Introduction To Microservice

Client App

Service
Consumer

account.wsdl

SOAP/HTTP

account.wsdl

Service
Provider

Account Service
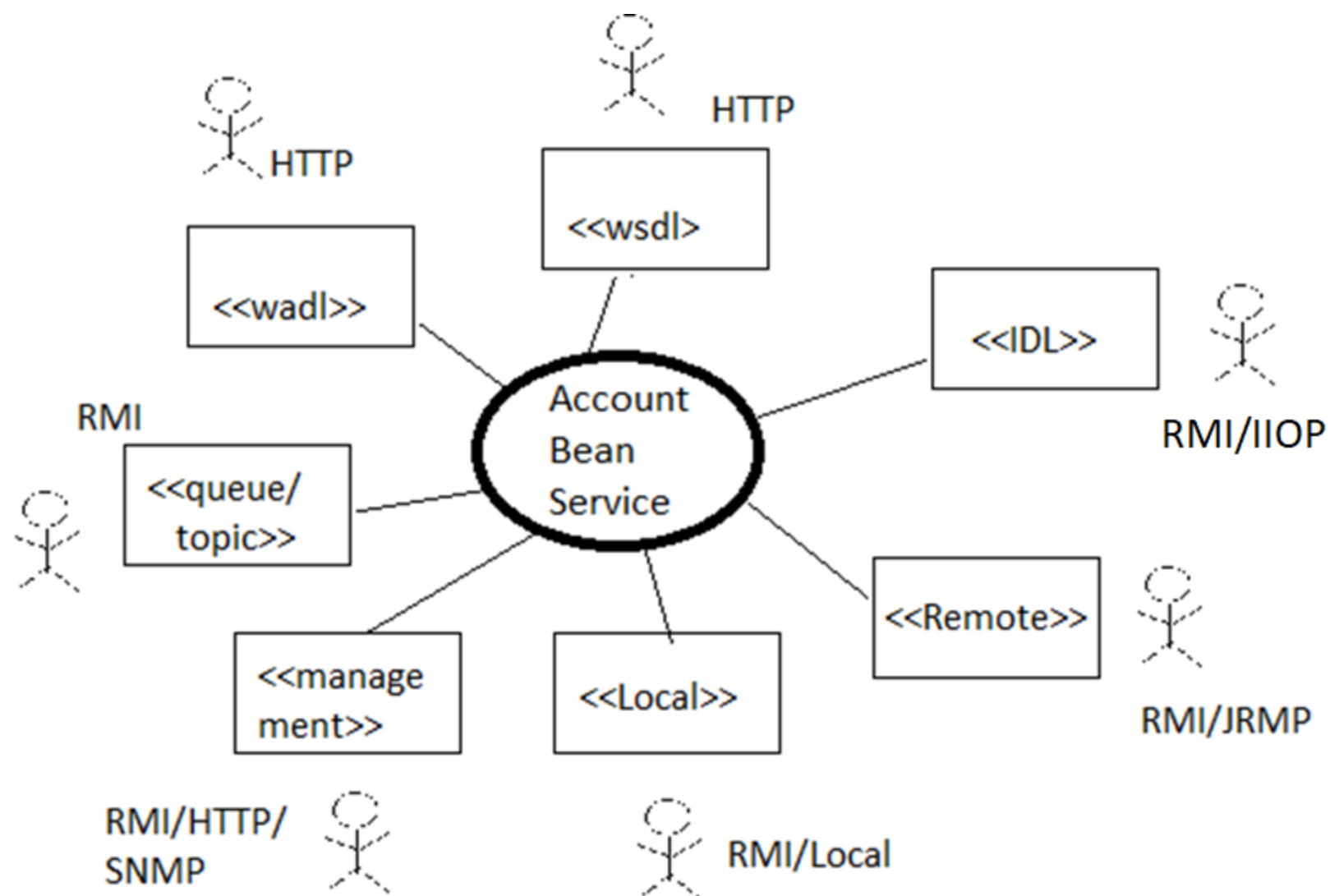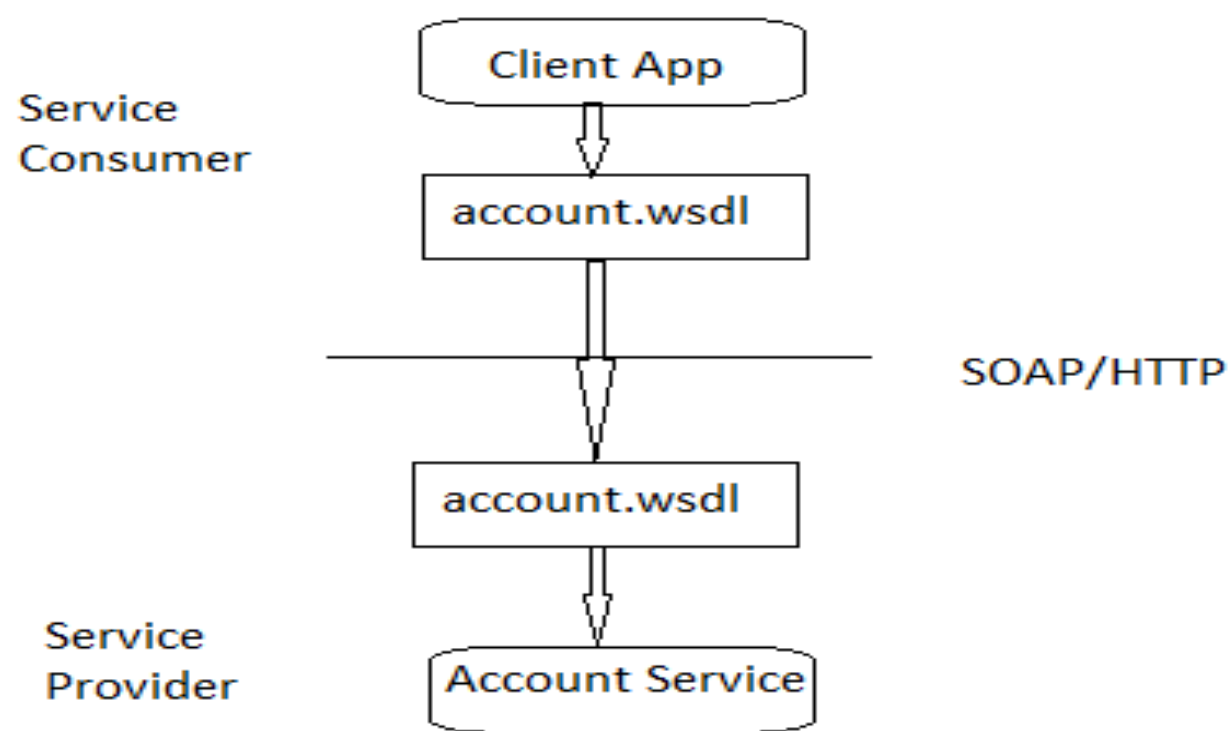
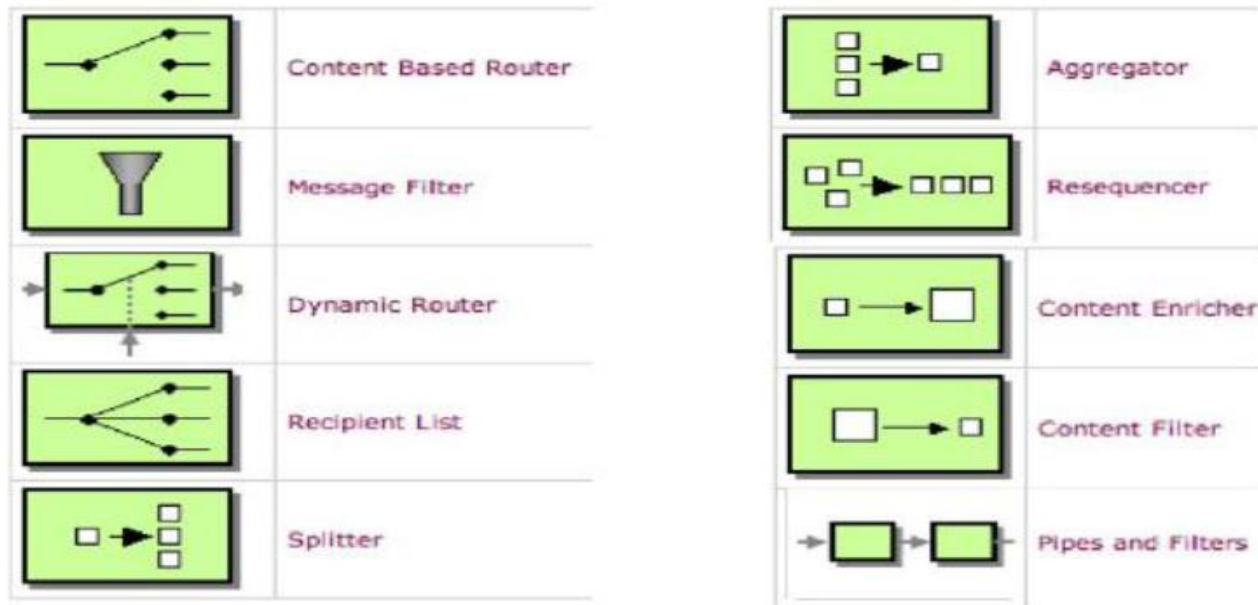# Webserivce OFF the BUS

SOA

Service Oriented Architecture (SOA) is an architectural style for implementing business processes as a set of loosely-coupled services.

Apache Camel is a
powerful Open Source
Integration Framework
based on known
Enterprise Integration Patterns

What is Apache Camel?

- Enterprise Integration Patterns
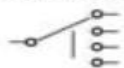
Camel

Integration Engine And Router

Camel Endpoints

* Camel can send messages to them
* Or Receive Messages from them

Filter Processor

Router Processor

Camel Processors

* Are use to wire Endpoints together
* Routing
* Transformation
* Mediation
* Interception
* Enrichment
* Validation
* Tracking
* Logging

JMS Component

JMS API

HTTP Component

Servlet API

File Component

File System

Camel Components

* Provide a uniform Endpoint Interface
* Act as connectors to all other systems

Web Container

Jetty | Tomcat | ...

JMS Provider ActiveMQ | IBM | Tibco | Sonic ...

HTTP Client

Local File System

What is Apache Camel?

- ## Camel's Architecture

Routing engine

A DSL wires endpoints and processors together to form routes.

CamelContext

Route 1

Route 2

Route N
```
from("file:c:\dir")
.filter()
.xpath(expression)
.to("jms:aQueue");
```

Message filter processor

. . .

Content - based router processor

File . . . . JMS HTTP

Processors

Handle things in between endpoints like

- EIPs
- Routing
- Transformation
- Mediation
- Enrichment
- Validation
- Interception

Components

- Provide a uniform endpoint interface
- Connect to other systems

Microservice architecture is a method of developing software applications as a suite of independently deployable, small, modular services in which each service runs a unique process and communicates through a well-defined, lightweight mechanism to serve a business goal.

Microservices are a service-oriented architectural pattern as well for defining distributed software architectures.

The pattern aims for better scalability, decoupling and control throughout the application development, testing and deployment cycle.

It relies on an inter-service communication protocol, which could be SOAP, REST and other technologies..
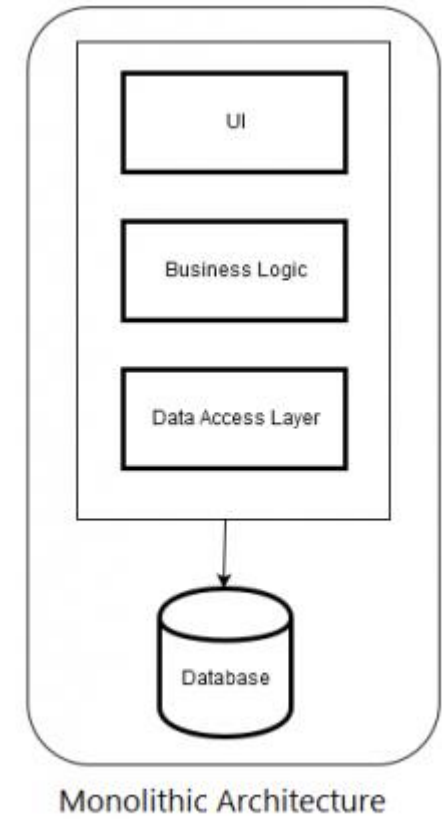
*The microservices style is usually organized around **business capabilities and priorities.***

Unlike a traditional monolithic development approach—where different teams each have a specific focus on, say, UIs, databases, technology layers, or server-side logic—microservice architecture utilizes cross-functional teams.

The responsibilities of each team are to make specific products based on one or more individual services communicating via message bus. That means that when changes are required, there won't necessarily be any reason for the project, as a whole, to take more time or for developers to have to wait for budgetary approval before individual services can be improved.

Online Shopping Center

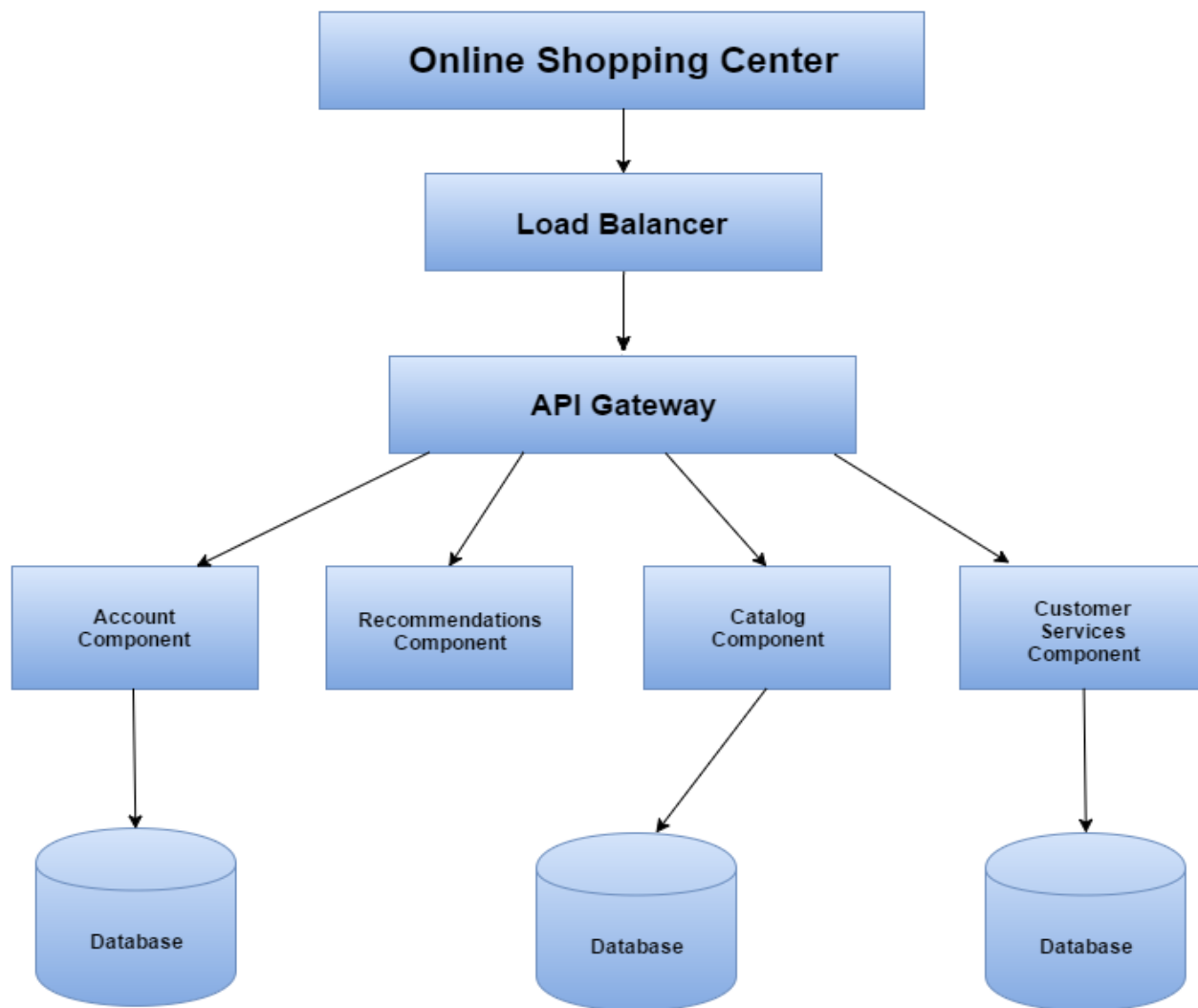**Load Balancer**

Account Component

Catalog Component

Recommendations Component

Customer Services Component

Database

Figure 1: Conventional Approach

UI

Business Logic

Data Access Layer

Database

Monolithic Architecture

Online Shopping Center

Load Balancer

API Gateway

| Account Component | Recommendations Component | Catalog Component | Customer Services Component |

Database

Database

Database

UI

Microservice

Microservice

Microservice

Microservice

Database

Database

Database

Microservices Architecture

Figure 2: Micro Services Approach
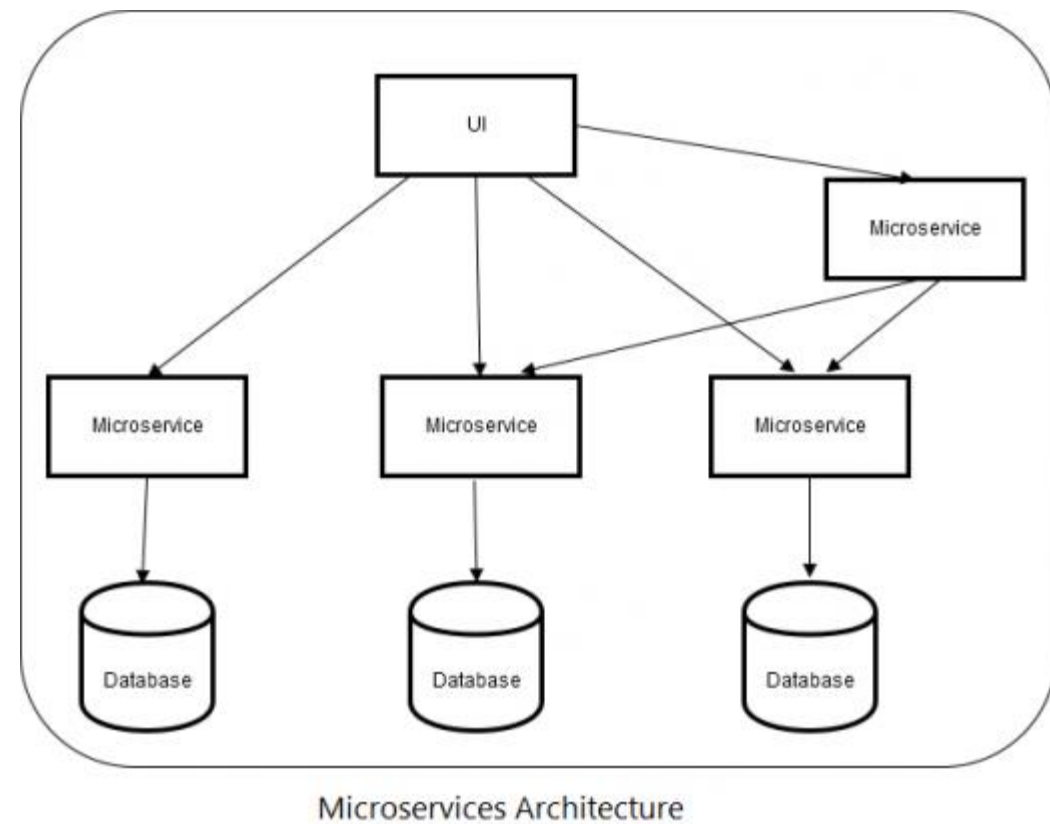
SOA Vs Microservices

SOA Architecture

| Services/APIs |
| Application |
| Application Server |
| Managed Runtime Environment |
| Operating System |
| Hypervisor |
| Storage |
| Network |

Hardware

Microservices Architecture

| Services/APIs |
| Managed Runtime Environment |
| Operating System |
| Hypervisor |
| Storage |
| Network |

Hardware

SOA Vs Microservices

SOA commonly relies on a shared data model with multiple hierarchies and complex relationships between dozens or hundreds of data structures and services.

It uses a tiered organizational architecture that contains a centralized messaging middleware for service coordination plus additional messaging functionality.

Microservices are a service-oriented architectural pattern as well for defining distributed software architectures.

The pattern aims for better scalability, decoupling and control throughout the application development, testing and deployment cycle.

It relies on an inter-service communication protocol, which could be SOAP, REST and other technologies..
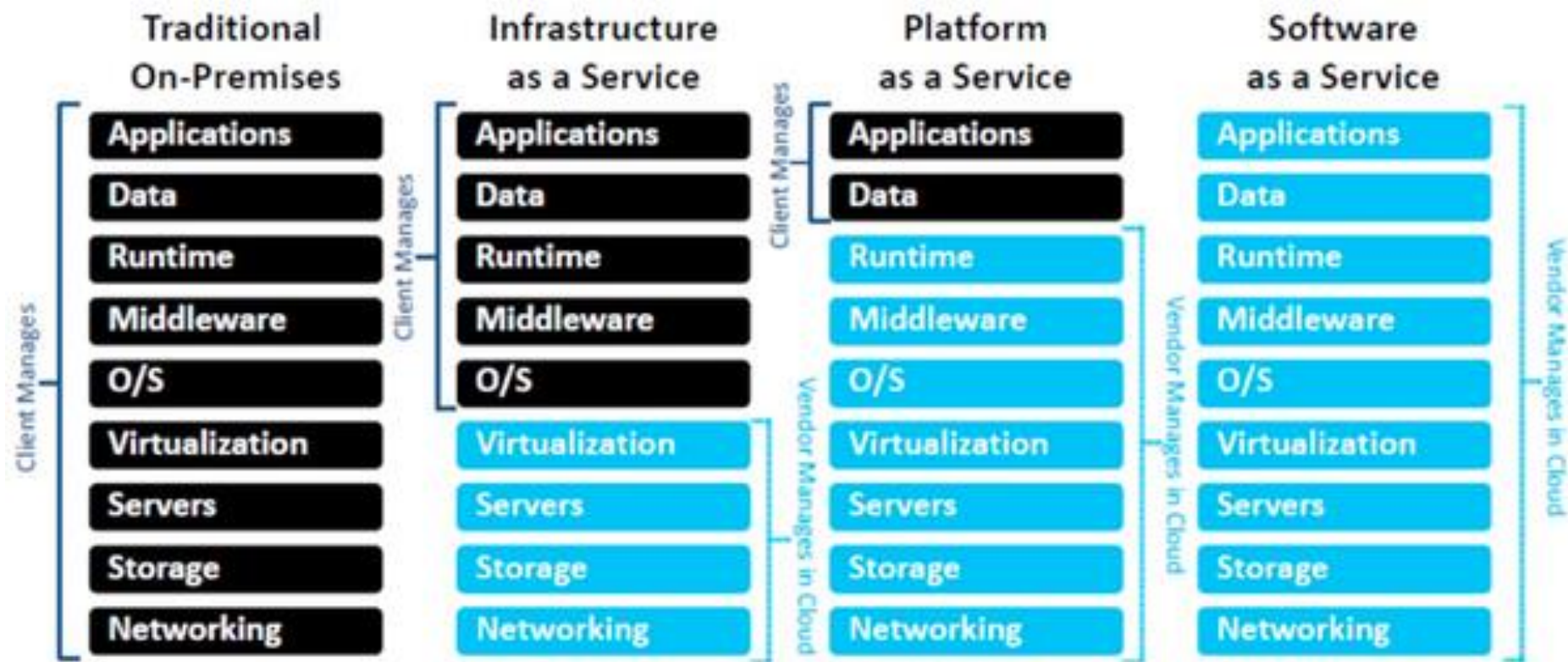
Microservices Benefits

❑ Smaller code base is easy to maintain.

❑ Easy to scale as individual component.

❑ Technology diversity i.e. we can mix libraries, databases, frameworks etc.

❑ Fault isolation i.e. a process failure should not bring whole system down.

❑ Better support for smaller and parallel team.

❑ Independent deployment

❑ Deployment time reduce

Microservices Challenges

❑ Difficult to achieve strong consistency across services

❑ ACID transactions do not span multiple processes.

❑ Distributed System so hard to debug and trace the issues

❑ Greater need for end to end testing

❑ Required cultural changes in across teams like Dev and Ops working together even in same team.

Microservices Infrastructure

❑ Platform as a Service like Pivotal Cloud Foundry help to deployment, easily run, scale, monitor etc.

❑ It support for continuous deployment, rolling upgrades fo new versions of code, running multiple versions of same service at same time

Traditional On-Premises	Infrastructure as a Service	Platform as a Service	Software as a Service
Applications	Applications	Applications	Applications
Data	Data	Data	Data
Runtime	Runtime	Runtime	Runtime
Middleware	Middleware	Middleware	Middleware
O/S	O/S	O/S	O/S
Virtualization	Virtualization	Virtualization	Virtualization
Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking

Customization; higher costs; slower time to value

Standardization; lower costs; faster time to value

Microservices Tooling Supports

1. Using Spring for creating Microservices

❑ Setup new service by using Spring Boot

❑ Expose resources via a RestController

❑ Consume remote services using RestTemplate/Feign

2. Adding Spring Cloud and Discovery server

❑ It is building blocks for Cloud and Microservices

❑ It provides microservices infrastructure like provide use services such as Service Discovery, Configuration server and Monitoring.

❑ It provides several other open source projects like Netflix OSS
.

❑ It provides PaaS like Cloud Foundry, AWS etc.,

❑ It uses Spring Boot style starters

spring-cloud-netflix

This project provides Netflix OSS integrations for Spring Boot apps through autoconfiguration and binding to the Spring Environment and other Spring programming model idioms.

With a few simple annotations you can quickly enable and configure the common patterns inside your application and build large distributed systems with battle-tested Netflix components.

The patterns provided include Service Discovery (Eureka), Circuit Breaker (Hystrix), Intelligent Routing (Zuul) and Client Side Load Balancing (Ribbon
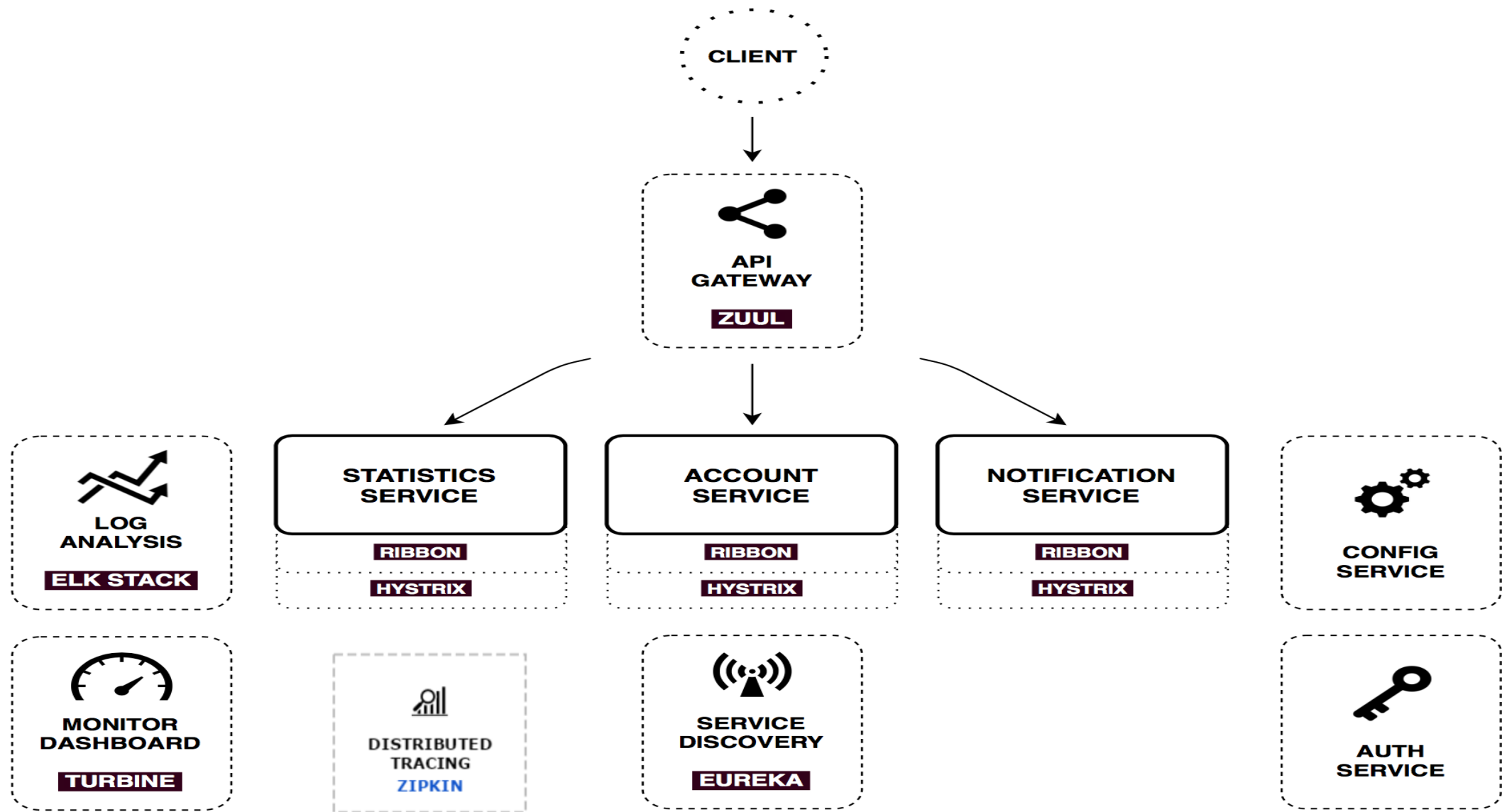
<u>Service Discovery:</u>

Eureka instances can be registered and clients can discover the instances using Spring-managed beans

<u>Circuit Breaker:</u>

Hystrix clients can be built with a simple annotation-driven method decorator
Embedded Hystrix dashboard with declarative Java configuration

<u>Declarative REST Client:</u>

Feign creates a dynamic implementation of an interface decorated with JAX-RS or Spring MVC annotations

<u>Client Side Load Balancer:</u>

Ribbon

External Configuration:

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.

Router and Filter:

Automatic registration of **Zuul** filters, and a simple convention over configuration approach to reverse proxy creation

Hystrix stream

Netflix Hystrix has a neat feature called the Hystrix stream that provides real-time metrics on the state of the Hystrix commands in an application. This data tends to be very raw though, a very cool interface called the Hystrix Dashboard consumes this raw data and presents a graphical information based on the underlying raw data.

Turbine stream

Hystrix stream provides information on a single application, Turbine provides a way to aggregate this information across all installations of an application in a cluster. Integrating turbine into a Spring-Cloud based application is straightforward, all it requires is information on which clusters to expose information on and how to aggregate information about the specific clusters.

<u>ELK – Elasticsearch, Logstash, Kibana:</u>

three different tools usually used together.

They are used for searching, analyzing, and visualizing log data in a real time.

<u>Zipkin is a distributed tracing system</u>

It helps gather timing data needed to troubleshoot latency problems in microservice architectures.

It manages both the collection and lookup of this data.