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?

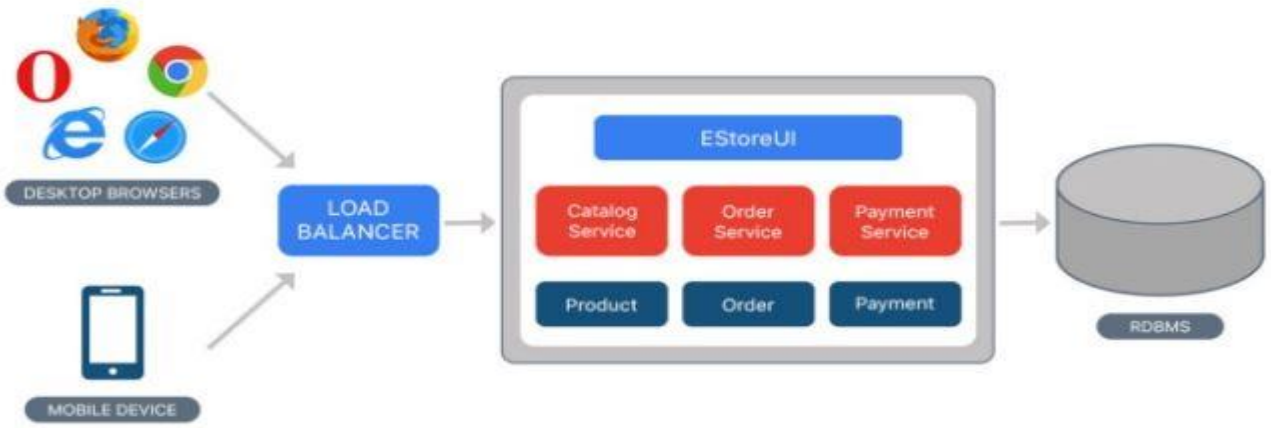
- Also know as Microservices Architecture
- An architectural style that structures an application as a suite of loosely coupled services, each of which has a single responsibility and can be deployed independently, scaled independently, and tested independently.[1]
- Microservices are individual software 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

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



Monolithic Architecture (for E-Commerce Application)

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

Microservices Architecture (for E-Commerce Application)

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

	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

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

I1: Decomposition Decision influences a lot.
P1: Domain Driven Design
C1: Domain Model and Artifact Mapping :- 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
P2: Capturing Service Dependency using Runtime Tracing
C2: High Cost and Low Coverage of Runtime Tracing

2. Database Decomposition

Strategies: By business capabilities, domain, horizontal and vertical decomposition
Can use: centralized databases, shared databases, no shared databases.

- Reason for shared databases between services:
  Business logic is severely coupled, reduce time for data synchronization and cascading queries.

I1: Data Coupling among Services
P1: Service Invocation Composition and Distributed Transaction. (Eg: Seata)
C1: Subsequent Refactoring and Network Latency

3. Deployment

- Solutions: Virtual Machines, Physical Machine, Containers, Virtual machines and Containers.
- Manage Containers using Kubernetes, Mesos, Docker Swarm, spring cloud.
- 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.

4. Service Communication Design

- Services must interact using an inter-process communication protocol such as HTTP, AMQP, and RPC.
- Communication styles: HTTPS/REST, RPC and Messaging
- Synchronous and Asynchronous patterns

5. API Gateway Design

- Application Program Interface(API) is a way which you can make sure two or more applications communicate with each other to process the client request.
- 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.

6. Service Registration and Discovery

- Helps to locate a service instance in a runtime environment.
- Locating services:
  Registration:- self registration, third party registration
  Discovery:- client-side discovery, server side discovery

7. Logging and Monitoring

- Helps to manage individual services using dashboards.
- 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.)
- Systems use self developed Application Performance Management (APM) system or commercial software.

I1: Complex and Asynchronous Service Invocation Chain
P1: Invasive and Non-invasive Tracing
C1: High Cost, Fragility, and Latency

I2: Service Incidents are Hard to Detect.
P2: Dashboard and Threshold
C2: Insufficient Automation

8. Performance and Availability Assurance

- 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 Approaches: Unit testing, Integration testing, stress testing, end-to-end testing, Component testing.

- Fuzz Testing for JVM based web applications[7]:
    1. Pick your target
    2. Enable your web service for fuzzing
    3. Configure the fuzz test
    4. Add HTTP request to get Fuzzer started.
    5. Wait until all the bugs have been collected.

10. Fault Localization

- 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 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
P1: Downward Compatibility and Upgrade Deadline
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

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

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