An Introduction to Microservices

Ravishka Rathnasuriya
PhD Student

Monolithic Architecture

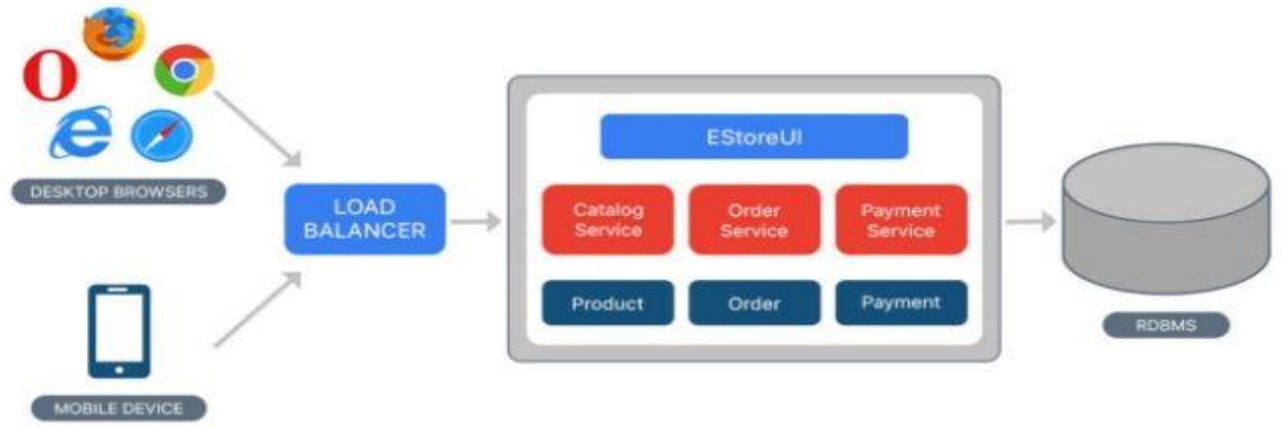
- Traditional unified model for the design of a software program.
- Other words, it is a big container where in all the software components of an application are assembled together and tightly packed.
- A single tired software application.
- Advantages at the early stages of development:
  - -Simple to develop
  - -Simple to test
  - -Simple to deploy
  - -Simple to scale horizontally.
- Once the application becomes large and complex, this approach has a number of drawbacks

Challenges to Monolithic Architecture

- Large and Complex applications
- Slow development
- Blocks continuous development
- Inflexible
- Unreliable
- Unscalable

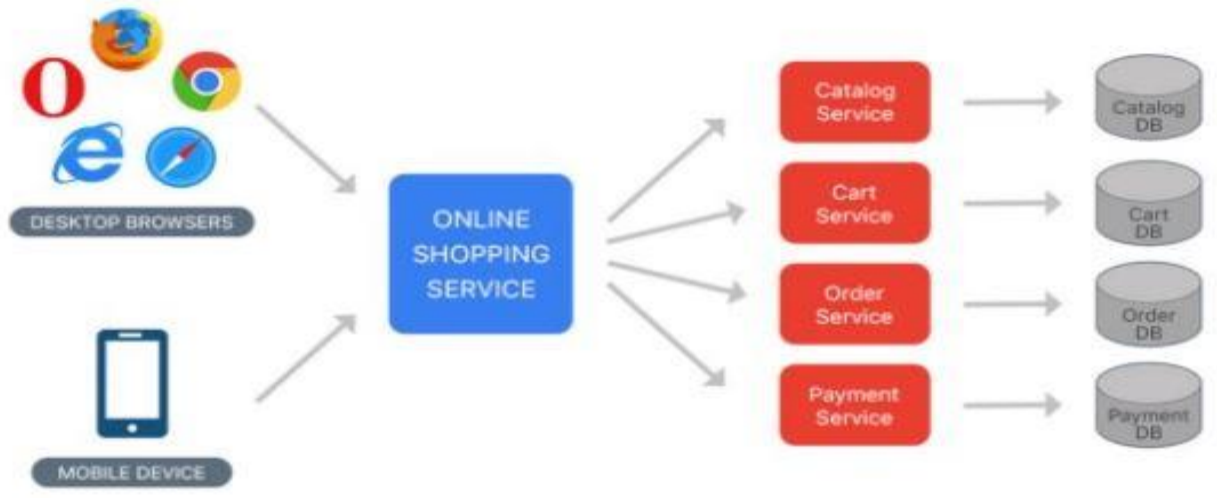
What are Microservices?

- Also know as Microservices Architecture
- An architectural style that structures an application as a suite of loosely coupled services, each of which has a single responsibility and can be deployed independently, scaled independently, and tested independently.[1]
- Microservices are individual software applications that communicate with each other through a well-defined network interface.
- Some of the features are:
  - -Small focused.
  - -Language Neutral
  - -Loosely Coupled
  - -Highly maintainable and testable



Monolithic Architecture (for E-Commerce Application)

[https://medium.com/koderlabs/introduction-to-monolithic-architecture-and-microservices-architecture-b211a5955c63]



Microservices Architecture (for E-Commerce Application)

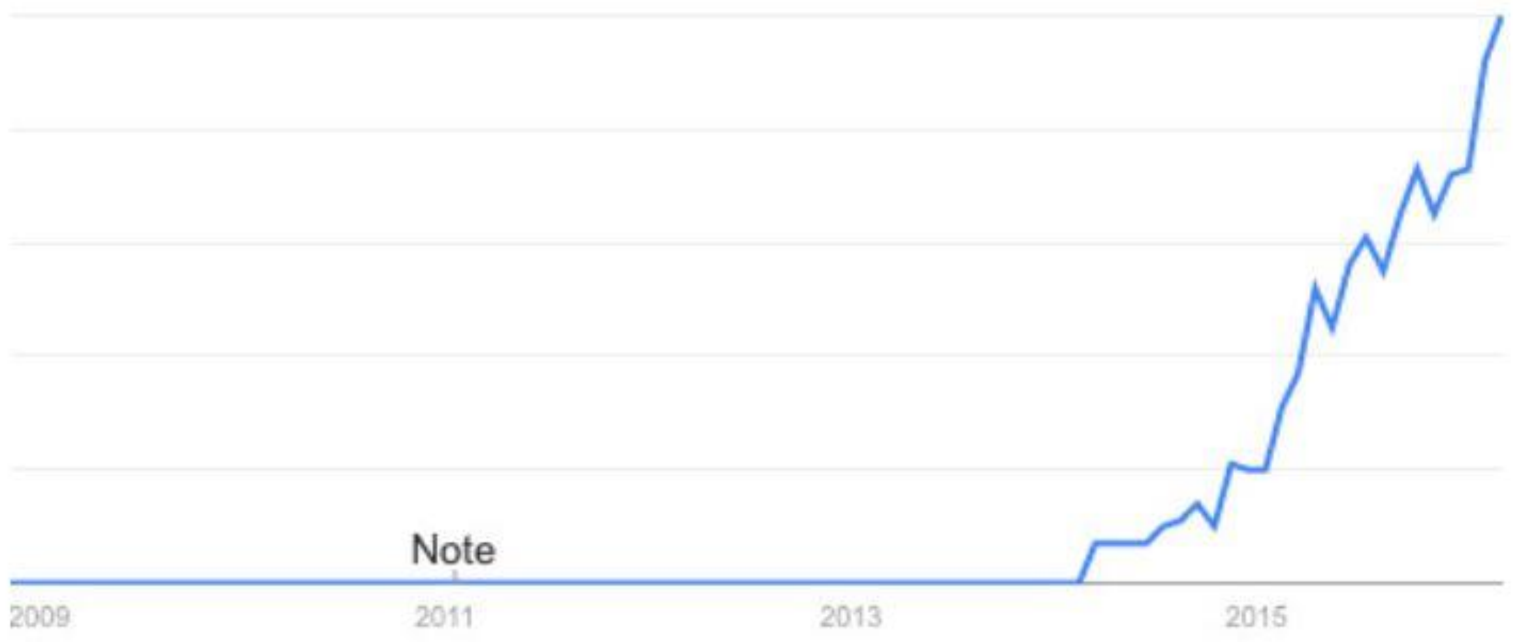
Why Microservices is Getting Popular?

1. Availability of technologies (Docker and virtual machines, Cloud platforms)
2. Faster time to the market
3. Dedicated teams work on discrete functions such as UI, database, server-side logic, and technological layers, microservices uses cross-functional teams to handle the entire life cycle of an application using a continuous delivery mode.
4. Easy management

More than three quarters (77 percent) of businesses have now adopted microservices.

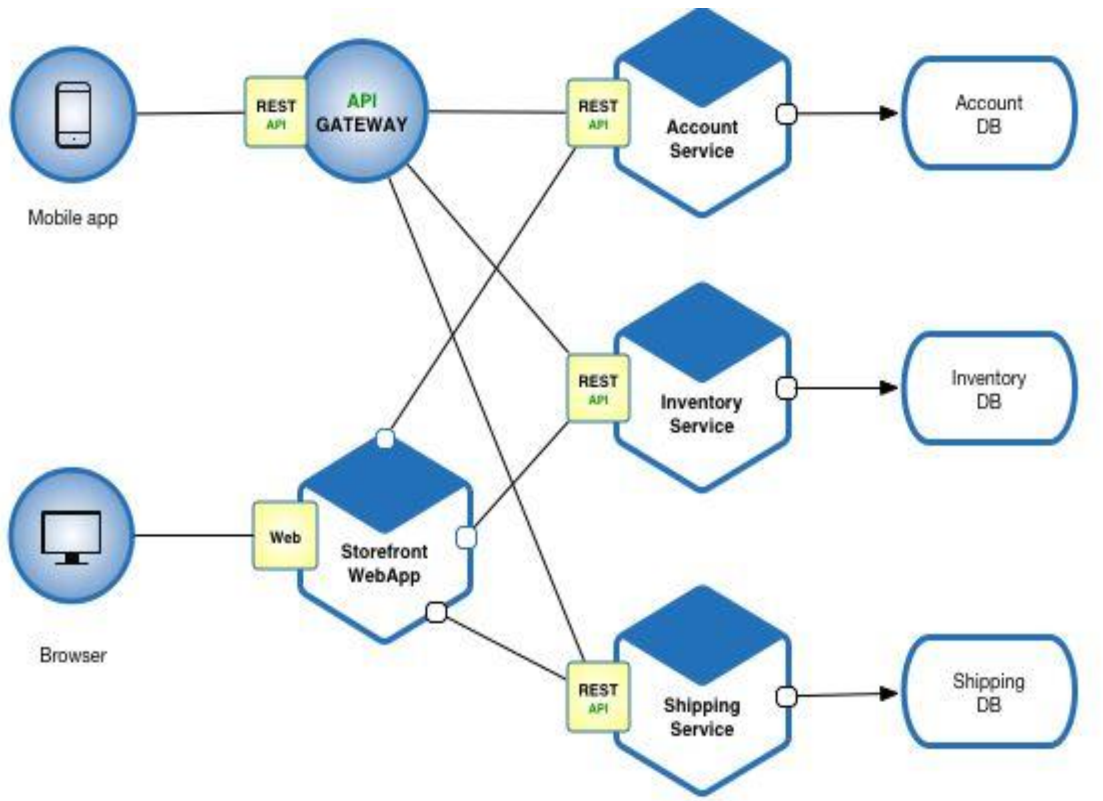
Most companies were influenced to use microservices by Amazon and Netflix

Trend of Use of Microservices



Advantages of Microservices

- ❖Faster Delivery
  - Parallel Development
  - Extendibility and Expandability
- ❖Improved Scalability and Availability
  - Flexible and Automatic scalability
  - Fault Tolerance and Fault Isolation
- ❖Greater Autonomy
  - Reduced communication cost
  - Flexible choice of technology stack



Companies that use Microservice Architecture

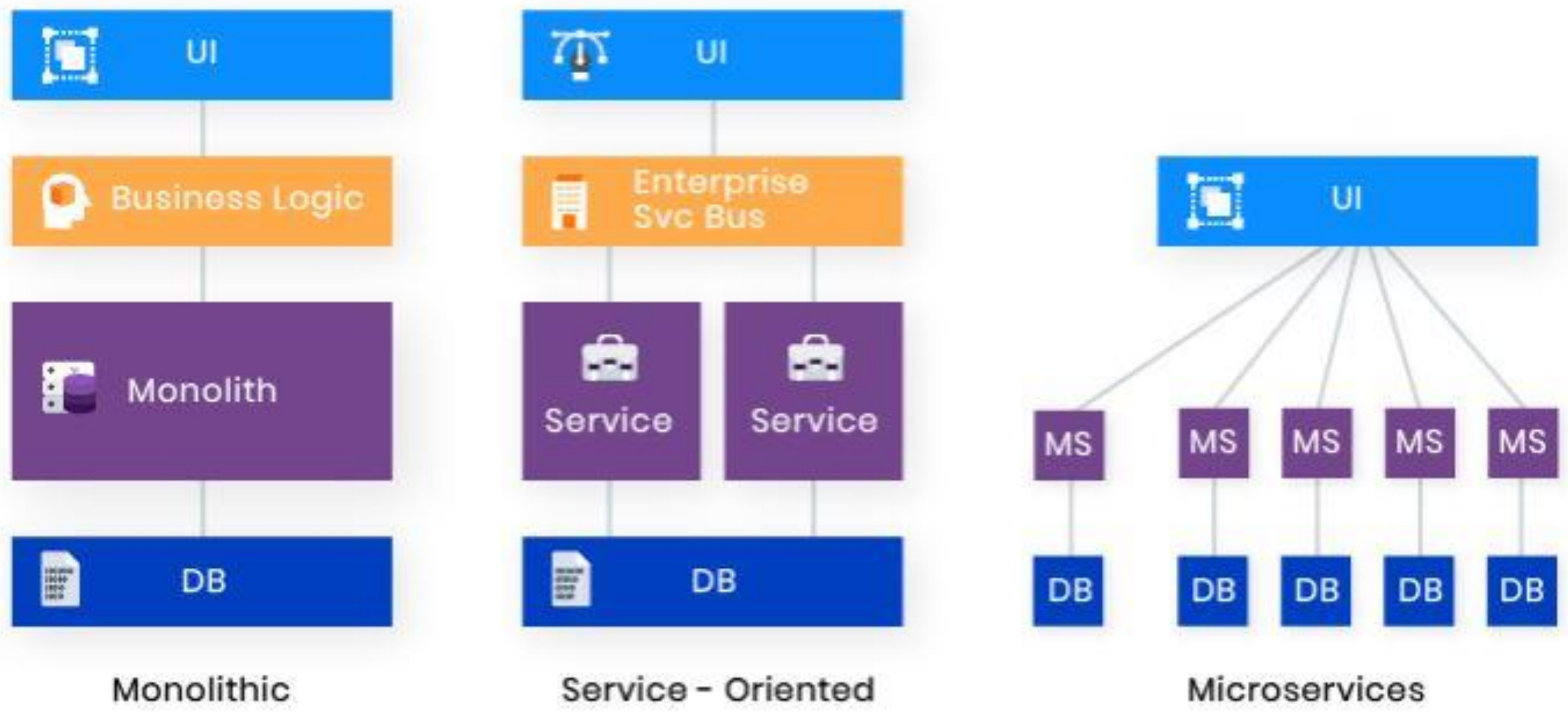
- Amazon
- Netflix
- Twitter
- Uber
- eBay
- PayPal
- Sound cloud
- Etsy

SOA vs Microservices

- Service Oriented Architecture. Also known as coarse-grained architecture.
- SOA breaks up the components required for applications into separate service modules that communicate with one another to meet specific business objectives.
  Each module is considerably smaller than a monolithic application, and can be deployed to serve different purposes in an enterprise. [6]

SOA delivers 4 services

- Functional services: used for business operations
- Enterprise services: implement the functionality
- Application services: specific for developing and deploying apps
- Infrastructure services: for non-functional processes such as security and authentication


UI
Business Logic
Monolith
DB
Monolithic
UI
Enterprise Svc Bus
Service
Service
DB
Service - Oriented
UI
MS
MS
MS
MS
MS
DB
DB
DB
DB
DB
Microservices

	Microservices	SOA
Architecture	Designed to host services which can function independently	Designed to share resources across services
Component sharing	Typically does not involve component sharing	Frequently involves component sharing
Granularity	Fine-grained services	Larger, more modular services
Data storage	Each service can have an independent data storage	Involves sharing data storage between services
Governance	Requires collaboration between teams	Common governance protocols across teams
Size and scope	Better for smaller and web-based applications	Better for large scale integrations
Communication	Communicates through an API layer	Communicates through an ESB
Coupling and cohesion	Relies on bounded context for coupling	Relies on sharing resources
Remote services	Uses REST and JMS	Uses protocols like SOAP and AMQP
Deployment	Quick and easy deployment	Less flexibility in deployment

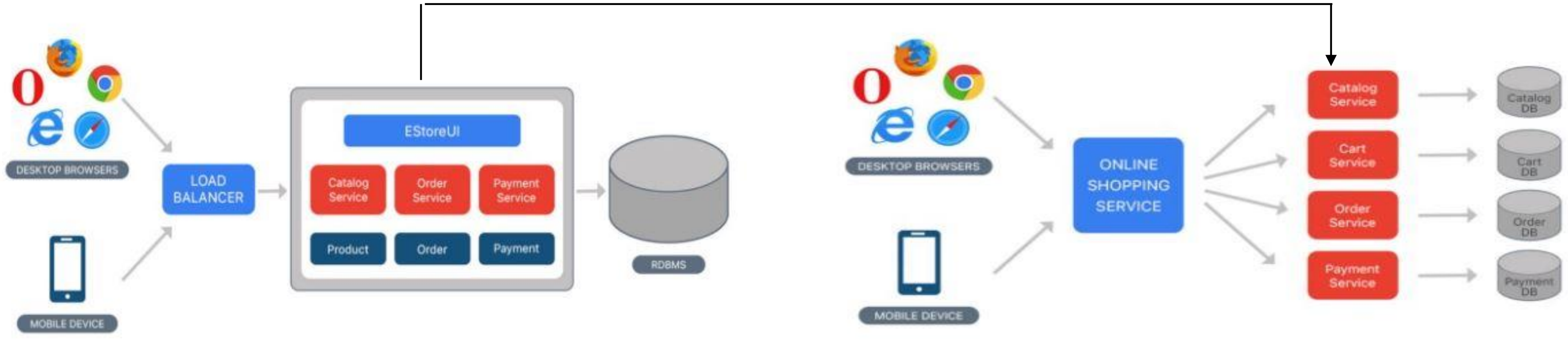
Capabilities of Microservice systems, Issues, Practices, and Challenges.

1. Service Decomposition

Breaking larger, coarse-grained services down to smaller, fine-grained services

Improve performance, can be developed independently, easy to scale

Strategies: By expert experience, Domain-driven design, data flow



Monolithic Architecture (for E-Commerce Application)

Microservices Architecture (for E-Commerce Application)

Service Decomposition

I1: Decomposition Decision influences a lot.

Improper service decompositions may cause serious quality problems.

Using a domain driven approach by mapping concepts in problem domain into the artefacts in the solution domain.

Challenges: How to conduct domain analysis to derive domain design, how to extract domain concepts, how to monitor and maintain consistency.

I2: Service Dependencies are Hard to Capture

When services are independently developed, missing service dependencies are harder to to measure service coupling to evaluate quality of decomposition

Capturing Service Dependency using Runtime Tracing

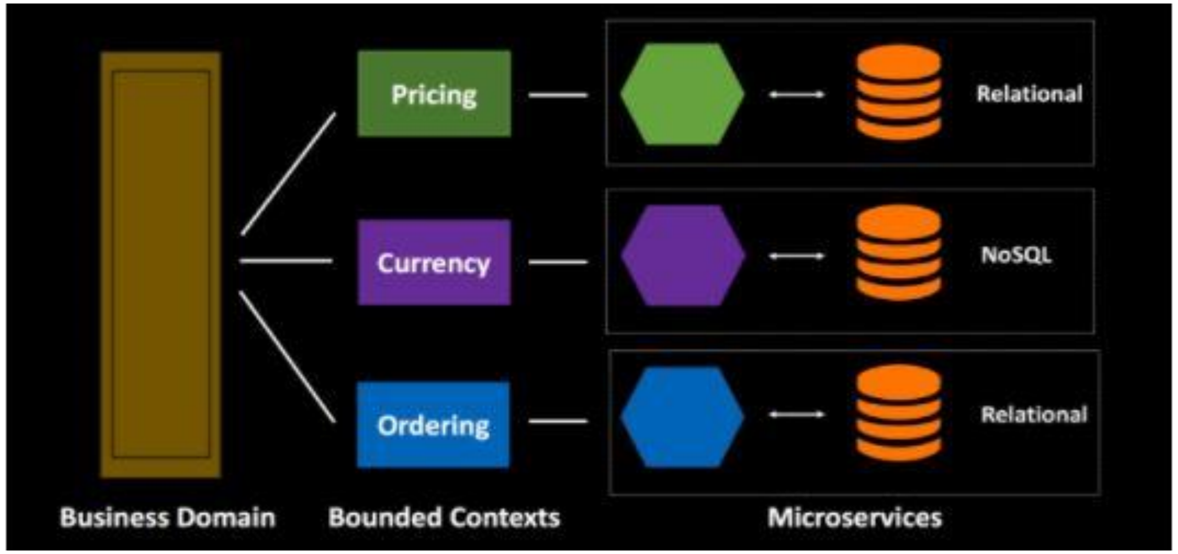
Challenges: High Cost and Low Coverage of Runtime Tracing

2. Database Decomposition

Split your database into multiple small databases that each has its own separate database with its own domain data.

Allow you to independently deploy and scale your microservices.

Recommended: Each service should manage its own database



Database Decomposition

I1: Data Coupling among Services

- Multiple services may have data coupling when data elements in their databases involve same data query or transaction. Cannot use cascading query and traditional transaction management cannot be used because of physical isolation

Service Invocation Composition and Distributed Transaction. (Eg: Seata)
Implementing cascading queries by composition of multiple service invocations and using distributed transaction frameworks like Seata

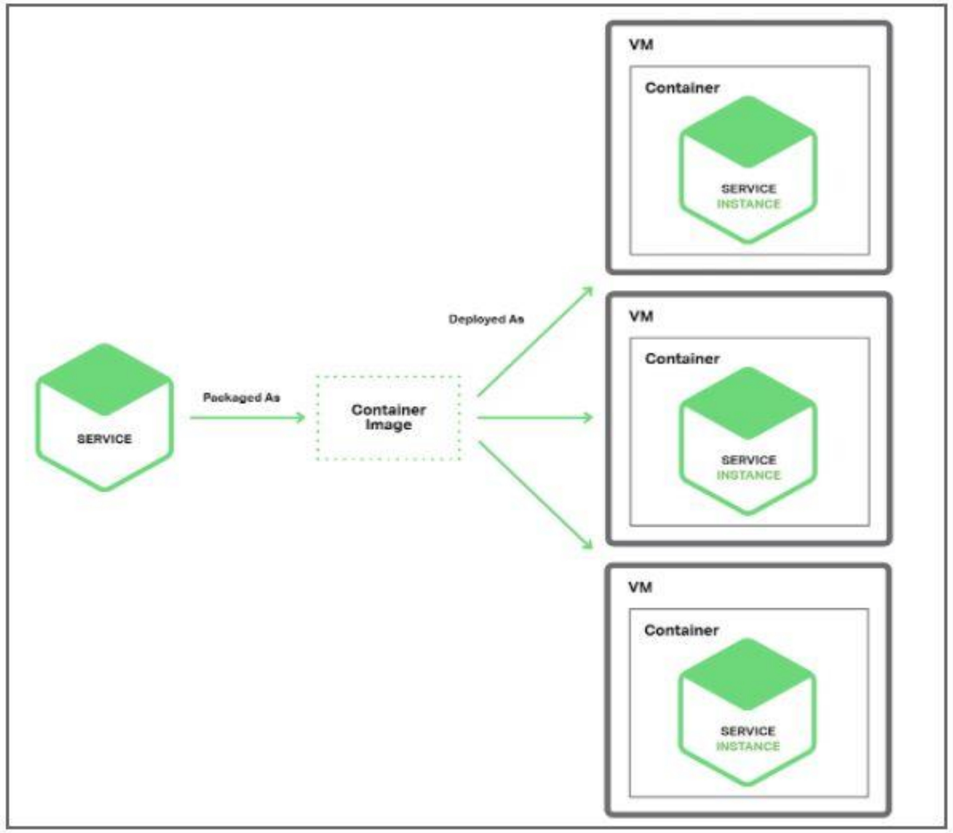
Challenges: Subsequent Refactoring and Network Latency
When database is decomposed into multiple parts, refactoring needs to be done.
Multi-service invocations may cause network latency

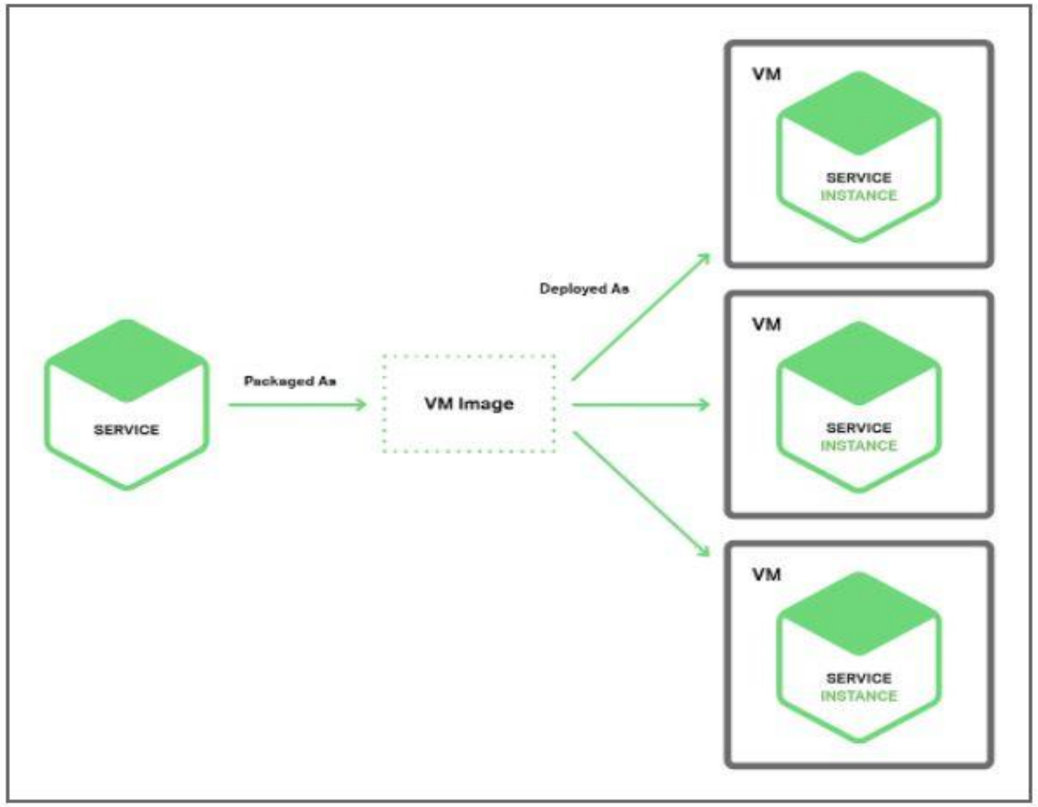
3. Deployment

- Each service is deployed as a set of service instances.
- Solutions: Virtual Machines, Physical Machine, Containers, Virtual machines and Containers.
- Manage Containers using Kubernetes, Mesos, Docker Swarm.
- Manage Virtual Machines using VMware vSphere.
- Systems are also hosted by cloud platforms.

I1: Complex Service Configuration
Eg: inconsistent memory limitations of JVM and Docker.

Deployment

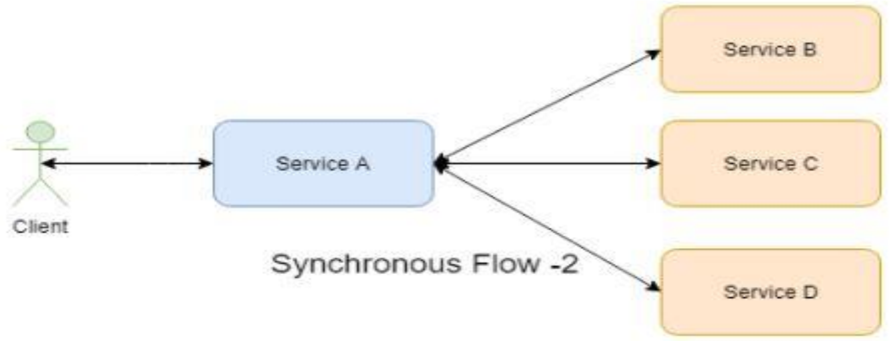

SERVICE
Packaged As
Container Image
Deployed As
VM
Container
SERVICE INSTANCE
VM
Container
SERVICE INSTANCE
VM
Container
SERVICE INSTANCE

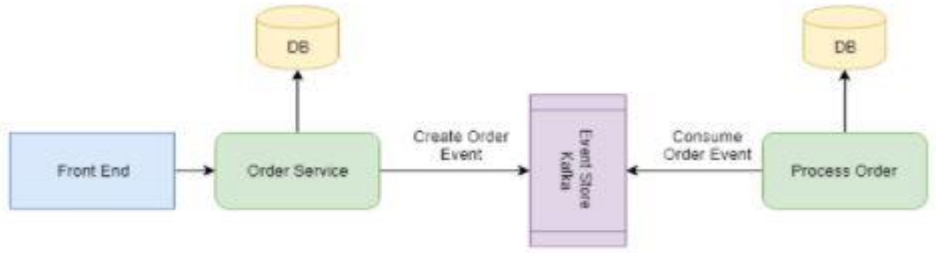

SERVICE
Packaged As
VM Image
Deployed As
VM
SERVICE INSTANCE
VM
SERVICE INSTANCE
VM
SERVICE INSTANCE

4. Service Communication Design

Inter-communication between microservices

- Services must interact using an inter-process communication protocol such as HTTP, AMQP, and RPC.
- Communication styles: HTTPS/REST, RPC and Messaging
- Synchronous: Client makes a request to the server an waits for its response. Thread is blocked until it receives communication back.
  Asynchronous: Client makes call to the server, server will process the request and complete it.

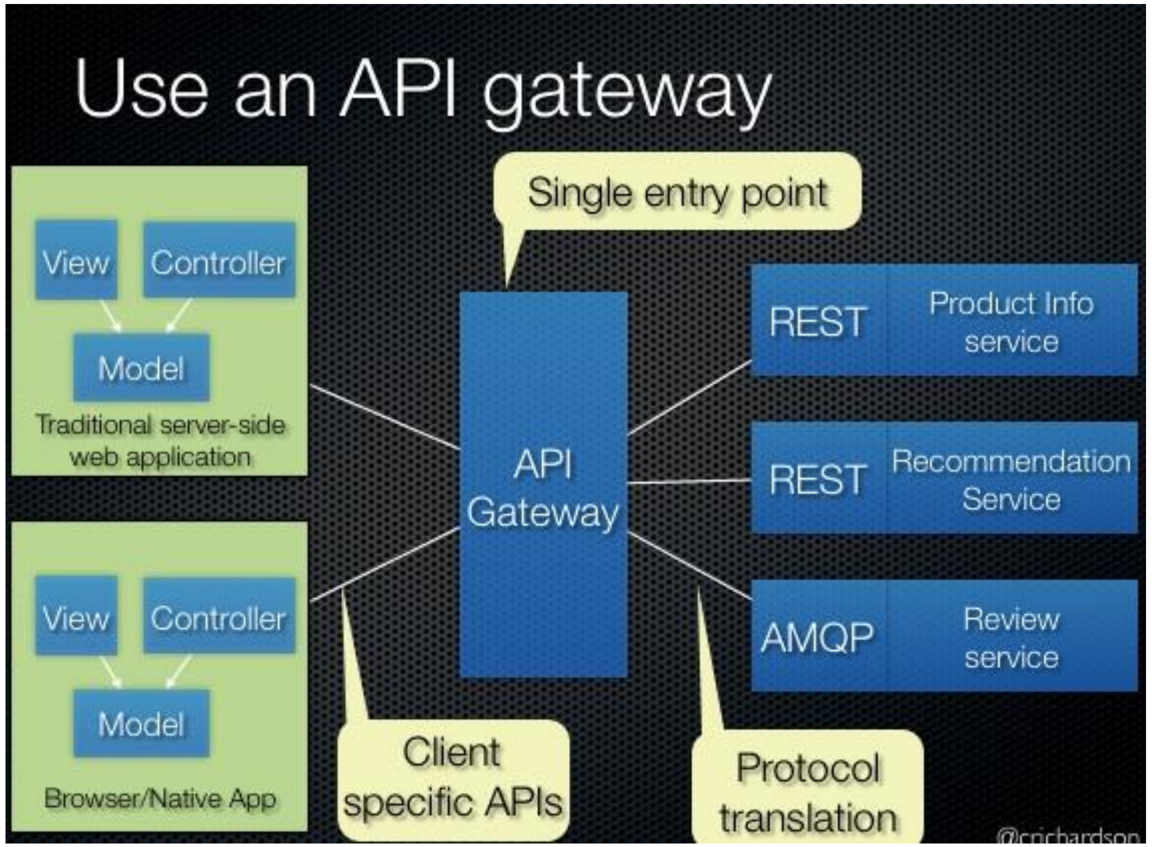


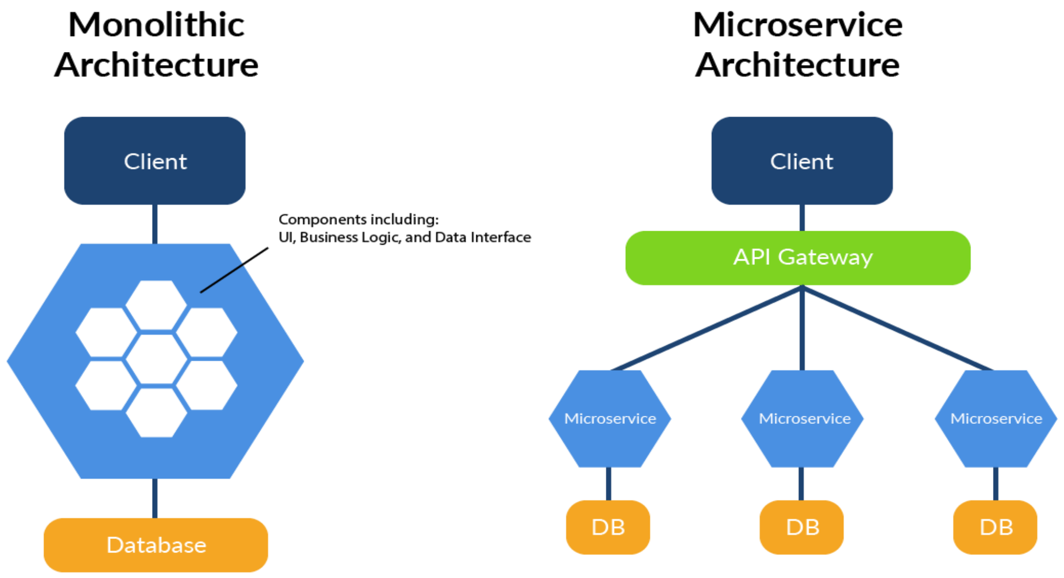


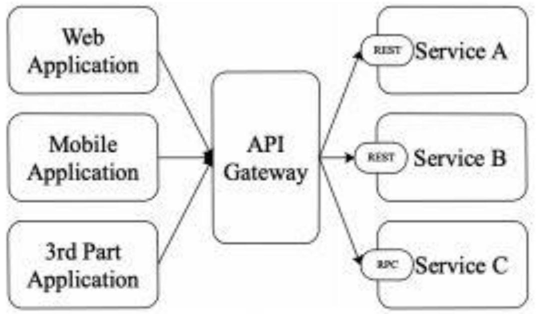
Event Driven - Asynchronous Flow

5. API Gateway Design

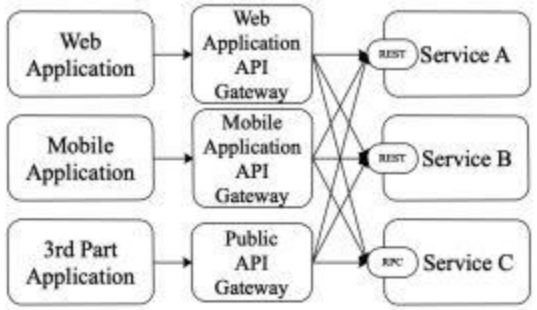
- Application Programming Interface(API) is a way which you can make sure two or more applications communicate with each other to process the client request.
- Helps to access the application data and use application’s functionality
- API gateway is a server that is a single entry point into the system. As soon as you send a request API Gateway will decide to which service the particular request has to be sent. Act as entry point to forward the clients requests to appropriate microservices.
- Responsibilities: Authentication, Monitoring, Load balancing, caching etc.
- Types of API gateways: Single, Multiple
- Kinds of Clients: Web, Mobile, and external 3rd party applications.







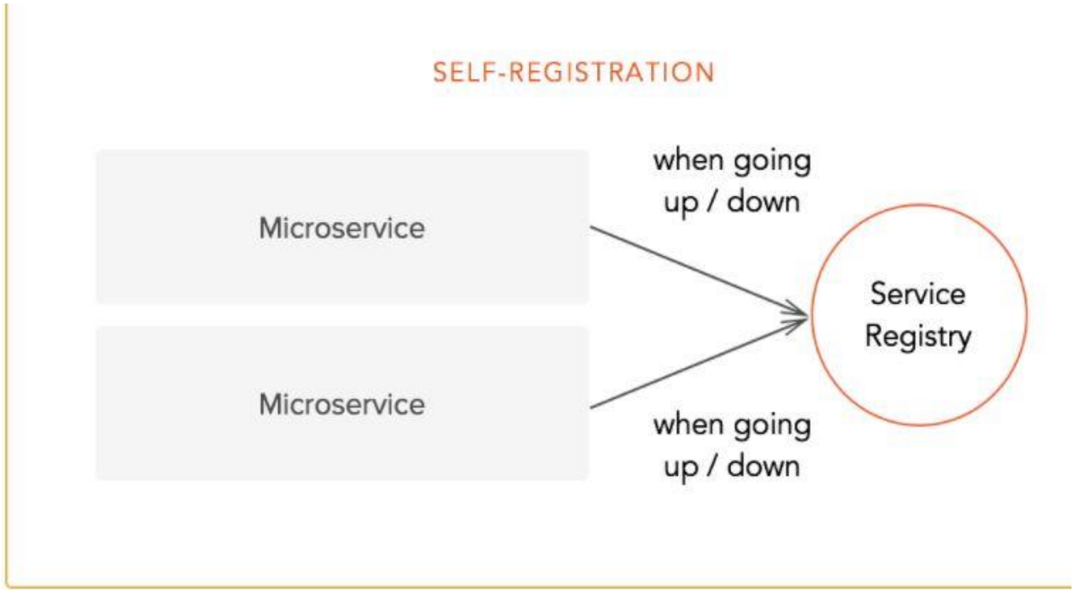
(a) Single API Gateway

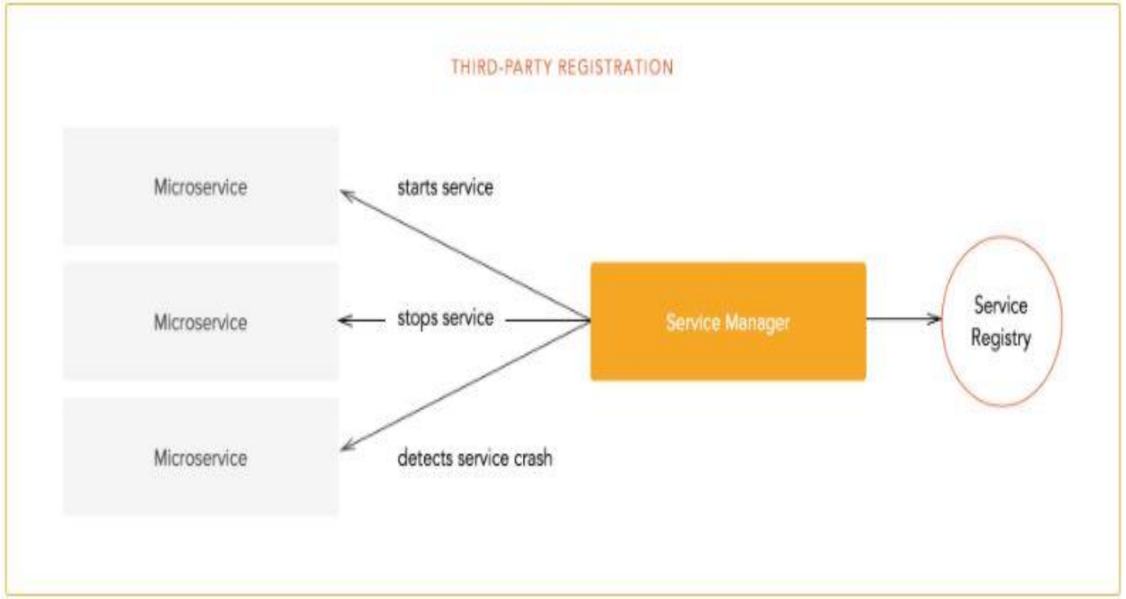


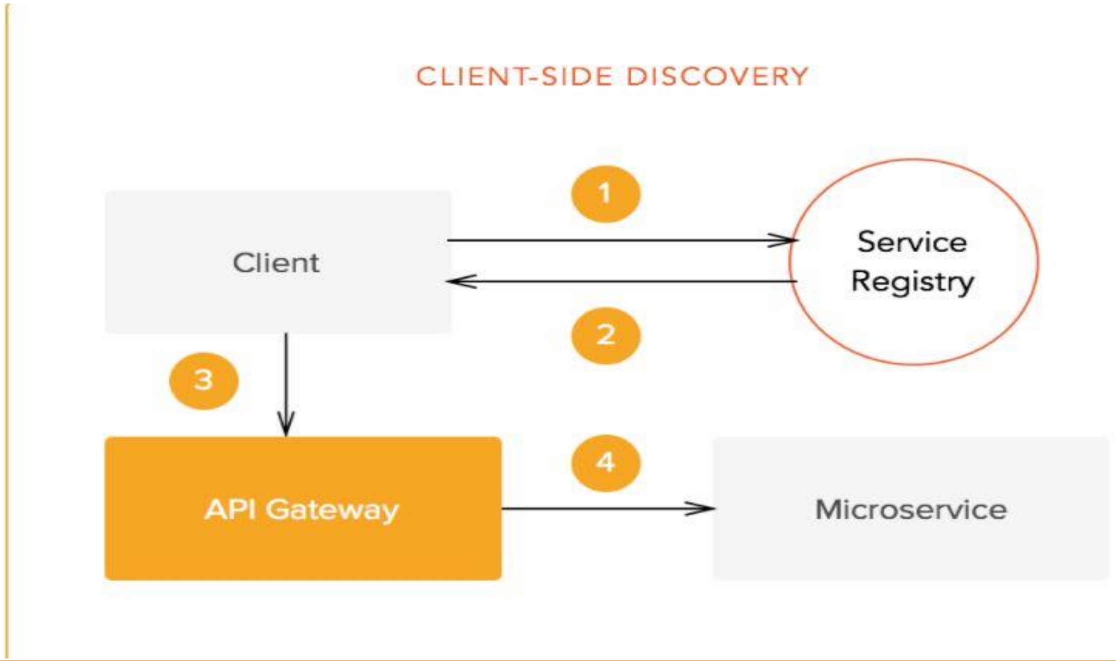
(b) Multiple API Gateway

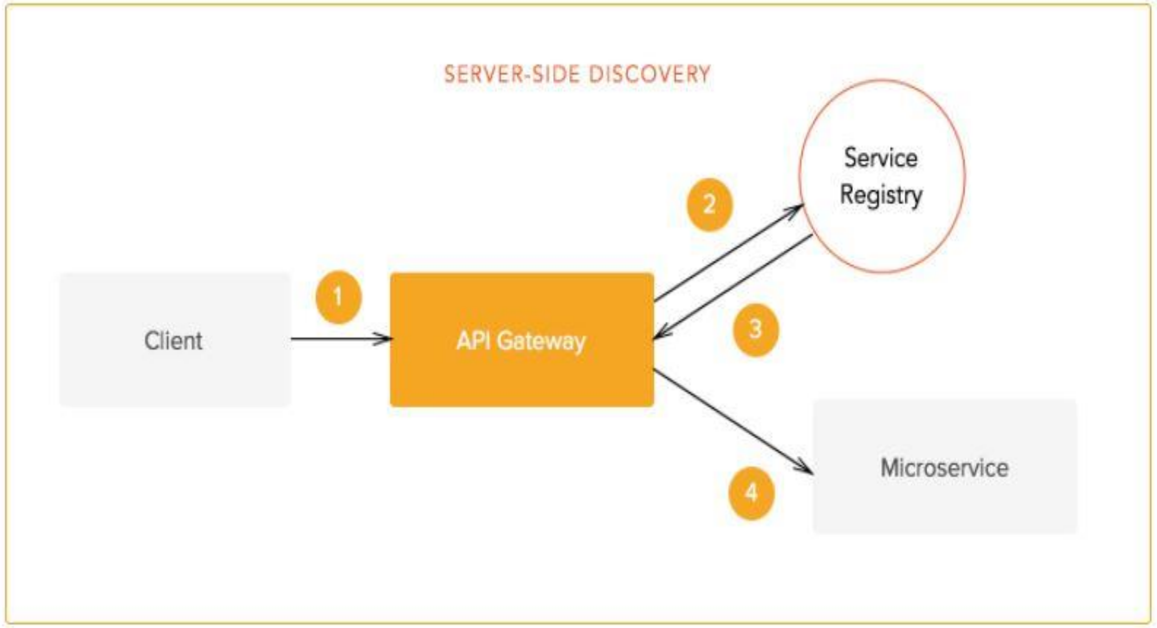
6. Service Registration and Discovery

- Helps to locate a service instance in a runtime environment. Information on how to dispatch requests to microservice instances.
- Locating services:
  Registration:- about who publishes or updates the information on how to reach each service. self registration: forces microservices to interact with the registry by themselves,
  third party registration : process or service that manages all other services
- Discovery When a client wants to access a service, it must find out where the service is located:-
  Eg: client-side discovery- forces clients to query a discovery service before performing the actual requests
  server side discovery- API gateway handle the discovery of the right endpoint (or endpoints) for a request


SELF-REGISTRATION
Microservice
Microservice
when going up / down
when going up / down
Service Registry

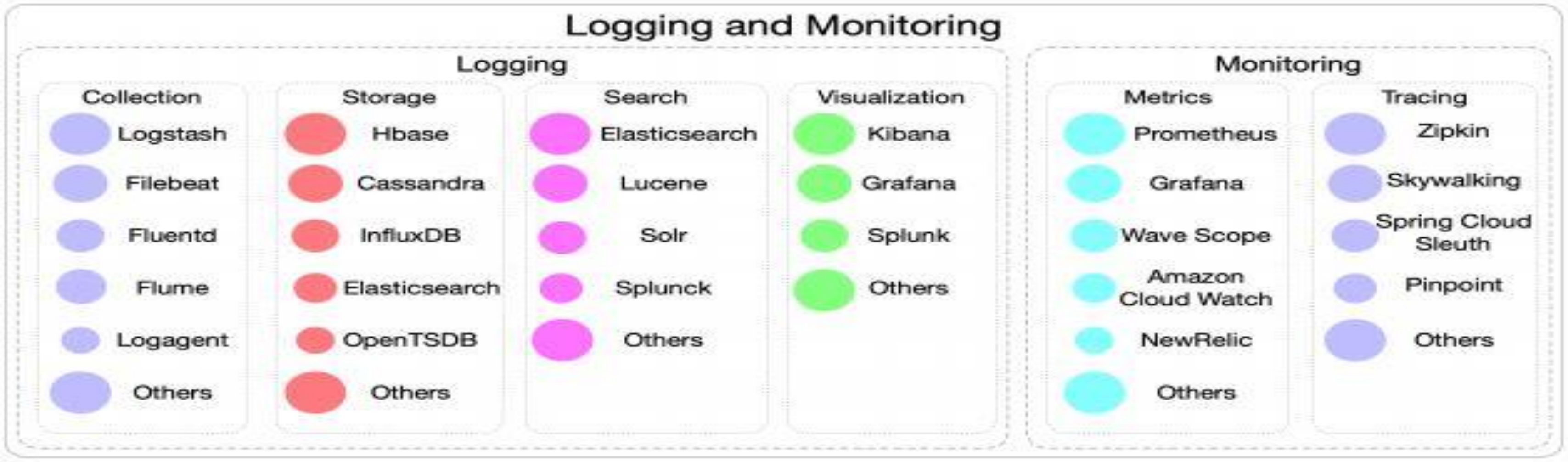

THIRD-PARTY REGISTRATION
Microservice
Microservice
Microservice
starts service
stops service
detects service crash
Service Manager
Service Registry


CLIENT-SIDE DISCOVERY
Client
1
2
Service Registry
3
API Gateway
4
Microservice


SERVER-SIDE DISCOVERY
Client
1
API Gateway
2
3
Service Registry
4
Microservice

7. Logging and Monitoring

- Helps to manage individual services using dashboards. Eg: Up/Down status, variety of operations, Business related metrics
- Logging and monitoring platforms are built with open-source systems
  Eg: ELK stack(Logstash for log collection, Elasticsearch for log indexing and retrieval, and
  Kibana for visualization.)



Logging and Monitoring

I1: Complex and Asynchronous Service Invocation Chain

Distributed tracing helps to pinpoint where the failures are occurred and what cause performance

Invasive and Non-invasive Tracing

By instrumenting probes or using network request in a service mesh

Challenges: High Cost, Fragility, and Latency

I2: Service Incidents are Hard to Detect.

Practice: Dashboard and Threshold

Conduct manual anomaly detection by analyzing the metric provided by dashboards (Prometheus, Grafana), setting thresholds to trigger predefined actions

Challenge: Insufficient Automation

Dashboards cannot display all metrics, need multiple monitoring platforms, hard to ensure accuracy and timeline thresholds

8. Performance and Availability Assurance

- Large number of calls between services and any request error could cause failures, and may lead to network latency and affect system performance.
- Strategies to handle the performance and availability issues:
  Timeout, Rate Limiters, Retry, and Circuit Breaker.
- Scaling Strategies: Horizontal and Vertical Scaling

I1: Inconsistency Across Stateful Services.
P1: Migrating States to External Storage
C1: High Refactoring Cost, Network Latency, and System Bottleneck

I2: Unpredictable and Uncontrollable Autoscaling strategy
P2: Semi-automated Scaling
C2: Predictable and Reliable Autoscaling

9. Testing

- Testing whether business logic meets the requirement and performance.
- Testing Approaches:

Unit testing: looking at each microservice at a granular level by breaking it into smaller testable functional parts

Integration testing: how well microservices connect, communicate, and interact as a whole system

Contract Testing: Contract testing simulates interaction scenarios between consumers and microservices API providers

Component testing: assessing how a microservice operates as a whole.

Testing

Fuzz Testing for JVM based web applications[7]:

1. Pick your target

   Get familiar with backend structure and architecture to recognize the interfaces where automated fuzz testing will more likely uncover critical vulnerabilities.

2. Enable your web service for fuzzing

   Fuzzer needs feedback from application under test. (JVM microservices adds Java agents)

   Eg: Code Intelligence Fuzz extension for VS-code

3. Configure the fuzz test

   Provide fuzzer with a description of API we want to test.

4. Add HTTP request to get Fuzzer started.

   Fuzzer operates on a simple HTTP request, easy to seed fuzzer with existing tooling like integration testing, API testing

5. Wait until all the bugs have been collected.

   All findings and exceptions will be listed with detailed feedback.

10. Fault Localization

Fault localization is a basic component in fault management systems, it identifies the reason for the fault.

Approaches for Fault Localization:
use logs for trouble shooting, monitoring tools(Distributed tracing), testing, remote debugging, and local debugging.

I1: Complex and Dynamic Service Interaction
P1: Local Debugging, Mock, Remote Debugging, Traffic Routing.
C1: Debugging Performance, Infrastructure Requirements, and Lack of Intelligence.

Proposed work:

MEPFLL (Microservice Error Prediction and Fault Localization), an approach of latent error prediction and fault localization for microservice applications by learning from system trace logs that is focus on predicting three common types of microservice application faults that are specifically relevant to microservice interactions and runtime environments, i.e., multi-instance faults, configuration faults, asynchronous interaction faults. [2]

Fault Localization Contd.

Proposed Work:
Delta Debugging is a methodology to automate the debugging of programs using a scientific approach of hypothesis-trial-result loop.

It starts with a failed test of a given program and the circumstances that may induce the failure.

Delta debugging then iteratively tests the program under different circumstances and determines the relevance of the circumstances to the failure based on the test results, until a minimal failure-inducing circumstance is found.

In each iteration, the circumstances are partitioned into subsets, and each subset and its complement are tested. If a subset or its complement makes the program fail, the potential failure-inducing circumstances are reduced and the delta debugging process proceeds to focus on the remaining circumstances and to reduce it further. [3]

11. Service Evolution

- Quality assessment of microservices
- Quality Assessment metrics:
  - -System Metrics: CPU, Memory, network latency, I/O, Thread
  - -Service Metrics: Query Per Second (QPS), Transaction Per Service (TPS), Error and exception, Success rate, Mean Time to Repair (MTTR)

I1: Evolution Compatibility

May have lots of services and update of a service may cause compatibility problem

Downward Compatibility and Upgrade Deadline

Provide multiple versions of a service API by adding a version prefix to the Uniform Resource Identifier. Usage of service APIs are monitored and eliminates the ones that are no longer used.

C1: High Maintenance Cost

Microservices Tools

Some of the popular tools,

- Operating Systems: Linux
- Programming Languages: Spring Boot, Elixir
- Tools for API Management and Testing: Postman, API Fortress
- Tools for Messaging: Apache Kafka, RabbitMQ
- Toolkits: Fabric8, Seneca
- Architectural Frameworks: Goa, Kong
- Tools for Orchestration: Kubernetes, Istio
- Tools for Monitoring: Prometheus, Logstash,
- Serverless Tools: Claudia, AWS Lambda

Microservices Security

Problems Faced in Microservices:

- User details might not be secure and also could be accessed by the 3rd party.
- Relying on a specific code reduces the flexibility of microservices.
- Security of individual microservice is on of the prominent problems in the architecture

Methods to secure microservices:

- Defense in Depth Mechanism – Apply a number of security layers to protect the services with most sensitive information.
- Tokens and API Gateway – Tokens are used to identify the user and are stored in the form of cookies. Data of tokens are needed to be encrypted. (Jason Web Format). Add an extra element to secure services through token authentication
- Distributed Tracing and Session Management:
  Distributed Tracing: Method to pinpoint the failures and identify the reason behind it.
  Session Management: Make the user data to be obtained from shared session storage.
- First Session and Mutual SSL: Data transferred between services will be encrypted.

AIOps, Microservices, and Cloud Platforms

Serverless: A fully managed system on a cloud platform. Advantages such as low maintenance cost, easy to scale, event driven architecture, pay-for-usage

Digital Modernization: Utilizing both microservices based architecture and serverless together.

Serverless can be utilized to asynchronous processing, scheduled jobs, ETL jobs.

Challenges arise:
1. Monitoring the high number of microservices
2. Identifying root cause for failure
3. Addressing the failure quickly
4. Testing across the various features.
5. Monitoring end-user conversions
6. Adapting to continuous upgrades and system changes.

Solution: Bringing Artificial Intelligence coupled with Machine Learning capabilities into DevOps will address new complexities in development, deployment, and APM

AIOps[5]

Four critical features needed for creating highly effective processes and systems:

1. **AIOps**: Analysis of the traffic, logs, usage with the help of machine learning, anomaly detection and alerting, and reliable root cause analysis

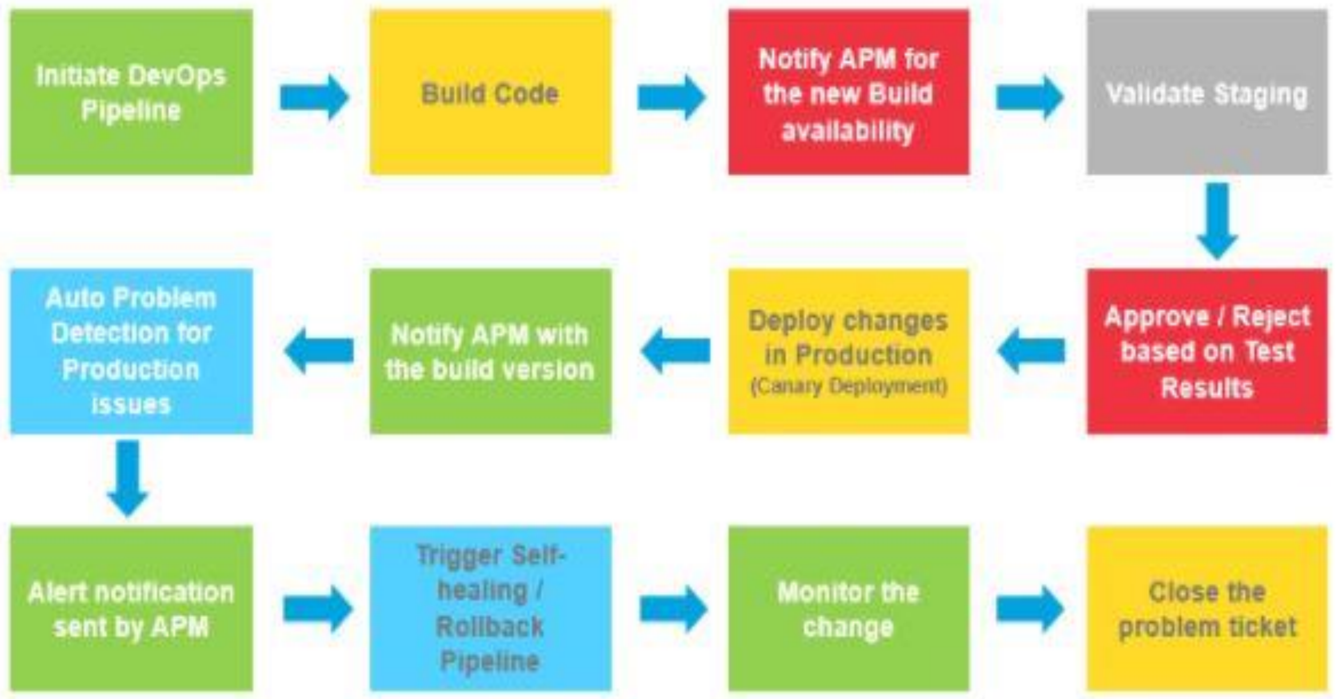
2. **Intelligence DevOps**: software quality is improved significantly with AI is driven performance and regression testing

3. **Remediation and Self Healing**: Auto-detect issues and alerts and trigger remediation and self-healing, provide prescriptive automation

4. **User Experience**: AIOps provide better insights for the usage of the system and measure conversions easily

AIOps Tools: Dynatrace, Cisco AppDynamics, New Relic

A typical workflow for automated remediation will look like this:


Initiate DevOps Pipeline
Build Code
Notify APM for the new Build availability
Validate Staging
Approve / Reject based on Test Results
Deploy changes in Production (Canary Deployment)
Notify APM with the build version
Auto Problem Detection for Production issues
Alert notification sent by APM
Trigger Self-healing / Rollback Pipeline
Monitor the change
Close the problem ticket

References

1. https://arxiv.org/abs/2106.07321
2. https://cspengxin.github.io/publications/fse19-zhou-microservice.pdf
3. https://cspengxin.github.io/publications/tsc19-deltadebugging.pdf
4. https://cspengxin.github.io/publications/tse19-msdebugging.pdf
5. https://dzone.com/articles/aiops-microservices-and-cloud-platforms
6. https://www.talend.com/resources/microservices-vs-soa/
7. https://blog.code-intelligence.com/fuzzing-microservices-in-5-steps

Tutorial on Microservices:
https://www.youtube.com/watch?v=tuJqH3AV0e8&t=10053s

Acknowledgement

Dr. Wei Yang