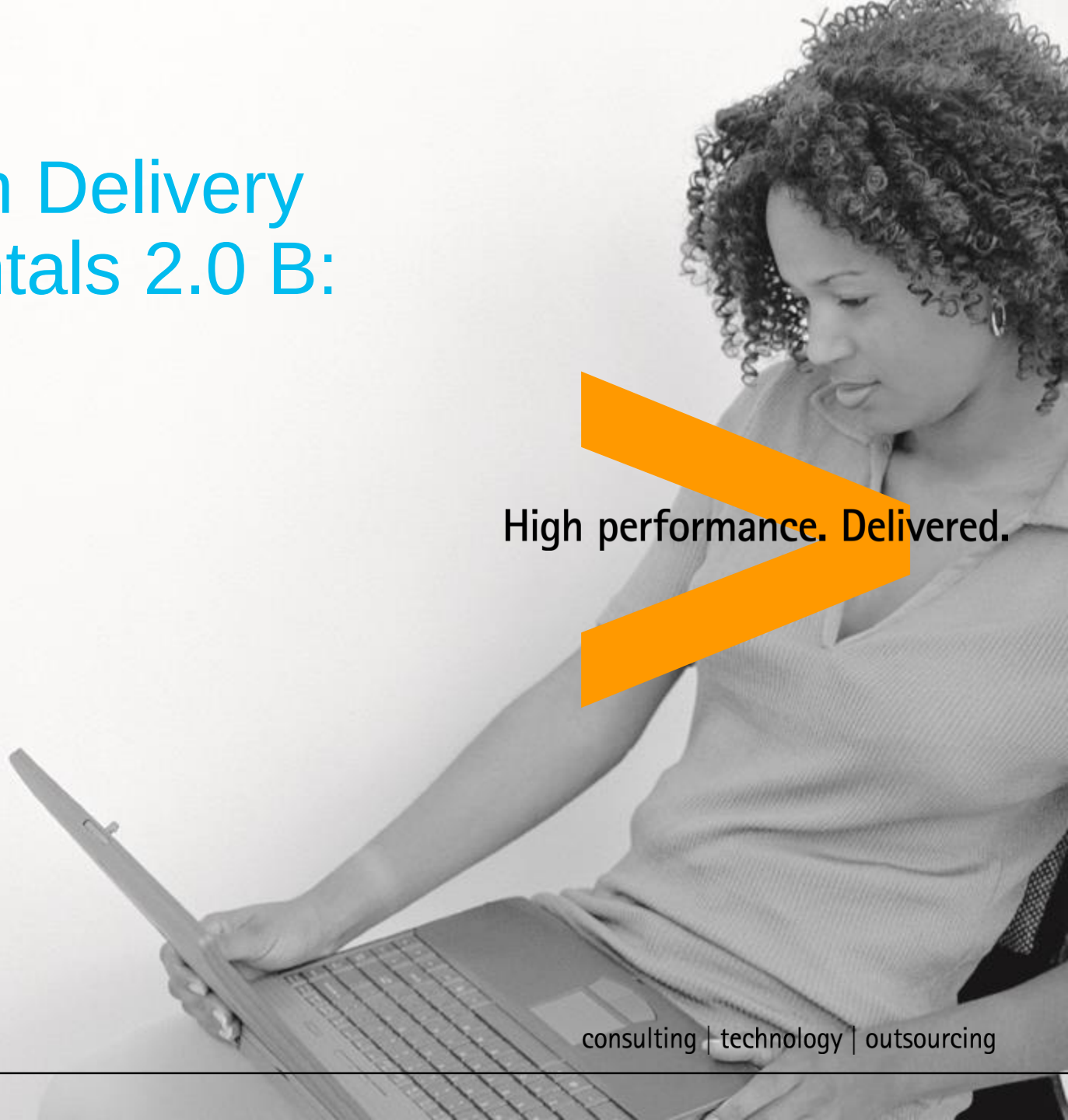
Application Delivery Fundamentals 2.0 B: Java

Micro Service



High performance. Delivered.

consulting | technology | outsourcing

Micro Service

Goals

- Independent deployment of components
- Independent scaling of components
- Independent implementation stacks for each component
- Easy self-serve deployments of components
- Repeatable deployments of components (external configuration management)
- Deployments without service interruptions

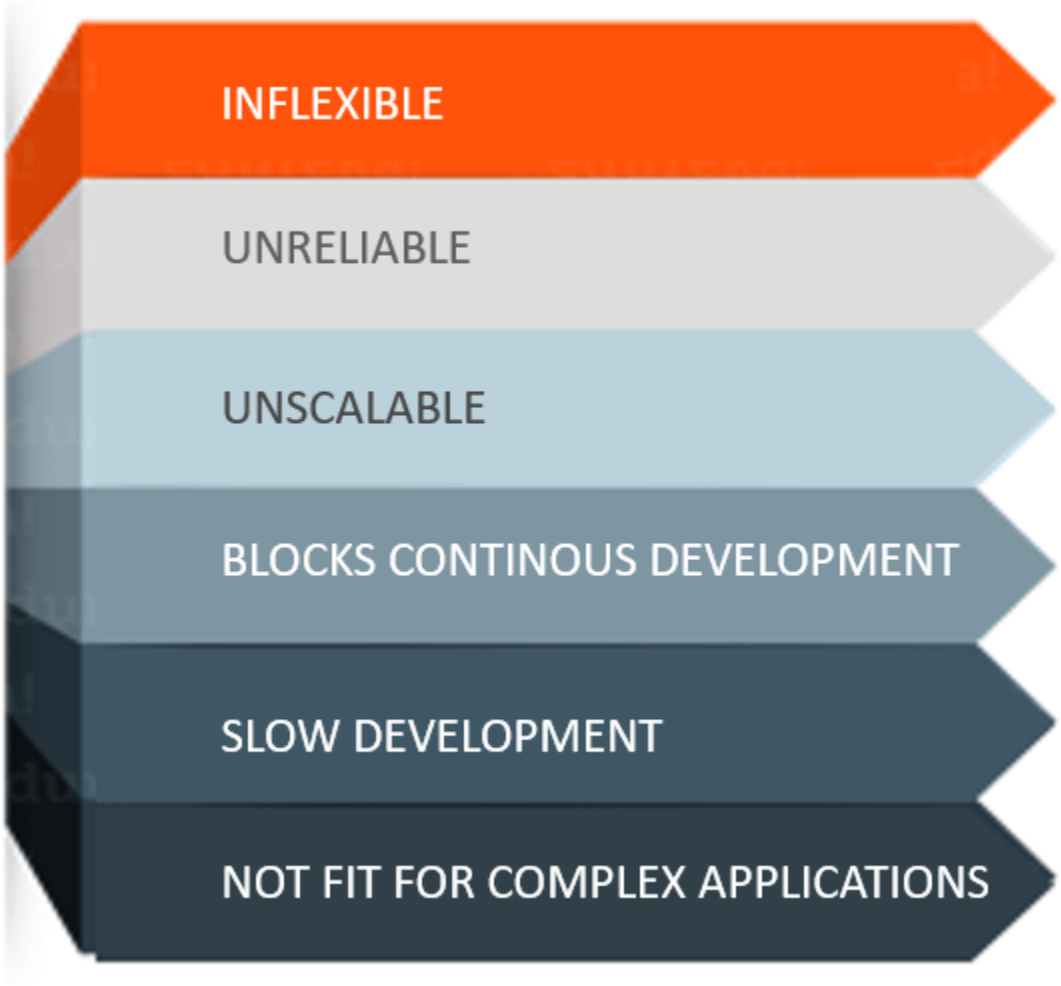
Micro Service

Goals

- Protection of system availability from individual Instance failure
- Automatic replacement of component instances when they fail (self-healing)
- Easy scaling of components by adjusting a simple parameter value
- Canary testing
- "Red/black" or "blue/green" deployments
- Instant reversal of new revision deployments

Monolithic Architecture

Microservices

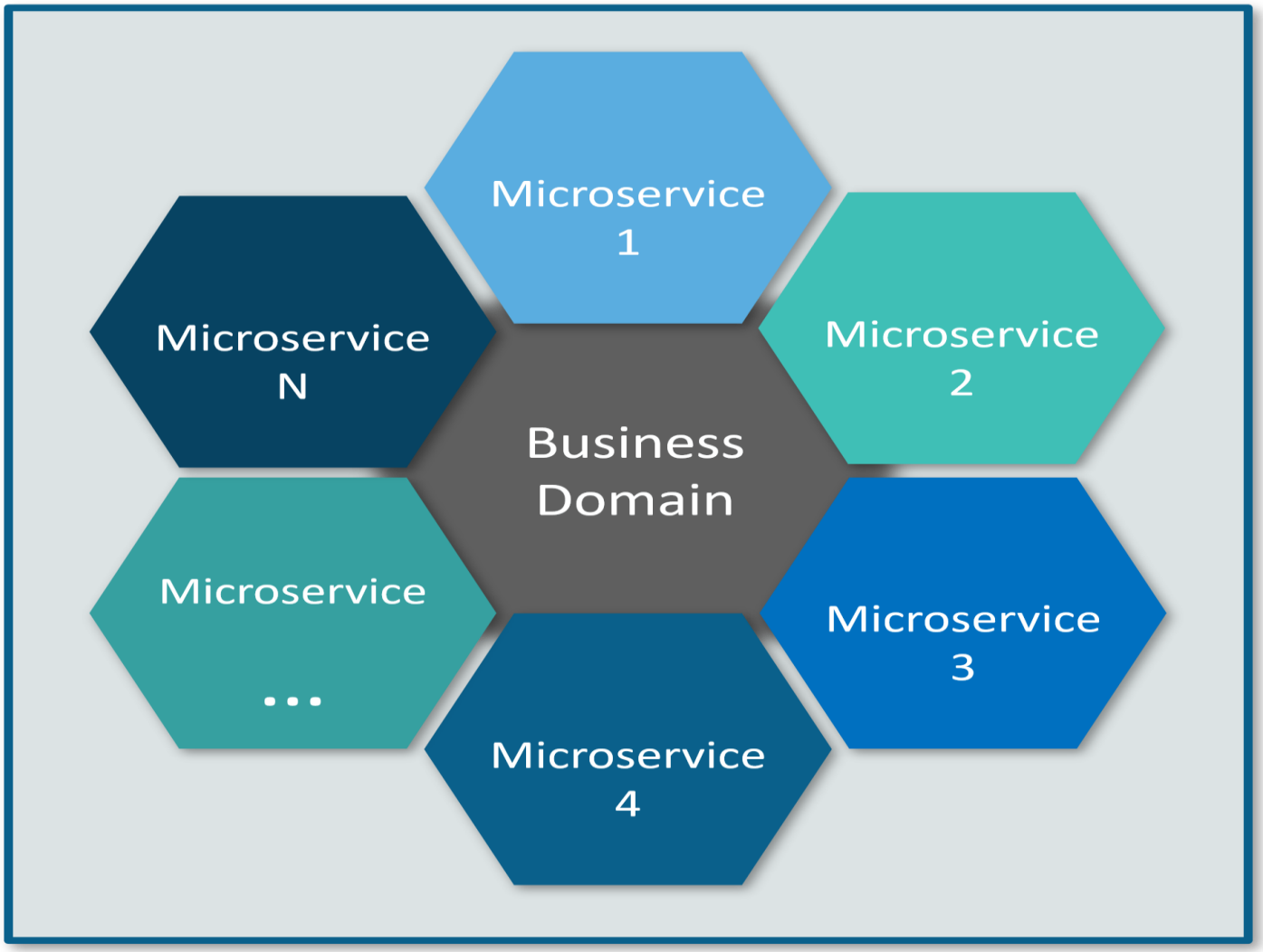


Micro Service

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

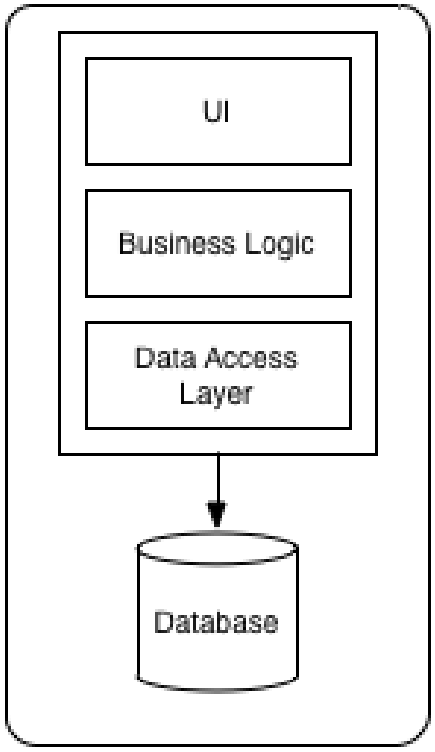
These services are built around business capabilities and independently deployable by fully automated deployment machinery...contd

Micro Service

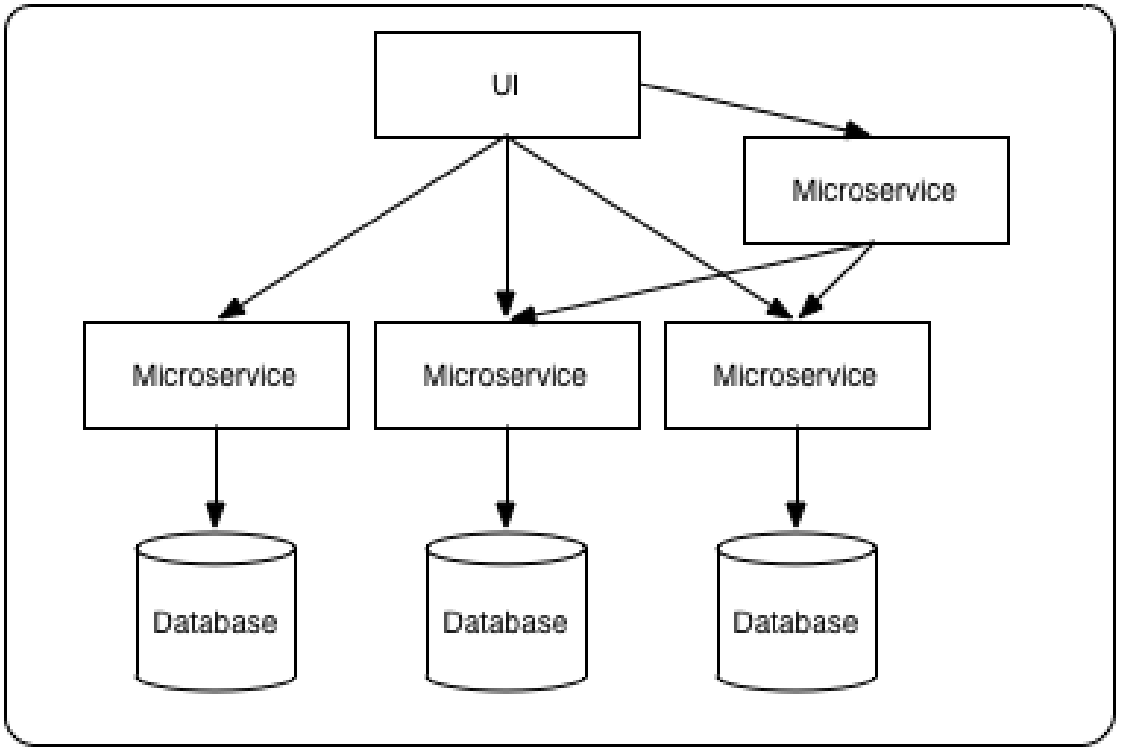
There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies - James Lewis and Martin Fowler

MicroService

Microservices

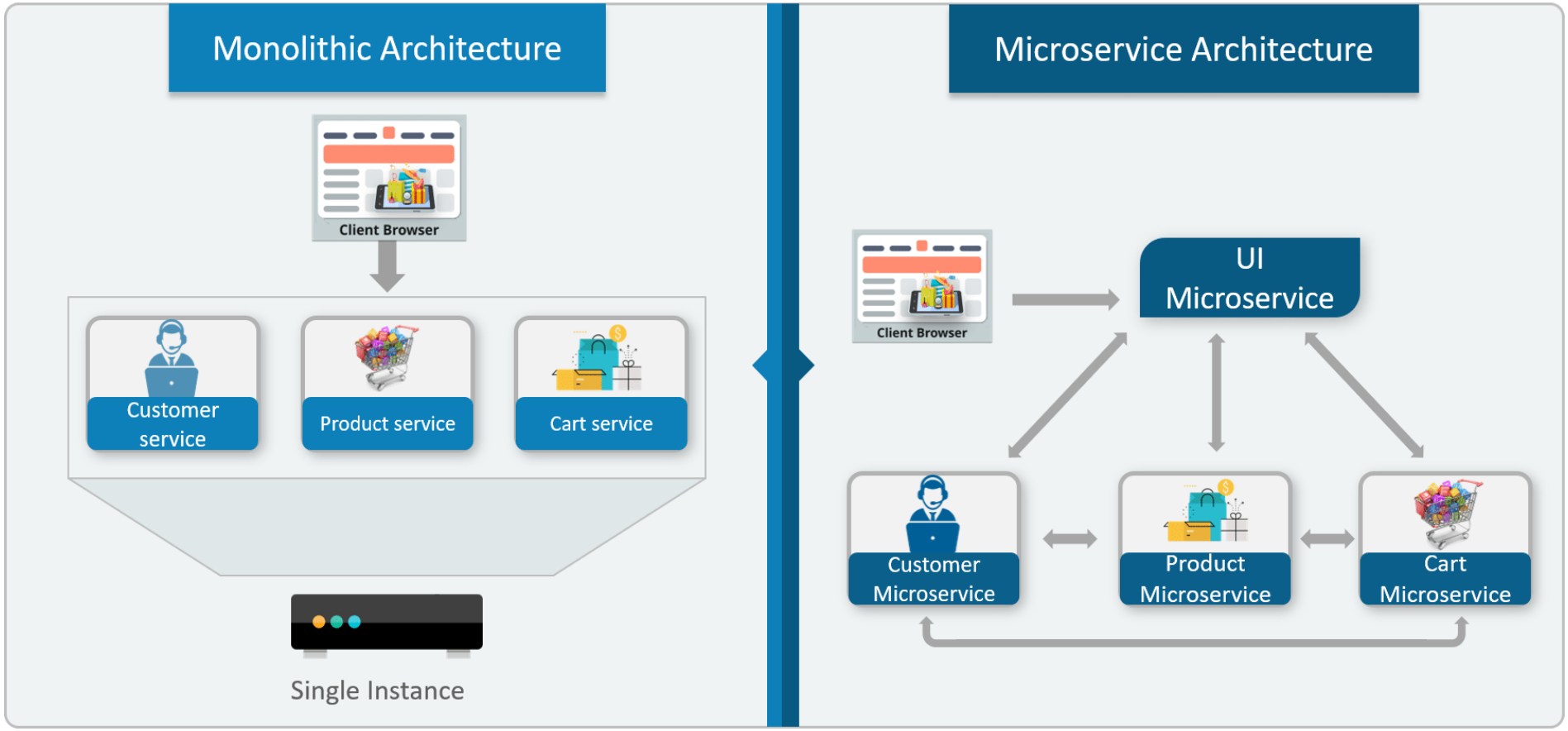


Monolithic Architecture



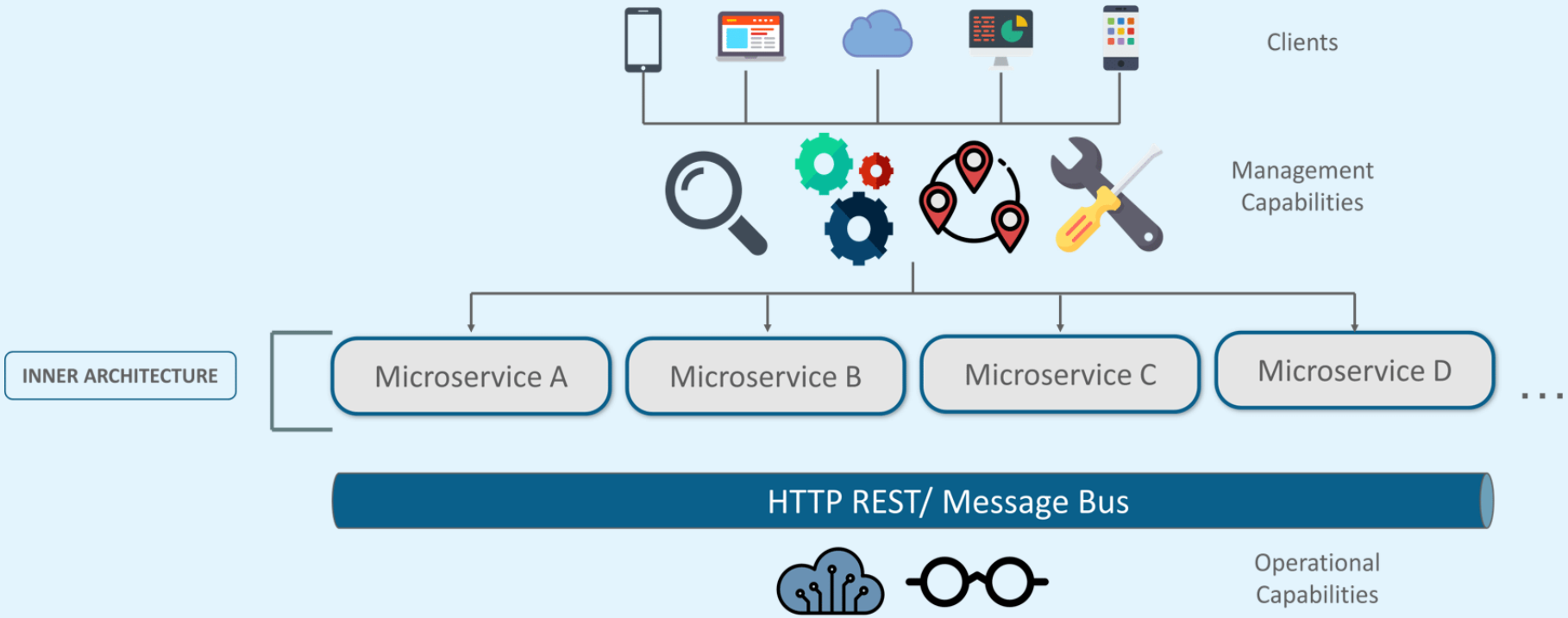
Microservices Architecture

Microservices



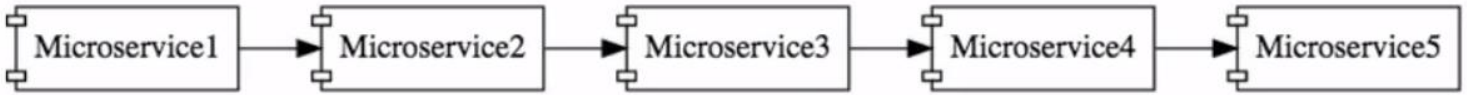
Microservice Architecture

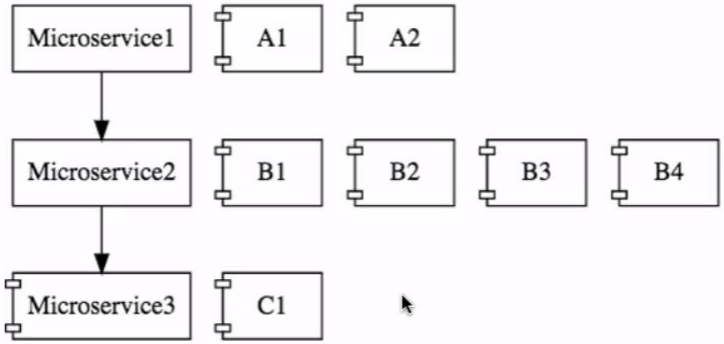
Microservices



Micro Service

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





What is Microservices

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

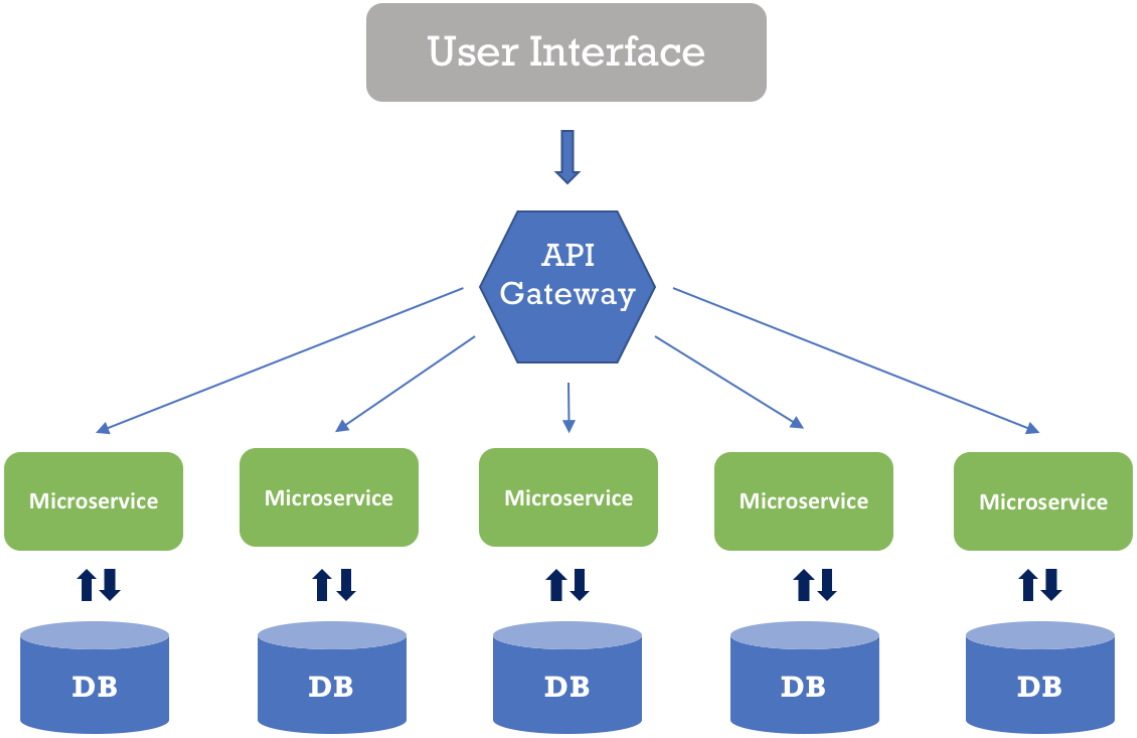
What is Microservices

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

What is Microservices

Microservices

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

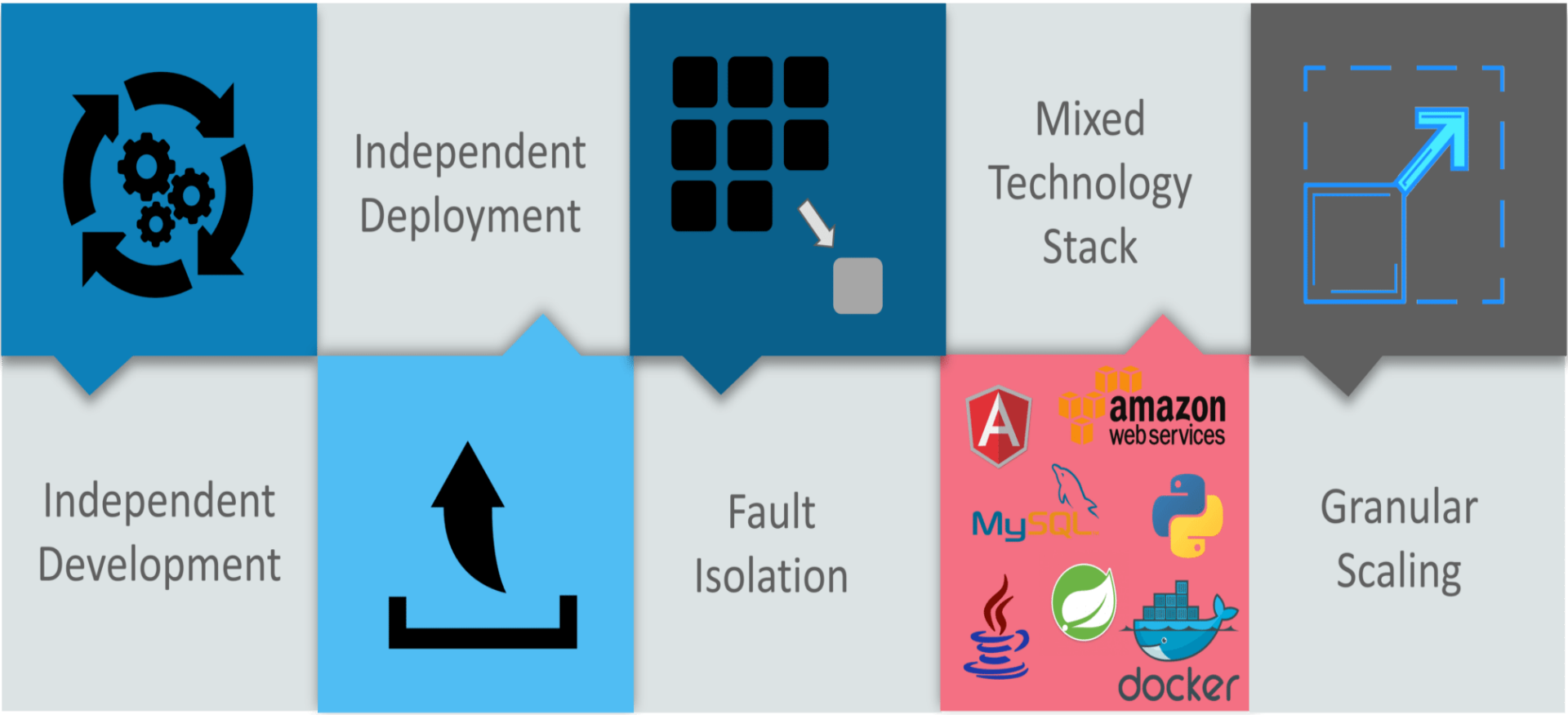


Microservices

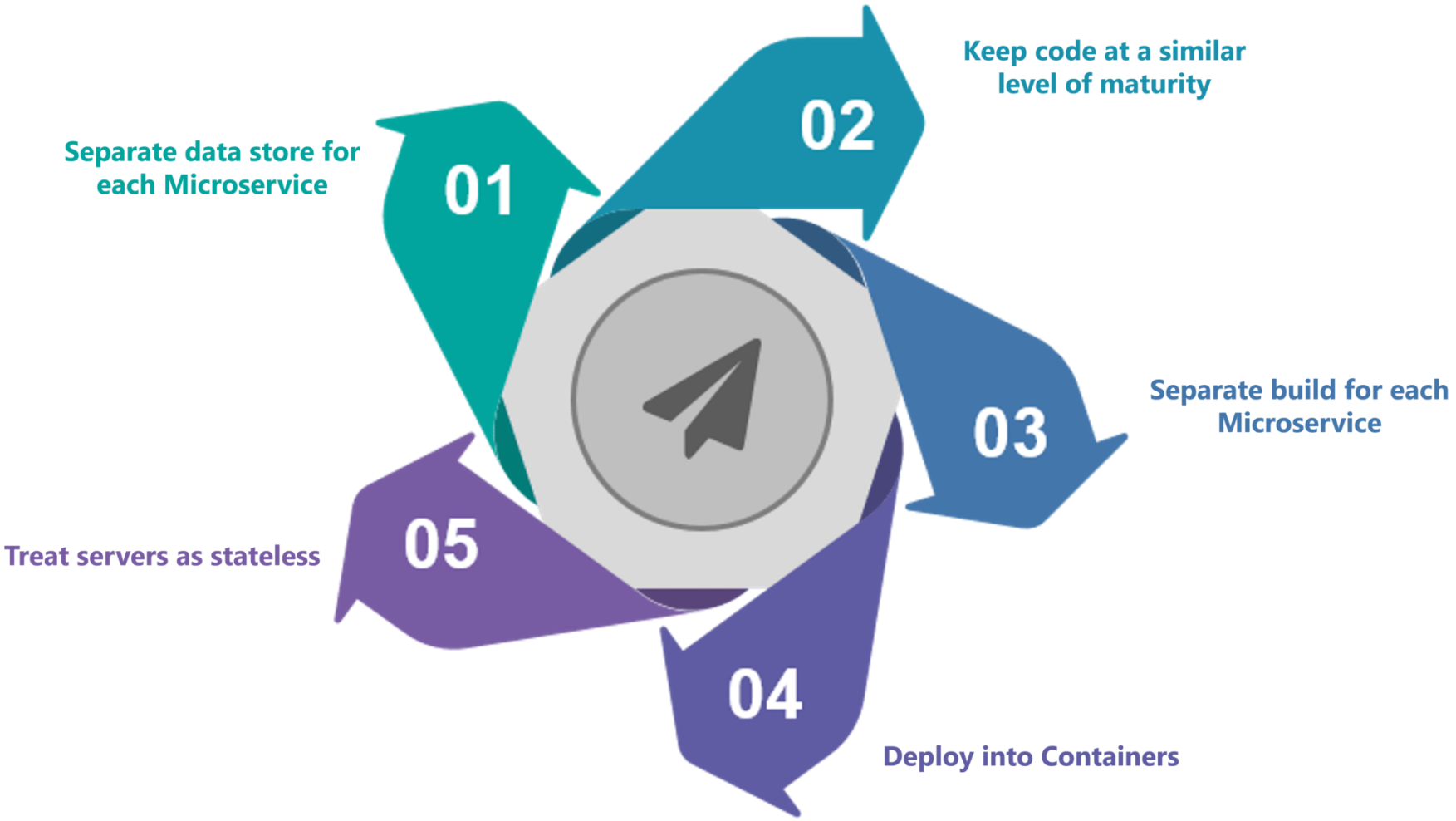
Microservices Features

Advantages Of Microservices

Microservices



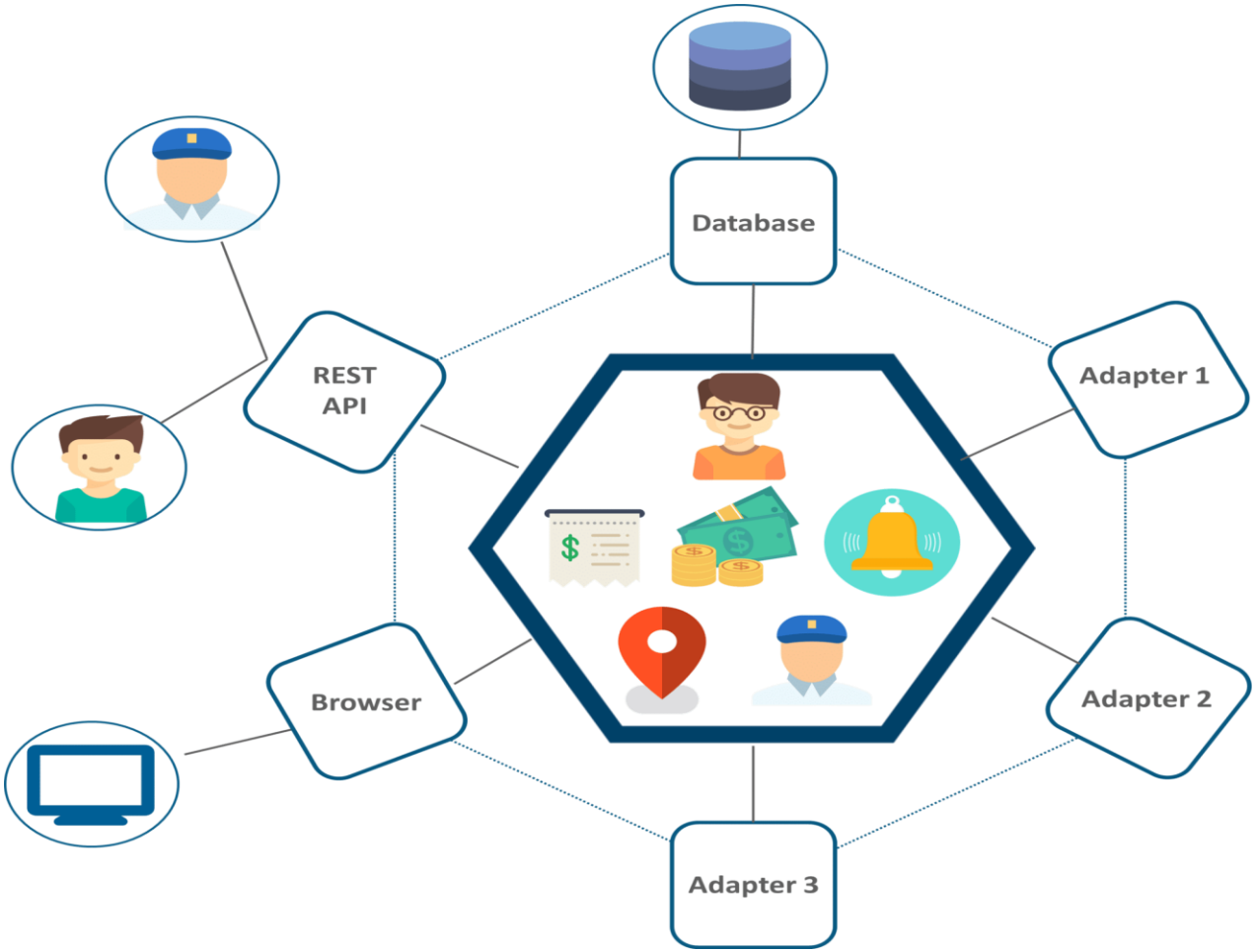
Best Practices To Design Microservices



UBER CASE STUDY

Microservices

UBER's Previous Architecture

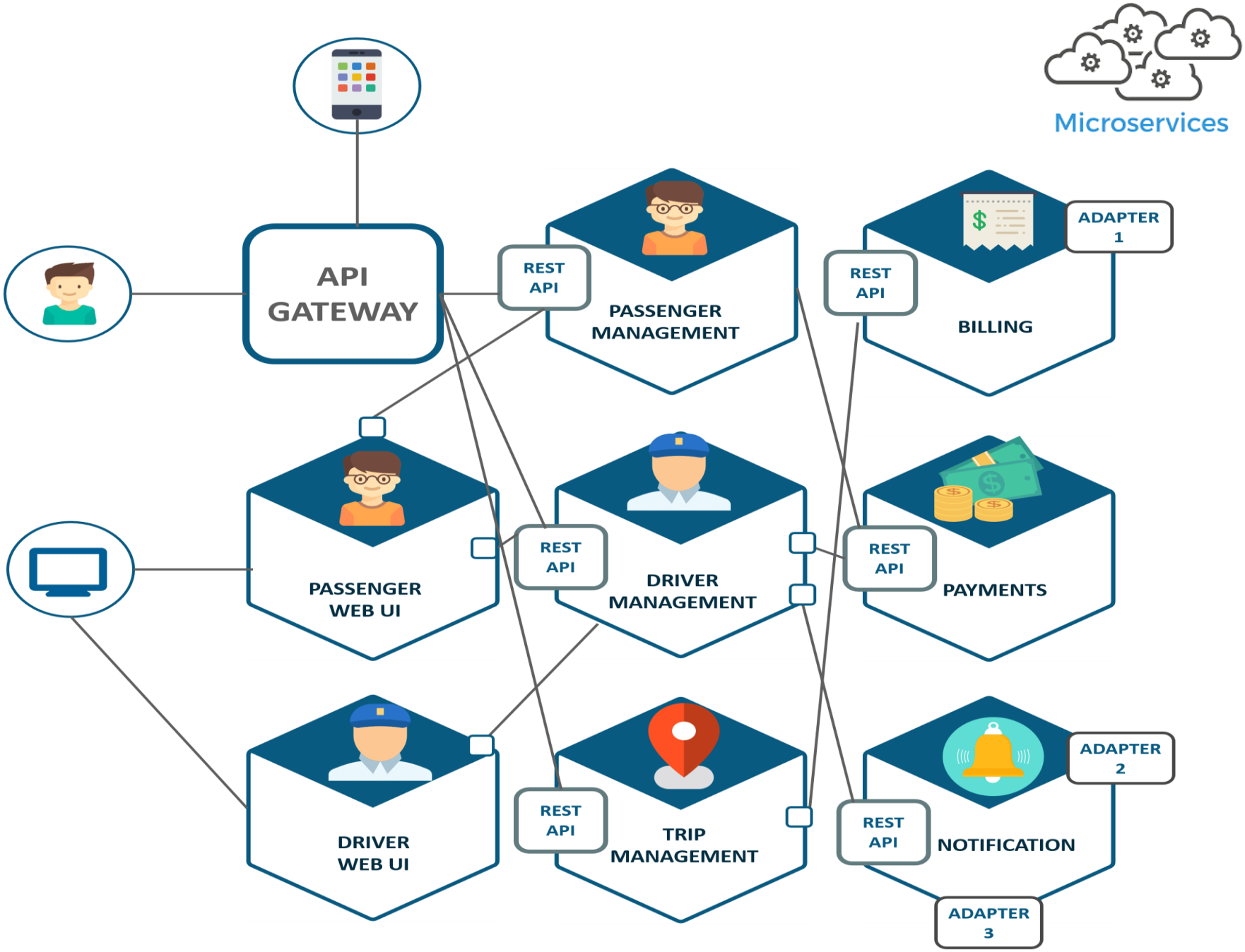


Problem Statement

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

Solution

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

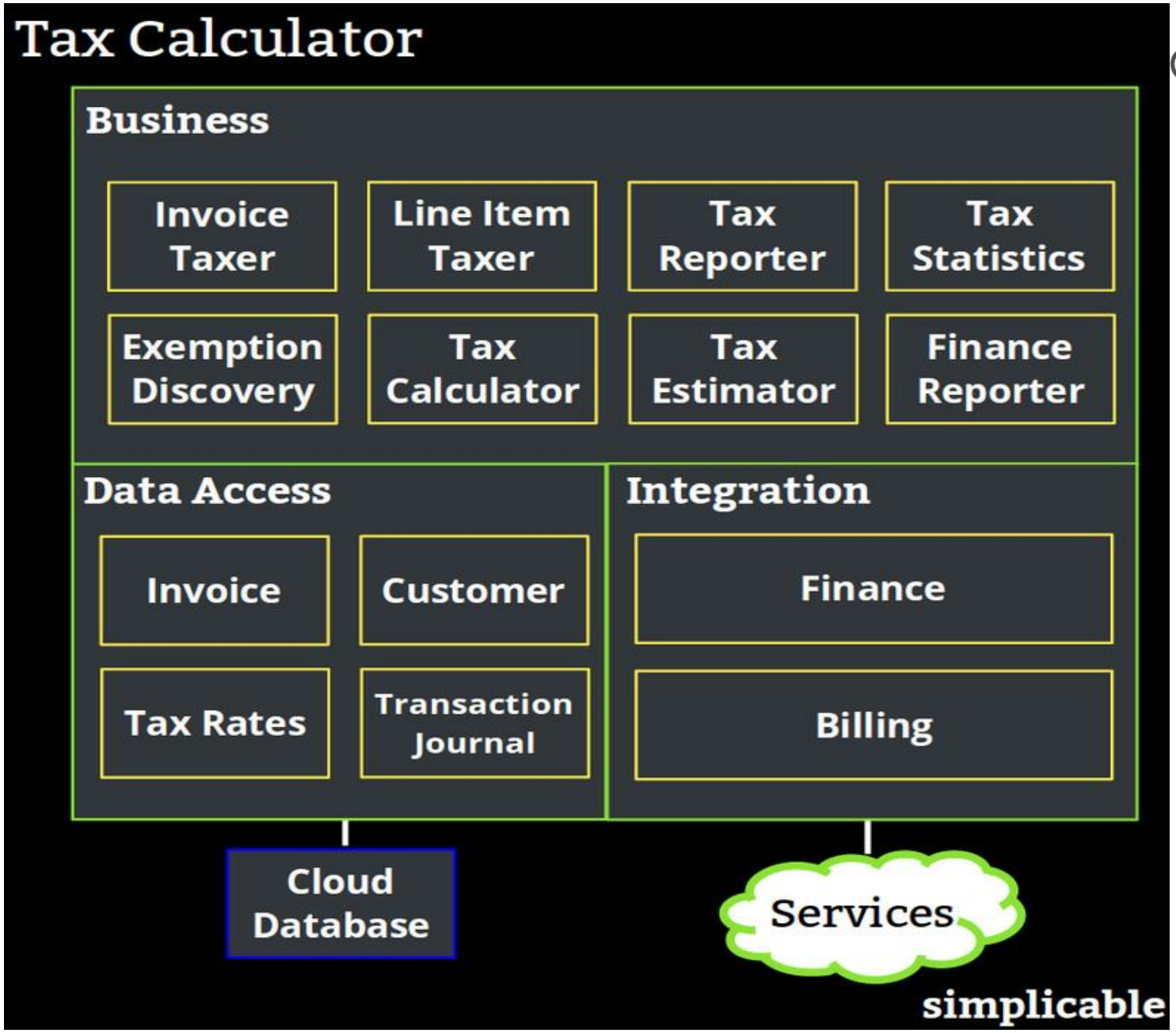


What are the Benefits of Microservices?

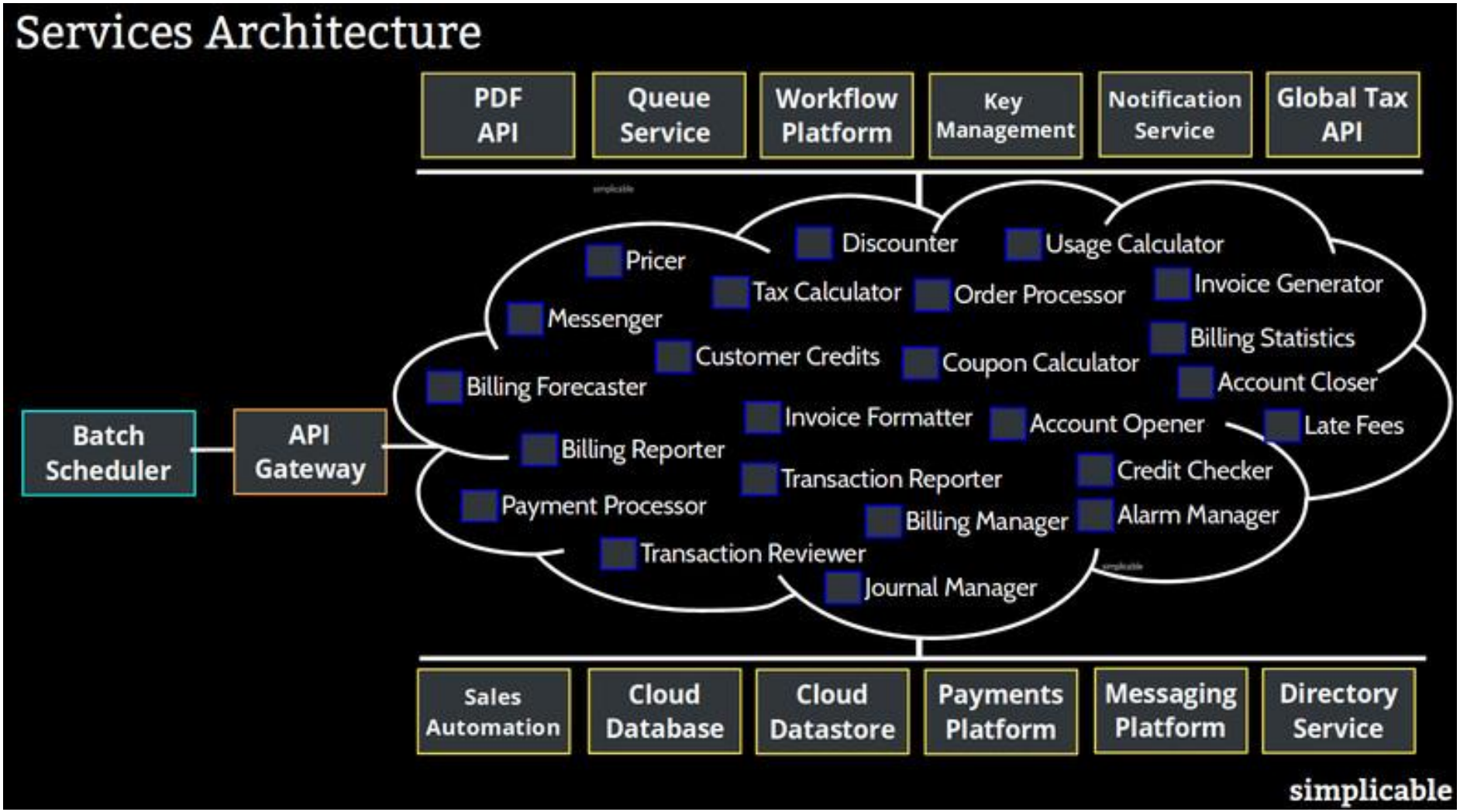
- Smaller applications are not dependent on the same coding language, the developers can use the programming language that they are most familiar with.
- That helps developers come up with a program faster with lower costs and fewer bugs.
- The agility and low costs can also come from being able to reuse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

Existing Scenarios

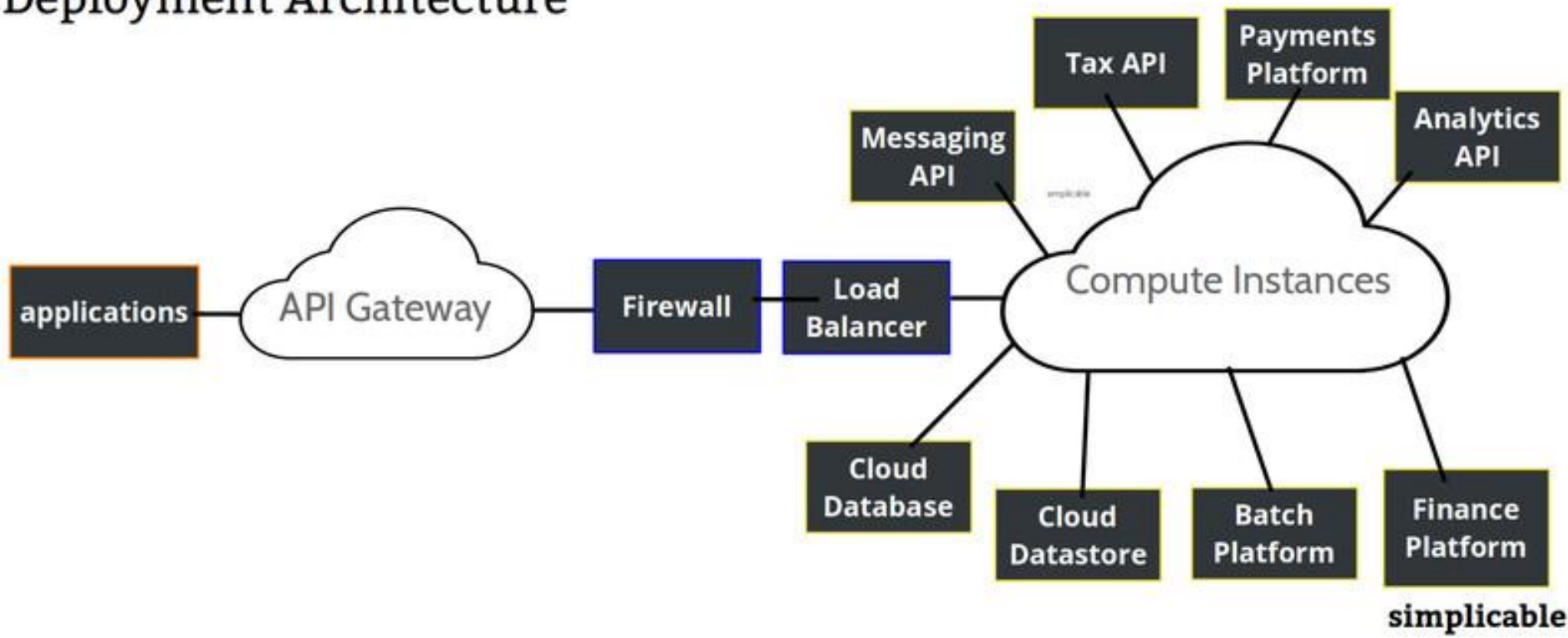
- Netflix has a widespread architecture that has evolved from monolithic to SOA. It receives more than *one billion*calls every day, from more than 800 different types of devices, to its streaming-video API. Each API call then prompts around five additional calls to the backend service.
- Amazon has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



Microservices



Deployment Architecture



Principles of Microservices

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

Principles of Microservices

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

Principles of Microservices

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

Principles of Microservices

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

Principles of Microservices

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

Principles of Microservices

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

Principles of Microservices

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

Benefits of Microservices

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

Benefits of Microservices

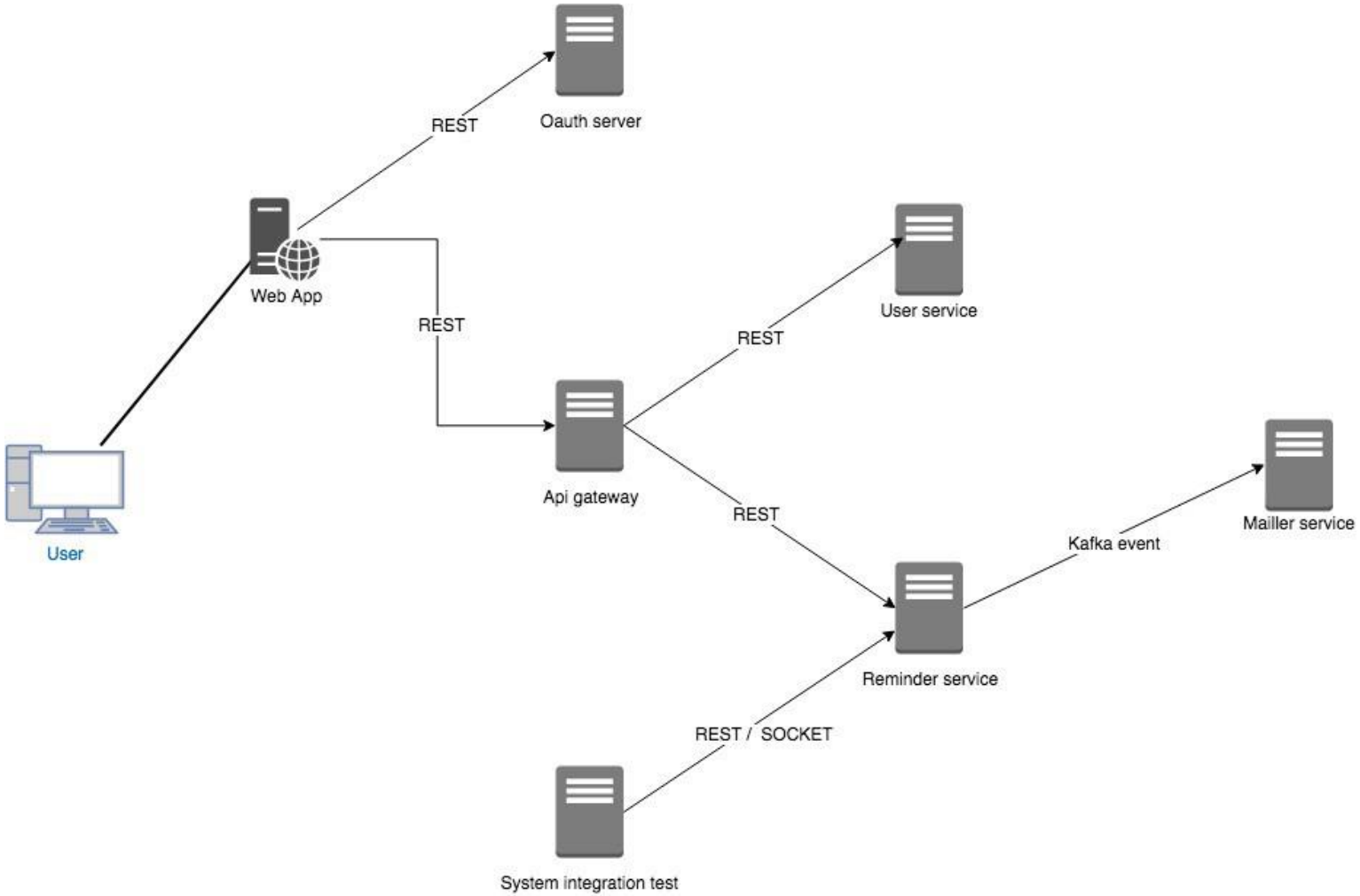
- Microservices enable to implement selective scalability i.e. each service could be independently scaled up or down and cost of scaling is comparatively less than monolithic approach.
- Microservices are self-contained, independent deployment modules enabling the substitution of one microservice with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

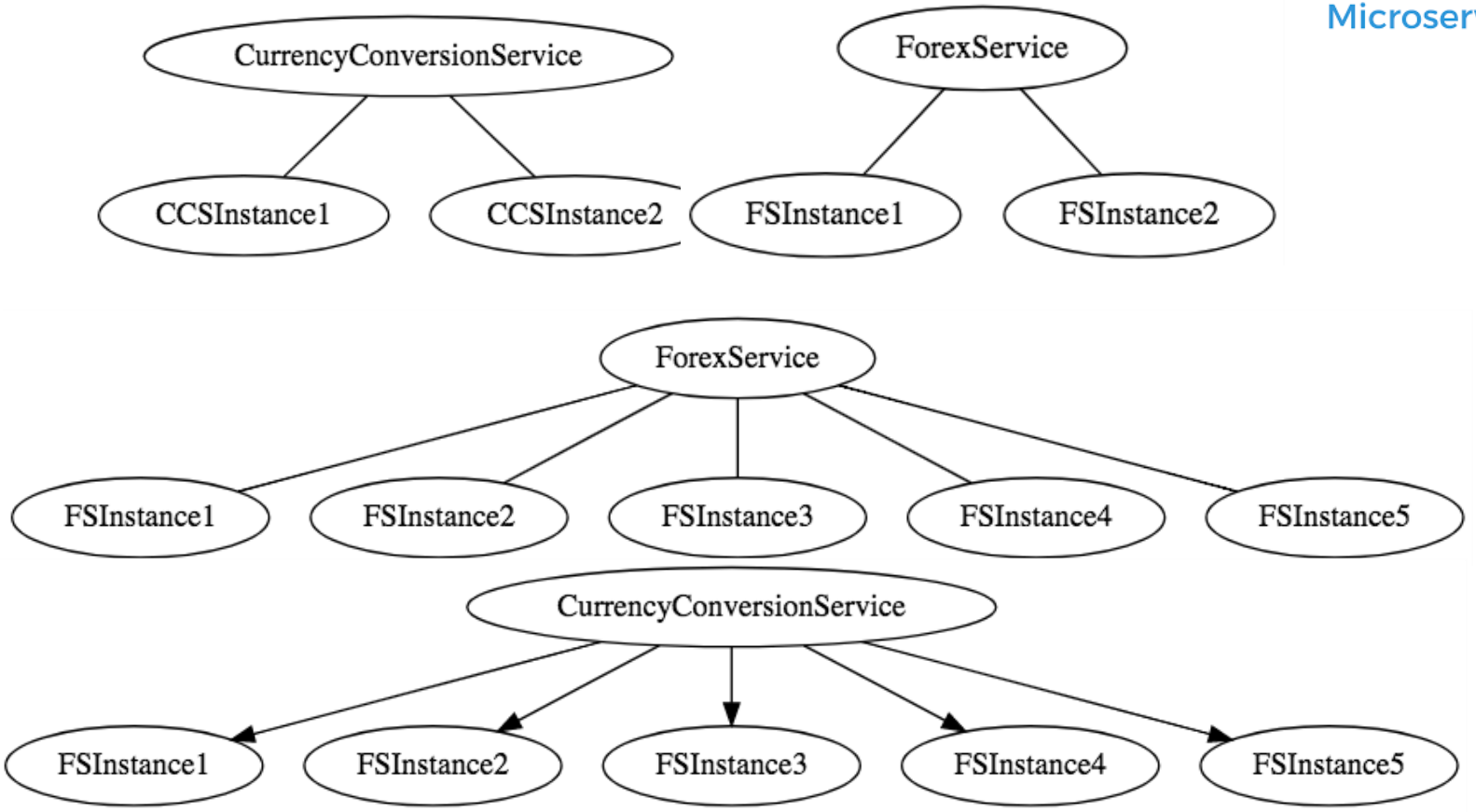
Benefits of Microservices

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

Benefits of Microservices

- As microservices package the service runtime environment along with the service itself, this enables having multiple versions of the service to coexist in the same environment.
- And finally, microservices also enable smaller, focused agile teams for development. Teams will be organized based on the boundaries of microservices.





Examples of Microservices Frameworks for Java

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

Pros

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

Pros

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

Pros

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

Cons

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

Cons

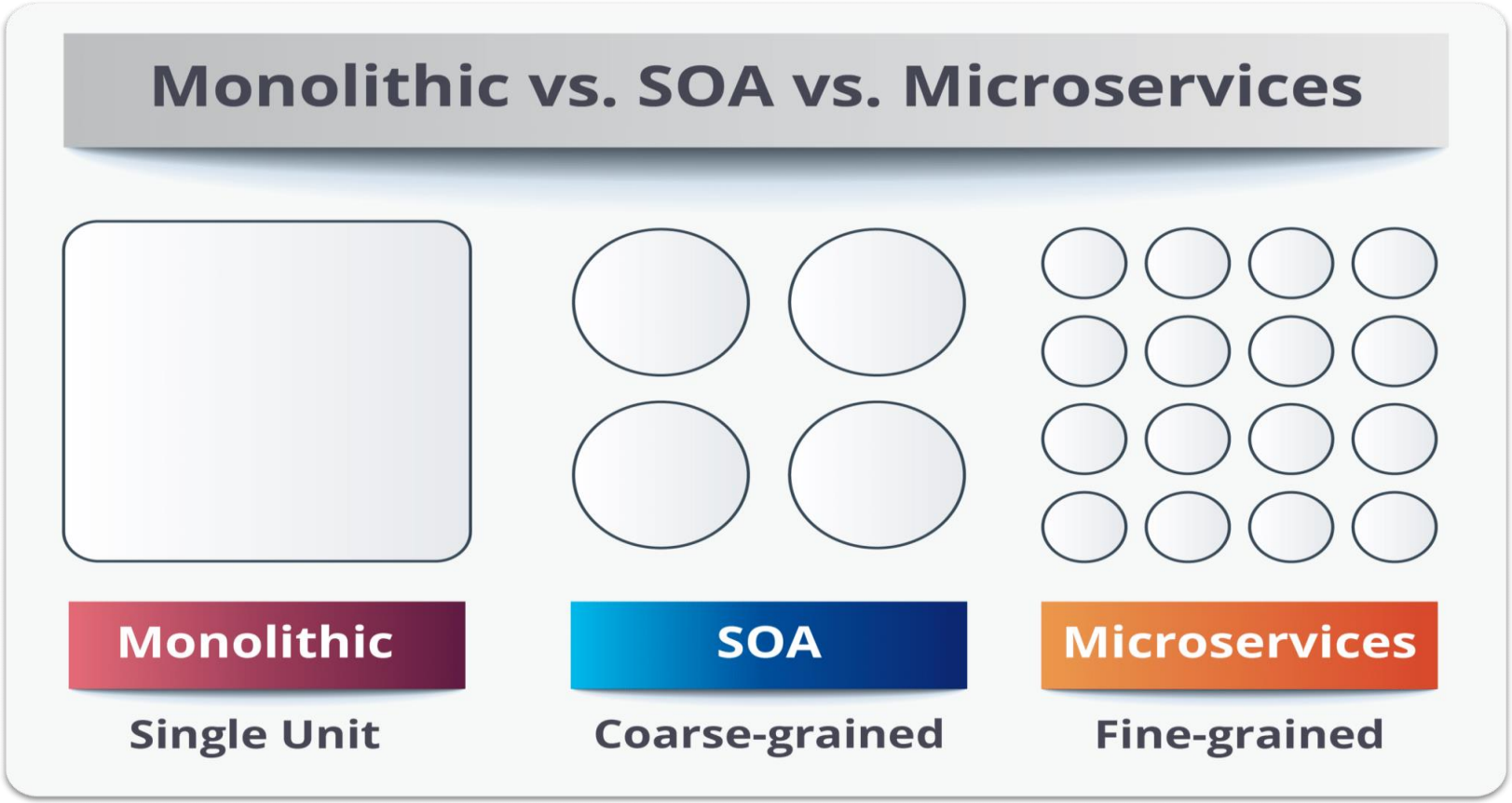
- When number of services increases, integration and managing whole products can become complicated
- In addition to several complexities of monolithic architecture, the developers have to deal with the additional complexity of a distributed system
- Developers have to put additional effort into implementing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

Cons

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

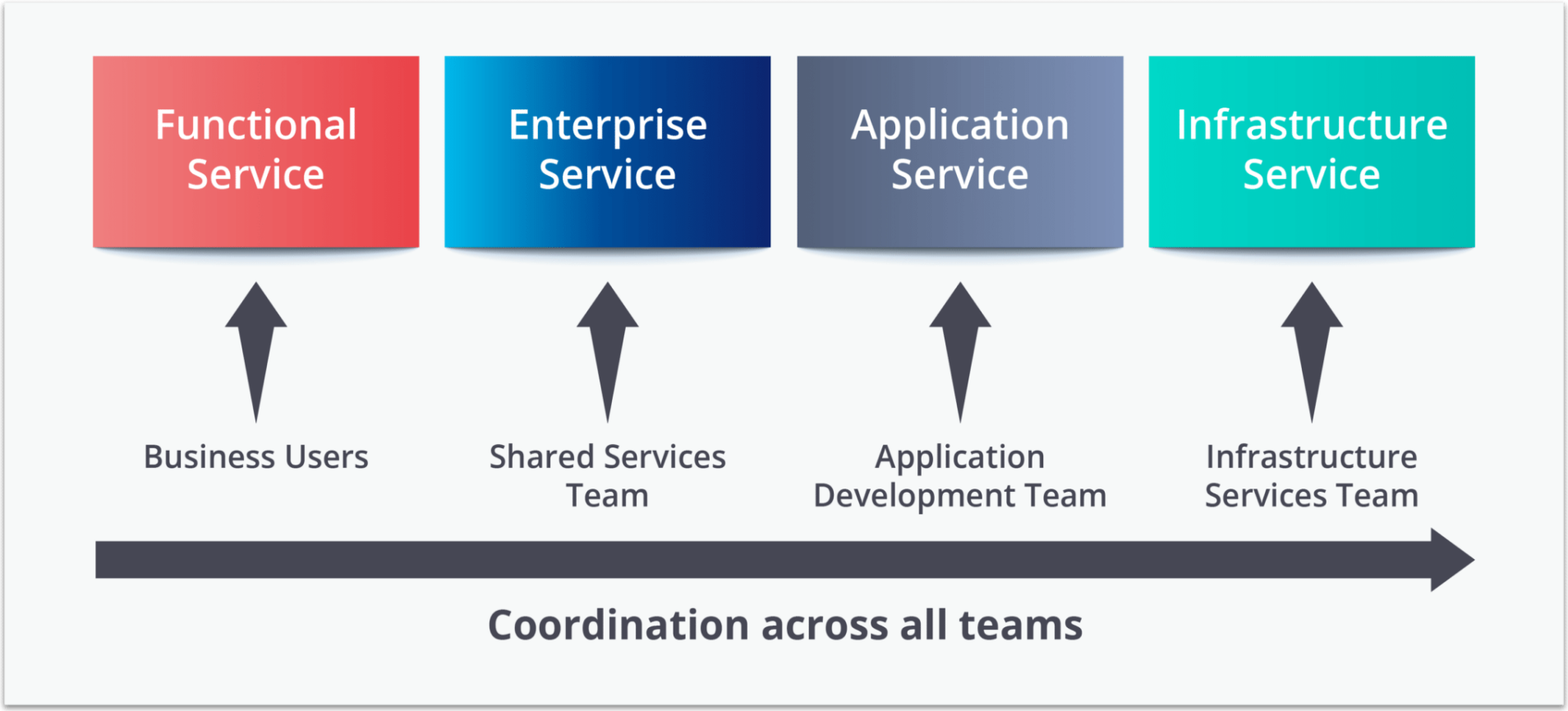
SOA vs Microservices

Microservices



Service Oriented Architecture

Microservices



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.

Scaling Challenge

X-AXIS SCALING

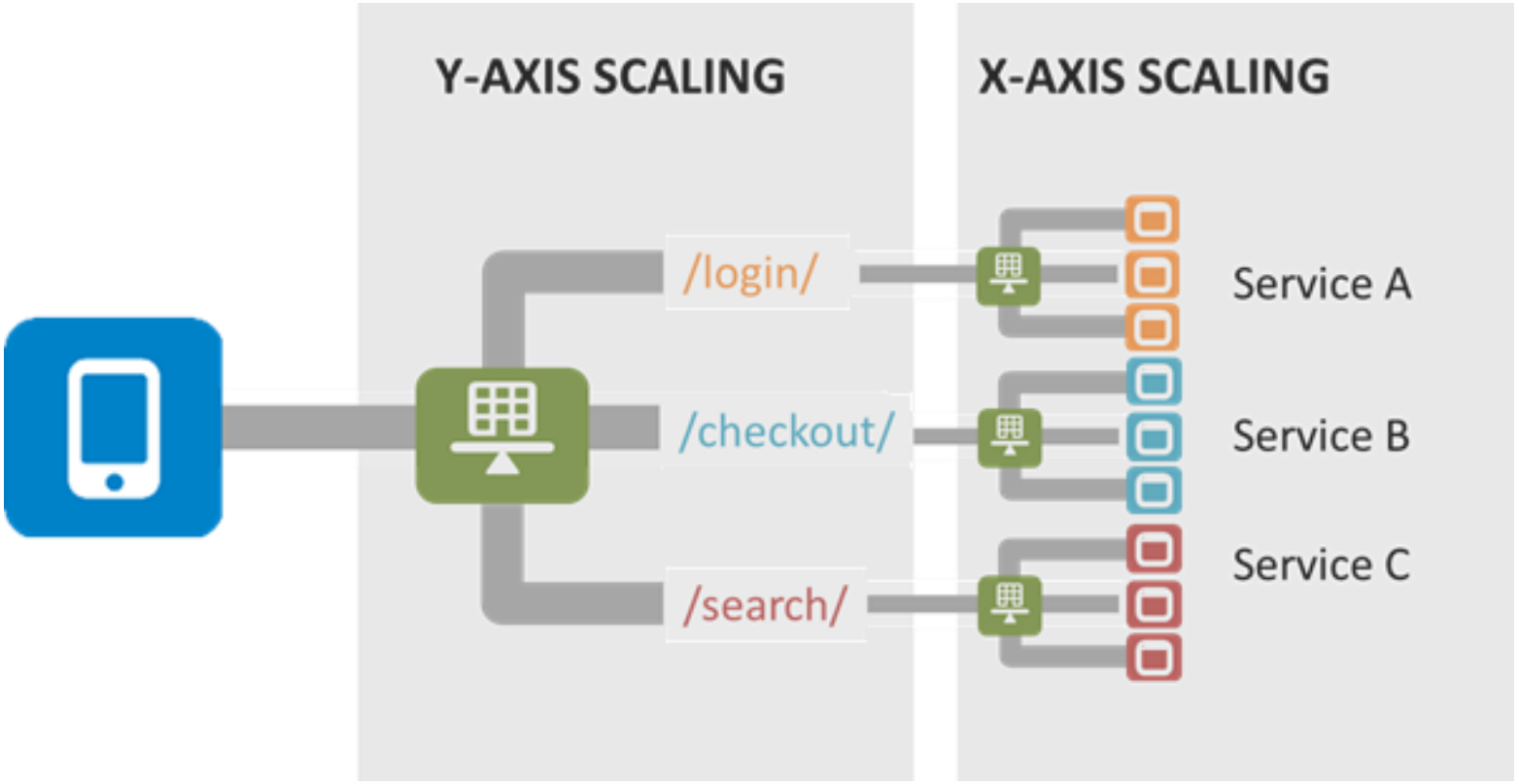
Network name: Horizontal scaling, scale out

Y-AXIS SCALING

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

Scaling Challenge

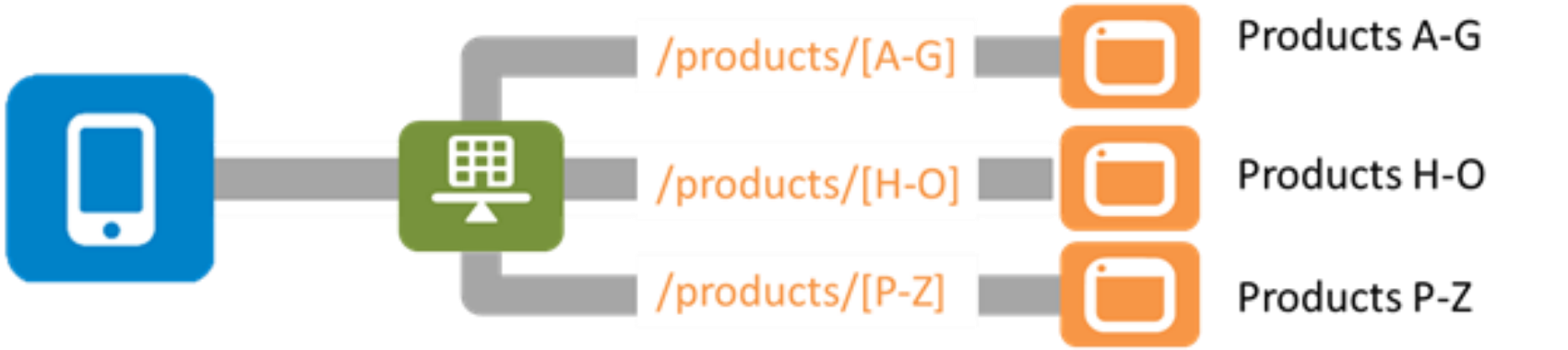
Microservices



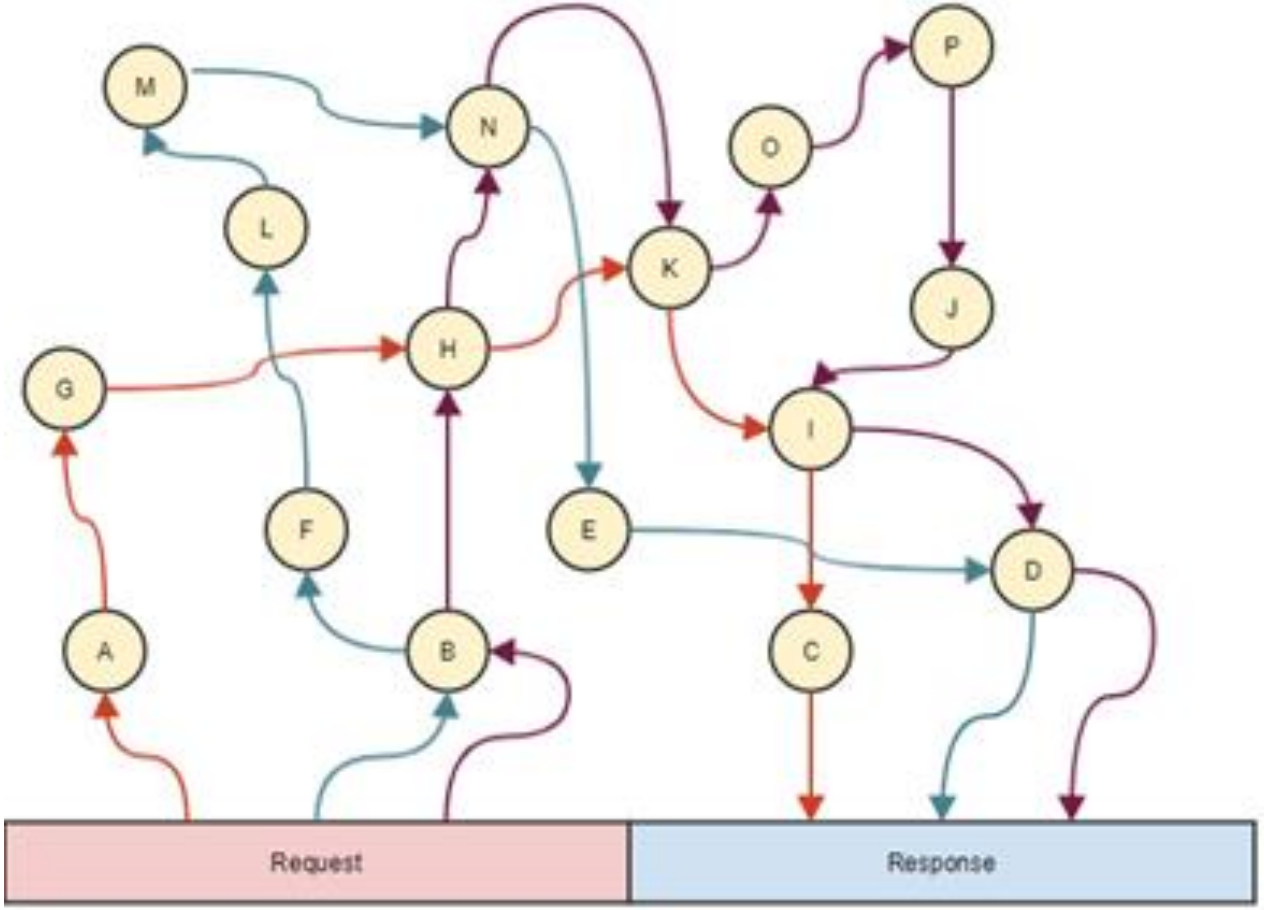
Scaling Challenge

Z-AXIS SCALING

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

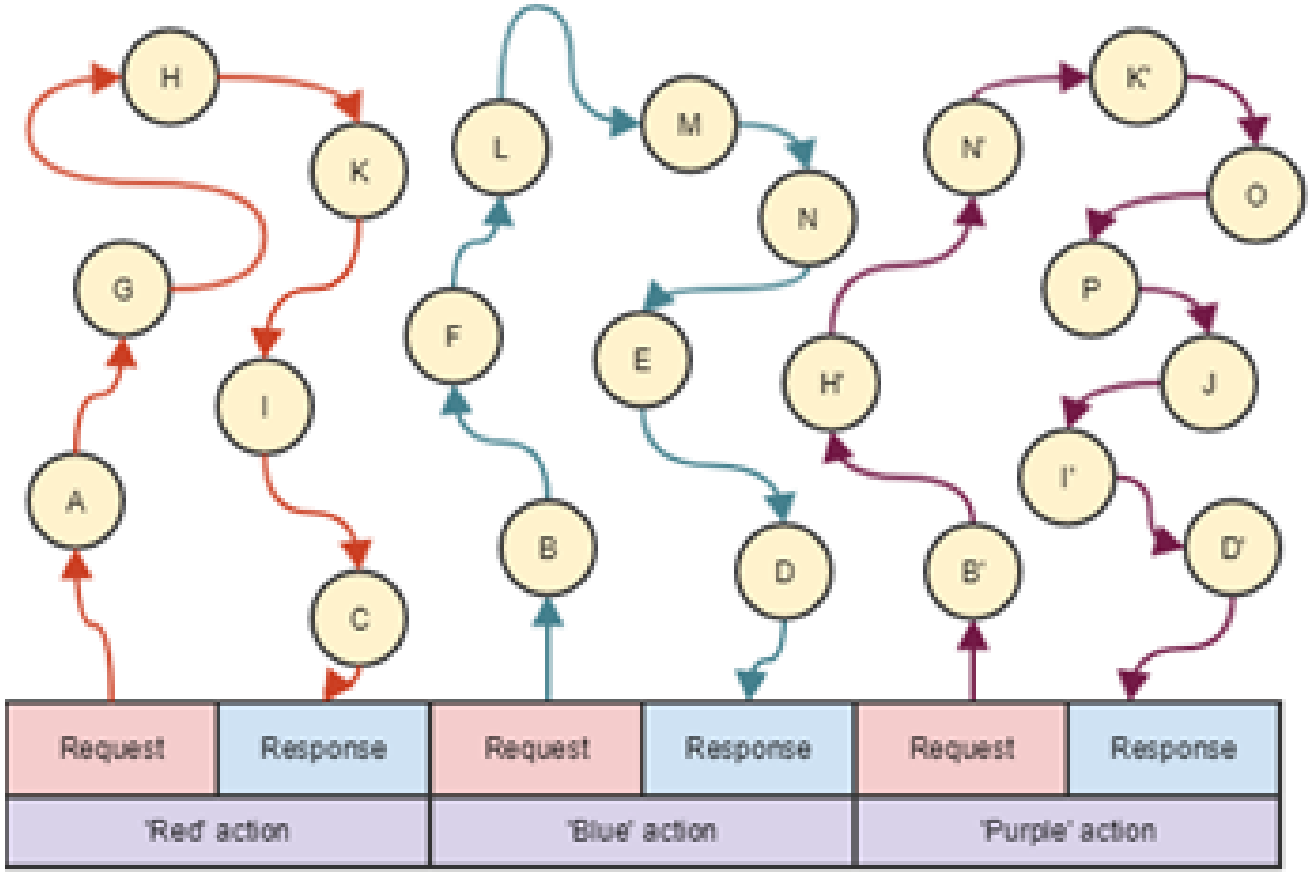


Saga Pattern



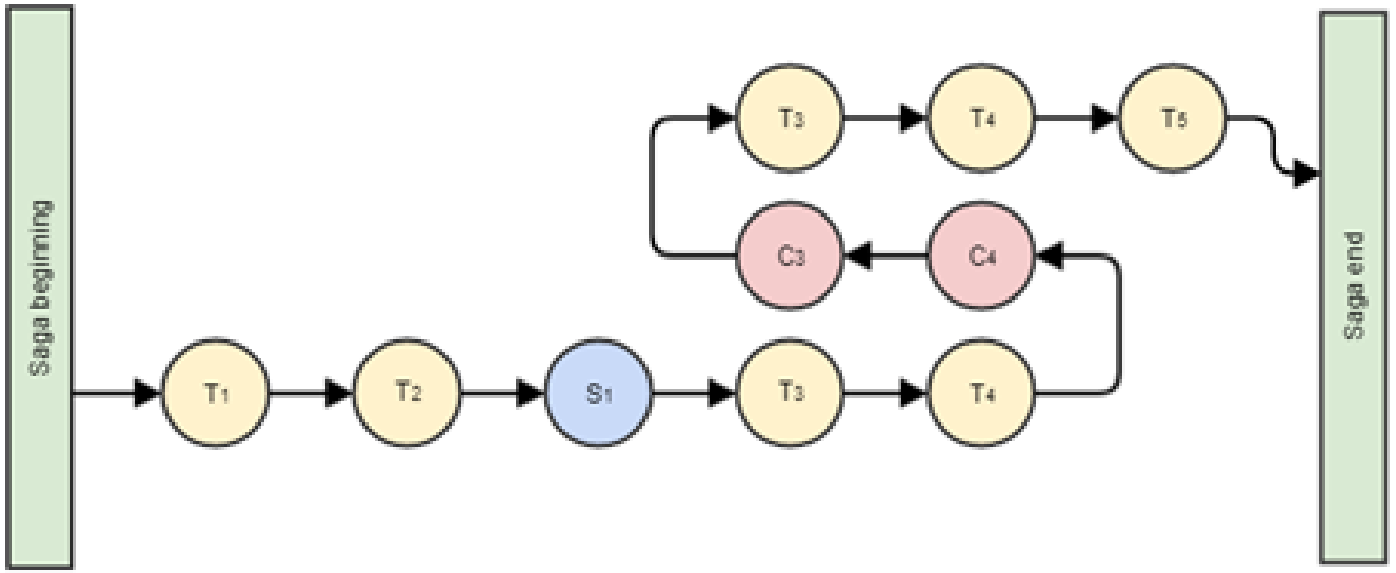
Saga Pattern

Microservices



Saga Pattern

Microservices



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.

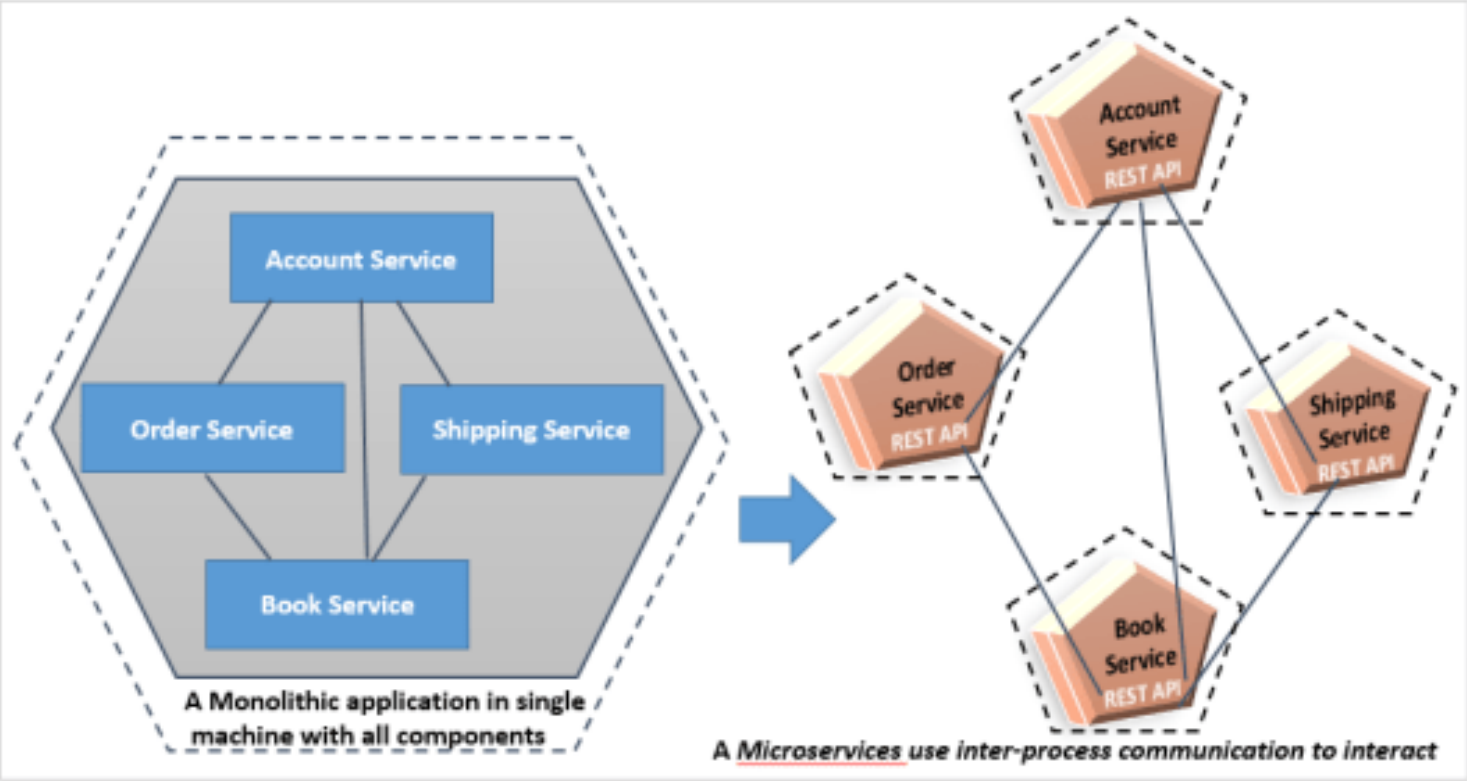
Proposed Solutions to Handle Challenges

- **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/OpensShift, 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

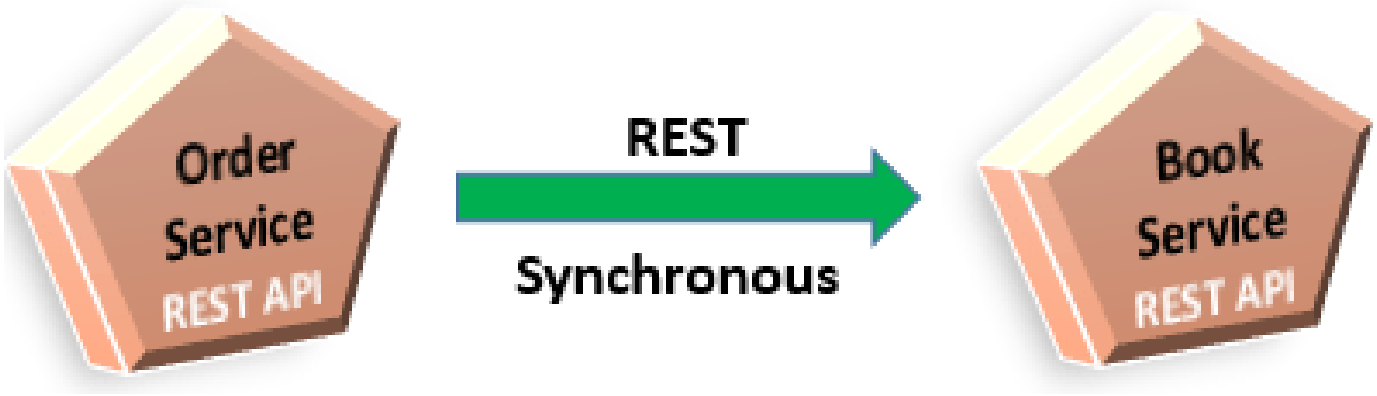


Microservices Inter-Service Communication

- 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

Microservices



- 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

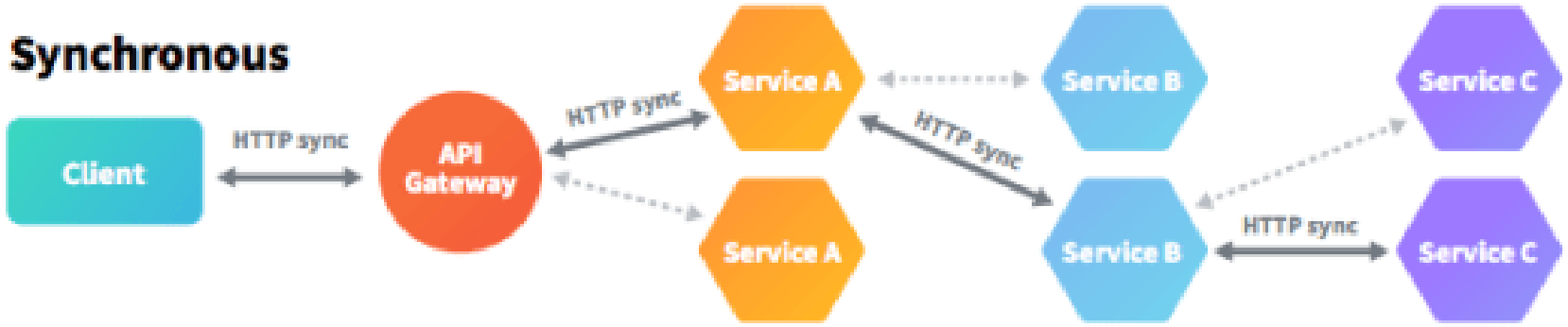
- The synchronous communication approach does have some drawbacks, such as timeouts and strong coupling.
- There are numerous protocols, such as REST, gRPC, and Apache Thrift, that can be used to interact with services synchronously.

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.

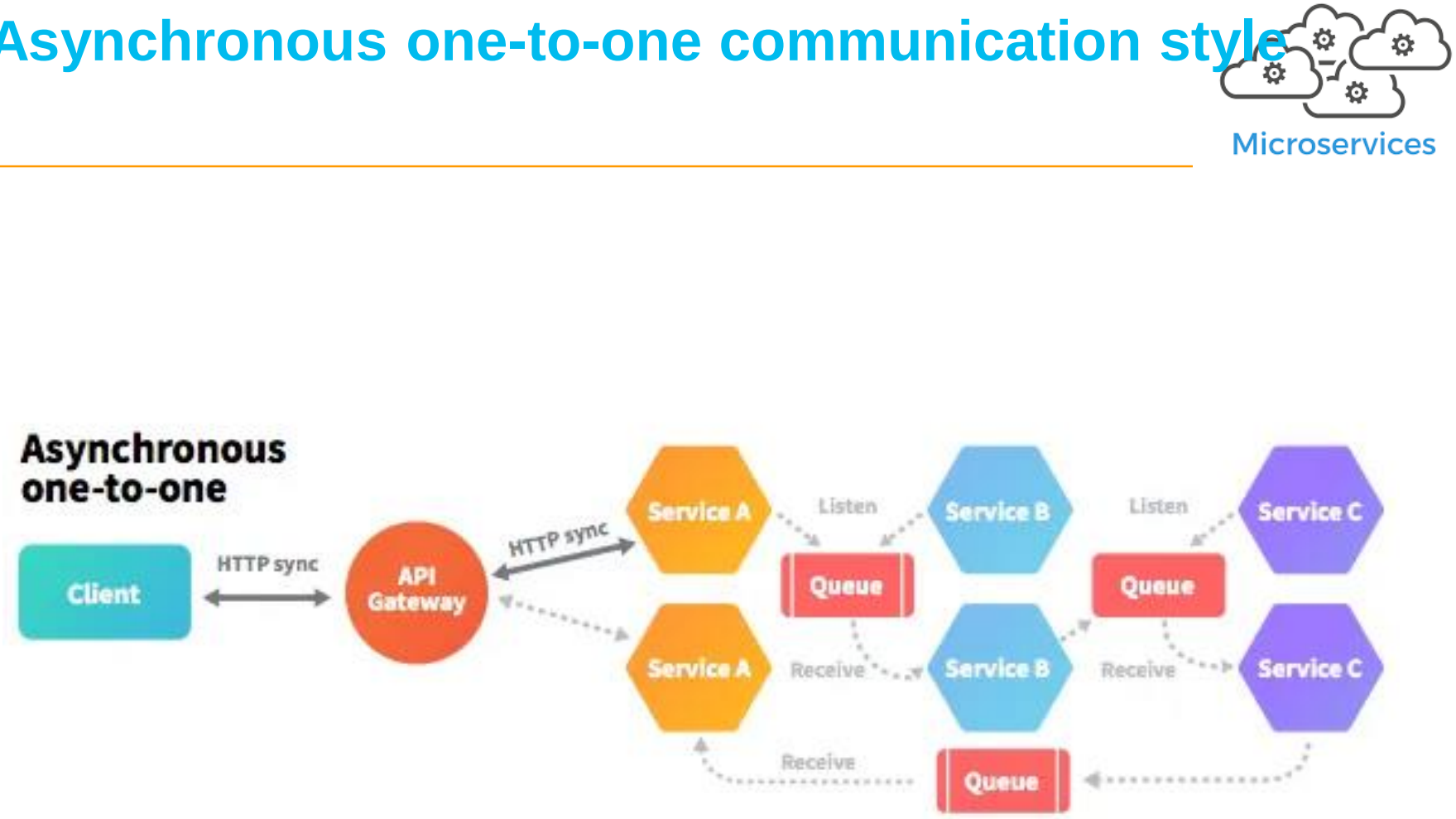
Asynchronous communication style

- 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.

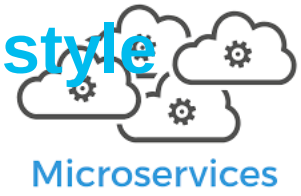
Asynchronous communication style

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

Asynchronous one-to-one communication style



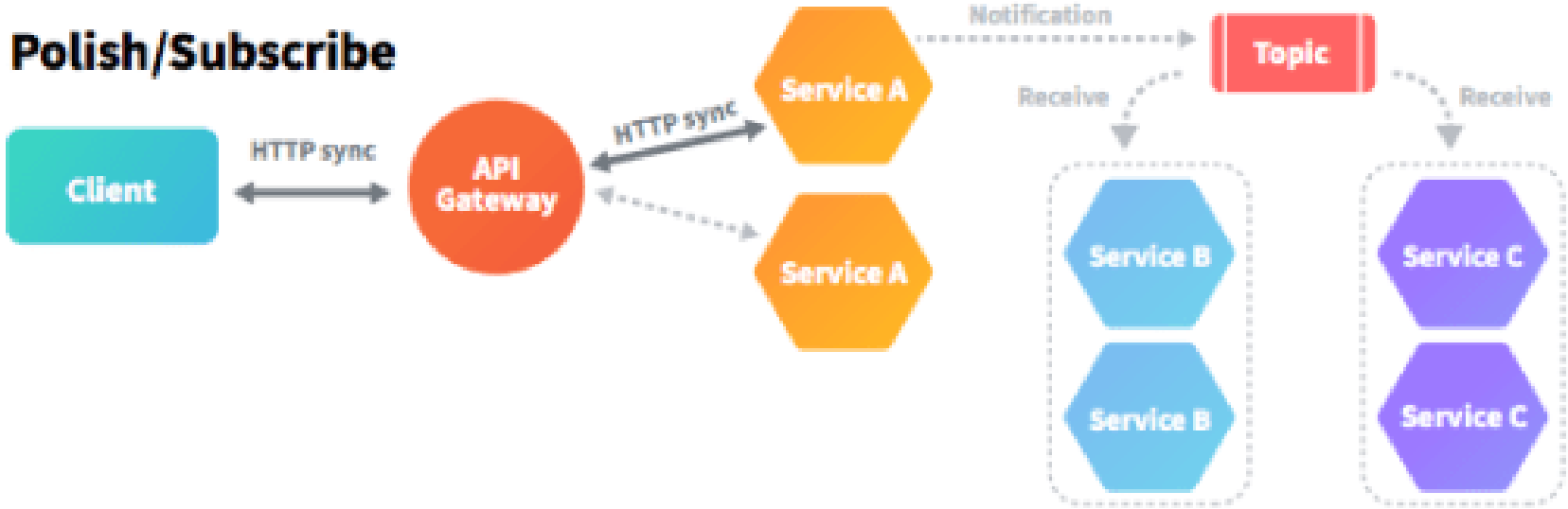
Asynchronous one-to-many communication style



- 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

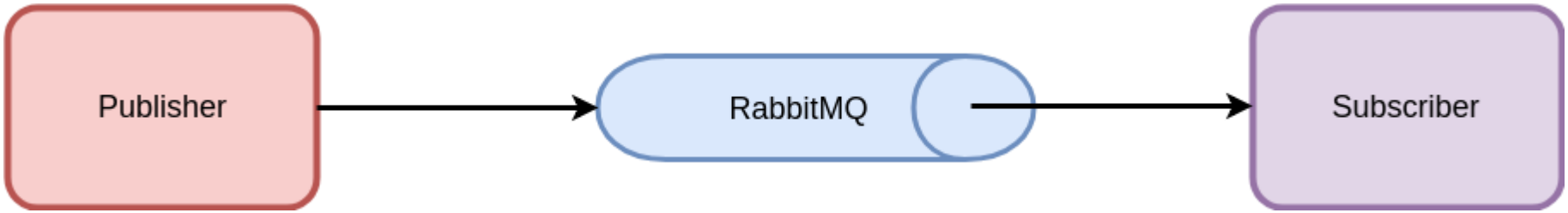
Asynchronous one-to-many communication style

Microservices



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

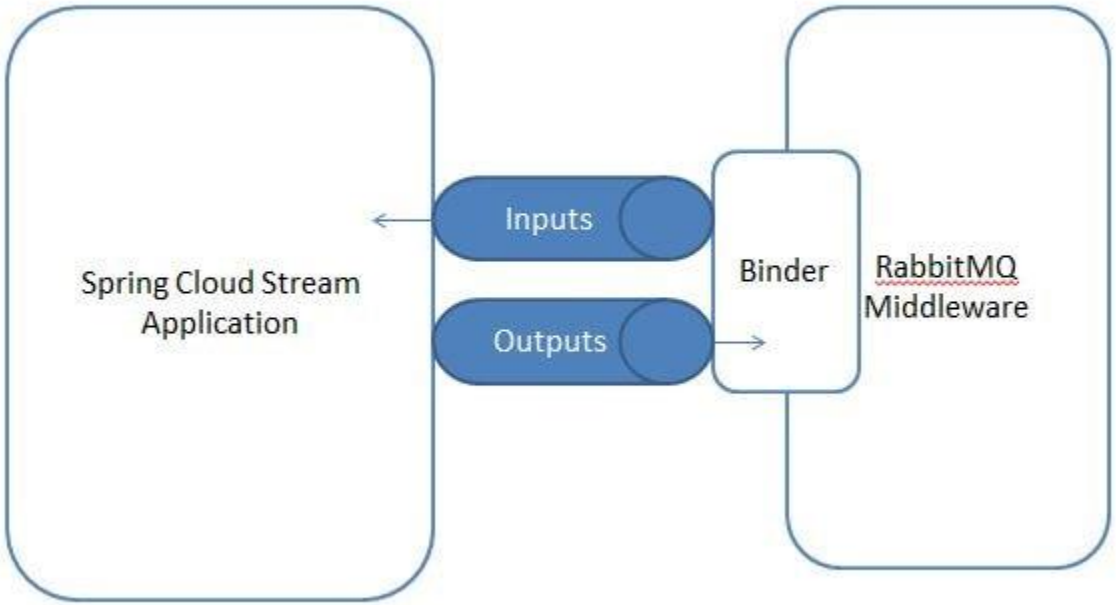
- Spring Cloud Stream supports a variety of binder implementations
  - RabbitMQ
  - Apache Kafka
  - Kafka Streams
  - Amazon Kinesis
  - Google PubSub *(partner maintained)*
  - Solace PubSub+ *(partner maintained)*
  - Azure Event Hubs *(partner maintained)*
  - Apache RocketMQ *(partner maintained)*

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).

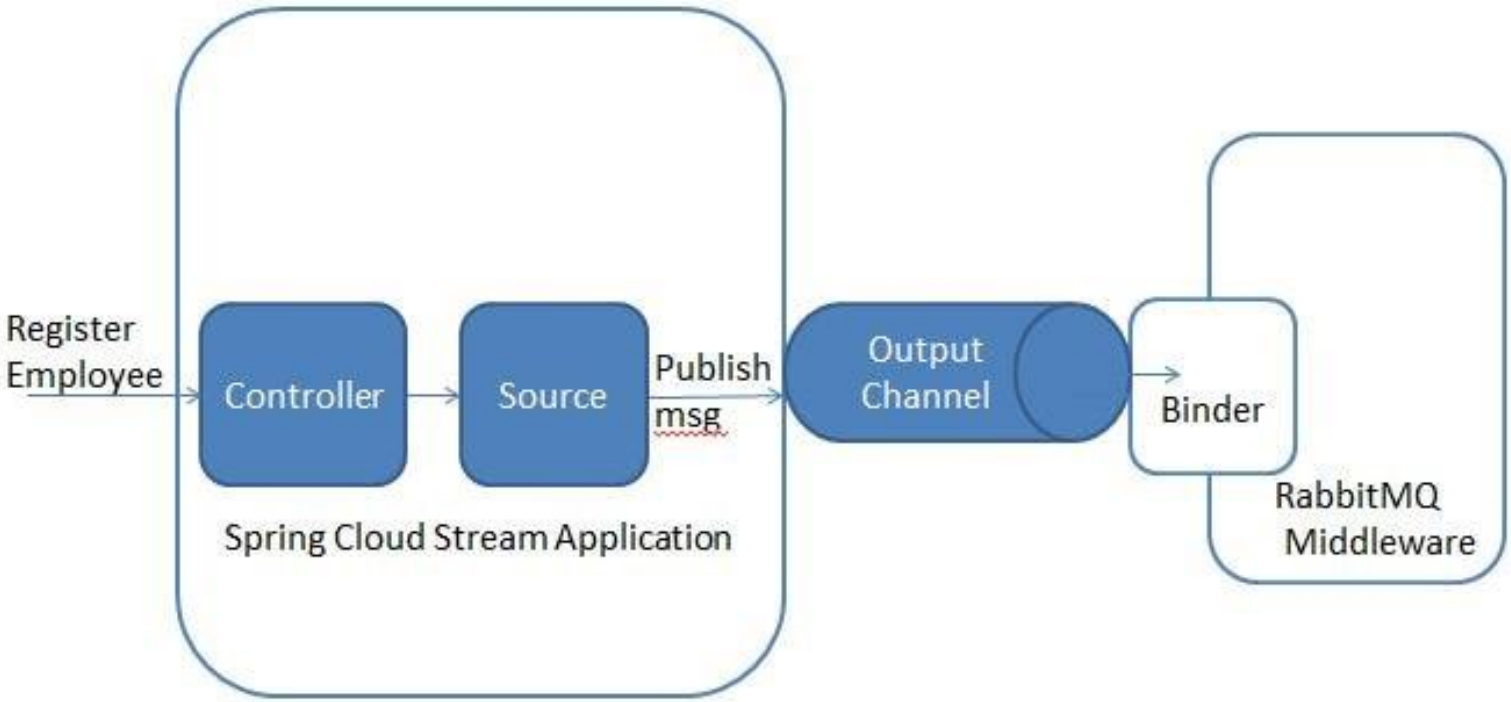
Core Components of Cloud Stream

Microservices



Core Components of Cloud Stream

Microservices



Asynchronous communication in Microservices

- Kafka
- RabbitMQ
- Google Pub/Sub
- Amazon Services
- ActiveMQ
- Azure Services

What is ZooKeeper

- ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.
- All of these kinds of services are used in some form or another by distributed applications.
- Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable.
- Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them, which make them brittle in the presence of change and difficult to manage.
- Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.

What is ZooKeeper?

- Apache ZooKeeper plays the very important role in system architecture as it works in the shadow of more exposed Big Data tools, as Apache Spark or Apache Kafka.
- Originally, the ZooKeeper framework was built at “Yahoo!”. Because it helps to access their applications in an easy manner.

Why Zookeeper

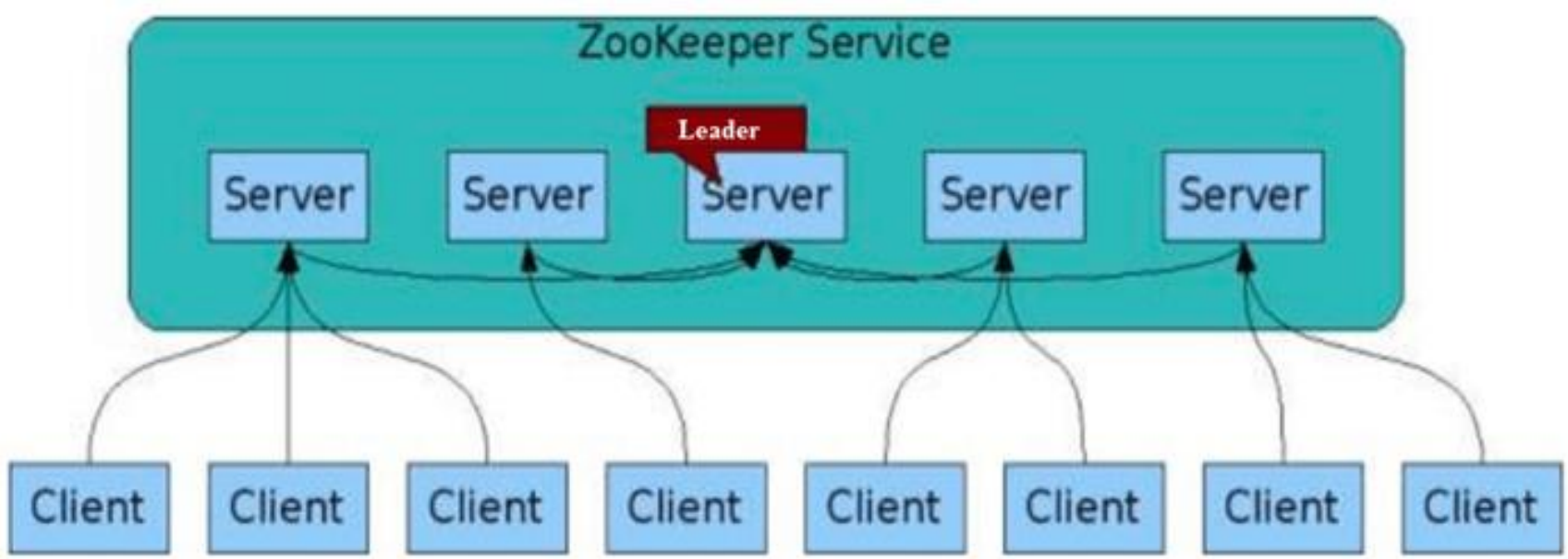
- It allows for mutual exclusion and cooperation between server processes.
- It ensures that your application runs consistently.
- The transaction process is never completed partially. It is either given the status of Success or failure. The distributed state can be held up, but it's never wrong.
- Irrespective of the server that it connects to, a client will be able to see the same view of the service
- Helps you to encode the data as per the specific set of rules

Why Zookeeper

- It helps to maintain a standard hierarchical namespace similar to files and directories.
- Computers, which run as a single system which can be locally or geographically connected.
- It allows to Join/leave node in a cluster and node status at the real time
- You can increase performance by deploying more machines.
- It allows you to elect a node as a leader for better coordination.
- ZooKeeper works fast with workloads where reads to the data are more common than writes.

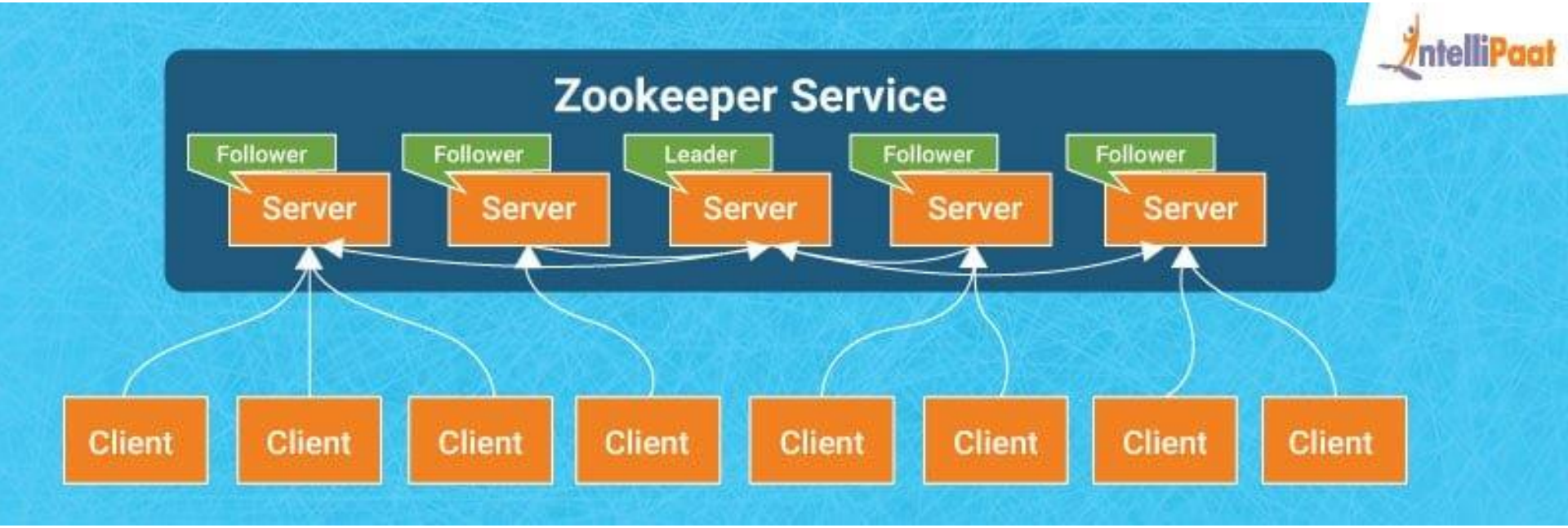
ZooKeeper Architecture: How it works?

Microservices



ZooKeeper Architecture: How it works?

Microservices



ZooKeeper Architecture

- Server: The server sends an acknowledge when any client connects. In the case when there is no response from the connected server, the client automatically redirects the message to another server.

- Client: Client is one of the nodes in the distributed application cluster. It helps you to accesses information from the server. Every client sends a message to the server at regular intervals that helps the server to know that the client is alive.

ZooKeeper Architecture

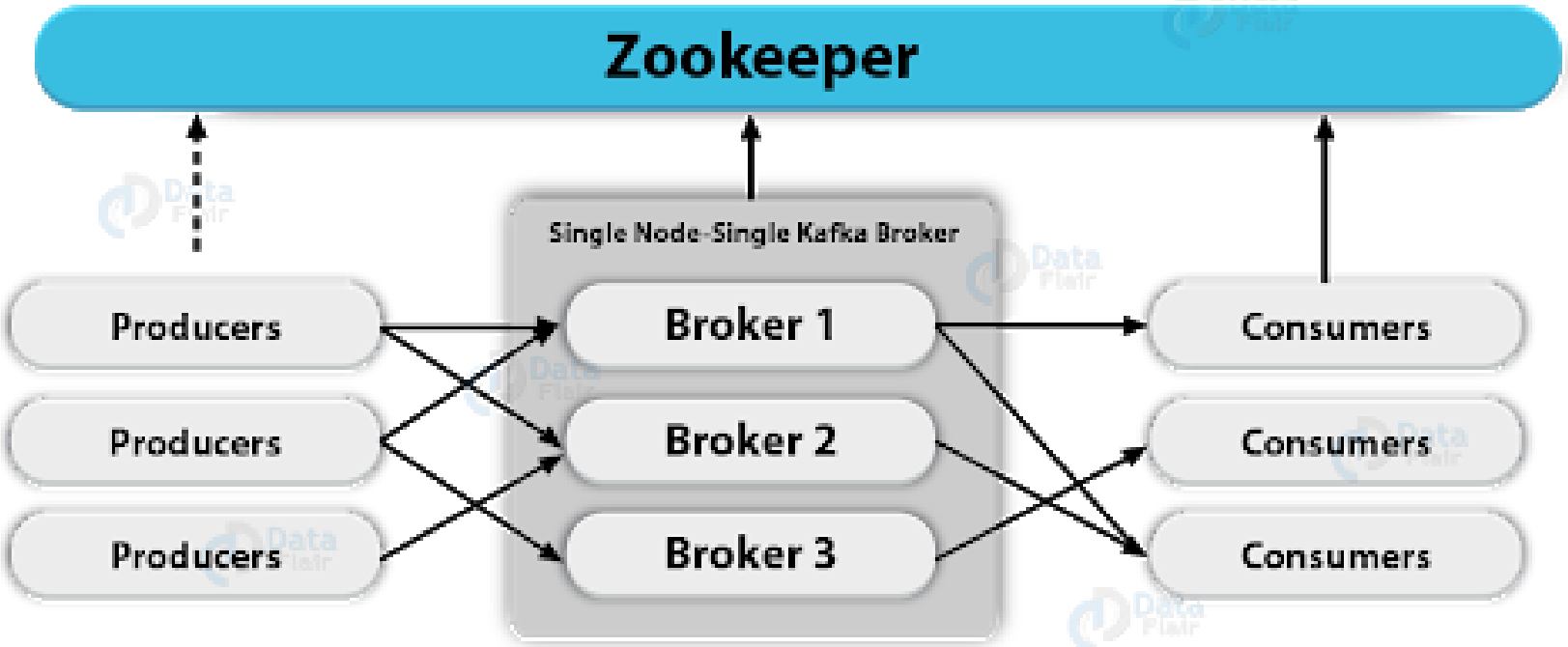
- Leader: One of the servers is designated a Leader. It gives all the information to the clients as well as an acknowledgment that the server is alive. It would performs automatic recovery if any of the connected nodes failed.
- Follower: Server node which follows leader instruction is called a follower.
- Client read requests are handled by the correspondingly connected Zookeeper server
- The client writes requests are handled by the Zookeeper leader.

ZooKeeper Architecture

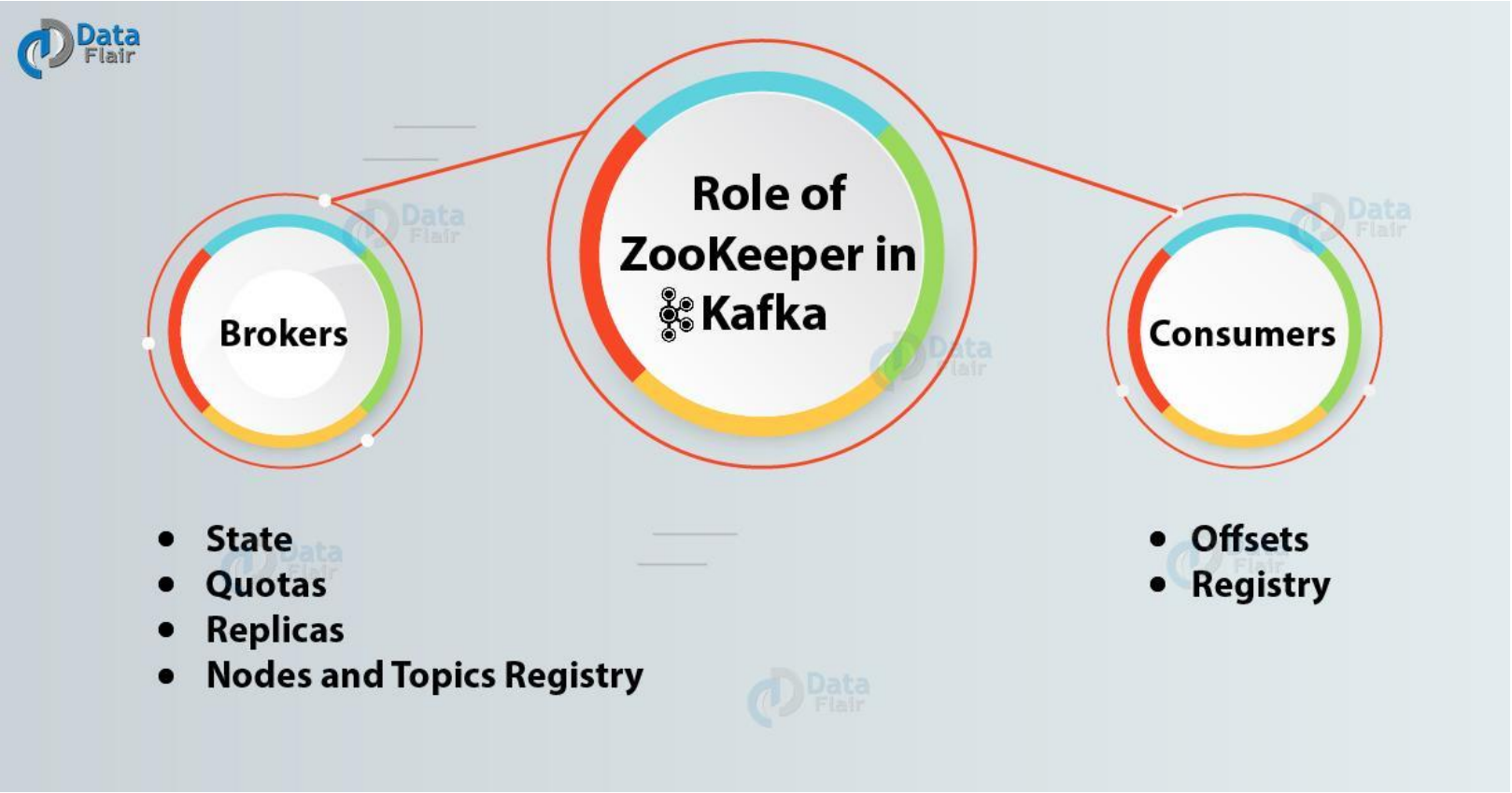
- Ensemble/Cluster: Group of Zookeeper servers which is called ensemble or a Cluster. You can use ZooKeeper infrastructure in the cluster mode to have the system at the optimal value when you are running the Apache.
- ZooKeeper WebUI: If you want to work with ZooKeeper resource management, then you need to use WebUI. It allows working with ZooKeeper using the web user interface, instead of using the command line. It offers fast and effective communication with the ZooKeeper application.

What is ZooKeeper?

Microservices



ZooKeeper in Kafka



Companies using Zookeeper

- Yahoo
- Facebook
- eBay
- Twitter
- Netflix
- Zynga
- Nutanix

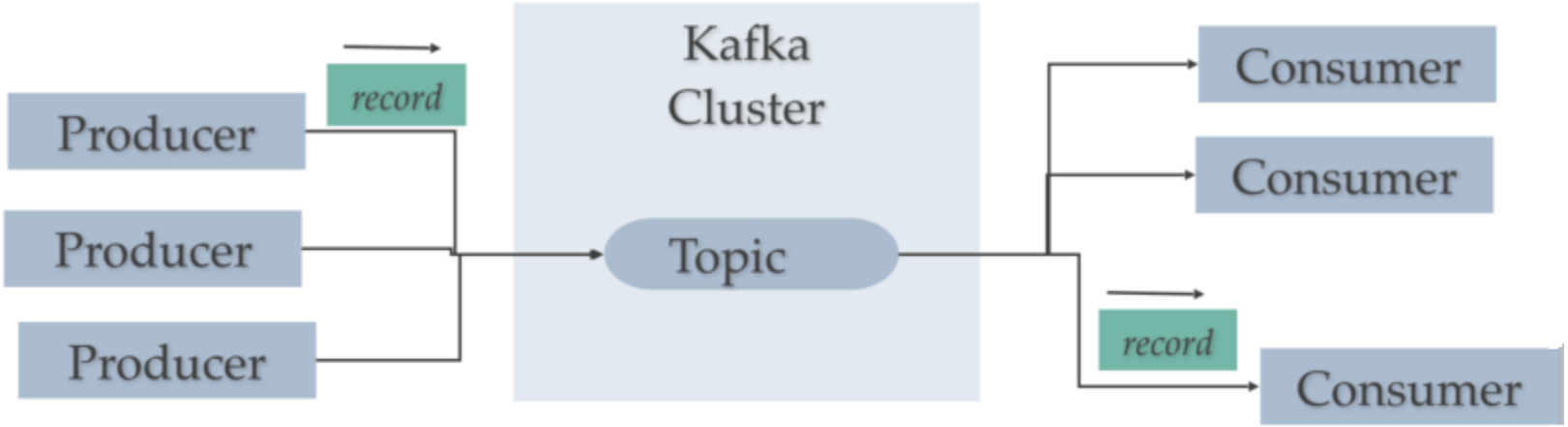
What is Kafka

- Kafka consists of Records, Topics, Consumers, Producers, Brokers, Logs, Partitions, and Clusters. Records can have key (optional), value and timestamp.
- Kafka Records are immutable.
- A Kafka Topic is a stream of records ("/orders", "/user-signups").
- You can think of a Topic as a feed name.
- A topic has a Log which is the topic’s storage on disk.
- A Topic Log is broken up into partitions and segments.

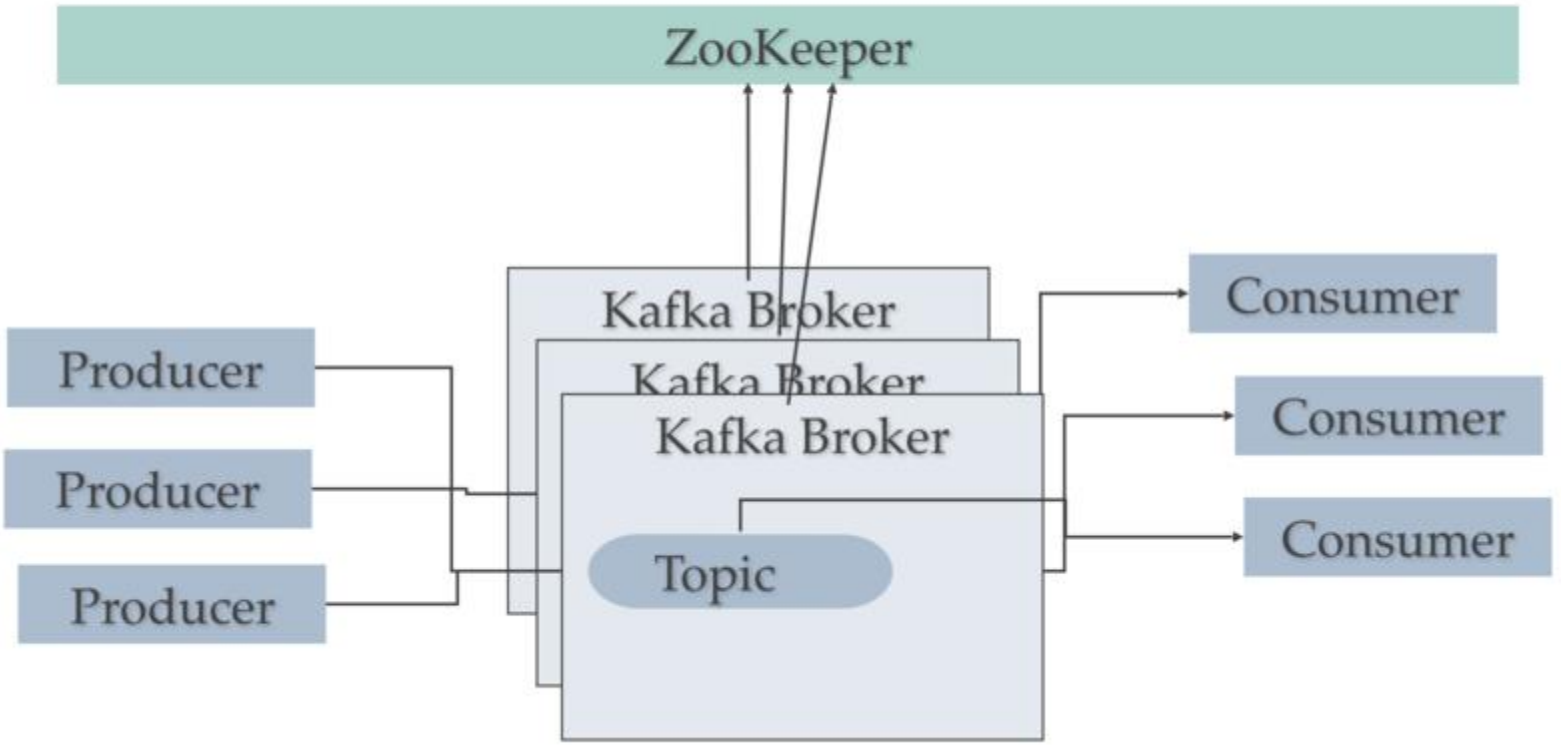
What is Kafka

- The Kafka Producer API is used to produce streams of data records.
- The Kafka Consumer API is used to consume a stream of records from Kafka.
- A Broker is a Kafka server that runs in a Kafka Cluster.
- Kafka Brokers form a cluster.
- The Kafka Cluster consists of many Kafka Brokers on many servers.
- Broker sometimes refer to more of a logical system or as Kafka as a whole.

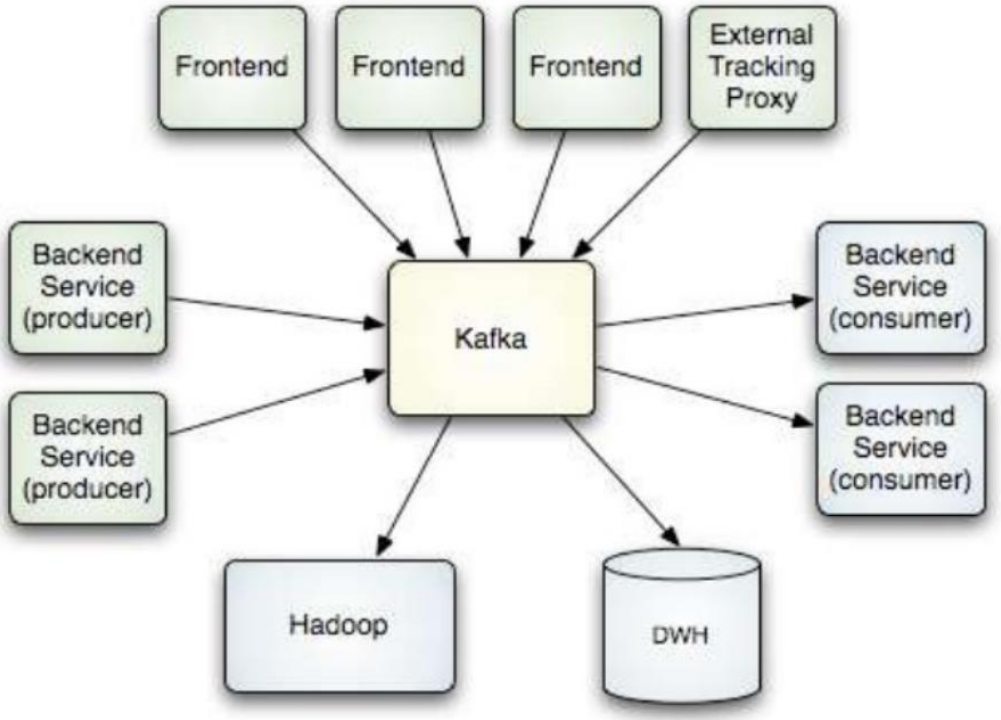
Kafka: Topics, Producers, and Consumers



ZooKeeper does coordination for Kafka Cluster

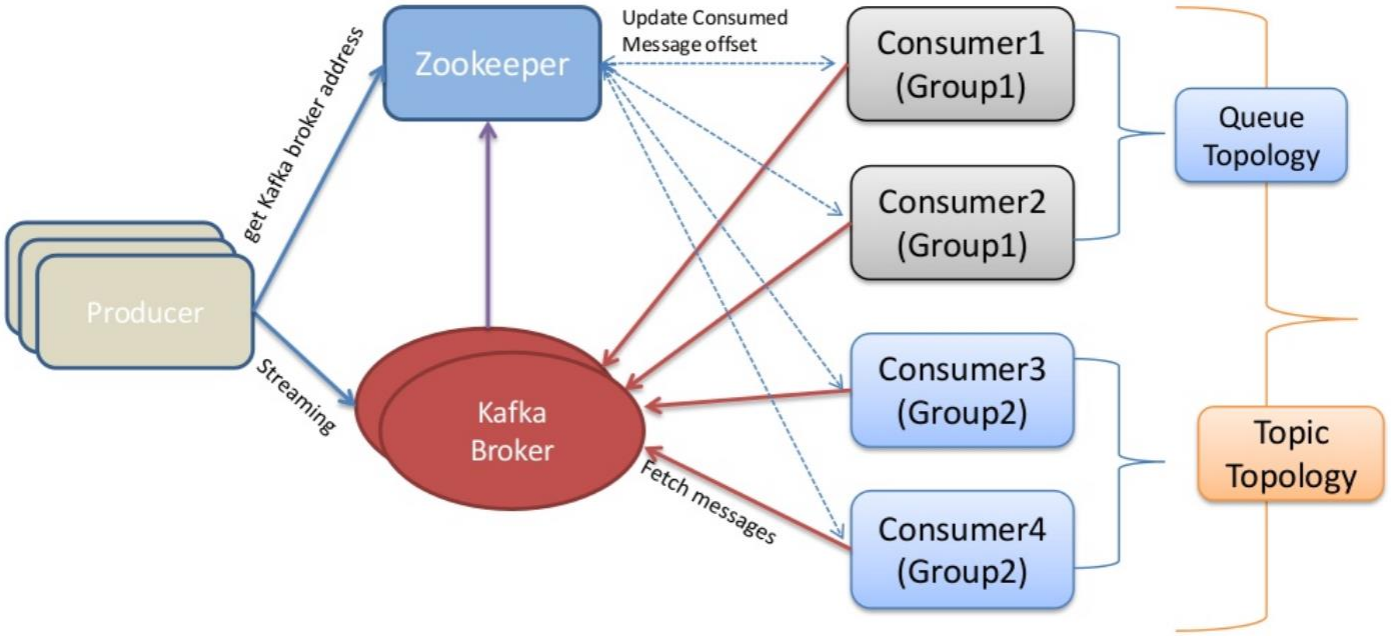


How it works



Credit : http://kafka.apache.org/design.html

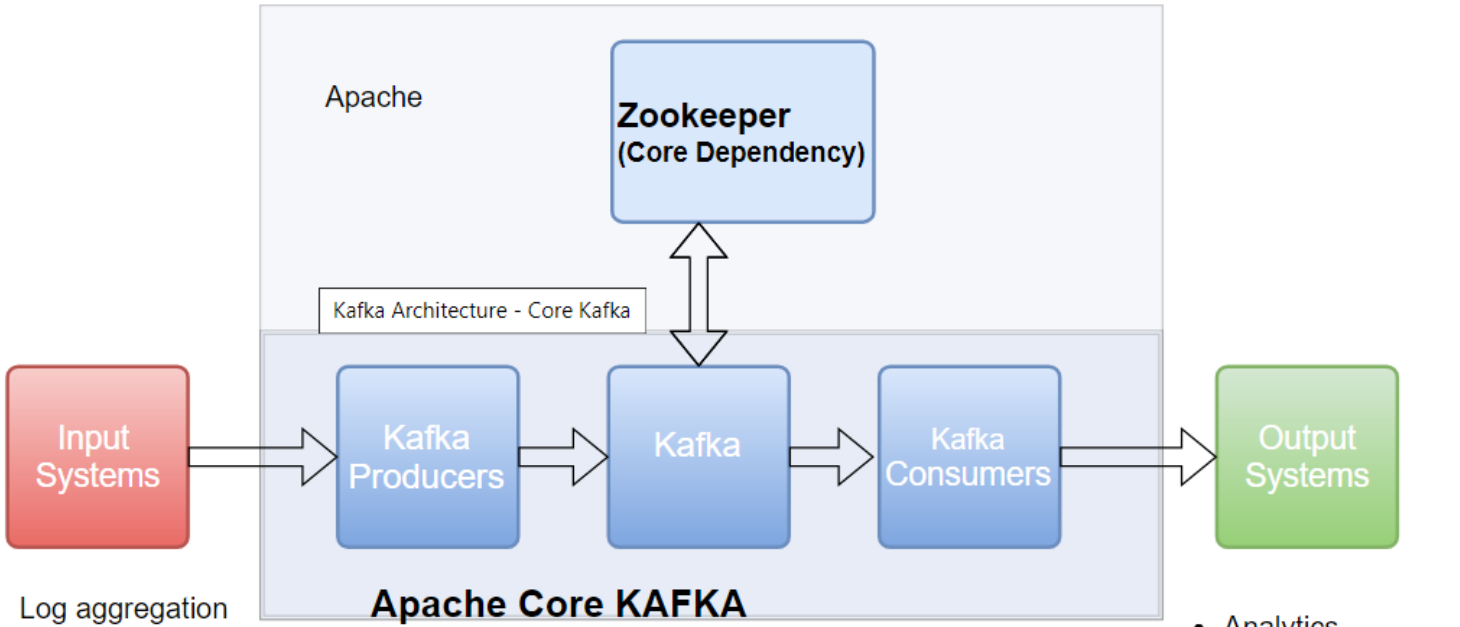
Real time transfer



ZooKeeper in Kafka

- Kafka needs ZooKeeper
- Kafka uses Zookeeper to do leadership election of Kafka Broker and Topic Partition pairs.
- Kafka uses Zookeeper to manage service discovery for Kafka Brokers that form the cluster.
- Zookeeper sends changes of the topology to Kafka, so each node in the cluster knows when a new broker joined, a Broker died, a topic was removed or a topic was added, etc.
- Zookeeper provides an in-sync view of Kafka Cluster configuration.

Kafka Architecture: Core Kafka



- Log aggregation
- Metrics
- KPIs
- Batch imports
- Audit trail
- User activity logs
- Web logs

Not part of core

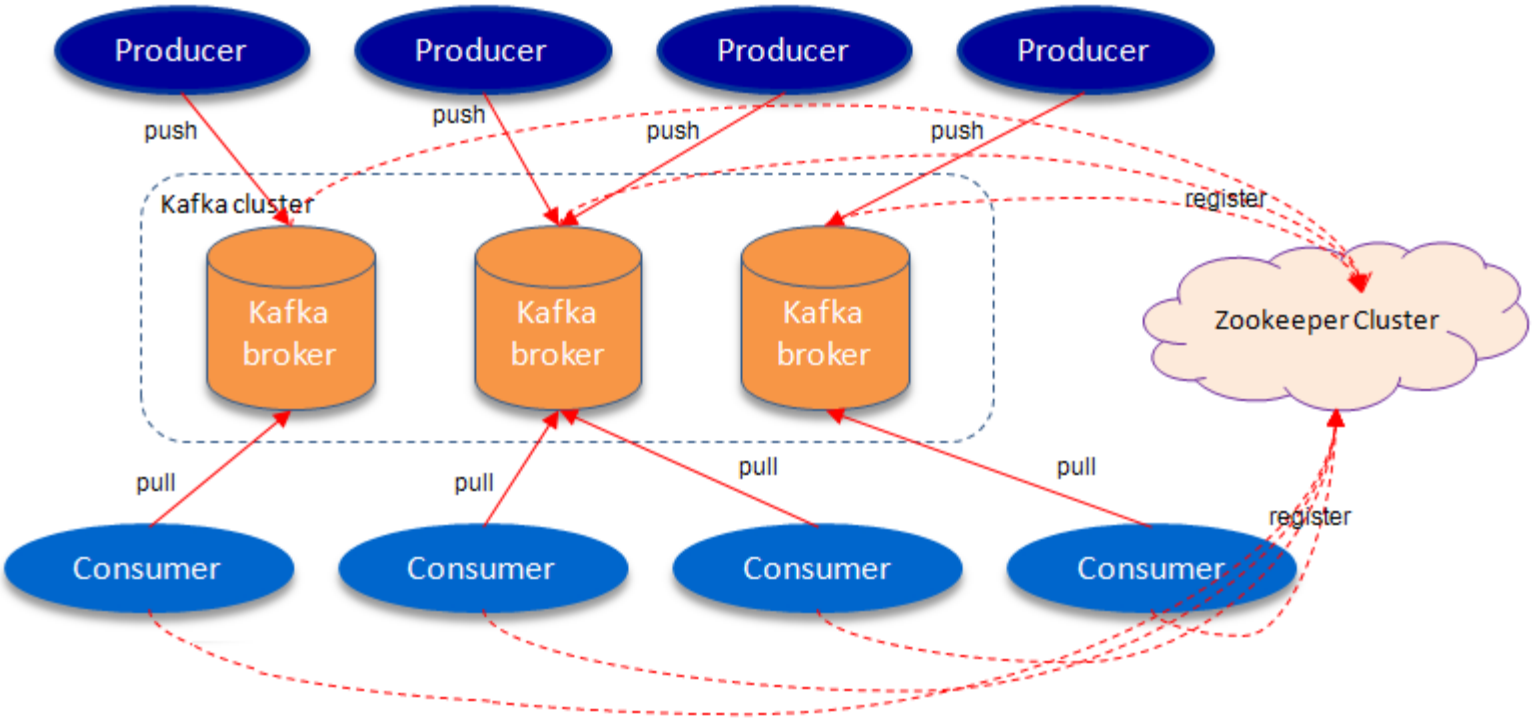
- Schema Registry
- Avro
- Kafka REST Proxy
- Kafka Connect
- Kafka Streams

Apache Kafka Core

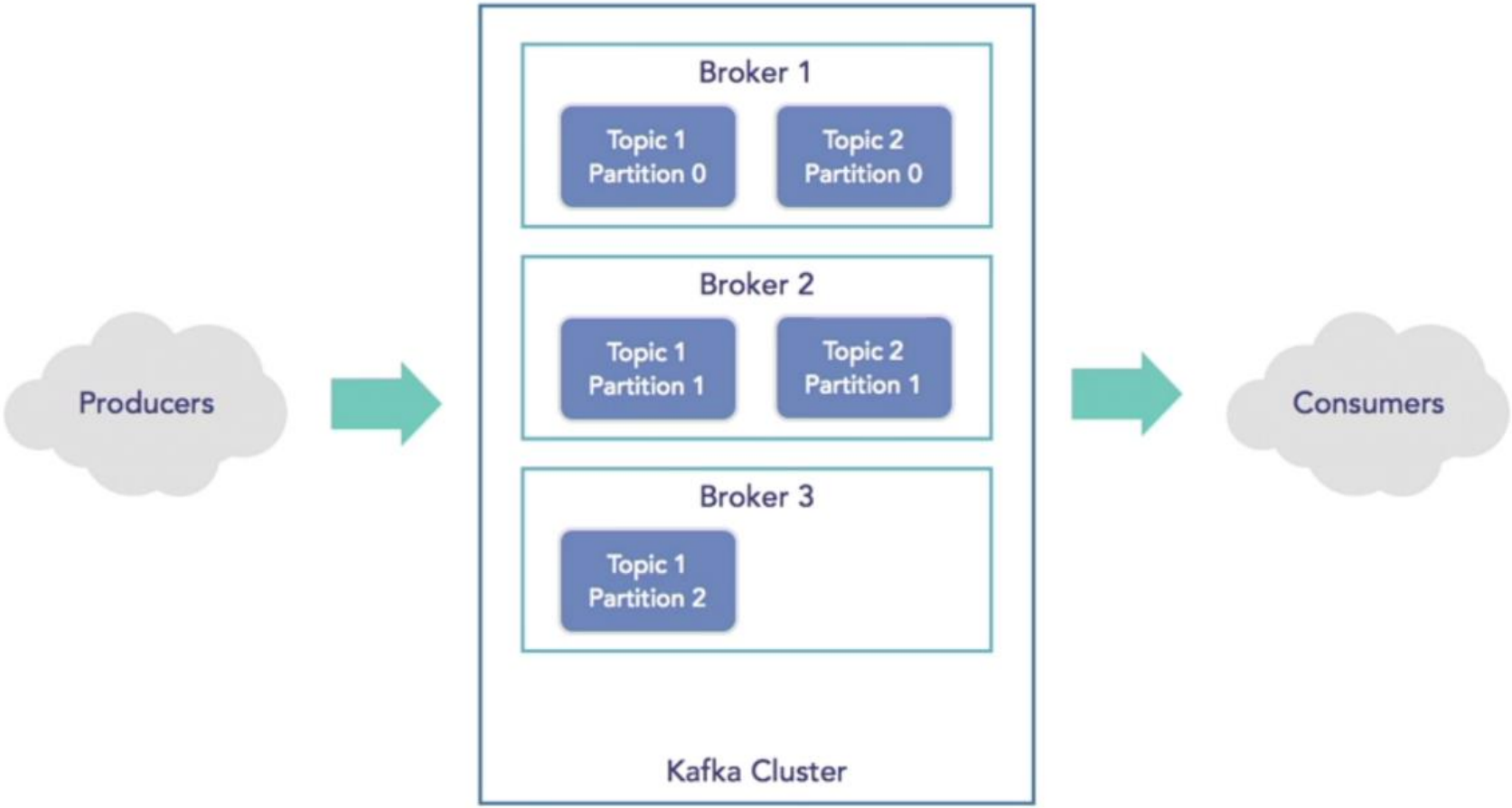
- Server/Broker
- Scripts to start libs
- Script to start up Zookeeper
- Utils to create topics
- Utils to monitor stats

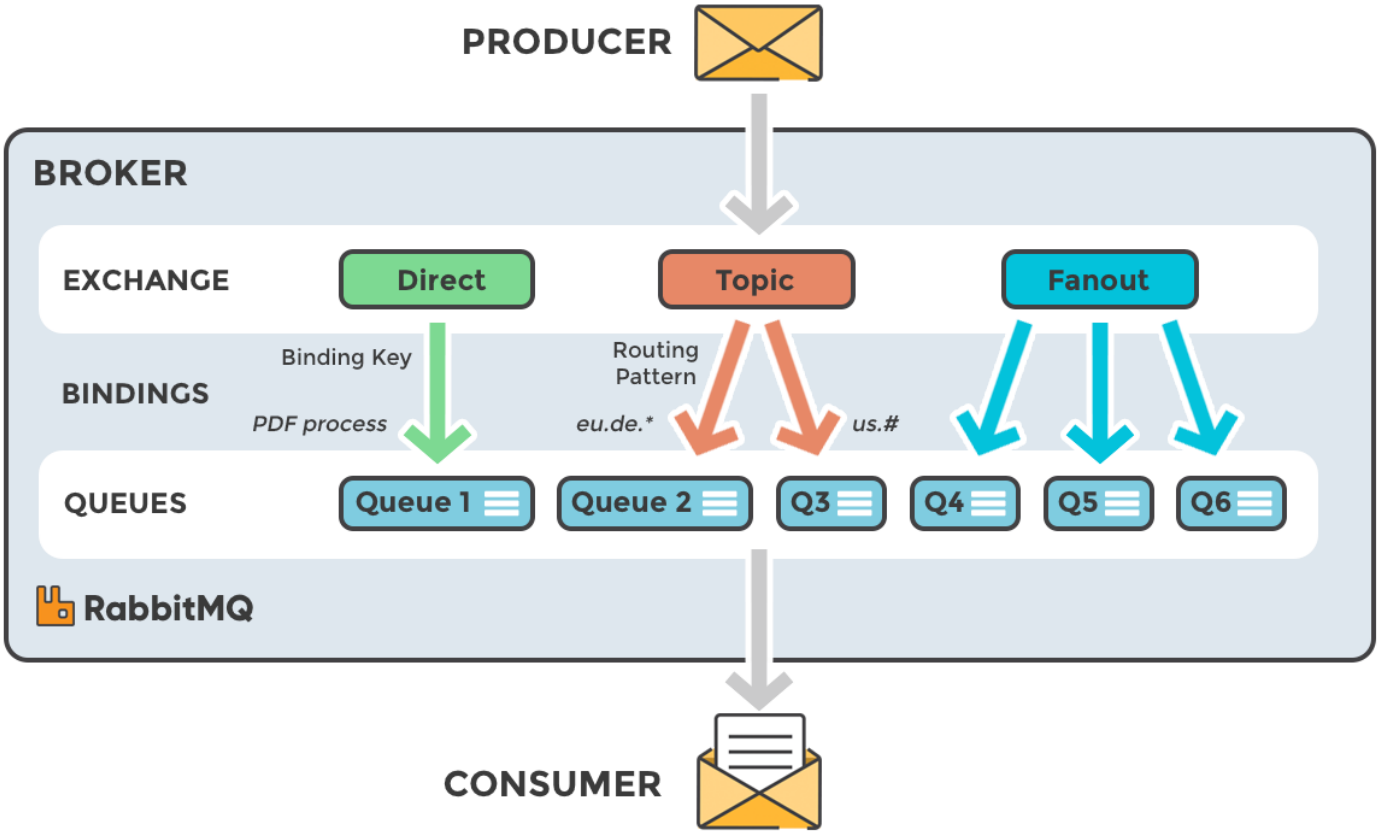
- Analytics
- Databases
- Machine Learning
- Dashboards
- Indexed for Search
- Business Intelligene

Kafka architecture

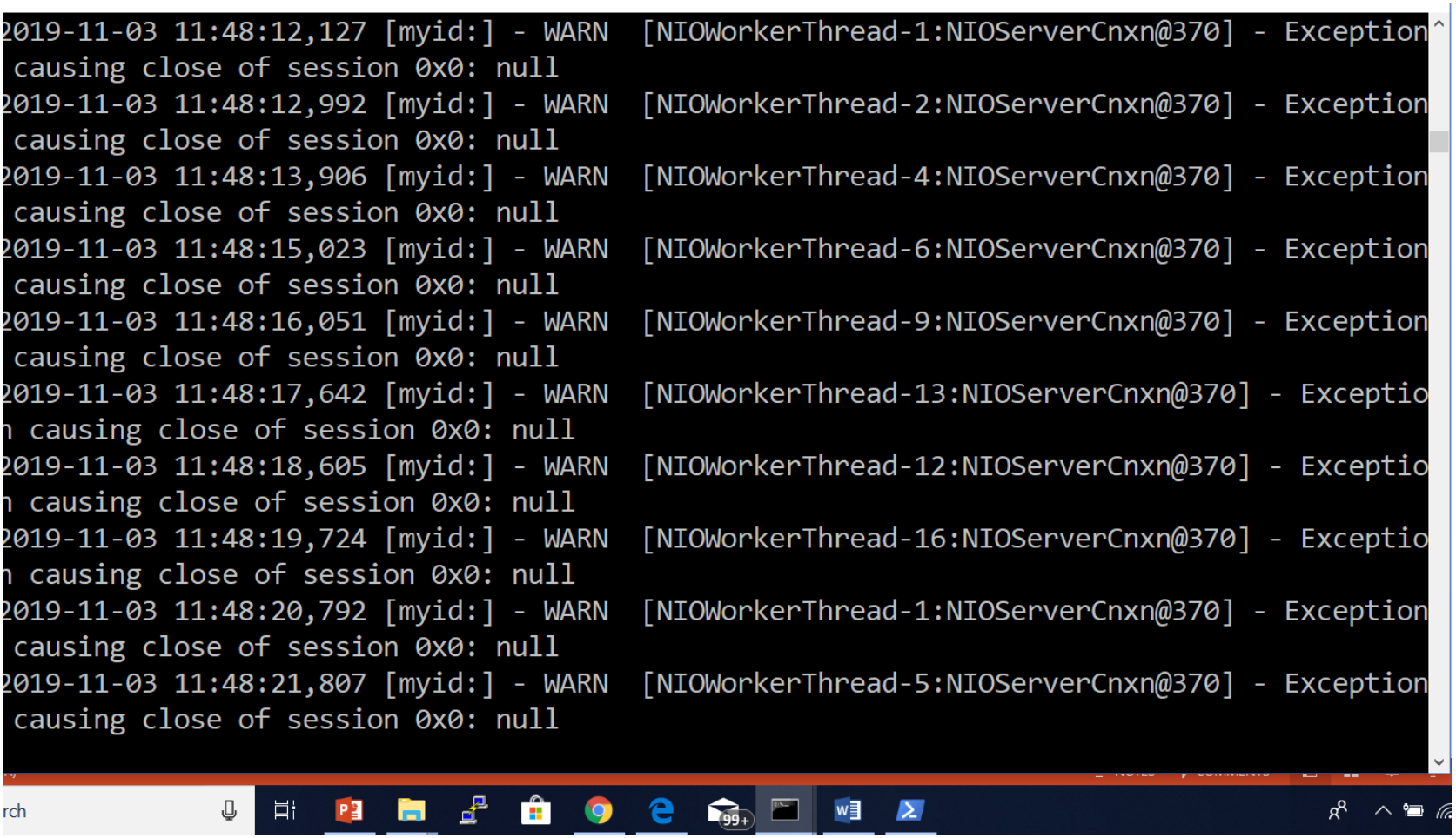


Kafka architecture





Zkserver from command prompt



C:\Windows\System32\cmd.exe - .\bin\windows\kafka-server-start.bat .\config\server.properties

```
tention.ms -> 86400000, cleanup.policy -> [delete], flush.ms -> 9223372036854775807, segm
ent.ms -> 604800000, segment.bytes -> 1073741824, retention.ms -> 604800000, message.time
stamp.difference.max.ms -> 9223372036854775807, segment.index.bytes -> 10485760, flush.me
ssages -> 9223372036854775807}. (kafka.log.LogManager)
[2019-11-03 11:43:35,072] INFO [Partition test11-0 broker=0] No checkpointed highwatermar
k is found for partition test11-0 (kafka.cluster.Partition)
[2019-11-03 11:43:35,073] INFO Replica loaded for partition test11-0 with initial high wa
termark 0 (kafka.cluster.Replica)
[2019-11-03 11:43:35,074] INFO [Partition test11-0 broker=0] test11-0 starts at Leader Ep
och 0 from offset 0. Previous Leader Epoch was: -1 (kafka.cluster.Partition)
[2019-11-03 11:48:44,857] INFO [GroupCoordinator 0]: Preparing to rebalance group console
-consumer-8213 in state PreparingRebalance with old generation 0 (__consumer_offsets-33)
(reason: Adding new member consumer-1-217673c0-2a77-44ea-b5dc-5b486eefded2) (kafka.coordi
nator.group.GroupCoordinator)
[2019-11-03 11:48:44,868] INFO [GroupCoordinator 0]: Stabilized group console-consumer-82
13 generation 1 (__consumer_offsets-33) (kafka.coordinator.group.GroupCoordinator)
[2019-11-03 11:48:44,882] INFO [GroupCoordinator 0]: Assignment received from leader for
group console-consumer-8213 for generation 1 (kafka.coordinator.group.GroupCoordinator)
[2019-11-03 11:51:33,455] INFO [GroupMetadataManager brokerId=0] Removed 0 expired offset
s in 3 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
```

Administrator: RabbitMQ Command Prompt (sbin dir) - kafka-console-producer.bat --broker-list localhost:9092 --topic test

```
[2019-11-03 11:43:19,657] ERROR org.apache.kafka.common.errors.TopicExistsException: Topic 'test' already exists.
 (kafka.admin.TopicCommand$)

E:\software\A08\file\kafka\bin\windows>kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --p
artitions 1 --topic test11
Created topic test11.

E:\software\A08\file\kafka\bin\windows>kafka-console-producer.bat --broker-list localhost:9092 --topic test
>hi
>how are yoy
>enjoy
>
```

Administrator: RabbitMQ Command Prompt (sbin dir) - kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic test --from-beginning

```
                                    consumption.

E:\software\A08\file\kafka\bin\windows>kafka-console-consumer.bat --bootstrap-server localhost:2181 --topic test
Processed a total of 0 messages
Terminate batch job (Y/N)? y

E:\software\A08\file\kafka\bin\windows>kafka-console-consumer.bat --bootstrap-server localhost:2181 --topic test
Processed a total of 0 messages
Terminate batch job (Y/N)? y

E:\software\A08\file\kafka\bin\windows>kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic test --fro
m-beginning
hi
how are yoy
enjoy
```

RabbitMQ

- The super simplified overview:
- Publishers send messages to exchanges
- Exchanges route messages to queues and other exchanges
- RabbitMQ sends acknowledgements to publishers on message receipt
- Consumers maintain persistent TCP connections with RabbitMQ and declare which queue(s) they consume
- RabbitMQ pushes messages to consumers
- Consumers send acknowledgements of success/failure
- Messages are removed from queues once consumed successfully

Category	Kafka	RabbitMQ	Amazon SQS	Google Pub/Sub
Message Rate (msgs/sec)	100k+	20k+	Varies (can be scaled)	10k+
Message Acknowledgements	No	Yes	Yes	Yes
Built in UI	No	Yes	Yes	Yes
Protocol	Kafka	AMQP	HTTP REST	HTTP REST
Additional features	Apache Storm, etc	Several plugins available to add more features	In-flight messages	-
Use cases	High ingestion platforms where speed,scale and efficiency is the prime concern	Robust systems where features like acknowled-gements, deeper management are important	Useful when deploying applications on AWS EC2, Elastic Beanstalk to facilitate low-latency communication	Works well with applications deployed on GCE

What design patterns are available for inter-service communication for microservices?

- **Saga Pattern**
- **API Composition**: Since table joins are not possible across databases, a dedicated service (or API gateway) coordinates the "joins", which are now at application layer rather than within the database.
- **Command Query Responsibility Segregation (CQRS):** Services keep materialized views based on data from multiple services. These views are updated via subscribed events. CQRS separates writes (commands) from the reads (queries).

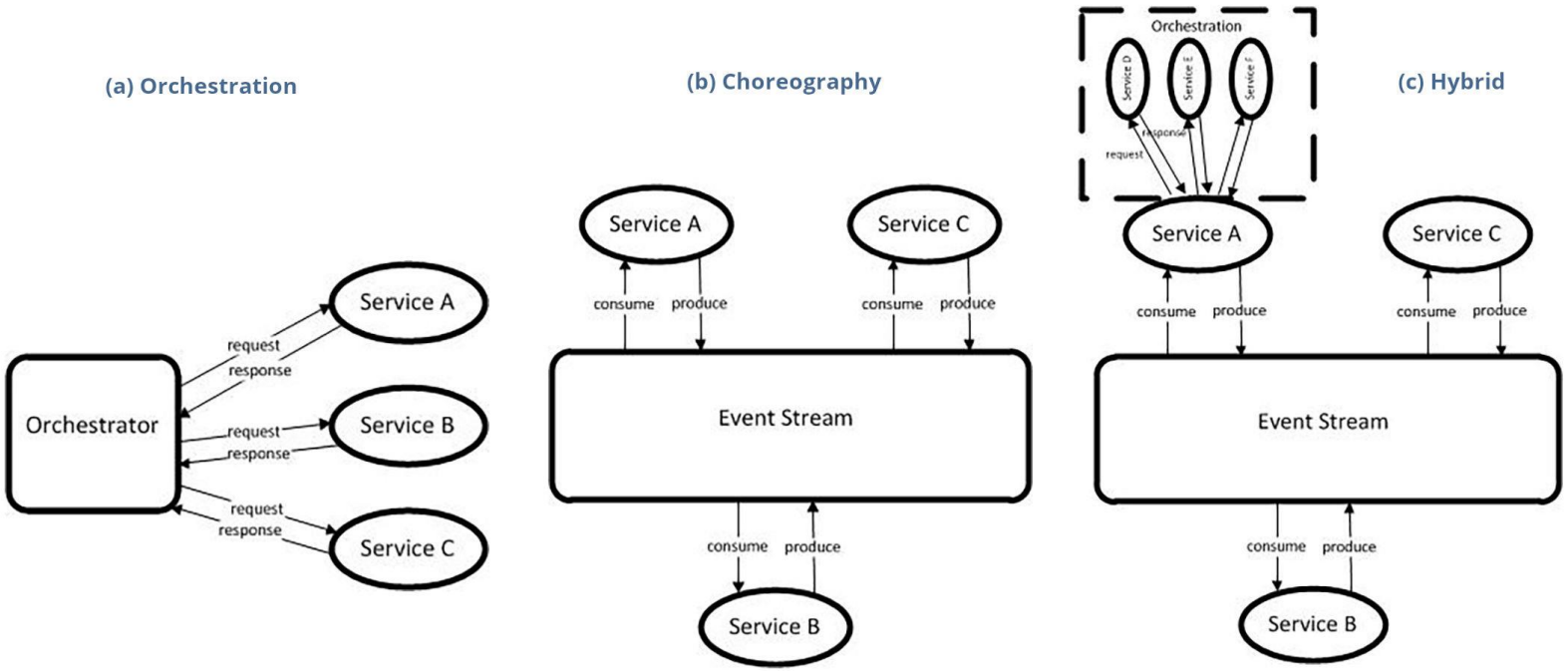
What design patterns are available for inter-service communication for microservices?

- **Event Sourcing:** Rather than store state, we store events. State may be computed from these events as desired. This is often used with CQRS: write events but derive states by replaying the events.
- **Orchestration**: A central controller or orchestrator coordinates interactions across microservices. API Composition is a form of orchestration.
- Choreography: Each microservice knows what to do when an event occurs, which are posted on an event stream/bus.

What design patterns are available for inter-service communication for microservices?

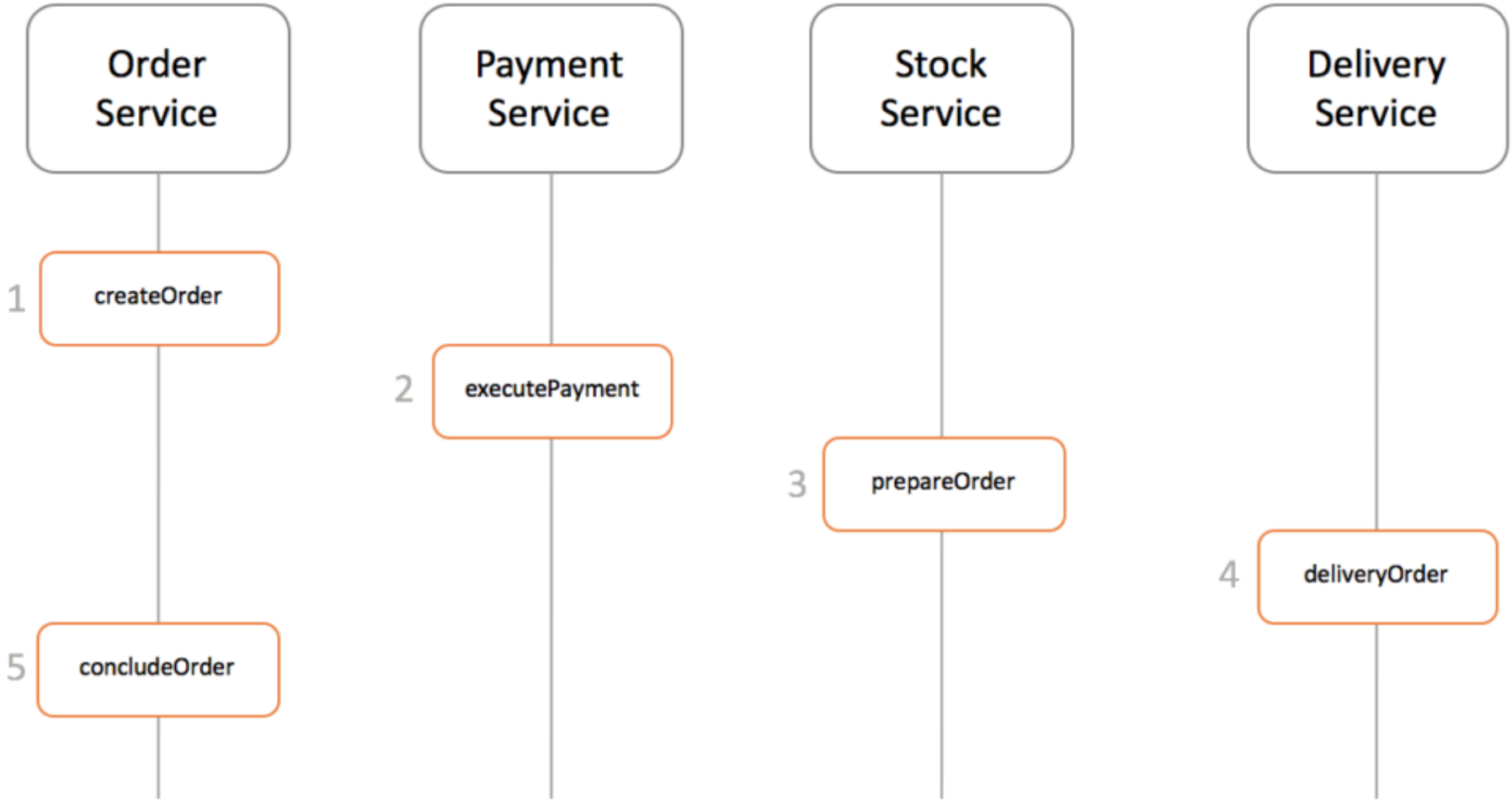
Microservices

- **Service Mesh**: Push application networking functions down to the infrastructure and not mix them with business logic**.**



Saga Design Pattern

Microservices

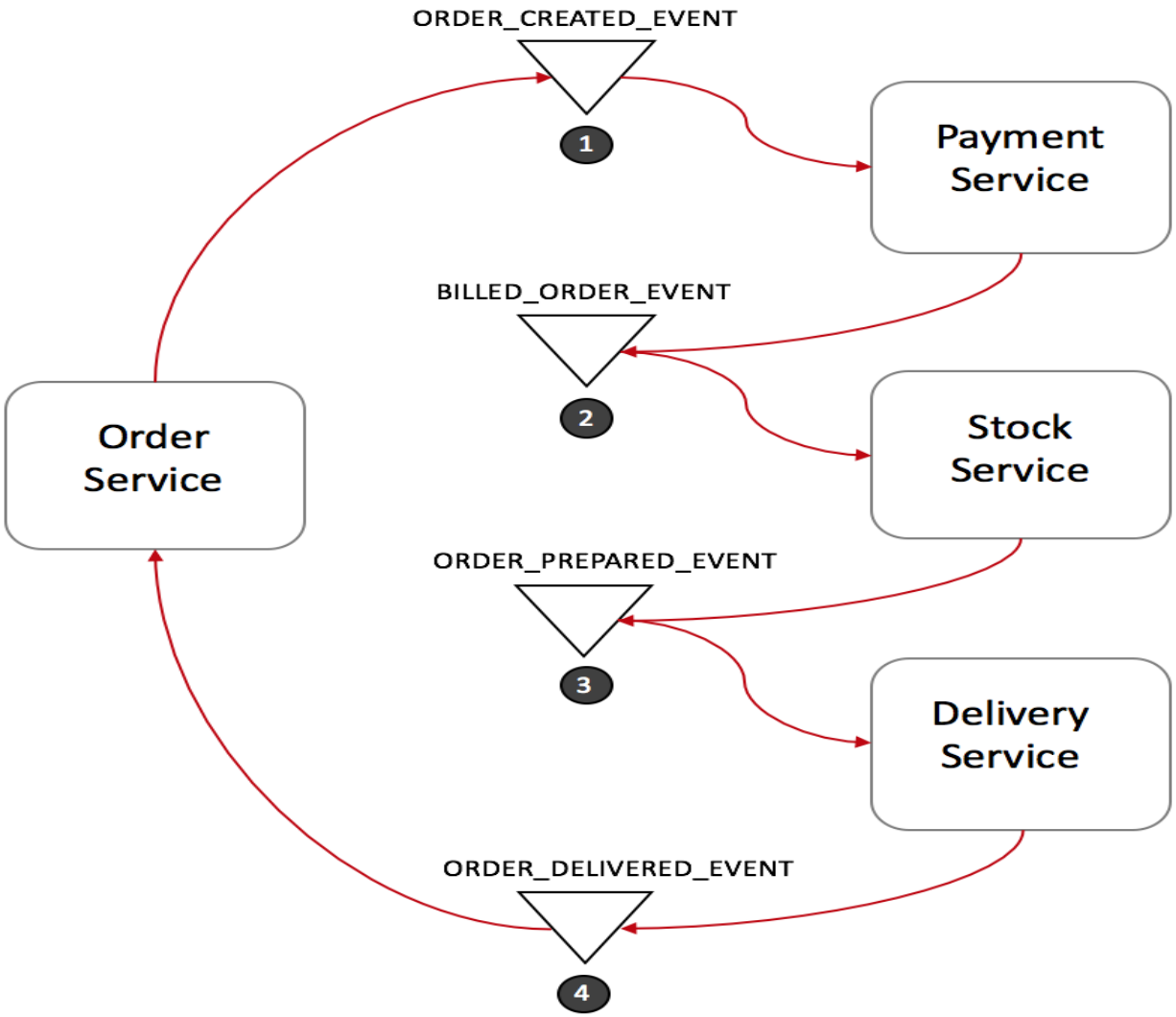


Saga Design Pattern

- 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

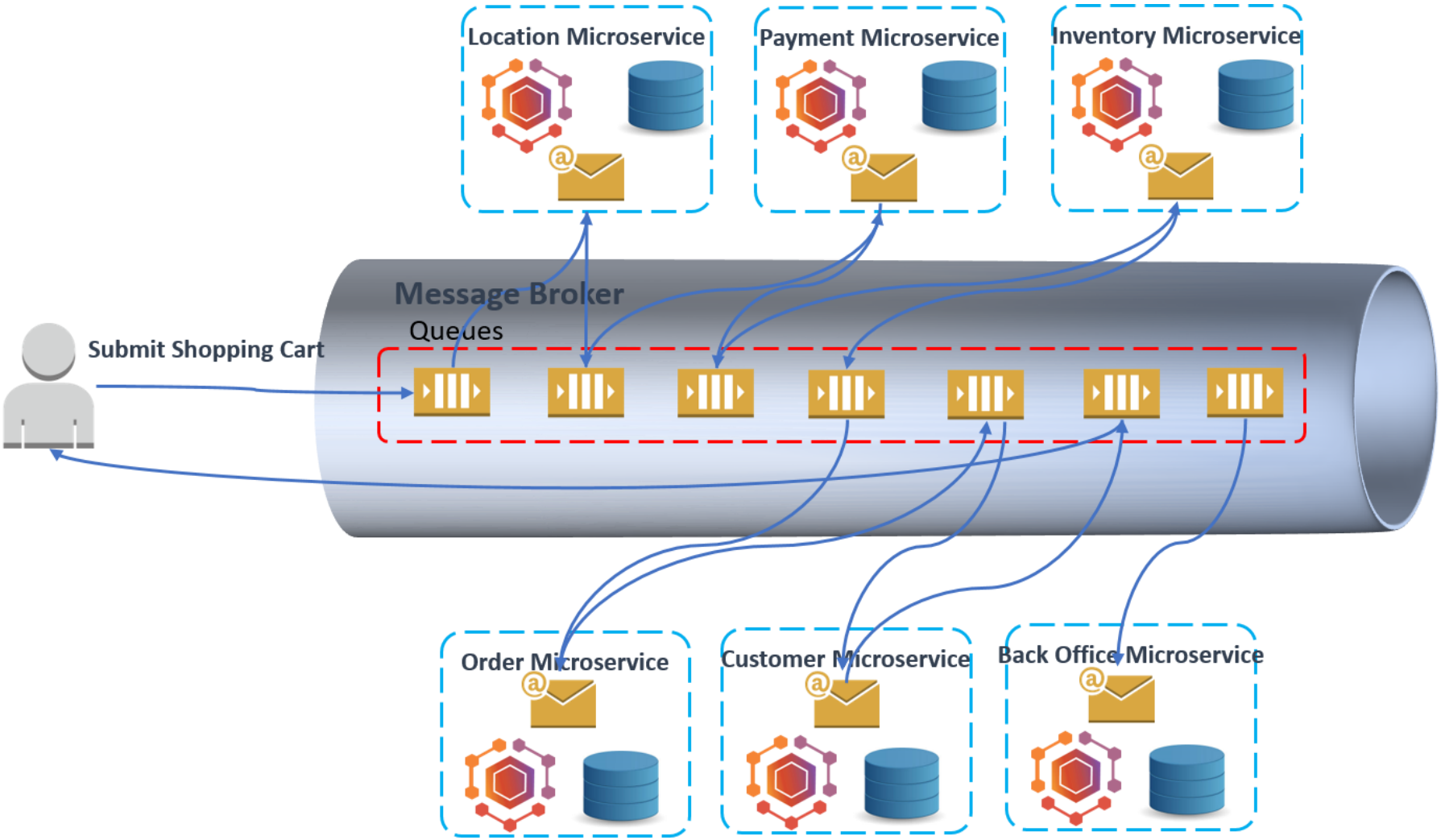
Event Choreography

Microservices



Event Choreography

Microservices

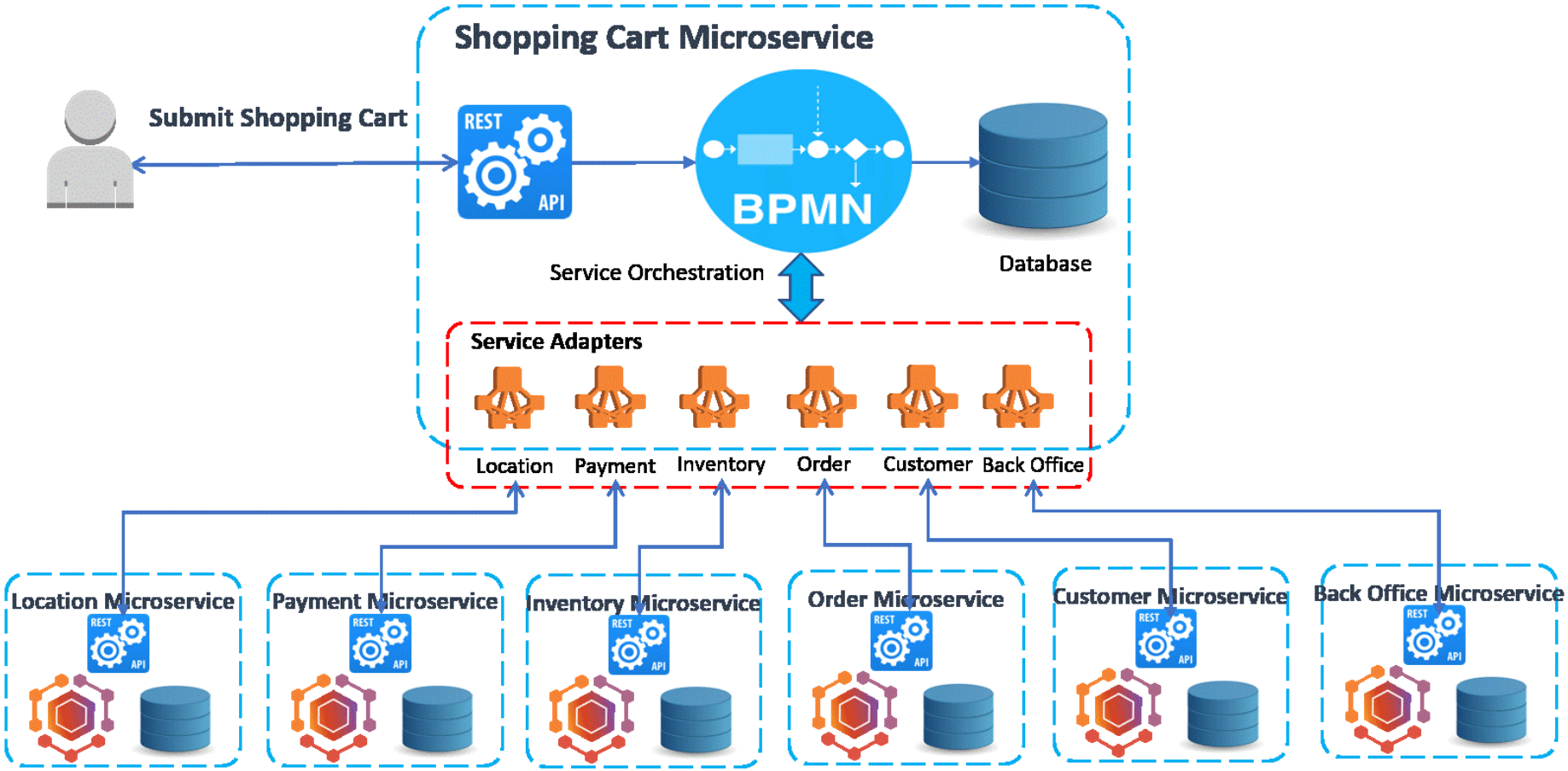


Event-Driven Service Choreography

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

Event Choreography

Microservices



Centralized Service Orchestration

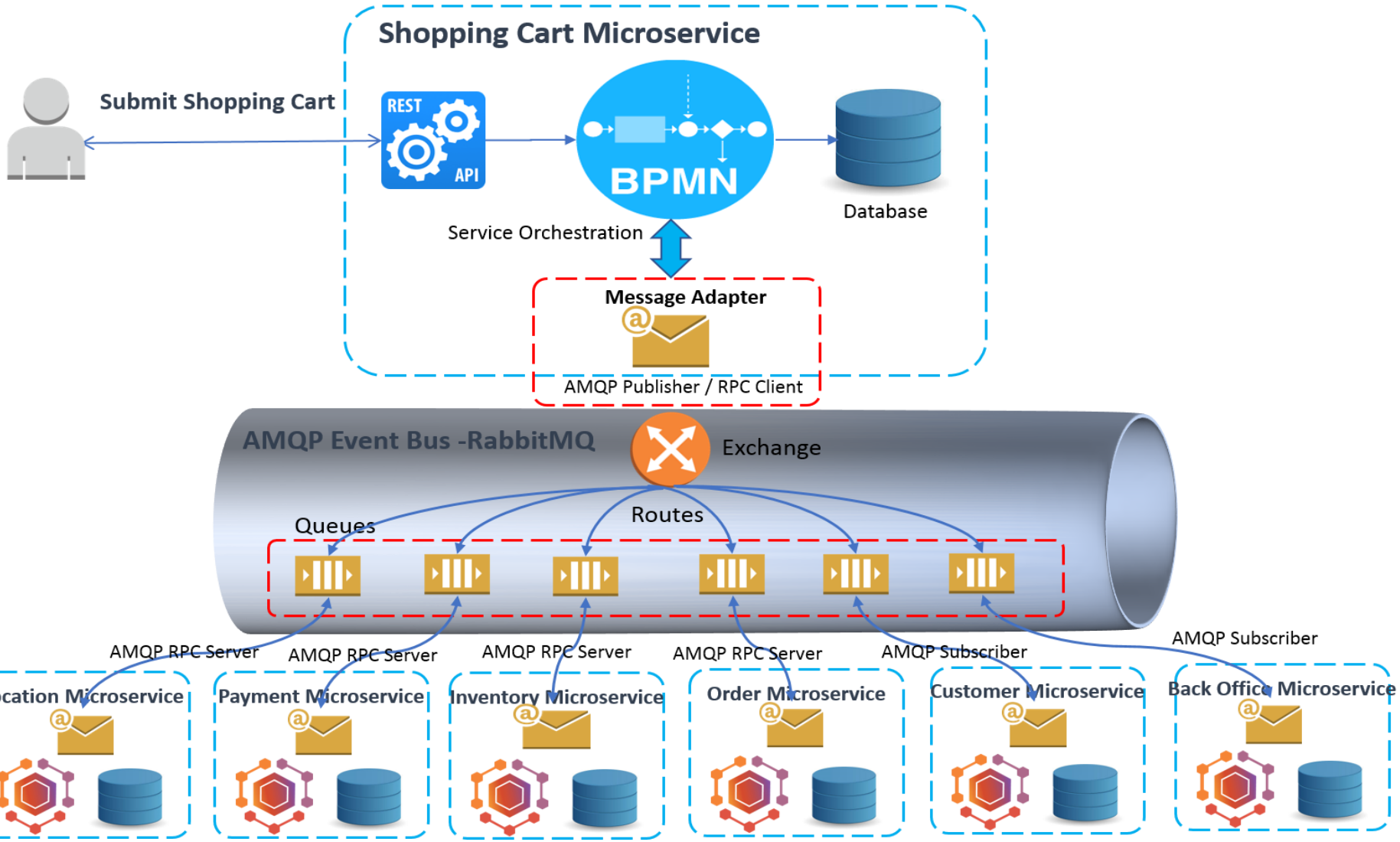
- Easy to maintain: the business flow is modeled as graphical BPMN process in a centralized orchestration microservice. The standard Business Process Model and Notation (BPMN) grants the businesses with the capability of understanding their internal business procedures in a graphical notation. Furthermore,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

Centralized Service Orchestration

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

Event Choreography

Microservices



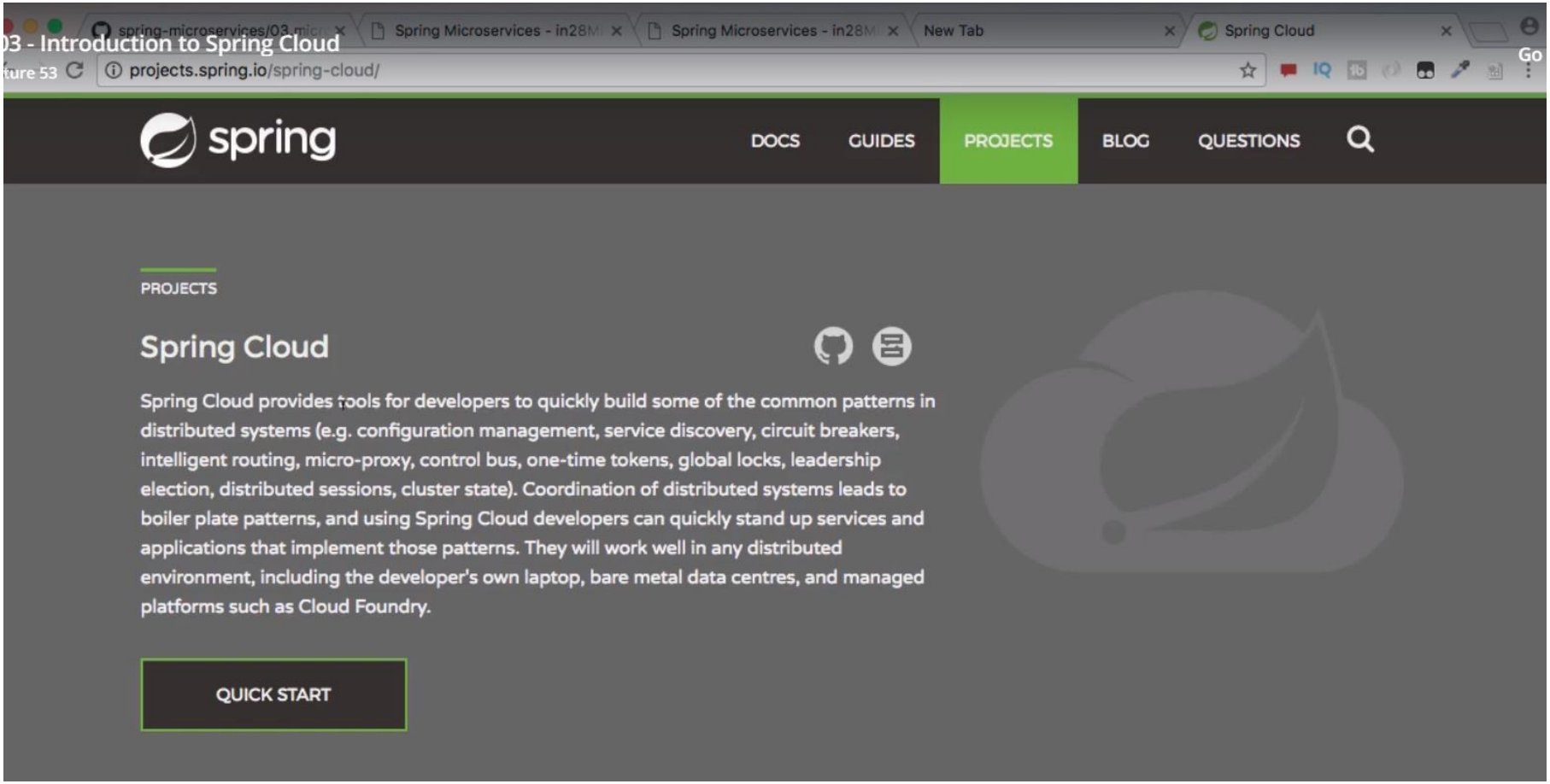
Solutions to Challenges with Microservice Architectures

- **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

Solutions to Challenges with Microservice Architectures

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

Spring Cloud



Spring Cloud

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

Spring Cloud Main Projects

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

Spring Cloud Main Projects

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

Spring Cloud Main Projects

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

Spring Cloud Main Projects

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

Spring Cloud Main Projects

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

Spring Cloud Main Projects

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

Spring Cloud Main Projects

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

Spring Cloud Main Projects

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

Spring Cloud Main Projects

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

Spring Cloud Main Projects

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

Spring Cloud Main Projects

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

Spring Cloud Config Server

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

Spring Cloud Config Server

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

Features

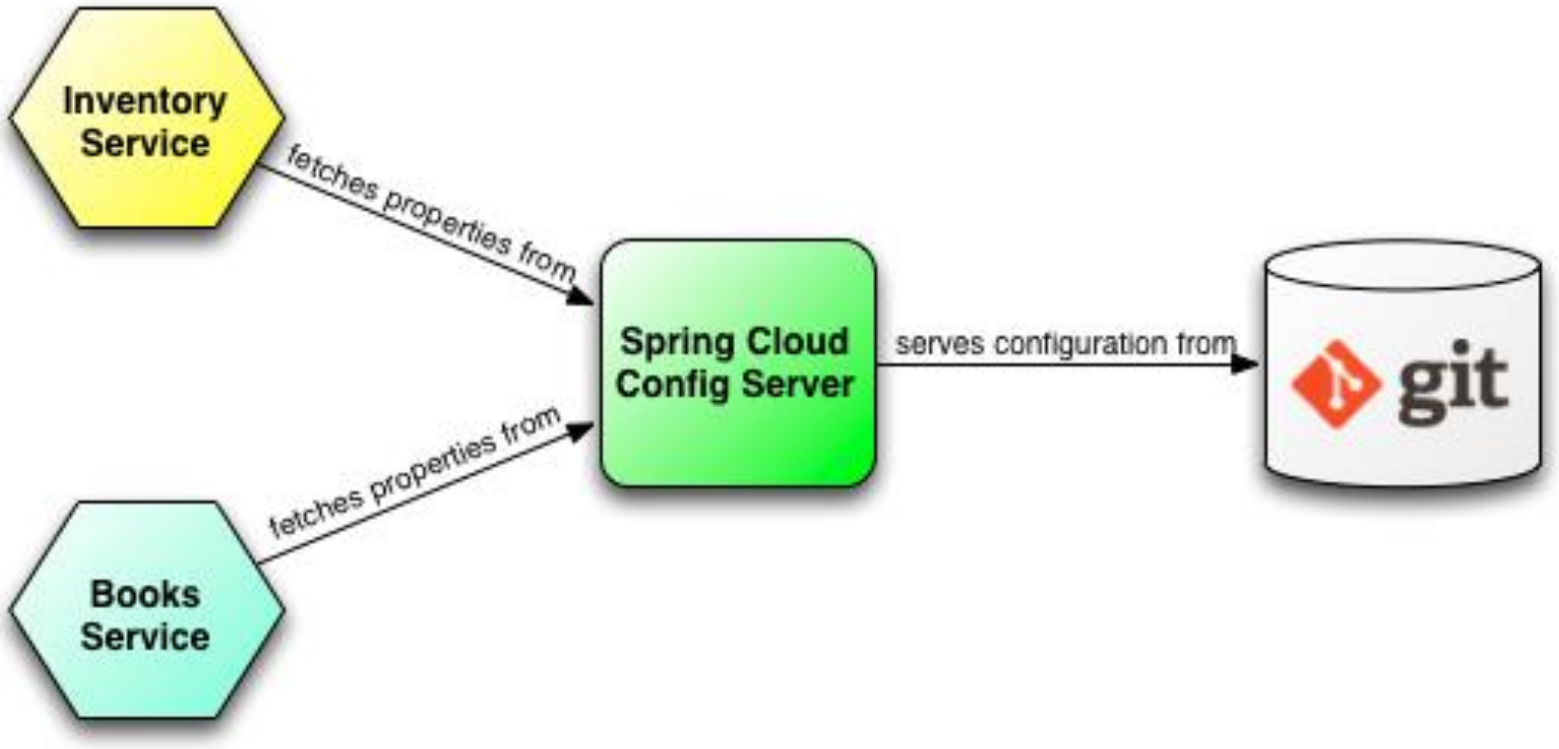
- Spring Cloud Config Server features:
- HTTP, resource-based API for external configuration (name-value pairs, or equivalent YAML content)
- Encrypt and decrypt property values (symmetric or asymmetric)
- Embeddable easily in a Spring Boot application using @EnableConfigServer

Client side Features

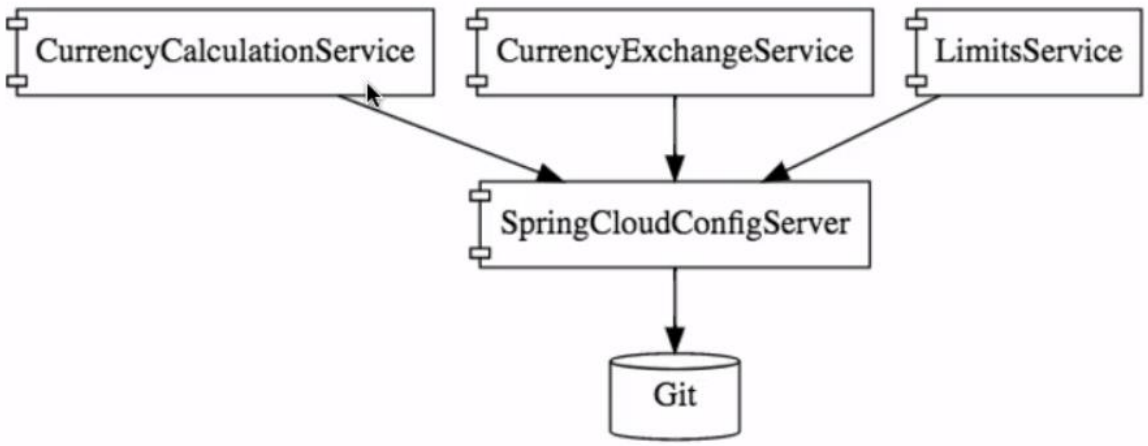
- Config Client features (for Spring applications):
- Bind to the Config Server and initialize Spring Environment with remote property sources
- Encrypt and decrypt property values (symmetric or asymmetric)

Spring Cloud Config Server

Microservices



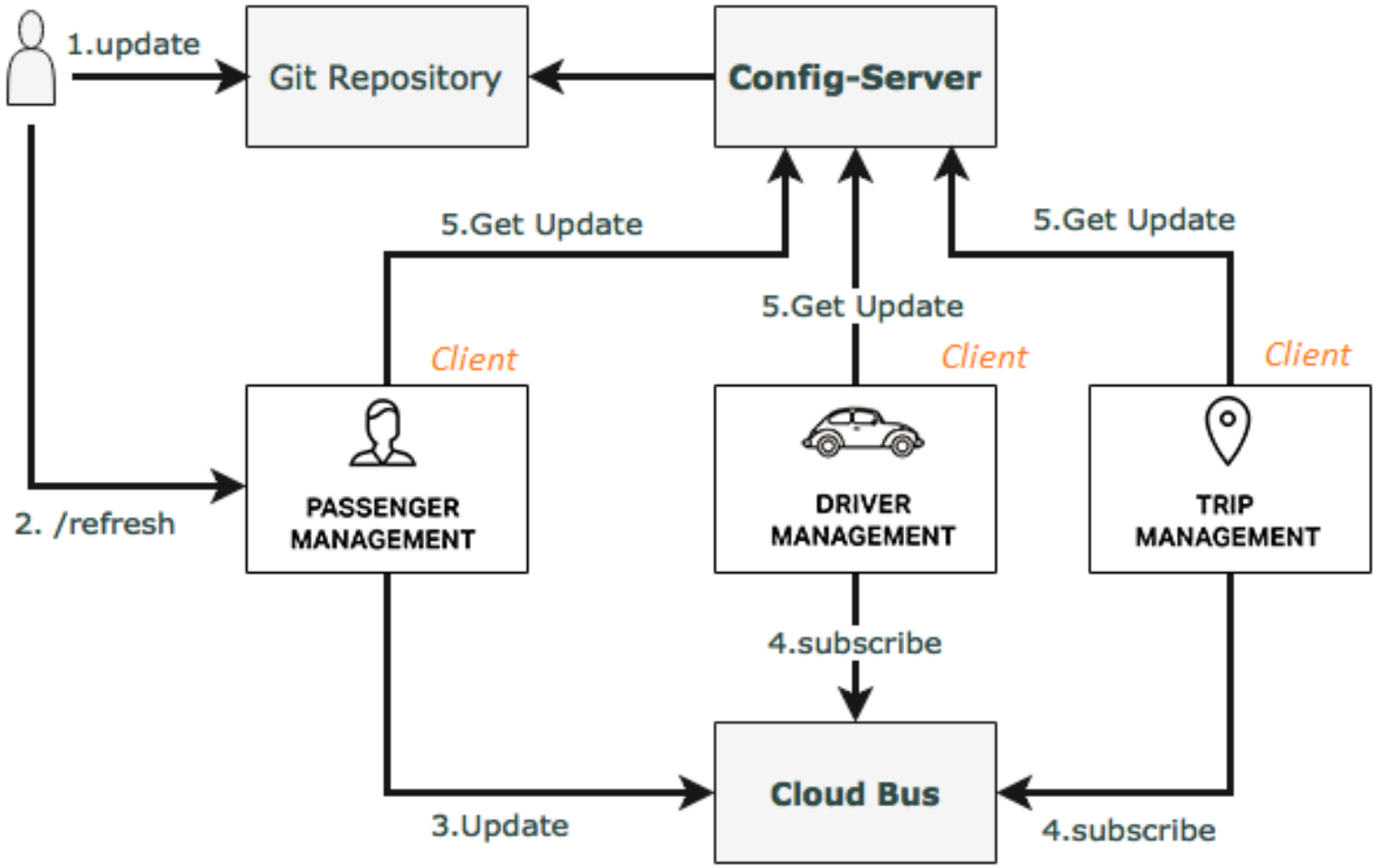
Spring Cloud



Spring Cloud Config Server

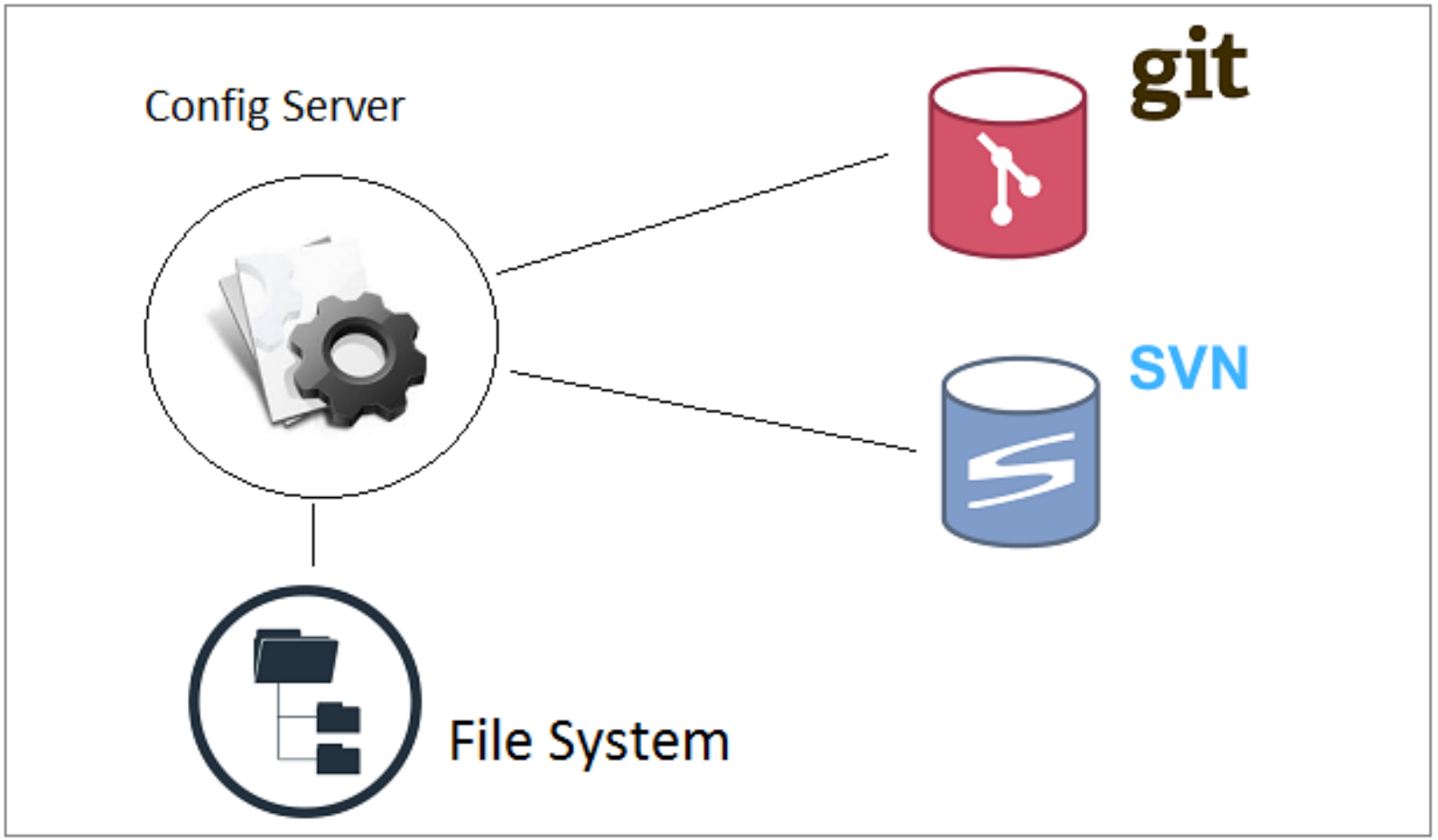
Spring Cloud Bus

Microservices



How does Config Server Store Data

Microservices

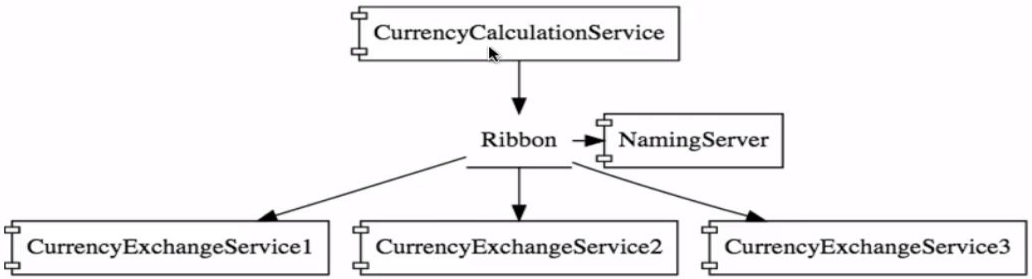


Spring Cloud

DYNAMIC SCALE UP AND DOWN

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

Spring Cloud



Ribbon Load Balancing

DYNAMIC SCALE UP AND DOWN

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

Spring Cloud

VISIBILITY AND MONITORING

- Zipkin Distributed Tracing
- Netflix API Gateway

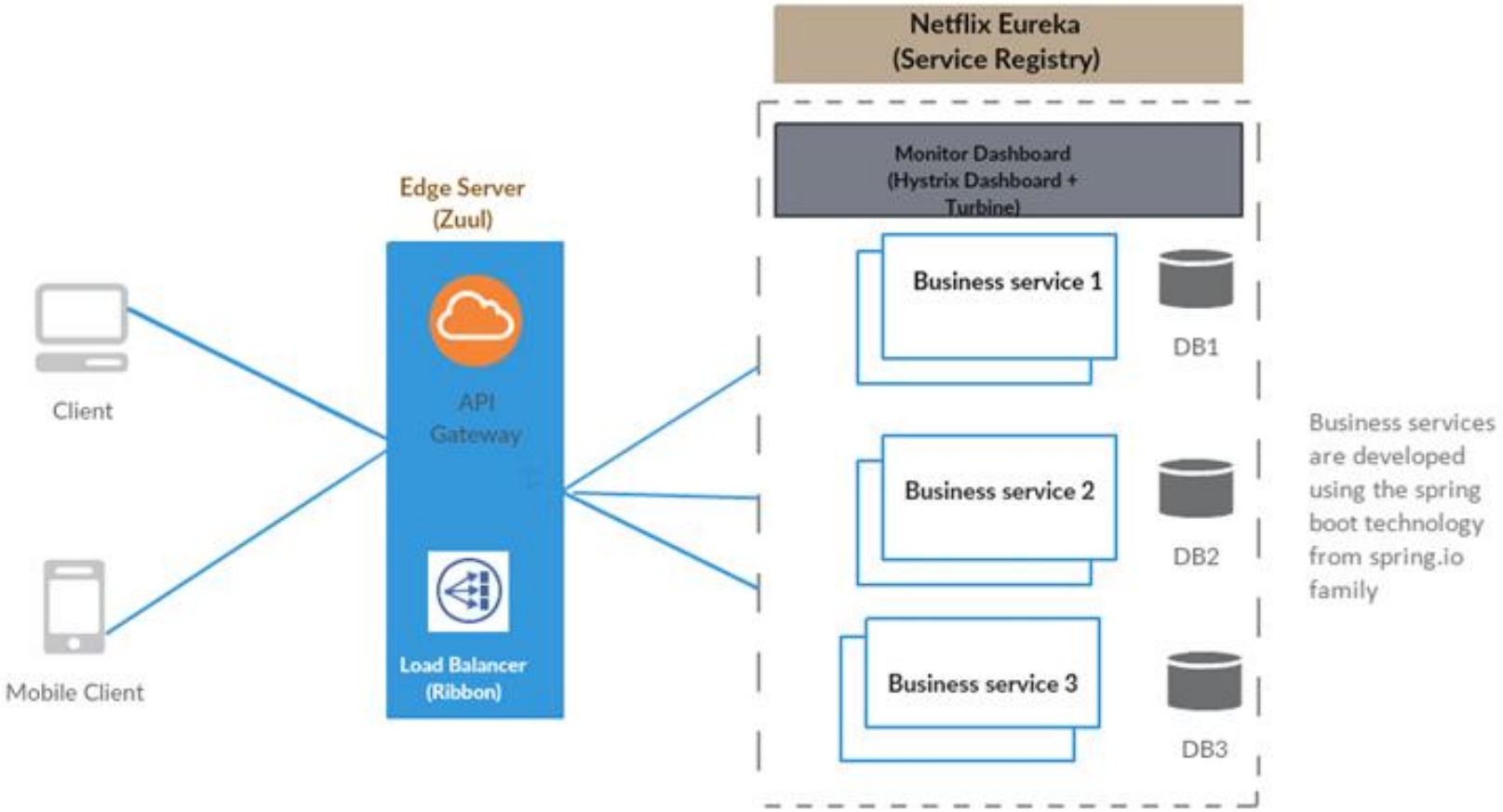
Important Spring Cloud Modules

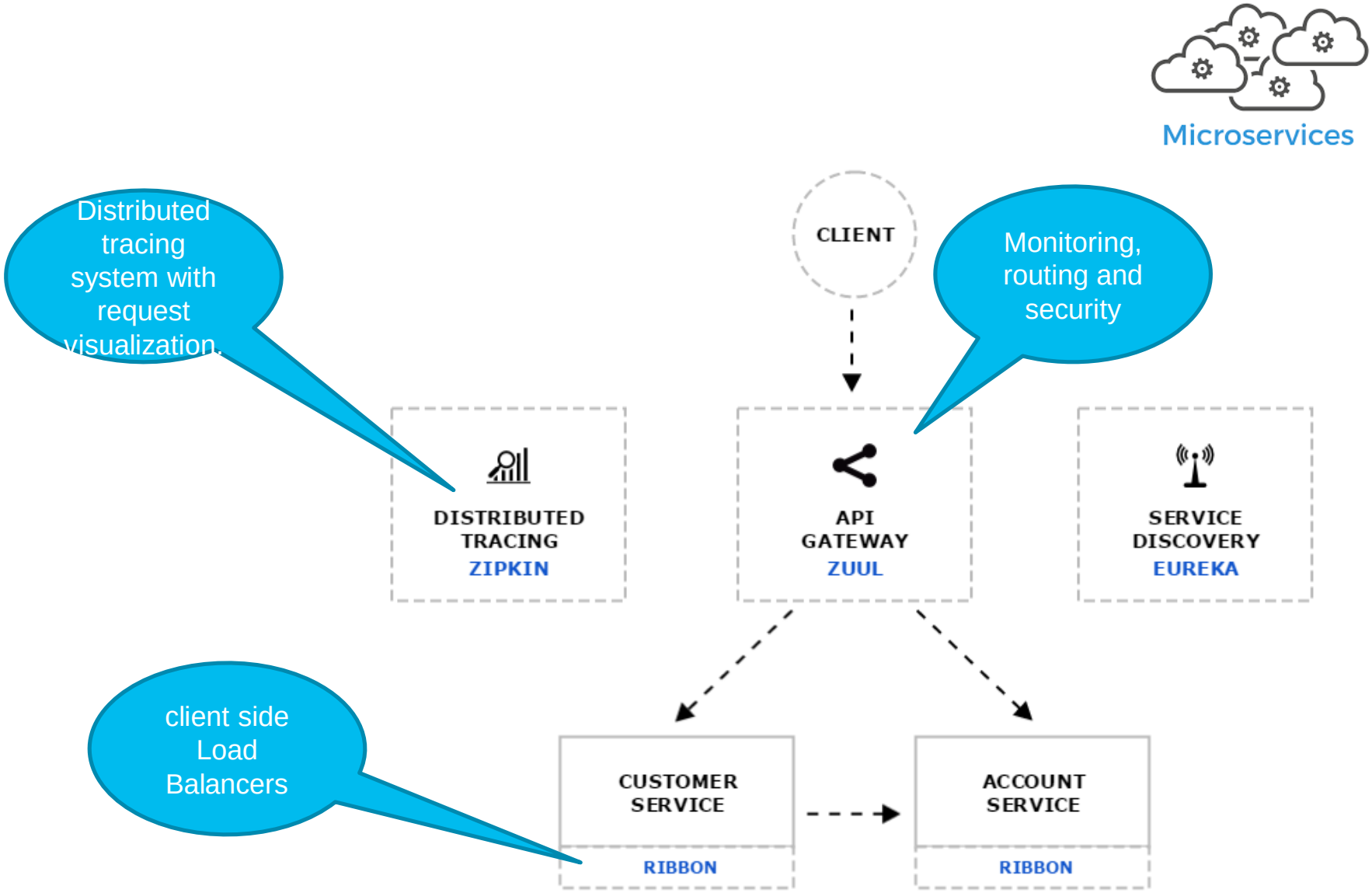
- Dynamic Scale Up and Down. Using a combination of
- Naming Server (Eureka)
- Ribbon (Client Side Load Balancing)
- Feign (Easier REST Clients)
- Visibility and Monitoring with
- Zipkin Distributed Tracing
- Netflix API Gateway
- Configuration Management with
- Spring Cloud Config Server
- Fault Tolerance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

Standardized ports

Ports

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





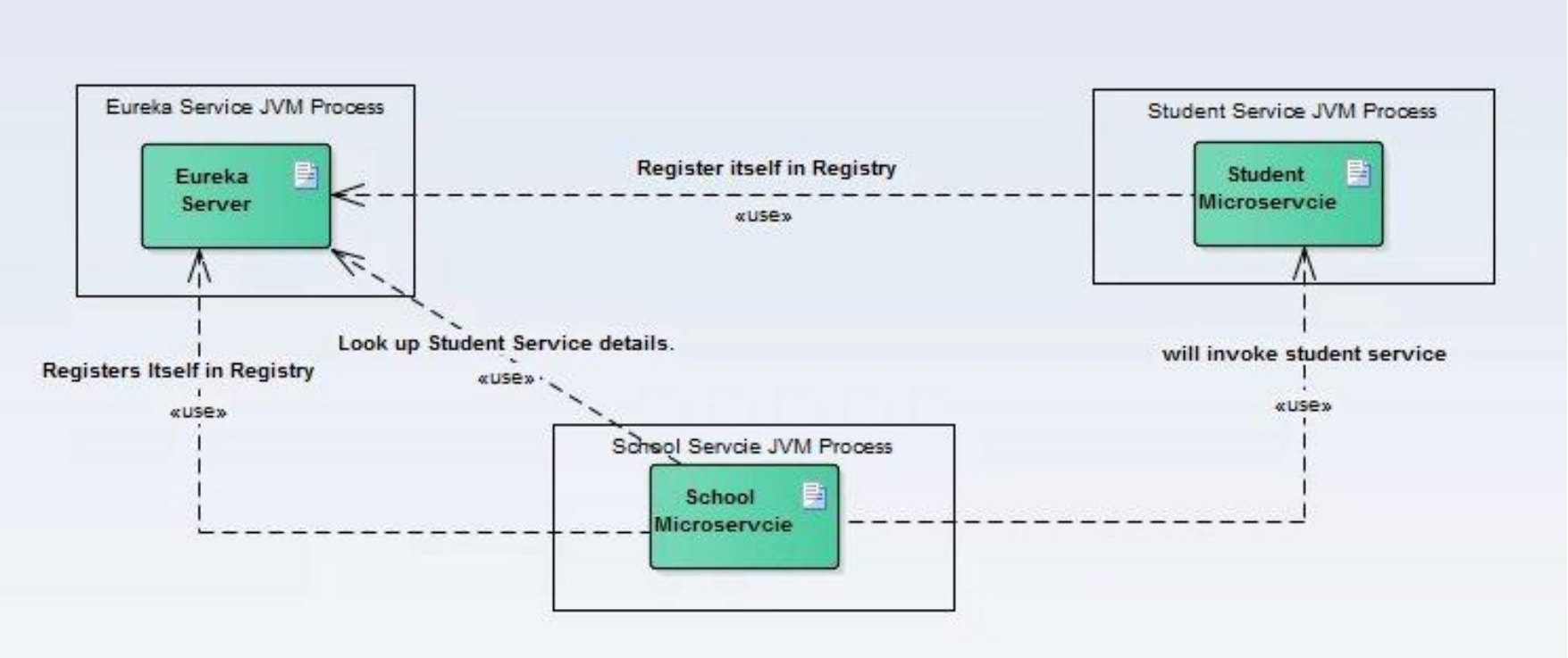
- Create stand-alone Spring applications
- Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
- Provide opinionated 'starter' POMs to simplify your Maven configuration
- Automatically configure Spring whenever possible
- Provide production-ready features such as metrics, health checks and externalized configuration
- Absolutely no code generation and no requirement for XML configuration

Why Eureka Server

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

Spring Cloud Service Discovery with Netflix Eureka

Microservices



Server Configuration

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

Server Configuration

Name	Description
spring.application.name	Unique name for a Eureka server service.
eureka.client.serviceUrl.defaultZone	It consults with other Eureka servers to sync the service registry. As it is in standalone mode, I am giving the local server address.
server.port	In which port the server will be bound.
eureka.client.register-with-eureka	This determines if this server registers itself as a client; as I said earlier, the Eureka server is also acting as a client so that it can sync the registry. The value being false means it prevents itself from acting as a client.
eureka.client.fetch-registry	Does not register itself in the service registry.

Client Configuration

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

Client Configuration

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

Client Configuration

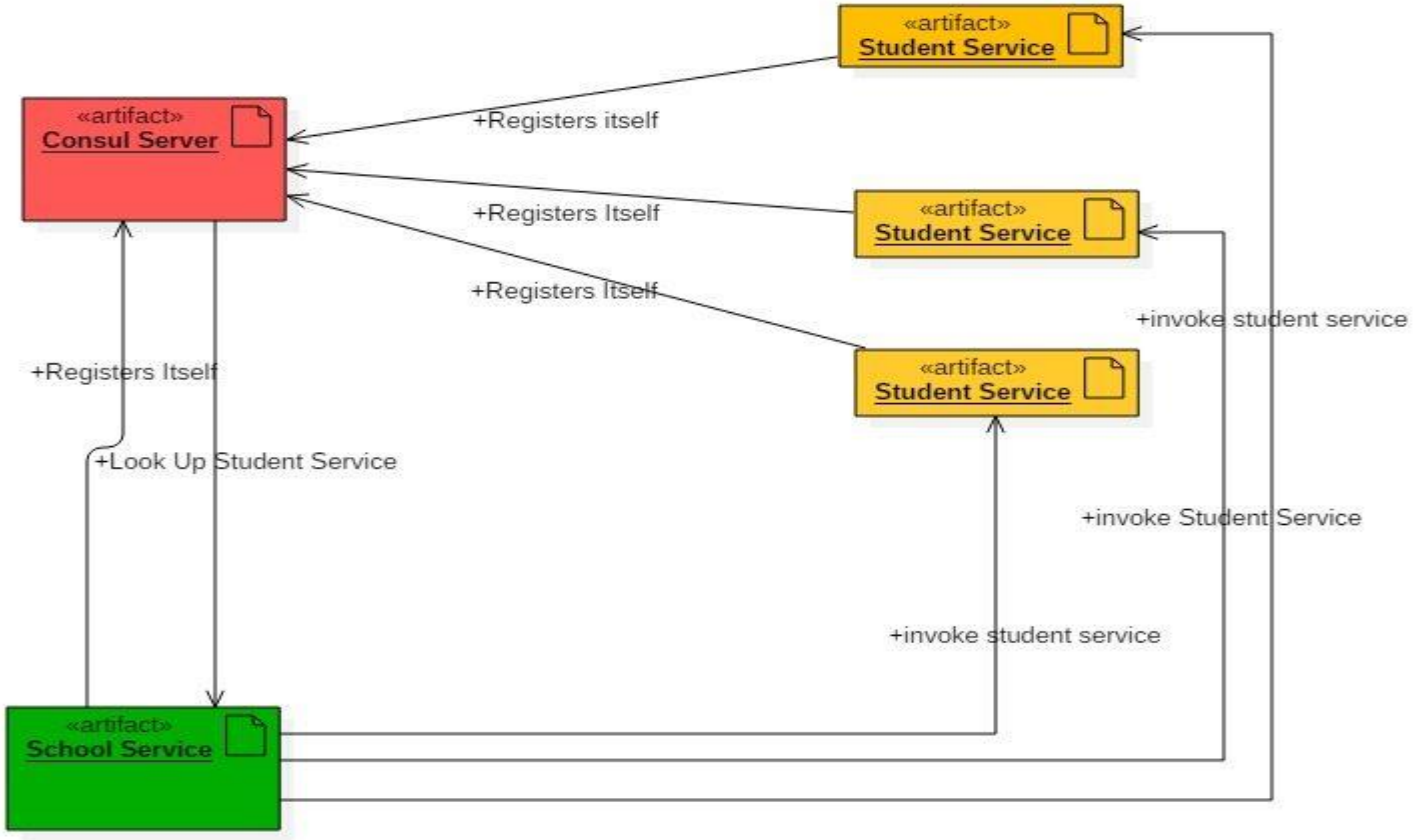
- management:
- security:
- enabled: false #disable the spring security on the management endpoints like /env, /refresh etc.
-
- logging:
- level:
- com.example.howtodoinjava: DEBUG

Consul Service Registration and Discovery Example

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

Consul Service Registration and Discovery Example

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

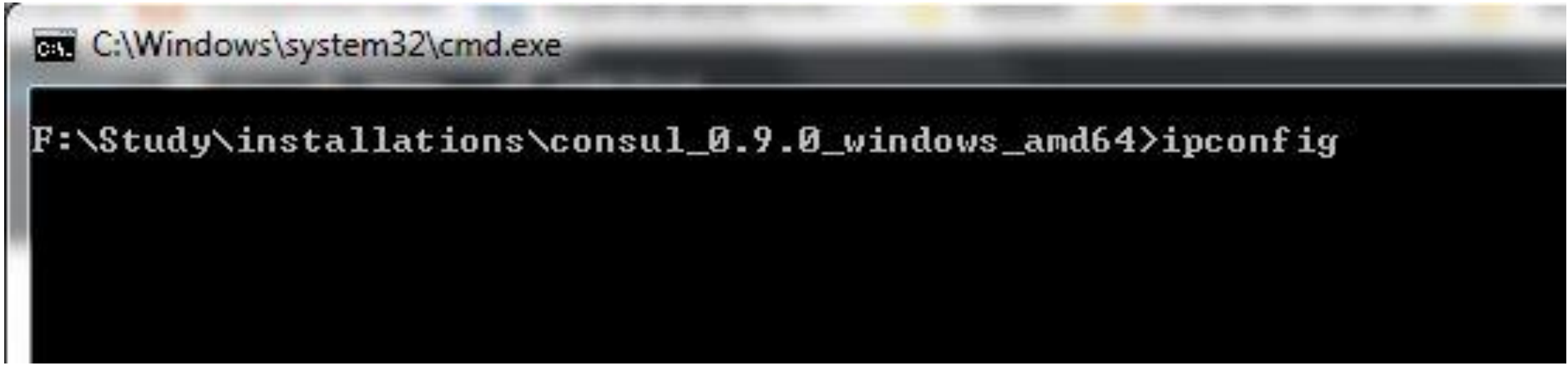


Configuring Consul in Local workstation

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

Configuring Consul in Local workstation

- Make sure you enter the correct bind address, it would be different depending on the LAN settings. Do a ipconfig in command prompt to know your IpV4 address and use it here.

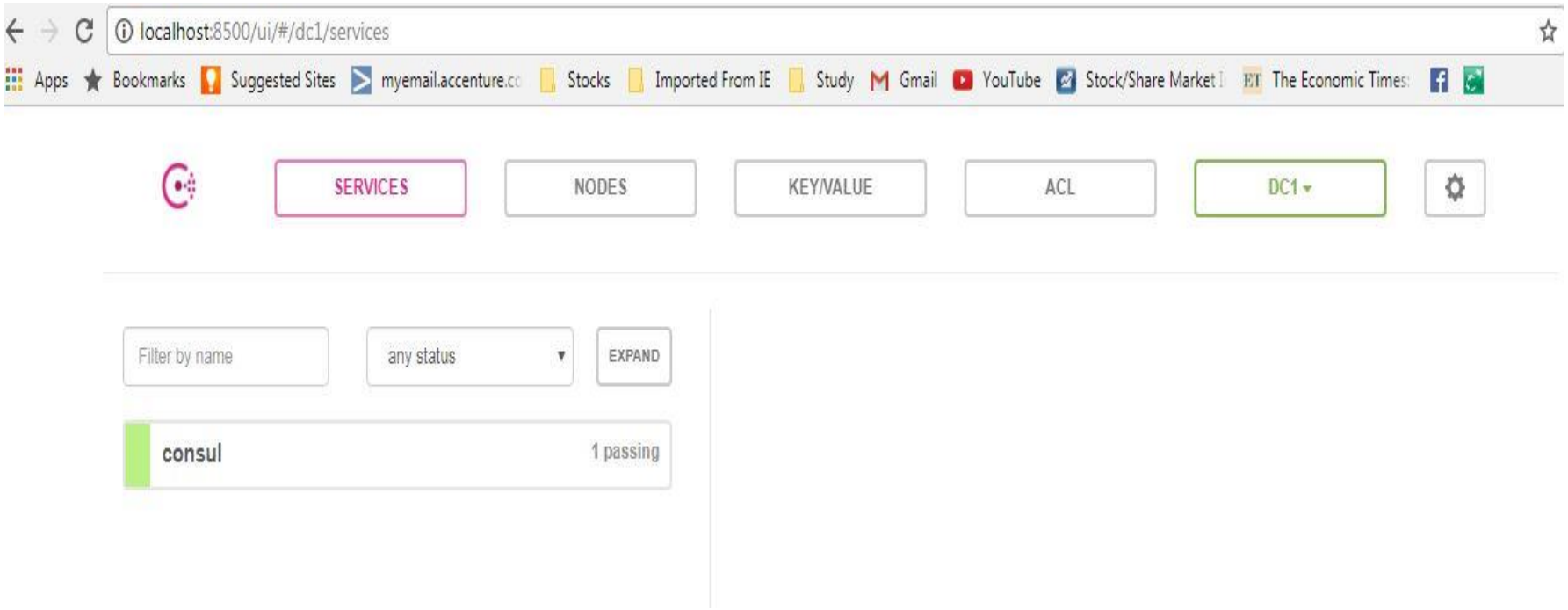


Configuring Consul in Local workstation


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

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

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

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

Configuring Consul in Local workstation

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



Consul Client

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

Service Configuration

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

Hystrix Circuit Breaker Pattern – Spring Cloud

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

What is Circuit Breaker Pattern?

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

What is Circuit Breaker Pattern?

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

What is Circuit Breaker Pattern?

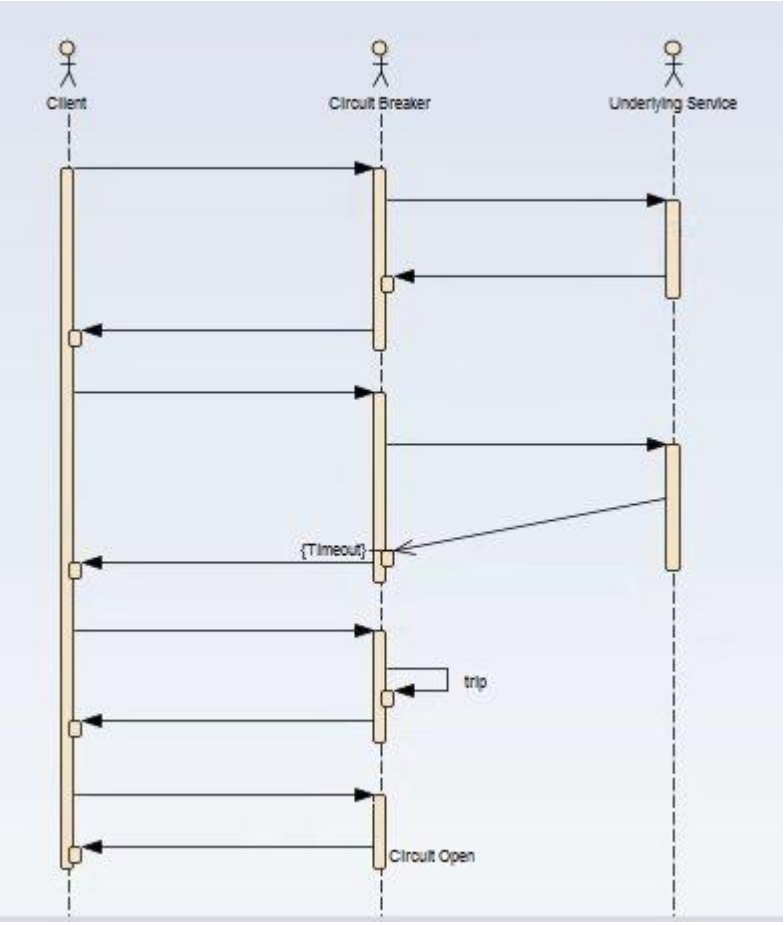
- Here circuit breaker pattern comes handy and it redirects traffic to a fall back path once it sees any such scenario.
- Also it monitors the defective service closely and restore the traffic once the service came back to normalcy.

What is Circuit Breaker Pattern?

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

What is Circuit Breaker Pattern?

Microservices



Hystrix Dashboard

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

Eureka | localhost:8765/ | Hystrix Circuit | localhost:9091/ | Consul by Hash | Spring Initializr | New Tab | localhost:9091/ | Hystrix Dashbo

localhost:9091/hystrix

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

Hystrix Dashboard

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

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

Delay: 2000 ms Title: Example Hystrix App

Monitor Stream

Type here to search

ENG 00:04 20/11/2018

Eureka | localhost:8765/ | Hystrix Circuit B | localhost:9091/ | Consul by Hash | Spring Initializr | New Tab | localhost:9091/ | Hystrix Monito

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

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

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

HYSTRIX
DEFEND YOUR APP

Circuit Sort: Error then Volume | Alphabetical | Volume | Error | Mean | Median | 90 | 99 | 99.5

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

callProductServiceAndGetData

0 0 0.0 %
0 0
0 0

Host: 0.0/s
Cluster: 0.0/s
Circuit Closed

Hosts	1	90th	0ms
Median	0ms	99th	0ms
Mean	0ms	99.5th	0ms

Thread Pools Sort: Alphabetical | Volume |

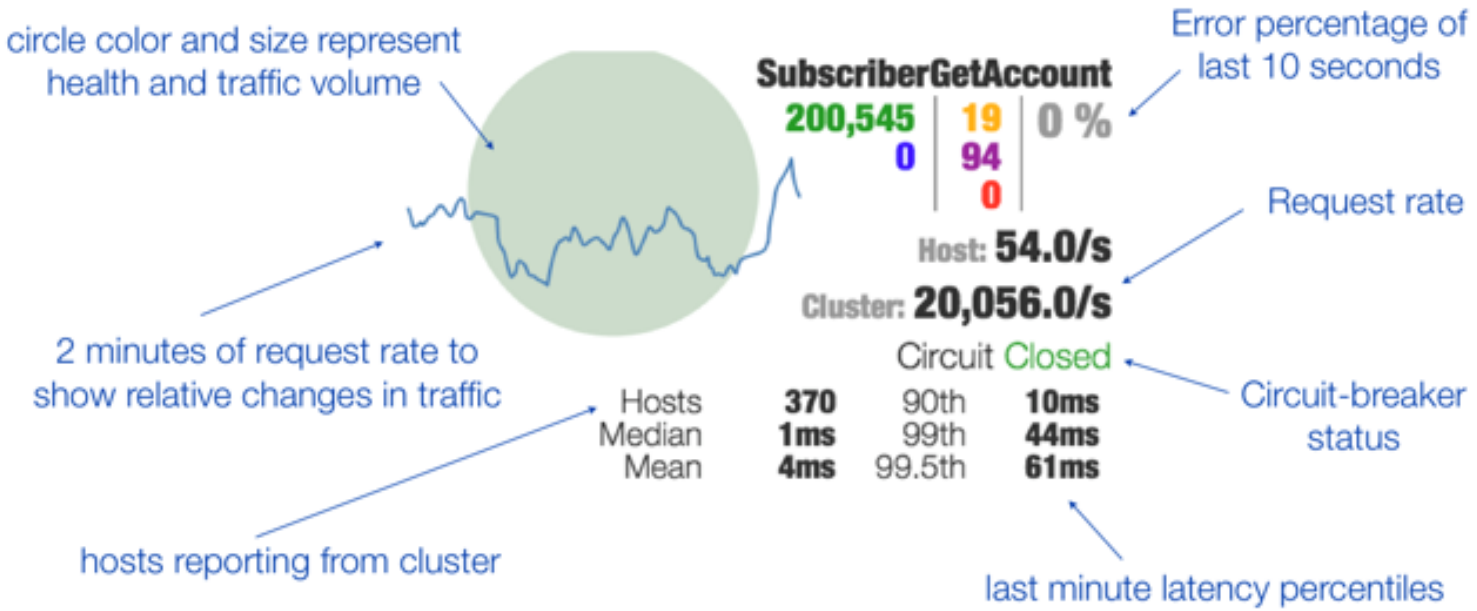
ProductDelegate

Host: 0.0/s
Cluster: 0.0/s

Active	0	Max Active	0
Queued	0	Executions	0
Pool Size	1	Queue Size	5

Type here to search

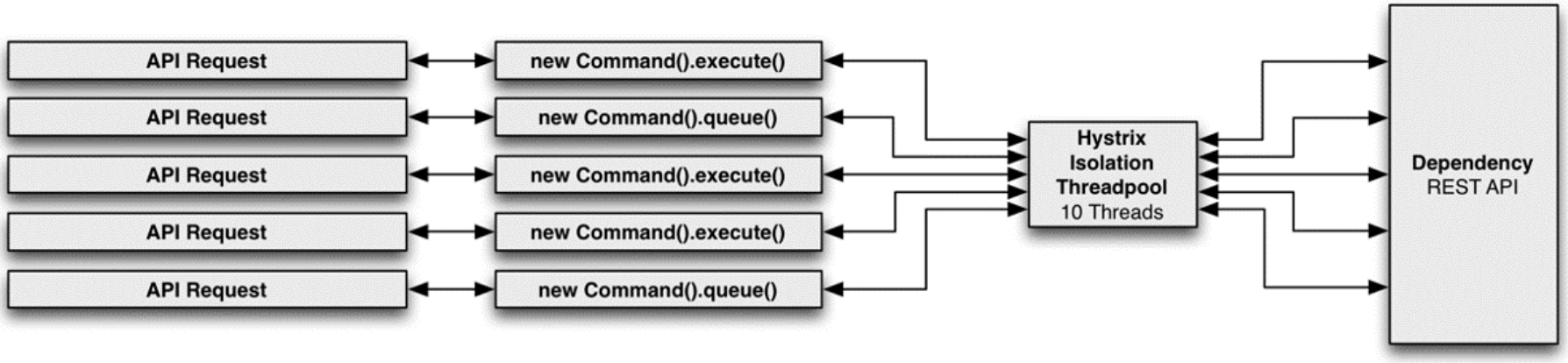
ENG 00:04 20/11/2018



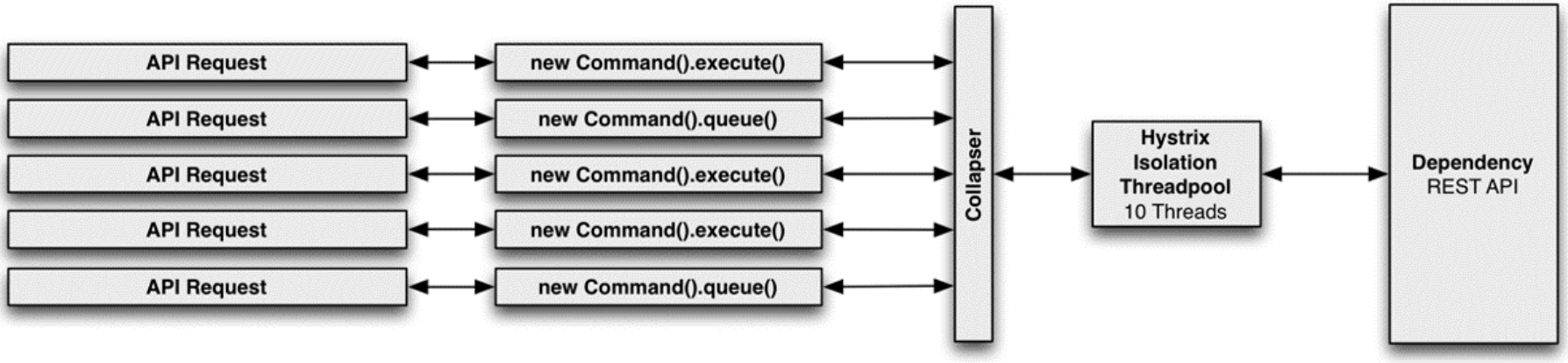
Rolling 10 second counters
with 1 second granularity

Successes	200,545	19	Thread timeouts
Short-circuited (rejected)	0	94	Thread-pool Rejections
		0	Failures/Exceptions

Without Collapsing: Request == Thread == Network Connection



With Collapsing: Requests within 'window' == 1 Thread == 1 Network Connection



Using Blue-Green Deployment to Reduce Downtime and Risk

- Blue-green deployment is a technique that reduces downtime and risk by running two identical production environments called Blue and Green.

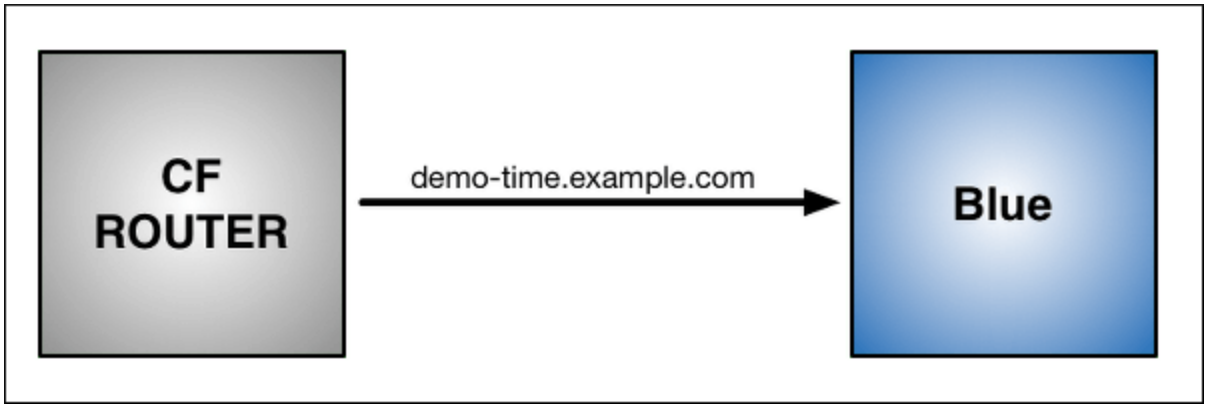
- At any time, only one of the environments is live, with the live environment serving all production traffic. For this example, Blue is currently live and Green is idle.

Using Blue-Green Deployment to Reduce Downtime and Risk

- As you prepare a new version of your software, deployment and the final stage of testing takes place in the environment that is *not* live: in this example, Green.
- Once you have deployed and fully tested the software in Green, you switch the router so all incoming requests now go to Green instead of Blue. Green is now live, and Blue is idle.

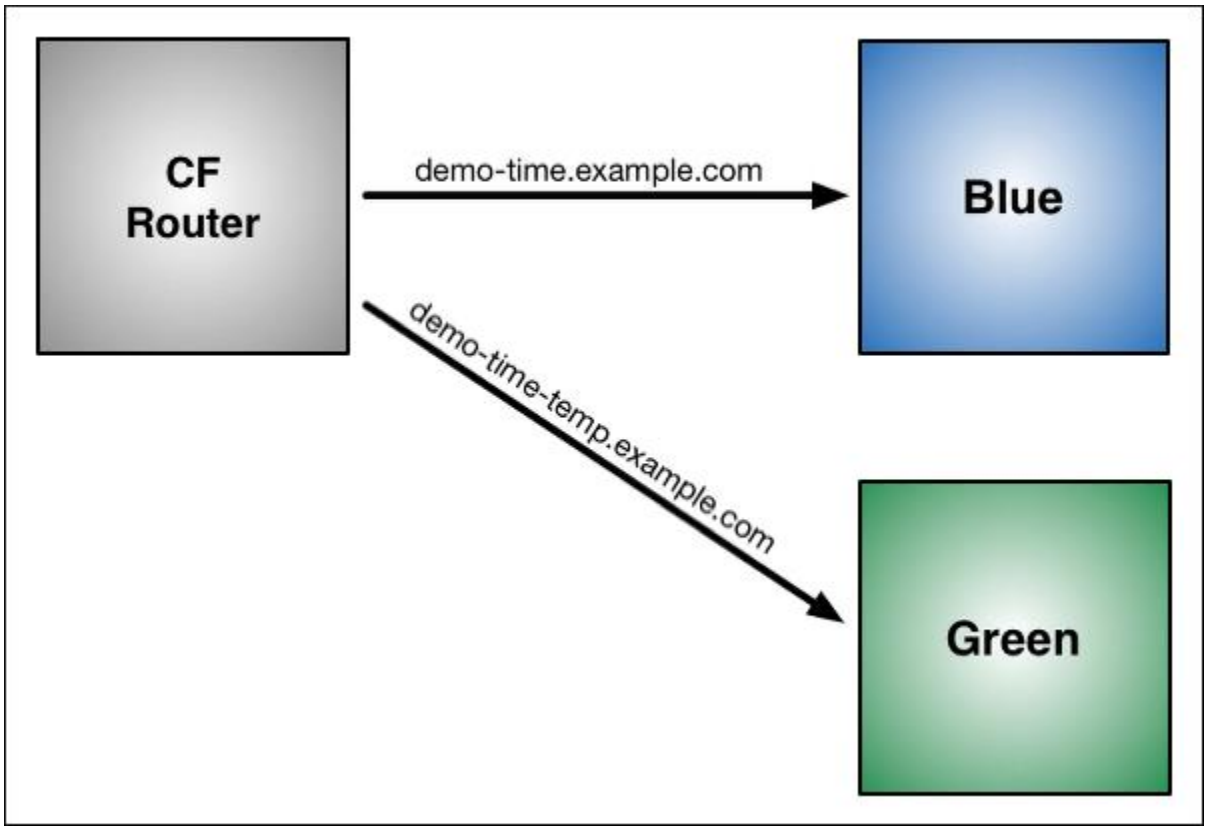
Blue-Green Deployment with Cloud Foundry

- $ cf push Blue -n demo-time



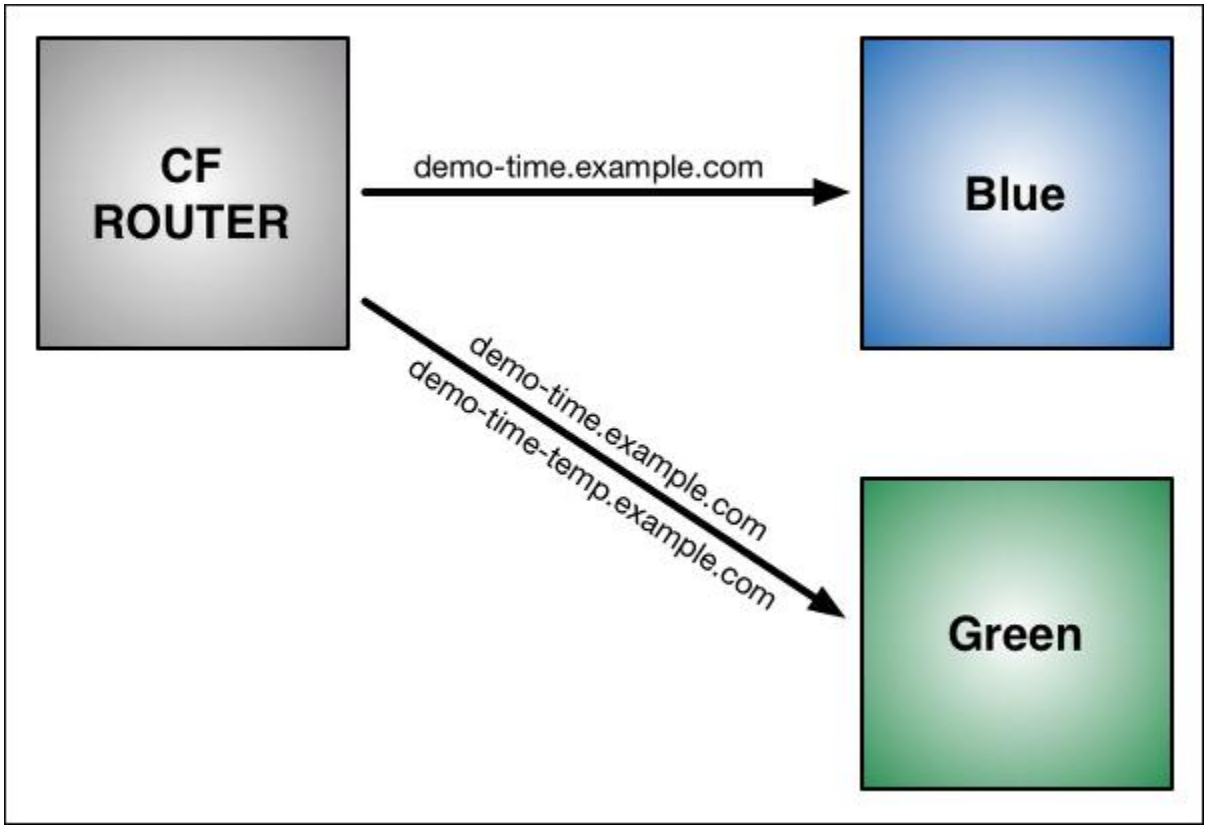
Blue-Green Deployment with Cloud Foundry

- $ cf push Green -n demo-time-temp



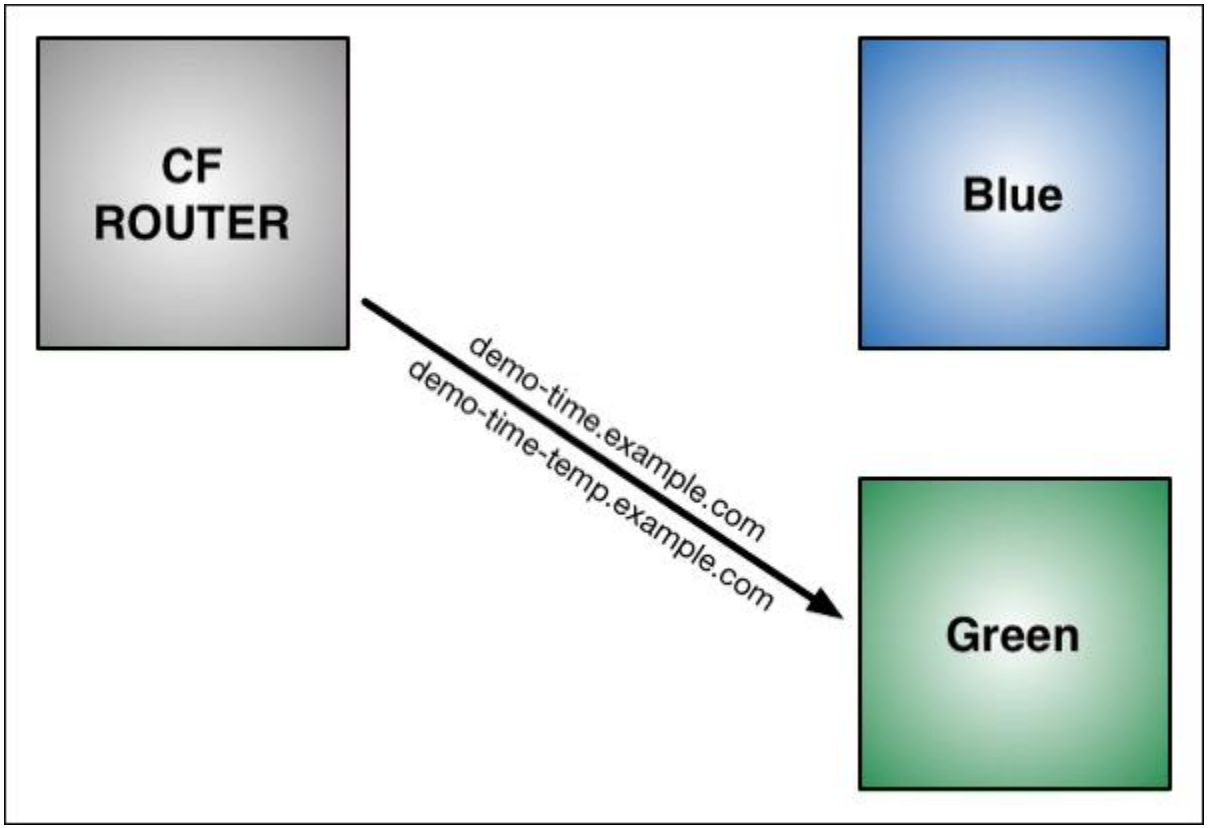
Blue-Green Deployment with Cloud Foundry

- $ cf map-route Green example.com -n demo-time
- Binding demo-time.example.com to Green... OK



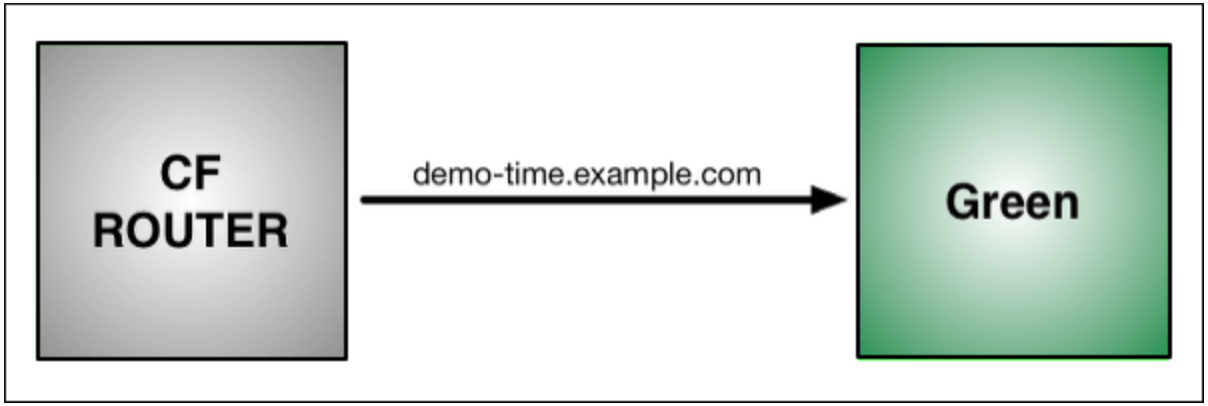
Blue-Green Deployment with Cloud Foundry

- $ cf unmap-route Blue example.com -n demo-time
- Unbinding demo-time.example.com from blue... OK



Blue-Green Deployment with Cloud Foundry

- cf delete-route



How to Deploy Spring Boot Application to Cloud Foundry Platform

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

What is Cloud Foundry

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

What is Cloud Foundry

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

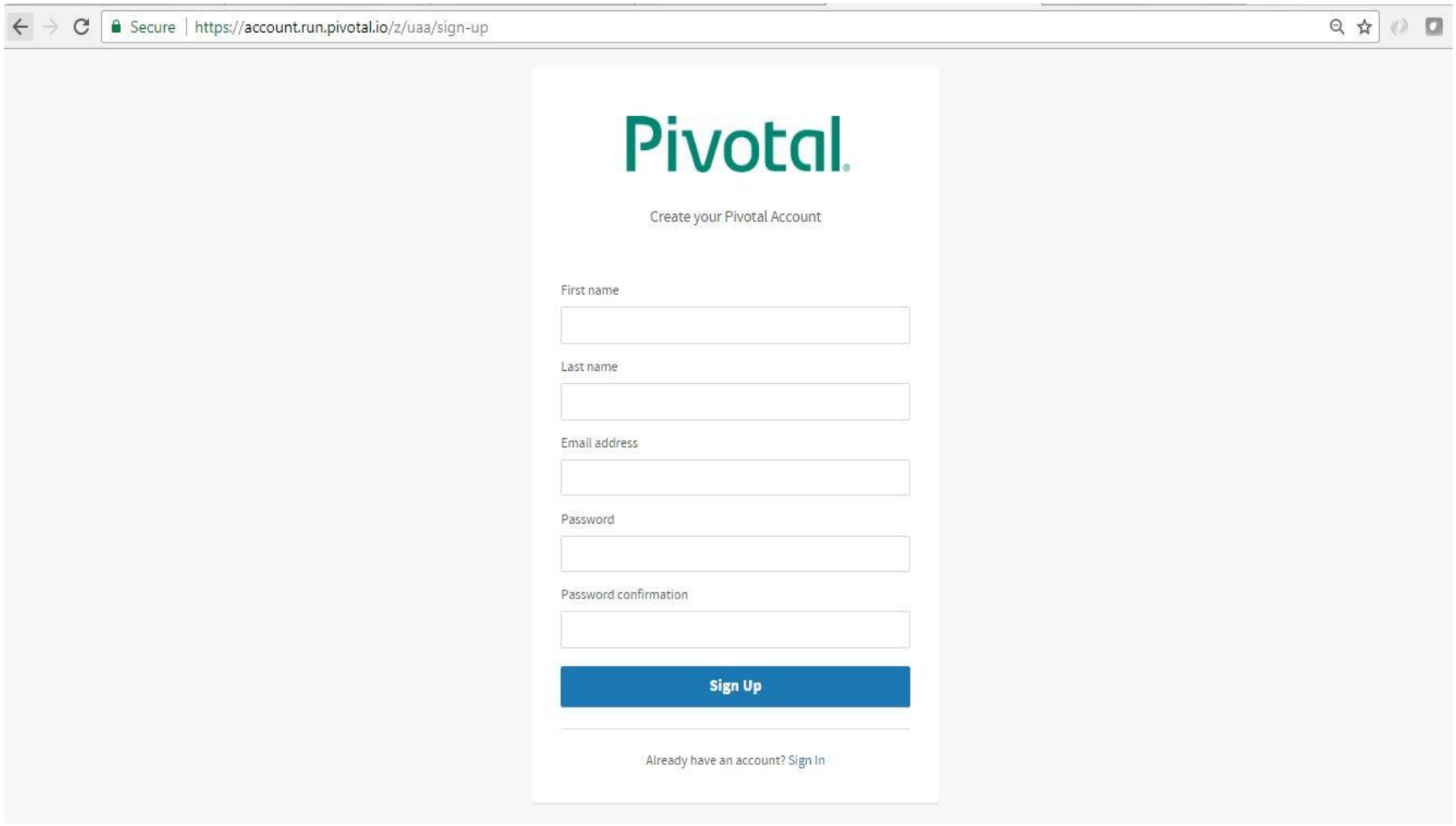
Cloud Foundry Installation for Windows

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

Cloud Foundry Installation for Windows

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

Setup PWS Console

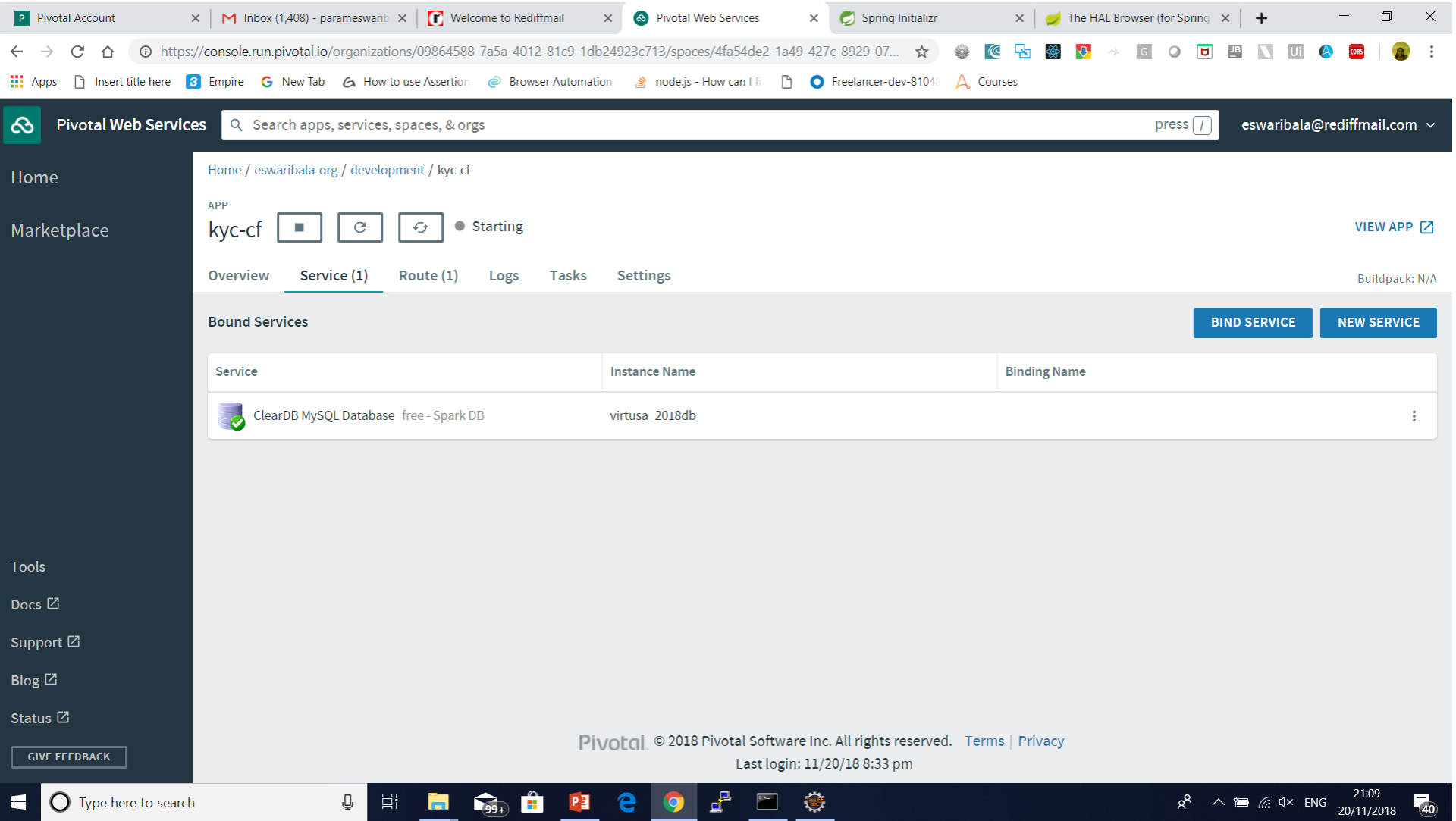


Login and logout from PWS Console using CLI

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

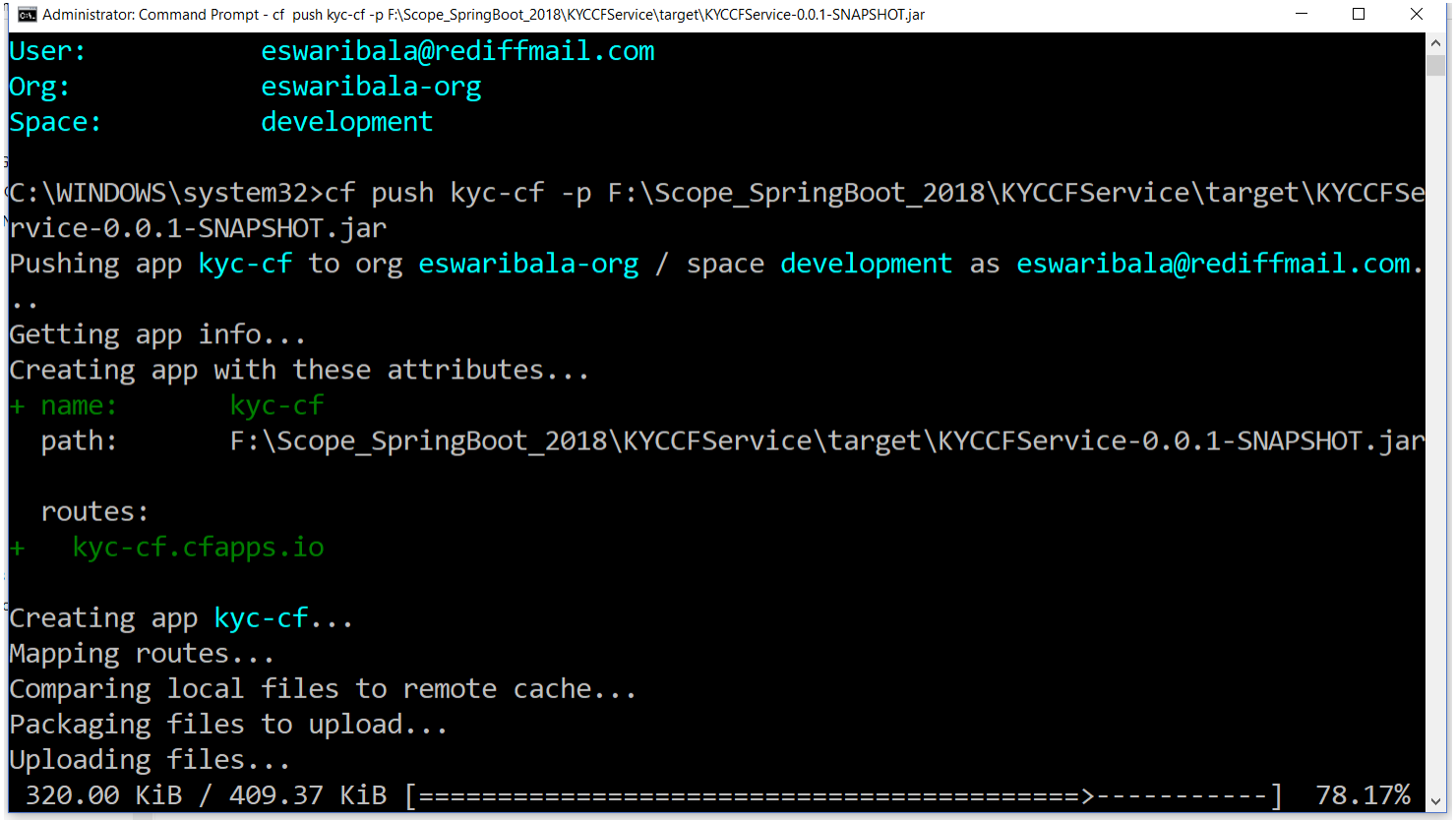
Create Database in cloud foundry

Microservices



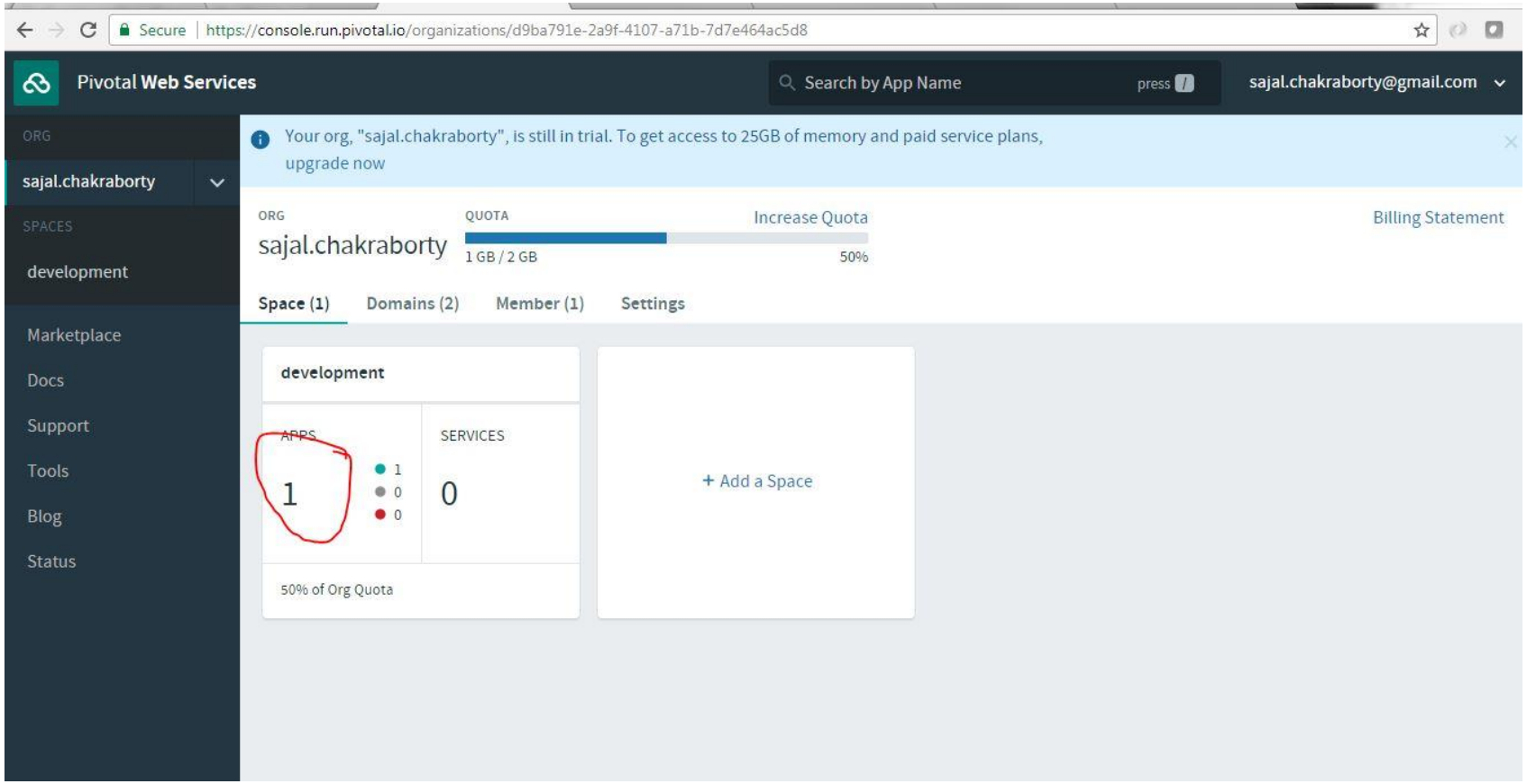
Push Application to Console

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



Verify Application Deployed

Microservices

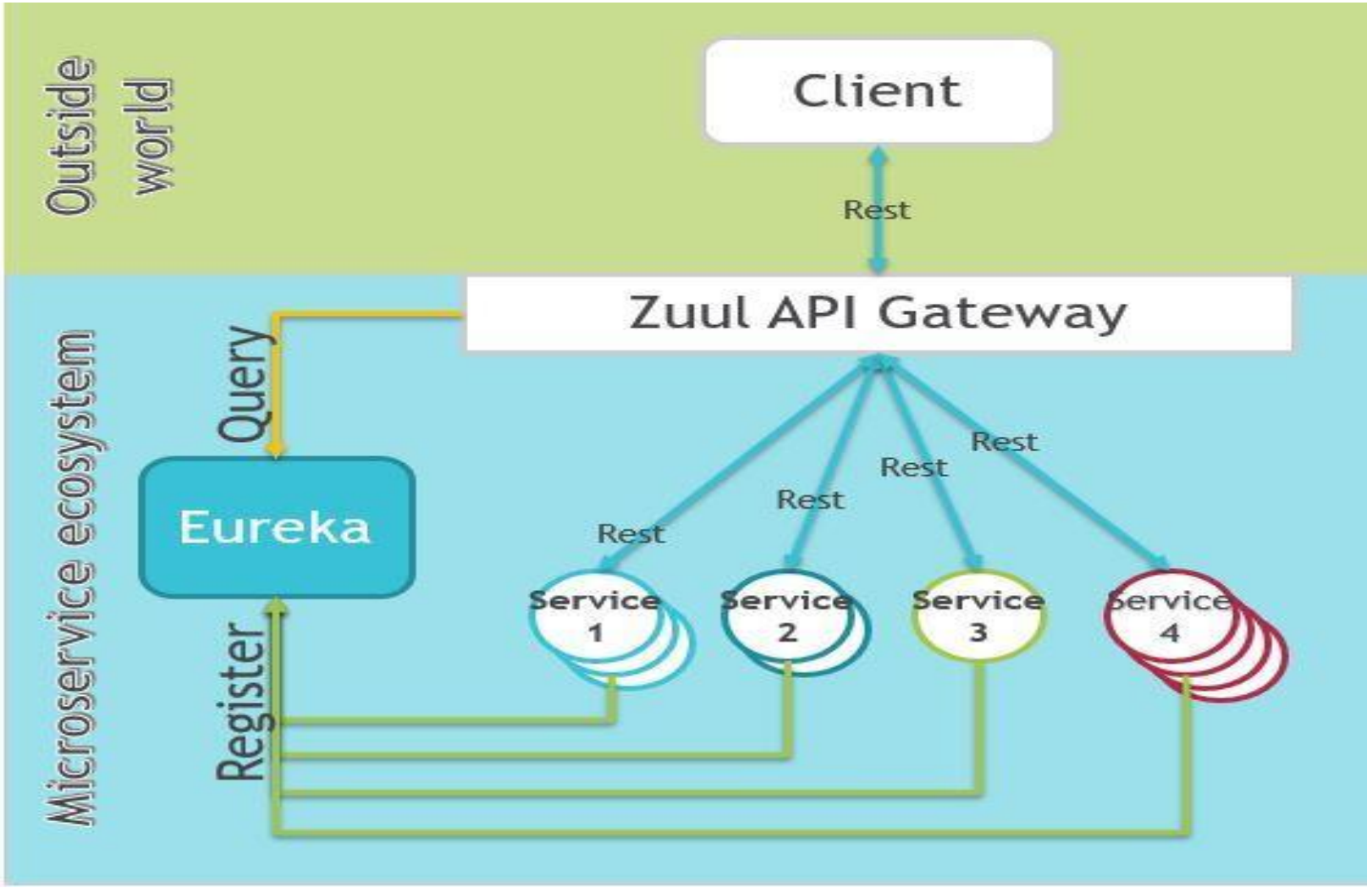


What is Zuul?

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

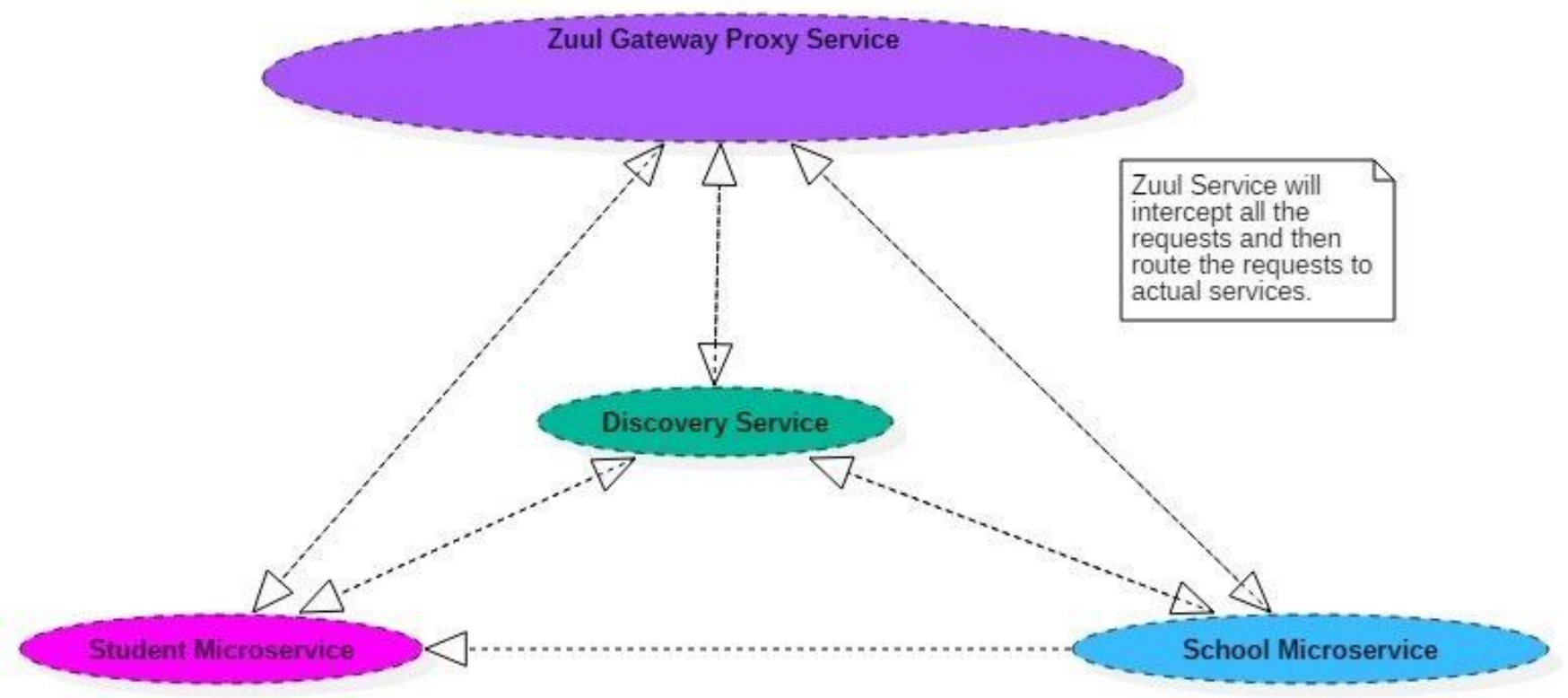
What is Zuul?

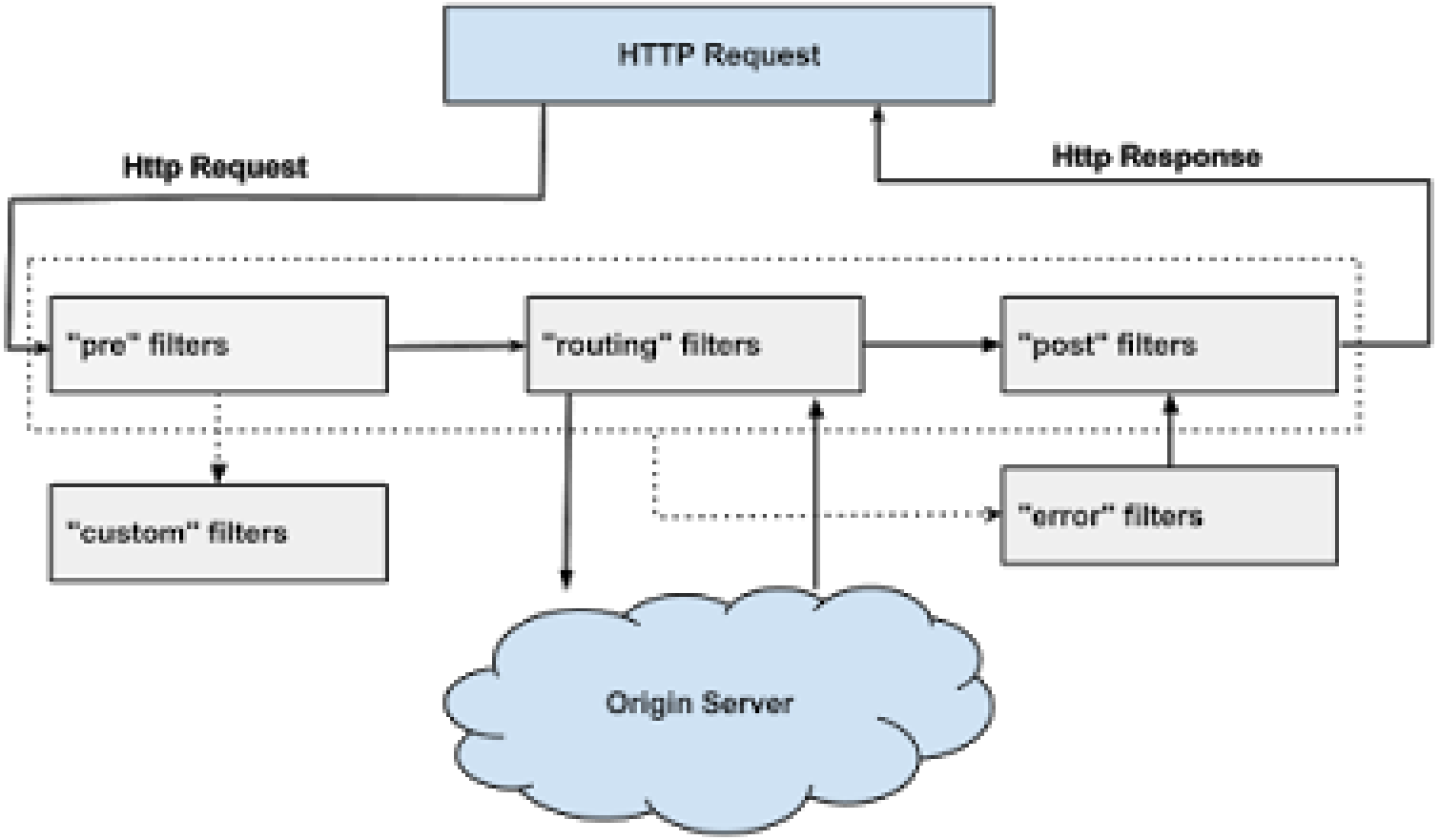
Microservices



What is Zuul?

Microservices





Why did we build Zuul?

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

Zuul Components

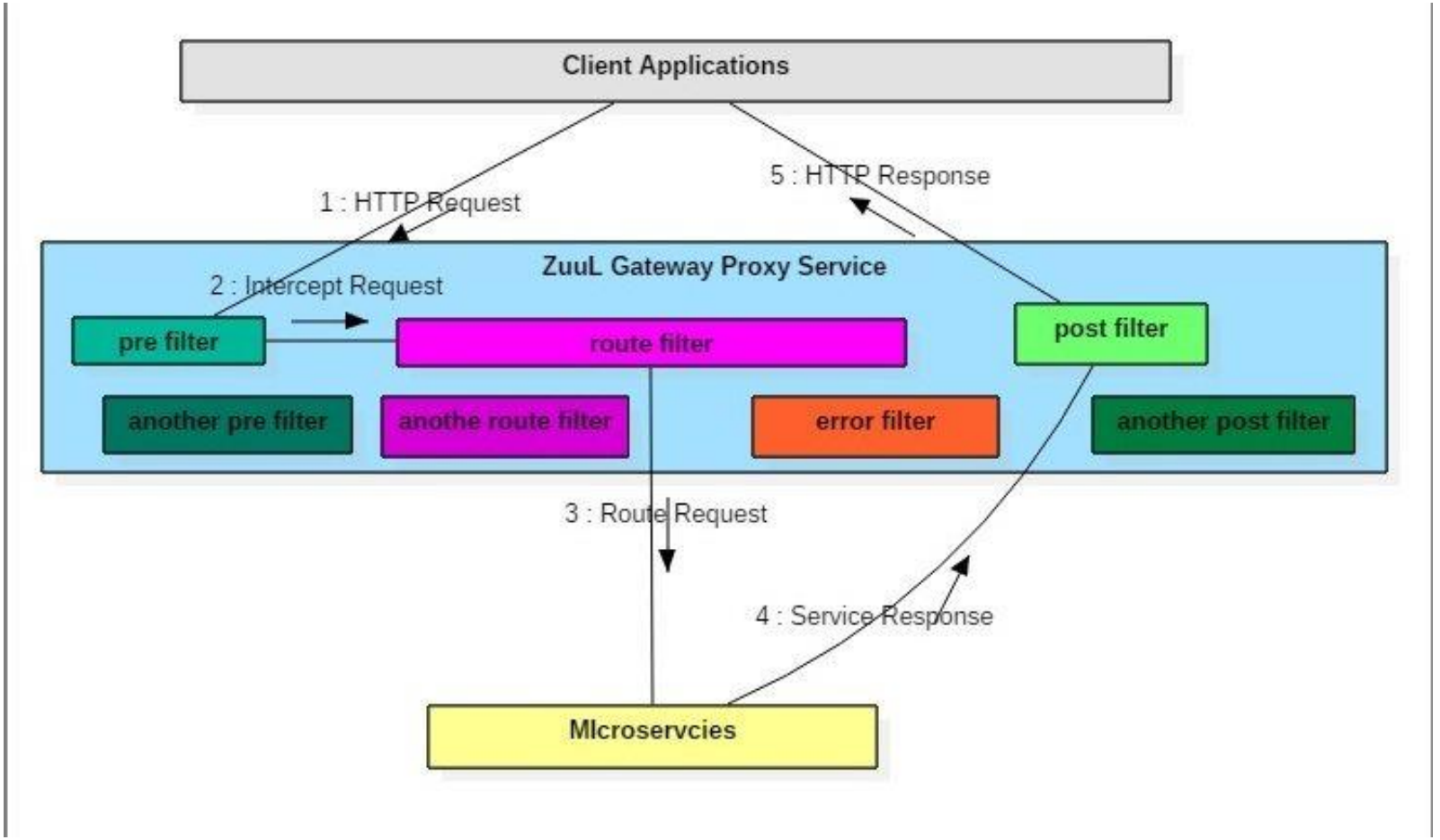
- Zuul has mainly four types of filters that enable us to intercept the traffic in different timeline of the request processing for any particular transaction.
- We can add any number of filters for a particular url pattern.
- **pre filters** – are invoked before the request is routed.
- **post filters**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

Zuul Components

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

Zuul Components

Microservices



Zuul Filters Responsibilities

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

Zuul Filters Responsibilities

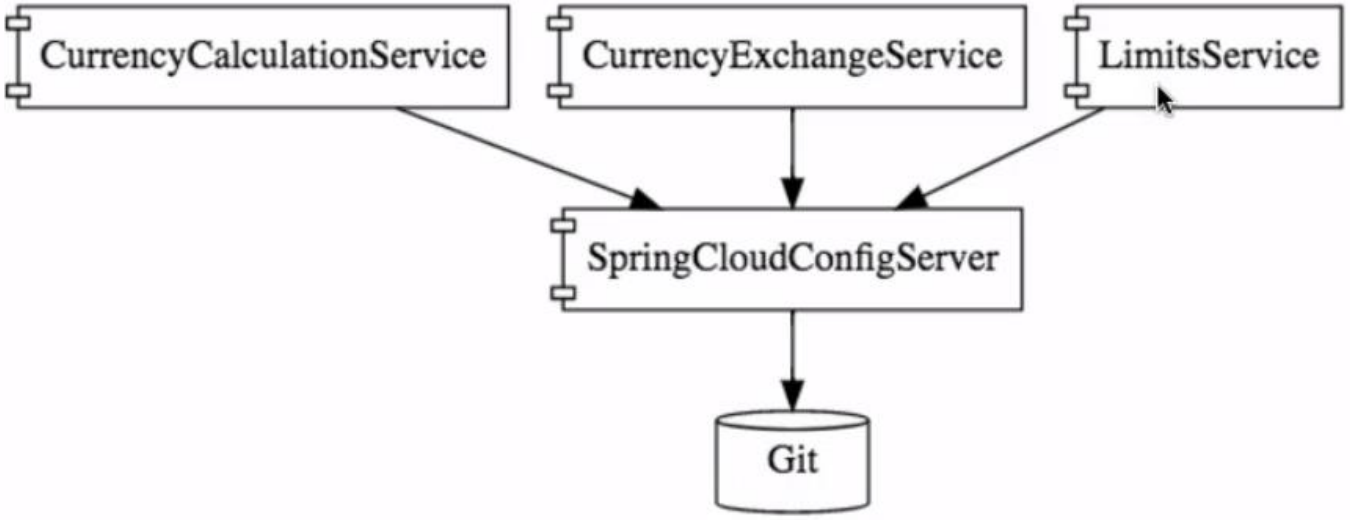
- **Dynamic Routing** can route requests to different backend clusters as needed.
- We can do **runtime stress testing** by gradually increasing the traffic to a new cluster in order to gauge performance in many scenarios e.g. cluster has new H/W and network setup or that has new version of production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

Zuul Filters Responsibilities

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

Spring Cloud Config Server with git

Microservices

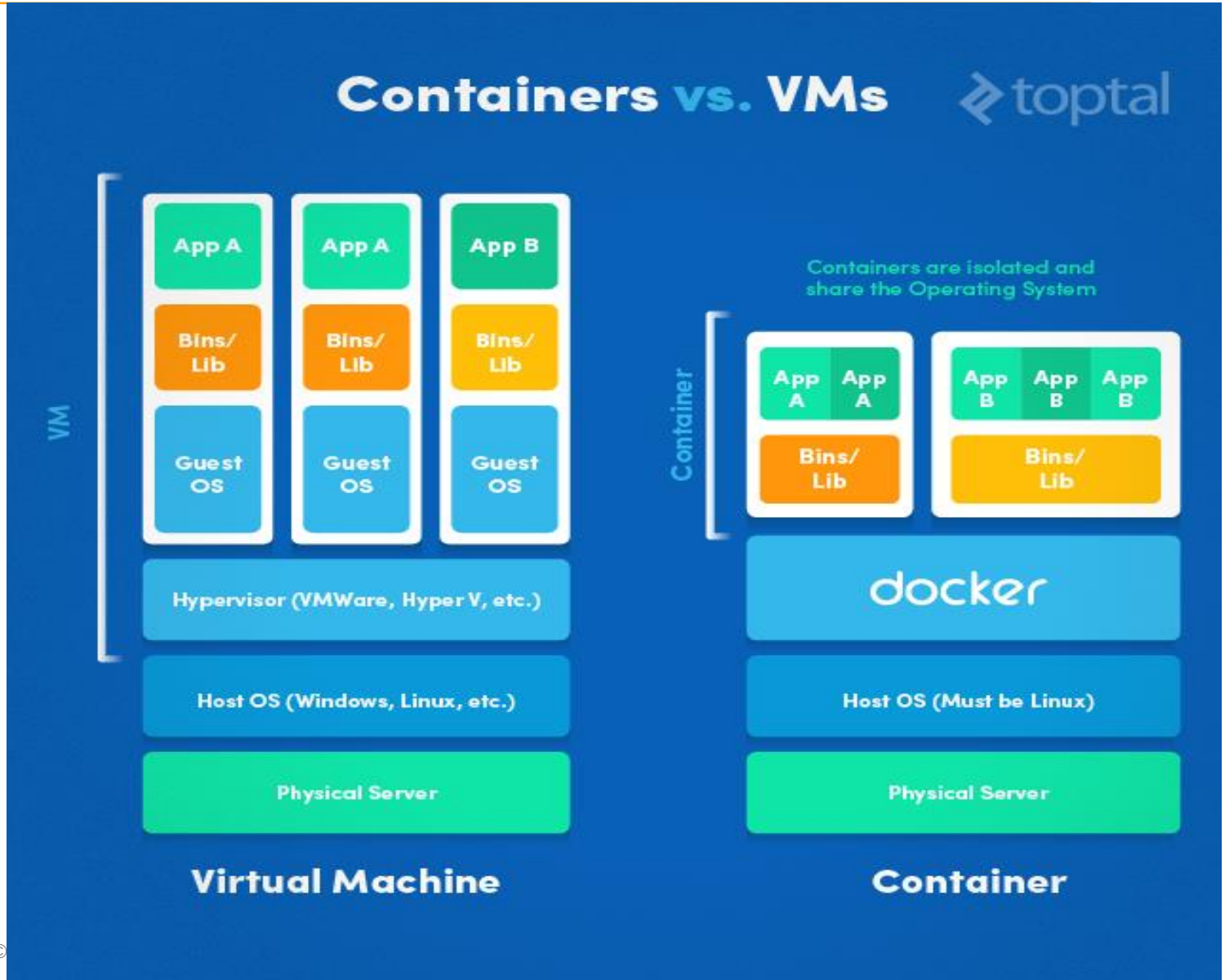


Spring Cloud Config Server

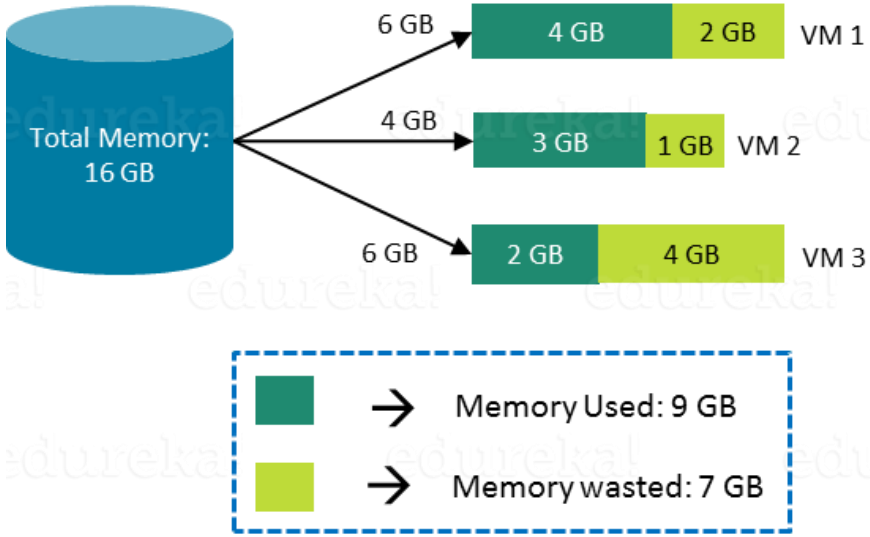
Docker

- 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



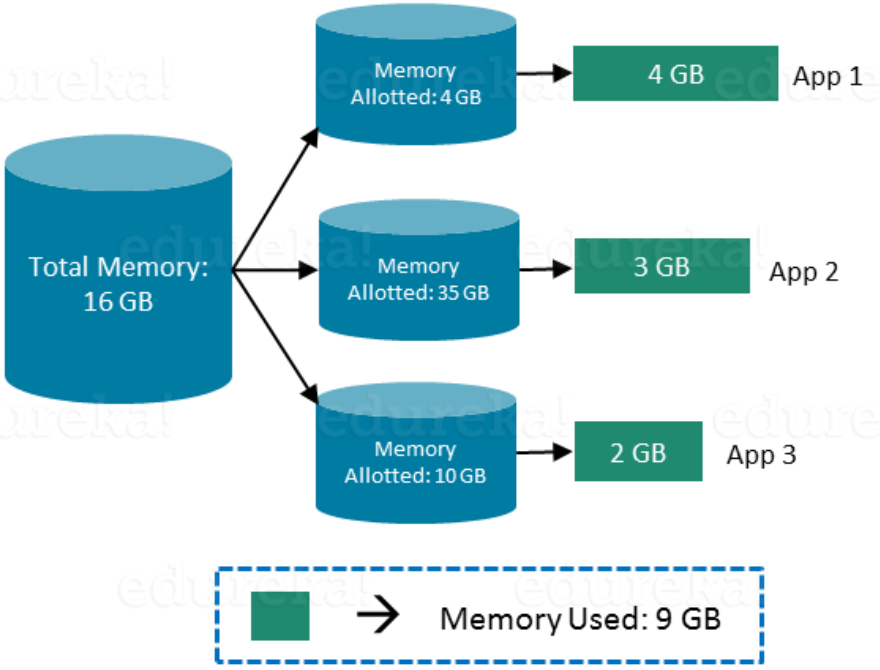
In case of Virtual Machines



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

In case of Docker

edureka!



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

What is Docker

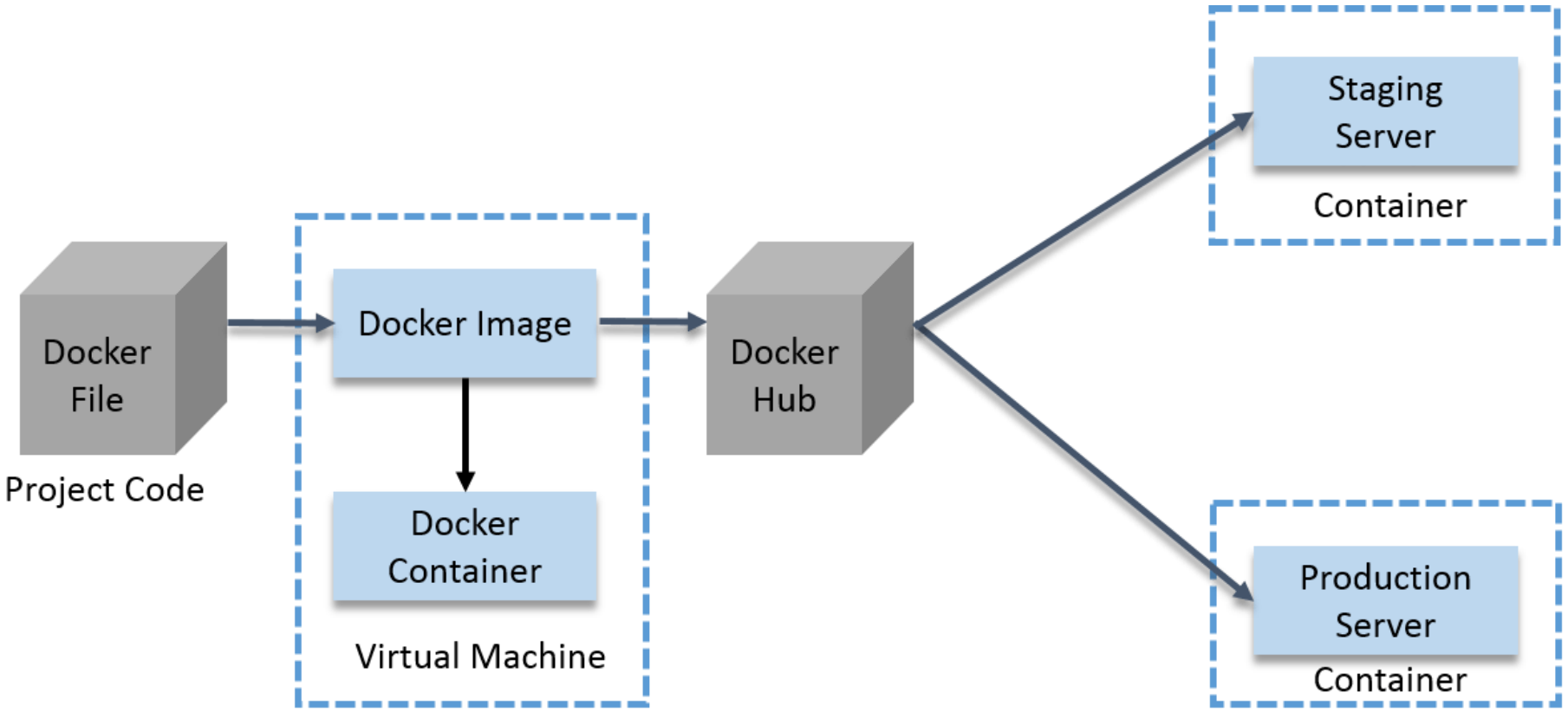
- What is Docker?
- At its heart, Docker is software which lets you create an *image* and then run instances of that image in a *container*.
- Docker maintain a vast repository of images, called the Docker Hub which you can use as starting points or as free storage for your own images.
- You can install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

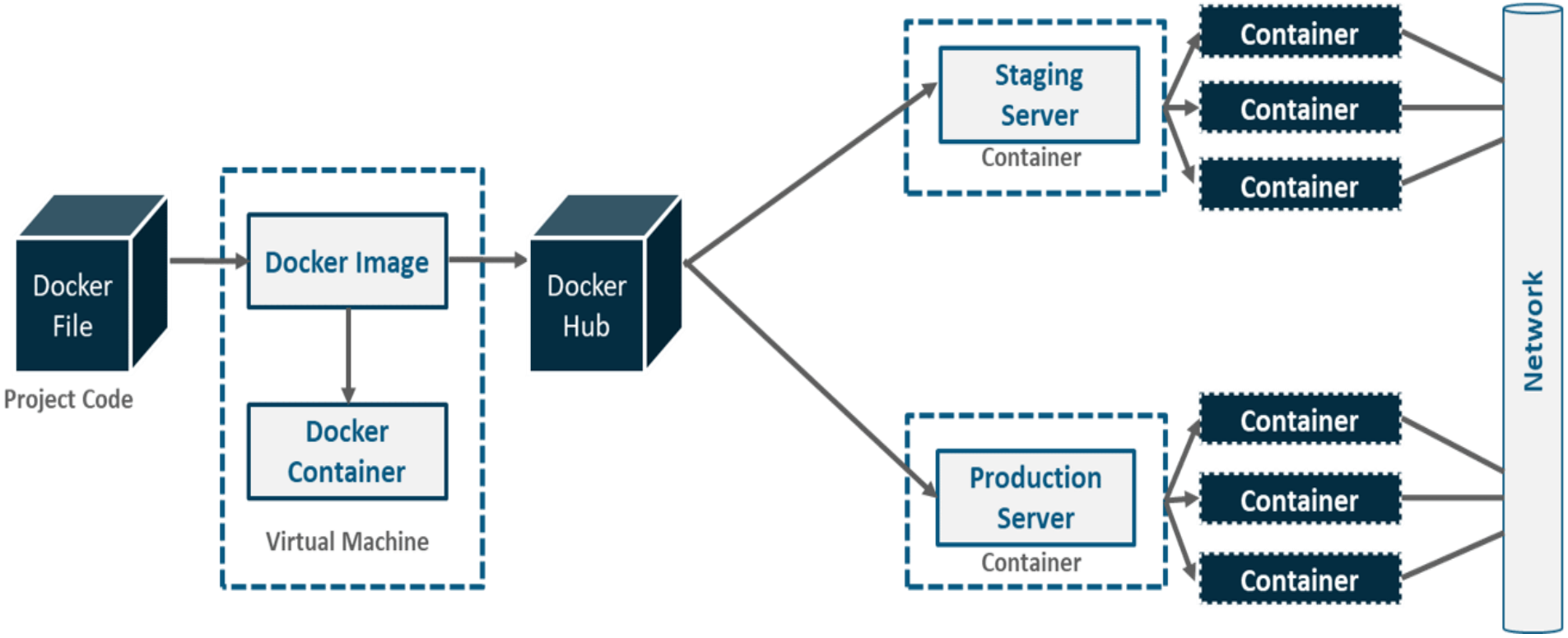
Difference between VM and Container

- 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.

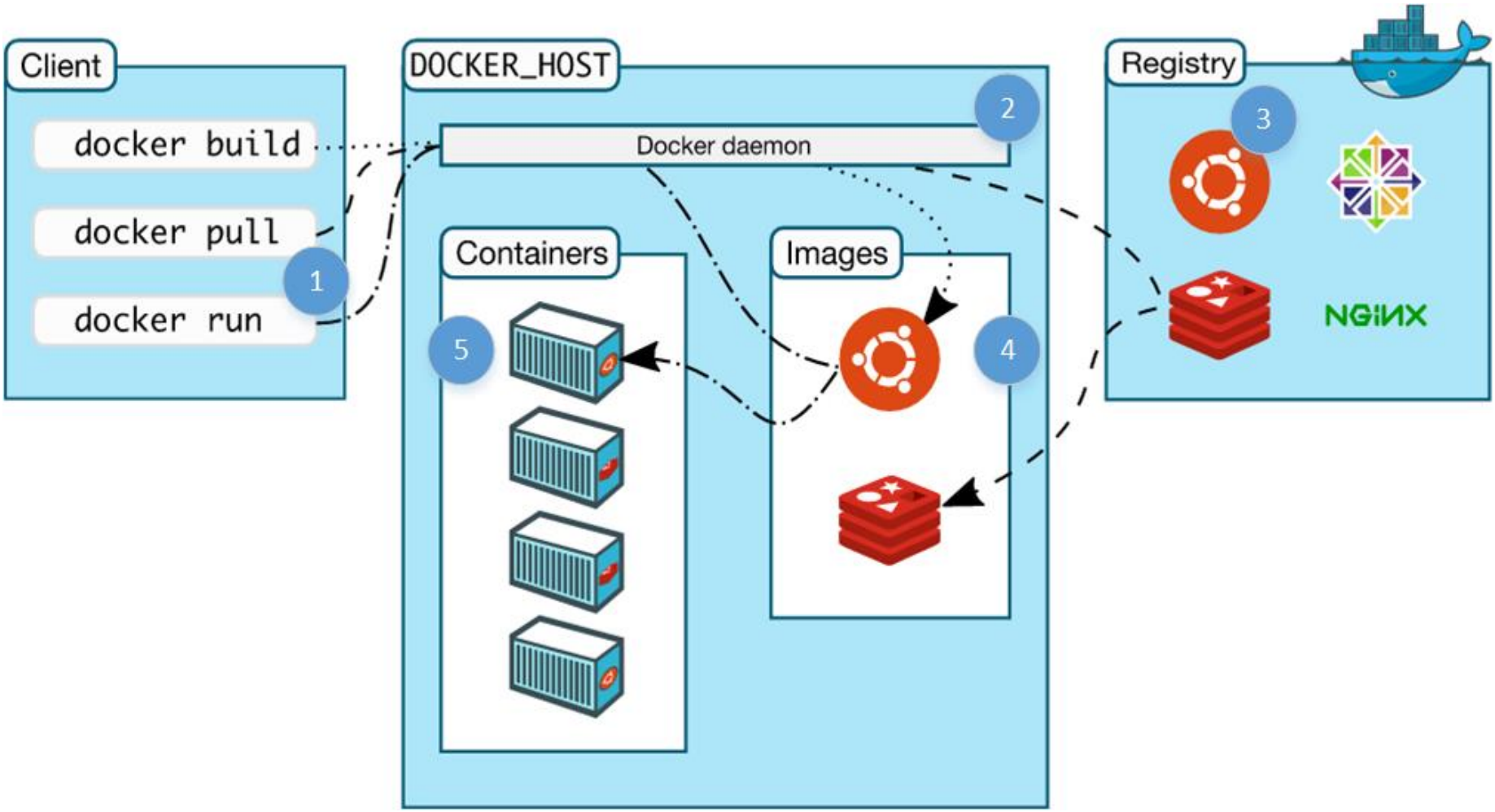
Dockerfile, Docker Image And Docker Container:

- 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.



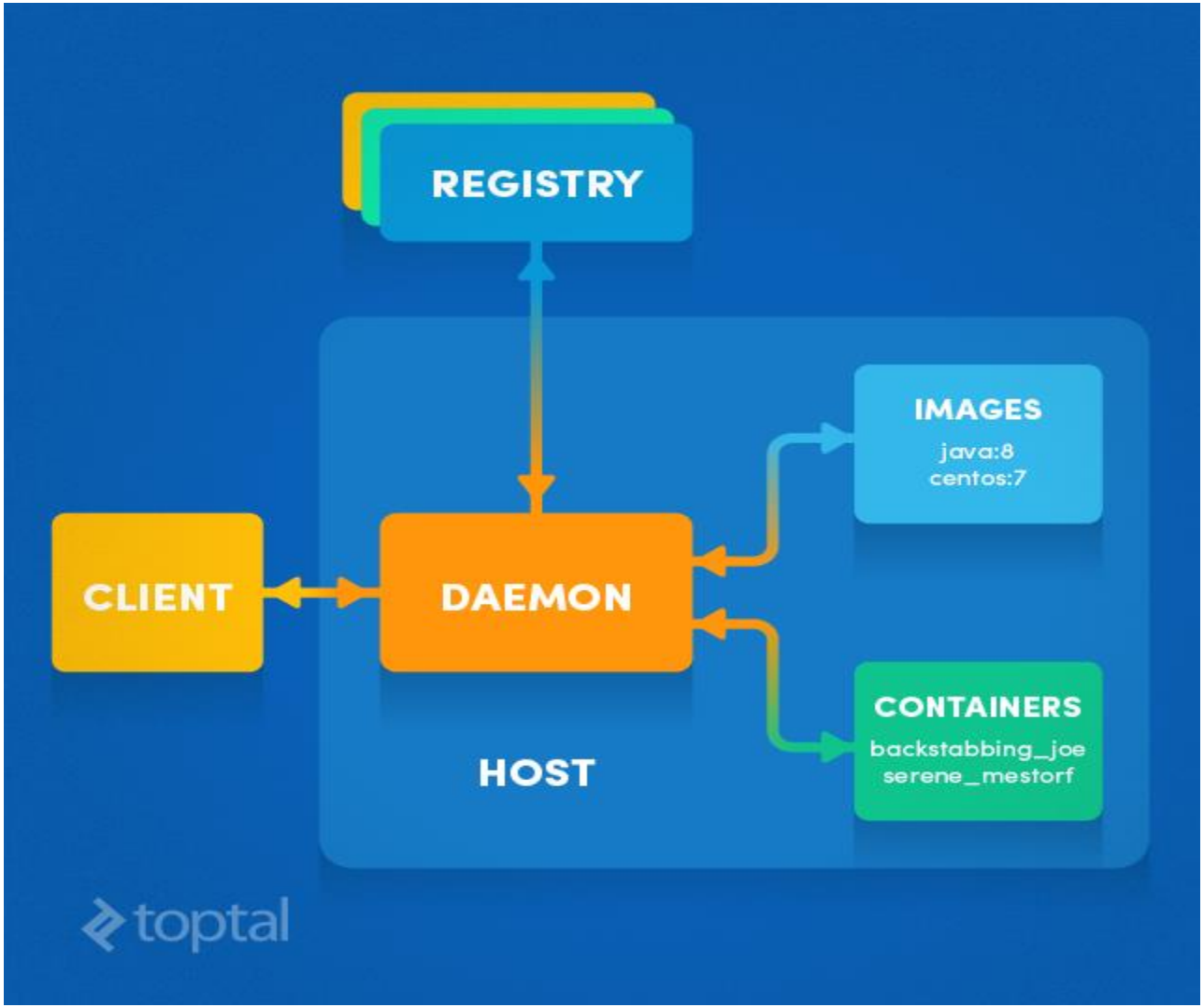


Docker



Docker Architecture

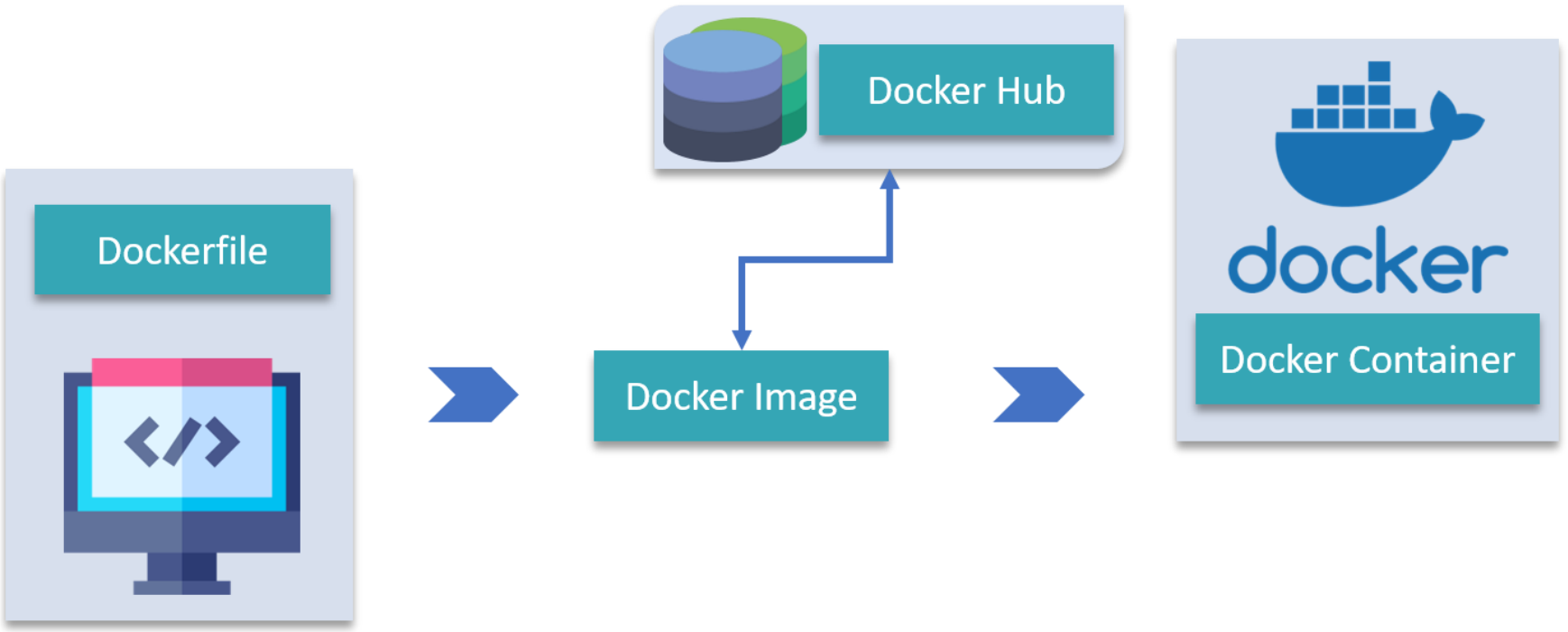
Microservices



Docker Hub:

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

Docker Hub



Docker Architecture

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

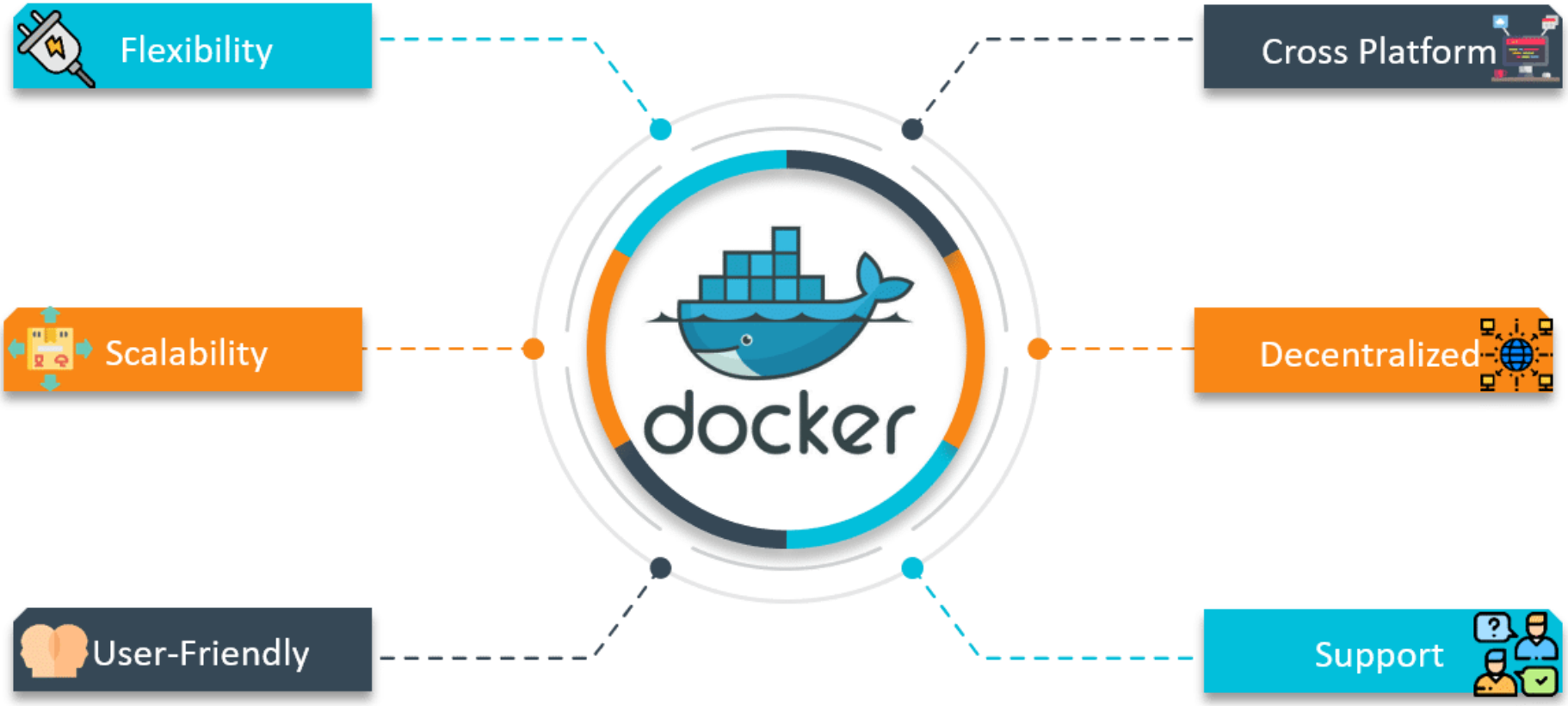
Docker Architecture

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

Docker Architecture

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

Goals of Docker Networking



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.

Creating Test Database Server

- docker run --name olddb -e MYSQL_ROOT_PASSWORD=vignesh –e MYSQL_DATABASE=virtusa_2018db –e MYSQL_USER=root -p 3306:3306 mysql/mysql-server:5.5
- docker run tells the engine we want to run an image (the image comes at the end, mysql:vlatest
- --name db names this container db.
- -d detach - i.e. run the container in the background.
- -e MYSQL_ROOT_PASSWORD=123 the -e is the flag that tells docker we want to provide an environment variable. The variable following it is what the MySQL image checks for setting the default root password.
- -p 3306:3306 tells the engine that we want to map the port 3306 from inside the container to out port 3306.

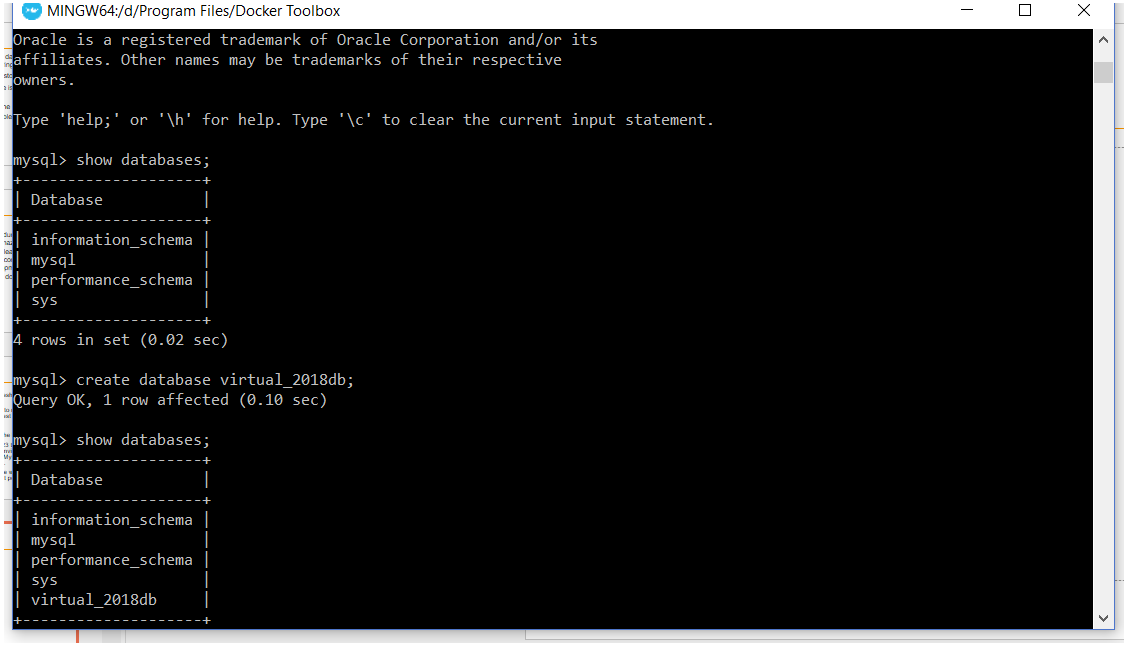
MINGW64:/d/Program Files/Docker Toolbox

```
Warning: Unable to load '/usr/share/zoneinfo/iso3166.tab' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/leapseconds' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/tzdata.zi' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/zone.tab' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/zone1970.tab' as time zone. Skipping it.
[Entrypoint] Not creating mysql user. MYSQL_USER and MYSQL_PASSWORD must be specified to create a mysql user.

[Entrypoint] ignoring /docker-entrypoint-initdb.d/*

[Entrypoint] Server shut down

[Entrypoint] MySQL init process done. Ready for start up.

[Entrypoint] Starting MySQL 5.5.62-1.1.8
181120 19:49:03 [Note] mysqld (mysqld 5.5.62) starting as process 1 ...

Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker kill $(docker ps -q)
304981f38447

Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker kill $(docker ps -a -q)
Error response from daemon: Cannot kill container: 304981f38447: Container 304981f384473c7caf3235c2fa998da2f0240efff01a1ed0d00381c25180b1dd is not running

Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker rm $(docker ps -a -q)
304981f38447

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

Creating Test Database Server

Microservices

- Docker ps
- docker exec -it olddb /bin/bash
- mysql –u root –p



Springboot Docker


```
<plugin>
    <groupId>com.spotify</groupId>
    <artifactId>docker-maven-plugin</artifactId>
    <version>VERSION GOES HERE</version>
    <configuration>
      <imageName>example</imageName>
      <baseImage>java</baseImage>
      <entryPoint>["java", "-jar", "/${project.build.finalName}.jar"]</entryPoint>
      <!-- copy the service's jar file from target into the root directory of the image -->
      <resources>
        <resource>
          <targetPath>/</targetPath>
          <directory>${project.build.directory}</directory>
          <include>${project.build.finalName}.jar</include>
        </resource>
      </resources>
    </configuration>
  </plugin>
```

Springboot Docker

- Spring boot Project Build

- Goal package docker:build
- mvn clean install dockerfile:build
- Docker folder created and docker image name example deployed in docker machine
- Check docker images
- Docker ps

Springboot Docker

- Docker run –t –name sampleapp –link v1db –p 8080:8080 example:latest

- Example:latest --- image name
- --name – container reference
- --link – db ref in container

docker run -h 192.168.99.100 -p 7070:7070 -t my-repo/example --name my-repo-image:latest

What Is Kubernetes? An Introduction To Container Orchestration Tool

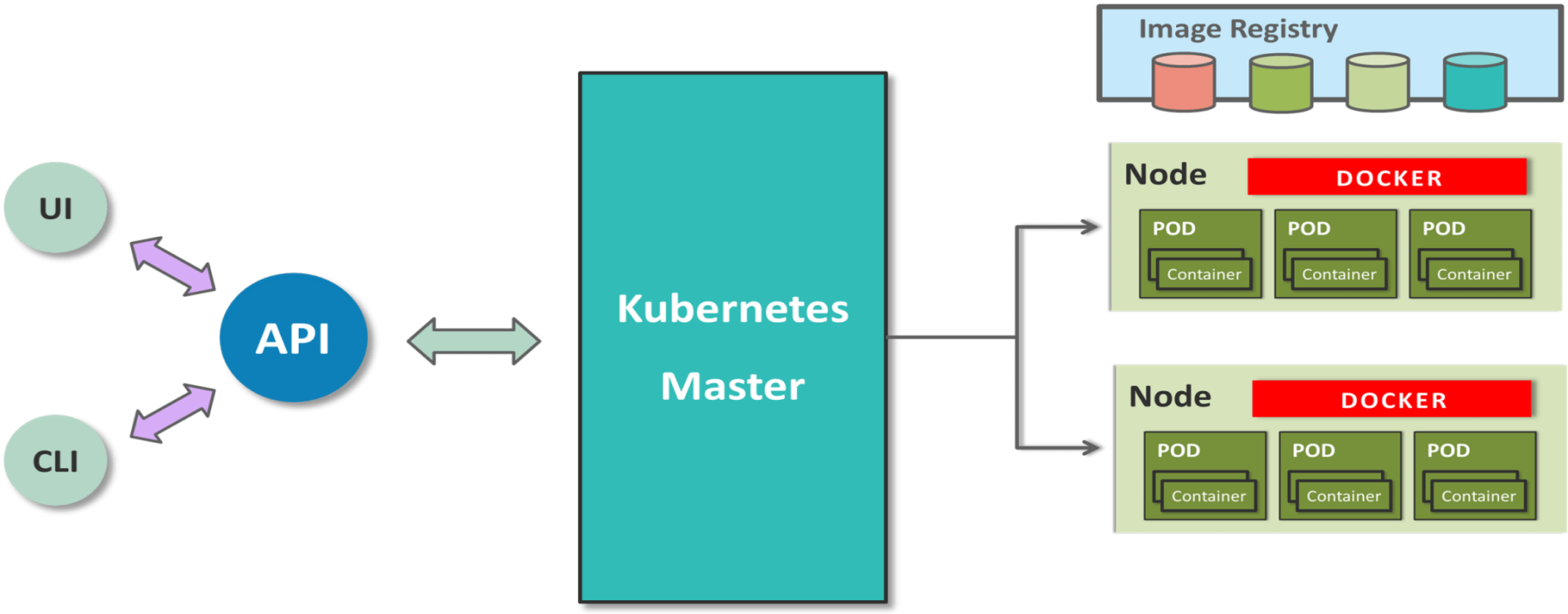
- **What Is Kubernetes?**
- Kubernetes is an open-source container management (orchestration) tool. It's container management responsibilities include container deployment, scaling & descaling of containers & container load balancing.
- Download from
- https://github.com/kubernetes/minikube
- Under curl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

What Is Kubernetes? An Introduction To Container Orchestration Tool

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

Features Of Kubernetes

1. Automatic Binpacking
2. Service Discovery & Load Balancing
3. Storage Orchestration
4. Self Healing
5. Secret & Configuration Management
6. Batch Execution
7. Horizontal Scaling
8. Automatic Rollbacks & Rollouts



Minikube –p cluster1 dashboard

Hello Minikube - Kubernetes | Overview - Kubernetes Dashboar

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

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

kubernetes | Search | + 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

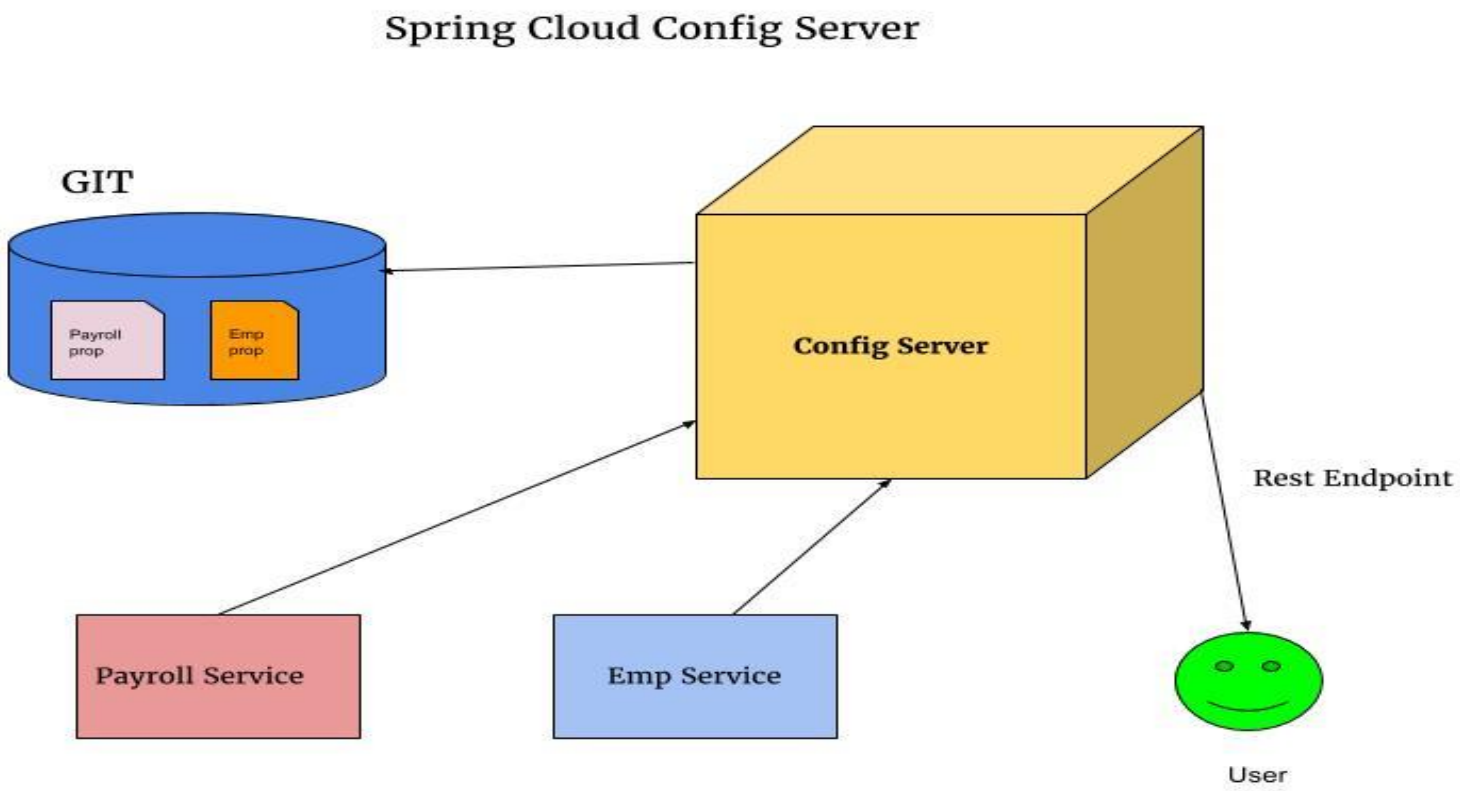
Secrets

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

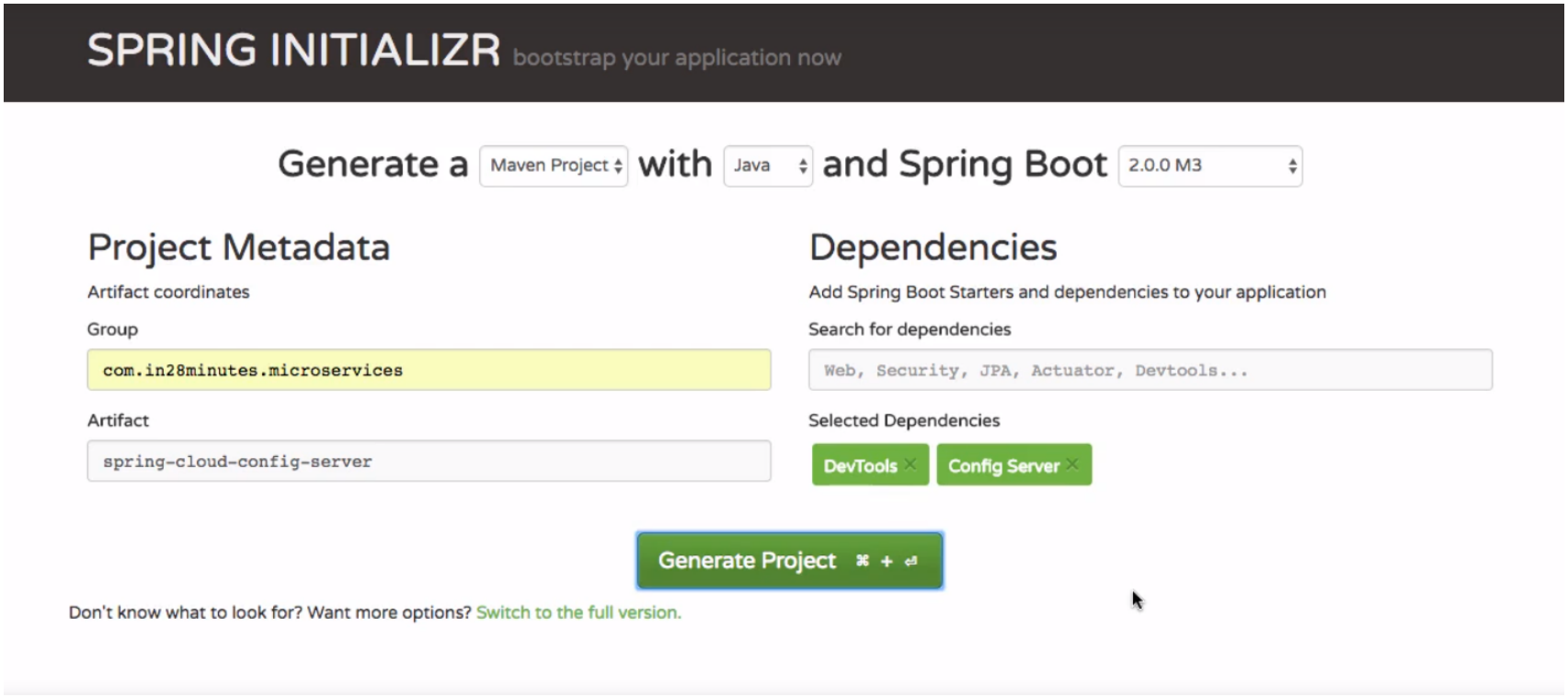
Type here to search | ENG | 22:50 | 20/02/2019

Spring Cloud Config Server with git

Microservices



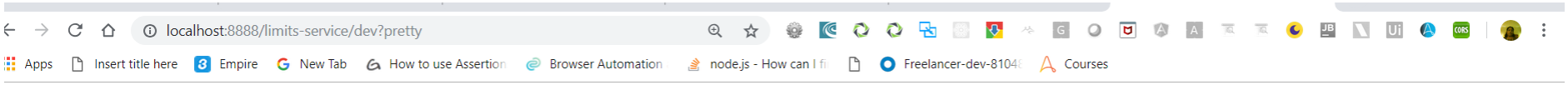
Spring Cloud Config Server with git



Spring Cloud Config Server with git


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

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

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

    git commit --amend --reset-author

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

Spring Cloud Config Server with git




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

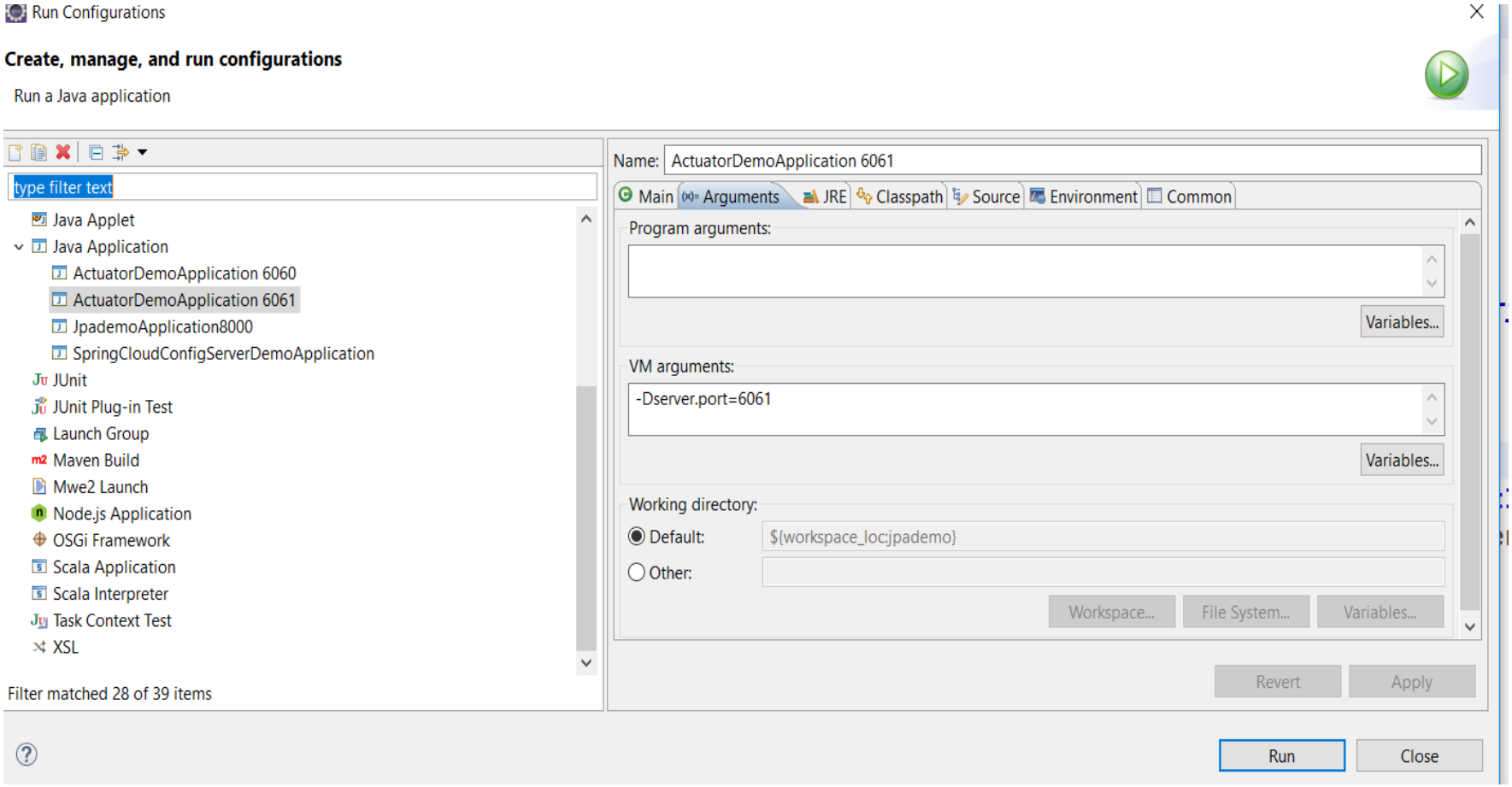
Micro service With Multiple Port


```
18 //@JsonIgnoreProperties(value= { assetValue })
19
20 //dynamic filter
21 @JsonFilter(value="AssetFilter")
22 public class Asset {
23     @Id
24     @GeneratedValue(strategy=GenerationType.IDENTITY)
25     @Column(name="Asset_Id")
26     private int assetId;
27     @Column(name="Asset_Name",nullable=false,length=50)
28     private String assetName;
29     @Column(name="Asset_Value")
30     private long assetValue;
31     @Transient
32     private int port;
33
34     public int getPort() {
35         return port;
36     }
37     public void setPort(int port) {
38         this.port = port;
39     }
40     public int getAssetId() {
41         return assetId;
```

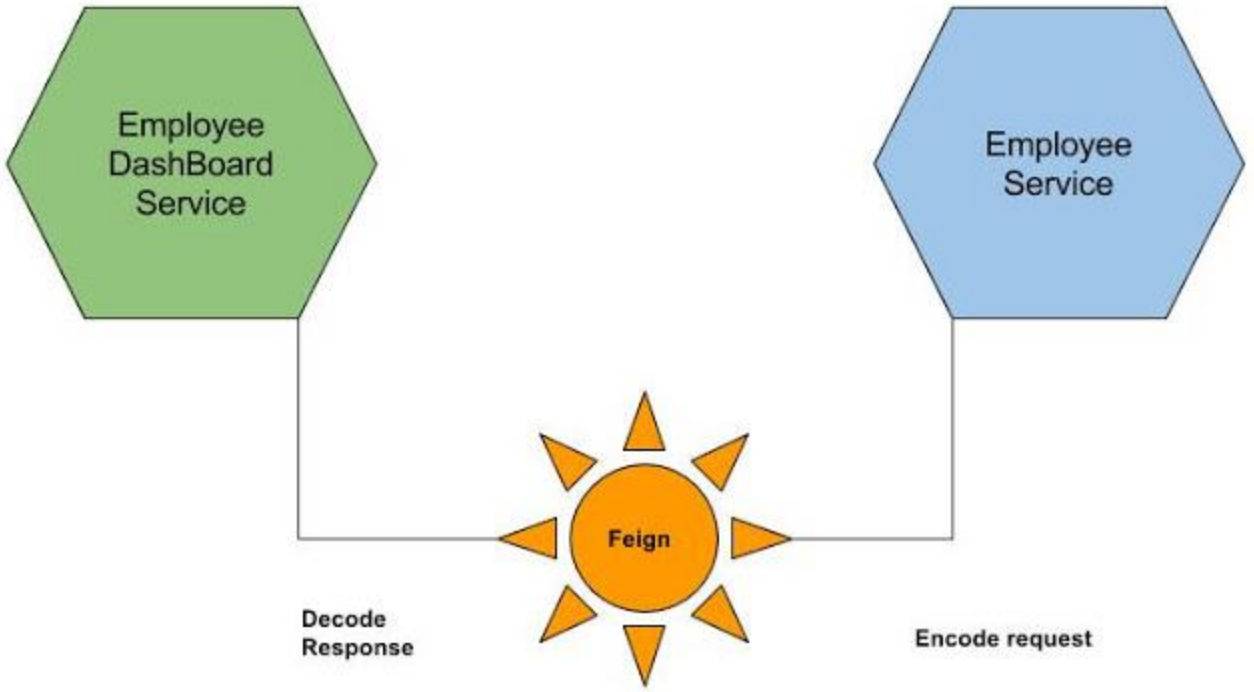
Micro service With Multiple Port


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

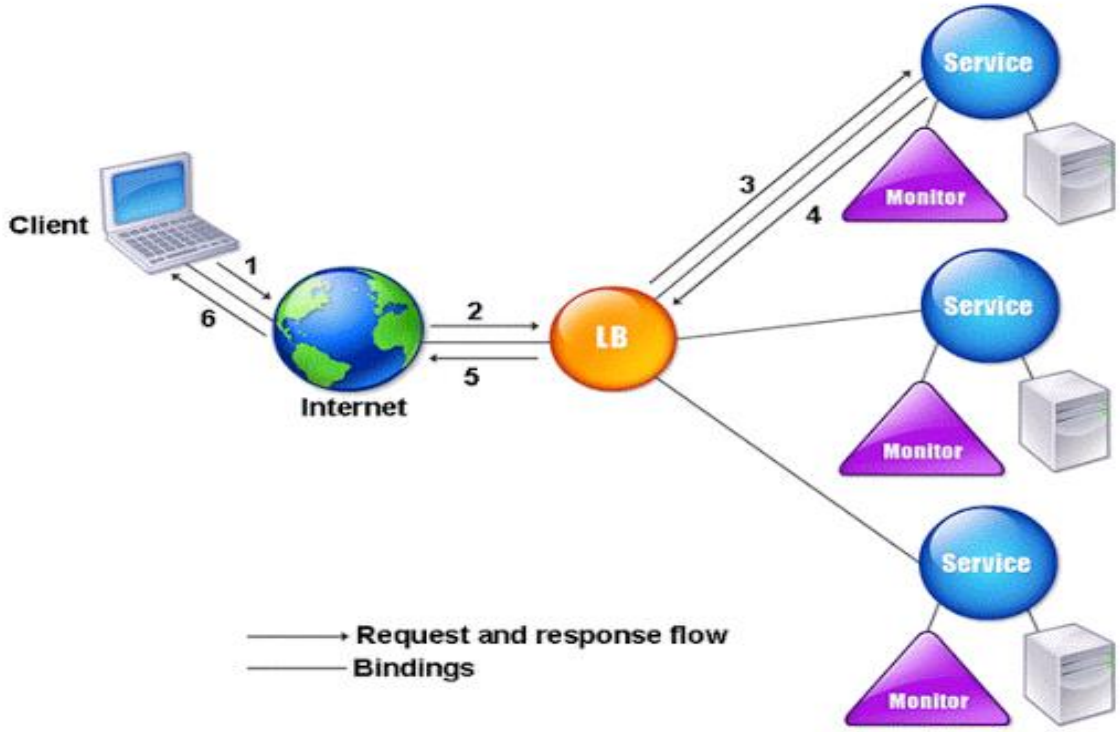
Micro service With Multiple Port

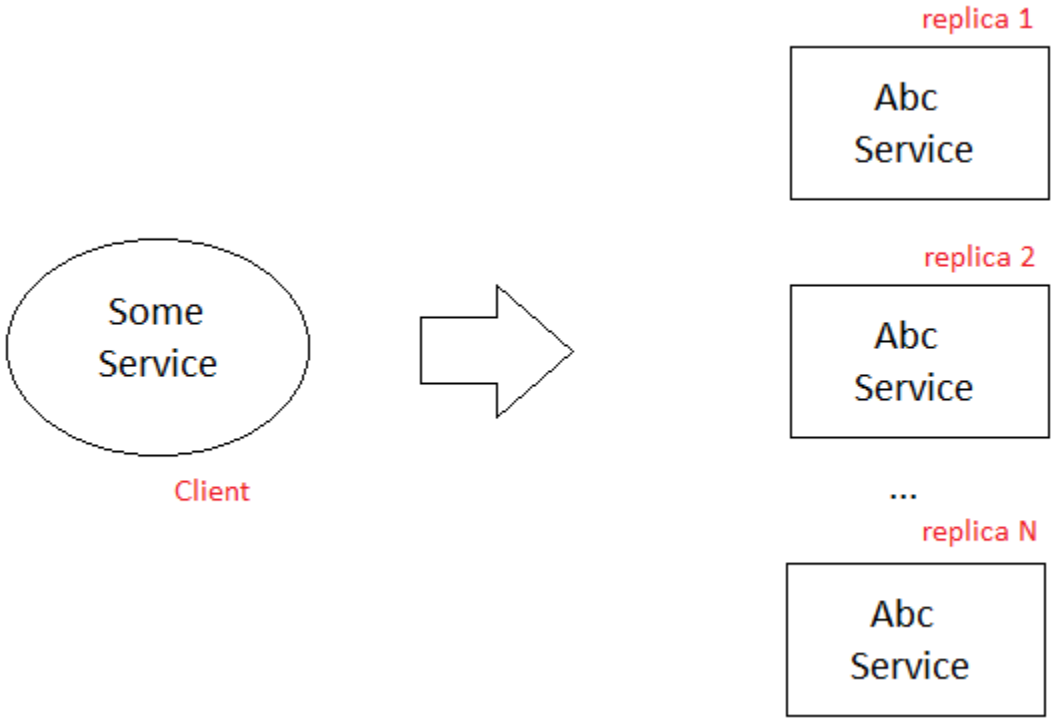


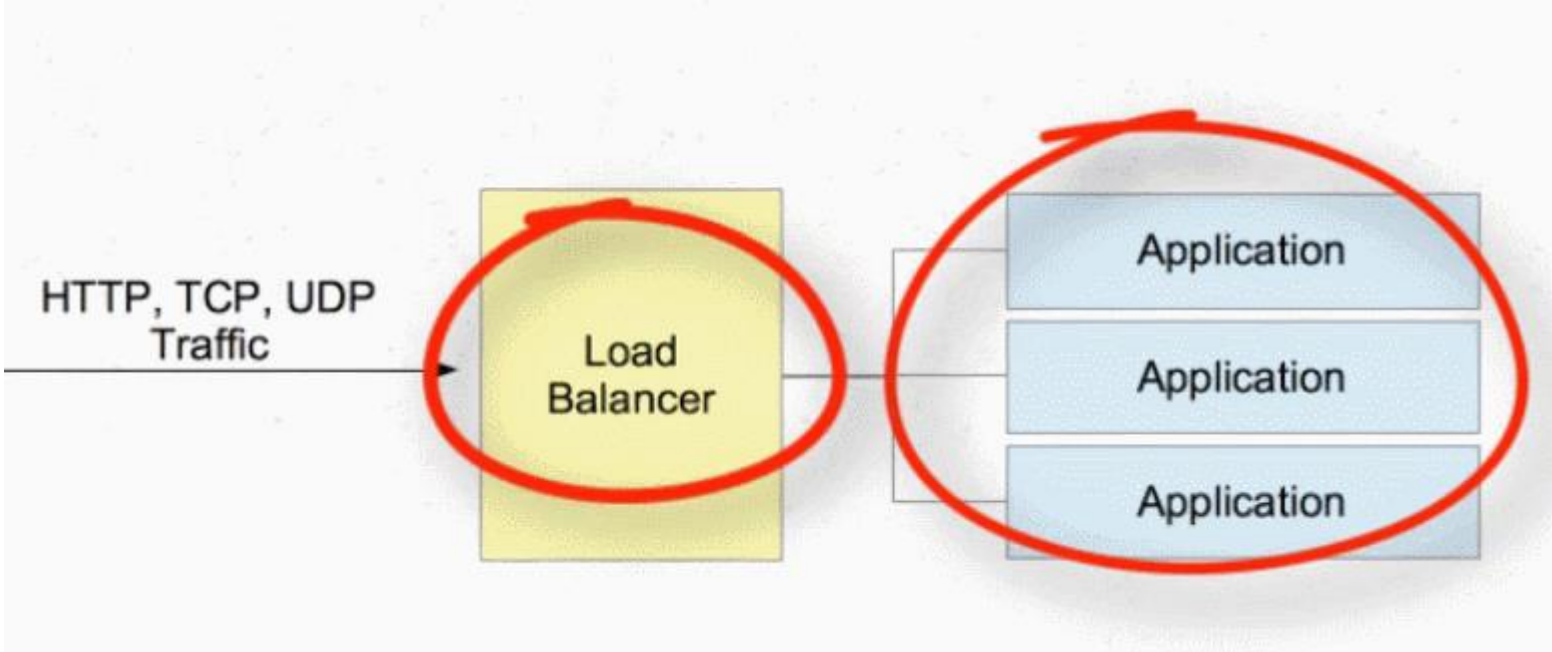
Feign Client

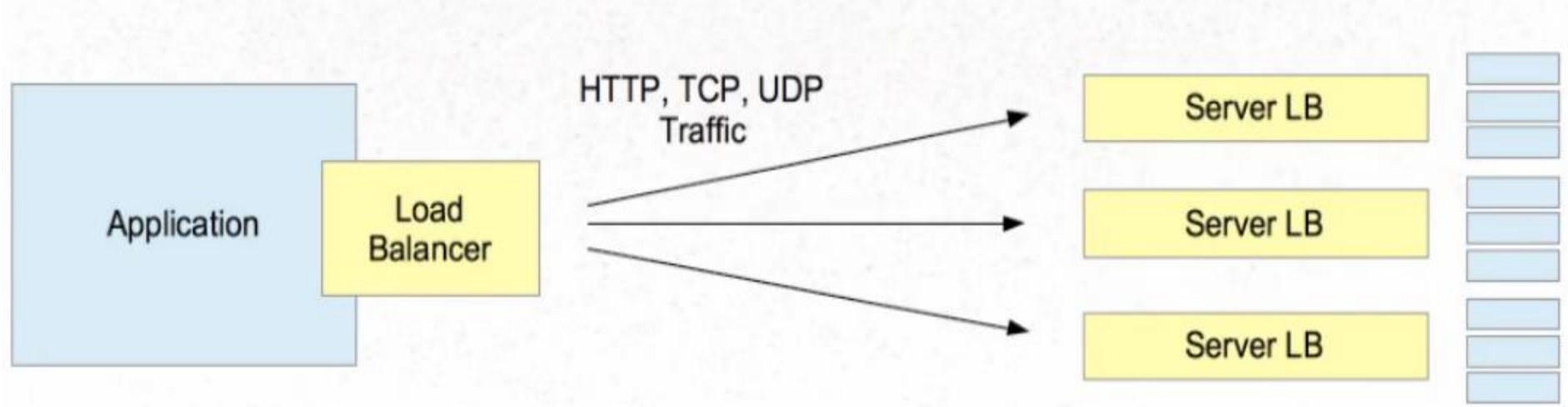


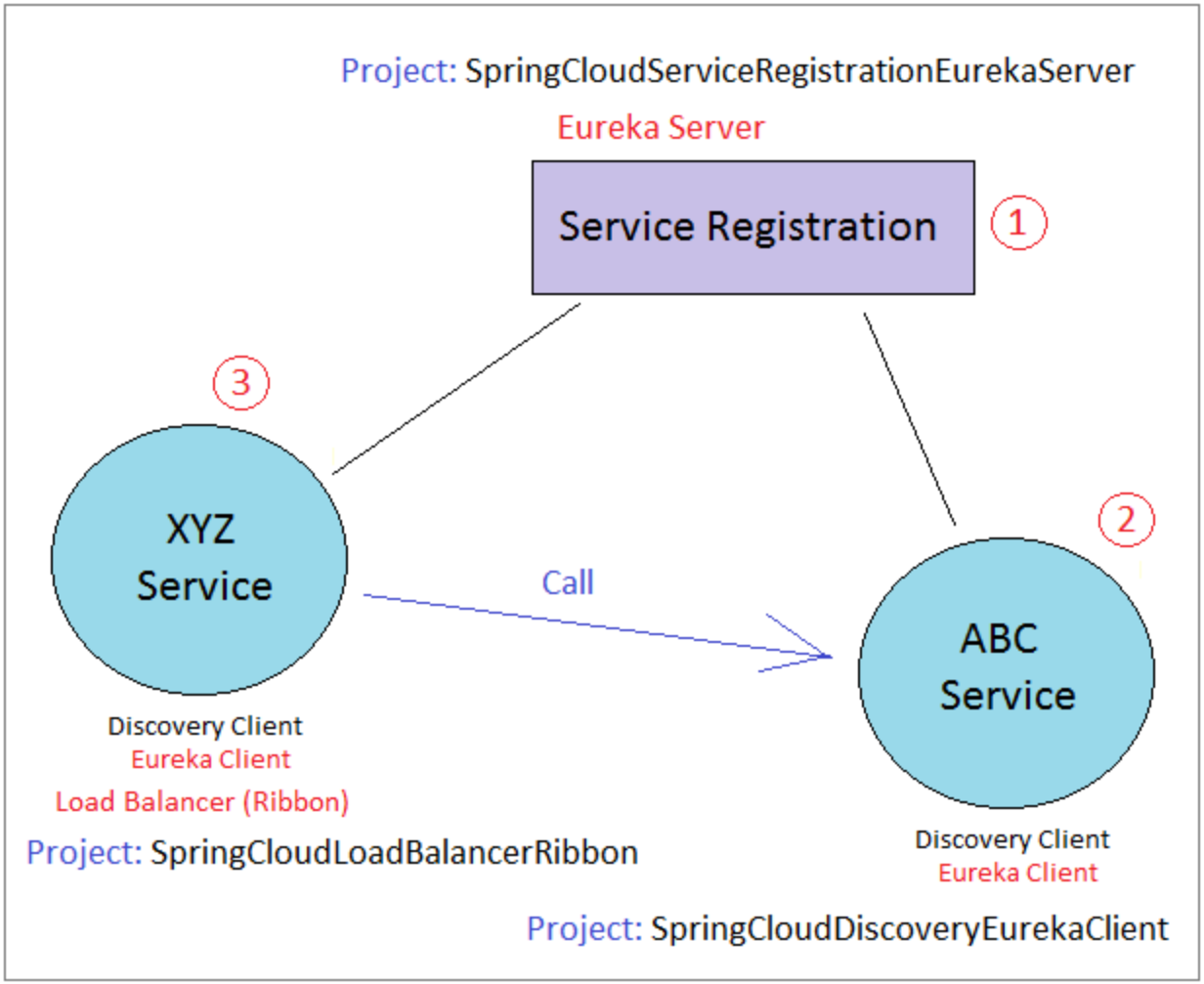
RIBBON LOAD BALANCER

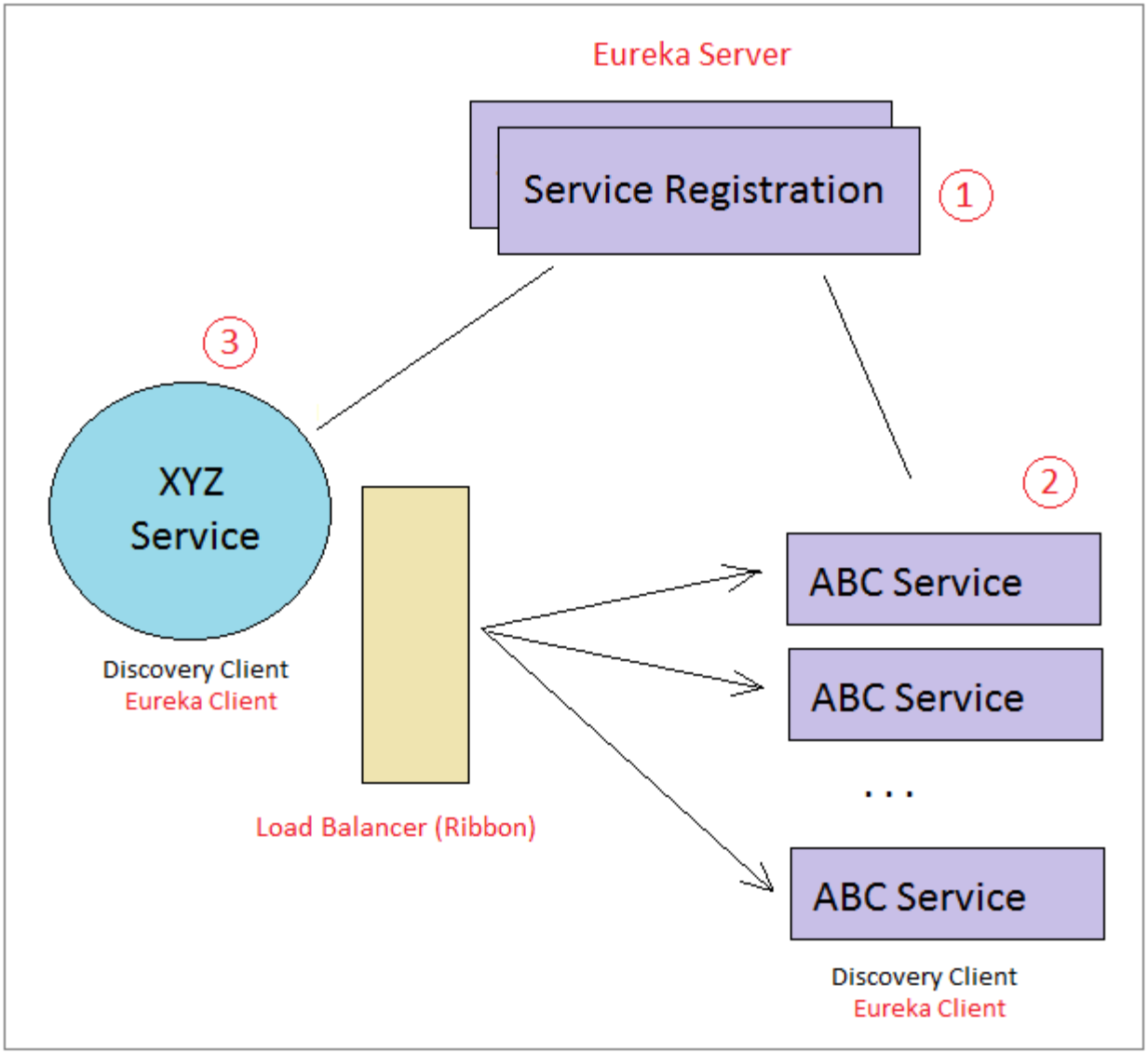












RIBBON LOAD BALANCER

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

RIBBON LOAD BALANCER

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

RIBBON LOAD BALANCER

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

RIBBON LOAD BALANCER

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

RIBBON LOAD BALANCER

The properties file (sample-client.properties)

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

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

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

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

#
```

RIBBON LOAD BALANCER

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

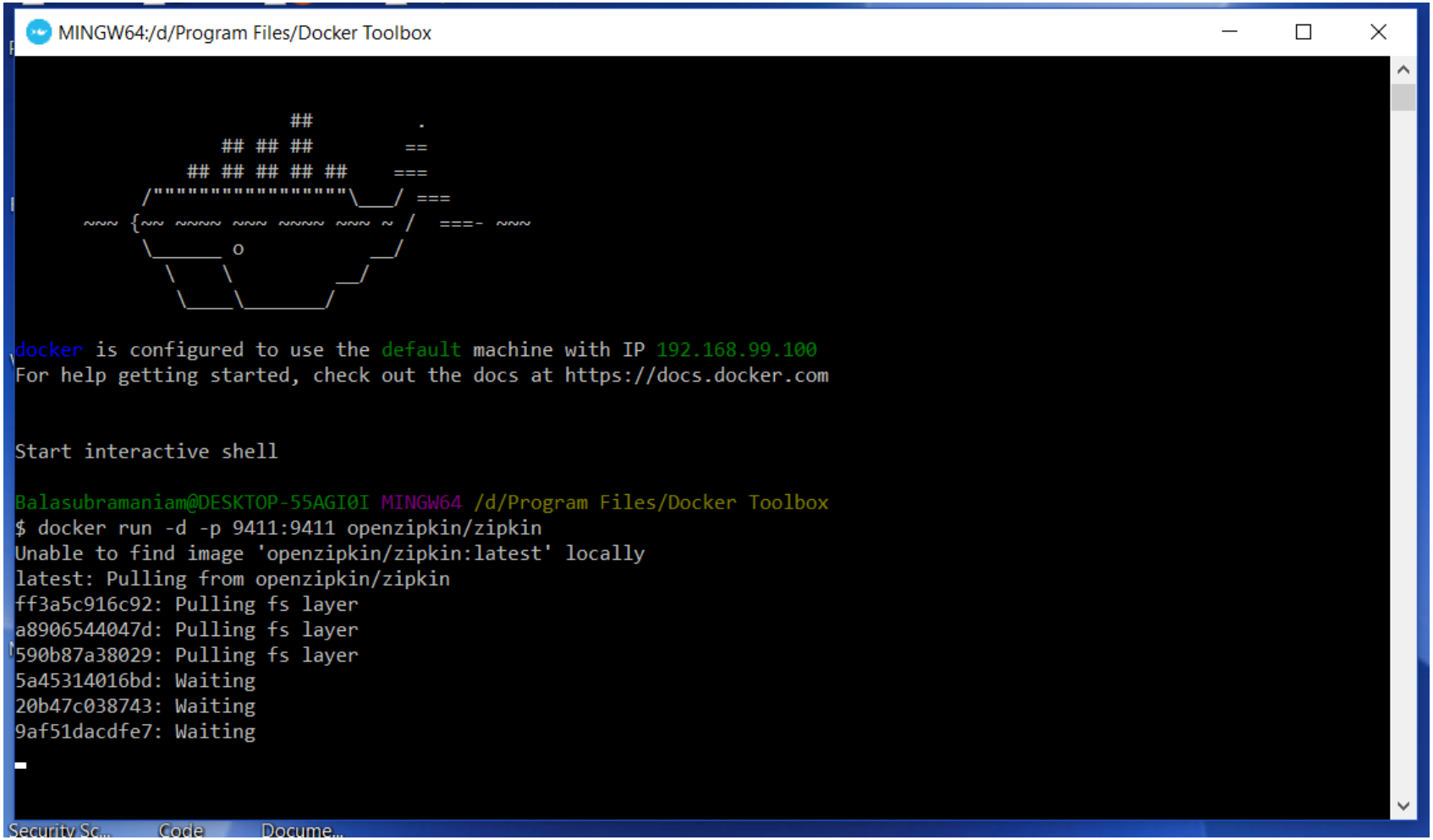
Read timeout used by Apache HttpClient
sample-client.ribbon.ReadTimeout=3000

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

API Gateway

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

Zipkin



AWS Lambda Serverless Environment

- **What is AWS Lambda?**
- Amazon explains, AWS Lambda (λ) as a ‘serverless’ compute service, meaning the developers, don’t have to worry about which AWS resources to launch, or how will they manage them, they just put the code on lambda and it runs, it’s that simple! It helps you to focus on core-competency i.e. App Building or the code.

AWS Lambda Serverless Environment

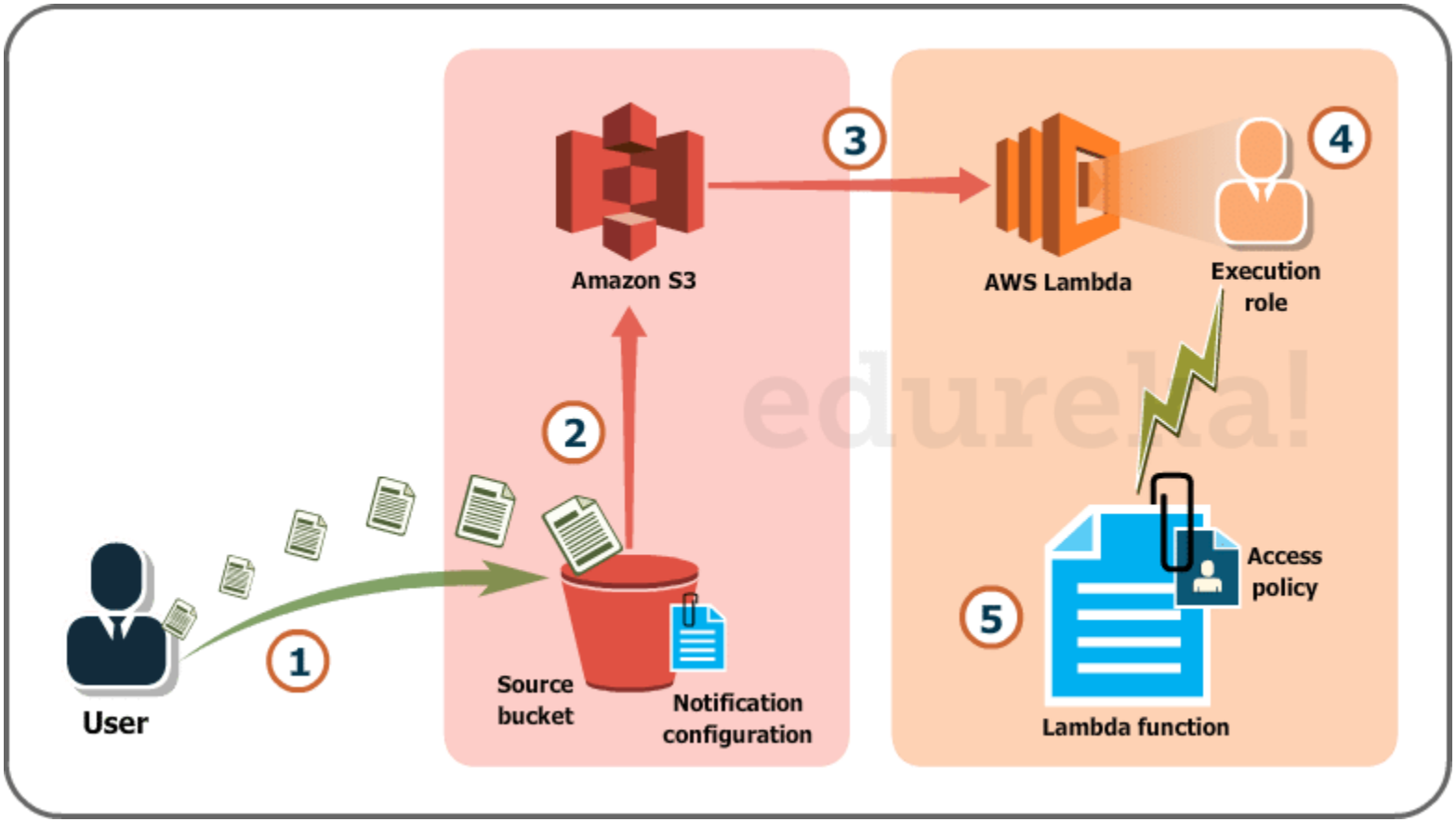
- **Where will I use AWS Lambda?**
- AWS Lambda executes your backend code, by automatically managing the AWS resources. When we say ‘manage’, it includes launching or terminating instances, health checkups, auto scaling, updating or patching new updates etc.

AWS Lambda Serverless Environment

- **So, how does it work?**
- The code that you want Lambda to run is known as a **Lambda function**. Now, as we know a function runs only when it is called, right? Here, **Event Source** is the entity which triggers a Lambda Function, and then the task is executed.

AWS Lambda Serverless Environment

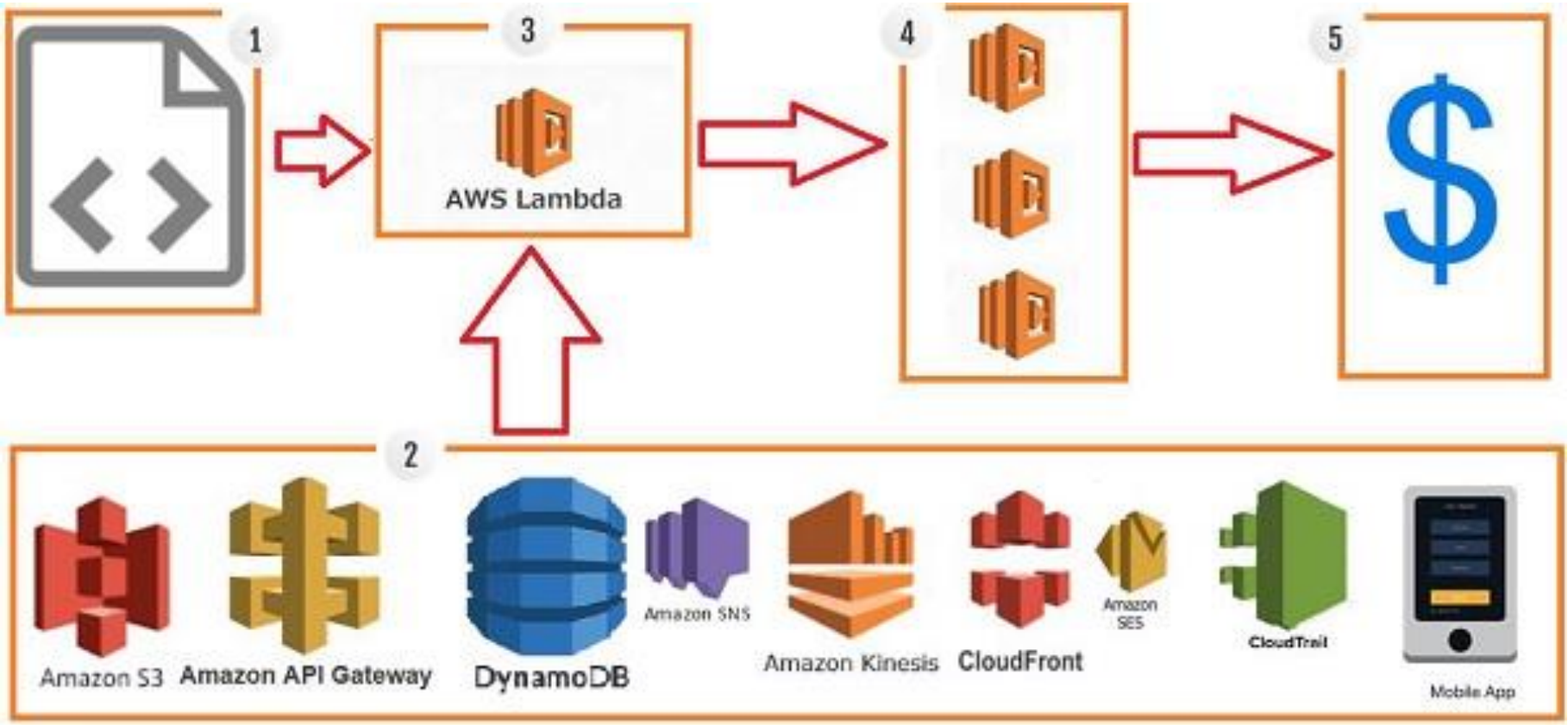
- **Let's take an example to understand it more clearly.**
- Suppose you have an app for image uploading. Now when you upload an image, there are a lot of tasks involved before storing it, such as resizing, applying filters, compression etc.
- So, this task of uploading of an image can be defined as an **Event Source** or the 'trigger' that will call the Lambda Function, and then all these tasks can be executed via the Lambda function.



AWS Lambda Serverless Environment

- The whole process, as you can see in the diagram, is divided into 5 steps, let's understand each one of them.
- User uploads an image (object) to a source bucket in S3 which has notification attached to it, for Lambda.
- The notification is read by S3 and it decides where to send that notification.
- S3 sends the notification to Lambda, this notification acts as an invoke call of the lambda function.
- Execution role in Lambda can be defined by using IAM (Identity and Access Management) to give access permission for the AWS resources, for this example here it would be S3.
- Finally, it invokes the desired lambda function which works on the object which has been uploaded to the S3 bucket.

How AWS Lambda Works?



AWS Lambda Steps

- **Step 1 –** Upload AWS lambda code in any of languages AWS lambda supports, that is NodeJS, Java, Python, C# and Go.
- **Step 2 –** These are few AWS services on which AWS lambda can be triggered.
- **Step 3 –** AWS Lambda which has the upload code and the event details on which the trigger has occurred. For example, event from Amazon S3, Amazon API Gateway, Dynamo dB, Amazon SNS, Amazon Kinesis, CloudFront, Amazon SES, CloudTrail, mobile app etc.

AWS Lambda Steps

- **Step 4 –** Executes AWS Lambda Code only when triggered by AWS services under the scenarios such as –
  - User uploads files in S3 bucket
  - http get/post endpoint URL is hit
  - data is added/updated/deleted in dynamo dB tables
  - push notification
  - data streams collection
  - hosting of website
  - email sending
  - mobile app, etc.
- **Step 5 –** Remember that AWS charges only when the AWS lambda code executes, and not otherwise.

Advantages of using AWS Lambda

- Ease of working with code(Infrastructure Burden Free)
- Log Provision
- Billing based on Usage
- Multi Language Support
- Ease of code authoring and deploying
-

Disadvantages of using AWS Lambda

- It is not suitable for small projects.
- You need to carefully analyze your code and decide the memory and timeout. Incase if your function needs more time than what is allocated, it will get terminated as per the timeout specified on it and the code will not be fully executed.
- Since AWS Lambda relies completely on AWS for the infrastructure, you cannot install anything additional software if your code demands it.

Events that Trigger AWS Lambda

- Entry into a S3 object
- Insertion, updation and deletion of data in Dynamo DB table
- Push notifications from SNS
- GET/POST calls to API Gateway
- Headers modification at viewer or origin request/response in CloudFront
- Log entries in AWS Kinesis data stream
- Log history in CloudTrail

Use Cases of AWS Lambda

- S3 Object and AWS Lambda
- Amazon S3 passes the event details to AWS Lambda when there is any file upload in S3. The details of the file upload or deletion of file or moving of file is passed to the AWS Lambda.
- The code in AWS Lambda can take the necessary step for when it receives the event details. For Example creating thumbnail of the image inserted into S3.

Use Cases of AWS Lambda

- DynamoDB and AWS Lambda
- DynamoDB can trigger AWS Lambda when there is data added, updated and deleted in the table. AWS Lambda event has all the details of the AWS DynamoDB table about the insert /update or delete.

- API Gateway and AWS Lambda
- API Gateway can trigger AWS Lambda on GET/POST methods. We can create a form and share details with API Gateway endpoint and use it with AWS Lambda for further processing, for Example, making an entry of the data in DynamoDB table.

Use Cases of AWS Lambda

- SNS and AWS Lambda
- SNS is used for push notification, sending SMS etc. We can trigger AWS lambda when there is any push notification happening in SNS. We can also send SMS to the phone number from AWS Lambda when it receives the trigger.

Use Cases of AWS Lambda

- Scheduled Events and AWS Lambda
- Scheduled Events can be used for cron jobs. It can trigger AWS Lambda to carry out the task at regular time pattern.
- CloudTrail and AWS Lambda
- CloudTrail can be helpful in monitoring the logs on the account. We can use AWS Lambda to further process the CloudTrail logs .

Use Cases of AWS Lambda

- Kinesis and AWS Lambda
- Kinesis is used to capture/store real time tracking data coming from website clicks, logs, social media feeds and a trigger to AWS Lambda can do additional processing on this logs.

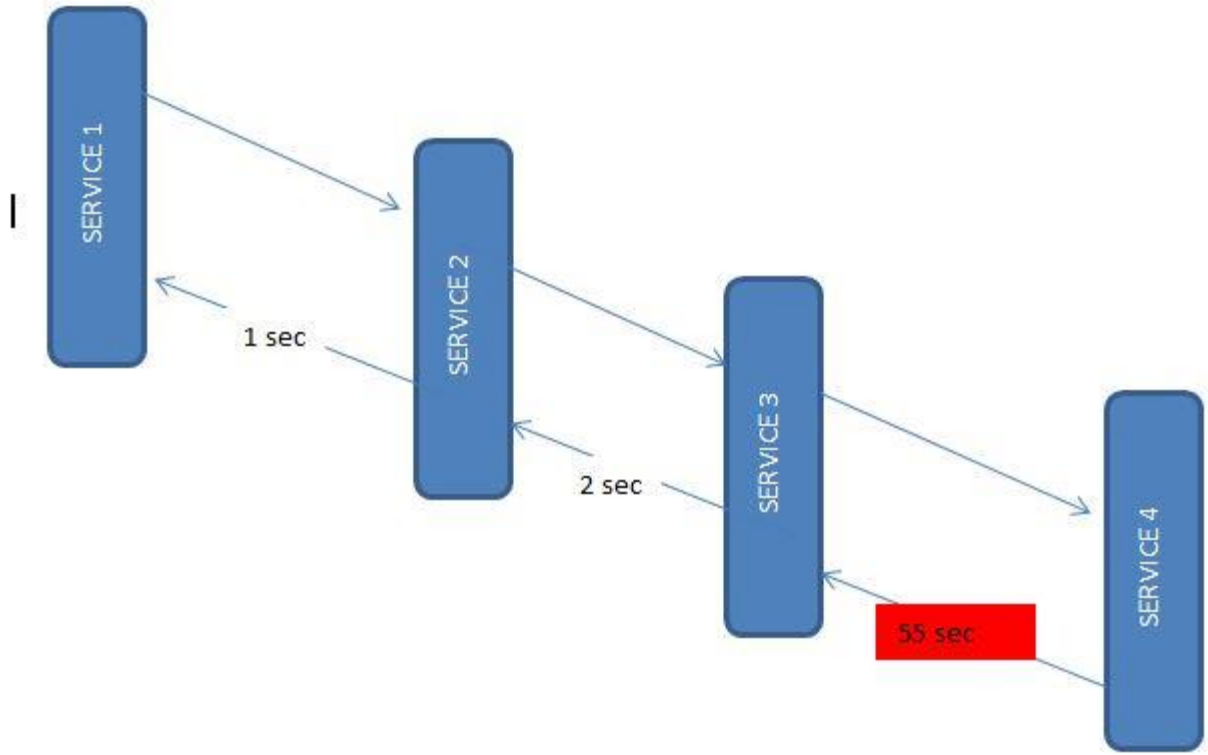
Zipkin server

- Download from
- download the latest Zipkin server from maven repository and run the executable jar file using below command.

- java -jar zipkin-server-2.12.1-exec.jar

- Once Zipkin is started, we can see the Web UI at http://localhost:9411/zipkin/

Microservices Interactions



Prometheus and grafana

- prometheus.exe --config.file="prometheus.yml" --web.listen-address=:9092

Questions

Module Summary

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