BUILDING MICROSERVICES

Motivation for Microservices

- Agility
  - Code refactoring
  - Big ball of mud
  - Slower release cycles
- Brittle
- Vertical scaling
- Configuration drift

Microservices

- Architecture style
- Structure an application into collection of services
  - Loosely coupled
  - Independently deployable
  - Organized into smaller teams

Challenges

- Decomposition of Monolith
- Configuration management
- Discovering Services
- Resiliency patterns
- Transactions
- Security
- Availability
- Monitoring and Tracing

Decomposition

→ Business capability

→ Domain driven design

Configuration management

- Centralized configuration
    - Eliminates configuration drift
- Options
    - Spring cloud config server
    - Kubernetes ConfigMaps

Discovering Services

- Enables load balancing
- Allows horizontal scaling of microservices
- Monitor health of each microservices and keep the pool of healthy microservices
- Options
  - Spring Cloud Eureka Server
  - Kubernetes Ingress Services

Resiliency Patterns

- → Prevents failing of downstream API’s
- → Allows for Bulk Head
- → Provides time for failing service to recover
- → Can optionally send a fallback implementation
- → Options
  - → Hystrix server

Transactions

- → Do not provide consistency
- → Prefer AP over C in the CAP theorem
- → Eventual consistency using Sagas
- → Database split across multiple microservices

Security

- Token based security
- Oauth 2

Availability

- API Gateway
- Common entry point
  - Assigned to the domain name
  - Rate limit
  - Enforcing policies
  - Aggregator
- Options
  - Zuul API Gateway
  - Kubernetes Ingress
  - Istio Service Mesh

Monitoring and Traceability

- Tracing HTTP requests
- Aggregating the logs to centralized server
- Set up monitoring and alerting mechanisms
- Options
  - Sleuth and Zipkin
  - ELK
  - Istio Service Mesh

Deployment

- Deploying microservices
  - Blue-green deployment
  - Canary deployment
  - Rolling deployment
- Options
  - Kubernetes Deployment
  - Istio