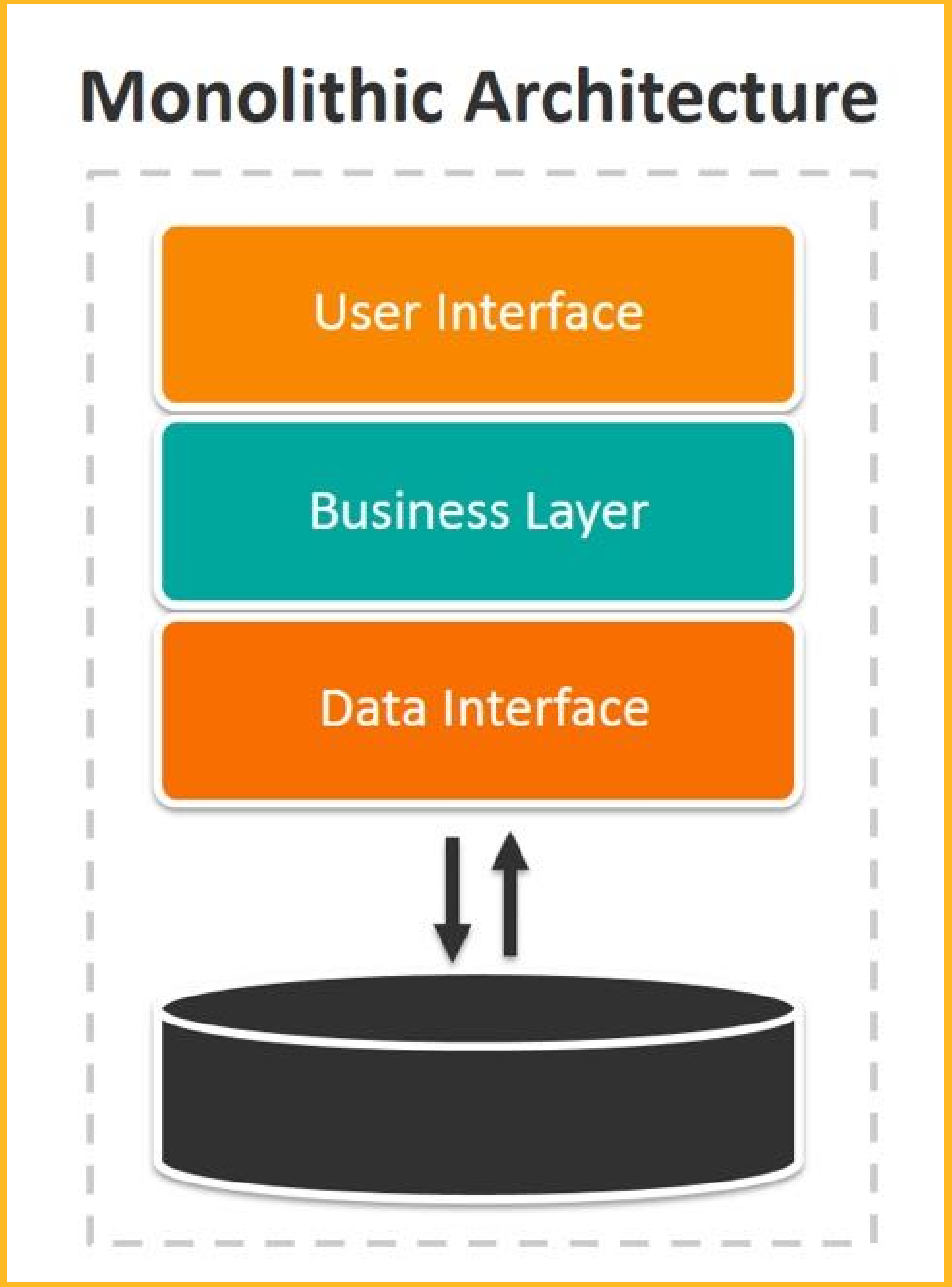
Checkpoint DevOps

wafa ben hamouda

Monolithic Architecture

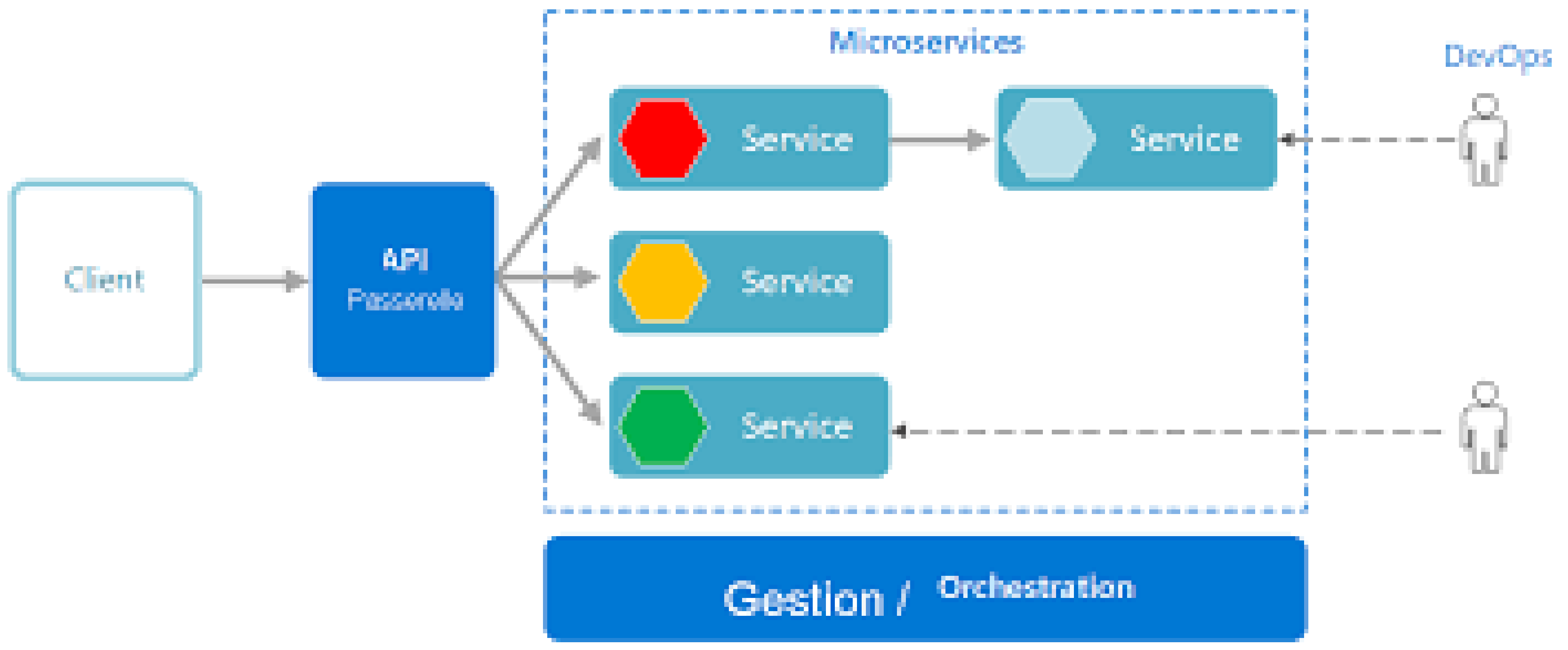


Monolithic architecture is a unified development model for software applications. It has three components:

Client-side user interface
Server-side application
Data interface

All three parts interact with a single database. Software built on this model operates with one base of code. As a result, whenever stakeholders want to make updates or changes, they access the same set of code. This can have ripple effects that impact user-side performance.

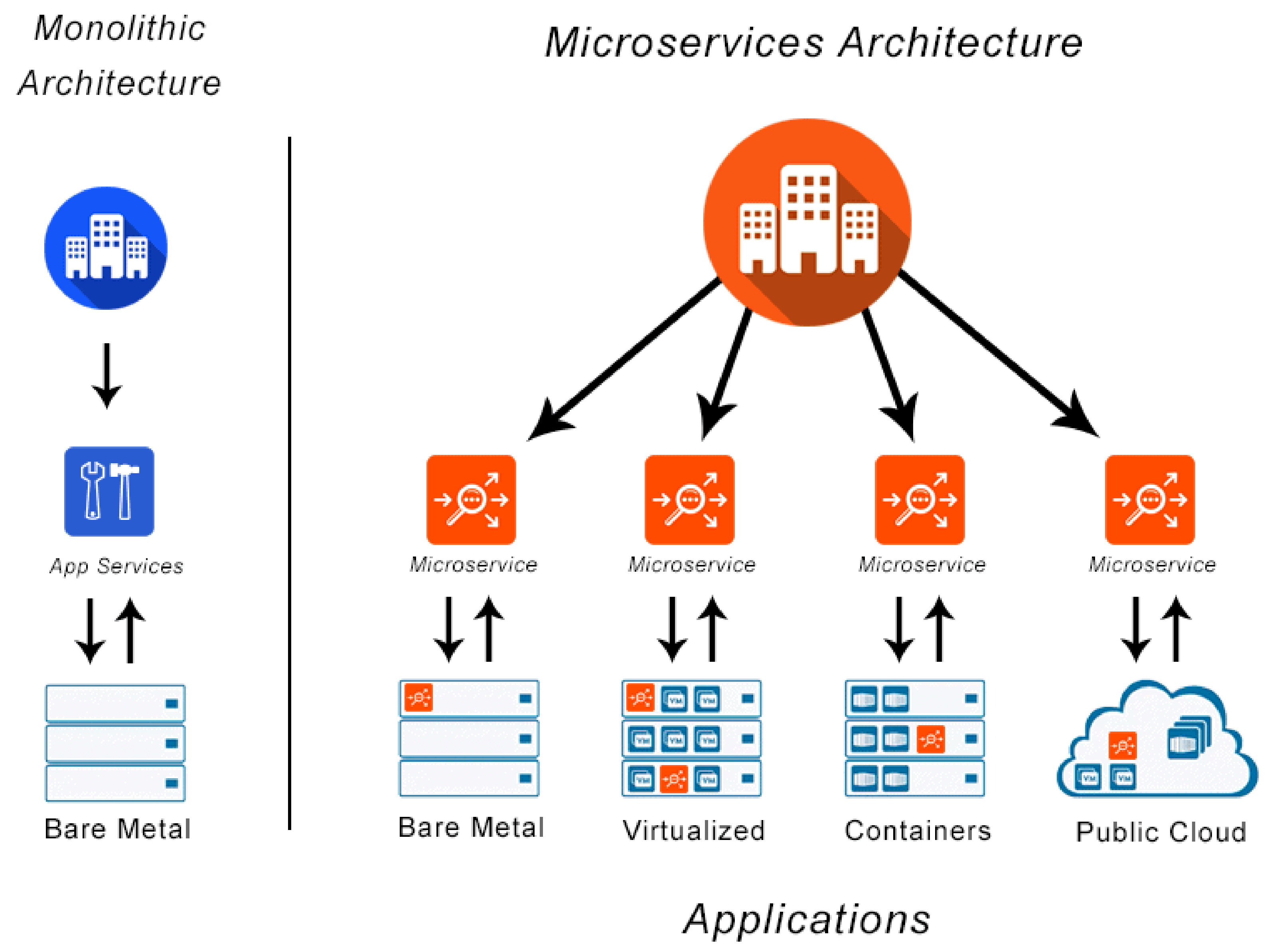
Microservice



Concretely, microservices are a software development method used to design an application as a set of modular services. Each module responds to a specific business objective and communicates with the other modules.

For Fowler, microservices should require the bare minimum in terms of centralized service management and can be created in different programming languages.

How the microservice works

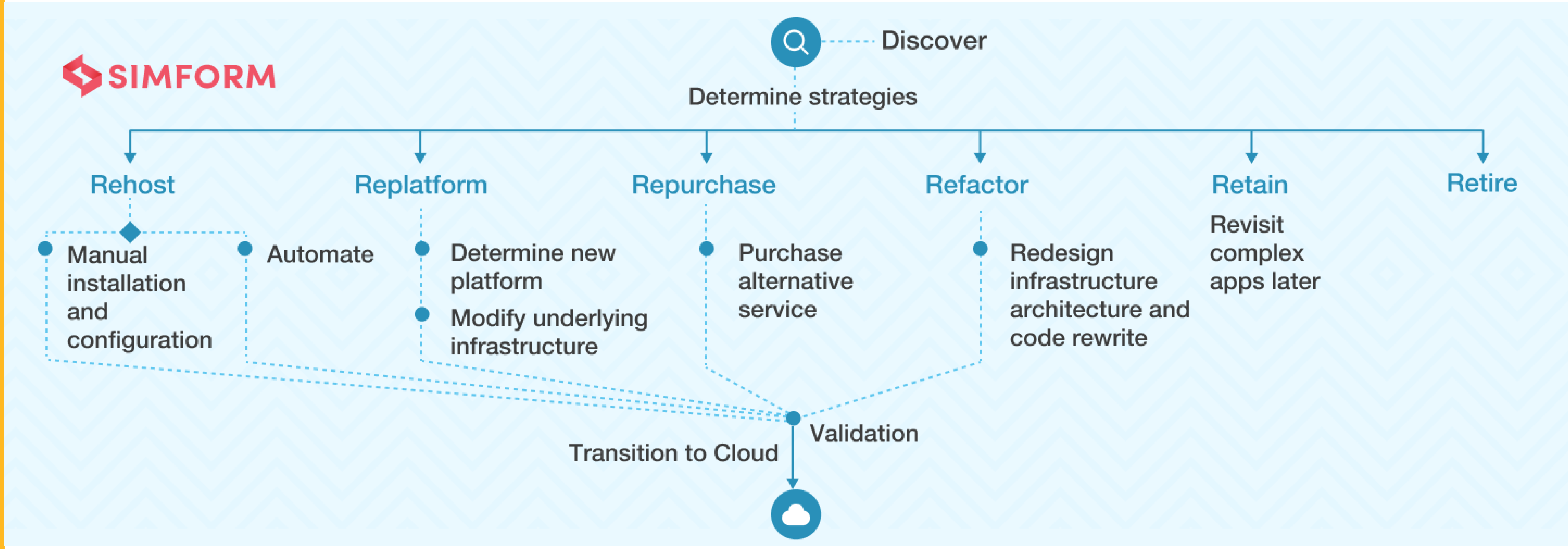


Normally, there are many microservices in such an architecture. And for the solution or application to work, the microservices must communicate and interact with each other.

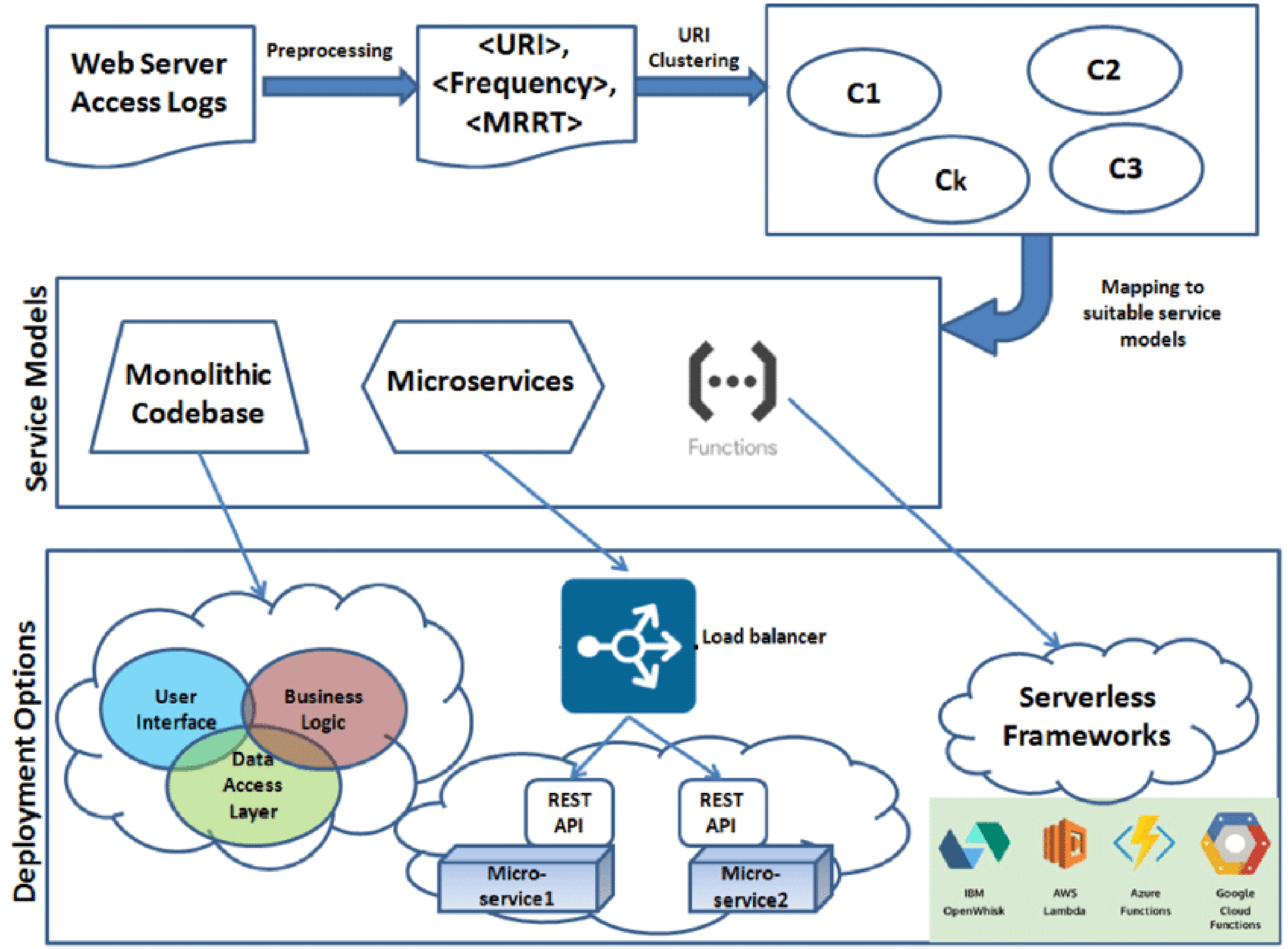
The microservice is therefore part of a larger ecosystem, functioning and working with other microservices to accomplish the functions of their parent application together.

What Is Cloud Migration?

Cloud migration is the process of moving some or all your digital operations to your cloud.
There are three main types of cloud migration you can perform — on-premises to cloud, cloud to cloud, or cloud to on-premises.
When performing any of these three migration types, there are five methods and strategies you can use.
These strategies are:


SIMFORM
Discover
Determine strategies
Rehost
Replatform
Repurchase
Refactor
Retain
Retire
Manual installation and configuration
Automate
Determine new platform
Modify underlying infrastructure
Purchase alternative service
Redesign infrastructure architecture and code rewrite
Revisit complex apps later
Validation
Transition to Cloud

The Refactoring Migration Approach



Advantages of Refactoring

The advantages of refactoring are mainly long-term benefits:
Long-term cost savings
—cloud-native and microservices architectures allow applications to rapidly change to adapt to new customer requirements, by adding new features or modifying existing functionality.
Increased resilience—by decoupling application components and wiring together managed solutions that provide high availability, the application inherits the durability of this cloud.

Disadvantages of Refactoring

The disadvantages of refactoring are:
Vendor lock-in
—the more cloud-native your application is, the more cloud features it is likely to consume. This makes applications tightly coupled to the public cloud you are using.
Time
—refactoring is resource-intensive and much more complex than a lift-and-shift migration, meaning projects take much longer to start showing business value.
Skills
—refactoring isn't for beginners. It requires advanced coding, automation, and DevOps skills.
Getting it wrong
—refactoring means changing many aspects of the application, so there is a high risk of errors at the code, configuration, and infrastructure level. Each mistake can cause delays, cost escalations, and possible outages.

How to refactor?

I. Complete Refactoring

In this type, 50% of the code is changed and the database is updated to utilize as many cloud-native features as required by the application. This strategy can improve performance, operations costs and IT teams' ability to meet the needs of the business. On the downside however, the process could be too costly or complex, and can introduce bugs.

2. Minimum Viable Refactoring

This requires only slight changes in the application, and is therefore, both quick and efficient. Users who take this approach often incorporate cloud-native security, management and perhaps a public cloud database into their migrated workload.

3. Containerization Refactoring

In this, applications are moved into containers with minimal modifications. The applications exist within the containers, which enables users to incorporate cloud-native features and improve portability.

This approach is found to be more complex because of the learning involved in adapting to new tools. But that is easily checked, as with the popularity of containers and their growing ecosystems, the costs and refactoring times continue to decrease.

4. Serverless Application Refactoring

This approach has similar issues as containerization as it changes the development and operations platform, which requires learning new tools and skills. Some modifications are required to make the application work effectively and take advantage of serverless systems on the public cloud.

Unlike containers however, serverless platforms don't provide portability, so lock-in is a major downside to this approach.

You can refactor your applications using either of these ways, but it is advisable to do Minimum Viable Refactoring for most of it. Refactoring is a highly variable activity, dependent on the current application complexity. And during its discovery assessment process, it’s not possible to predict how long an application refactor will take. It could be around three-to-six months per application depending on complexity and previous experience.

Hence, a targeted timeline, refactoring in parts, and checking progress with collected data are some of the best practices to keep in mind while taking up Refactoring cloud migration approach. Because of these reasons, this approach is chosen by very few enterprises that have the time, money, and resources for it.