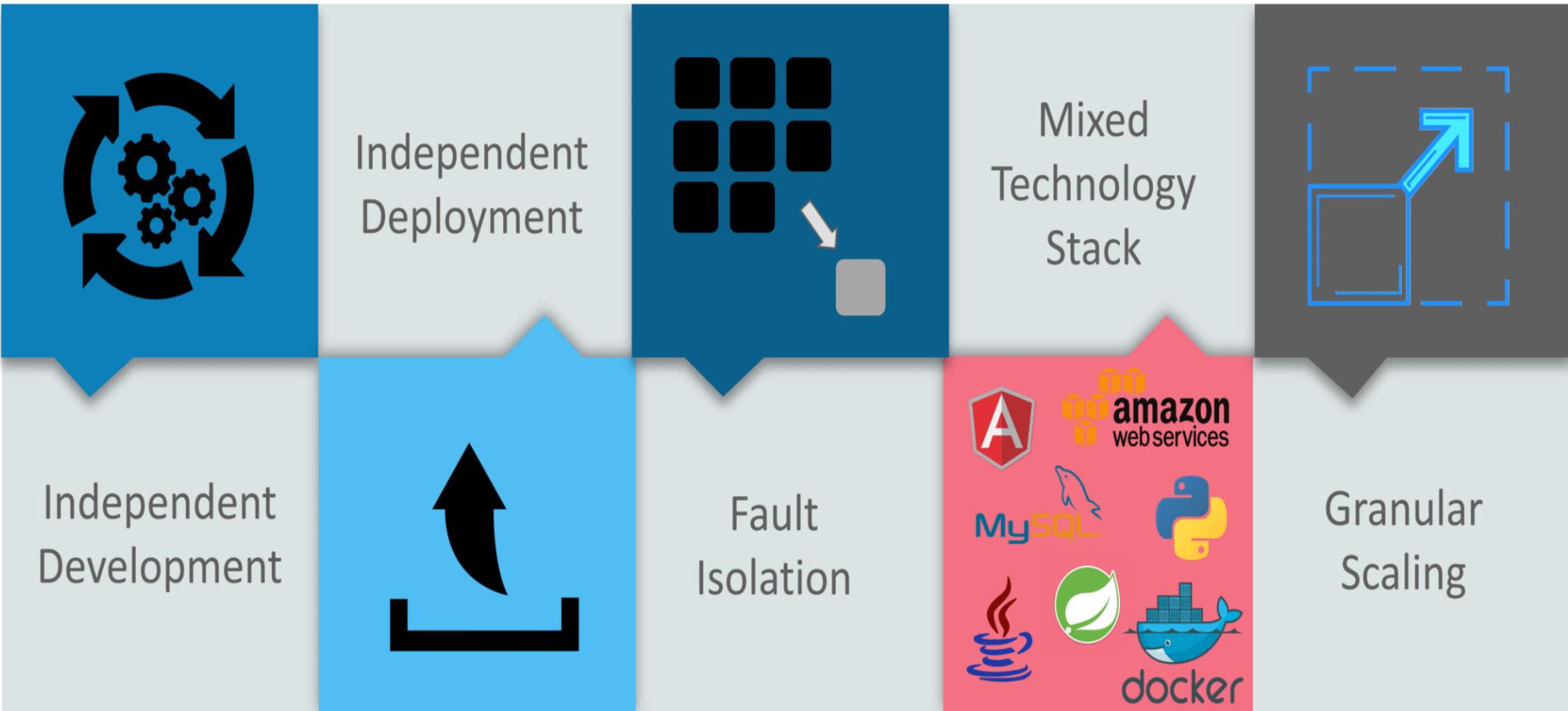


# Microservices Features

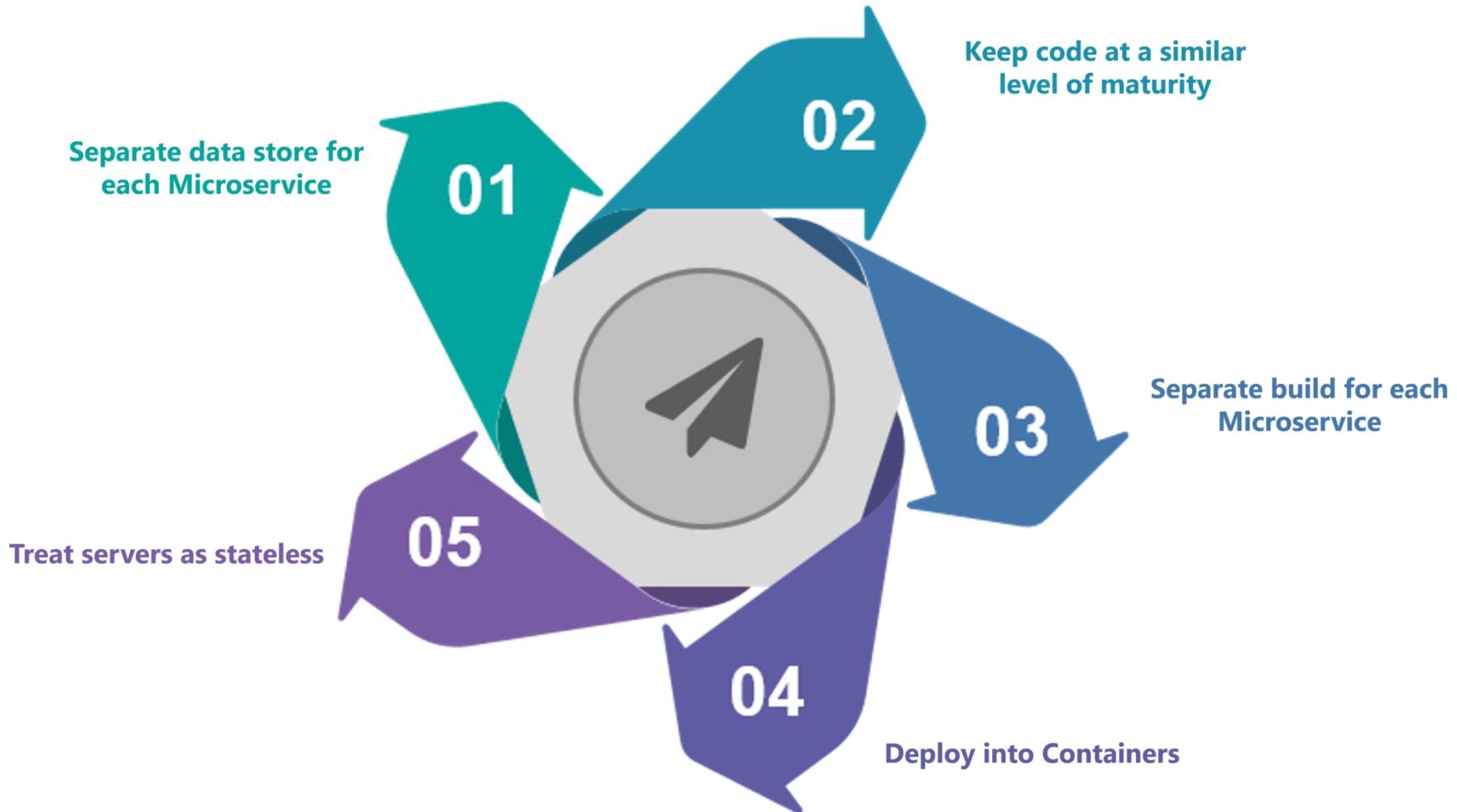




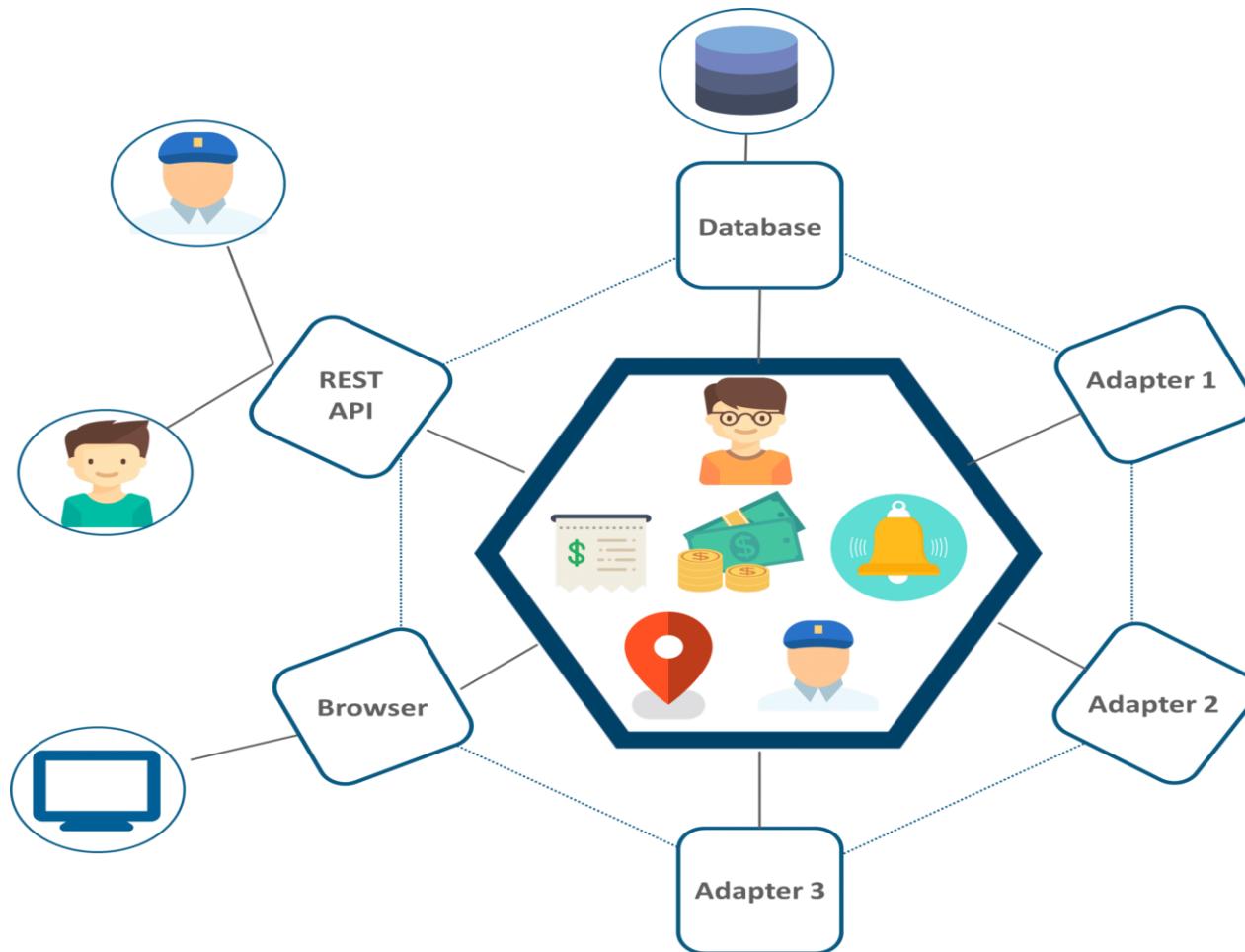
# Advantages Of Microservices



# Best Practices To Design Microservices



## UBER's Previous Architecture





## Problem Statement

---

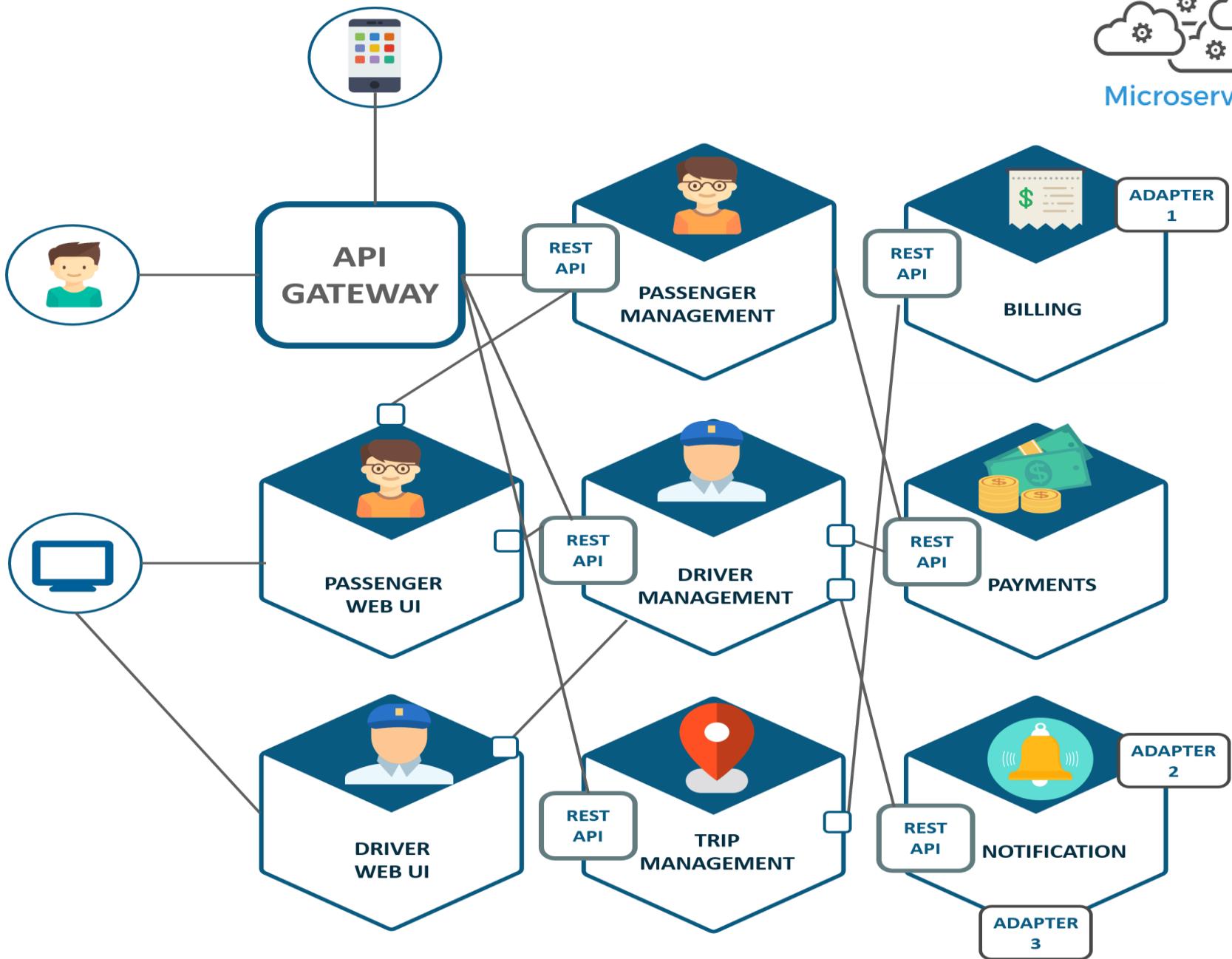
- While UBER started expanding worldwide this kind of framework introduced various challenges. The following are some of the prominent challenges
- All the features had to be re-built, deployed and tested again and again to update a single feature.
- Fixing bugs became extremely difficult in a single repository as developers had to change the code again and again.
- Scaling the features simultaneously with the introduction of new features worldwide was quite tough to be handled together.



## Solution

---

- To avoid such problems UBER decided to change its architecture and follow the other hyper-growth companies like Amazon, Netflix, Twitter and many others.
- Thus, UBER decided to break its monolithic architecture into multiple codebases to form a microservice architecture.



# What are the Benefits of Microservices?



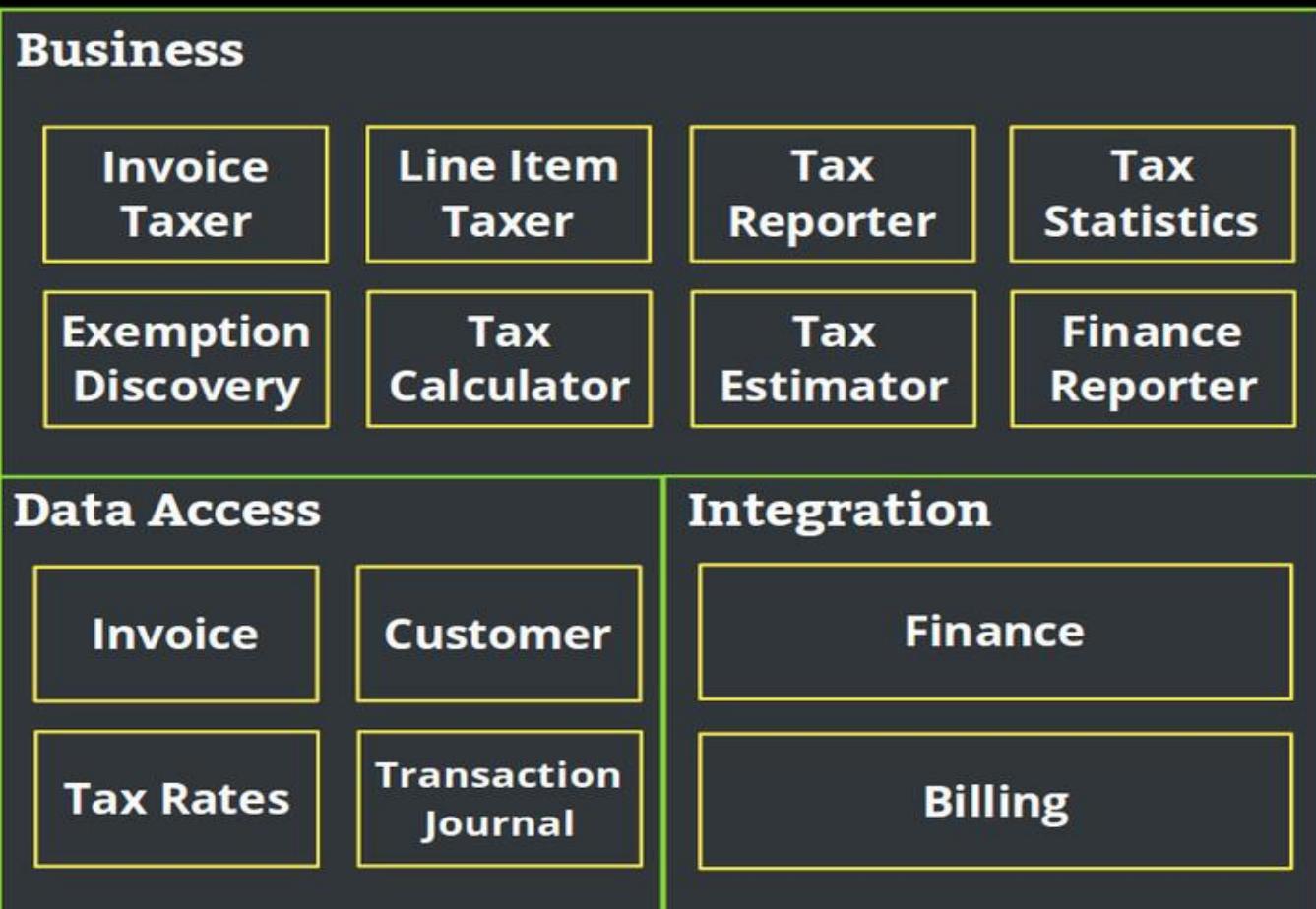
- Smaller applications are not dependent on the same coding language, the developers can use the programming language that they are most familiar with.
- That helps developers come up with a program faster with lower costs and fewer bugs.
- The agility and low costs can also come from being able to reuse these smaller programs on other projects, making it more efficient.



## Existing Scenarios

- Netflix has a widespread architecture that has evolved from monolithic to SOA. It receives more than *one billion* calls every day, from more than 800 different types of devices, to its streaming-video API. Each API call then prompts around five additional calls to the backend service.
- Amazon has also migrated to microservices. They get countless calls from a variety of applications—including applications that manage the web service API as well as the website itself—which would have been simply impossible for their old, two-tiered architecture to handle.
- The auction site eBay is yet another example that has gone through the same transition. Their core application comprises several autonomous applications, with each one executing the business logic for different function areas.

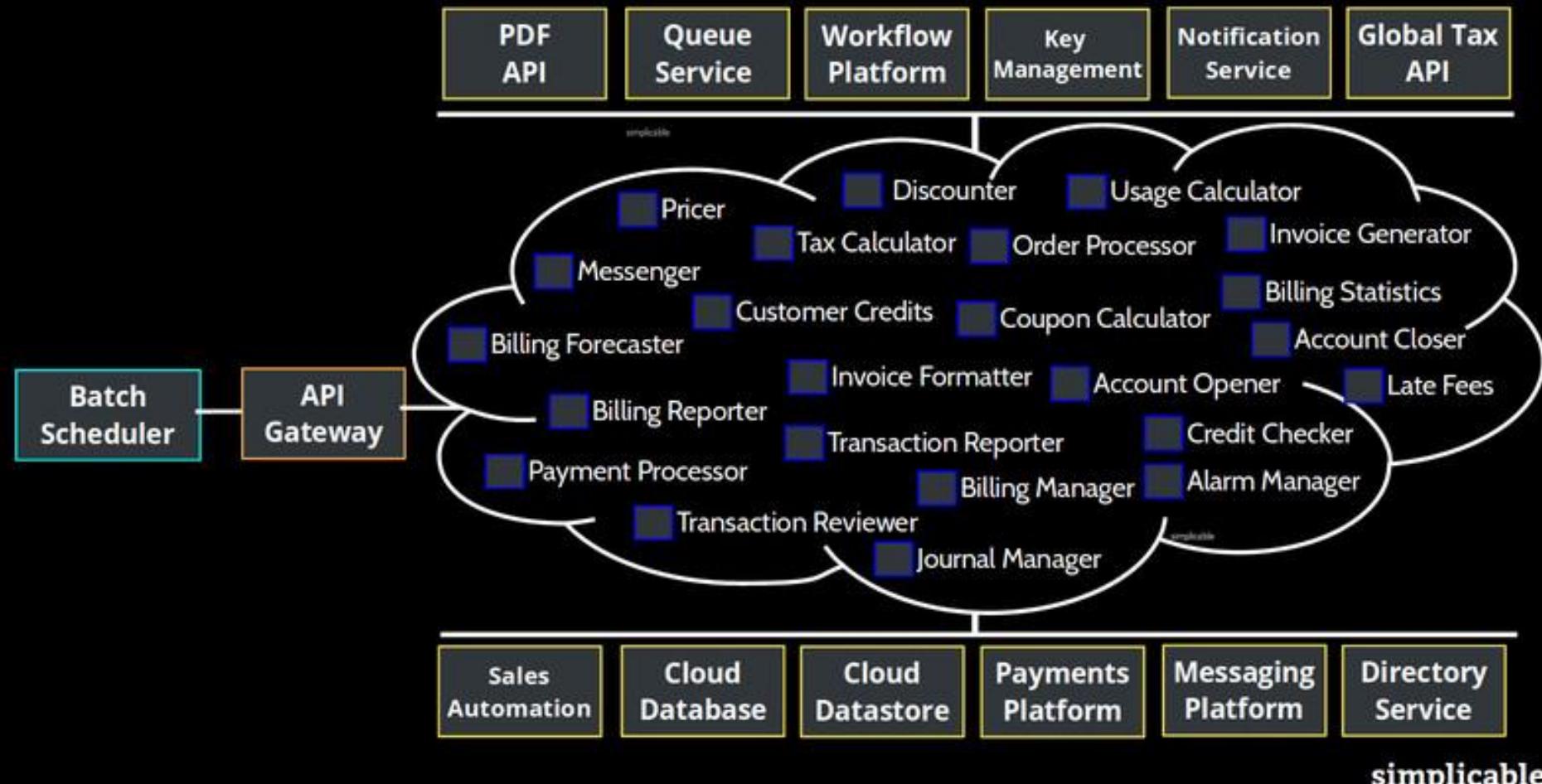
# Tax Calculator



Services

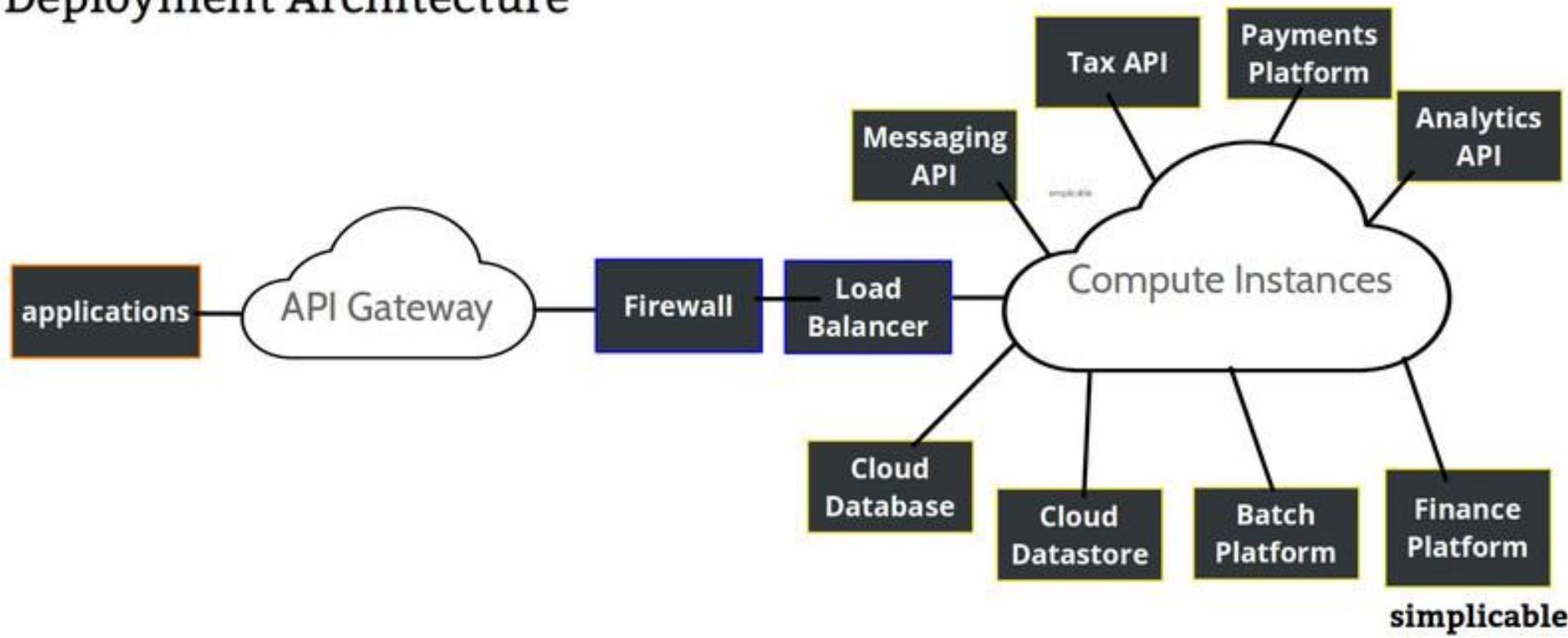
simplicable

# Services Architecture





## Deployment Architecture



# Principles of Microservices



- **Single responsibility principle**
- The single responsibility principle is one of the principles defined as part of the SOLID design pattern. It implies that a unit, either a class, a function, or a microservice, should have one and only one responsibility.
- At no point of time, one microservice should have more than one responsibility.

# Principles of Microservices



- **Built around business capabilities**
- **Microservices should focus on certain business function** and ensure that it helps in getting things done. A microservice shall never restrict itself from adopting appropriate technology stack or backend database storage which is most suitable for solving the business purpose.
- This is often the constraint when we design monolithic applications where we try to solve multiple business solutions with some compromises in some areas. Microservices enable you to choose what's best for the problem in hand.

# Principles of Microservices



- **You build it, you own it!**
- Another important aspect of such design is related to responsibilities pre-and-post development. In large organization, usually one team develops the app location, and after some knowledge transfer sessions it hand over the project to maintenance team. In microservices, the team which build the service – owns it, and is responsible for maintaining it in future.

# Principles of Microservices



- **Infrastructure Automation**
- Preparing and building infrastructure for microservices is another very important need. **A service shall be independently deployable** and shall bundle all dependencies, including library dependencies, and even execution environments such as web servers and containers or virtual machines that abstract physical resources.
- One of the major **differences between microservices and SOA** is in their level of autonomy. While most SOA implementations provide service-level abstraction, microservices go further and abstract the realization and execution environment.

# Principles of Microservices



- **Infrastructure Automation**
- In traditional application developments, we build a WAR or an EAR, then deploy it into a JEE application server, such as with JBoss, WebLogic, WebSphere, and so on. We may deploy multiple applications into the same JEE container. In ideal scenario, in the microservices approach, each microservice will be built as a fat Jar, embedding all dependencies and run as a standalone Java process.

# Principles of Microservices



- **Design for Failure**
- A microservice shall be designed with failure cases in mind. What if the service fails, or go down for some time. These are very important questions and must be solved before actual coding starts – to clearly estimate **how service failures will affect the user experience**.
- Fail fast is another concept used to build fault-tolerant, resilient systems. This philosophy advocates systems that expect failures versus building systems that never fail. Since services can fail at any time, it's important to be able to detect the failures quickly and, if possible, automatically restore service.

# Principles of Microservices



- **Design for Failure**
- Microservice applications put a lot of emphasis on real-time monitoring of the application, checking both architectural elements (how many requests per second is the database getting) and business relevant metrics (such as how many orders per minute are received). Semantic monitoring can provide an early warning system of something going wrong that triggers development teams to follow up and investigate.

# Benefits of Microservices



- With microservices, architects and developers can choose fit for purpose architectures and technologies for each microservice (polyglot architecture). This gives the flexibility to design better-fit solutions in a more cost-effective way.
- As services are fairly simple and smaller in size, enterprises can afford to experiment new processes, algorithms, business logic, and so on. It enables enterprises to do disruptive innovation by offering the ability to experiment and fail fast.

# Benefits of Microservices



- Microservices enable to implement selective scalability i.e. each service could be independently scaled up or down and cost of scaling is comparatively less than monolithic approach.
- Microservices are self-contained, independent deployment modules enabling the substitution of one microservice with another similar microservice, when second one is not performing as per our need. It helps in taking right buy-versus-build decisions which are often the challenge for many enterprises.

# Benefits of Microservices

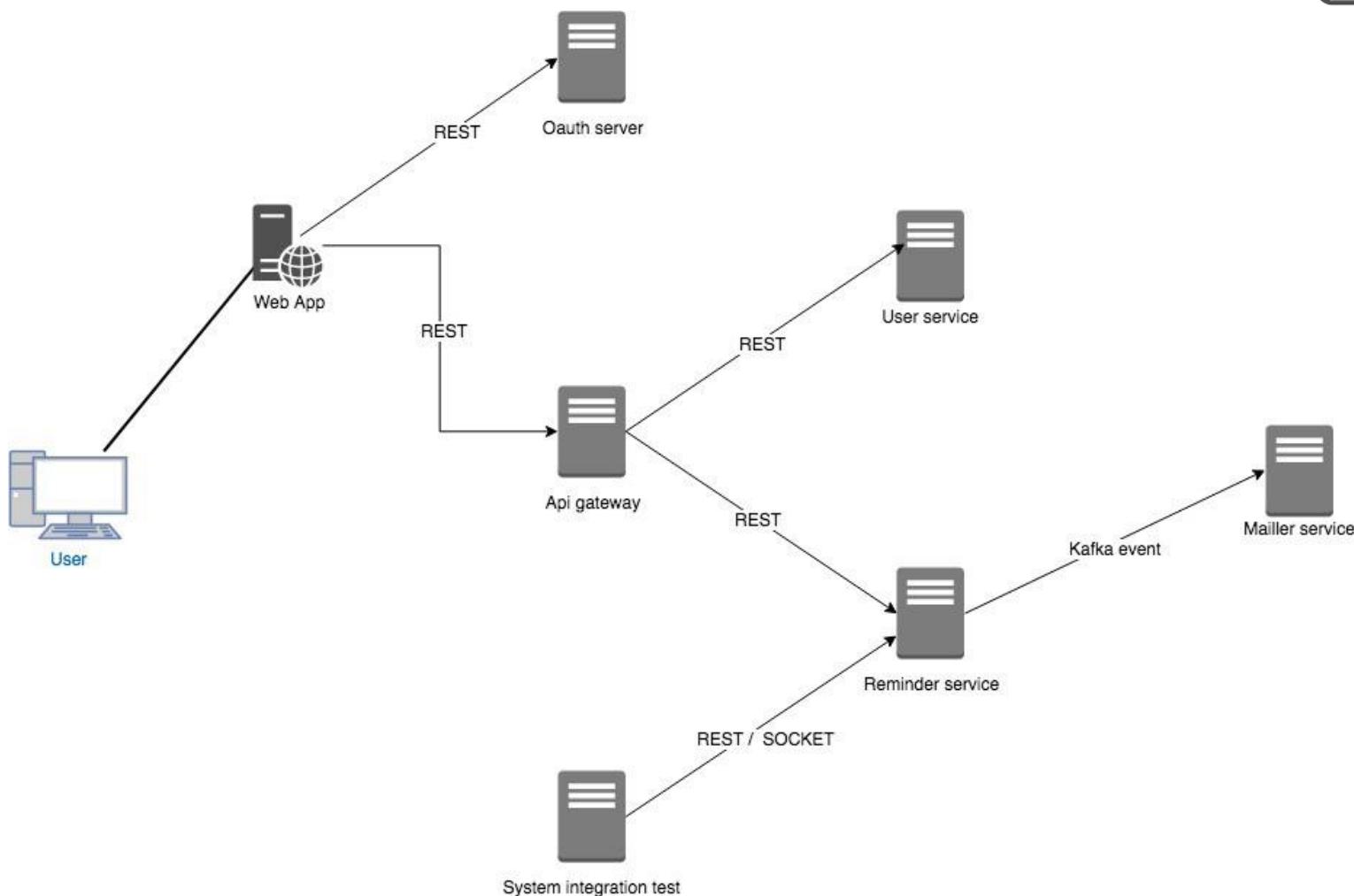


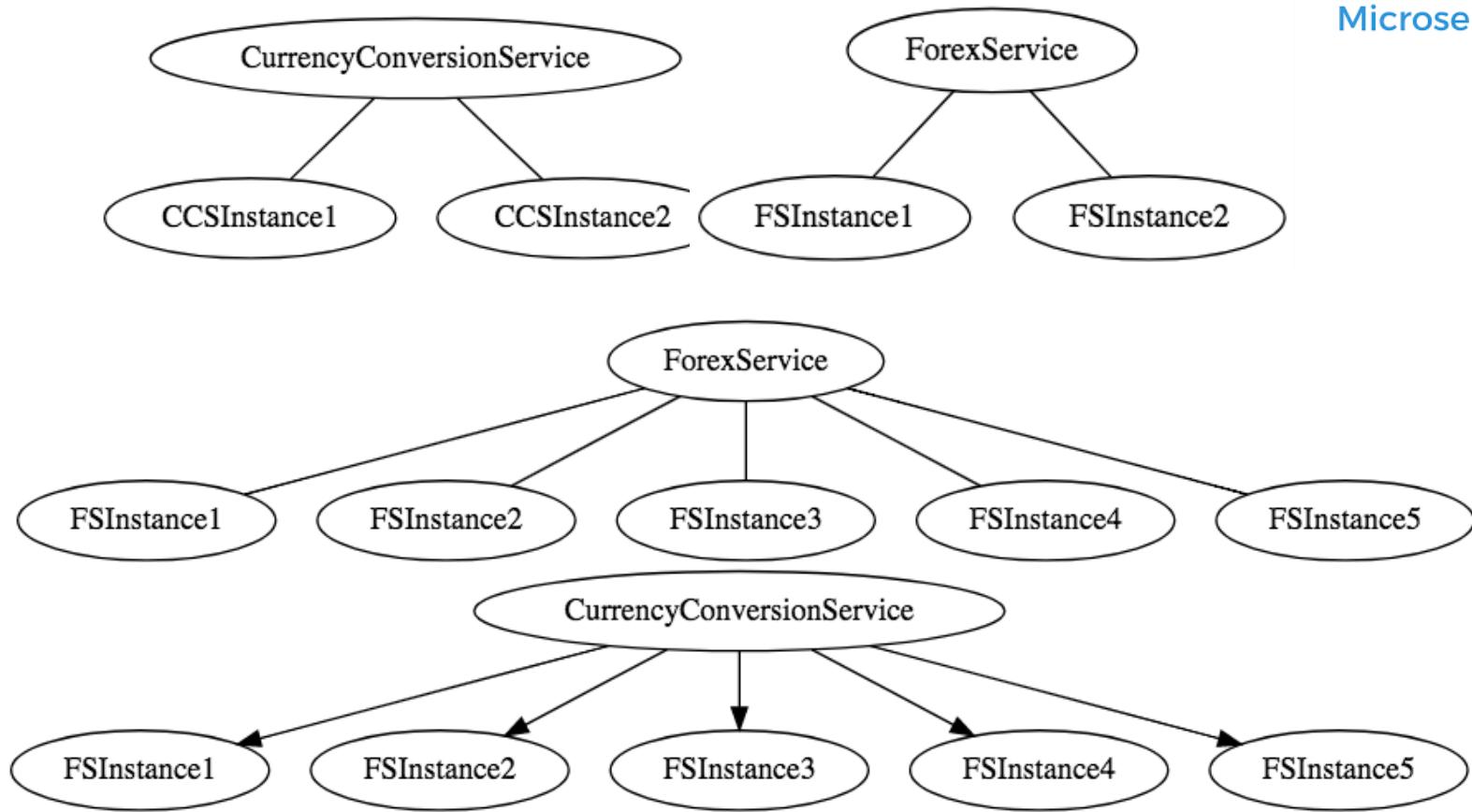
- Microservices help us build systems that are organic in nature(Organic systems are systems that grow laterally over a period of time by adding more and more functions to it). Because microservices are all about independently manageable services – it enable to add more and more services as the need arises with minimal impact on the existing services.
- Technology changes are one of the barriers in software development. With microservices, it is possible to change or upgrade technology for each service individually rather than upgrading an entire application.

# Benefits of Microservices



- 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.
- And finally, microservices also enable smaller, focused agile teams for development. Teams will be organized based on the boundaries of microservices.





# Examples of Microservices Frameworks for Java



- There are several microservices frameworks that you can use for developing for Java. Some of these are:
- **Spring Boot.** This is probably the best Java microservices framework that works on top of languages for Inversion of Control, Aspect Oriented Programming, and others.
- **Jersey.** This open source framework supports JAX-RS APIs in Java is very easy to use.
- **Swagger.** Helps you in documenting API as well as gives you a development portal, which allows users to test your APIs.



## Pros

---

- Microservice architecture gives developers the freedom to independently develop and deploy services
- A microservice can be developed by a fairly small team
- Code for different services can be written in different languages (though many practitioners discourage it)
- Easy integration and automatic deployment (using open-source continuous integration tools such as Jenkins, Hudson, etc.)



## Pros

---

- Easy to understand and modify for developers, thus can help a new team member become productive quickly
- The developers can make use of the latest technologies
- The code is organized around business capabilities
- Starts the web container more quickly, so the deployment is also faster



## Pros

---

- When change is required in a certain part of the application, only the related service can be modified and redeployed—no need to modify and redeploy the entire application
- Better fault isolation: if one microservice fails, the other will continue to work (although one problematic area of a monolith application can jeopardize the entire system)
- Easy to scale and integrate with third-party services
- No long-term commitment to technology stack



## Cons

---

- Due to distributed deployment, testing can become complicated and tedious
- Increasing number of services can result in information barriers
- The architecture brings additional complexity as the developers have to mitigate fault tolerance, network latency, and deal with a variety of message formats as well as load balancing
- Being a distributed system, it can result in duplication of effort



## Cons

---

- When number of services increases, integration and managing whole products can become complicated
- In addition to several complexities of monolithic architecture, the developers have to deal with the additional complexity of a distributed system
- Developers have to put additional effort into implementing the mechanism of communication between the services



## Cons

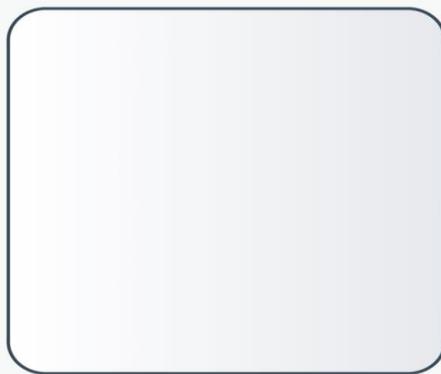
---

- Handling use cases that span more than one service without using distributed transactions is not only tough but also requires communication and cooperation between different teams
- The architecture usually results in increased memory consumption
- Partitioning the application into microservices is very much an art.



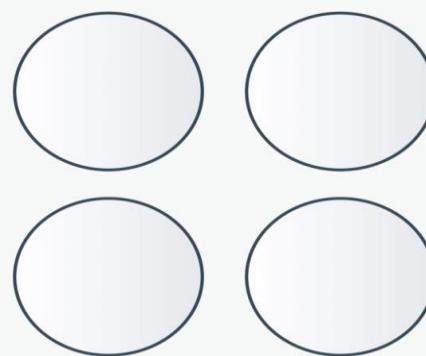
# SOA vs Microservices

## Monolithic vs. SOA vs. Microservices



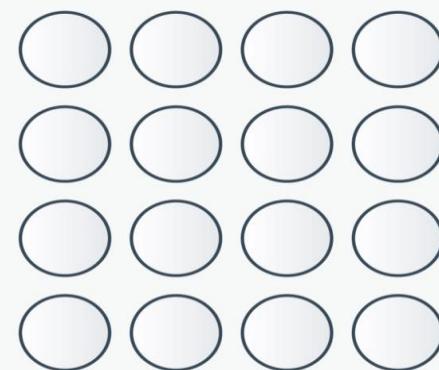
**Monolithic**

**Single Unit**



**SOA**

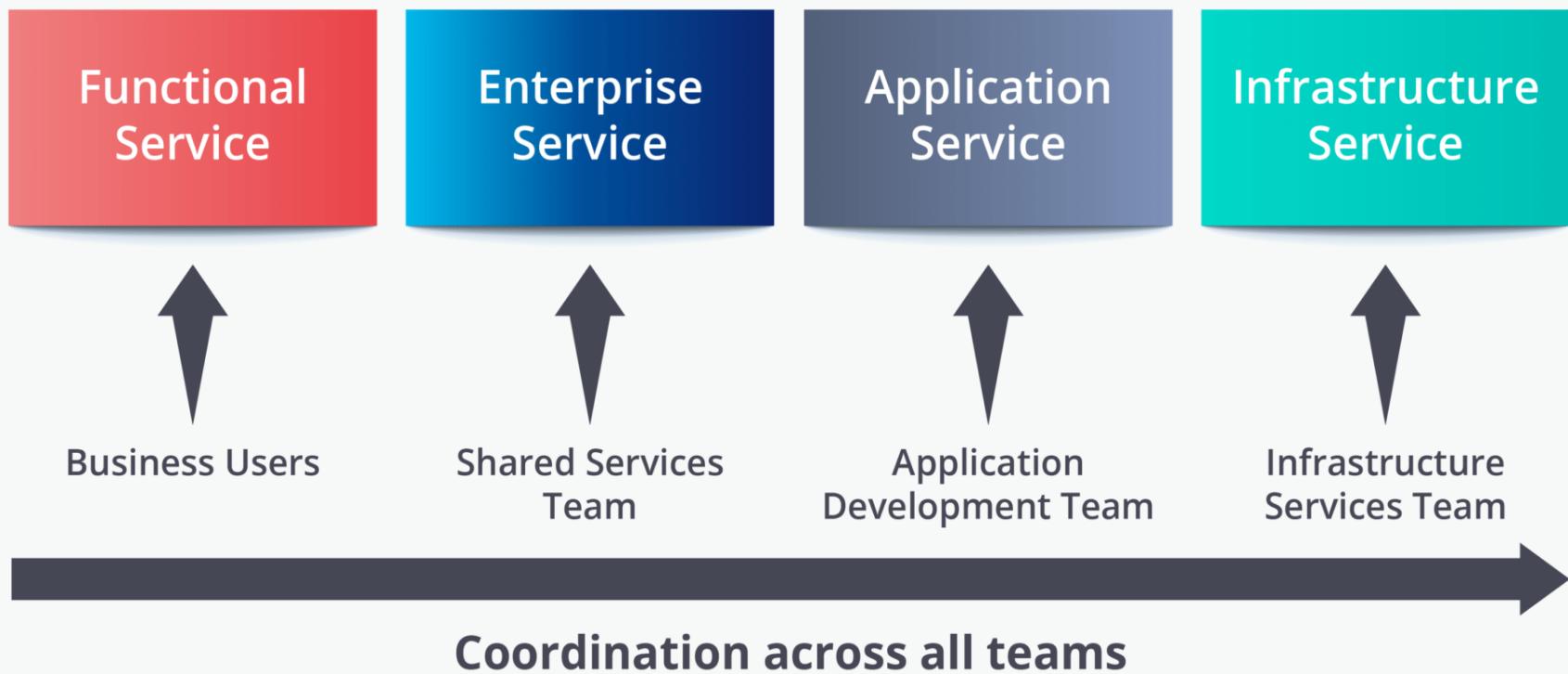
**Coarse-grained**



**Microservices**

**Fine-grained**

# Service Oriented Architecture





## SOA ARCHITECTURE



Services/APIs



Application



Application Server



Managed Runtime Environment



Operating System



Hypervisor



Storage



Network

## MICROSERVICES

Services/APIs



Managed Runtime Environment



Operating System



Hypervisor



Storage



Network



SOA	MSA
Follows “ <b>share-as-much-as-possible</b> ” architecture approach	Follows “ <b>share-as-little-as-possible</b> ” architecture approach
Importance is on <b>business functionality</b> reuse	Importance is on the concept of “ <b>bounded context</b> ”
They have <b>common governance</b> and <b>standards</b>	They focus on <b>people collaboration</b> and freedom of other options
Uses <b>Enterprise Service bus (ESB)</b> for communication	Simple messaging system
They support <b>multiple message protocols</b>	They use <b>lightweight protocols</b> such as <b>HTTP/REST</b> etc.
<b>Multi-threaded</b> with more overheads to handle I/O	<b>Single-threaded</b> usually with the use of Event Loop features for non-locking I/O handling
Maximizes application service reusability	Focuses on <b>decoupling</b>
<b>Traditional Relational Databases</b> are more often used	<b>Modern Relational Databases</b> are more often used
A systematic change requires modifying the monolith	A systematic change is to create a new service
DevOps / Continuous Delivery is becoming popular, but not yet mainstream	Strong focus on DevOps / Continuous Delivery

# Challenges with Microservice Architectures



- Quick Setup needed : You cannot spend a month setting up each microservice. You should be able to create microservices quickly.
- Automation : Because there are a number of smaller components instead of a monolith, you need to automate everything - Builds, Deployment, Monitoring etc.
- Visibility : You now have a number of smaller components to deploy and maintain. Maybe 100 or maybe 1000 components. You should be able to monitor and identify problems automatically. You need great visibility around all the components.



# Challenges with Microservice Architectures

- Bounded Context : Deciding the boundaries of a microservice is not an easy task. Bounded Contexts from Domain Driven Design is a good starting point. Your understanding of the domain evolves over a period of time. You need to ensure that the microservice boundaries evolve.
- Configuration Management : You need to maintain configurations for hundreds of components across environments. You would need a Configuration Management solution
- Dynamic Scale Up and Scale Down : The advantages of microservices will only be realized if your applications can scaled up and down easily in the cloud.
- Pack of Cards : If a microservice at the bottom of the call chain fails, it can have knock on effects on all other microservices. Microservices should be fault tolerant by Design.



# Challenges with Microservice Architectures

---

- Debugging : When there is a problem that needs investigation, you might need to look into multiple services across different components. Centralized Logging and Dashboards are essential to make it easy to debug problems.
- Consistency : You cannot have a wide range of tools solving the same problem. While it is important to foster innovation, it is also important to have some decentralized governance around the languages, platforms, technology and tools used for implementing/deploying/monitoring microservices.

# Scaling Challenge

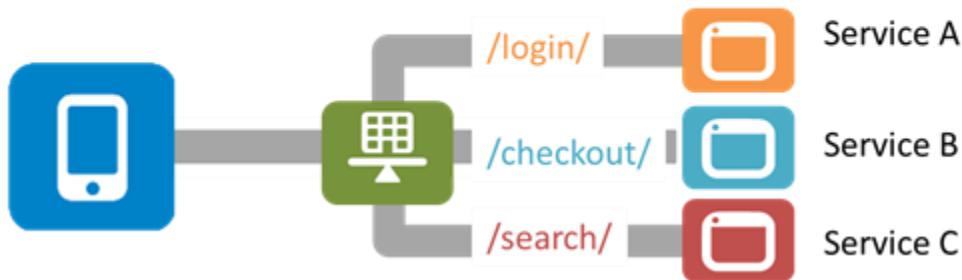
## X-AXIS SCALING

Network name: Horizontal scaling, scale out



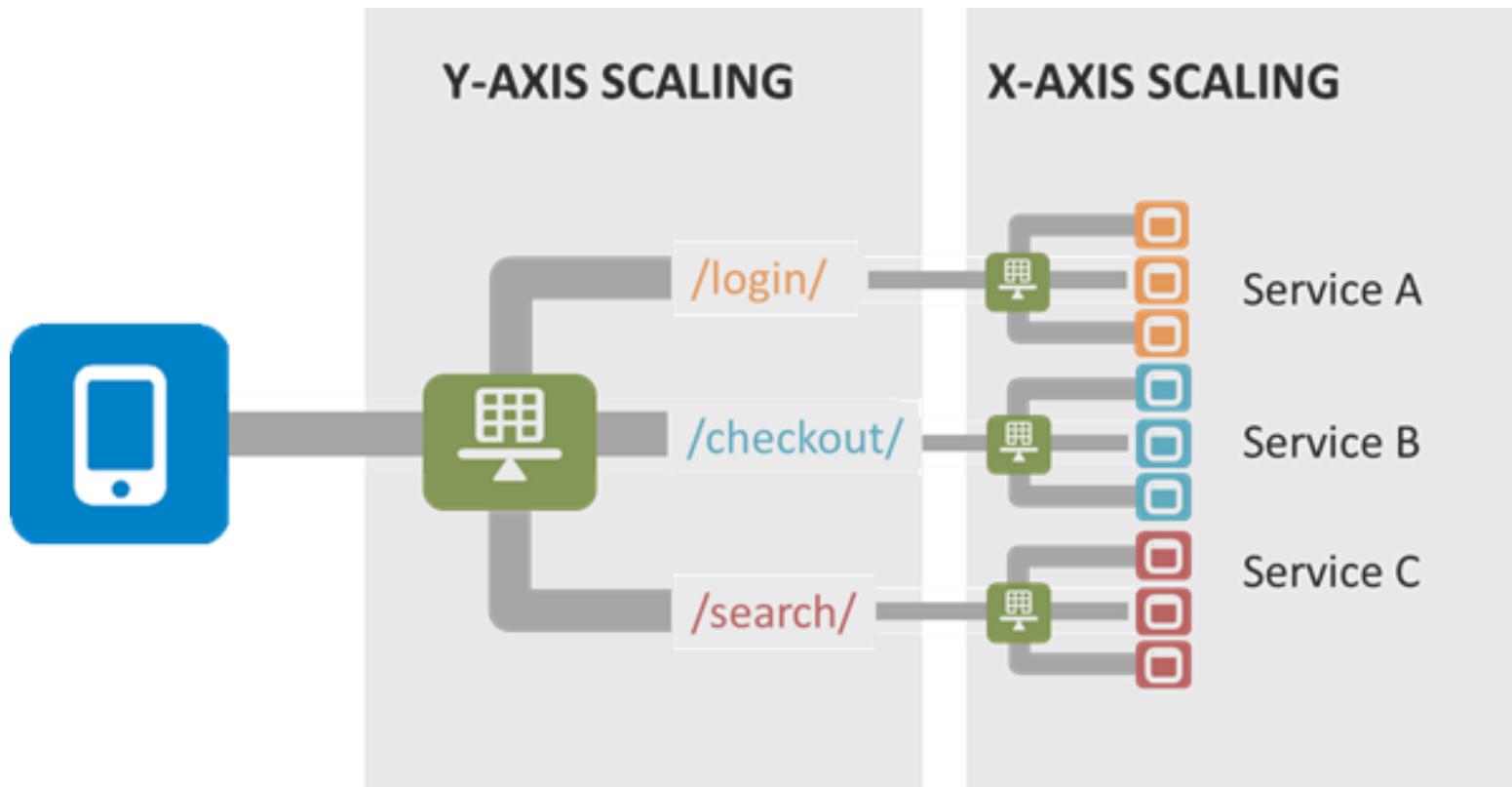
## Y-AXIS SCALING

Network name: Layer 7 Load Balancing, Content switching, HTTP Message Steering





# Scaling Challenge



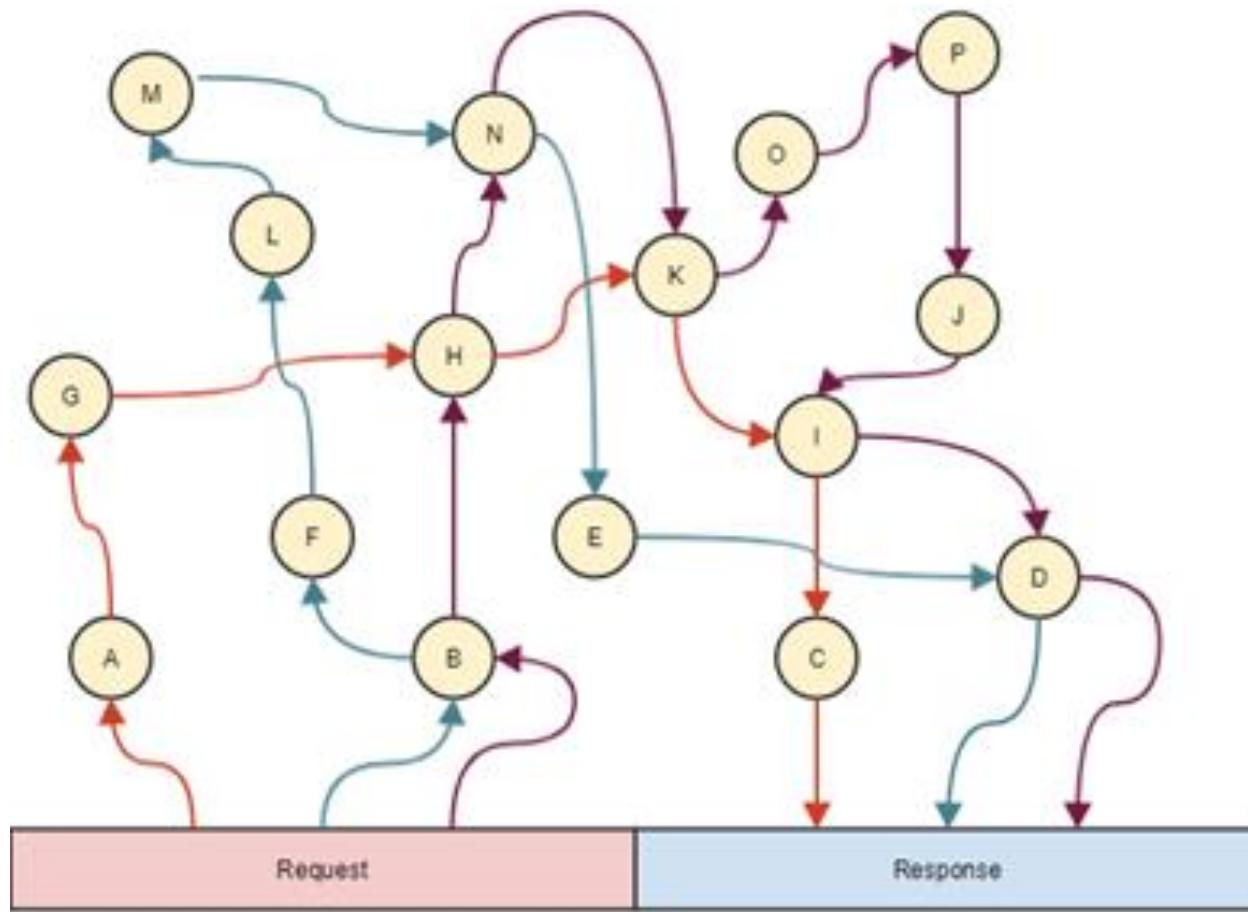
# Scaling Challenge

## Z-AXIS SCALING

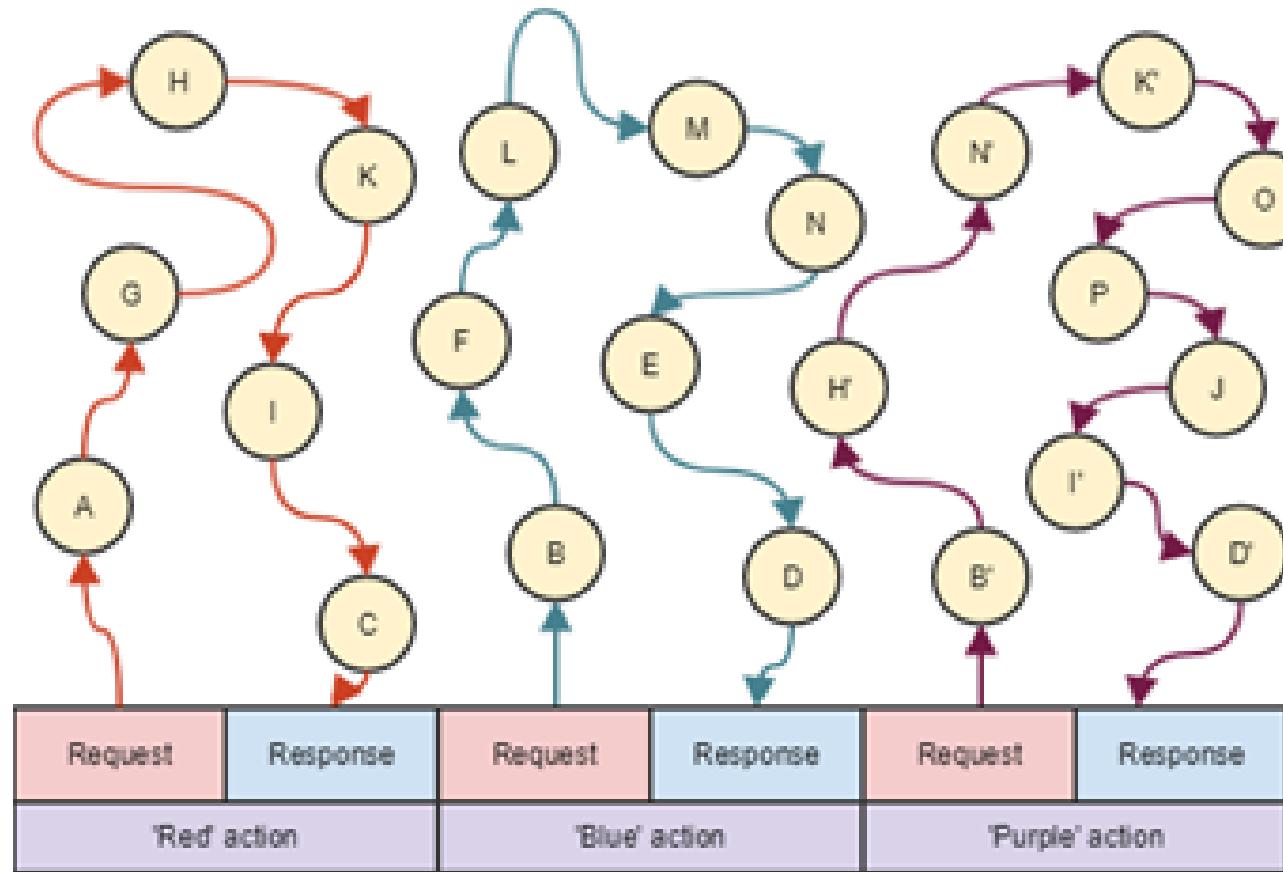
Network name: Layer 7 Load Balancing, Content switching, HTTP Message Steering



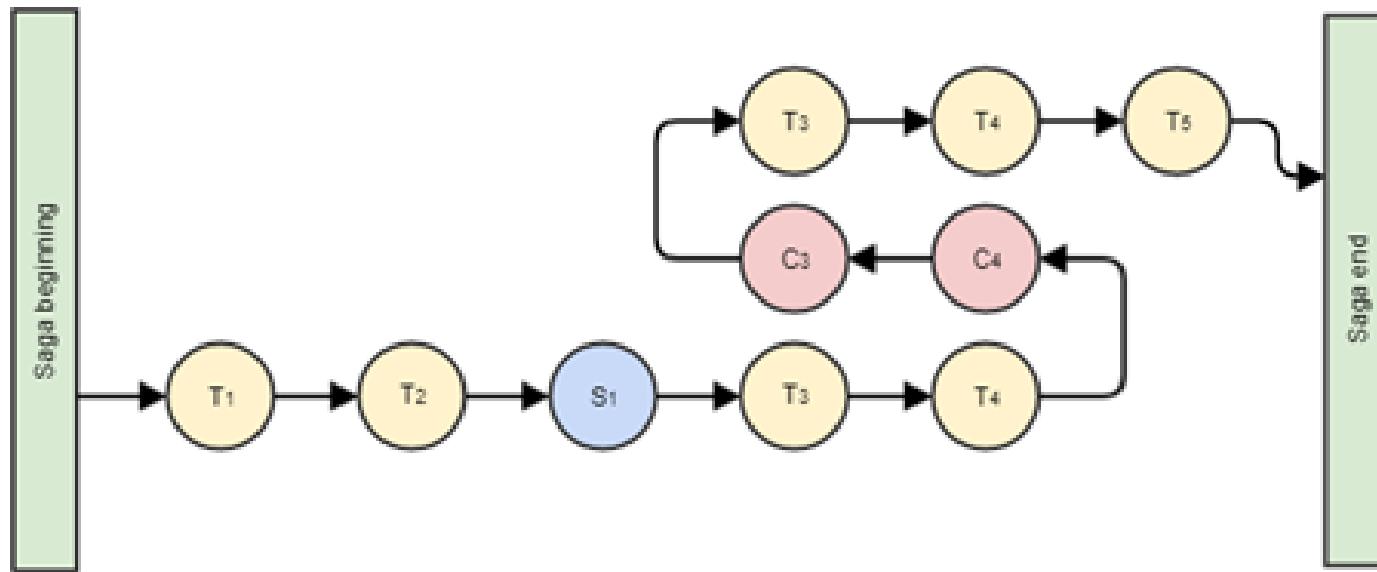
# Saga Pattern



# Saga Pattern



# Saga Pattern





# Proposed Solutions to Handle Challenges

- **1. Data Synchronization** — We have event sourcing architecture to address this issue using the async messaging platform. The saga design pattern can address this challenge.
- **2. Security** — An API Gateway can solve these challenges. Kong is very popular and is open-source, and is being used by many companies in production. Custom solutions can also be developed for API security using JWT token, Spring Security, and Netflix Zuul/ Zuul2. There are enterprise solutions available, too, like Apigee and Okta (2-step authentication). Openshift is used for public cloud security for its top features, like Red Hat Linux Kernel-based security and namespace-based app-to-app security.
- **3. Versioning** — This will be taken care of by API registry and discovery APIs using the dynamic Swagger API, which can be updated dynamically and shared with consumers on the server.
- **4. Discovery** — This will be addressed by API discovery tools like Kubernetes and OpenShift. It can also be done using Netflix Eureka at the code level. However, doing it in with the orchestration layer will be better and can be managed by these tools rather doing and maintaining it through code and configuration.



# Proposed Solutions to Handle Challenges

- **5. Data Staleness** — The database should be always updated to give recent data. The API will fetch data from the recent and updated database. A timestamp entry can also be added with each record in the database to check and verify the recent data. Caching can be used and customized with an acceptable eviction policy based on business requirements.
- **6. Debugging and Logging** — There are multiple solutions for this. Externalized logging can be used by pushing log messages to an async messaging platform like Kafka, Google PubSub, etc. A correlation ID can be provided by the client in the header to REST APIs to track the relevant logs across all the pods/Docker containers. Also, local debugging can be done individually on each microservice using the IDE or checking the logs.
- **7. Testing** — This issue can be addressed with unit testing by mocking REST APIs or integrated/dependent APIs which are not available for testing using WireMock, BDD, Cucumber, integration testing, performance testing using JMeter, and any good profiling tool like Jprofiler, DynaTrace, YourToolKit, VisualVM, etc.

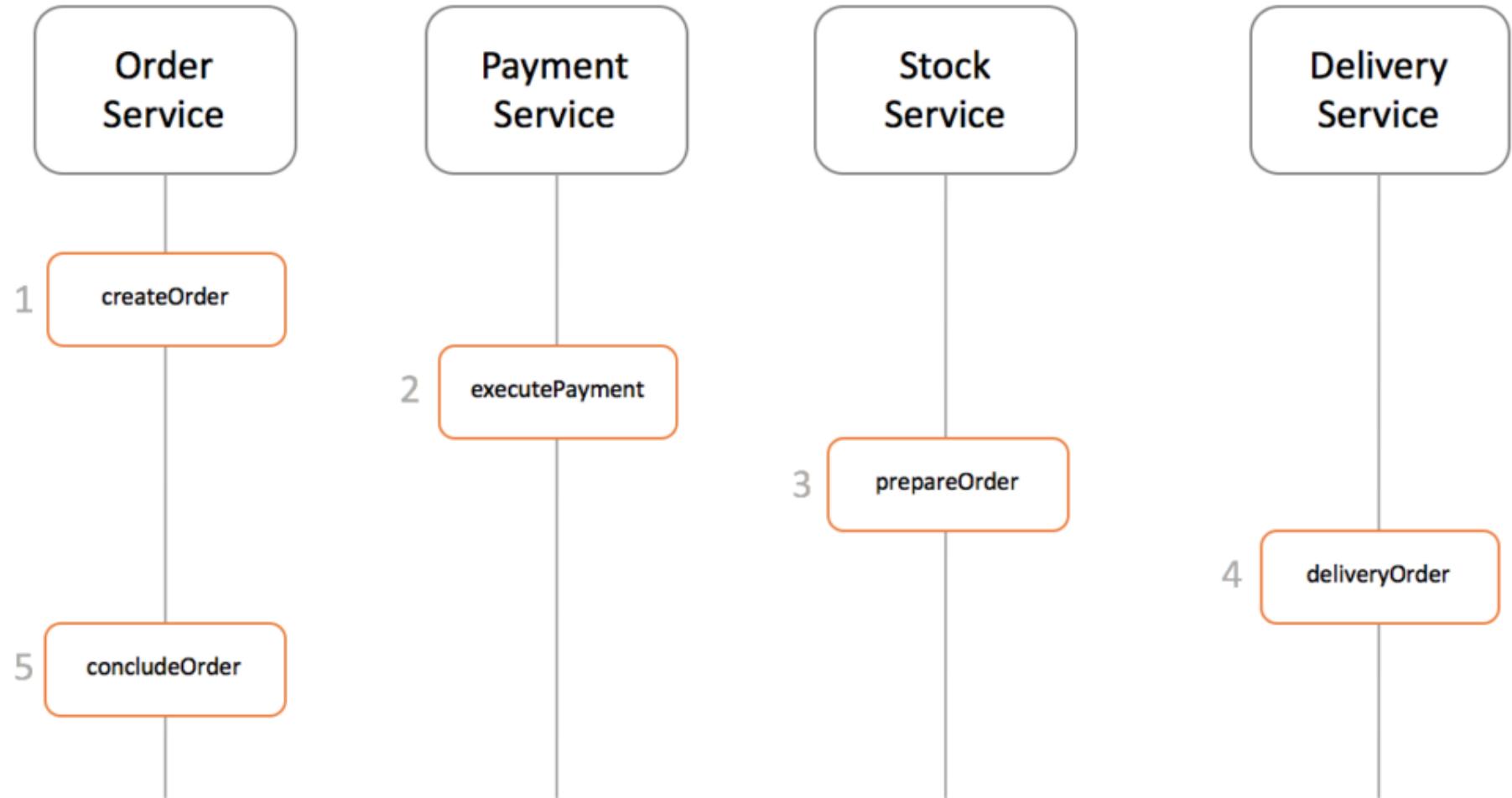


## Proposed Solutions to Handle Challenges

- **8. Monitoring** — Monitoring can be done using open-source tools like Prometheus in combination with Grafana by creating gauge and matrices, Kubernetes/OpenShift, Influx DB, Apigee, combined with Grafana, and Graphite.
- **9. DevOps Support** — Microservices deployment and support-related challenges can be addressed using state-of-the-art DevOps tools like GCP, Kubernetes, and OpenShift with Jenkins.
- **10. Fault Tolerance** — Netflix Hystrix can be used to break the circuit if there is no response from the API for the given SLA/ETA.



# Saga Design Pattern





# Saga Design Pattern

- Events/Choreography: When there is no central coordination, each service produces and listen to other service's events and decides if an action should be taken or not.
- Command/Orchestration: when a coordinator

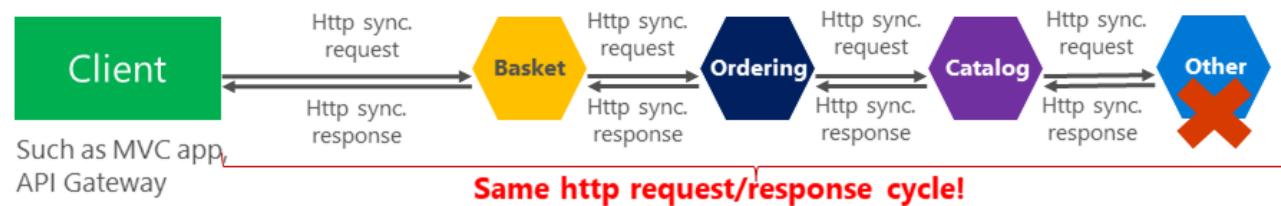


# Saga Design Pattern

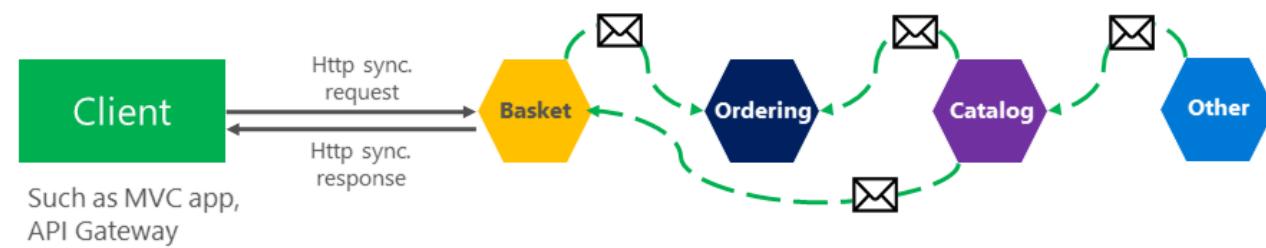
Synchronous vs. async communication across microservices

## Anti-pattern

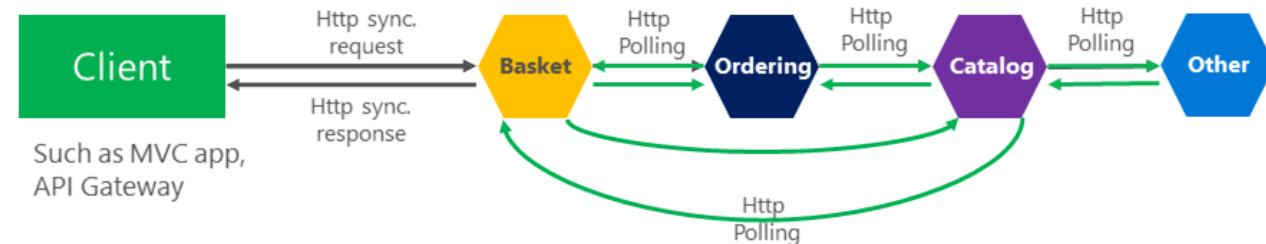
**Synchronous**  
all request/response cycle



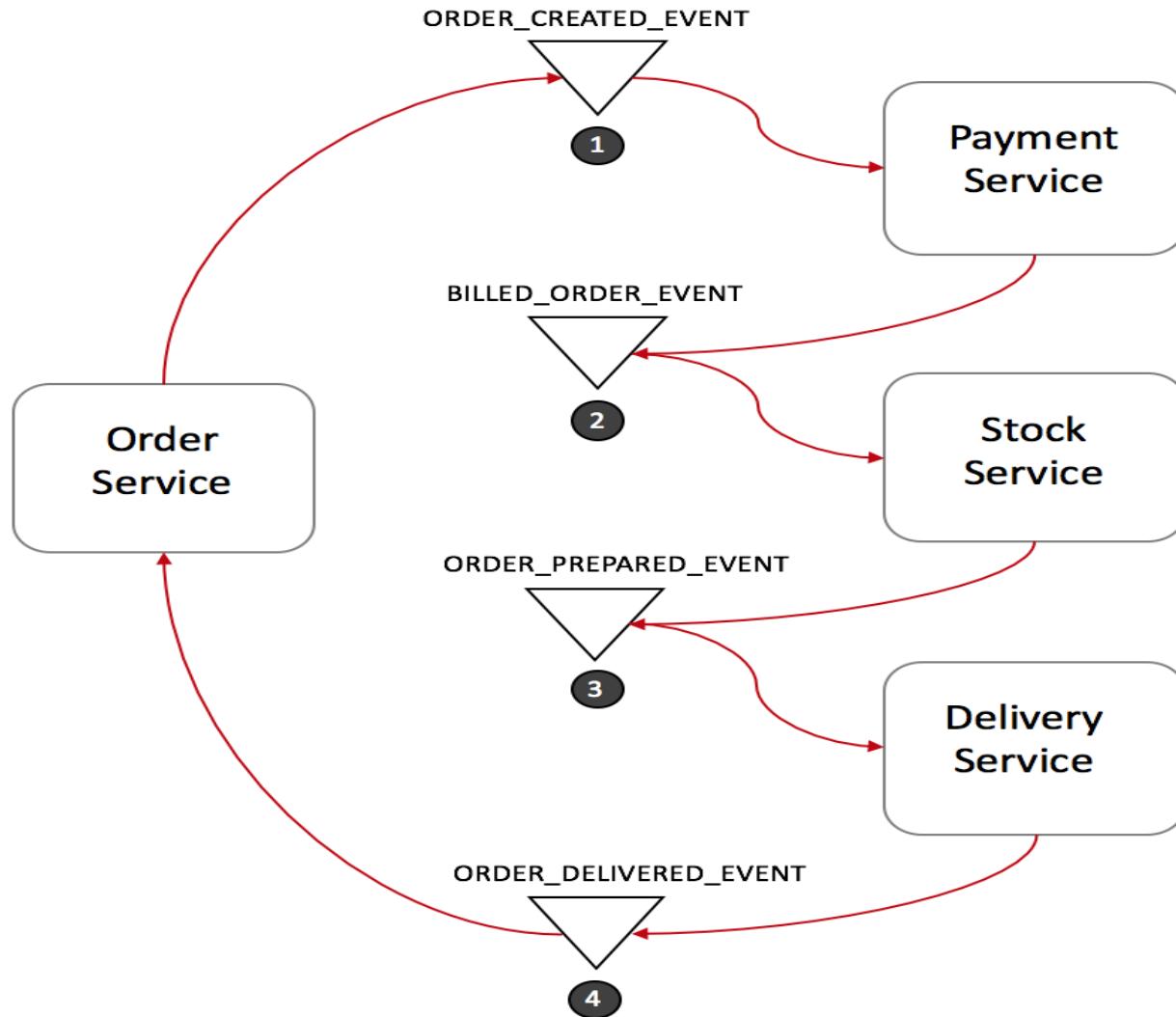
**Asynchronous**  
Comm. across internal microservices  
(EventBus: like **AMQP**)



**"Asynchronous"**  
Comm. across internal microservices  
(Polling: **Http**)

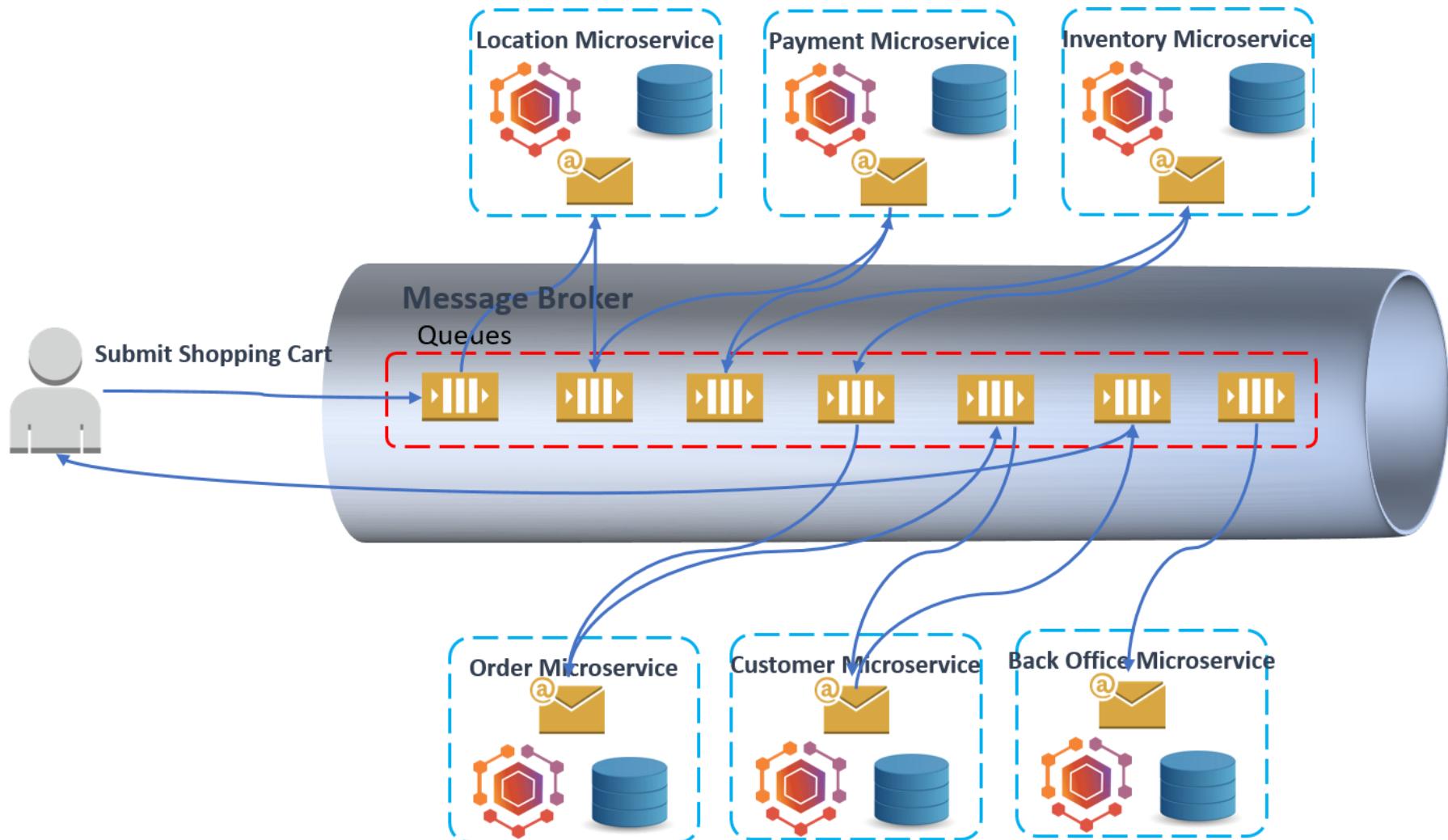


# Event Choreography





# Event Choreography



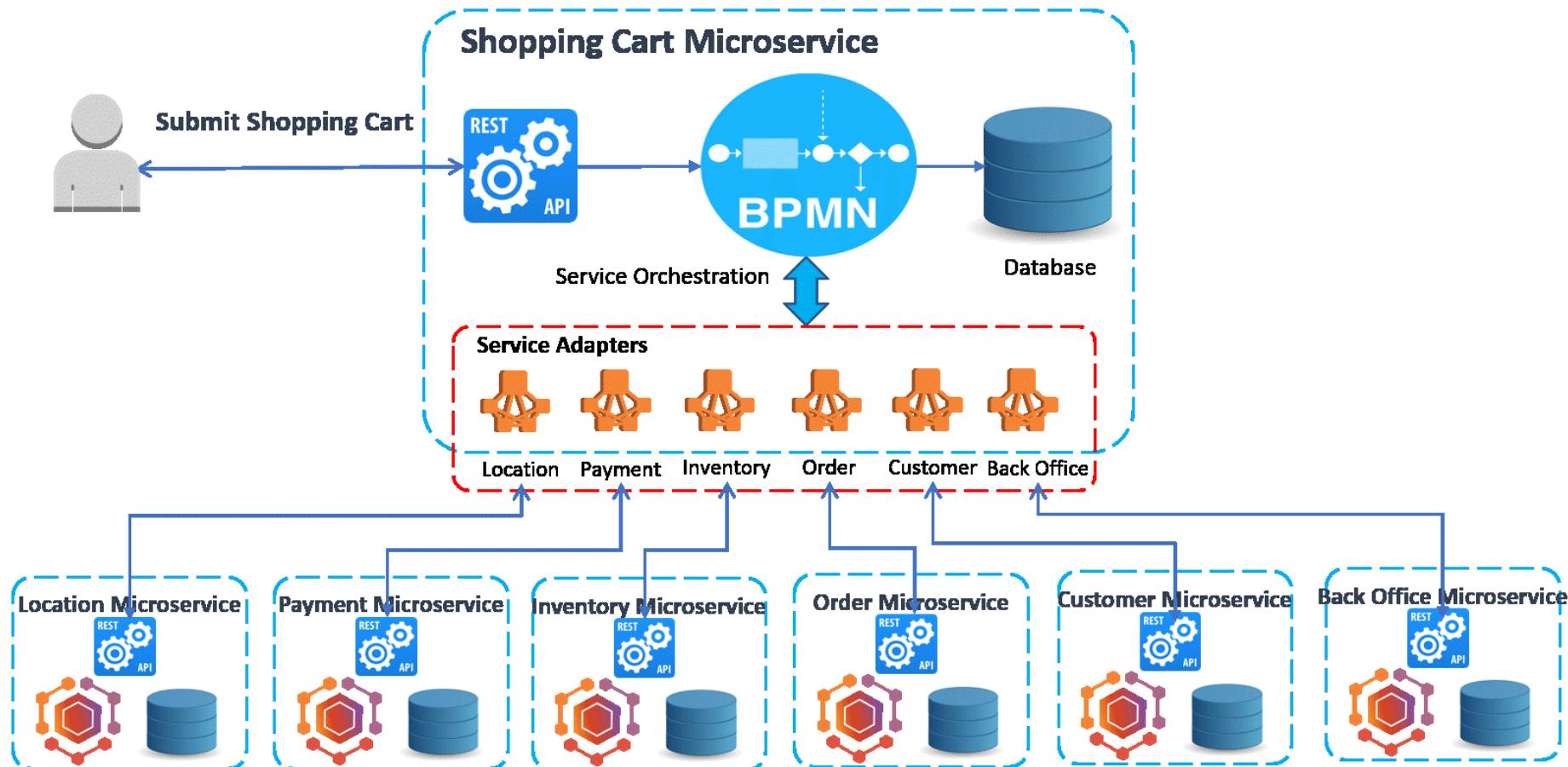
# Event-Driven Service Choreography



- Difficult to maintain: Due to the decentralized solution, the business flow is spreading across multiple services. Any business change to the process flow will lead to changes in multiple services.
- Difficult to manage: The runtime state of the process instance is stored separately.
- Difficult to rollback: The business process is choreographed by multiple services which leave the transaction's ACID becomes extremely difficult.



# Event Choreography





# Centralized Service Orchestration

- Easy to maintain: the business flow is modeled as graphical BPMN process in a centralized orchestration microservice. The standard Business Process Model and Notation (BPMN) grants the businesses with the capability of understanding their internal business procedures in a graphical notation. Furthermore, organizations can better communicate and report these procedures in a standard manner. Any further changes to the process flow will be mitigated by implementing the workflow diagrams.
- Easy to manage: BPMN process engine keeps track of all process instances, their state, audit and statistics information.
- Easy to rollback: Transaction's ACID can be guaranteed by the compensation flow as the BPMN out-of-the-box feature.

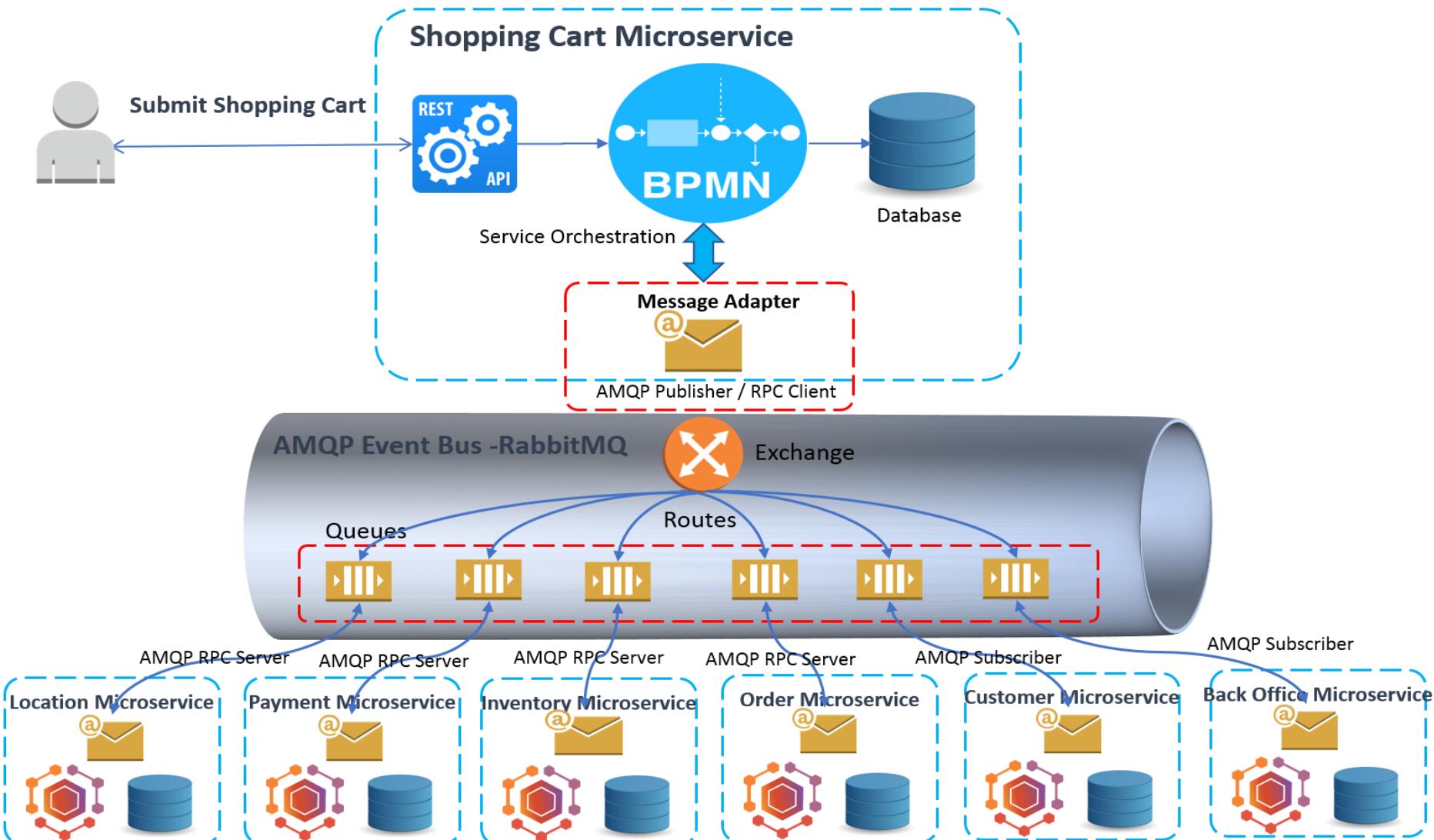


# Centralized Service Orchestration

- Tight-coupling: The orchestration pattern must build and maintain a point-to-point connection between the services. Point-to-point means that one service calls the API of another service which results in a mesh of communication paths between all services. Integrating, changing or removing services from the service repository will be a hard task since you must be aware of each connection between the services.
- Complexity in building service adapters: Orchestration service need to develop adapters to the peer services, which must maintain all details of the service communications, such as service location, service interfaces, and data model translation. Whenever a peer service changes, the adapter will be impacted.



# Event Choreography



# Introduction to Containers and Docker



- Containerization is an approach to software development in which an application or service, its dependencies, and its configuration (abstracted as deployment manifest files) are packaged together as a container image.
- The containerized application can be tested as a unit and deployed as a container image instance to the host operating system (OS).

# Introduction to Containers and Docker

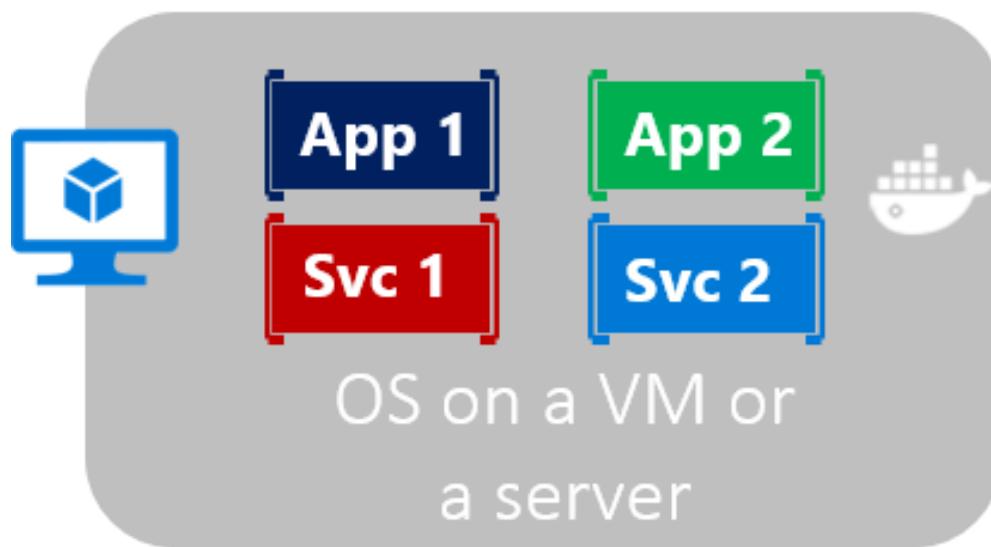


- Just as shipping containers allow goods to be transported by ship, train, or truck regardless of the cargo inside, software containers act as a standard unit of software deployment that can contain different code and dependencies.
- Containers also isolate applications from each other on a shared OS. Containerized applications run on top of a container host that in turn runs on the OS (Linux or Windows).
- Containers therefore have a significantly smaller footprint than virtual machine (VM) images.

# Multiple containers running on a container host



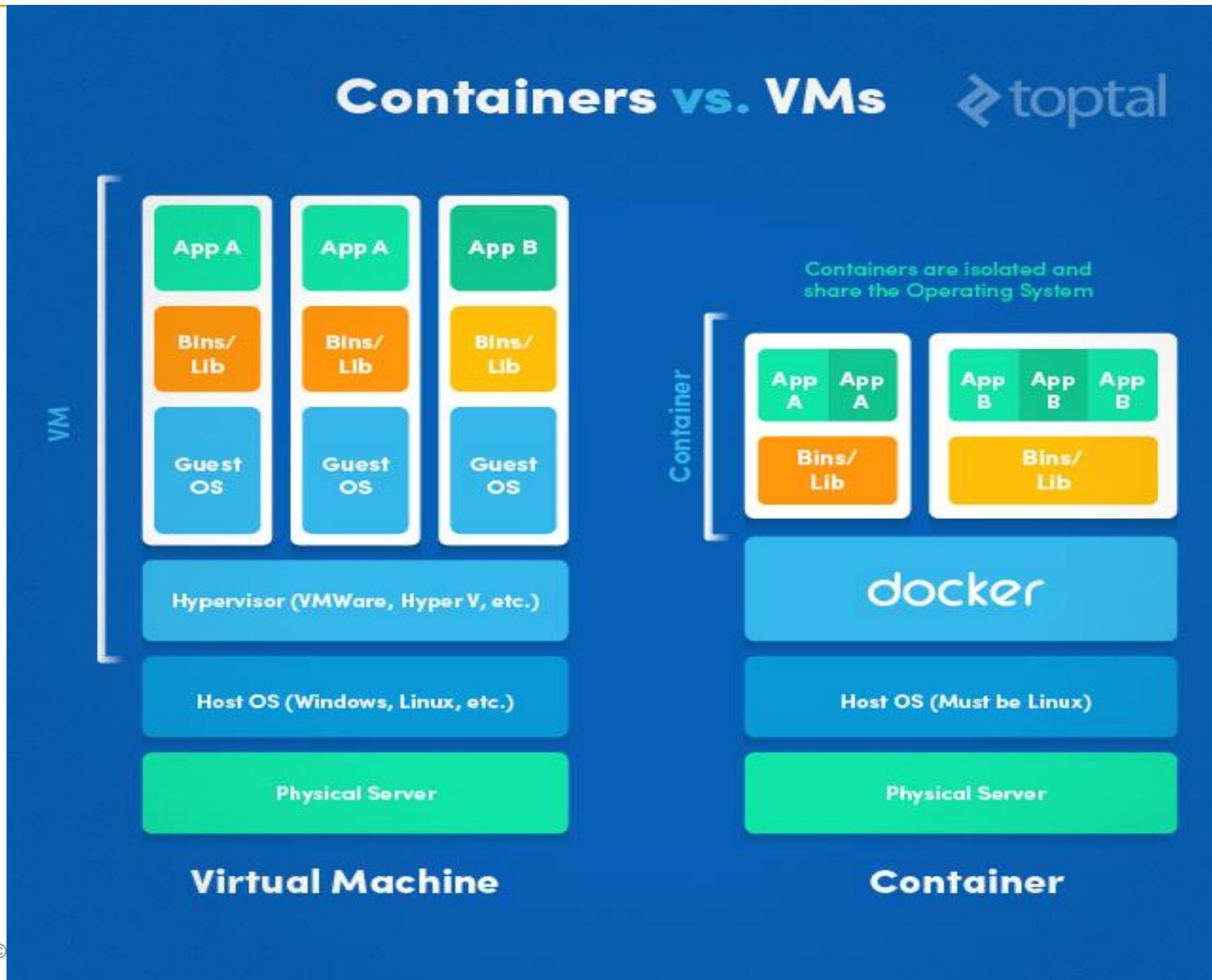
## Docker Host





- Docker is an excellent tool for managing and deploying microservices.
- Each microservice can be further broken down into processes running in separate Docker containers, which can be specified with Dockerfiles and Docker Compose configuration files.
- Combined with a provisioning tool such as Kubernetes, each microservice can then be easily deployed, scaled, and collaborated on by a developer team.
- Specifying an environment in this way also makes it easy to link microservices together to form a larger application.

# Docker

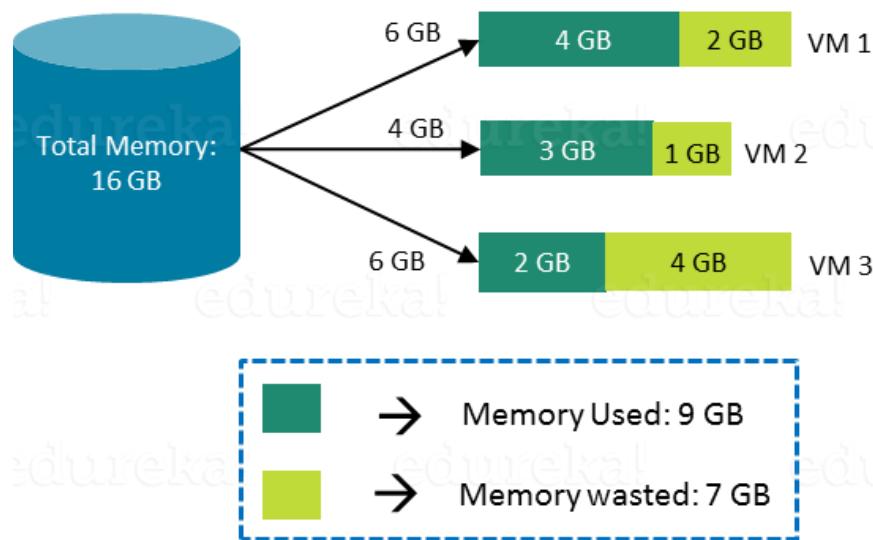




## VM vs Container

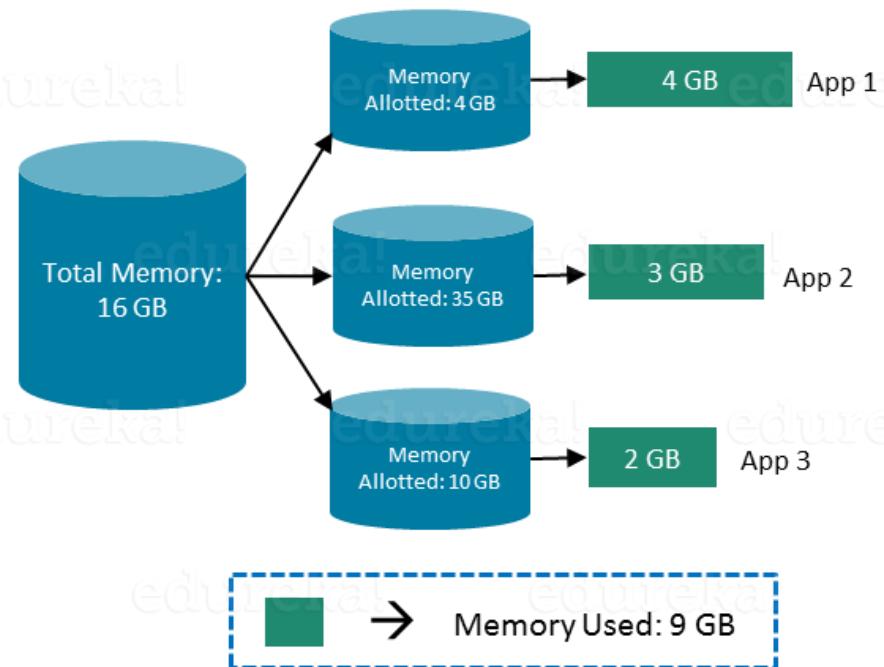
- Virtual machines include the application, the required libraries or binaries, and a full guest operating system. Full virtualization requires more resources than containerization.
- Containers include the application and all its dependencies.
- However, they share the OS kernel with other containers, running as isolated processes in user space on the host operating system. (Except in Hyper-V containers, where each container runs inside of a special virtual machine per container.)

## In case of Virtual Machines



7 Gb of Memory is blocked and cannot be allotted to a new VM

## In case of Docker



Only 9 GB memory utilized;  
7 GB can be allotted to a new Container



# Difference between VM and Container

- Applications running in virtual machines, apart from the hypervisor, require a full instance of the operating system and any supporting libraries.
- Containers, on the other hand, share the operating system with the host.
- Hypervisor is comparable to the container engine (represented as Docker on the image) in a sense that it manages the lifecycle of the containers.
- The important difference is that the processes running inside the containers are just like the native processes on the host, and do not introduce any overheads associated with hypervisor execution.
- Additionally, applications can reuse the libraries and share the data between containers.



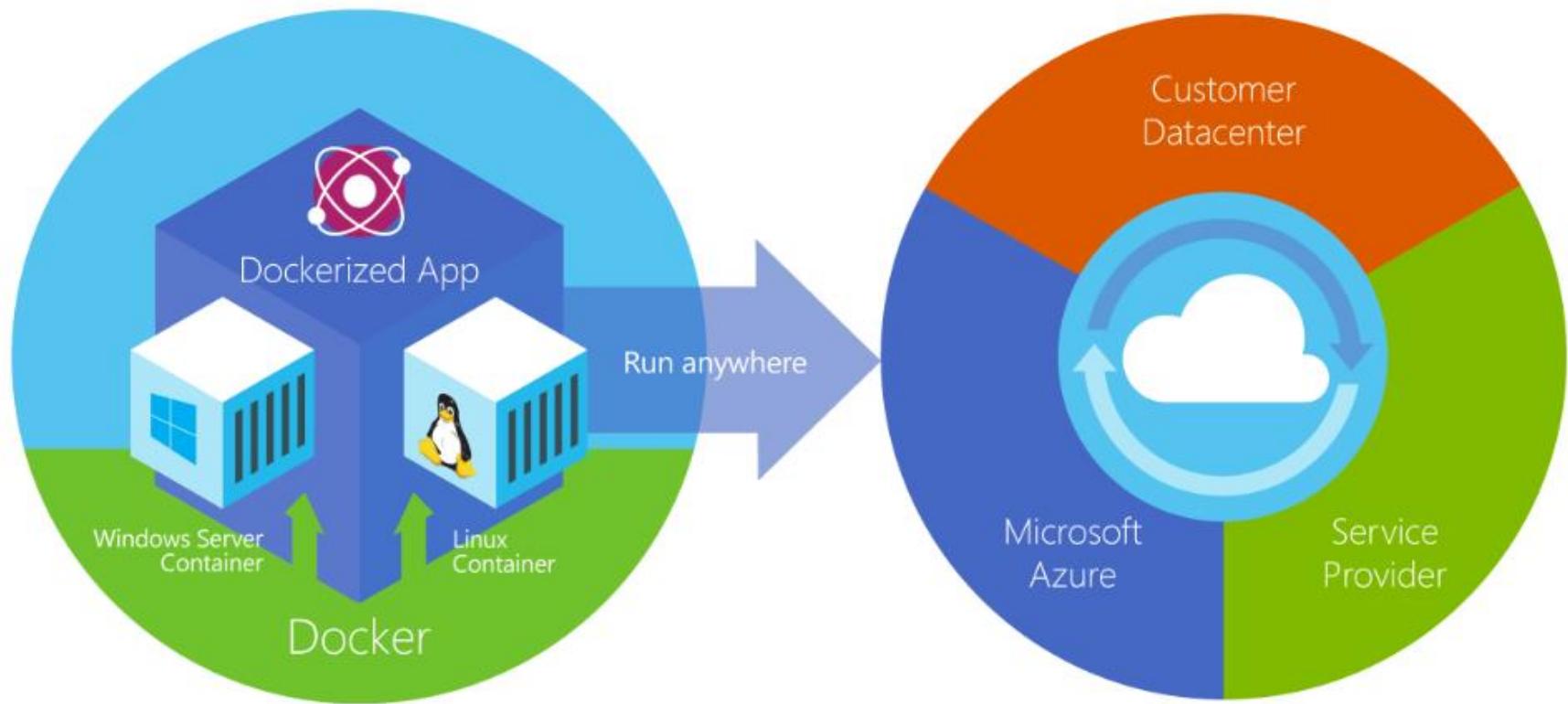
# What is Docker

---

- What is Docker?
- At its heart, Docker is software which lets you create an *image* and then run instances of that image in a *container*.
- Docker maintains a vast repository of images, called the Docker Hub which you can use as starting points or as free storage for your own images.
- You can install Docker, choose an image you'd like to use, then run an instance of it in a container.



# What is Docker





# What is Docker

---

- Docker image containers can run natively on Linux and Windows.
- However, Windows images can run only on Windows hosts and Linux images can run on Linux hosts and Windows hosts (using a Hyper-V Linux VM, so far), where host means a server or a VM.

# Dockerfile, Docker Image And Docker Container



- **Container image:** A package with all the dependencies and information needed to create a container.
- An image includes all the dependencies (such as frameworks) plus deployment and execution configuration to be used by a container runtime.
- Usually, an image derives from multiple base images that are layers stacked on top of each other to form the container's filesystem.
- An image is immutable once it has been created.
- A Docker Image is created by the sequence of commands written in a file called as Dockerfile.
- **Dockerfile:** A text file that contains instructions for how to build a Docker image. It's like a batch script, the first line states the base image to begin with and then follow the instructions to install required programs, copy files and so on, until you get the working environment you need.

# Dockerfile, Docker Image And Docker Container



- **Build:** The action of building a container image based on the information and context provided by its Dockerfile, plus additional files in the folder where the image is built. You can build images with the Docker **docker build** command.
- **Container:** An instance of a Docker image. A container represents the execution of a single application, process, or service.
- It consists of the contents of a Docker image, an execution environment, and a standard set of instructions.
- When scaling a service, you create multiple instances of a container from the same image. Or a batch job can create multiple containers from the same image, passing different parameters to each instance.

# Dockerfile, Docker Image And Docker Container



- **Volumes:** Offer a writable filesystem that the container can use. Since images are read-only but most programs need to write to the filesystem, volumes add a writable layer, on top of the container image, so the programs have access to a writable filesystem.
- The program doesn't know it is accessing a layered filesystem, it is just the filesystem as usual. Volumes live in the host system and are managed by Docker.
- **Tag:** A mark or label you can apply to images so that different images or versions of the same image (depending on the version number or the target environment) can be identified.

# Dockerfile, Docker Image And Docker Container



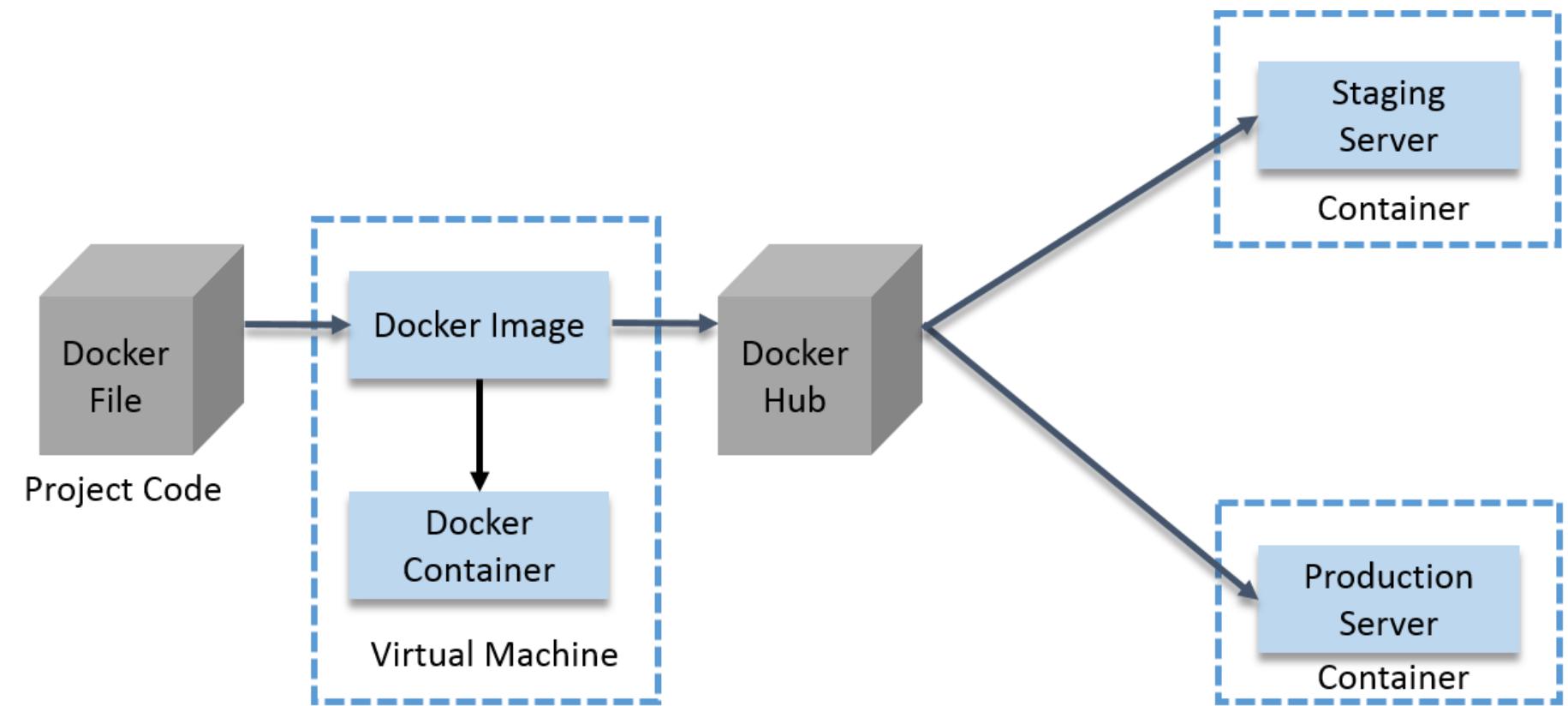
- **Repository (repo):** A collection of related Docker images, labeled with a tag that indicates the image version.
- Some repos contain multiple variants of a specific image, such as an image containing SDKs (heavier), an image containing only runtimes (lighter), etc. Those variants can be marked with tags.
- A single repo can contain platform variants, such as a Linux image and a Windows image.

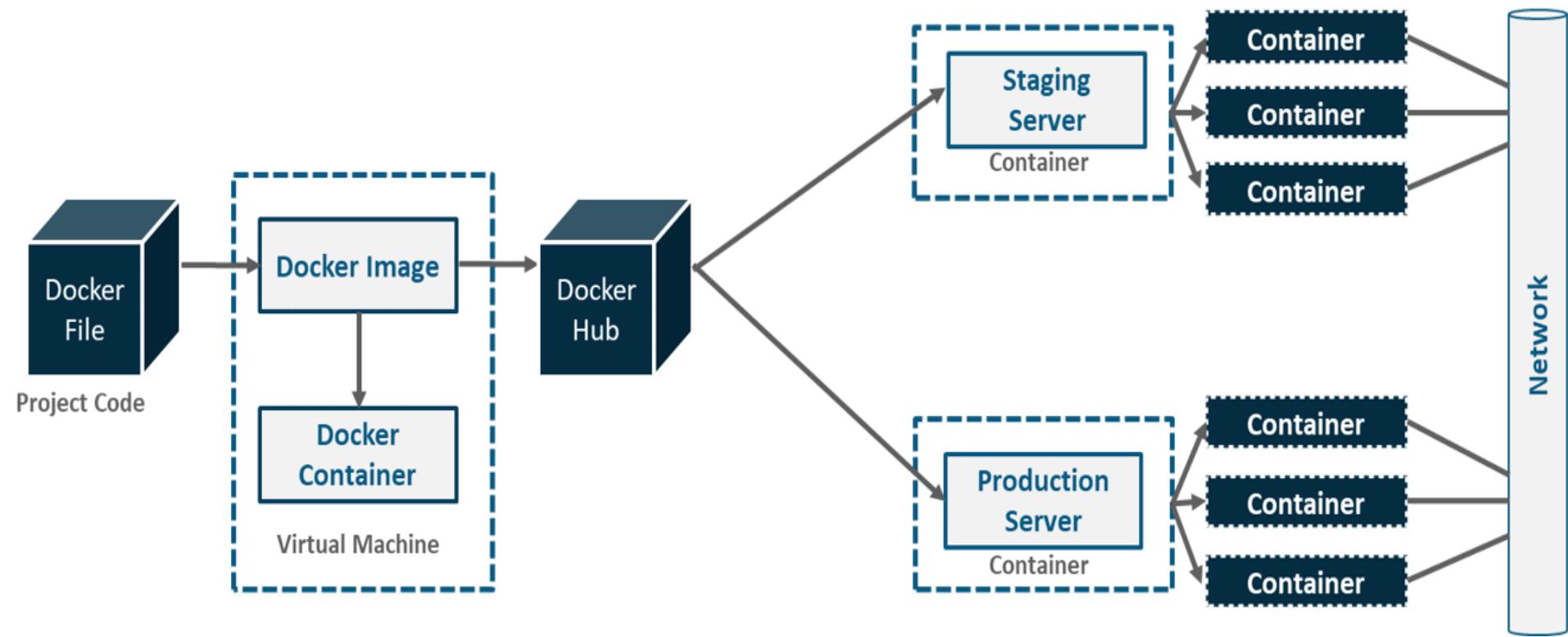
# Dockerfile, Docker Image And Docker Container



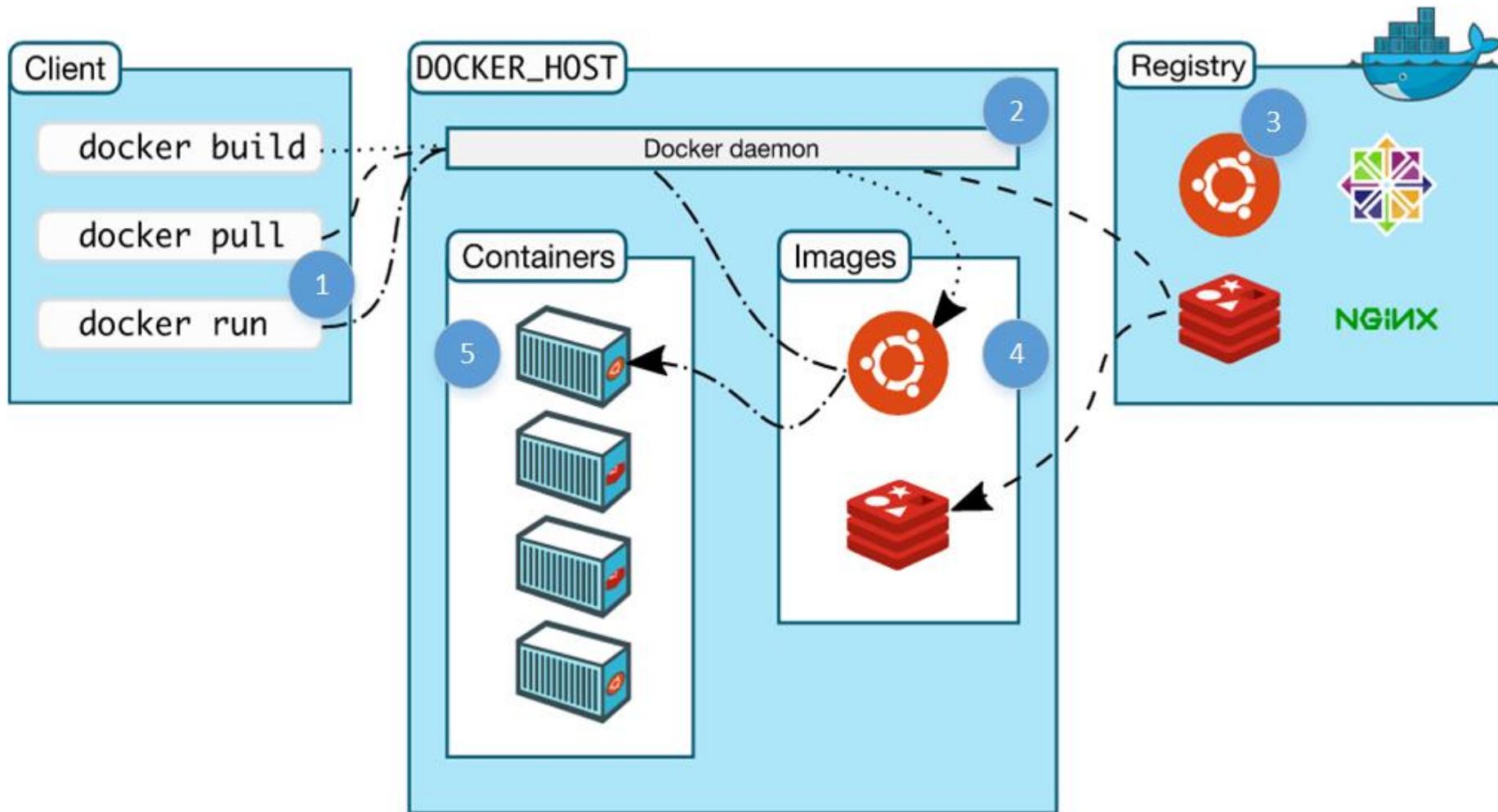
- **Registry:** A service that provides access to repositories. The default registry for most public images is Docker Hub(owned by Docker as an organization).
- A registry usually contains repositories from multiple teams. Companies often have private registries to store and manage images they've created. Azure Container Registry is another example.
- **Docker Hub:** A public registry to upload images and work with them. Docker Hub provides Docker image hosting, public or private registries, build triggers and web hooks, and integration with GitHub and Bitbucket.





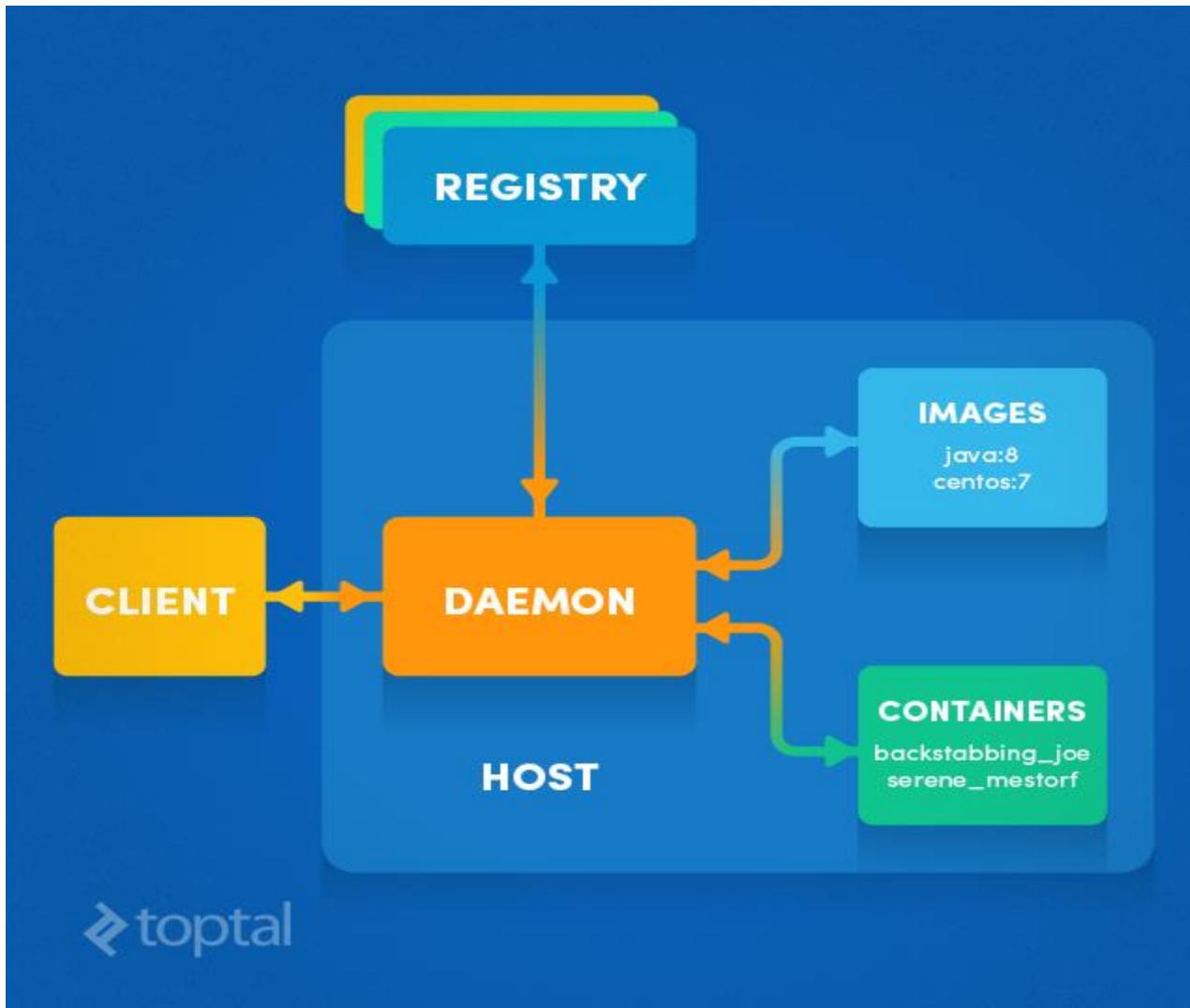


# Docker





# Docker Architecture



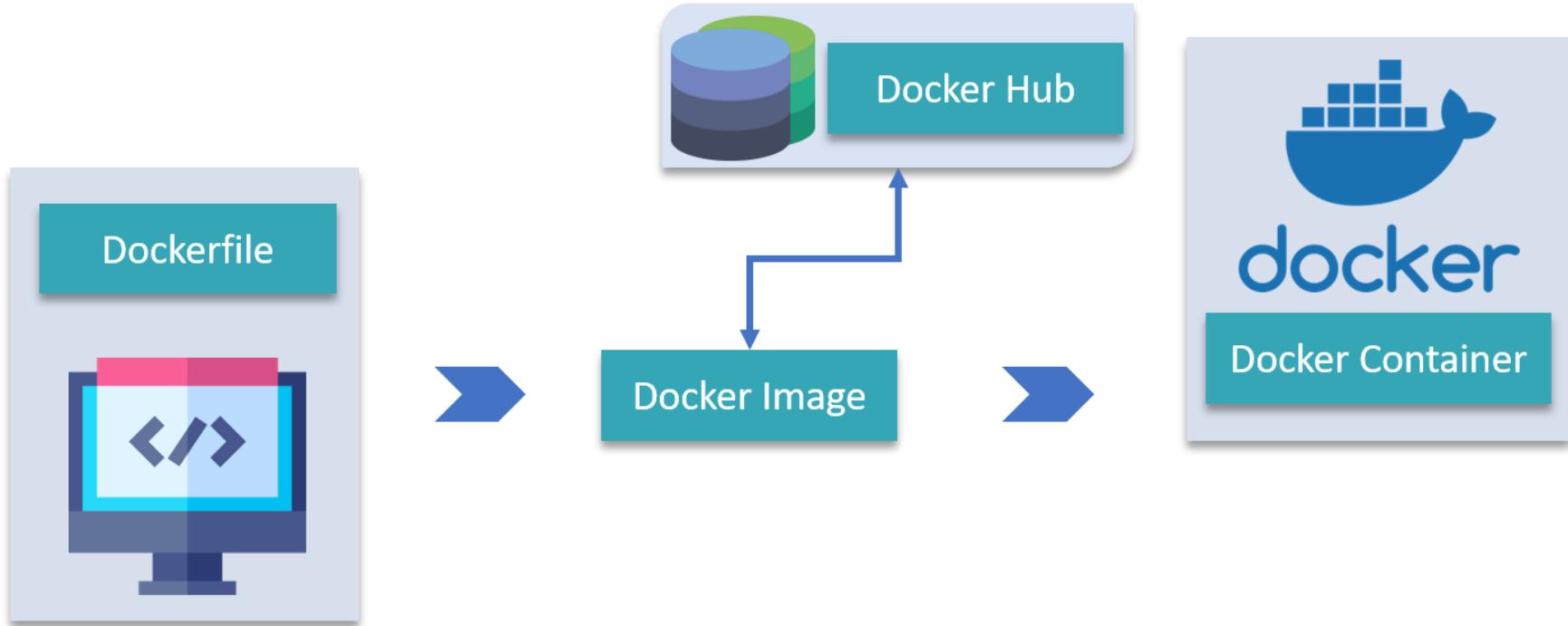
 toptal

## Docker Hub:



- Docker Hub is like GitHub for Docker Images. It is basically a cloud registry where you can find Docker Images uploaded by different communities, also you can develop your own image and upload on Docker Hub, but first, you need to create an account on DockerHub.

# Docker Hub





# Docker Architecture

- It consists of a Docker Engine which is a client-server application with three major components:
- A server which is a type of long-running program called a daemon process (the docker command).
- A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
- A command line interface (CLI) client (the docker command).
- The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI.



# Docker Architecture

- By default, the main registry is the Docker Hub which hosts public and official images.
- Organizations can also host their private registries if they desire.
- Images can be downloaded from registries explicitly (`docker pull imageName`) or implicitly when starting a container. Once the image is downloaded it is cached locally.
- Containers are the instances of images - they are the living thing. There could be multiple containers running based on the same image.



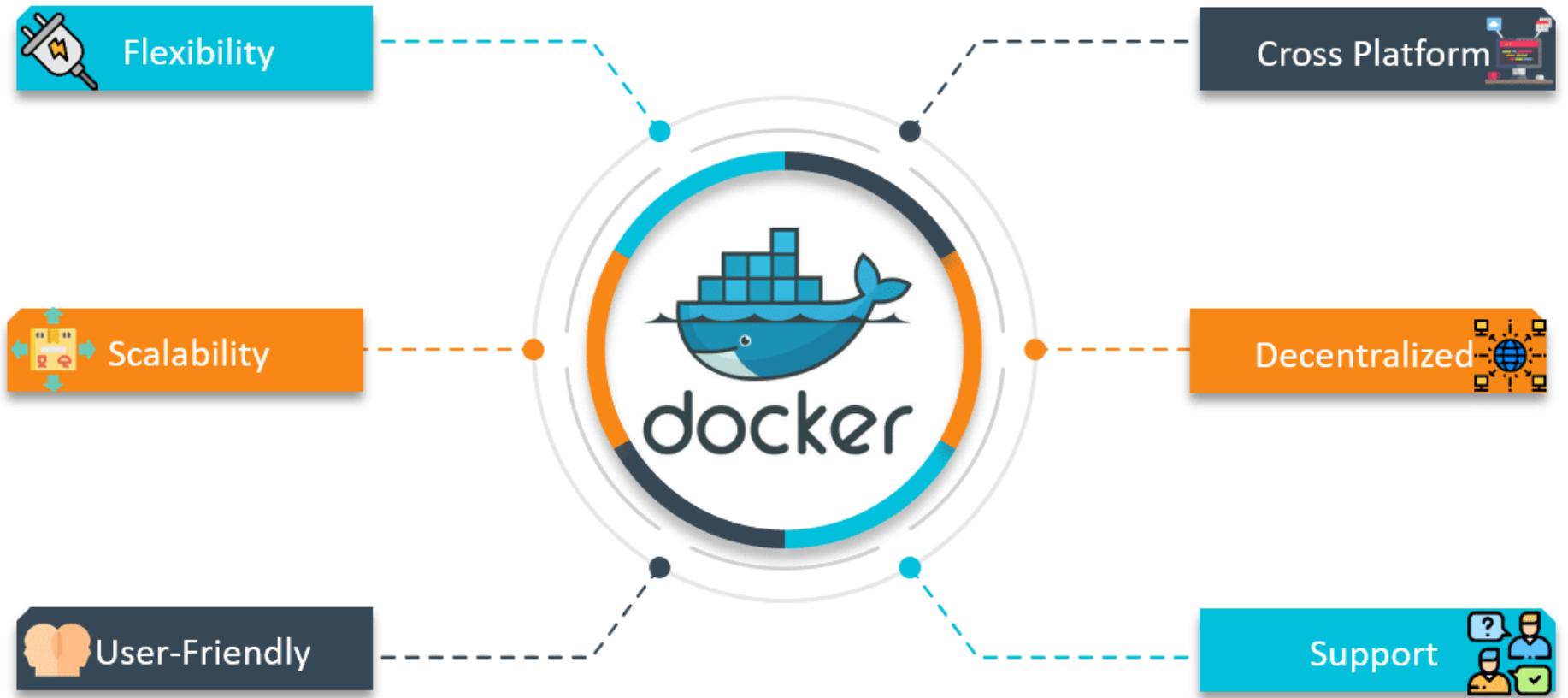
# Docker Architecture

---

- At the center, there is the Docker daemon responsible for creating, running, and monitoring containers.
- It also takes care of building and storing images.
- Finally, on the left-hand side there is a Docker client. It talks to the daemon via HTTP.
- Unix sockets are used when on the same machine, but remote management is possible via HTTP based API.



# Goals of Docker Networking



# Choosing Between .NET Core and .NET Framework for Docker Containers



- There are two supported frameworks for building server-side containerized Docker applications with .NET: .NET Framework and .NET Core.
- They share many .NET platform components, and you can share code across the two.
- However, there are fundamental differences between them, and which framework you use will depend on what you want to accomplish.



## General Guidance

---

- You should use .NET Core, with Linux or Windows Containers, for your containerized Docker server application when:
- You have cross-platform needs. For example, you want to use both Linux and Windows Containers.
- Your application architecture is based on micro services.
- You need to start containers fast and want a small footprint per container to achieve better density or more containers per hardware unit in order to lower your costs.



## General Guidance

---

- You should use .NET Framework for your containerized Docker server application when:
  - Your application currently uses .NET Framework and has strong dependencies on Windows.
  - You need to use Windows APIs that are not supported by .NET Core.
  - You need to use third-party .NET libraries or NuGet packages that are not available for .NET Core.

# Decision table: .NET frameworks to use for Docker



Microservices

Architecture / App Type	Linux containers	Windows Containers
Microservices on containers	.NET Core	.NET Core
Monolithic app	.NET Core	.NET Framework .NET Core
Best-in-class performance and scalability	.NET Core	.NET Core
Windows Server legacy app ("brown-field") migration to containers	--	.NET Framework
New container-based development ("green-field")	.NET Core	.NET Core

# Decision table: .NET frameworks to use for Docker

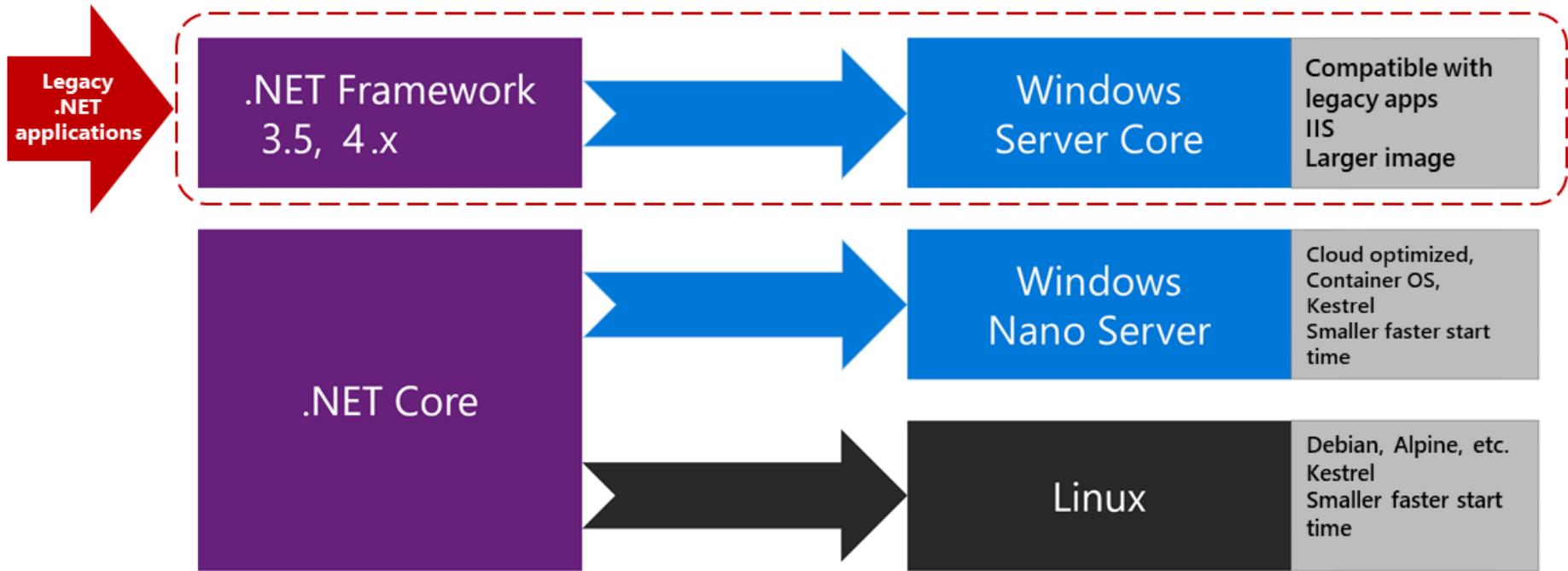


ASP.NET Core	.NET Core	.NET Core (recommended) .NET Framework
ASP.NET 4 (MVC 5, Web API 2, and Web Forms)	--	.NET Framework
SignalR services	.NET Core 2.1 or higher version	.NET Framework .NET Core 2.1 or higher version
WCF, WF, and other legacy frameworks	WCF in .NET Core (only the WCF client library)	.NET Framework WCF in .NET Core (only the WCF client library)
Consumption of Azure services	.NET Core (eventually all Azure services will provide client SDKs for .NET Core)	.NET Framework .NET Core (eventually all Azure services will provide client SDKs for .NET Core)

# What OS to target with .NET containers



## What OS to target with .NET containers



# What OS to target with .NET containers



Image	Comments
microsoft/dotnet:2.2-runtime	.NET Core 2.2 multi-architecture: Supports Linux and Windows Nano Server depending on the Docker host.
microsoft/dotnet:2.2-aspnetcore-runtime	ASP.NET Core 2.2 multi-architecture: Supports Linux and Windows Nano Server depending on the Docker host. The aspnetcore image has a few optimizations for ASP.NET Core.
microsoft/dotnet:2.2-aspnetcore-runtime-alpine	.NET Core 2.2 runtime-only on Linux Alpine distro
microsoft/dotnet:2.2-aspnetcore-runtime-nanoserver-1803	.NET Core 2.2 runtime-only on Windows Nano Server (Windows Server version 1803)



# Official .NET Docker images

- The Official .NET Docker images are Docker images created and optimized by Microsoft.
- They are publicly available in the Microsoft repositories on Docker Hub.
- Each repository can contain multiple images, depending on .NET versions, and depending on the OS and versions (Linux Debian, Linux Alpine, Windows Nano Server, Windows Server Core, etc.).

# Architecting container and microservice-based applications

---



- **Container design principles**
- In the container model, a container image instance represents a single process.
- By defining a container image as a process boundary, you can create primitives that can be used to scale the process or to batch it.
- When you design a container image, an ENTRYPOINT definition will be in the Dockerfile.
- This defines the process whose lifetime controls the lifetime of the container.
- When the process completes, the container lifecycle ends. Containers might represent long-running processes like web servers, but can also represent short-lived processes like batch jobs, which formerly might have been implemented as Azure WebJobs.

# Architecting container and microservice-based applications

---



- **Container design principles**
- If the process fails, the container ends, and the orchestrator takes over.
- If the orchestrator was configured to keep five instances running and one fails, the orchestrator will create another container instance to replace the failed process.
- In a batch job, the process is started with parameters. When the process completes, the work is complete..

# Architecting container and microservice-based applications

---

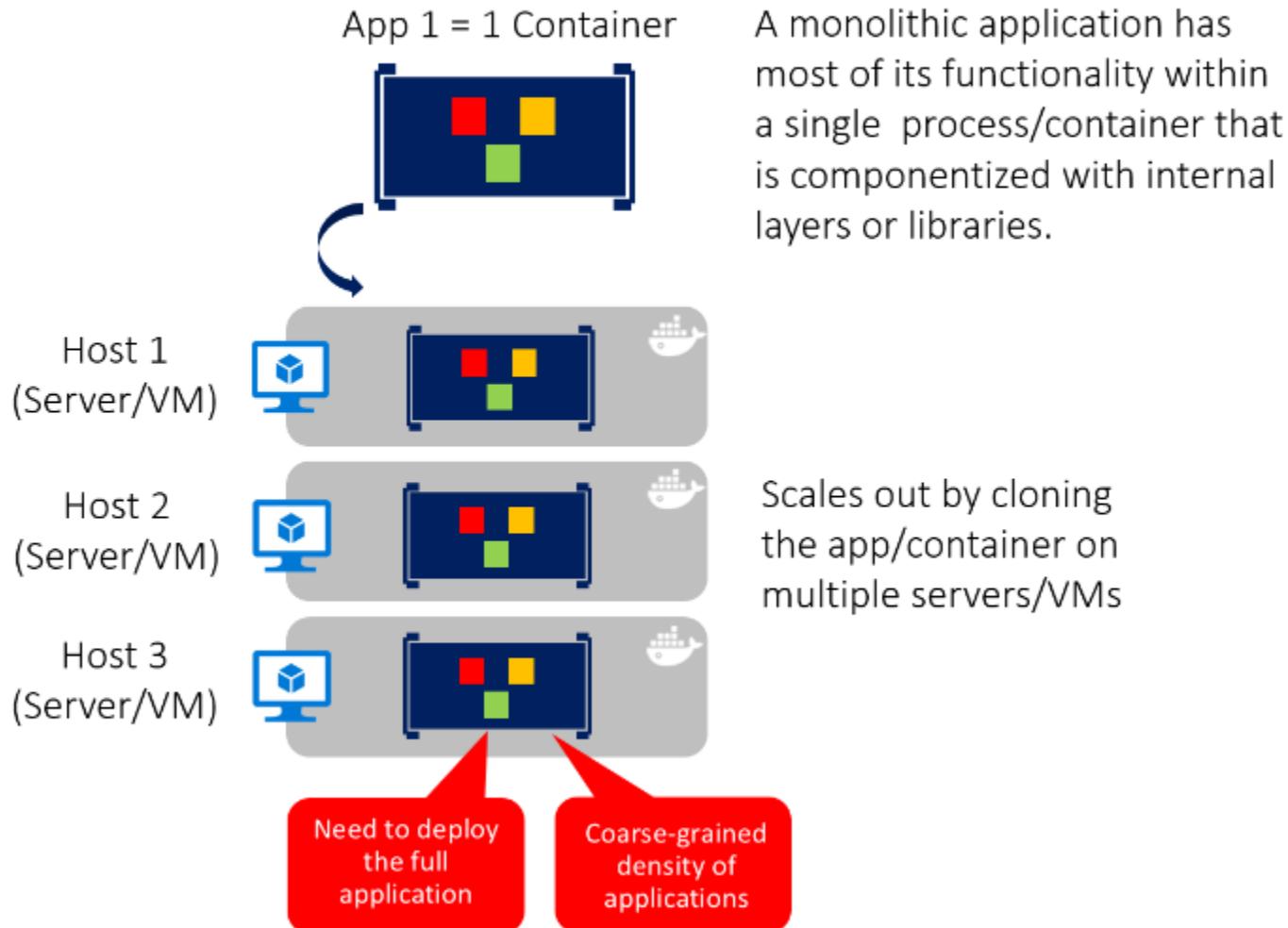


- **Container design principles**
- Scenario where we want multiple processes running in a single container.
- For that scenario, since there can be only one entry point per container, we need to run a script within the container that launches as many programs as needed.
- For example, you can use Supervisor or a similar tool to take care of launching multiple processes inside a single container.

# Containerizing monolithic applications



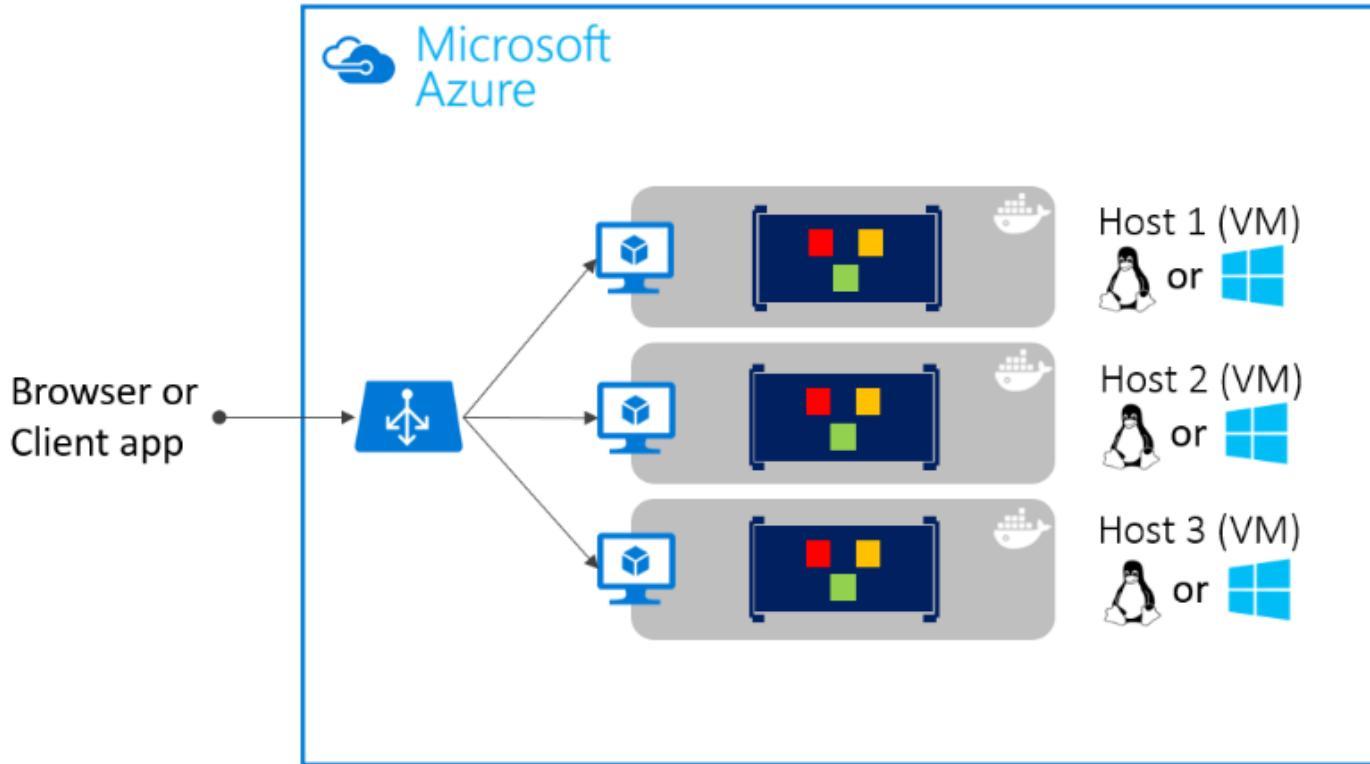
## Monolithic Containerized application



# Containerizing monolithic applications



## Architecture in Docker infrastructure for monolithic applications





# Docker Compose

- Docker Compose is basically used to run multiple Docker Containers as a single server. Let me give you an example:
- Suppose if I have an application which requires WordPress, Maria DB and PHP MyAdmin. I can create one file which would start both the containers as a service without the need to start each one separately. It is really useful especially if you have a microservice architecture.



# Docker Container

Column	Description
Container ID	The unique ID of the container. It is a SHA-256.
Image	The name of the container image from which this container is instantiated.
Status	The status of the container (created, restarting, running, removing, paused, exited, or dead).
Ports	The list of container ports that have been mapped to the host.
Names	The name assigned to this container (multiple names are possible).



# Docker Client and Docker Engine

- **Docker Client** : This is the utility we use when we run any docker commands e.g. docker run (docker container run) , docker images , docker ps etc. It allows us to run these commands which a human can easily understand.
- **Docker Daemon/Engine**: This is the part which does rest of the magic and knows how to talk to the kernel, makes the system calls to create, operate and manage containers, which we as users of docker dont have to worry about.

# Azure Kubernetes



A screenshot of a Microsoft Edge browser window. The address bar shows "Not secure | 40.118.245.185". The toolbar has various icons for apps, search, and navigation. Several tabs are open, including "Insert title here", "Empire", "New Tab", "How to use Asserti...", "Browser Automatio...", "node.js - How can I...", "Freelancer-dev-810...", "Courses", and "New Tab".

webappimage Home Privacy

Use this space to summarize your privacy and cookie use policy. [Learn More.](#)

Accept

## Welcome

Learn about [building Web apps with ASP.NET Core.](#)



# Azure Kubernetes

```
F:\dotnetfidelity2019>kubectl describe pod ey-api-5f685855d4-tp7kp
Name:           ey-api-5f685855d4-tp7kp
Namespace:      default
Priority:       0
PriorityClassName: <none>
Node:           aks-agentpool-16062465-1/10.240.0.6
Start Time:     Tue, 14 May 2019 05:19:43 +0530
Labels:         app=ey-api
                pod-template-hash=5f685855d4
Annotations:    <none>
Status:         Running
IP:             10.244.1.6
Controlled By: ReplicaSet/ey-api-5f685855d4
Containers:
  ey-api:
    Container ID:  docker://086479851aa64a08f53c75ca1dadbd63468762420e8707edfacc3ac5294adf12
    Image:          eycontainerregistry2019.azurecr.io/eydemowebapp:latest
    Image ID:       docker-pullable://eycontainerregistry2019.azurecr.io/eydemowebapp@sha256:9fc1c84d36510d53adae37df6253e166b076f0b914c3ce876508986000efe26e
    Port:          80/TCP
```



# Azure Kubernetes

```
F:\dotnetfidelity2019>kubectl create secret docker-registry acr-auth --docker-server eycontainerregistry2019.azurecr.io --docker-username=eycontainerregistry2019 --docker-password=Z2m1pqVuf1p1Y0iAEWiAVjoJ/CDckq9C --docker-email=Parameswaribala@gmail.com  
secret/acr-auth created  
  
F:\dotnetfidelity2019>kubectl delete --all services --namespace=default  
service "ey-api" deleted  
service "kubernetes" deleted  
  
F:\dotnetfidelity2019>kubectl delete --all deployments --namespace=default  
deployment.extensions "ey-api" deleted  
  
F:\dotnetfidelity2019>kubectl delete --all services --namespace=default  
service "kubernetes" deleted  
  
F:\dotnetfidelity2019>kubectl get pods  
No resources found.  
  
F:\dotnetfidelity2019>kubectl create -f eydemodeployment.yml  
deployment.apps/ey-api created
```



# Azure Kubernetes

```
F:\Administrator>kubectl get svc
ey-api   LoadBalancer   10.0.108.206   <pending>      80:30472/TCP   2m
ey-api   LoadBalancer   10.0.108.206   40.118.245.185  80:30472/TCP   2m4s

F:\dotnetfidelity2019>kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
ey-api-5f685855d4-tp7kp   1/1     Running   0          12m

F:\dotnetfidelity2019>kubectl autoscale deployment ey-api --min=3 --max=3 --cpu-percent=80
horizontalpodautoscaler.autoscaling/ey-api autoscaled

F:\dotnetfidelity2019>kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
ey-api-5f685855d4-tp7kp   1/1     Running   0          14m

F:\dotnetfidelity2019>kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
ey-api-5f685855d4-18bsz   1/1     Running   0          61s
ey-api-5f685855d4-n4dh7   1/1     Running   0          61s
ey-api-5f685855d4-tp7kp   1/1     Running   0          15m

F:\dotnetfidelity2019>
```



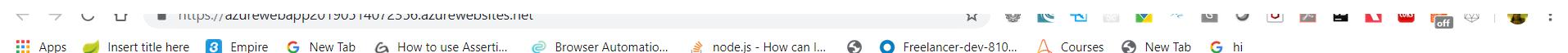
# Azure and ASPnetcore

---

- Update-Database
- Or
- Dotnet ef database update



# Azure and ASPnetcore



azurewebapp Home Privacy

Register Login

Use this space to summarize your privacy and cookie use policy. [Learn More.](#)

Accept

## Welcome

Learn about [building Web apps with ASP.NET Core](#).



# Azure RBAC

---

- Access management for cloud resources is a critical function for any organization that is using the cloud.
- Role-based access control (RBAC) helps you manage who has access to Azure resources, what they can do with those resources, and what areas they have access to.
- RBAC is an authorization system built on Azure Resource Manager that provides fine-grained access management of Azure resources.

# What can I do with RBAC?



- Allow one user to manage virtual machines in a subscription and another user to manage virtual networks
- Allow a DBA group to manage SQL databases in a subscription
- Allow a user to manage all resources in a resource group, such as virtual machines, websites, and subnets
- Allow an application to access all resources in a resource group

# Best practice for using RBAC



- Using RBAC, you can segregate duties within your team and grant only the amount of access to users that they need to perform their jobs.
- Instead of giving everybody unrestricted permissions in your Azure subscription or resources, you can allow only certain actions at a particular scope.

# Best practice for using RBAC



	Reader	Resource-specific or custom role	Contributor	Owner
Subscription	Observers	Users managing resources		Admins
Resource group				
Resource		Automated processes		

# How RBAC works



- The way you control access to resources using RBAC is to create role assignments.
- This is a key concept to understand – it's how permissions are enforced.
- A role assignment consists of three elements: security principal, role definition, and scope.

# How RBAC works



- **Security principal**
- A *security principal* is an object that represents a user, group, service principal, or managed identity that is requesting access to Azure resources.



# How RBAC works



- User - An individual who has a profile in Azure Active Directory. You can also assign roles to users in other tenants.
- Group - A set of users created in Azure Active Directory. When you assign a role to a group, all users within that group have that role.
- Service principal - A security identity used by applications or services to access specific Azure resources. You can think of it as a *user identity* (username and password or certificate) for an application.
- Managed identity - An identity in Azure Active Directory that is automatically managed by Azure. You typically use managed identities when developing cloud applications to manage the credentials for authenticating to Azure services.

# Role definition

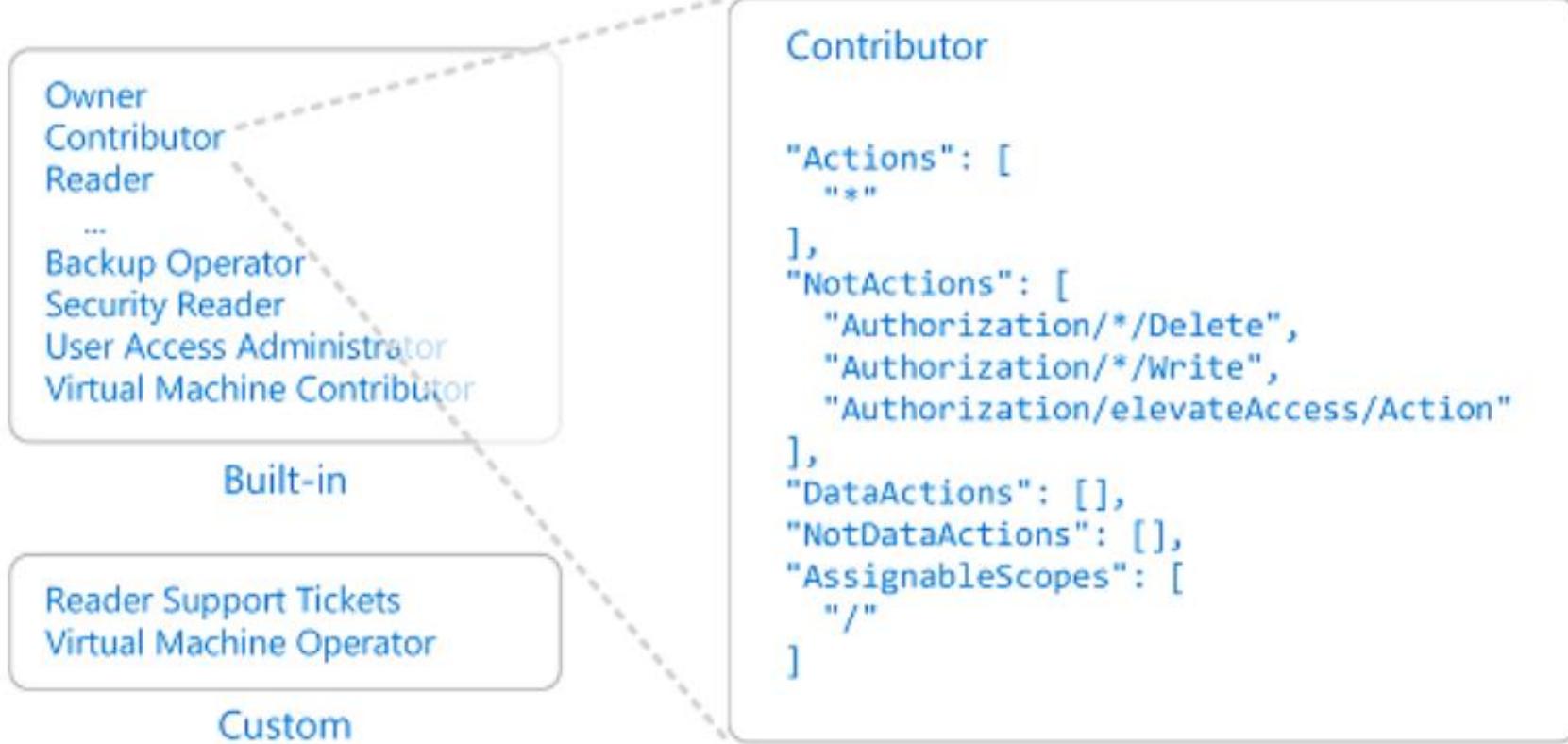


- A *role definition* is a collection of permissions.
- It's sometimes just called a *role*.
- A role definition lists the operations that can be performed, such as read, write, and delete.
- Roles can be high-level, like owner, or specific, like virtual machine reader.

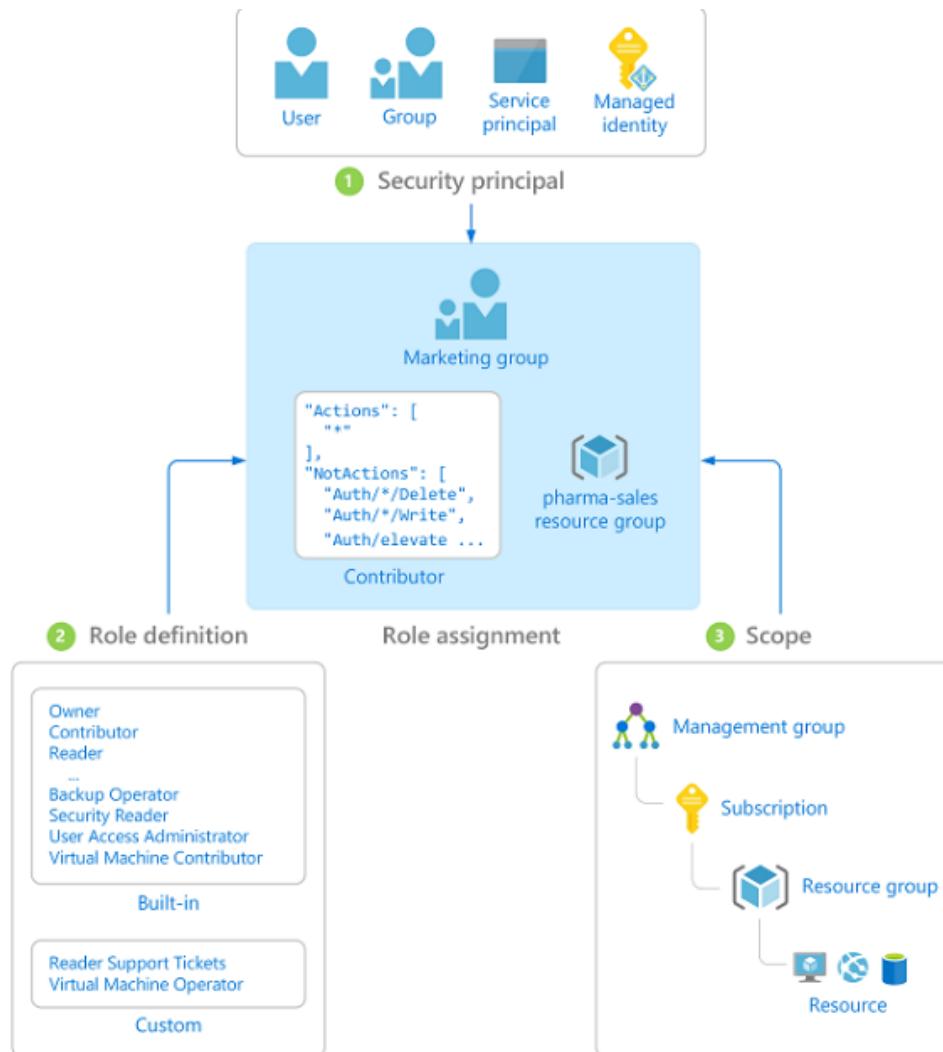
# Role definition

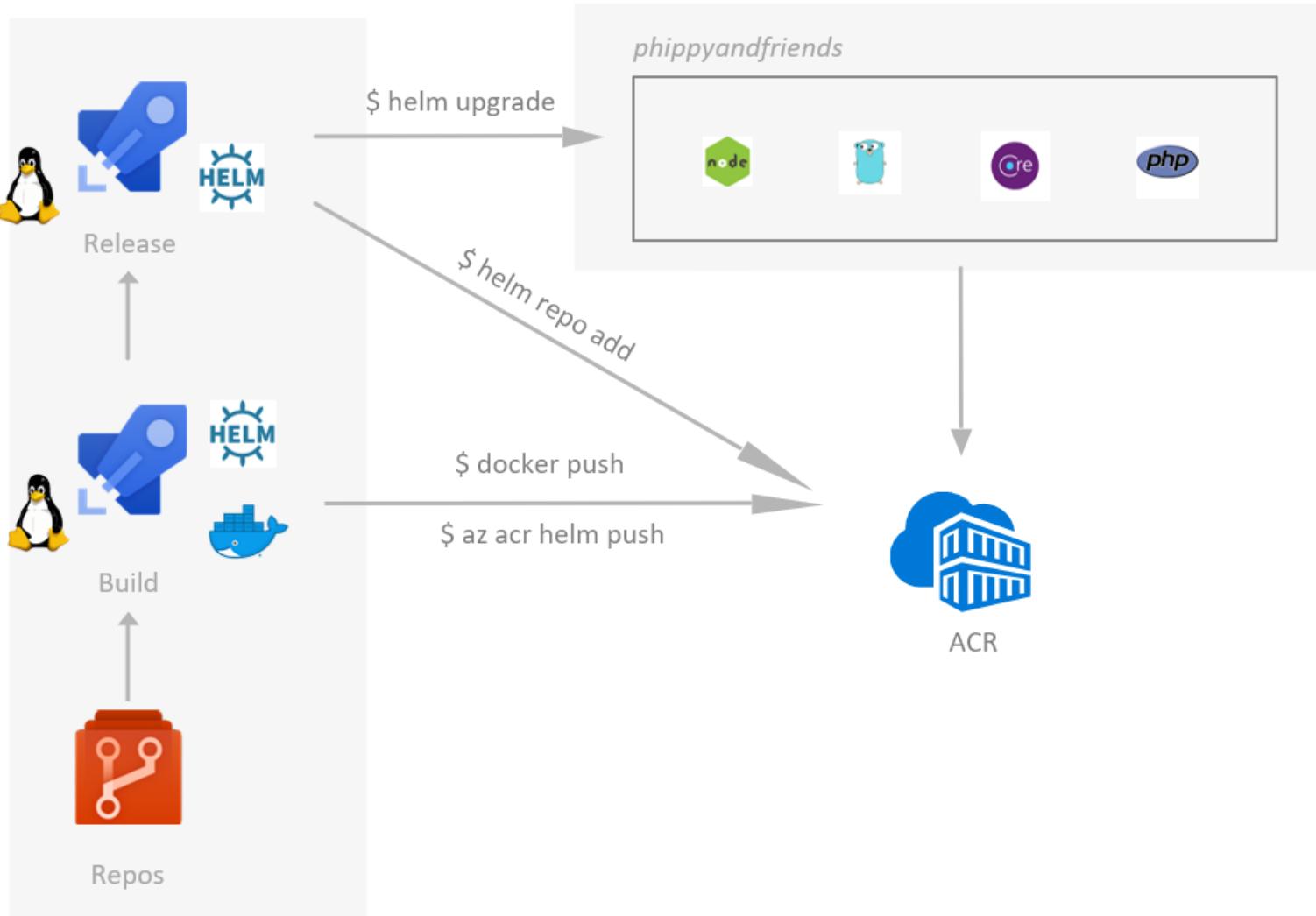


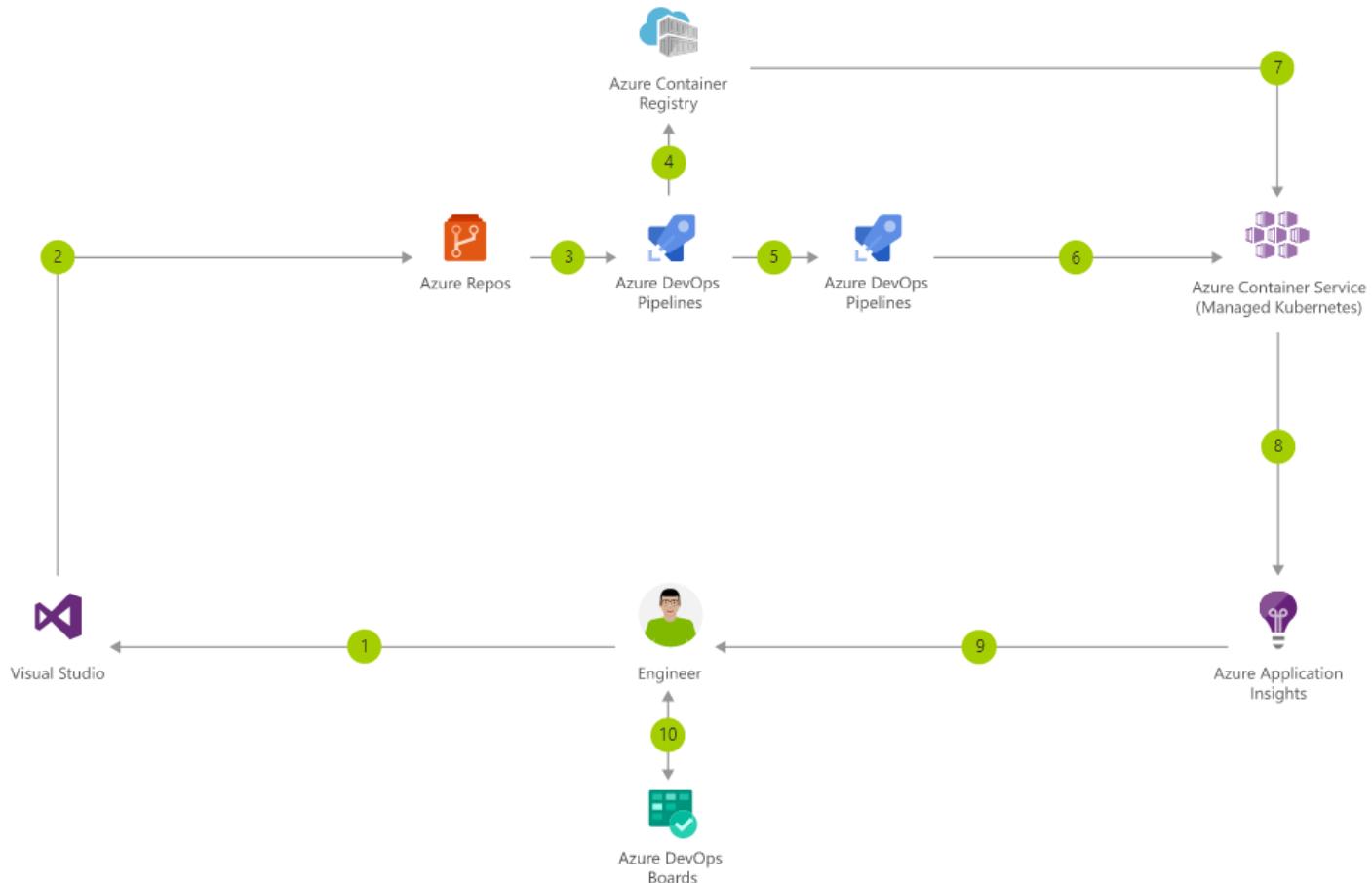
## 2 Role definition

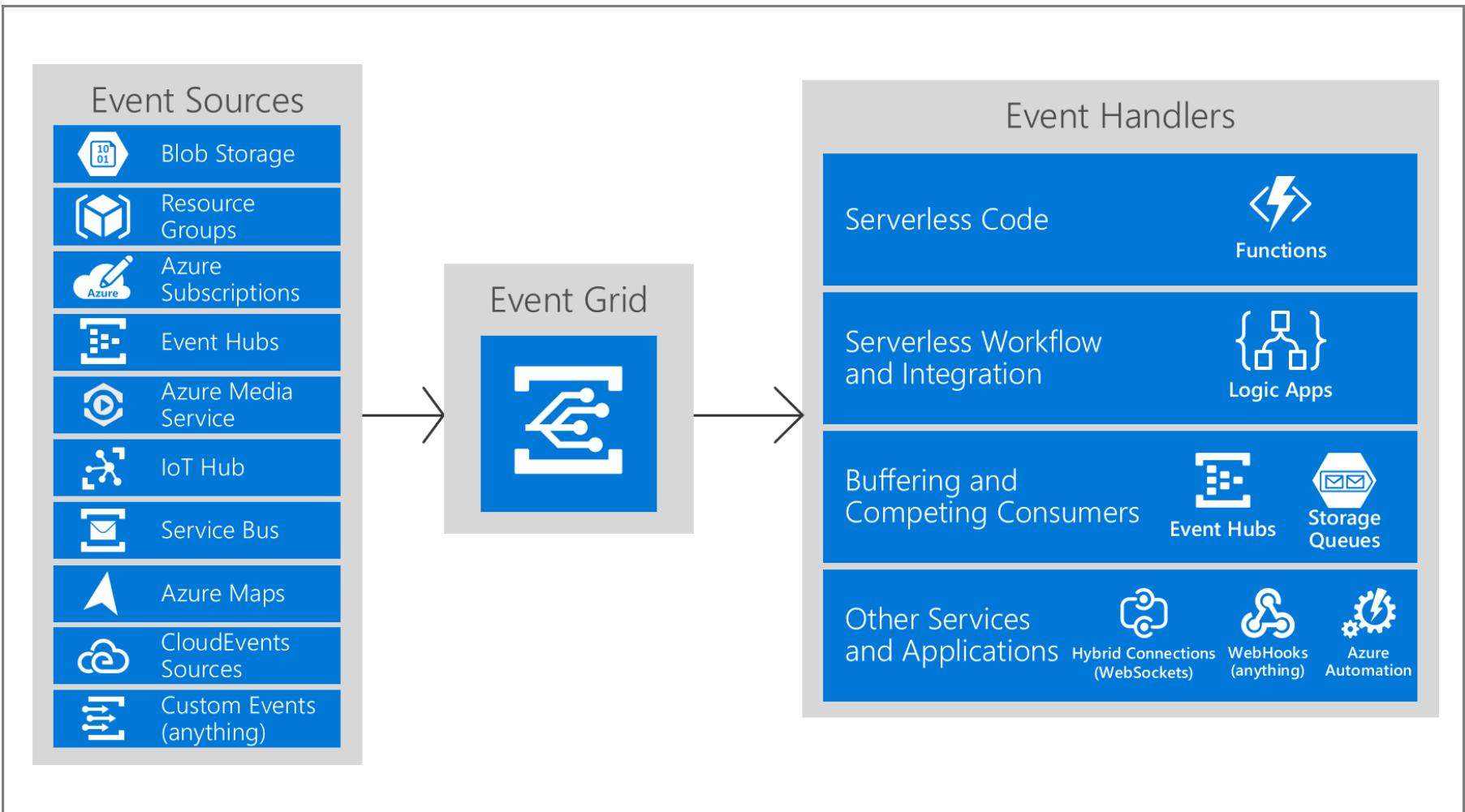


# Role definition











# Event Grid

---

- **Events** - What happened.
- **Event sources** - Where the event took place.
- **Topics** - The endpoint where publishers send events.
- **Event subscriptions** - The endpoint or built-in mechanism to route events, sometimes to more than one handler. Subscriptions are also used by handlers to intelligently filter incoming events.
- **Event handlers** - The app or service reacting to the event.



# Event Grid

---

- **Capabilities**
- Here are some of the key features of Azure Event Grid:
- **Simplicity** - Point and click to aim events from your Azure resource to any event handler or endpoint.
- **Advanced filtering** - Filter on event type or event publish path to make sure event handlers only receive relevant events.
- **Fan-out** - Subscribe several endpoints to the same event to send copies of the event to as many places as needed.



## Event Grid

- **Reliability** - 24-hour retry with exponential backoff to make sure events are delivered.
- **Pay-per-event** - Pay only for the amount you use Event Grid.
- **High throughput** - Build high-volume workloads on Event Grid with support for millions of events per second.
- **Built-in Events** - Get up and running quickly with resource-defined built-in events.
- **Custom Events** - Use Event Grid route, filter, and reliably deliver custom events in your app.



```
C:\WINDOWS\system32>setx EVENT_GRID_URL https://evntgridtopic.westus2-1.eventgrid.azure.net/api/events /M
```

SUCCESS: Specified value was saved.

```
C:\WINDOWS\system32>setx EVENT_GRID_KEY 8WPAX5iqJT7IZ4YVJla749MYwO/f+DcKiA+AVoB//VU= /M
```

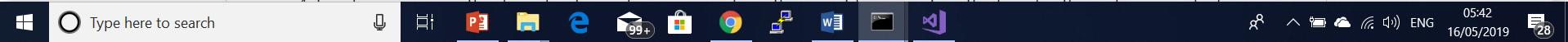
SUCCESS: Specified value was saved.

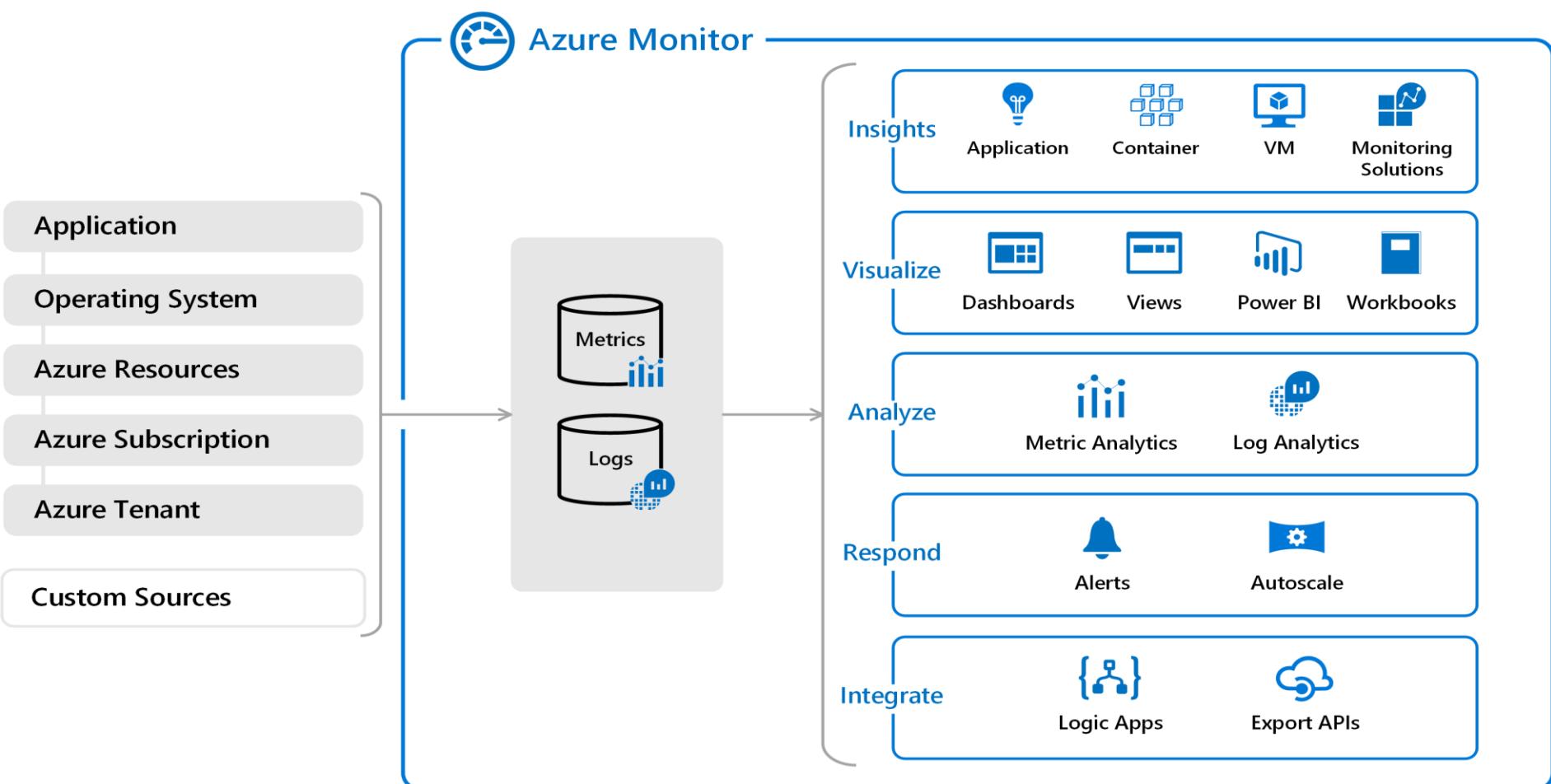
```
C:\WINDOWS\system32>
```



```
C:\Windows\System32\cmd.exe
F:\dotnetfidelity2019\build-event-grid-master\broadcaster>dotnet run "send tester message"
"
[{"Id":"53144afe-7d83-43bd-b175-3f0075b4cc2a","EventType":"BuildMessage","Subject":"send
tester message","EventTime":"2019-05-16T05:42:22.3817298+05:30","Data":{ "message": "send t
ester message" }}]
Response: OK - .
Successfully published.

F:\dotnetfidelity2019\build-event-grid-master\broadcaster>
```







# Azure Monitor Log Query

---

- Select resource group
- let startDatetime = todatetime("2019-05-15 20:12:42.9");
- let duration = totimespan(25m);
- Heartbeat
- | where TimeGenerated between(startDatetime .. (startDatetime+duration) )
- | extend timeFromStart = TimeGenerated - startDatetime



# Azure Monitor Log Query

- `Perf | where TimeGenerated > ago(30m) | summarize count() by bin(TimeGenerated, 5m) | render timechart`
- -----

# State and data in Docker applications



- A process doesn't maintain persistent state.
- While a container can write to its local storage, assuming that an instance will be around indefinitely would be like assuming that a single location in memory will be durable.
- Assume that container images, like processes, have multiple instances or will eventually be killed.
- If they're managed with a container orchestrator, you should assume that they might get moved from one node or VM to another.

# State and data in Docker applications



- The following solutions are used to manage persistent data in Docker applications:
- From the Docker host, as Docker Volumes:
- **Volumes** are stored in an area of the host filesystem that's managed by Docker.
- **Bind mounts** can map to any folder in the host filesystem, so access can't be controlled from Docker process and can pose a security risk as a container could access sensitive OS folders.
- **tmpfs mounts** are like virtual folders that only exist in the host's memory and are never written to the filesystem.

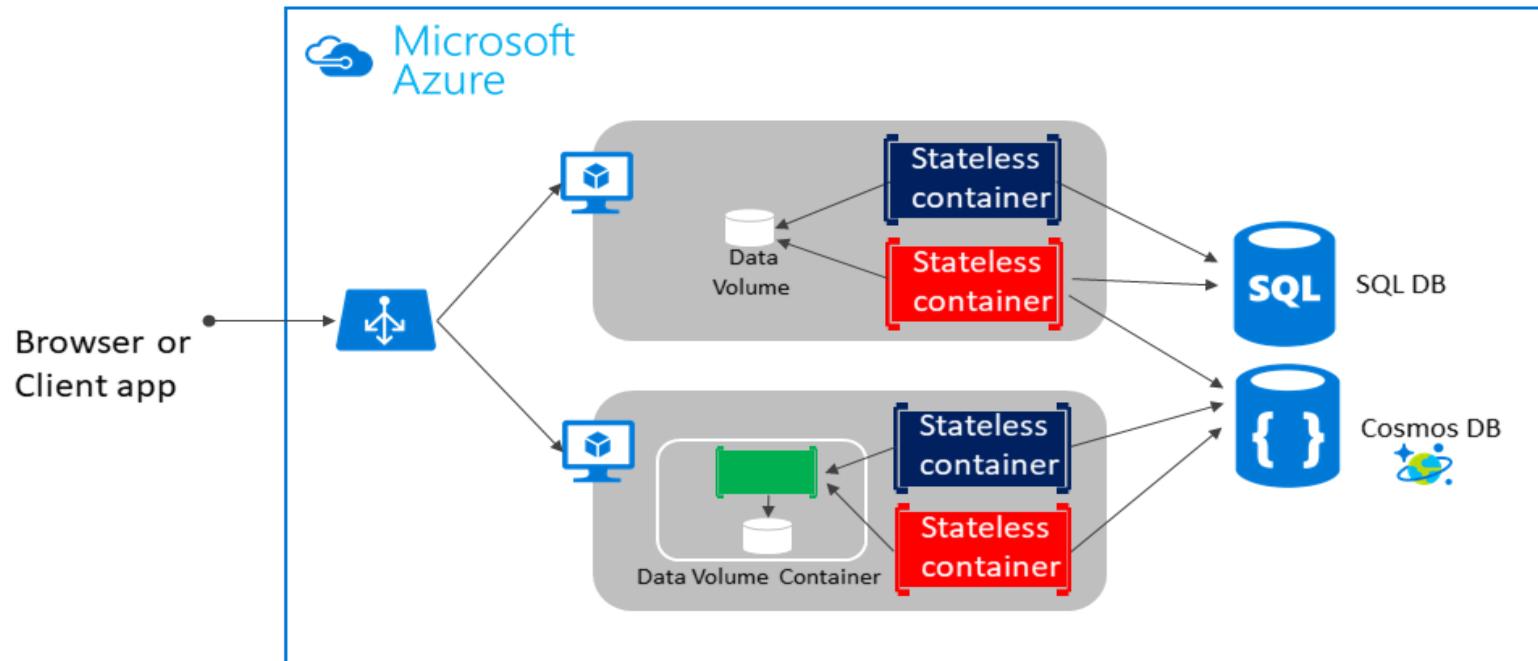
# State and data in Docker applications



- From remote storage:
- Azure Storage, which provides geo-distributable storage, providing a good long-term persistence solution for containers.
- Remote relational databases like Azure SQL Database or NoSQL databases like Azure Cosmos DB, or cache services like Redis.



## Data Volume and Data Volume Container





# Domain Driven Design

- Domain-Driven Design process, you end up with these object types:
- A list of Entities, some of which are Aggregates, including identified root Aggregates and Entities
- Defined Repositories
- A list of Value Objects that are associated with one or more Entities
- A list of Services that correspond to functions that aren't part of any particular Entity

# Domain Driven Design



*Designing a city analogy*

Unplanned



Big Ball Of Mud

Planned



Domain Driven Design



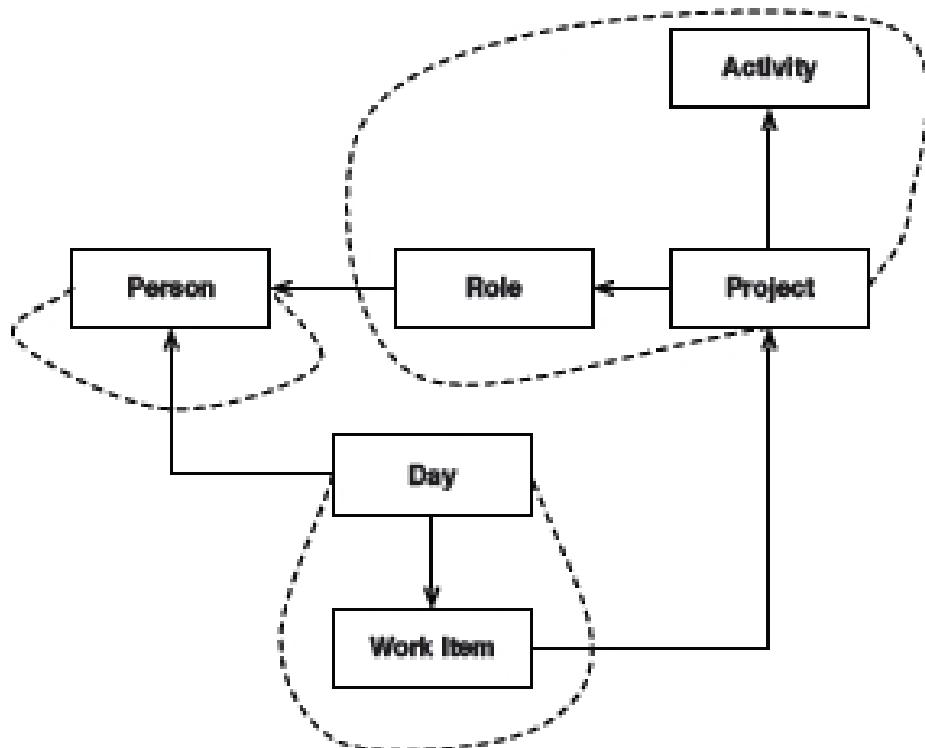
# Domain Driven Design

---

- **Dealing with Structure**
- **Entities**
- **Cardinality of Associations**
- **Services**
- **Aggregates**
  - The root has global identity and the others have local identity
  - The root checks that all invariants are satisfied
  - Entities outside the aggregate only hold references to the root
  - Deletes remove everything in the aggregate
  - When an object changes, all invariants must be satisfied.



# Domain Driven Design





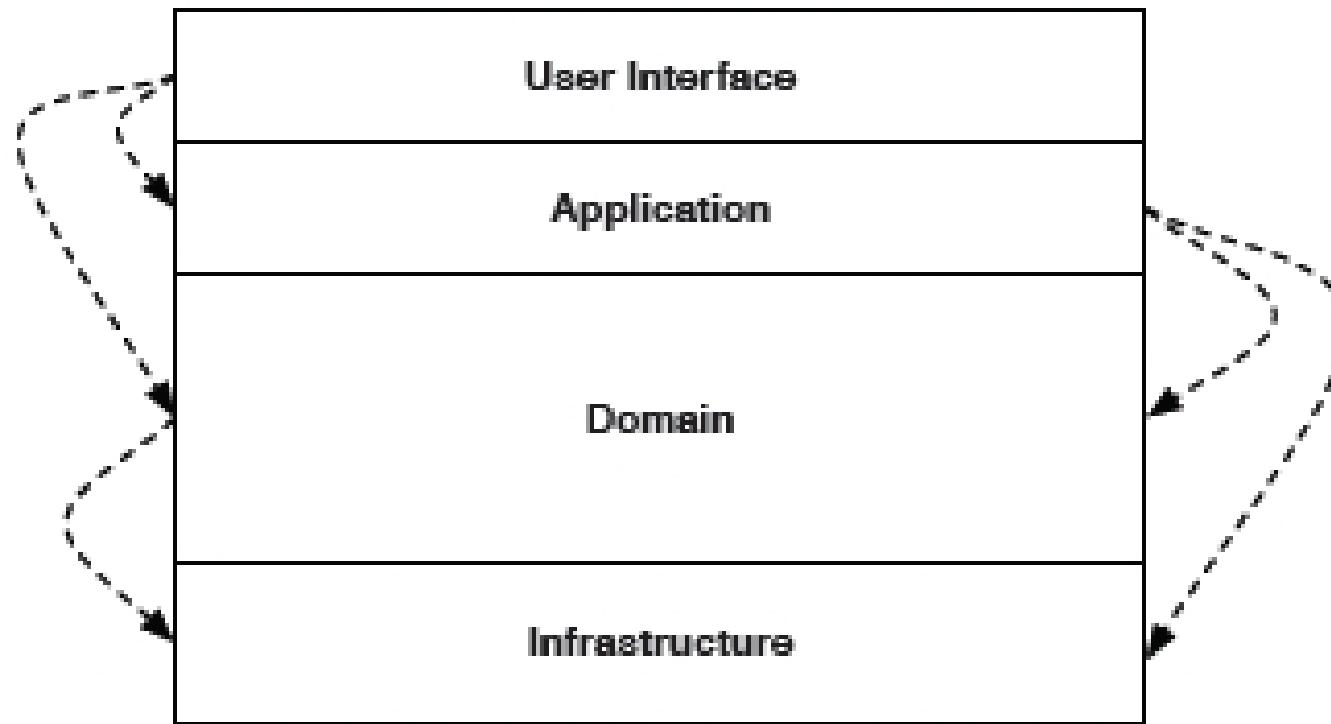
# Domain Driven Design

---

- Dealing with Life Cycles
- Factories
- Dealing with Behavior
- Specification Pattern
- Strategy Pattern
- Composite Pattern



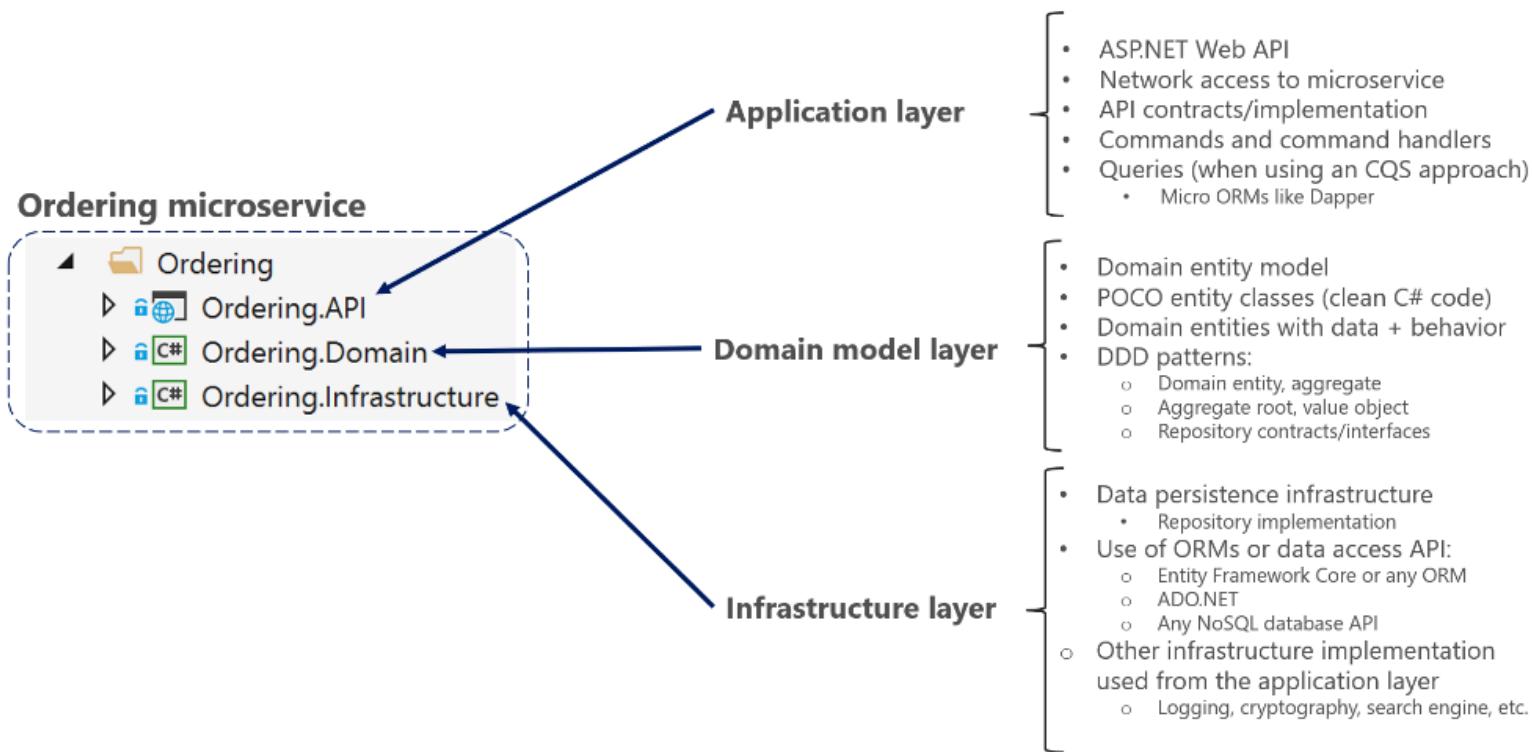
# Domain Driven Design



# Domain Driven Design



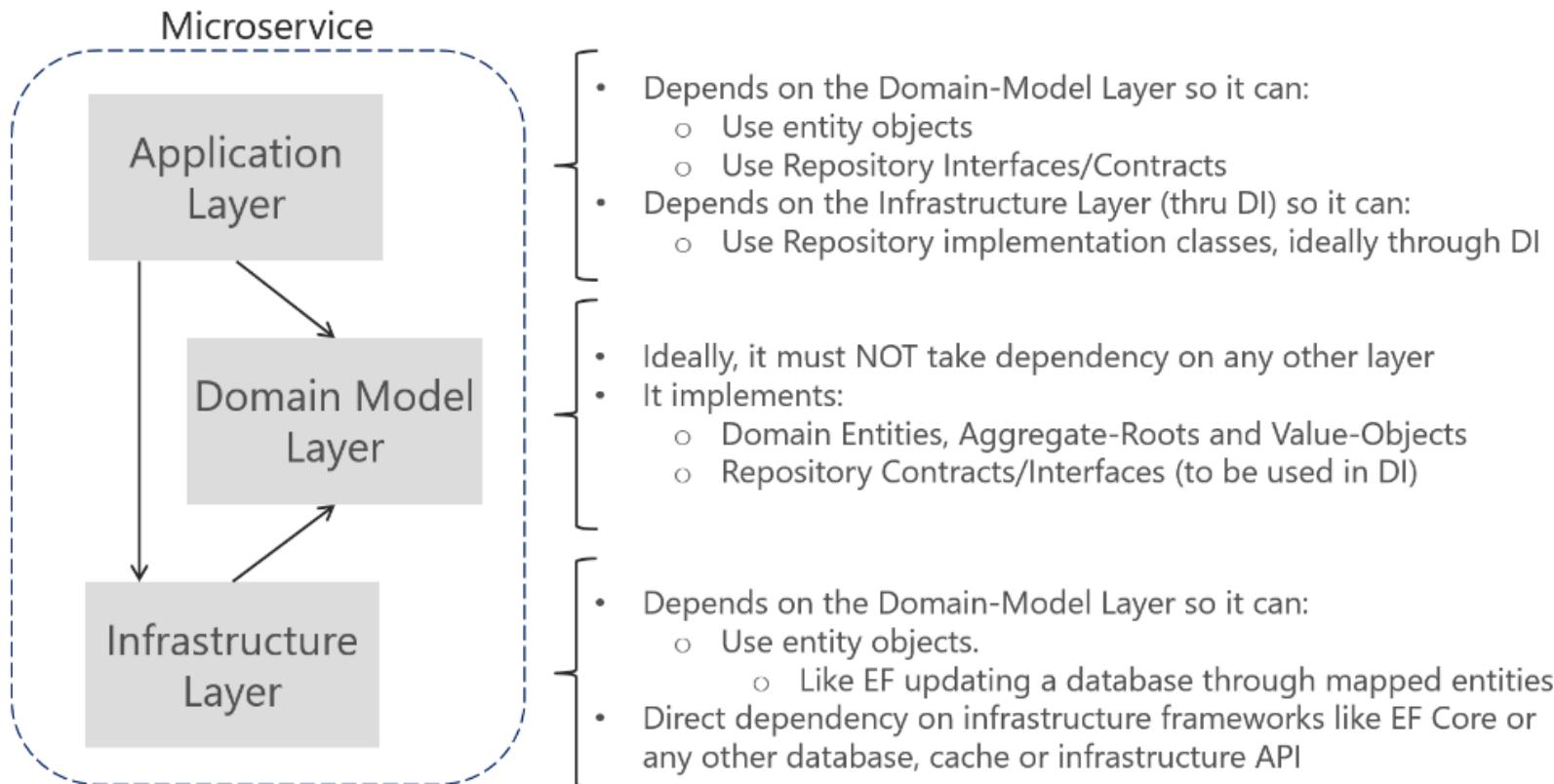
## Layers in a Domain-Driven Design Microservice





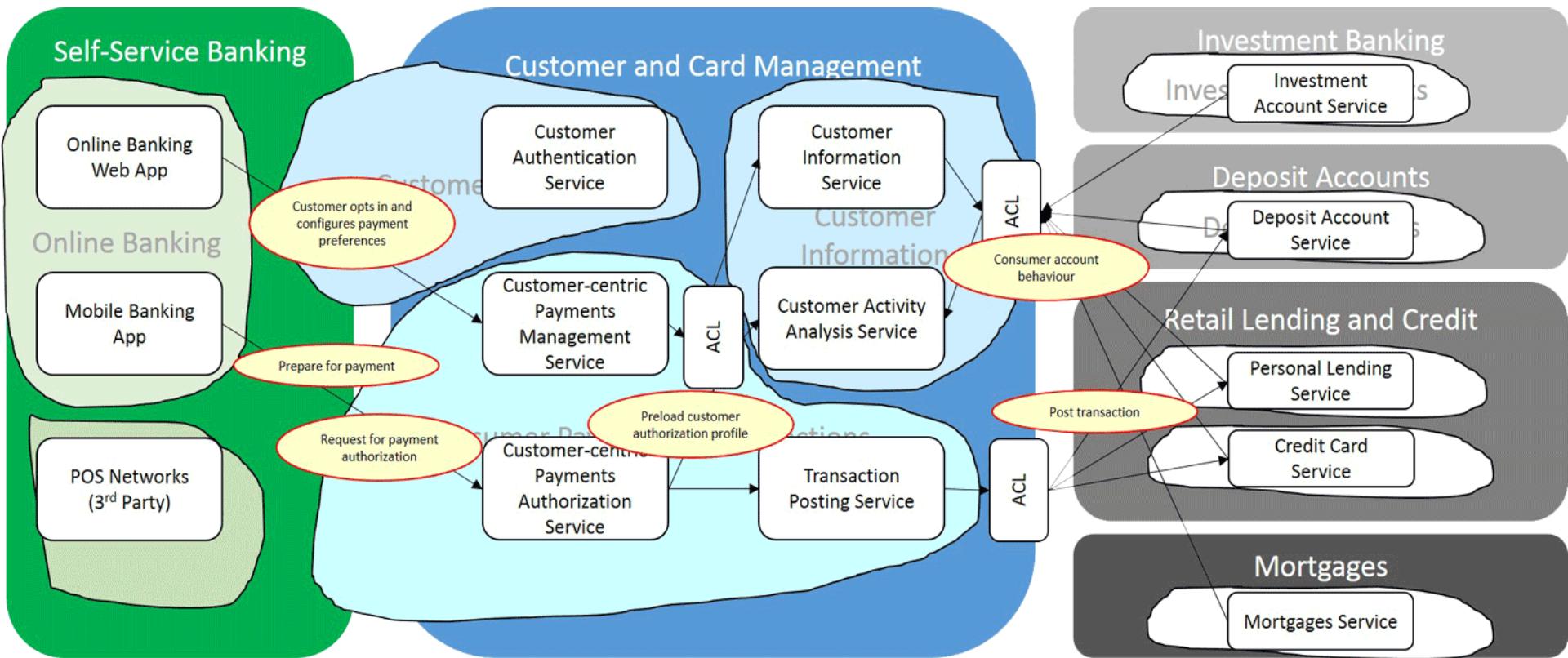
# Domain Driven Design

## Dependencies between Layers in a Domain-Driven Design service





# Domain Driven Design



# Strategies to handle partial failure



- **Use asynchronous communication (for example, message-based communication) across internal microservices.**
- **Use retries with exponential backoff.**
- **Work around network timeouts.**
- **Use the Circuit Breaker pattern.**
- **Limit the number of queued requests.**

# Resilience policies



Policy	Premise	Aka	How does the policy mitigate?
<b>Retry</b> (policy family) <a href="#">(quickstart)</a> ; <a href="#">(deep)</a>	Many faults are transient and may self-correct after a short delay.	"Maybe it's just a blip"	Allows configuring automatic retries.
<b>Circuit-breaker</b> (policy family) <a href="#">(quickstart)</a> ; <a href="#">(deep)</a>	When a system is seriously struggling, failing fast is better than making users/callers wait.  Protecting a faulting system from overload can help it recover.	"Stop doing it if it hurts"  "Give that system a break"	Breaks the circuit (blocks executions) for a period, when faults exceed some pre-configured threshold.
<b>Timeout</b> <a href="#">(quickstart)</a> ; <a href="#">(deep)</a>	Beyond a certain wait, a success result is unlikely.	"Don't wait forever"	Guarantees the caller won't have to wait beyond the timeout.

# Resilience policies



<b>Bulkhead Isolation</b> ( <a href="#">quickstart</a> ; <a href="#">deep</a> )	<p>When a process faults, multiple failing calls backing up can easily swamp resource (eg threads/CPU) in a host.</p> <p>A faulting downstream system can also cause 'backed-up' failing calls upstream.</p> <p>Both risk a faulting process bringing down a wider system.</p>	<p>"One fault shouldn't sink the whole ship"</p>	Constrains the governed actions to a fixed-size resource pool, isolating their potential to affect others.
<b>Cache</b> ( <a href="#">quickstart</a> ; <a href="#">deep</a> )	<p>Some proportion of requests may be similar.</p>	<p>"You've asked that one before"</p>	<p>Provides a response from cache if known.</p> <p>Stores responses automatically in cache, when first retrieved.</p>
<b>Fallback</b> ( <a href="#">quickstart</a> ; <a href="#">deep</a> )	<p>Things will still fail - plan what you will do when that happens.</p>	<p>"Degrade gracefully"</p>	Defines an alternative value to be returned (or action to be executed) on failure.



# Polly

- Polly is a .NET resilience and transient-fault-handling library that allows developers to express policies such as Retry, Circuit Breaker, Timeout, Bulkhead Isolation, and Fallback in a fluent and thread-safe manner.
- **Adding Polly to our solution**
- Install-Package Polly



# Polly – Retry Policy

---

- Policy
- .Handle<Exception>()
- .Retry(3, (exception, retryCount) =>
- {
- var result = YourFunction();
- });



# Polly – Retry Policy

---

- Policy
- .Handle<Exception>()
- .Retry(3, (exception, retryCount) =>
- {
- var result = YourFunction();
- });



# Polly – Retry Policy

---

- Policy
- .Handle<Exception>()
- .RetryForever(exception =>
- {
- YourFunction();
- });



# Polly – wait and retry

---

- Policy
- .Handle<Exception>()
- .WaitAndRetry(new[]
- {
- TimeSpan.FromSeconds(1),
- TimeSpan.FromSeconds(2),
- TimeSpan.FromSeconds(3)
- }, (exception, timeSpan, context) => {
- YourFunction();
- });



## Polly – circuit breaker

- Action<Exception, TimeSpan> onBreak = (exception, timespan) => { ... };
- Action onReset = () => { ... };
- CircuitBreakerPolicy breaker = Policy
  - .Handle<Exception>()
  - .CircuitBreaker(2, TimeSpan.FromMinutes(1), onBreak, onReset);

# What Is Kubernetes? An Introduction To Container Orchestration Tool



- **What Is Kubernetes?**
- Kubernetes is an open-source container management (orchestration) tool. Its container management responsibilities include container deployment, scaling & descaling of containers & container load balancing.
- Download from
- <https://github.com/kubernetes/minikube>
- Under curl - windows
- <https://kubernetes.io/docs/tasks/tools/install-kubectl/#install-kubectl>

# What Is Kubernetes? An Introduction To Container Orchestration Tool



- **Why Use Kubernetes?**
- Companies out there maybe using Docker or Rocket or maybe simply Linux containers for containerizing their applications. But, whatever it is, they use it on a massive scale. They don't stop at using 1 or 2 containers in Prod. But rather, **10's or 100's** of containers for load balancing the traffic and ensuring high availability.
- .



# Features Of Kubernetes

1

Automatic Binpacking

2

Service Discovery &  
Load Balancing

3

Storage Orchestration

4

Self Healing

5

Batch Execution

6

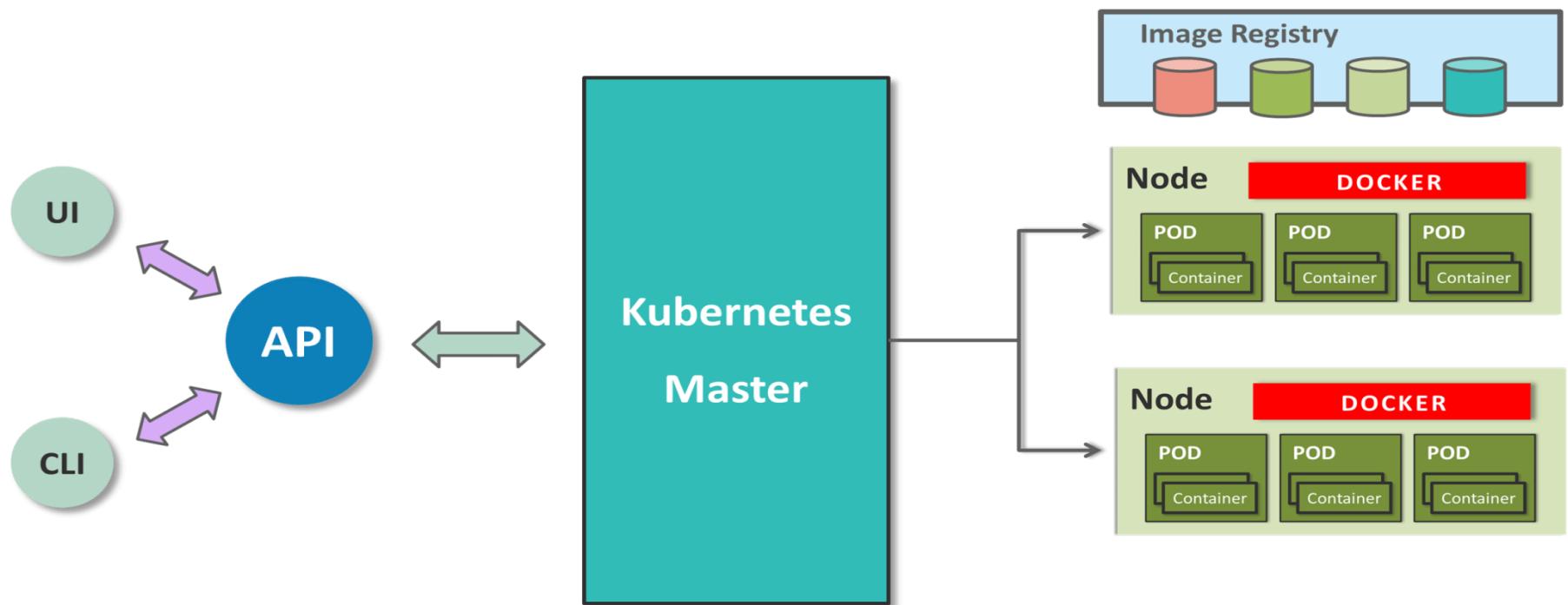
Horizontal Scaling

7

Secret & Configuration  
Management

8

Automatic Rollbacks  
& Rollouts



# Minikube –p cluster1 dashboard



Screenshot of the Kubernetes Dashboard running on Minikube. The browser tabs show "Hello Minikube - Kubernetes" and "Overview - Kubernetes Dashboard".

The dashboard interface includes:

- Cluster** sidebar: Namespaces, Nodes, Persistent Volumes, Roles, Storage Classes.
- Namespace** dropdown: default.
- Overview** tab selected.
- Services** section: A table showing a single service named "kubernetes".

Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age
kubernetes	component: apiserver provider: kubernetes	10.96.0.1	kubernetes:443 TCP	-	6 minutes
- Secrets** section: A table showing a single secret named "default-token-r827j".

Name	Type	Age
default-token-r827j	kubernetes.io/service-account-token	6 minutes
- Workloads** sidebar: Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, Replication Controllers.
- System tray** at the bottom: Windows logo, search bar, task icons (File Explorer, Edge, etc.), system status (Wi-Fi, battery, volume), date/time (20/02/2019, 22:50), and notification badge (36).

# AWS Cluster



No.1 Tamil website in the world | Credentials - parameswarabala@... | Getting Started with CQRS – Part | Getting Started with Amazon EKS | Amazon EKS

https://us-east-2.console.aws.amazon.com/eks/home?region=us-east-2#/clusters/test

Apps Insert title here Empire New Tab How to use Asserti... Browser Automatio... node.js - How can I ... Freelancer-dev-810... Courses New Tab Google hi

aws awsblr Ohio Support

**Amazon Container Services**

EKS > Clusters > test

## test

**General configuration**

Kubernetes version	Platform version	Status
1.11	eks.1	ACTIVE
API server endpoint	Certificate authority	
https://8FC057FCC5404B401C7366F6781BC7ED.sk1.us-east-2.eks.amazonaws.com	LS0tLS1CRUdjTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUN5RENDQWJDZ0f3SUJBZ0lCQURBTkJna3Foa2lHOXcwQkFRc0ZBREFWTvJnd0VRWURWUVFERXdwcmRXSmwKY201bGRHVnpNQjRYRFRNU1ESXlOekUzTWpnME1Gb1hEVEk1TURJe	
Cluster ARN	Role ARN	
arn:aws:eks:us-east-2:724457010663:cluster/test	arn:aws:iam::724457010663:role/Manager	

**Networking**

VPC Subnets Security groups



# EKS Steps

---

- **Assumptions and Prerequisites**
- You should have an AWS account with an active subscription and be able to log in using [AWS IAM](#) account credentials. If you don't have either of these create an AWS account & create an IAM user in your AWS account.
- You should install the latest version of the AWS command-line interface (CLI), to a location in your system path. In case you haven't, [install it using these instructions.](#)



# EKS Steps

- **Step 1: Create an AWS IAM service role and a VPC**
- The first step is to create an IAM role that Kubernetes can assume to create AWS resources. To do this:
- Navigate to AWS IAM Console & in “Roles” section, click the “Create role” button
- Select “AWS service” as the type of entity and “EKS” as the service
- Enter a name for the service role and click “Create role” to create the role

# EKS Steps



AWS Services Resource Groups Archana Global Support

## Create role

Review

Provide the required information below and review this role before you create it.

**Role name\*** eksServiceRole

Use alphanumeric and '+,.,@-\_ characters. Maximum 64 characters.

**Role description** Allows EKS to manage clusters on your behalf.

Maximum 1000 characters. Use alphanumeric and '+,.,@-\_ characters.

**Trusted entities** AWS service: eks.amazonaws.com

**Policies** AmazonEKSClusterPolicy AmazonEKSServicePolicy

**Permissions boundary** Permissions boundary is not set

\* Required

Cancel Previous Create role



# EKS Steps

AWS Services Resource Groups N. Virginia Support

CloudFormation > Stacks > Create stack

Step 1 Specify template

Step 2 Specify stack details

Step 3 Configure stack options

Step 4 Review

## Create stack

**Stack name**

Stack name: AmazonEKSVPC

Stack name can include letters (A-Z and a-z), numbers (0-9), and dashes (-).

**Parameters**

Parameters are defined in your template and allow you to input custom values when you create or update a stack.

**Worker Network Configuration**

**VpcBlock**  
The CIDR range for the VPC. This should be a valid private (RFC 1918) CIDR range.  
192.168.0.0/16

**Subnet01Block**  
CidrBlock for subnet 01 within the VPC  
192.168.64.0/18

**Subnet02Block**  
CidrBlock for subnet 02 within the VPC  
192.168.128.0/18

**Subnet03Block**  
CidrBlock for subnet 03 within the VPC  
192.168.192.0/18

Cancel Previous Next

Feedback English (US) © 2008 - 2018, Amazon Internet Services Private Ltd. or its affiliates. All rights reserved. Privacy Policy Terms of Use



## EKS Steps

---

- **Step 2: Create an Amazon EKS cluster**
- At this point, you are ready to create a new Amazon EKS cluster. To do this:
  - Navigate to the Amazon EKS console and click on “Create cluster” button
  - Enter details into the EKS cluster creation form such as cluster name, role ARN, VPC, subnets and security groups
  - Click “Create” to create the Amazon EKS cluster

# EKS Steps



Screenshot of the AWS EKS Create Cluster configuration page.

**Cluster configuration:**

- Cluster name:** nginxcluster
- Kubernetes version:** 1.10
- Role name:** eksServiceRole

**Networking:**

- VPC:** vpc-0f8853ef93162beff - 192.168.0.0/16
- Subnets:** Three subnets selected:
  - subnet-0643f874c1b0a572e (AmazonEKSVPCCSubnet02) - us-east-1b, 192.168.128.0/18
  - subnet-09a8ca2ac0e5e6d45 (AmazonEKSVPCCSubnet03) - us-east-1c, 192.168.192.0/18
  - subnet-048f24dea6adb8526 (AmazonEKSVPCCSubnet01) - us-east-1a, 192.168.64.0/18
- Security groups:** Two security groups selected:
  - sg-0033319d94c70f6b8 (default) - default VPC security group
  - sg-09e6ccaae3498cf9 (AmazonEKSVPCCControlPlaneSecurityGroup-B1U8RVRCV8DA) - Cluster communication with worker nodes

**Buttons:** Cancel, Create



# EKS Steps

- **Step 3: Configure *kubectl* for Amazon EKS cluster**
- Kubernetes uses a command-line utility called ***Kubectl*** for communicating with Kubernetes cluster. Amazon EKS clusters also require the AWS IAM Authenticator for Kubernetes to allow IAM authentication for your Kubernetes cluster. So, install both of these binaries. Instructions for downloading and setup are in the Amazon EKS documentation.
- **Note:** Ensure that you have at least version of the AWS CLI installed and your system's Python version must be Python 2.7.9 or greater.
- Next, you have to create a ***kubeconfig*** file for your cluster with the AWS CLI **update-kubeconfig** command as follows:
- Use the AWS CLI **update-kubeconfig** command to create or update your kubeconfig for your cluster
- Test your configuration



# EKS Steps

Windows PowerShell

```
PS C:\kubern> aws eks update-kubeconfig --name nginxcluster  
Updated context arn:aws:eks:us-east-1:397527253565:cluster/nginxcluster in C:\Users\archana_ch\.kube\config
```

```
PS C:\kubern> kubectl get svc  
NAME      TYPE      CLUSTER-IP    EXTERNAL-IP   PORT(S)      AGE  
kubernetes  ClusterIP  10.100.0.1  <none>        443/TCP    18m
```

```
PS C:\kubern> ■
```



# EKS Steps

- Step 4: Launch and configure Amazon EKS worker nodes
- Note: Wait for your cluster status to show as ACTIVE. If you launch your worker nodes before the cluster is active, the worker nodes will fail to register with the cluster and you will have to relaunch them.
- Once the control plane of your cluster has been activated, the next step is to add nodes to it. To do this:
- Navigate to the AWS CloudFormation console and click on “Create stack” option
- On the “Select Template” page, select the option to “Specify an Amazon S3 template URL” and enter the URL
- On the “Specify Details” page, enter details as shown below. Review the details and click on “Create”
- Once stack creation is complete, select the stack name in the list of available stacks and select the “Outputs” section in the lower left pane. Make a note of Role ARN



# EKS Steps

Sales > Resource Groups > Create stack

Step 1 Specify template

Step 2 Specify stack details

Step 3 Configure stack options

Step 4 Review

### Create stack

**Stack name**

Stack name: nginxcluster-worker-nodes  
Stack name can include letters (A-Z and a-z), numbers (0-9), and dashes (-).

**Parameters**

Parameters are defined in your template and allow you to input custom values when you create or update a stack.

**EKS Cluster**

**ClusterName**  
The cluster name provided when the cluster was created. If it is incorrect, nodes will not be able to join the cluster.  
nginxcluster

**ClusterControlPlaneSecurityGroup**  
The security group of the cluster control plane.  
AmazonEKSVPCControlPlaneSecurityGroup-B1U8RVRCV8DA (sg-09e6ccaae5498cf9)

**Worker Node Configuration**

**NodeGroupName**  
Unique identifier for the Node Group.  
mynodegroup

**NodeAutoScalingGroupMinSize**  
Minimum size of Node Group ASG.  
1

**NodeAutoScalingGroupMaxSize**  
Maximum size of Node Group ASG.  
3

**NodeInstanceType**  
EC2 instance type for the node instances.  
t2.medium

**NodeImageId**  
AMI id for the node instances.  
ami-0a0b913ef3249b655

**NodeVolumeSize**  
Node volume size.  
20

**KeyName**  
The EC2 Key Pair to allow SSH access to the instances.  
mykeypair

**BootstrapArguments**  
Arguments to pass to the bootstrap script. See files/bootstrap.sh in <https://github.com/awslabs/amazon-eks-ami>

**Worker Network Configuration**

**VpcId**  
The VPC of the worker instances.  
vpc-0f8853ef93162beff (192.168.0.0/16) (AmazonEKSVPCC-VPC)

**Subnets**  
The subnets where workers can be created.

subnet-0643f874c1b0a572e (192.168.128.0/18) (AmazonEKSVPCC-Subnet02) X

subnet-09a8ca2ac0e5e6d45 (192.168.192.0/18) (AmazonEKSVPCC-Subnet03) X

subnet-048f24dea6adb8526 (192.168.64.0/18) (AmazonEKSVPCC-Subnet01) X

Cancel Previous Next

# EKS Steps

- Now to enable worker nodes to join Kubernetes cluster follow below steps:
- On your local system, create a file named `aws-auth-cm.yaml` and fill it with the content below. Replace the AWS-ARN with the node instance role that you copied from the stack output earlier
- 

```
aws-auth-cm.yaml
1  apiVersion: v1
2  kind: ConfigMap
3  metadata:
4    name: aws-auth
5    namespace: kube-system
6  data:
7    mapRoles: |
8      - rolearn: <AWS-ARN>
9        username: system:node:{{EC2PrivateDNSName}}
10       groups:
11         - system:bootstrappers
12         - system:nodes
```



# EKS Steps

- Apply the configuration. This command may take a few minutes to finish.
- Watch the status of your nodes and wait for them to reach the Ready state

```
Windows PowerShell
PS C:\kubern> kubectl apply -f aws-auth-cm.yaml
configmap "aws-auth" created
PS C:\kubern> kubectl get nodes
NAME                      STATUS    ROLES      AGE     VERSION
ip-192-168-127-82.ec2.internal  NotReady  <none>    11s    v1.10.3
ip-192-168-158-125.ec2.internal  NotReady  <none>    8s     v1.10.3
ip-192-168-249-192.ec2.internal  NotReady  <none>    11s    v1.10.3
PS C:\kubern> kubectl get nodes
NAME                      STATUS    ROLES      AGE     VERSION
ip-192-168-127-82.ec2.internal  Ready    <none>    44s    v1.10.3
ip-192-168-158-125.ec2.internal  Ready    <none>    41s    v1.10.3
ip-192-168-249-192.ec2.internal  Ready    <none>    44s    v1.10.3
PS C:\kubern>
```



# EKS Steps

- Launch a simple nginx application
- To create an application you need to create a Kubernetes object of type Deployment. On your local system, create a file named nginx.yaml and fill it with the content below.
- Now, as a backup for application, you also need to create a Kubernetes object of Service type. A Kubernetes Service is an abstraction which defines a logical set of Pods running somewhere in your cluster, that all provide the same functionality as ones which you created earlier. On your local system, create a file named nginx-svc.yaml and fill it with the content below

```
nginx.yaml
1 apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
2 kind: Deployment
3 metadata:
4   name: nginx
5 spec:
6   selector:
7     matchLabels:
8       run: nginx
9   replicas: 2 # tells deployment to run 2 pods matching the template
10  template:
11    metadata:
12      labels:
13        run: nginx
14    spec:
15      containers:
16        - name: nginx
17          image: nginx:1.7.9
18          ports:
19            - containerPort: 80
```

```
nginx-svc.yaml
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: nginx
5   labels:
6     run: nginx
7 spec:
8   ports:
9     # the port that this service should serve on
10    - port: 80
11      protocol: TCP
12   selector:
13     run: nginx
14   type: LoadBalancer
```



# EKS Steps

- Create the nginx application and nginx service
- List the running services and capture the external IP address & port
- After your external IP address is available, point a web browser to that address at the respective port to view your nginx application

```
PS C:\kubern> kubectl create -f nginx.yaml
deployment.apps "nginx" created
PS C:\kubern> kubectl create -f nginx-svc.yaml
service "nginx" created
PS C:\kubern> kubectl get services -o wide
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP
kubernetes     ClusterIP  10.100.0.1   <none>
nginx          LoadBalancer 10.100.185.162 a306dc8fe8d111e8a2000a28798916d-747318031.us-east-1.elb.amazonaws.com
:31499/TCP    4m        run=nginx
PS C:\kubern> kubectl describe svc nginx
Name:           nginx
Namespace:      default
Labels:          run=nginx
Annotations:    <none>
Selector:        run=nginx
Type:           LoadBalancer
IP:             10.100.185.162
LoadBalancer Ingress: a306dc8fe8d111e8a2000a28798916d-747318031.us-east-1.elb.amazonaws.com
Port:           <unset>  80/TCP
TargetPort:     80/TCP
NodePort:       <unset>  31499/TCP
Endpoints:      192.168.187.235:80,192.168.90.184:80
Session Affinity: None
External Traffic Policy: Cluster
Events:
  Type  Reason          Age   From            Message
  ----  ----          ----  ----            -----
  Normal  EnsuringLoadBalancer  4m   service-controller  Ensuring load balancer
  Normal  EnsuredLoadBalancer  4m   service-controller  Ensured load balancer
PS C:\kubern>
```



# EKS Steps

Welcome to nginx!

Not secure ae1888e52e8bd11e8a2000a28798916d-1273662397.us-east-1.elb.amazonaws.com

# Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](http://nginx.org).  
Commercial support is available at [nginx.com](http://nginx.com).

*Thank you for using nginx.*

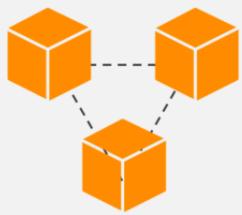


# EKS Steps

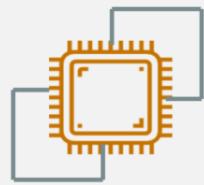
- **Step 6: Cleaning up the application & assigned resources**
- When you are finished experimenting with your application, you should clean up the resources that

```
Windows PowerShell

PS C:\kubern> kubectl delete -f nginx.yaml
deployment.apps "nginx" deleted
PS C:\kubern> kubectl delete svc nginx
service "nginx" deleted
PS C:\kubern> kubectl get services -o wide
NAME      TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE    SELECTOR
kubernetes  ClusterIP  10.100.0.1  <none>        443/TCP   1h    <none>
PS C:\kubern>
```



*Provision an Amazon EKS cluster*



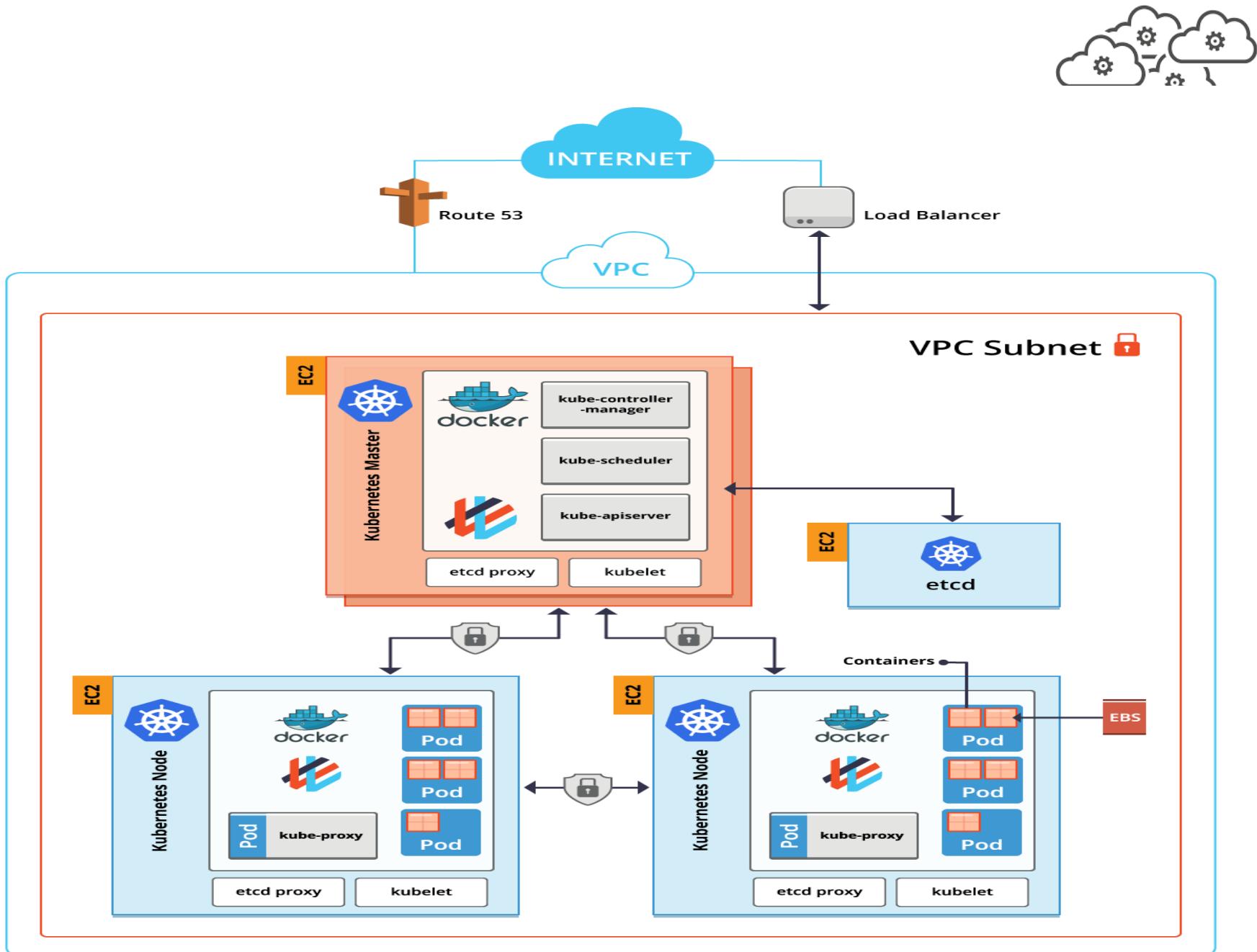
*Deploy worker nodes*



*Connect to EKS*



*Run Kubernetes apps on EKS cluster*



# AWS Cluster



No.1 Tamil website in the world | Credentials - parameswaribala@... | Getting Started with CQRS - Part 1 | Getting Started with Amazon EKS | Amazon EKS

https://us-east-2.console.aws.amazon.com/eks/home?region=us-east-2#/clusters

aws Services Resource Groups

Amazon Container Services

Amazon ECS Clusters Task definitions

Amazon EKS Clusters

Amazon ECR Repositories

EKS > Clusters

Clusters (1)

Find clusters by name

Cluster name Kubernetes version Status

Cluster name	Kubernetes version	Status
test	1.11	ACTIVE

Feedback English (US) © 2008 - 2019, Amazon Internet Services Private Ltd. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Type here to search

23 28 27/02/2019

# Command and Query Responsibility Segregation (CQRS) pattern



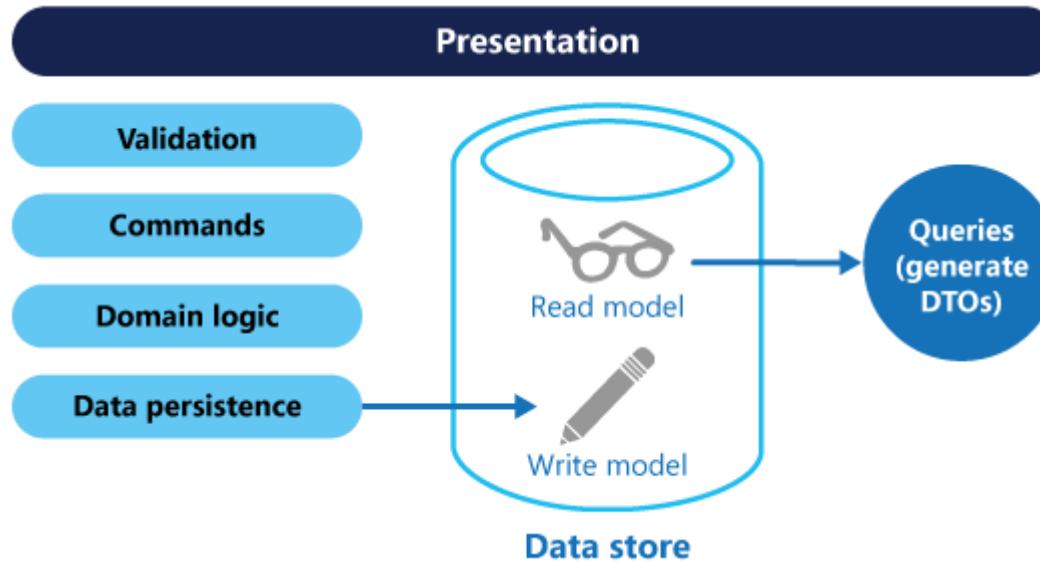
- Segregate operations that read data from operations that update data by using separate interfaces.
- This can maximize performance, scalability, and security.
- Supports the evolution of the system over time through higher flexibility, and prevents update commands from causing merge conflicts at the domain level.

# Command and Query Responsibility Segregation (CQRS) pattern

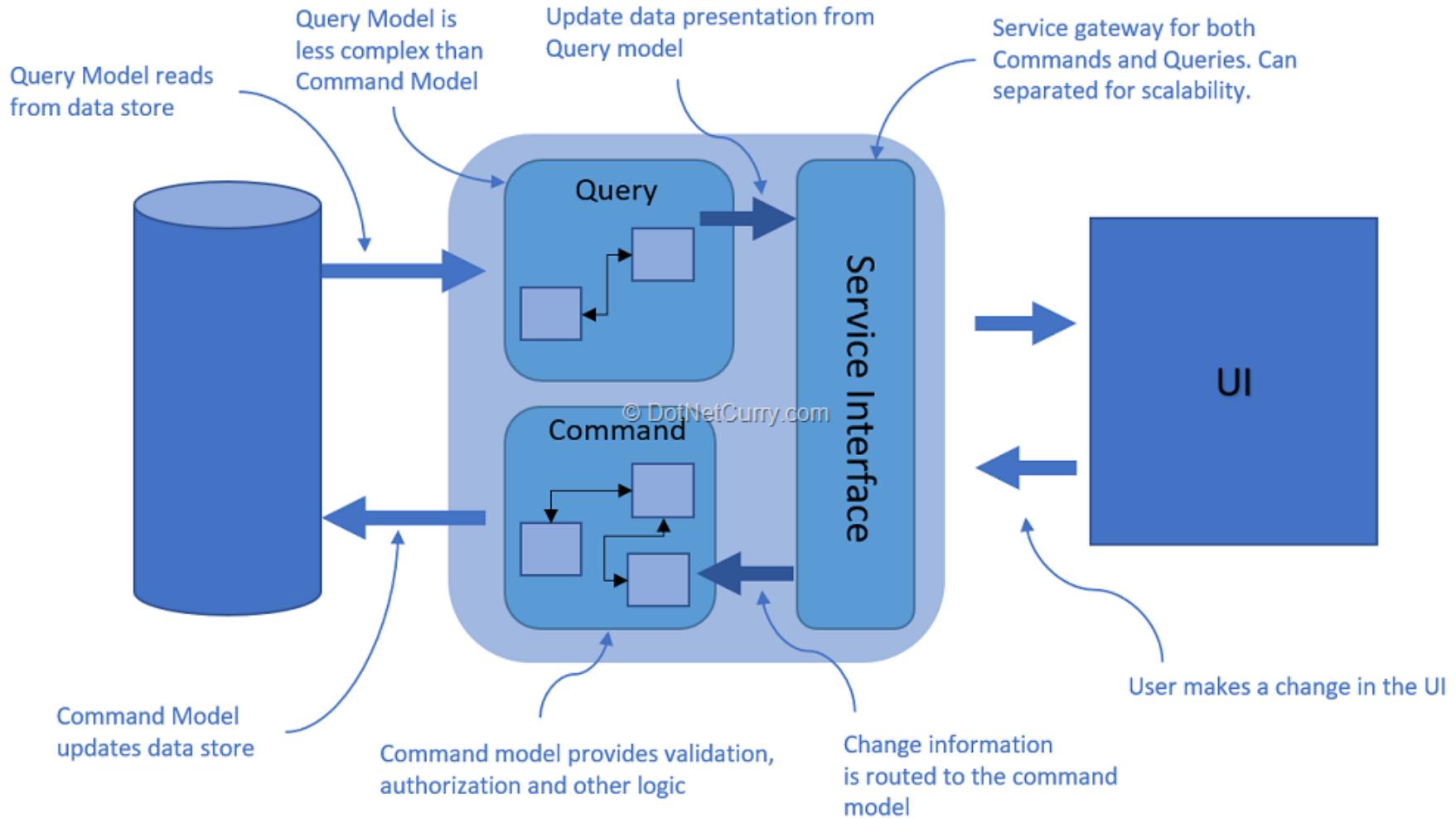


- In traditional data management systems, both commands (updates to the data) and queries (requests for data) are executed against the same set of entities in a single data repository.
- These entities can be a subset of the rows in one or more tables in a relational database such as SQL Server.
- Typically in these systems, all create, read, update, and delete (CRUD) operations are applied to the same representation of the entity.
- For example, a data transfer object (DTO) representing a customer is retrieved from the data store by the data access layer (DAL) and displayed on the screen.

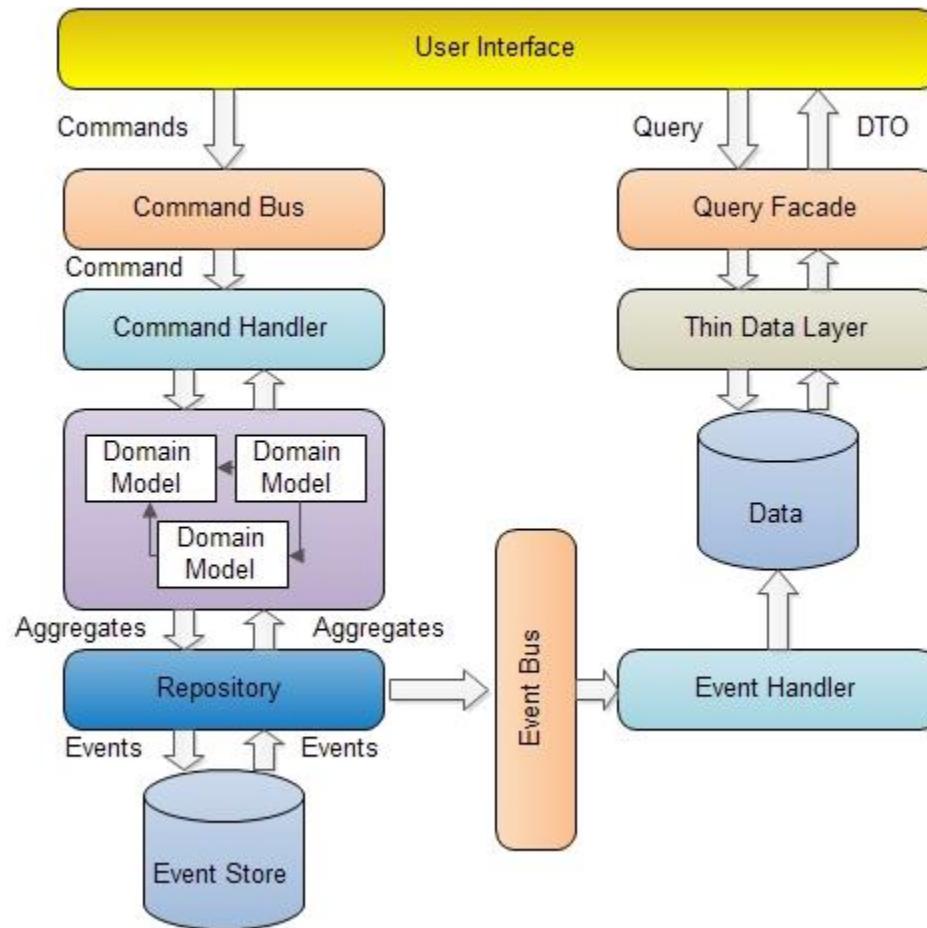
# Command and Query Responsibility Segregation (CQRS) pattern



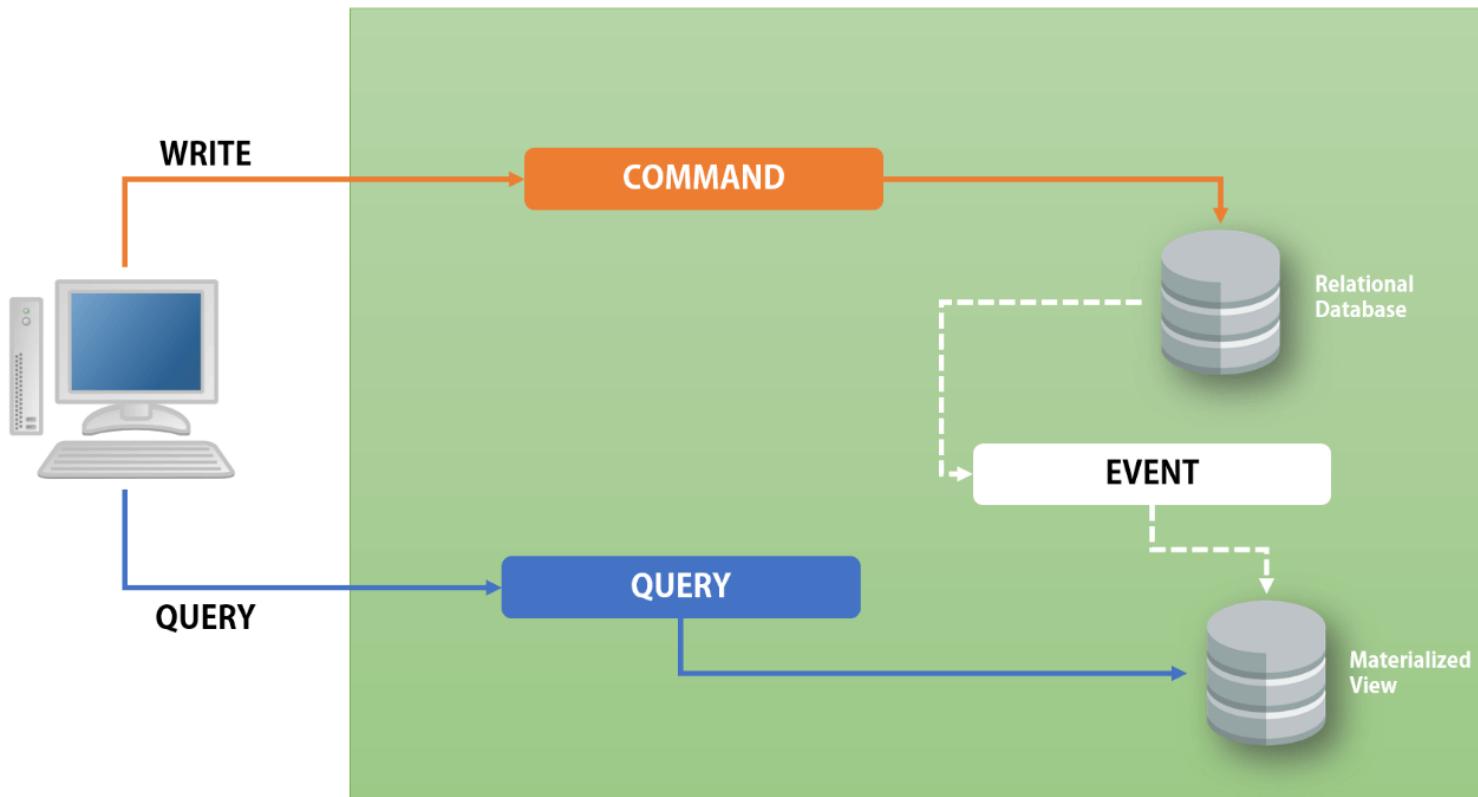
# Command and Query Responsibility Segregation (CQRS) pattern



# CQRS Command Query Responsibility Segregation



# CQRS



- This pattern was first introduced by Greg Young and Udi Dahan.
- They took inspiration from a pattern called Command Query Separation which was defined by Bertrand Meyer in his book “Object Oriented Software Construction”.
- The main idea behind CQS is: “A method should either change state of an object, or return a result, but not both.”
- **Commands** - change the state of an object or entire system (sometimes called as modifiers or mutators).
- **Queries** - return results and do not change the state of an object.



Secure | https://www.rabbitmq.com

RabbitMQ by Pivotal. Features Get Started Support Community Docs Blog

RabbitMQ is the most widely deployed open source message broker.

#### Updates

- |                                 |             |
|---------------------------------|-------------|
| <a href="#">RabbitMQ 3.6.11</a> | 16 Aug 2017 |
| <a href="#">RabbitMQ 3.6.10</a> | 25 May 2017 |
| <a href="#">RabbitMQ 3.6.9</a>  | 29 Mar 2017 |



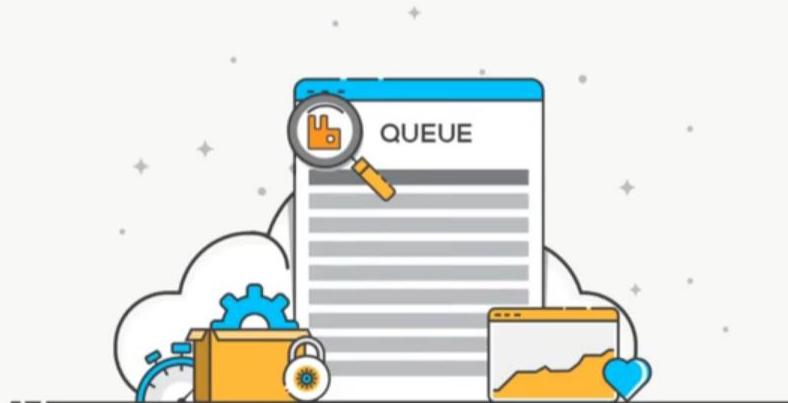
## rabbitmq-plugins enable rabbitmq\_management

Secure | https://www.cloudamqp.com

CloudAMQP Pricing Documentation Support Blog Login Sign Up

# RabbitMQ as a Service

Perfectly configured and optimized RabbitMQ clusters ready in 2 minutes.



Get a managed RabbitMQ server today

Part of the 

# Solutions to Challenges with Microservice Architectures



- **Spring Cloud**
- **Spring Cloud provides solutions to cloud enable your microservices.**
- **It leverages and builds on top of some of the Cloud solutions opensourced by Netflix (Netflix OSS).**

# Spring Cloud



The screenshot shows a web browser window with multiple tabs open. The active tab is titled "Spring Cloud" and has the URL [projects.spring.io/spring-cloud/](https://projects.spring.io/spring-cloud/). The page content is as follows:

**spring** PROJECTS DOCS GUIDES PROJECTS BLOG QUESTIONS 🔍

**PROJECTS**

## Spring Cloud

Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems (e.g. configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, one-time tokens, global locks, leadership election, distributed sessions, cluster state). Coordination of distributed systems leads to boiler plate patterns, and using Spring Cloud developers can quickly stand up services and applications that implement those patterns. They will work well in any distributed environment, including the developer's own laptop, bare metal data centres, and managed platforms such as Cloud Foundry.

[QUICK START](#)



# Spring Cloud

- Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems (e.g. configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, one-time tokens, global locks, leadership election, distributed sessions, cluster state).
- Coordination of distributed systems leads to boiler plate patterns, and using Spring Cloud developers can quickly stand up services and applications that implement those patterns.
- They will work well in any distributed environment, including the developer's own laptop, bare metal data centres, and managed platforms such as Cloud Foundry.



# Spring Cloud Main Projects

---

- **Spring Cloud Config**
- Centralized external configuration management backed by a git repository.
- The configuration resources map directly to Spring Environment but could be used by non-Spring applications if desired.
- **Spring Cloud Netflix**
- Integration with various Netflix OSS components (Eureka, Hystrix, Zuul, Archaius, etc.).



# Spring Cloud Main Projects

---

- Spring Cloud Bus
- An event bus for linking services and service instances together with distributed messaging. Useful for propagating state changes across a cluster (e.g. config change events).
- Spring Cloud Cloudfoundry
- Integrates your application with Pivotal Cloud Foundry. Provides a service discovery implementation and also makes it easy to implement SSO and OAuth2 protected resources.



# Spring Cloud Main Projects

---

- Spring Cloud Open Service Broker
- Provides a starting point for building a service broker that implements the Open Service Broker API.
- Spring Cloud Cluster
- Leadership election and common stateful patterns with an abstraction and implementation for Zookeeper, Redis, Hazelcast, Consul.



# Spring Cloud Main Projects

---

- Spring Cloud Consul
- Service discovery and configuration management with Hashicorp Consul.
- Spring Cloud Security
- Provides support for load-balanced OAuth2 rest client and authentication header relays in a Zuul proxy.
- Spring Cloud Sleuth
- Distributed tracing for Spring Cloud applications, compatible with Zipkin, HTrace and log-based (e.g. ELK) tracing.



# Spring Cloud Main Projects

---

- **Spring Cloud Data Flow**
- **A cloud-native orchestration service for composable microservice applications on modern runtimes. Easy-to-use DSL, drag-and-drop GUI, and REST-APIs together simplifies the overall orchestration of microservice based data pipelines.**
- **Spring Cloud Stream**
- **A lightweight event-driven microservices framework to quickly build applications that can connect to external systems. Simple declarative model to send and receive messages using Apache Kafka or RabbitMQ between Spring Boot apps.**



# Spring Cloud Main Projects

---

- **Spring Cloud Stream App Starters**
- Spring Cloud Stream App Starters are Spring Boot based Spring Integration applications that provide integration with external systems.
- **Spring Cloud Task**
- A short-lived microservices framework to quickly build applications that perform finite amounts of data processing. Simple declarative for adding both functional and non-functional features to Spring Boot apps.



# Spring Cloud Main Projects

---

- **Spring Cloud Task App Starters**
- Spring Cloud Task App Starters are Spring Boot applications that may be any process including Spring Batch jobs that do not run forever, and they end/stop after a finite period of data processing.
- **Spring Cloud Zookeeper**
- Service discovery and configuration management with Apache Zookeeper.



# Spring Cloud Main Projects

- **Spring Cloud AWS**
- Easy integration with hosted Amazon Web Services. It offers a convenient way to interact with AWS provided services using well-known Spring idioms and APIs, such as the messaging or caching API. Developers can build their application around the hosted services without having to care about infrastructure or maintenance.
- **Spring Cloud Connectors**
- Makes it easy for PaaS applications in a variety of platforms to connect to backend services like databases and message brokers (the project formerly known as "Spring Cloud").



# Spring Cloud Main Projects

---

- **Spring Cloud Starters**
  - Spring Boot-style starter projects to ease dependency management for consumers of Spring Cloud.  
(Discontinued as a project and merged with the other projects after Angel.SR2.)
- **Spring Cloud CLI**
  - Spring Boot CLI plugin for creating Spring Cloud component applications quickly in Groovy
- **Spring Cloud Contract**
  - Spring Cloud Contract is an umbrella project holding solutions that help users in successfully implementing the Consumer Driven Contracts approach.



# Spring Cloud Main Projects

---

- **Spring Cloud Gateway**
- Spring Cloud Gateway is an intelligent and programmable router based on Project Reactor.
- **Spring Cloud OpenFeign**
- Spring Cloud OpenFeign provides integrations for Spring Boot apps through autoconfiguration and binding to the Spring Environment and other Spring programming model idioms.
- **Spring Cloud Pipelines**
- Spring Cloud Pipelines provides an opinionated deployment pipeline with steps to ensure that your application can be deployed in zero downtime fashion and easily rolled back if something goes wrong.



# Spring Cloud Main Projects

---

- **Spring Cloud Function**
- Spring Cloud Function promotes the implementation of business logic via functions. It supports a uniform programming model across serverless providers, as well as the ability to run standalone (locally or in a PaaS).
- .



# Spring Cloud Config Server

- Spring Cloud Config provides server and client-side support for externalized configuration in a distributed system.
- With the Config Server you have a central place to manage external properties for applications across all environments.
- The concepts on both client and server map identically to the Spring Environment and PropertySource abstractions, so they fit very well with Spring applications, but can be used with any application running in any language.



# Spring Cloud Config Server

- As an application moves through the deployment pipeline from dev to test and into production you can manage the configuration between those environments and be certain that applications have everything they need to run when they migrate.
- The default implementation of the server storage backend uses git so it easily supports labelled versions of configuration environments, as well as being accessible to a wide range of tooling for managing the content. It is easy to add alternative implementations and plug them in with Spring configuration.



## Features

---

- Spring Cloud Config Server features:
- HTTP, resource-based API for external configuration (name-value pairs, or equivalent YAML content)
- Encrypt and decrypt property values (symmetric or asymmetric)
- Embeddable easily in a Spring Boot application using @EnableConfigServer



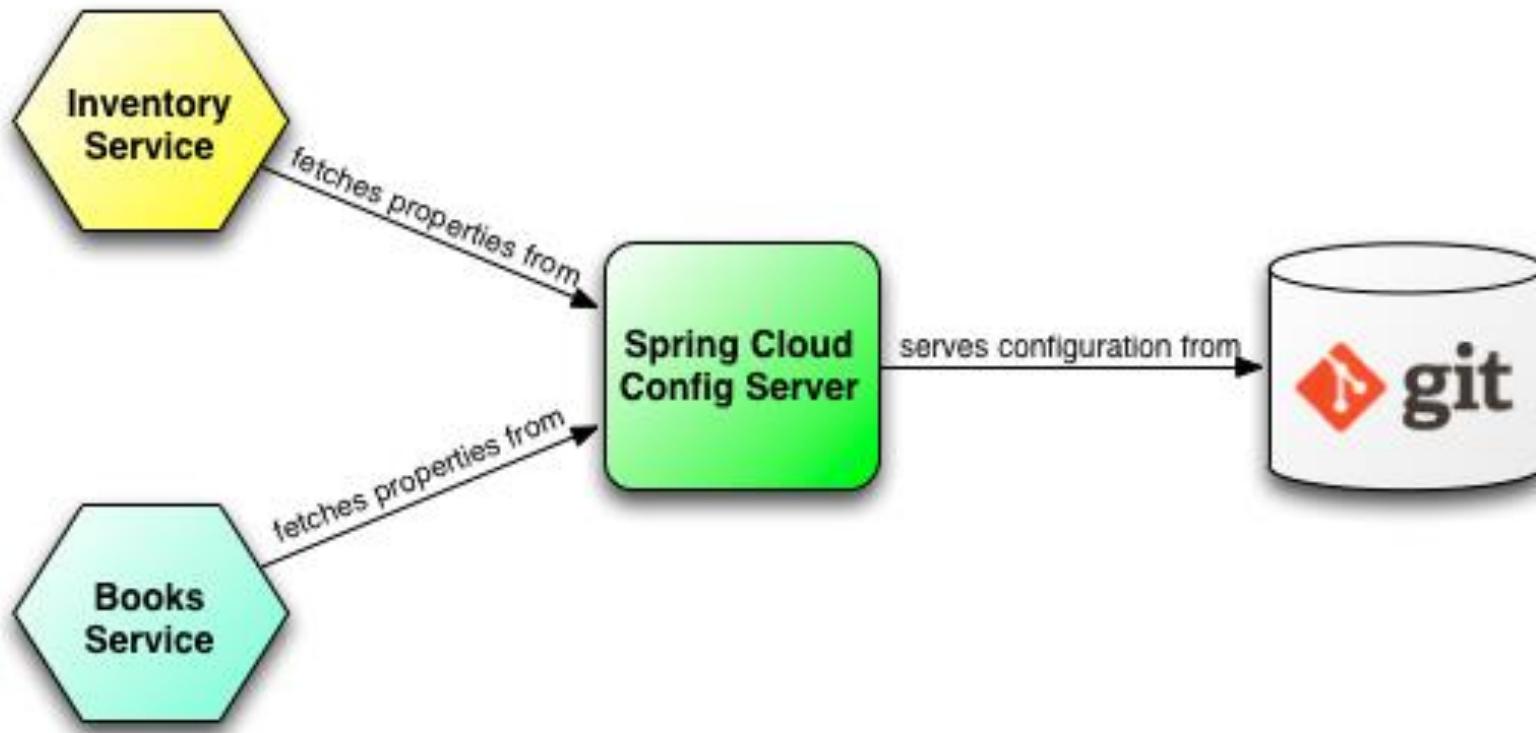
## Client side Features

---

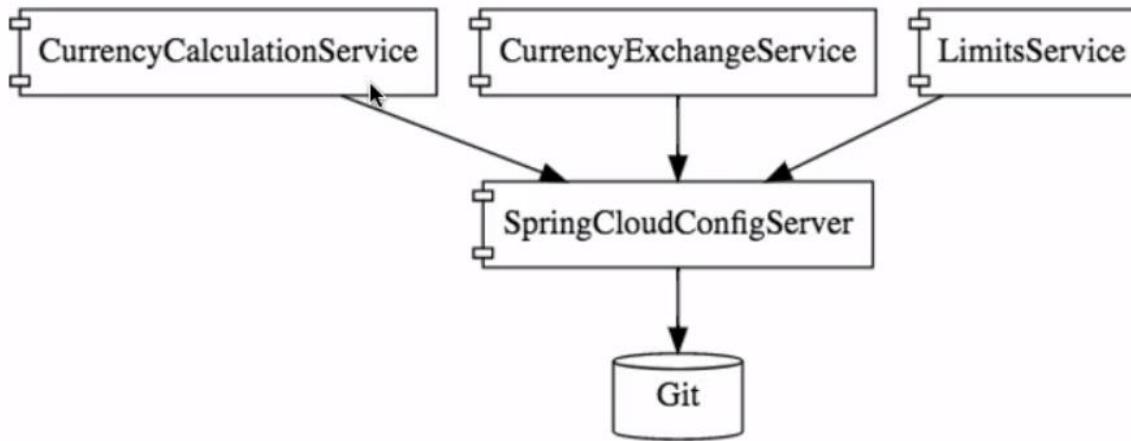
- Config Client features (for Spring applications):
- Bind to the Config Server and initialize Spring Environment with remote property sources
- Encrypt and decrypt property values (symmetric or asymmetric)



# Spring Cloud Config Server



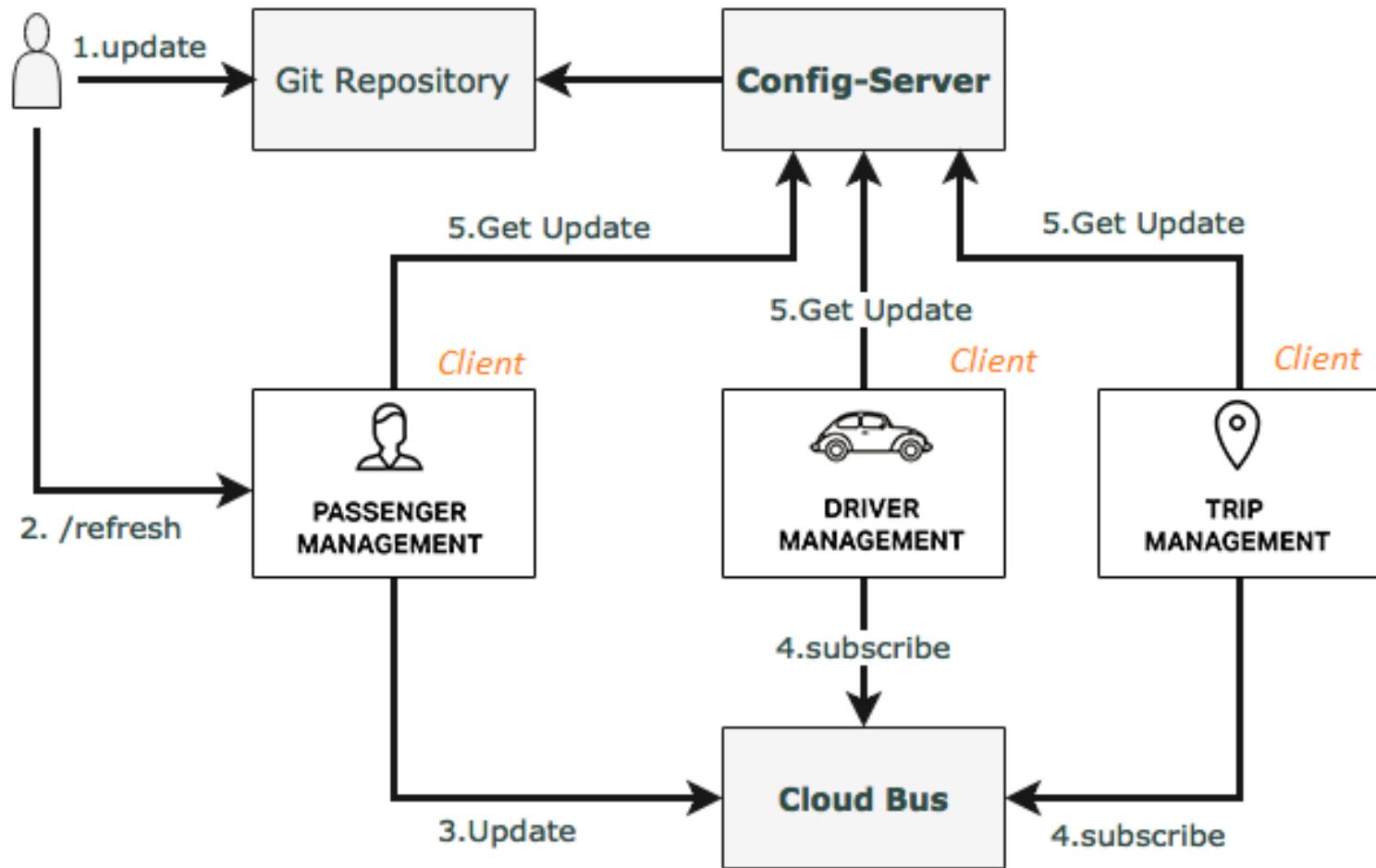
# Spring Cloud



*Spring Cloud Config Server*

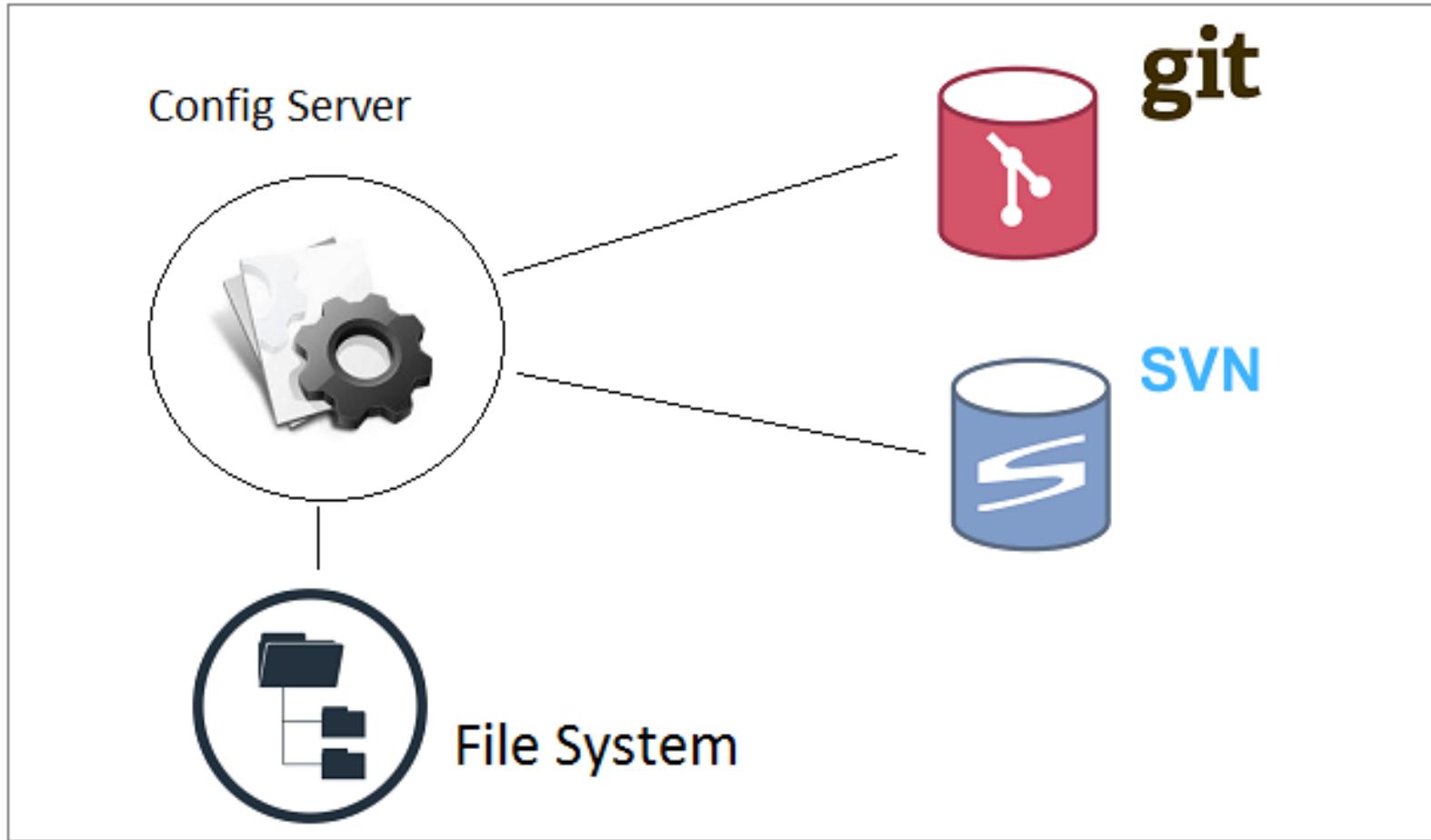


# Spring Cloud Bus





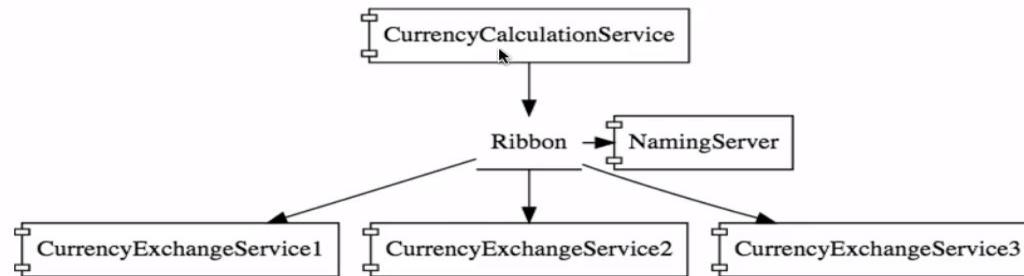
# How does Config Server Store Data



## DYNAMIC SCALE UP AND DOWN

- Naming Server (Eureka)
- Ribbon (Client Side Load Balancing)
- Feign (Easier REST Clients)

# Spring Cloud



*Ribbon Load Balancing*

## DYNAMIC SCALE UP AND DOWN

- Naming Server (Eureka)
- Ribbon (Client Side Load Balancing)
- Feign (Easier REST Clients)

## VISIBILITY AND MONITORING

- Zipkin Distributed Tracing
- Netflix API Gateway

# Important Spring Cloud Modules



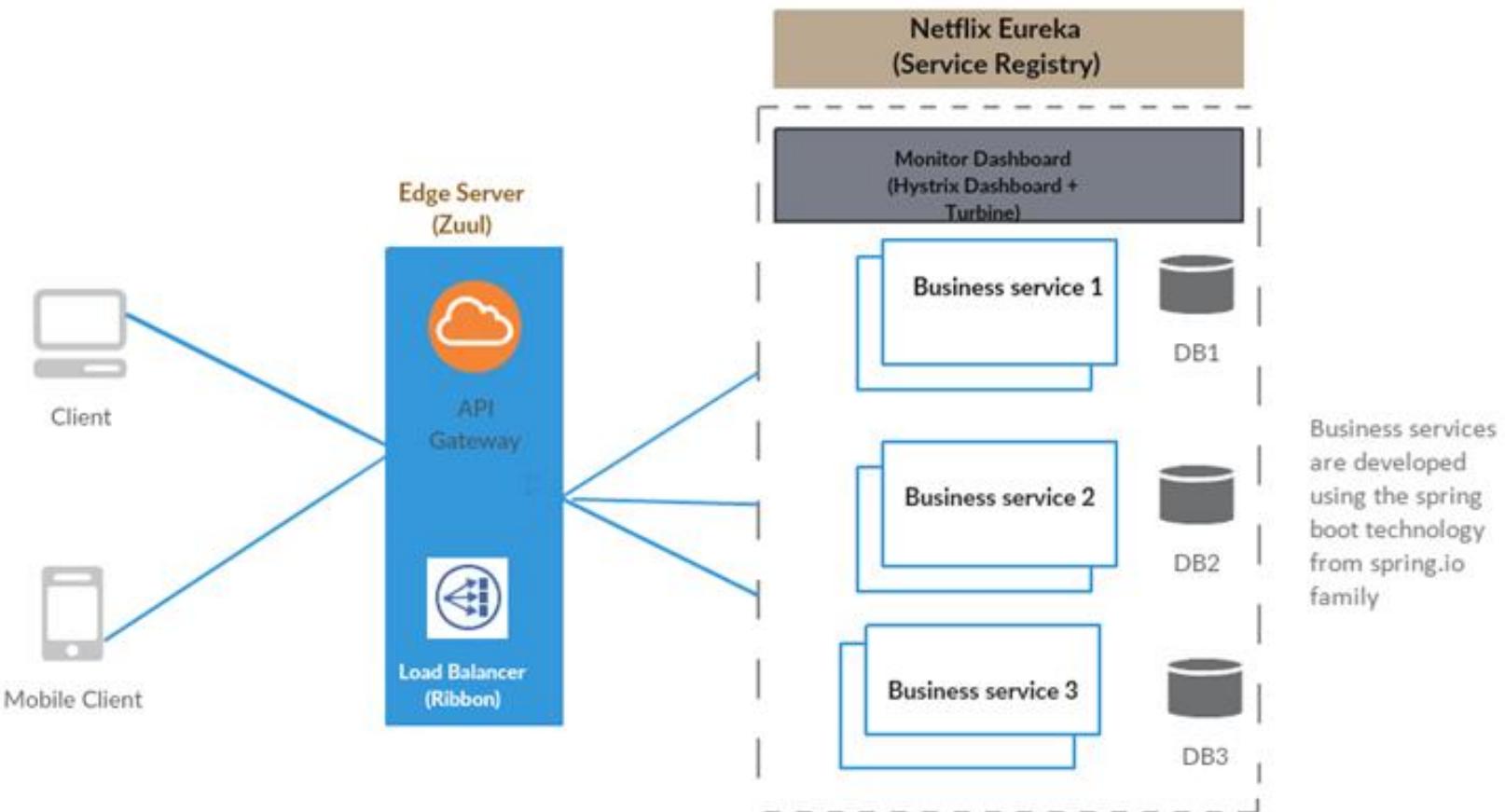
- Dynamic Scale Up and Down. Using a combination of
- Naming Server (Eureka)
- Ribbon (Client Side Load Balancing)
- Feign (Easier REST Clients)
- Visibility and Monitoring with
- Zipkin Distributed Tracing
- Netflix API Gateway
- Configuration Management with
- Spring Cloud Config Server
- Fault Tolerance with
- Hystrix

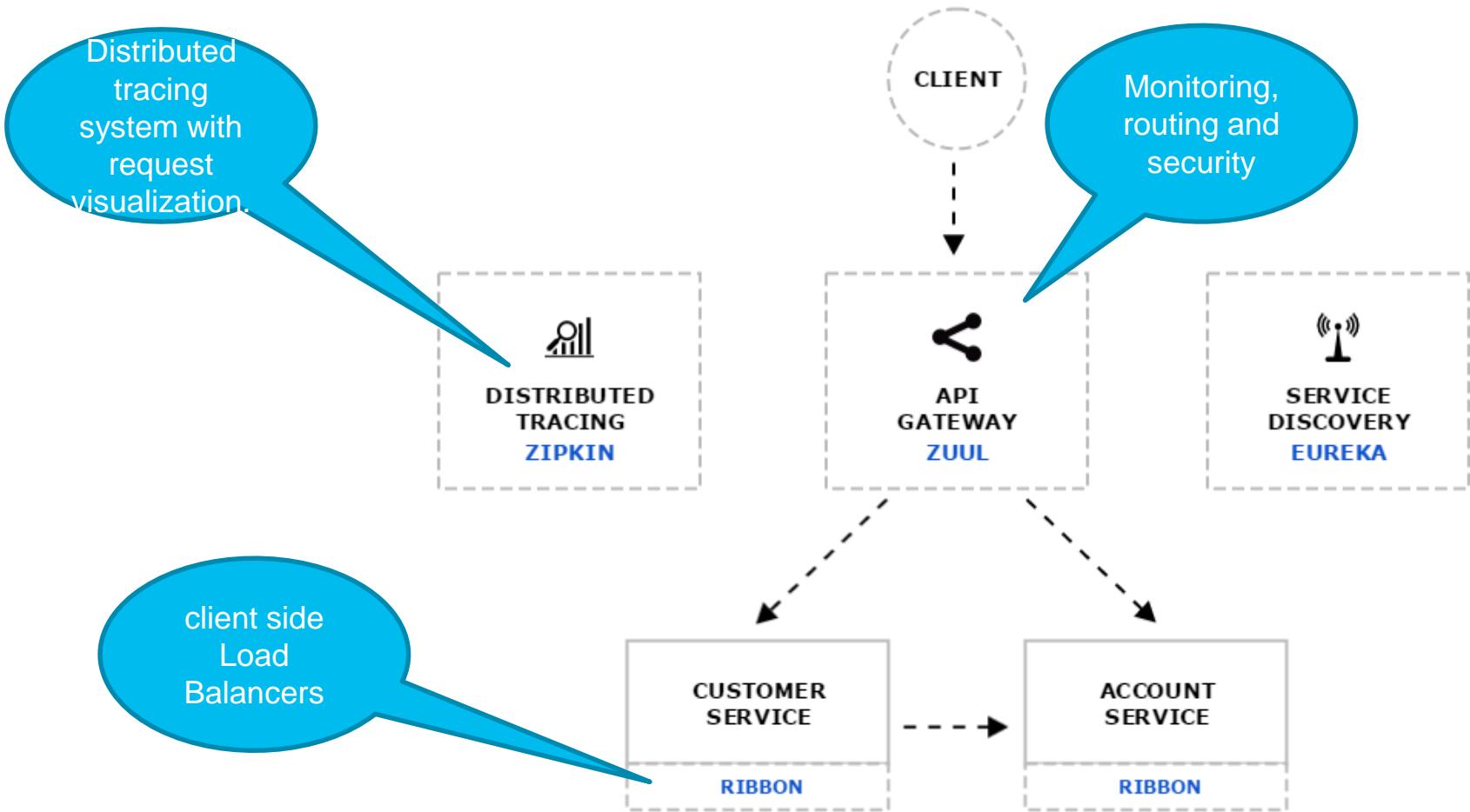


# Standardized ports

## Ports

Application	Port
Limits Service	8080, 8081, ...
Spring Cloud Config Server	8888
Currency Exchange Service	8000, 8001, 8002, ..
Currency Conversion Service	8100, 8101, 8102, ...
Netflix Eureka Naming Server	8761
Netflix Zuul API Gateway Server	8765
Zipkin Distributed Tracing Server	9411







Create stand-alone Spring applications

Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)

Provide opinionated 'starter' POMs to simplify your Maven configuration

Automatically configure Spring whenever possible

Provide production-ready features such as metrics, health checks and externalized configuration

Absolutely no code generation and no requirement for XML configuration

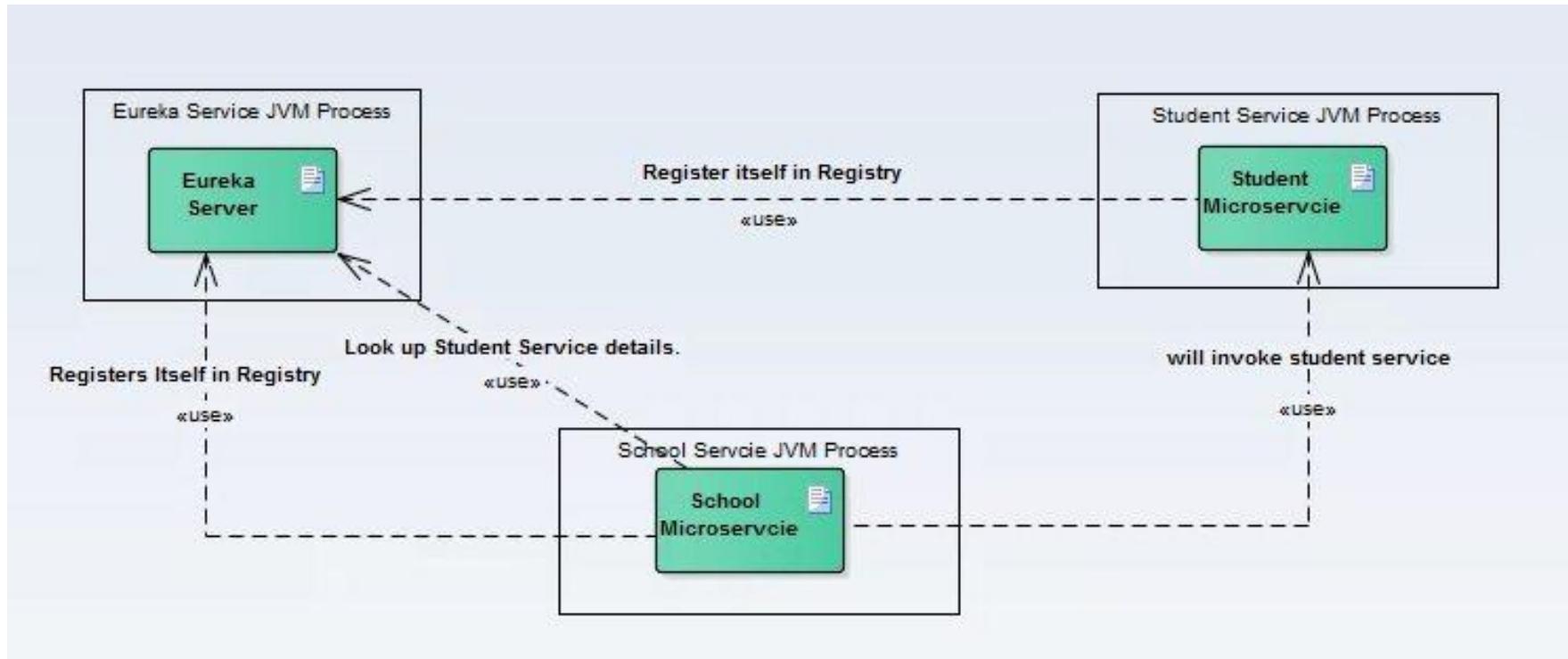


# Why Eureka Server

---

- In the distributed computing are there is a concept called 'Service registration and discovery' where one dedicated server is responsible to maintain the registry of all the Microservice that has been deployed and removed. This will act like a phone book of all other applications/microservices.

# Spring Cloud Service Discovery with Netflix Eureka





# Server Configuration

---

- server:
- port: \${PORT:8761} # Indicate the default PORT where this service will be started
- 
- eureka:
- client:
  - registerWithEureka: false #telling the server not to register himself in the service registry
  - fetchRegistry: false
- server:
- waitTimeInMsWhenSyncEmpty: 0 #wait time for subsequent sync



# Client Configuration

---

- server:
- port: 8098 #default port where the service will be started
- 
- eureka: #tells about the Eureka server details and its refresh time
- instance:
- leaseRenewalIntervalInSeconds: 1
- leaseExpirationDurationInSeconds: 2
- client:



# Client Configuration

---

- serviceUrl:
- defaultZone: http://127.0.0.1:8761/eureka/
- healthcheck:
- enabled: true
- lease:
- duration: 5
- 
- spring:
- application:
- name: student-service #current service name to be used by the eureka server
-



# Client Configuration

---

- management:
- security:
- enabled: false #disable the spring security on the management endpoints like /env, /refresh etc.
- 
- logging:
- level:
- com.example.howtодоinjava: DEBUG

# Consul Service Registration and Discovery Example

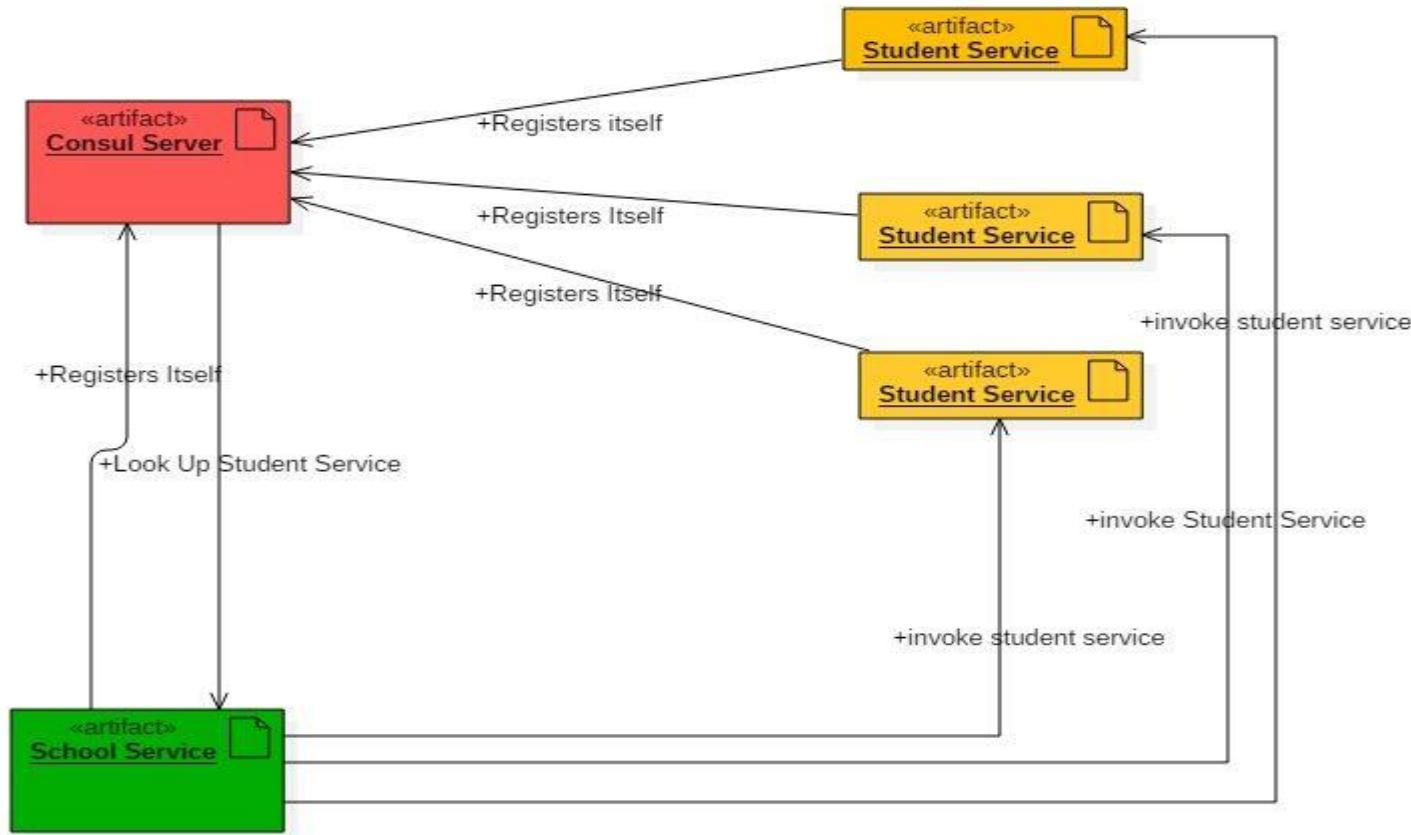


- Consul provides multiple features like service discovery, configuration management, health checking and key-value store etc.

# Consul Service Registration and Discovery Example



- **Consul Agent** – running on localhost acting as discovery/registry server functionality.



# Configuring Consul in Local workstation



- Download from Consul portal. Choose particular package based on the operating System. Once downloaded the zip, we need to unzip it to desired place.
- Start Consul Agent in local workstation – The Zip file that we have unzipped, has only one exe file called `consul.exe`. We will start a command prompt here and use below command to start the agent.
- `consul agent -server -bootstrap-expect=1 -data-dir=consul-data -ui -bind=192.168.99.1`

# Configuring Consul in Local workstation



- Make sure you enter the correct bind address, it would be different depending on the LAN settings. Do a ipconfig in command prompt to know your IPv4 address and use it here.

A screenshot of a Windows Command Prompt window titled "cmd C:\Windows\system32\cmd.exe". The window shows the command "F:\Study\installations\consul\_0.9.0\_windows\_amd64>ipconfig" entered and ready to be executed.

# Configuring Consul in Local workstation



```
C:\Windows\system32\cmd.exe - consul agent -server -bootstrap-expect=1 -data-dir=consul-data -ui -bind=192.168.6.1

F:\Study\installations\consul_0.9.0_windows_amd64>consul agent -server -bootstrap-expect=1 -data-dir=consul-data -ui -bind=192.168.6.1
==> WARNING: BootstrapExpect Mode is specified as 1; this is the same as Bootstrap mode.
==> WARNING: Bootstrap mode enabled! Do not enable unless necessary
==> Starting Consul agent...
==> Consul agent running!
  Version: 'v0.9.0'
    Node ID: 'f8db8d09-ac67-e997-9306-aeb20e4ecd3e'
    Node name: 'Sajal-HP'
    Datacenter: 'dc1'
      Server: true <Bootstrap: true>
    Client Addr: 127.0.0.1 <HTTP: 8500, HTTPS: -1, DNS: 8600>
    Cluster Addr: 192.168.6.1 <LAN: 8301, WAN: 8302>
  Gossip encrypt: false, RPC-TLS: false, TLS-Incoming: false

==> Log data will now stream in as it occurs:

2017/07/20 13:31:42 [INFO] raft: Initial configuration <index=1>: [{Suffrage:Voter ID:192.168.6.1:8300 Address:192.168.6.1:8300}]
2017/07/20 13:31:42 [INFO] raft: Node at 192.168.6.1:8300 [Follower] entering Follower state <Leader: "">
2017/07/20 13:31:42 [INFO] serf: EventMemberJoin: Sajal-HP.dc1 192.168.6.1
2017/07/20 13:31:42 [WARN] serf: Failed to re-join any previously known node
2017/07/20 13:31:42 [INFO] serf: EventMemberJoin: Sajal-HP 192.168.6.1
2017/07/20 13:31:42 [INFO] consul: Handled member-join event for server "Sajal-HP.dc1" in area "wan"
2017/07/20 13:31:42 [INFO] consul: Adding LAN server Sajal-HP <Addr: tcp/192.168.6.1:8300> <DC: dc1>
2017/07/20 13:31:42 [WARN] serf: Failed to re-join any previously known node
2017/07/20 13:31:42 [INFO] agent: Started DNS server 127.0.0.1:8600 <udp>
2017/07/20 13:31:42 [INFO] agent: Started DNS server 127.0.0.1:8600 <tcp>
2017/07/20 13:31:42 [INFO] agent: Started HTTP server on 127.0.0.1:8500
2017/07/20 13:31:49 [ERR] agent: failed to sync remote state: No cluster leader
2017/07/20 13:31:49 [ERR] http: Request GET /v1/catalog/services?wait=2s&index=169, error: No cluster leader from=127.0.0.1:53785
2017/07/20 13:31:50 [ERR] http: Request GET /v1/catalog/services, error: No cluster leader from=127.0.0.1:53789
2017/07/20 13:31:51 [ERR] http: Request GET /v1/catalog/services, error: No cluster leader from=127.0.0.1:53792
2017/07/20 13:31:52 [WARN] raft: Heartbeat timeout from "" reached, starting election
2017/07/20 13:31:52 [INFO] raft: Node at 192.168.6.1:8300 [Candidate] entering Candidate state in term 66
2017/07/20 13:31:52 [INFO] raft: Election won. Tally: 1
2017/07/20 13:31:52 [INFO] raft: Node at 192.168.6.1:8300 [Leader] entering Leader state
2017/07/20 13:31:52 [INFO] consul: cluster leadership acquired
2017/07/20 13:31:52 [INFO] consul: New leader elected: Sajal-HP
2017/07/20 13:31:52 [WARN] agent: Check 'service:student-service-9099' is now critical
2017/07/20 13:31:52 [WARN] agent: Check 'service:student-service-9097' is now critical
2017/07/20 13:31:54 [INFO] agent: Synced check 'service:student-service-9097'
2017/07/20 13:31:54 [INFO] agent: Synced check 'service:school-service-8098'
2017/07/20 13:31:54 [INFO] agent: Synced check 'service:student-service-9099'
2017/07/20 13:31:54 [INFO] agent: Synced check 'service:student-service-9098'
```

# Configuring Consul in Local workstation



- **Test whether Consul Server is running** – Consul runs on default port and once agent started successfully, browse <http://localhost:8500/ui> and you should see a console screen like –

A screenshot of a web browser displaying the Consul UI at [#/dc1/services](http://localhost:8500/ui). The browser's address bar shows the URL. The page has a header with tabs: SERVICES (highlighted in pink), NODES, KEY/VALUE, ACL, DC1 (highlighted in green with a dropdown arrow), and a settings gear icon. Below the tabs are filters: 'Filter by name' (text input), 'any status' (dropdown menu), and 'EXPAND' (button). A main table lists services. One row is visible, showing a green bar icon, the service name 'consul', and the status '1 passing'. The rest of the table is cut off by a horizontal scroll bar.



# Consul Client

---

- **Create Student Project**
- Create a Spring boot project from initializer portal with four dependencies i.e.
- Actuator
- Web
- Rest Repositories
- Consul Discovery



# Service Configuration

- `server.port=9098` – will start the service in default 9098 port.
- `spring.application.name: student-service` – will registers itself in consul server using student-service tag and also other services will lookup this service with this name itself.
- `management.security.enabled=false` – is not actually required for this exercise, but it will disable spring security in the management endpoints provided by actuator module.

# Hystrix Circuit Breaker Pattern – Spring Cloud



- It is generally required to enable fault tolerance in the application where some underlying service is down/throwing error permanently, we need to fall back to different path of program execution automatically.
- This is related to distributed computing style of Eco system using lots of underlying Microservices.
- This is where circuit breaker pattern helps and Hystrix is an tool to build this circuit breaker.

# What is Circuit Breaker Pattern?



- If we design our systems on microservice based architecture, we will generally develop many Microservices and those will interact with each other heavily in achieving certain business goals.
- Now, all of us can assume that this will give expected result if all the services are up and running and response time of each service is satisfactory.

# What is Circuit Breaker Pattern?



- Now what will happen if any service, of the current Eco system, has some issue and stopped servicing the requests.
- It will result in timeouts/exception and the whole Eco system will get unstable due to this single point of failure.

# What is Circuit Breaker Pattern?



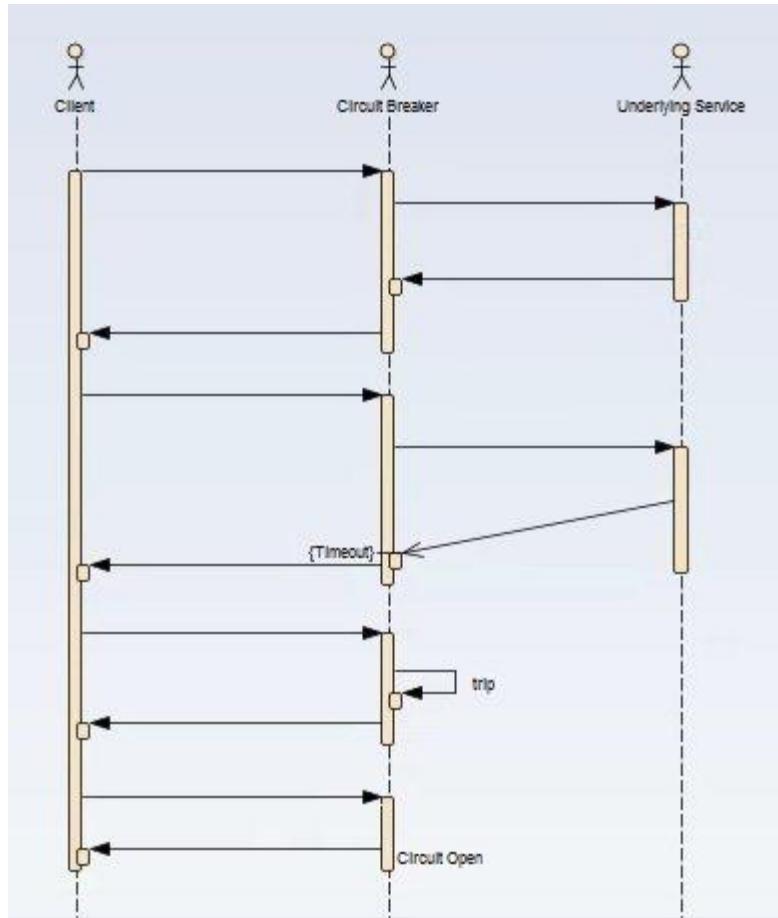
- Here circuit breaker pattern comes handy and it redirects traffic to a fall back path once it sees any such scenario.
- Also it monitors the defective service closely and restore the traffic once the service came back to normalcy.

# What is Circuit Breaker Pattern?



- So circuit breaker is a kind of a wrapper of the method which is doing the service call and it monitors the service health and once it gets some issue, the circuit breaker trips and all further calls goto the circuit breaker fall back and finally restores automatically once the service came back !! That's cool right?

# What is Circuit Breaker Pattern?



# Hystrix Dashboard



- **<http://localhost:9091/hystrix>**
- **<http://localhost:9091/actuator/hystrix.stream>** – It's a continuous stream that Hystrix generates. It is just a health check result along with all the service calls that are being monitored by Hystrix. Sample output will look like in browser –



Eureka | localhost:8765/ | Hystrix Circuit E | localhost:9091/ | Consul by Hash | Spring Initializr | New Tab | localhost:9091/ | Hystrix Dashboard | + | - | X

localhost:9091/hystrix

Insert title here Empire New Tab How to use Assertion Browser Automation node.js - How can I fi Freelancer-dev-8104 Courses



## Hystrix Dashboard

<http://localhost:9091/actuator/hystrix.stream>

Cluster via Turbine (default cluster): http://turbine-hostname:port/turbine.stream  
Cluster via Turbine (custom cluster): http://turbine-hostname:port/turbine.stream?cluster=[clusterName]  
Single Hystrix App: http://hystrix-app:port/actuator/hystrix.stream

Delay:  ms Title:





Eureka | localhost:8765/ | Hystrix Circuit E | localhost:9091/ | Consul by Hash | Spring Initializr | New Tab | localhost:9091/ | Hystrix Monitor | + | - | X

localhost:9091/hystrix/monitor?stream=http%3A%2F%2Flocalhost%3A9091%2Factuator%2Fhystrix.stream

Apps Insert title here Empire New Tab How to use Assertion Browser Automation node.js - How can I fi Freelancer-dev-8104 Courses



## Hystrix Stream: http://localhost:9091/actuator/hystrix.stream

### Circuit



Success | Short-Circuited | Bad Request | Timeout | Rejected | Failure | Error %

### Thread Pools



# How to Deploy Spring Boot Application to Cloud Foundry Platform

---



- **Cloud Foundry** is one of the Cloud Providers. It is a PaaS service where we can easily deploy and manage our applications and the Cloud Foundry will take care of the rest of the cloud based offerings like scalability, high availability etc.

# What is Cloud Foundry



- Cloud Foundry is an open-source platform as a service (PaaS) that provides you with a choice of clouds, developer frameworks, and application services.
- It is open source and it is governed by the Cloud Foundry Foundation.
- The original Cloud Foundry was developed by VMware and currently it is managed by Pivotal, a joint venture company by GE, EMC and VMware.

# What is Cloud Foundry



- Now since Cloud Foundry is open source product many popular organizations currently provides this platform separately and below are the list of current certified providers.
- Pivotal Cloud Foundry
- IBM Bluemix
- HPE Helion Stackato 4.0
- Atos Canopy
- CenturyLink App Fog
- GE Predix
- Huawei FusionStage
- SAP Cloud Platform
- Swisscom Application Cloud

# Cloud Foundry Installation for Windows



- Download the [CF Windows installer](#). It will prompt for the download. Save the zip file distribution.
- Unpack the zip file to a suitable place in your workstation.
- After successfully **unzip** operation, double click on the cf CLI executable.
- When prompted, click **Install**, then Close. Here are the sample steps for the same. This is very straight forward, you can select the default values.

# Cloud Foundry Installation for Windows



- Verify the installation by opening a terminal window and type cf. If your installation was successful, the cf CLI help listing appears. This indicates that you are ready to go with any cloud foundry platform from your local workstation.

# Setup PWS Console



Secure | https://account.run.pivotal.io/z/uaa/sign-up

**Pivotal®**

Create your Pivotal Account

First name

Last name

Email address

Password

Password confirmation

**Sign Up**

Already have an account? [Sign In](#)

# Login and logout from PWS Console using CLI



- Login to PWS – We will use **cf login -a api.run.pivotal.io** command to login to pivotal web service console from CLI tool that we have installed in our local workstation. It will logon the CLI tool to PWS platform so that we can deploy and manage our applications from our workstation. After giving command, it will ask for registered email and password and once provided successfully, it will logon to the platform.
- Logout from PWS Console – We will use command **cf logout** to logout from the platform, once we have all the work done for that session.



# Create Database in cloud foundry

Pivotal Account | M Inbox (1,408) - parameswarib | R Welcome to Rediffmail | Pivotal Web Services | Spring Initializr | The HAL Browser (for Spring) | +

Apps Insert title here Empire New Tab How to use Assertion Browser Automation node.js - How can I fi Freelancer-dev-8104 Courses

press / eswaribala@rediffmail.com

Pivotal Web Services Search apps, services, spaces, & orgs

Home / eswaribala-org / development / kyc-cf

APP kyc-cf Starting

VIEW APP

Overview Service (1) Route (1) Logs Tasks Settings Buildpack: N/A

Bound Services

BIND SERVICE NEW SERVICE

Service	Instance Name	Binding Name
ClearDB MySQL Database free - Spark DB	virtusa_2018db	:

GIVE FEEDBACK

Pivotal © 2018 Pivotal Software Inc. All rights reserved. Terms | Privacy  
Last login: 11/20/18 8:33 pm

Type here to search

21:09 ENG 20/11/2018

# Push Application to Console



- cf push kyc-cf -p target\spring-helloworld-cf-0.0.1-SNAPSHOT.jar

```
Administrator: Command Prompt - cf push kyc-cf -p F:\Scope_SpringBoot_2018\KYCCFService\target\KYCCFService-0.0.1-SNAPSHOT.jar
User:          eswaribala@rediffmail.com
Org:          eswaribala-org
Space:        development

C:\WINDOWS\system32>cf push kyc-cf -p F:\Scope_SpringBoot_2018\KYCCFService\target\KYCCFService-0.0.1-SNAPSHOT.jar
Pushing app kyc-cf to org eswaribala-org / space development as eswaribala@rediffmail.com.
..
Getting app info...
Creating app with these attributes...
+ name:      kyc-cf
  path:      F:\Scope_SpringBoot_2018\KYCCFService\target\KYCCFService-0.0.1-SNAPSHOT.jar

  routes:
+   kyc-cf.cfapps.io

Creating app kyc-cf...
Mapping routes...
Comparing local files to remote cache...
Packaging files to upload...
Uploading files...
  320.00 KiB / 409.37 KiB [=====>-----] 78.17%
```



# Verify Application Deployed

Secure | https://console.run.pivotal.io/organizations/d9ba791e-2a9f-4107-a71b-7d7e464ac5d8

Pivotal Web Services

Search by App Name

sajal.chakraborty@gmail.com

Your org, "sajal.chakraborty", is still in trial. To get access to 25GB of memory and paid service plans, upgrade now

ORG  
sajal.chakraborty

SPACES  
development

Marketplace

Docs

Support

Tools

Blog

Status

ORG QUOTA  
sajal.chakraborty 1 GB / 2 GB Increase Quota 50%

Billing Statement

Space (1) Domains (2) Member (1) Settings

development	
APPS	SERVICES
1	0

+ Add a Space

50% of Org Quota

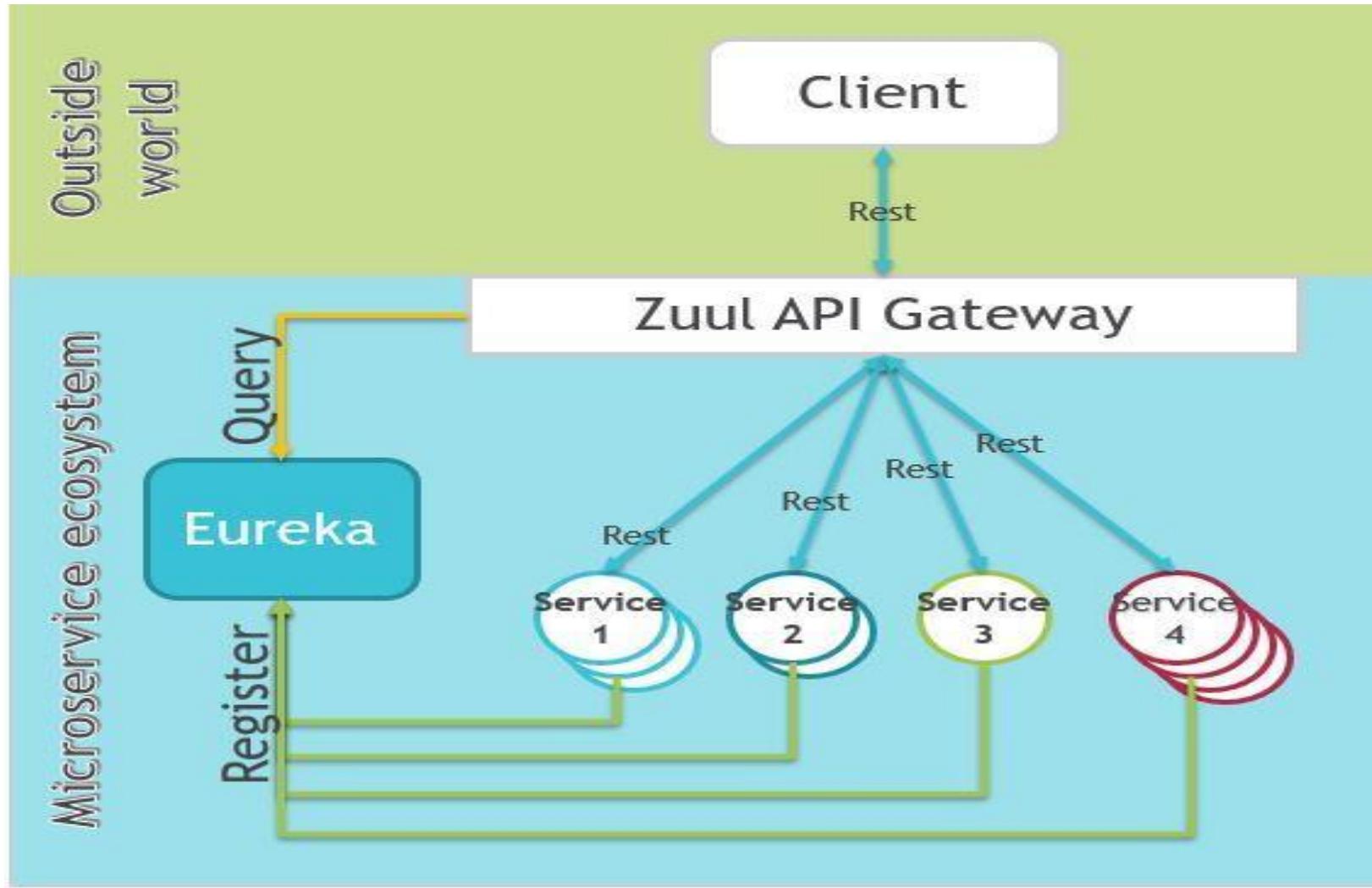
A screenshot of the Pivotal Web Services console showing the organization details for 'sajal.chakraborty'. A red circle highlights the number '1' under the 'APPS' column in the 'development' space, indicating one application has been deployed. The quota section shows 1 GB used out of 2 GB, which is 50% of the org's quota.

# What is Zuul?

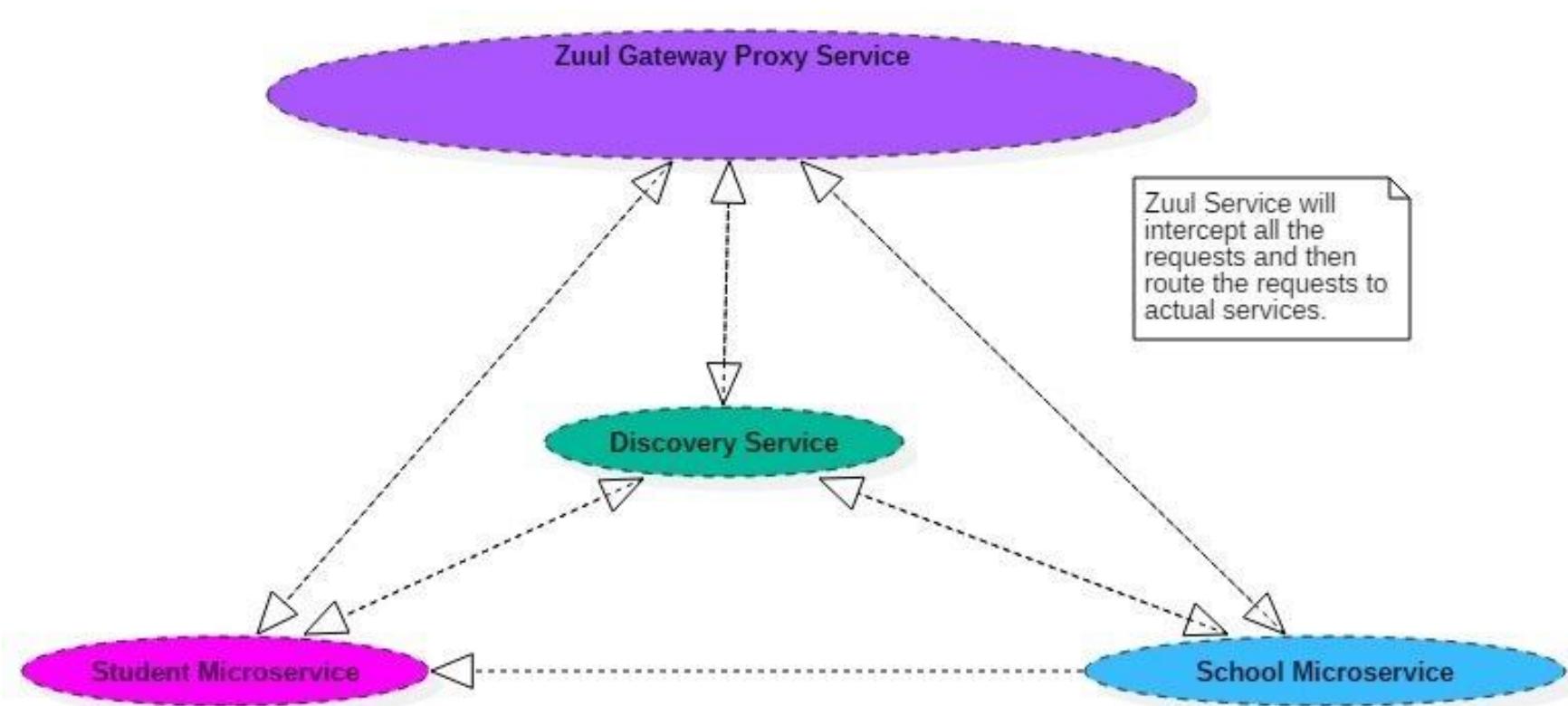


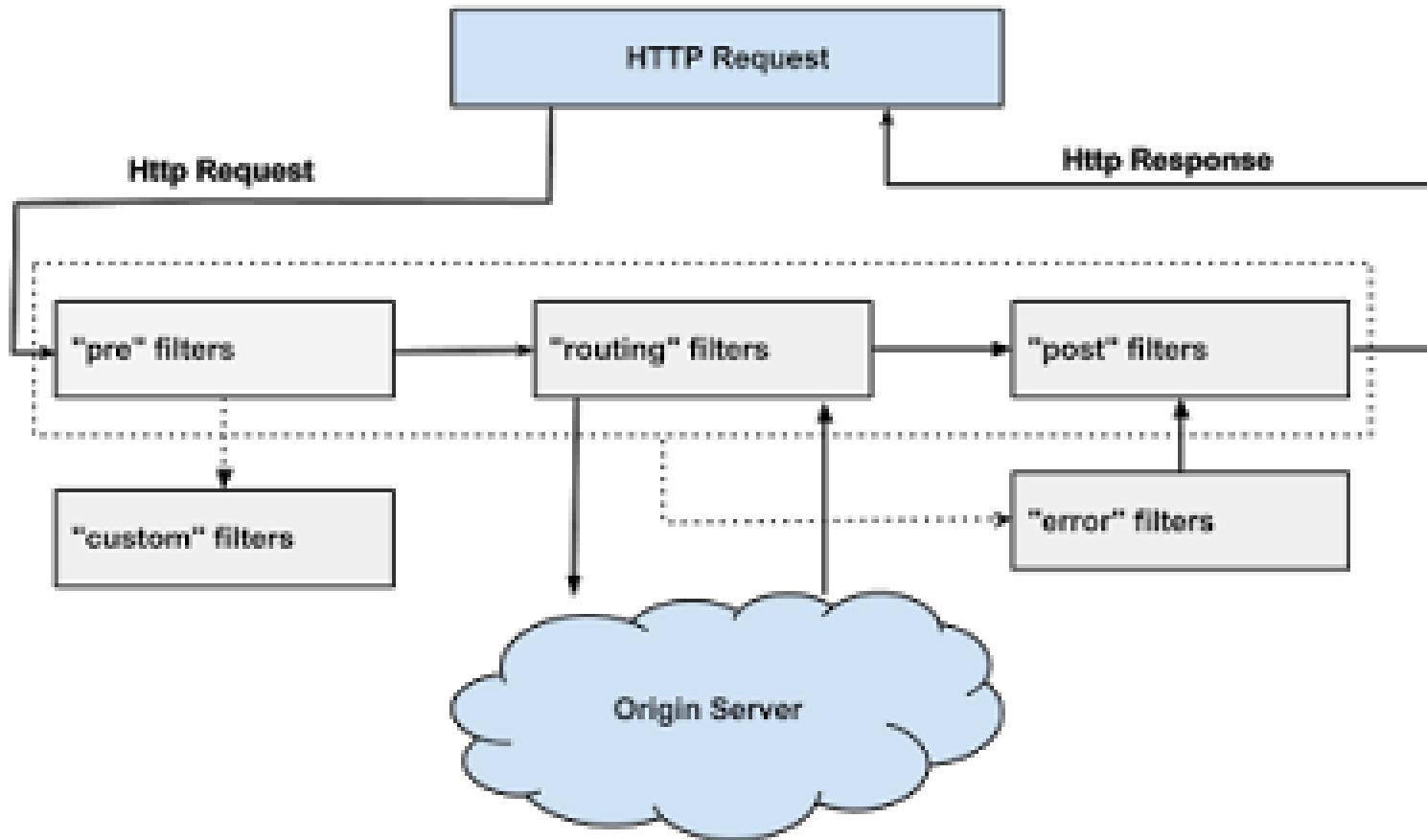
- *Zuul is the front door for all requests from devices and web sites to the backend of the Netflix streaming application.*
- *As an edge service application, Zuul is built to enable dynamic routing, monitoring, resiliency and security.*
- *It also has the ability to route requests to multiple Amazon Auto Scaling Groups as appropriate.*

# What is Zuul?



# What is Zuul?





# *Why did we build Zuul?*



- *The volume and diversity of Netflix API traffic sometimes results in production issues arising quickly and without warning.*
- *We need a system that allows us to rapidly change behaviour in order to react to these situations.*

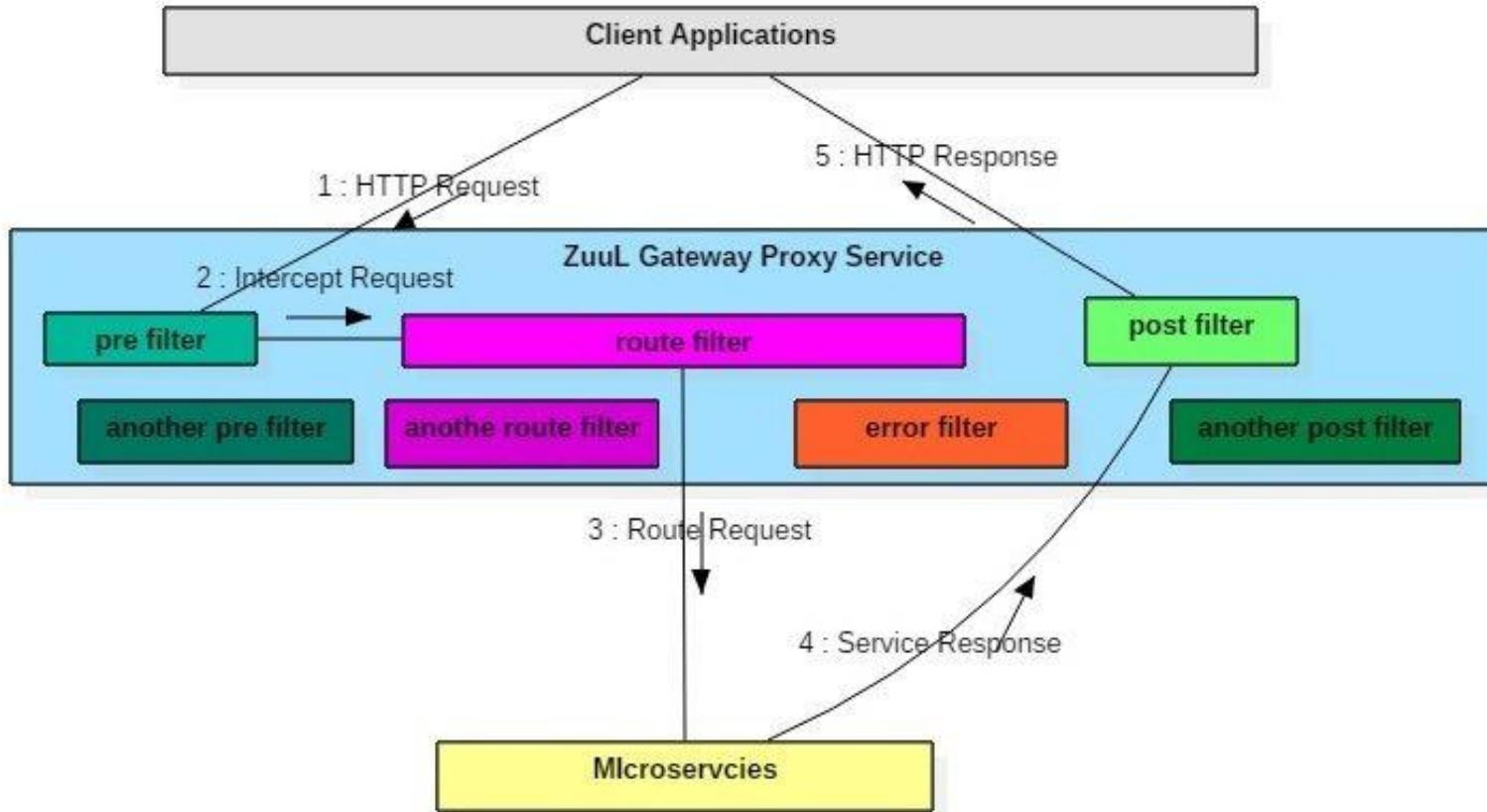


- Zuul has mainly four types of filters that enable us to intercept the traffic in different timeline of the request processing for any particular transaction.
- We can add any number of filters for a particular url pattern.
- **pre filters** – are invoked before the request is routed.
- **post filters** – are invoked after the request has been routed.
-



- **route filters** – are used to route the request.
- **error filters** – are invoked when an error occurs while handling the request.
-

# Zuul Components



# Zuul Filters Responsibilities



- Apply **microservice authentication and security** in the gateway layer to protect the actual services
- We can do **microservices insights and monitoring** of all the traffic that are going in to the ecosystem by enabling some logging to get meaningful data and statistics at the edge in order to give us an accurate view of production.

# Zuul Filters Responsibilities



- **Dynamic Routing** can route requests to different backend clusters as needed.
- We can do **runtime stress testing** by gradually increasing the traffic to a new cluster in order to gauge performance in many scenarios e.g. cluster has new H/W and network setup or that has new version of production code deployed.

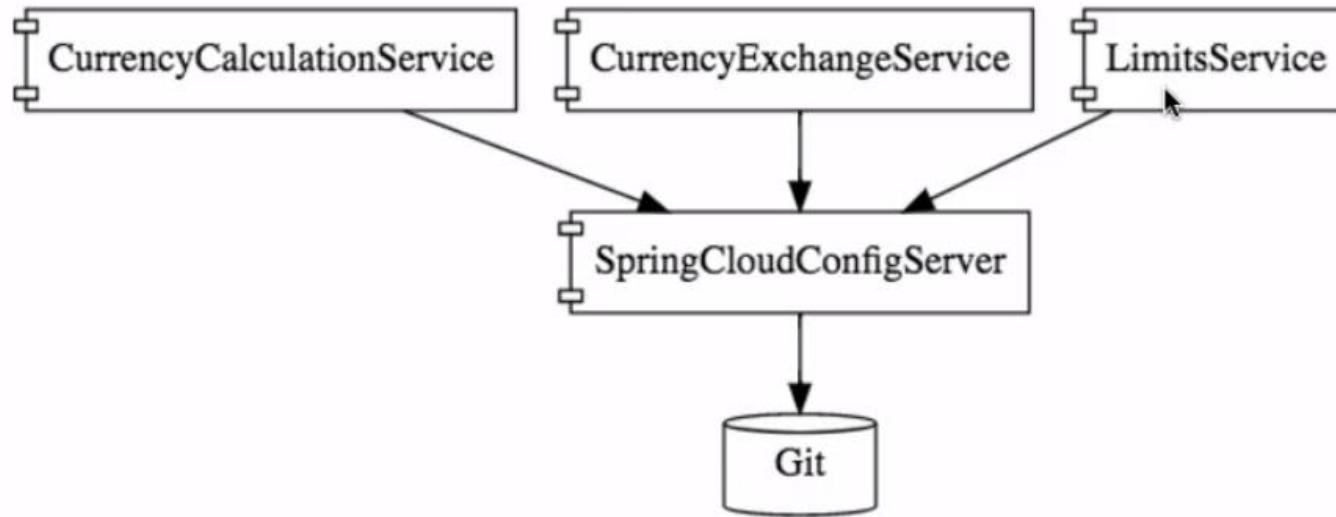
# Zuul Filters Responsibilities



- We can do **dynamic load shedding** i.e. allocating capacity for each type of request and dropping requests that go over the limit.
- We can apply **static response handling** i.e. building some responses directly at the edge instead of forwarding them to an internal cluster for processing.

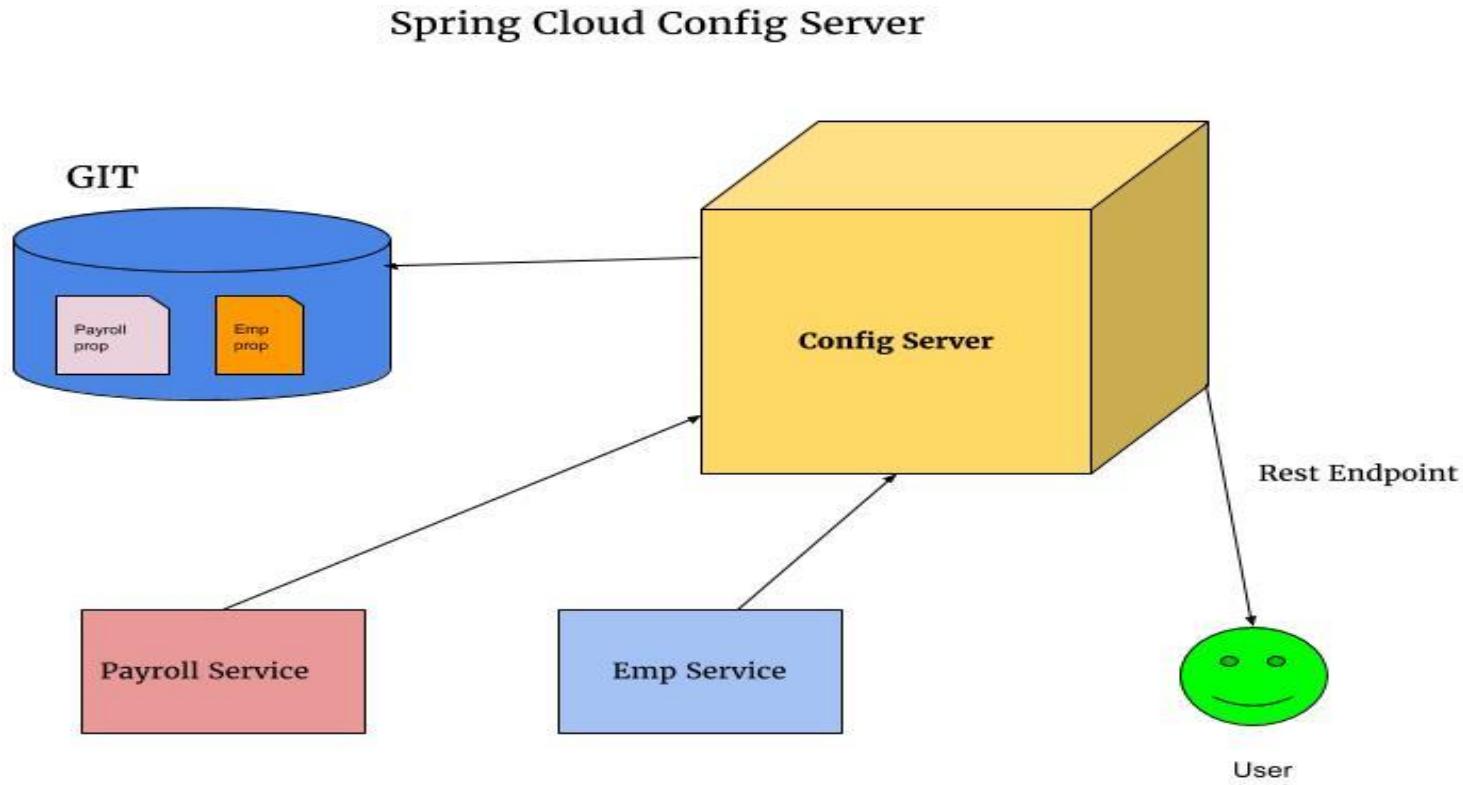


# Spring Cloud Config Server with git



*Spring Cloud Config Server*

# Spring Cloud Config Server with git





# Spring Cloud Config Server with git

SPRING INITIALIZR bootstrap your application now

Generate a  with  and Spring Boot

## Project Metadata

Artifact coordinates

Group

com.in28minutes.microservices

Artifact

spring-cloud-config-server

## Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Web, Security, JPA, Actuator, Devtools...

Selected Dependencies

DevTools ✕ Config Server ✕

Generate Project 

Don't know what to look for? Want more options? [Switch to the full version.](#)



# Spring Cloud Config Server with git

```
repository
Rangas-MacBook-Pro:03.microservices rangaraokaranam$ mkdir git-localconfig-repo
Rangas-MacBook-Pro:03.microservices rangaraokaranam$ cd git-localconfig-repo/
Rangas-MacBook-Pro:git-localconfig-repo rangaraokaranam$ git init
Initialized empty Git repository in /in28Minutes/git/spring-micro-services/03.microservices/git-localconfig-repo/.git/
Rangas-MacBook-Pro:git-localconfig-repo rangaraokaranam$ git add -A
Rangas-MacBook-Pro:git-localconfig-repo rangaraokaranam$ git commit -m "first commit"
[master (root-commit) 0898c54] first commit
  Committer: Ranga Rao Karanam <rangaraokaranam@Rangas-MacBook-Pro.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:
```

```
git config --global user.name "Your Name"
git config --global user.email you@example.com
```

After doing this, you may fix the identity used for this commit with:

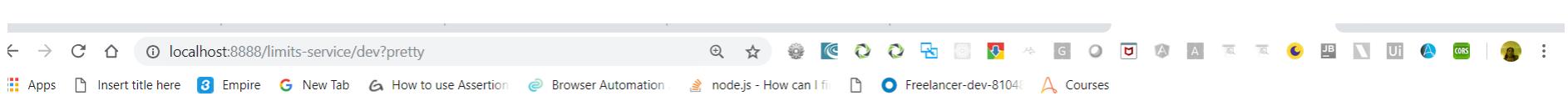
```
git commit --amend --reset-author
```

```
1 file changed, 2 insertions(+)
create mode 100644 limits-service.properties
Rangas-MacBook-Pro:git-localconfig-repo rangaraokaranam$
```



# Spring Cloud Config Server with git

```
1spring.application.name=spring-cloud-config-server
2server.port=8888
3spring.cloud.config.server.git.uri=file:///D:/Microservices/ConfigServerGit
```



```
{"name":"limits-service","profiles":["dev"],"label":null,"version":"b07af9f124423cd6326ac8e10c7d310c95f5d7ab","state":null,"propertySources": [{"name":"file:///D:/Microservices/ConfigServerGit/limits-service-dev.properties","source":{"limits-service.minimum":"1"}}, {"name":"file:///D:/Microservices/ConfigServerGit/limits-service.properties","source":{"limits-service.minimum":"88","limits-service.maximum":"888"}]}]
```



# Micro service With Multiple Port

```
10 // @JsonIgnoreProperties(value = { assetValue })  
11  
12 //dynamic filter  
13 @JsonFilter(value="AssetFilter")  
14 public class Asset {  
15     @Id  
16     @GeneratedValue(strategy=GenerationType.IDENTITY)  
17     @Column(name="Asset_Id")  
18     private int assetId;  
19     @Column(name="Asset_Name",nullable=false,length=50)  
20     private String(assetName);  
21     @Column(name="Asset_Value")  
22     private long assetValue;  
23     @Transient  
24     private int port;  
25  
26     public int getPort() {  
27         return port;  
28     }  
29     public void setPort(int port) {  
30         this.port = port;  
31     }  
32     public int getAssetId() {  
33         return assetId;  
34     }
```



# Micro service With Multiple Port

```
49
50
51 }
52 @RequestMapping(path="/getassetbyid/{id}",method=RequestMethod.GET,produces=MediaType.APPLICATION_JSON)
53 public @ResponseBody MappingJacksonValue getAssetInfo(@PathVariable int id)
54 {
55     Asset asset = assetService.findById(id);
56     asset.setPort(Integer.parseInt(environment.getProperty("local.server.port")));
57     SimpleBeanPropertyFilter propertyFilter=SimpleBeanPropertyFilter.filterOutAllExcept("assetId");
58     FilterProvider filters = new SimpleFilterProvider().addFilter("AssetFilter", propertyFilter);
59     MappingJacksonValue mapping=new MappingJacksonValue(asset);
60     mapping.setFilters(filters);
61     return mapping;
62
63 }
64 @CrossOrigin(origins = "*")
65 @RequestMapping(path="/getassetbyname/{name}",method=RequestMethod.GET,produces=MediaType.APPLICATION_JSON)
66 public @ResponseBody Asset getAssetByNameInfo(@PathVariable String name)
67 {
68     System.out.println(name);
69     return assetService.findByName(name);
```



# Micro service With Multiple Port

Run Configurations

## Create, manage, and run configurations

Run a Java application

type filter text

- Java Applet
- Java Application
  - ActuatorDemoApplication 6060
  - ActuatorDemoApplication 6061
  - JpademoApplication8000
  - SpringCloudConfigServerDemoApplication
- JUnit
- JUnit Plug-in Test
- Launch Group
- Maven Build
- Mwe2 Launch
- Node.js Application
- OSGi Framework
- Scala Application
- Scala Interpreter
- Task Context Test
- XSL

Filter matched 28 of 39 items

Name: ActuatorDemoApplication 6061

Main Arguments JRE Classpath Source Environment Common

Program arguments:

VM arguments:  
-Dserver.port=6061

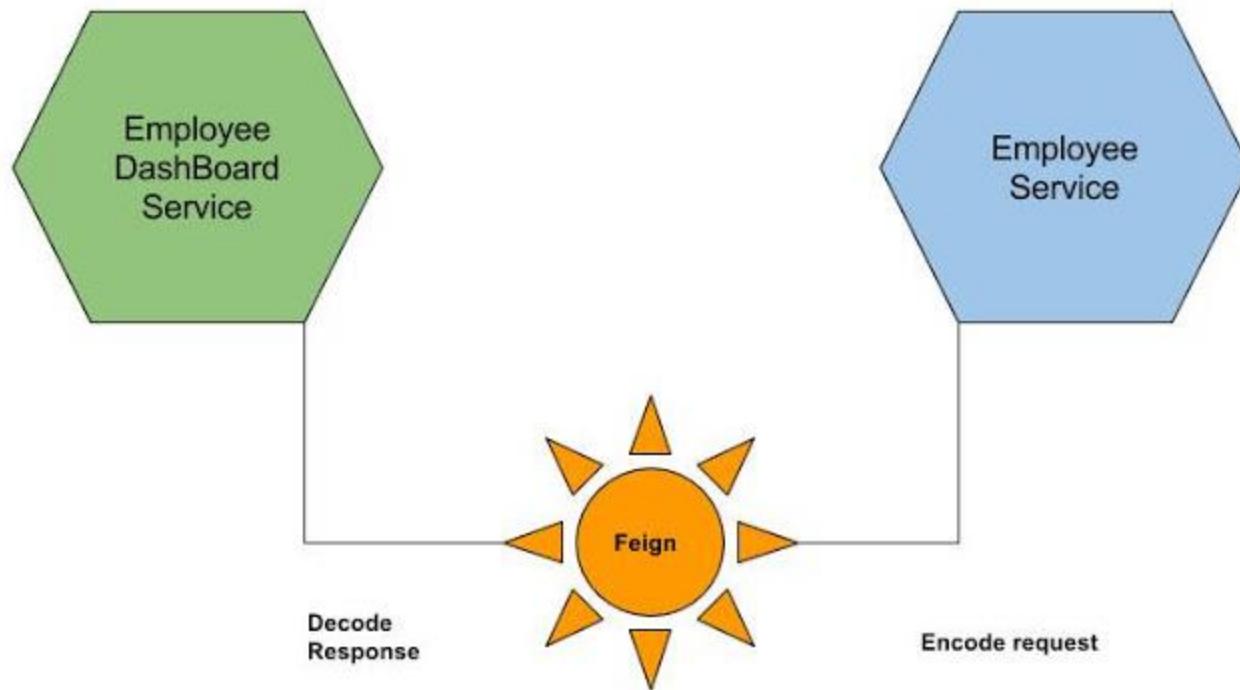
Working directory:  
 Default: \${workspace\_loc:jpademo}  
 Other:

Workspace... File System... Variables...

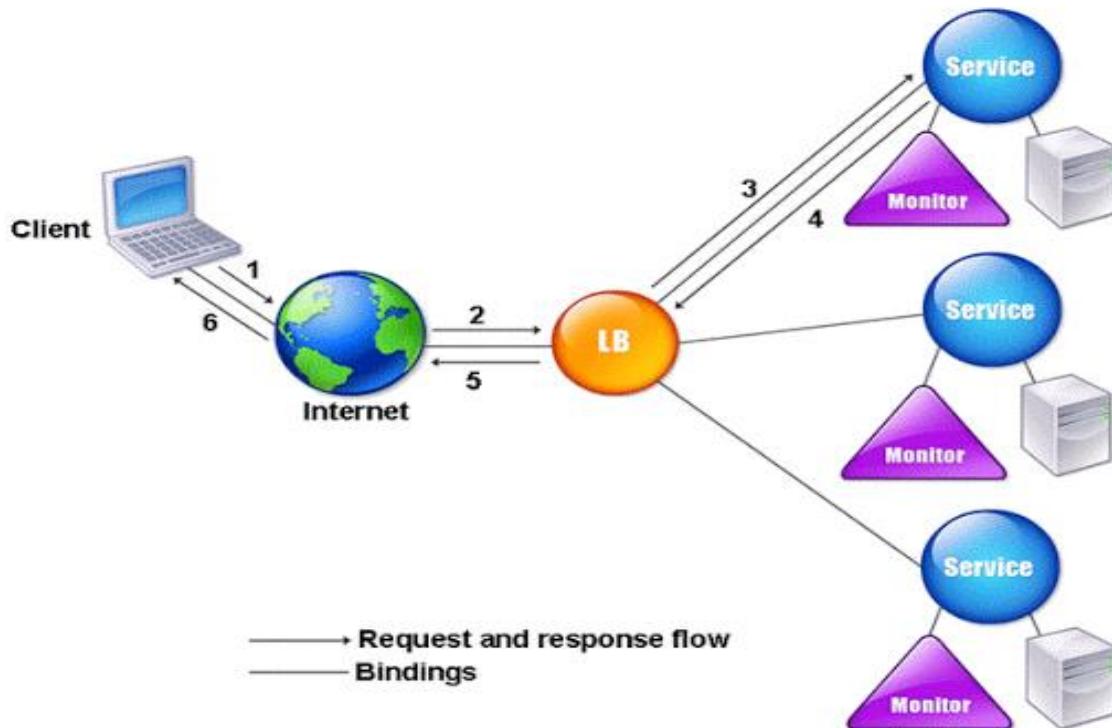
Revert Run Close Apply

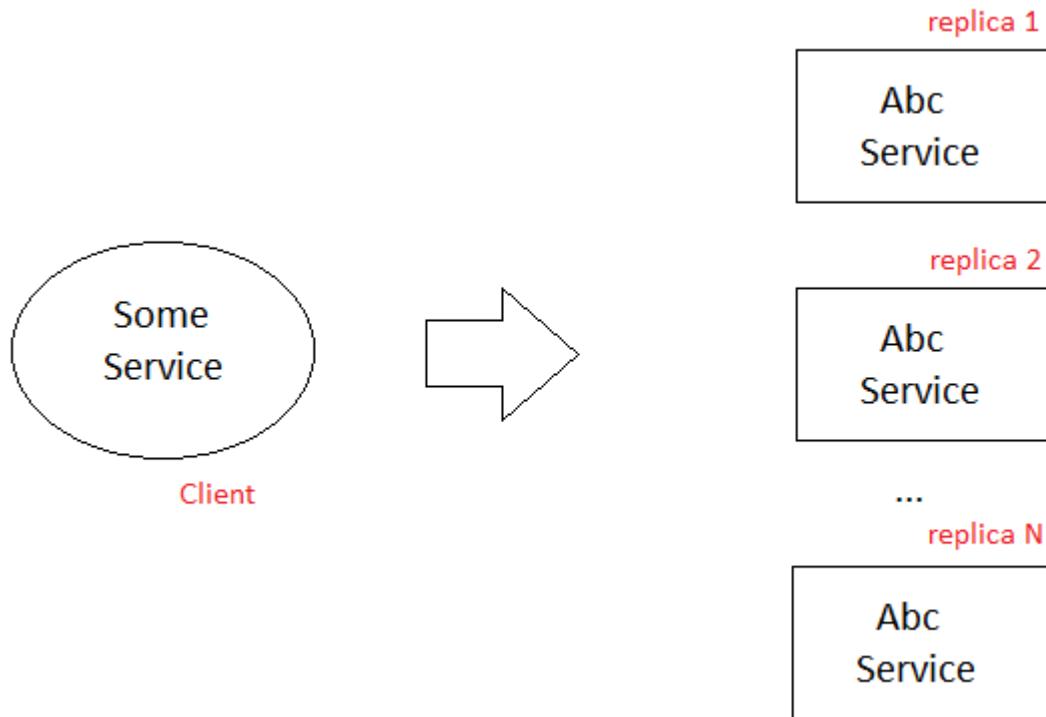


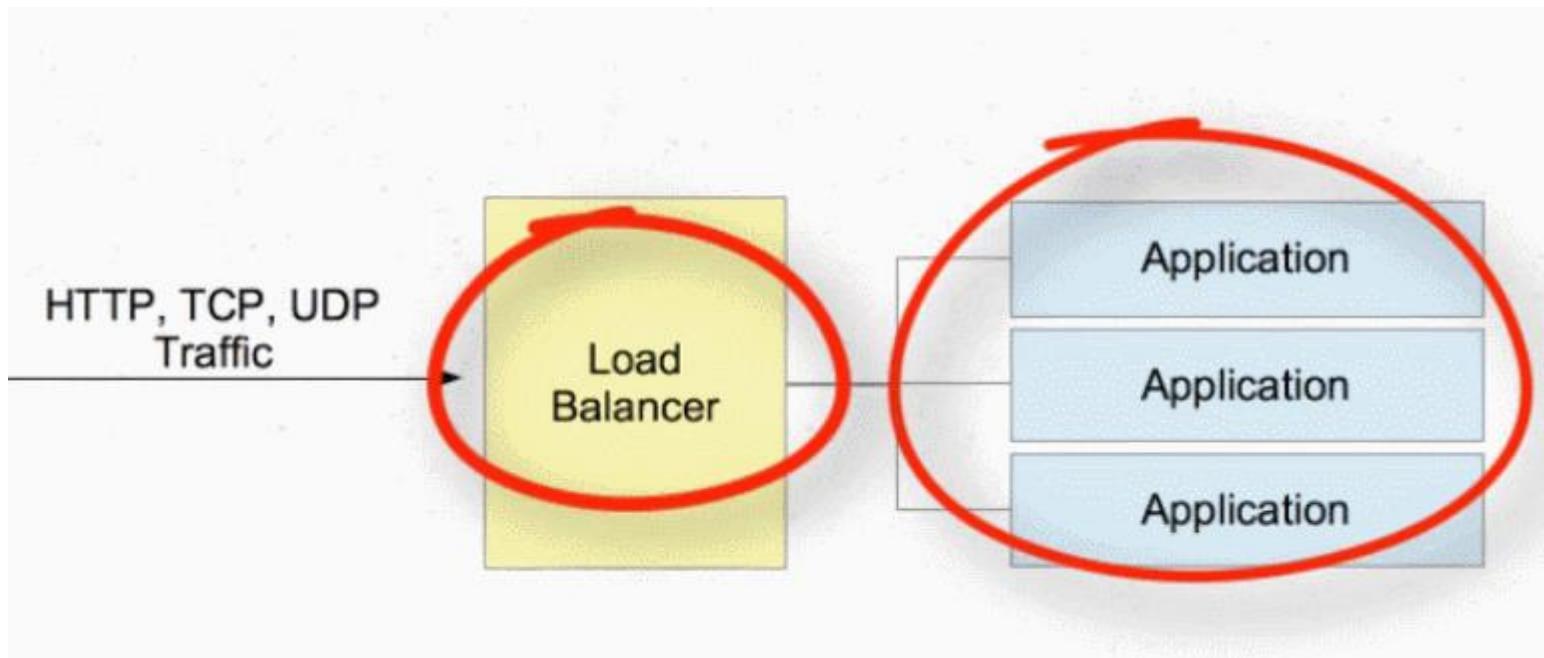
# Feign Client

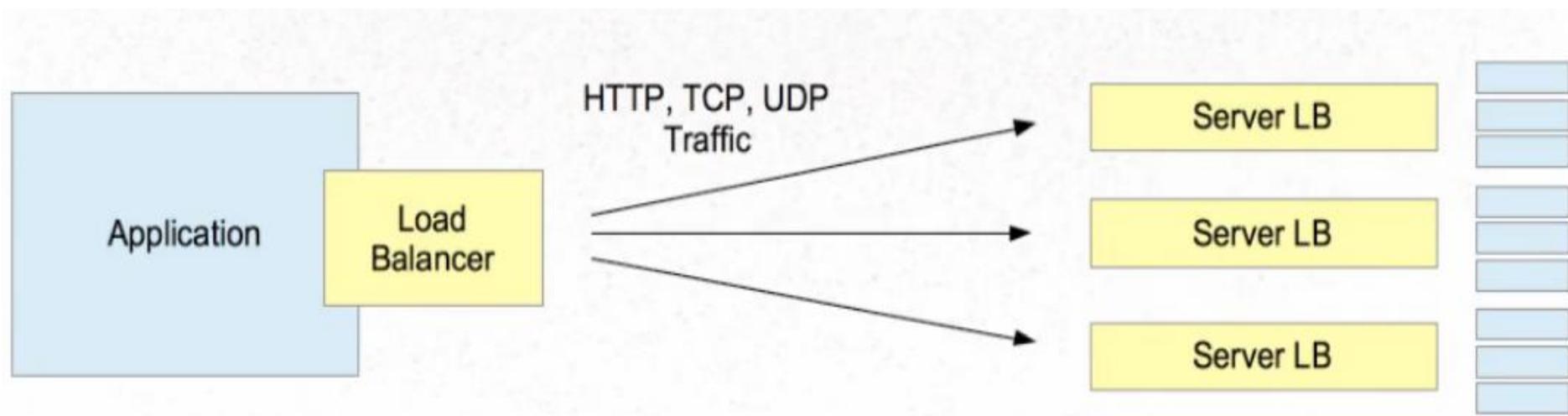


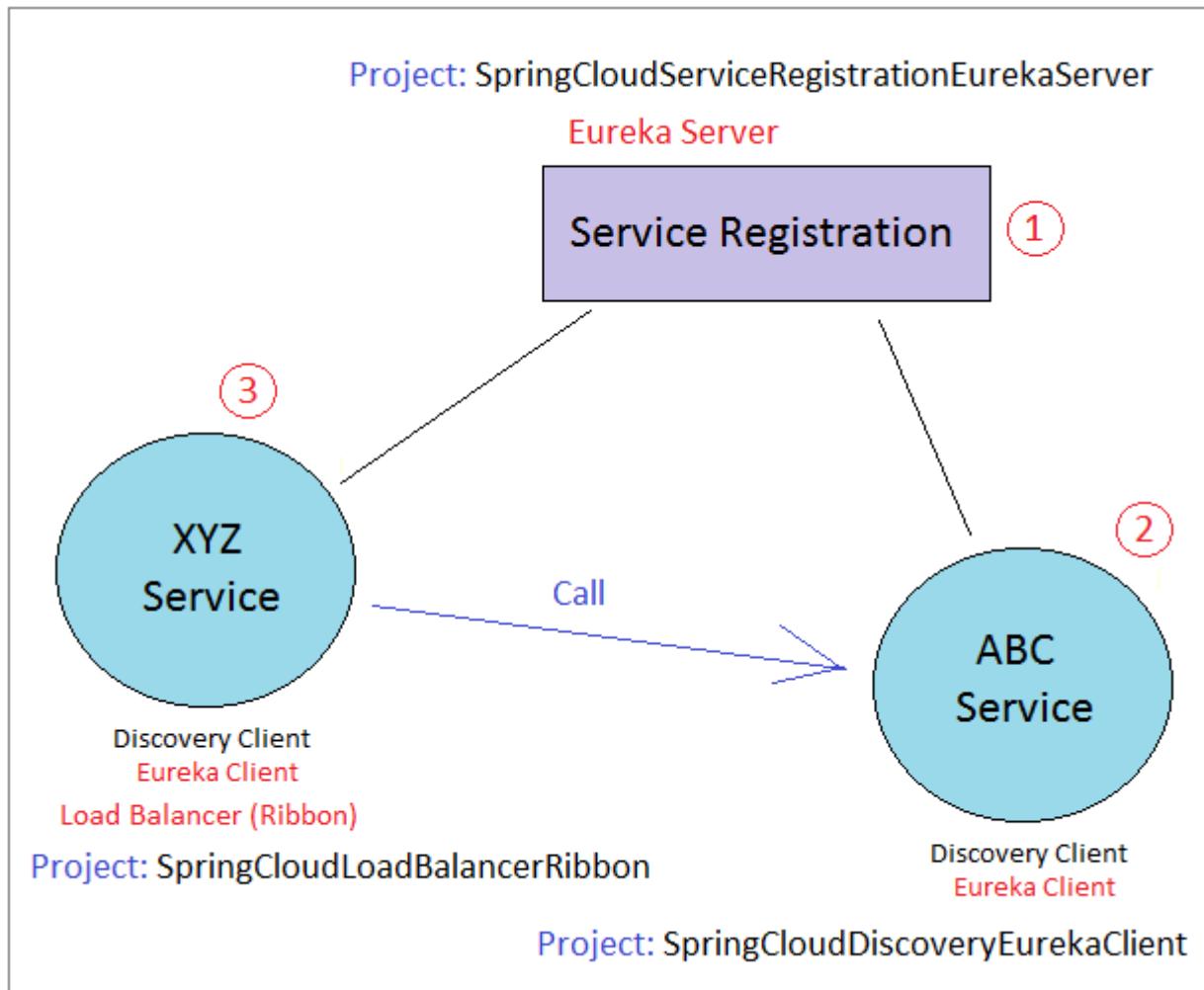
# RIBBON LOAD BALANCER

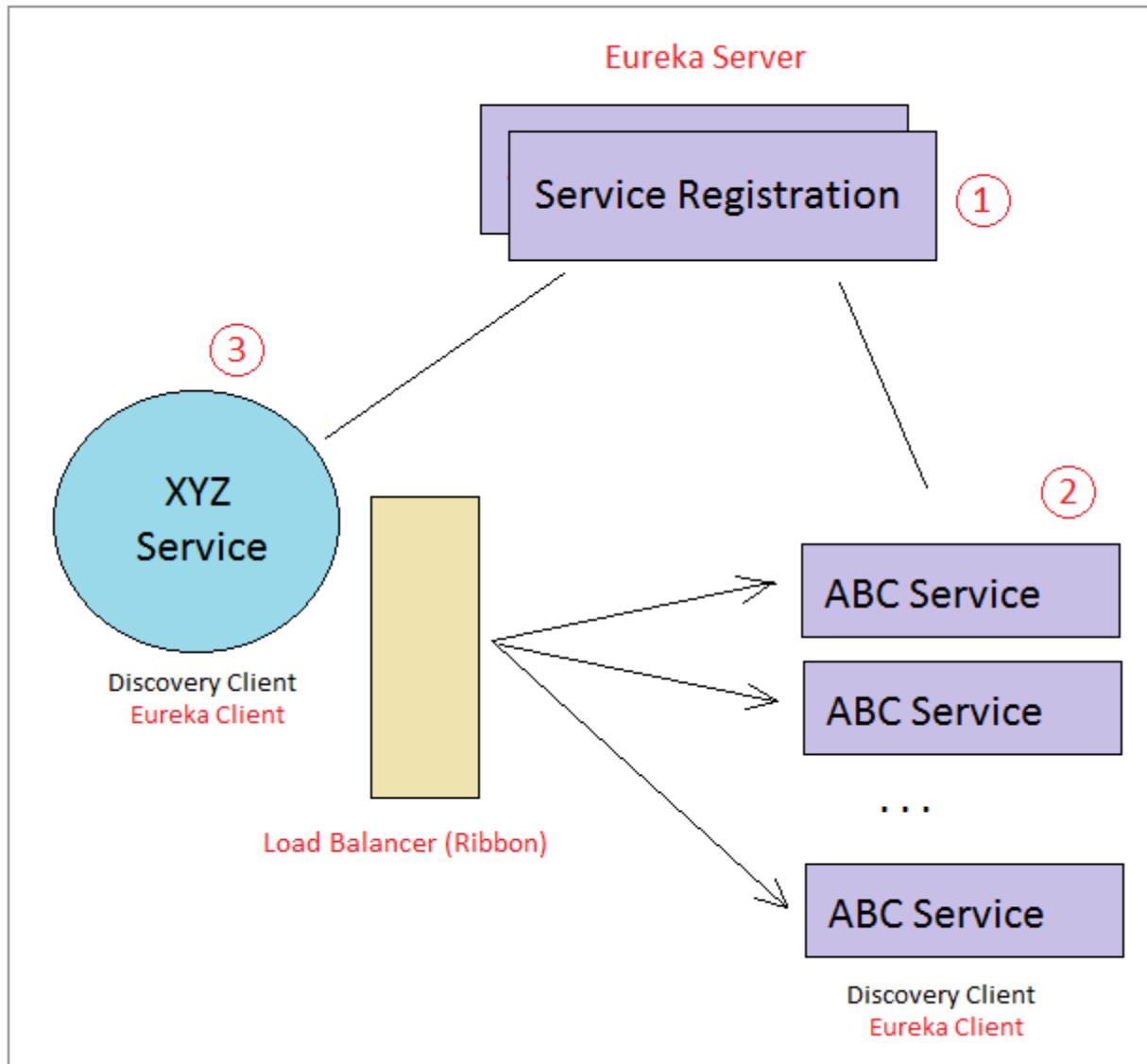














# RIBBON LOAD BALANCER

---

- Ribbon is a client-side load balancer, which gives you a lot of control over the behavior of HTTP and TCP clients.
- Ribbon's Client component offers a good set of configuration options such as connection timeouts, retries, retry algorithm (exponential, bounded back off) etc.
- Ribbon comes built in with a pluggable and customizable Load Balancing component.



# RIBBON LOAD BALANCER

---

- Some of the load balancing strategies offered are listed below:
- Simple Round Robin LB
- Weighted Response Time LB
- Zone Aware Round Robin LB
- Random LB



# RIBBON LOAD BALANCER

---

- Ribbon provides the following features:
- Load balancing
- Fault tolerance
- Multiple protocols (HTTP, TCP, UDP) support in an asynchronous and reactive model
- Caching and batching



# RIBBON LOAD BALANCER

Bean Type	Bean Name	Class Name
IClientConfig	ribbonClientConfig	DefaultClientConfigImpl
IRule	ribbonRule	ZoneAvoidanceRule
IPing	ribbonPing	DummyPing
ServerList<Server>	ribbonServerList	ConfigurationBasedServerList
ServerListFilter<Server>	ribbonServerListFilter	ZonePreferenceServerListFilter
ILoadBalancer	ribbonLoadBalancer	ZoneAwareLoadBalancer
ServerListUpdater	ribbonServerListUpdater	PollingServerListUpdater



# RIBBON LOAD BALANCER

The properties file (sample-client.properties)

```
# Max number of retries on the same server (excluding the first try)
sample-client.ribbon.MaxAutoRetries=1

# Max number of next servers to retry (excluding the first server)
sample-client.ribbon.MaxAutoRetriesNextServer=1

# Whether all operations can be retried for this client
sample-client.ribbon.OkToRetryOnAllOperations=true

# Interval to refresh the server list from the source
sample-client.ribbon.ServerListRefreshInterval=2000

#
```



# RIBBON LOAD BALANCER

Connect timeout used by Apache HttpClient  
sample-client.ribbon.ConnectTimeout=3000

# Read timeout used by Apache HttpClient  
sample-client.ribbon.ReadTimeout=3000

# Initial list of servers, can be changed via Archaius dynamic property at runtime  
sample-  
client.ribbon.listOfServers=www.microsoft.com:80, www.yahoo.com:80, www.google.co  
m:80



# API Gateway

---

- Authentication, authorization and security
- Rate Limits
- Fault Tolerance
- Service Aggregation

# Zipkin



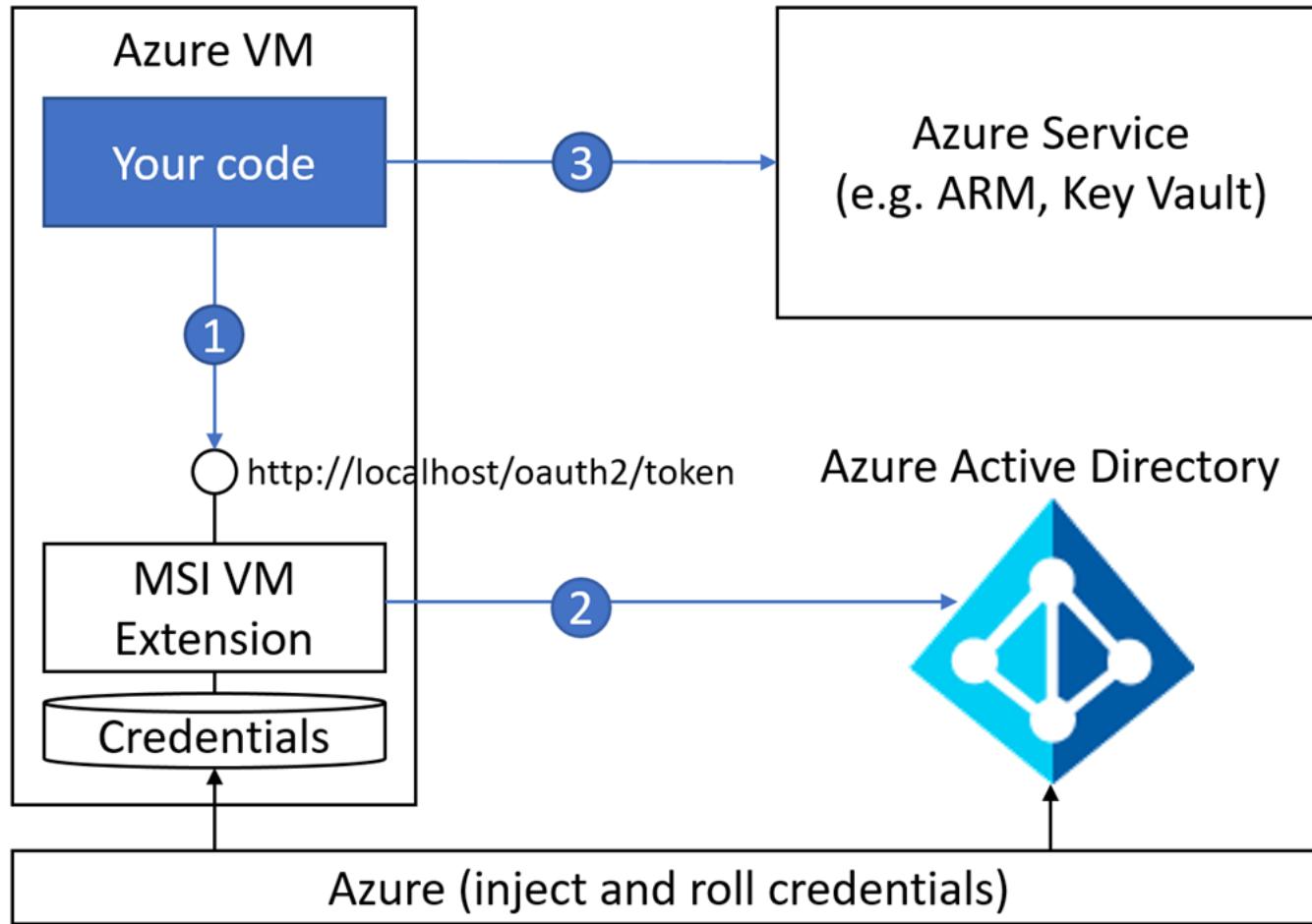


## AWS Vault

---

- Azure Key Vault is a cloud service that provides a secure store for secrets.
- We can securely store keys, passwords, certificates, and other secrets.
- Azure key vaults may be created and managed through the Azure portal.

# AWS Valult





# AWS Vault

RE: Microservice Using Dotnet - 2 | e! Amazon EKS | Building a Kuberne... | Amazon EKS | rpsvalut - Microsoft Azure | Azure Quickstart - Set and retriev... | +

Apps Insert title here Empire New Tab How to use Assert... Browser Automatio... node.js - How can I ... Freelancer-dev-810... Courses New Tab hi

Microsoft Azure  Home > Microsoft.KeyVault - Overview > rpsvalut

**rpsvalut** Key vault

Delete Move

Resource group (change) : Syed-BO  
Location : South India  
Subscription (change) : MSDN Platforms  
Subscription ID : 017d16e9-a9db-430b-b26c-1dfc4e605dc5  
DNS Name : https://rpsvalut.vault.azure.net/  
Sku (Pricing tier) : Standard  
Directory ID : 2f5c8d48-aacc-4e8d-a137-396450661599  
Directory Name : SyedLab

Overview Activity log Access control (IAM) Tags Diagnose and solve problems

Monitoring

Show data for last: 1 hour 6 hours 12 hours 1 day 7 days 30 days Click for additional metrics.

Total requests

100  
90  
80  
70  
60  
50  
40  
30  
20  
10  
0

6 am 12 pm 6 pm March

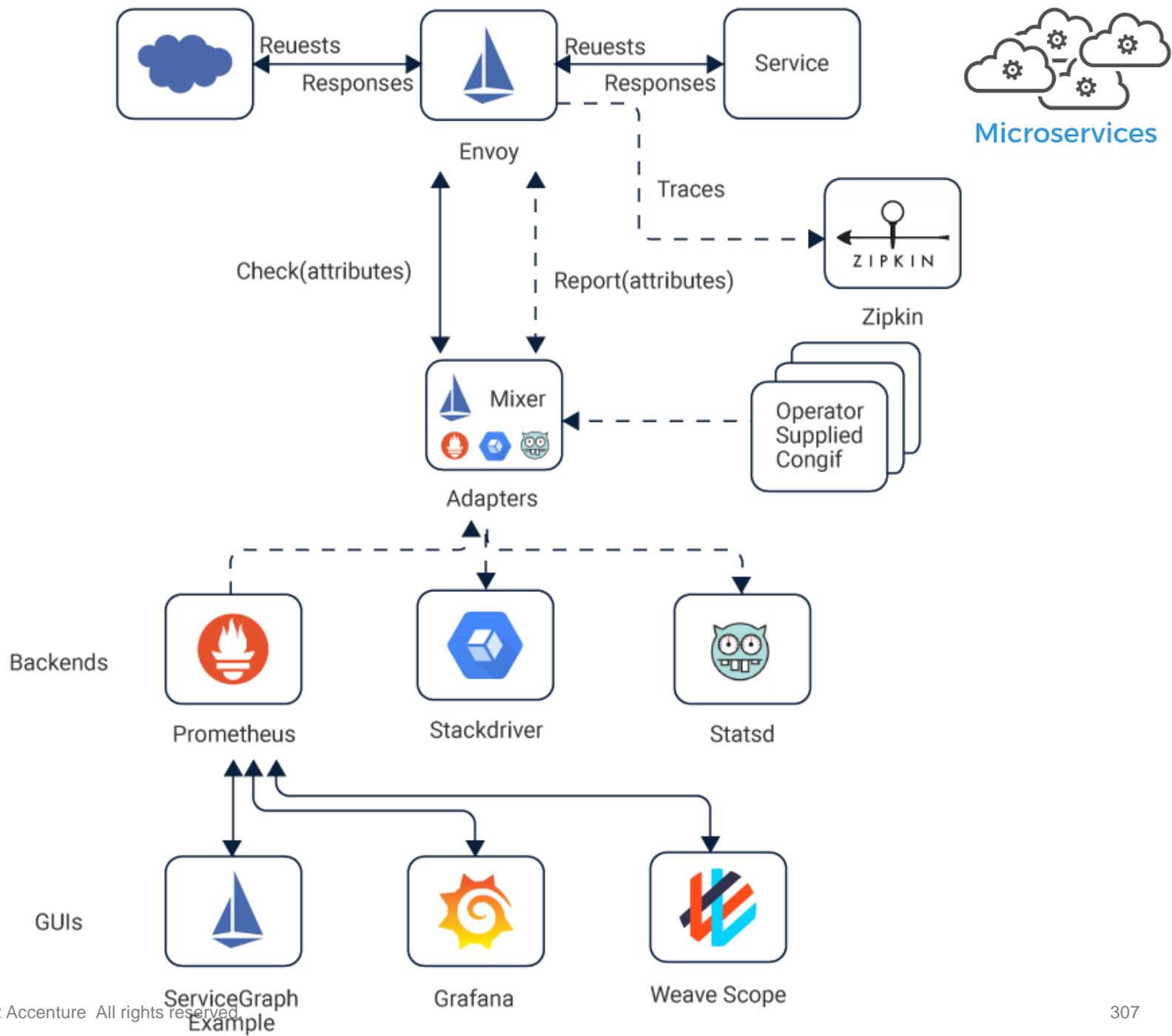
Type here to search

04:19 01/03/2019 305

# What is Istio Service Mesh?



- Istio service mesh provides several capabilities for traffic monitoring, access control, discovery, security, resiliency, and other useful things to a bundle of services.
- It delivers all that and strikingly does not require any changes to the code of any of those services.



# What is Istio Service Mesh?



- The term service mesh is used to describe the network of micro services that make up such applications and the interactions between them.
- As a service mesh grows in size and complexity, it can become harder to understand and manage. Its requirements can include discovery, load balancing, failure recovery, metrics, and monitoring.
- A service mesh also often has more complex operational requirements, like A/B testing, canary releases, rate limiting, access control, and end-to-end authentication.
- Istio provides behavioral insights and operational control over the service mesh as a whole, offering a complete solution to satisfy the diverse requirements of microservice applications.



# Why Istio

- Istio makes it easy to create a network of deployed services with load balancing, service-to-service authentication, monitoring, and more, without any changes in service code.
- You add Istio support to services by deploying a special sidecar proxy throughout your environment that intercepts all network communication between microservices, then configure and manage Istio using its control plane functionality, which includes:
  - Automatic load balancing for HTTP, gRPC, WebSocket, and TCP traffic.
  - Fine-grained control of traffic behavior with rich routing rules, retries, failovers, and fault injection.
  - A pluggable policy layer and configuration API supporting access controls, rate limits and quotas.
  - Automatic metrics, logs, and traces for all traffic within a cluster, including cluster ingress and egress.
  - Secure service-to-service communication in a cluster with strong identity-based authentication and authorization.



# Architecture

- An Istio service mesh is logically split into a **data plane** and a **control plane**.
- The **data plane** is composed of a set of intelligent proxies (Envoy) deployed as sidecars. These proxies mediate and control all network communication between microservices along with Mixer, a general-purpose policy and telemetry hub.
- The **control plane** manages and configures the proxies to route traffic. Additionally, the control plane configures Mixers to enforce policies and collect telemetry.

# What is Istio Service Mesh?



- Envoy
- Envoy is an open-source extension and service proxy provider, built for cloud-extensive meshes. The Istio mesh creates an extendible proxy system through Envoy.
- Mixer
- The mixer is a part of the service mesh that helps in enforcing safety protocols, allowing access controls and implementing usage policies and works independently from the mesh.
- Pilot
- Pilot provides all services for the Istio Envoy sidecars and allows for a more coherent traffic management system with high level routing.

# What is Istio Service Mesh?



- To make this possible, Istio deploys an Istio proxy (called an Istio sidecar) next to each service.
- All of the traffic meant for assistance is directed to the proxy, which uses policies to decide how, when, or if that traffic should be deployed to the service.
- It also enables sophisticated techniques such as canary deployments, fault injections, and circuit breakers.



- Dynamic service discovery
- Load balancing
- TLS termination
- HTTP/2 and gRPC proxies
- Circuit breakers
- Health checks
- Staged rollouts with %-based traffic split
- Fault injection
- Rich metrics



- Mixer is a platform-independent component.
- Mixer enforces access control and usage policies across the service mesh, and collects telemetry data from the Envoy proxy and other services.
- The proxy extracts request level attributes, and sends them to Mixer for evaluation.
- You can find more information on this attribute extraction and policy evaluation in our [Mixer Configuration documentation](#).
- Mixer includes a flexible plugin model. This model enables Istio to interface with a variety of host environments and infrastructure backends. Thus, Istio abstracts the Envoy proxy and Istio-managed services from these details.



# Pilot

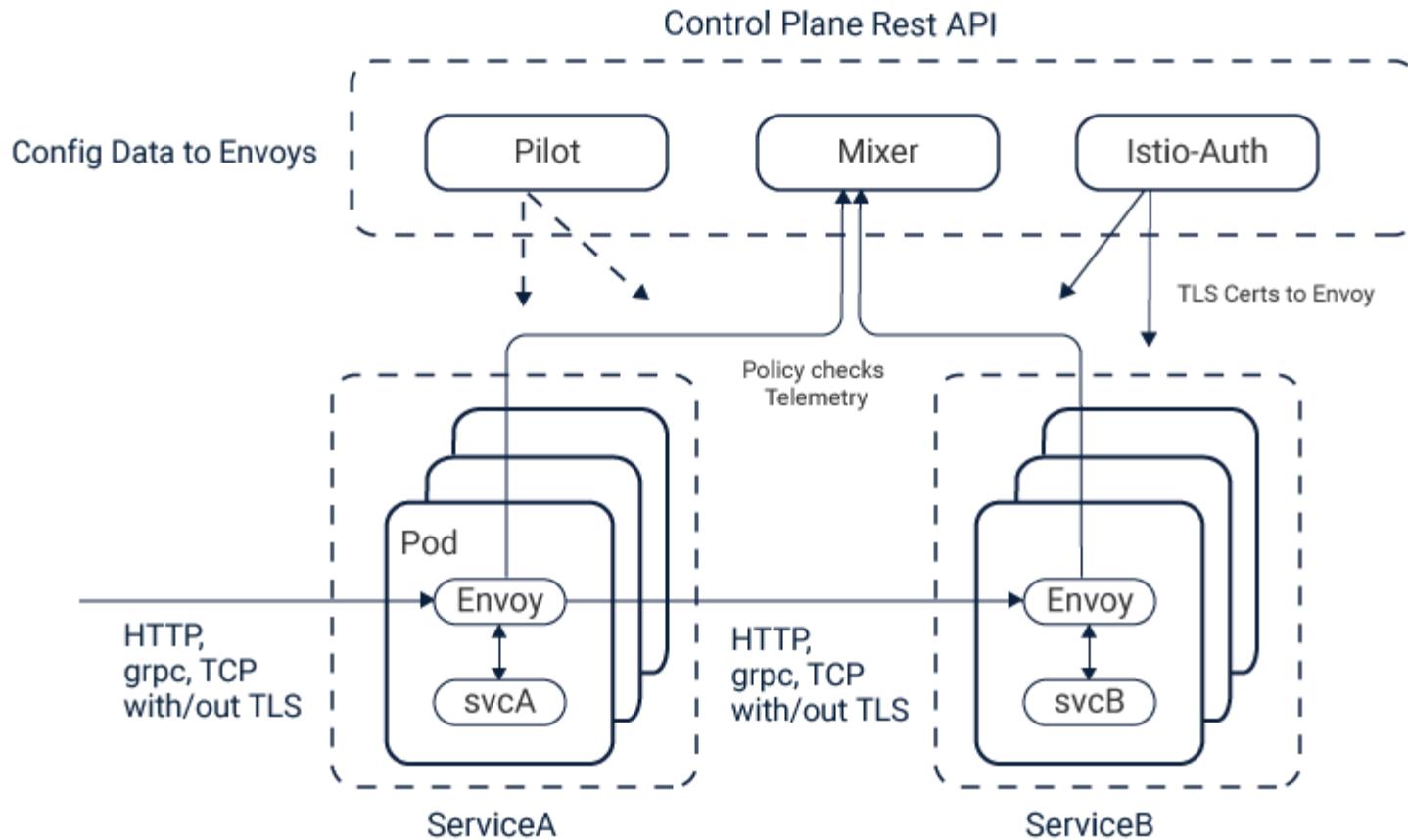
- Pilot provides service discovery for the Envoy sidecars, traffic management capabilities for intelligent routing (e.g., A/B tests, canary deployments, etc.), and resiliency (timeouts, retries, circuit breakers, etc.).
- Pilot converts high level routing rules that control traffic behavior into Envoy-specific configurations, and propagates them to the sidecars at runtime.
- Pilot abstracts platform-specific service discovery mechanisms and synthesizes them into a standard format that any sidecar conforming with the Envoy data plane APIs can consume.
- This loose coupling allows Istio to run on multiple environments such as Kubernetes, Consul, or Nomad, while maintaining the same operator interface for traffic management.

- Citadel provides strong service-to-service and end-user authentication with built-in identity and credential management.
- You can use Citadel to upgrade unencrypted traffic in the service mesh. Using Citadel, operators can enforce policies based on service identity rather than on network controls.
- Starting from release 0.5, you can use Istio's authorization feature to control who can access your services.

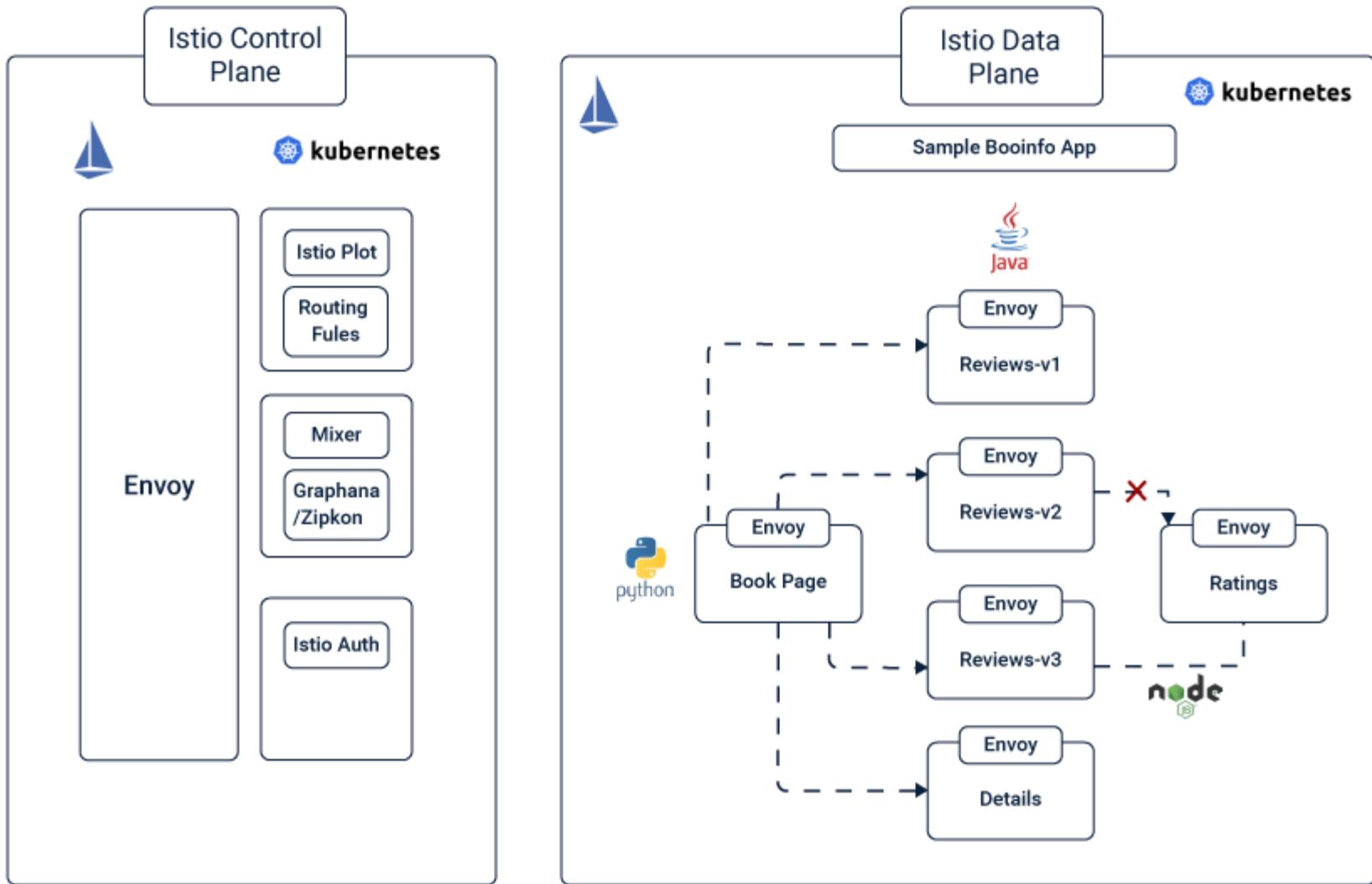


- Galley validates user authored Istio API configuration on behalf of the other Istio control plane components.
- Over time, Galley will take over responsibility as the top-level configuration ingestion, processing and distribution component of Istio.
- It will be responsible for insulating the rest of the Istio components from the details of obtaining user configuration from the underlying platform (e.g. Kubernetes).

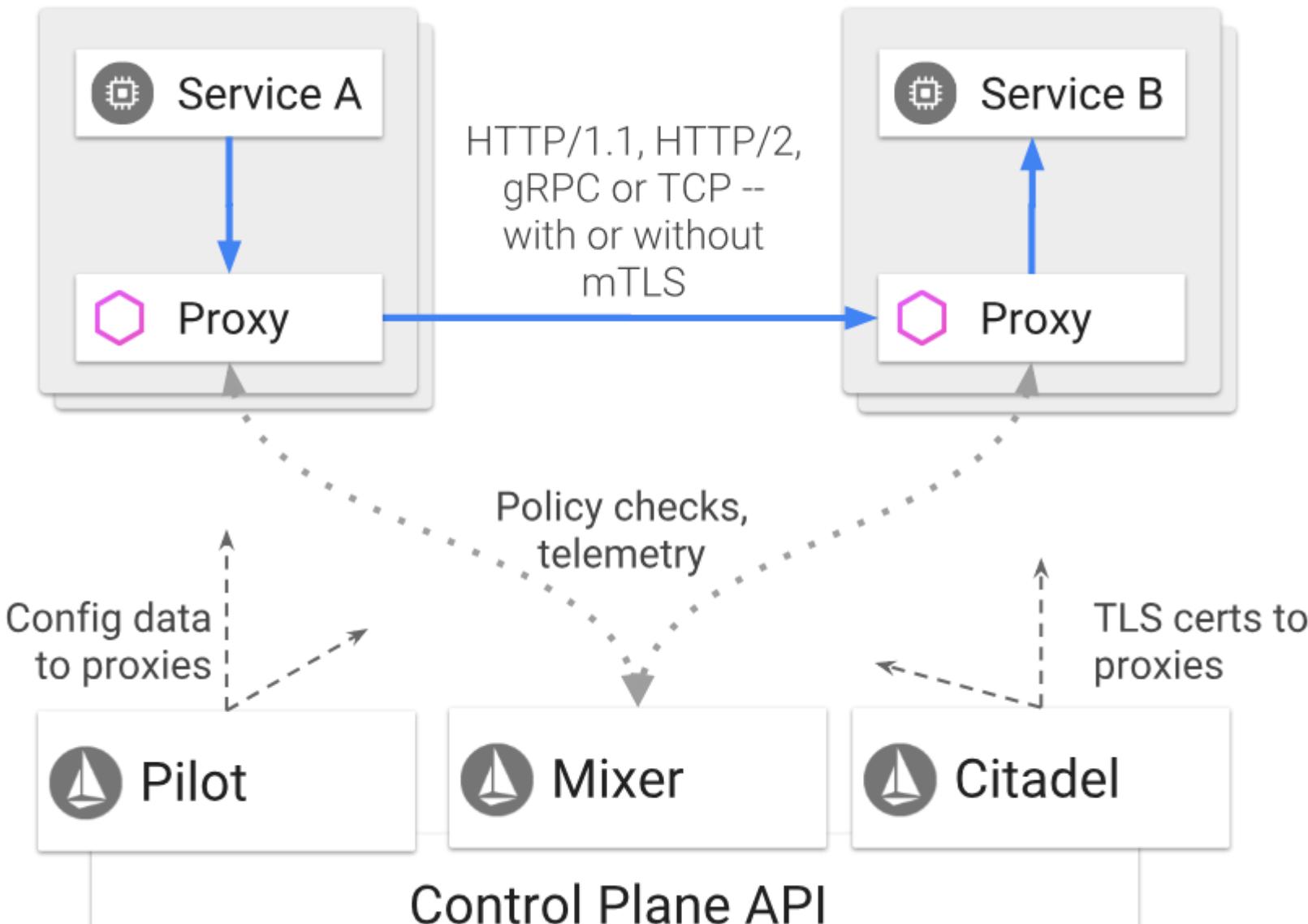
# What is Istio Service Mesh?



# What is Istio Service Mesh?



# What is Istio Service Mesh?





# Istio Download

---

- <https://github.com/istio/istio/releases>
- minikube start --memory=8192 --cpus=4 --kubernetes-version=v1.10.0 \
- --extra-config=controller-manager.cluster-signing-cert-file="/var/lib/localkube/certs/ca.crt" \
- --extra-config=controller-manager.cluster-signing-key-file="/var/lib/localkube/certs/ca.key" \
- --vm-driver=virtualbox



## Istio Download

---

- <https://github.com/istio/istio/releases>
- **install Istio with Helm and Tiller on Minikube**
- `kubectl create -f install/kubernetes/helm/helm-service-account.yaml`
- `helm init --service-account tiller`



## Istio Download

---

- Install Istio with automatic sidecar injection
- \$ helm install install/kubernetes/helm/istio --name istio --namespace istio-system
- kubectl get svc -n istio-system
- kubectl get pods -n istio-system
- kubectl apply -f samples/bookinfo/kube/bookinfo.yaml
- kubectl get services



# Istio Download

- Confirm the Bookinfo app
- \$ export INGRESS\_HOST=\$(minikube ip)
- \$ export INGRESS\_PORT=\$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="http")].nodePort}' )
- \$ export GATEWAY\_URL=\$INGRESS\_HOST:\$INGRESS\_PORT
- # Check
- \$ curl -o /dev/null -s -w "%{http\_code}\n" http://\$GATEWAY\_URL/productpage
- 200
- # Check on a browser
- \$ xdg-open http://\$GATEWAY\_URL/productpage

# Azure Kubernetes Service (AKS)

Description	Pros	Cons	Pricing
<p>Kubernetes is a container orchestration platform. Azure Kubernetes Service (AKS) is a managed service offering. With AKS Microsoft manages the master nodes of an AKS cluster reducing complexity and operational overhead.</p> <p>Azure Kubernetes Service (AKS) manages a hosted Kubernetes environment, making it easy to deploy and manage containerized applications without container orchestration experience.</p> <p><b>Official Link:</b> <a href="https://docs.microsoft.com/en-us/azure/aks/intro-kubernetes">https://docs.microsoft.com/en-us/azure/aks/intro-kubernetes</a></p>	<p><b>General</b></p> <ul style="list-style-type: none"><li>• Kubernetes has wide adoption including managed offerings on all major cloud platforms (AWS, GCP, Azure) helping avoid vendor lock in.</li><li>• Wide community and industry support.</li><li>• Scalability and modularity</li><li>• Kubernetes has a very mature and proven underlying architecture. Its design is built on over 10 years of operational experience of the Google engineers.</li><li>• More control over the orchestration platform for custom or specific needs.</li><li>• Monitoring available from Microsoft Azure monitoring at a Kubernetes cluster and container level.</li><li>• The power of a full container orchestration solution.</li><li>• Flexibility with networking.</li><li>• Rich ecosystem of addons.</li></ul> <p><b>Container Specific</b></p> <ul style="list-style-type: none"><li>• Supports Docker hub, Azure Container registry, private Container registry.</li><li>• Supports use of Docker compose or Helm charts.</li></ul>	<p><b>General</b></p> <ul style="list-style-type: none"><li>• Can't run code directly on Kubernetes; only supports containers.</li><li>• As a free service, AKS does not offer a financially-backed service level agreement. Microsoft will strive to attain at least 99.5% availability for the Kubernetes API server. The availability of the agent nodes in your cluster is covered by the Azure Virtual Machines SLA.</li><li>• Steep learning curve. Need an understanding of how an orchestration platform and Docker containers work.</li><li>• If something breaks can require complex troubleshooting.</li></ul> <p><b>Container Specific</b></p> <ul style="list-style-type: none"><li>• Only supports Linux containers in AKS currently.</li></ul>	<p>As a managed Kubernetes service, AKS is free - you only pay for the agent nodes within your clusters, not for the masters.</p>

# Azure Service Fabric (ASF)

Description	Pros	Cons	Pricing
<p>Azure Service Fabric is a distributed platform for running applications based on microservices and containers. Service Fabric also serves as the orchestration platform for microservices and containers.</p> <p><b>Official Link:</b> <a href="https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-overview">https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-overview</a></p> <p><b>NOTE:</b> On Azure there are two Service Fabric offerings. #1 is a Service Fabric Cluster and #2 is Service Fabric Mesh. The cluster offering is a Service Fabric instance you manage. Mesh is a managed Service Fabric but is currently in preview.</p>	<p><b>General</b></p> <ul style="list-style-type: none"><li>Service Fabric can run in Azure but also can run on-premises or other clouds helping to avoid vendor lock-in.</li><li>Service Fabric is now open source.</li><li>Ability to run code or a container.</li><li>Ability to have an application that combines containers and Service Fabric microservices. For example, NGINX for web front end running as a container and the rest of the application components running as services all on Service Fabric.</li><li>Automatic load balancing.</li><li>Automatic high availability.</li><li>Automated load distribution and scheduling.</li><li>Proven stability and enterprise use as it is used to power majority of Microsoft Azure cloud services.</li></ul> <p><b>Container Specific</b></p> <ul style="list-style-type: none"><li>Service Fabric supports both Linux and Windows containers.</li></ul>	<p><b>General</b></p> <ul style="list-style-type: none"><li>The Service Fabric service itself is a free service and therefore does not carry an SLA. The Service Fabric availability is based on the SLA's of the underlying Azure services including virtual machines and storage.</li><li>Semi-steep learning curve. Need an understanding of how an orchestration/micro services platform and Docker containers work.</li><li>Service Fabric is now open source, but the community support and expertise are not as wide spread as a platform such as Kubernetes.</li><li>If something breaks can require complex troubleshooting.</li></ul> <p><b>Container Specific</b></p> <ul style="list-style-type: none"><li>Service Fabrics focus is on programming frameworks for .NET and Java libraries. Containers are also supported but they are second-class citizens on the platform as they run as guest workloads and therefore do not benefit from the full feature sets as the programming frameworks.</li></ul>	Charged for the compute instances, storage, networking resources, and IP addresses used in a Service Fabric cluster on Azure. No charge for the Service Fabric itself.



# Cloud Native Functionalities

- Architecturally, it is standard practice to leverage as much as cloud-native functionalities(PaaS/CaaS as compared to IaaS) as possible, or leveraging a standard that is portable across the cloud vendors.
- It leads to increased developer agility, stronger operational control, awesome developer tooling, offloading cross-cutting concerns and functionality to cloud like auditing, role-based access control, authentication, authorization, DevOps integration, log analytics, telemetry etc.
- Cloud native computing foundation has put up an awesome set of resources and tooling information that you can look to get to the right practices and architecture for a solid future ready cloud architecture.



# Kubernetes Offering for Native Cloud

- **Azure Kubernetes Service** is a CaaS offering which sits somewhere between a PaaS and an IaaS.
- That said, it does not offer complete native cloud integration and offers a managed version of the open source Kubernetes orchestration engine.
- Although, it should be noted that Kubernetes is one of the open source standards of cloud-native foundation and the recommended architectural tooling for an inter-operable cloud architecture.
- That said, AKS miss some of the native functionality that could be offered by Azure out of the box.
- It means some of the functionalities like SSL encryption/termination, load balancing(partial), SCM, kudus, auto-scaling etc. has to managed on our own.
- It does provide native functionalities like log analytics, managed service identity, encryption etc.



# Service Fabric

- **Azure Service Fabric** is yet another PaaS offering, but unlike App Service, it is a different beast.
- As a matter of fact, App Service internally is built on the Service Fabric.
- It is an amazing product for large scale complex production application where 24x7 availability and resiliency is of utmost importance.
- It has a ton of native functionality, including native development framework, development tooling, powerful native orchestration, cluster management, auto-scaling, self-healing, etc.
- It's a proprietary Microsoft stack and if you are not worried about using Microsoft tooling like Visual Studio, .Net framework and the likes; Service Fabric definitely wins on the cloud-native features and is also inter-operable because it is also recently open sourced by Azure.



# 12 Factor Applications

---

- **The Twelve Factors**
- **I. Codebase**
  - One codebase tracked in revision control, many deploys
- **II. Dependencies**
  - Explicitly declare and isolate dependencies
- **III. Config**
  - Store config in the environment
- **IV. Backing services**
  - Treat backing services as attached resources
- **V. Build, release, run**
  - Strictly separate build and run stages



# 12 Factor Applications

---

- VI. Processes
  - Execute the app as one or more stateless processes
- VII. Port binding
  - Export services via port binding
- VIII. Concurrency
  - Scale out via the process model
- IX. Disposability
  - Maximize robustness with fast startup and graceful shutdown
- X. Dev/prod parity
  - Keep development, staging, and production as similar as possible
- XI. Logs
  - Treat logs as event streams
- XII. Admin processes
  - Run admin/management tasks as one-off processes



# Service Fabric

- **Service Fabric** is also built for the micro-service and event-driven architectures and pretty much support everything that AKS offers from the architectural standpoint.
- Where AKS is strictly a service orchestrator and handles deployments, Service fabric also offers a development framework that allows building modern stateful/stateless reactive and 12-factor applications.
- It is, however, a very opinionated framework with a large inclination towards Microsoft proprietary tooling.



# Service Fabric

---

- **Service Fabric** as we know is a Microsoft proprietary technology.
- Although it was open sourced some time back, it still lacks the large community support and open source tooling for the DevOps.
- It has the great support of Azure Secure DevOps toolkit, but that is again tied to proprietary MS stack.
- Also, it has a large learning curve to understand the operational features and unlike Kubernetes, does not have a lot of community tooling and resources(books, courses) etc.



# Service Fabric

---

- **Service Fabric** also offers tight control over the underlying infrastructure.
- However, topology control is limited considering the proprietary technology and limited open source tooling and community work.
- While definitely, it is possible to control everything on service fabric, it is definitely not something everyone can do.



# Service Fabric

---

- Azure Service Fabric is a distributed systems platform that makes it easy to package, deploy, and manage scalable and reliable microservices and containers.
- Service Fabric also addresses the significant challenges in developing and managing cloud native applications.
- Developers and administrators can avoid complex infrastructure problems and focus on implementing mission-critical, demanding workloads that are scalable, reliable, and manageable.
- Service Fabric represents the next-generation platform for building and managing these enterprise-class, tier-1, cloud-scale applications running in containers.

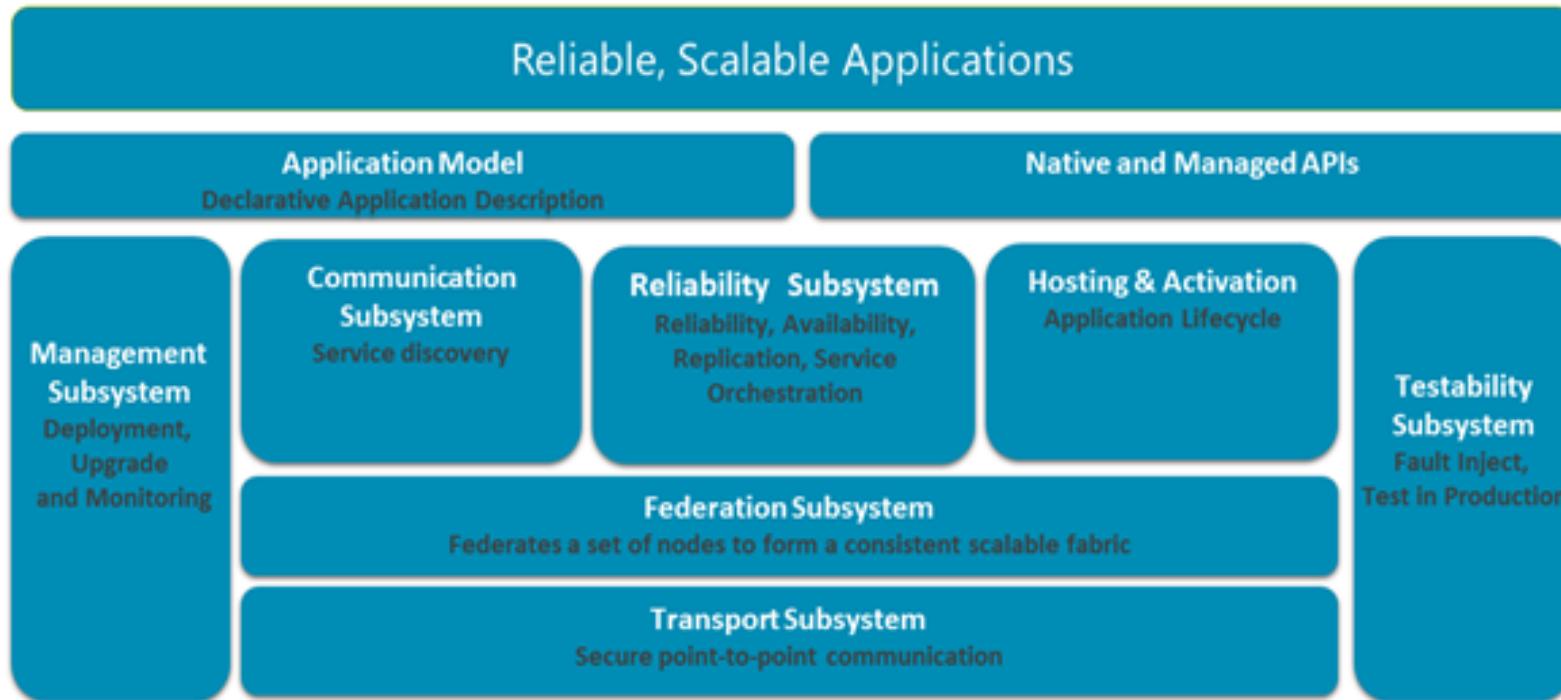


# Service Fabric

- Deploy to Azure or to on-premises datacenters that run Windows or Linux with zero code changes. Write once, and then deploy anywhere to any Service Fabric cluster.
- Develop scalable applications that are composed of microservices by using the Service Fabric programming models, containers, or any code.
- Develop highly reliable stateless and stateful microservices. Simplify the design of your application by using stateful microservices.
- Use the novel Reliable Actors programming model to create cloud objects with self contained code and state.
- Deploy and orchestrate containers that include Windows containers and Linux containers. Service Fabric is a data aware, stateful, container orchestrator.
- Deploy applications in seconds, at high density with hundreds or thousands of applications or containers per machine.



# Service Fabric





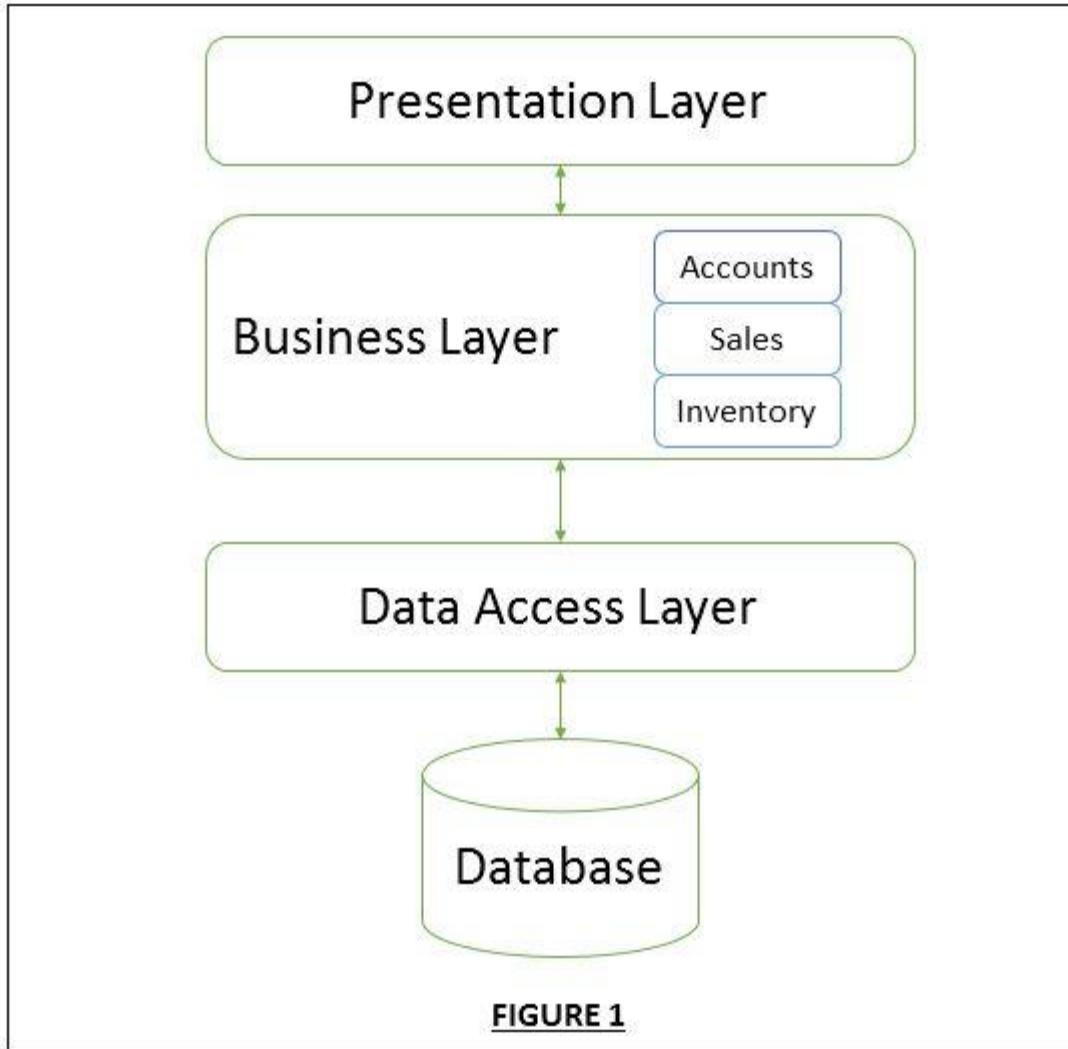
# Service Fabric

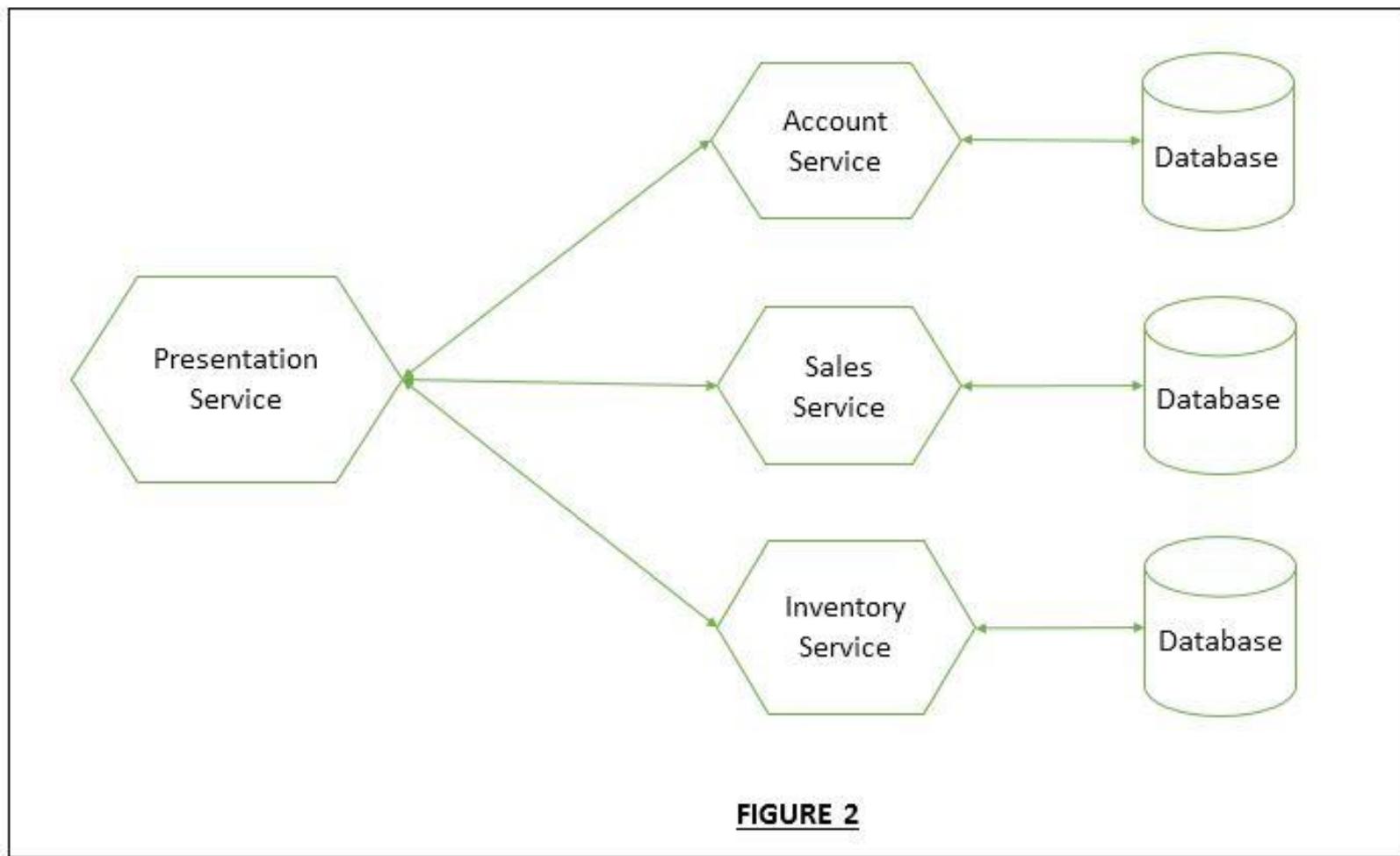
- Deploy different versions of the same application side by side, and upgrade each application independently.
- Manage the lifecycle of your applications without any downtime, including breaking and nonbreaking upgrades.
- Scale out or scale in the number of nodes in a cluster. As you scale nodes, your applications automatically scale.
- Monitor and diagnose the health of your applications and set policies for performing automatic repairs.
- Watch the resource balancer orchestrate the redistribution of applications across the cluster. Service Fabric recovers from failures and optimizes the distribution of load based on available resources.



# Service Fabric Offerings

- **Stateless service** : Is a service that does not need to maintain any state, so this could be a per request type of thing, or something that will create a new processing pipeline per message or something like that
- **Stateful service** : Is a service where we need to store state. This is done via ReliableDictionary in the ServiceFabric
- **Actor service** : Is intended for tiny discrete bits of functionality that send messages to each other. think 1000nds of actors all doing very small bits of work sending messages to each other
- **Guest executable** : Allows you to package up virtually anything (exe, node app whatever) and have it run on the fabric
- **Container** : Container that will run on the fabric
- **ASP Core** : Either web Api type app or stateful http app







# Service Fabric

- Microsoft Azure Service Fabric provides the necessary mechanism to build Microservices-based applications.

Under the hood it facilitates the below capabilities.

- Provides necessary APIs to build Microservices applications
- Manages state for the services in the Microservices applications
- Provides hosting mechanism for Microservices based applications
- Handles scaling out for independent services in the Microservices application
- Guarantees High Availability by spinning out a new service instance for the Microservices application when one of the instances goes down
- Provides mechanism to monitor node and service instance health
- Helps in managing life cycle of Microservices application using a feature rich and intuitive dashboard.
- Provides all that is needed to build, manage and deploy Microservices based application.

New-SelfSignedCertificate –DnsName <**Computer name**> -  
CertStoreLocation “cert:\LocalMachine\My”



```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> New-SelfSignedCertificate -DnsName DESKTOP-55AG101 -CertStoreLocation "cert:\LocalMachine\My"

PSParentPath: Microsoft.PowerShell.Security\Certificate::LocalMachine\My

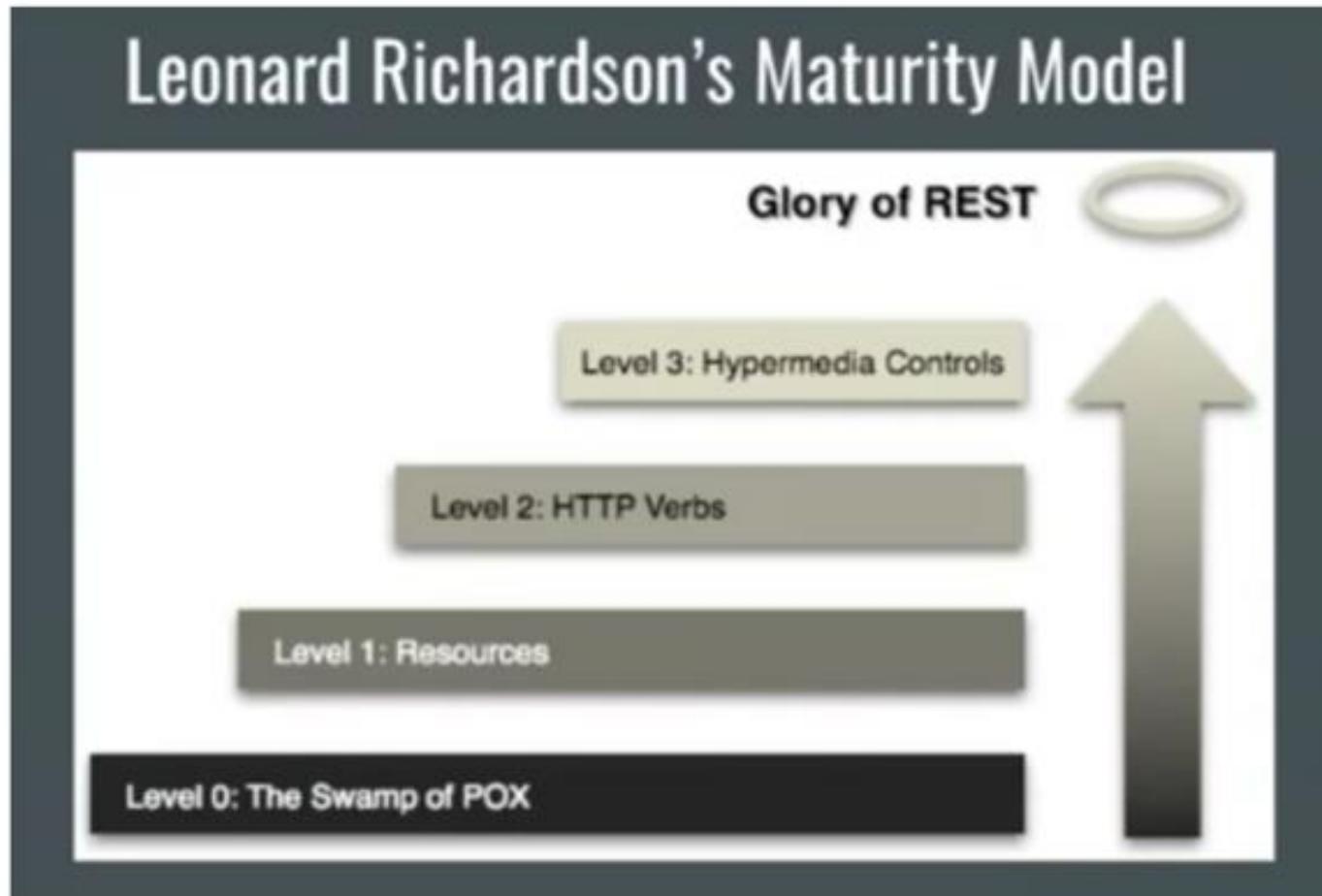
Thumbprint          Subject
-----          -----
5D0E3674408193802110352654BA9B09FACA8874 CN=DESKTOP-55AG101

PS C:\WINDOWS\system32>
```



- <https://portal.azure.com/#create/azure-oss.jenkins>

## Best Practice #1





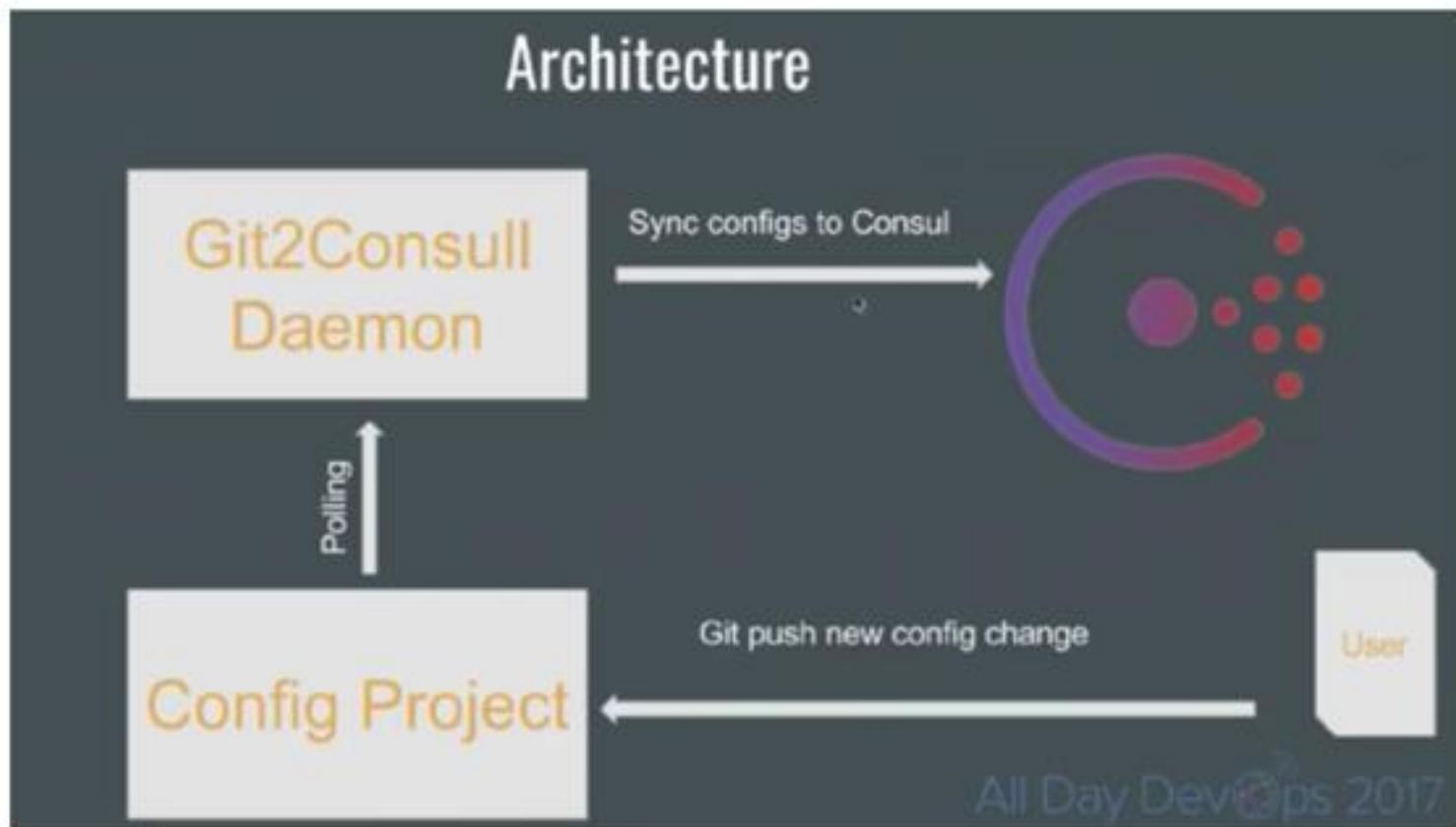
## Best Practice #2

---

Using Spring HATEOAS. This helps you use navigable, restful APIs.



## Best Practice #3





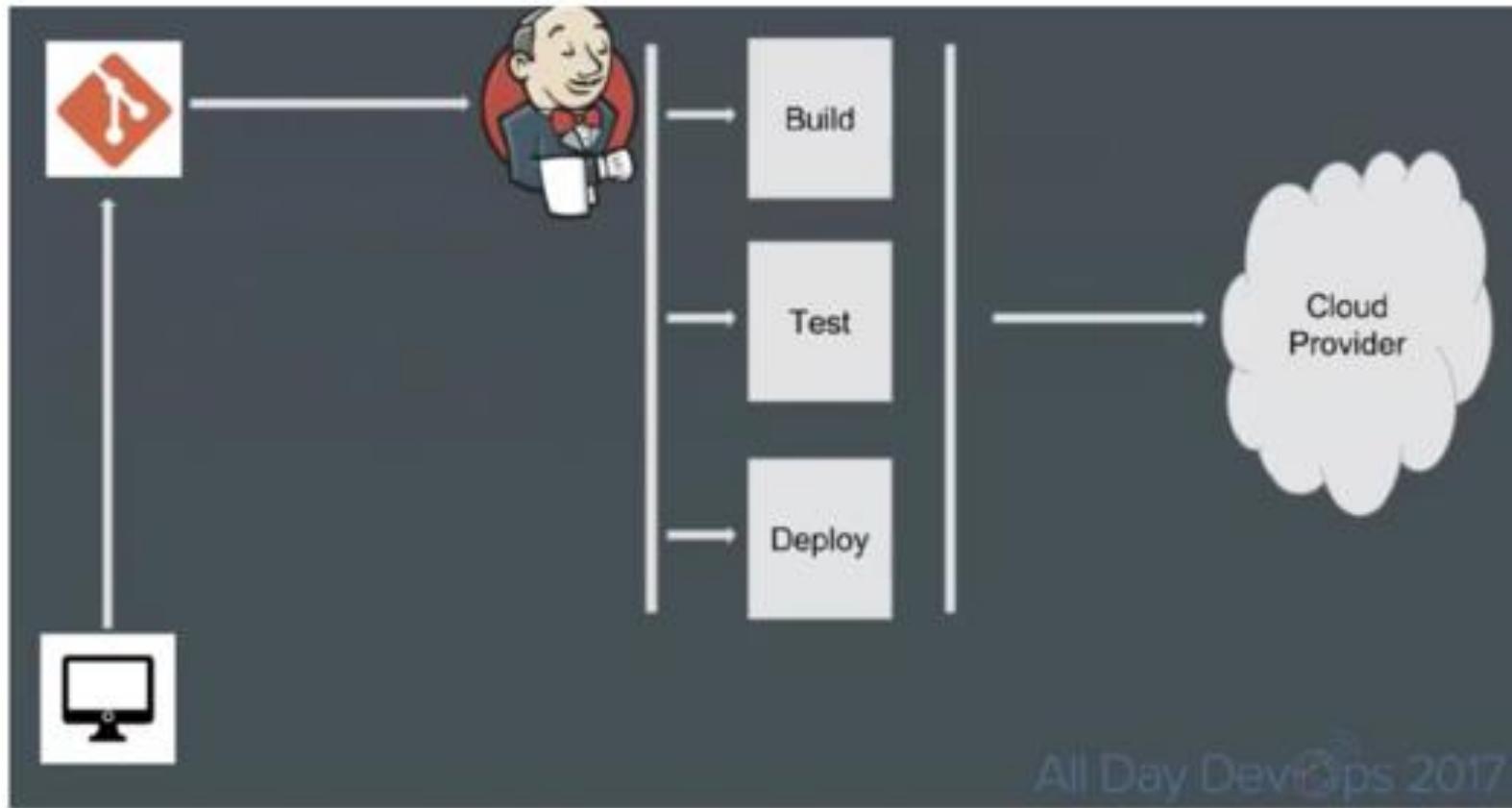
## Best Practice #4

---

Client code generation. Hüseyin suggests, “either using Swagger to generate your client code on any supported language or use feign client with a little annotation and client side load balancing with Ribbon.”



## Best Practice #5

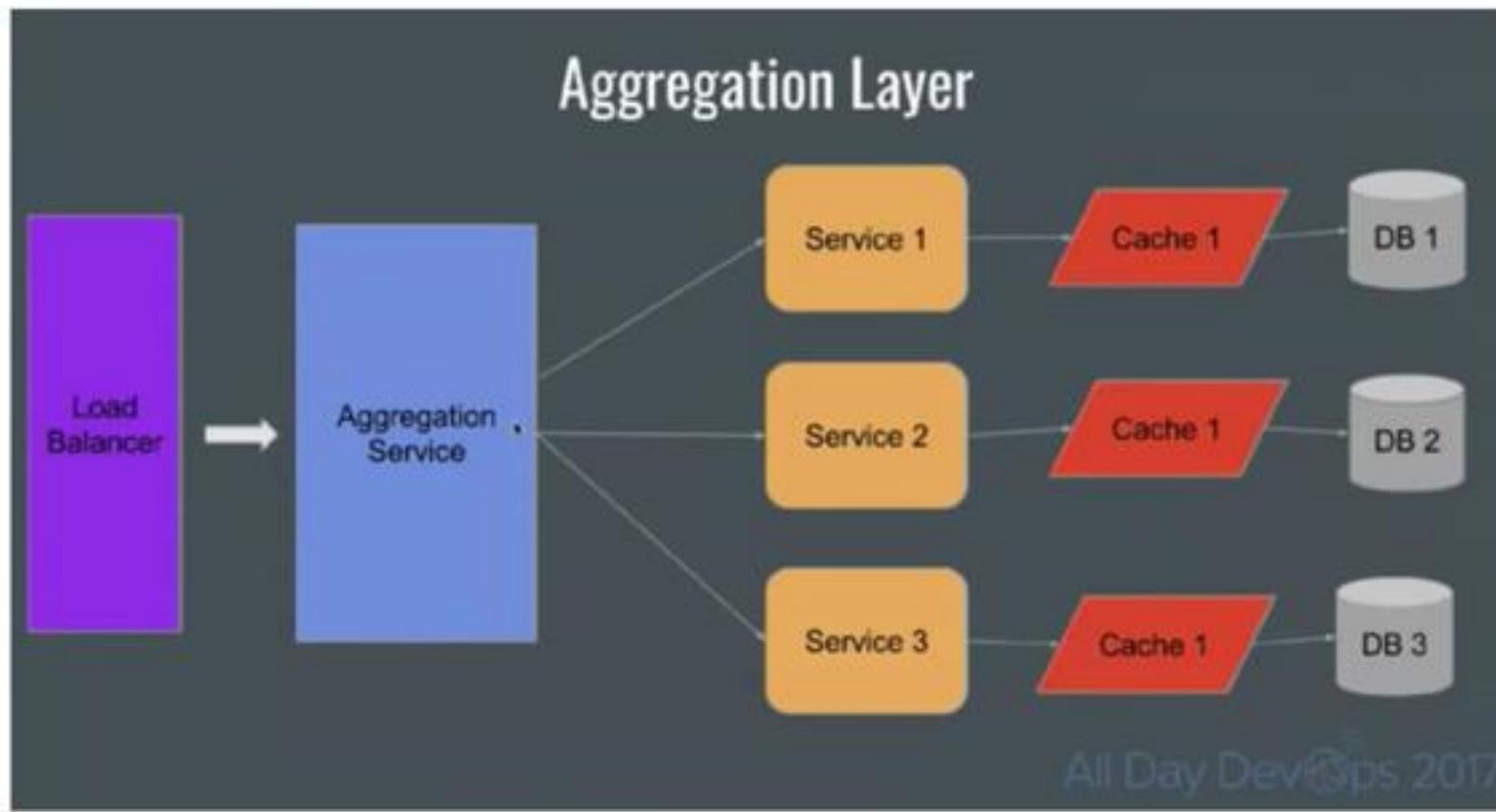




## Best Practice #6



## Best Practice #7





## Best Practice #8

---

Event sourcing and CQRS (Command and Query Responsibility Segregation). A Command alters the state of an object, but does not return data. A Query returns data, but does not alter the state of the object.

# Questions



# Module Summary

---

- Spring Integration Framework.
- Message, Channel and Adapter
- Understood the different Component Integration
- Understood the Event-Driven Architecture

