Microservices

Patterns . Tools . Technologies

Glarimy Technology Services www.glarimy.com | krishna@glarimy.com August 2018

Microservices

Patterns

Solution Architecture

Problem Statement

- Server-side enterprise application
- Variety of different clients
  - desktop browsers, mobile browsers, native mobile applications
- Northbound API for 3rd parties
  - REST API
- Enterprise Integration
  - Web services and/or message broker.
- HTML/JSON/XML responses
- *What's the application's deployment architecture?*

Forces

- Understandability and Maintainability
- Quick response to frequent business requirements: Agile
- 24/7 Availability and High Scalability
- Team dynamics: New faces, rotation
- Technology Adaptation

Monolithic Architecture

Problem Statement

- Server-side enterprise application
- Variety of different clients
  - desktop browsers, mobile browsers, native mobile applications
- Northbound API for 3rd parties
  - REST API
- Enterprise Integration
  - Web services and/or message broker.
- HTML/JSON/XML responses
- What's the application's deployment architecture?

Forces

- Understandability and Maintainability
- Quick response to frequent business requirements: Agile
- 24/7 Availability and High Scalability
- Team dynamics: New faces, rotation
- Technology Adaptation

Solution

- Tiered and Layered Architecture: UI+Service+Data, MVC, DAO
- One single distribution: WAR

Negative Impact

- Loaded IDE, Delayed deployments, Build Nightmares, Outdated Technologies

Microservice Architecture

Problem Statement

- Server-side enterprise application
- Variety of different clients
  - desktop browsers, mobile browsers, native mobile applications
- Northbound API for 3rd parties
  - REST API
- Enterprise Integration
  - Web services and/or message broker.
- HTML/JSON/XML responses
- What's the application's deployment architecture?

Forces

- Understandability and Maintainability
- Quick response to frequent business requirements: Agile
- 24/7 Availability and High Scalability
- Team dynamics: New faces, rotation
- Technology Adaptation

Solution

- Application as a set of loosely coupled collaborating services
- Polyglot databases and technologies
- Self-contained deployments
- Focussed Teams with CI/CD governance

Negative Impact

- Complexity associated with any distributed systems

Decomposition

Problem Statement

- Microservice Architecture
- Small, agile, autonomous and cross-functional teams
- Continuous Delivery and Deployment
- Independent Deployment
- Simplified Testing
- Polyglot Databases and Technology Stacks
- *How to decompose the application into services?*

Forces

- Architectural stability
- Loosely coupled and cohesive services
- Single Responsibility Principle
- Common Closure Principle

Decomposition by Business Capability

Problem Statement

- Microservice Architecture
- Small, agile, autonomous and cross-functional teams
- Continuous Delivery and Deployment
- Independent Deployment
- Simplified Testing
- Polyglot Databases and Technology Stacks
- *How to decompose the application into services?*

Forces

- Architectural stability
- Loosely coupled and cohesive services
- Single Responsibility Principle
- Common Closure Principle

Solution

- Discover services on the lines of business capability
- Core and Expandable Architecture
- Organization Structure
- High-Level Domain Model

Challenges

- Very difficult to discover services: There is no single way

Decomposition by Subdomains

Problem Statement

- Microservice Architecture
- Small, agile, autonomous and cross-functional teams
- Continuous Delivery and Deployment
- Independent Deployment
- Simplified Testing
- Polyglot Databases and Technology Stacks
- *How to decompose the application into services?*

Forces

- Architectural stability
- Loosely coupled and cohesive services
- Single Responsibility Principle
- Common Closure Principle

Solution

- Discover services on the lines of Domain Driven Design
- Core Subdomains: In-house development
- Supporting Subdomains: Outsourcing
- Generic Subdomains: Off the shelf solutions

Challenges

- Very difficult to discover services: Requires strong understanding of the business.

Deployment

Problem Statement

- Microservice architecture
- Smaller services
- Multiple instances for throughput
- Multiple instances for availability
- *How to package and deploy the services?*

Forces

- Heterogenous set of languages, frameworks & versions
- Multiple instances of each service
- Independent deployment
- Resource consumption constraints
- Monitoring
- Quick and frequent deployment

One Host - Many Service Instances

Problem Statement
- Microservice architecture
- Smaller services
- Multiple instances for throughput
- Multiple instances for availability
- *How to package and deploy the services?*

Forces
- Heterogenous set of languages, frameworks & versions
- Multiple instances of each service
- Independent deployment
- Resource consumption constraints
- Monitoring
- Quick and frequent deployment

Solution
- Multiple instances of service per host
  - Multiple JVMs on a host
  - One Tomcat per JVM
  - One or more services per Tomcat
- Leads to better utilisation of the host resources

Negative Impact
- Conflicting resource requirements
- Conflicting dependency versions
- Difficult demarcation
- Difficulty in monitoring individual services

One Host - One Service Instance

Problem Statement

- Microservice architecture
- Smaller services
- Multiple instances for throughput
- Multiple instances for availability
- *How to package and deploy the services?*

Forces

- Heterogenous set of languages, frameworks & versions
- Multiple instances of each service
- Independent deployment
- Resource consumption constraints
- Monitoring
- Quick and frequent deployment

Solution

- Single instance of service per host
    - Only one JVM on a host
    - One Tomcat per JVM
    - One service per Tomcat
- No conflict in resource requirements
- No dependency conflict
- Full control on demarcation
- Difficulty in monitoring individual services

Negative Impact

- Less efficient utilisation of the host resources

One VM - One Service Instance

Problem Statement

- Microservice architecture
- Smaller services
- Multiple instances for throughput
- Multiple instances for availability
- *How to package and deploy the services?*

Forces

- Heterogenous set of languages, frameworks & versions
- Multiple instances of each service
- Independent deployment
- Resource consumption constraints
- Monitoring
- Quick and frequent deployment

Solution

- Single instance of service per VM like EC2 instance
    - Only one JVM on a VM
    - One Tomcat per JVM
    - One service per Tomcat
- Services are isolated … better demarcation and no conflicts
- IaaS case: unmatched scaling

Negative Impact

- Building VM image is slow and time consuming

One Container - One Service Instance

Problem Statement

- Microservice architecture
- Smaller services
- Multiple instances for throughput
- Multiple instances for availability
- *How to package and deploy the services?*

Forces

- Heterogenous set of languages, frameworks & versions
- Multiple instances of each service
- Independent deployment
- Resource consumption constraints
- Monitoring
- Quick and frequent deployment

Solution

- Single instance of service per container
    - Only one JVM on a container
    - One Tomcat per JVM
    - One service per Tomcat
- Services are isolated … better demarcation and no conflicts
- Very good for scaling
- Easy deployment and relatively less load on the host

Negative Impact

- Relatively newer technology

Serverless Deployment

Problem Statement

- Microservice architecture
- Smaller services
- Multiple instances for throughput
- Multiple instances for availability
- *How to package and deploy the services?*

Forces

- Heterogenous set of languages, frameworks & versions
- Multiple instances of each service
- Independent deployment
- Resource consumption constraints
- Monitoring
- Quick and frequent deployment

Solution

- Public clouds with opaque and elastic infrastructure
- Deploy only the functionality
    - AWS Lambda, GC Funtions, Azure Functions
- No heavy lifting of image creation and deployment

Negative Impact

- Limited possibilities in terms of supported technologies
- Limited integration capabilities beyond REST API and MQTT
- Elastic, means no pre-provisioning
- Bad for long-duration services
- Vendor-locking

External API

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Multiple kinds of clients
- Client needs several services
- *How to expose the services to client?*

Forces

- Granularity of the API
- Different data needs from the same service
- Mobile Vs WAN Vs LAN performance
- Number and location of services varies
- Service partitioning/aggregation over a period of time
- Change in protocols

API Gateway

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Multiple kinds of clients
- Client needs several services
- *How to expose the services to client?*

Forces

- Granularity of the API
- Different data needs from the same service
- Mobile Vs WAN Vs LAN performance
- Number and location of services varies
- Service partitioning/aggregation over a period of time
- Change in protocols

Solution

- API Gateway as the single point of entry
- All call reach API Gateway
- The API Gateway proxies the requests to the actual services
- Event-driven mechansim (Node/Vertx/Reactor)

Negative Impact

- Impact of changes in service on the single API Gate way
- One-size-fits-all may not really true
- Latency

Backend to Frontend

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Multiple kinds of clients
- Client needs several services
- *How to expose the services to client?*

Forces

- Granularity of the API
- Different data needs from the same service
- Mobile Vs WAN Vs LAN performance
- Number and location of services varies
- Service partitioning/aggregation over a period of time
- Change in protocols

Solution

- Different API Gateways for different contexts (like client types)
- Specific calls reach specific API Gateway
- Specific API Gateway proxies the requests to the actual services
- Event-driven mechanism (Vertx/Node/Reactor)

Negative Impact

- Impact of changes in service on each of the API Gate ways
- Latency

Security

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Multiple instances of the services
- *How to secure the service?*

Forces

- Stateless calls
- No server-side sessions
- Ever-changing granularity of the services
- Client capabilities
- Social Networks

Access Tokens

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Multiple instances of the services
- *How to secure the service?*

Forces

- Stateless calls
- No server-side sessions
- Ever-changing granularity of the services
- Client capabilities
- Social Networks

Solution

- Access Tokens
- Client sends access token in every call
- Authentication at the gateway
- JSON Web Tokens or JWT
- Open Authentication

Cross Cutting Concerns

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Configuration
- Logging
- Monitoring
- Health Checks
- *How to deal with the cross cutting concerns across the application?*

Forces

- Quick development
- Optimal investment on something common across services

Chasis

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Configuration
- Logging
- Monitoring
- Health Checks
- *How to deal with the cross cutting concerns across the application?*

Forces

- Quick development
- Optimal investment on something common across services

Solution

- Build a chassis
- Spring Boot Actuator
    - /actuator
- External Log Storage, Log Aggregation, Correlation IDSs
    - AWS Cloudwath
    - LogStash

Externalisation

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Configuration
- Logging
- Monitoring
- Health Checks
- *How to deal with the cross cutting concerns across the application?*

Forces

- Quick development
- Optimal investment on something common across services

Solution

- Build a chassis
- Spring Boot Actuator
  - /actuator
- External Log Storage, Log Aggregation, Correlation IDSs
  - AWS Cloudwath
  - LogStash
- External Configuration
  - Docker Compose supplies the configuration as OS variables

Collaboration

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Collaboration
- *How two microservices of the application communicate with each other?*

Forces

- Frequent Communication
- Heavy payloads
- Binary payloads
- Synchronous and Asynchronous
- Coupling

Remote Procedure Invocation

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Collaboration
- *How two microservices of the application communicate with each other?*

Forces

- Frequent Communication
- Heavy payloads
- Binary payloads
- Synchronous and Asynchronous
- Coupling

Solution

- Use Request/Reply RPI calls
- REST: HTTP/1
- gRPC: HTTP/2 with duplex communication
- Apache Thrift with binary payloads

Negative Impact

- No support for notifications, pub/sub
- Reduced availability: If one goes down, the other goes down
- Difficulty associated with discovering other service

Messaging

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Collaboration
- *How two microservices of the application communicate with each other?*

Forces

- Frequent Communication
- Heavy payloads
- Binary payloads
- Synchronous and Asynchronous
- Coupling

Solution

- Use messaging for inter-service communication
- Data-pipelining with Apache Kafka: publish/subscribe, fault-tolerant, stream processing
- Rabbit MQ: Publish/subscribe

Negative Impact

- Additional Complexity with yet another technology
- Clients need to discover message broker
- General complexity associated with asynchronous communication

Data Management

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Common data domain
- What is the database architecture?

Forces

- Services must be loosely coupled
- Business transactions spanning across services
- Queries across services
- Data joining across several services
- Replication of data for scaling
- Different storage requirements

Shared Database

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Common data domain
- What is the database architecture?

Forces

- Services must be loosely coupled
- Business transactions spanning across services
- Queries across services
- Data joining across several services
- Replication of data for scaling
- Different storage requirements

Solution

- Use single database server for the whole application
- Stick to ACID transactions

Negative Impact

- Development-time coupling
- Run-time coupling
- Chosen database may not satisfy the storage requirements of some of the services
- Scaling issues

Database per Service

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Common data domain
- What is the database architecture?

Forces

- Services must be loosely coupled
- Business transactions spanning across services
- Queries across services
- Data joining across several services
- Replication of data for scaling
- Different storage requirements

Solution

- Each service have it's own database
- Enforce demarcation, may be using different user-ids for different services
  - Tables per service
  - Schema per service
  - Database Server per service

Negative Impact

- Cross-service transaction are still complex
- Joining data across services is still challenging

Command-Query Segregation

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Common data domain
- What is the database architecture?

Forces

- Services must be loosely coupled
- Business transactions spanning across services
- Queries across services
- Data joining across several services
- Replication of data for scaling
- Different storage requirements

Solution

- Let the services of the application divide based on the responsibility
    - Services that insert/delete/update records (commanding service)
    - Services that query the records (querying service)
- Let the commanding service fire data-change events
- Let the querying service subscribe to the events and update their views
- De-normalized databases

Negative Impact

- Increased complexity
- Replication lag
- Eventual consistency

Event Sourcing

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Common data domain
- What is the database architecture?

Forces

- Services must be loosely coupled
- Business transactions spanning across services
- Queries across services
- Data joining across several services
- Replication of data for scaling
- Different storage requirements

Solution

- Let the commands result in only state-change events
- Let the queries build the state of the entity by replaying the events
- Inherently atomic
- Use it with command-query segregation
- Provision of snapshots … to reduce amount replay
- Suitable for event driven systems
- Temporal audit

Negative Impact

- Relatively newer technology
- Increased complexity

API Composition

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Common data domain
- Database per service
- What is the query join strategy?

Forces

- Services must be loosely coupled
- Business transactions spanning across services
- Queries across services
- Data joining across several services
- Replication of data for scaling
- Different storage requirements

Solution

- Let an API composer join the query results in-memory

Negative Impact

- Very in-efficient

Service Discovery

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Service collaboration
- How a client service discover a provider service?

Forces

- Each instance of a service runs at a specific location (host & Port)
- Number of services/instance and locations change dynamically
- Containers are assigned with dynamic IP addresses

Service Registry

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Service collaboration
- How a client service discover a provider service?

Forces

- Each instance of a service runs at a specific location (host & Port)
- Number of services/instance and locations change dynamically
- Containers are assigned with dynamic IP addresses

Solution

- Implement a naming registry of the services
- Register a service location against a name at the service startup
- Deregister the service location at the service shutdown
- Registry polls the services time to time
- Several readymade offerings
    - Netflix Eureka with Spring Boot
    - Consul
    - etcd
    - Apache ZooKeeper

Challenges

- Registry must be highly available
- Fixed location of the registry service must be known to the clients

Self Registration

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Service collaboration
- Service Registry is implemented
- How a client registers with the registry?

Forces

- Instances must be registers at startup
- Instance must be deregistered at shutdown
- Crashed instances must be removed from the registry
- Incapable instances must be identified and removed from the registry

Solution

- Instances registers and deregisters themselves
- Instances periodically renew the registration (heart-beats)
- Instances can provide fine granular states (STARTING, STARTED, PAUSED …)
- Build as part of chassis

Challenges

- Coupling between service and registry
- Incapable service may not aware the reality
- Chassis for various platforms

Registrar

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Service collaboration
- Service Registry is implemented
- How a client registers with the registry?

Forces

- Instances must be registers at startup
- Instance must be deregistered at shutdown
- Crashed instances must be removed from the registry
- Incapable instances must be identified and removed from the registry

Solution

- A 3rd party registrar takes note of start/shutdown of services and updates the register
- A service composer takes this responsibility
  - Eg. Kubernetes
  - Eg. Netflix Prana for non-JVM applications
- Individual services are free from working with the registers
- Build as part of chassis

Challenges

- Registrar can have only superficial knowledge of the state of the service
- Adds one more crucial component to the infrastructure

Client-side Discovery

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Service collaboration
- Service Registry is implemented
- How a client discover the service?

Forces

- Each instance of a service runs at a specific location (host & Port)
- Number of services/instance and locations change dynamically
- Containers are assigned with dynamic IP addresses
- Registry is having the info of the services

Solution

- Let the client queries the registry to find the service
- Lesser number of variables

Challenges

- Apart from the services, the clients are also coupled with the registry
- Each language should have the chassis … for all the micro services

Server-side Discovery

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Service collaboration
- Service Registry is implemented
- How a client discover the service?

Forces

- Each instance of a service runs at a specific location (host & Port)
- Number of services/instance and locations change dynamically
- Containers are assigned with dynamic IP addresses
- Registry is having the info of the services

Solution

- Let the load balancer query the registry to find the service, on behalf of the client
- No coupling between the client and registry

Challenges

- More moving parts
- More protocols to be supported by the API Gateway
- Each language should have the chassis … for all the micro services

Reliability

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Service collaboration
- Service Registry is implemented
- How to mitigate if a service is unavailable?
- How to prevent cascading affect?

Forces

- A depending service may be unavailable
- A depending service may be taking too much time
- A client retrying for the failed service consumes resources

Circuit Breaker

Problem Statement

- Microservice architecture
- Smaller and multiple services
- Service collaboration
- Service Registry is implemented
- How to mitigate if a service is unavailable?
- How to prevent cascading affect?

Forces

- A depending service may be unavailable
- A depending service may be taking too much time
- A client retrying for the failed service consumes resources

Solution

- Wrap the calls to the service with a monitor
- The monitor works like electric circuit breaker
- When the monitor is CLOSED, the calls are passed to the service
- When the monitor is OPEN, the calls returned to the client with error
- When the monitor is HALF-OPEN, the calls attempted on the service
    - If service responds, monitor CLOSES, otherwise OPENS
- Timeout/threshold to move from CLOSED to OPEN
- Timeout to move from OPEN to HALF-OPEN
- NetFlix Hystrix

Challenges

- Choosing the timeouts/thresholds
- Yet another cog in the wheel

Microservices

Tools and Technologies

Spring Boot

Eureka . Hysterix . Ribbon . Circuit Breaker

Glarimy Technology Services www.glarimy.com | krishna@glarimy.com August 2018

Microservices

Tools and Technologies

Docker Containers

Microservices

Tools and Technologies

Docker Compose

Microservices

Tools and Technologies

CI/CD

Git. Jenkins . Pipelines . Dockers

Glarimy Technology Services www.glarimy.com | krishna@glarimy.com August 2018

Case Study

Problem Statement

- An NMS system is to be built for managing about 5000 printers of a customer
- IT department of the customer uses this NMS to install new software on the printers and to monitor various parameters of the printers.
- The NMS comprises of about 100 agents which takes care of SNMP communication with the printers.
- The agents communicate with the NMS server on behalf of the printers.
- Discovery, heart-beats, error-reporting and etc generate huge traffic between the printers & the NMS.
- Also, every operation on NMS and/or printers needs to be logged which runs into millions of log records every day.

Architectural Requirements:

1. The printers and agents run within the intranet of the customer
2. The NMS components run on the cloud
3. Number of printers and thereby the agents may vary
4. The volume of operations and logs vary between extremes during a week
5. The databases and logs are to be made available for other 3rd party reporting & data analytic applications
6. Provision must be available for managing future or 3rd party printers
7. The UI is to be made available both on mobile and desktop web browser with different capabilities

Thanks

Glarimy Technology Services, Bengaluru, India
www.glarimy.com | https://github.com/glarimy
Krishna Mohan Koyya
krishna@glarimy.com