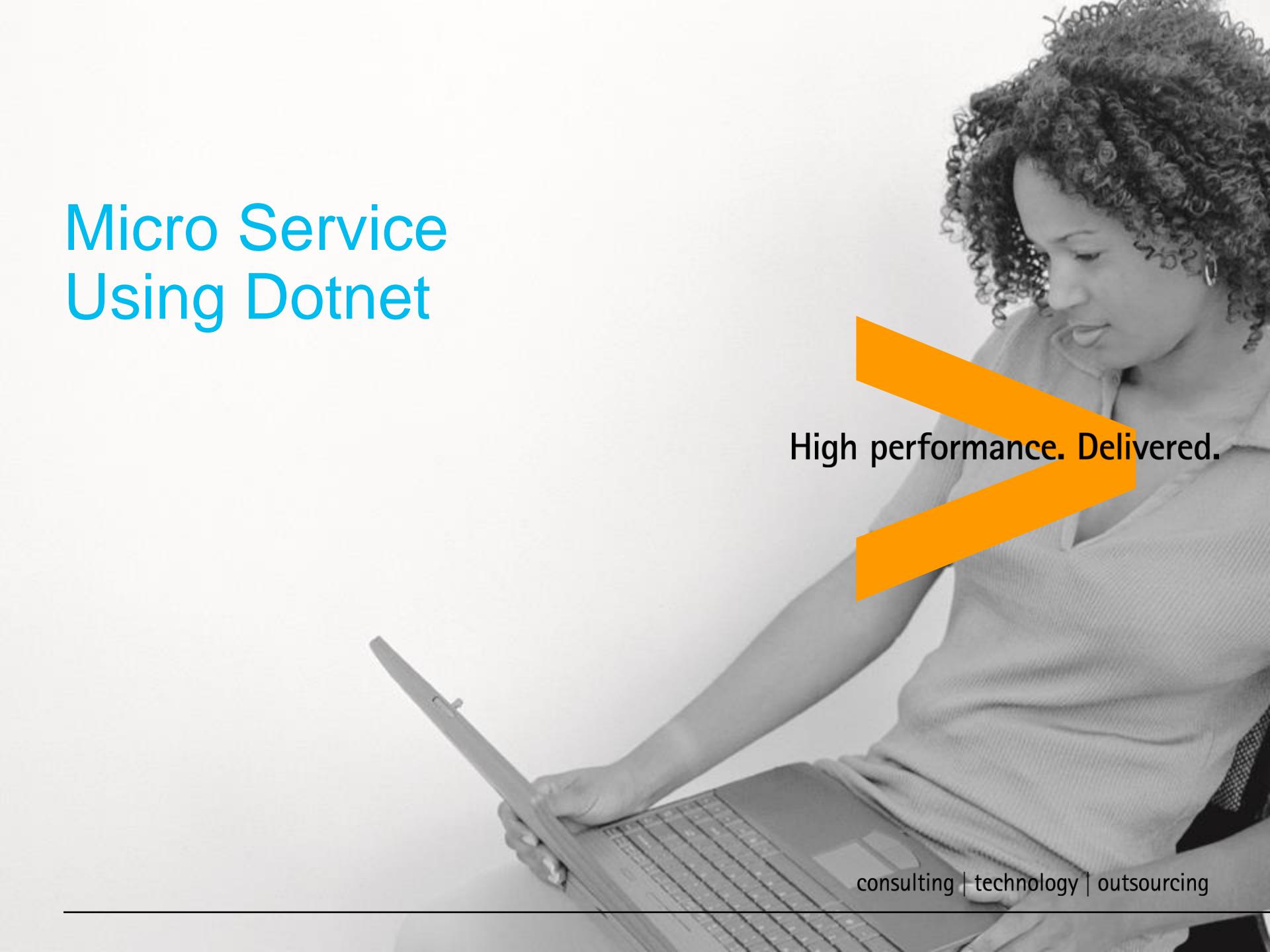


Micro Service Using Dotnet

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



- Monolith means composed all in one piece.
- The Monolithic application describes a single-tiered software application in which different components combined into a single program from a single platform.



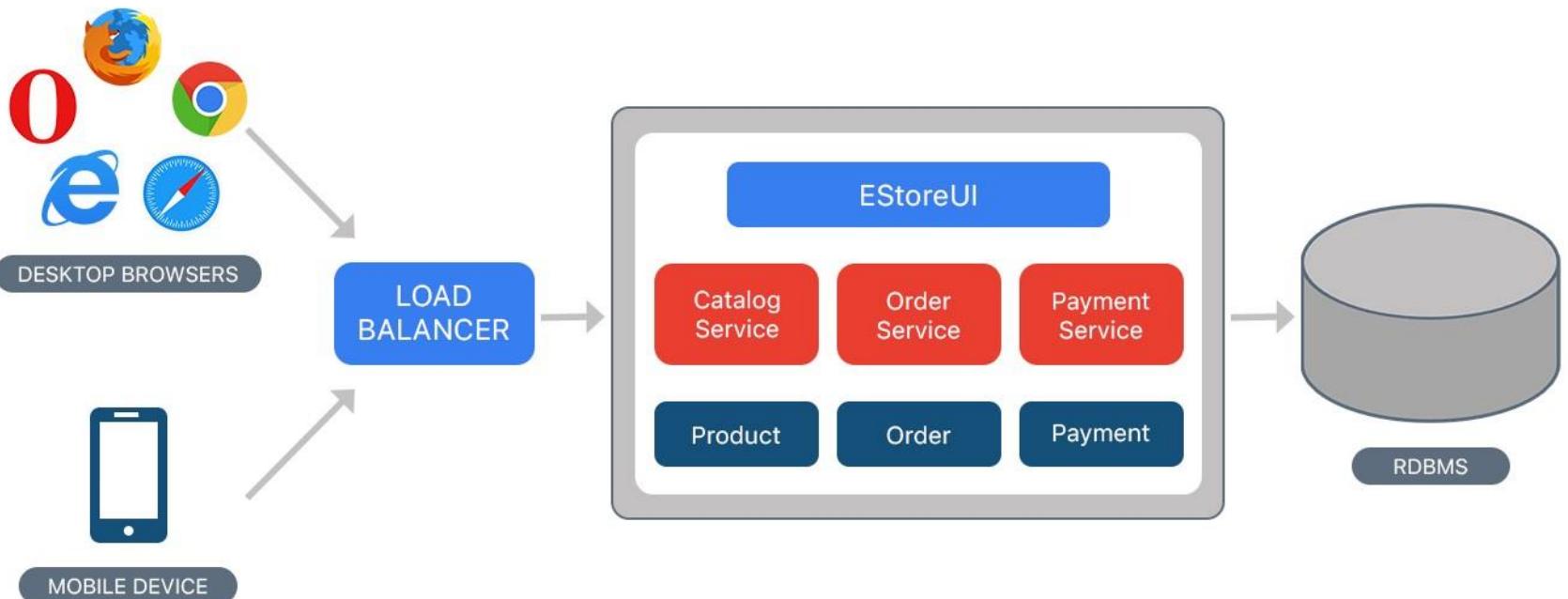
Monolithic Architecture

Components can be:

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



Example of Monolithic Approach





Drawbacks

- Maintenance — If Application is too large and complex to understand entirely, it is challenging to make changes fast and correctly.
- The size of the application can slow down the start-up time.
- You must redeploy the entire application on each update.
- Monolithic applications can also be challenging to scale when different modules have conflicting resource requirements.
- Reliability — Bug in any module (e.g. memory leak) can potentially bring down the entire process.
- Moreover, since all instances of the application are identical, that bug impact the availability of the entire application



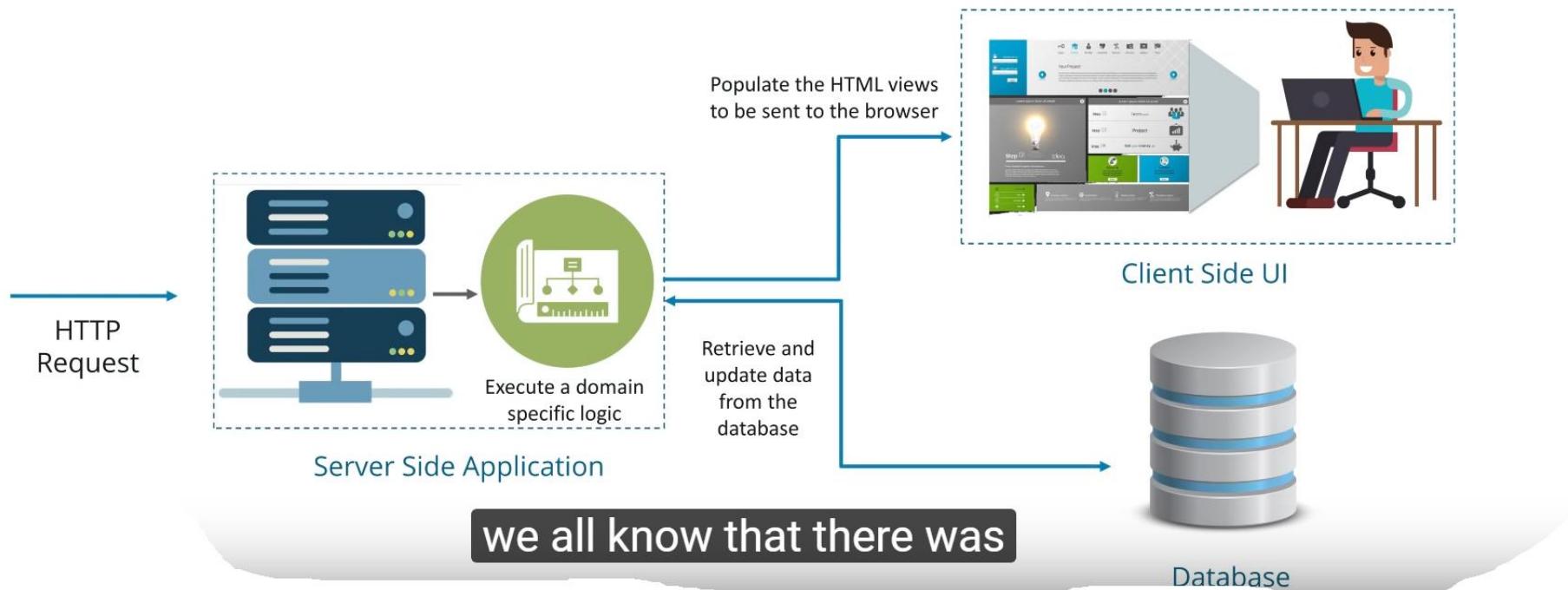
Drawbacks

- Regardless of how easy the initial stages may seem, Monolithic applications have difficulty to adopting new and advance technologies.
- Since changes in languages or frameworks affect an entire application.
- It requires efforts to thoroughly work with the app details.
- Hence it is costly considering both time and efforts.



Monolithic Architecture

Monolithic Architecture is like a big container wherein all the software components of an application are assembled together and tightly packaged



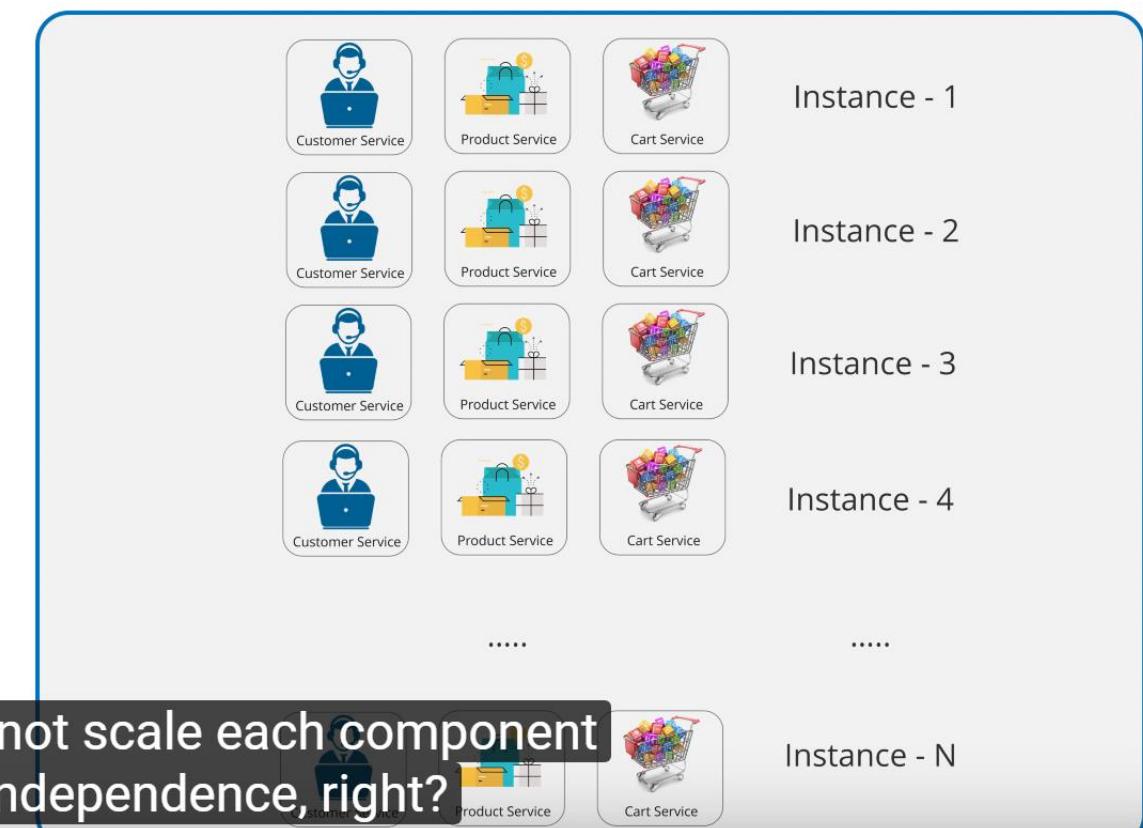
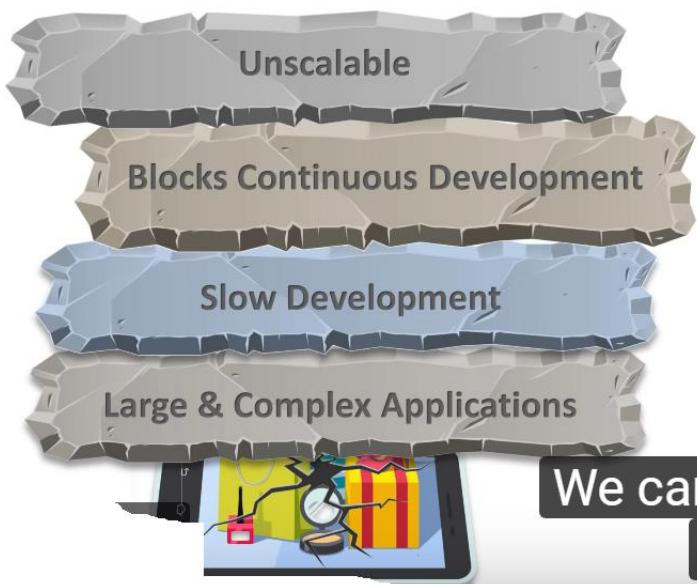


Monolithic Architecture



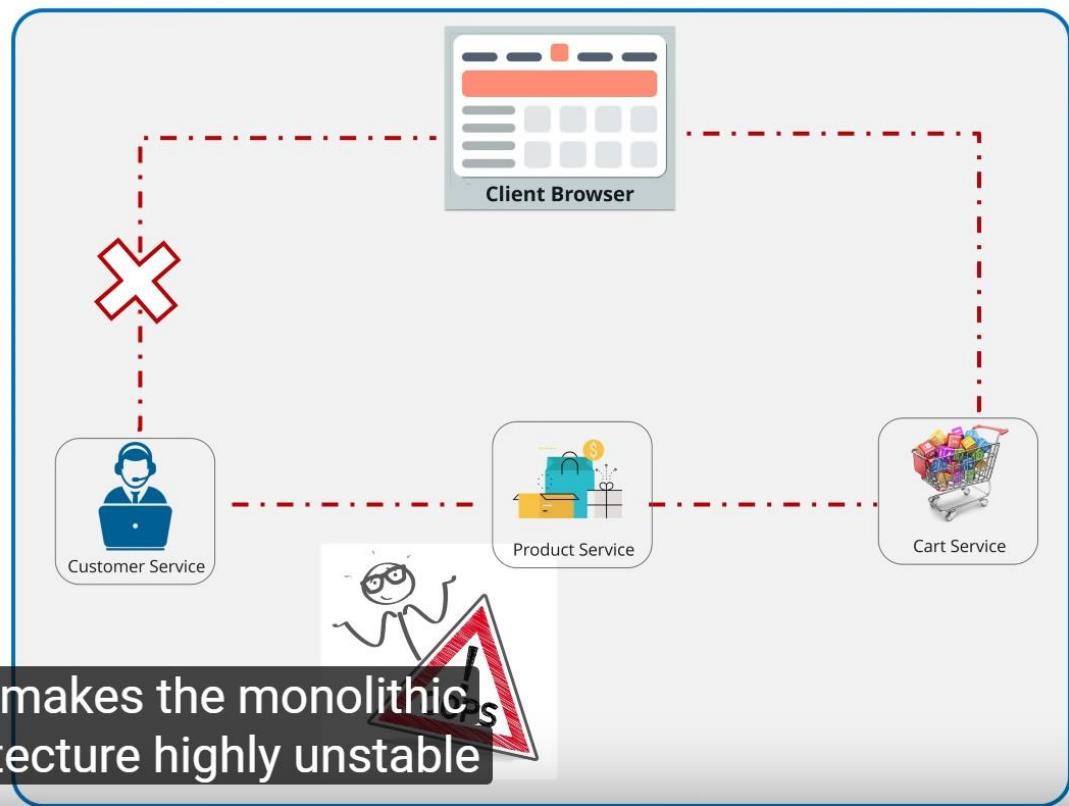


Monolithic Architecture





Monolithic Architecture

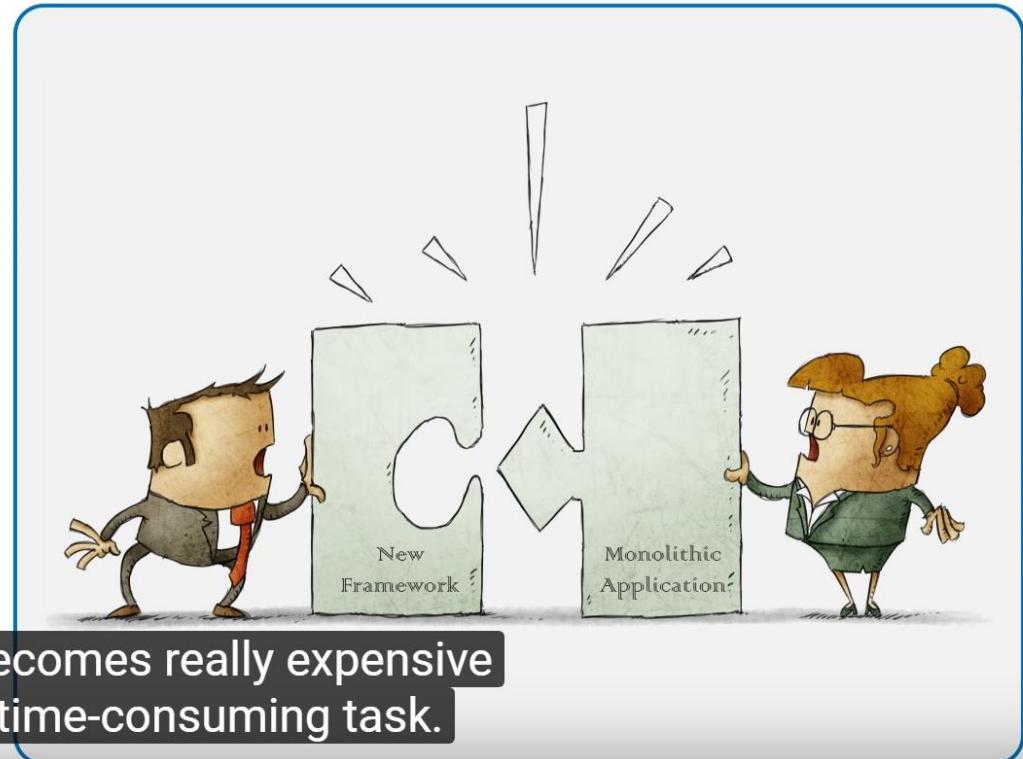




Monolithic Architecture

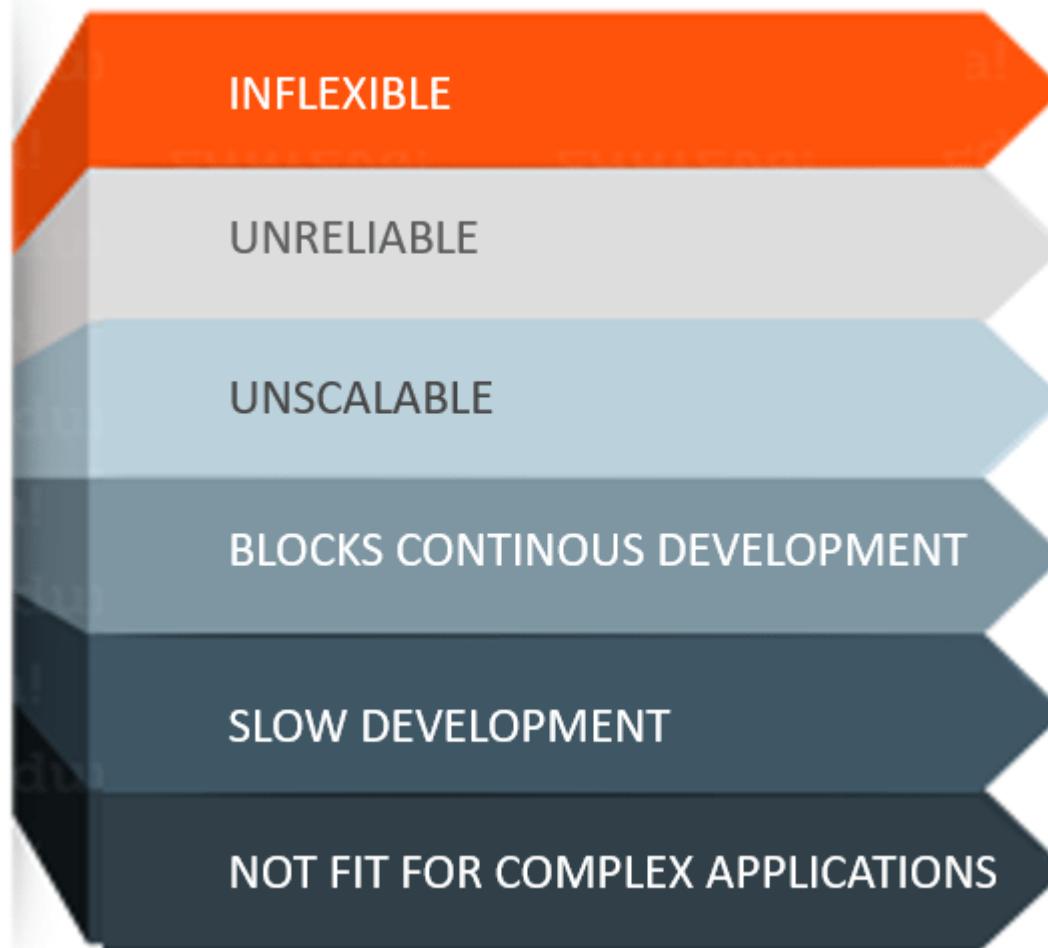


So it becomes really expensive
and time-consuming task.





Monolithic Architecture



Micro Service

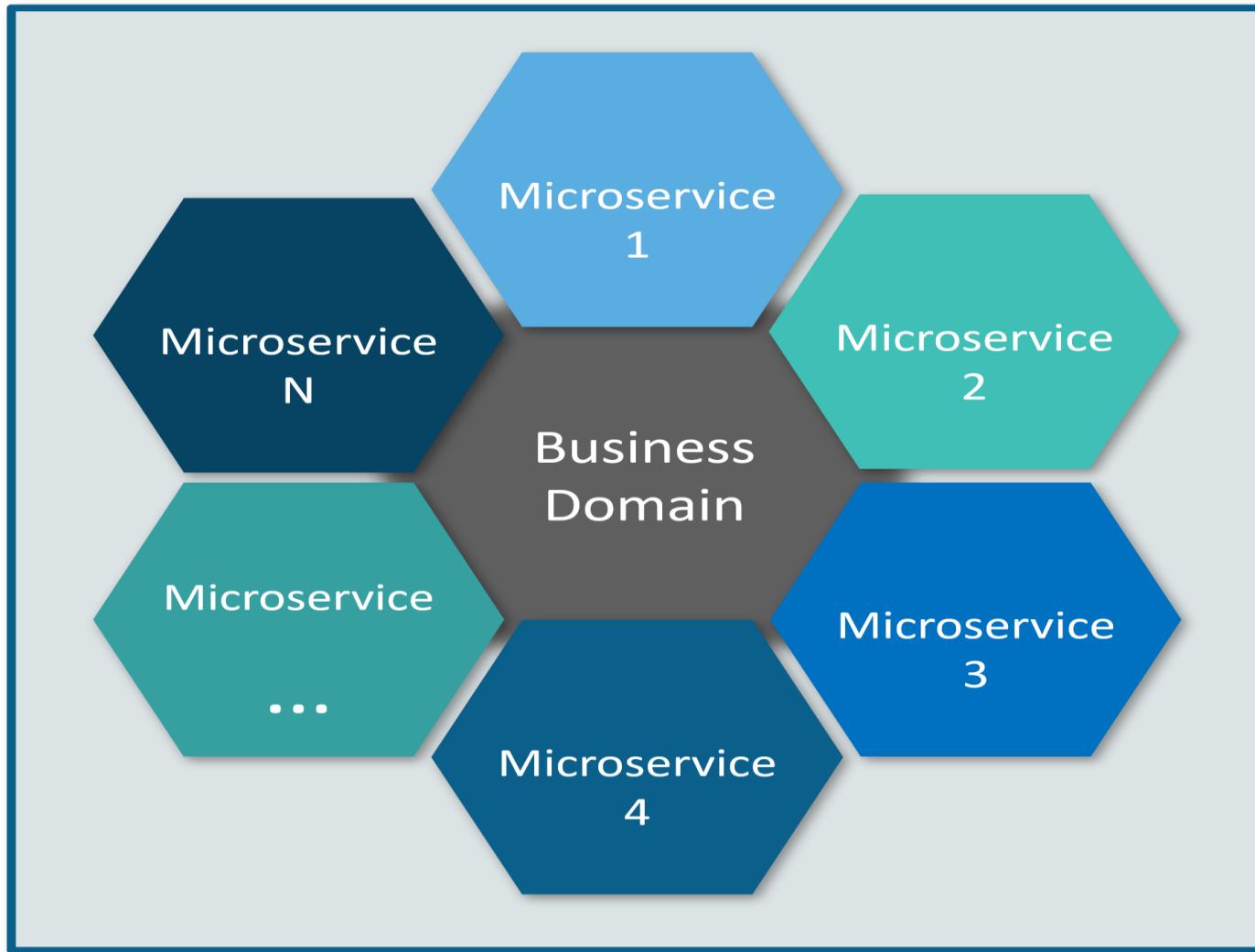


Small autonomous services that work together - Sam Newman

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





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

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*



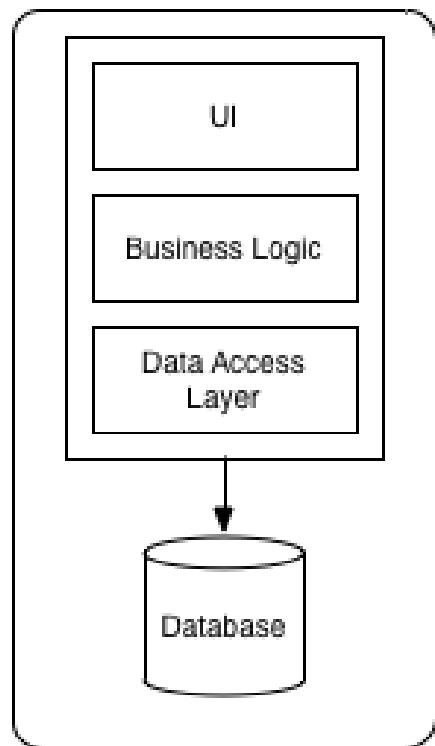
Micro Service



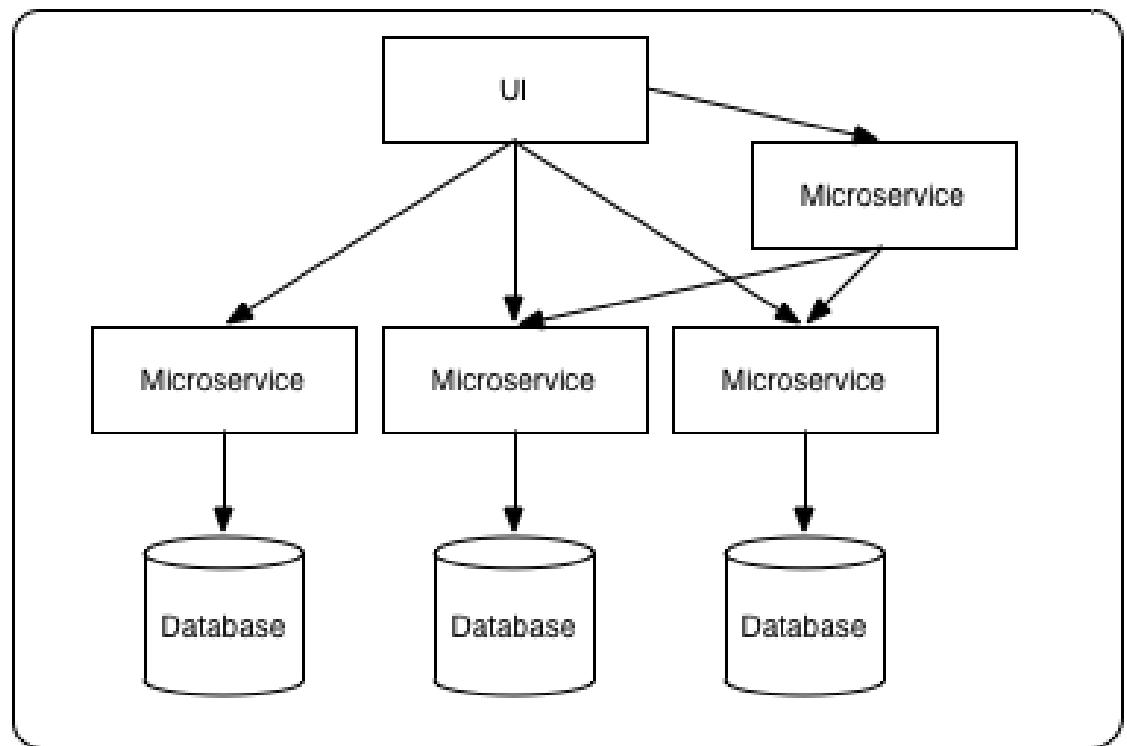
There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies - James Lewis and Martin Fowler



Microservice



Monolithic Architecture

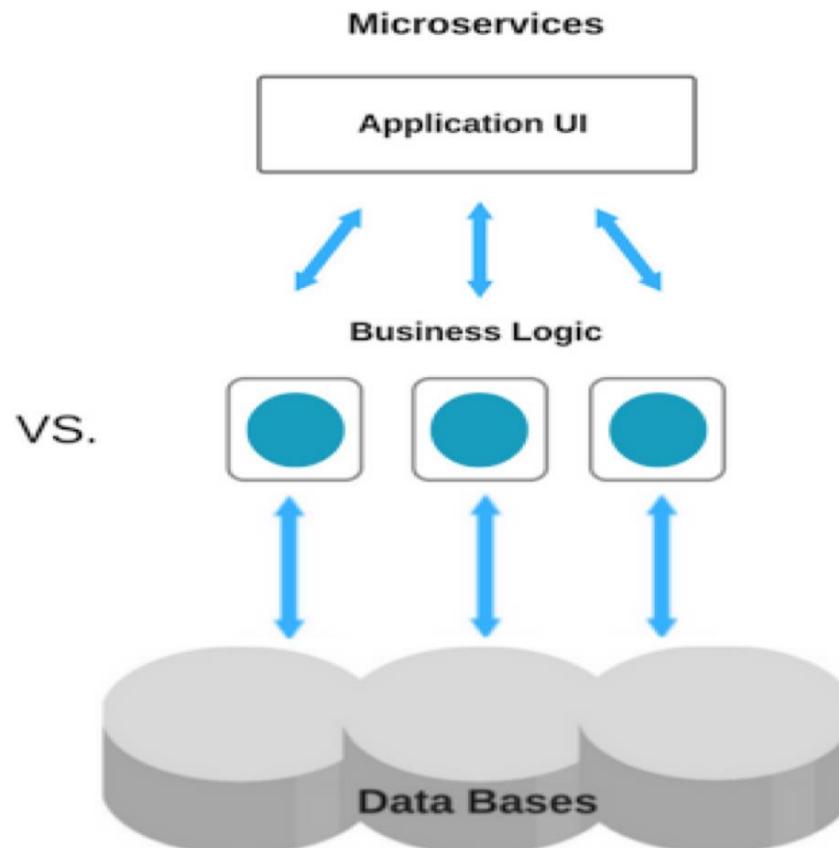
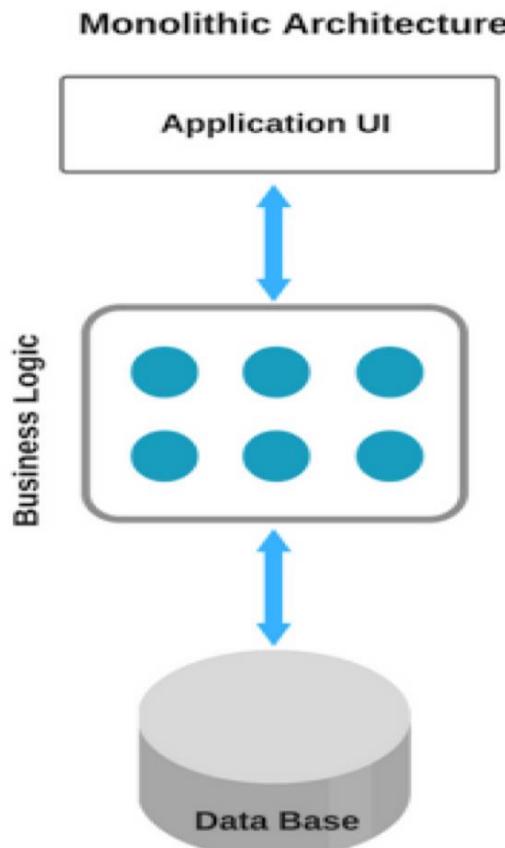


Microservices Architecture

MicroService

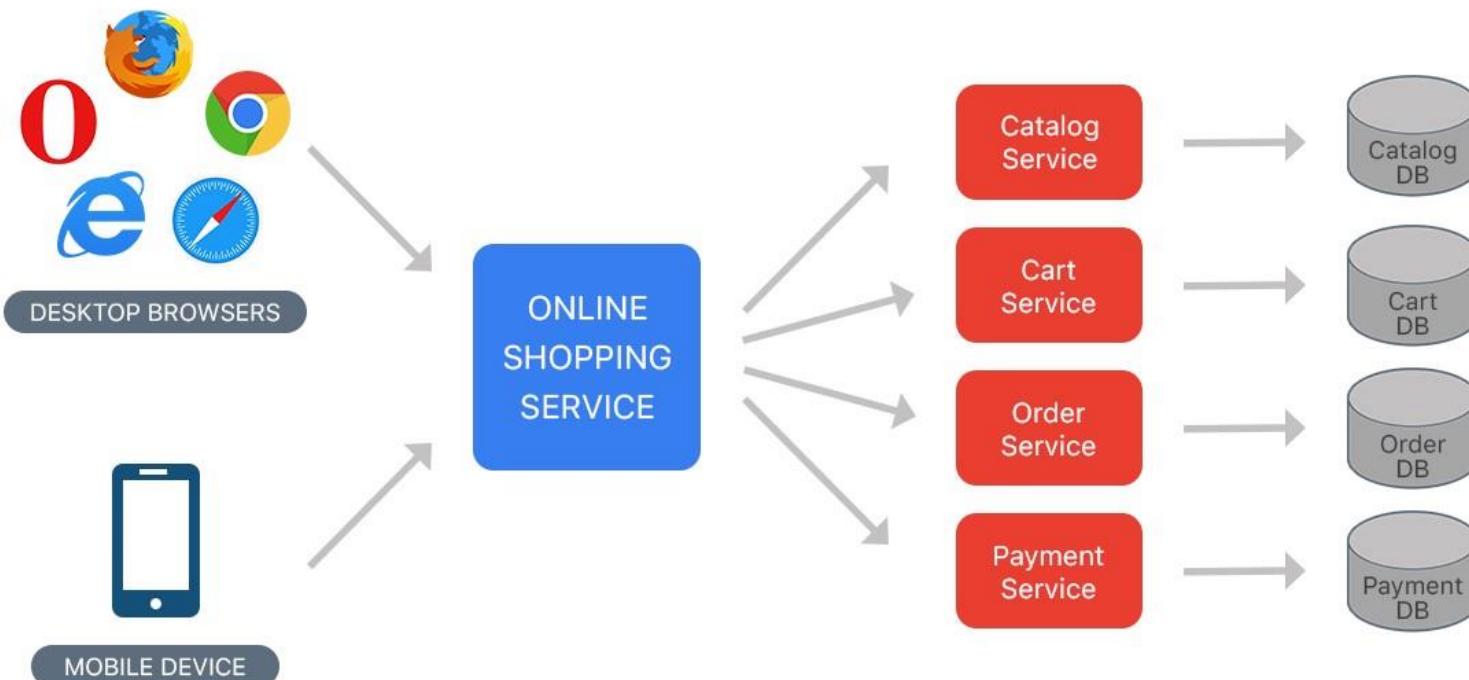


The difference between the monolithic and microservices architecture

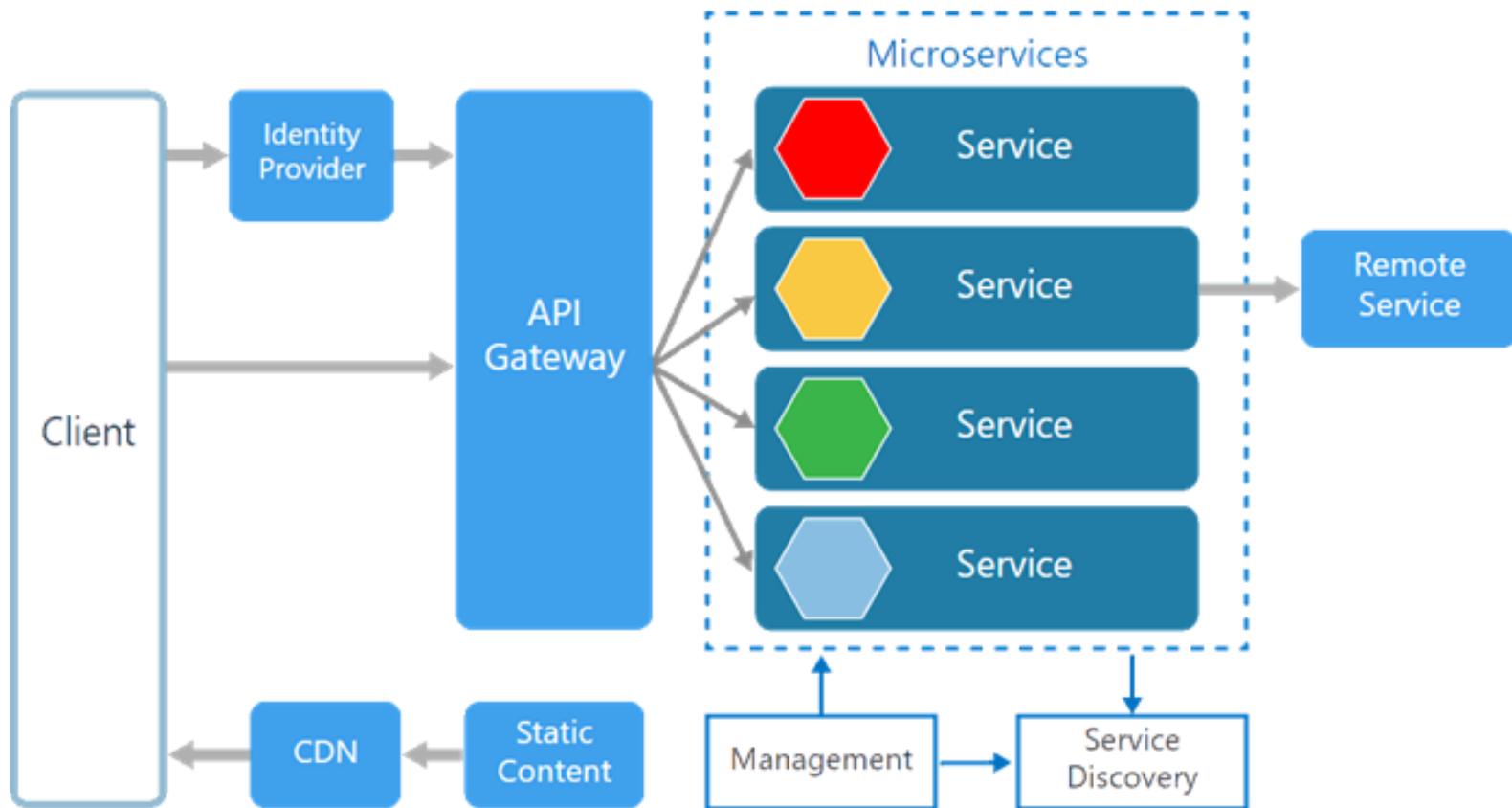


VS.

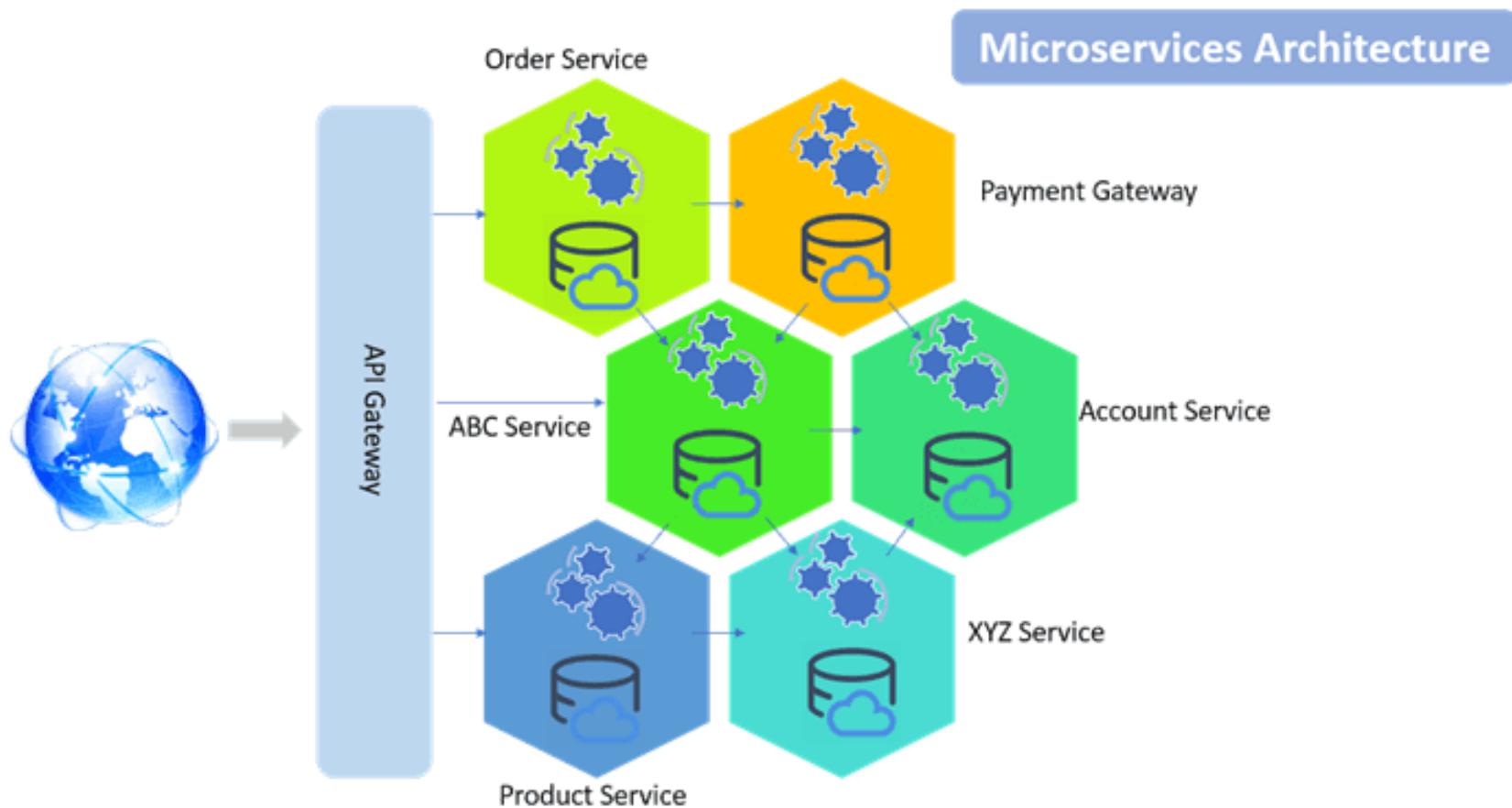
Micro Service



Micro Service

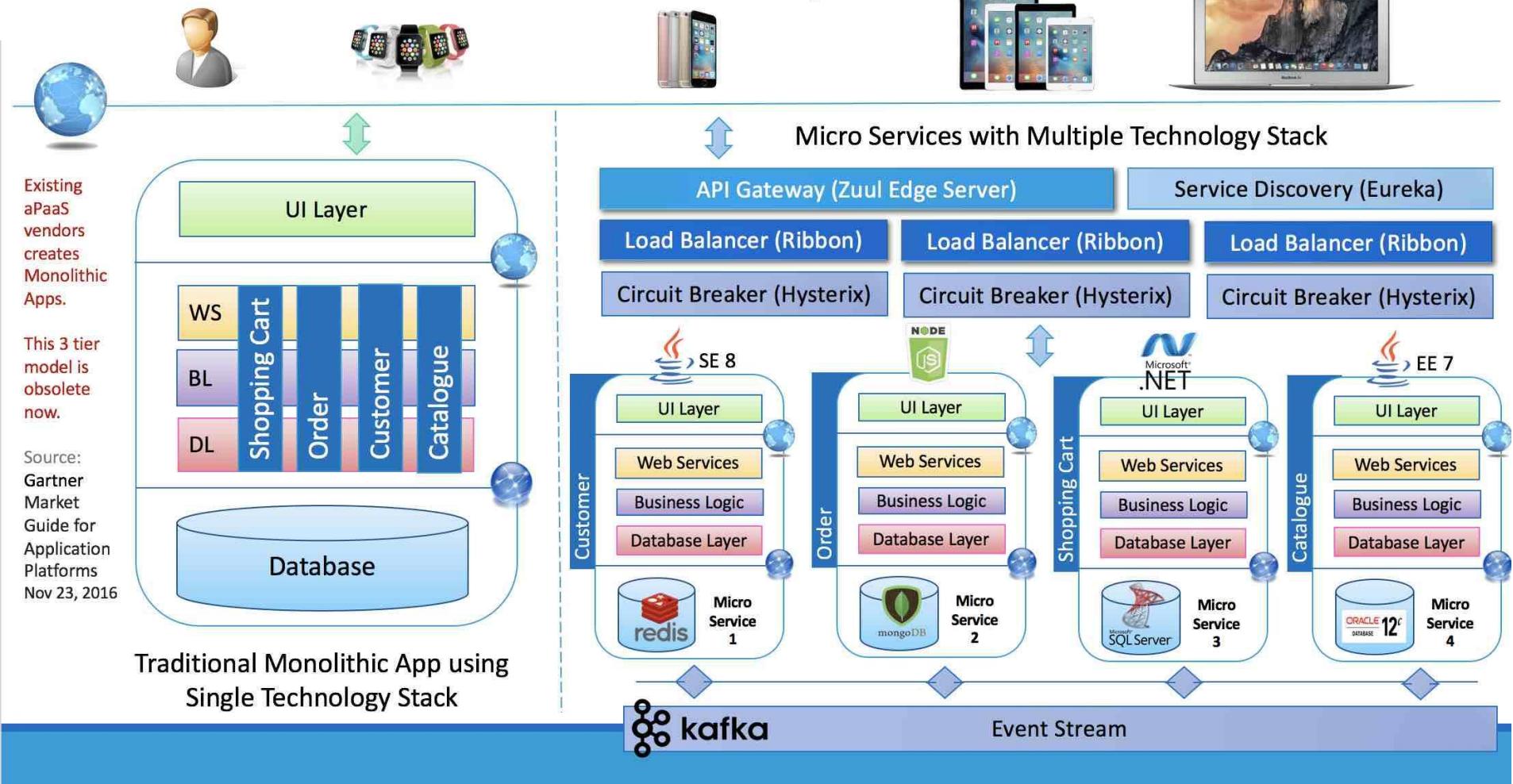


Micro Service



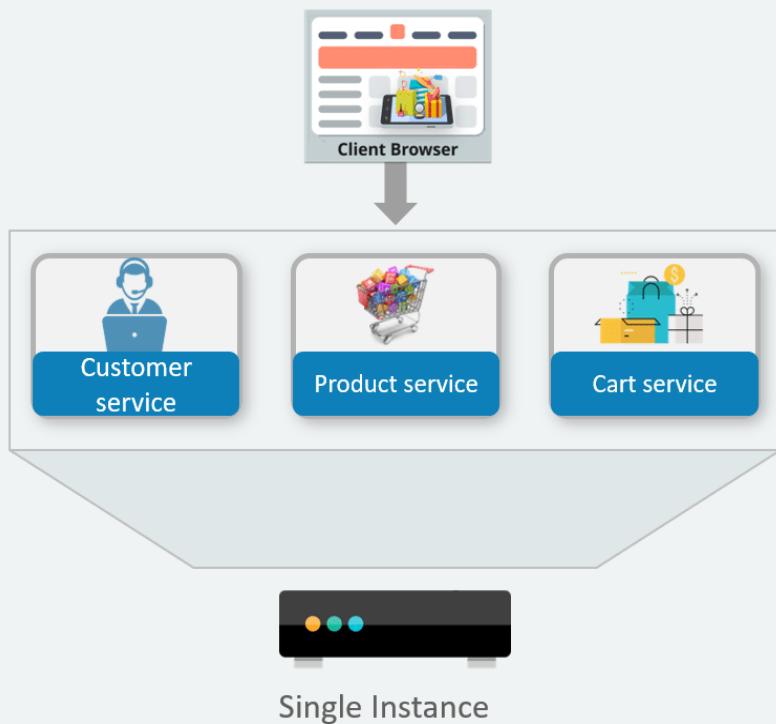


Monolithic vs. Micro Services Example

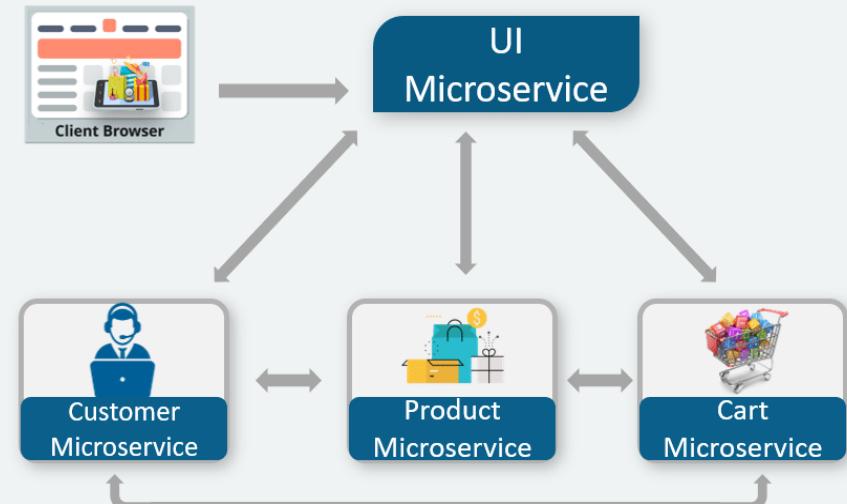




Monolithic Architecture

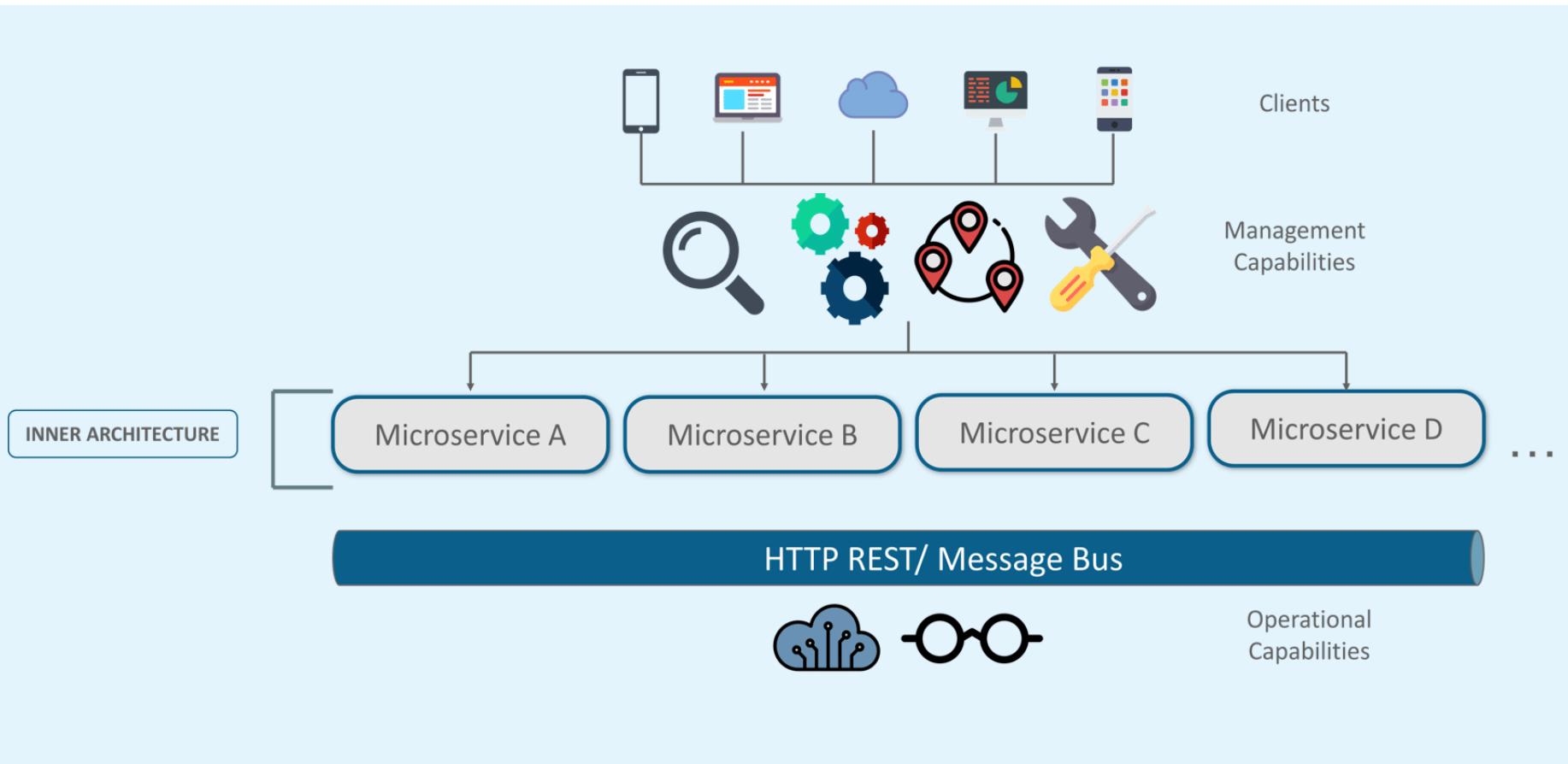


Microservice Architecture





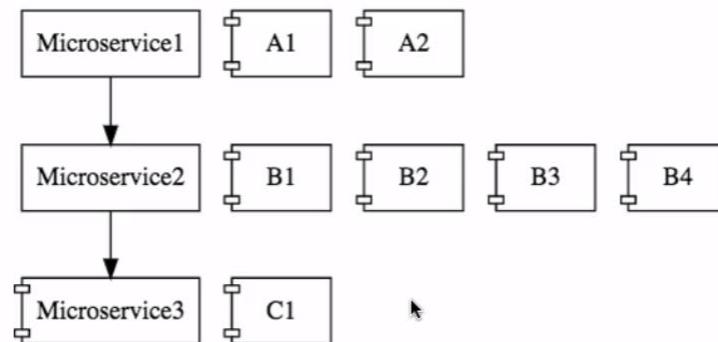
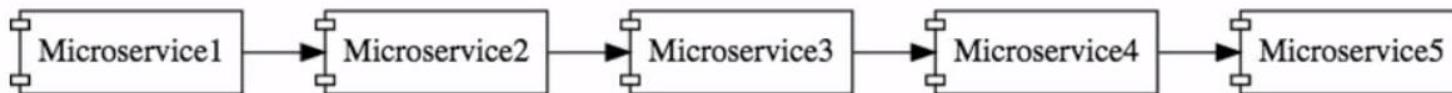
Microservice Architecture



Micro Service



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





What is Microservice

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



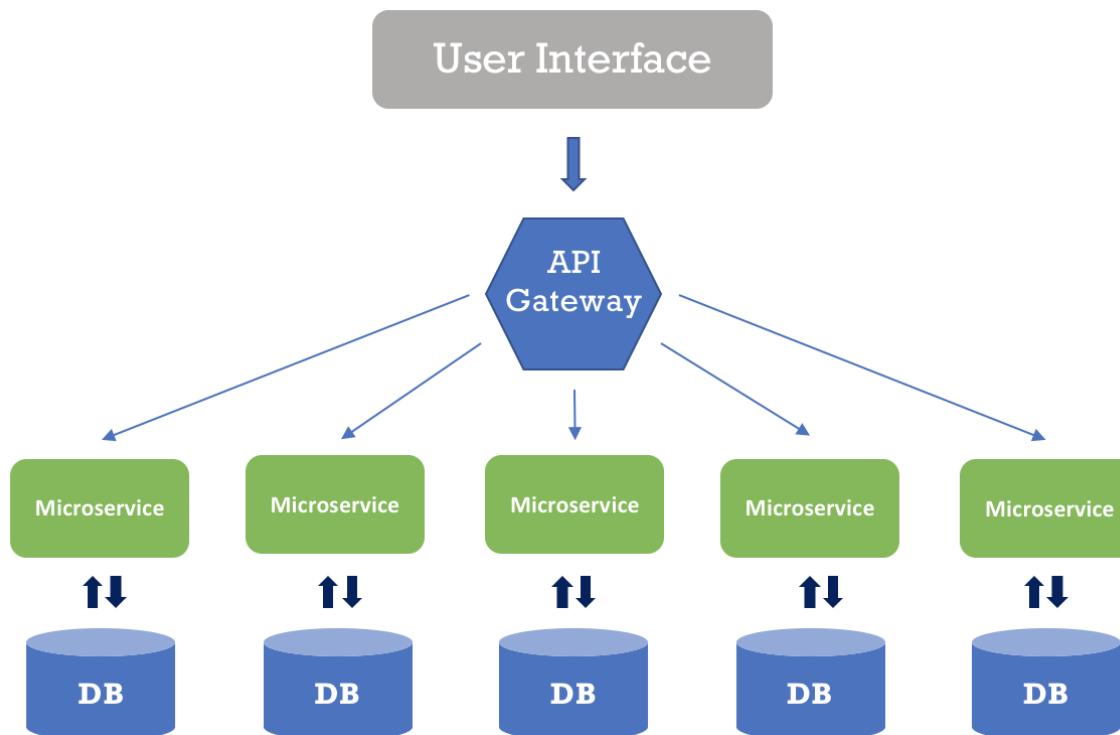
What is Microservice

- 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.

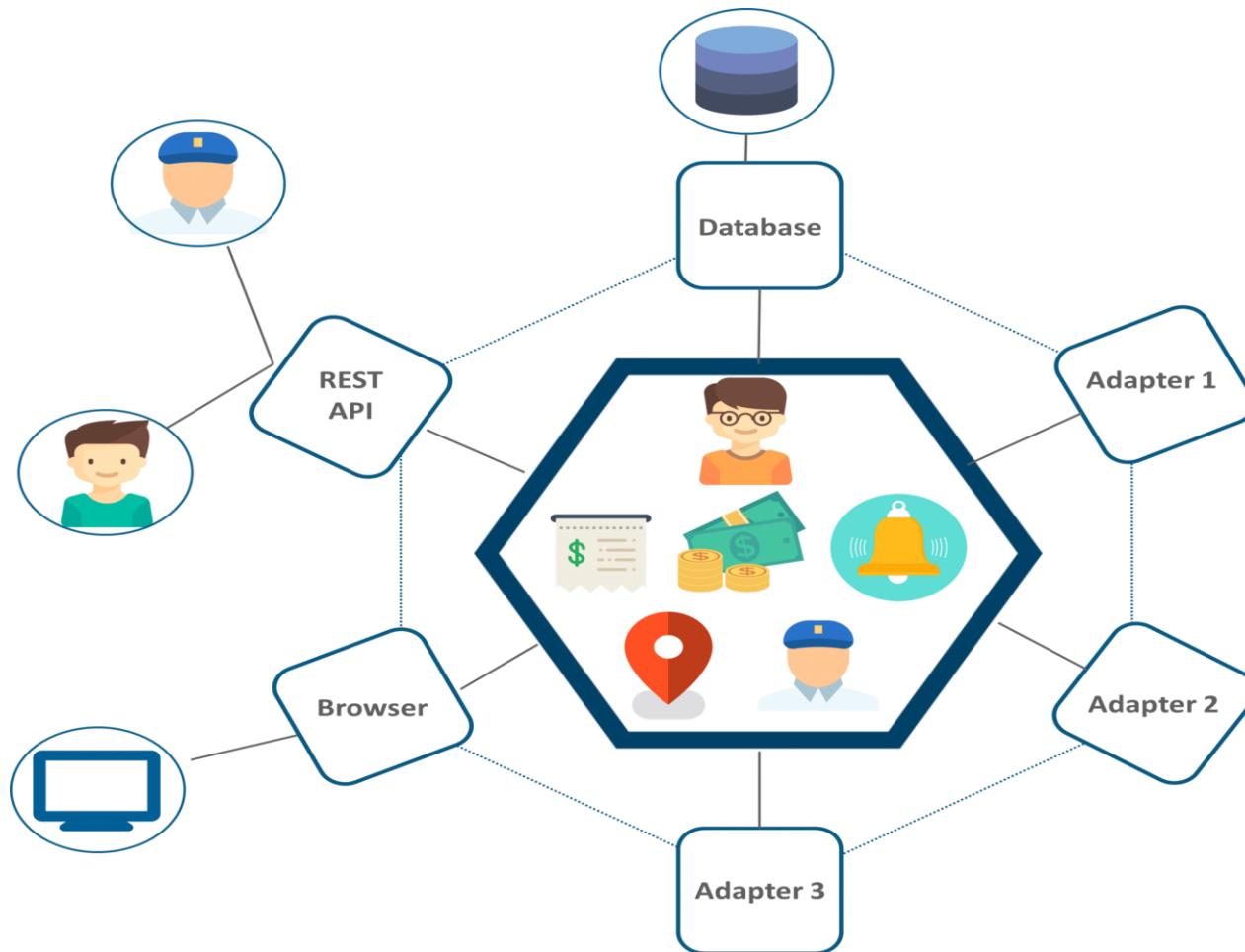


What is Microservices

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



UBER's Previous Architecture





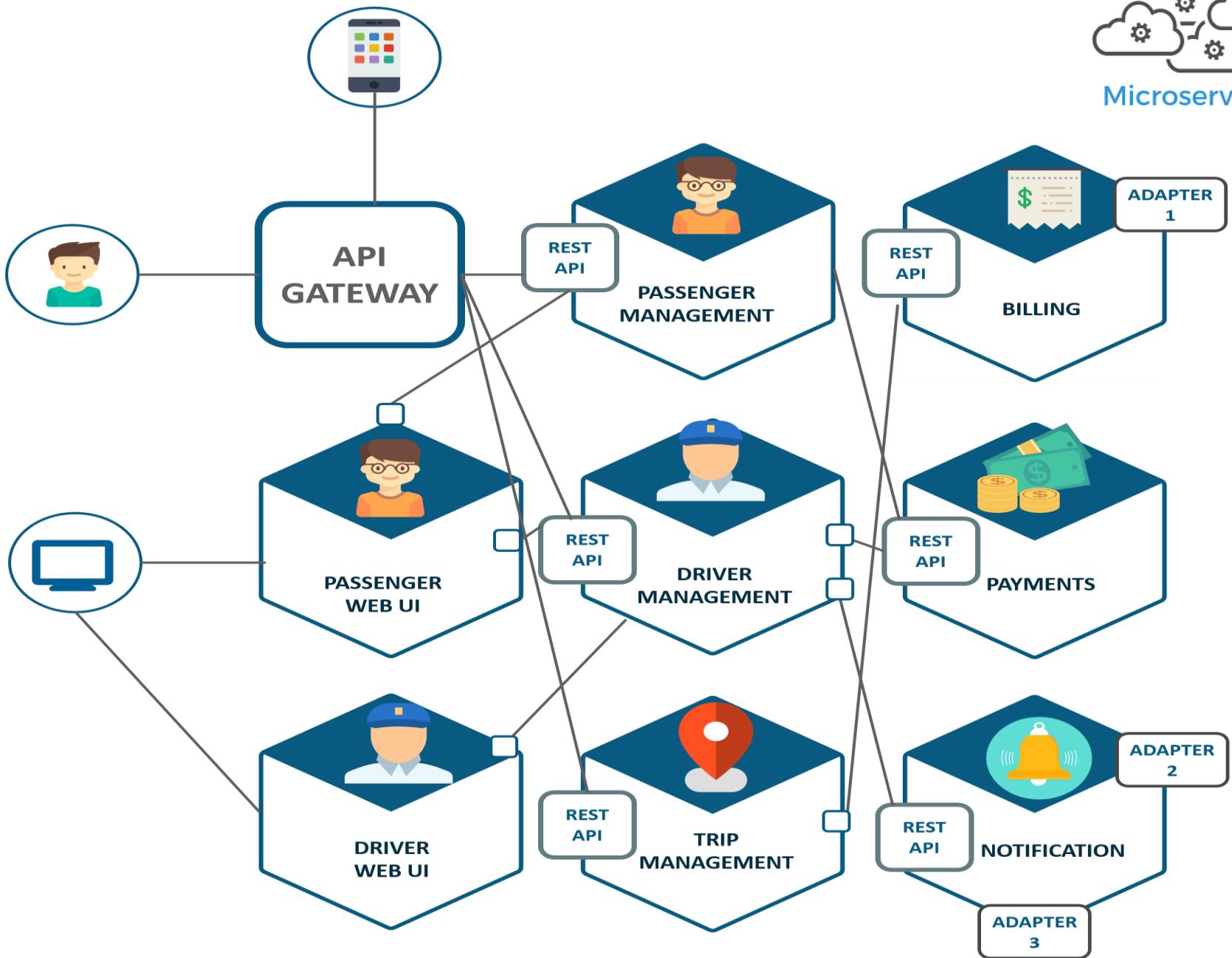
Problem Statement

- 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.

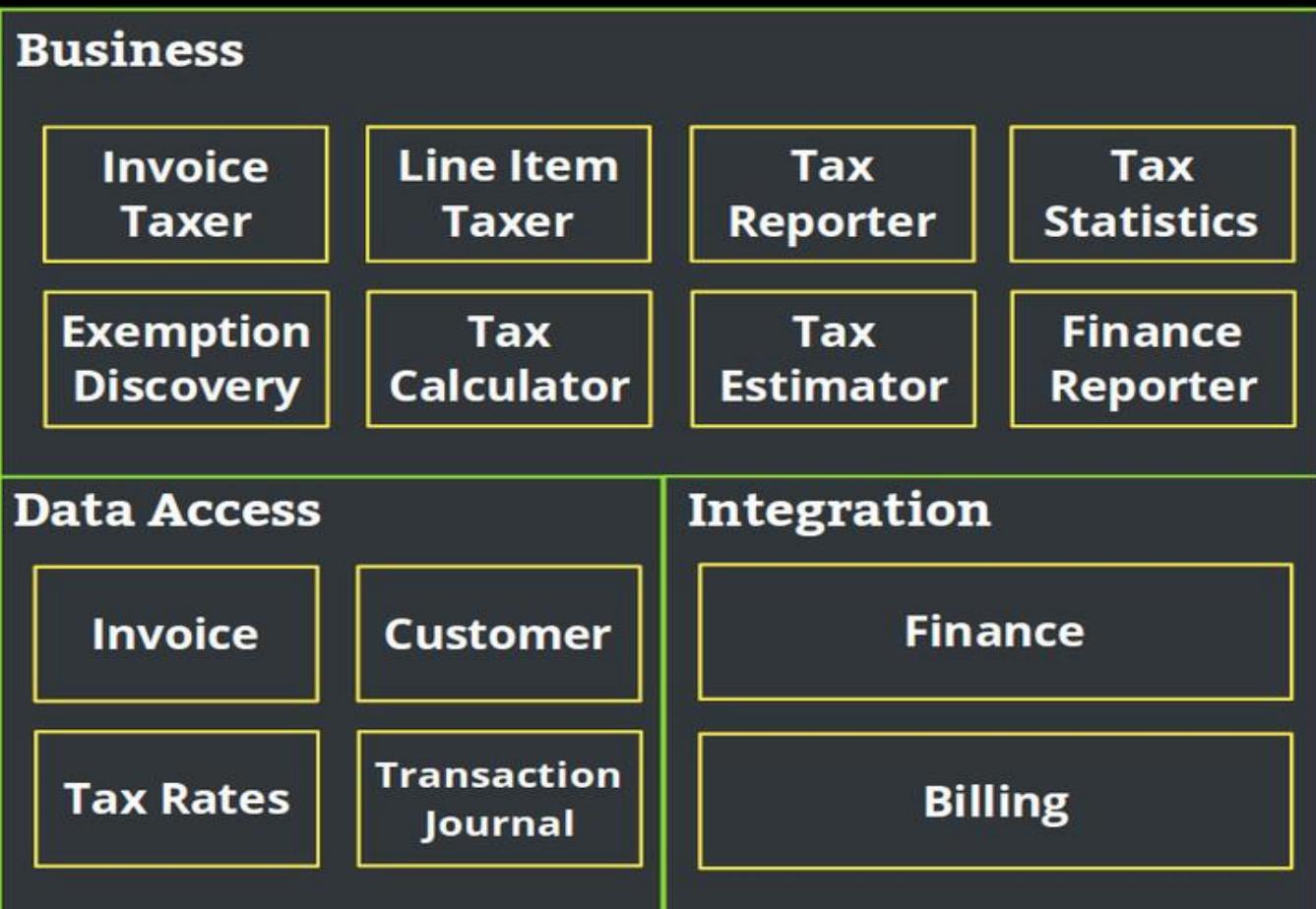




Existing Scenarios

- 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



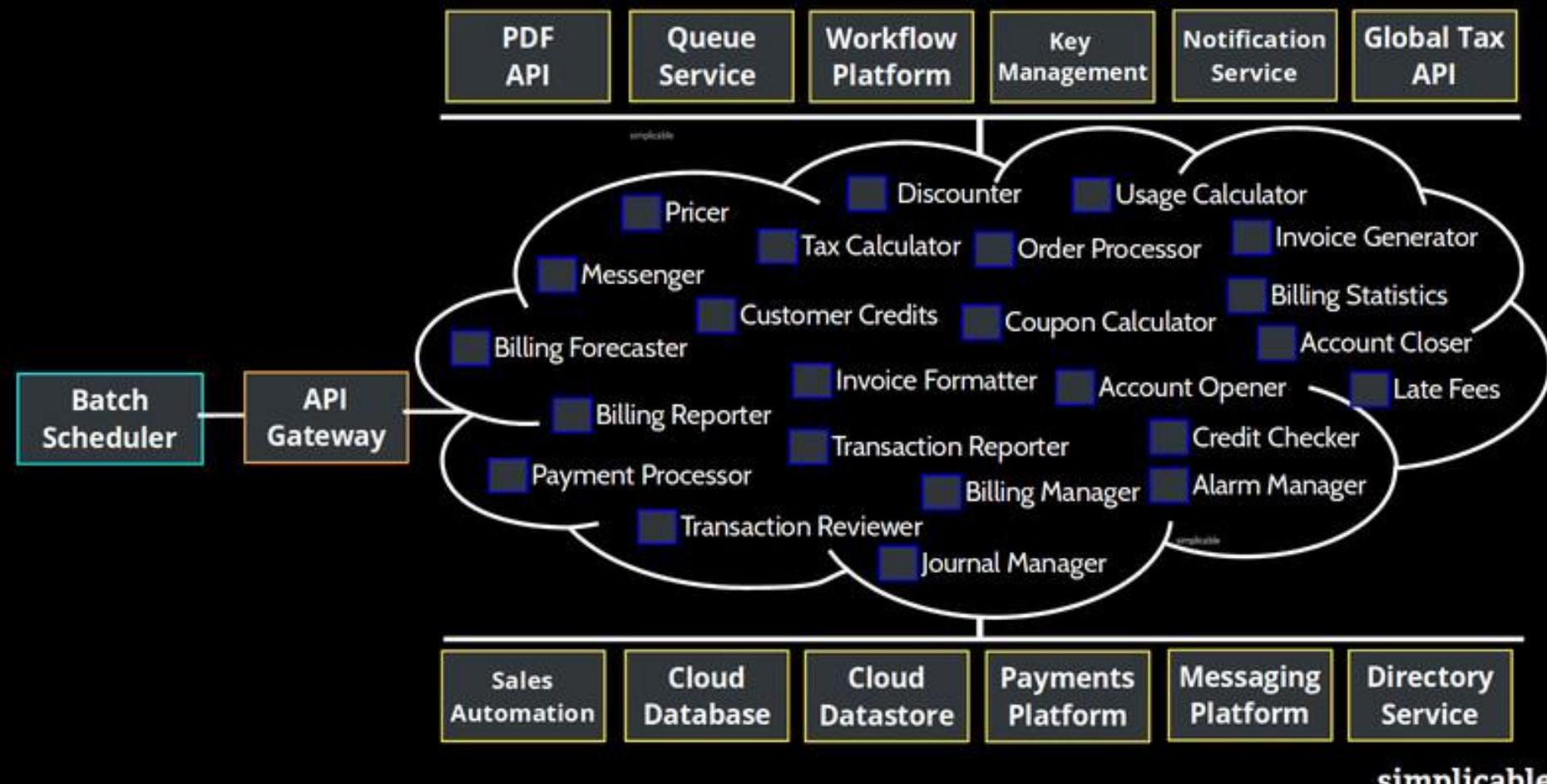
Cloud
Database

Services

simplicable



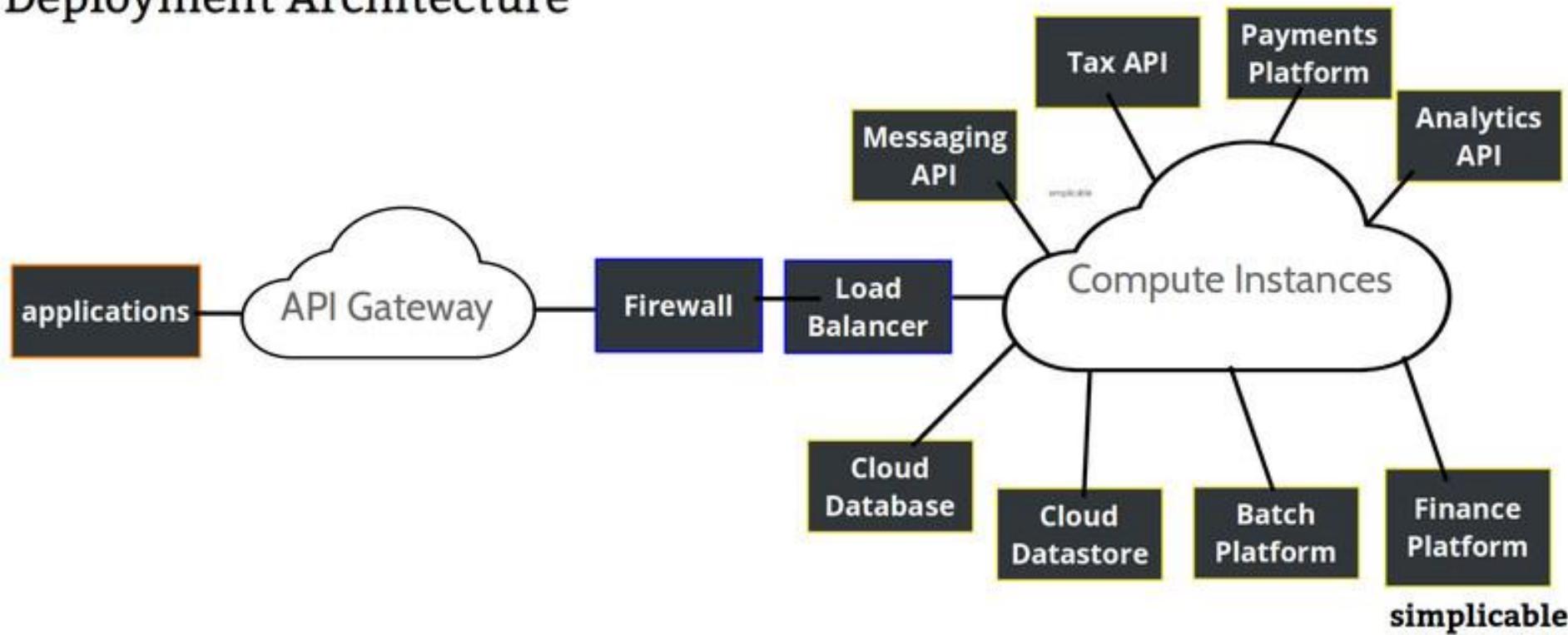
Services Architecture



simplicable



Deployment Architecture

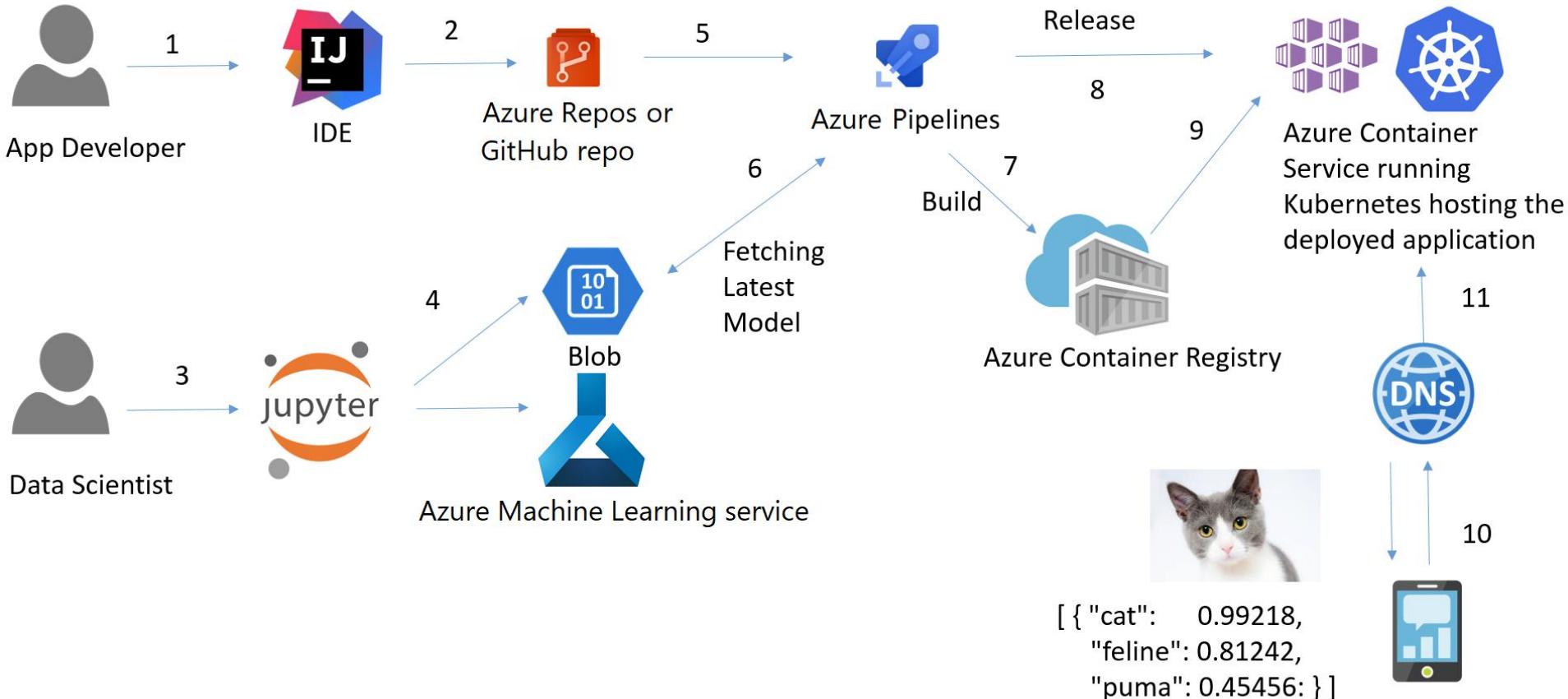




Principles of Micro service

- Independent & Autonomous Services
- Scalability
- Decentralization
- Resilient Services
- Real-Time Load Balancing
- Availability
- Continuous delivery through DevOps Integration
- Seamless API Integration and Continuous Monitoring
- Isolation from Failures
- Auto -Provisioning

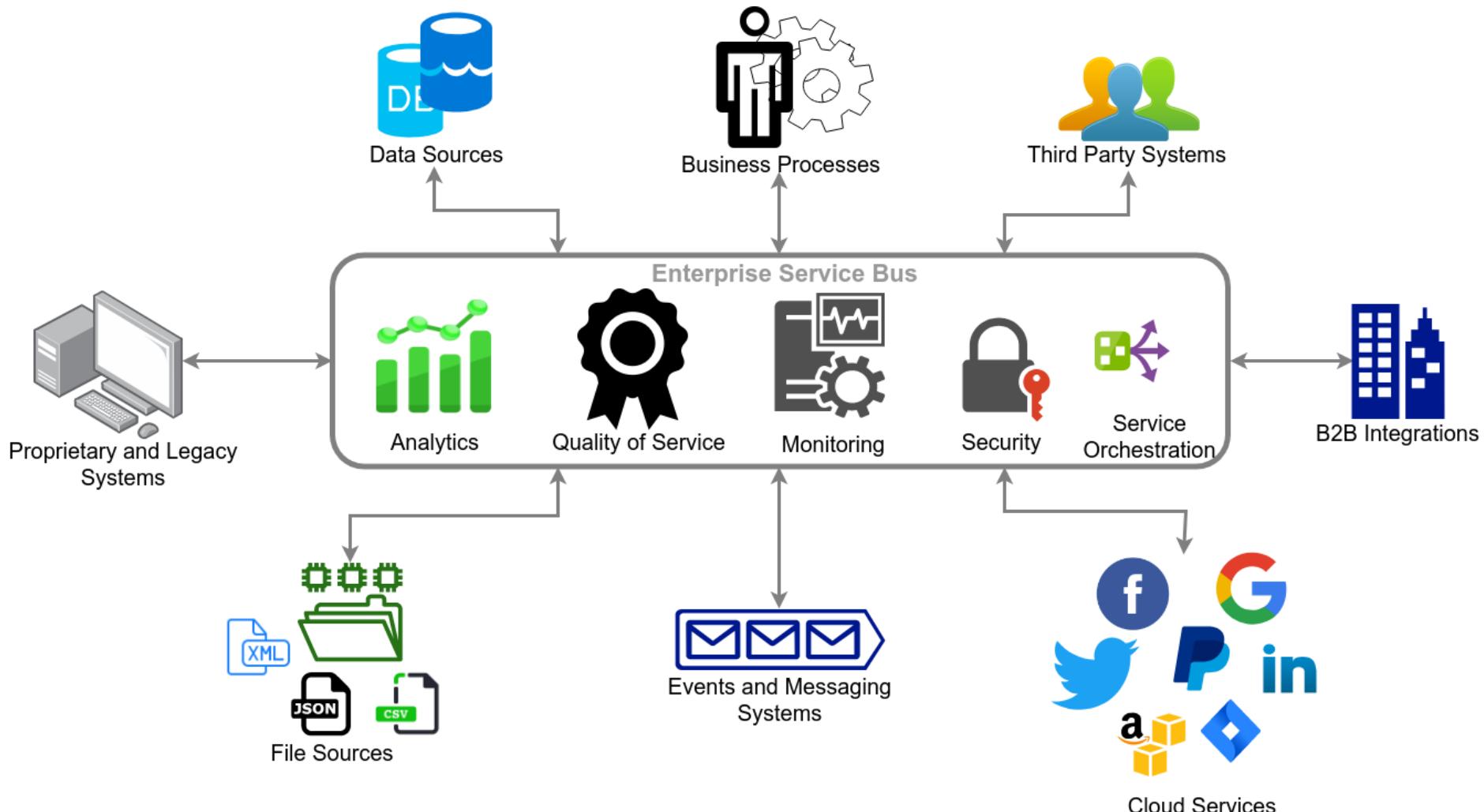
Continuous delivery through DevOps Integration



API Integration ESB or Microservices



ESB for API Integration



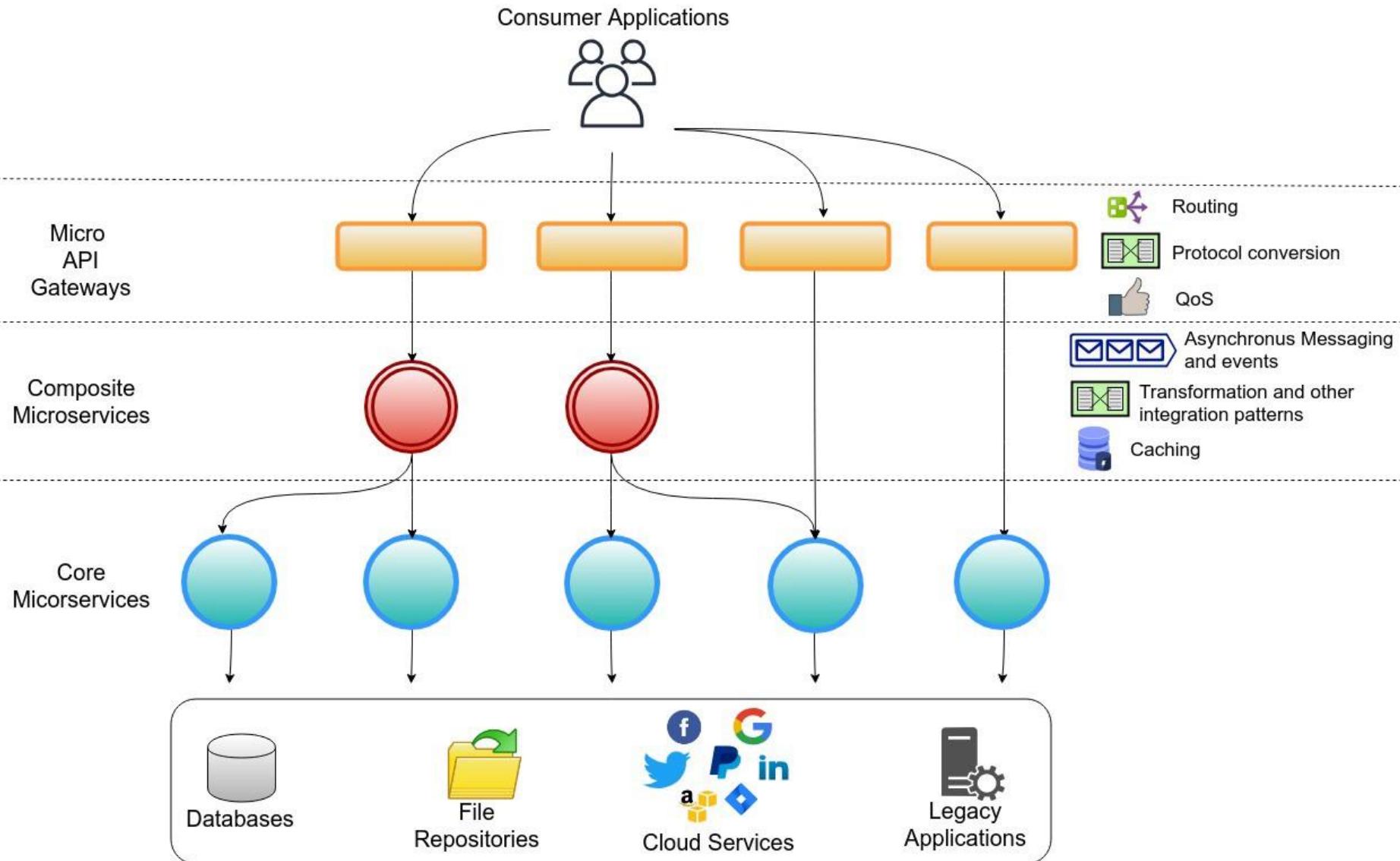
Disadvantages of ESB



- **Added delay in round-trip**
- **Connectivity requirements(All calls through ESB)**
- **Single point of failure**
- **Complexity in scalability**



Microservice for API Integration



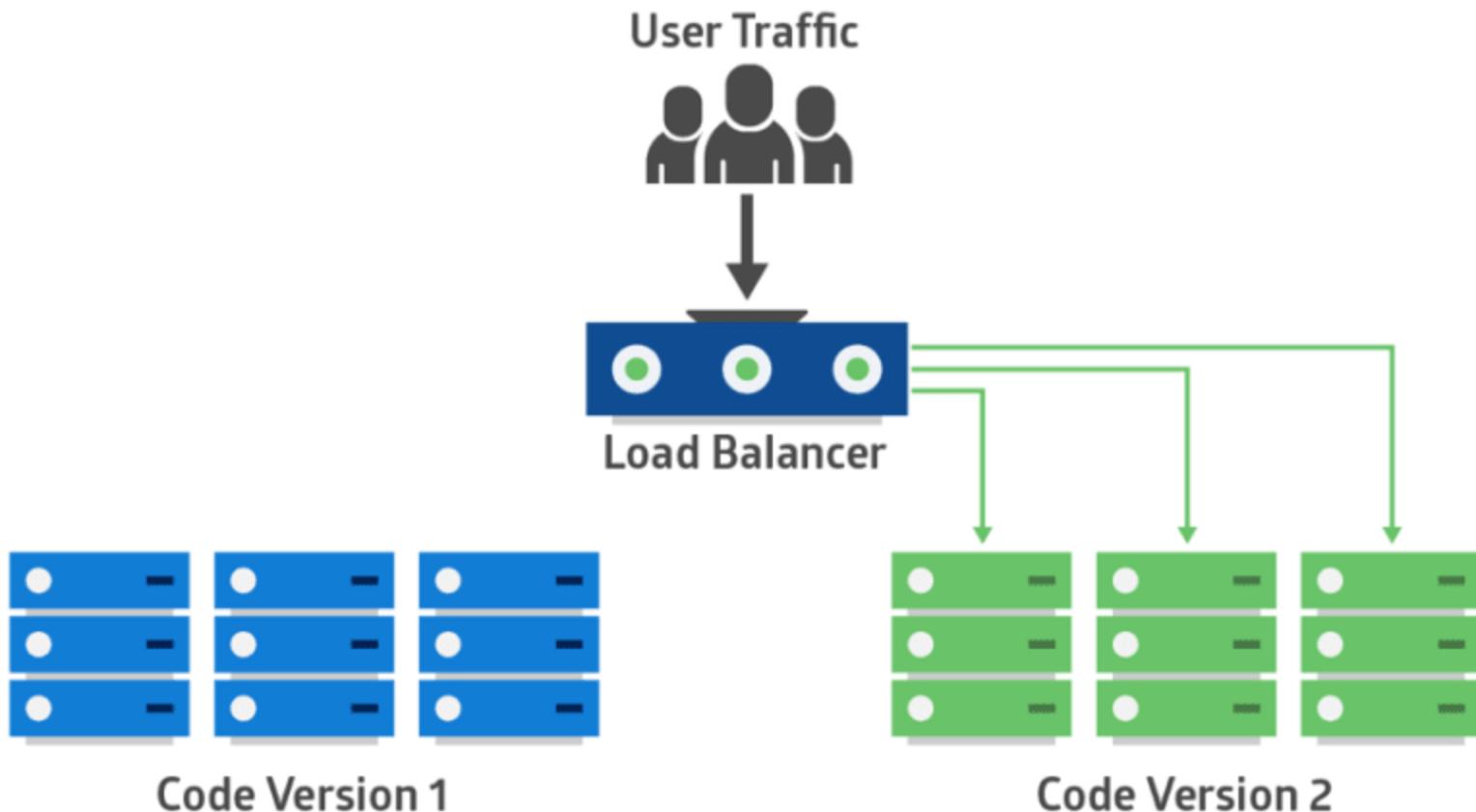


ESB or Microservice

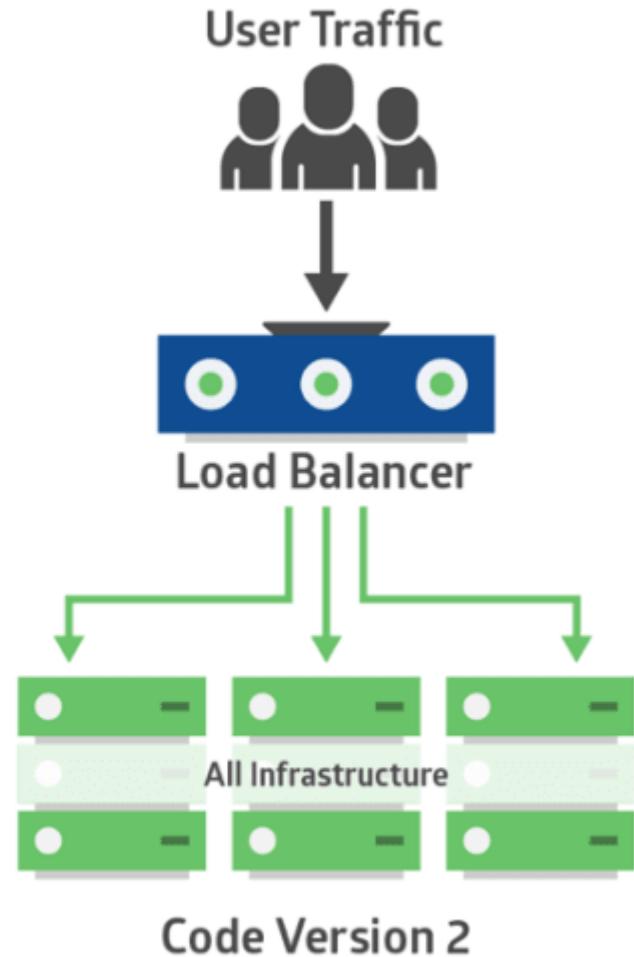
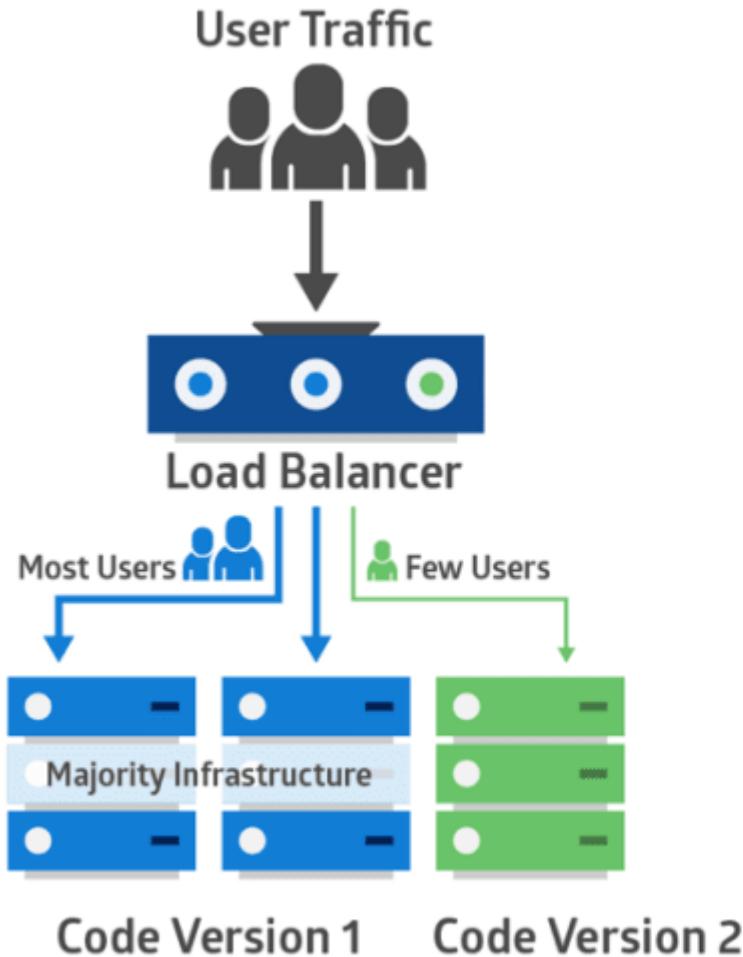
- When you select the suitable solution for the organization, following factors need to be assessed.
- Types of services to be integrated
- Communication protocols required
- Performance numbers
- Transaction requirements
- Security restrictions
- Inter-dependencies of services
- Business timelines
- Target expenses for the infrastructure
- Skill set of the team



Blue Green Deployment



Canary Testing



TWELVE-FACTOR APPS AND MICROSERVICES



- Modern architecture aims to develop large and complex applications in software as service (SaaS) models.
- The twelve-factor methodology provides guidelines for developing SaaS-based applications.
- Microservices, containers and their ecosystem fit well into the twelve-factor methodology.

TWELVE-FACTOR APPS AND MICROSERVICES



TABLE 1 SOLUTION COMPONENTS FOR 12 FACTOR APPS

FACTORS	BRIEF DETAILS <i>(Ref: https://www.12factor.net/)</i>	MICROSERVICE ECOSYSTEM COMPONENT
Codebase	One codebase tracked in revision control, many deploys	Source control systems such as gitlab or bitbucket can be leveraged for this. Source control systems provide in-built support for code revisions and version controls. Deployment can be done through build pipeline and CI/CD pipeline.
Dependencies	Explicitly declare and isolate dependencies	Build and packaging libraries such as Maven, Gradle and npm allow us to declare the dependent libraries along with the library version.
Config	Store config in the environment	Environment specific configurations (such as URLs, connection strings etc.) can be injected to the configuration files (such as application.properties in Spring application) in the build pipeline.
Backing services	Treat backing services as attached resources	Backing services such as storage, database or an external service should be accessible and managed through configuration files to ensure portability.
Build, release, run	Strictly separate build and run stages	Source code branches and automated CI/CD pipelines can be leveraged to manage environment specific releases.
Processes	Execute the app as one or more stateless processes	Stateless is the core tenets of microservices. Implementing token-based security helps us implement stateless authentication and authorization.

TWELVE-FACTOR APPS AND MICROSERVICES



TABLE 1 SOLUTION COMPONENTS FOR 12 FACTOR APPS

FACTORS	BRIEF DETAILS (Ref: https://www.12factor.net/)	MICROSERVICE ECOSYSTEM COMPONENT
Port binding	Export services via port binding	The services can be made visible through exposed ports. Container infrastructure provides configuration files (such as service.yaml in Docker) to bind the ports for services.
Concurrency	Scale out via the process model	By leveraging independent deployment feature of microservices, we can individually scale the most needed microservice by using on-demand scaling feature of containers.
Disposability	Maximize robustness with fast startup and graceful shutdown	Individual containers/pods can be started quickly. Container orchestrator can handle container shutdown gracefully.
Dev/prod parity	Keep development, staging, and production as similar as possible	We can achieve parity in environment dependencies, server dependencies, configurations through a container model.
Logs	Treat logs as event streams	Each microservice can log to standard output, which can be picked up by tools such as Kibana or Splunk to manage and visualize the logs centrally.
Admin processes	Run admin/management tasks as one-off processes	Container orchestration is managed by tools such as Kubernetes, while log management is carried out by tools such as Kibana or Splunk. Other application-specific administration can be deployed as a separate microservice.



Microservice Reference Architecture

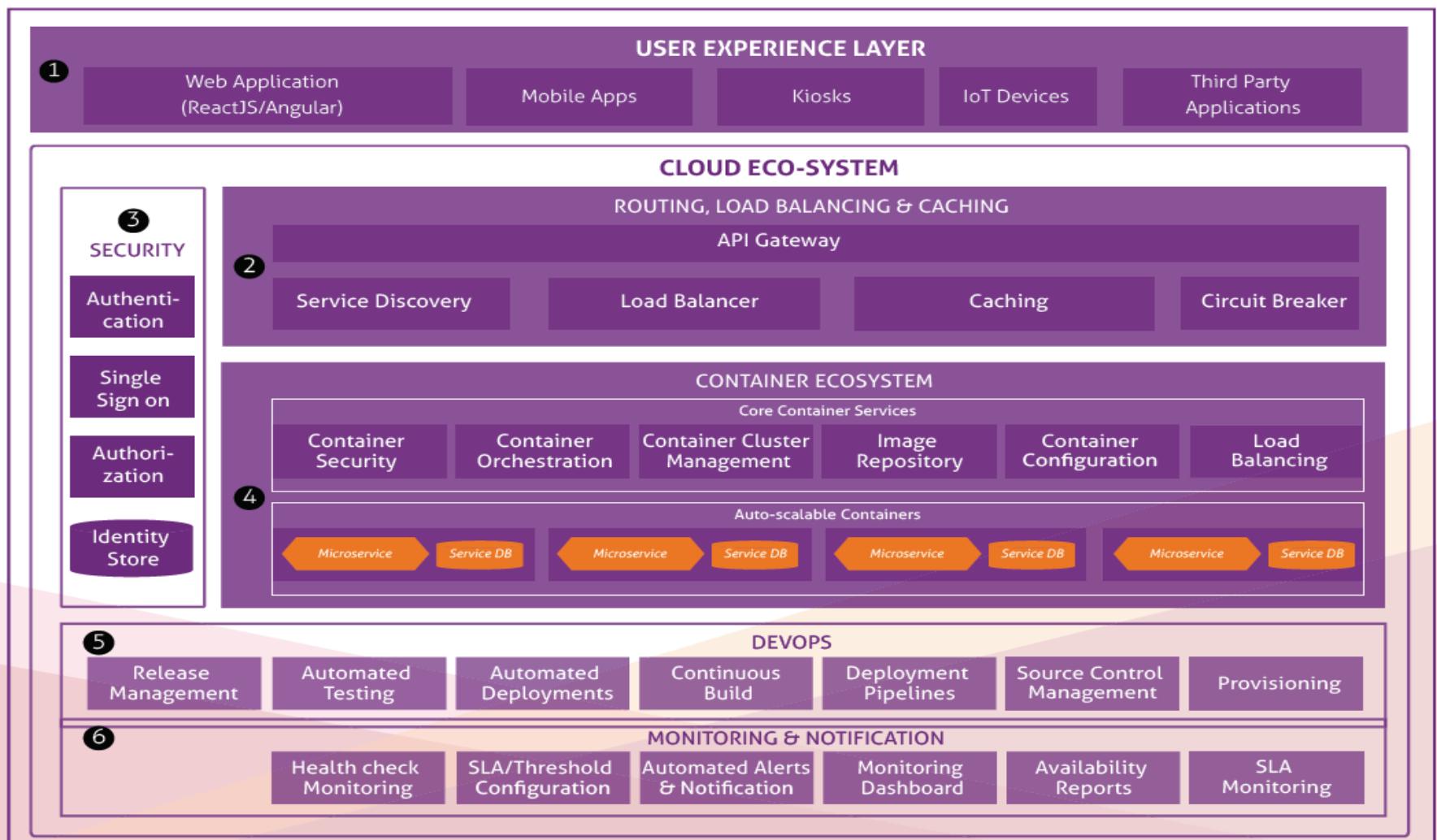


Figure 1: Microservices Reference Architecture



Sample Microservice Interaction

CLOUD NATIVE MICROSERVICE FRAMEWORK – SOLUTION COMPONENTS

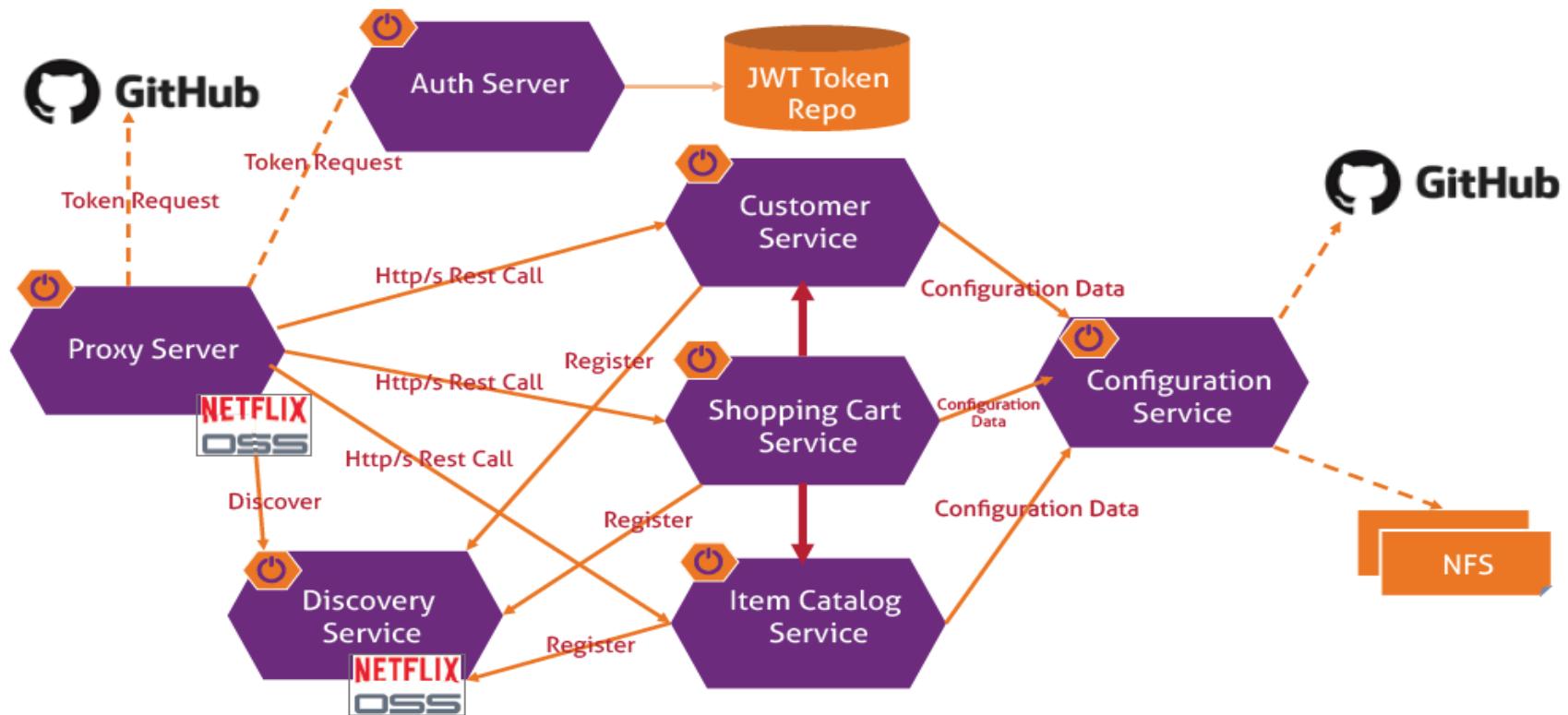


Figure 2: Sample Microservice Interaction Diagram



Sample Microservice Interaction

INFRA SERVICE	DETAILS
Auth Server	Custom Authorization server implementation which provides a configurable (in-memory, JWT) identity token. The token will be used to verify a user's authenticity every time the client tries to access the above business service
Proxy Server	We can leverage the Netflix OSS- Zuul service.
Service Discovery	We can leverage the Netflix OSS- Eureka service
Configuration Service	Custom cloud configuration service implementation. It provides configurable in-file or Git storage for service configuration.



Sample Microservice Interaction

ARCHITECTURE COMPONENT	SAMPLE PRODUCT STACK
Api Gateway	Netflix OSS- Zuul
Authentication Service	Spring – Security OAuth2, OpenID Connect
Service Discovery	Netflix OSS- Eureka, Apache Zookeeper
Configuration Service	Spring – Cloud Config Server
Microservice	Spring – Boot, Vert.x, Dropwizard
Monitoring	Netflix OSS- Turbine, Prometheus, Splunk, ELK (Elasticsearch, Logstash, Kibana), CAdvisor Visualization – Grafana, Kibana
Circuit Breaker	Netflix OSS- Hystrix
Microservices Testing	Wiremock
Container Ecosystem	Docker – Container technology Docker Swarm, Kubernetes – Container orchestrator

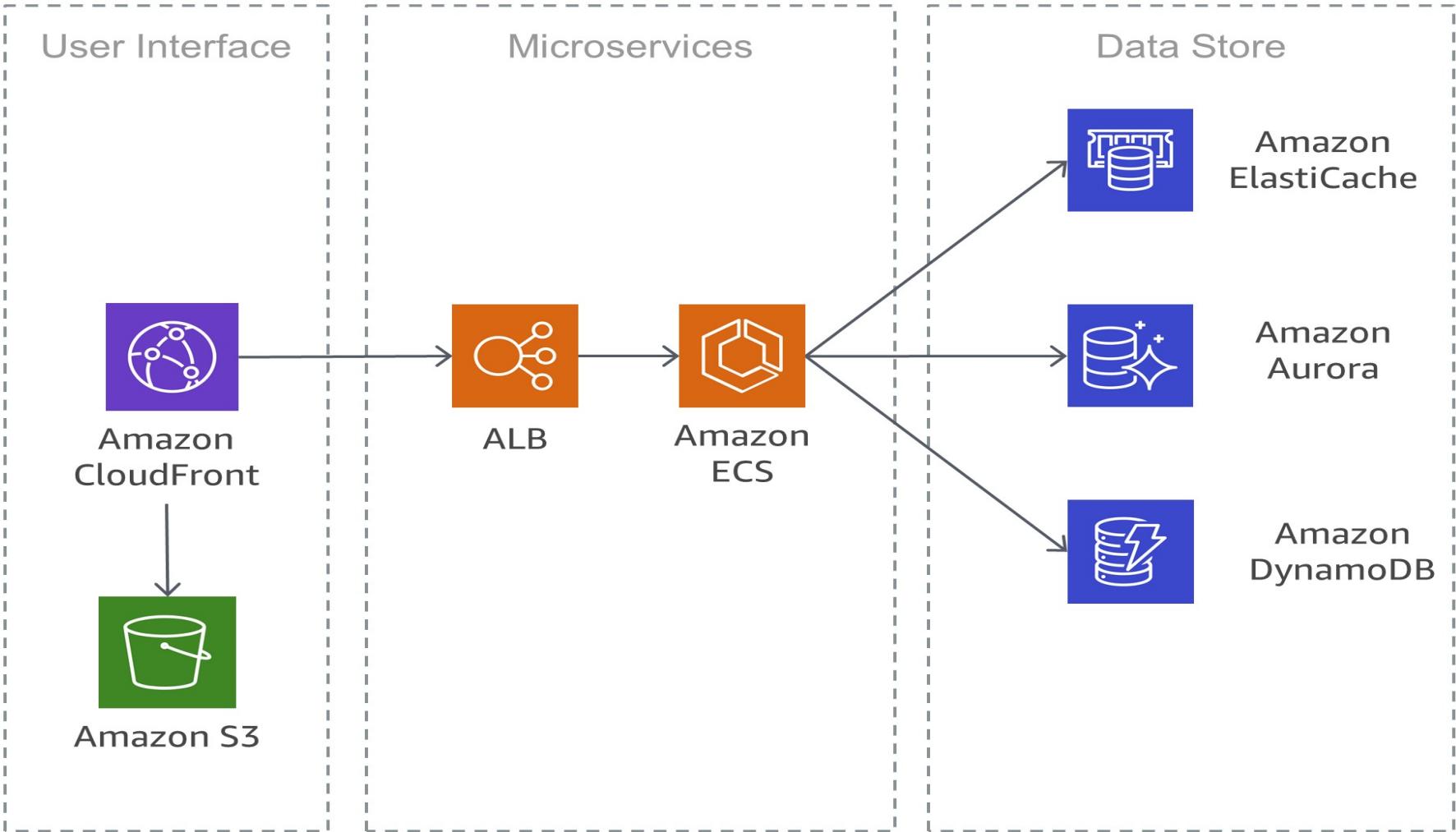


Cloud Deployment Architecture - AWS

Cloudfront CDN: Cloudfront native CDN allows to effectively deliver the content for the consumer from different geographies.

- **S3 Bucket:** ReactJS/Angular-based web applications will be deployed on S3 bucket & it is integrated with AWS Cloud front. The frontend is secured with WAF (Web Application Firewall). Deploying frontend static content on S3 makes it highly scalable & cost effective.
- **Custom Services Layer:** Backend microservices are developed using NodeJS/ Spring Boot and are deployed on containers using AWS Cloud-native container orchestration services ECS. These services are accessed through AWS API Gateway.

Simple Microservices Architecture on AWS





Cloud Deployment Architecture - AWS

AWS-BASED MICROSERVICES DEPLOYMENT ARCHITECTURE

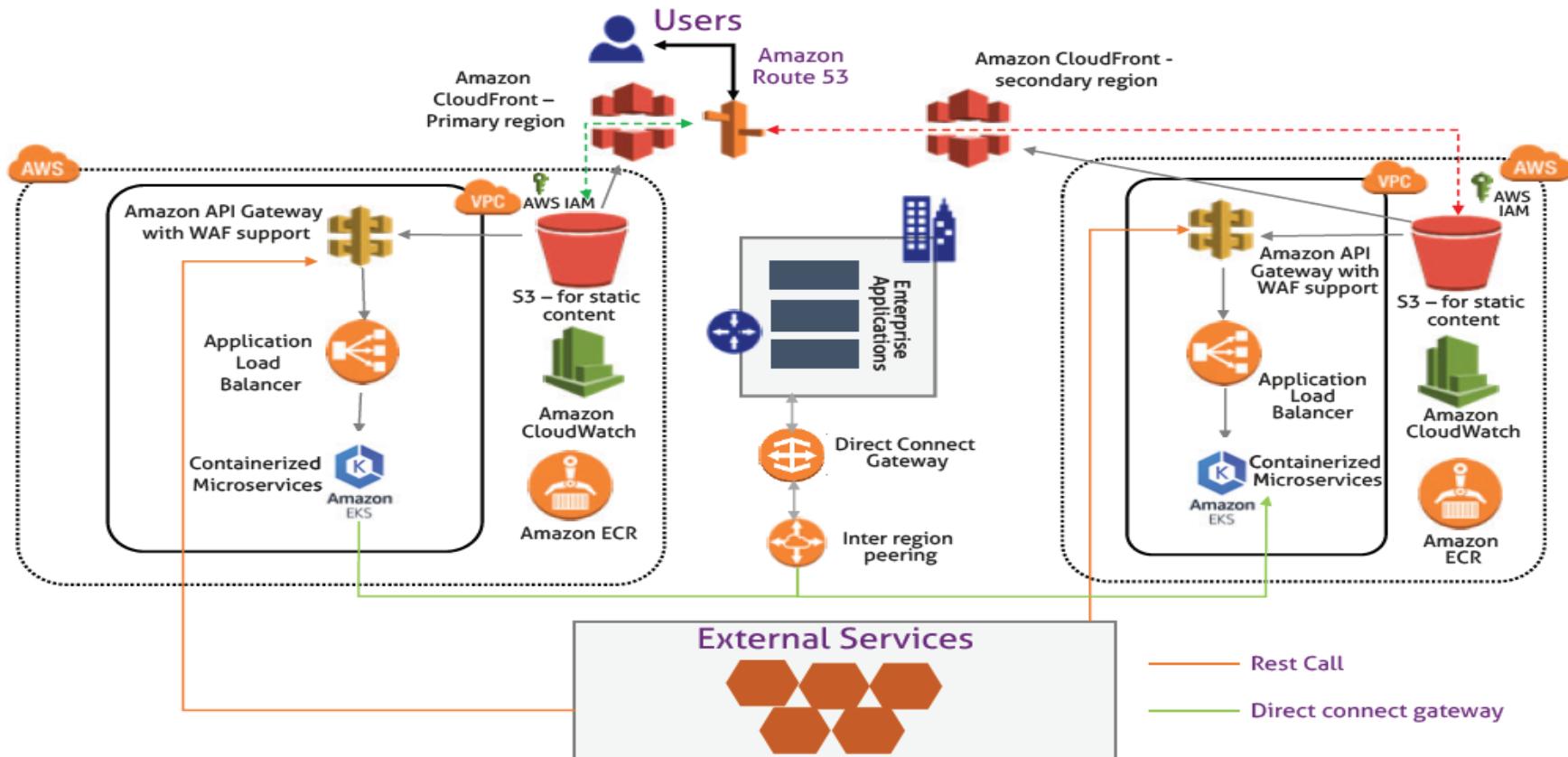


Figure 3: Sample AWS-based Microservices Deployment Architecture

Cloud Deployment Architecture - Azure



AZURE-BASED MICROSERVICES DEPLOYMENT ARCHITECTURE

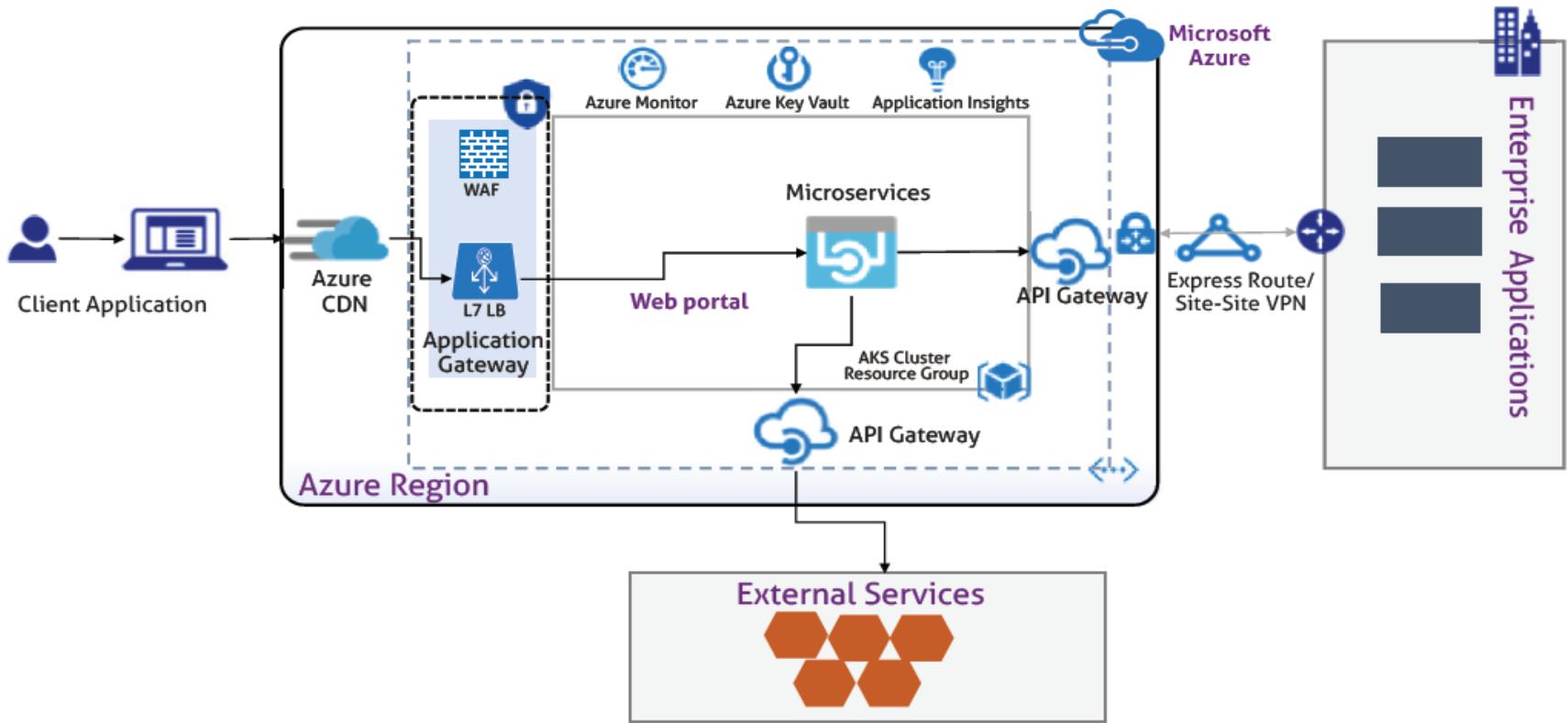


Figure 4: Sample Azure-based Microservices Deployment Architecture



Microservices Features

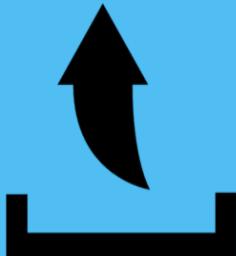




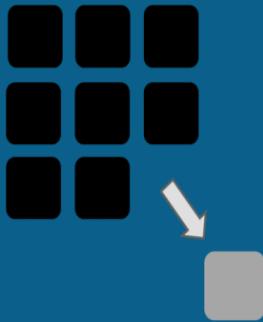
Advantages Of Microservices



Independent Development



Independent Development



Independent Deployment

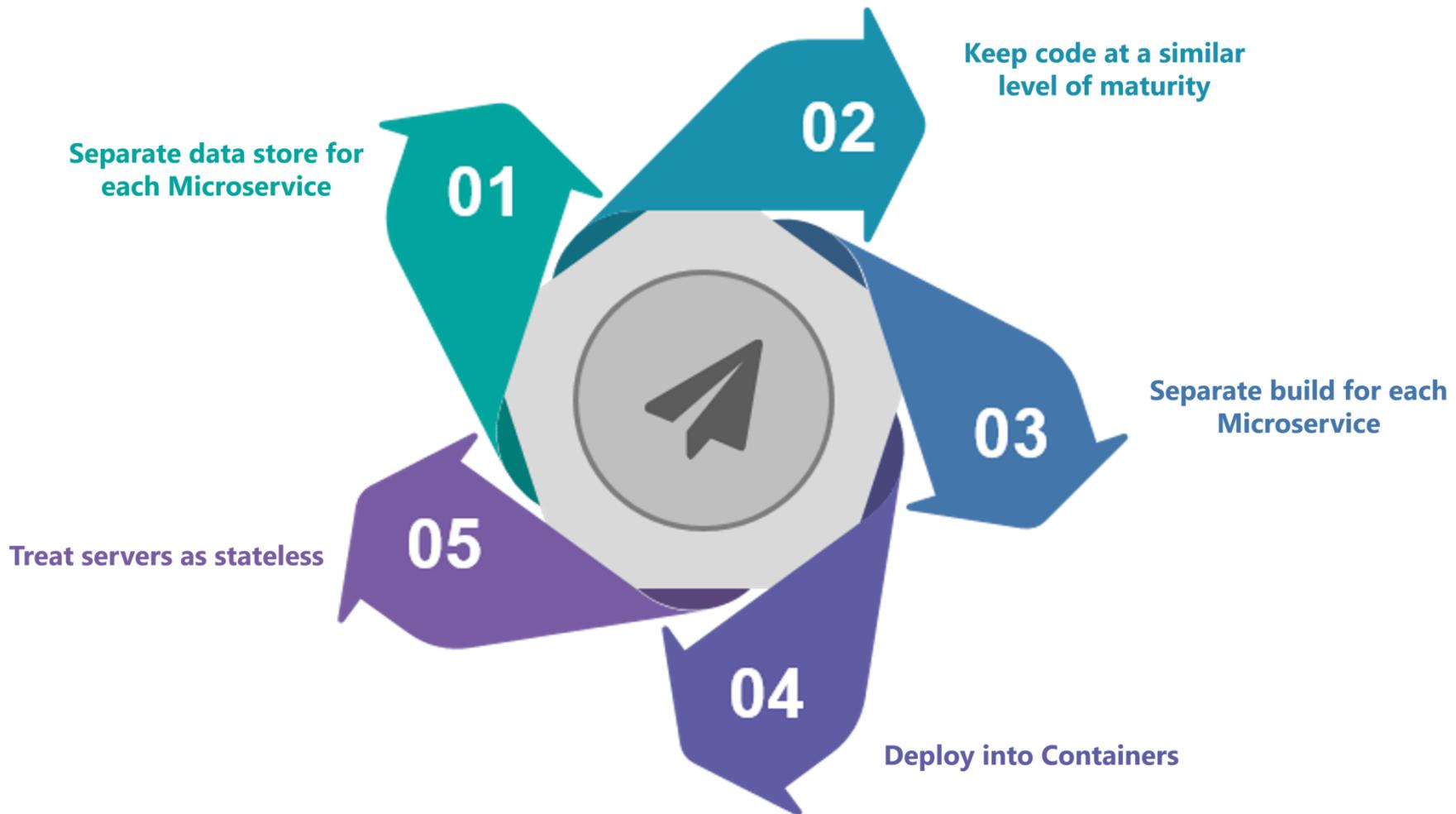
Fault Isolation

Mixed Technology Stack



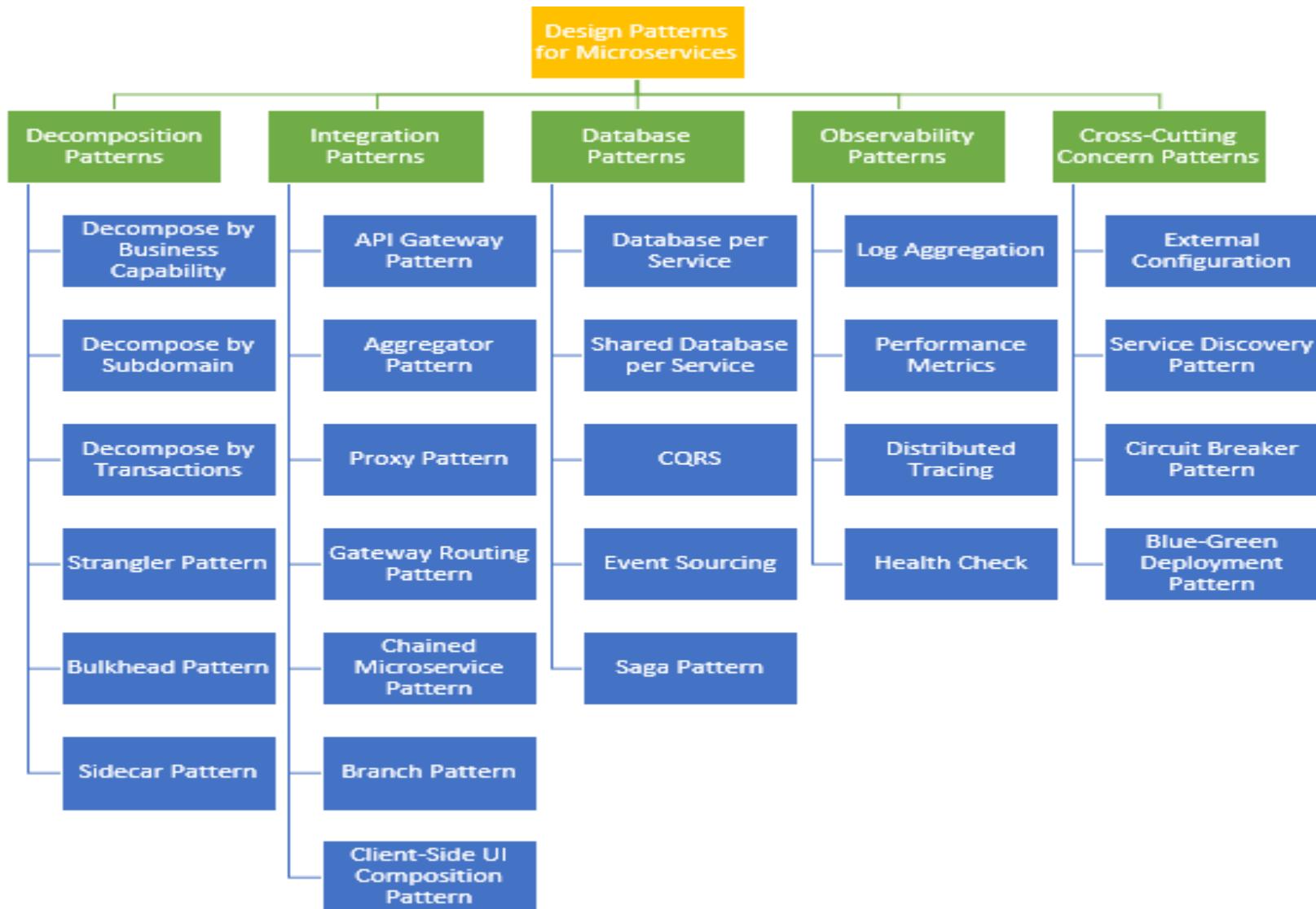
Granular Scaling

Best Practices To Design Microservices





Design Patterns of Micro services



Decomposition Patterns

Decompose by Business Capability



- Problem
- Microservices is all about making services loosely coupled, applying the single responsibility principle.
- However, breaking an application into smaller pieces has to be done logically.
- How do we decompose an application into small services?

Decompose by Business Capability



- **Solution**
- One strategy is to decompose by business capability.
- A business capability is something that a business does in order to generate value.
- The set of capabilities for a given business depend on the type of business.
- For example, the capabilities of an insurance company typically include sales, marketing, underwriting, claims processing, billing, compliance, etc.
- Each business capability can be thought of as a service, except it's business-oriented rather than technical.



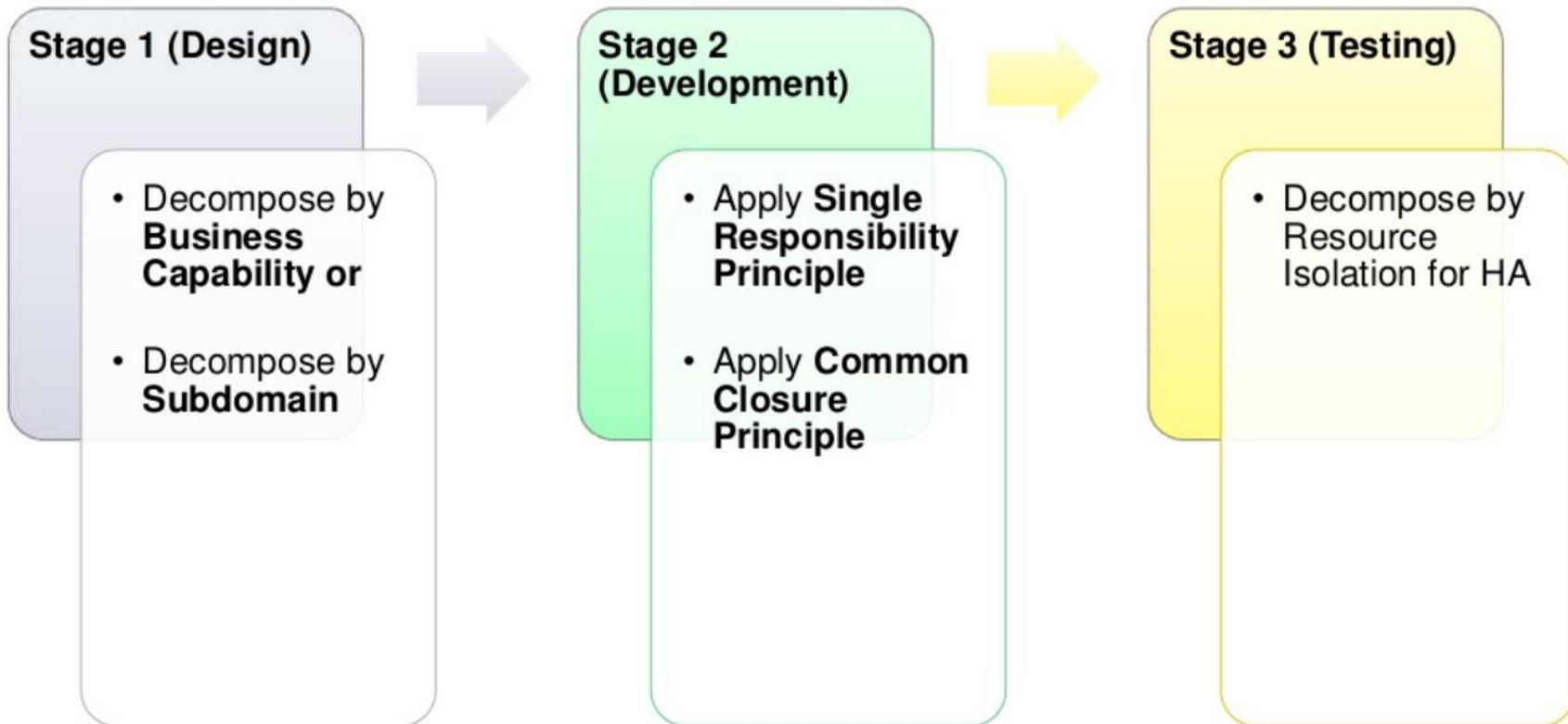
Decompose by Business Capability

- Microservice is all about making services loosely coupled, applying the single responsibility principle.
- It decomposes by business capability.
- Define services corresponding to business capabilities.
- A business capability is a concept from business architecture modeling.
- A business capability often corresponds to a business object, e.g.
- Order Management is responsible for orders
- Customer Management is responsible for customers.
- For instance, in an e-commerce solution, the main business capabilities are order management, product promotions, service management and others. We can create microservices based on these.



Microservice Decomposition Strategy

The process to define Microservice decomposition.





Decompose by Subdomain

- Decomposing an application using business capabilities might be a good start, but there will be so-called “God Classes” which will not be easy to decompose.
- These classes will be common among multiple services.
- A domain consists of multiple subdomains.
- Each subdomain corresponds to a different part of the business.



Decompose by Subdomain

- Subdomains can be classified as follows:
- Core — key differentiator for the business and the most valuable part of the application
- Supporting — related to what the business does but not a differentiator. These can be implemented in-house or outsourced
- Generic — not specific to the business and are ideally implemented using off the shelf software



Decompose by Subdomain

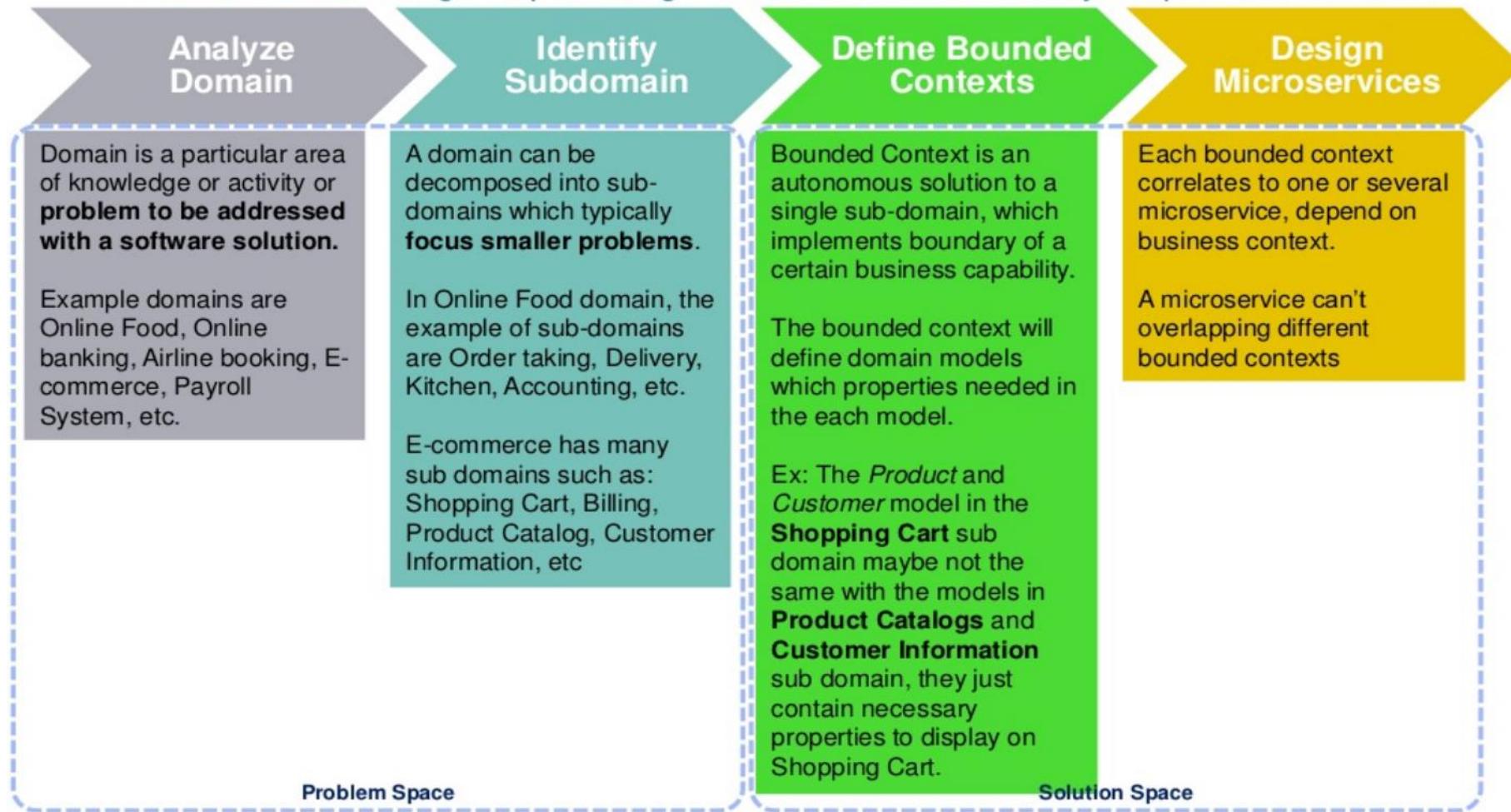
- The subdomains of an Order management include:
 - Product catalog service
 - Inventory management services
 - Order management services
 - Delivery management services

Decompose by Subdomain



Decompose by Subdomain Pattern

Domain-driven design help to design Microservices that loosely coupled and cohesive.

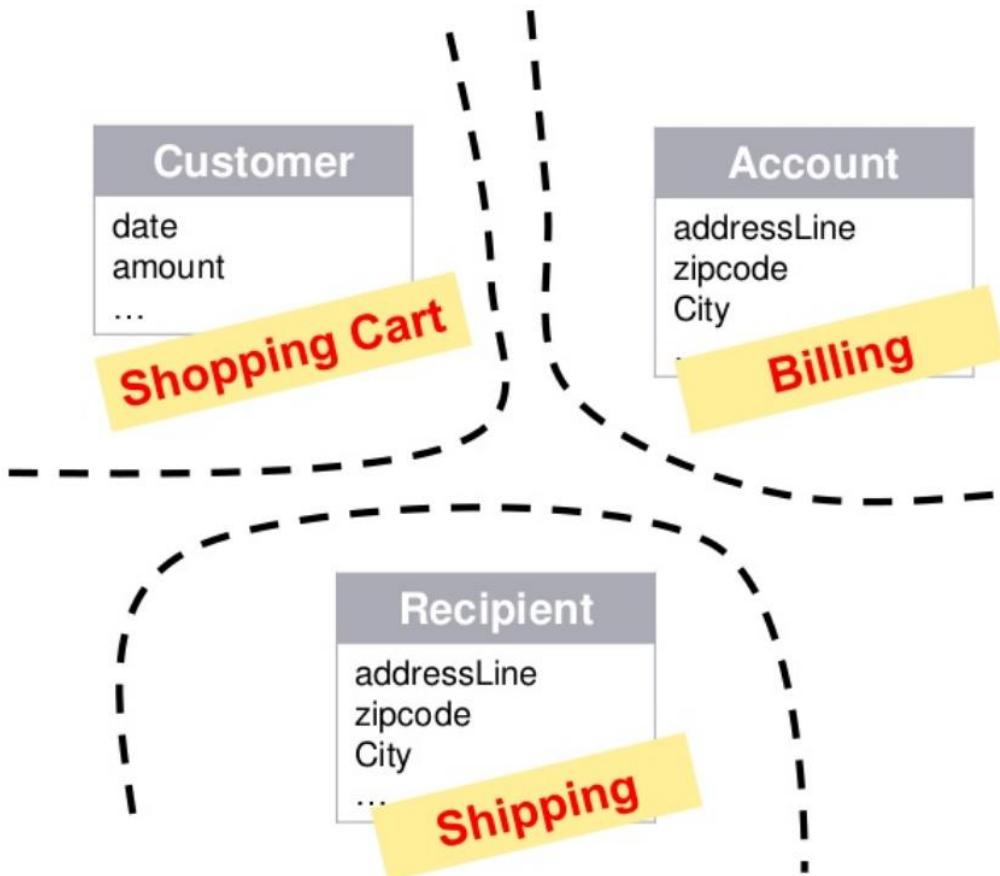


Decompose by Subdomain



If We Using Bounded Context

Split by business domains or business capabilities



- Achieve Autonomous Team
- Avoid distributed monolith application
- Avoid cascading failure
- Simplify the complexity of the business logic
- Achieve Business and IT alignment

But

- Create partial duplication of data
- How to correlate or integrate the data?
- How to create data consistency and data integrity?
- How to create effective and efficient query?

Decompose by Subdomain



2

DDD: Bounded Context – Strategic Design



An App User's Journey can run across multiple Bounded Context / Micro Services.



Areas of the domain treated independently

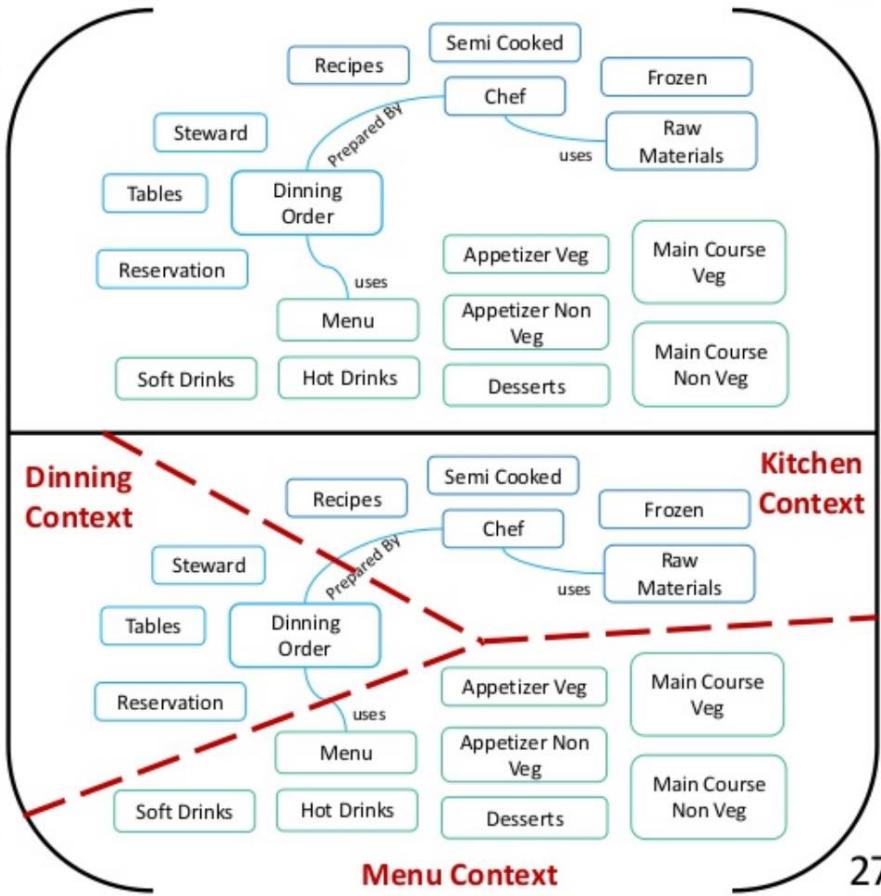
Discovered as you assess requirements and build language

8/10/2018

Source: Domain-Driven Design
Reference by Eric Evans



Understanding Bounded Context (DDD) of a Restaurant App



27

78

Decompose by Transactions / Two-phase commit (2pc) pattern



- Identify the main transactions of the application and develop microservices for them.
- For instance, the main transactions of an e-commerce application are login, checkout, search and such; We can create microservices for these transactions.

Decompose by Transactions / Two-phase commit (2pc) pattern



- You can decompose services over the transactions.
- Then there will be multiple transactions in the system.
- The distributed transaction consists of two steps:
- Prepare phase — during this phase, all participants of the transaction prepare for commit and notify the coordinator that they are ready to complete the transaction
- Commit or Rollback phase — during this phase, either a commit or a rollback command is issued by the transaction coordinator to all participants

Decompose by Transactions / Two-phase commit (2pc) pattern



- The problem with 2PC is that it is quite slow compared to the time for operation of a single microservice.
- Coordinating the transaction between microservices, even if they are on the same network, can really slow the system down, so this approach isn't usually used in a high load scenario.



Decomposition based on resources

- We can create microservices based on nouns or resources and define the operations.
- For instance, in an e-commerce solution, ‘products’ is a resource and we can define the list all products
- (GET /products), query particular product (GET /product/{1}), insert product (PUT /product/{}).



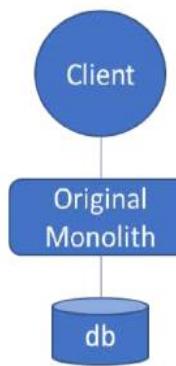
Strangler Pattern

- This creates two separate applications that live side by side in the same URI space.
- **Over time, the newly refactored application “strangles” or replaces the original application and shuts off monolithic application.**
- Application steps are transform, coexist, and eliminate:
- Transform — Create a parallel new site with modern approaches.
- Coexist — Leave the existing site where it is for a time. Redirect from the existing site to the new one so the functionality is implemented incrementally.
- Eliminate — Remove the old functionality from the existing site.

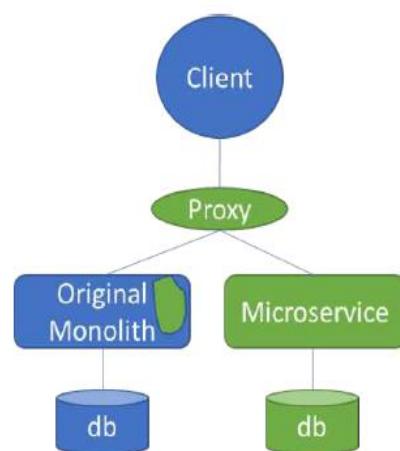
Strangler Pattern



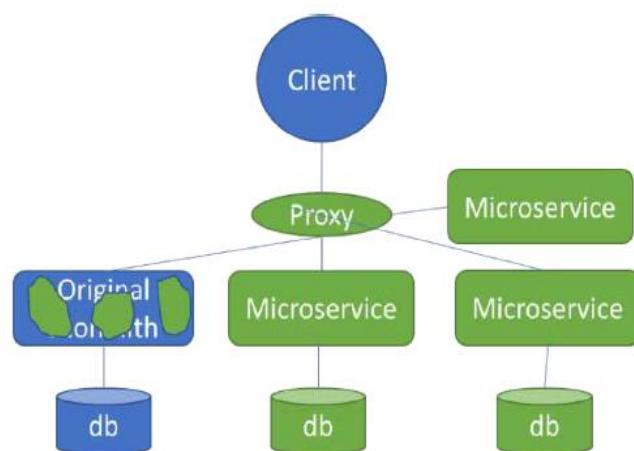
Transform – Create a parallel microservice



Co-exist – Incrementally redirect the traffic from the legacy to microservice



Eliminate – eliminate the legacy module





Bulkhead Pattern

- Isolate elements of an application into pools so that if one fails, the others will continue to function.
- This pattern is named Bulkhead because it resembles the sectioned partitions of a ship's hull.
- Partition service instances into different groups, based on consumer load and availability requirements.
- This design helps to isolate failures, and gives sustain service functionality for some consumers, even during a failure.



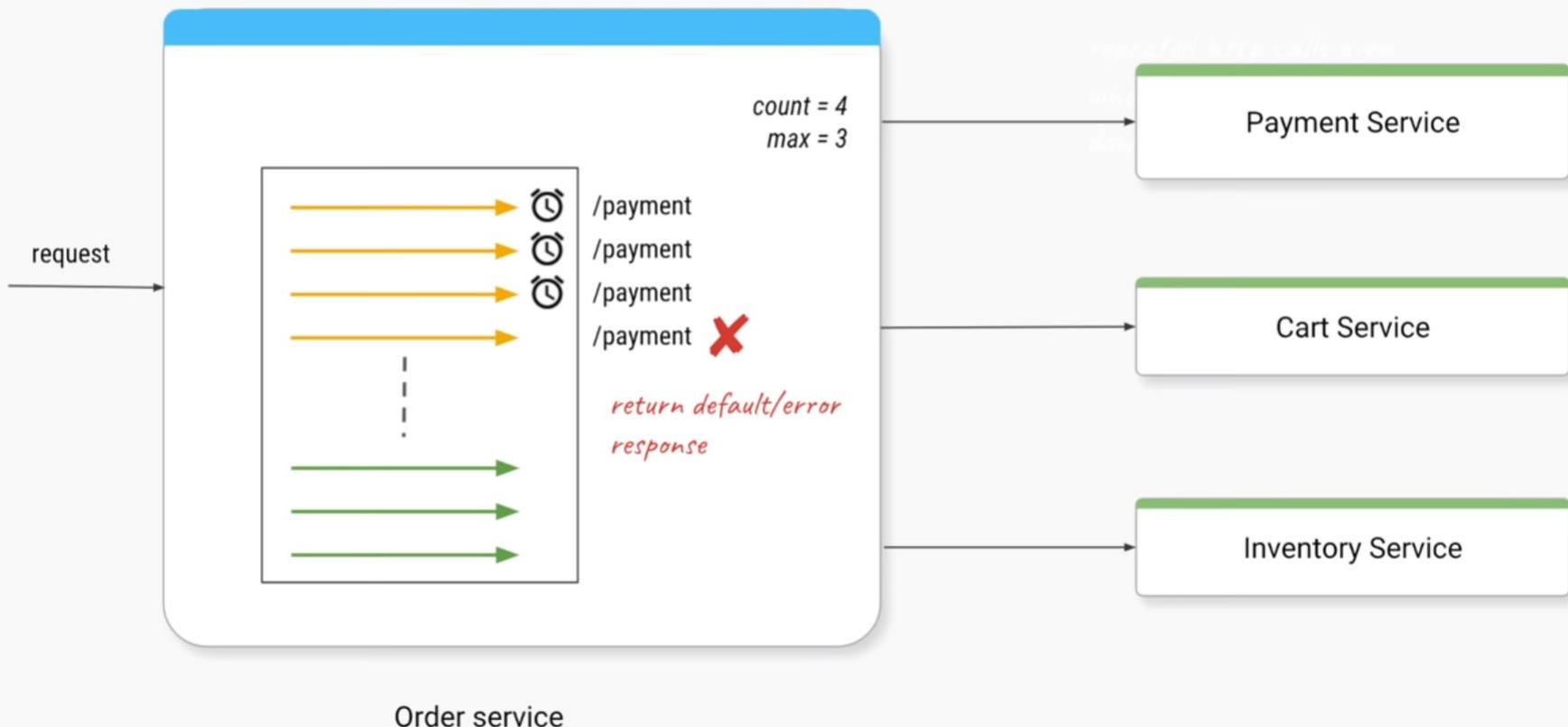
Bulkhead Pattern

- Bulkhead is a term borrowed from cargo ships.
- In a cargo ship, the bulkhead is a wall built between different cargo sections, which makes sure that a fire or flood in one section is restricted to that section and other sections are not impacted.
- Failure in one service or a group of services should not bring down the whole application.
- To implement the bulkhead pattern, we need to make sure that all our services work independently of each other and failure in one will not create a failure in another service.
- Techniques such as maintaining a single-responsibility pattern, an asynchronous-communication pattern, or fail-fast and failure-handling patterns help us to achieve ...



Bulkhead

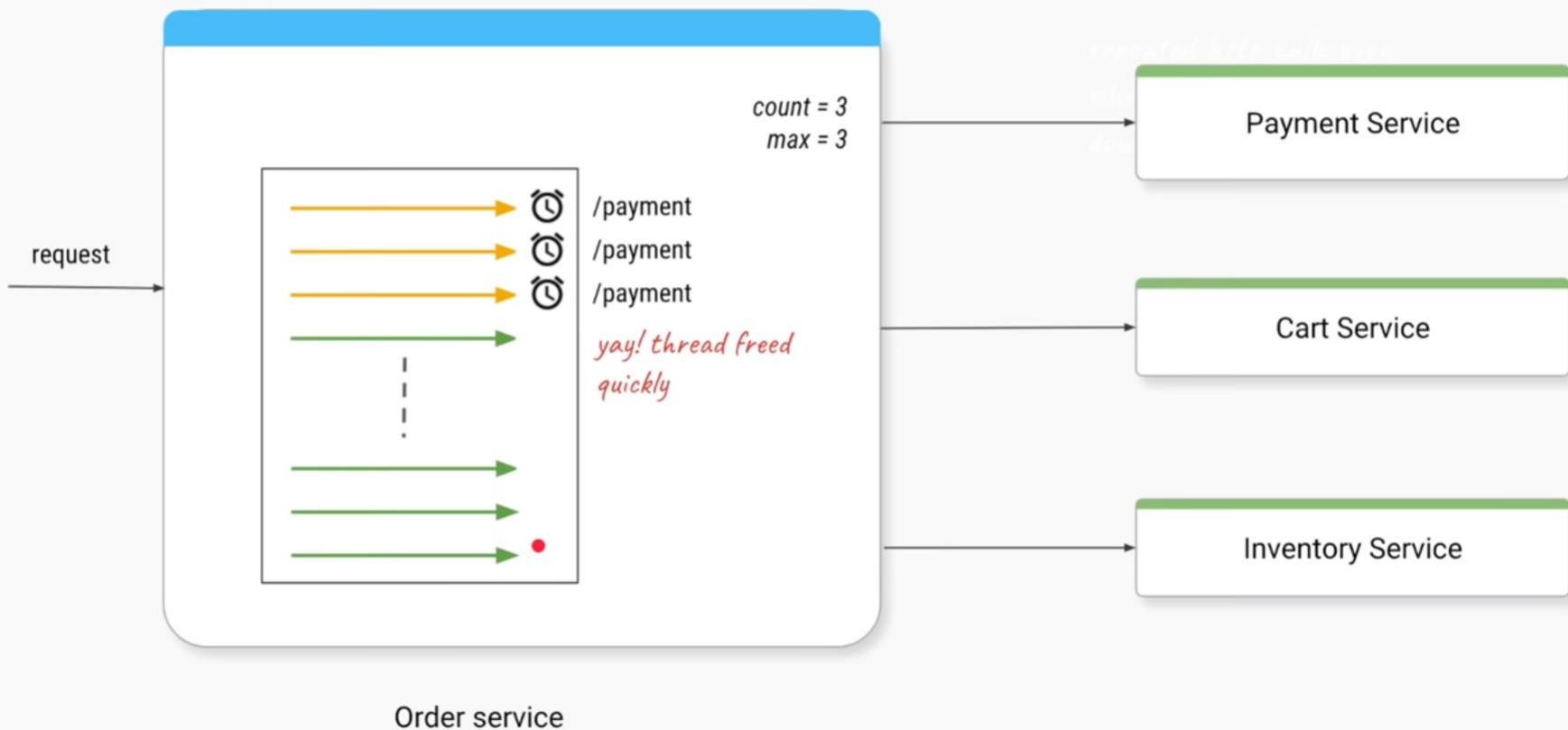
Return error/default-response for any new requests





Bulkhead

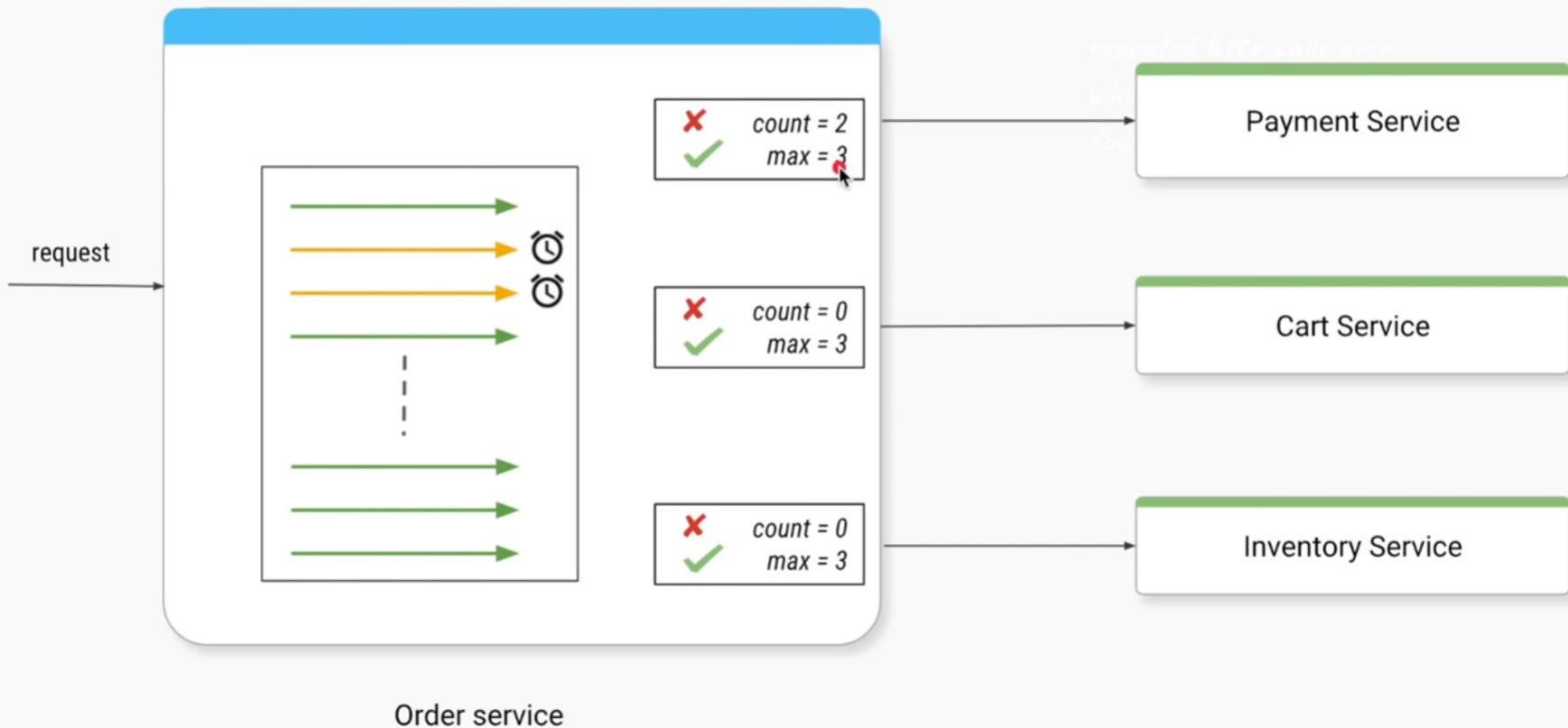
Load shedding / Fail-fast (helps avoid threadpool exhaustion)





Bulkhead

Bulkhead pattern





Bulkhead

Sample skeleton code

```
// current requests for payment service          --- not thread-safe ---
int count = 0;

@PostMapping("/pay")
public ResponseEntity<Payment> makePayment() {

    if (count <= THRESHOLD) {
        count++;
        Payment response = paymentService.makePayment();
        count--;
        return response;
    } else {
        return "default-response"; ←———— fail-fast
    }
}
```

(thread freed quickly)



Bulkhead

Code (resilience4j)

```
// Create once
BulkheadConfig config = BulkheadConfig.custom()
    .maxConcurrentCalls(150)
    .maxWaitTime(100)
    .build();
```

}] custom config

```
Bulkhead bulkheadForPayment = Bulkhead.of("payments", config);

// decorate call with bulkhead
Supplier<String> pay = Bulkhead.decorateSupplier(bulkheadForPayment, this::makePayment);

// execute the call
Try<String> result = Try.ofSupplier(pay);
if (result.isSuccess()) {
    return result.get();
} else { ← fail-fast
    return "default-response";
}
```

← decorate call to remote service

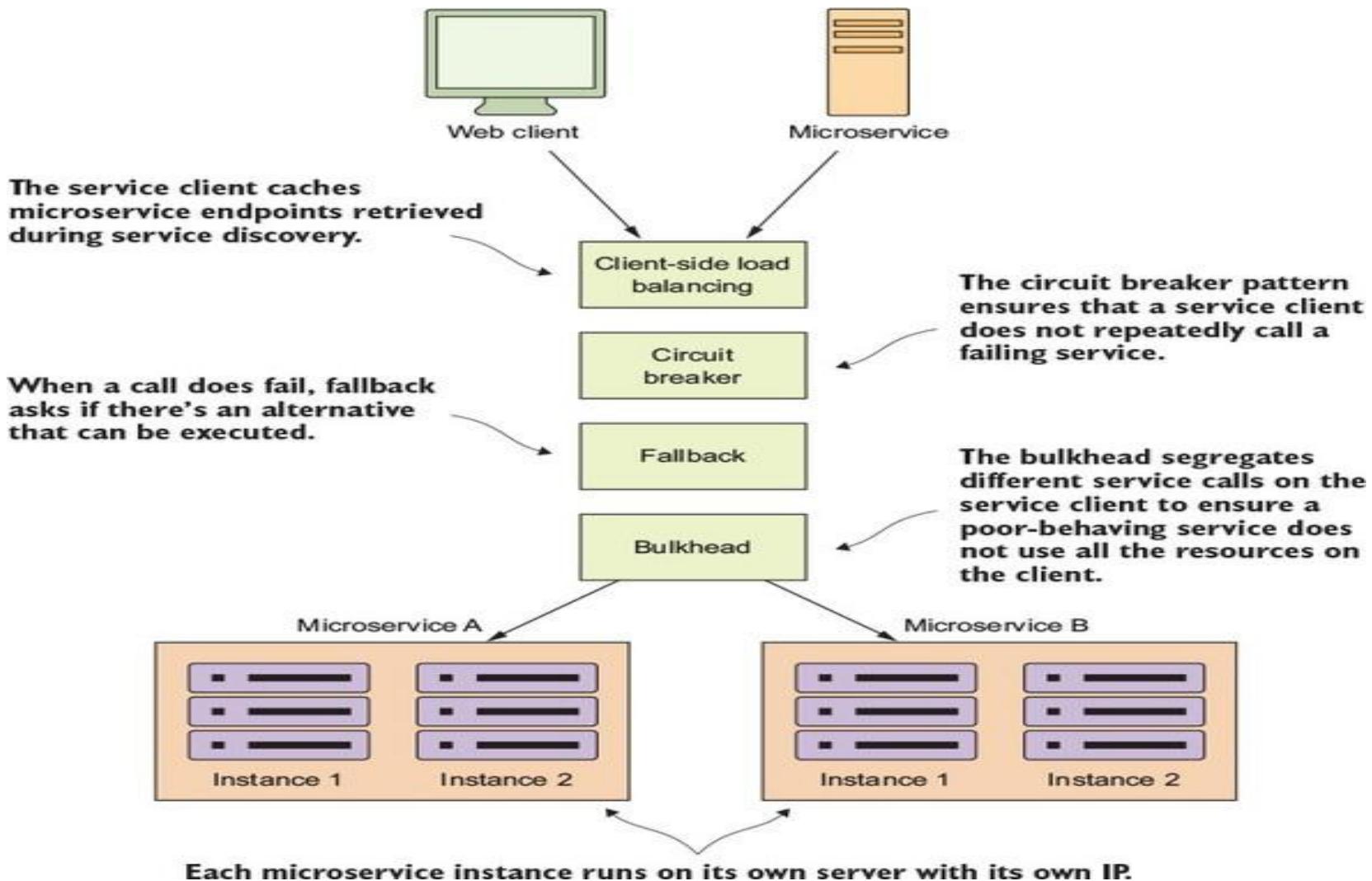


Bulkhead

```
33     //select where by customerId
34     @Bulkhead(name = "getCustomerByIdService", fallbackMethod = "getFallbackgetCustomerById",
35                 type = Type.SEMAPHORE)
36     // maximum 5 concurrent requests allowed
37     //@Bulkhead(5)
38     public Customer getCustomerById(long customerId)
39     {
40         /*
41          * try { Thread.sleep(3000); } catch (InterruptedException e) { // TODO
42          * Auto-generated catch block e.printStackTrace(); }
43          */
44         return this.customerRepo.findById(customerId).orElse(null);
45     }
46     public Customer getFallbackgetCustomerById(long customerId, Exception e) {
47         System.out.println("Falling back : " +customerId);
48         return new Customer();
49     }
50 }
```

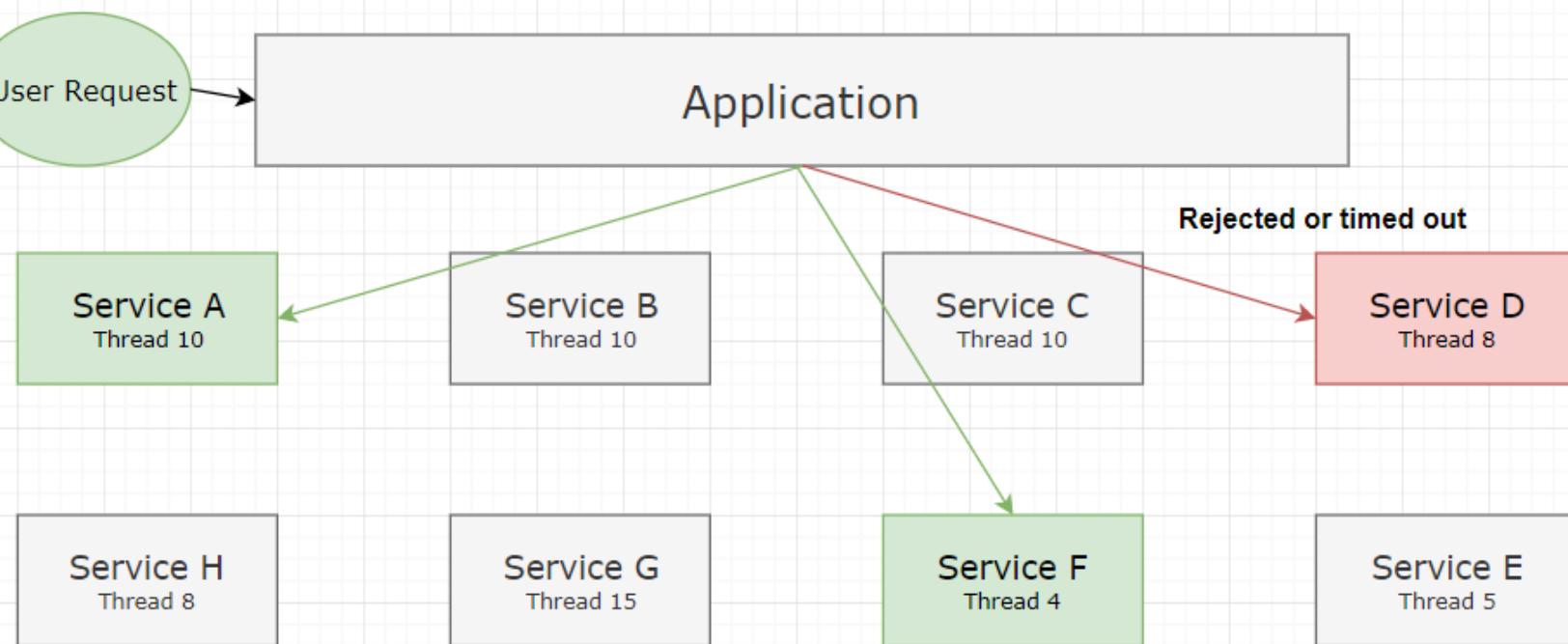


Bulkhead Pattern



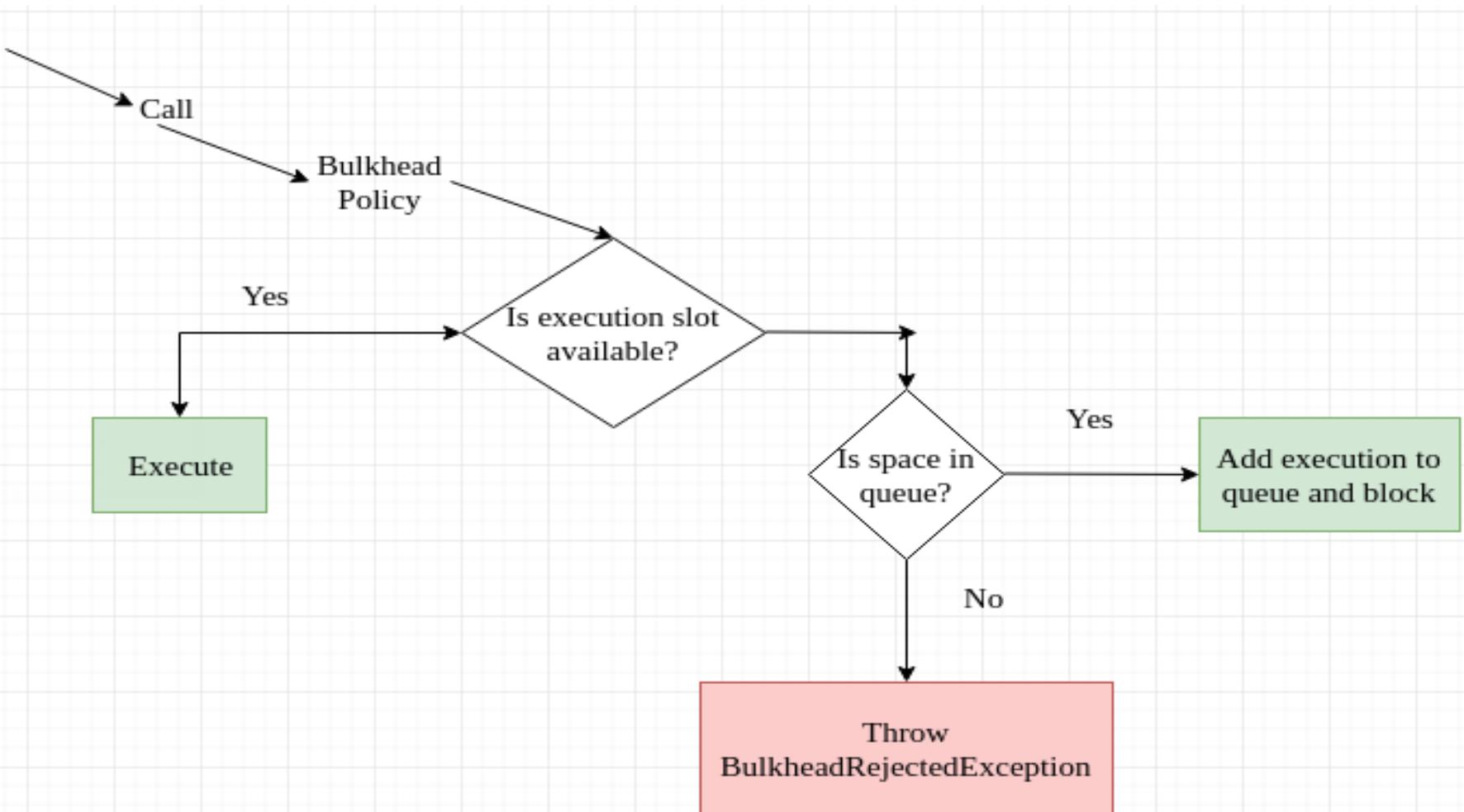


Bulkhead Pattern





Bulkhead Pattern





Sidecar Pattern

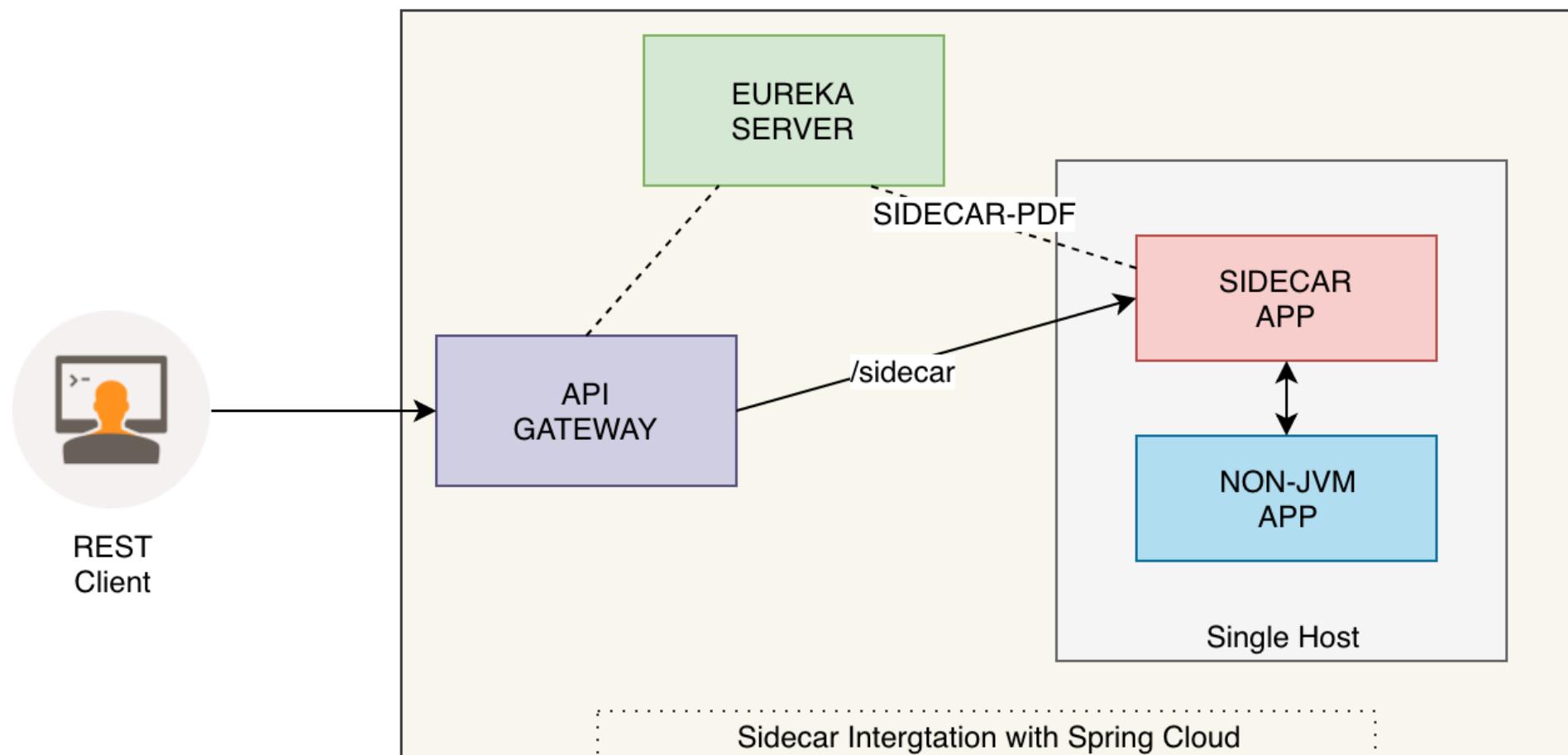
- Deploy components of an application into a separate processor container to provide isolation and encapsulation.
- This pattern can also enable applications to be composed of heterogeneous components and technologies.
- This pattern is named Sidecar because it resembles a sidecar attached to a motorcycle.
- In the pattern, the sidecar is attached to a parent application and provides supporting features for the application.
- The sidecar also shares the same lifecycle as the parent application, is created and retired alongside the parent.
- The sidecar pattern is sometimes referred to as the sidekick pattern and is the last decomposition pattern.



Sidecar Pattern

- The problem statement
- There are multiple problems related to inter service communication when polyglot comes into picture. few of them are:
- How does non-JVM apps discover JVM microservices in Spring Cloud environment and vice-versa.
- In any cloud native application, we would never want to hard code host and port of apps for discovery purpose.
- How does non-JVM app communicates (i.e. calls REST endpoints) with JVM apps and vice versa.
- How do we utilize the client side load balancing features for non-JVM apps.
- How do we utilize the config server for non-JVM apps?

Sidecar for the Rescue



Integration Patterns



API Gateway Pattern

- An API gateway provides a centralized access point for invoking a microservice.
- The API gateway handles security (such as authentication, authorization), governance (such as logging service, monitoring service), request routing, protocol transformation, data transformation and the aggregation of responses from multiple services.

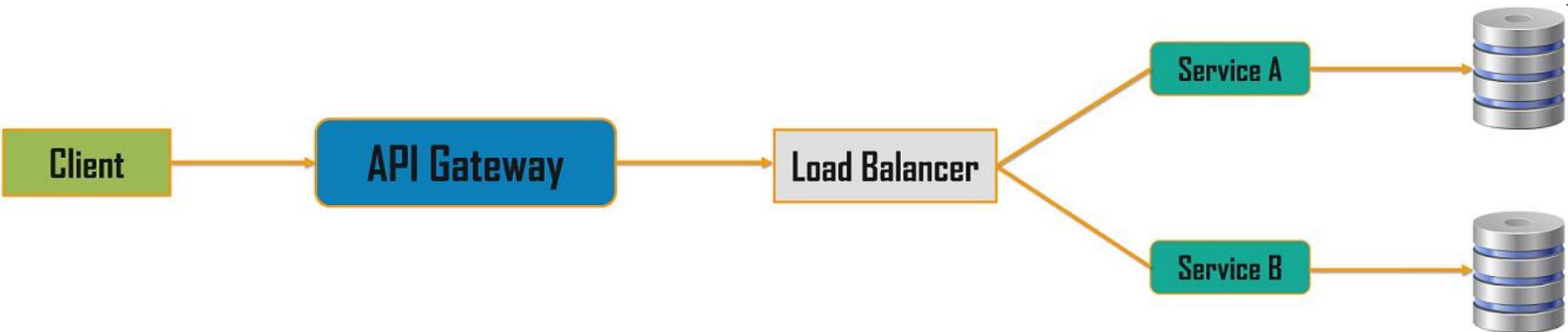


API Gateway Pattern

- An API Gateway helps to address many concerns raised by the microservice implementation.
- An API Gateway is the single point of entry for any microservice call.
- It can work as a proxy service to route a request to the concerned microservice.
- It can aggregate the results to send back to the consumer.
- This solution can create a fine-grained API for each specific type of client.
- It can also convert the protocol request and respond.
- It can also offload the authentication/authorization responsibility of the microservice.



API Gateway Design Pattern





Aggregator Pattern

- When breaking the business functionality into several smaller logical pieces of code, it becomes necessary to think about how to collaborate the data returned by each service.
- This responsibility cannot be left with the consumer.
- The Aggregator pattern helps to address this.
- It talks about how we can aggregate the data from different services and then send the final response to the consumer.

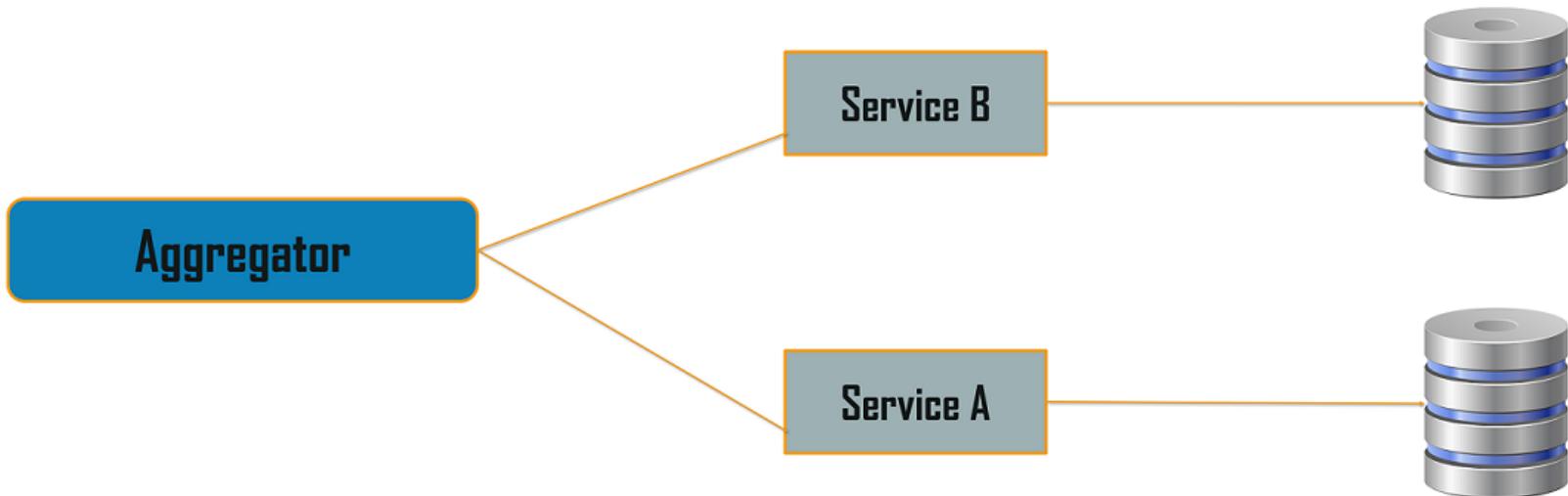


Aggregator Pattern

- A composite microservice will make calls to all the required microservices, consolidate the data, and transform the data before sending back.
- An API Gateway can also partition the request to multiple microservices and aggregate the data before sending it to the consumer.
- When a single microservice needs responses from multiple microservices, a composite service can take the responsibility of aggregating the response.

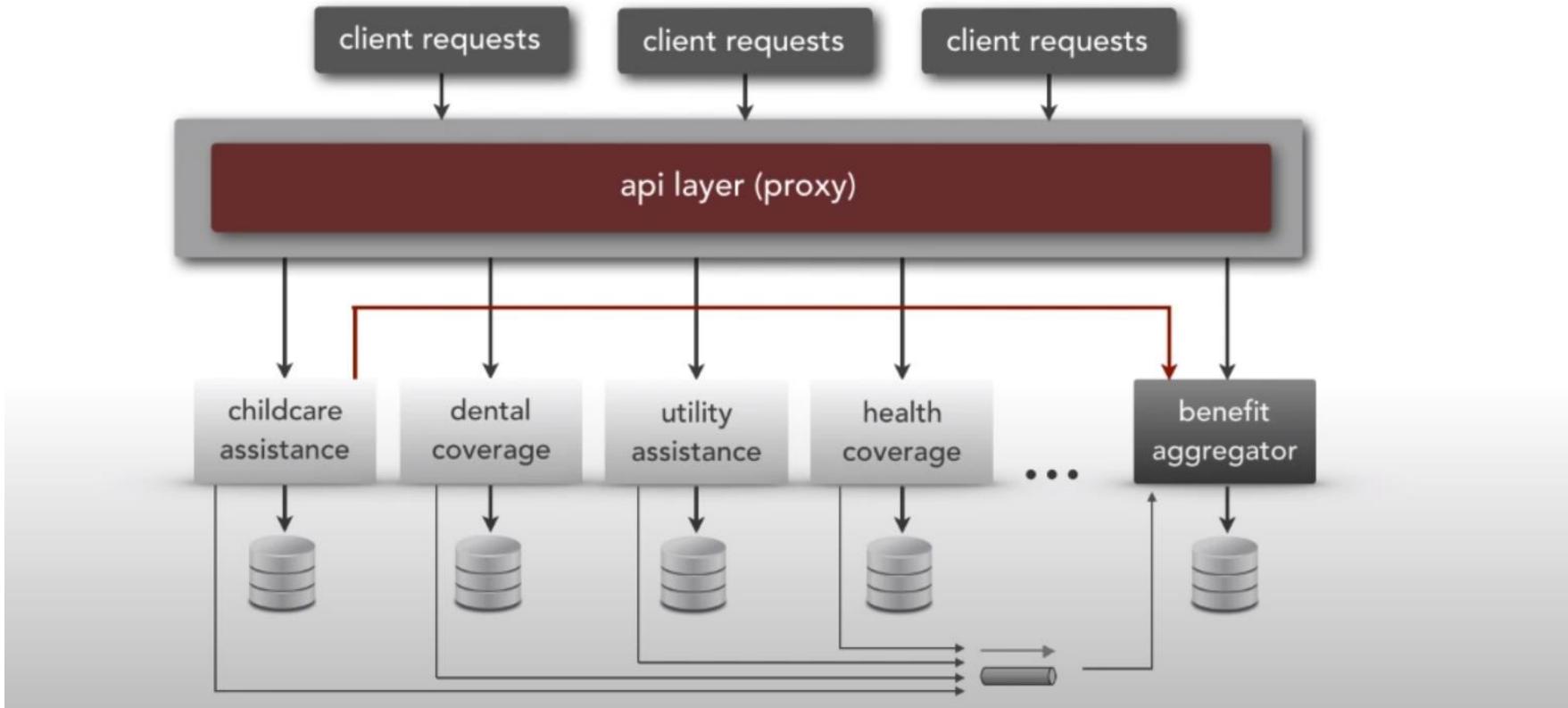


Aggregator Pattern





microservice aggregator

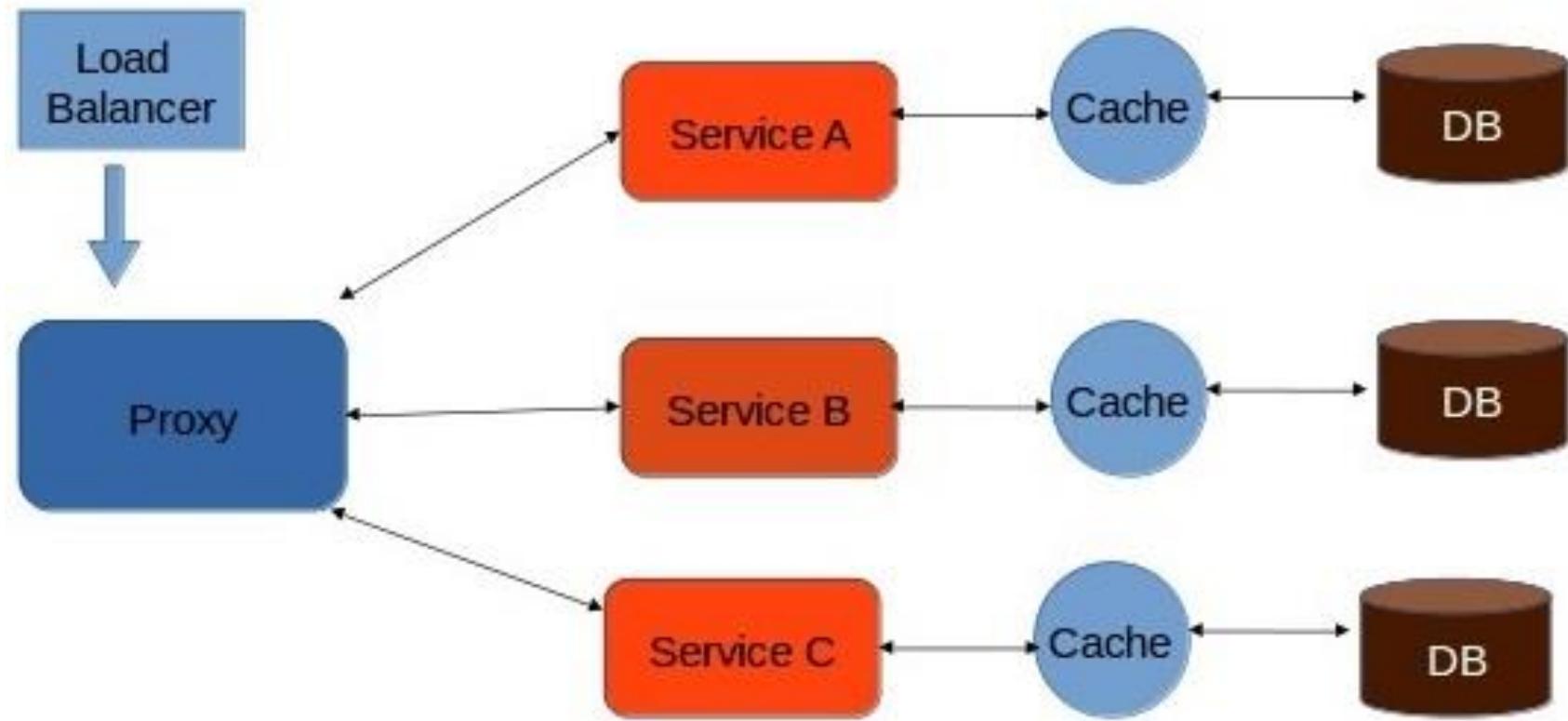




Proxy Pattern

- Proxy microservice design pattern is a variation of Aggregator.
- Proxy means ‘in place of’, representing’ or ‘in place of’ or ‘on behalf of’ are literal meanings of proxy and that directly explains Proxy Design Pattern.
- Proxies are also called surrogates, handles, and wrappers.
- They are closely related in structure, but not purpose, to Adapters and Decorators.
- A real world example can be a cheque or credit card is a proxy for what is in our bank account. It can be used in place of cash, and provides a means of accessing that cash when required. And that's exactly what the Proxy pattern does – “Controls and manage access to the object they are protecting”.

Proxy Pattern





UI composition pattern

- The end user interface layer is laid out into various sections, which individually invokes the corresponding microservice asynchronously.
- Modern user interfaces use single page application (SPA) built by Angular or ReactJS frameworks.



Backend for frontend

- Instead of creating a general-purpose microservice, we can design a microservice and its response specifically for the client agents (such as desktop browsers, mobile devices etc.).
- This tight coupling of client agents with the corresponding backend service helps us to efficiently create response data.



Gateway Routing Pattern

- The API gateway is responsible for request routing.
- An API gateway implements some API operations by routing requests to the corresponding service.
- When it receives a request, the API gateway consults a routing map that specifies which service to route the request to.
- A routing map might, for example, map an HTTP method and path to the HTTP URL of service.
- This function is identical to the reverse proxying features provided by web servers such as NGINX.

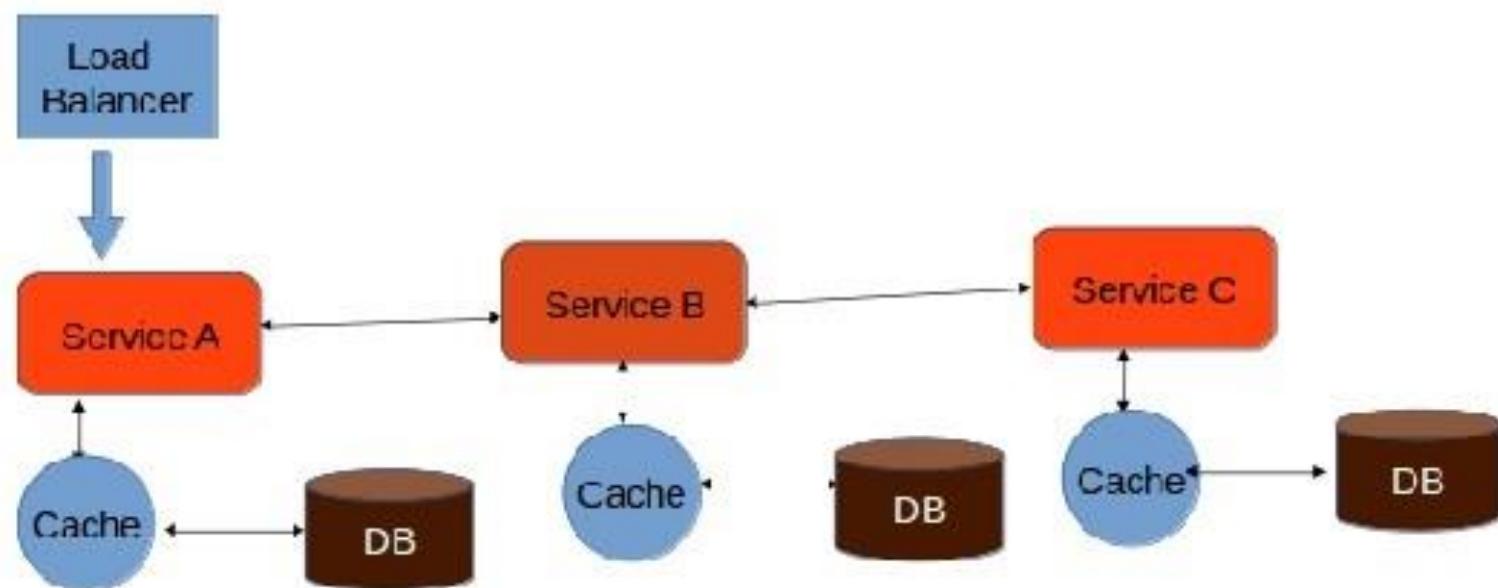


Chained Micro Service Pattern

- There will be multiple dependencies of for single services or microservice eg: Sale microservice has dependency products microservice and order microservice.
- Chained microservice design pattern will help you to provide the consolidated outcome to your request.
- The request received by a microservice: 1, which is then communicating with microservice-2 and it may be communicating with microservice-3.
- All these services are synchronous calls.

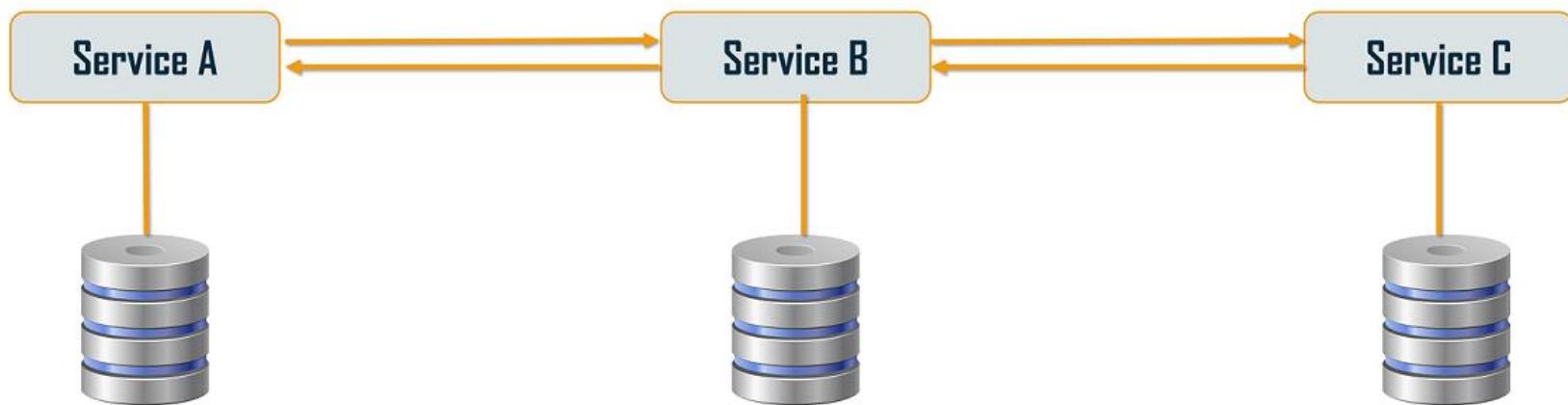


Chained Micro Service Pattern





Chained or Chain of Responsibility Pattern

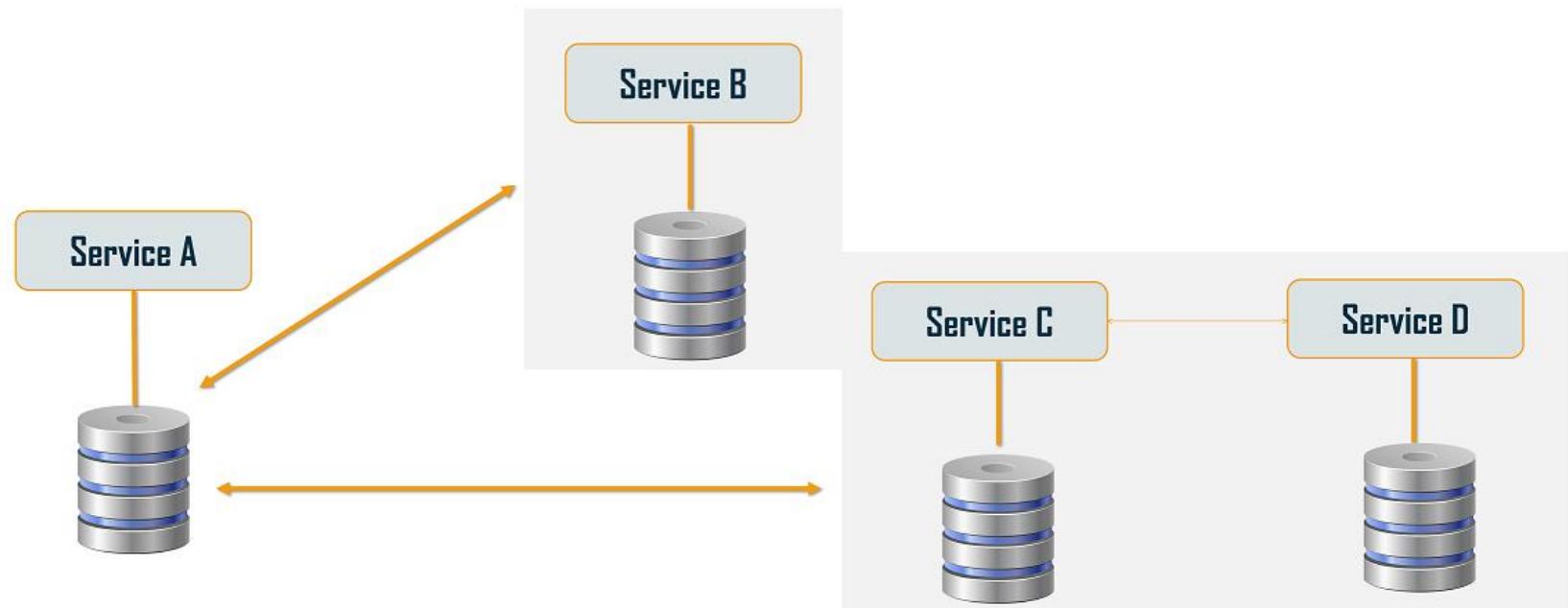




Branch Pattern

- Branch microservice is a design pattern in which you can simultaneously process the requests and responses from two or more independent microservices.
- So, unlike the chained design pattern, the request is not passed in a sequence, but the request is passed to two or more mutually exclusive microservices chains.
- This design pattern extends the Aggregator design pattern and provides the flexibility to produce responses from multiple chains or single chain.
- For example, if you consider an e-commerce application, then you may need to retrieve data from multiple sources and this data could be a collaborated output of data from various services.
- So, you can use the branch pattern, to retrieve data from multiple sources.

Branch Pattern



Database Patterns



DATA-RELATED PATTERNS

- As microservices are self-contained and designed for independent scalability, we end up having service-specific databases (such as database server per service, database schema per service and service-specific tables).
- Due to this design, we face challenges such as:
 - A single service that reads or updates data from multiple databases
 - A single business transaction spanning multiple services and databases
 - Replication of data in the databases



Database per Service

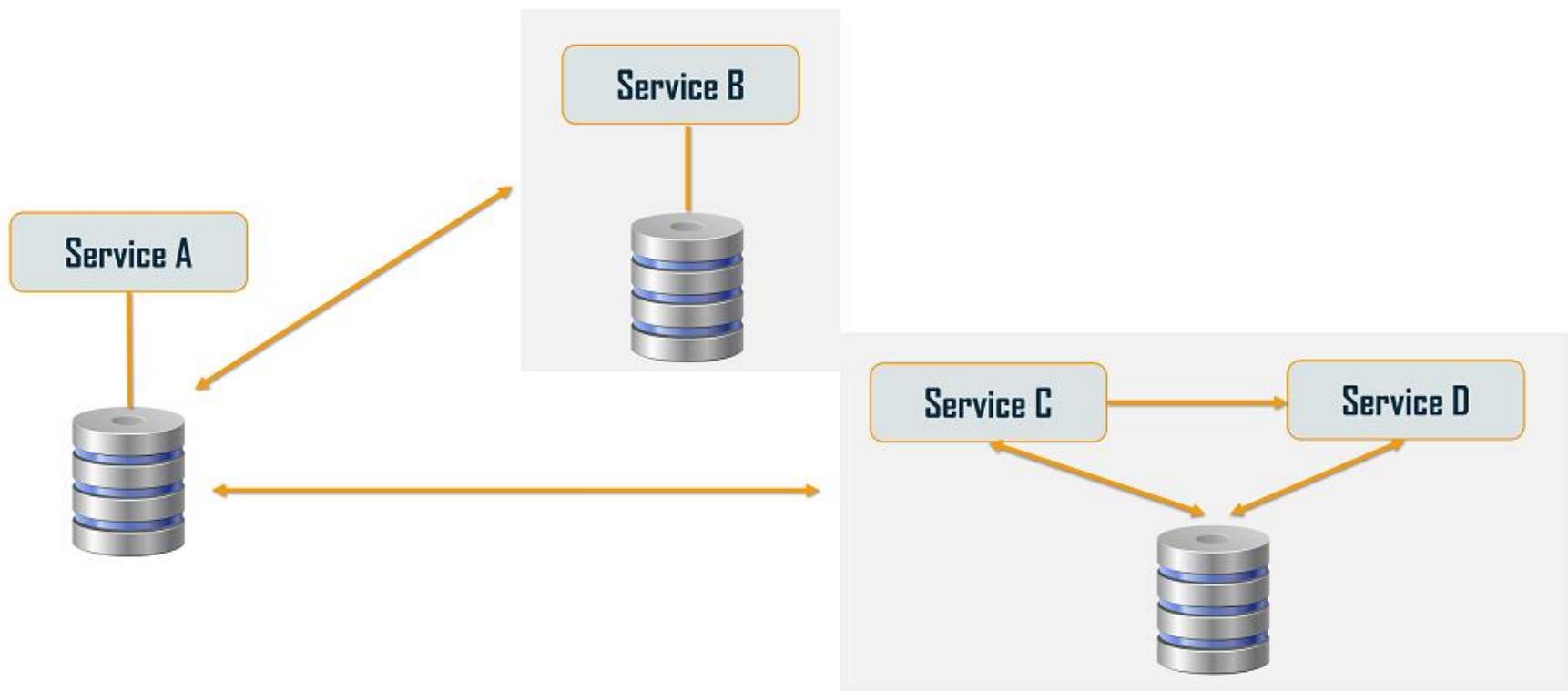
- To solve the above concerns, one database per microservice must be designed; it must be private to that service only.
- It should be accessed by the microservice API only.
- It cannot be accessed by other services directly.
- For example, for relational databases, we can use private-tables-per-service, schema-per-service, or database-server-per-service.



Shared Database per service

- It is an anti-pattern for microservices.
- But if the application is a monolith and trying to break into microservices, denormalization is not that easy.
- In the later phase, we can move to DB per service pattern.
- A shared database per service is not ideal, but that is the working solution for the above scenario.
- Most people consider this an anti-pattern for microservices, but for brownfield applications, this is a good start to break the application into smaller logical pieces.
- This should not be applied for greenfield applications.

Database or Shared Data Pattern



Command Query Responsibility Segregation (CQRS)



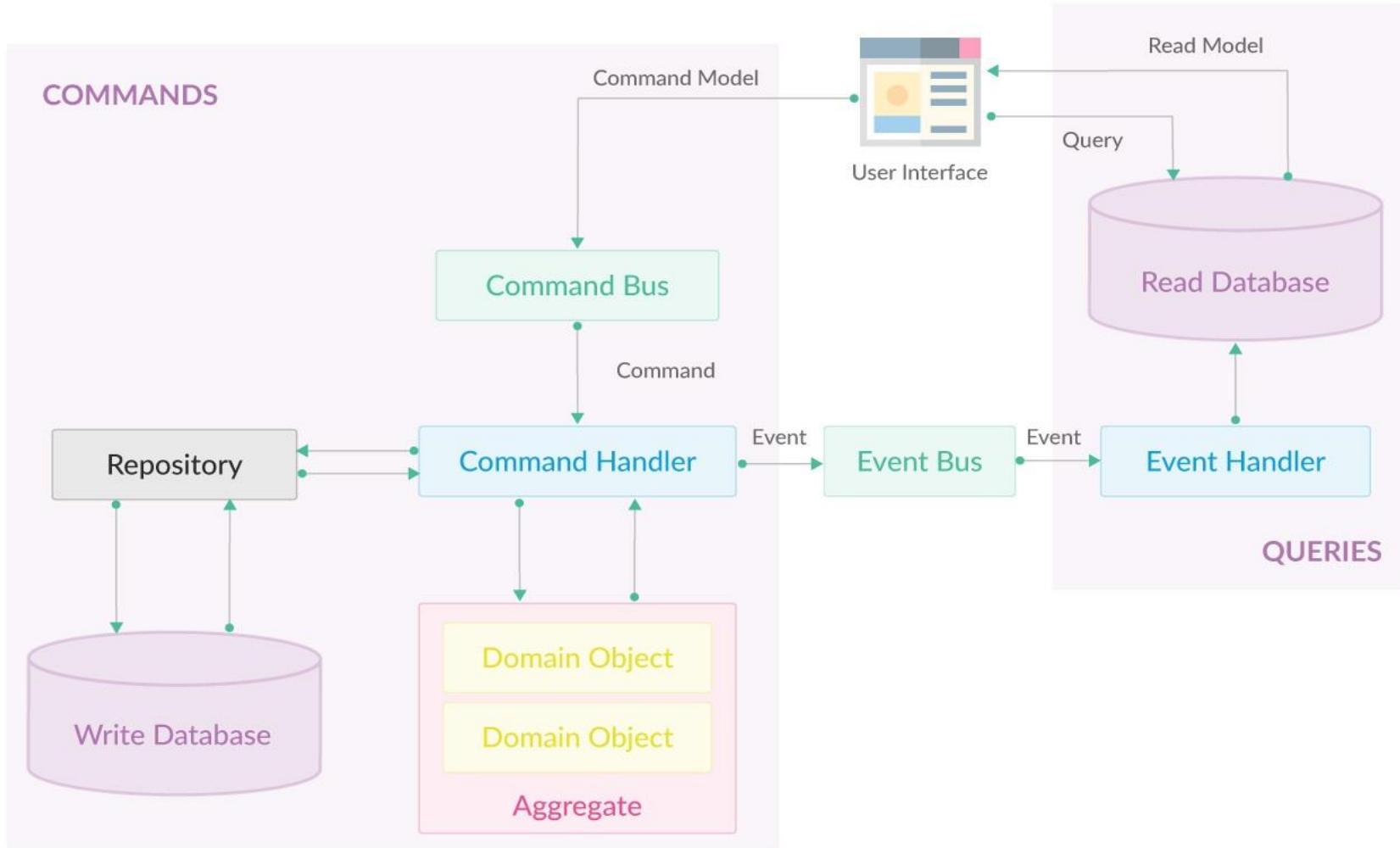
- Database handling is split into two categories, the **command part** for handling data creation, update, deletion and the **query part** that uses materialized views to retrieve data.
- The materialized view is updated by subscribing to data change events.
- Event sourcing pattern is used along with CQRS to create immutable events.

Command Query Responsibility Segregation (CQRS)



- Once we implement database-per-service, there is a requirement to query, which requires joint data from multiple services.
- CQRS suggests splitting the application into two parts—the command side and the query side.
- The command side handles the Create, Update, and Delete requests.
- The query side handles the query part by using the materialized views.

Command Query Responsibility Segregation (CQRS)



Command Query Responsibility Segregation (CQRS)



- **Commands represent user intent.** They contain all the necessary information about actions that user would like to perform.
- **Command Bus** is a type of queue that receives commands and passes them to Command Handlers.
- **Command Handlers** contain actual business logic which validates and processes data received in commands. Command handlers are responsible for generation and propagation of domain events to Event Bus
- **Event Bus** dispatches events to event handlers subscribed for specific event types. Event bus can propagate events both asynchronously or synchronously if consecutive events are dependent.
- **Event Handlers** are responsible for handling specific incoming events. Their role is to save the new state of the application in read repository, and perform terminal actions like sending emails, storing files etc.

Command Query Responsibility Segregation (CQRS)



- Queries are objects, which represent the actual state of application available for a user. Getting data for UI should be done through these objects.
- We can distinguish many advantages of CQRS approach including the following:
- We can split development tasks between people more experienced who will be working on the business logic and those ones who will be working on queries part.
- We have to be careful because this advantage may hurt knowledge transfer.
- We can achieve a great read/write performance by scaling commands and queries on multiple different servers.

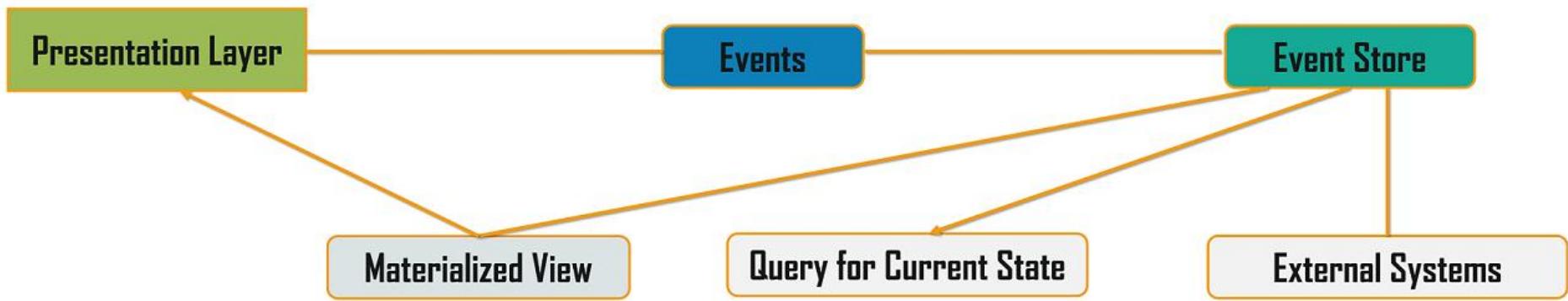
Command Query Responsibility Segregation (CQRS)



- Using two different repositories (read/write) which are synchronized gives us automatic backup without any additional effort.
- Reads don't hit write database, so they can go faster when using Event Sourcing.
- We can structure read data directly for the views, without thinking about domain logic, which simplifies views and can improve performance.



Event Sourcing Design Pattern





Event Sourcing

- The event sourcing pattern is generally used along with it to create events for any data change.
- Materialized views are kept updated by subscribing to the stream of events.
- Event Sourcing Most applications work with data, and the typical approach is for the application to maintain the current state.
- For example, in the traditional create, read, update, and delete (CRUD) model a typical data process is to read data from the store.
- It contains limitations of locking the data, often using transactions.



Event Sourcing

- The Event Sourcing pattern defines an approach to handle operations on data that's driven by a sequence of events, each of which is recorded in an append-only store.
- Application code sends a series of events that imperatively describe each action that has occurred on the data to the event store, where they're persisted.
- Each event represents a set of changes to the data (such as `AddedItemToOrder`).
- The events are persisted in an event store that acts as the system of record.

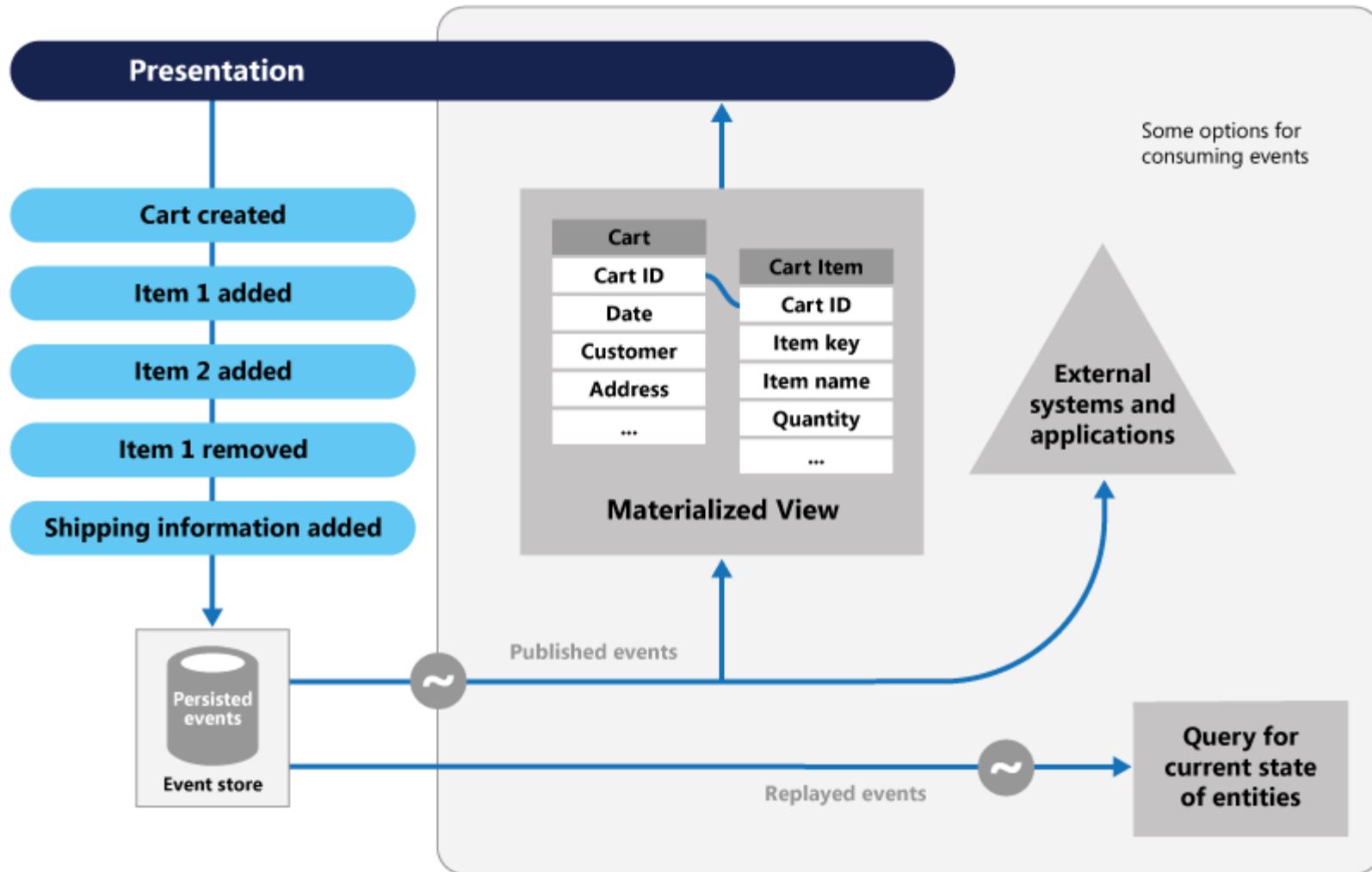


Event Sourcing

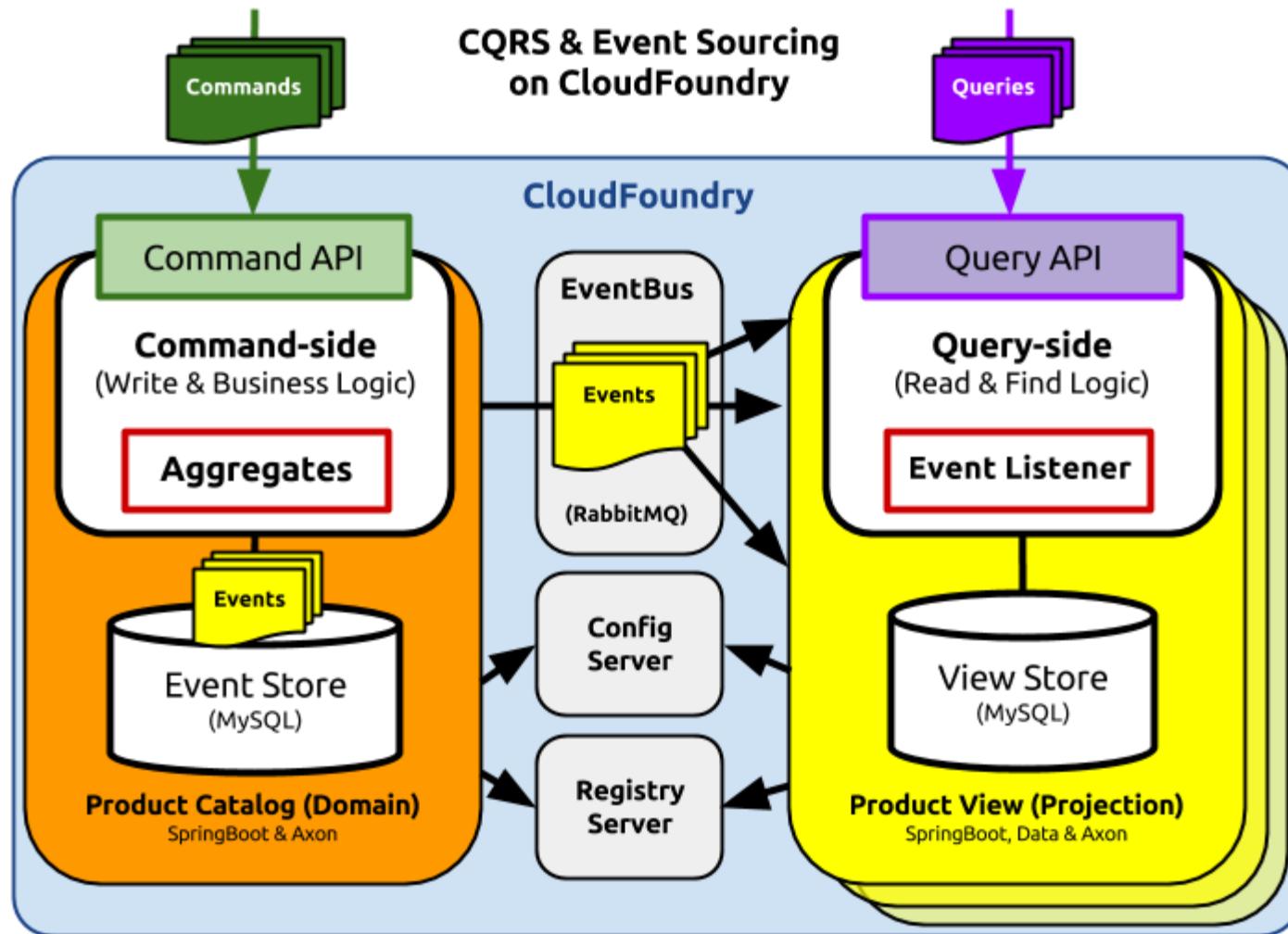
- Typical uses of the events published by the event store are to maintain materialized views of entities as actions in the application change them, and for integration with external systems.
- For example, a system can maintain a materialized view of all customer orders that are used to populate parts of the UI.
- As the application adds new orders, adds or removes items on the order, and adds shipping information, the events that describe these changes can be handled and used to update the materialized view. .



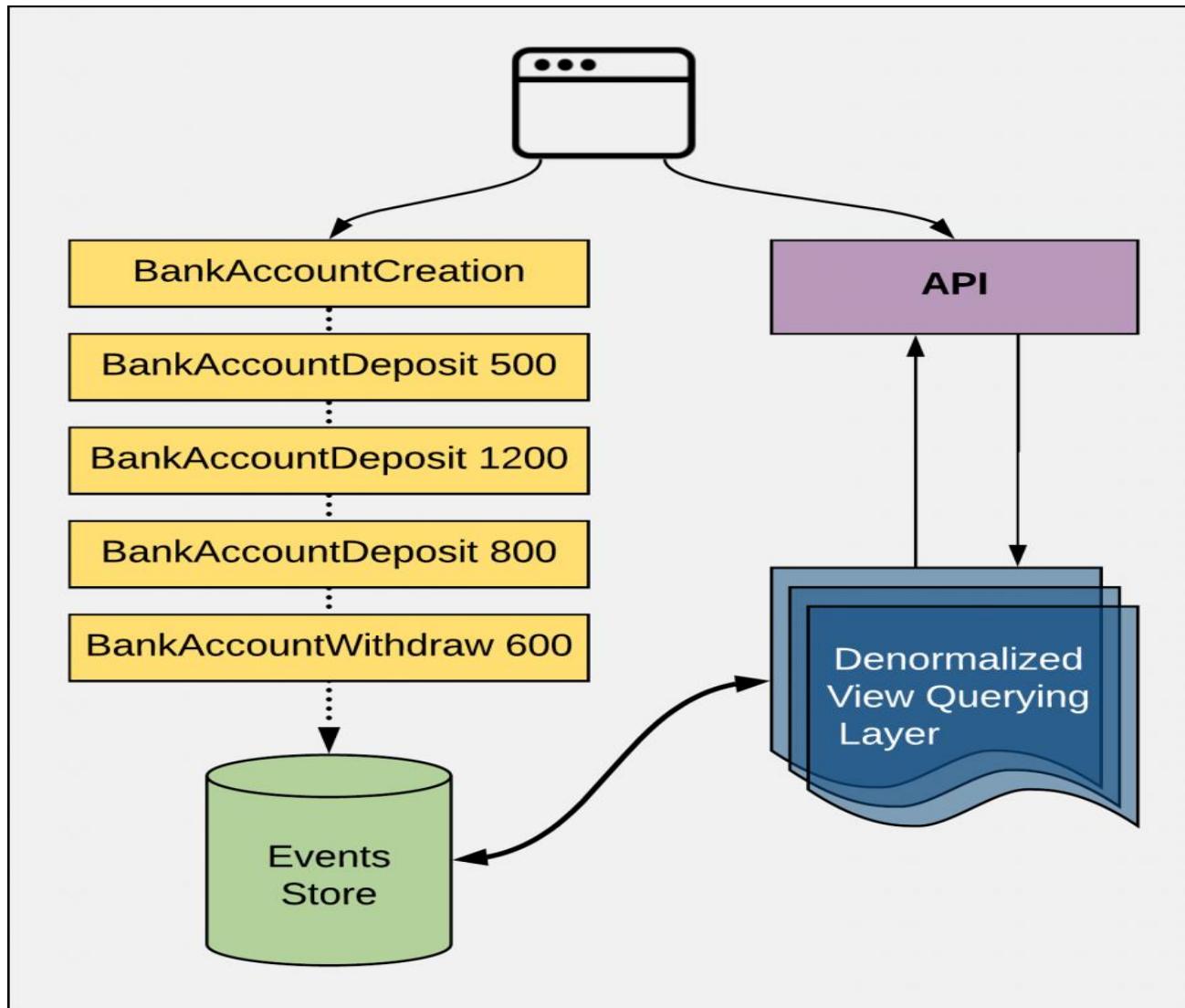
Event Sourcing



Event Sourcing

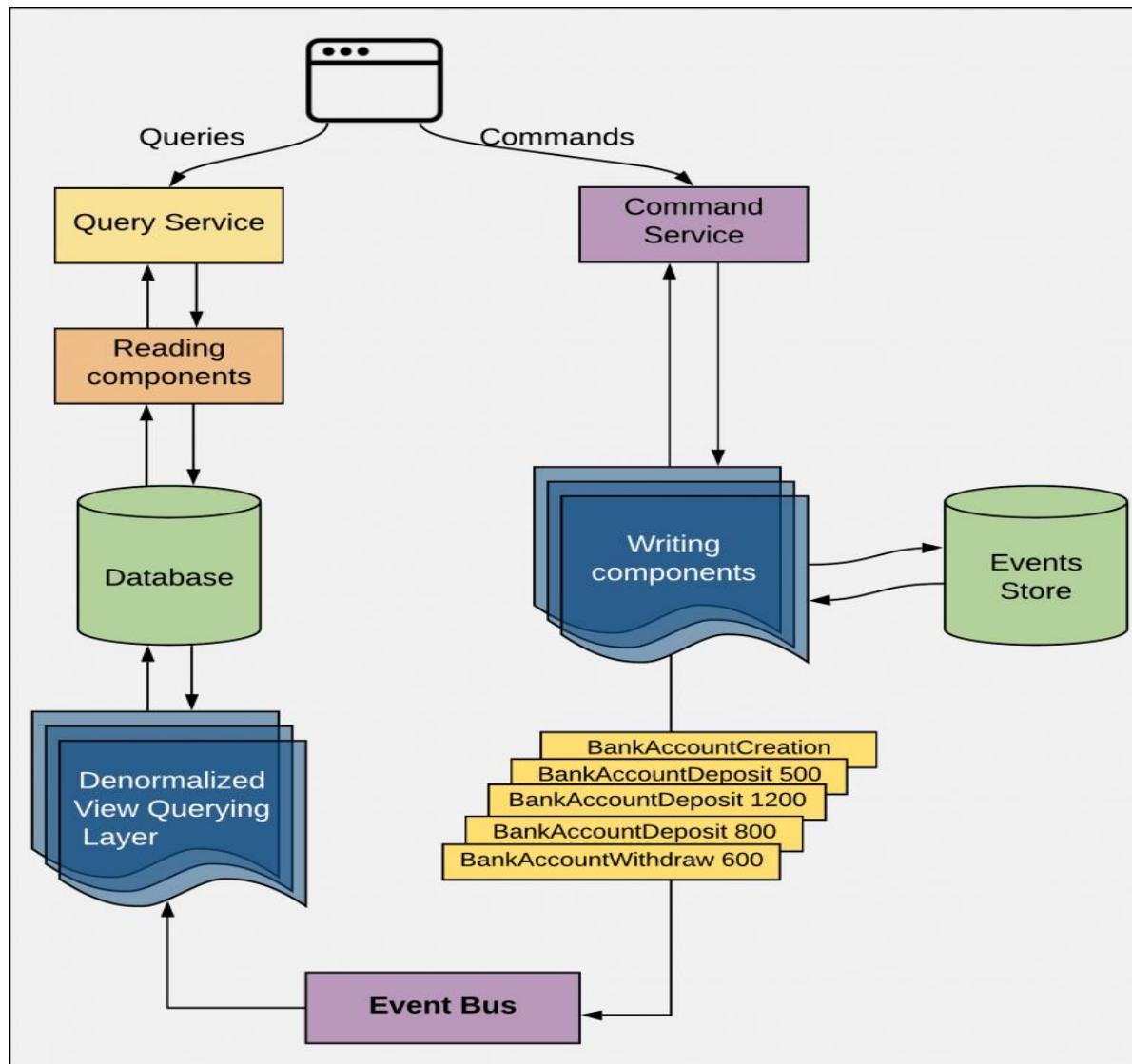


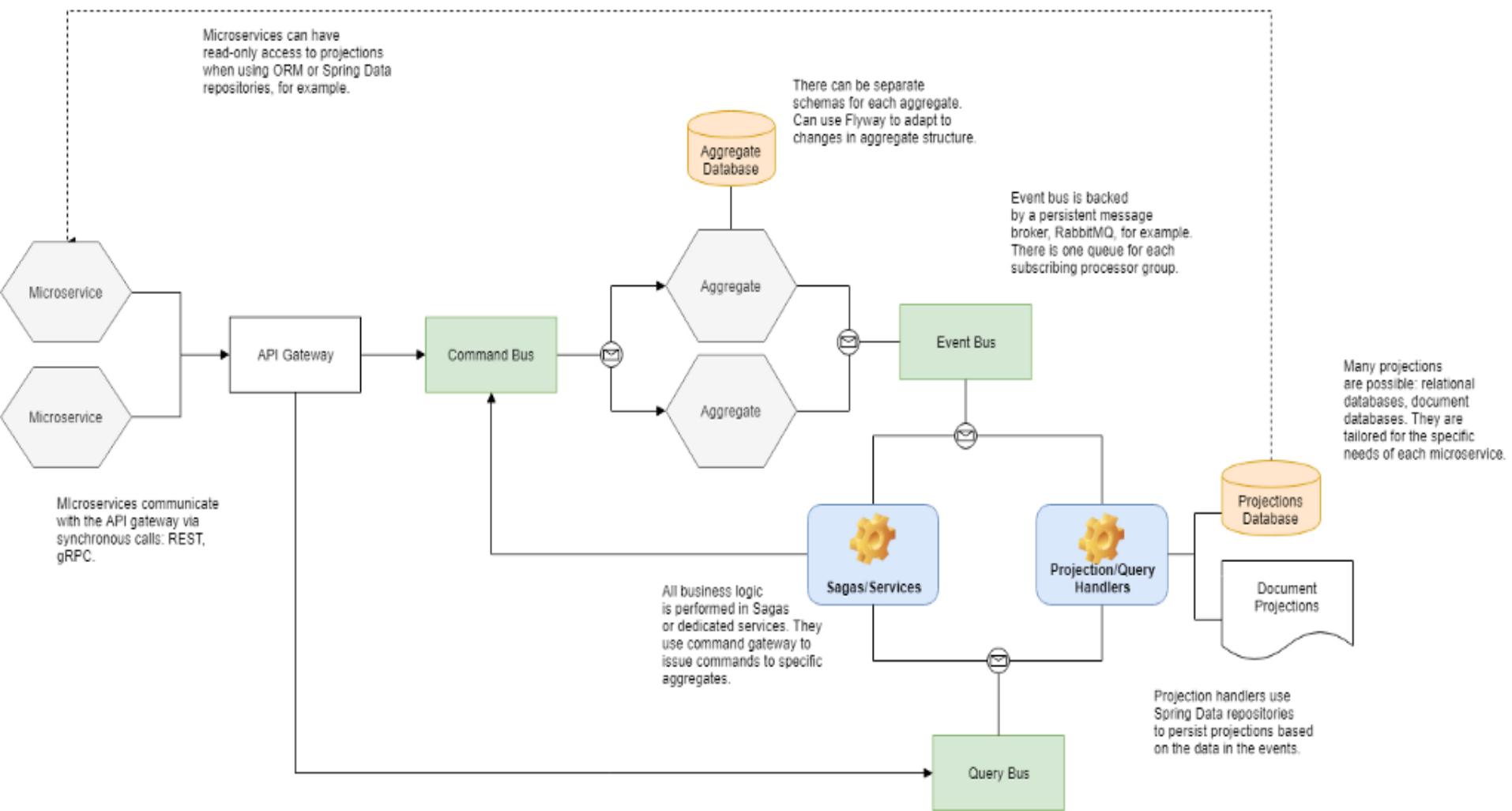
Event Sourcing



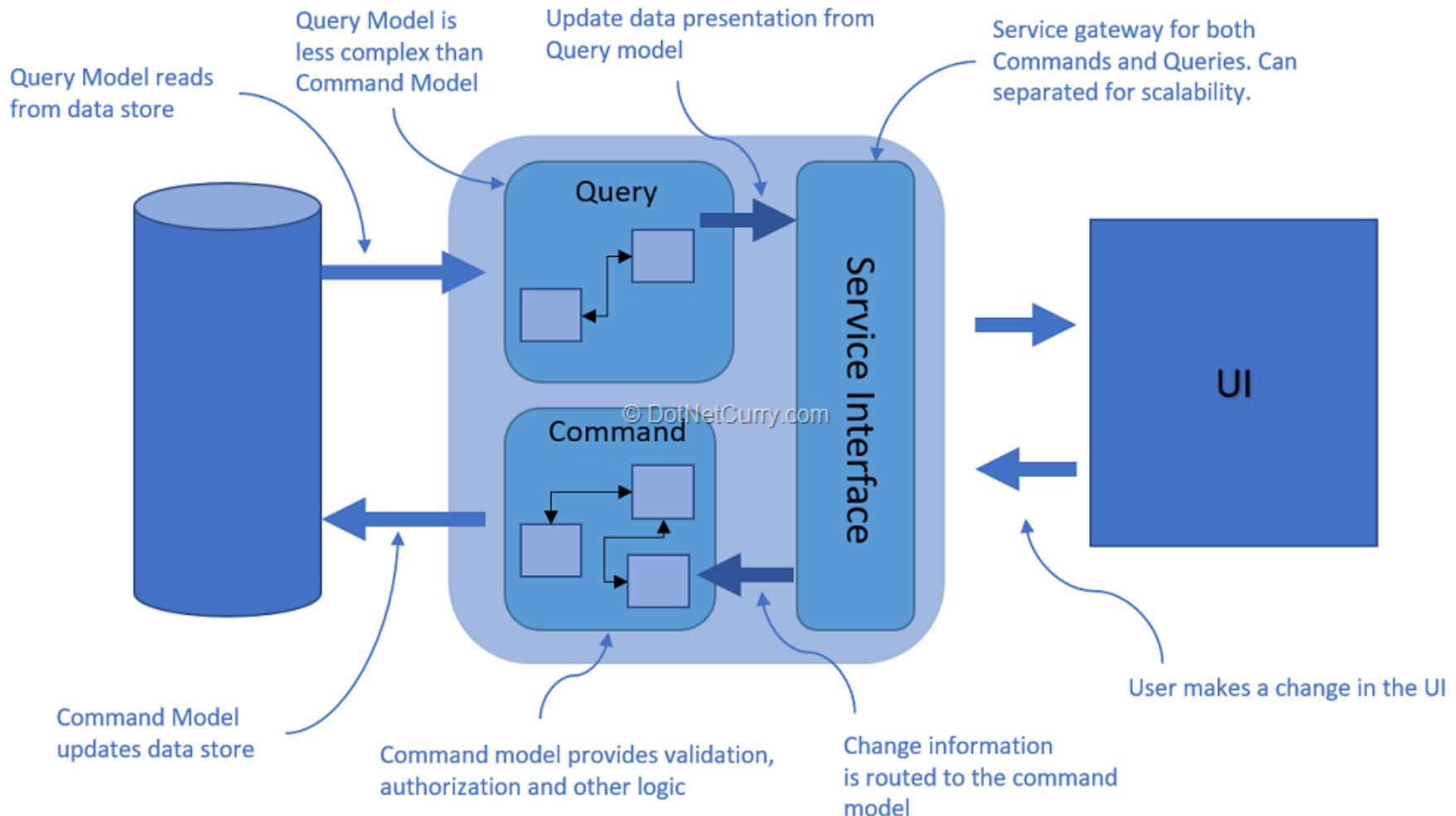


CQRS and Event Sourcing





Command and Query Responsibility Segregation (CQRS) pattern



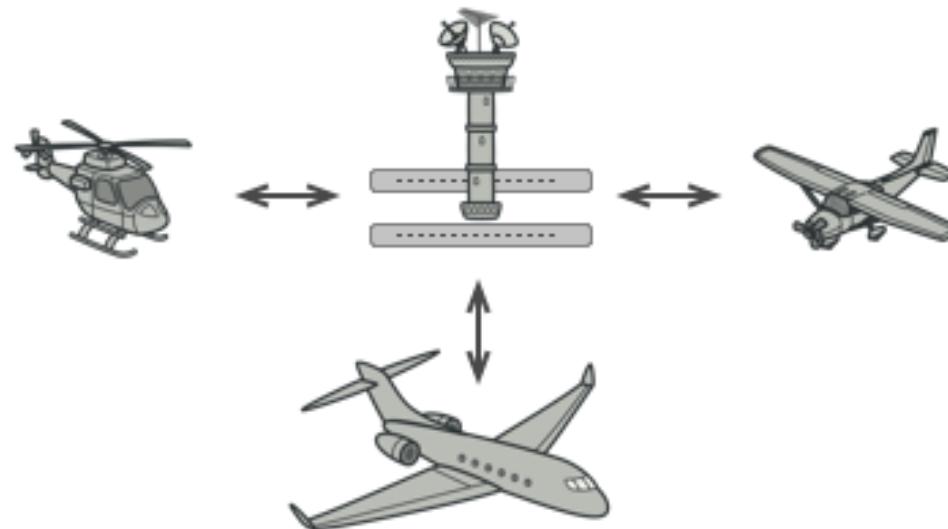


Mediator Pattern

- While building applications, especially ASP.NET Core Applications, it's highly important to keep in mind that we always have to keep the code inside controllers as minimal as possible.
- Theoretically, Controllers are just routing mechanisms that take in a request and sends it internally to other specific services/libraries and return the data.
- It wouldn't really make sense to put all your validations and logics within the Controllers.
- Mediator pattern is yet another design pattern that dramatically reduces the coupling between various components of an application by making them communicate indirectly, usually via a special mediator object.

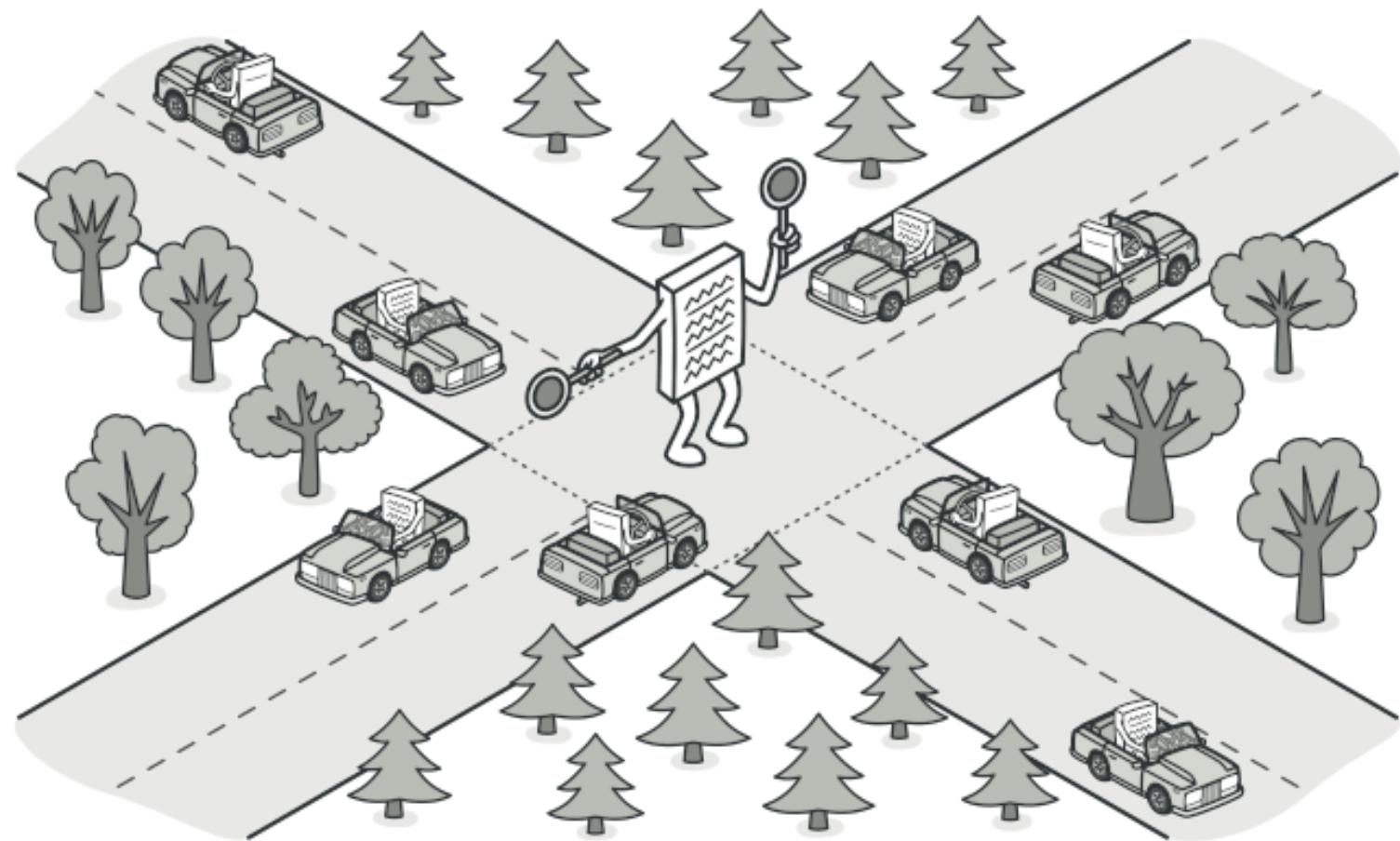
Mediator Pattern

- Mediator is a behavioral design pattern that lets you reduce chaotic dependencies between objects.
- The pattern restricts direct communications between the objects and forces them to collaborate only via a mediator object.



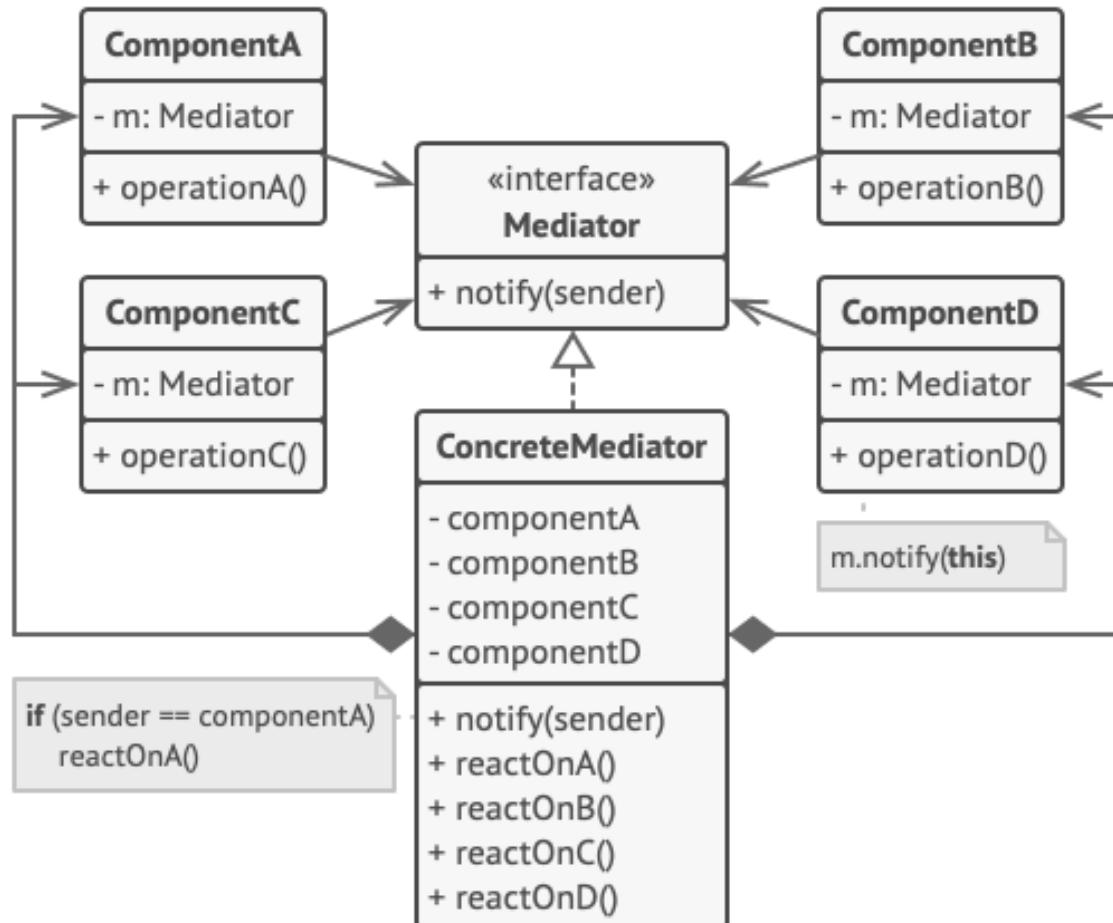


Mediator Pattern



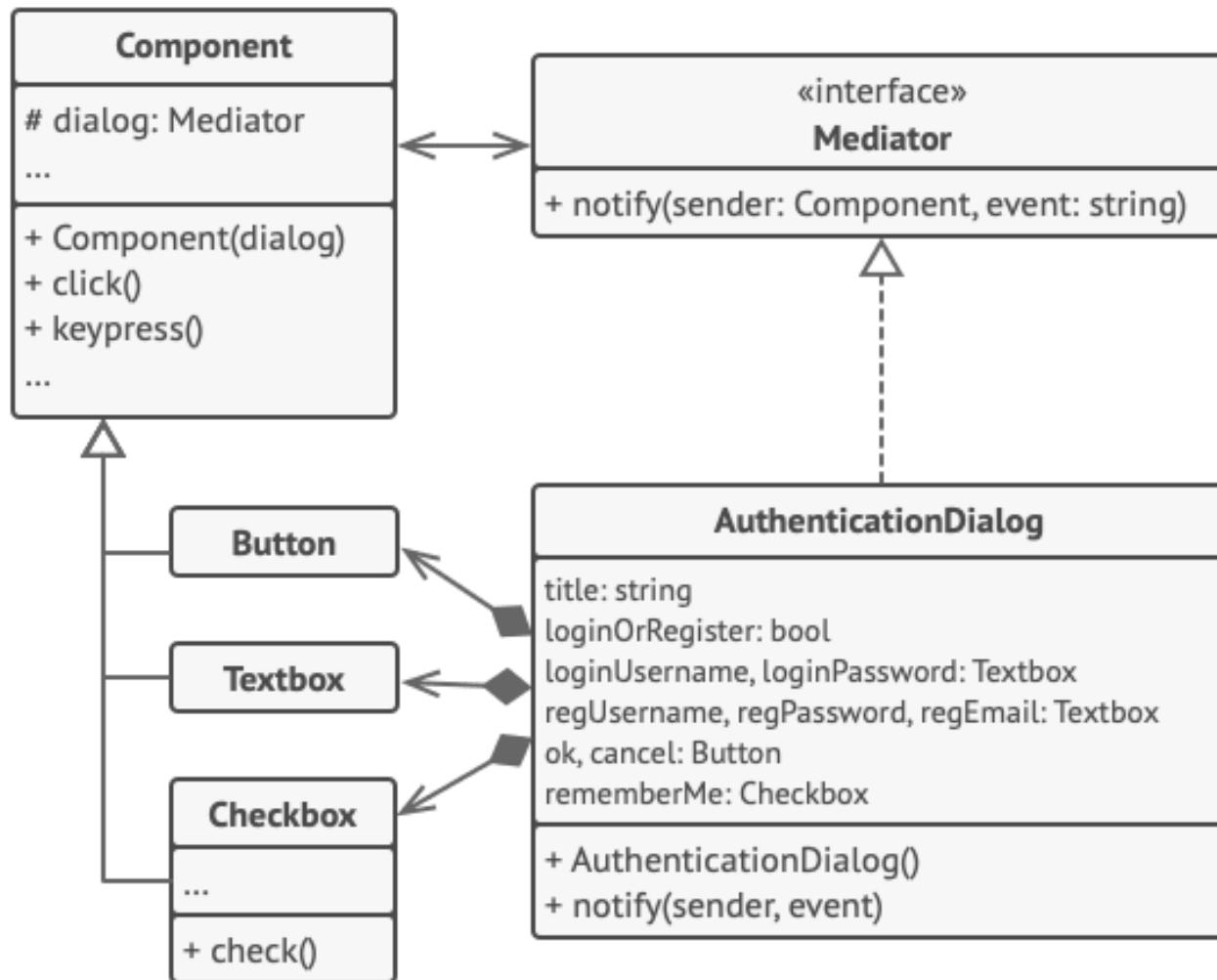


Mediator Pattern





Mediator Pattern





Media Pattern

- **Applicability**
- Use the Mediator pattern when it's hard to change some of the classes because they are tightly coupled to a bunch of other classes.
- Use the pattern when you can't reuse a component in a different program because it's too dependent on other components.
- Use the Mediator when you find yourself creating tons of component subclasses just to reuse some basic behavior in various contexts.



Media Pattern

- **Single Responsibility Principle.** You can extract the communications between various components into a single place, making it easier to comprehend and maintain.
- **Open/Closed Principle.** You can introduce new mediators without having to change the actual components.
- You can reduce coupling between various components of a program.
- You can reuse individual components more easily.



Mediator Pattern

- MediatR Library
- MediatR is a library that helps implements Mediator Pattern in .NET. The first thing we need to do, is to install the following packages.
- Install-Package MediatR
- Install-Package
MediatR.Extensions.Microsoft.DependencyInjection

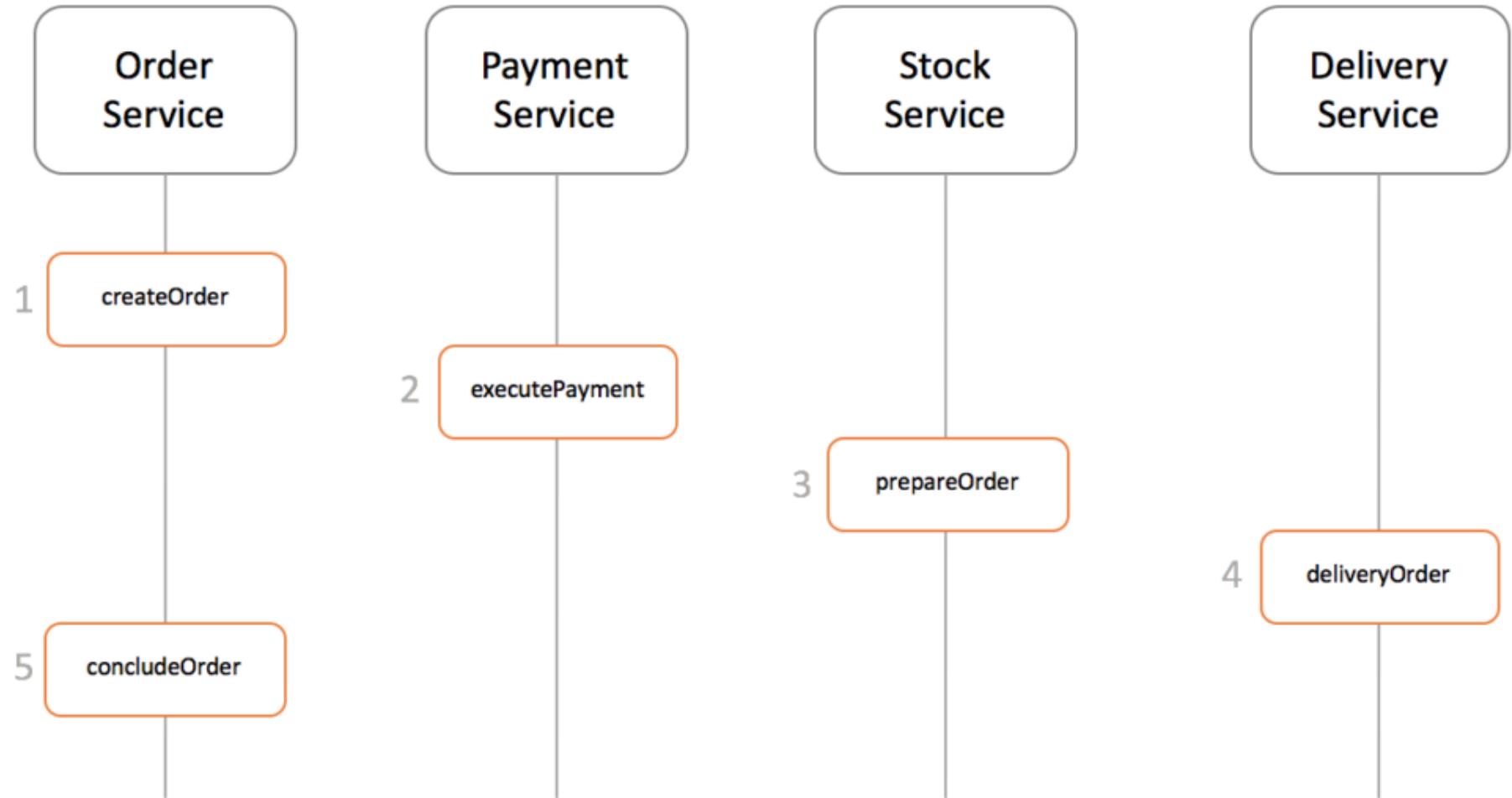


Saga Pattern

- When a business transaction needs to manage data consistency that is spread across multiple databases, we could use Saga pattern.
- As a part of the Saga pattern, each transaction is orchestrated locally or centrally to execute it entirely and handle the failure/rollback scenario.
- For instance, if a business transaction needs to handle data related to order and customer service, each of these services produces and listens to each other to handle the transaction.



Saga Design Pattern



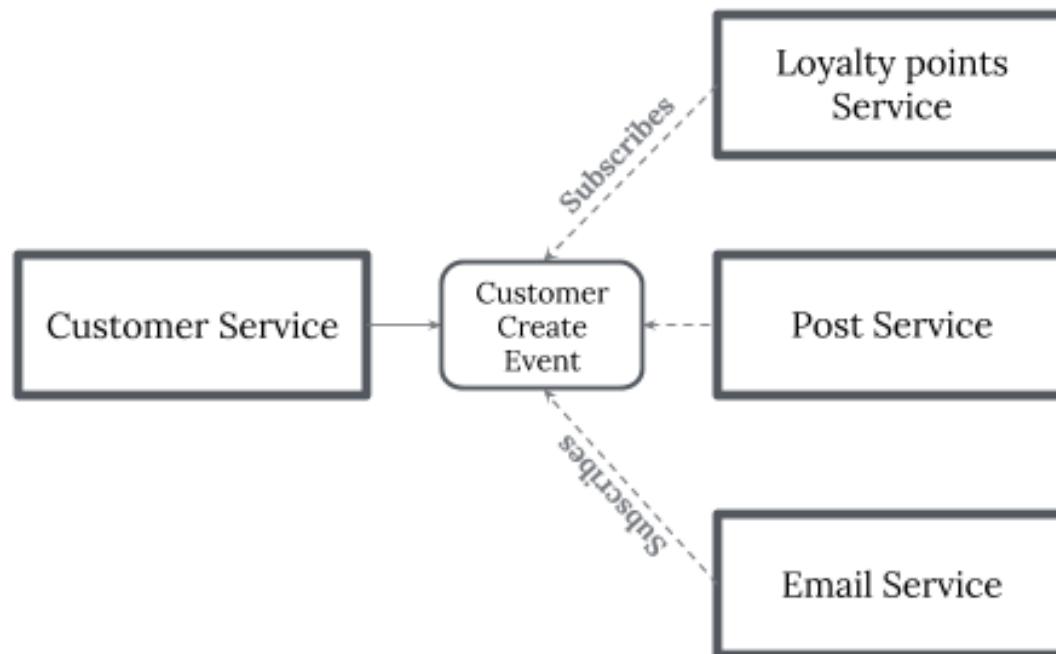


Saga Pattern

- When each service has its own database and a business transaction spans multiple services, how do we ensure data consistency across services?
- Each request has a compensating request that is executed when the request fails.
- It can be implemented in two ways:
- Choreography—When there is no central coordination, each service produces and listens to another service's events and decides if an action should be taken or not.
- Choreography is a way of specifying how two or more parties; none of which have any control over the other parties' processes, or perhaps any visibility of those processes—can coordinate their activities and processes to share information and value.

Saga Pattern

- Use choreography when coordination across domains of control/visibility is required.
- It dictates acceptable patterns of requests and responses between parties.



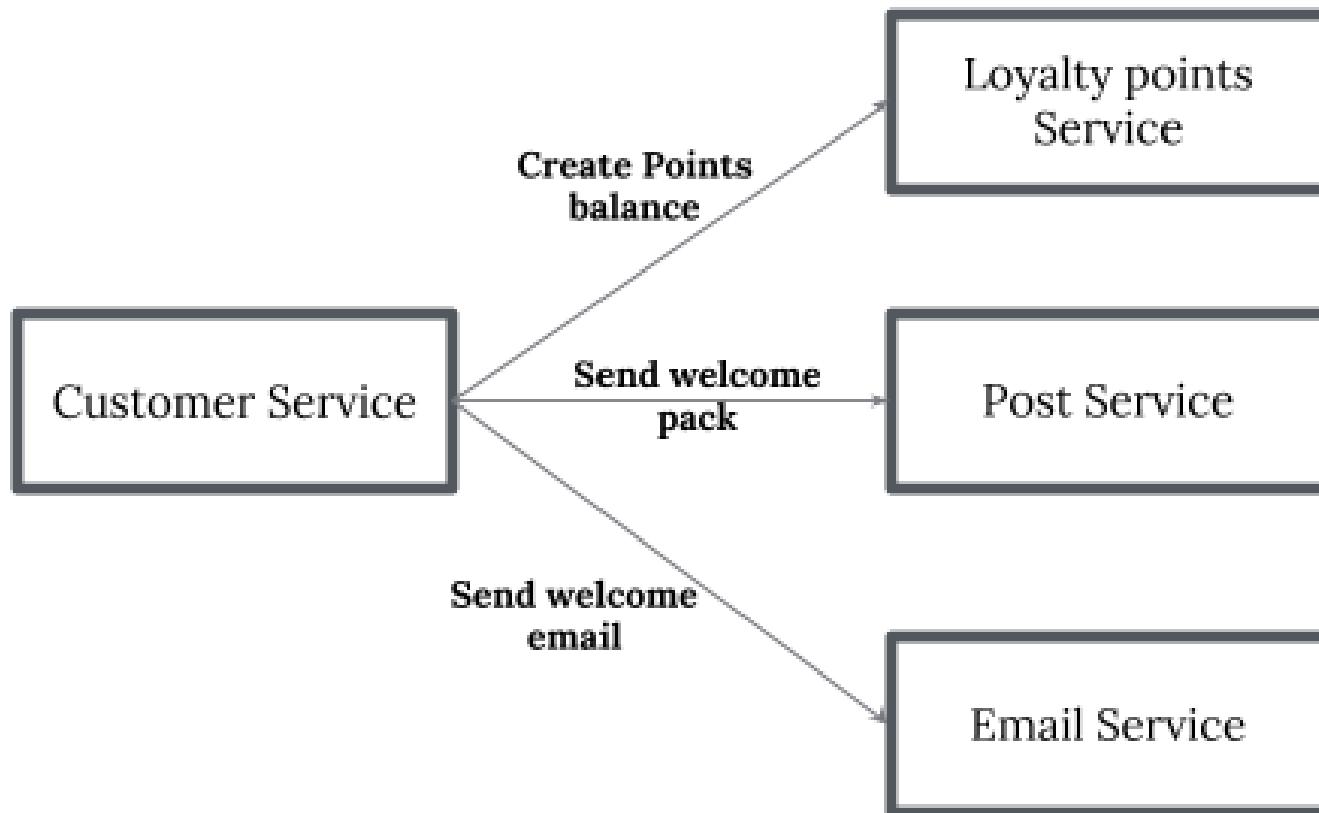


Saga Pattern

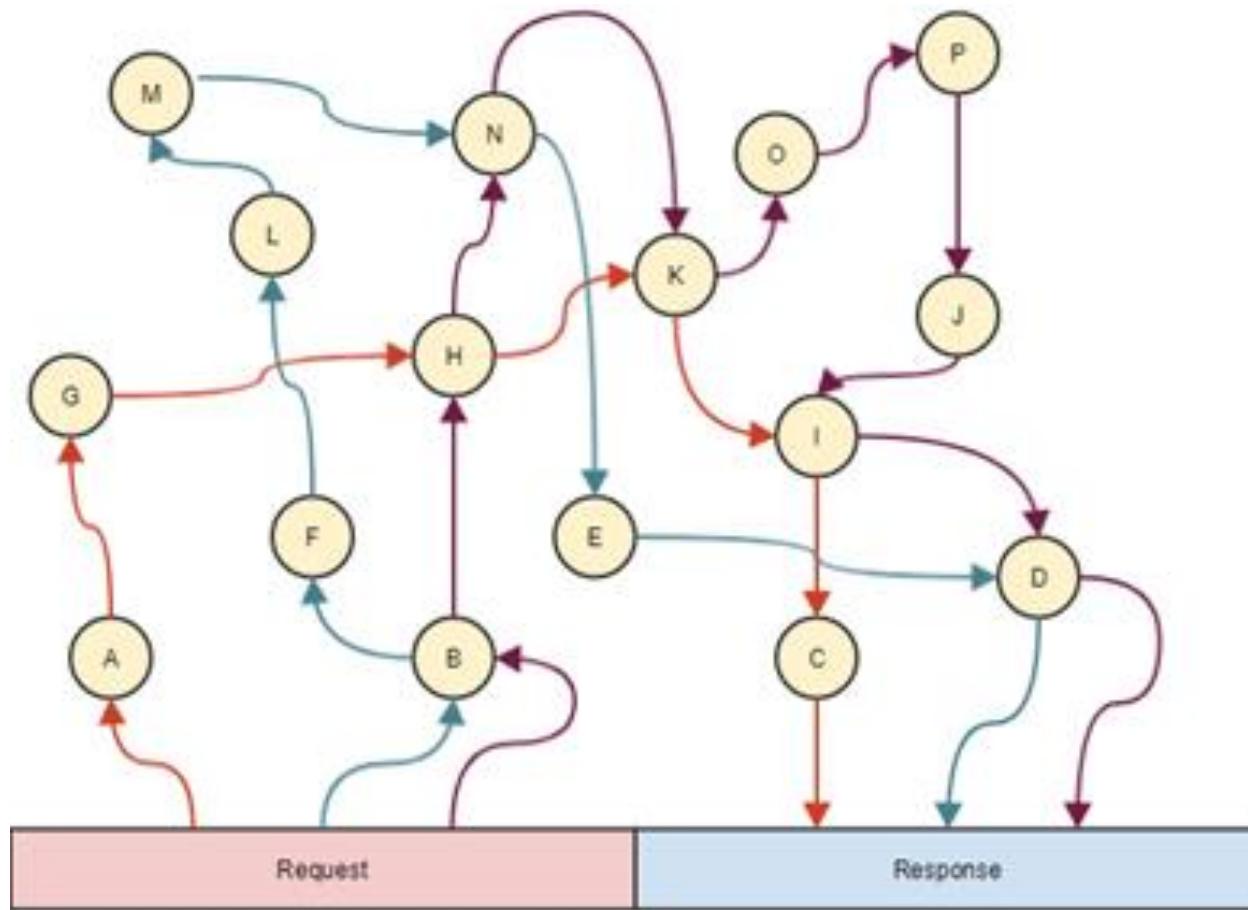
- Orchestration—An orchestrator (object) takes responsibility for a saga's decision making and sequencing business logic.
- when you have control over all the actors in a process.
when they're all in one domain of control and you can control the flow of activities.
- This is, of course, most often when you're specifying a business process that will be enacted inside one organization that you have control over.



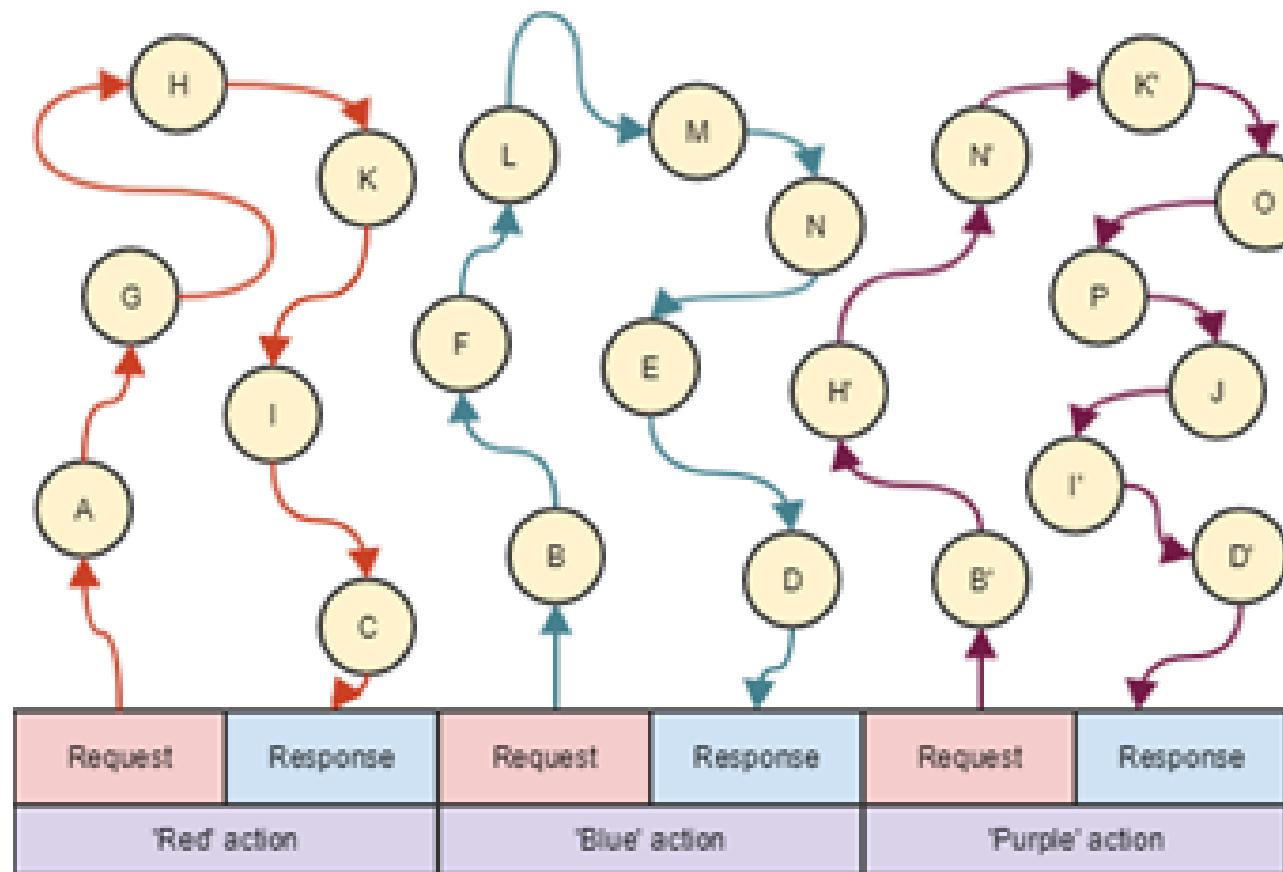
Saga Pattern



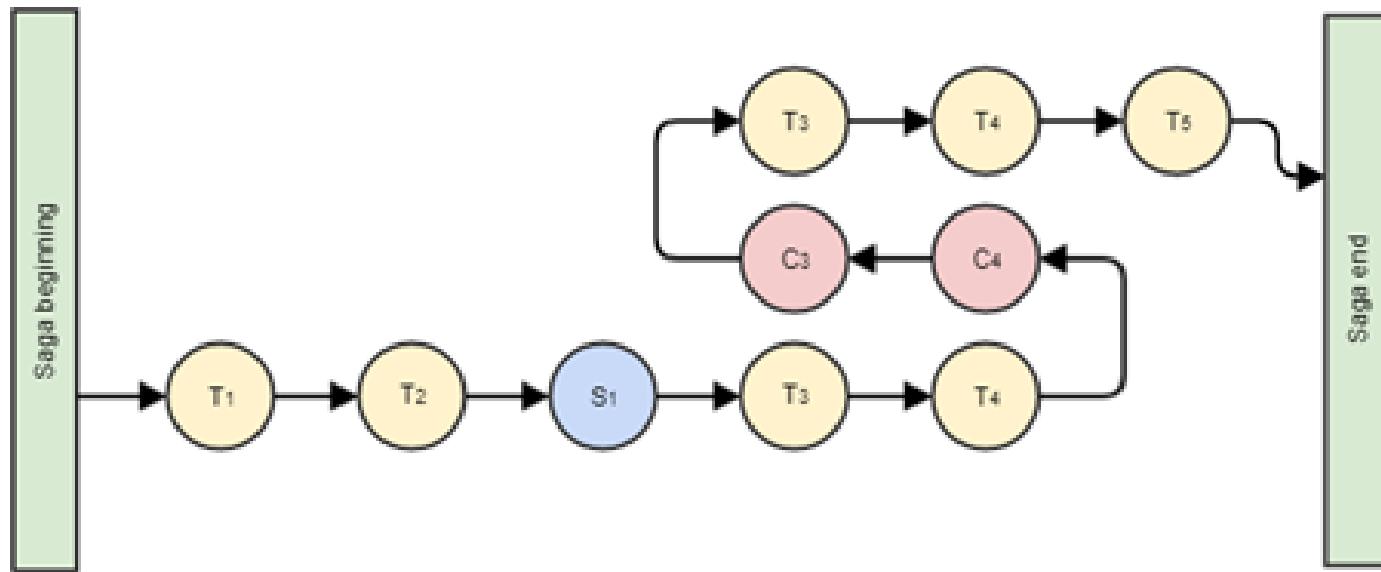
Saga Pattern



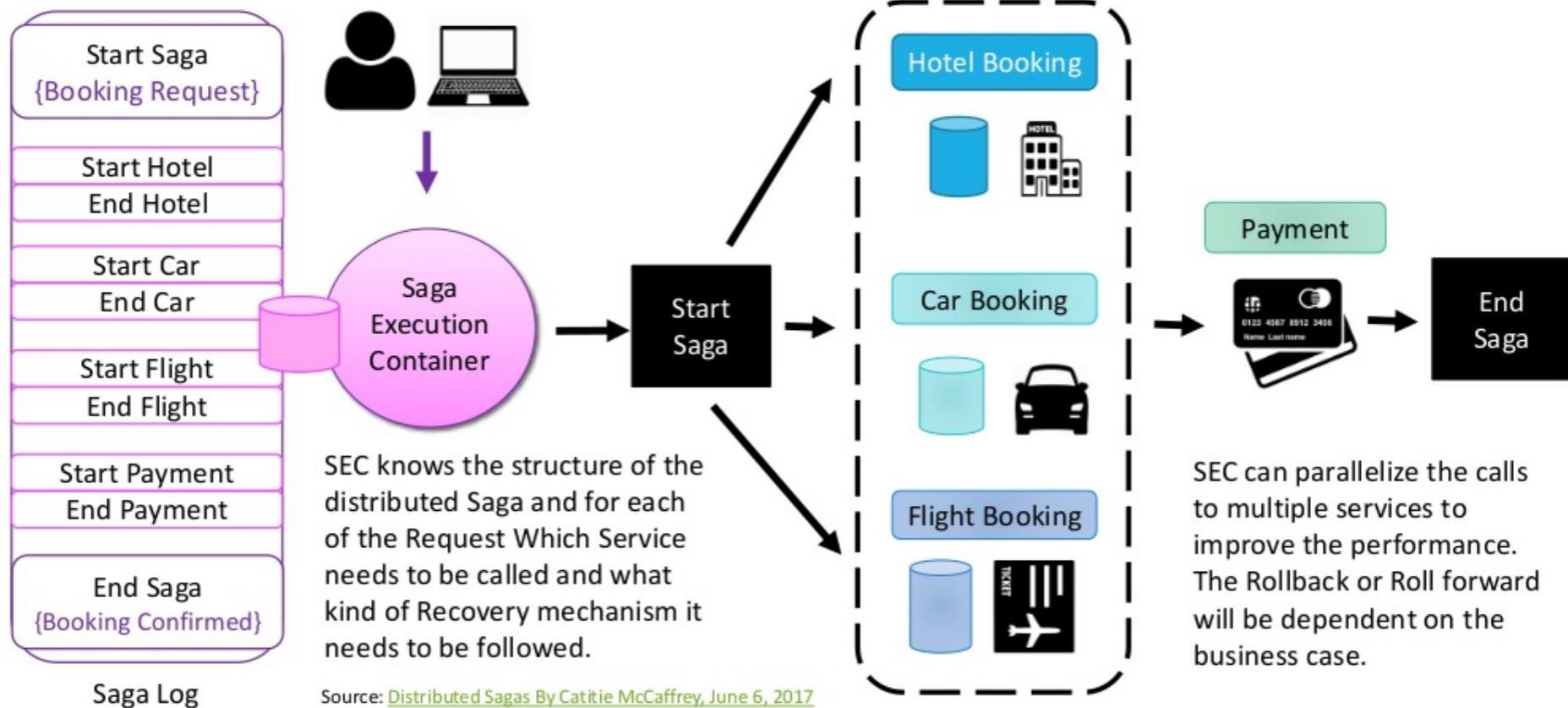
Saga Pattern



Saga Pattern



Use Case : Travel Booking – Distributed Saga (SEC)



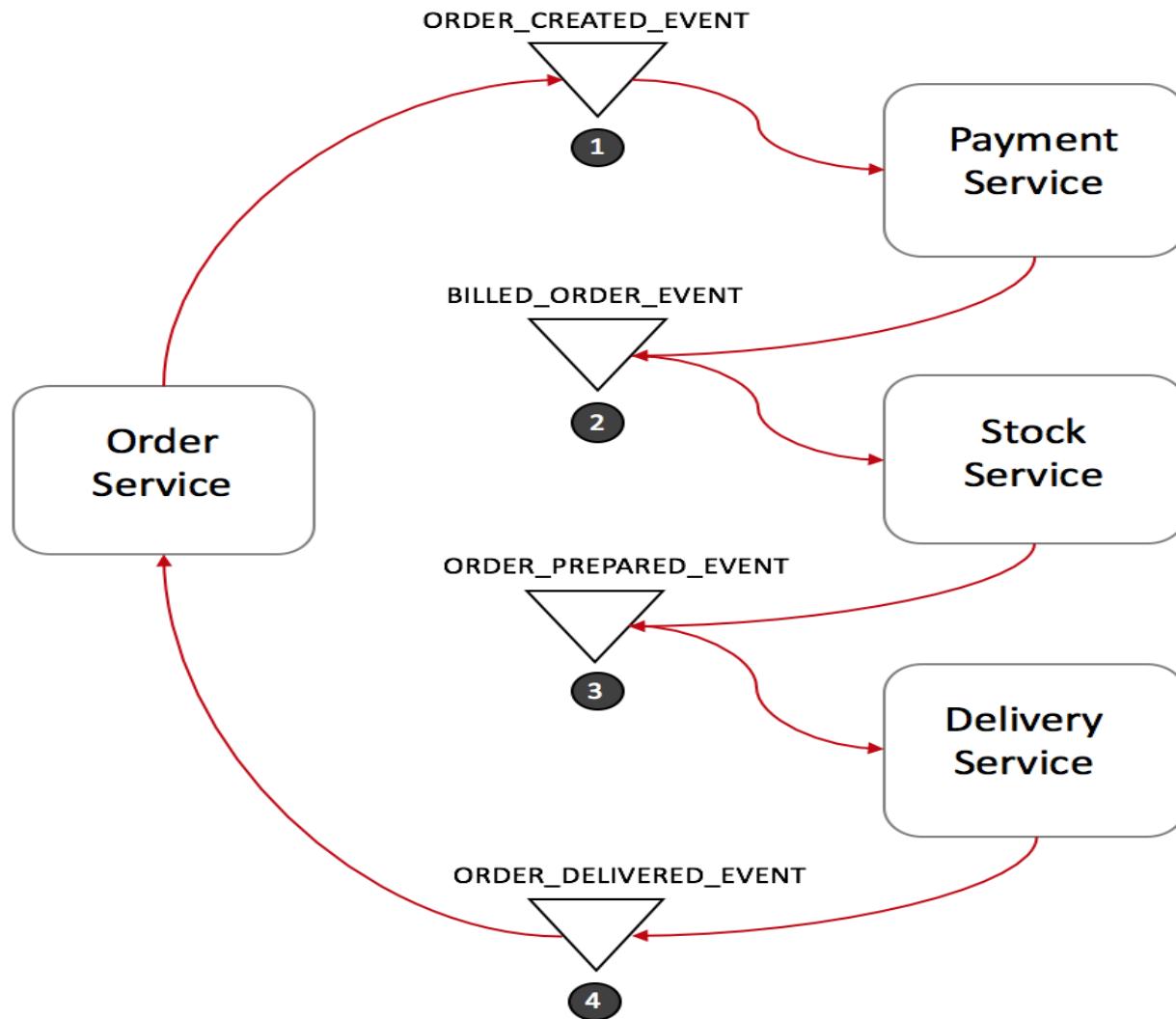


Saga Design Pattern

- 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

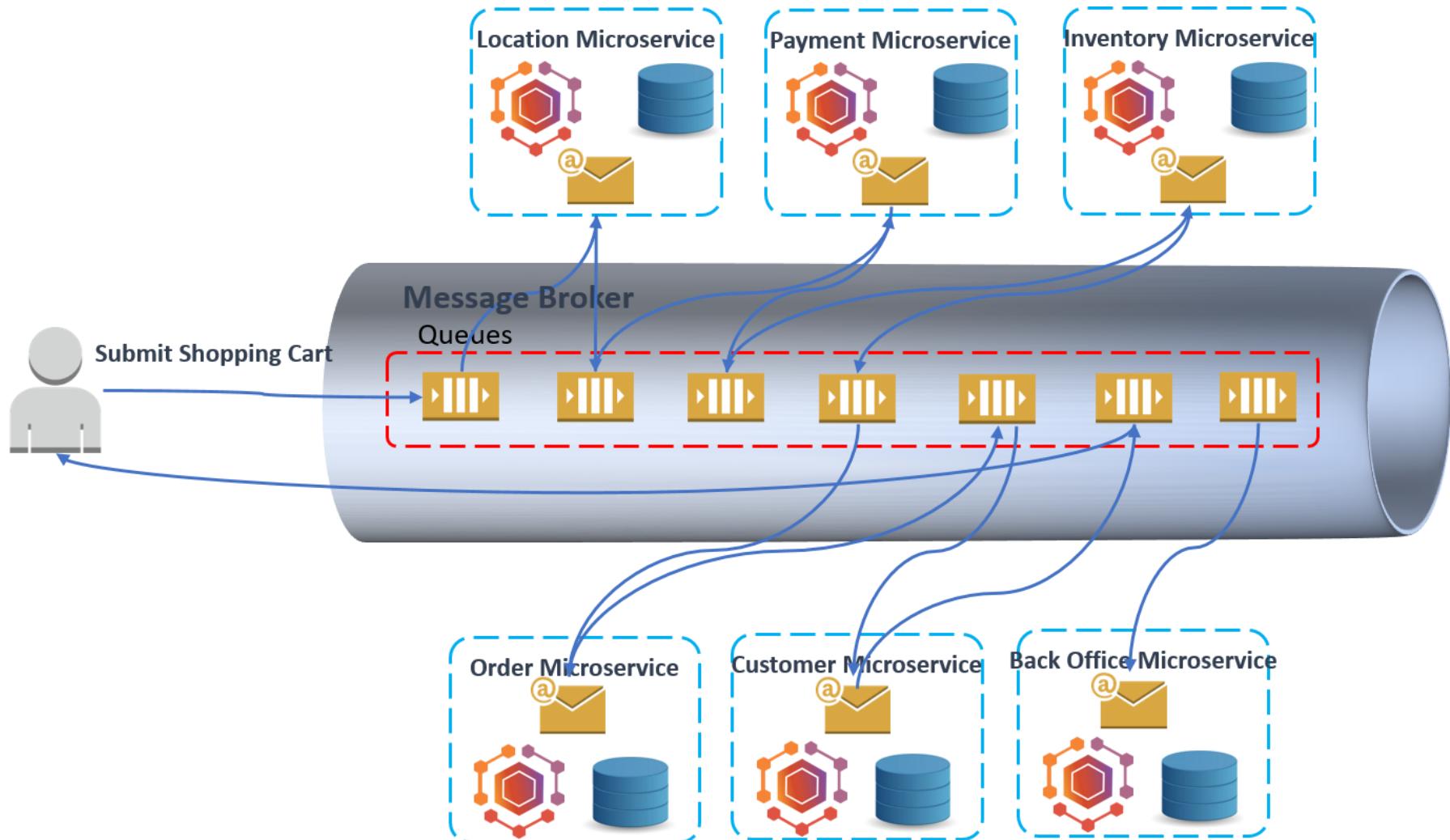


Event Choreography





Event Choreography



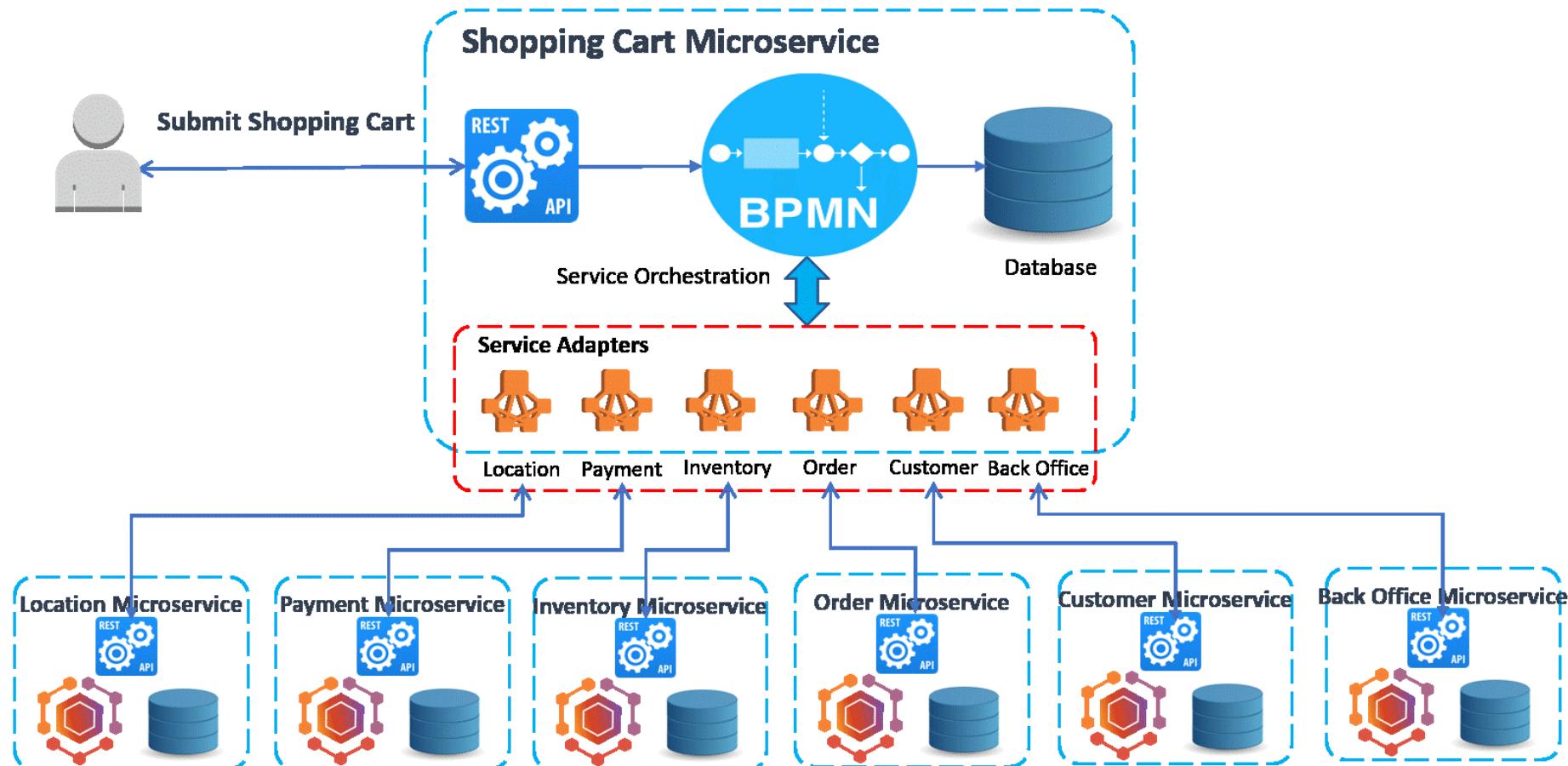
Event-Driven Service Choreography



- 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.



Centralized Service Orchestration





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.

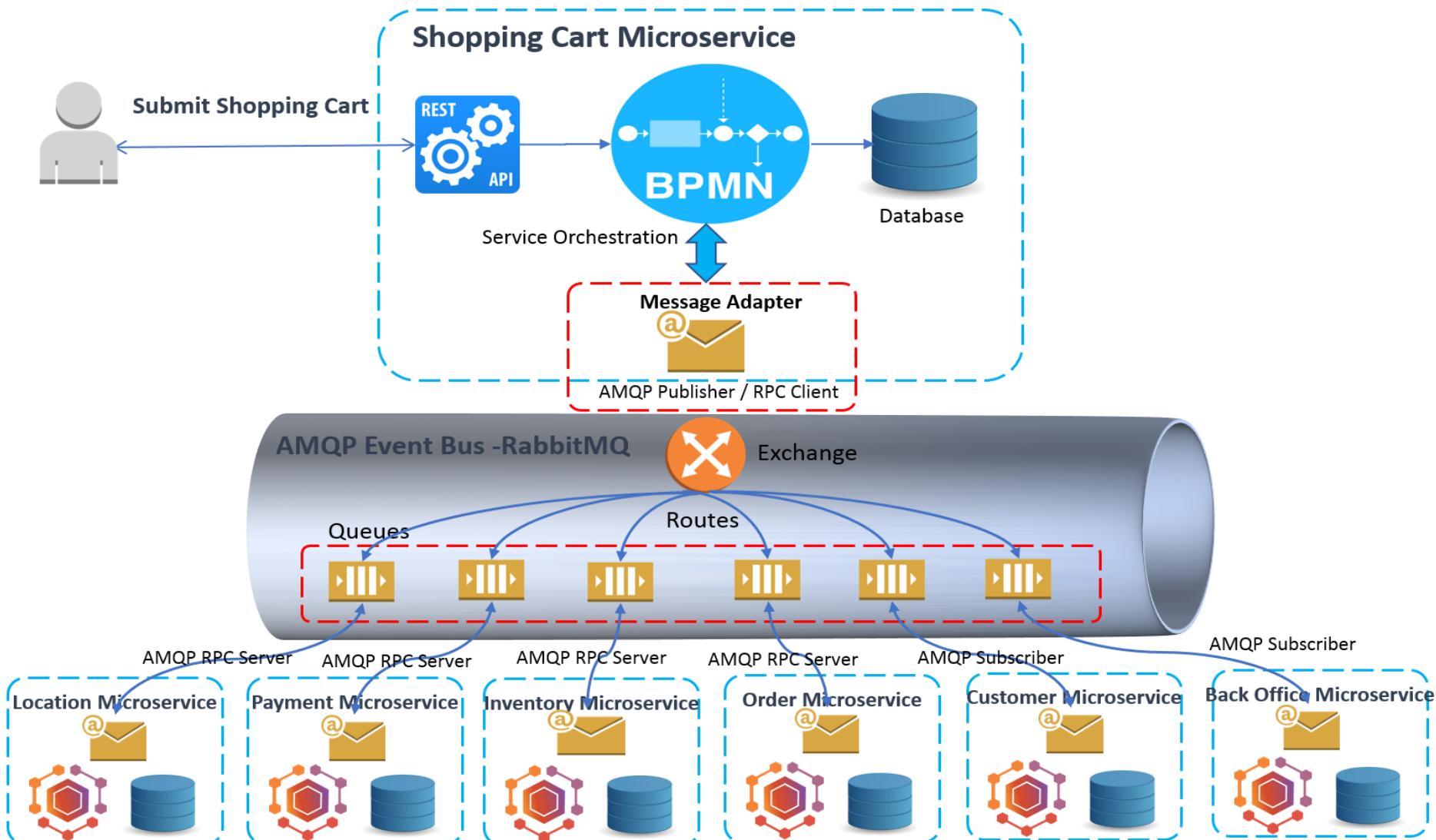


Centralized Service Orchestration

- 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.



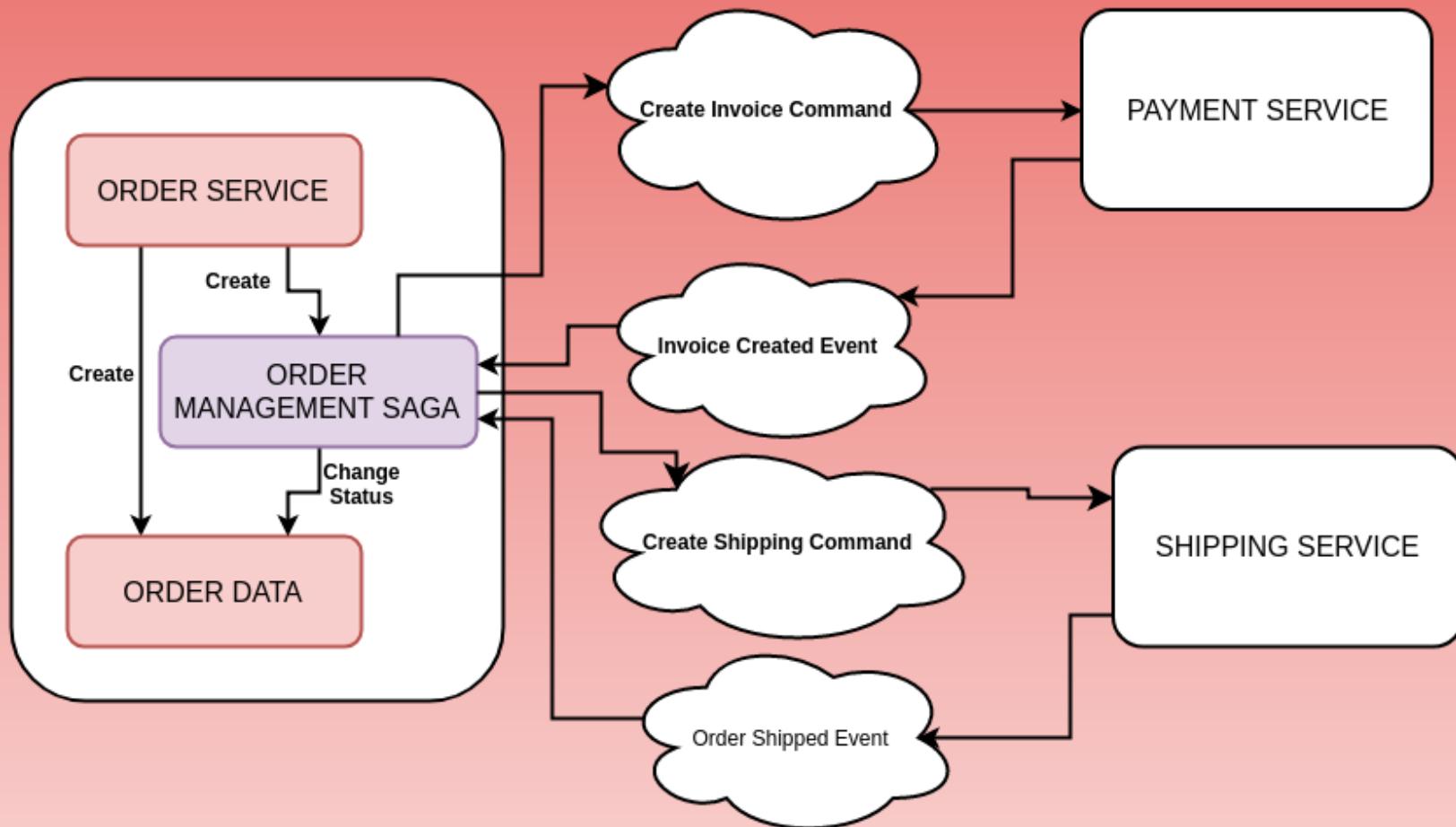
Centralized Service Orchestration





Saga Pattern

ORCHESTRATION BASED SAGA





High-Level Components

- Order Service
- Payment Service
- Shipping Service
- Core-APIs – Integration glue
- Axon Server – Axon Server is part of the Axon Platform.

Observability Pattern



Log Aggregation

- Since microservices are deployed into individual containers, the logs generated by each of the containers (a.k.a pods) need to be aggregated to create a centralized log repository.
- Microservices can log to standard output or to a log file.
- The log management systems such as Splunk or Kibana can aggregate the log stream in real time to a centralized log repository and we can query the real-time logs.



Performance Metrics

- Performance monitoring services such as Prometheus, AppDynamics and NewRelic can be used to monitor the performance metrics of microservices.
- The performance metrics are depicted visually and we can configure the performance thresholds and notification triggers.



Distributed Tracing

- When the request flows across various layers and microservices, it is necessary to trace the request end-to-end for error handling and for performance troubleshooting scenarios.
- In distributed tracking, we create a unique request ID (such as x-request-id) that is passed across all layers and microservices and logged for troubleshooting purposes.



Health Checks

- In order to properly distribute the load and route the traffic accordingly, each microservice has to publish health check endpoint (such as /health) that provides the status of the overall health of the service.
- The health check service should check the status of dependent systems (such as databases, storage systems) and host connectivity to provide the overall health status of the service.

CROSS-CUTTING CONCERN PATTERNS



External Configuration

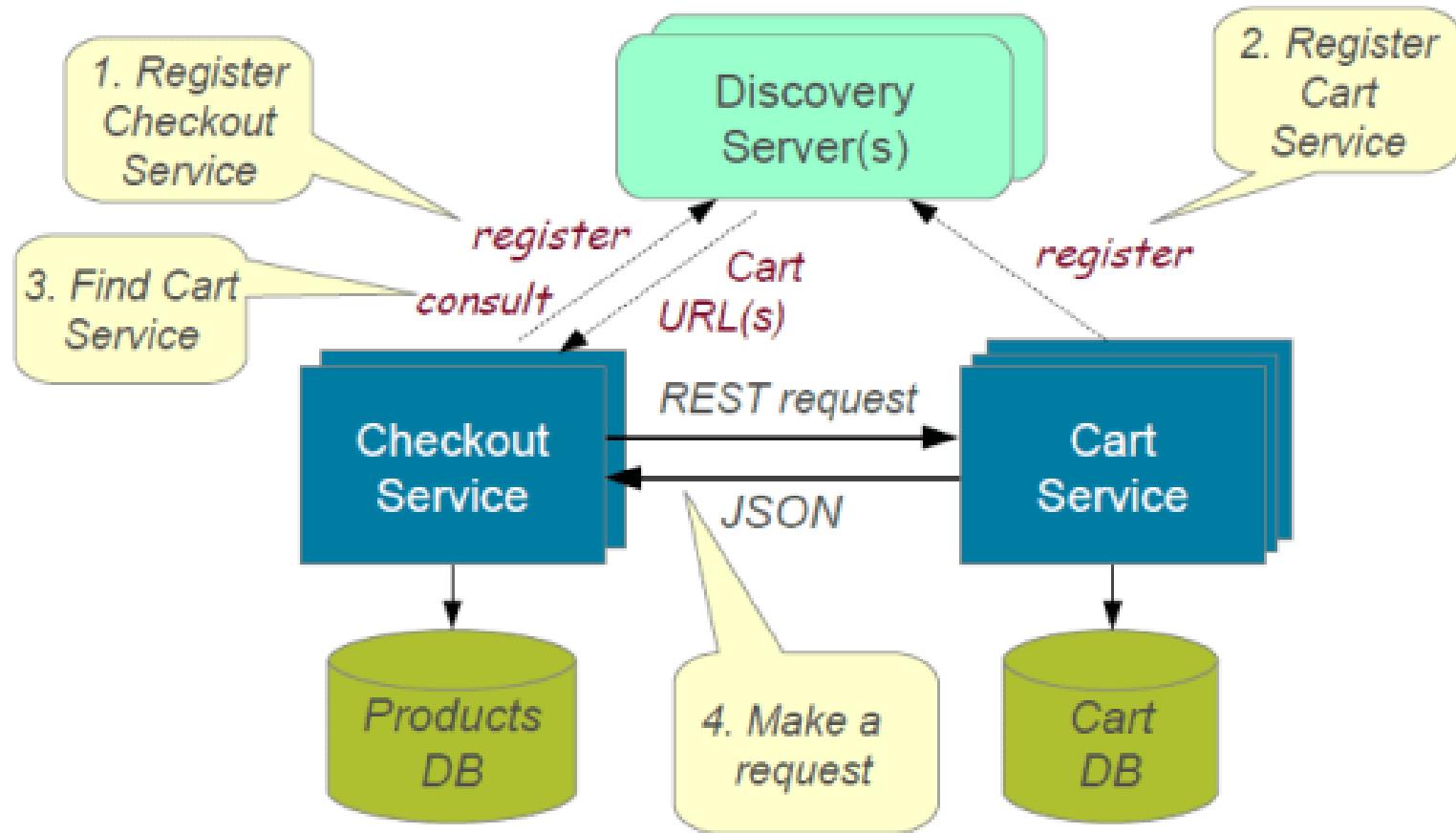
- All environment-specific configurations such as connection strings, application properties and URLs should be loaded from an external configuration file.
- The CI/CD pipeline can inject the environment-specific configuration values during the build.



Service Discovery

- Centralized service discovery module should handle the responsibilities such as service registration/ de-registration and request routing, based on service health.
- For client side service discovery service registry is used for load balancing and for server side service discovery, server side load balancing is used.

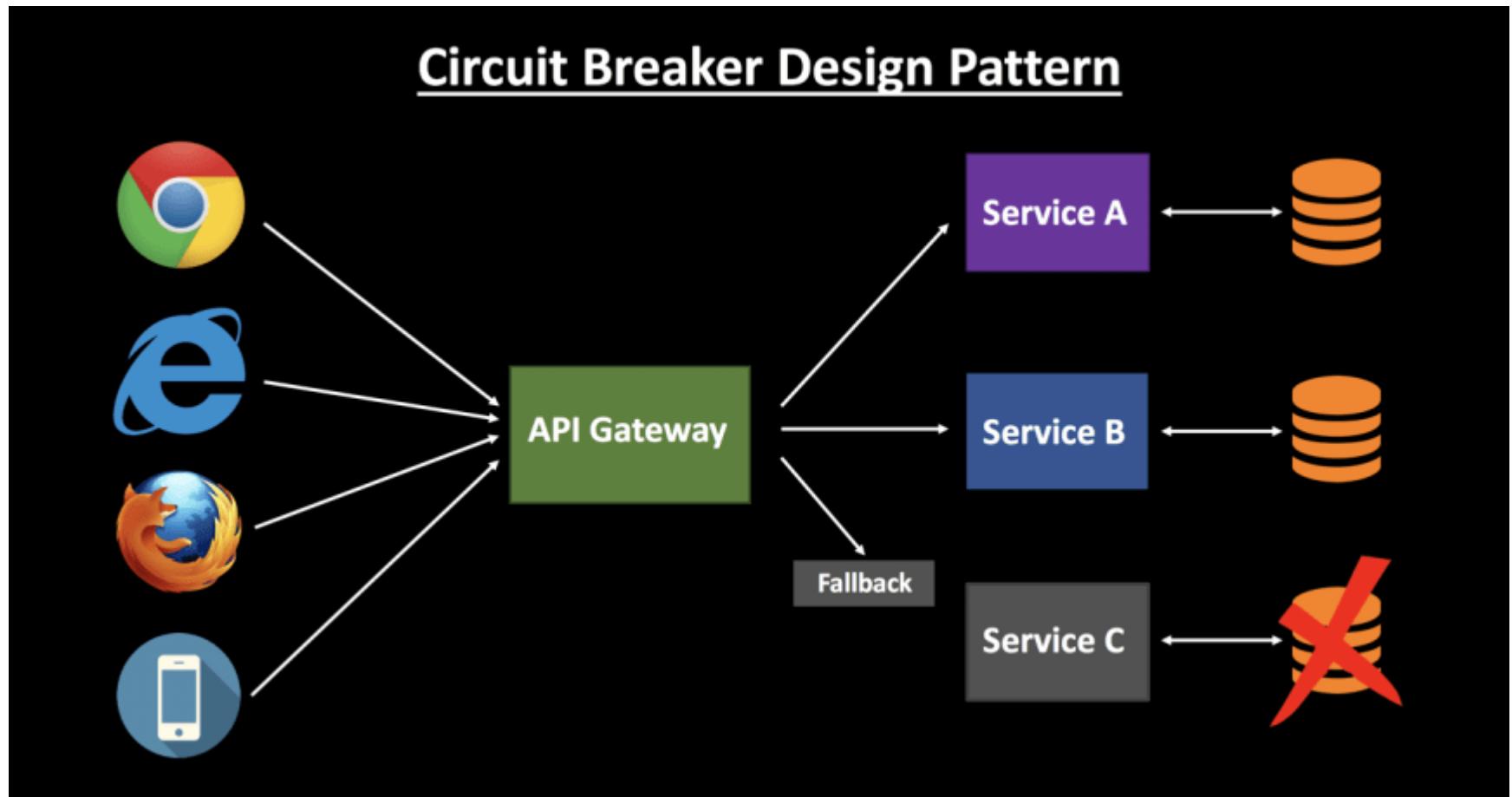
Service Discovery





The circuit breaker

- When one of the services in the request-processing pipeline fails, the circuit breaker is responsible in terms of handling the failure and preventing the cascading of the error.
- The circuit breaker can monitor the error from a dependent service and fallback to a default handler in case of error.
- Netflix Hystrix is an example of a circuit breaker



Solutions to Challenges with Microservice Architectures



- **Spring Boot**
- *Enable building production ready applications quickly*
- Provide non-functional features
- embedded servers (easy deployment with containers)
- metrics (monitoring)
- health checks (monitoring)
- externalized configuration

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



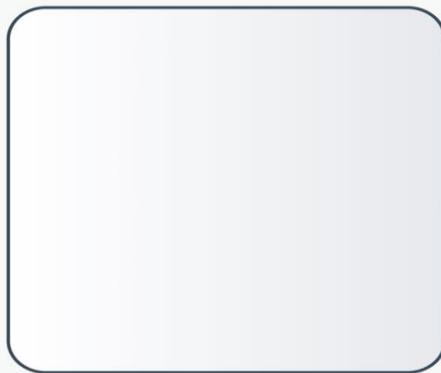
Cons

- 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.



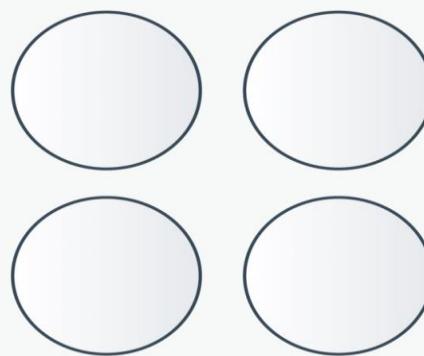
SOA vs Microservices

Monolithic vs. SOA vs. Microservices



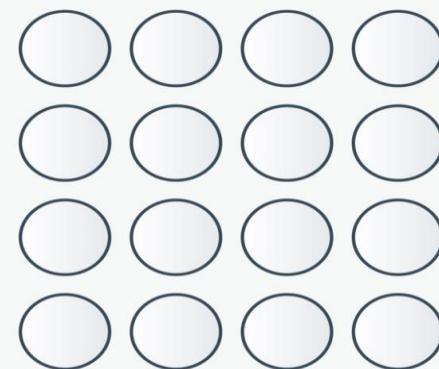
Monolithic

Single Unit



SOA

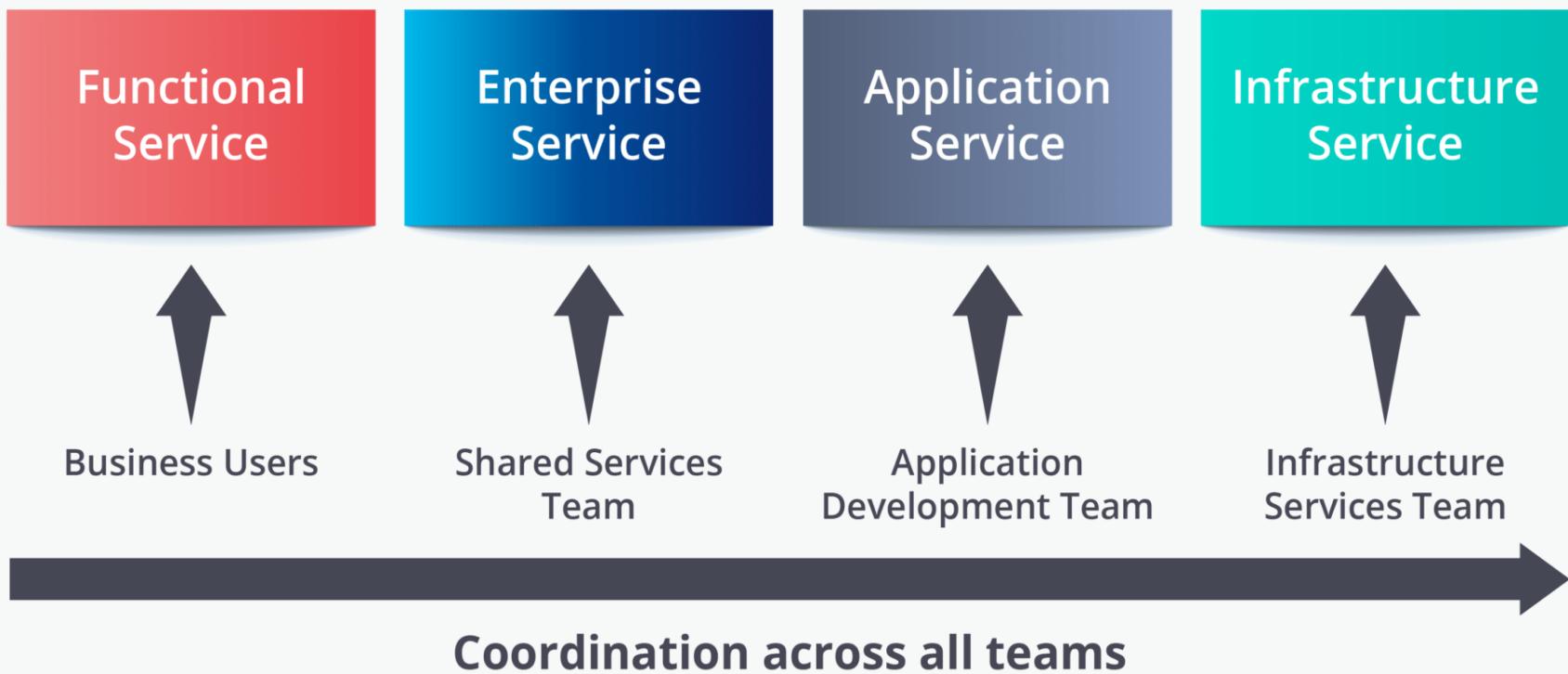
Coarse-grained



Microservices

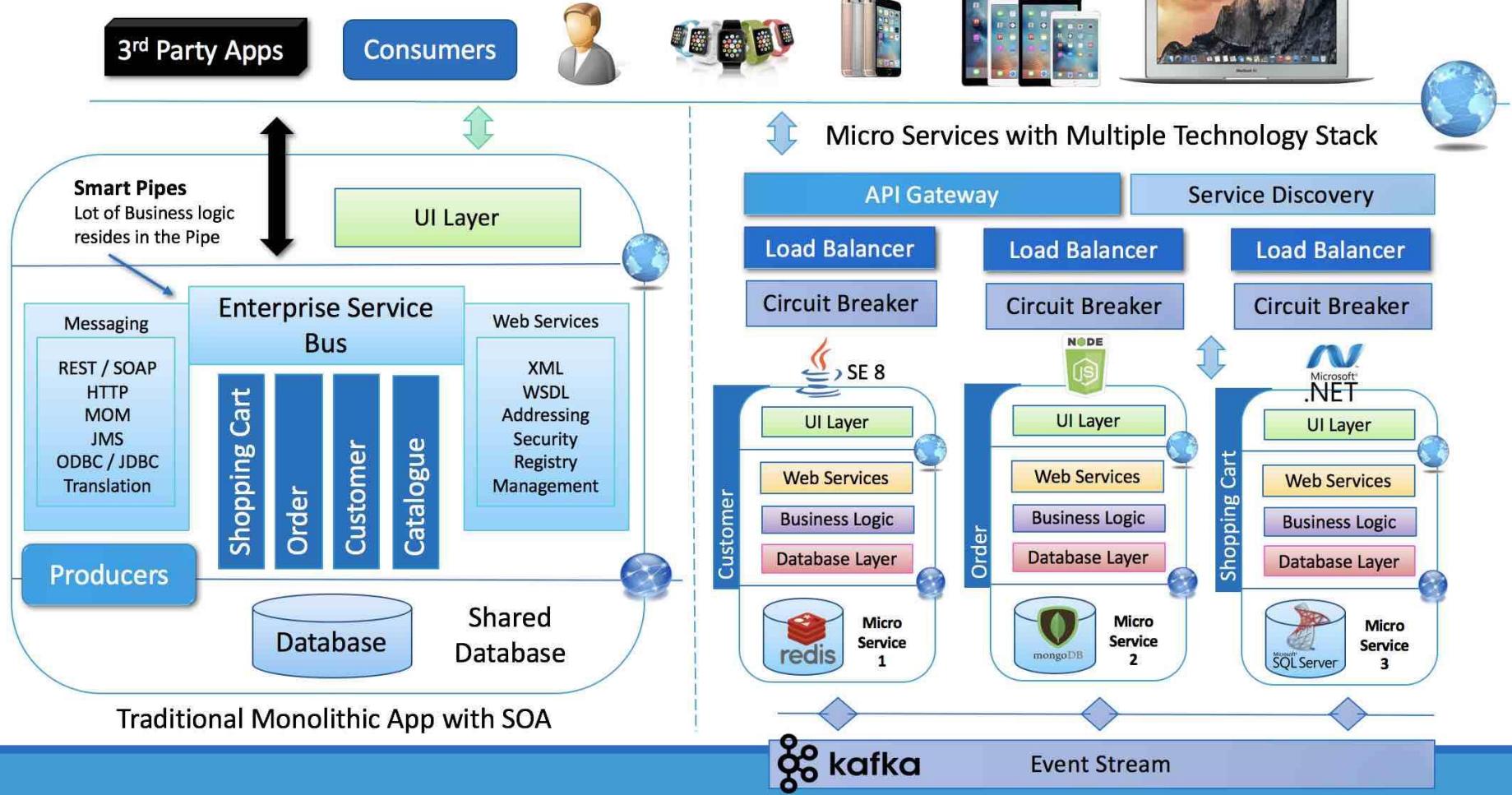
Fine-grained

Service Oriented Architecture





SOA vs. Micro Services Example





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 “ share-as-much-as-possible ” architecture approach	Follows “ share-as-little-as-possible ” architecture approach
Importance is on business functionality reuse	Importance is on the concept of “ bounded context ”
They have common governance and standards	They focus on people collaboration and freedom of other options
Uses Enterprise Service bus (ESB) for communication	Simple messaging system
They support multiple message protocols	They use lightweight protocols such as HTTP/REST etc.
Multi-threaded with more overheads to handle I/O	Single-threaded usually with the use of Event Loop features for non-locking I/O handling
Maximizes application service reusability	Focuses on decoupling
Traditional Relational Databases are more often used	Modern Relational Databases 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

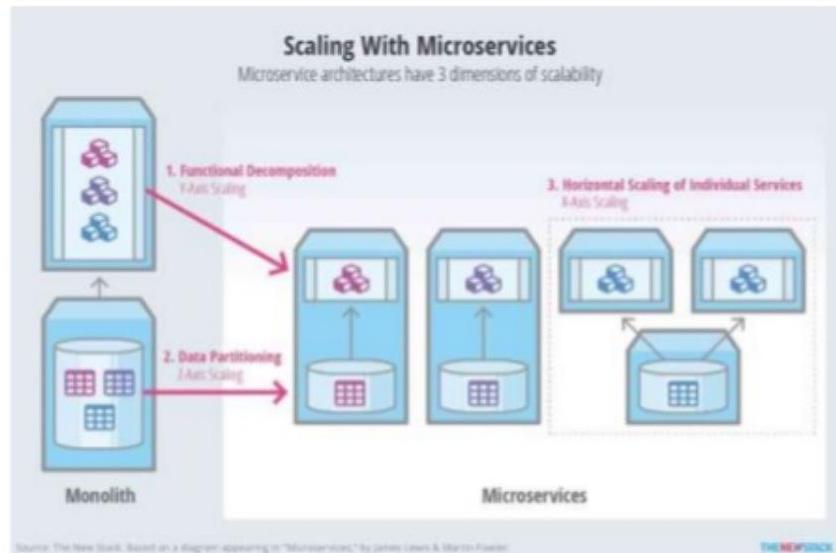
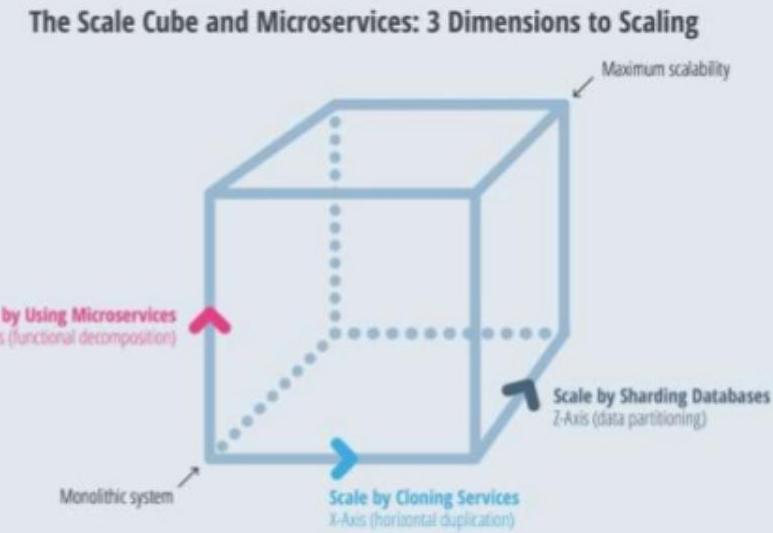
- **Bounded Context** : Deciding the boundaries of a microservice is not an easy task. Bounded Contexts from Domain Driven Design is a good starting point. Your understanding of the domain evolves over a period of time. You need to ensure that the microservice boundaries evolve.
- **Configuration Management** : 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.

Scale Cube and Micro Services



1. Y Axis Scaling – Functional Decomposition : Business Function as a Service
2. Z Axis Scaling – Database Partitioning : Avoid locks by Database Sharding
3. X Axis Scaling – Cloning of Individual Services for Specific Service Scalability

Scaling Challenge

X-AXIS SCALING

Network name: Horizontal scaling, scale out



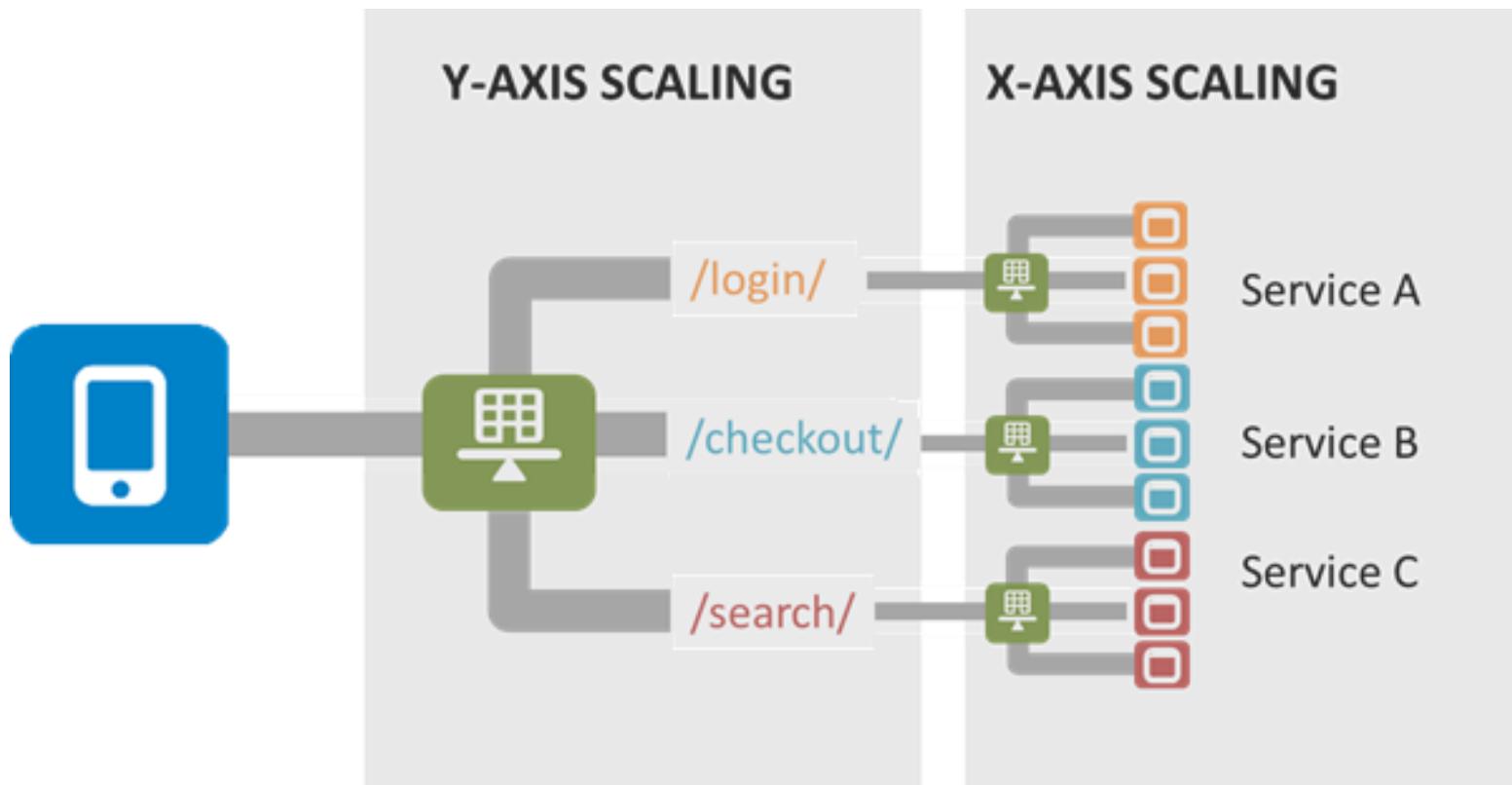
Y-AXIS SCALING

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





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

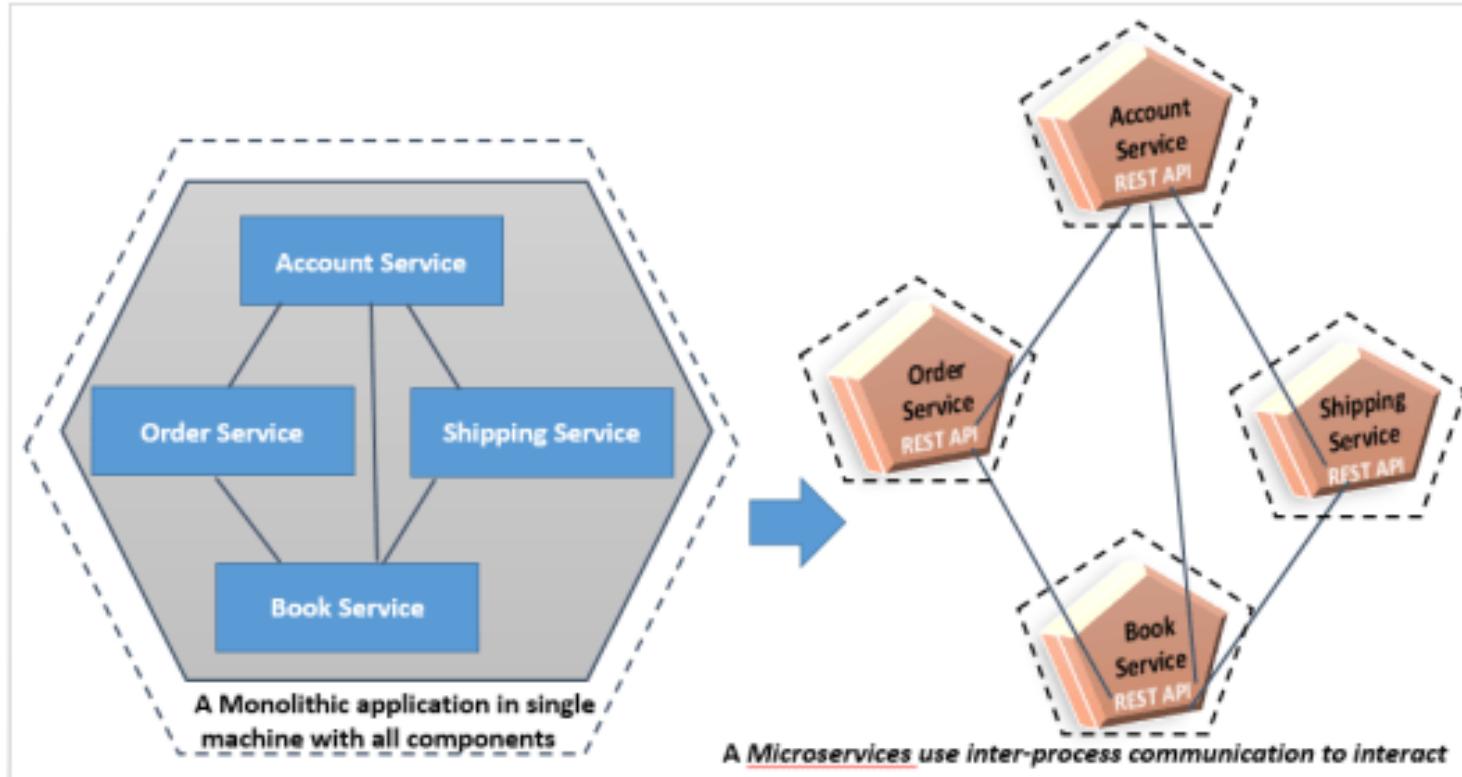
- **5. Data Staleness** — The database should be always updated to give recent data. The API will fetch data from the recent and updated database. A timestamp entry can also be added with each record in the database to check and verify the recent data. Caching can be used and customized with an acceptable 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.

Microservices Inter-Service Communication

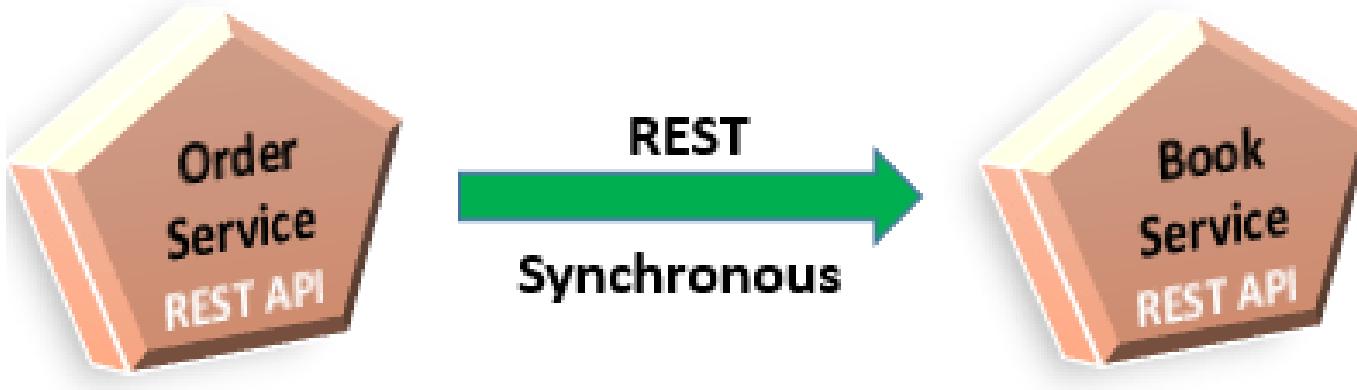


Microservices Inter-Service Communication



- In Microservice Architecture, we can classify our inter-service communication into two approaches like the following:
- Synchronous communication style
- Asynchronous communication style

Synchronous communication style



- The client sends a request to the server and waits for a response from the service (Mostly JSON over HTTP).
- For example, Spring Cloud Netflix provides the most common pattern for synchronous REST communication such as Feign or Hystrix.

Synchronous communication style



- The synchronous communication approach does have some drawbacks, such as timeouts and strong coupling.
- There are numerous protocols, such as REST, gRPC, and Apache Thrift, that can be used to interact with services synchronously.

Synchronous one-to-one communication style

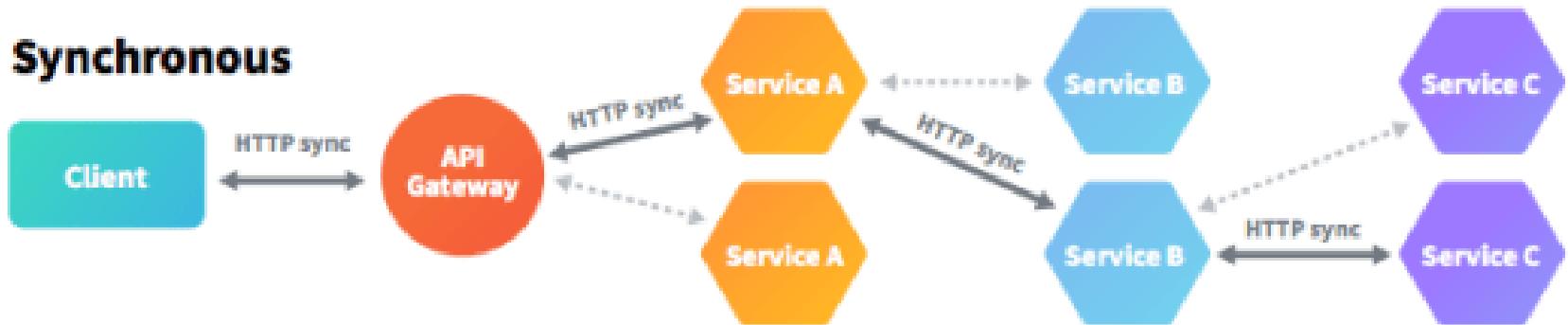


- Most synchronous communications are one-to-one.
- In synchronous one-to-one communication, you can also use multiple instances of a service to scale the service.
- However, if you do, you have to use a load-balancing mechanism on the client side.
- Each service contains meta-information about all instances of the calling service.
- This information is provided by the service discovery server, an example of which is Netflix Eureka.

Synchronous one-to-one communication style



- There are several load-balancing mechanisms you can use.
- One of these is Netflix Ribbon, which carries out load-balancing on the client side, as illustrated in the following diagram:



Synchronous one-to-one communication style



- As you can see in the preceding diagram, we have multiple instances of a particular service, but the services are still communicating one-to-one.
- That means that each service communicates to an instance of another service.
- The load balancer chooses which method should be called.
- The following is a list of some of the most common load-balancing methods available:
 - Round-Robin: This is the simplest method that routes requests across all the instances sequentially.
 - Least Connections: This is a method in which the request goes to the instance that has the fewest number of connections at the time.
 - Weighted Round-Robin: This is an algorithm that assigns a weight to each instance and forwards the connection according to this weight.
 - IP Hash: This is a method that generates a unique hash key from the source IP address and determines which instance receives the request.

Asynchronous communication style



- In this communication style, the client service doesn't wait for the response coming from another service.
- So, the client doesn't block a thread while it is waiting for a response from the server. Such type of communications is possible by using lightweight messaging brokers.
- The message producer service doesn't wait for a response.
- It just generates a message and sends message to the broker.
- It waits for the only acknowledgement from the message broker to know the message has been received by a message broker or not.

Asynchronous communication style



- The Order Service generates a message to A Message Broker and then forgets about it.
- The Book Service that subscribes to a topic is fed with all the messages belonging to that topic.
- The services don't need to know each other at all, they just need to know that messages of a certain type exist with a certain payload.



Asynchronous communication style



- There are various tools to support lightweight messaging, you just choose one of the following message brokers that is delivering your messages to consumers running on respective microservices:
 - RabbitMQ
 - Apache Kafka
 - Apache ActiveMQ
 - NSQ
- The above tools are based on the AMQP (Advanced Message Queuing Protocol).
- This protocol provides messaging based inter-service communication.
- Spring Cloud Stream also provides mechanisms for building message-driven microservices using either the RabbitMQ or the Apache Kafka.

Asynchronous communication style

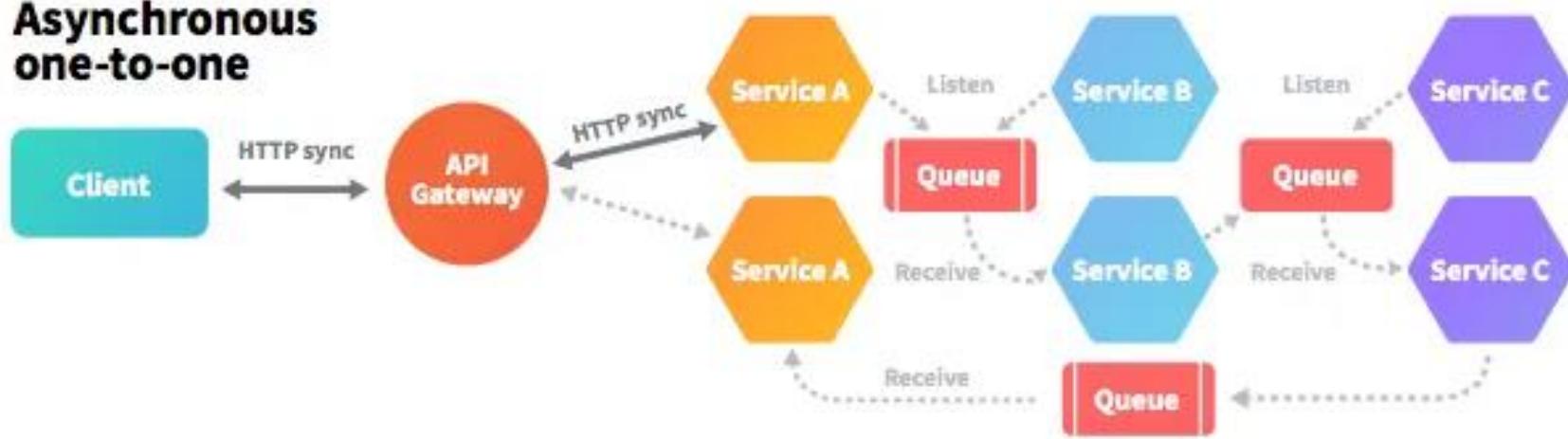


- Asynchronous one-to-one communication style
- Asynchronous one-to-many communication style

Asynchronous one-to-one communication style



Asynchronous one-to-one



Asynchronous one-to-many communication style

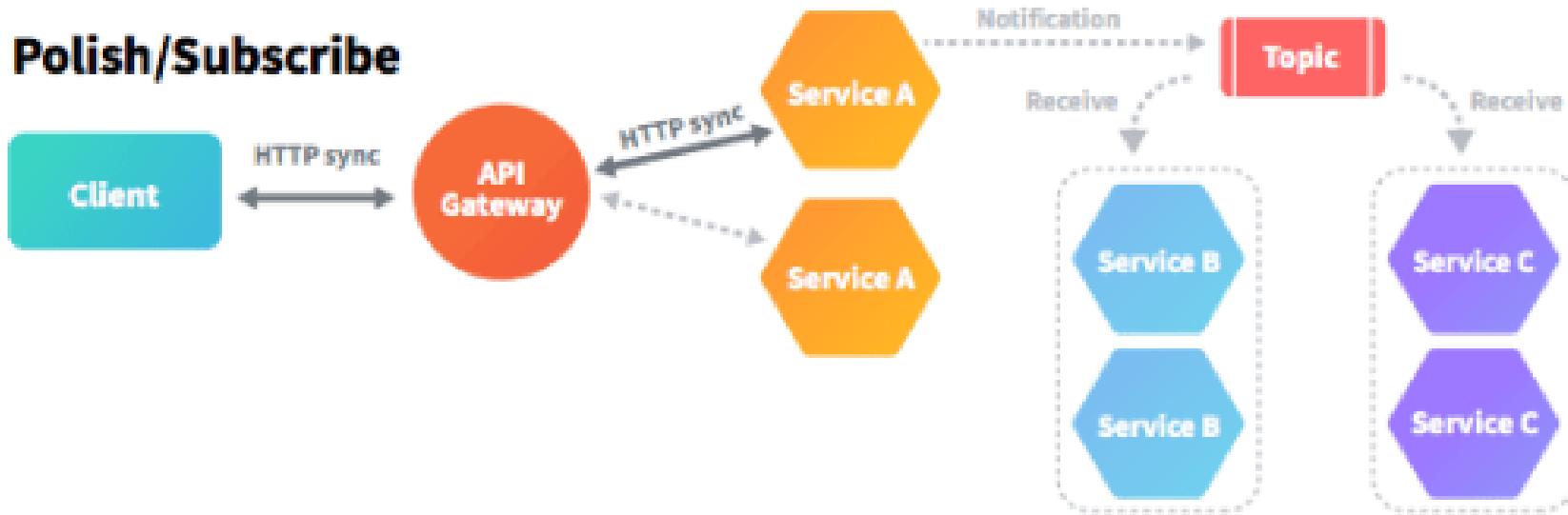


- Publish/subscribe: In this approach, a Client publishes a notification message to the message broker and this notification message is consumed by zero or more interesting services.
- Publish/async responses: In this approach, a Client publishes a request message and then waits for a certain amount of time for responses from interested services

Asynchronous one-to-many communication style



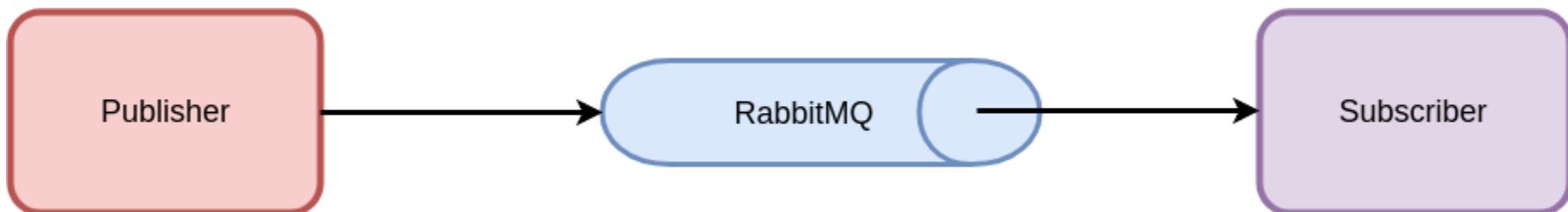
Push/Subscribe



Connecting Microservices Through Messaging



- Spring Cloud Stream
 - It is a framework for building highly scalable event-driven microservices connected with shared messaging systems.
 - The framework provides a flexible programming model built on already established and familiar Spring idioms and best practices, including support for persistent pub/sub semantics, consumer groups, and stateful partitions.





Binder Implementations

- Spring Cloud Stream supports a variety of binder implementations
 - RabbitMQ
 - Apache Kafka
 - Kafka Streams
 - Amazon Kinesis
 - Google PubSub (*partner maintained*)
 - Solace PubSub+ (*partner maintained*)
 - Azure Event Hubs (*partner maintained*)
 - Apache RocketMQ (*partner maintained*)



Core Components of Cloud Stream

- The core building blocks of Spring Cloud Stream are:
- Destination Binders: Components responsible to provide integration with the external messaging systems.
- Destination Bindings: Bridge between the external messaging systems and application provided Producers and Consumers of messages (created by the Destination Binders).
- Message: The canonical data structure used by producers and consumers to communicate with Destination Binders (and thus other applications via external messaging systems).

Docker



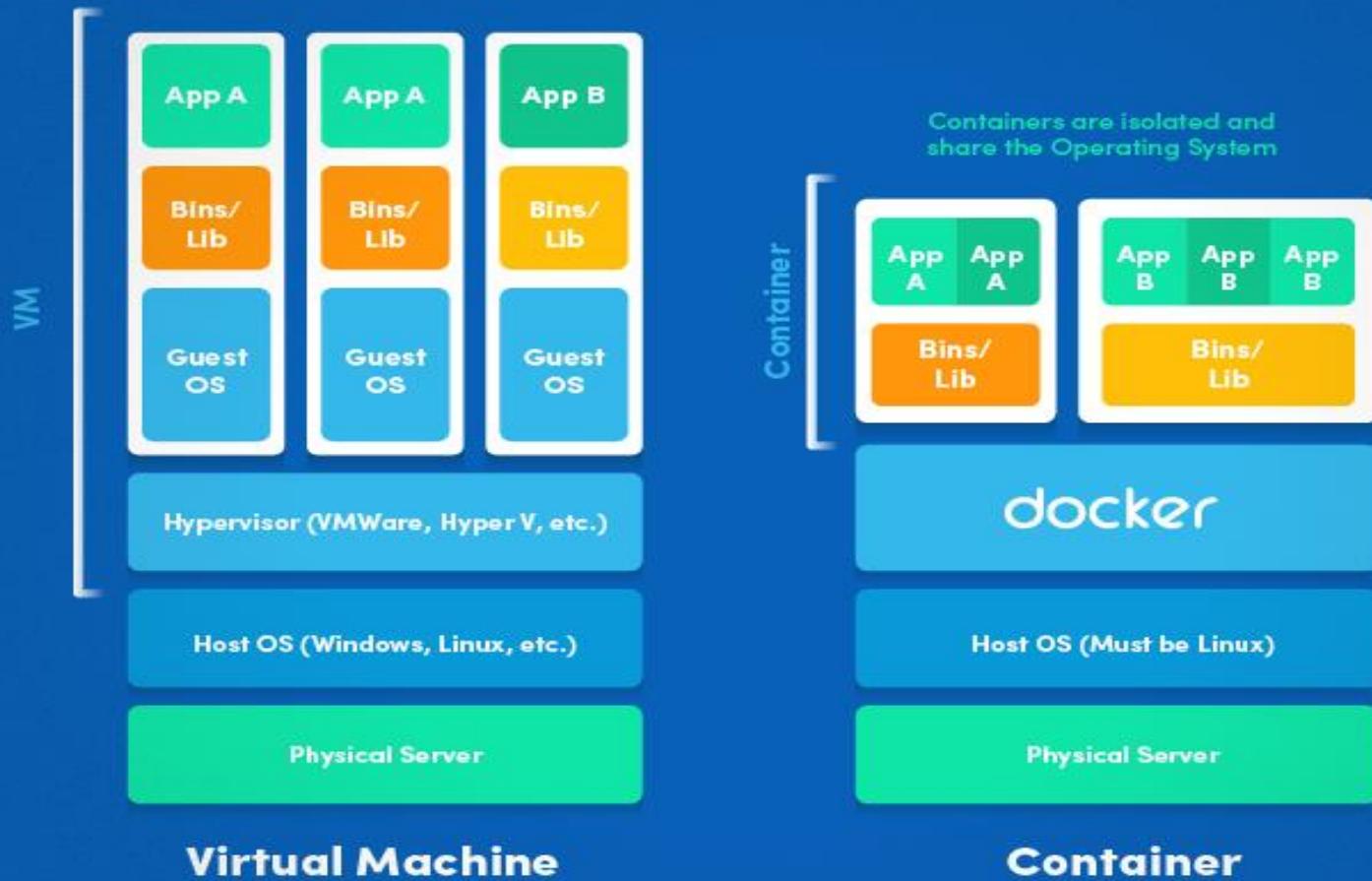
- Docker is an excellent tool for managing and deploying microservices.
- Each microservice can be further broken down into processes running in separate Docker containers, which can be specified with Dockerfiles and Docker Compose configuration files.
- Combined with a provisioning tool such as Kubernetes, 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

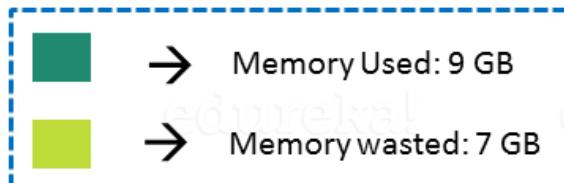
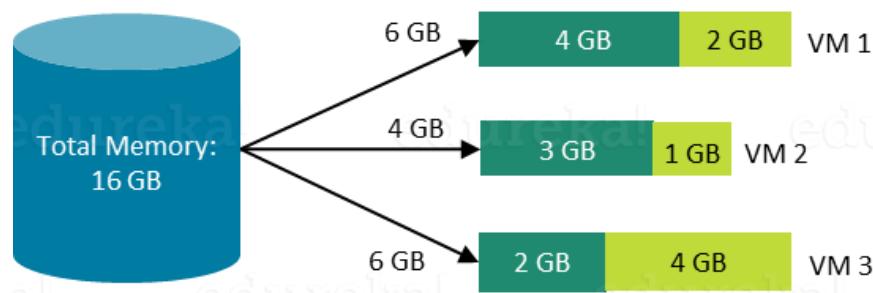


Containers vs. VMs

 toptal

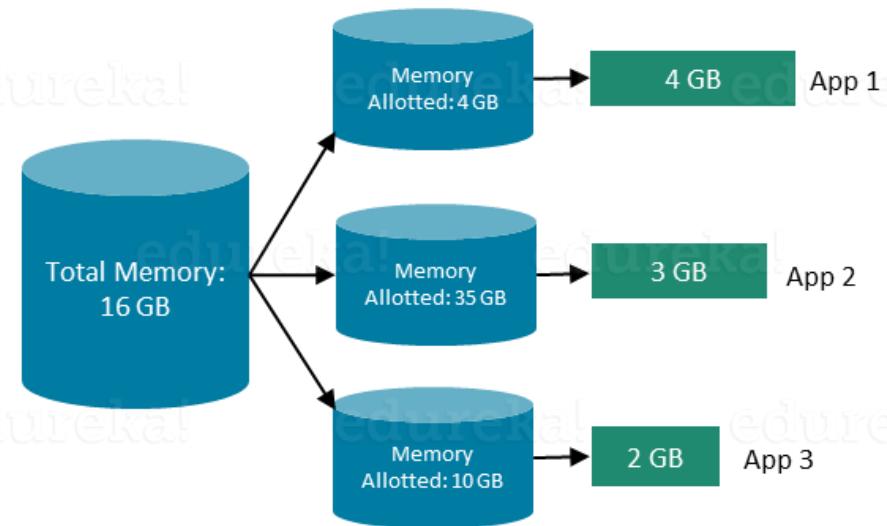


In case of Virtual Machines



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

In case of Docker



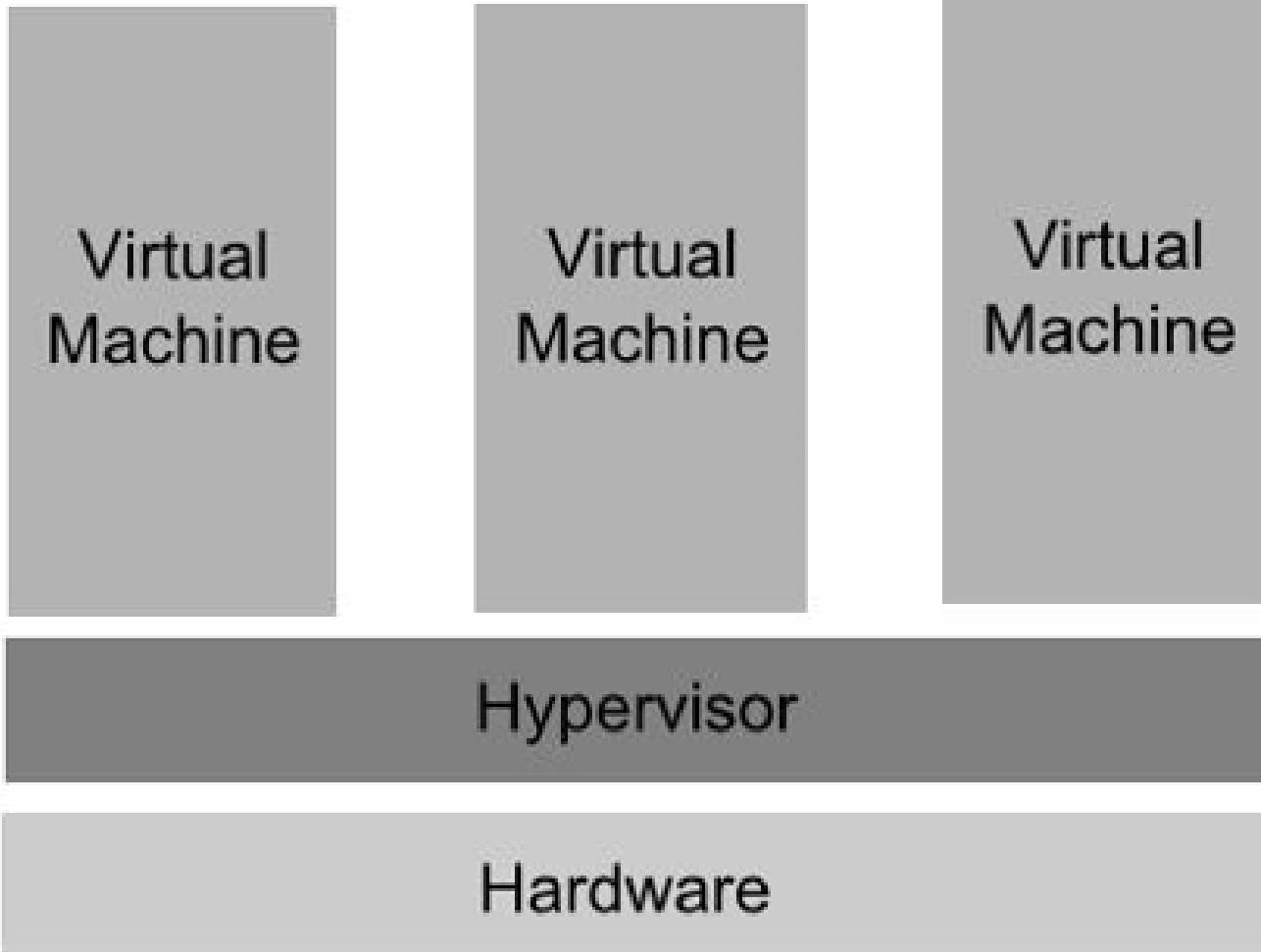
Only 9 GB memory utilized;
7 GB can be allotted to a new Container



Hypervisor

- A hypervisor is a software that makes virtualization possible.
- It is also called Virtual Machine Monitor.
- It divides the host system and allocates the resources to each divided virtual environment.

Hypervisor





Hypervisor

- Hyper-V will only run on processors which support hardware assisted virtualization.
- The x86 family of CPUs provide a range of protection levels also known as rings in which code can execute.
- Ring 0 has the highest level privilege and it is in this ring that the operating system kernel normally runs.
- Code executing in ring 0 is said to be running in system space, kernel mode or supervisor mode.
- All other code, such as applications running on the operating system, operate in less privileged rings, typically ring 3.



Hypervisor

- Under Hyper-V hypervisor virtualization a program known as a hypervisor runs directly on the hardware of the host system in ring 0.
- The task of this hypervisor is to handle tasks such CPU and memory resource allocation for the virtual machines in addition to providing interfaces for higher level administration and monitoring tools.



Hypervisor

- If the hypervisor is going to occupy ring 0 of the CPU, the kernels for any guest operating systems running on the system must run in less privileged CPU rings.
- Unfortunately, most operating system kernels are written explicitly to run in ring 0 for the simple reason that they need to perform tasks that are only available in that ring, such as the ability to execute privileged CPU instructions and directly manipulate memory.



Hypervisor

- One solution to this problem is to modify the guest operating systems, replacing any privileged operations that will only run in ring 0 of the CPU with calls to the hypervisor (known as hypercalls).
- The hypervisor in turn performs the task on behalf of the guest system.

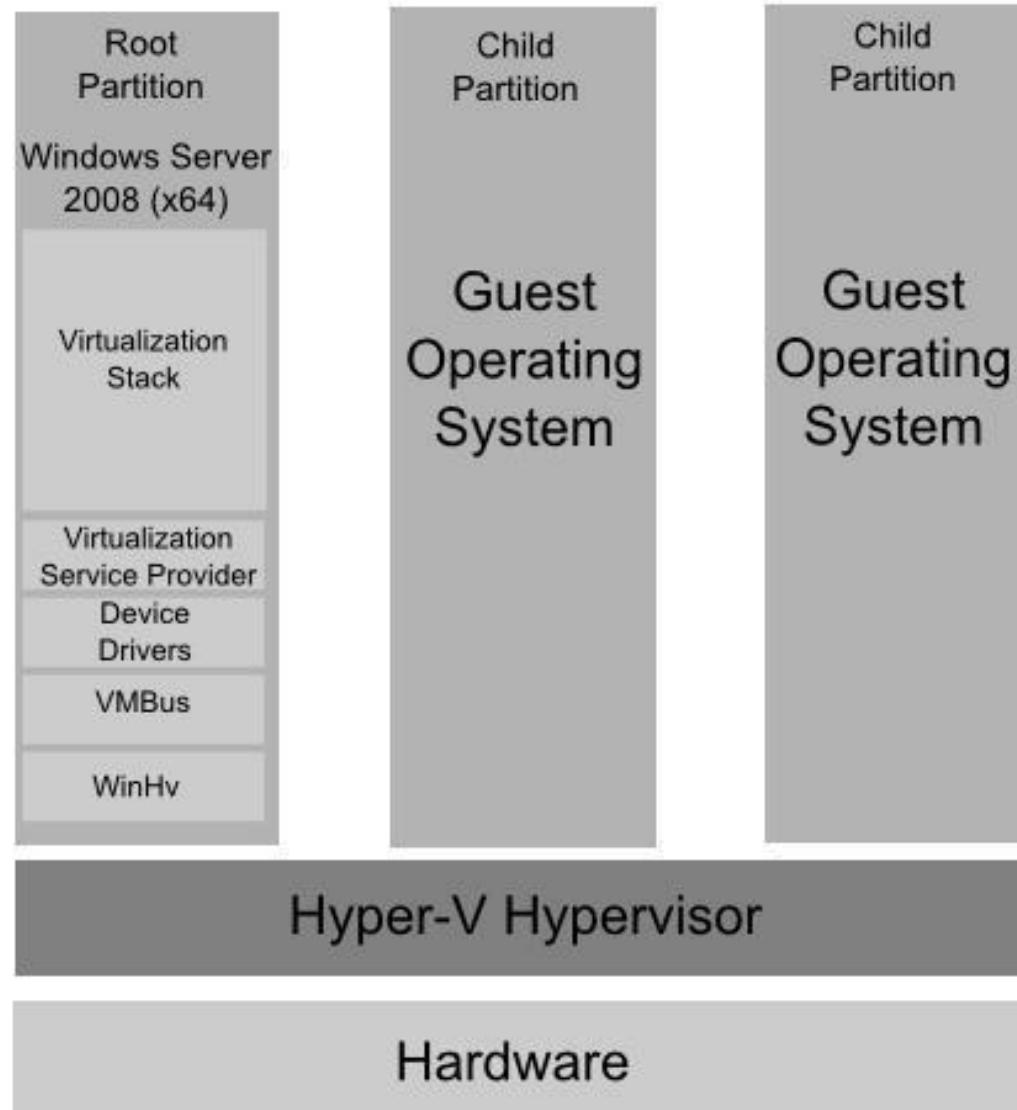


Hypervisor

- Another solution is to leverage the hardware assisted virtualization features of the latest generation of processors from both Intel and AMD.
- These technologies, known as Intel VT and AMD-V respectively, provide extensions necessary to run unmodified guest virtual machines.
- These new processors provide an additional privilege mode (referred to as ring -1) above ring 0 in which the hypervisor can operate, essentially leaving ring 0 available for unmodified guest operating systems.



Hyper-V Root and Child Partitions

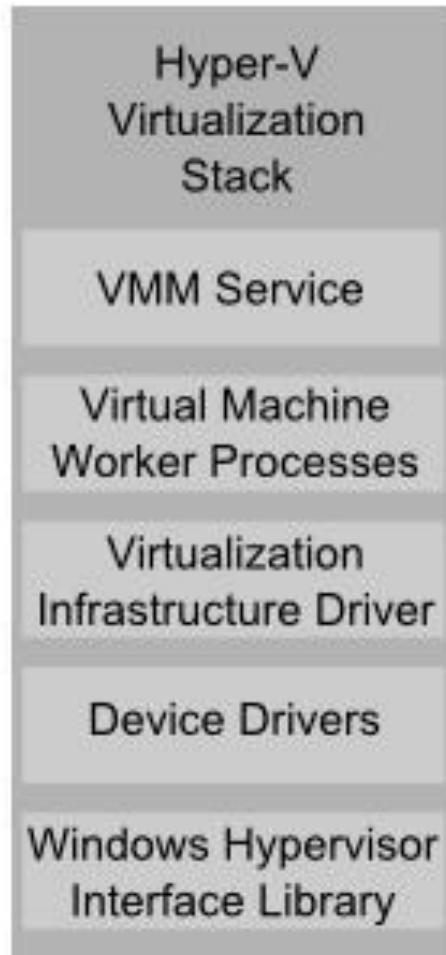




Hyper-V Root and Child Partitions

- The root partition is essentially a virtual machine which runs a copy of 64-bit Windows Server 2008 which, in turn, acts as a host for a number of special Hyper-V components.
- The root partition is responsible for providing the device drivers for the virtual machines running in the child partitions, managing the child partition lifecycles, power management and event logging.
- The root partition operating system also hosts the Virtualization Stack which is responsible for performing a wide range of virtualization functions.
- Child partitions host the virtual machines in which the guest operating systems run. Hyper-V supports both Hyper-V Aware (also referred to as enlightened) and Hyper-V Unaware guest operating systems.

The Virtualization Stack and Other Root Partition Components



The Virtualization Stack and Other Root Partition Components



Component	Description
Virtual Machine Management Service (VMM Service)	<p>Manages the state of virtual machines running in the child partitions (active, offline, stopped etc) and controls the tasks that can be performed on a virtual machine based on current state (such as taking snapshots). Also manages the addition and removal of devices. When a virtual machine is started, the VMM Service is also responsible for creating a corresponding <i>Virtual Machine Worker Process</i>.</p>
Virtual Machine Worker Process	<p>Virtual Machine Worker Processes are started by the VMM Service when virtual machines are started. A Virtual Machine Worker Process (named vmwp.exe) is created for each Hyper-V virtual machine and is responsible for much of the management level interaction between the parent partition Windows Server 2008 system and the virtual machines in the child partitions. The duties of the Virtual Machine Worker Process include creating, configuring, running, pausing, resuming, saving, restoring and snapshotting the associated virtual machine. It also handles IRQs, memory and I/O port mapping through a <i>Virtual Motherboard (VMB)</i>.</p>

The Virtualization Stack and Other Root Partition Components



Component	Description
Virtual Devices	<p>Virtual Devices are managed by the Virtual Motherboard (VMB). Virtual Motherboards are contained within the Virtual Machine Worker Processes, of which there is one for each virtual machine. Virtual Devices fall into two categories, <i>Core VDevs</i> and <i>Plug-in VDevs</i>. Core VDevs can either be <i>Emulated Devices</i> or <i>Synthetic Devices</i>.</p>
Virtual Infrastructure Driver	<p>Operates in kernel mode (i.e. in the privileged CPU ring) and provides partition, memory and processor management for the virtual machines running in the child partitions. The Virtual Infrastructure Driver (Vid.sys) also provides the conduit for the components higher up the Virtualization Stack to communicate with the hypervisor.</p>

The Virtualization Stack and Other Root Partition Components



Component	Description
Windows Hypervisor Interface Library	<p>A DLL (named WinHv.sys) located in the parent partition Windows Server 2008 instance and any guest operating systems which are <i>Hyper-V aware</i> (in other words modified specifically to operate in a Hyper-V child partition). Allows the operating system's drivers to access the hypervisor using standard Windows API calls instead of hypercalls.</p>
VMBus	<p>Part of Hyper-V Integration Services, the VMBus facilitates highly optimized communication between child partitions and the parent partition</p>

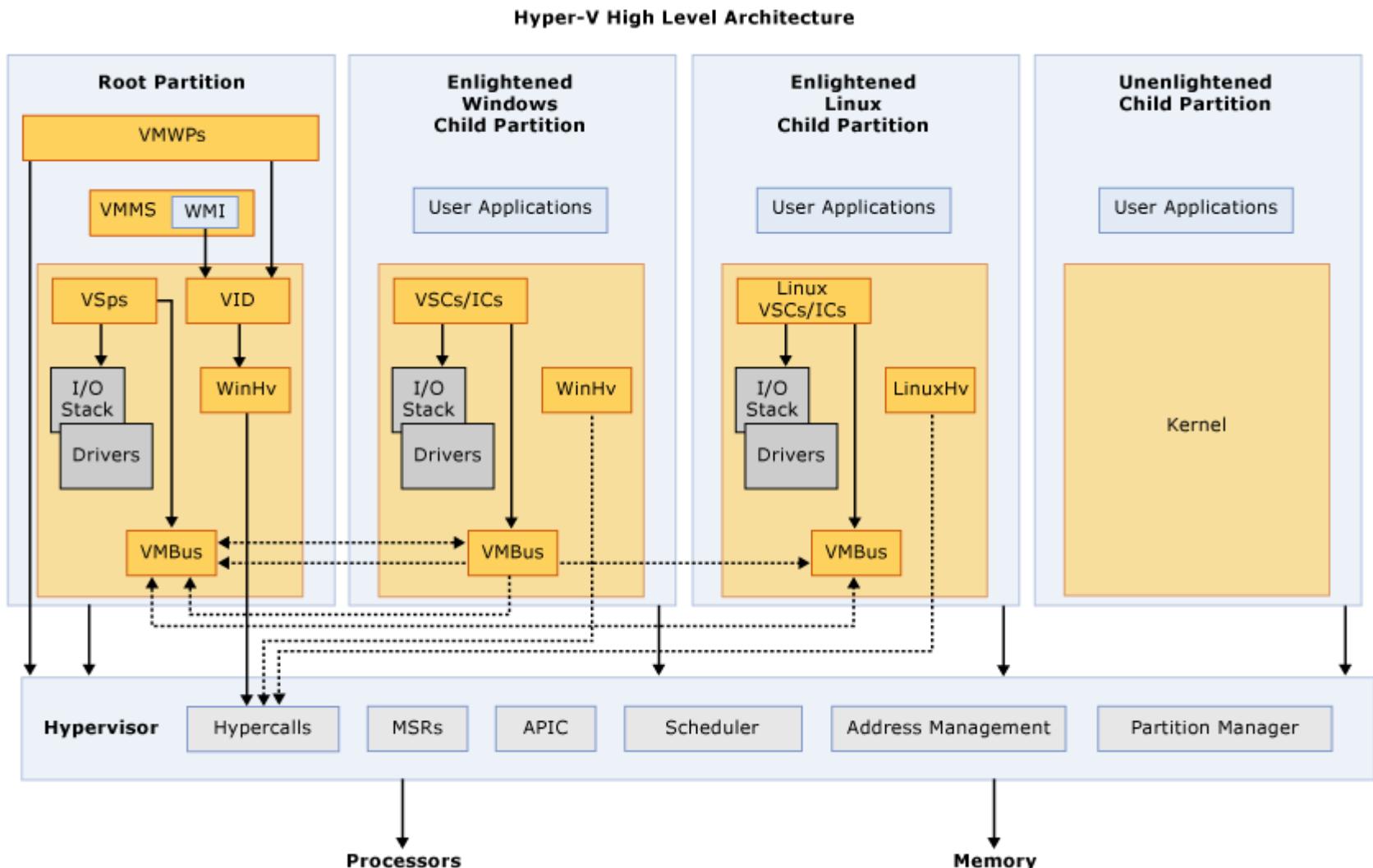
The Virtualization Stack and Other Root Partition Components



Component	Description
Virtualization Service Providers	Resides in the parent partition and provides synthetic device support via the VMBus to Virtual Service Clients (VSCs) running in child partitions.
Virtualization Service Clients	Virtualization Service Clients are synthetic device instances that reside in child partitions. They communicate with the VSPs in the parent partition over the VMBus to fulfill the child partition's device access requests.



Hyper-V Architecture





Hyper-V Architecture

- APIC – Advanced Programmable Interrupt Controller – A device which allows priority levels to be assigned to its interrupt outputs.
- Child Partition – Partition that hosts a guest operating system - All access to physical memory and devices by a child partition is provided via the Virtual Machine Bus (VMBus) or the hypervisor.
- Hypercall – Interface for communication with the hypervisor - The hypercall interface accommodates access to the optimizations provided by the hypervisor.



Hyper-V Architecture

- Hypervisor – A layer of software that sits between the hardware and one or more operating systems. Its primary job is to provide isolated execution environments called partitions. The hypervisor controls and arbitrates access to the underlying hardware.
- IC – Integration component – Component that allows child partitions to communicate with other partitions and the hypervisor.
- I/O stack – Input/output stack
- MSR – Memory Service Routine



Hyper-V Architecture

- Root Partition – Sometimes called parent partition. Manages machine-level functions such as device drivers, power management, and device hot addition/removal. The root (or parent) partition is the only partition that has direct access to physical memory and devices.
- VID – Virtualization Infrastructure Driver – Provides partition management services, virtual processor management services, and memory management services for partitions.



Hyper-V Architecture

- VMBus – Channel-based communication mechanism used for inter-partition communication and device enumeration on systems with multiple active virtualized partitions. The VMBus is installed with Hyper-V Integration Services.
- VMMS – Virtual Machine Management Service – Responsible for managing the state of all virtual machines in child partitions.
- VMWP – Virtual Machine Worker Process – A user mode component of the virtualization stack.
- The worker process provides virtual machine management services from the Windows Server 2008 instance in the parent partition to the guest operating systems in the child partitions.
- The Virtual Machine Management Service spawns a separate worker process for each running virtual machine.



Hyper-V Architecture

- VSC – Virtualization Service Client – A synthetic device instance that resides in a child partition.
- VSCs utilize hardware resources that are provided by Virtualization Service Providers (VSPs) in the parent partition.
- They communicate with the corresponding VSPs in the parent partition over the VMBus to satisfy a child partitions device I/O requests.
- VSP – Virtualization Service Provider – Resides in the root partition and provide synthetic device support to child partitions over the Virtual Machine Bus (VMBus).



Hyper-V Architecture

- WinHv – Windows Hypervisor Interface Library - WinHv is essentially a bridge between a partitioned operating system's drivers and the hypervisor which allows drivers to call the hypervisor using standard Windows calling conventions
- WMI – The Virtual Machine Management Service exposes a set of Windows Management Instrumentation (WMI)-based APIs for managing and controlling virtual machines.



hyper-v containers

- Windows Hyper-v container is a windows server container that runs in a VM.
- Every hyper-v container creates its own VM.
- This means that there is no kernel sharing between the different hyper-v containers.
- This is useful for cases where additional level of isolation is needed by customers who don't like the traditional kernel sharing done by containers.
- The same Docker image and CLI can be used to manage hyper-v containers.
- Creation of hyper-v containers is specified as a runtime option.
- There is no difference when building or managing containers between windows server and hyper-v container.
- Startup times for hyper-v container is higher than windows native container since a new lightweight VM gets created each time.



hyper-v containers

- Creation of hyper-v containers is specified as a runtime option.
- There is no difference when building or managing containers between windows server and hyper-v container.
- Startup times for hyper-v container is higher than windows native container since a new lightweight VM gets created each time.



hyper-v containers

There are 2 modes of hyper-v container.

1. Windows hyper-v container – Here, hyper-v container runs on top of Windows kernel. Only Windows containers can be run in this mode.
2. Linux hyper-v container – Here, hyper-v container runs on top of Linux kernel. This mode was not available earlier and it was introduced as part of Dockercon 2017. Any Linux flavor can be used as the base kernel. Docker's Linuxkit project can be used to build the Linux kernel needed for the hyper-v container. Only Linux containers can be run in this mode.

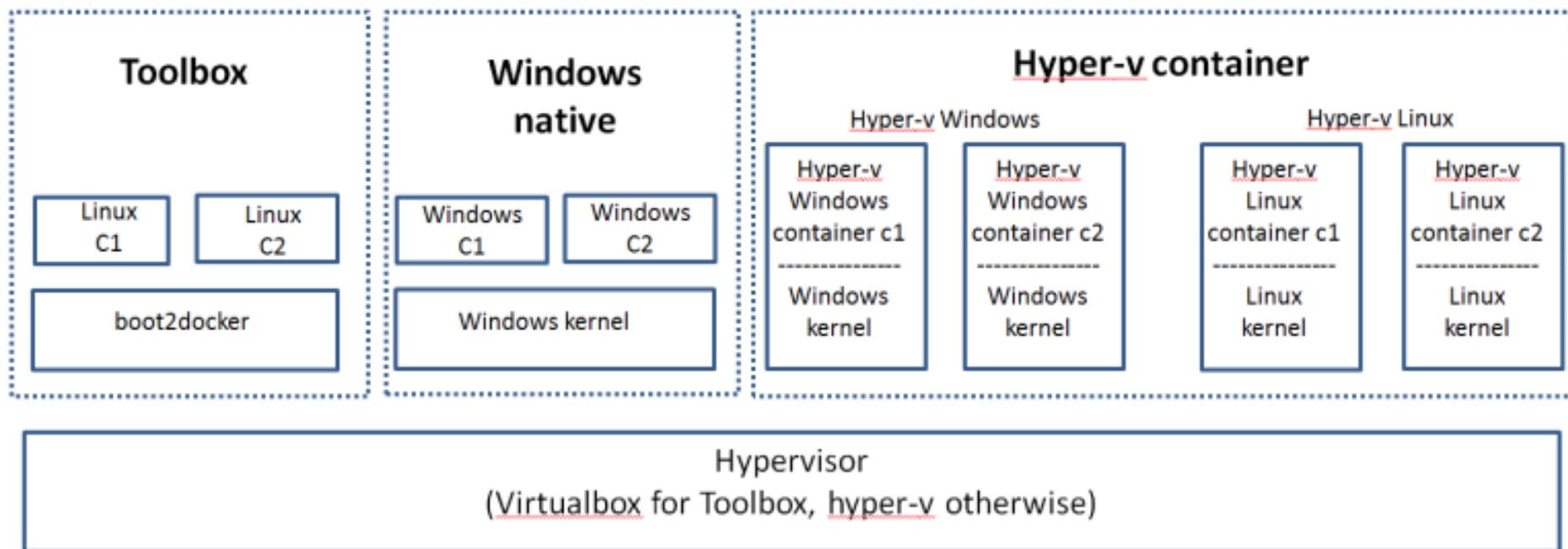


hyper-v containers

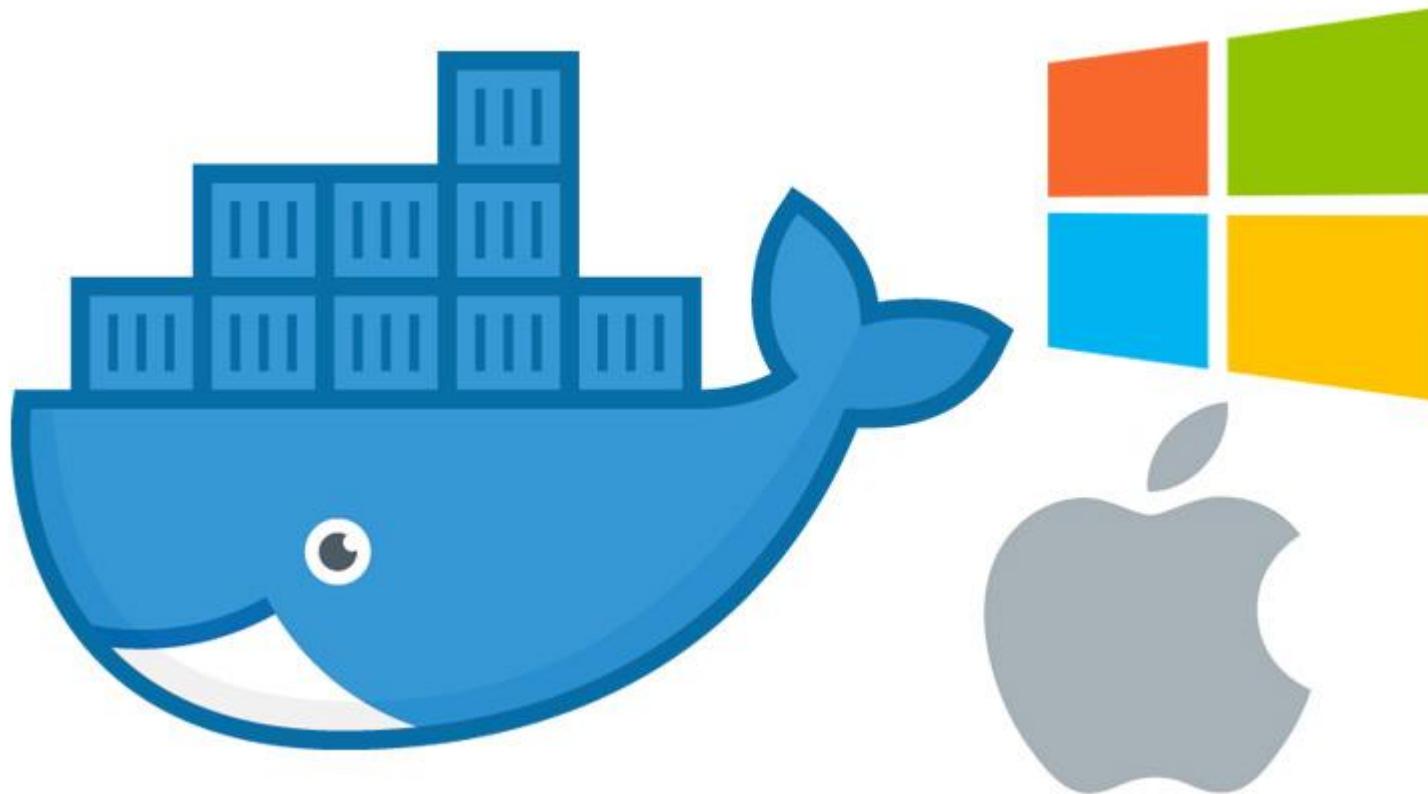
1. We cannot use Docker Toolbox and hyper-v containers at the same time. Virtualbox cannot run when “Docker for Windows” is installed.

2. <https://nickjanetakis.com/blog/should-you-use-the-docker-toolbox-or-docker-for-mac-windows>

Docker



Should You Install Docker with the Docker Toolbox or Docker for Mac / Windows?



Should You Install Docker with the Docker Toolbox or Docker for Mac / Windows?



- If you're on MacOS or Windows you can install Docker with:
 - Docker for Mac / Windows (now known as Docker Desktop)
 - Docker Toolbox
 - Running your own Virtual Machine and installing Docker yourself

Should You Install Docker with the Docker Toolbox or Docker for Mac / Windows?



- If you're on MacOS or Windows you can install Docker with:
 - Docker for Mac / Windows (now known as Docker Desktop)
 - Docker Toolbox
 - Running your own Virtual Machine and installing Docker yourself

Docker for Mac / Docker for Windows (Docker Desktop)



- Pros
- Offers the most “native” experience, you can easily use any terminal you want since Docker is effectively running on localhost from MacOS / Windows’ POV.
- Docker is heavily developing and polishing this solution.

Docker for Mac / Docker for Windows (Docker Desktop)



- **Cons**
- On certain MacOS hardware combos the volume performance can be a little slow.
- I can legit say there are not any “wow this sucks!” cons for Windows, it’s really solid.



Docker Toolbox

- Pros
- Offers an “out of the box” Docker experience if you have no other choice.



Docker Toolbox

- Cons
- You need to either use the Docker Quickstart Terminal, or configure your own terminal to connect to the Docker Daemon running a VM.
- Not a native solution, so you'll need to access your Docker Machine's IP address if you're developing web apps. Example: 192.168.99.100 instead of localhost.



Docker Toolbox

- Cons
- Unless you jump through hoops, your code needs to live in your Windows user directory such as C:\Users\eswari\src\myapp. Otherwise Docker won't be able to find it.
- Suffers from typical VirtualBox edge case bugs and mount performance issues.



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.

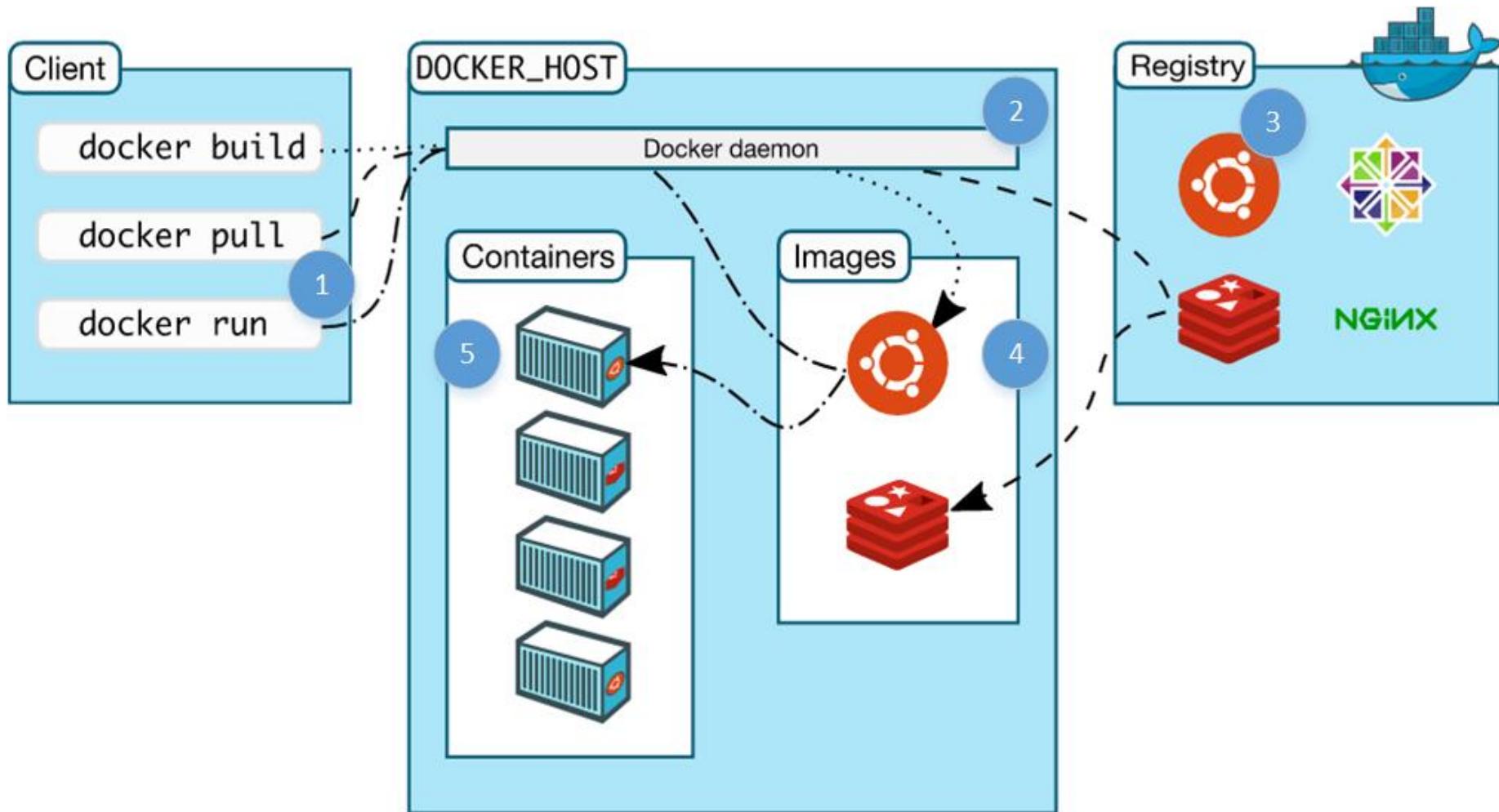


Difference between VM and Container

- 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.

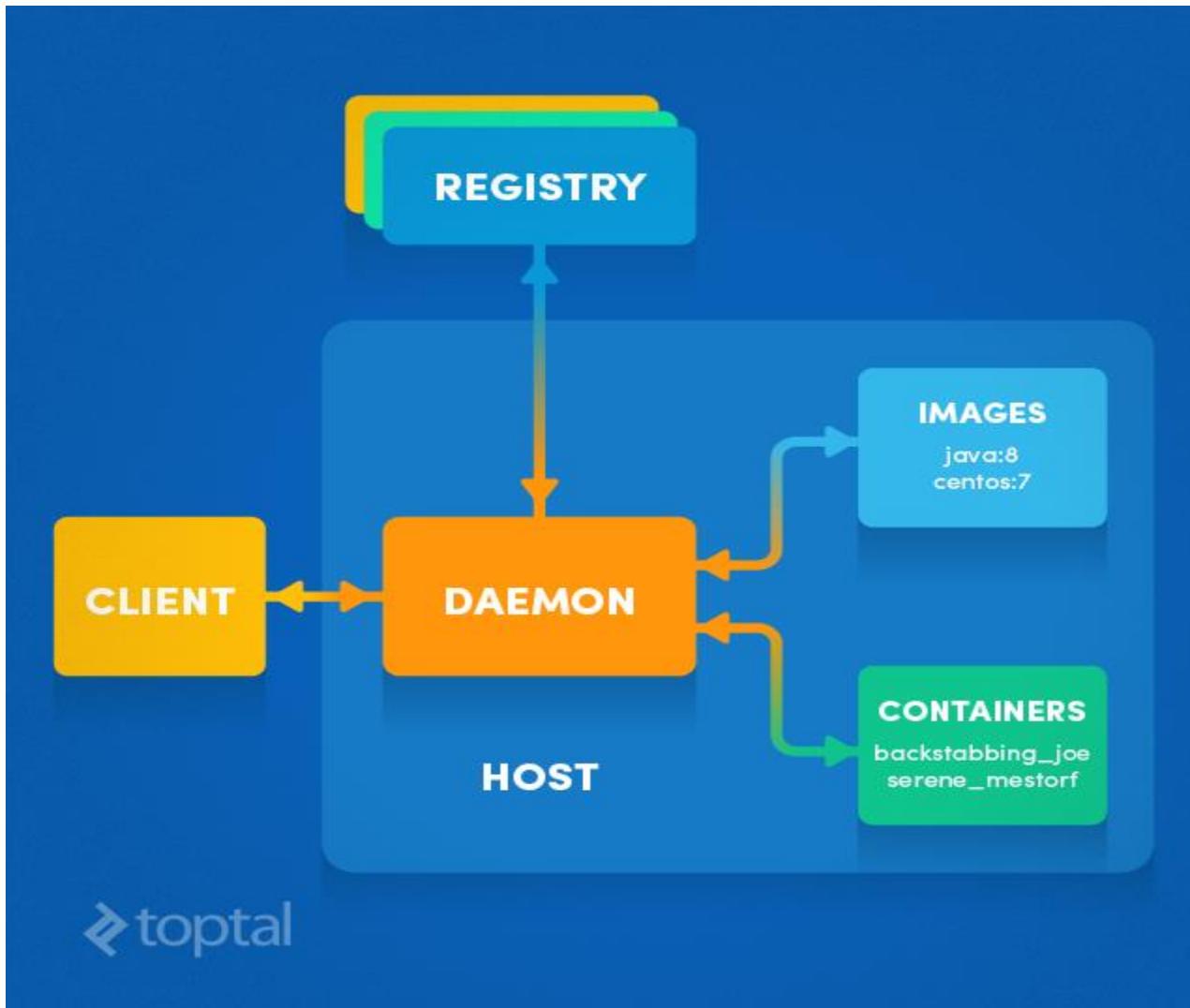


Docker Architecture

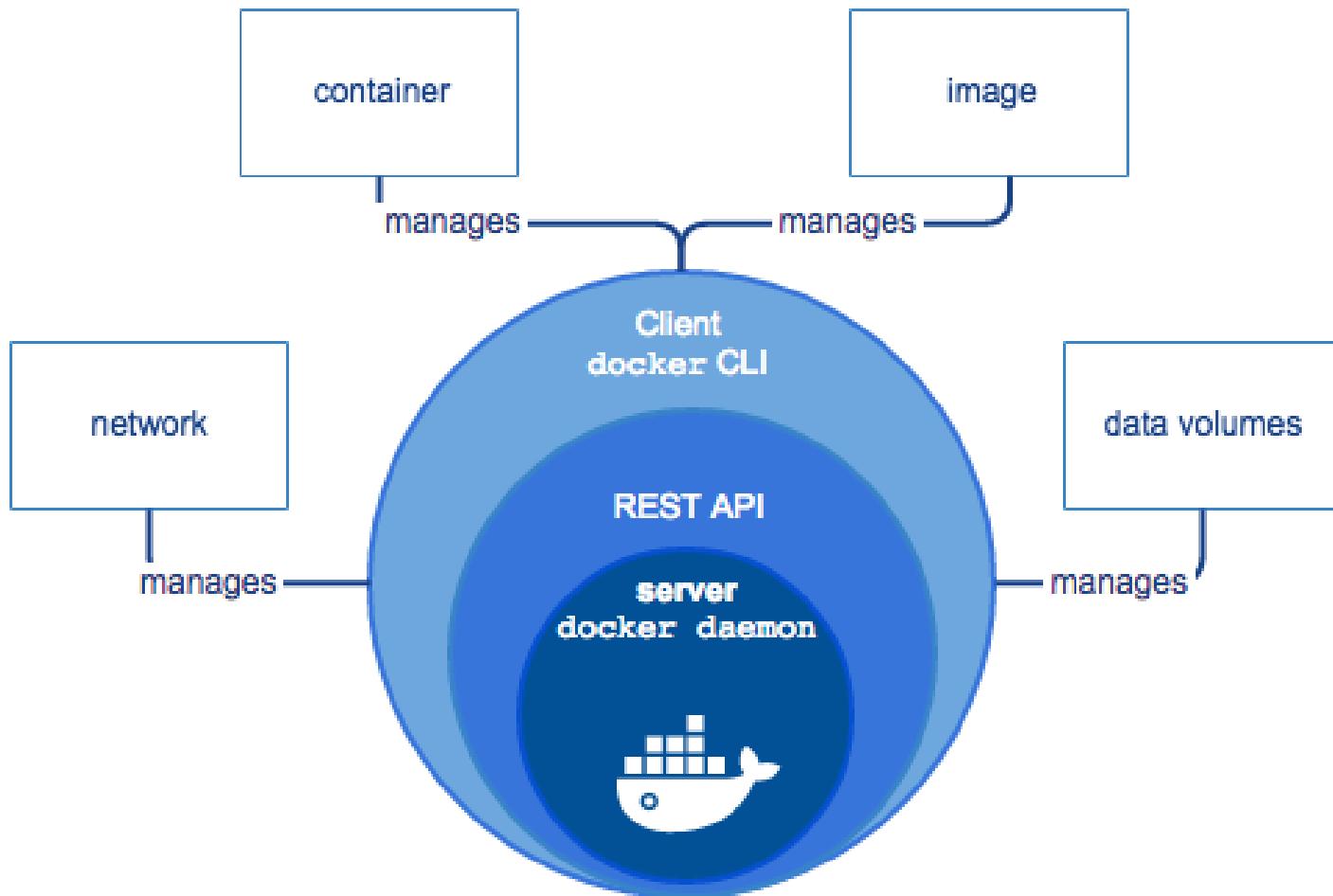




Docker Architecture



Docker Engine





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

- 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.

Dockerfile, Docker Image And Docker Container



- A Docker Image is created by the sequence of commands written in a file called as Dockerfile.
- When this Dockerfile is executed using a docker command it results into a Docker Image with a name.
- When this Image is executed by “docker run” command it will by itself start whatever application or service it must start on its execution.



Image Demo

- <https://learnk8s.io/spring-boot-kubernetes-guide>





Docker Network

- Docker implements networking in an application-driven manner and provides various options while maintaining enough abstraction for application developers.
- There are basically two types of networks available - the default Docker network and user-defined networks.
- By default, you get three different networks on the installation of Docker - none, bridge, and host.



Docker Network

- The none and host networks are part of the network stack in Docker.
- The bridge network automatically creates a gateway and IP subnet and all containers that belong to this network can talk to each other via IP addressing.
- This network is not commonly used as it does not scale well and has constraints in terms of network usability and service discovery.



Docker Network

- The other type of networks is user-defined networks. Administrators can configure multiple user-defined networks.
- There are three types:
- Bridge network: Similar to the default bridge network, a user-defined Bridge network differs in that there is no need for port forwarding for containers within the network to communicate with each other.
- The other difference is that it has full support for automatic network discovery.



Docker Network

- Overlay network: An Overlay network is used when you need containers on separate hosts to be able to communicate with each other, as in the case of a distributed network.
- However, a caveat is that swarm mode must be enabled for a cluster of Docker engines, known as a swarm, to be able to join the same group.
- Macvlan network: When using Bridge and Overlay networks a bridge resides between the container and the host.
- A Macvlan network removes this bridge, providing the benefit of exposing container resources to external networks without dealing with port forwarding.
- This is realized by using MAC addresses instead of IP addresses.



Storage

- You can store data within the writable layer of a container but it requires a storage driver.
- Being non-persistent, it perishes whenever the container is not running.
- It is not easy to transfer this data.
- In terms of persistent storage, Docker offers four options:
- Data Volume, Data Volume Container, Directory Mounts, Storage Plugins



Storage

- There are storage plugins from various companies to automate the storage provisioning process. For example,
- HPE 3PAR
- EMC (ScaleIO, XtremIO, VMAX, Isilon)
- NetApp
- There are also plugins that support public cloud providers like:
- Azure File Storage
- Google Compute Platform.



Registries

- A Docker registry is a storage and distribution system for named Docker images.
- The same image might have multiple different versions, identified by their tags.
- A Docker registry is organized into Docker repositories , where a repository holds all the versions of a specific image.
- The registry allows Docker users to pull images locally, as well as push new images to the registry (given adequate access permissions when applicable).



Registries

- By default, the Docker engine interacts with DockerHub , Docker's public registry instance.
- However, it is possible to run on-premise the open-source Docker registry/distribution, as well as a commercially supported version called Docker Trusted Registry .



Docker Hub

- DockerHub is a hosted registry solution by Docker Inc.
- Besides public and private repositories, it also provides automated builds, organization accounts, and integration with source control solutions like Github and Bitbucket .



Docker Hub

- A public repository is accessible by anyone running Docker, and image names include the organization/user name.
- For example, `docker pull jenkins/jenkins` will pull the Jenkins CI server image with tag `latest` from the Jenkins organization.
- There are hundreds of thousands public images available.
- Private repositories restrict access to the repository creator or members of its organization.



Docker Hub

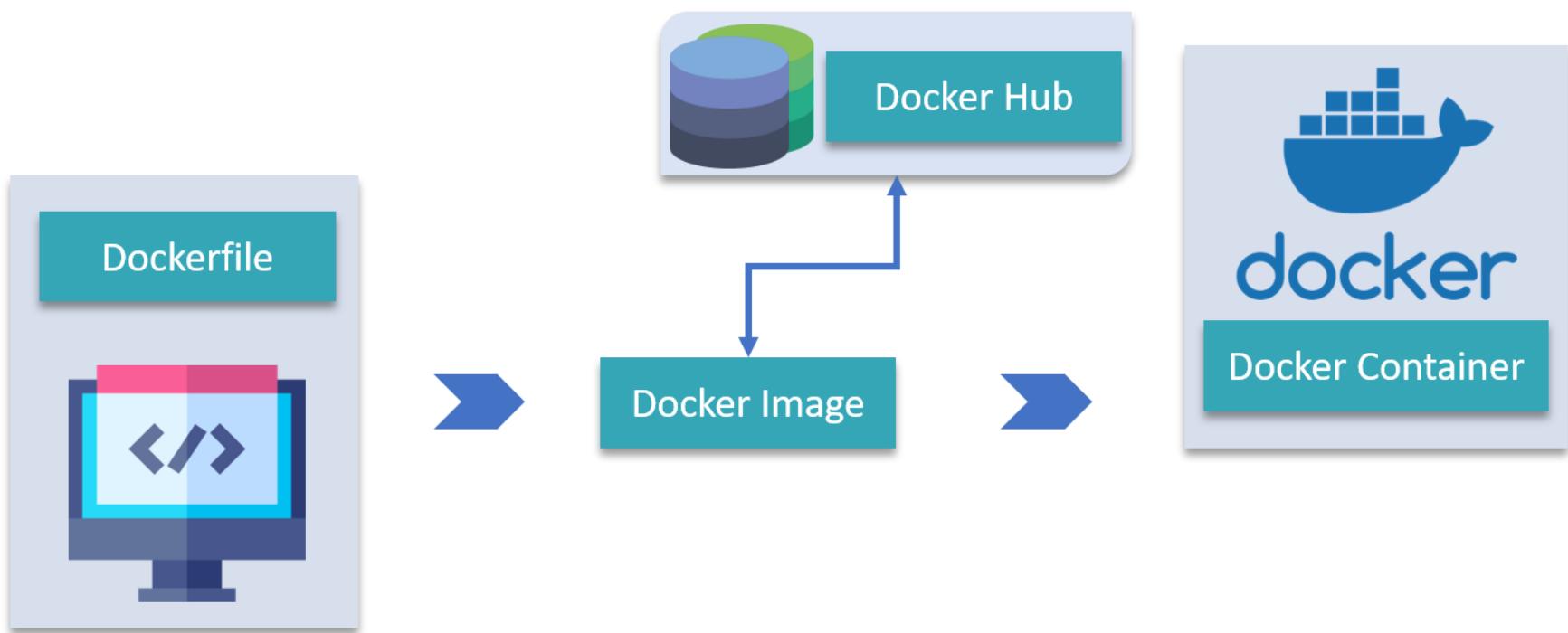
- DockerHub supports official repositories, which include images verified for security and best practices.
- These do not require an organization/user name, for example `docker pull nginx` will pull the latest image of the Nginx load balancer.
- DockerHub can perform automated image builds if the DockerHub repository is linked to a source control repository which contains a build context (`Dockerfile` and all any files in the same folder).
- A commit in the source repository will trigger a build in DockerHub.

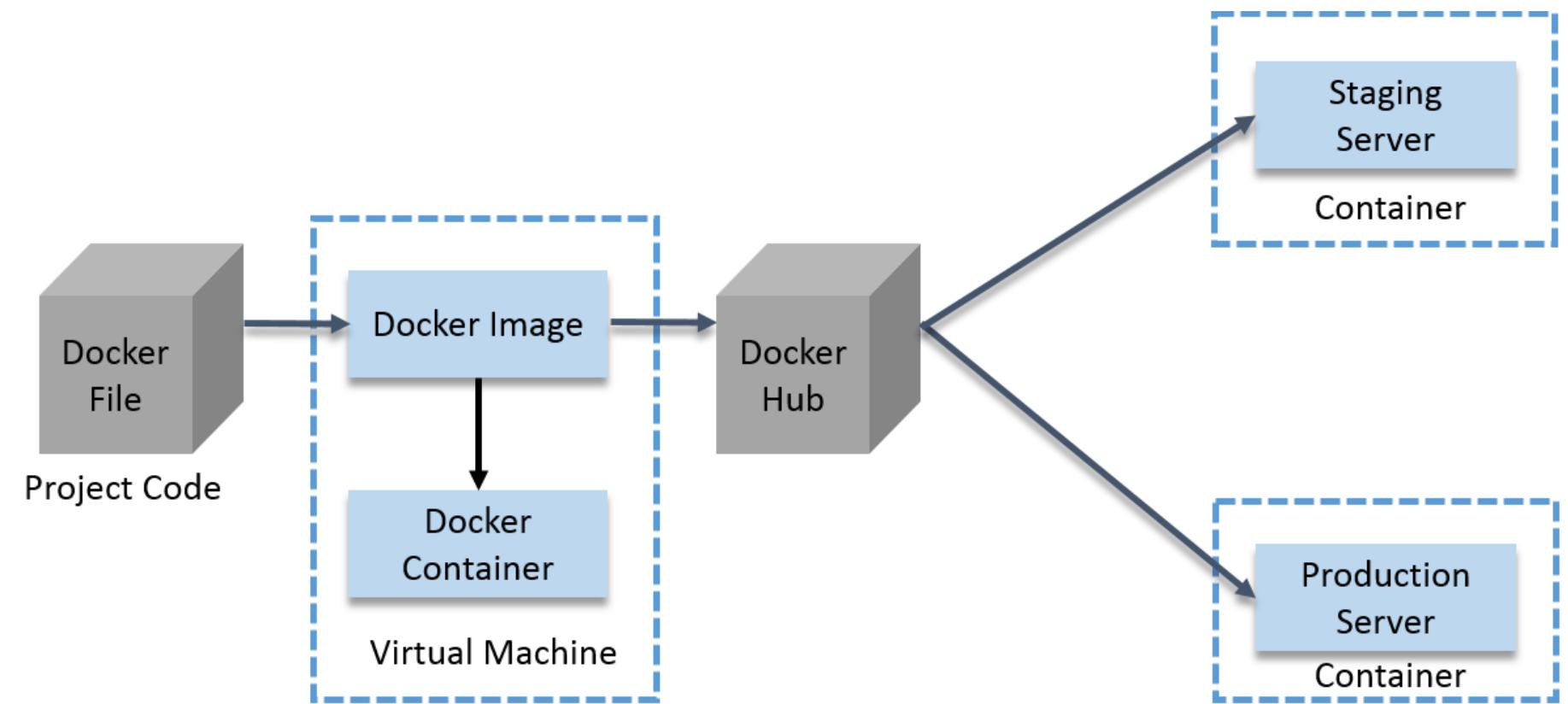
Docker Hub:

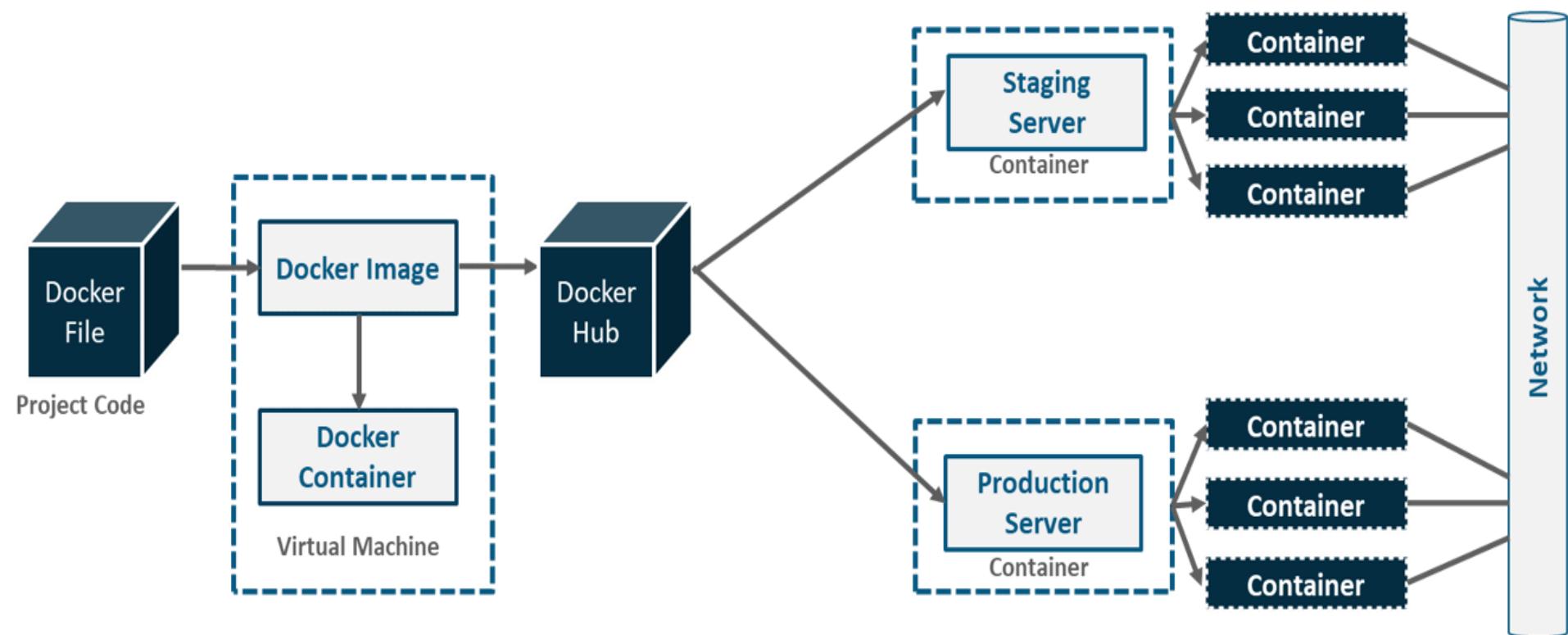


- Docker Hub is like GitHub for Docker Images. It is basically a cloud registry where you can find Docker Images uploaded by different communities, also you can develop your own image and upload on Docker Hub, but first, you need to create an account on DockerHub.

Docker Hub









Other Public Registries

- Other companies host paid online Docker registries for public use.
- Cloud providers like AWS and Google, who also offer container-hosting services, market the high availability of their registries.



Other Public Registries

- Amazon Elastic Container Registry (ECR) integrates with AWS Identity and Access Management (IAM) service for authentication. It supports only private repositories and does not provide automated image building.
- Google Container Registry (GCR) authentication is based on Google's Cloud Storage service permissions. It supports only private repositories and provides automated image builds via integration with Google Cloud Source Repositories, GitHub, and Bitbucket.



Other Public Registries

- Azure Container Registry (ACR) supports multi-region registries and authenticates with Active Directory. It supports only private repositories and does not provide automated image building.
- CoreOS Quay supports OAuth and LDAP authentication. It offers both (paid) private and (free) public repositories, automatic security scanning and automated image builds via integration with GitLab, GitHub, and Bitbucket.
- Private Docker Registry supports OAuth, LDAP and Active Directory authentication. It offers both private and public repositories, free up to 3 repositories (private or public).



Private Registries

- Use cases for running a private registry on-premise (internal to the organization) include:
- Distributing images inside an isolated network (not sending images over the Internet)
- Creating faster CI/CD pipelines (pulling and pushing images from internal network), including faster deployments to on-premise environments
- Deploying a new image over a large cluster of machines
- Tightly controlling where images are being stored



Private Registries

- Running a private registry system, especially when delivery to production depends on it, requires operational skills such as ensuring availability, logging and log processing, monitoring, and security.
- Strong understanding of http and overall network communications is also important.

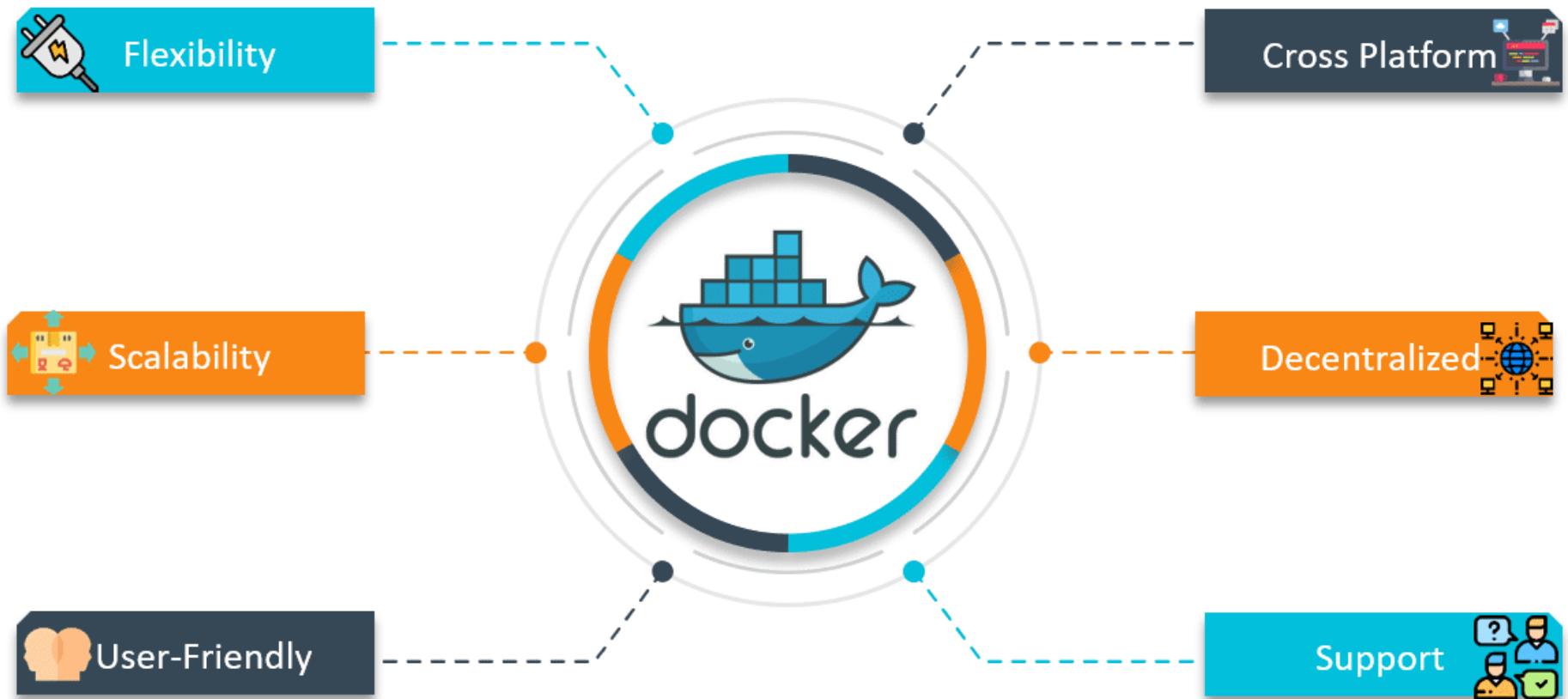


Private Registries

- Docker Trusted Registry is Docker Inc's commercially supported version, providing high availability via replication, image auditing, signing and security scanning, integration with LDAP and Active Directory.
- Harbor is a VMWare open source offering which also provides high availability via replication, image auditing, integration with LDAP and Active Directory.
- GitLab Container Registry is tightly integrated with GitLab CI's workflow, with minimal setup.
- JFrog Artifactory for strong artifact management (not only Docker images but any artifact).



Goals of Docker Networking





Creating Test Database Server

- This is a great Docker use case.
- We might not want to run our production database in Docker (perhaps we'll just use Amazon RDS for example), but we can spin up a clean MySQL database in no time as a Docker container for development - leaving our development machine clean and keeping everything we do controlled and repeatable.



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.



Best practices for writing Dockerfiles

- Docker builds images automatically by reading the instructions from a Dockerfile -- a text file that contains all commands, in order, needed to build a given image.
- A Dockerfile adheres to a specific format and set of instructions which you can find at Dockerfile reference.
- A Docker image consists of read-only layers each of which represents a Dockerfile instruction.
- The layers are stacked and each one is a delta of the changes from the previous layer.



Best practices for writing Dockerfiles

- FROM ubuntu:18.04
- COPY . /app
- RUN make /app
- CMD python /app/app.py
- FROM creates a layer from the ubuntu:18.04 Docker image.
- COPY adds files from your Docker client's current directory.
- RUN builds your application with make.
- CMD specifies what command to run within the container.



Best practices for writing Dockerfiles

- https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

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.

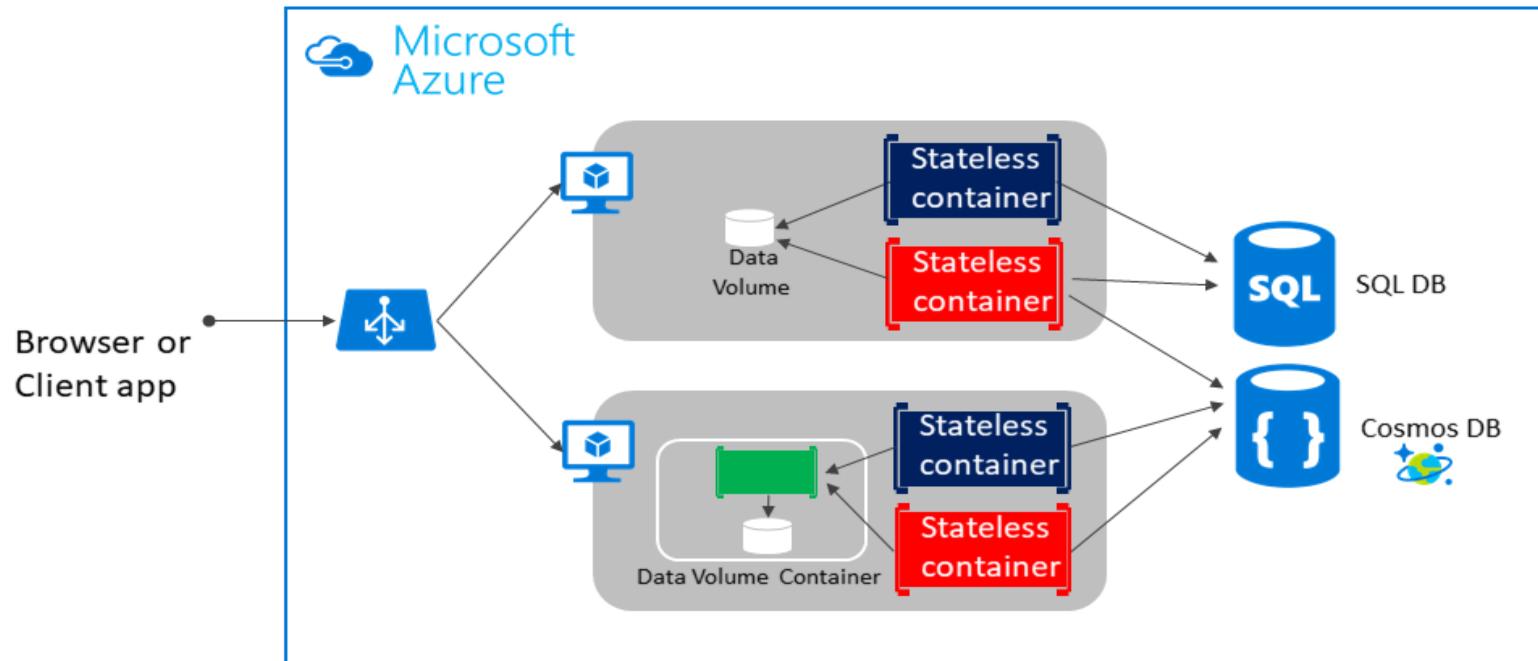
State and data in Docker applications



- 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





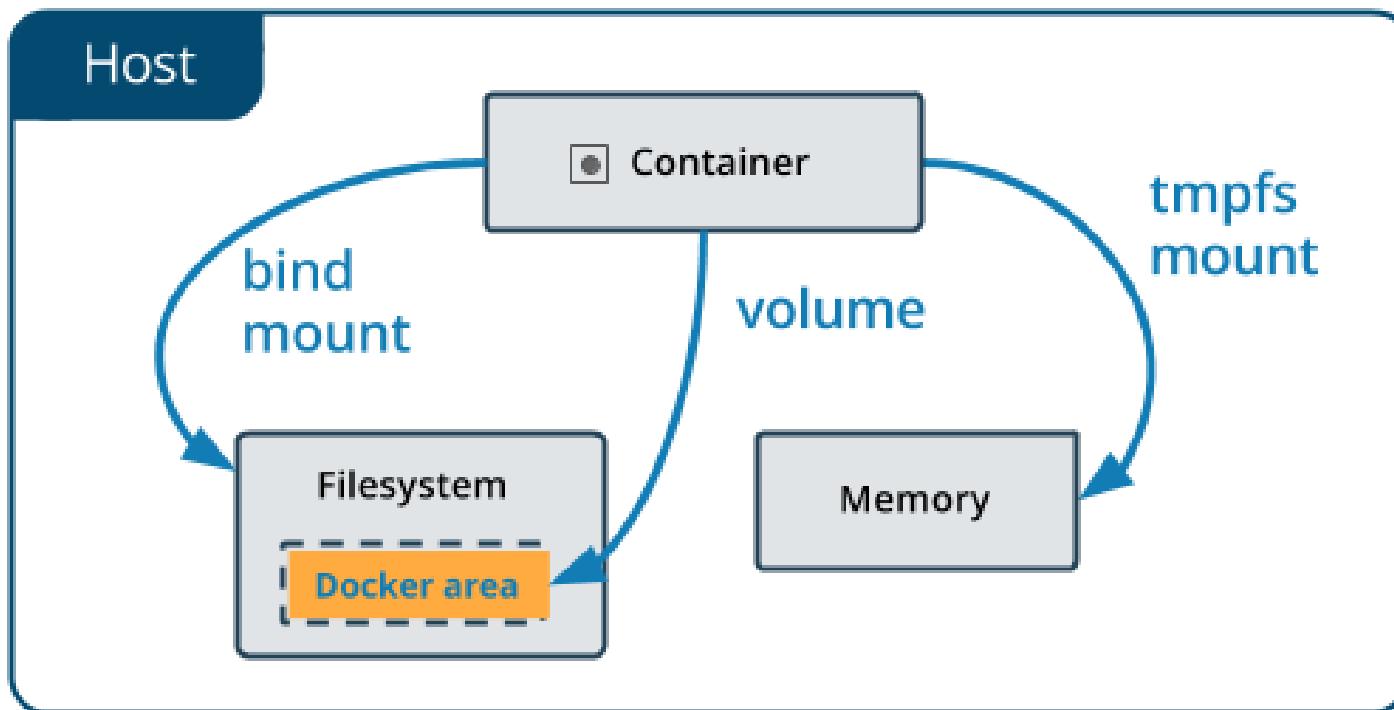
Docker Volume

- Volumes are the preferred mechanism for persisting data generated by and used by Docker containers.
- While bind mounts are dependent on the directory structure of the host machine, volumes are completely managed by Docker.



Docker Volume

- Volumes have several advantages over bind mounts:
 - Volumes are easier to back up or migrate than bind mounts.
 - You can manage volumes using Docker CLI commands or the Docker API.
 - Volumes work on both Linux and Windows containers.
 - Volumes can be more safely shared among multiple containers.
 - Volume drivers let you store volumes on remote hosts or cloud providers, to encrypt the contents of volumes, or to add other functionality.
 - New volumes can have their content pre-populated by a container





Manage Data in Docker

- There are 2 ways in which you can manage data in Docker:
 - Data volumes
 - Data volume containers



Manage Data in Docker

- A data volume is a specially designed directory in the container.
- It is initialized when the container is created.
- By default, it is not deleted when the container is stopped.
- It is not even garbage collected when there is no container referencing the volume.
- The data volumes are independently updated.
- Data volumes can be shared across containers too.
They could be mounted in read-only mode too.



Manage Data in Docker

- Mounting a Data volume
- docker container run -it -v/udata --tty ubuntu /bin/bash
- cd udata
- touch file1.txt
- exit
- docker container restart 2eec01eb7368
- docker attach 2eec01eb7368exit
- docker container rm 2eec01eb7368
- docker volume ls



Manage Data in Docker

- After removing container also the data volume is still present on the host.
- This is a dangling or ghost volume and could remain there on your machine consuming space.
- Do remember to clean up if you want.
- Alternatively, there is also a -v option while removing the container



```
root@46a4edc2cf30:/  
Error response from daemon: You cannot remove a running container e26b6d0c5de3bc889381c20b2d169ce8a61bf701dfe46aea127e0e03d314fb0e. Sto  
p the container before attempting removal or force remove  
C:\WINDOWS\system32>docker container stop e26b6d0c5de3  
e26b6d0c5de3  
C:\WINDOWS\system32>docker container rm e26b6d0c5de3  
e26b6d0c5de3  
C:\WINDOWS\system32>docker container run -it --tty ubuntu /bin/bash  
Unable to find image 'ubuntu:latest' locally  
latest: Pulling from library/ubuntu  
a4a2a29f9ba4: Pull complete  
127c9761dcba: Pull complete  
d13bf203e905: Pull complete  
4039240d2e0b: Pull complete  
Digest: sha256:35c4a2c15539c6c1e4e5fa4e554dac323ad0107d8eb5c582d6ff386b383b7dce  
Status: Downloaded newer image for ubuntu:latest  
root@46a4edc2cf30:/# ls  
bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp usr var  
root@46a4edc2cf30:/#
```



```
root@2eec01eb7368:/udata
```

```
laughingeinstein
```

```
C:\WINDOWS\system32>docker container rm 46a4edc2cf30  
46a4edc2cf30
```

```
C:\WINDOWS\system32>docker container run -it -v/udata --tty ubuntu /bin/bash
```

```
root@2eec01eb7368:/# ls  
bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp udata usr var  
root@2eec01eb7368:/# cd udata  
root@2eec01eb7368:/udata#
```



```
Administrator: Command Prompt
"LowerDir": "/var/lib/docker/overlay2/b79f152c6131e7d3c11ec0c3b019497c8a4f17ac12411a0a49719b21508c35a1-init/diff:/var/lib/docker/overlay2/3eab3aebc8c580252e330fe3efa59b8d92c1339cb4dbf46e8fc374dfef63f569/diff:/var/lib/docker/overlay2/3872c83edf6b5c30265e21691152c9126a2ffee424afc81202f02f07b1819210/diff:/var/lib/docker/overlay2/25d3f8faffb92bef36d1b620a6ababbd344c1cb58f41fb565620be058a0dcf8e/diff:/var/lib/docker/overlay2/e4ebde681507d7b624d11dae676fbca8cb273d91e18e55a3d511a5b4fd0cd9ac/diff",
    "MergedDir": "/var/lib/docker/overlay2/b79f152c6131e7d3c11ec0c3b019497c8a4f17ac12411a0a49719b21508c35a1/merged",
    "UpperDir": "/var/lib/docker/overlay2/b79f152c6131e7d3c11ec0c3b019497c8a4f17ac12411a0a49719b21508c35a1/diff",
    "WorkDir": "/var/lib/docker/overlay2/b79f152c6131e7d3c11ec0c3b019497c8a4f17ac12411a0a49719b21508c35a1/work"
},
    "Name": "overlay2"
},
"Mounts": [
{
    "Type": "volume",
    "Name": "7b3c1be1134c62d876416f35818c1545d2a96c4aff4443b1457ebb0b08f840f3",
    "Source": "/Var/lib/docker/volumes/7b3c1be1134c62d876416f35818c1545d2a96c4aff4443b1457ebb0b08f840f3/_data",
    "Destination": "/udata",
    "Driver": "local",
    "Mode": "",
    "RW": true,
    "Propagation": ""
}
],
"Config": {
    "Hostname": "2eec01eb7368",
    "Domainname": "",
    "User": "",
    "AttachStdin": true,
    "AttachStdout": true,
    "AttachStderr": true,
    "Tty": true,
    "OpenStdin": true,
    "StdinOnce": true,
    "Env": [

```





```
root@2eec01eb7368: /udata
```

94d81de361e1	mysql	"docker-entrypoint.s..."	3 hours ago	Up 2 hours
6/tcp, 33060/tcp	virtusa-mysql			
36acbeba8c36	docker/getting-started	"nginx -g 'daemon of..."	3 hours ago	Created
	laughing_einstein			

```
C:\WINDOWS\system32>docker container restart 2eec01eb7368  
2eec01eb7368
```

```
C:\WINDOWS\system32>docker attach 2eec01eb7368  
root@2eec01eb7368:/# ls  
bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp udata usr var  
root@2eec01eb7368:/# cd udata  
root@2eec01eb7368:/udata# ls  
file1.txt  
root@2eec01eb7368:/udata#
```



Manage Data in Docker

- The sequence of steps are as follows:
- We restarted the container (`container1`)
- We attached to the running container (`container1`)
- We did a `ls` and found that our volume `/data` is still there.
- We did a `cd` into the `/data` volume and did a `ls` there. And our file is still present.



Mounting a Host Directory as a Data volume

Screenshot of the Docker Settings interface showing the 'Resources' section.

The left sidebar shows the following sections:

- General
- Resources** (selected)
- ADVANCED
 - FILE SHARING
- PROXIES
- NETWORK
- Docker Engine
- Command Line
- Kubernetes

The main content area is titled "Resources File sharing". It contains the following text:
These directories (and their subdirectories) can be bind mounted into Docker containers. You can check the [documentation](#) for more details.

Path	Action
E:\DockerFileShare	⊖
C:\path\to\exported\directory	⊕

```
docker container run -it -v e:/DockerFileShare/project/web01:/mnt/test ubuntu /bin/bash
```



Data volume containers

- Creating a Data volume container is very useful if you want to share data between containers or you want to use the data from non-persistent containers.
- The process is really two step:
 - You first create a Data volume container
 - Create another container and mount the volume from the container created in Step 1.



Data volume containers

- Creating a Data volume container is very useful if you want to share data between containers or you want to use the data from non-persistent containers.
- The process is really two step:
 - You first create a Data volume container
 - Create another container and mount the volume from the container created in Step 1.



Data volume containers

```
Administrator: Command Prompt - docker run -it -v /data --name container1 busybox
local          4e7fc43d326fc162732551862b8ec241772a468b891ee93e716c870af082b228
local          7b3c1be1134c62d876416f35818c1545d2a96c4aff4443b1457ebb0b08f840f3
local          030cf2d5467d3086b88f8d6e1d613a56426f16c9e398407b1082f168b3b01e4e
local          5834bcb24dcc7c574889dde2c5091956a1915c11a11baad89bf166a3bd00d9dd
local          a6176c52cd63f2ce3a787f61f7374bd36b5810406d4ae2d55bedcb2cd493dc2

C:\WINDOWS\system32>docker run -it -v /data --name container1 busybox
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
76df9210b28c: Pull complete
Digest: sha256:95cf004f559831017cdf4628aaf1bb30133677be8702a8c5f2994629f637a209
Status: Downloaded newer image for busybox:latest
/ # cd data
/data # touch file1.txt
/data # touch file2.txt
/data # ls
file1.txt  file2.txt
/data #
```



Data volume containers

```
Administrator: Command Prompt - docker run -it -v /data --name container1 busybox
local          4e7fc43d326fc162732551862b8ec241772a468b891ee93e716c870af082b228
local          7b3c1be1134c62d876416f35818c1545d2a96c4aff4443b1457ebb0b08f840f3
local          030cf2d5467d3086b88f8d6e1d613a56426f16c9e398407b1082f168b3b01e4e
local          5834bcb24dcc7c574889dde2c5091956a1915c11a11baad89bf166a3bd00d9dd
local          a6176c52cd63f2ce3a787f61f7374bd36b5810406d4ae2d55bedcb2cd493dc2

C:\WINDOWS\system32>docker run -it -v /data --name container1 busybox
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
76df9210b28c: Pull complete
Digest: sha256:95cf004f559831017cdf4628aaaf1bb30133677be8702a8c5f2994629f637a209
Status: Downloaded newer image for busybox:latest
/ # cd data
/data # touch file1.txt
/data # touch file2.txt
/data # ls
file1.txt  file2.txt
/data #
```



Docker network

```
Administrator: Command Prompt
76df9210b28c: Pull complete
Digest: sha256:95cf004f559831017cdf4628aaf1bb30133677be8702a8c5f2994629f637a209
Status: Downloaded newer image for busybox:latest
/ # cd data
/data # touch file1.txt
/data # touch file2.txt
/data # ls
file1.txt  file2.txt
/data # exit

C:\WINDOWS\system32>docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
5cc9ae61fdda   bridge    bridge      local
3857a50b891b   host      host       local
e43a4f132e84   none     null       local

C:\WINDOWS\system32>
```



Docker network

- **docker network inspect bridge**
- Docker automatically creates a subnet and gateway for the bridge network, and docker run automatically adds containers to it. I
- If you have containers running on your network, docker network inspect displays networking information for your containers.



Docker network

- Any containers on the same network may communicate with one another via IP addresses.
- Docker does not support automatic service discovery on bridge.
- You must connect containers with the --link option in your docker run command.



Docker network

- The Docker bridge supports port mappings and docker run --link allowing communications between containers on the docker0 network.
- However, these error-prone techniques require unnecessary complexity.
- It's better to define your own networks instead.



Docker network

- None
- This offers a container-specific network stack that lacks a network interface.
- This container only has a local loopback interface (i.e., no external network interface).



Docker network

- Host
- This enables a container to attach to your host's network (meaning the configuration inside the container matches the configuration outside the container).



Defining your own networks

- You can create multiple networks with Docker and add containers to one or more networks.
- Containers can communicate within networks but not across networks.
- A container with attachments to multiple networks can connect with all of the containers on all of those networks.
- This lets you build a “hub” of sorts to connect to multiple networks and separate concerns.



Defining your own networks

- Bridge networks (similar to the default docker0 network) offer the easiest solution to creating your own Docker network.
- While similar, you do not simply clone the default0 network, so you get some new features and lose some old ones.
- Follow along below to create your own `my_isolated_bridge_network` and run your Postgres container `my_mysql_db` on that network:



Defining your own networks

- `docker network create --driver bridge my_isolated_bridge_network`
- `docker network inspect my_isolated_bridge_network`
- `docker network ls`
- `docker run --net=my_isolated_bridge_network --name=my_psql_db postgres`
- `docker run --network=my_isolated_bridge_network --name=my_app hello-world`



Defining your own networks

Administrator: Command Prompt

```
"EnableIPv6": false,
"IPAM": {
    "Driver": "default",
    "Options": {},
    "Config": [
        {
            "Subnet": "172.18.0.0/16",
            "Gateway": "172.18.0.1"
        }
    ],
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
        "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
        "fce4adc125341807e7b334cb2908436a2a5b9596aa6b08749203f9c6e18d6dbb": {
            "Name": "competent_darwin",
            "EndpointID": "ca6a9d07770e8226967cf6c601d327cfe19b7b15396a09229f0e540f83f066e1",
            "MacAddress": "02:42:ac:12:00:02",
            "IPv4Address": "172.18.0.2/16",
            "IPv6Address": ""
        }
    },
    "Options": {},
    "Labels": {}
}
]
```

C:\WINDOWS\system32>





Creating an overlay network

- If you want native multi-host networking, you need to create an overlay network.
- These networks require a valid key-value store service, such as Consul, Etcd, or ZooKeeper.
- You must install and configure your key-value store service before creating your network.
- Your Docker hosts (you can use multiple hosts with overlay networks) must communicate with the service you choose.
- Each host needs to run Docker.
- You can provision the hosts with Docker Machine.



Creating an overlay network

- Create the overlay network in a similar manner to the bridge network (network name `my_multi_host_network`):
- `docker network create --driver overlay my_multi_host_network`
- Launch containers on each host; make sure you specify the network name:
- `docker run -itd -net=my_multi_host_network my_python_app`

Getting Started with Artifactory as a Docker Registry

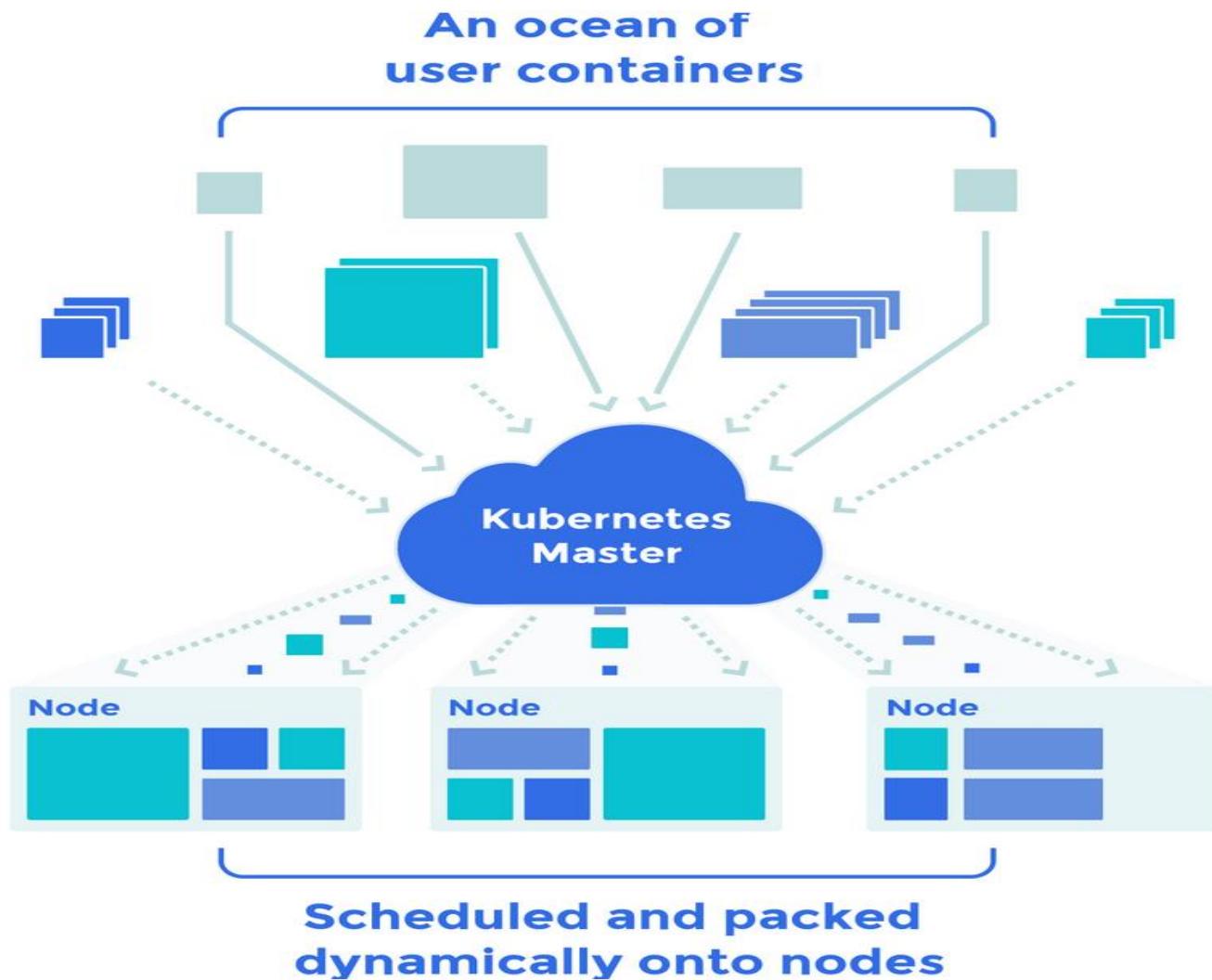


- Login to your repository use the following command with your Artifactory Cloud credentials.
- `docker login ${server-name}-${repo-name}.jfrog.io`
- Pull an image using the following command.
- `docker pull ${server-name}-${repo-name}.jfrog.io/<image name>`
- Push an image by first tagging it and then using the push command.
- `docker tag <image name> ${server-name}-${repo-name}.jfrog.io/<image name>`
- `docker push ${server-name}-${repo-name}.jfrog.io/<image name>`

Getting Started with Artifactory as a Docker Registry



- In this example, the Artifactory Cloud server is named **acme**.
- Start by creating a virtual Docker repository called dockerv2-virtual.
- Pull the hello-world image
- docker pull hello-world
- Login to repository dockerv2-virtual
- docker login acme-dockerv2-virtual.jfrog.io
- Tag the hello-world image
- docker tag hello-world acme-dockerv2-virtual.jfrog.io/hello-world
- Push the tagged hello-world image to dockerv2-virtual
- docker push acme-dockerv2-virtual.jfrog.io/hello-world





Docker Swarm

- Docker Swarm is Docker's native feature to support clustering of Docker machines.
- This enables multiple machines running Docker Engine to participate in a cluster, called Swarm.
- The Docker engines contributing to a Swarm are said to be running in Swarm mode.
- Machines enter into the Swarm mode by either initializing a new swarm or by joining an existing swarm.
- To the end user the swarm would seem like a single machine.
- A Docker engine participating in a swarm is called a node



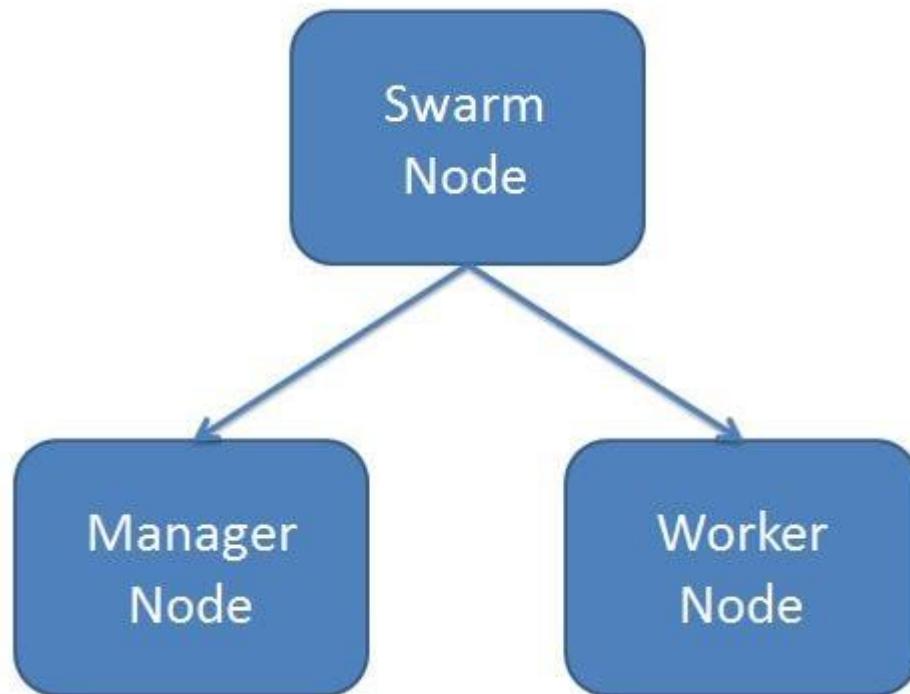
Docker Swarm

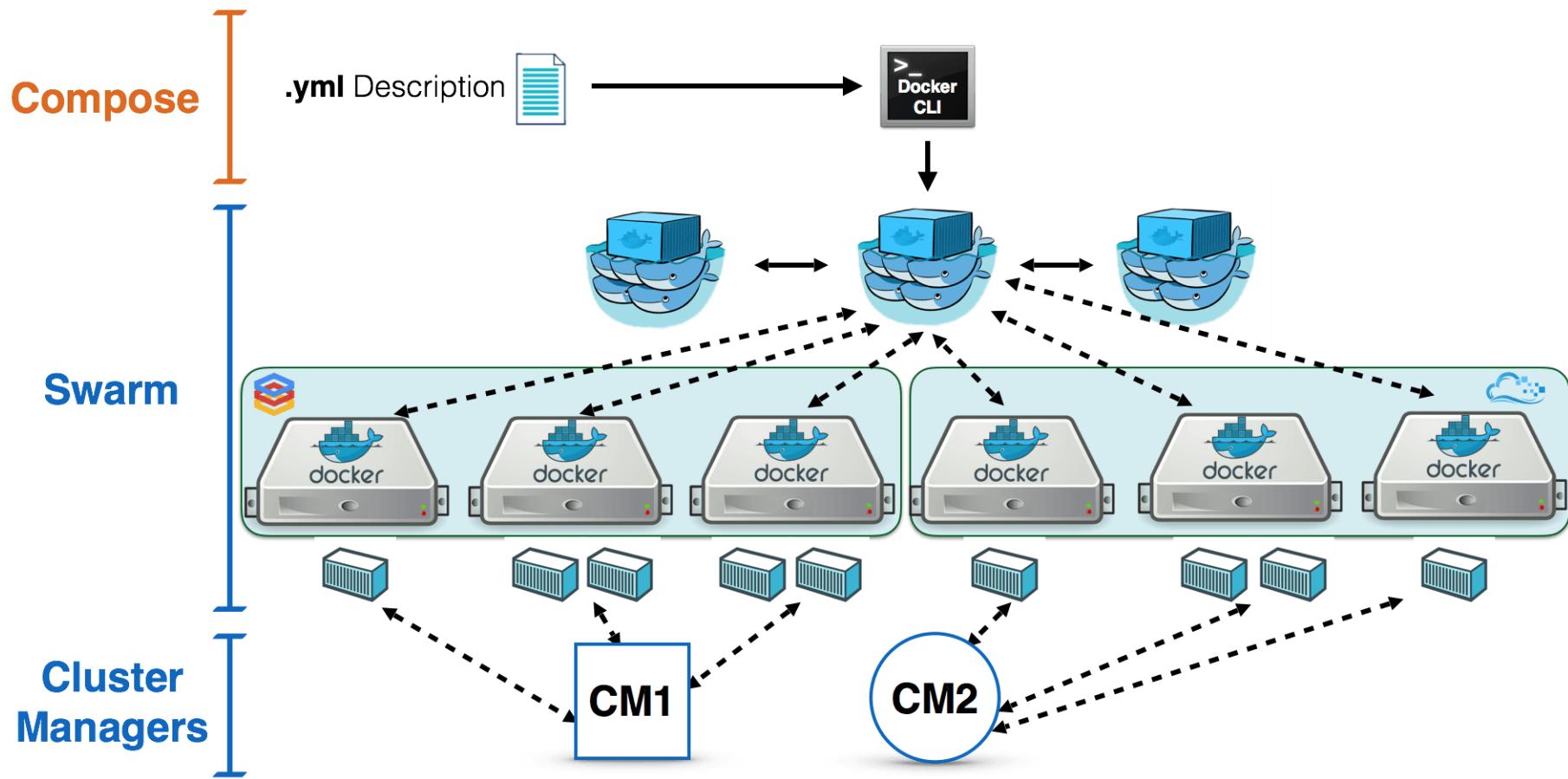
- A node can either be a manager node or a worker node.
- The manager node performs cluster management and orchestration while the worker nodes perform tasks allocated by the manager.
- A manager node itself, unless configured otherwise, is also a worker node.
- The central entity in the Docker Swarm infrastructure is called a service.
- A Docker swarm executes services. The user submits a service to the manager node to deploy and execute.
- A service is made up of many tasks. A task is the most basic work unit in a Swarm.
- A task is allocated to each worker node by the manager node.

Docker Swarm



- The Docker ecosystem consists of tools from development to production deployment frameworks.
- In that list, docker swarm fits into cluster management.
- A mix of docker-compose, swarm, overlay network and a good service discovery tool such as etcd or consul can be used for managing a cluster of Docker containers.
- Docker swarm is still maturing in terms of functionalities when compared to other open-source container cluster management tools.
- Considering the vast docker contributors, it won't be so long for docker swarm to have all the best functionalities other tools possess.
- Docker has documented a good production plan for using docker swarm in production.







```
Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker service create --name eurekaswarm -p 8761:8761 eureka-app:latest
image eureka-app:latest could not be accessed on a registry to record
its digest. Each node will access eureka-app:latest independently,
possibly leading to different nodes running different
versions of the image.
```

```
0uw58z9g5vbjl13zzjop10k9
overall progress: 1 out of 1 tasks
1/1: running  [=====>]
verify: Service converged
```

```
Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
0uw58z9g5vbj	eurekaswarm	replicated	1/1	eureka-app:latest	*:8761->8761/tcp

```
Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker service scale eurekaswarm=3
eurekaswarm scaled to 3
overall progress: 3 out of 3 tasks
1/3: running  [=====>]
2/3: running  [=====>]
3/3: running  [=====>]
verify: Service converged
```

```
Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
0uw58z9g5vbj	eurekaswarm	replicated	3/3	eureka-app:latest	*:8761->8761/tcp

```
Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
```



What is Kubernetes?

- Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.
- It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.
- The name Kubernetes originates from Greek, meaning helmsman or pilot.
- Google open-sourced the Kubernetes project in 2014.
- Kubernetes combines over 15 years of Google's experience running production workloads at scale with best-of-breed ideas and practices from the community.

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



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.
- .



Why use Kubernetes?

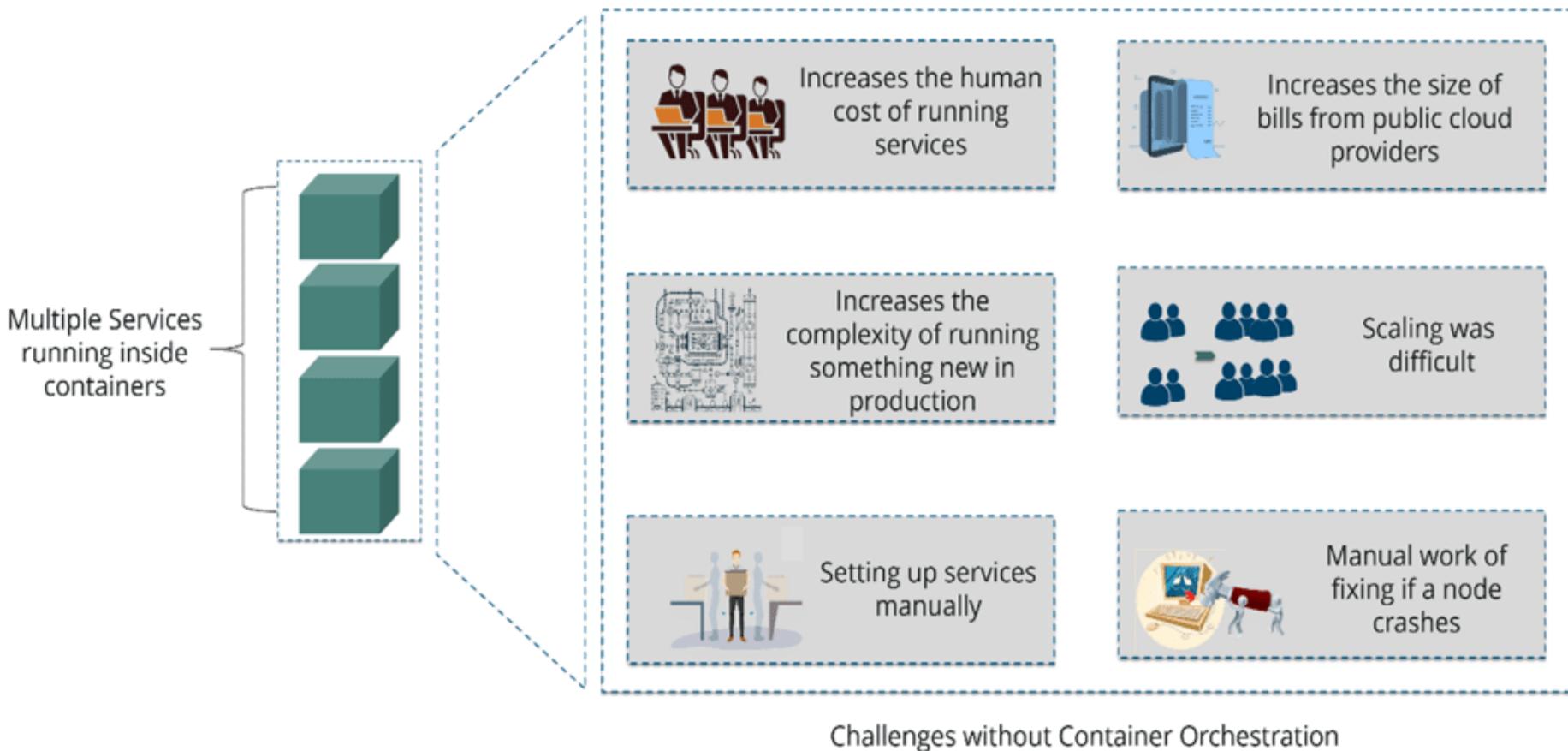
- Kubernetes can run on-premises bare metal, OpenStack, public clouds Google, Azure, AWS, etc.
- Helps you to avoid vendor lock issues as it can use any vendor-specific APIs or services except where Kubernetes provides an abstraction, e.g., load balancer and storage.
- Containerization using kubernetes allows package software to serve these goals. It will enable applications that need to be released and updated without any downtime.
- Kubernetes allows you to assure those containerized applications run where and when you want and helps you to find resources and tools which you want to work.



What task are performed by Kubernetes?

- Kubernetes is the Linux kernel which is used for distributed systems.
- It helps you to abstract the underlying hardware of the nodes(servers) and offers a consistent interface for applications that consume the shared pool of resources.

What is the need for Container Orchestration?



Kubernetes vs Docker Swarm



Features	Kubernetes	Docker Swarm
Installation & Cluster Config	Setup is very complicated, but once installed cluster is robust.	Installation is very simple, but the cluster is not robust.
GUI	GUI is the Kubernetes Dashboard.	There is no GUI.
Scalability	Highly scalable and scales fast.	Highly scalable and scales 5x faster than Kubernetes.
Auto-scaling	Kubernetes can do auto-scaling.	Docker swarm cannot do auto-scaling.
Load Balancing	Manual intervention needed for load balancing traffic between different containers and pods.	Docker swarm does auto load balancing of traffic between containers in the cluster.
Rolling Updates & Rollbacks	Can deploy rolling updates and does automatic rollbacks.	Can deploy rolling updates, but not automatic rollback.
DATA Volumes	Can share storage volumes only with the other containers in the same pod.	Can share storage volumes with any other container.
Logging & Monitoring	In-built tools for logging and monitoring.	3rd party tools like ELK stack should be used for logging and monitoring.

What are the features of Kubernetes?



01

Automated Scheduling

Kubernetes provides advanced scheduler to launch container on cluster nodes

02

Self Healing Capabilities

Rescheduling, replacing and restarting the containers which are died.

03

Automated rollouts and rollback

Kubernetes supports rollouts and rollbacks for the desired state of the containerized application

04

Horizontal Scaling and Load Balancing

Kubernetes can scale up and scale down the application as per the requirements



Features of Kubernetes

- Automated Scheduling
- Self-Healing Capabilities
- Automated rollouts & rollback
- Horizontal Scaling & Load Balancing
- Offers environment consistency for development, testing, and production
- Infrastructure is loosely coupled to each component can act as a separate unit
- Provides a higher density of resource utilization
- Offers enterprise-ready features
- Application-centric management
- Auto-scalable infrastructure
- You can create predictable infrastructure



Kubernetes Basics

- Cluster:
- It is a collection of hosts(servers) that helps you to aggregate their available resources. That includes ram, CPU, ram, disk, and their devices into a usable pool.
- Master:
- The master is a collection of components which make up the control panel of Kubernetes. These components are used for all cluster decisions. It includes both scheduling and responding to cluster events.



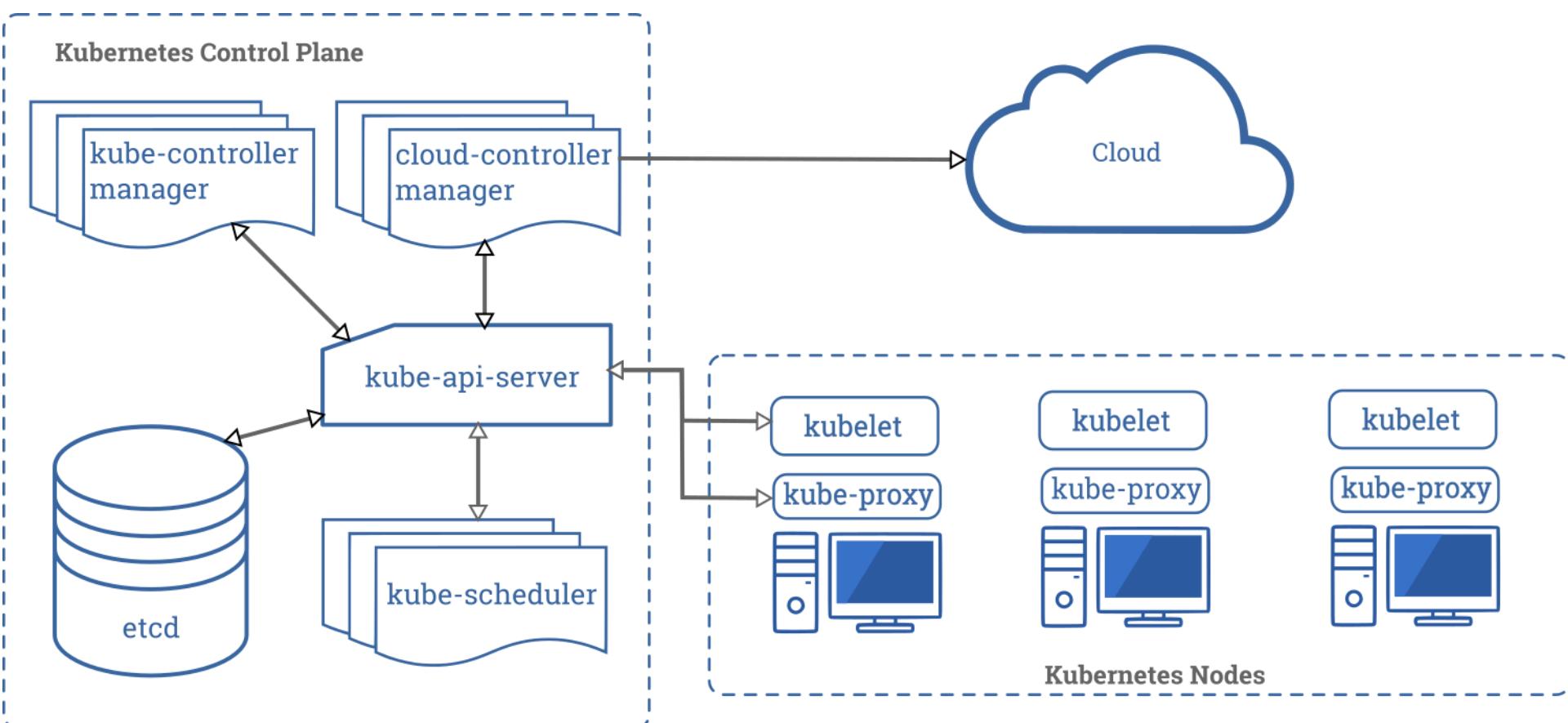
Kubernetes Basics

- Node:
 - It is a single host which is capable of running on a physical or virtual machine. A node should run both kube-proxy, minikube, and kubelet which are considered as a part of the cluster.
- Namespace:
 - It is a logical cluster or environment. It is a widely used method which is used for scoping access or dividing a cluster.



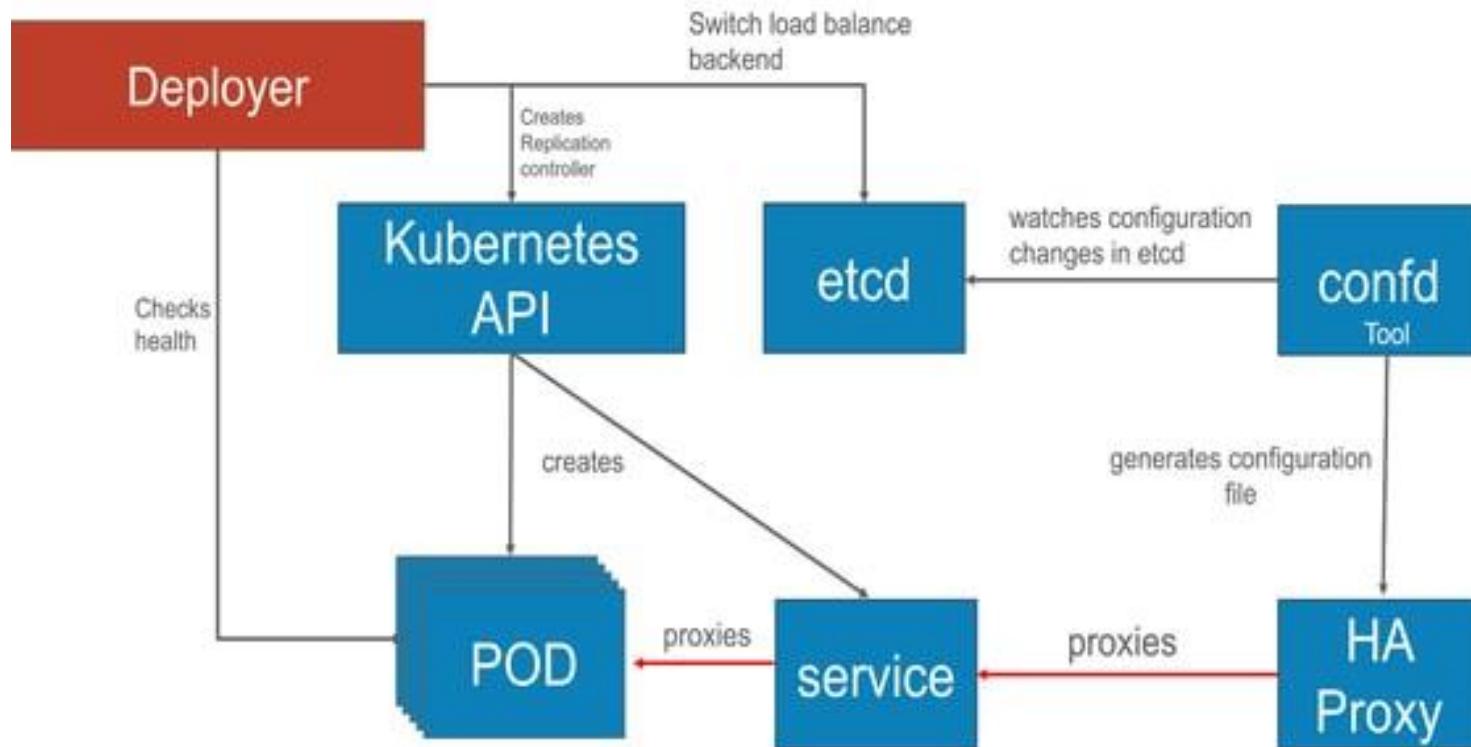
Kubernetes Architecture and Components

- Kubernetes has a decentralized architecture that does not handle tasks sequentially.
- It functions based on a declarative model and implements the concept of a ‘desired state.’





Kubernetes architecture

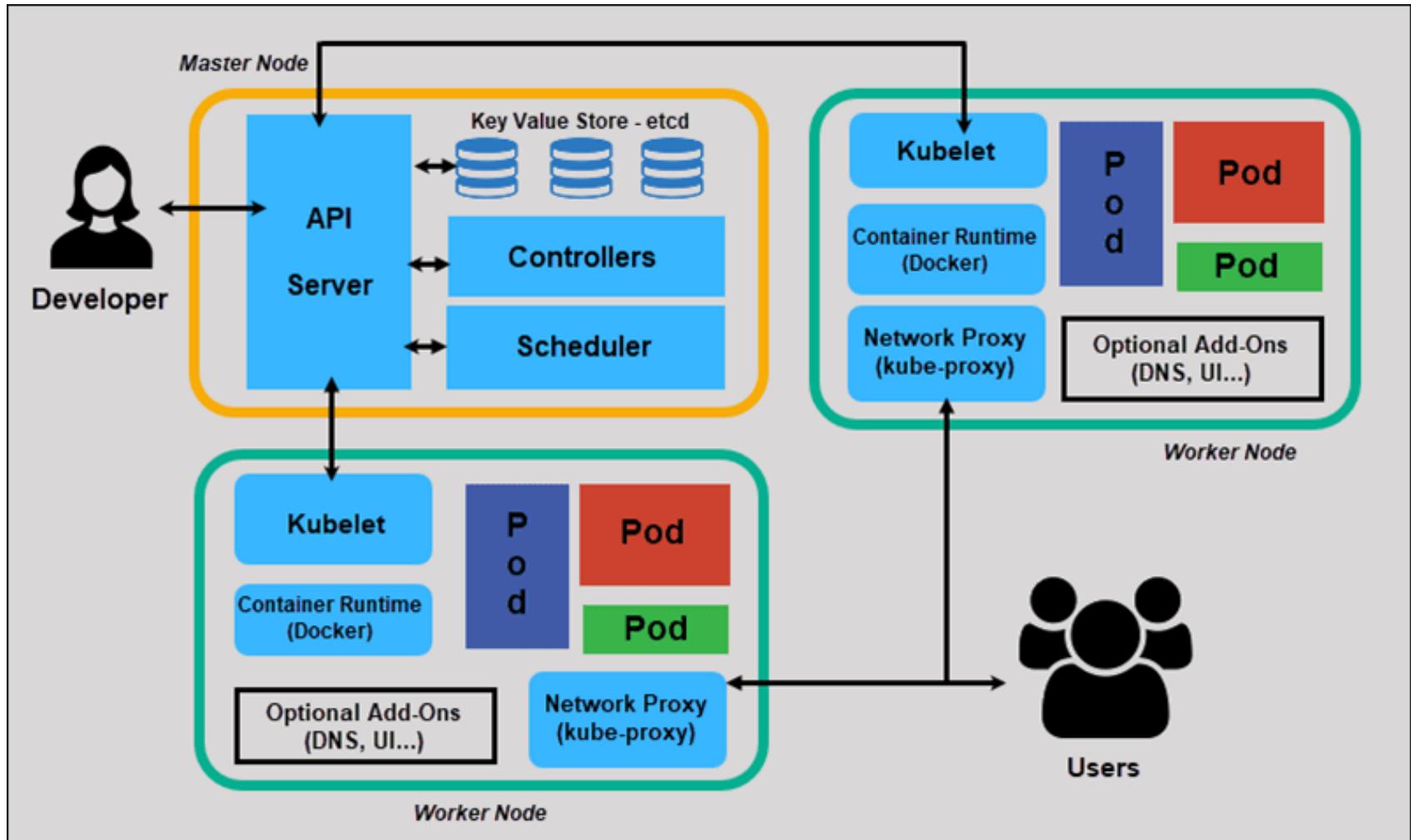




Kubernetes Architecture and Components

- These steps illustrate the basic Kubernetes process:
- An administrator creates and places the desired state of an application into a manifest file.
- The file is provided to the Kubernetes API Server using a CLI or UI.
- Kubernetes' default command-line tool is called kubectl.
- Kubernetes stores the file (an application's desired state) in a database called the Key-Value Store (etcd).
- Kubernetes then implements the desired state on all the relevant applications within the cluster.
- Kubernetes continuously monitors the elements of the cluster to make sure the current state of the application does not vary from the desired state.

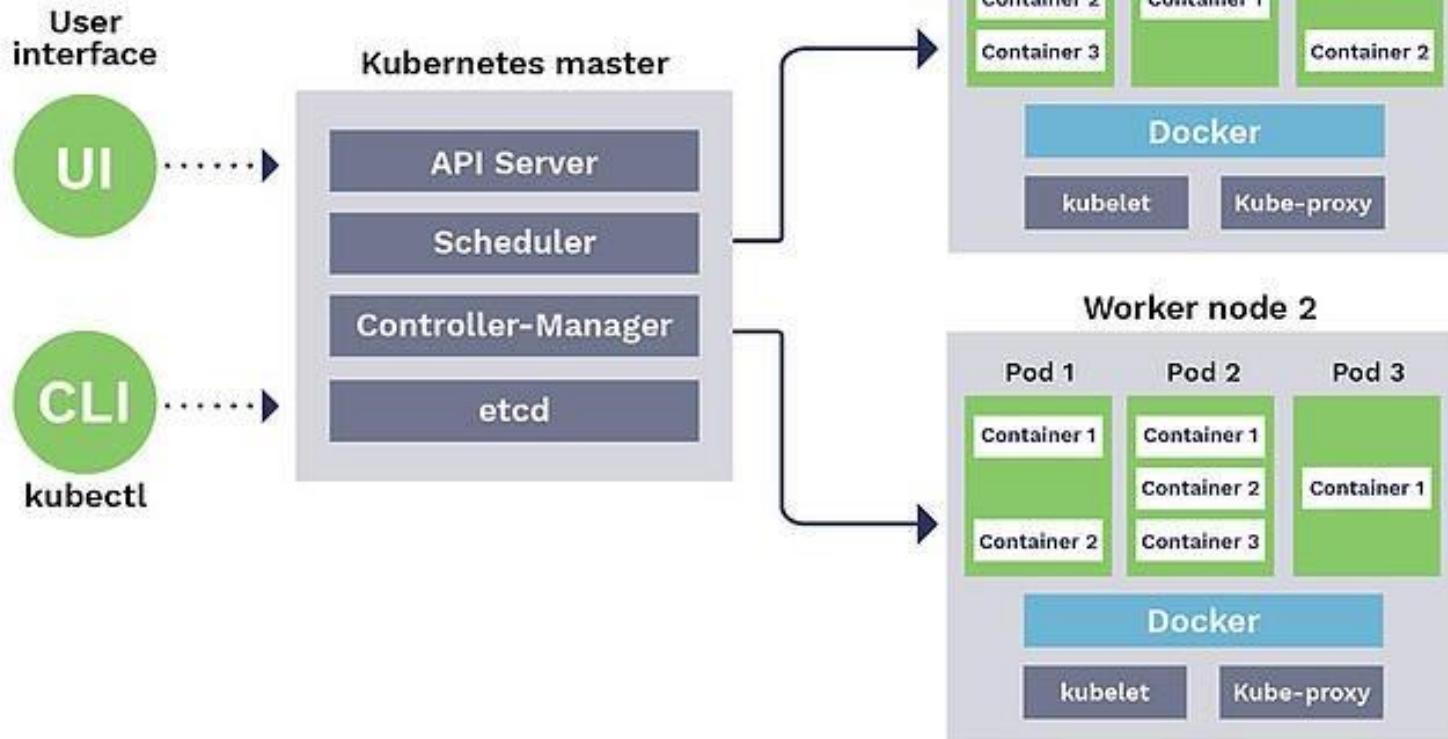
Kubernetes Architecture and Components



Kubernetes Architecture and Components

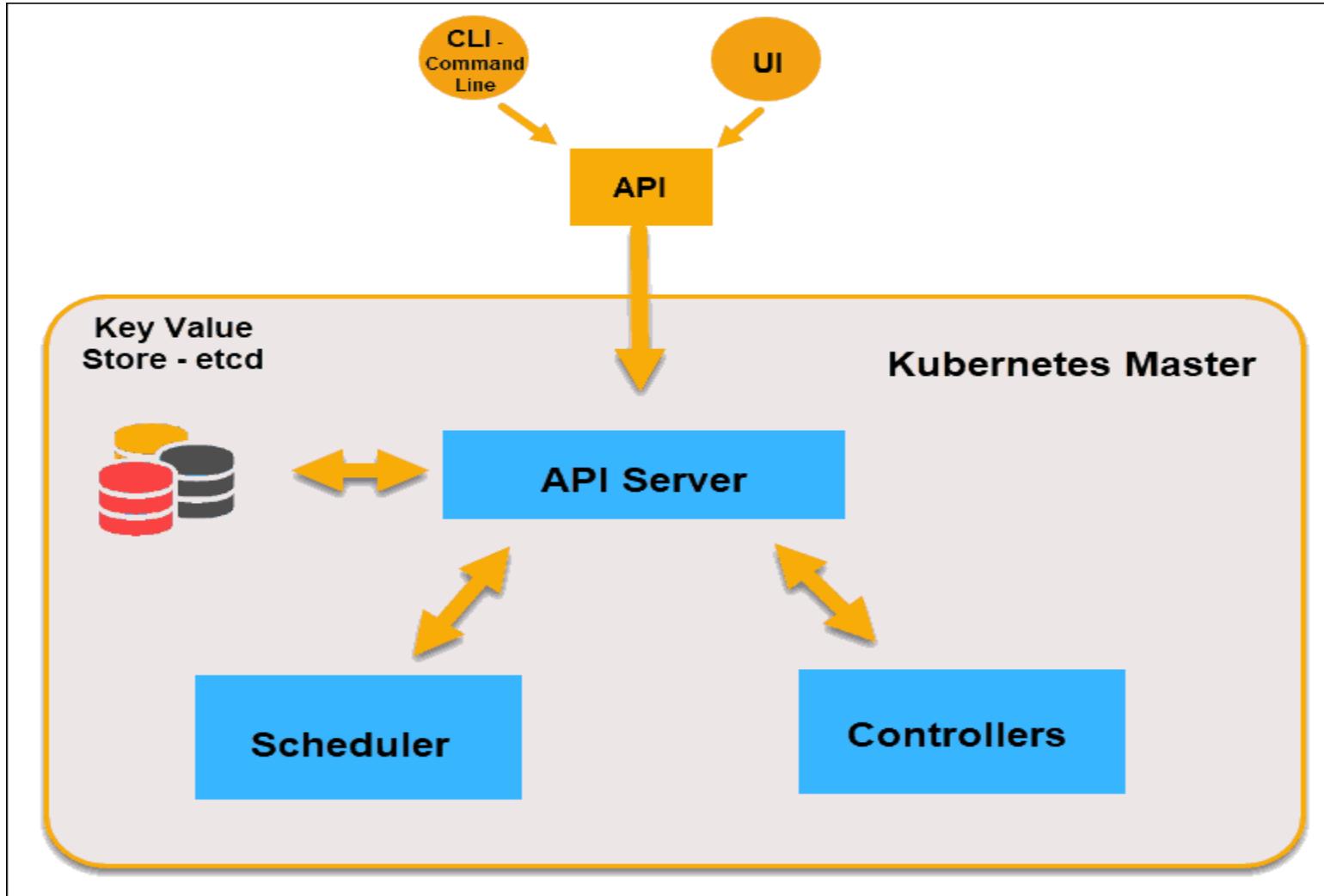


Kubernetes architecture





What is Master Node in Kubernetes Architecture?





API Server

- The API Server is the front-end of the control plane and the only component in the control plane that we interact with directly.
- Internal system components, as well as external user components, all communicate via the same API.



Key-Value Store (etcd)

- The Key-Value Store, also called etcd, is a database Kubernetes uses to back-up all cluster data.
- It stores the entire configuration and state of the cluster.
- The Master node queries etcd to retrieve parameters for the state of the nodes, pods, and containers.



Controller

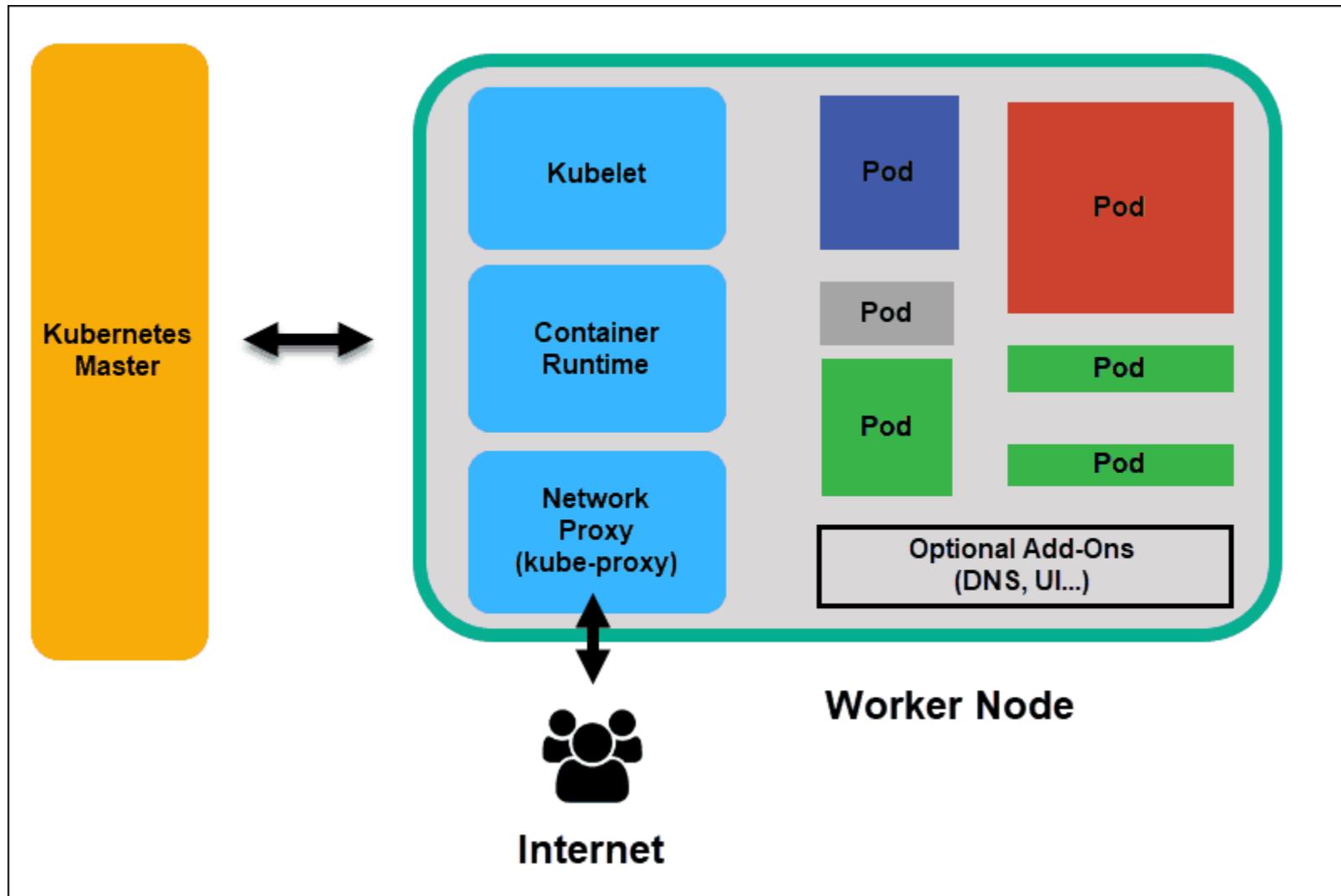
- The role of the Controller is to obtain the desired state from the API Server.
- It checks the current state of the nodes it is tasked to control, and determines if there are any differences, and resolves them, if any.

Scheduler



- A **Scheduler** watches for new requests coming from the API Server and assigns them to healthy nodes.
- It ranks the quality of the nodes and deploys pods to the best-suited node.
- If there are no suitable nodes, the pods are put in a pending state until such a node appears.

What is Worker Node in Kubernetes Architecture





Kubelet

- The kubelet runs on every node in the cluster.
- It is the principal Kubernetes agent.
- By installing kubelet, the node's CPU, RAM, and storage become part of the broader cluster.
- It watches for tasks sent from the API Server, executes the task, and reports back to the Master.
- It also monitors pods and reports back to the control panel if a pod is not fully functional.
- Based on that information, the Master can then decide how to allocate tasks and resources to reach the desired state.



Container Runtime

- The container runtime pulls images from a container image registry and starts and stops containers.
- A 3rd party software or plugin, such as Docker, usually performs this function.



Kube-proxy

- The kube-proxy makes sure that each node gets its IP address, implements local iptables and rules to handle routing and traffic load-balancing.



- A pod is the smallest element of scheduling in Kubernetes. Without it, a container cannot be part of a cluster. If you need to scale your app, you can only do so by adding or removing pods.
- The pod serves as a ‘wrapper’ for a single container with the application code. Based on the availability of resources, the Master schedules the pod on a specific node and coordinates with the container runtime to launch the container



Deployment

Scaling, updates, and rollbacks

Pod

Smallest Unit of Deployment in Kubernetes

Container

(Application Code)

Pod



- In instances where pods unexpectedly fail to perform their tasks, Kubernetes does not attempt to fix them.
- Instead, it creates and starts a new pod in its place.
- This new pod is a replica, except for the DNS and IP address.
- This feature has had a profound impact on how developers design applications.
- Due to the flexible nature of Kubernetes architecture, applications no longer need to be tied to a particular instance of a pod.
- Instead, applications need to be designed so that an entirely new pod, created anywhere within the cluster, can seamlessly take its place. To assist with this process, Kubernetes uses services



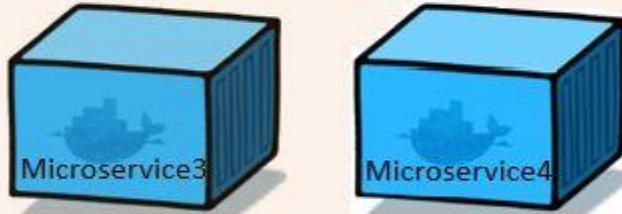
POD1



POD2



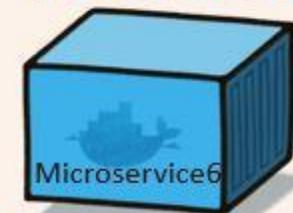
POD3



POD4



POD5



...

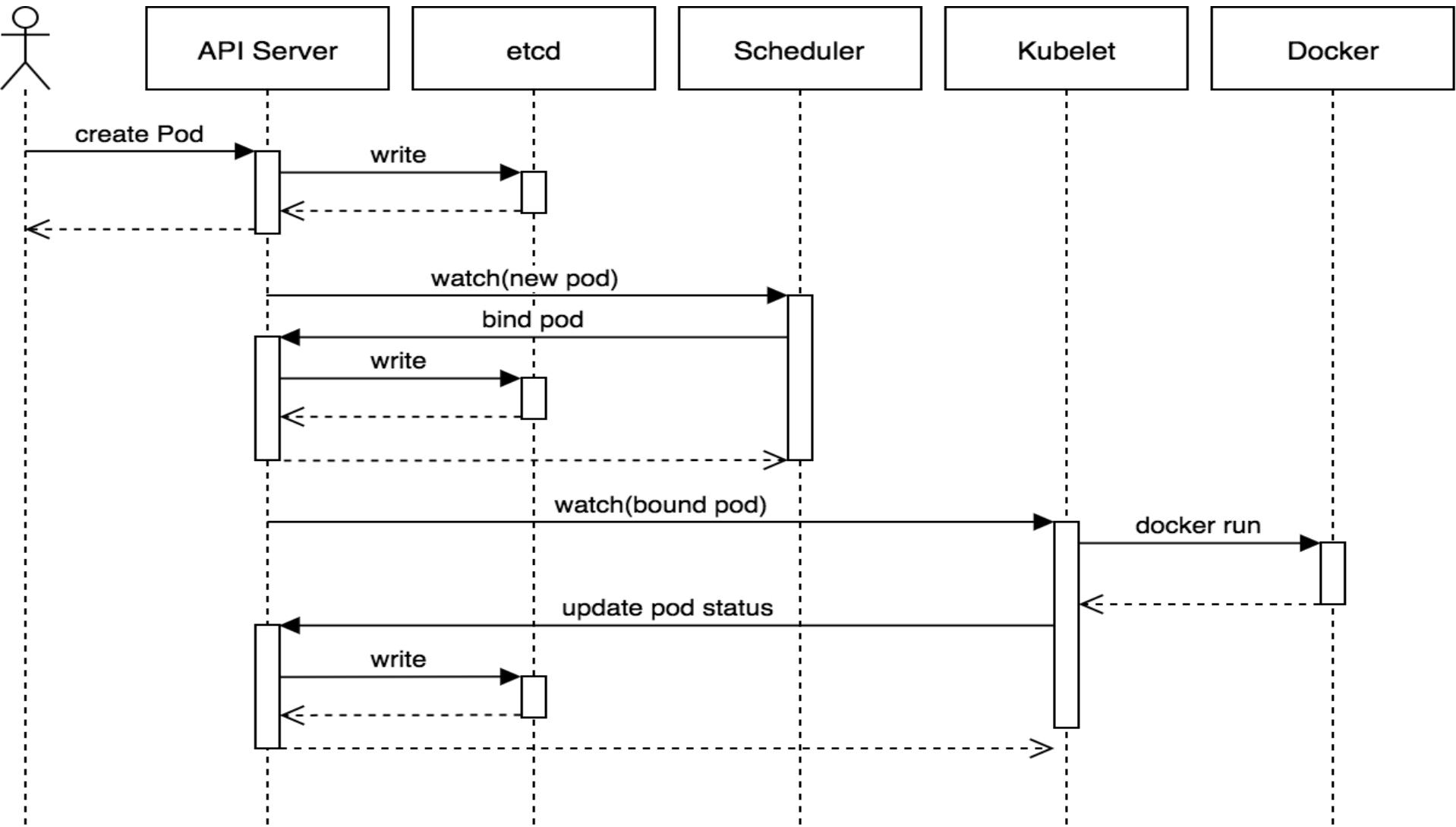


States of a Pod

- Through its lifecycle, a Pod can attain following states:
- Pending: The pod is accepted by the Kubernetes system but its container(s) is/are not created yet.
- Running: The pod is scheduled on a node and all its containers are created and at-least one container is in Running state.
- Succeeded: All container(s) in the Pod have exited with status 0 and will not be restarted.
- Failed: All container(s) of the Pod have exited and at least one container has returned a non-zero status.
- CrashLoopBackoff: The container fails to start and is tried again and again.

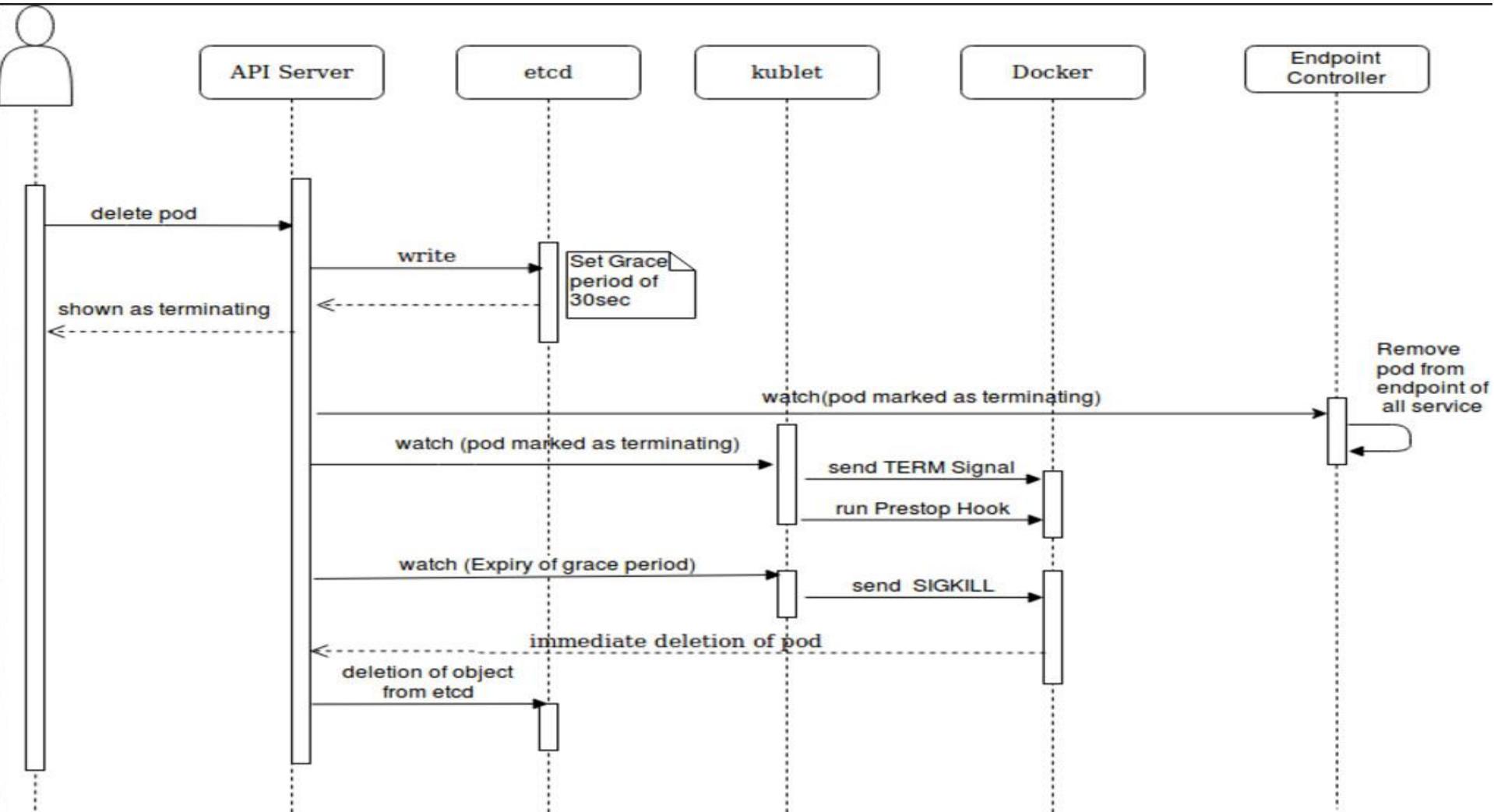


Birth of a Pod





Pod Termination





Kubernetes Services

- Pods are not constant. One of the best features Kubernetes offers is that non-functioning pods get replaced by new ones automatically.
- However, these new pods have a different set of IPs. It can lead to processing issues, and IP churn as the IPs no longer match. If left unattended, this property would make pods highly unreliable.
- Services are introduced to provide reliable networking by bringing stable IP addresses and DNS names to the unstable world of pods.
- By controlling traffic coming and going to the pod, a Kubernetes service provides a stable networking endpoint – a fixed IP, DNS, and port.
- Through a service, any pod can be added or removed without the fear that basic network information would change in any way.



How Do Kubernetes Services Work?

- Pods are associated with services through key-value pairs called labels and selectors.
- A service automatically discovers a new pod with labels that match the selector.
- This process seamlessly adds new pods to the service, and at the same time, removes terminated pods from the cluster.
- For example, if the desired state includes three replicas of a pod and a node running one replica fails, the current state is reduced to two pods.
- Kubernetes observes that the desired state is three pods.
- It then schedules one new replica to take the place of the failed pod and assigns it to another node in the cluster.



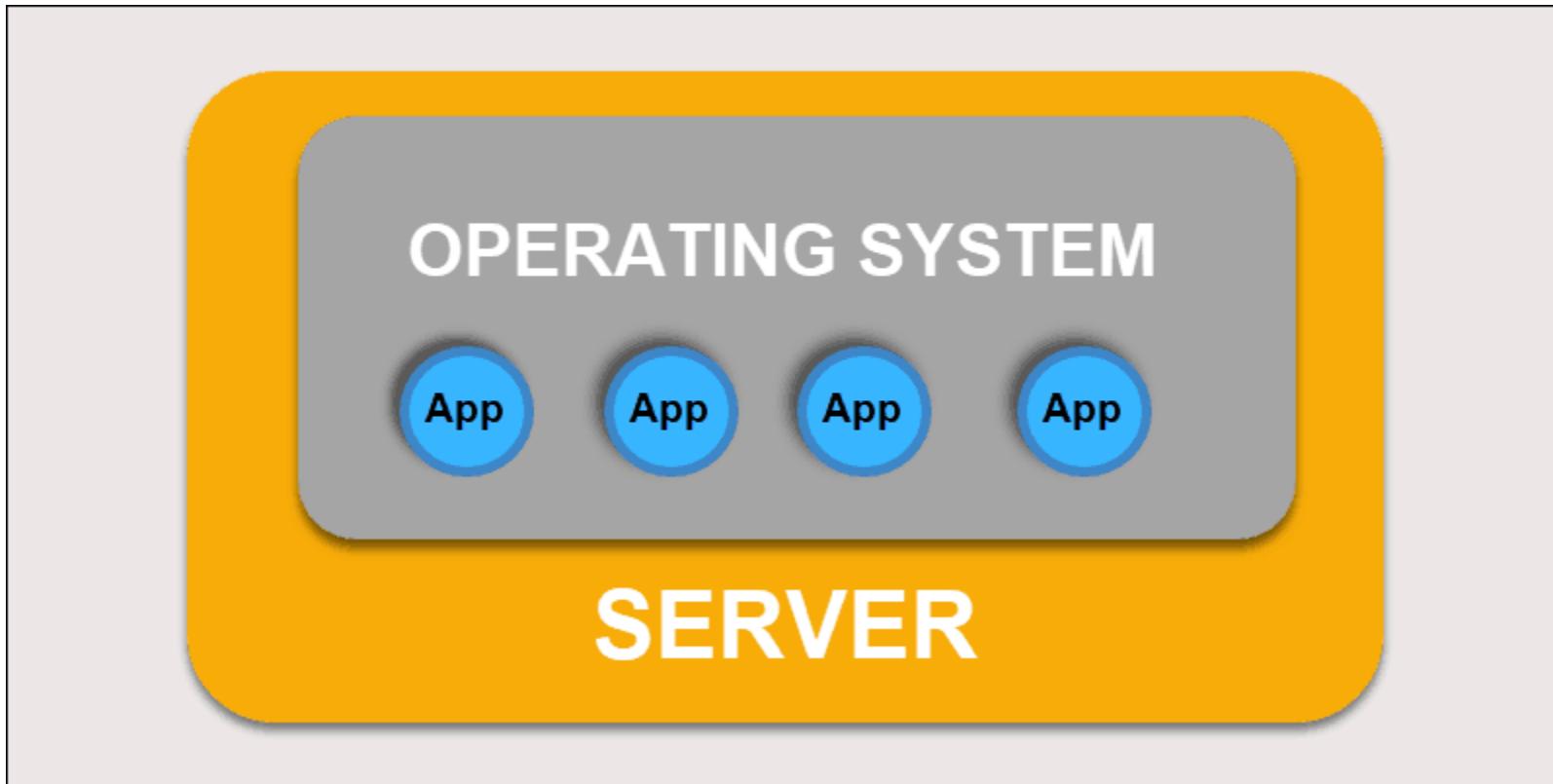
How Do Kubernetes Services Work?

- The same would apply when updating or scaling the application by adding or removing pods.
- Once we update the desired state, Kubernetes notices the discrepancy and adds or removes pods to match the manifest file.
- The Kubernetes control panel records, implements, and runs background reconciliation loops that continuously check to see if the environment matches user-defined requirements.



What is Container Deployment?

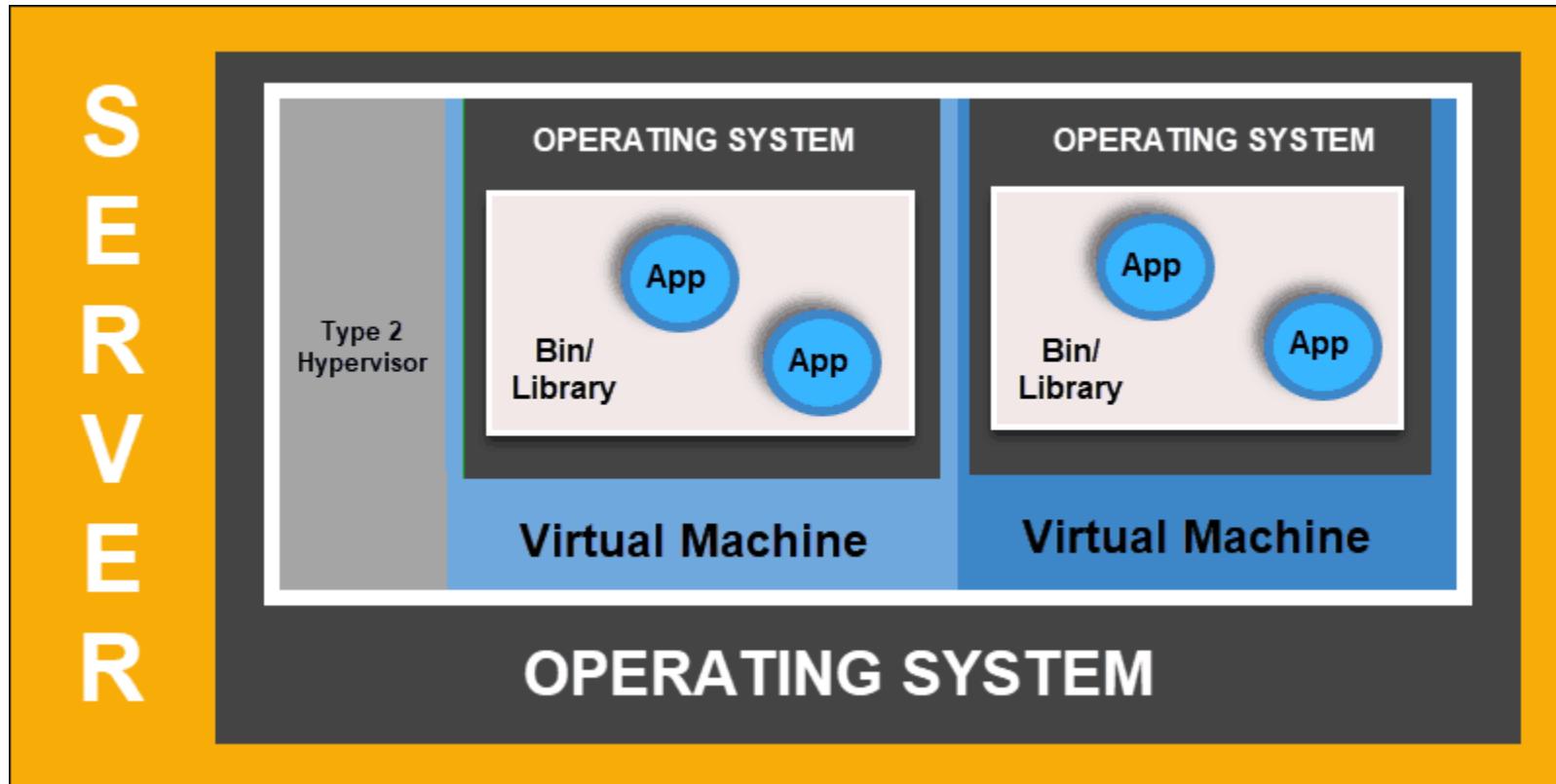
Traditional Deployment





What is Container Deployment?

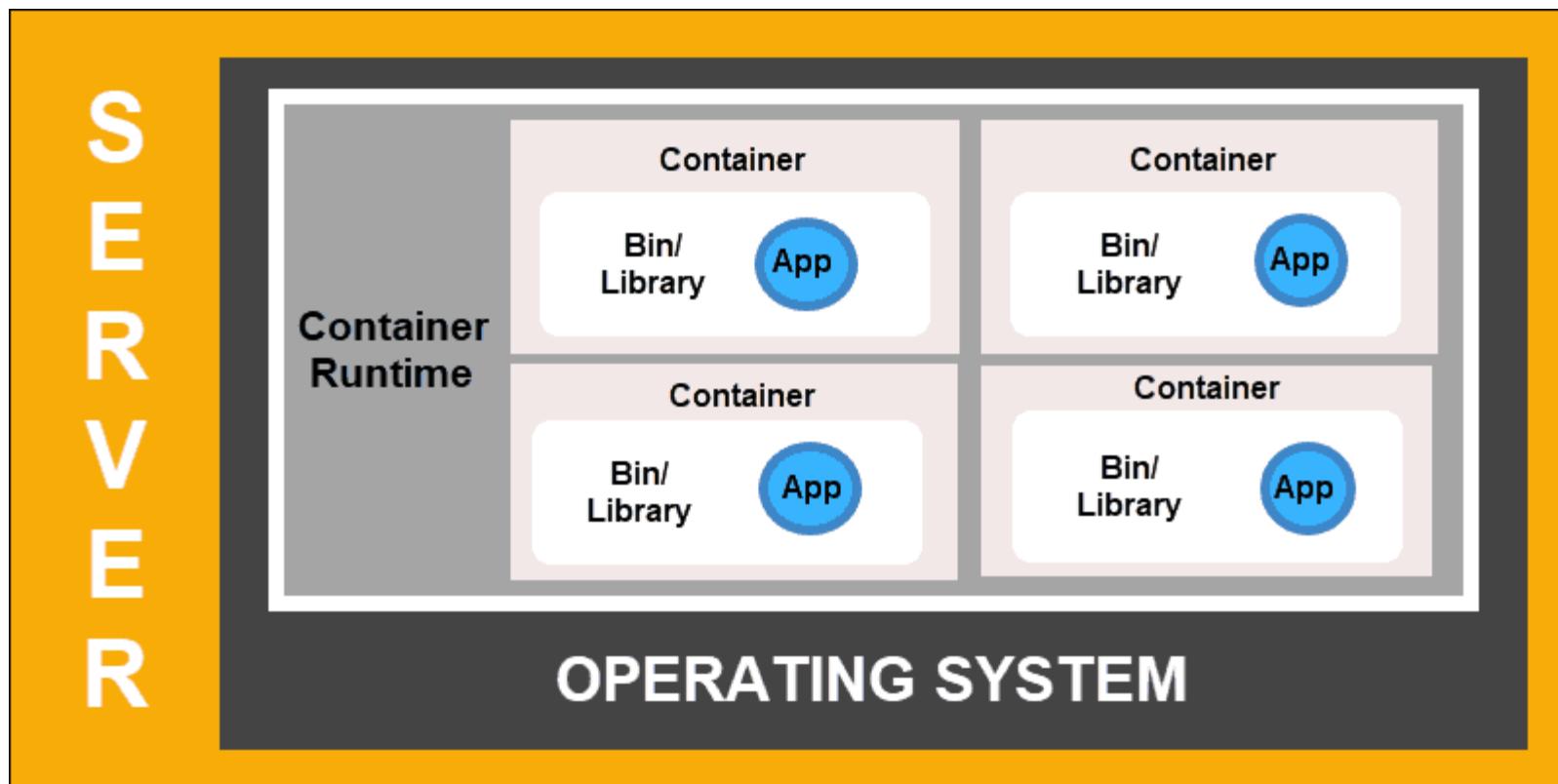
virtualized Deployment





What is Container Deployment?

ContainerDeployment





ReplicaSet

- A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time.
- As such, it is often used to guarantee the availability of a specified number of identical Pods.



ReplicaSet

- A ReplicaSet is defined with fields, including a selector that specifies how to identify Pods it can acquire, a number of replicas indicating how many Pods it should be maintaining, and a pod template specifying the data of new Pods it should create to meet the number of replicas criteria.
- A ReplicaSet then fulfills its purpose by creating and deleting Pods as needed to reach the desired number.
- When a ReplicaSet needs to create new Pods, it uses its Pod template.



ReplicaSet

- When to use a ReplicaSet
- A ReplicaSet ensures that a specified number of pod replicas are running at any given time.
- However, a Deployment is a higher-level concept that manages ReplicaSets and provides declarative updates to Pods along with a lot of other useful features.
- Therefore, we recommend using Deployments instead of directly using ReplicaSets, unless you require custom update orchestration or don't require updates at all.



controllers/frontend.yaml

```
apiVersion: apps/v1
kind: Replicaset
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # modify replicas according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3
```

kubectl get rs



What is a Deployment?

- A Deployment provides declarative updates for Pods and Replica Sets (the next-generation Replication Controller).
- You only need to describe the desired state in a Deployment object, and the Deployment controller will change the actual state to the desired state at a controlled rate for you.
- You can define Deployments to create new resources, or replace existing ones by new ones.



What is a Deployment?

- A typical use case is:
- Create a Deployment to bring up a Replica Set and Pods.
- Check the status of a Deployment to see if it succeeds or not.
- Later, update that Deployment to recreate the Pods (for example, to use a new image).
- Rollback to an earlier Deployment revision if the current Deployment isn't stable.
- Pause and resume a Deployment.



Creating a Deployment

- Here is an example Deployment. It creates a Replica Set to bring up 3 nginx Pods.
- ```
{% include code.html language="yaml" file="nginx-deployment.yaml"
ghlink="/docs/concepts/workloads/controllers/nginx-deployment.yaml" %}
```
- Run the example by downloading the example file and then running this command:
- ```
$ kubectl create -f docs/user-guide/nginx-deployment.yaml --record
```
- deployment "nginx-deployment" created



Creating a Deployment

- \$ kubectl get deployments
- | NAME | DESIRED | CURRENT | UP-TO-DATE |
|------------------|---------|---------|------------|
| AVAILABLE | AGE | | |
| nginx-deployment | 3 | 0 | 0 |
| | | | 1s |
- \$ kubectl get rs
- | NAME | DESIRED | CURRENT |
|-----------------------------|---------|---------|
| READY | AGE | |
| nginx-deployment-2035384211 | 3 | 3 |
| | | 0 |
| | 18s | |



Creating a Deployment

- \$ kubectl get pods --show-labels



Rolling Back a Deployment

- Sometimes you may want to rollback a Deployment; for example, when the Deployment is not stable, such as crash looping.
- By default, two previous Deployment's rollout history are kept in the system so that you can rollback anytime you want.



Rolling Back a Deployment

- Note: a Deployment's revision is created when a Deployment's rollout is triggered.
- This means that the new revision is created if and only if the Deployment's pod template (i.e. `.spec.template`) is changed, e.g. updating labels or container images of the template.
- Other updates, such as scaling the Deployment, will not create a Deployment revision -- so that we can facilitate simultaneous manual- or auto-scaling.
- This implies that when you rollback to an earlier revision, only the Deployment's pod template part will be rolled back.



Rolling Back a Deployment

- `kubectl set image deployment/nginx-deployment nginx=nginx:1.91`
- `kubectl rollout status deployments nginx-deployment`
- `kubectl rollout history deployment/nginx-deployment`
- `kubectl rollout history deployment/nginx-deployment --revision=2`



Scaling a Deployment

- `kubectl scale deployment nginx-deployment --replicas 10`
- `kubectl get deploy`
- `kubectl set image deploy/nginx-deployment nginx=nginx:sometag`



Working with Kubernetes Objects

- Kubernetes objects are persistent entities in the Kubernetes system.
- Kubernetes uses these entities to represent the state of your cluster.
- Specifically, they can describe:
- What containerized applications are running (and on which nodes)
- The resources available to those applications
- The policies around how those applications behave, such as restart policies, upgrades, and fault-tolerance



Init containers

- A Pod can have multiple containers running apps within it, but it can also have one or more init containers, which are run before the app containers are started.
- Init containers are exactly like regular containers, except:
- Init containers always run to completion.
- Each init container must complete successfully before the next one starts.
- If a Pod's init container fails, Kubernetes repeatedly restarts the Pod until the init container succeeds.
- However, if the Pod has a restartPolicy of Never, Kubernetes does not restart the Pod.



Differences from regular containers

- Init containers support all the fields and features of app containers, including resource limits, volumes, and security settings.
- However, the resource requests and limits for an init container are handled differently, as documented in Resources.
- Also, init containers do not support lifecycle, livenessProbe, readinessProbe, or startupProbe because they must run to completion before the Pod can be ready.



Create a Pod that has an Init Container

```
apiVersion: v1
kind: Pod
metadata:
  name: init-demo
spec:
  containers:
    - name: nginx
      image: nginx
      ports:
        - containerPort: 80
      volumeMounts:
        - name: workdir
          mountPath: /usr/share/nginx/html
    # These containers are run during pod initialization
  initContainers:
    - name: install
      image: busybox
      command:
        - wget
        - "-O"
        - "/work-dir/index.html"
        - http://kubernetes.io
      volumeMounts:
        - name: workdir
          mountPath: "/work-dir"
  dnsPolicy: Default
  volumes:
    - name: workdir
      emptyDir: {}
```



Working with Kubernetes Objects

- To work with Kubernetes objects--whether to create, modify, or delete them--you'll need to use the Kubernetes API.
- When you use the kubectl command-line interface, for example, the CLI makes the necessary Kubernetes API calls for you.



Object Spec and Status

- Almost every Kubernetes object includes two nested object fields that govern the object's configuration: the object spec and the object status.
- For objects that have a spec, you have to set this when you create the object, providing a description of the characteristics you want the resource to have: its desired state.
- The status describes the current state of the object, supplied and updated by the Kubernetes system and its components.
- The Kubernetes control plane continually and actively manages every object's actual state to match the desired state you supplied.



Object Spec and Status

- For example: in Kubernetes, a Deployment is an object that can represent an application running on your cluster.
- When you create the Deployment, you might set the Deployment spec to specify that you want three replicas of the application to be running.
- The Kubernetes system reads the Deployment spec and starts three instances of your desired application--updating the status to match your spec.
- If any of those instances should fail (a status change), the Kubernetes system responds to the difference between spec and status by making a correction--in this case, starting a replacement instance.



Describing a Kubernetes object

- Deployment.yaml
- apiVersion - Which version of the Kubernetes API you're using to create this object
- kind - What kind of object you want to create
- metadata - Data that helps uniquely identify the object, including a name string, UID, and optional namespace
- spec - What state you desire for the object



Network Terms

- Layer 2 Networking
- Layer 2 is the data link layer providing Node-to-Node data transfer.
- It defines the protocol to establish and terminate a connection between two physically connected devices. It also defines the protocol for flow control between them.



Network Terms

- Layer 4 Networking
- The transport layer controls the reliability of a given link through flow control. In TCP/IP, this layer refers to the TCP protocol for exchanging data over an unreliable network.



Network Terms

- Layer 7 Networking
- The application layer is the layer closest to the end user, which means both the application layer and the user interact directly with the software application.
- This layer interacts with software applications that implement a communicating component. Typically, Layer 7 Networking refers to HTTP.



Network Terms

- NAT — Network Address Translation
- NAT or network address translation is an IP-level remapping of one address space into another. The mapping happens by modifying network address information in the IP header of packets while they are in transit across a traffic routing device.



Network Terms

- SNAT — Source Network Address Translation
- SNAT simply refers to a NAT procedure that modifies the source address of an IP packet. This is the typical behaviour for the NAT described above.
- **DNAT — Destination Network Address Translation**
- DNAT refers to a NAT procedure that modifies the destination address of an IP packet. DNAT is used to publish a service resting in a private network to a publicly addressable IP address.



Network Terms

- Network Namespace
- In networking, each machine (real or virtual) has an Ethernet device (that we will refer to as eth0).
- All traffic flowing in and out of the machine is associated with that device.
- In truth, Linux associates each Ethernet device with a network namespace — a logical copy of the entire network stack, with its own routes, firewall rules, and network devices.
- Initially, all the processes share the same default network namespace from the init process, called the root namespace.
- By default, a process inherits its network namespace from its parent and so, if you don't make any changes, all network traffic flows through the Ethernet device specified for the root network namespace.



Network Terms

- veth — Virtual Ethernet Device Pairs
- Computer systems typically consist of one or more networking devices — eth0, eth1, etc — that are associated with a physical network adapter which is responsible for placing packets onto the physical wire.
- Veth devices are virtual network devices that are always created in interconnected pairs.
- They can act as tunnels between network namespaces to create a bridge to a physical network device in another namespace, but can also be used as standalone network devices.



Network Terms

- bridge — Network Bridge
- A network bridge is a device that creates a single aggregate network from multiple communication networks or network segments.
- Bridging connects two separate networks as if they were a single network.
- Bridging uses an internal data structure to record the location that each packet is sent to as a performance optimization.



Network Terms

- **iptables** — Packet Mangling Tool
- **iptables** is a program that allows a Linux system administrator to configure the netfilter and the chains and rules it stores.
- Each rule within an IP table consists of a number of classifiers (**iptables** matches) and one connected action (**iptables** target).



Network Terms

- IPVS — IP Virtual Server
- IPVS implements transport-layer load balancing as part of the Linux kernel.
- IPVS is a tool similar to iptables. It is based on the Linux kernel's netfilter hook function, but uses a hash table as the underlying data structure.
- That means, when compared to iptables, IPVS redirects traffic much faster, has much better performance when syncing proxy rules, and provides more load balancing algorithms.



Network Terms

- DNS — The Domain Name System
- The Domain Name System (DNS) is a decentralized naming system for associating system names with IP addresses.
- It translates domain names to numerical IP addresses for locating computer services.



The Kubernetes Networking Model

- Kubernetes makes opinionated choices about how Pods are networked. In particular, Kubernetes dictates the following requirements on any networking implementation:
 - all Pods can communicate with all other Pods without using network address translation (NAT).
 - all Nodes can communicate with all Pods without NAT.
 - the IP that a Pod sees itself as is the same IP that others see it as.

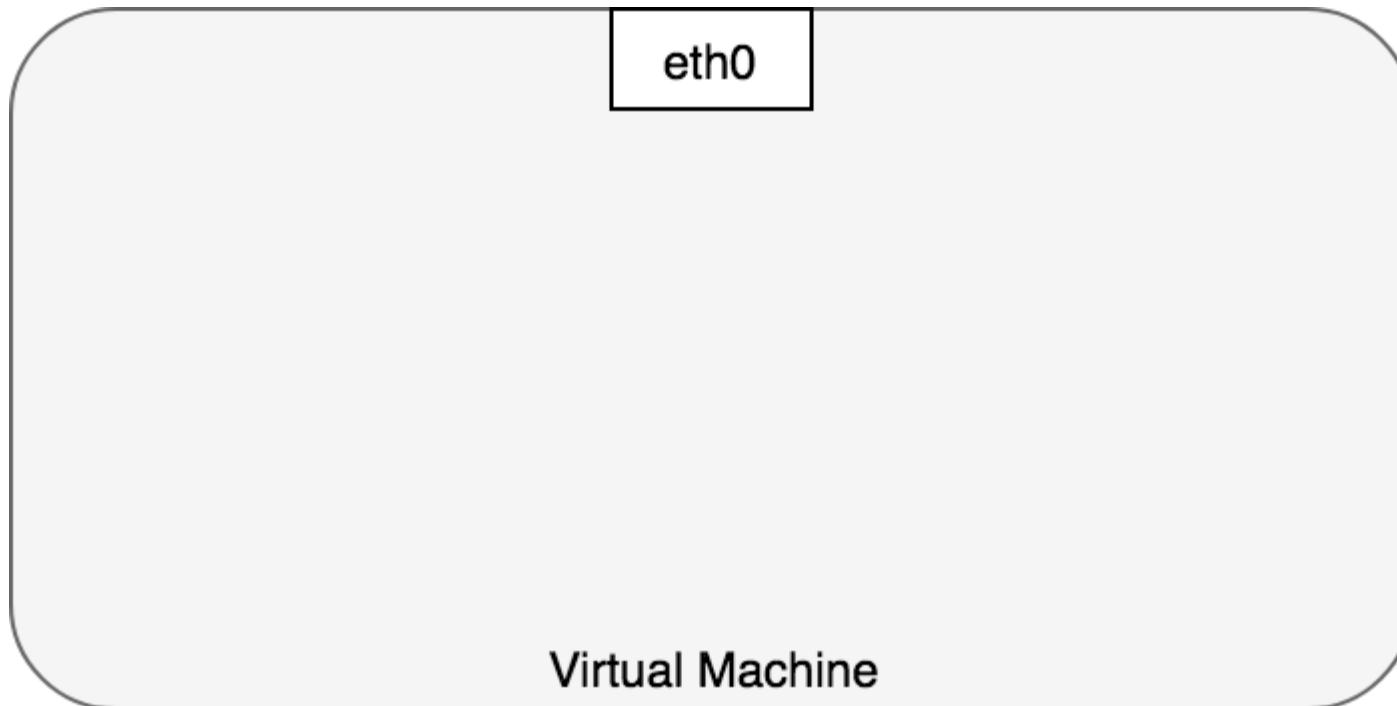


The Kubernetes Networking Model

- Given these constraints, we are left with four distinct networking problems to solve:
 - Container-to-Container networking
 - Pod-to-Pod networking
 - Pod-to-Service networking
 - Internet-to-Service networking



Container-to-Container Networking



- **Typically, we view network communication in a virtual machine as interacting directly with an Ethernet device,**



Container-to-Container Networking

- Linux, each running process communicates within a network namespace that provides a logical networking stack with its own routes, firewall rules, and network devices.
- In essence, a network namespace provides a brand new network stack for all the processes within the namespace.
- As a Linux user, network namespaces can be created using the ip command.
- For example, the following command will create a new network namespace called ns1.
- **ip netns add ns1**



Container-to-Container Networking

- When the namespace is created, a mount point for it is created under `/var/run/netns`, allowing the namespace to persist even if there is no process attached to it.
- You can list available namespaces by listing all the mount points under `/var/run/netns`, or by using the `ip` command.
- `$ ls /var/run/netns`
- `ns1`
- `$ ip netns`
- `ns1`



eswaribala@DESKTOP-55AGI0I: ~

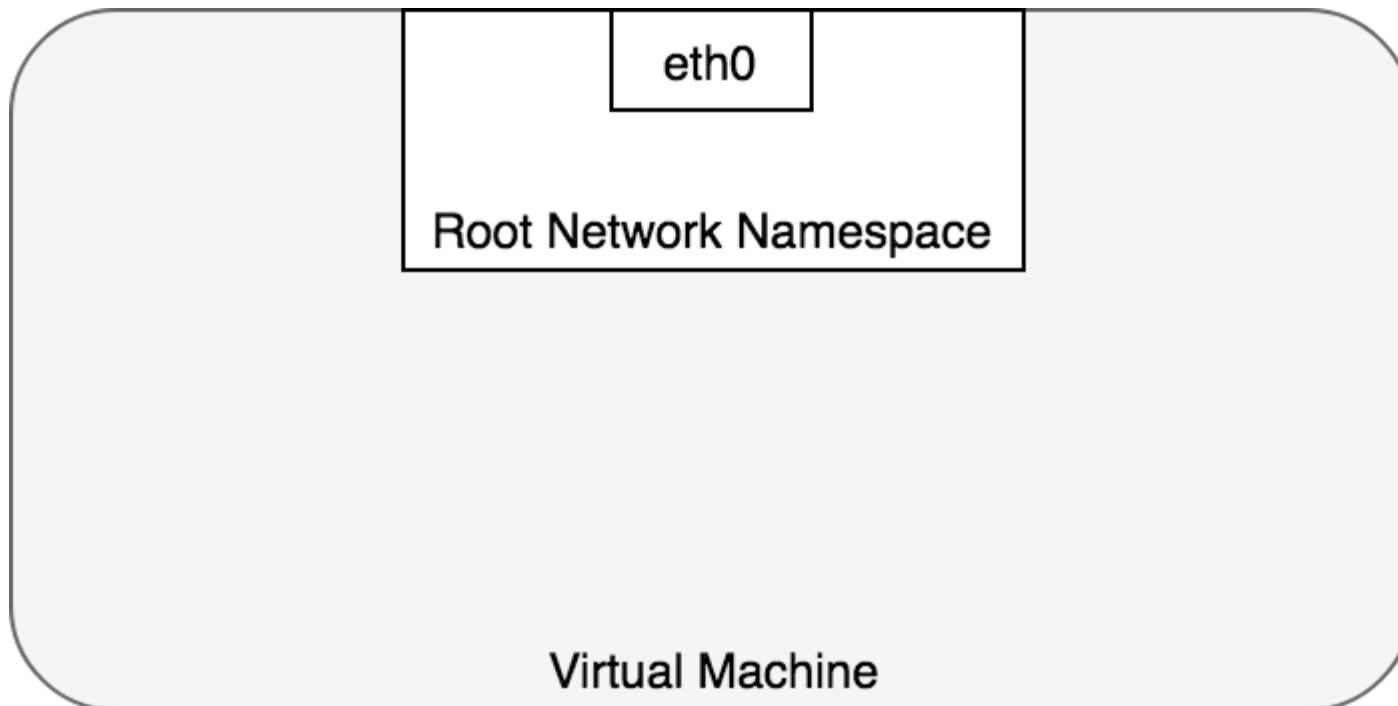
This message is shown once once a day. To disable it please create the
/home/eswaribala/.hushlogin file.

```
eswaribala@DESKTOP-55AGI0I:~$ ip netns add ns1
mkdir /run/netns failed: Permission denied
eswaribala@DESKTOP-55AGI0I:~$ sudo ip netns add ns1
[sudo] password for eswaribala:
eswaribala@DESKTOP-55AGI0I:~$ ls /var/run/netns
ns1
eswaribala@DESKTOP-55AGI0I:~$ ip netns
ns1
eswaribala@DESKTOP-55AGI0I:~$
```



Root Network Namespace

- By default, Linux assigns every process to the root network namespace to provide access to the external world



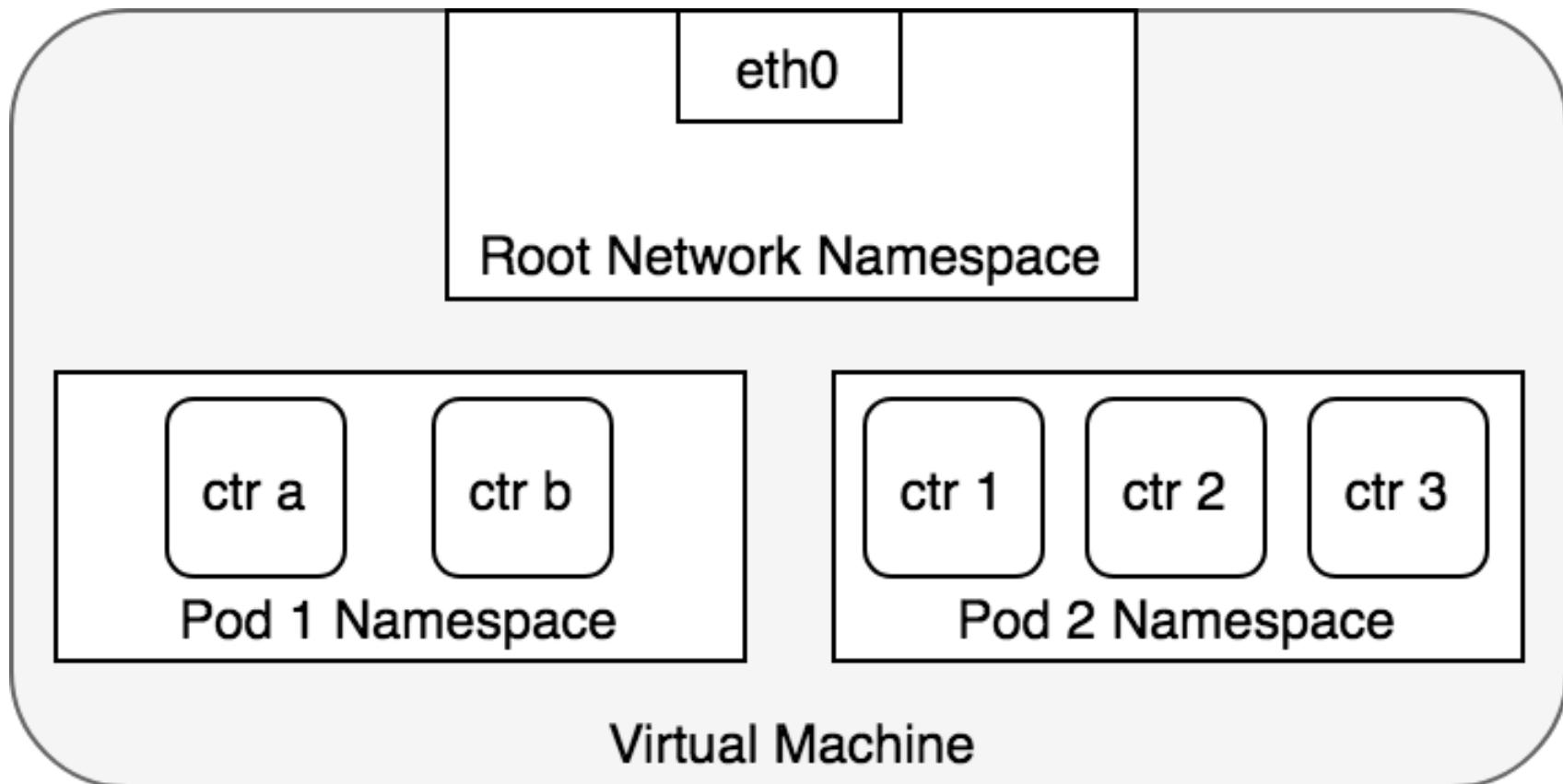


Network Namespace/pod

- In terms of Docker constructs, a Pod is modelled as a group of Docker containers that share a network namespace.
- Containers within a Pod all have the same IP address and port space assigned through the network namespace assigned to the Pod, and can find each other via localhost since they reside in the same namespace.
- We can create a network namespace for each Pod on a virtual machine.
- This is implemented, using Docker, as a “Pod container” which holds the network namespace open while “app containers” (the things the user specified) join that namespace with Docker’s `–net=container:` function.

Network Namespace/pod

Applications within a Pod also have access to shared volumes, which are defined as part of a Pod and are made available to be mounted into each application's filesystem.





Pod-to-Pod Networking

- In Kubernetes, every Pod has a real IP address and each Pod communicates with other Pods using that IP address.
- The task at hand is to understand how Kubernetes enables Pod-to-Pod communication using real IPs, whether the Pod is deployed on the same physical Node or different Node in the cluster.
- We are considering that pods reside on the same machine to avoid the complications of going over the internal network to communicate across Nodes.

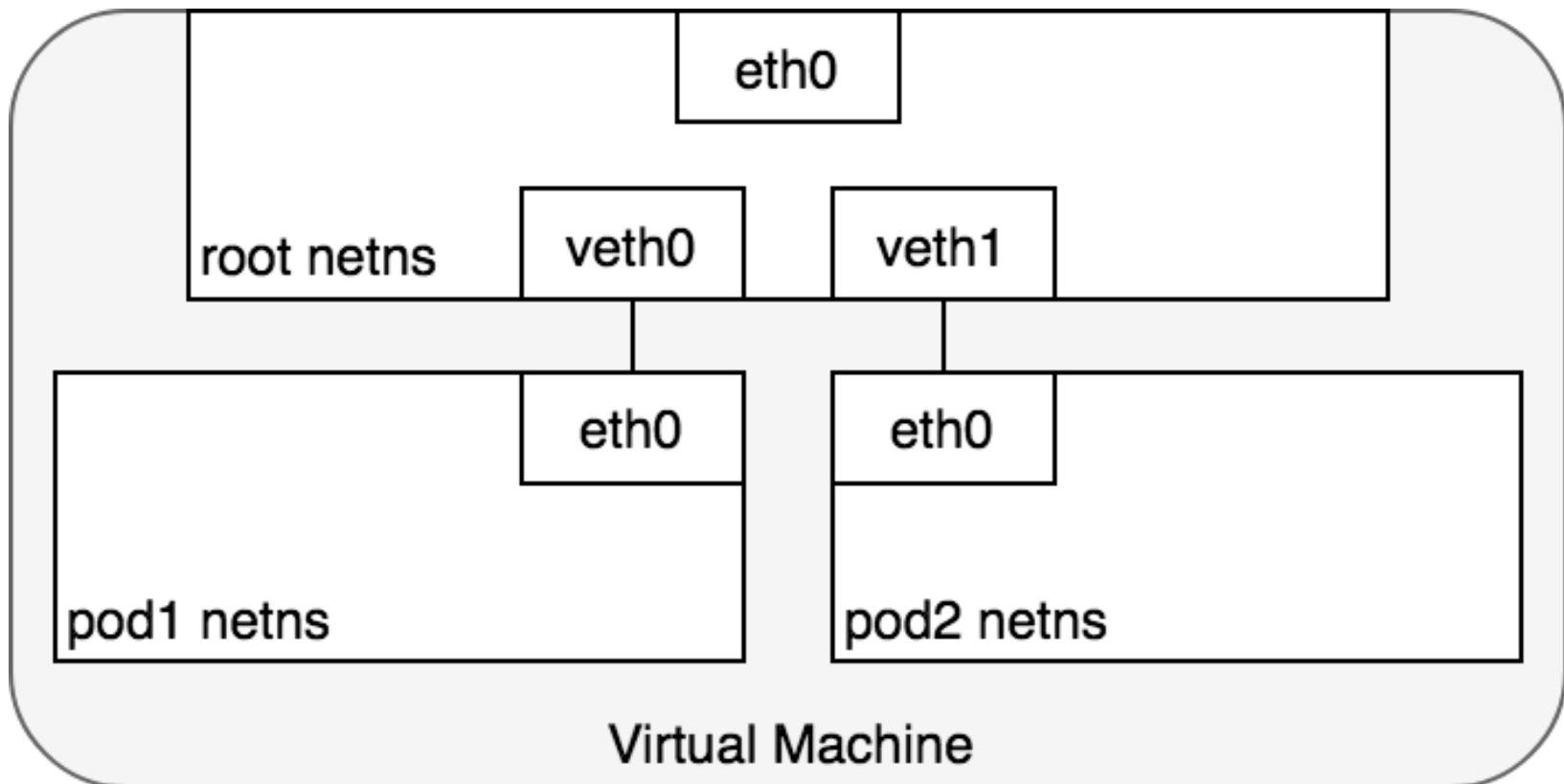


Pod-to-Pod Networking

- From the Pod's perspective, it exists in its own Ethernet namespace that needs to communicate with other network namespaces on the same Node.
- Namespaces can be connected using a Linux Virtual Ethernet Device or veth pair consisting of two virtual interfaces that can be spread over multiple namespaces.
- To connect Pod namespaces, we can assign one side of the veth pair to the root network namespace, and the other side to the Pod's network namespace.
- Each veth pair works like a patch cable, connecting the two sides and allowing traffic to flow between them.
- This setup can be replicated for as many Pods as we have on the machine.



Pod-to-Pod Networking





Pod-to-Pod Networking

- At this point, we've set up the Pods to each have their own network namespace so that they believe they have their own Ethernet device and IP address, and they are connected to the root namespace for the Node.
- Now, we want the Pods to talk to each other through the root namespace, and for this we use a network bridge.



Pod-to-Pod Networking

- A Linux Ethernet bridge is a virtual Layer 2 networking device used to unite two or more network segments, working transparently to connect two networks together.
- The bridge operates by maintaining a forwarding table between sources and destinations by examining the destination of the data packets that travel through it and deciding whether or not to pass the packets to other network segments connected to the bridge.
- The bridging code decides whether to bridge data or to drop it by looking at the MAC-address unique to each Ethernet device in the network.

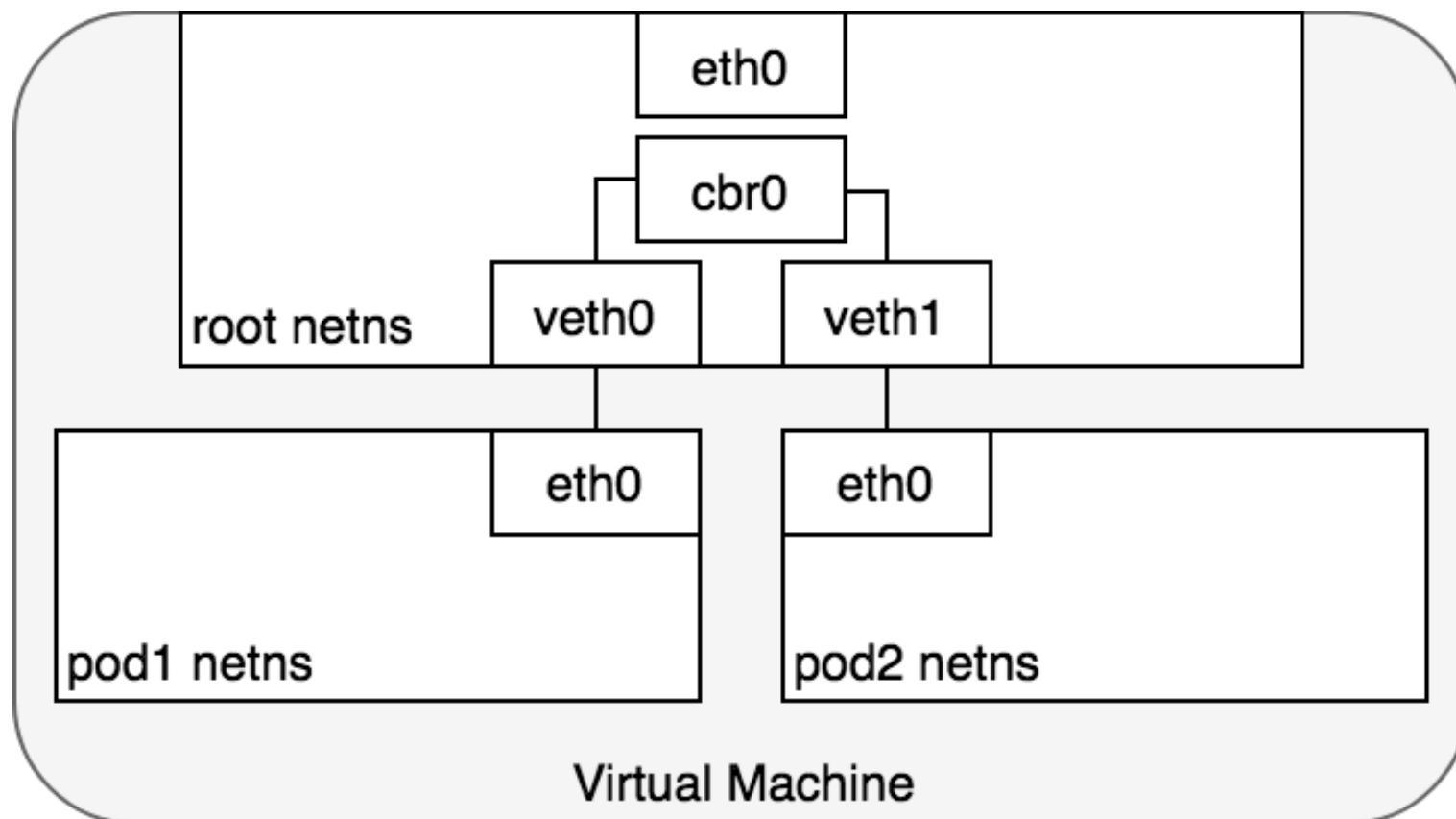


Pod-to-Pod Networking

- Bridges implement the ARP protocol to discover the link-layer MAC address associated with a given IP address.
- When a data frame is received at the bridge, the bridge broadcasts the frame out to all connected devices (except the original sender) and the device that responds to the frame is stored in a lookup table.
- Future traffic with the same IP address uses the lookup table to discover the correct MAC address to forward the packet to.

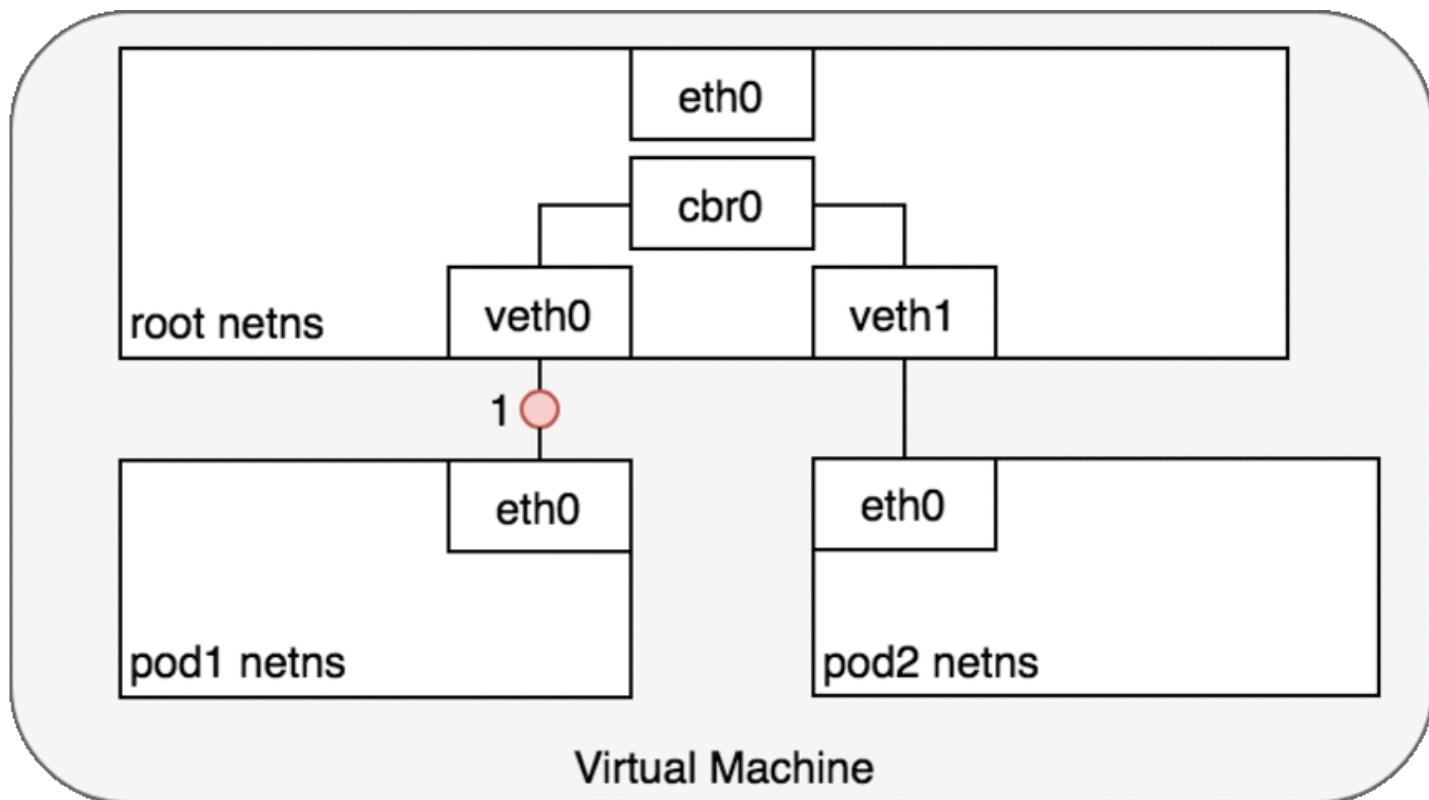


Pod-to-Pod Networking



Life of a Packet

src: pod1
dst: pod2



Virtual Machine



Life of a Packet

- Pod 1 sends a packet to its own Ethernet device eth0 which is available as the default device for the Pod.
- For Pod 1, eth0 is connected via a virtual Ethernet device to the root namespace, veth0 (1).
- The bridge cbr0 is configured with veth0 a network segment attached to it.
- Once the packet reaches the bridge, the bridge resolves the correct network segment to send the packet to — veth1 using the ARP protocol (3).



Life of a Packet

- When the packet reaches the virtual device veth1, it is forwarded directly to Pod 2's namespace and the eth0 device within that namespace (4).
- Throughout this traffic flow, each Pod is communicating only with eth0 on localhost and the traffic is routed to the correct Pod.
- The development experience for using the network is the default behaviour that a developer would expect.



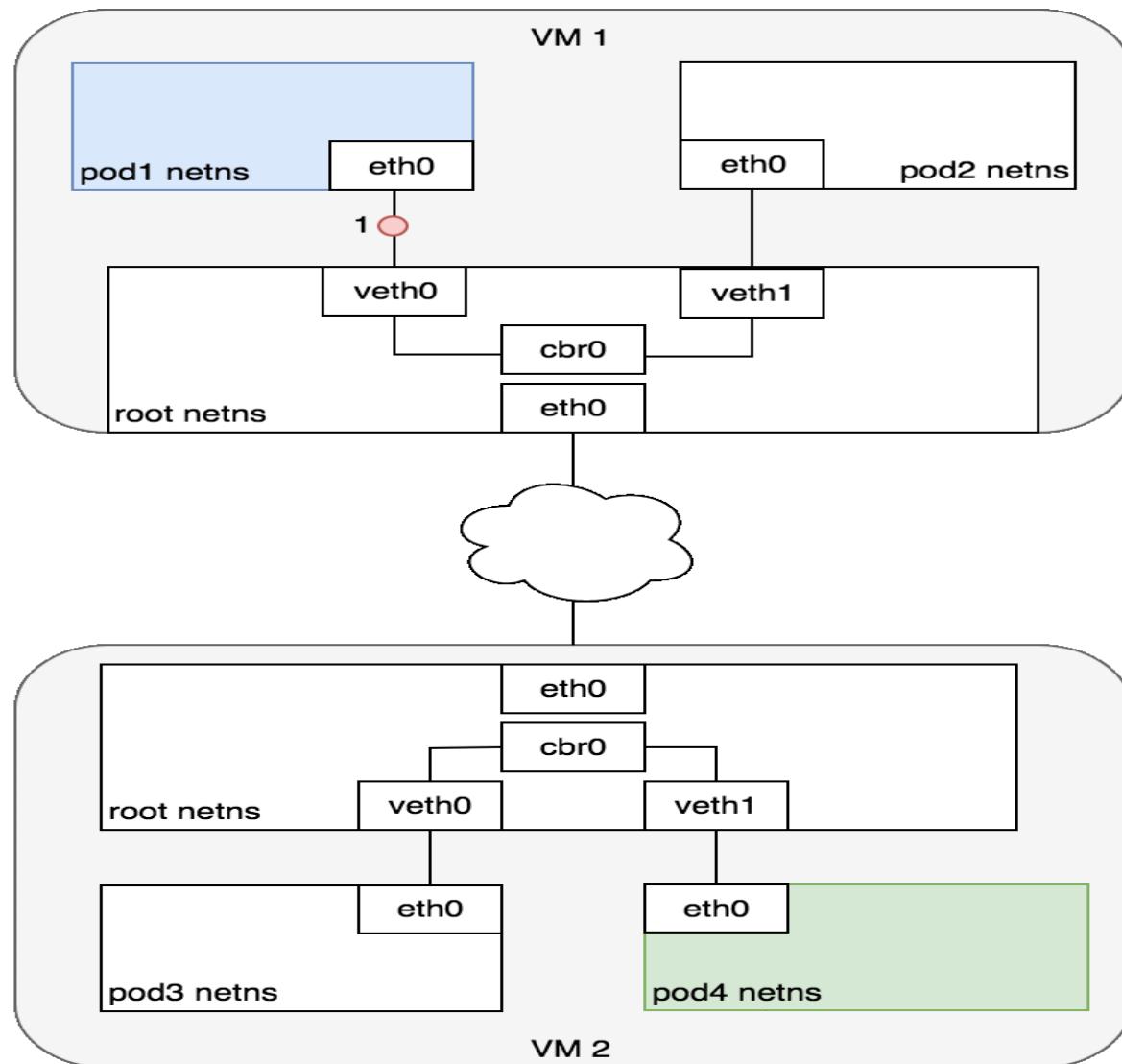
Life of a Packet

- Kubernetes' networking model dictates that Pods must be reachable by their IP address across Nodes.
- That is, the IP address of a Pod is always visible to other Pods in the network, and each Pod views its own IP address as the same as how other Pods see it.
- We now turn to the problem of routing traffic between Pods on different Nodes.

Life of a packet: Pod-to-Pod, across Nodes



src: pod1
dst: pod4





Pod-to-Service Networking

- A Kubernetes Service manages the state of a set of Pods, allowing you to track a set of Pod IP addresses that are dynamically changing over time.
- Services act as an abstraction over Pods and assign a single virtual IP address to a group of Pod IP addresses.
- Any traffic addressed to the virtual IP of the Service will be routed to the set of Pods that are associated with the virtual IP.
- This allows the set of Pods associated with a Service to change at any time — clients only need to know the Service's virtual IP, which does not change.



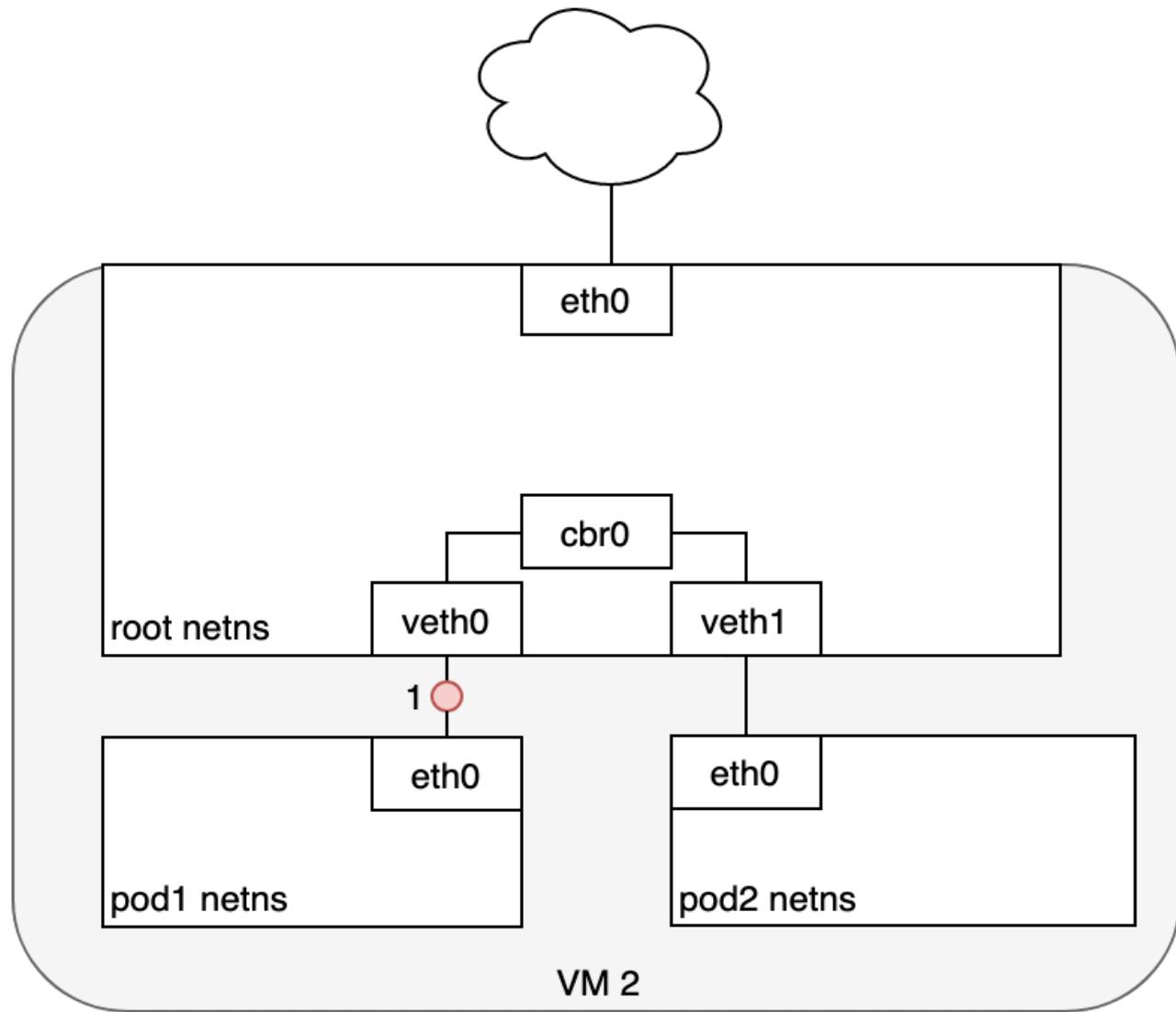
Pod-to-Service Networking

- When creating a new Kubernetes Service, a new virtual IP (also known as a cluster IP) is created on your behalf.
- Anywhere within the cluster, traffic addressed to the virtual IP will be load-balanced to the set of backing Pods associated with the Service.
- In effect, Kubernetes automatically creates and maintains a distributed in-cluster load balancer that distributes traffic to a Service's associated healthy Pods.

Pod to Service



src: pod1
dst: svc1





Pod to Service

- The packet first leaves the Pod through the eth0 interface attached to the Pod's network namespace (1).
- Then it travels through the virtual Ethernet device to the bridge (2).
- The ARP protocol running on the bridge does not know about the Service and so it transfers the packet out through the default route — eth0 (3).
- Here, something different happens. Before being accepted at eth0, the packet is filtered through iptables.
- After receiving the packet, iptables uses the rules installed on the Node by kube-proxy in response to Service or Pod events to rewrite the destination of the packet from the Service IP to a specific Pod IP (4).

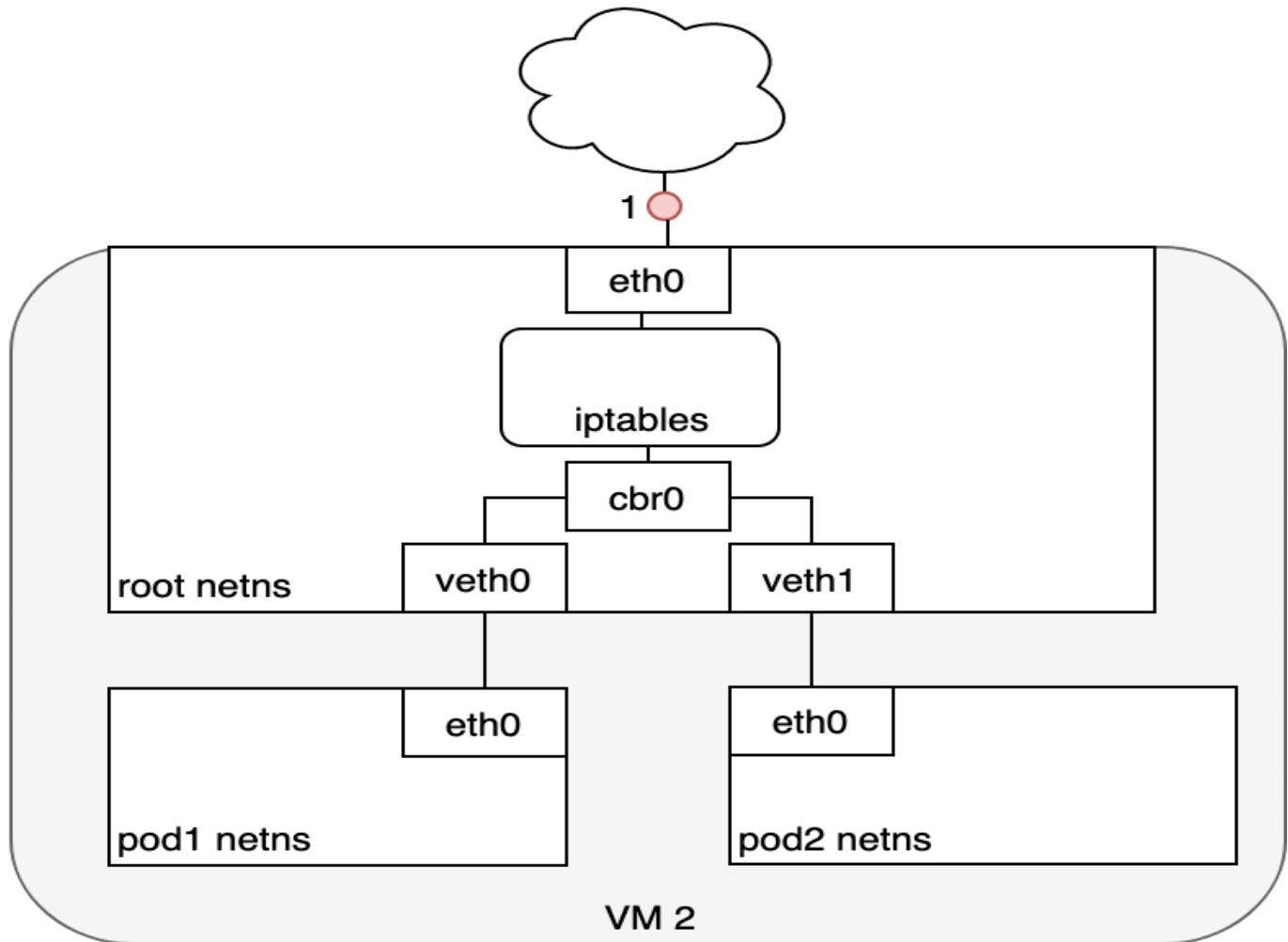


Pod to Service

- The packet is now destined to reach Pod 4 rather than the Service's virtual IP.
- The Linux kernel's conntrack utility is leveraged by iptables to remember the Pod choice that was made so future traffic is routed to the same Pod (barring any scaling events).
- In essence, iptables has done in-cluster load balancing directly on the Node. Traffic then flows to the Pod using the Pod-to-Pod routing we've already examined (5).



Service to POD



src: pod4
dst: pod1



Using DNS

- Kubernetes can optionally use DNS to avoid having to hard-code a Service's cluster IP address into your application.
- Kubernetes DNS runs as a regular Kubernetes Service that is scheduled on the cluster.
- It configures the kubelets running on each Node so that containers use the DNS Service's IP to resolve DNS names.
- Every Service defined in the cluster (including the DNS server itself) is assigned a DNS name.
- DNS records resolve DNS names to the cluster IP of the Service or the IP of a POD, depending on your needs.
- SRV records are used to specify particular named ports for running Services.



Using DNS

- A DNS Pod consists of three separate containers:
- `kubedns`: watches the Kubernetes master for changes in Services and Endpoints, and maintains in-memory lookup structures to serve DNS requests.
- `dnsmasq`: adds DNS caching to improve performance.
- `sidecar`: provides a single health check endpoint to perform healthchecks for `dnsmasq` and `kubedns`.

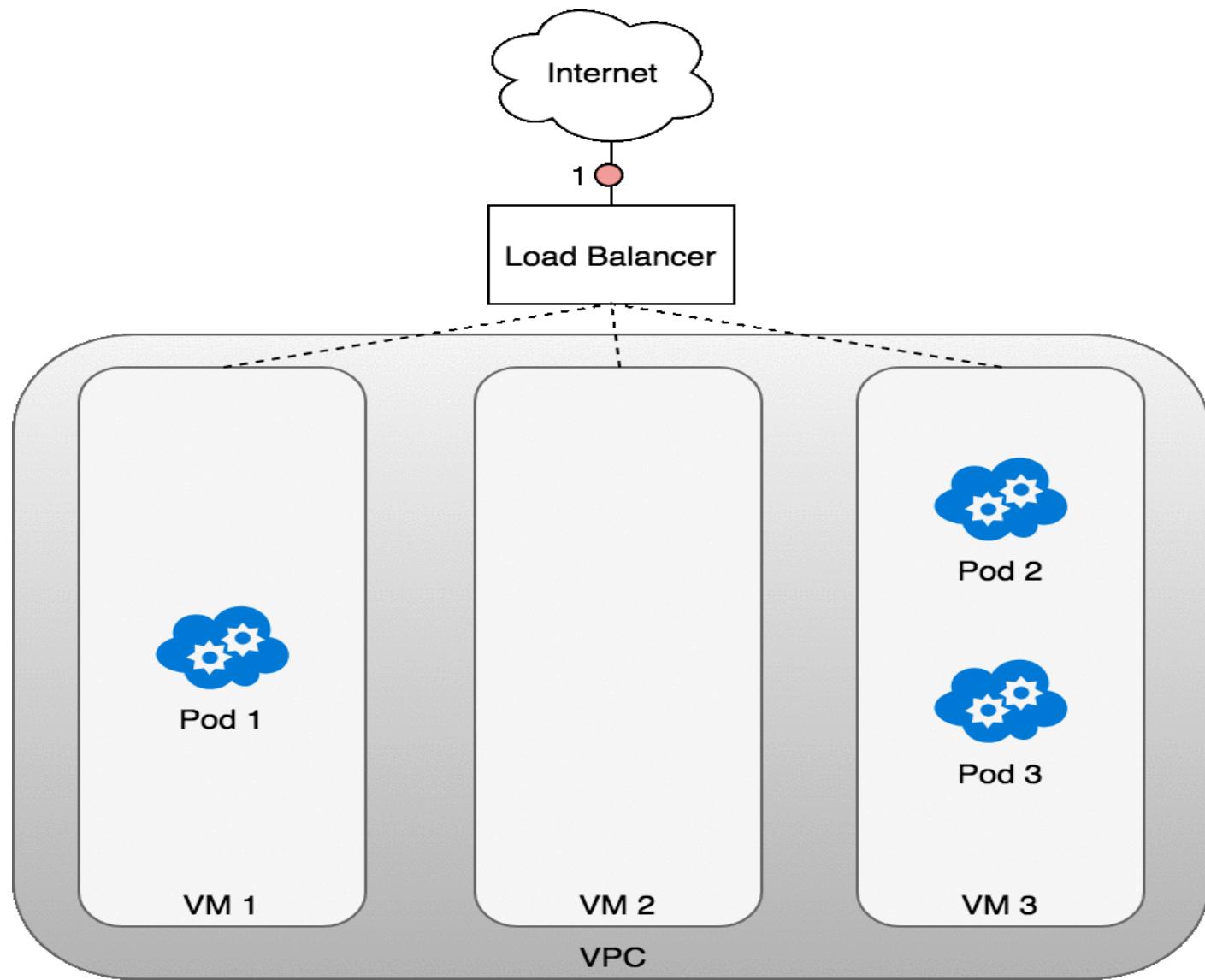
Ingress — Routing Internet traffic to Kubernetes

- Ingress — getting traffic into your cluster — is a problem to solve.
- Again, this is specific to the network you are running, but in general, Ingress is divided into two solutions that work on different parts of the network stack:
- (1) a Service LoadBalancer and
- (2) an Ingress controller.



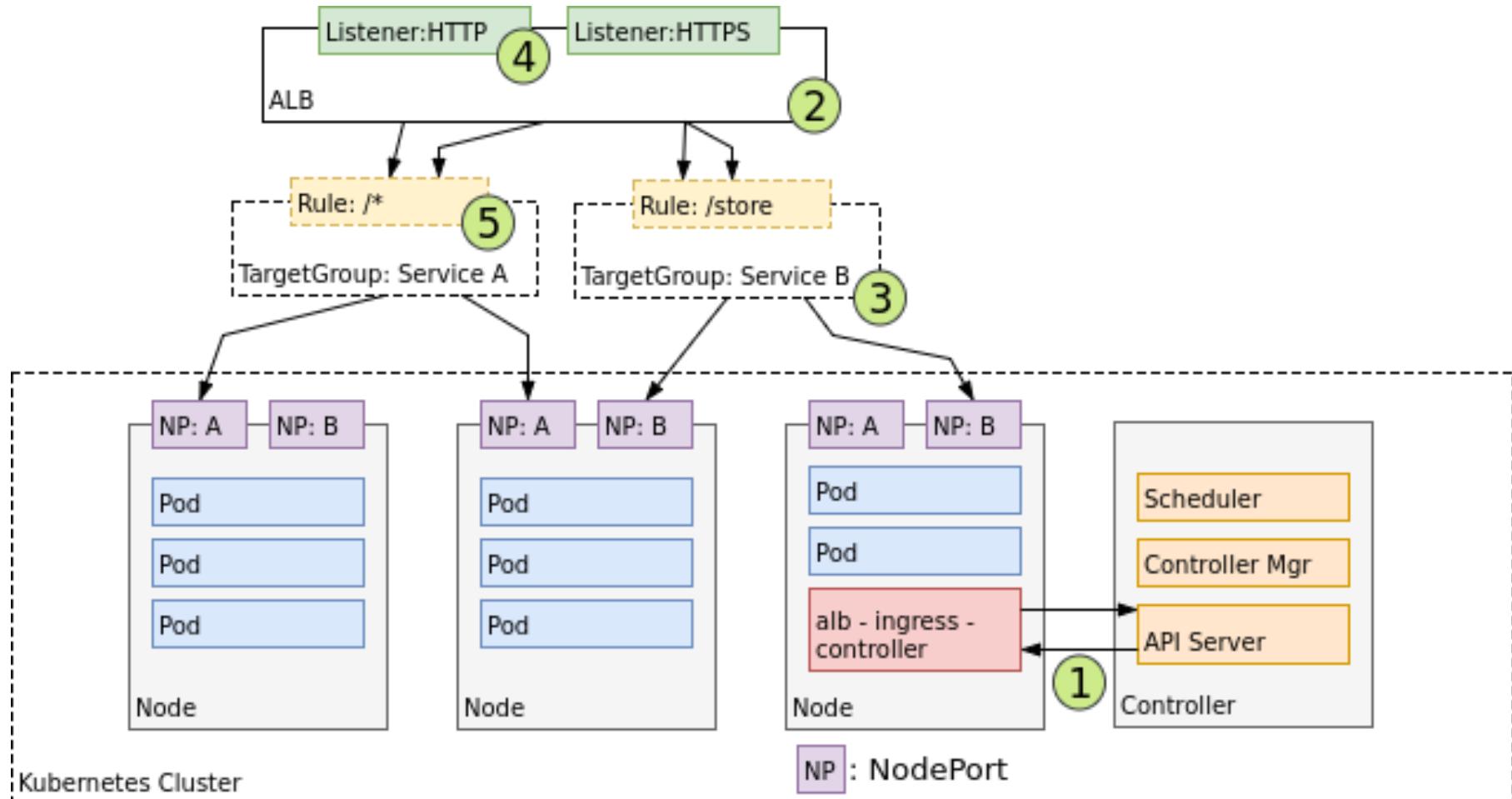
Layer 4 Ingress: LoadBalancer

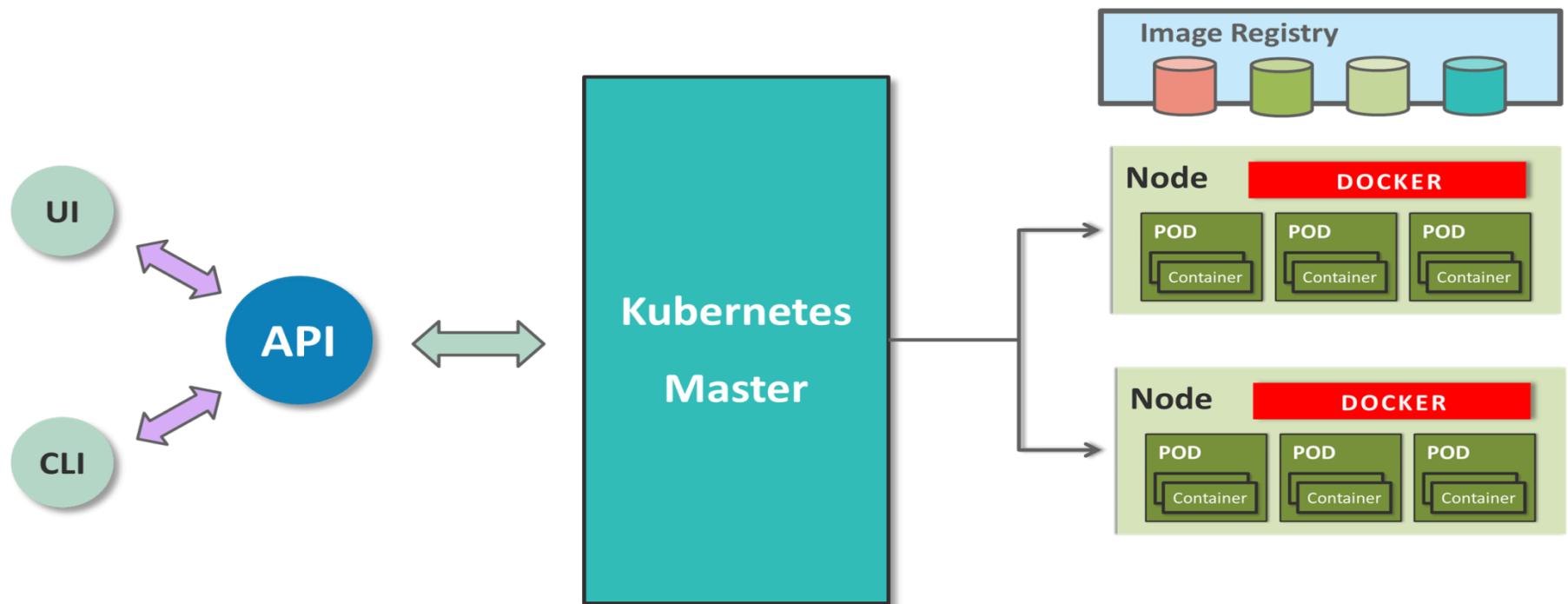
- When you create a Kubernetes Service you can optionally specify a LoadBalancer to go with it.
- The implementation of the LoadBalancer is provided by a cloud controller that knows how to create a load balancer for your service.
- Once your Service is created, it will advertise the IP address for the load balancer.
- As an end user, you can start directing traffic to the load balancer to begin communicating with your Service





Ingress Controller





Pod has group of containers that are run on the same host. So, if we regularly deploy single containers, then our container and the pod will be one and the same.



Representation Of Kubernetes Cluster

App.yaml

Deployment

Pod1

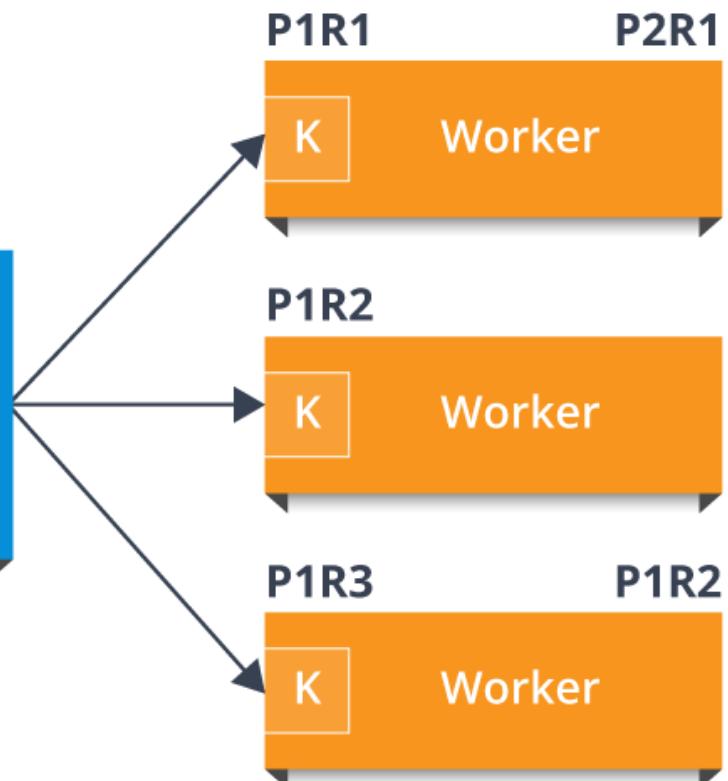
Container Image 1 Container Image 2

Replicas = 3

Pod2

Container Image 3

Replicas = 2



P - Pod R - Replica

Different types of services in Kubernetes



Cluster IP	Node Port	Load Balancer	External Name
<ul style="list-style-type: none">• Exposes the service on a cluster-internal IP.• Makes the service only reachable from within the cluster.• This is the default Service Type.	<ul style="list-style-type: none">• Exposes the service on each Node's IP at a static port.• A Cluster IP service to which Node Port service will route, is automatically created.	<ul style="list-style-type: none">• Exposes the service externally using a cloud provider's load balancer.• Services, to which the external load balancer will route, are automatically created.	<ul style="list-style-type: none">• Maps the service to the contents of the External Name field by returning a CNAME record with its value.• No proxying of any kind is set up.

Minikube –p cluster1 dashboard



Hello Minikube - Kubernetes Overview - Kubernetes Dashboard

127.0.0.1:52161/api/v1/namespaces/kube-system/services/http:kubernetes-dashboard:/proxy#!/overview?namespace=default

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

kubernetes + CREATE

☰ Overview

Cluster

Namespaces

Nodes

Persistent Volumes

Roles

Storage Classes

Namespace

default

Overview

Workloads

Cron Jobs

Daemon Sets

Deployments

Jobs

Pods

Replica Sets

Replication Controllers

Discovery and Load Balancing

Services

Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age
kubernetes	component: apiserver provider: kubernetes	10.96.0.1	kubernetes:443 TCP	-	6 minutes

Config and Storage

Secrets

Name	Type	Age
default-token-r827j	kubernetes.io/service-account-token	6 minutes

Type here to search

22:50 20/02/2019

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



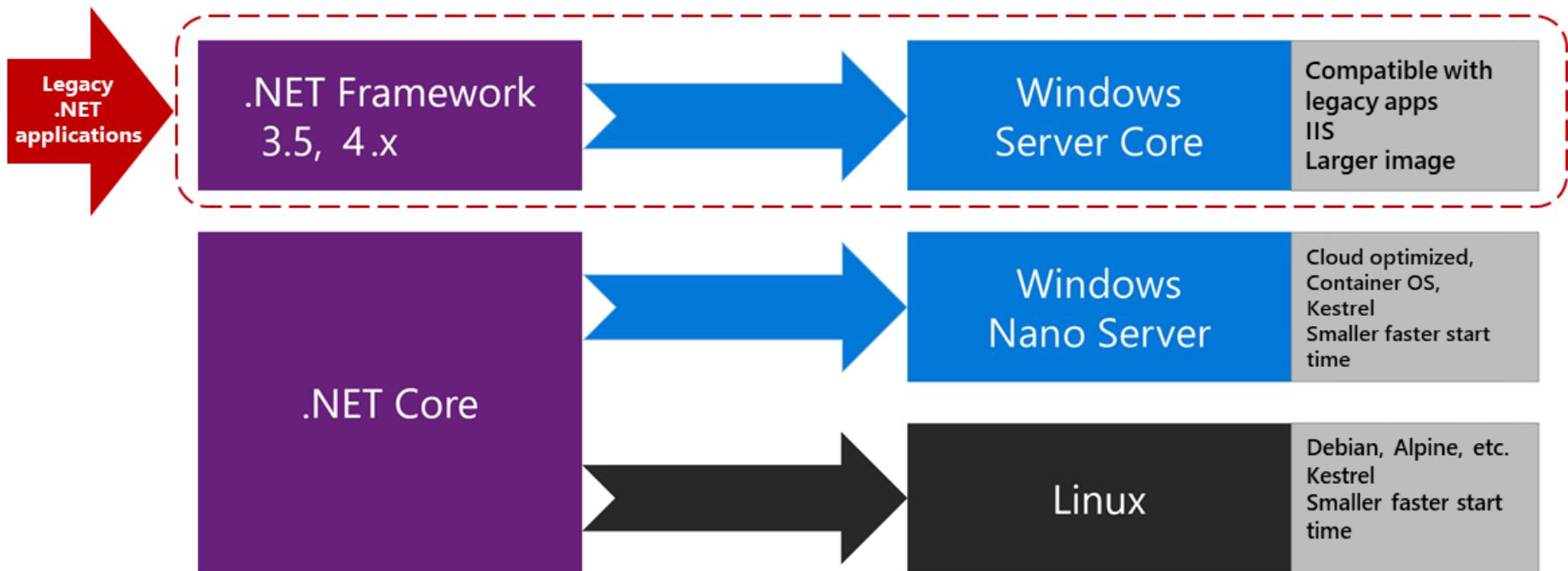
Microservices

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	<p>.NET Core 2.2 multi-architecture: Supports Linux and Windows Nano Server depending on the Docker host.</p>
microsoft/dotnet:2.2-aspnetcore-runtime	<p>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.</p>
microsoft/dotnet:2.2-aspnetcore-runtime-alpine	<p>.NET Core 2.2 runtime-only on Linux Alpine distro</p>
microsoft/dotnet:2.2-aspnetcore-runtime-nanoserver-1803	<p>.NET Core 2.2 runtime-only on Windows Nano Server (Windows Server version 1803)</p>



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

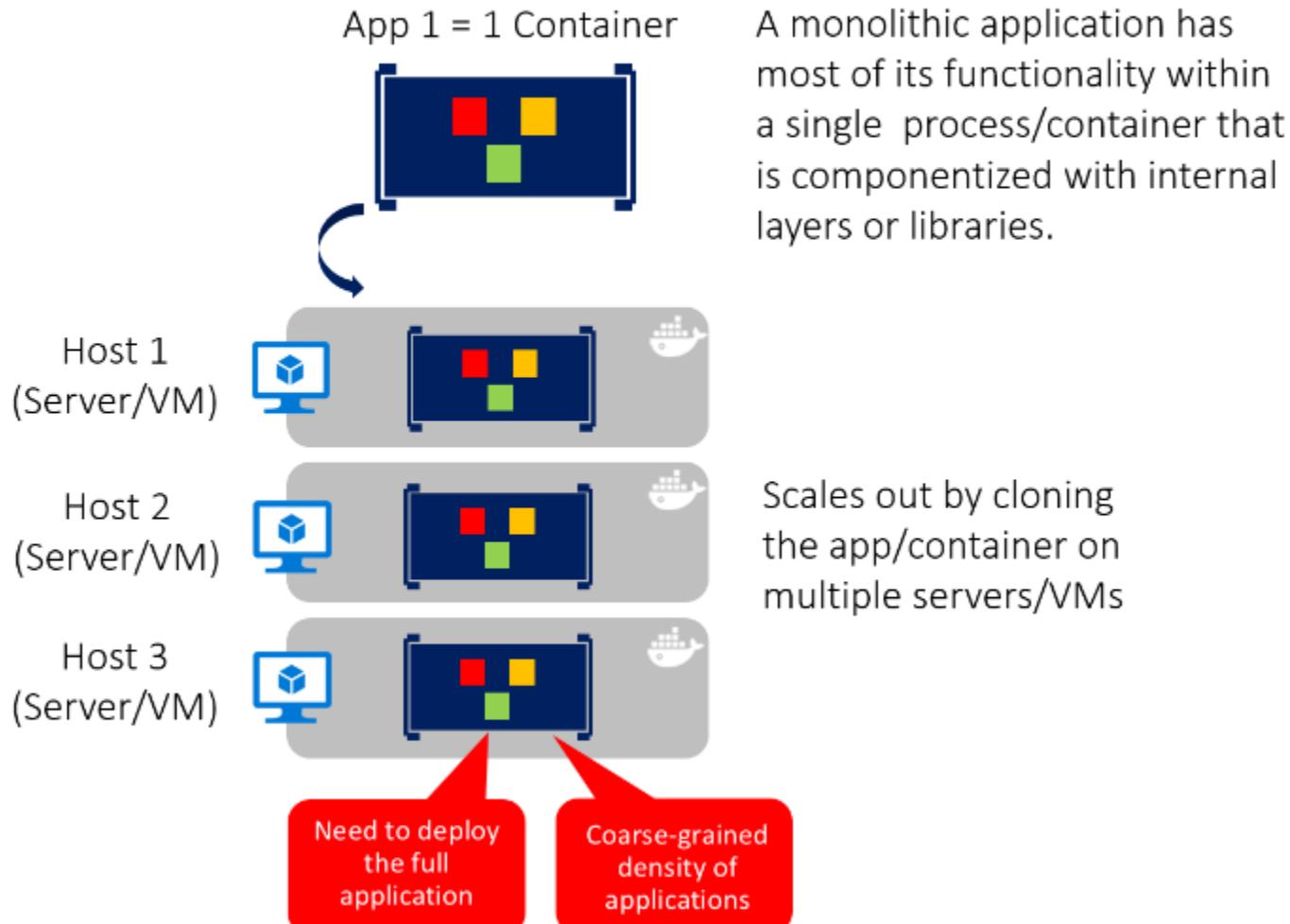


- **Container design principles**
- Scenario where we want multiple processes running in a single container.
- For that scenario, since there can be only one entry point per container, we need to run a script within the container that launches as many programs as needed.
- For example, you can use Supervisor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

Containerizing monolithic applications



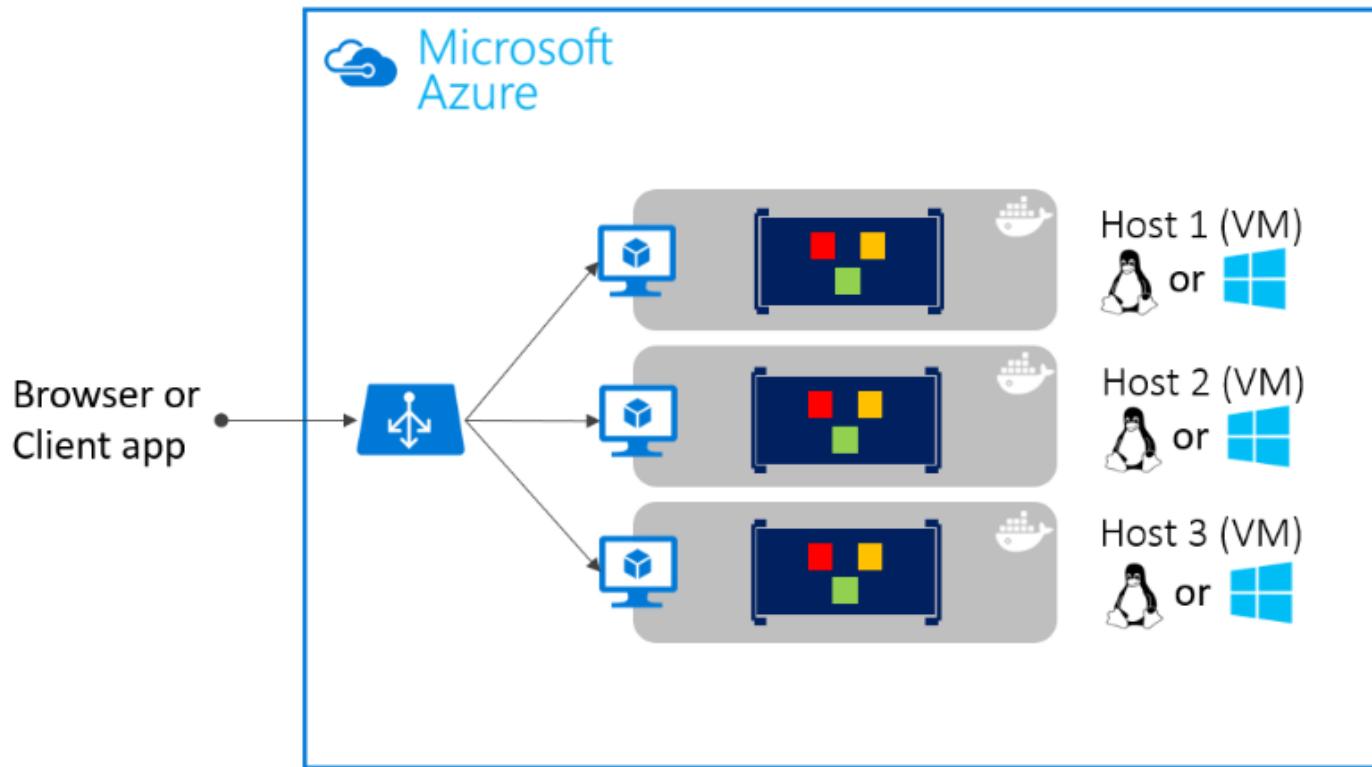
Monolithic Containerized application



Containerizing monolithic applications



Architecture in Docker infrastructure for monolithic applications





Azure Kubernetes

A screenshot of a web browser window. The address bar at the top shows "Not secure | 40.118.245.185". The page content includes a navigation bar with links like "Insert title here", "Empire", "New Tab", and several tabs open in the background. Below the navigation bar, there's a header with "webappimage", "Home", and "Privacy" links. A teal-colored callout box with rounded corners contains the text "Use this space to summarize your privacy and cookie use policy. [Learn More.](#)" on the left and "Accept" on the right. The main content area features the word "Welcome" in large, dark gray letters, followed by the subtext "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@sha256:9fc1c84d36510d53adae
37df6253e166b076f0b914c3ce876508986000efe26e
    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.yaml  
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>
```

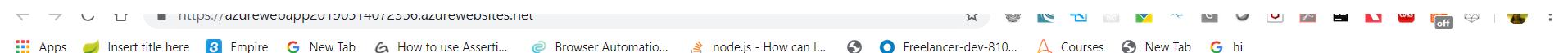


Azure and ASPnetcore

- 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

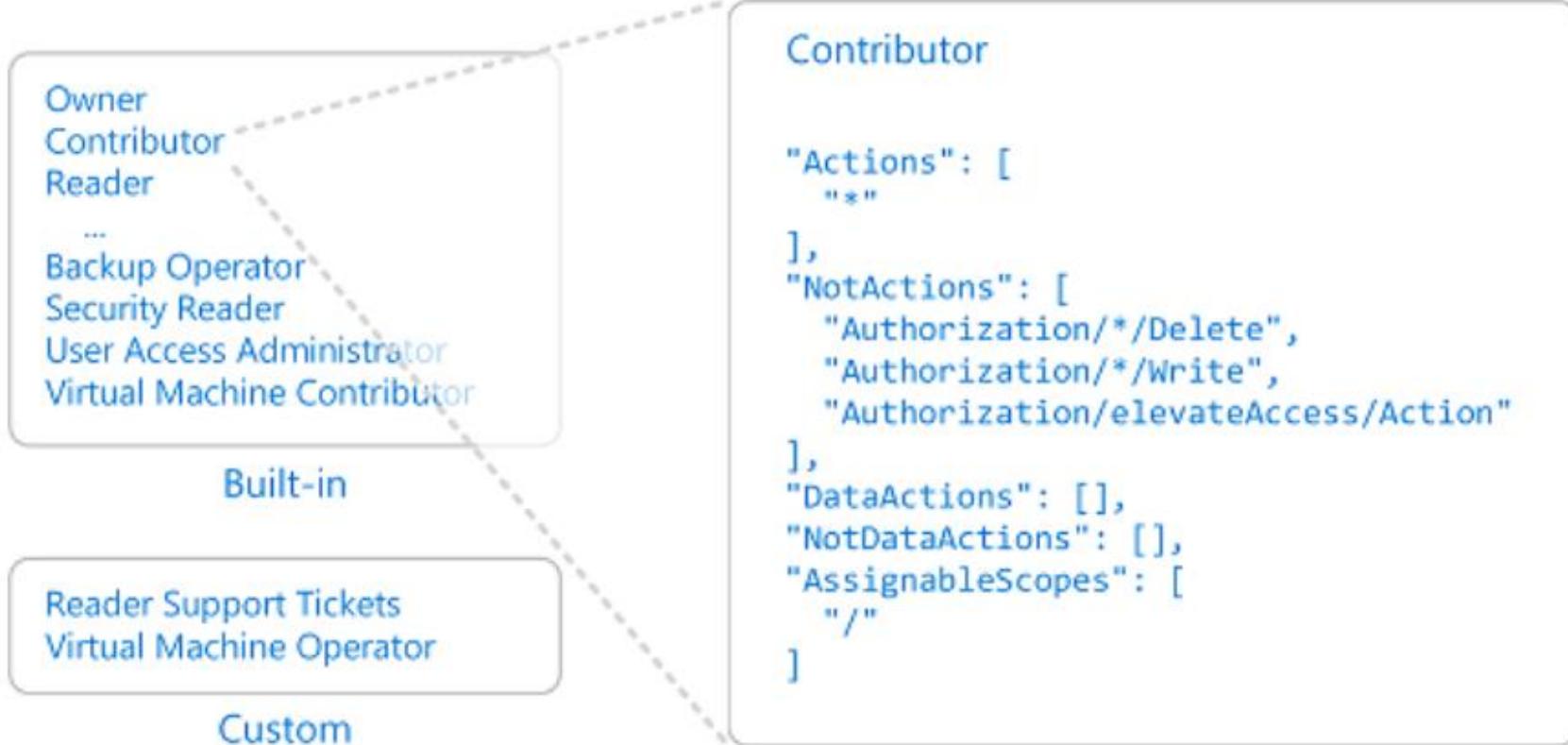


- A *role definition* is a collection of permissions.
- It's sometimes just called a *role*.
- A role definition lists the operations that can be performed, such as read, write, and delete.
- Roles can be high-level, like owner, or specific, like virtual machine reader.

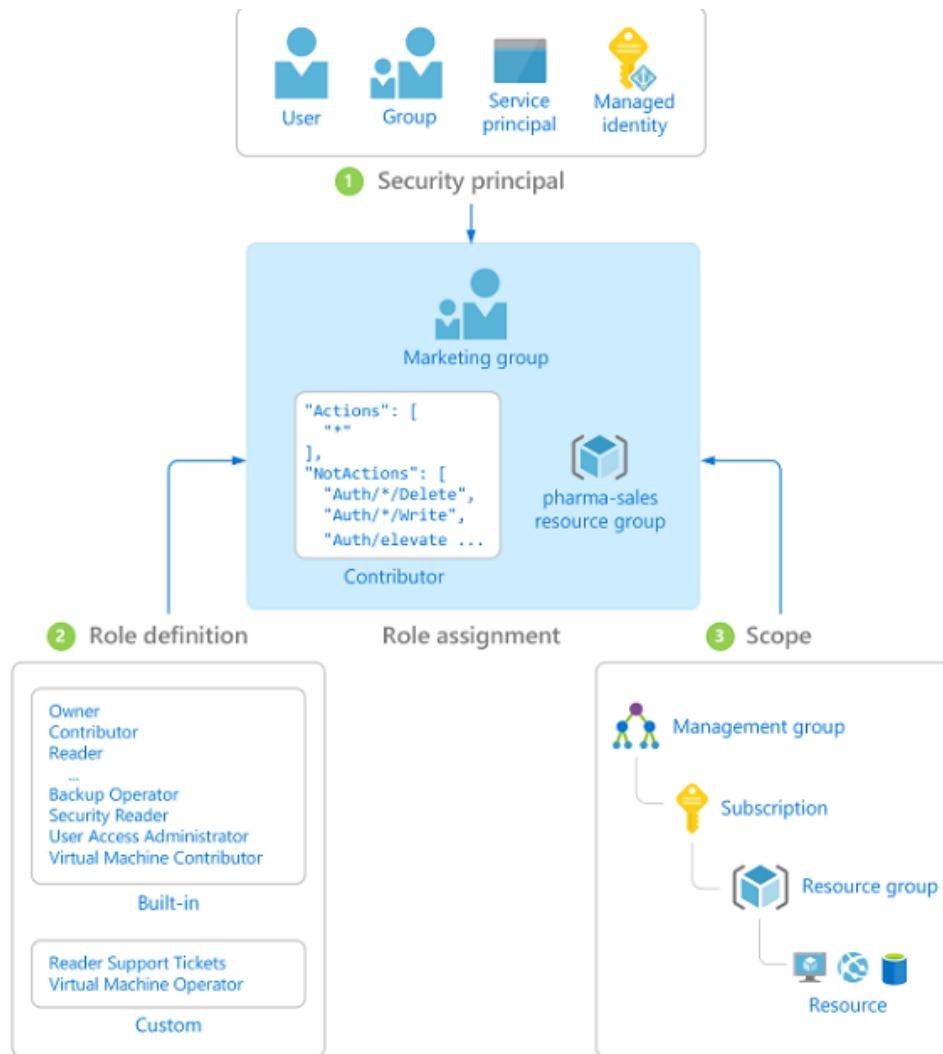
Role definition

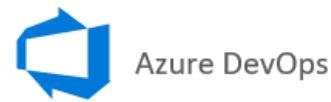


2 Role definition



Role definition

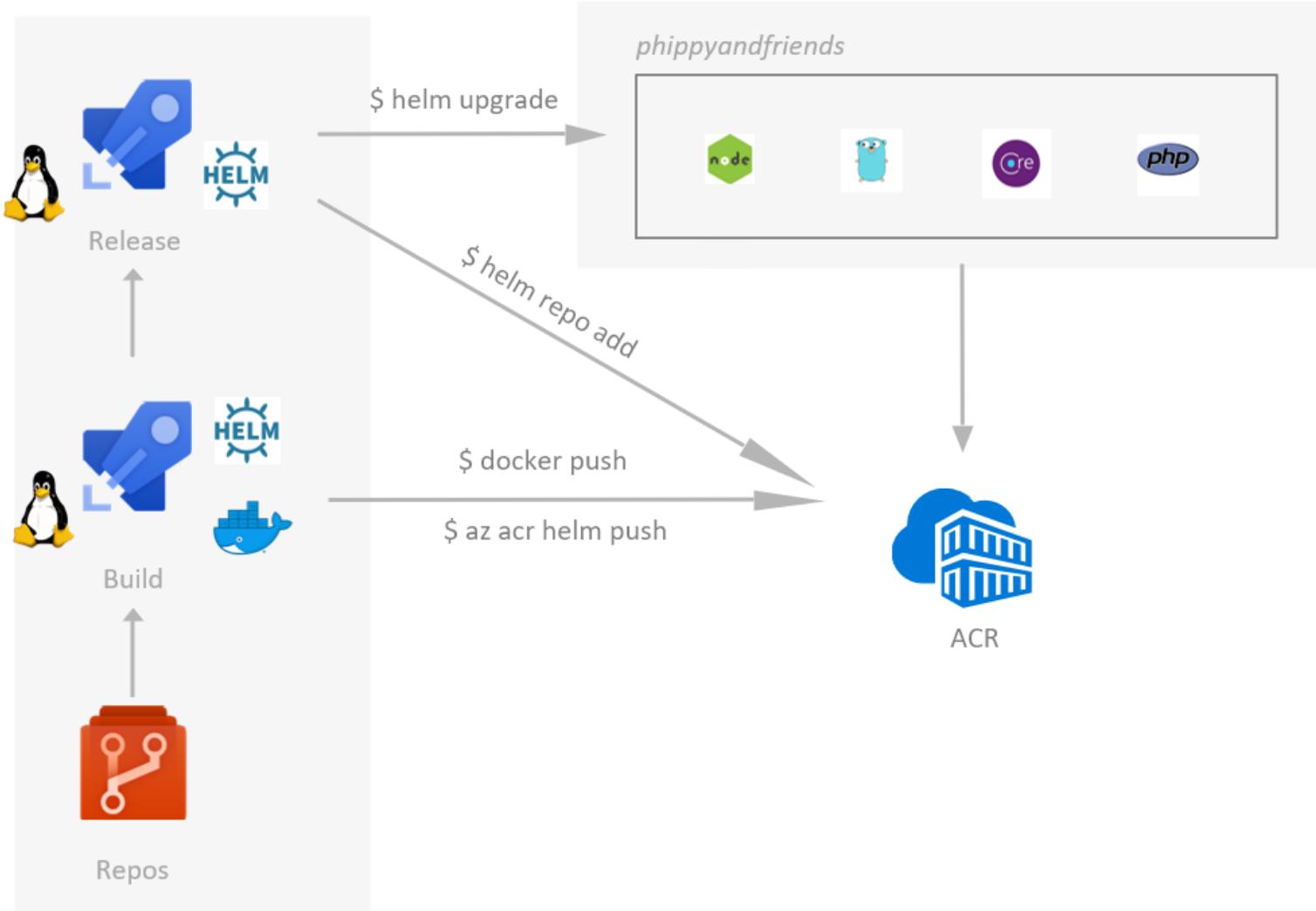


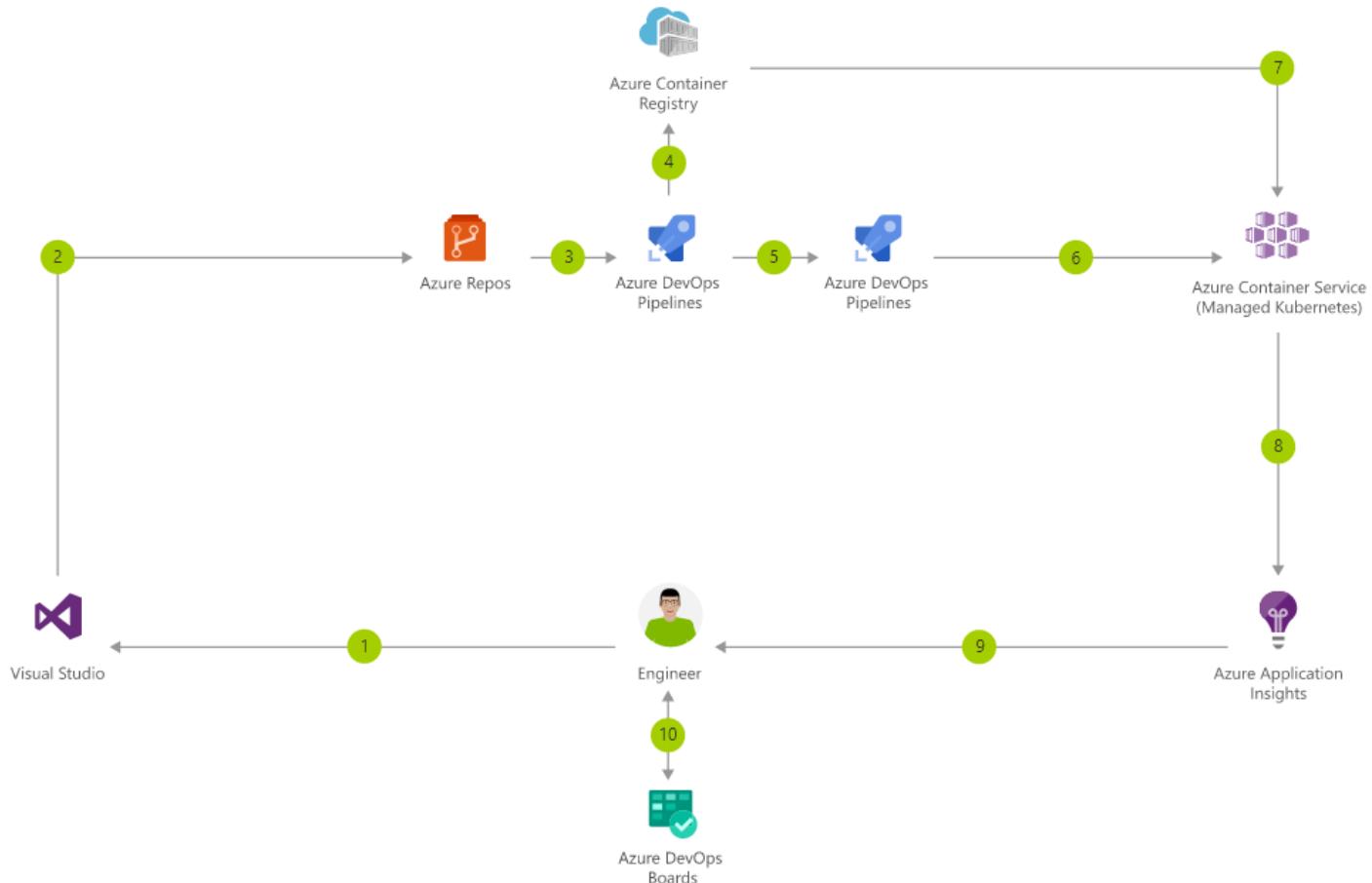


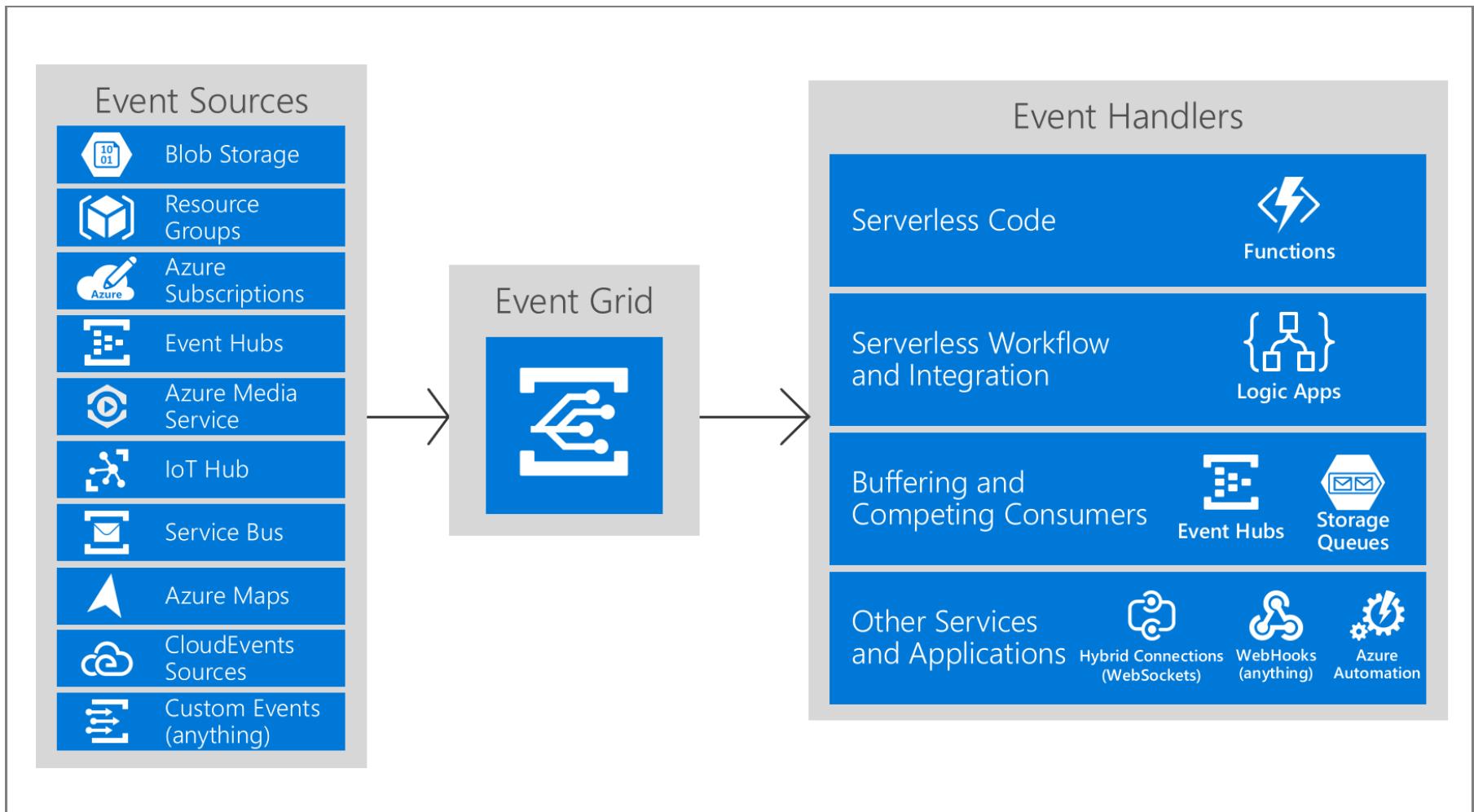
Azure DevOps



AKS cluster









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
```

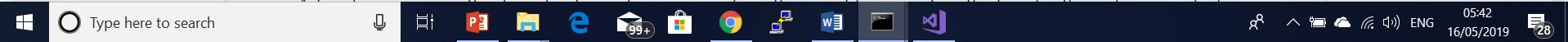
SUCCESS: Specified value was saved.

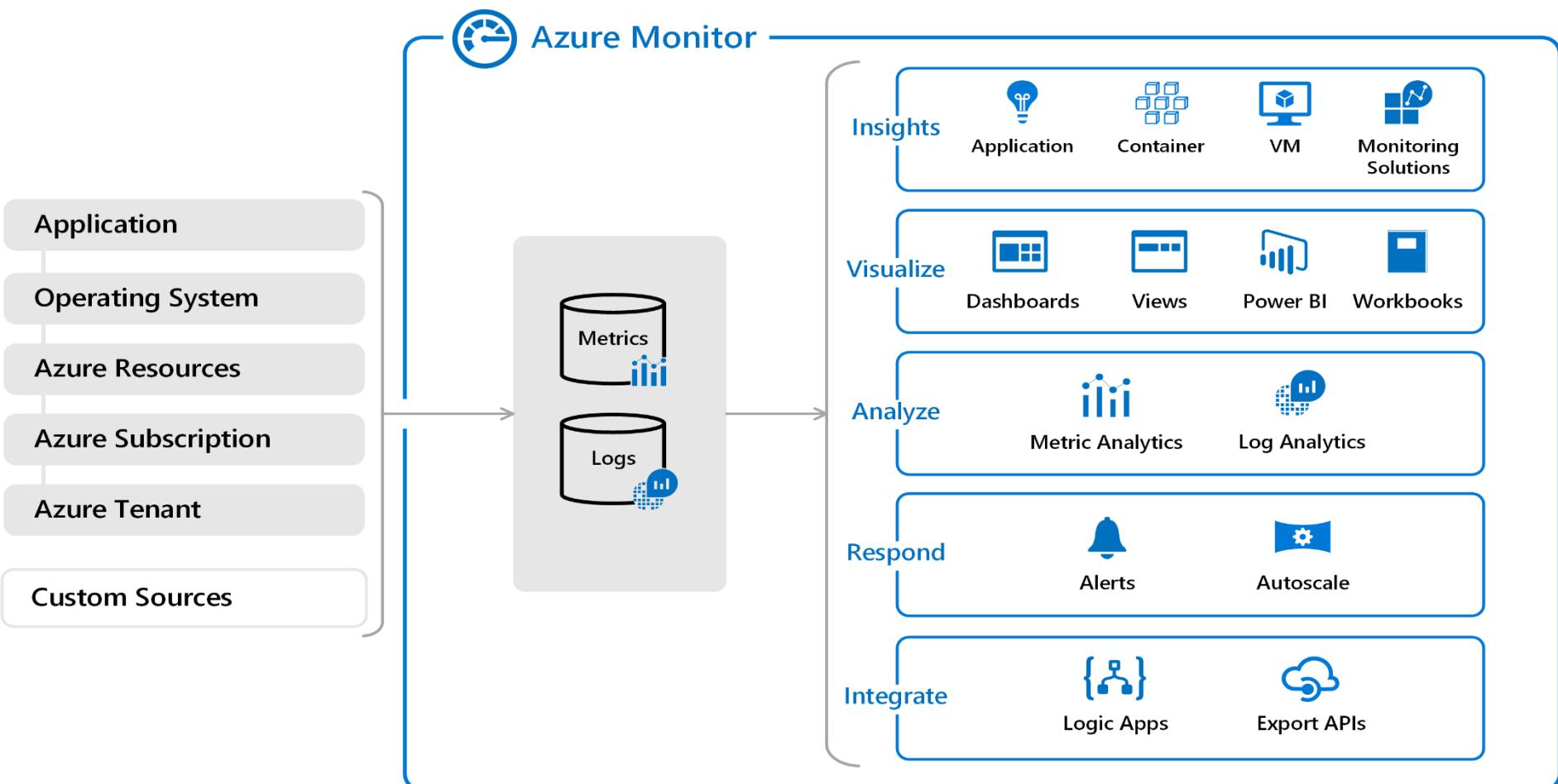
```
C:\WINDOWS\system32>
```



```
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`
- -----

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?
Retry (policy family) (quickstart) ; (deep)	Many faults are transient and may self-correct after a short delay.	"Maybe it's just a blip"	Allows configuring automatic retries.
Circuit-breaker (policy family) (quickstart) ; (deep)	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.
Timeout (quickstart) ; (deep)	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



Bulkhead Isolation (quickstart ; deep)	<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.
Cache (quickstart ; deep)	<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>
Fallback (quickstart ; deep)	<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);



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:**
 - subnet-0643f874c1b0a572e (AmazonEKSVPCCSubnet02)
 - subnet-09a8ca2ac0e5e6d45 (AmazonEKSVPCCSubnet03)
 - subnet-048f24dea6adb8526 (AmazonEKSVPCCSubnet01)
- Security groups:**
 - sg-0033319d94c70f6b8 (default) - default VPC security group
 - sg-09e6ccaae3498cf9 (AmazonEKSVPCCControlPlaneSecurityGroup-B1U8RVRCV8DA) - Cluster communication with worker nodes

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 Services Resource Groups Archana 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: 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

A screenshot of a web browser window. The title bar says "Welcome to nginx!". The address bar shows a URL starting with "ae1888e52e8bd11e8a2000a28798916d-1273662397.us-east-1.elb.amazonaws.com", which is highlighted with a green box. The page content is a standard "Welcome to nginx!" page with text about the server's status and links to documentation and support.

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.
Commercial support is available at nginx.com.

Thank you for using nginx.

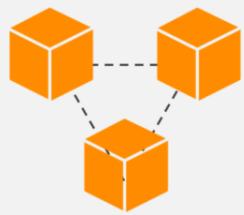


EKS Steps

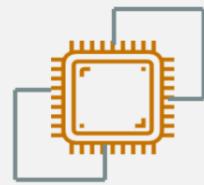
- **Step 6: Cleaning up the application & assigned resources**
- When you are finished experimenting with your application, you should clean up the resources that you created for it.

```
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



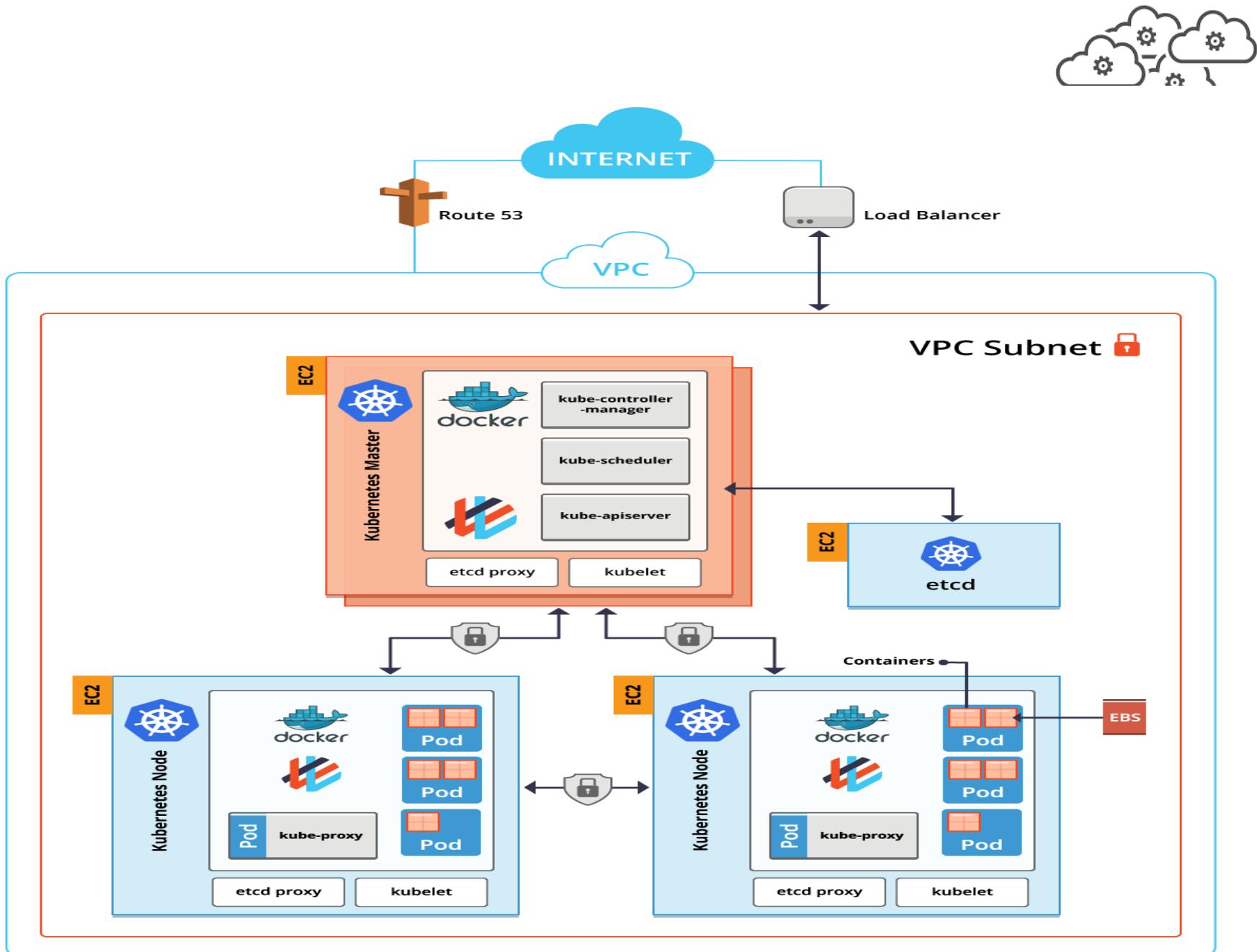
Deploy worker nodes



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
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

23:08 27/02/2019



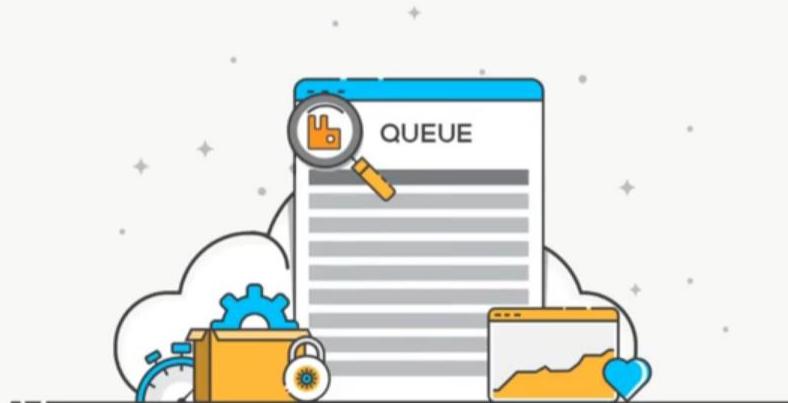
rabbitmq-plugins enable rabbitmq_management

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 Dev Academy



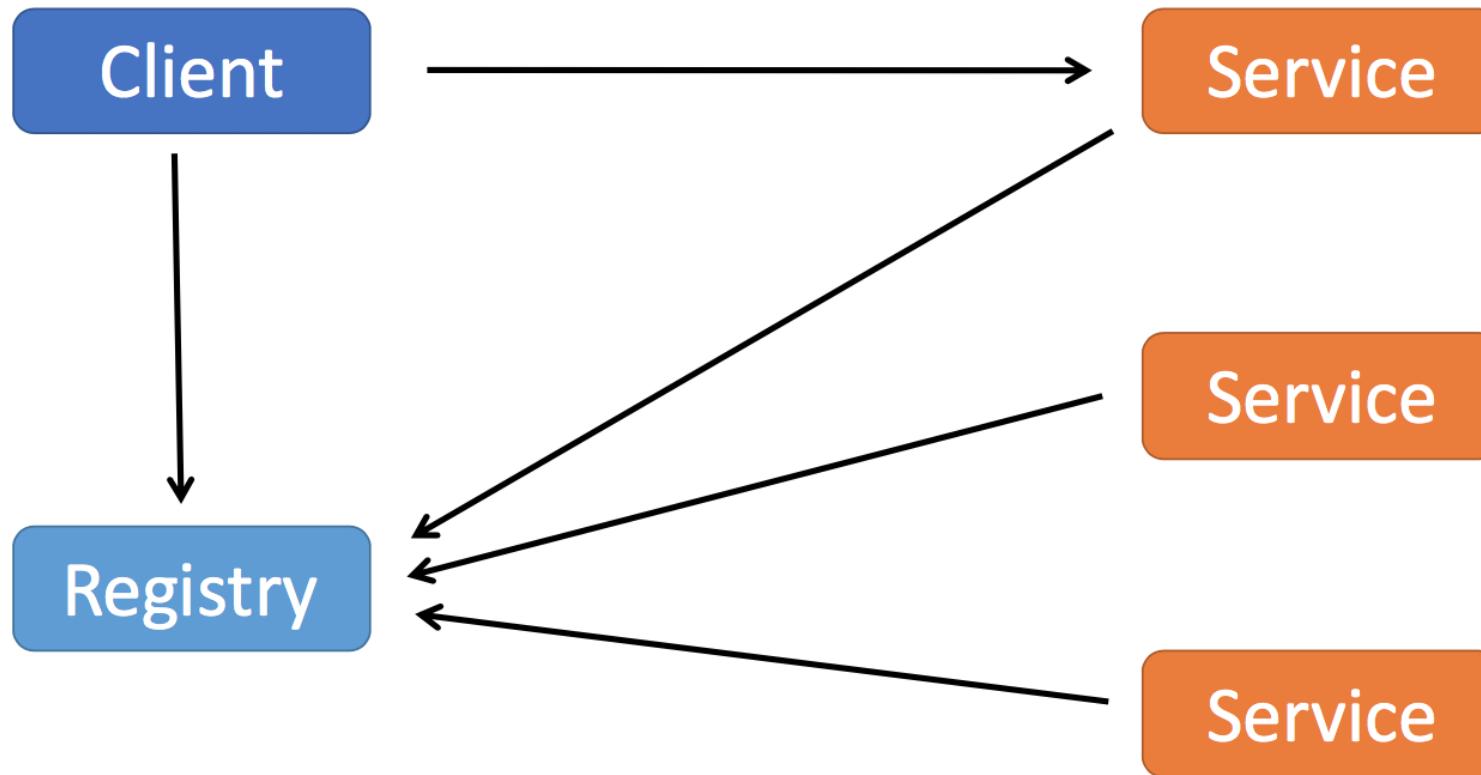
Solutions to Challenges with Microservice Architectures



- **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).**



Service Discovery



```
docker run -p 8500:8500 -p 8600:8600/udp --name=consul consul:latest agent -server -bootstrap -ui -client=0.0.0.0
```



Service Discovery

localhost:8500/ui/dc1/services

Apps Projects Gmail YouTube Maps Pluralsight

dc1 Services Nodes Key/Value ACL Intentions Help Settings

Services 1 total

Search

✓ consul
1 Instance

```
docker run -p 8500:8500 -p 8600:8600/udp --name=consul consul:latest agent -server -bootstrap -ui -client=0.0.0.0
```



Service Discovery

localhost:8500/ui/dc1/services/CatalogApi/instances/b9ad58f595f2/CatalogApi-v1-9001/health-checks

Apps Projects Gmail YouTube Maps Pluralsight

CatalogApi-v1-9001

http://localhost

Service Name	Node Name
CatalogApi	b9ad58f595f2

[Health Checks](#) [Tags & Meta](#)

× Service 'CatalogApi' check

ServiceName	CheckID	Type	Notes
CatalogApi	service:CatalogApi-v1-9001	http	-

Output

```
Agent alive and reachable
```

✓ Serf Health Status

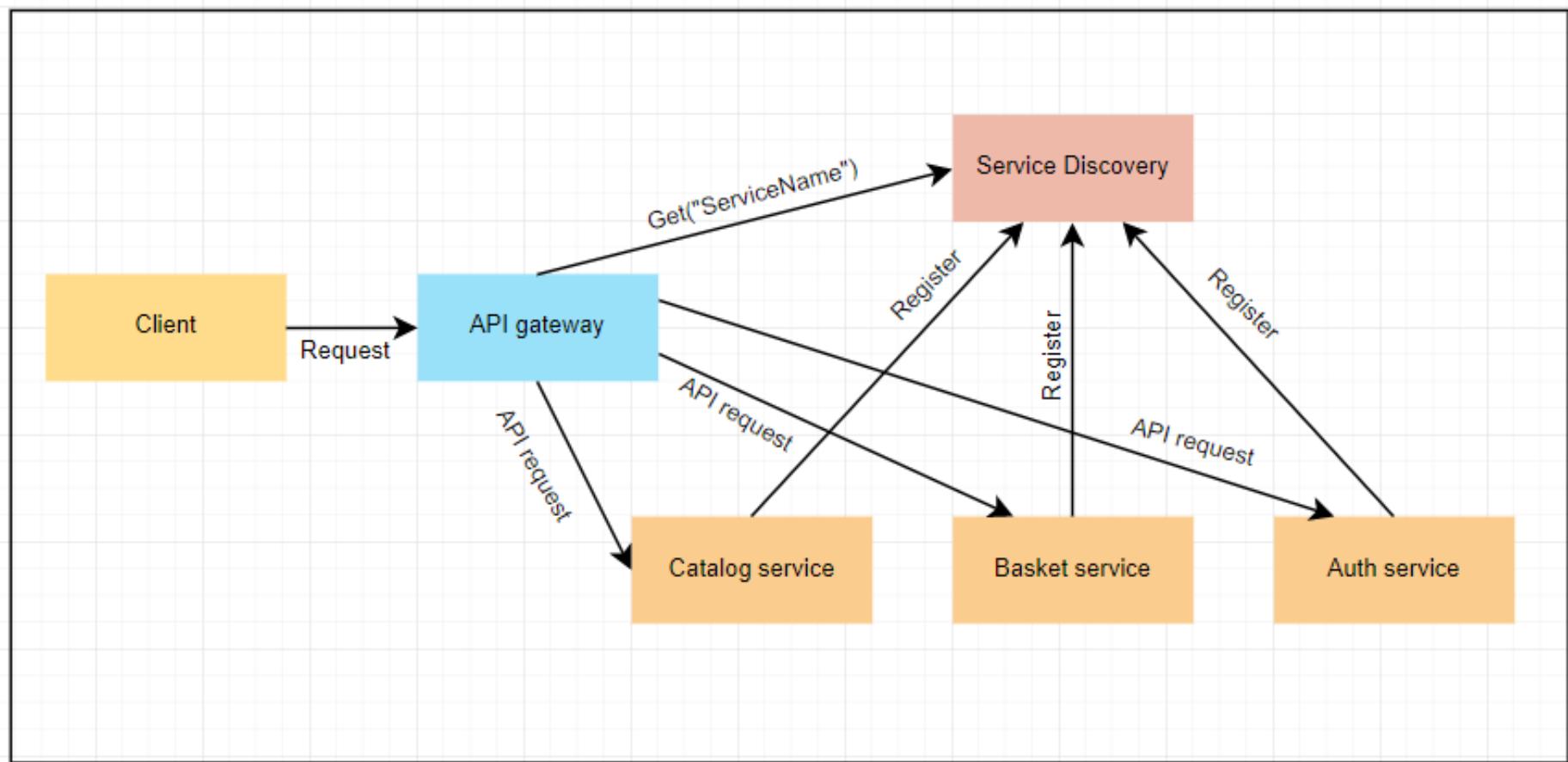
NodeName	CheckID	Type	Notes
b9ad58f595f2	serfHealth		-

Output

```
Agent alive and reachable
```



Service Discovery





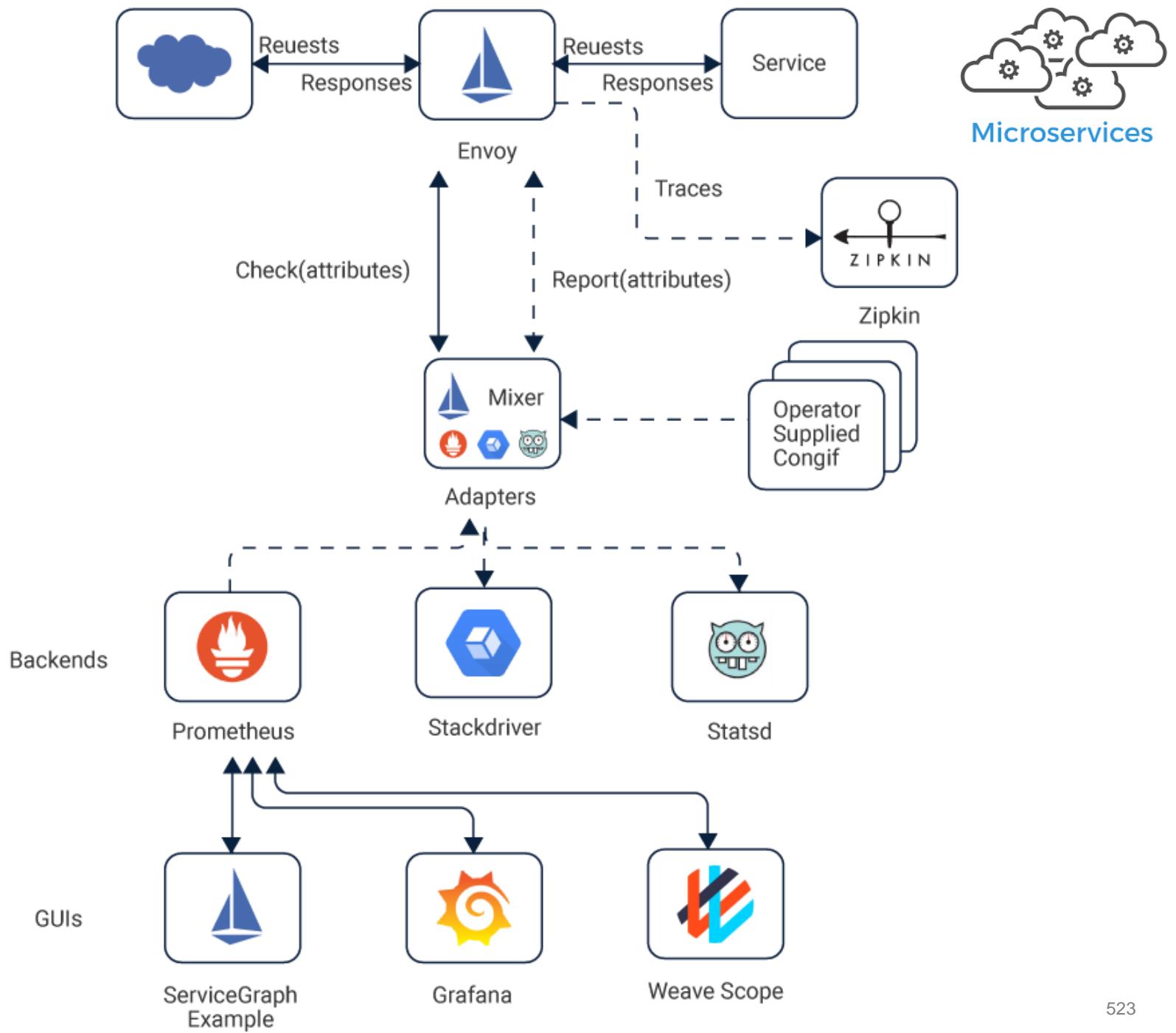
Service Discovery



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.

Envoy



- 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

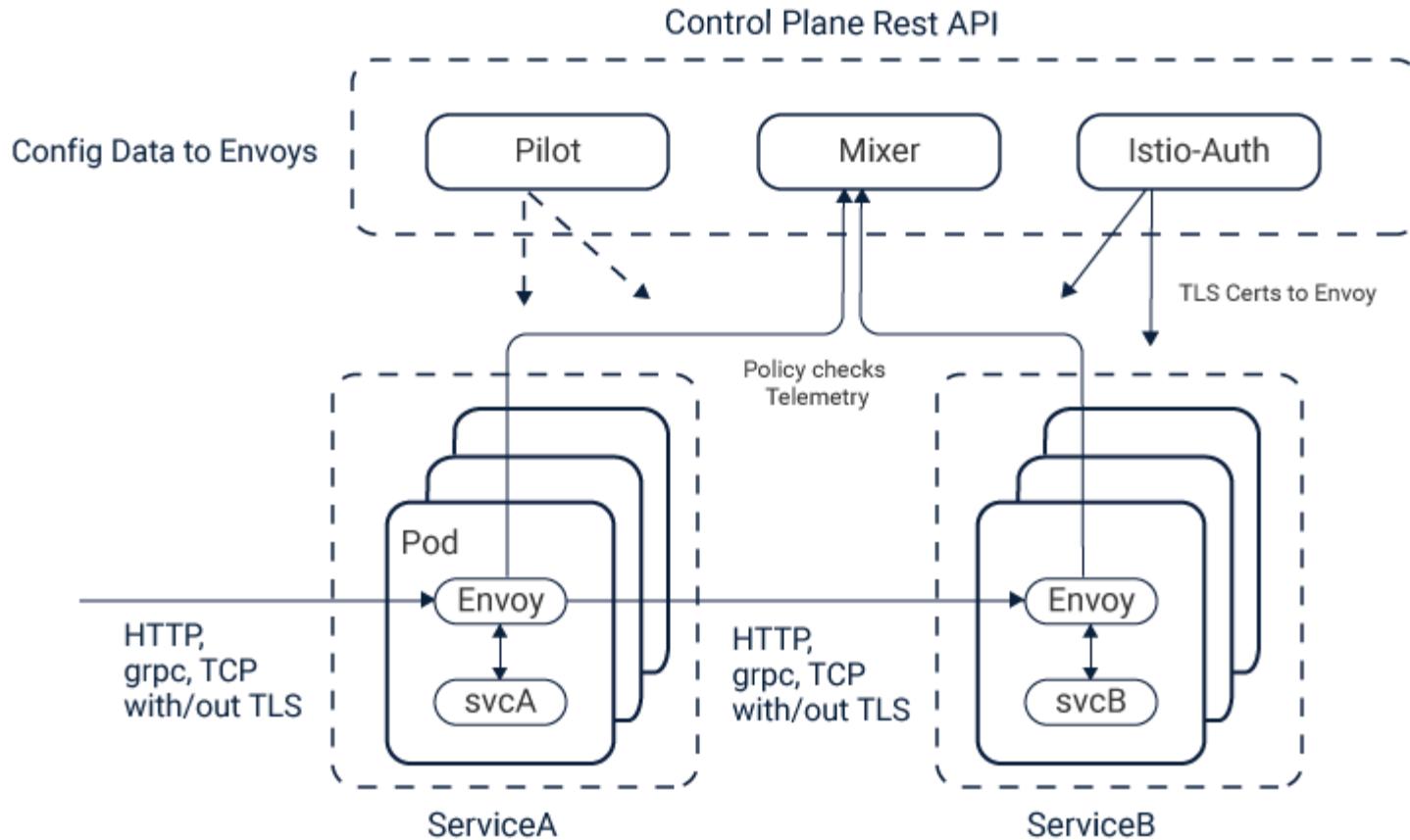
- Pilot provides service discovery for the Envoy sidecars, traffic management capabilities for intelligent routing (e.g., A/B tests, canary deployments, etc.), and resiliency (timeouts, retries, circuit breakers, etc.).
- Pilot converts high level routing rules that control traffic behavior into Envoy-specific 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.

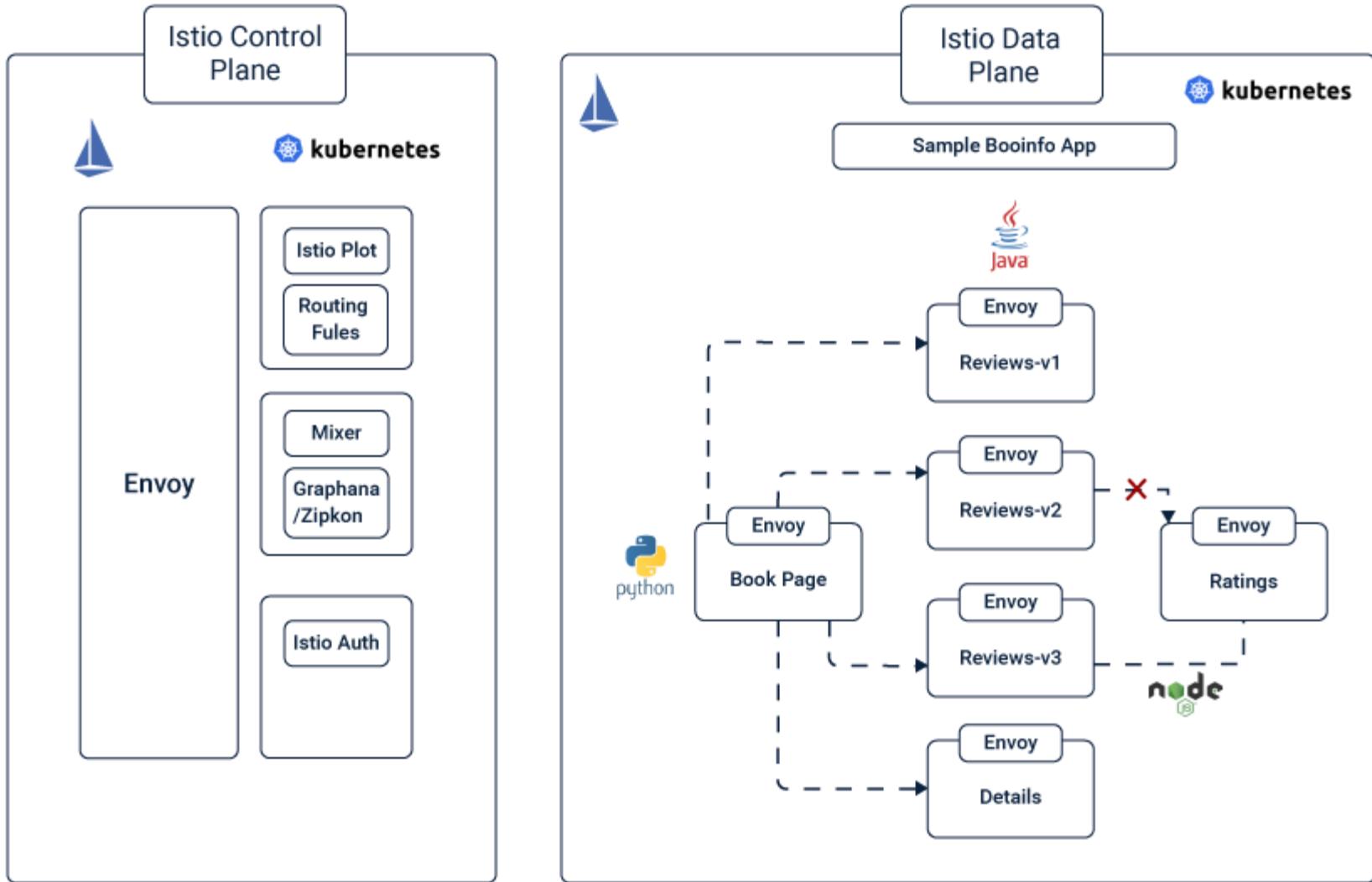


- Galley validates user authored Istio API configuration on behalf of the other Istio control plane components.
- Over time, Galley will take over responsibility as the top-level configuration ingestion, processing and distribution component of Istio.
- It will be responsible for insulating the rest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

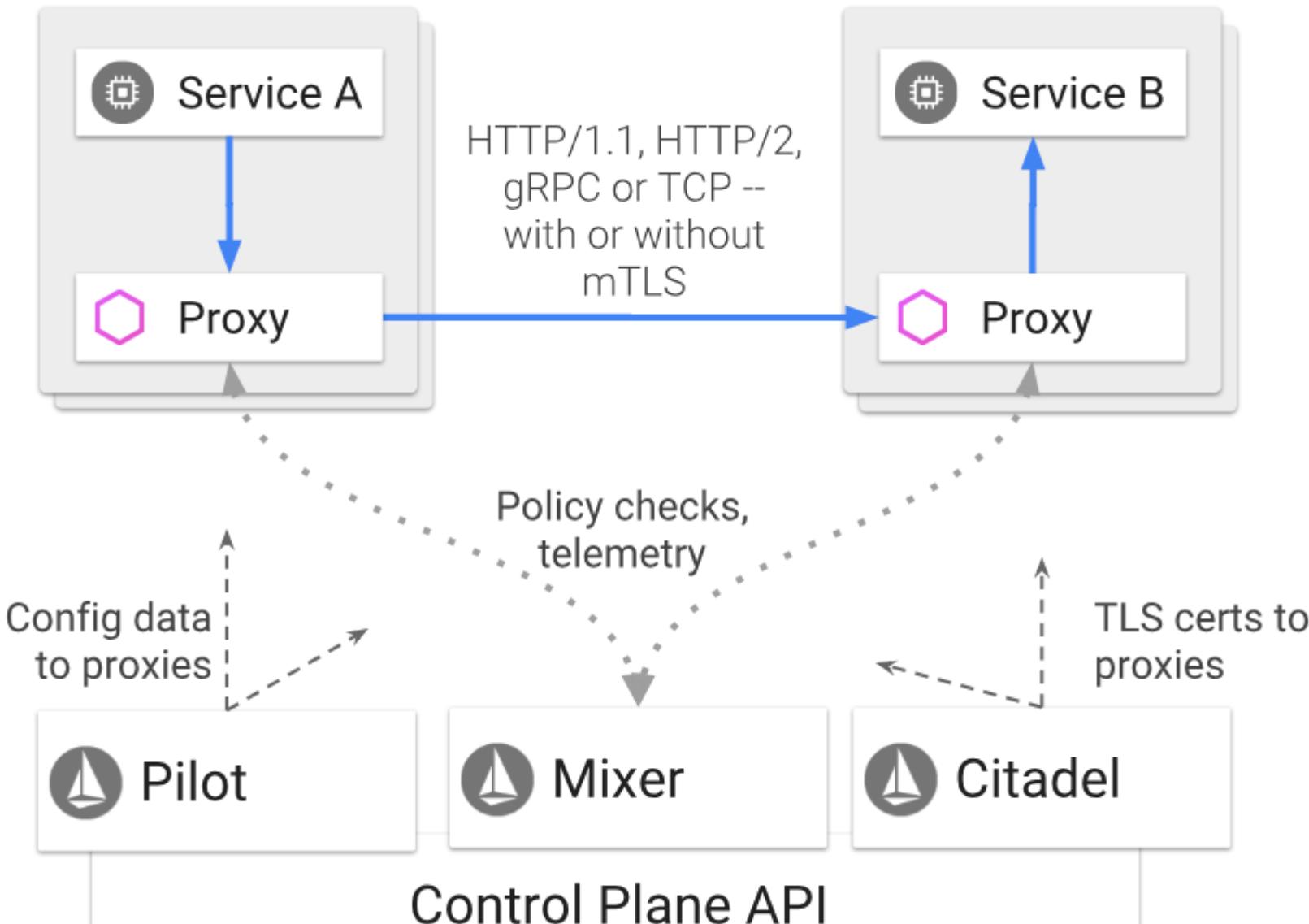
What is Istio Service Mesh?



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>Official Link: https://docs.microsoft.com/en-us/azure/aks/intro-kubernetes</p>	<p>General</p> <ul style="list-style-type: none">• Kubernetes has wide adoption including managed offerings on all major cloud platforms (AWS, GCP, Azure) helping avoid vendor lock in.• Wide community and industry support.• Scalability and modularity• Kubernetes has a very mature and proven underlying architecture. Its design is built on over 10 years of operational experience of the Google engineers.• More control over the orchestration platform for custom or specific needs.• Monitoring available from Microsoft Azure monitoring at a Kubernetes cluster and container level.• The power of a full container orchestration solution.• Flexibility with networking.• Rich ecosystem of addons. <p>Container Specific</p> <ul style="list-style-type: none">• Supports Docker hub, Azure Container registry, private Container registry.• Supports use of Docker compose or Helm charts.	<p>General</p> <ul style="list-style-type: none">• Can't run code directly on Kubernetes; only supports containers.• 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.• Steep learning curve. Need an understanding of how an orchestration platform and Docker containers work.• If something breaks can require complex troubleshooting. <p>Container Specific</p> <ul style="list-style-type: none">• Only supports Linux containers in AKS currently.	<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>Official Link: https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-overview</p> <p>NOTE: 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>General</p> <ul style="list-style-type: none">Service Fabric can run in Azure but also can run on-premises or other clouds helping to avoid vendor lock-in.Service Fabric is now open source.Ability to run code or a container.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.Automatic load balancing.Automatic high availability.Automated load distribution and scheduling.Proven stability and enterprise use as it is used to power majority of Microsoft Azure cloud services. <p>Container Specific</p> <ul style="list-style-type: none">Service Fabric supports both Linux and Windows containers.	<p>General</p> <ul style="list-style-type: none">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.Semi-steep learning curve. Need an understanding of how an orchestration/micro services platform and Docker containers work.Service Fabric is now open source, but the community support and expertise are not as wide spread as a platform such as Kubernetes.If something breaks can require complex troubleshooting. <p>Container Specific</p> <ul style="list-style-type: none">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.	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.

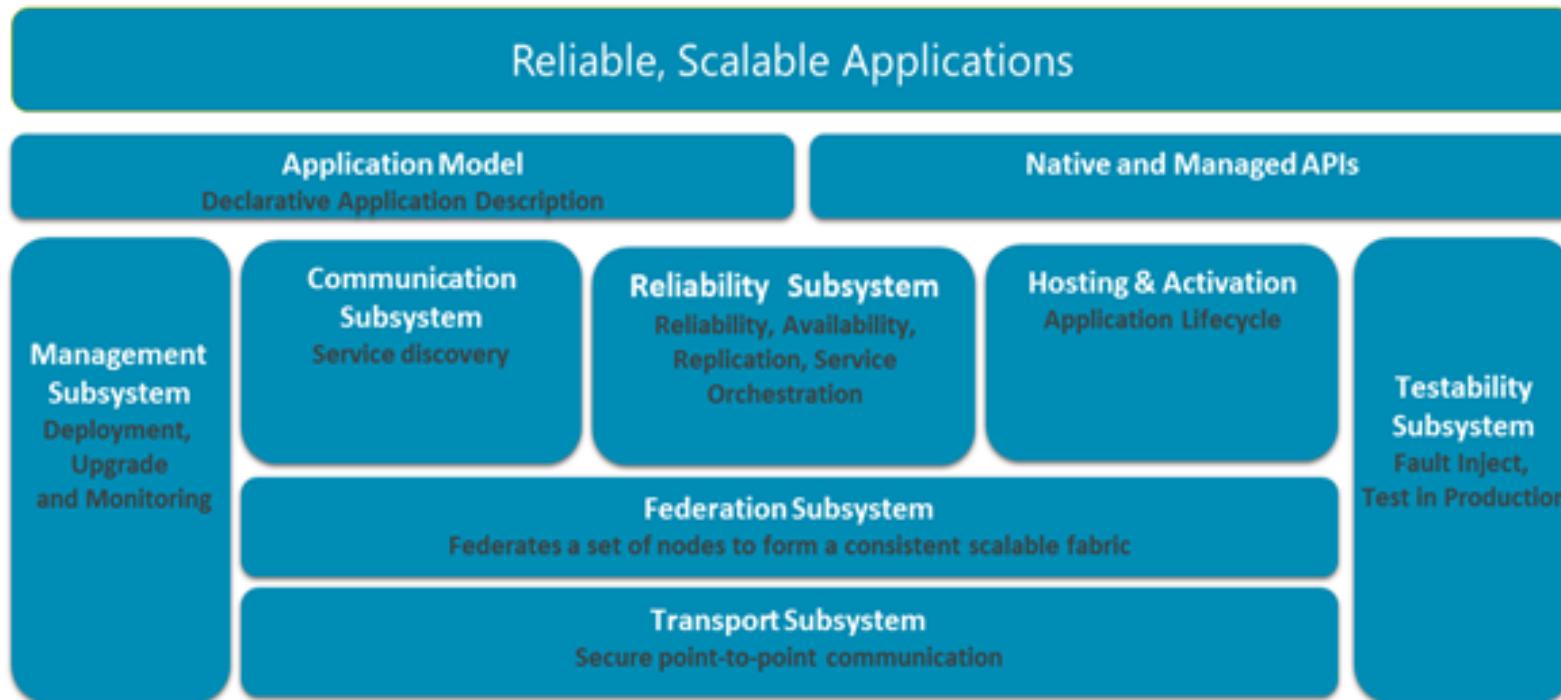


Service Fabric

- 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





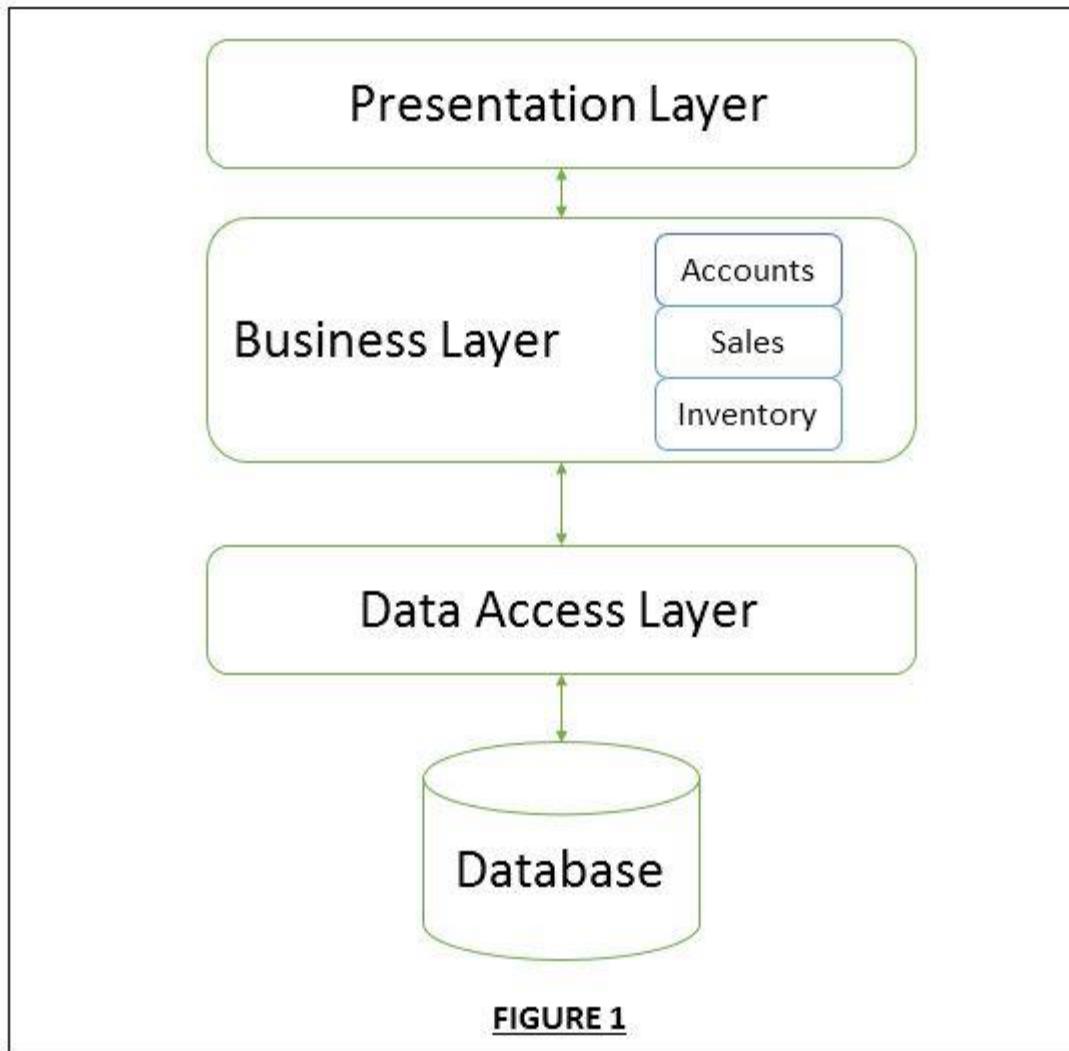
Service Fabric

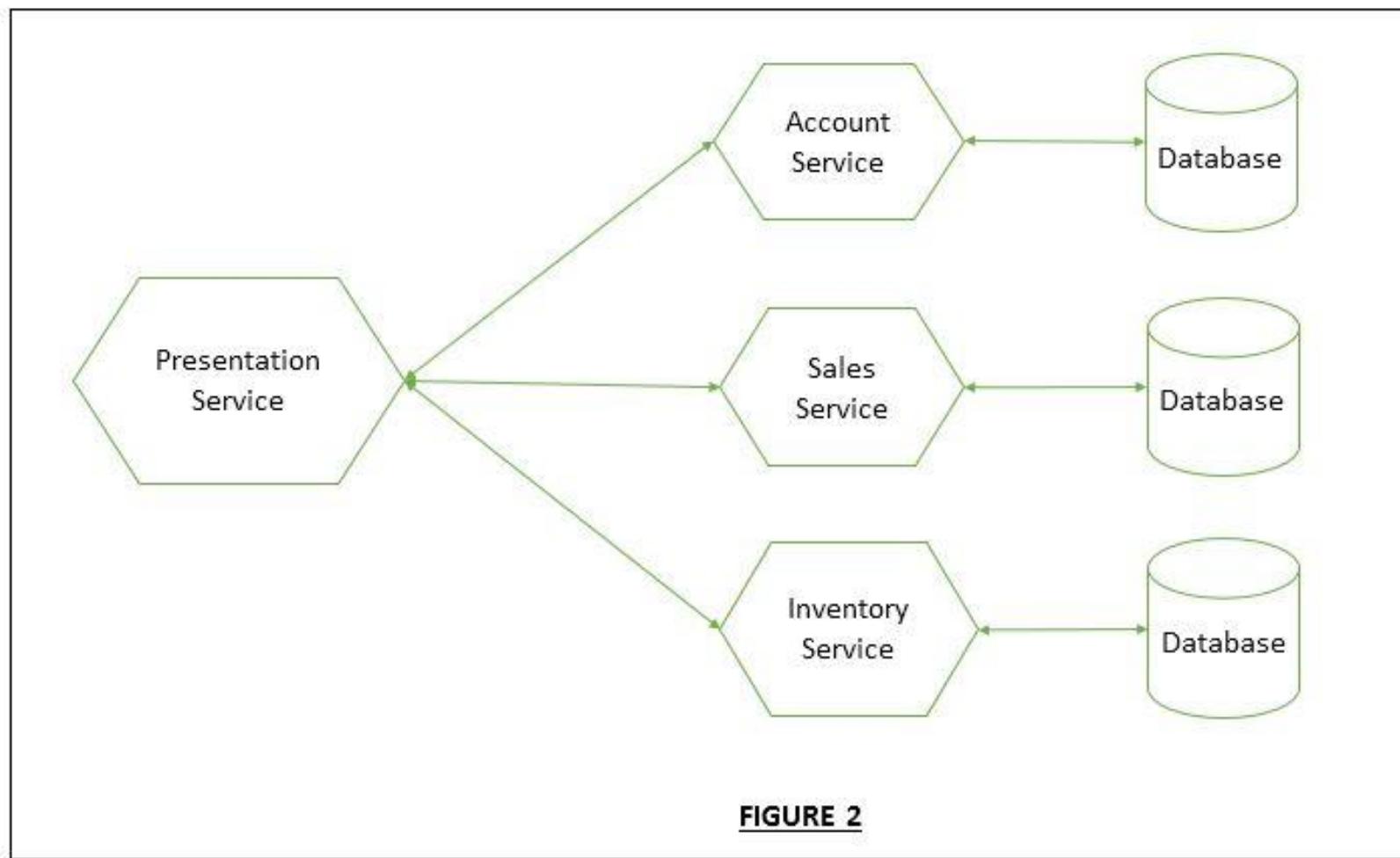
- Deploy different versions of the same application side by side, and upgrade each application independently.
- Manage the lifecycle of your applications without any downtime, including breaking and nonbreaking upgrades.
- Scale out or scale in the number of nodes in a cluster. As you scale nodes, your 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

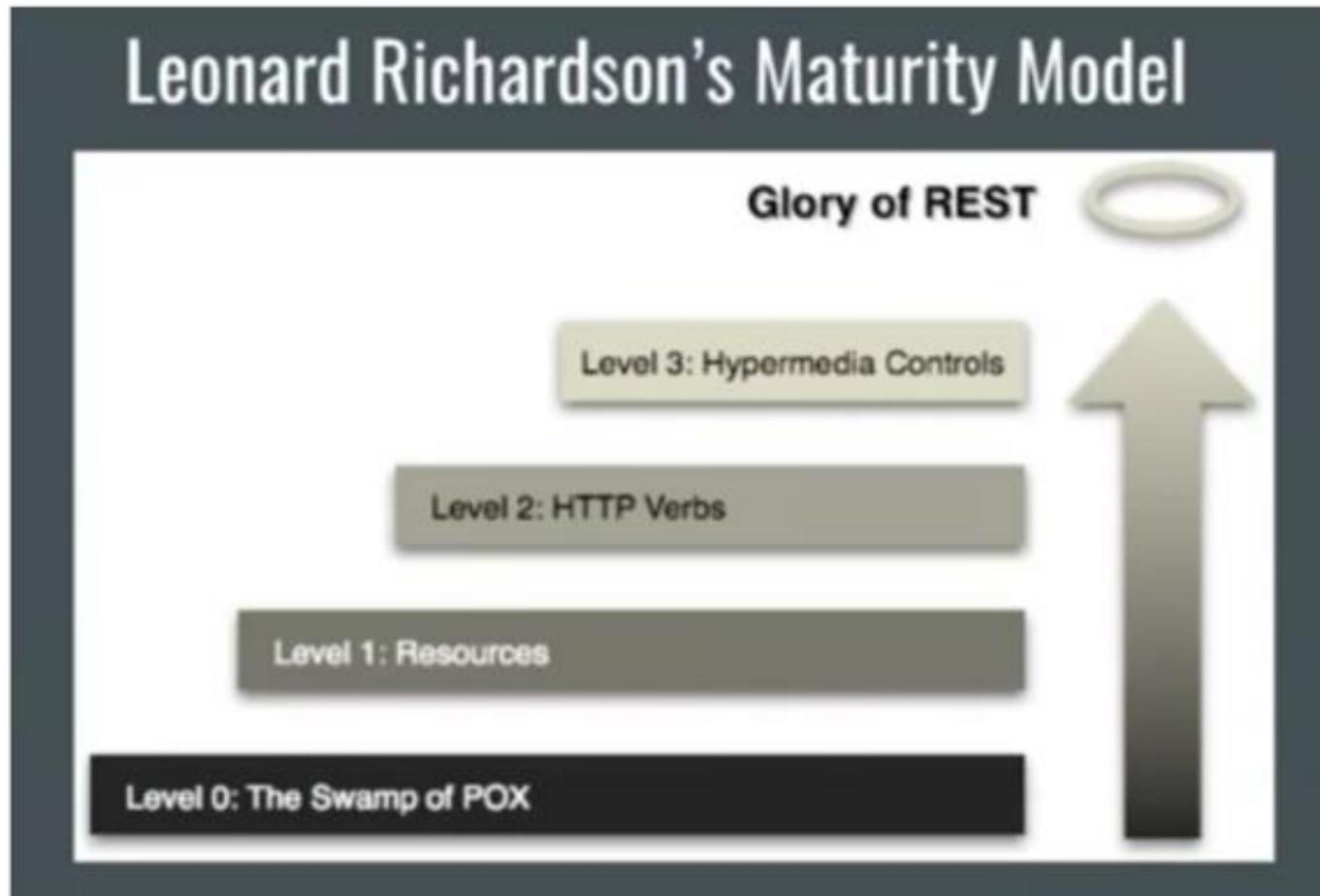
PS C:\WINDOWS\system32>
```

jenkins



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

Best Practice #1



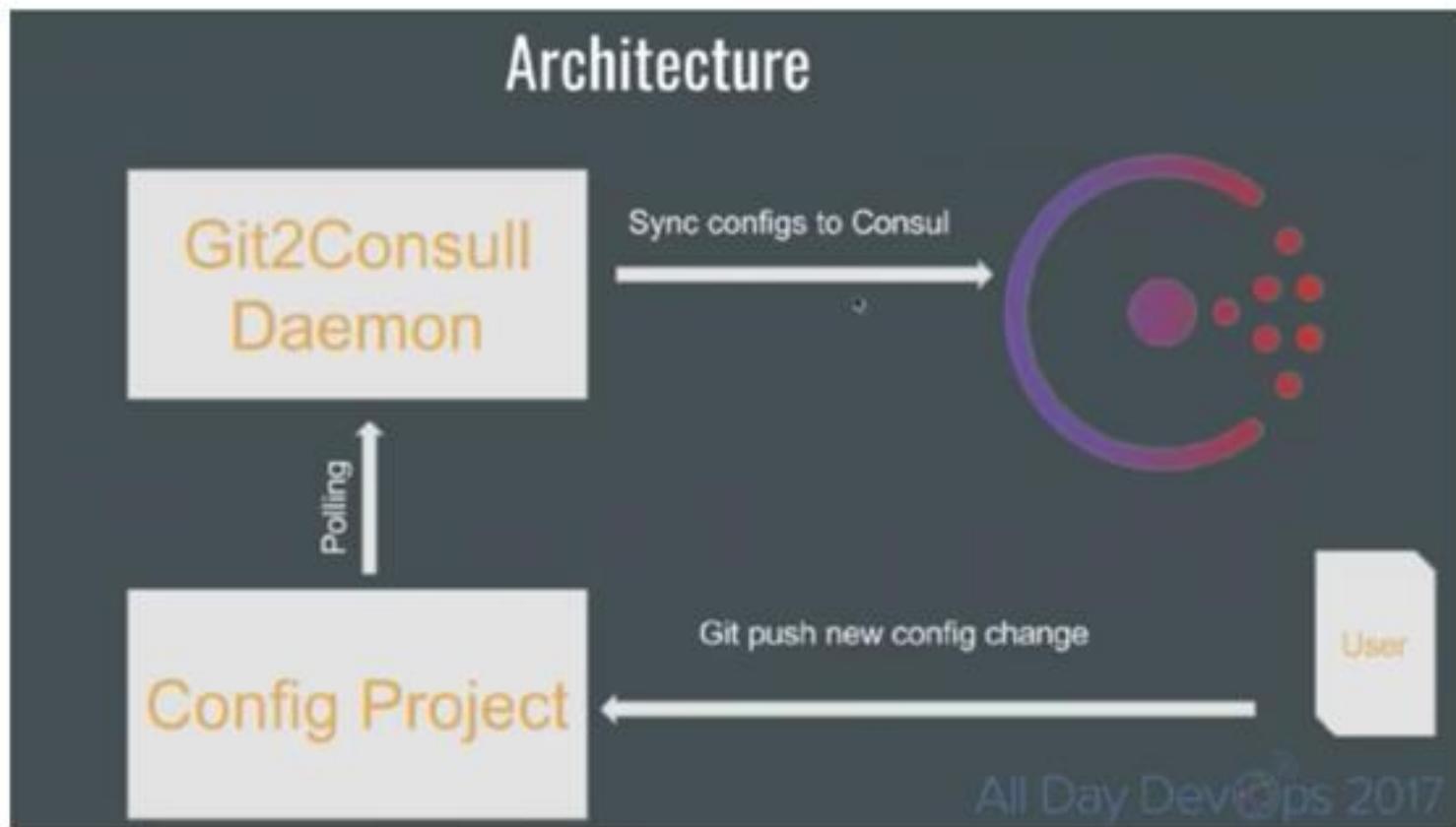


Best Practice #2

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



Best Practice #3



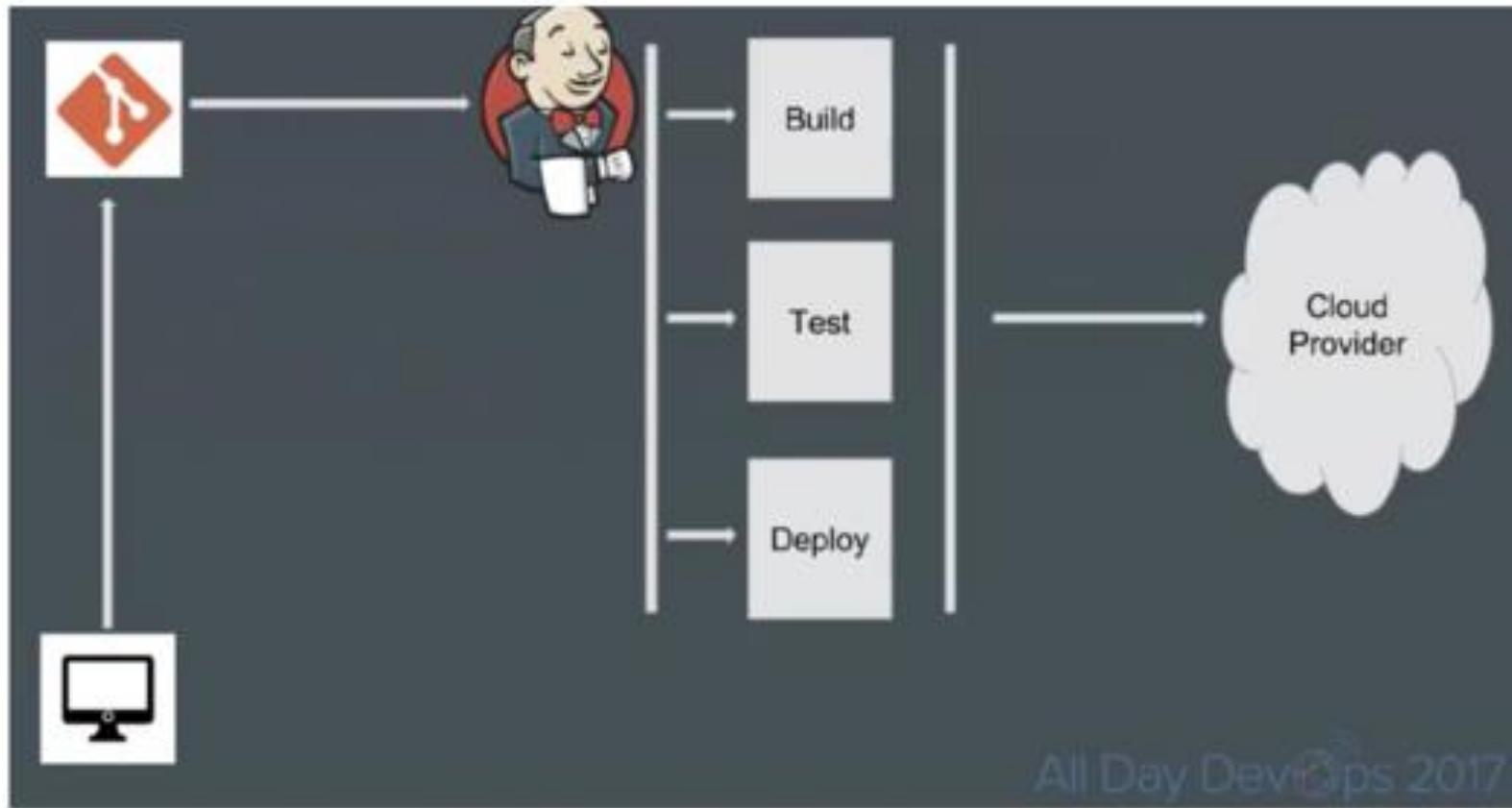


Best Practice #4

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

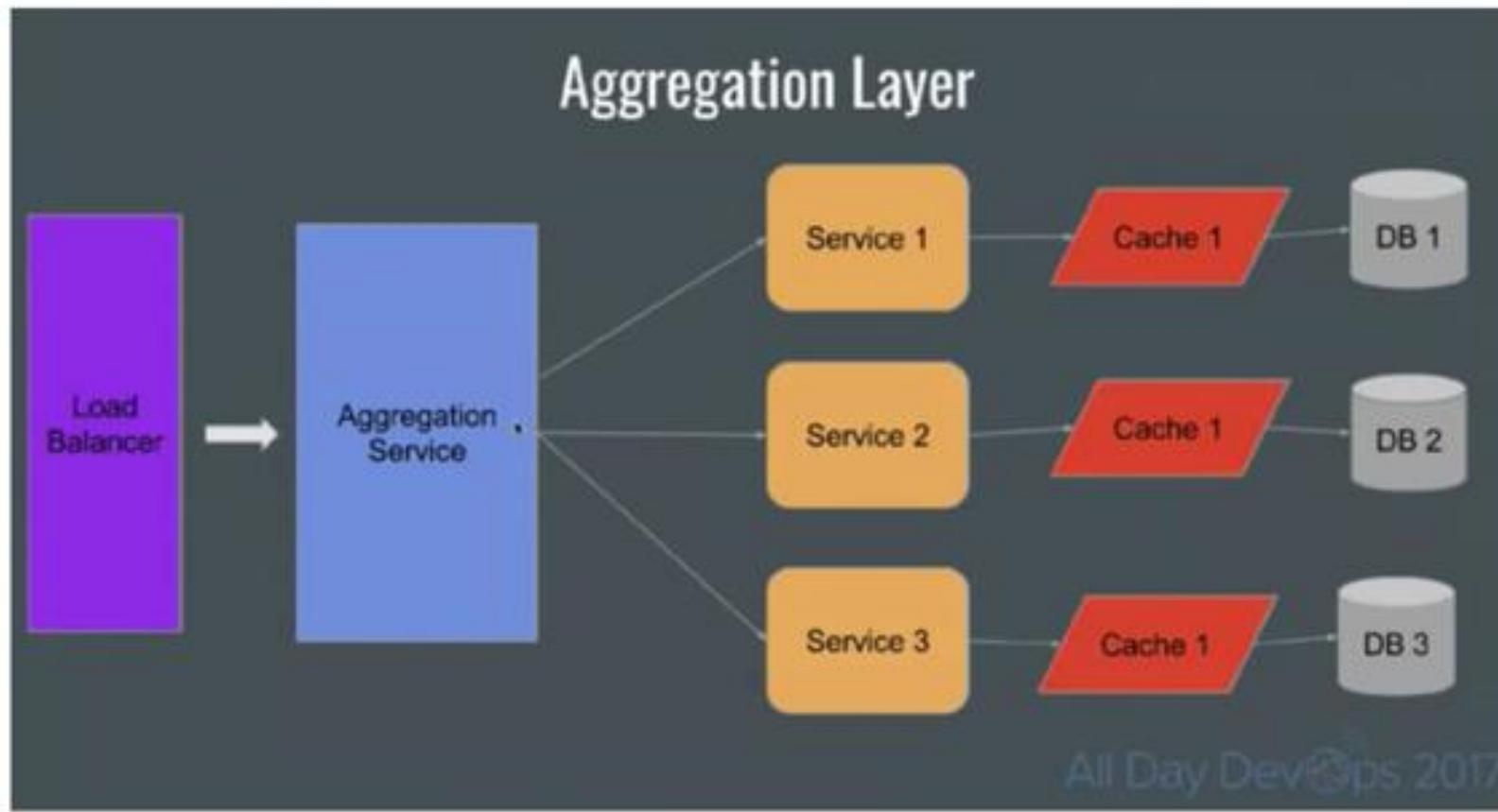




Best Practice #6



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.



Helpers

- Add-Migration InitialCreate
- Update-Database

Helpers



Screenshot of Microsoft Visual Studio showing the configuration of a .NET Core API project named "CustomerApi".

The "CustomerApi" tab is selected in the Solution Explorer. The "Debug" configuration is selected in the Configuration dropdown.

Debug Settings:

- Enable native code debugging
- Enable SQL Server debugging

Web Server Settings:

- App URL: https://localhost:44343/
- IIS Express Bitness: Default
- Hosting Model: Default (In Process)
- Enable SSL https://localhost:44343/ [Copy](#)
- Enable Anonymous Authentication
- Enable Windows Authentication

Output Window:

```
iisexpress.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.3\System.Xml.ReaderWriter.dll'. Skipped loading 'iisexpress.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.3\System.Net.WebClient.dll'. Skipped loading 'iisexpress.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.3\System.Security.Cryptography.Primitives.dll' 'iisexpress.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.3\System.Text.Encoding.CodePages.dll'. Skipped loading 'iisexpress.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.3\System.Diagnostics.Process.dll'. Skipped loading The thread 0x5050 has exited with code 0 (0x0). 'iisexpress.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.AspNetCore.App\3.1.3\Microsoft.AspNetCore.Http.Extensions.dll'. The thread 0x5e5c has exited with code 0 (0x0). The thread 0x1990 has exited with code 0 (0x0). The program '[19336] iisexpress.exe' has exited with code -1 (0xffffffff).
```

Toolbars and Status Bar:

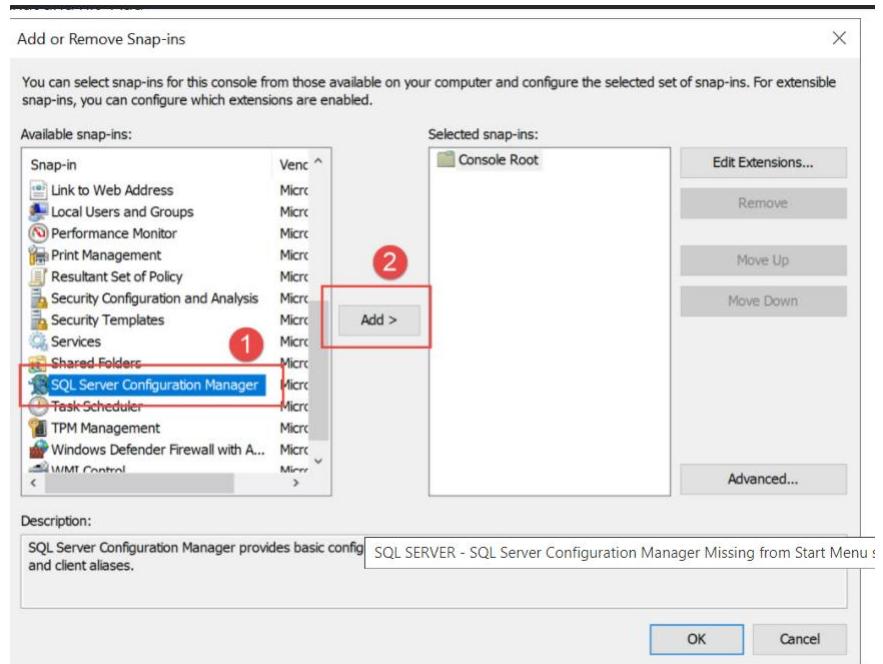
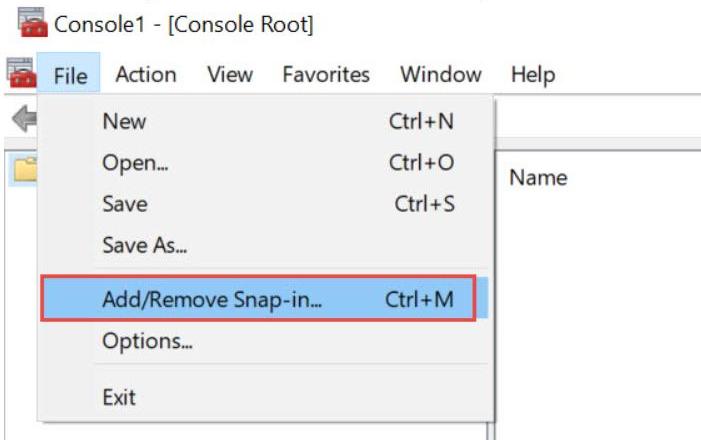
- File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
- Search (Ctrl+Q) CustomerApi
- R Live Share ADMIN
- Add to Source Control
- Ready
- Type here to search
- 23:00 23/08/2020 570

Helpers



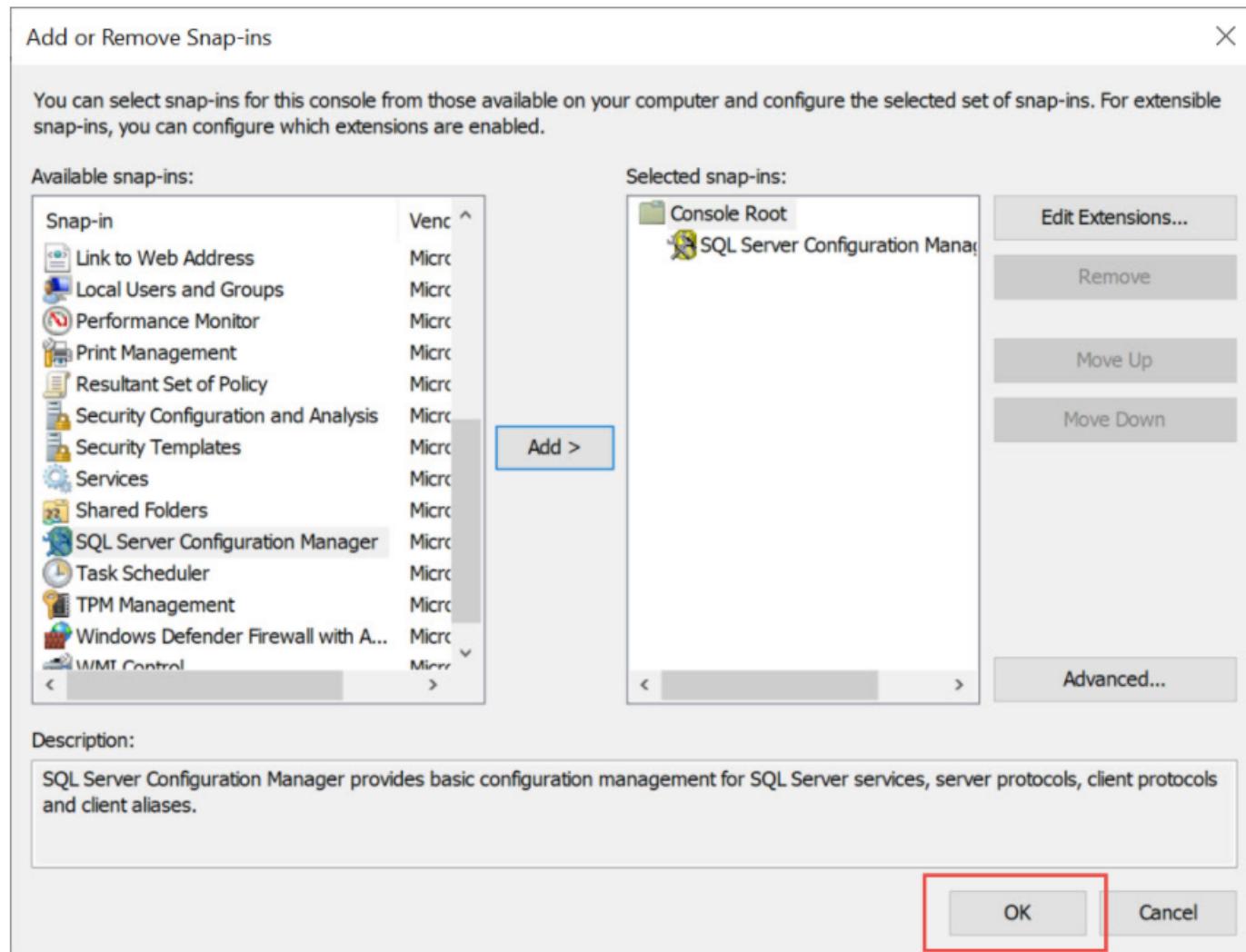
1. Open MMC.exe by going to Start > Run > mmc.exe

2. In the menu bar, go to "File" and choose "Add/Remove Snap-in"



Helpers

4. And then hit OK.





Helpers

- **docker container run -p 1433:1433 -d --name mssql -v mssql_data:/var/opt/mssql -e SA_PASSWORD=vignesh -e ACCEPT_EULA=Y --network=test_network microsoft/mssql-server-linux**
- **docker run -e "ACCEPT_EULA=Y" -e "SA_PASSWORD=Vignesh@95" -p 1403:1433 -d --name mssqllinux microsoft/mssql-server-linux**



Helpers

- docker exec -i mssqllinux /opt/mssql-tools/bin/sqlcmd -S localhost -U SA -P "Vignesh@95" -Q "CREATE DATABASE CUSTOMERDB"
- docker exec -i mssqllinux /opt/mssql-tools/bin/sqlcmd -S localhost -U SA -P "Vignesh@95" -Q "SELECT name, database_id, create_date from sys.databases"



Helpers

- docker exec -it mssqllinux bash
- /opt/mssql-tools/bin/sqlcmd -S localhost -U SA -P Vignesh@95
- Use master
- GO



Helpers

- To configure the remote access option
- In Object Explorer, right-click a server and select Properties.
- Click the Connections node.
- Under Remote server connections, select or clear the Allow remote connections to this server check box.
- Or
- `EXEC sp_configure 'remote access', 0 ; GO RECONFIGURE ; GO`

Questions



Module Summary

- Microservices using docker and Kubernets.
- Message, Channel and Adapter
- Understood the different Component Integration
- Understood the Event-Driven Architecture

