Spring Cloud로 개발하는 마이크로서비스 애플리케이션

+

강의 소개

Part I

- *Section 0: Microservice와 Sprig Cloud 소개*
- *Section 1: Service Discovery*
- *Section 2: API Gateway Service*
- *Section 3: E-commerce 애플리케이션*
- *Section 4: Users Microservice - ①*
- *Section 5: Catalogs, Orders Microservice*
- *Section 6: Users Microservice - ②*
- *Section 7: Configuration Service*
- *Section 8: Spring Cloud Bus*

Part II

- *Section 9: 암호화 처리를 위한 Encryption과 Decryption*
- *Section 10: 마이크로서비스간 통신*
- *Section 11: 데이터 동기화를 위한 Kafka 활용 ①*
- *Section 12: 데이터 동기화를 위한 Kafka 활용 ②*
- *Section 13: 장애 처리와 Microservice 분산 추적*
- *Section 14: Microservice 모니터링*
- *Section 15: 애플리케이션 배포를 위한 컨테이너 가상화*
- *Section 16: 애플리케이션 배포 – Docker Container*
- *Appendix: Microservice 패턴*

njone company

Section 1.

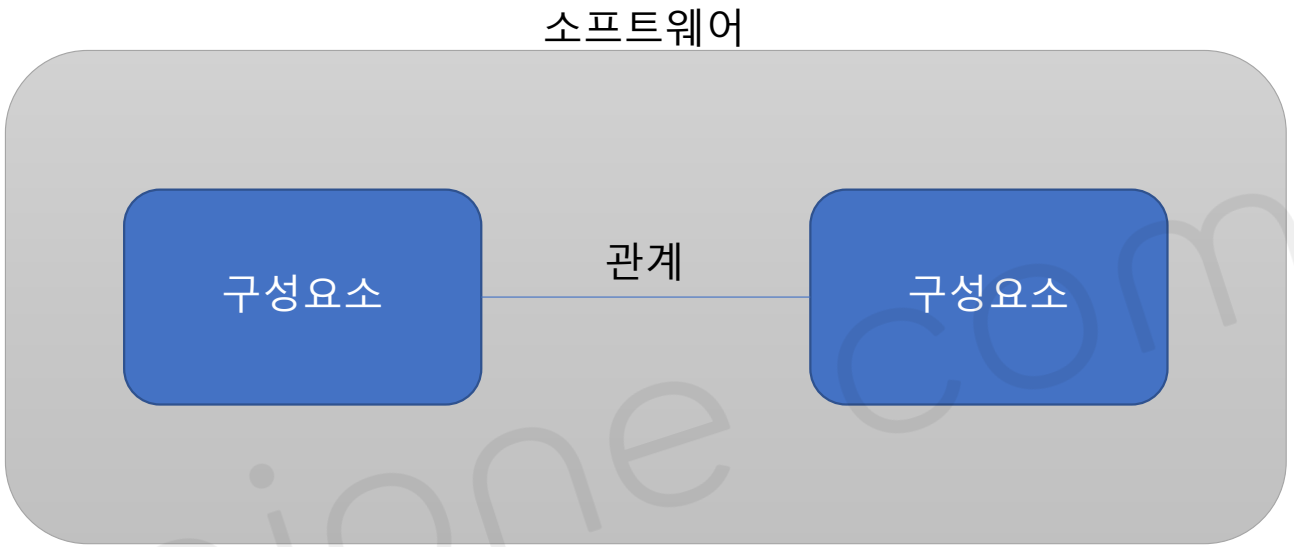
Cloud Native와 Microservice

- Cloud Native
- Microservice
- Spring Cloud
- 필수 SW 설치

njone company

Software Architecture

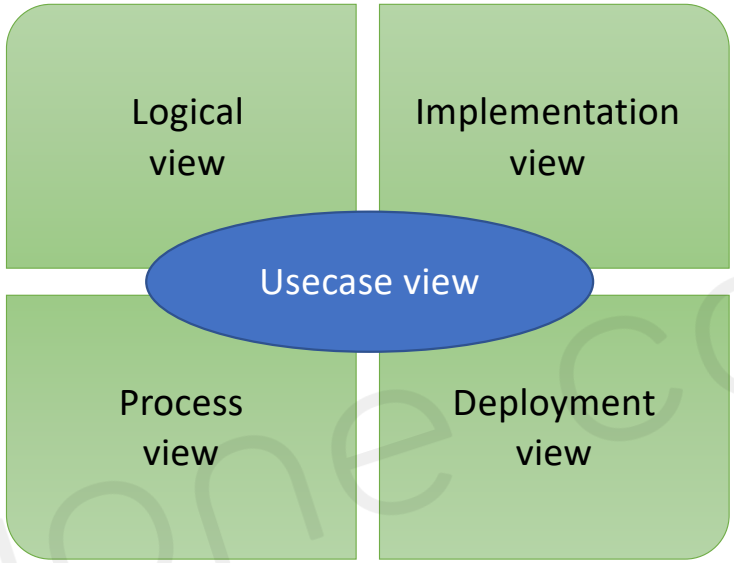
- SW를 구성하는 요소와 요소 간의 관계 정의
  - 전체 구성 관계, 포함 관계, 호출 관계
  - SW 설계자, 개발, 사용자 등 이해 관계자들간의 커뮤니케이션 도구



njone company

Software Architecture

- Architecture
  - 이해관계자들이 시스템을 이해하는 수준은 모두 다름 → 관점 → View
  - UML(Unified Modeling Language)



njone company

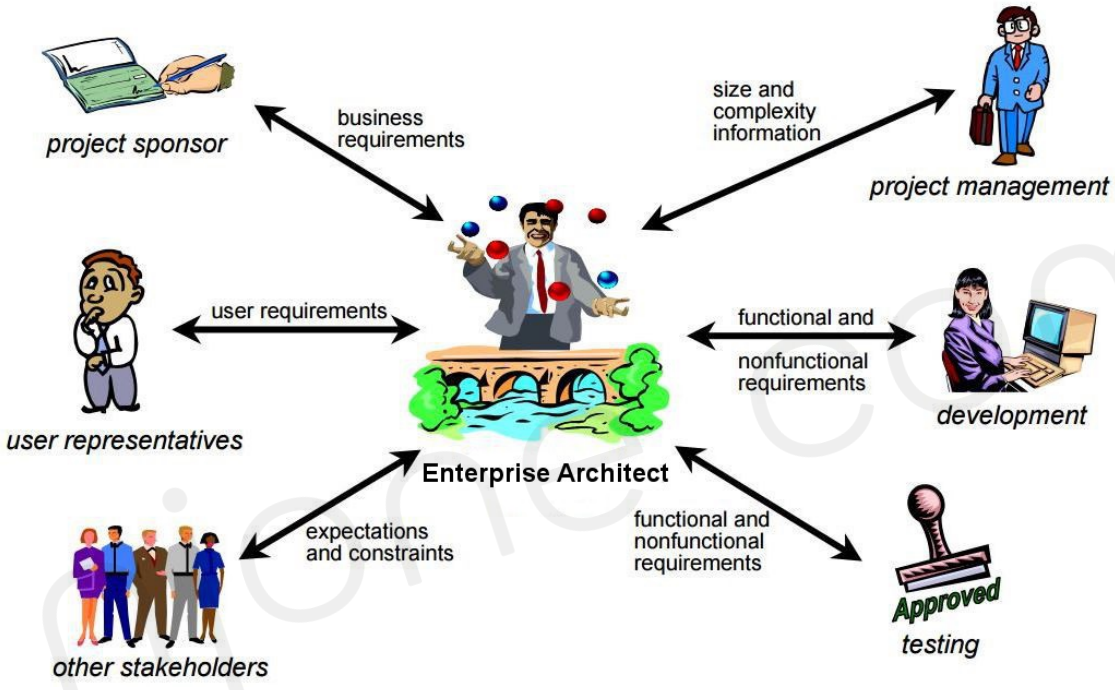
Software Architecture

- Architecture의 역할

"Architecture is both the ***process*** and the product of ***planning***, ***designing***, and ***constructing*** buildings or any other structures. Architectural works, in the material form of buildings, are often perceived as cultural symbols and as works of art. ***Historical civilizations*** are often identified with their surviving ***architectural achievements***. "

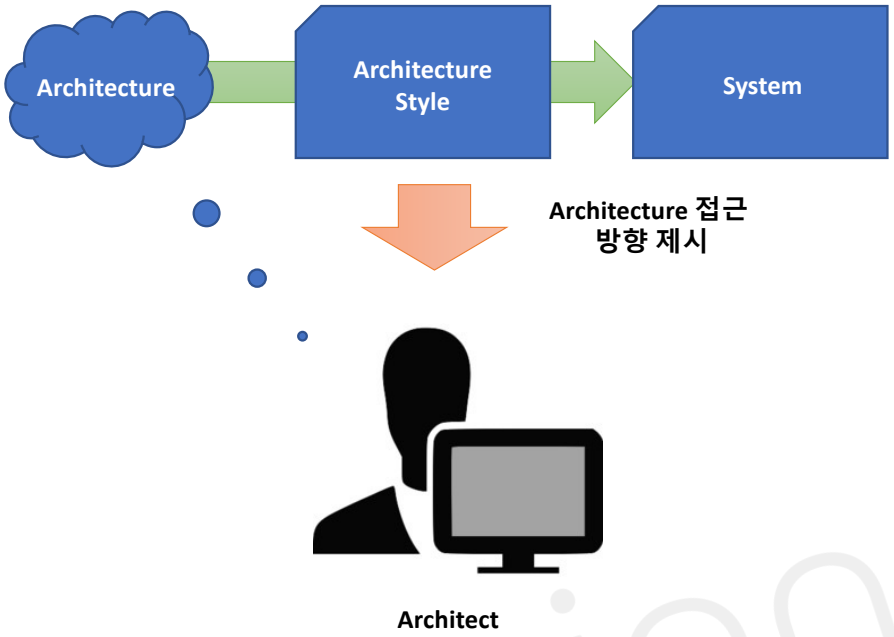
Software Architecture

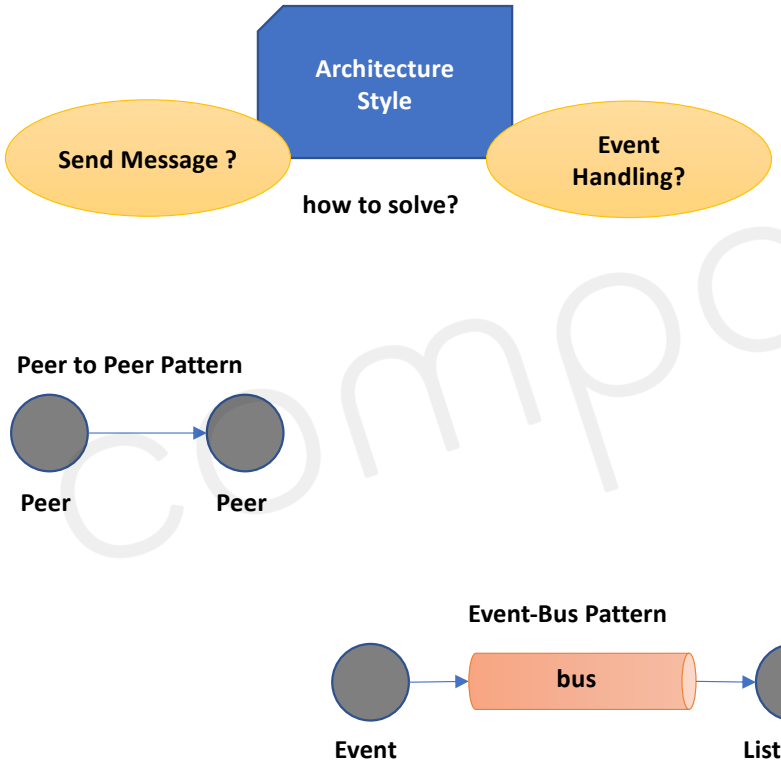
- Architecture의 역할
  - 무형의 지식 집합체(코드)
  - 기능적인 측면 보다는 안정적인 운영을 위한 전체적인 구조를 설계 ex) 오케스트라



Software Architecture

- Architecture 스타일과 패턴





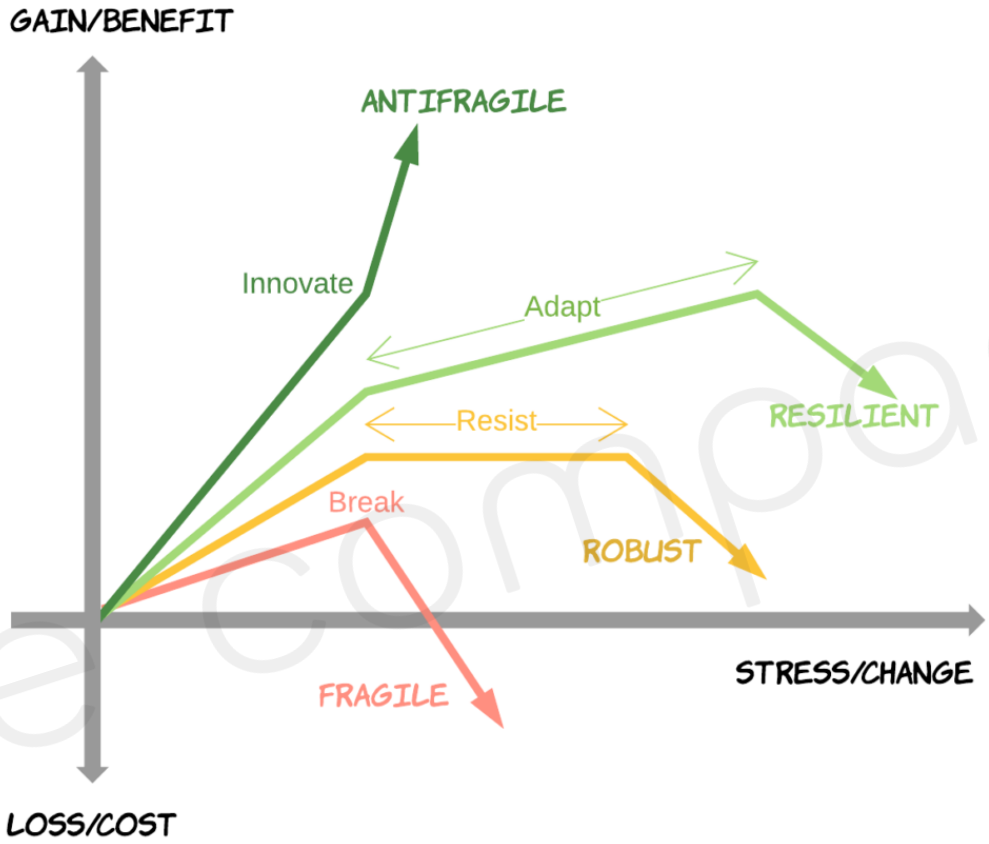
njone company

Software Architecture

- ***The History of IT System***
  - 1960 ~ 1980s : ***Fragile, Cowboys***
    - Mainframe, Hardware
  - 1990 ~ 2000s : ***Robust, Distributed***
    - Changes
  - 2010s ~ : ***Resilient/Anti-Fragile, Cloud Native***
    - Flow of value의 지속적인 개선

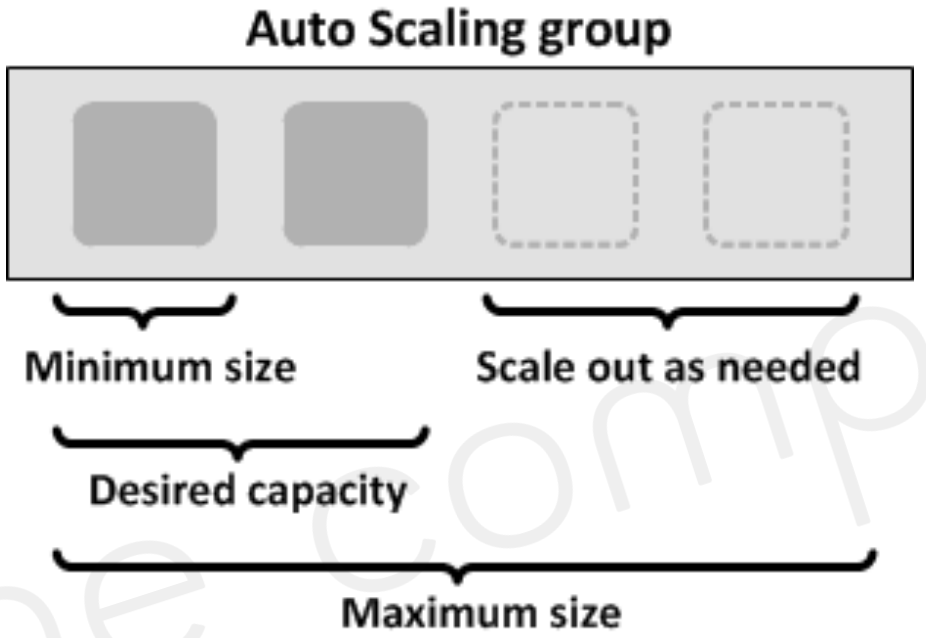
A different kind of structure and culture

Domain	Fragile	Robust	Anti-Fragile
Transport	Racing car	Tank, 4x4	Horse
Transport in London	Train	Bus	Bicycle
Market	Stock exchange	Supermarket	Bazaar
Knowledge	Science book	Journal	Wiki
IT Culture	Cowboys	ITIL	**DevOps**
IT Architecture	Monolithic	Distributed	**Cloud Native**



Software Architecture

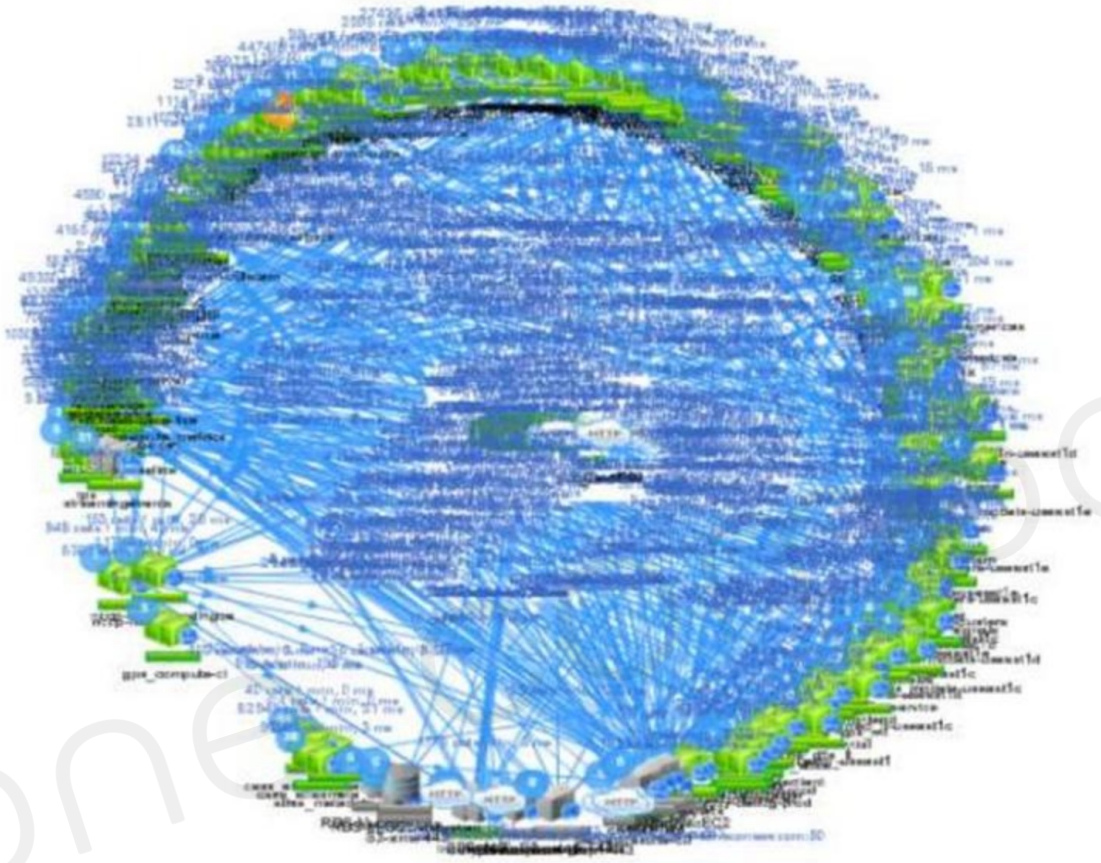
- ***Antifragile***
  - ***Auto scaling***
  - *Microservices*
  - *Chaos engineering*
  - *Continuous deployments*



njone company

Software Architecture

- ***Antifragile***
  - *Auto scaling*
  - ***Microservices***
  - *Chaos engineering*
  - *Continuous deployments*



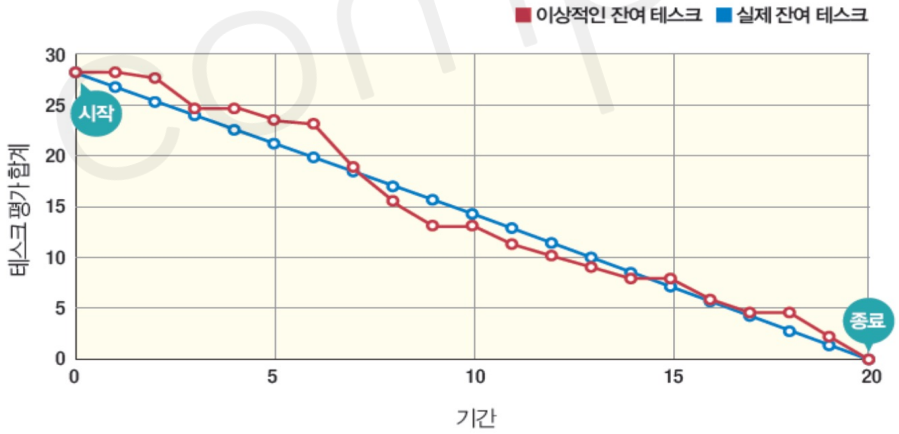
njone company

Software Architecture

- ***Antifragile***
  - *Auto scaling*
  - *Microservices*
  - ***Chaos engineering***
  - *Continuous deployments*

스토리	작업 목록	진행사항	변경사항	종료
As a User, I... 8 points	Code the... 9 Test the... 8 Code the... 2 Code the... 5 Code the... 8 Test the... 4	Code the... DC 4 Test the... SC 5	Test the... SC 6	Code the... DC Test the... 6 Test the... Test the... SC Test the... SC 6
As a User, I... 5 points	Code the... 5 Test the... 8 Code the... 4 Code the... 6	Code the... DC 5		Test the... SC Test the... SC Test the... SC 6

- *변동*
- *예견된 불확실성*
- *예견되지 않는 불확실성*
- *카오스 불확실성*



Software Architecture

- ***Antifragile***
  - *Auto scaling*
  - *Microservices*
  - *Chaos engineering*
  - ***Continuous deployments***

njonecompany

Cloud Native Architecture

- ***확장 가능한 아키텍처***
  - *시스템의 수평적 확정에 유연*
  - *확장된 서버로 시스템의 부하 분산, 가용성 보장*
  - *시스템 또는, 서비스 애플리케이션 단위의 패키지 (컨테이너 기반 패키지)*
  - *모니터링*
- 탄력적 아키텍처
  - *서비스 생성 - 통합 - 배포, 비즈니스 환경 변화에 대응 시간 단축*
  - *분활 된 서비스 구조*
  - *무상태 통신 프로토콜*
  - *서비스의 추가와 삭제 자동으로 감지*
  - *변경된 서비스 요청에 따라 사용자 요청 처리 (동적 처리)*
- 장애 격리 (Fault isolation)
  - *특정 서비스에 오류가 발생해도 다른 서비스에 영향 주지 않음*

njone company

Cloud Native Architecture

- 확장 가능한 아키텍처
  - *시스템의 수평적 확정에 유연*
  - *확장된 서버로 시스템의 부하 분산, 가용성 보장*
  - *시스템 또는, 서비스 애플리케이션 단위의 패키지(컨테이너 기반 패키지)*
  - *모니터링*
- ***탄력적 아키텍처***
  - *서비스 생성- 통합- 배포, 비즈니스 환경 변화에 대응 시간 단축*
  - *분활 된 서비스 구조*
  - *무상태 통신 프로토콜*
  - *서비스의 추가와 삭제 자동으로 감지*
  - *변경된 서비스 요청에 따라 사용자 요청 처리(동적 처리)*
- 장애 격리 (Fault isolation)
  - *특정 서비스에 오류가 발생해도 다른 서비스에 영향 주지 않음*

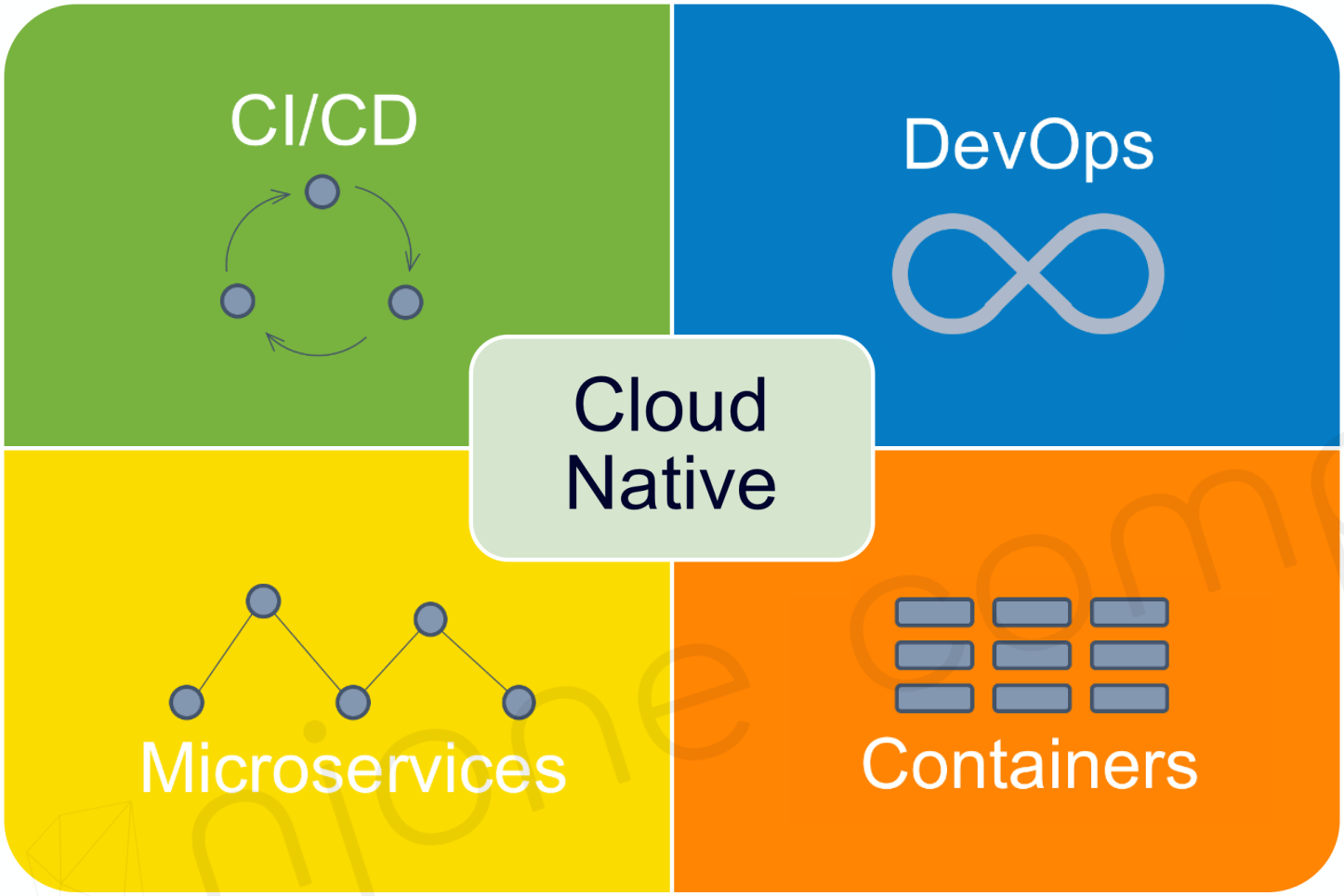
njone company

Cloud Native Architecture

- 확장 가능한 아키텍처
  - *시스템의 수평적 확정에 유연*
  - *확장된 서버로 시스템의 부하 분산, 가용성 보장*
  - *시스템 또는, 서비스 애플리케이션 단위의 패키지 (컨테이너 기반 패키지)*
  - *모니터링*
- ***탄력적 아키텍처***
  - *서비스 생성 - 통합 - 배포, 비즈니스 환경 변화에 대응 시간 단축*
  - *분활 된 서비스 구조*
  - *무상태 통신 프로토콜*
  - *서비스의 추가와 삭제 자동으로 감지*
  - *변경된 서비스 요청에 따라 사용자 요청 처리 (동적 처리)*
- ***장애 격리 (Fault isolation)***
  - *특정 서비스에 오류가 발생해도 다른 서비스에 영향 주지 않음*

njone company

Cloud Native Application



njone company

Cloud Native Application

Microservices

A microservices architecture is an application development approach in which a large application is built as a suite of modular components or services

Containerization

Containers are a type of software that can virtually package and isolate applications for deployment

Continuous delivery

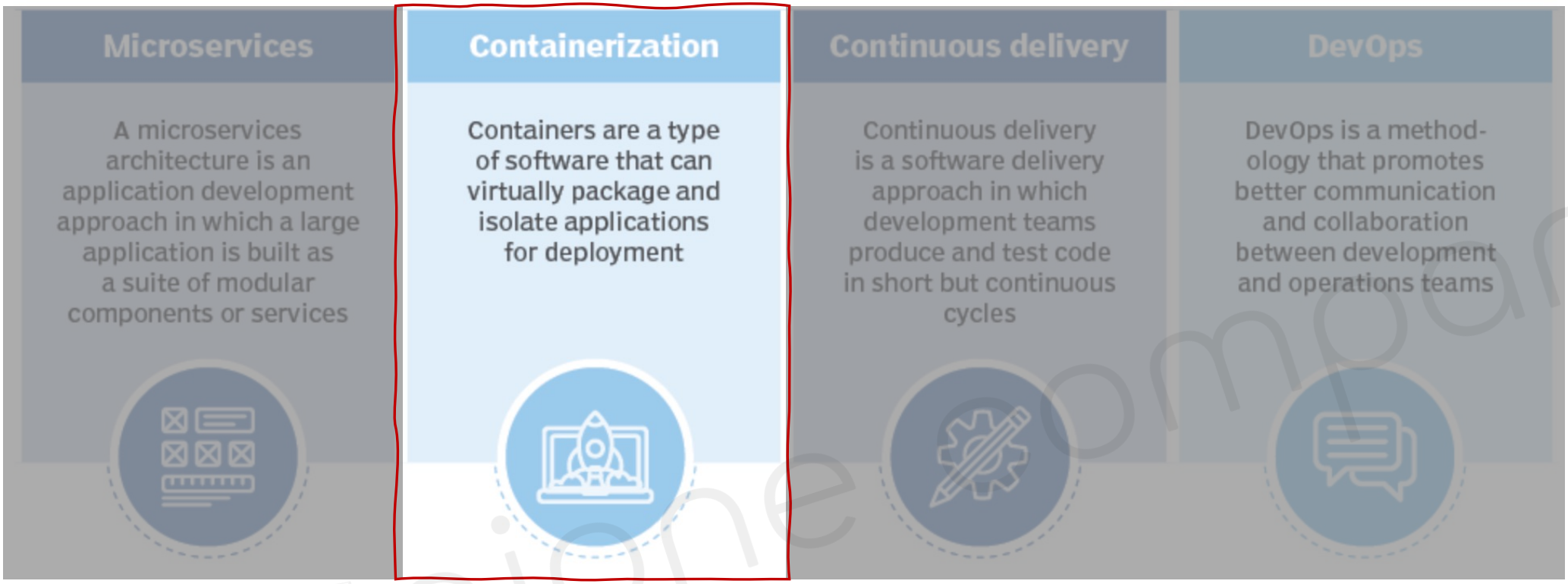
Continuous delivery is a software delivery approach in which development teams produce and test code in short but continuous cycles

DevOps

DevOps is a method-ology that promotes better communication and collaboration between development and operations teams

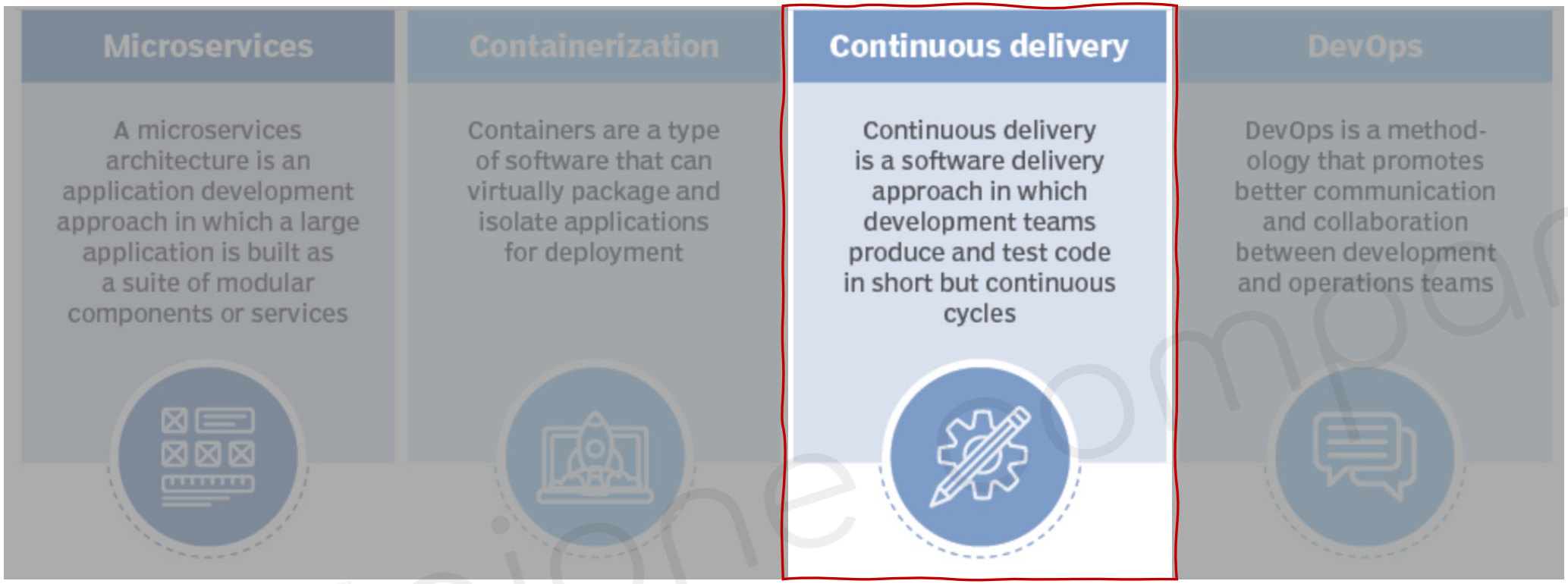
njone company

Cloud Native Application



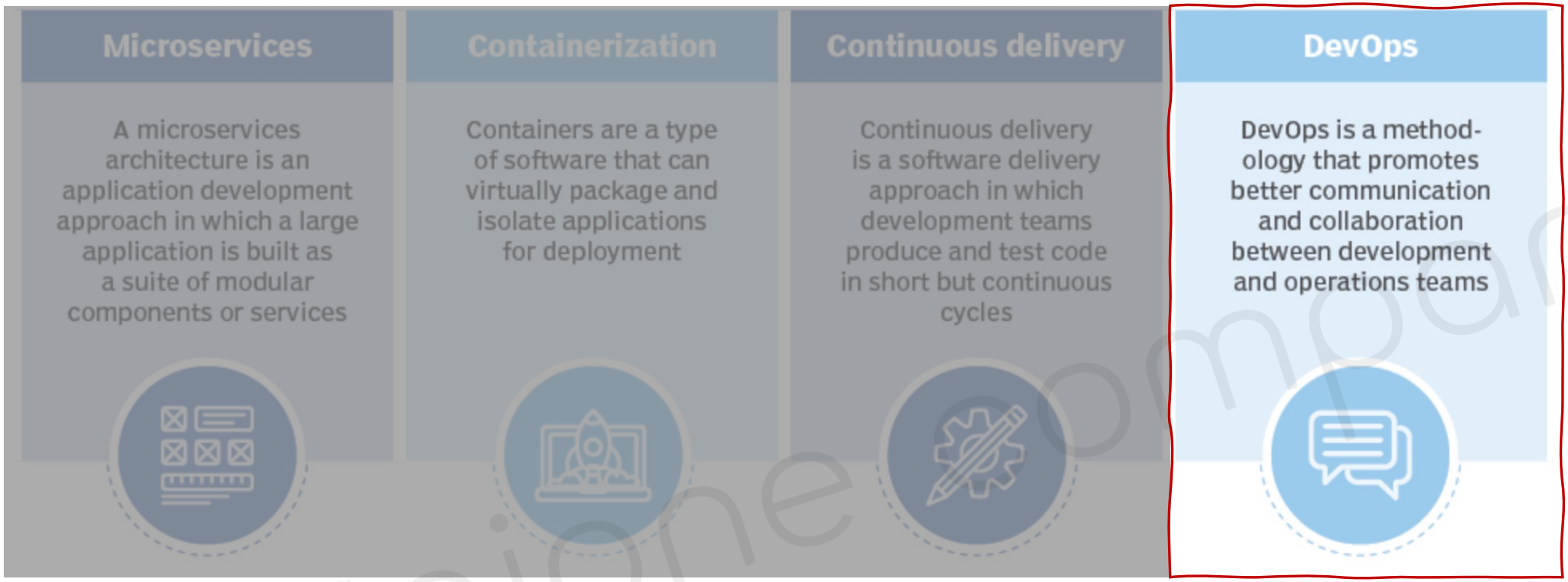
njone company

Cloud Native Application



njone company

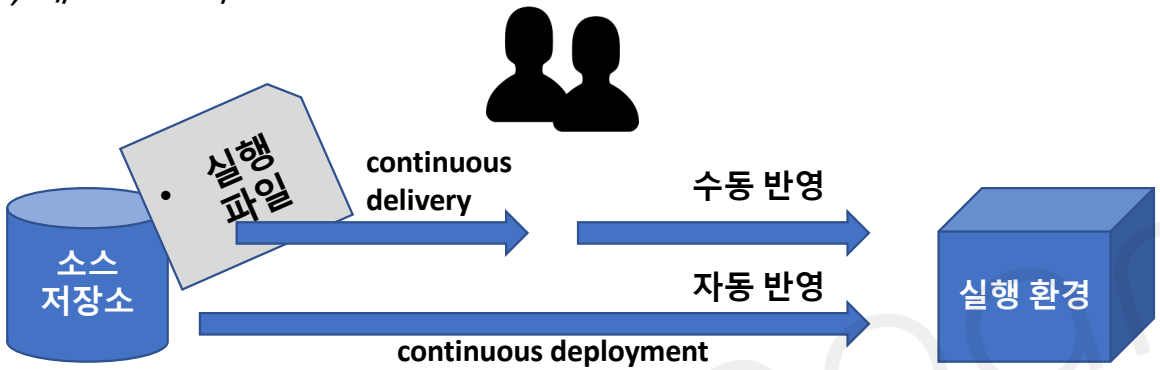
Cloud Native Application



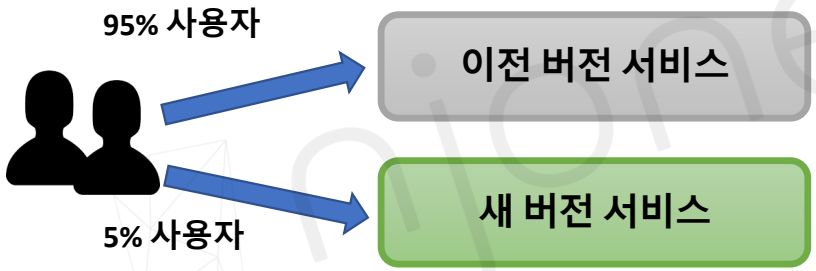
njone.com

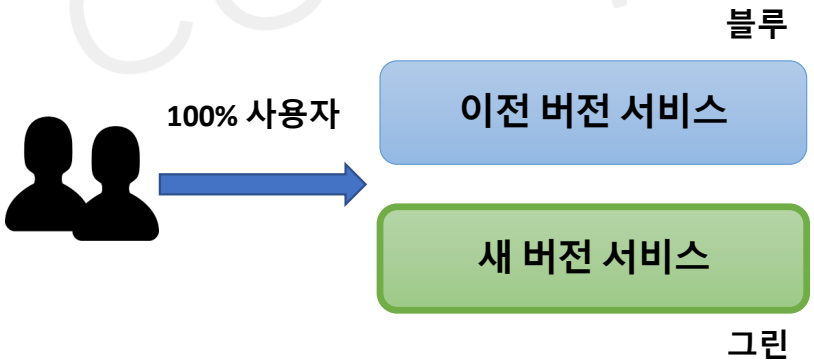
Cloud Native Application - CI/CD

- 지속적인 통합, CI(Continuous Integration)
  - *통합 서버, 소스 관리(SCM), 빌드 도구, 테스트 도구*
  - *ex) Jenkins, Team CI, Travis CI*
- 지속적 배포
  - ***Continuous Delivery***
  - ***Continuous Deployment***
  - Pipe line

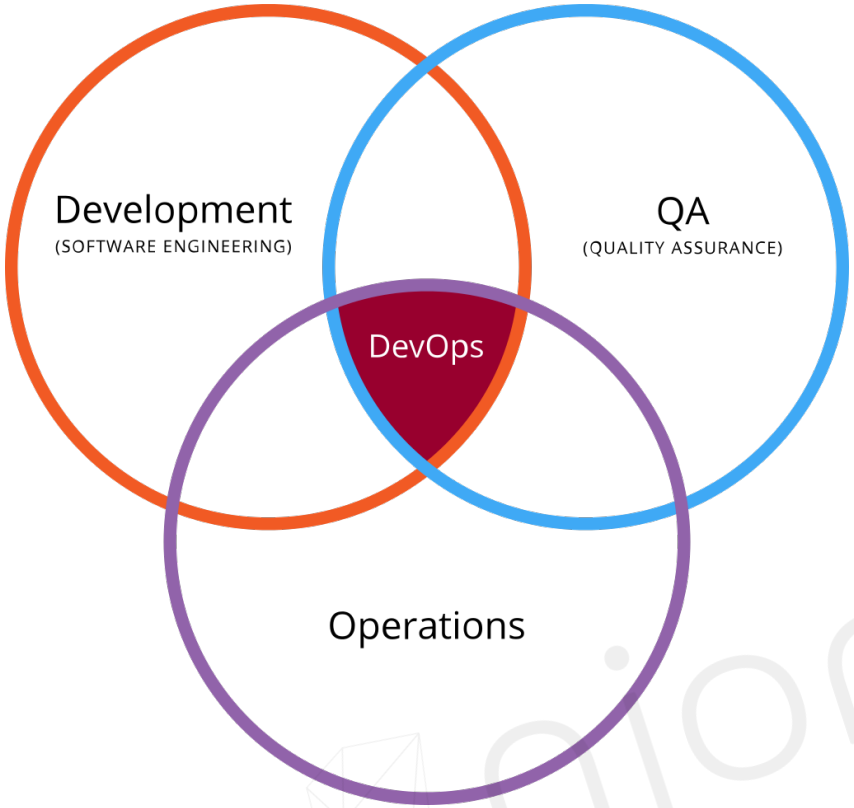


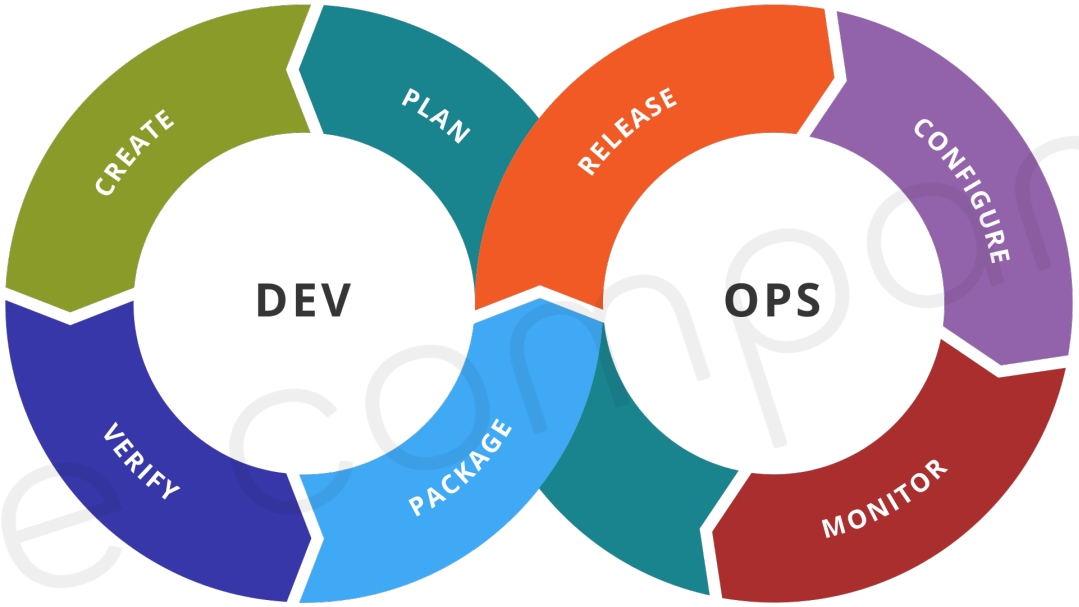
- 카나리 배포와 블루그린 배포





Cloud Native Application - DevOps

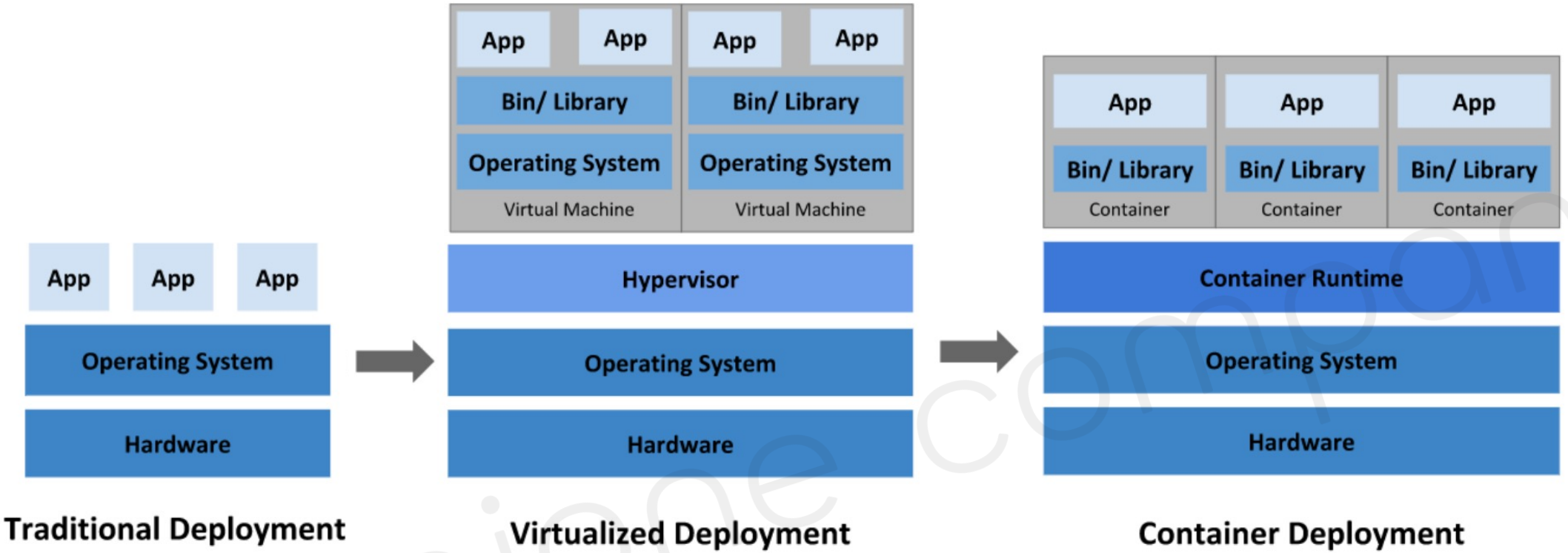




njonecompany

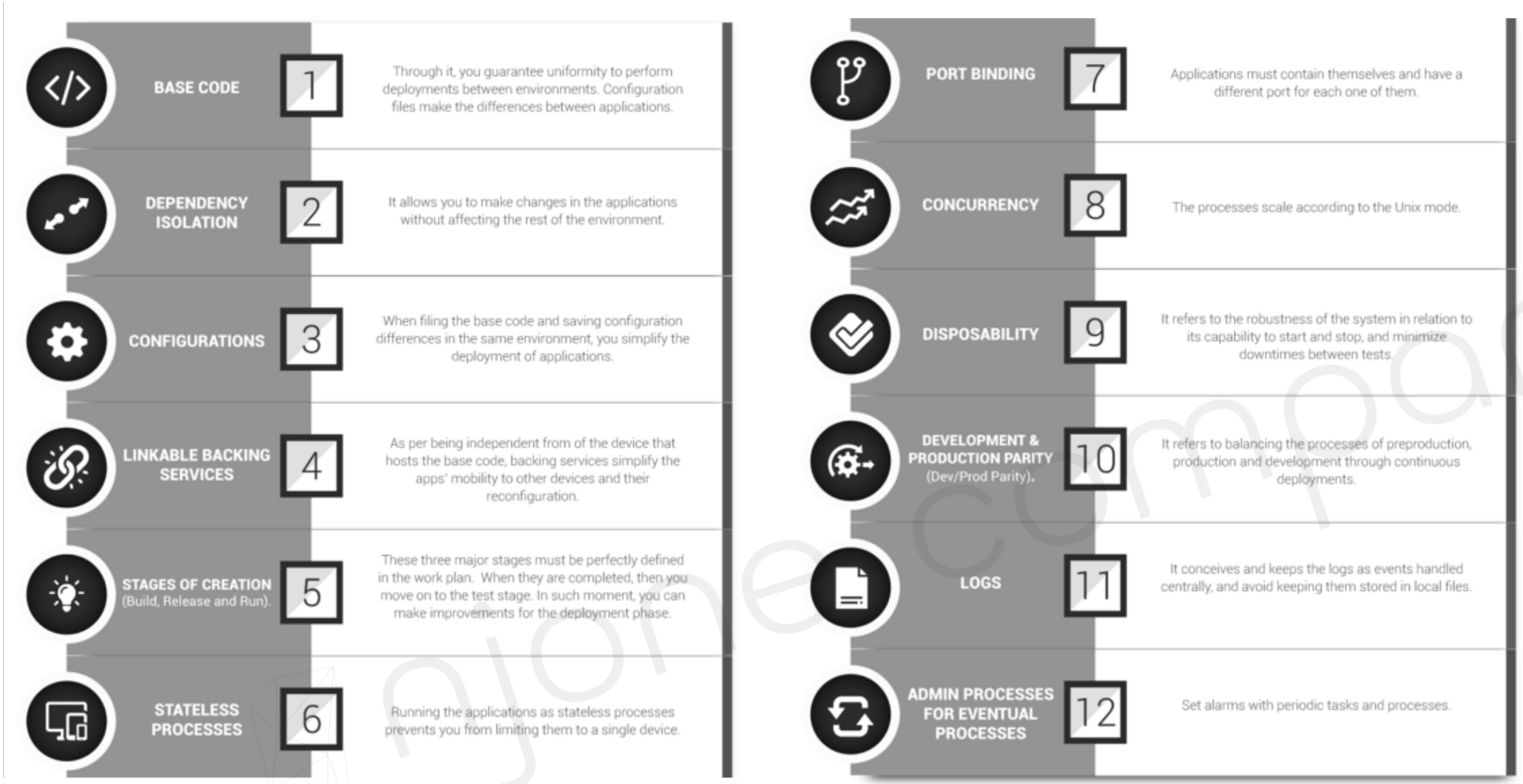
Cloud Native Application - Container 가상화

njone company



12 Factors(https://12factor.net/)

njone company



12 Factors + 3

1.One codebase, one application

2.API first

3.Dependency management

4.Design, build, release, and run

5.Configuration, credentials, and code

6.Logs

7.Disposability

8.Backing services

9. Environment parity

10.Administrative processes

11.Port binding

12.Stateless processes

13.Concurrency

14.Telemetry

15.Authentication and authorization

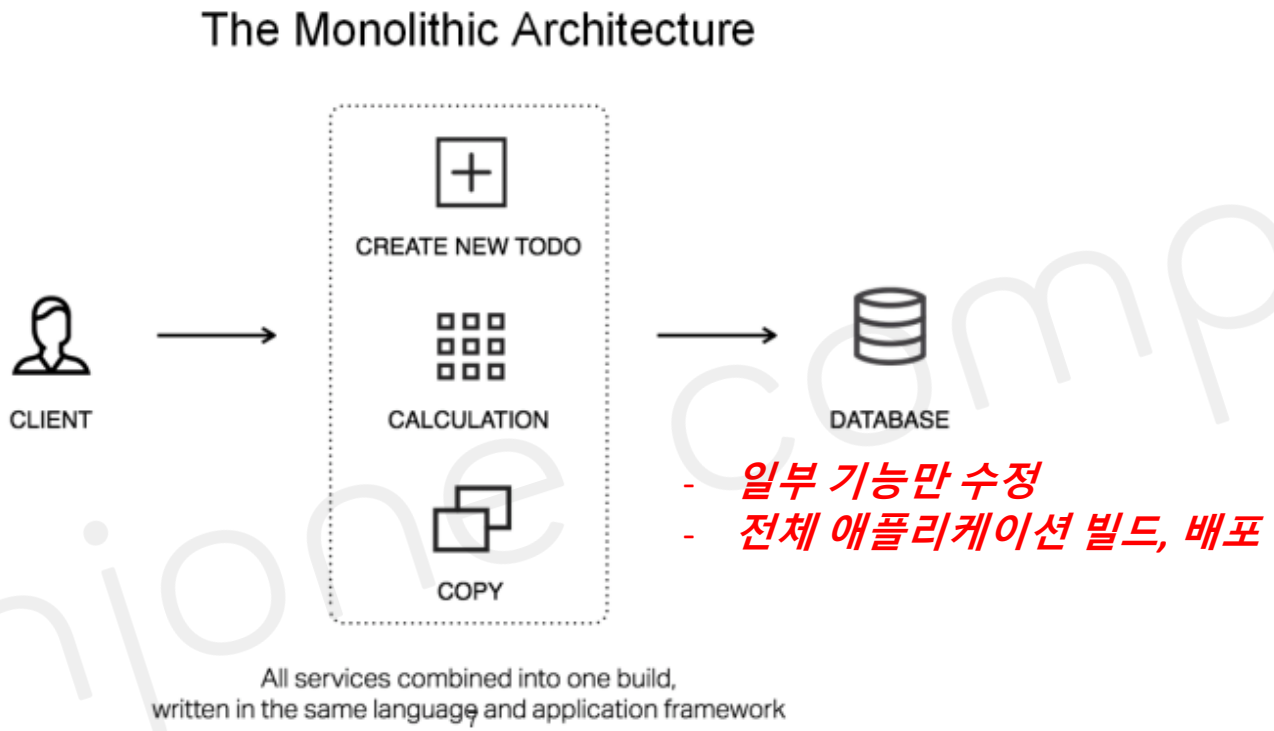
njone company

Monolithic vs. MSA

- ***Monolith vs Microservice Architecture***

Monolithic vs. MSA

- ***Monolith Architecture***
  - 모든 업무 로직이 하나의 애플리케이션 형태로 패키지 되어 서비스
  - 애플리케이션에서 사용하는 데이터가 한곳에 모여 참조되어 서비스되는 형태



- ***일부 기능만 수정***
- ***전체 애플리케이션 빌드, 배포***

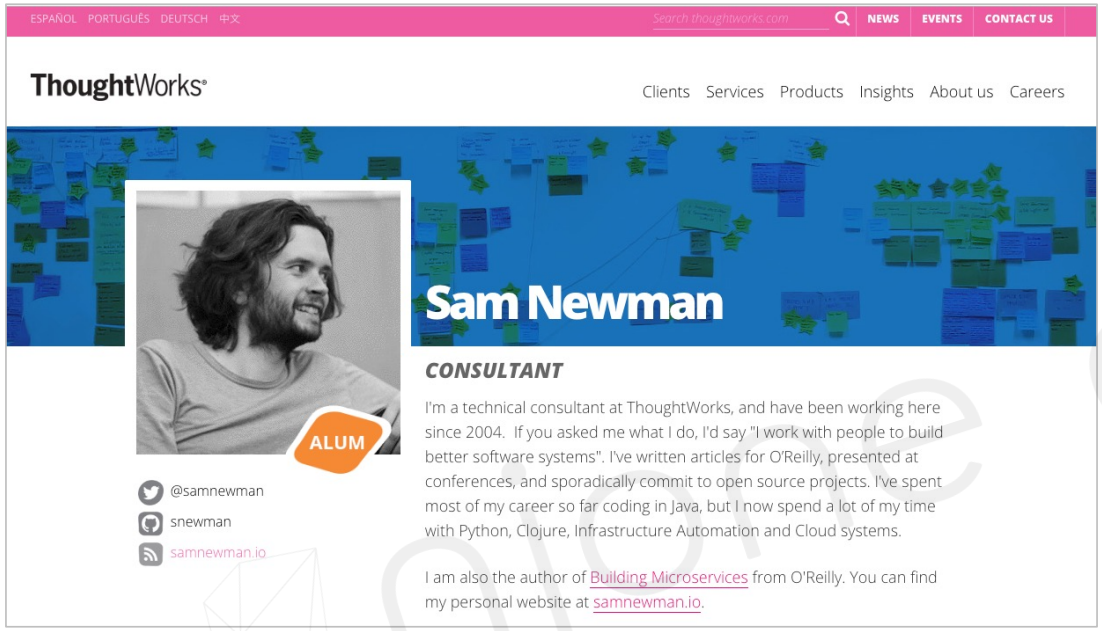
njone company

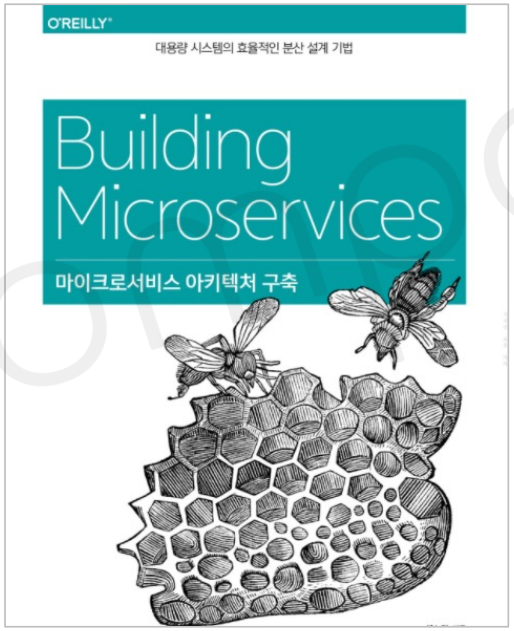
Monolithic vs. MSA

- ***What is the Microservice?***

Small autonomous services that work together

- Sam Newman





Monolithic vs. MSA

- ***What is the Microservice?***

In short, the microservice architectural style is an approach to developing a ***single application*** as a suite of ***small services***, each running in its own process and communicating with lightweight mechanisms, on an HTTP resource API....contd

njone company

Monolithic vs. MSA

- ***What is the Microservice?***

These services are built around ***business capabilities*** and ***independently deployable*** by fully **automated deployment** machinery...contd

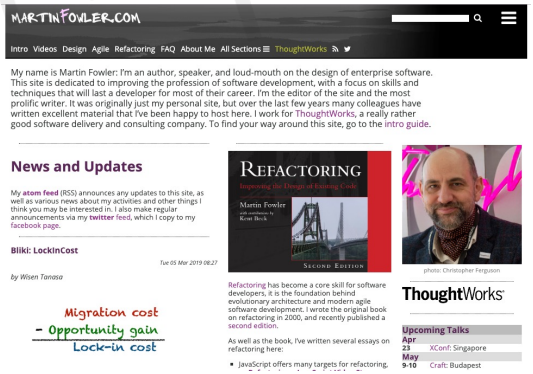
njone company

Monolithic vs. MSA

- ***What is the Microservice?***

There is a bare minimum of ***centralized management*** of these services, which may be written in ***different programming languages*** and use ***different data storage*** technologies

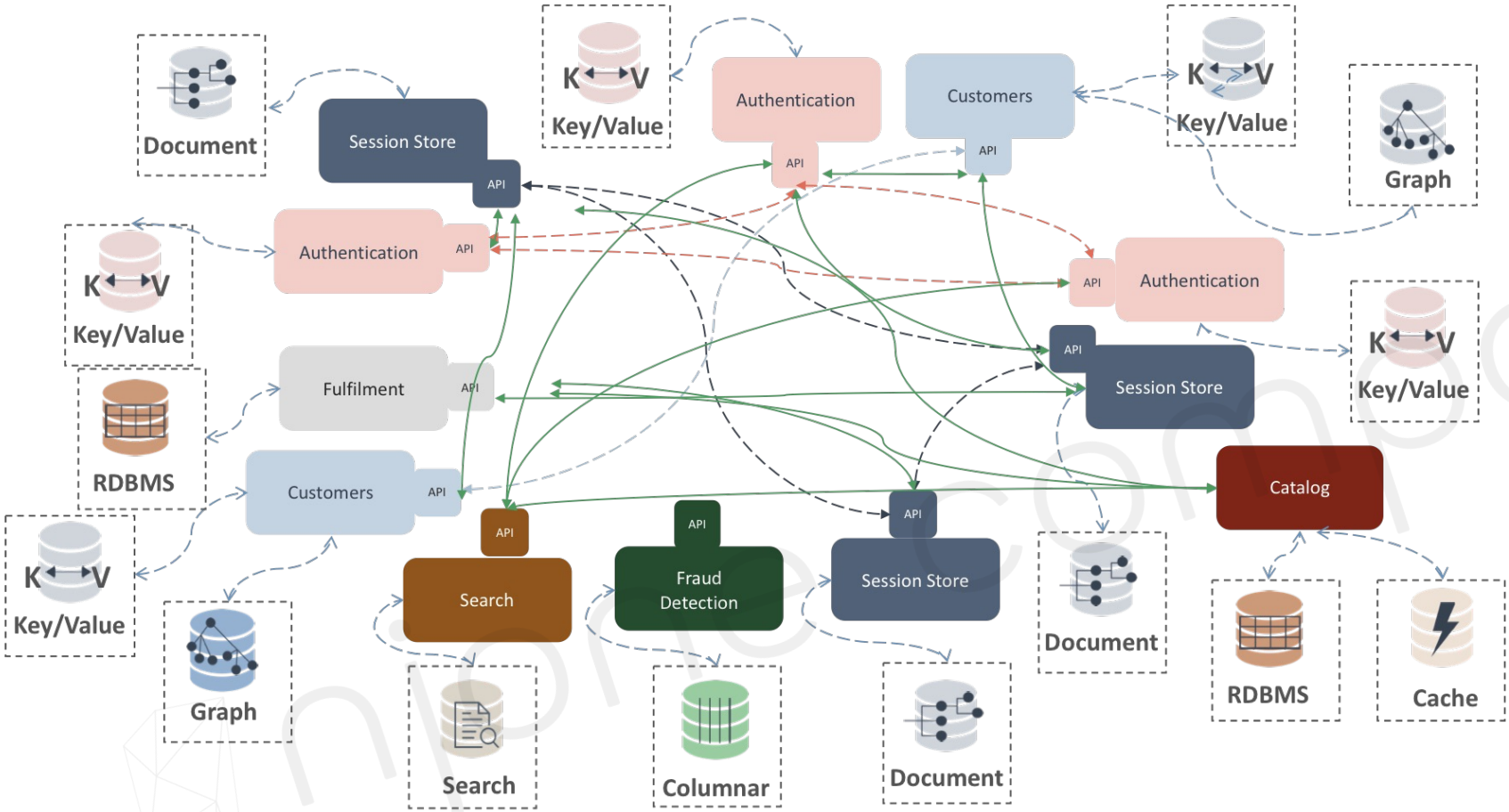
- James Lewis and Martin Fowler



Monolithic vs. MSA

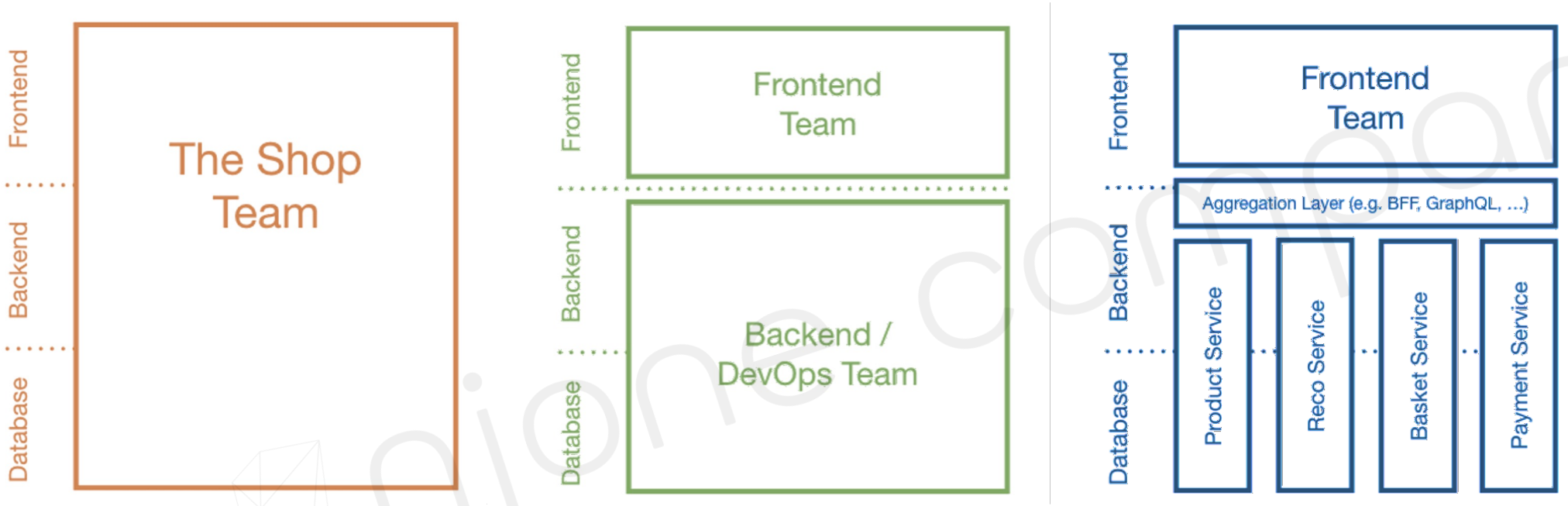
njone company

- ***Polyglot Persistence***

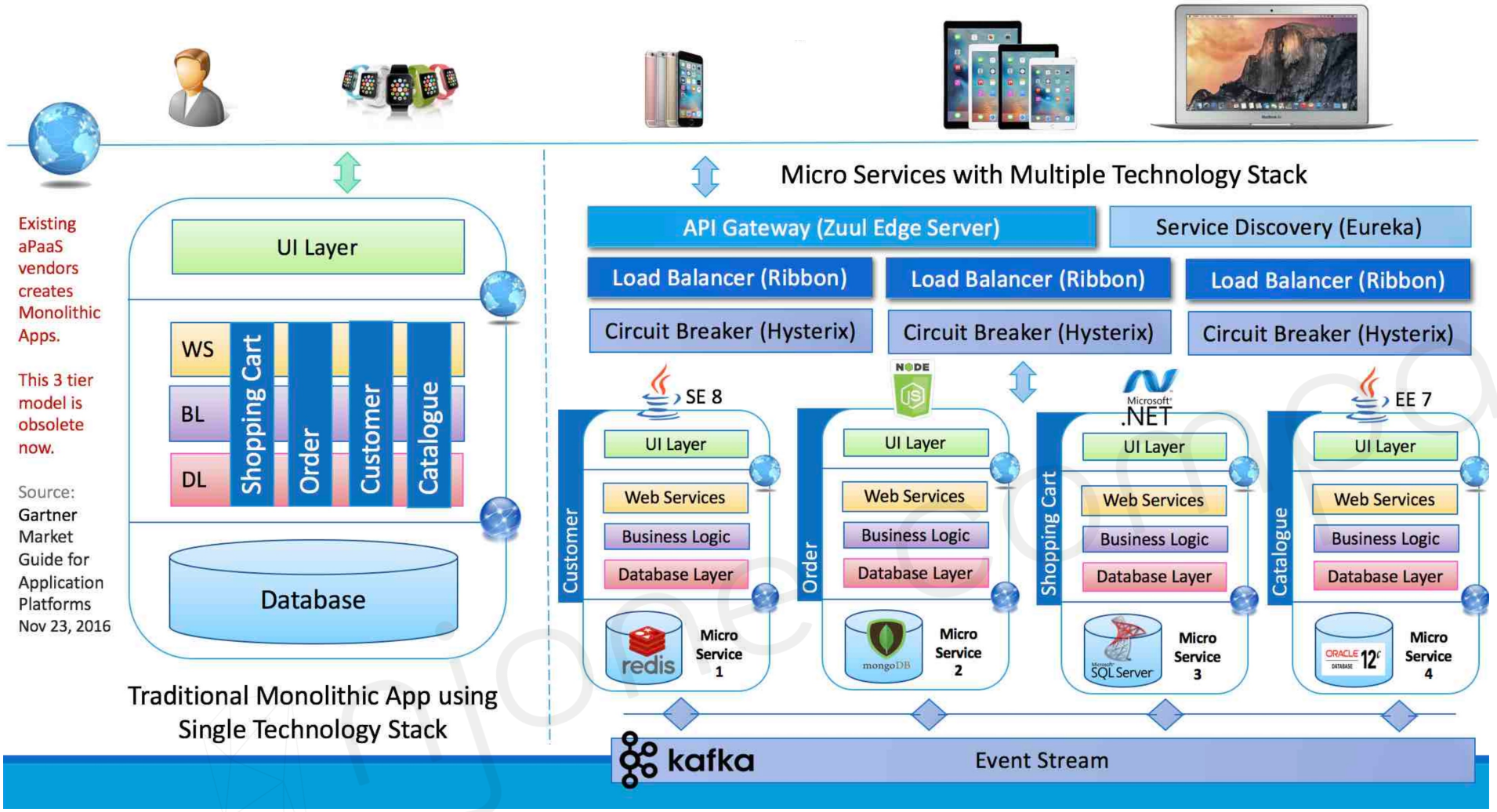


Monolithic vs. MSA

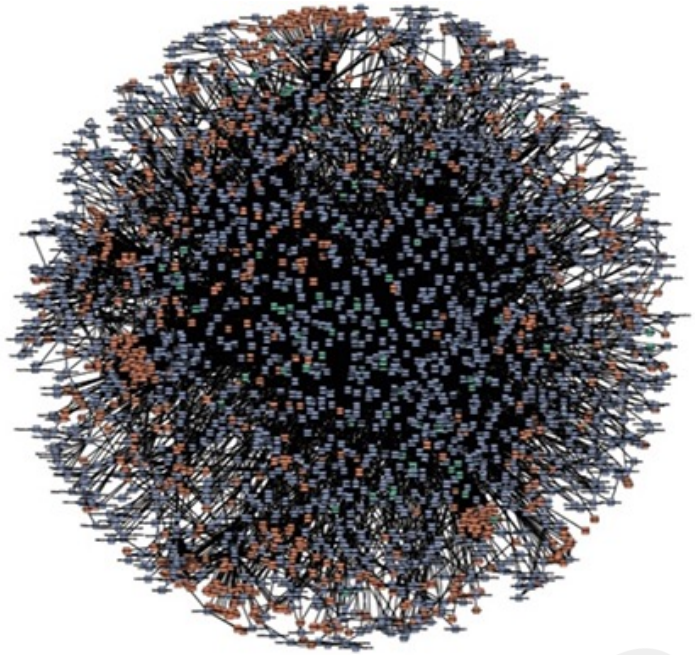
- ***Monolith** vs **Front & Back** vs **Microservice Architecture***



Monolithic vs. MSA



Microservice



amazon.com

NETFLIX

njone company

Microservice

- ***In 2002, Amazon founder and CEO Jeff Bezos's Email to Employees.***

1. All teams will henceforth expose their data and functionality through service interfaces.
2. Teams must communicate with each other through these interfaces.
3. There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
4. It doesn't matter what technology they use. HTTP, Corba, Pubsub, custom protocols — doesn't matter.
5. All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
6. Anyone who doesn't do this will be fired.
7. Thank you; have a nice day!

njone company

Microservice의 특징

- ***Monolith Pros & Cons***

- Easy source code navigation
- Easy debugging
- Easy deployment
- Performance
- Relatively easy

- Tight coupling
- Limited knowledge
- Undesirable change
- Limited tech stack
- Hard to scale

njone company

Microservice의 특징

- ***Microservice Pros & Cons***

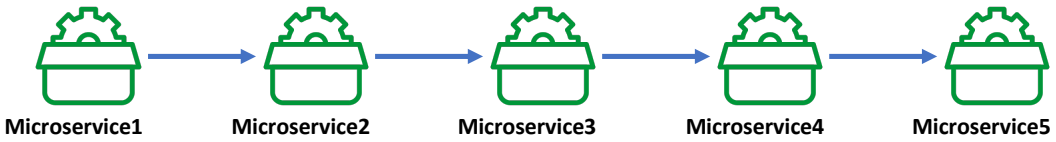
- Loose coupling
- Knowledge maintenance
- Less undesirable change
- Limited tech stack
- Easy to scale

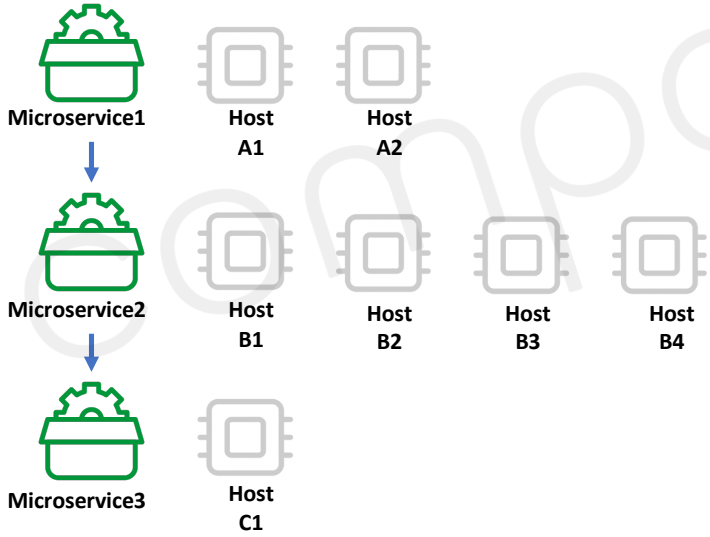
- Network latency
- Reduced availability
- Data consistency
- Object design complexity

njone company

Microservice의 특징

1) *Challenges*
2) *Small Well Chosen Deployable Units*
3) *Bounded Context*
4) *RESTful*
5) *Configuration Management*
6) *Cloud Enabled*
7) *Dynamic Scale Up And Scale Down*
8) *CI/CD*
9) *Visibility*





njone company

Microservice의 특징

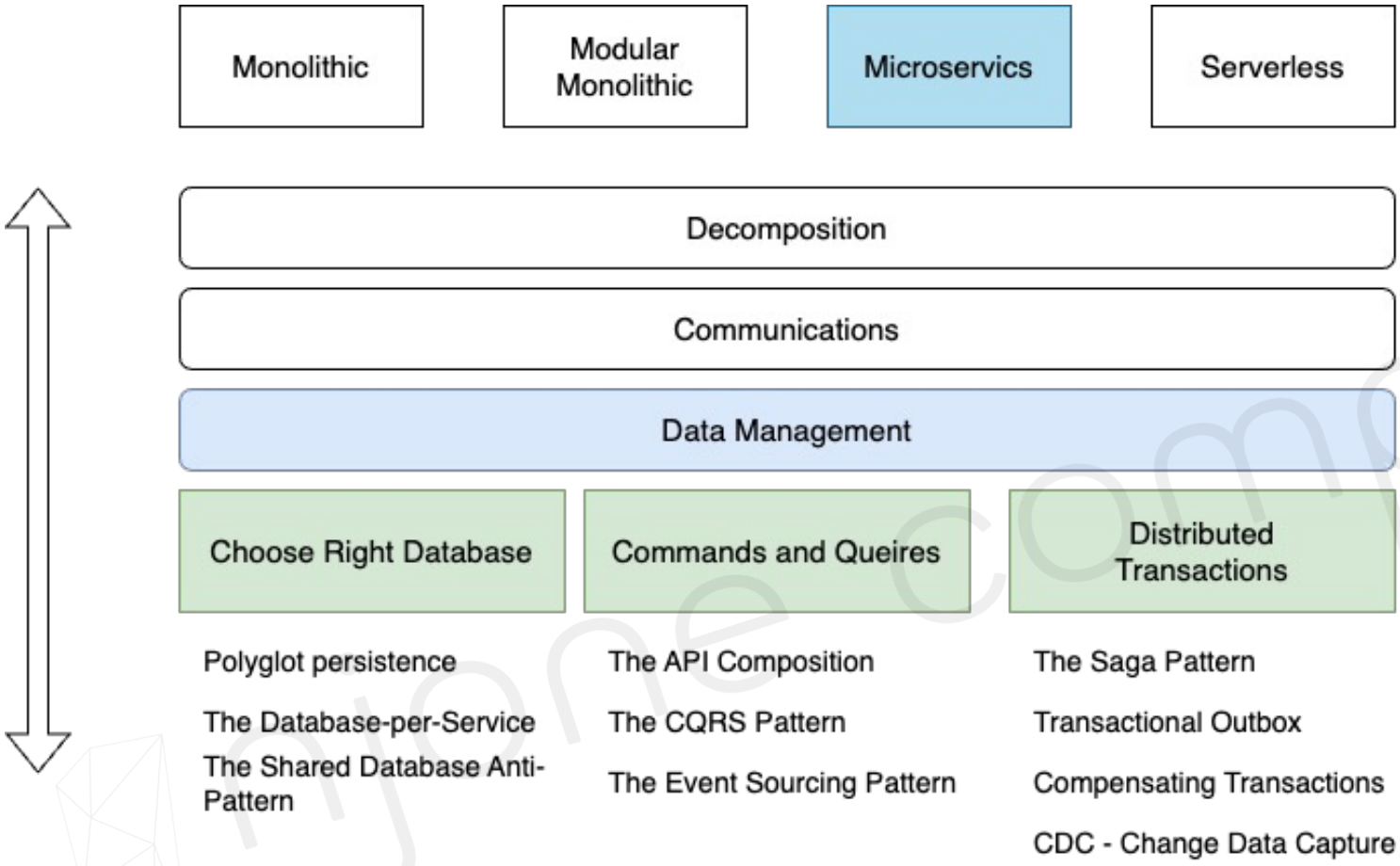
- ***Architecture Design***

Decomposition	• Service boundaries • Service decomposition
Communications	• Microservices Sync Communications • Microservices Async Communications
Data Management	• Microservices Choosing Database • Microservices Data Commands & Queries
Transaction Management	• Microservices Distributed Transactions • Microservices EDA
Deployment	• Microservices Deployments • Containers and Orchestrators
Resilience	• Microservices Resilience • Microservices Observability and Monitoring

njone company

Microservice의 특징

- ***Architecture Design***



Microservice의 특징

~~**“Everything should be a microservice”**~~

Q1) Multiple Rates of Change

Q2) Independent Life Cycles

Q3) Independent Scalability

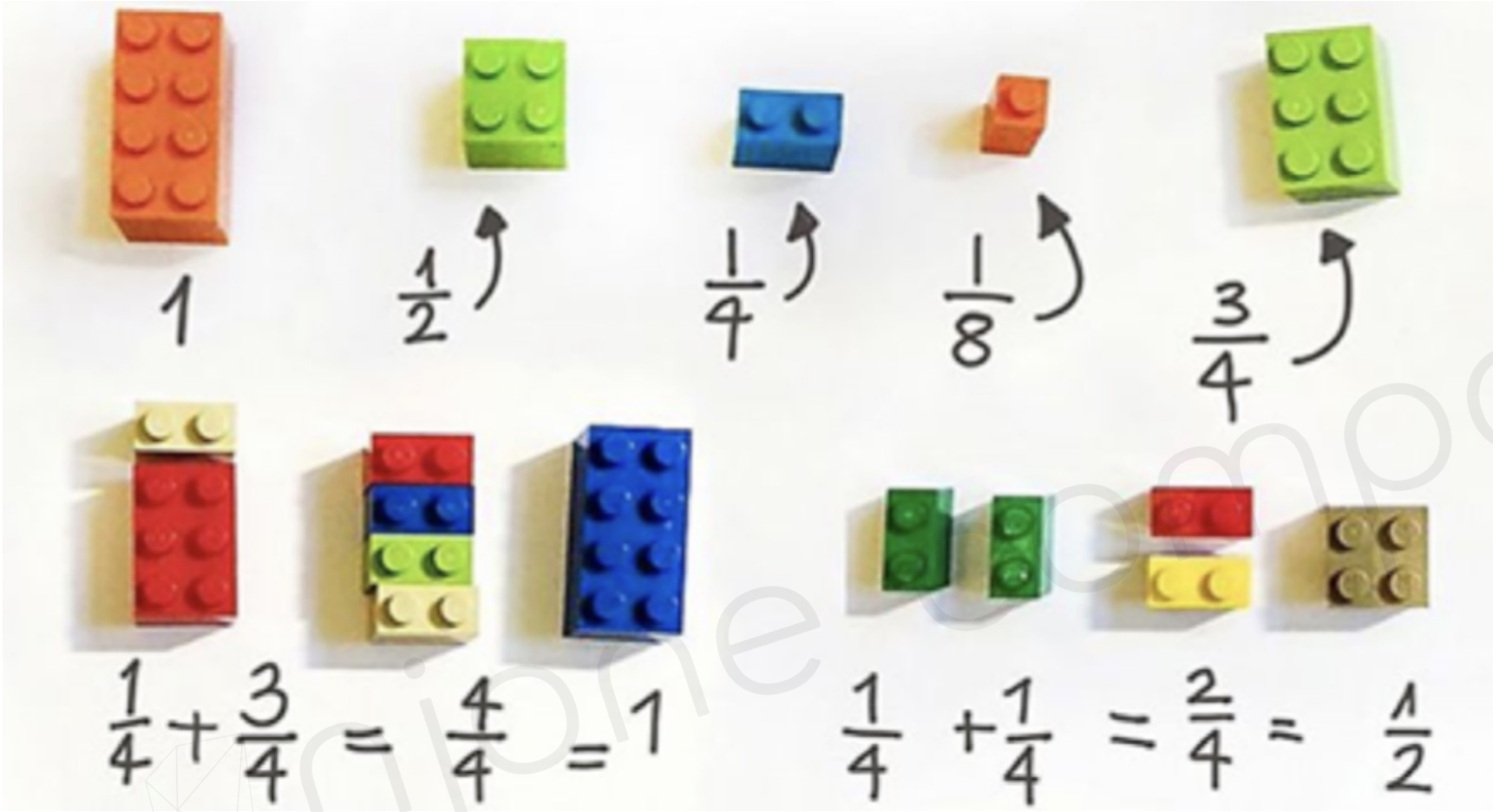
Q4) Isolated Failure

Q5) Simplify Interactions with External Dependencies

Q6) Polyglot Technology

njone company

Microservice의 특징

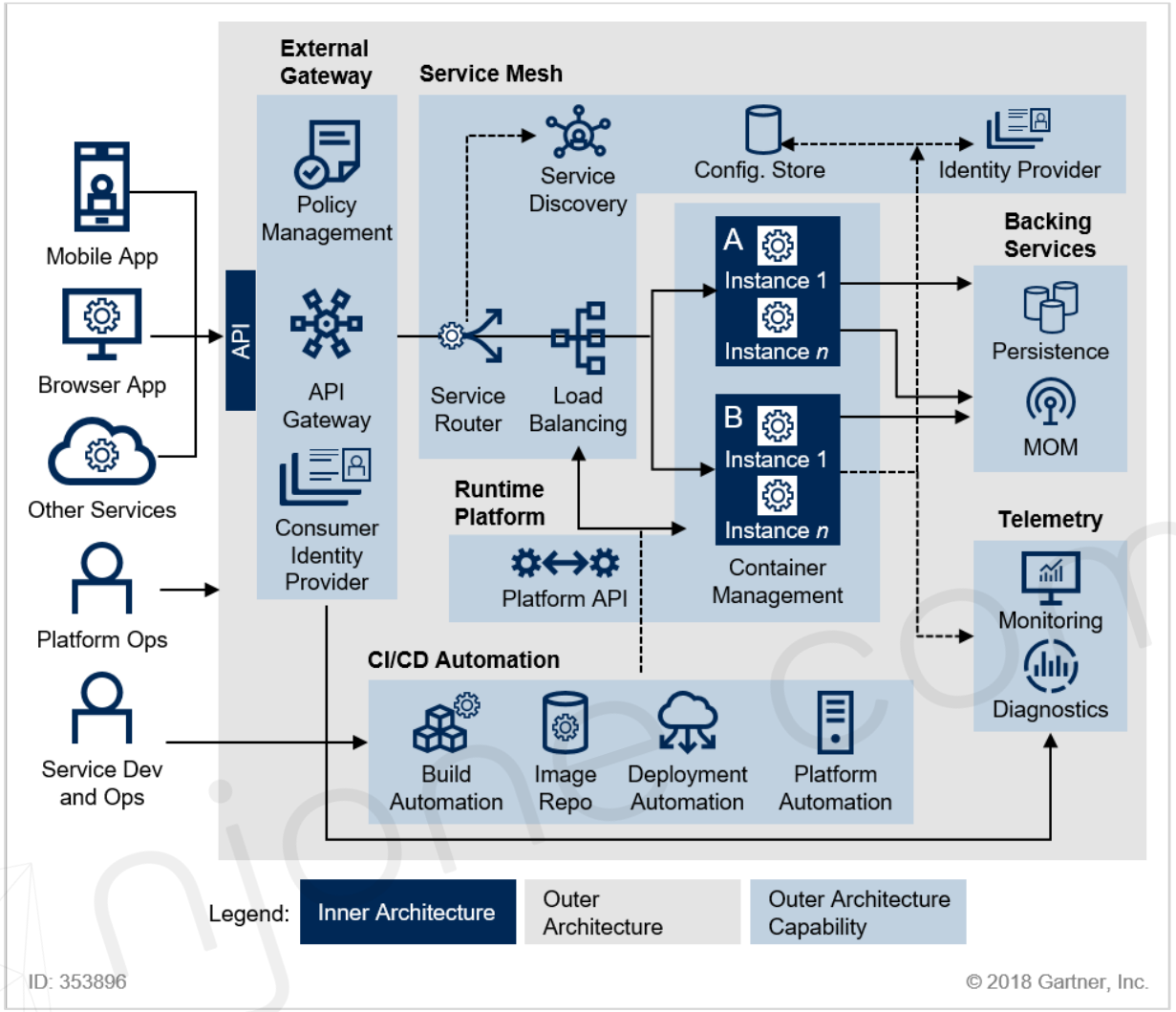


Microservice의 특징

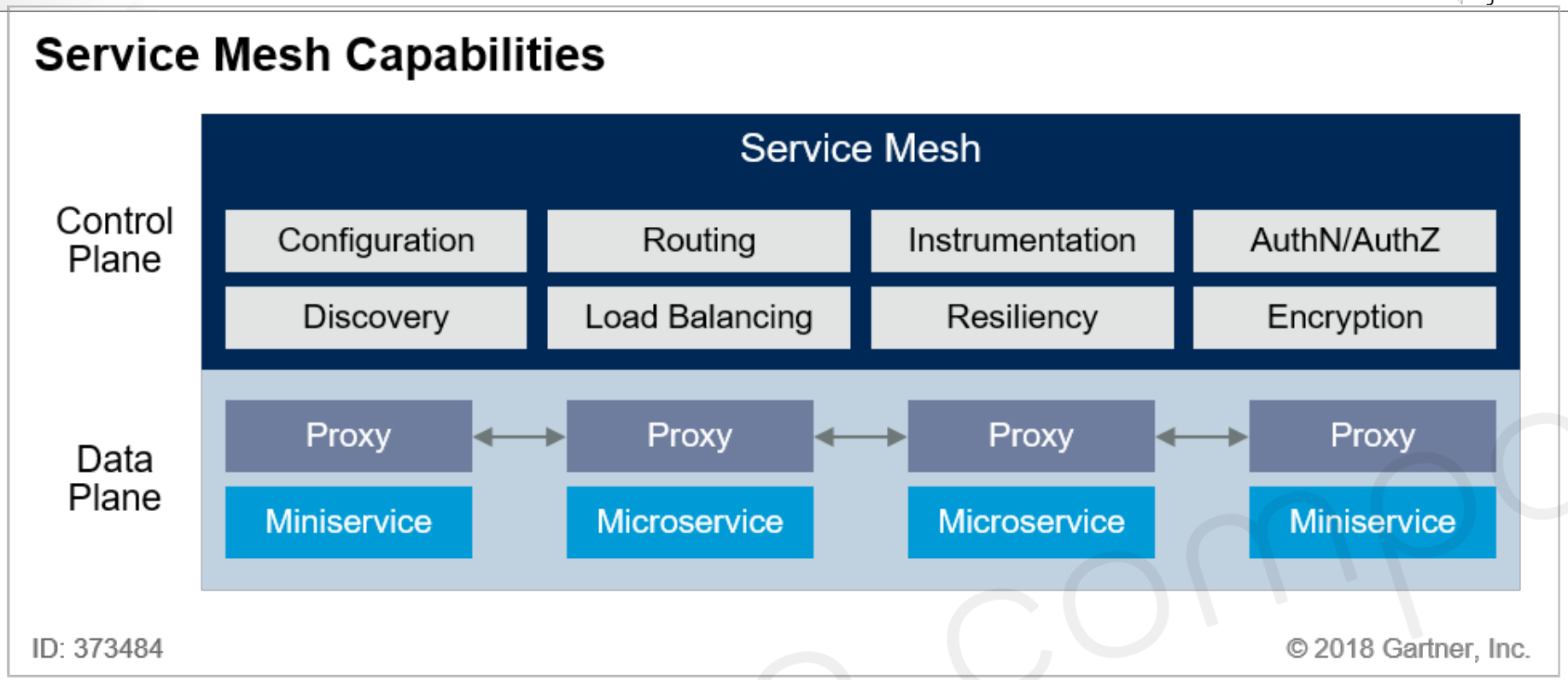
- ***Microservice Team Structure***
  - Two Pizza team
  - Teams communicating through API contracts
  - Develop, test and deploy each service independently
  - Consumer Driven Contract

njone company

MSA 표준 구성요소


External Gateway
Service Mesh
Mobile App
Browser App
Other Services
Platform Ops
Service Dev and Ops
API
Policy Management
API Gateway
Consumer Identity Provider
Service Discovery
Config. Store
Identity Provider
Service Router
Load Balancing
A
Instance 1
Instance n
B
Instance 1
Instance n
Container Management
Backing Services
Persistence
MOM
Runtime Platform
Platform API
Telemetry
Monitoring
Diagnostics
CI/CD Automation
Build Automation
Image Repo
Deployment Automation
Platform Automation
Legend:
Inner Architecture
Outer Architecture
Outer Architecture Capability
ID: 353896
© 2018 Gartner, Inc.

Service Mesh Capabilities

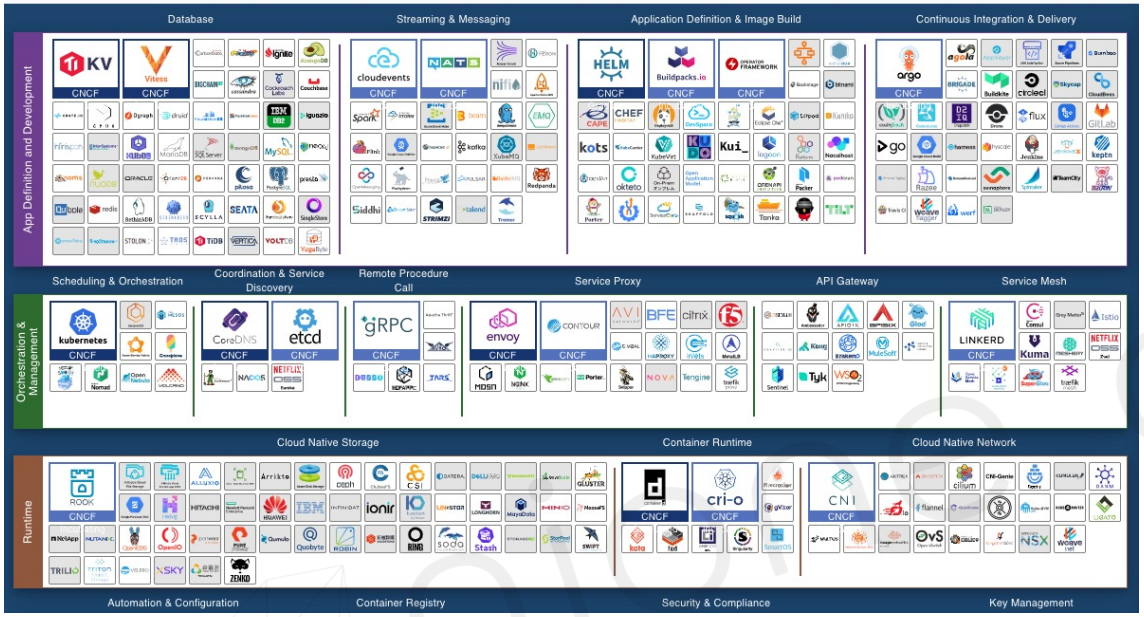


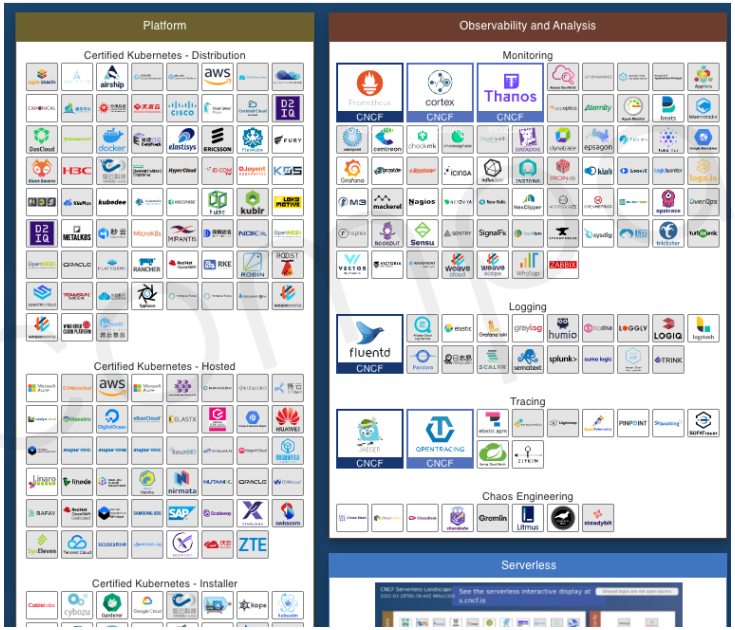
- MSA 인프라 → ***미들웨어***
  - *프록시 역할, 인증, 권한 부여, 암호화, 서비스 검색, 요청 라우팅, 로드 밸런싱*
  - ***자가 치유 복구 서비스***
- 서비스간의 통신과 관련된 기능을 자동화

njone company

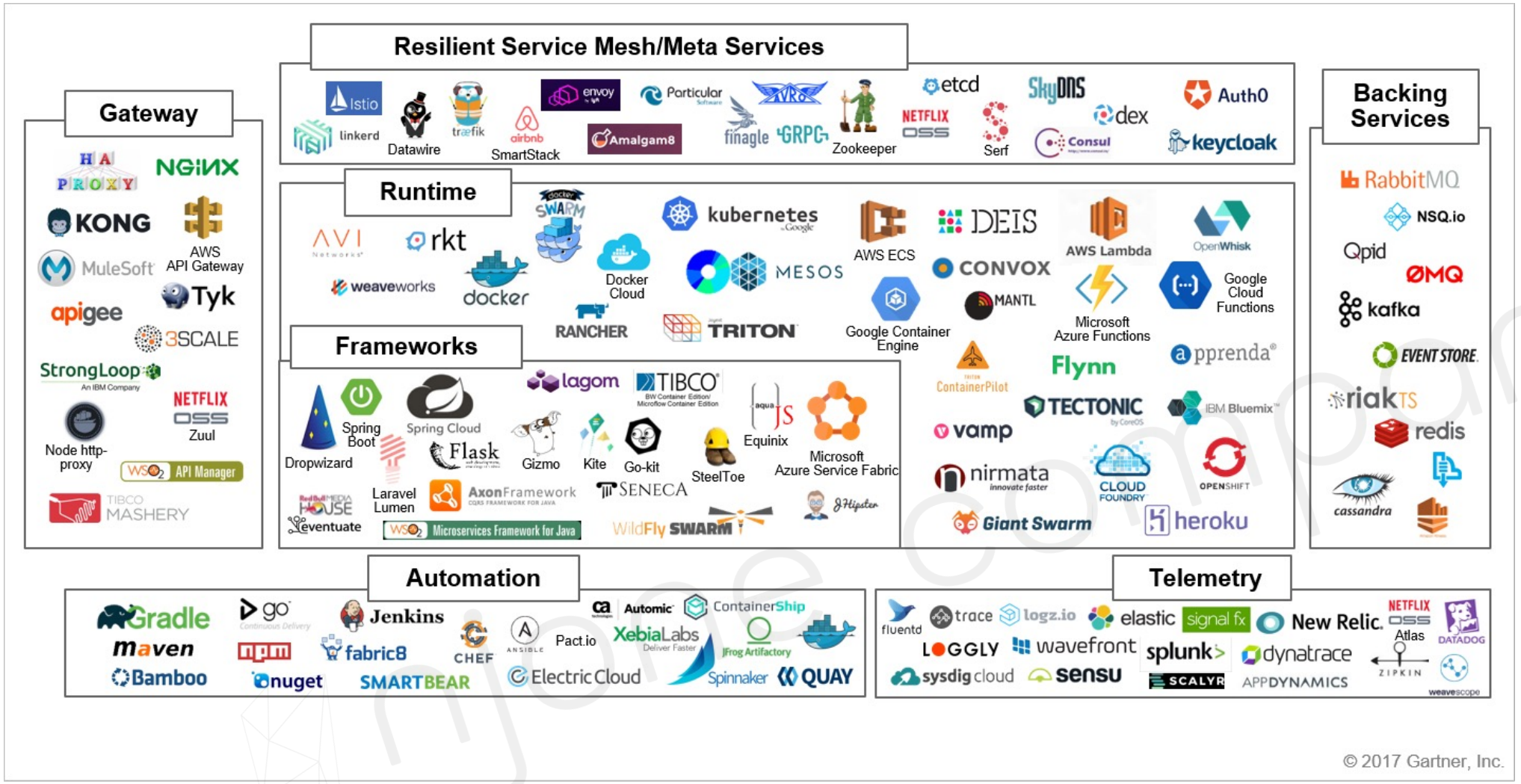
MSA 표준 구성요소

- CNCF(Cloud Native Computing Foundation)
  - Cloud Native Interactive Landscape
  - https://landscape.cncf.io/





MSA 기반 기술



© 2017 Gartner, Inc.

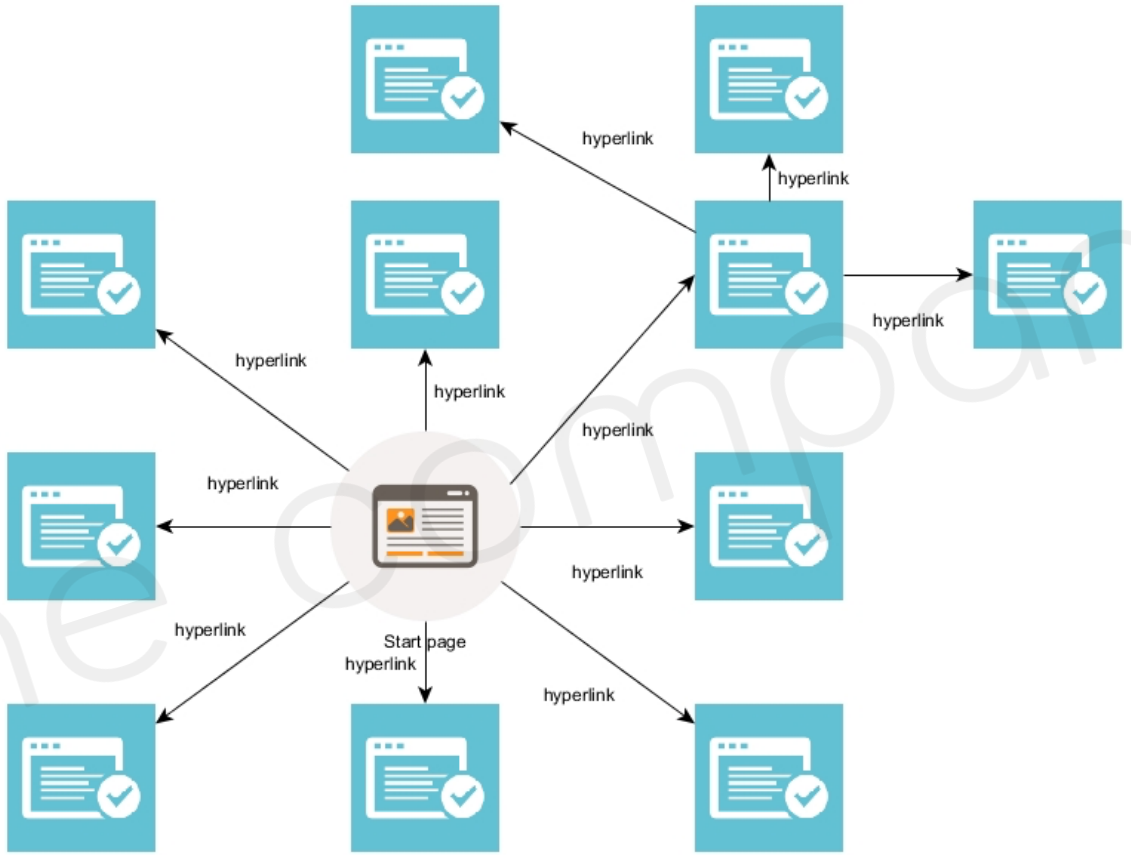
RESTful Web Service

njone company

- “A way to grade your API according to the constraints of REST.”

by Leonard Richardson

- LEVEL 0
  - Expose soap web services in rest style
  - http://server/getPosts
  - http://server/deletePosts
  - http://server/doThis
- LEVEL 1
  - Expose resources with proper uri
  - http://server/accounts
  - http://server/accounts/10
  - note: improper use of http methods
- LEVEL 2
  - Level1 + HTTP Methods
- LEVEL 3
  - Level2 + HATEOAS
  - DATA + NEXT POSSIBLE ACTIONS



RESTful Web Service

- Consumer first
- Make best use of HTTP
- Request methods
  - GET
  - POST
  - PUT
  - DELETE
- Response Status
  - 200
  - 404
  - 400
  - 201
  - 401
- No secure info in URI
- Use plurals
  - prefer /users to /user
  - prefer /users/1 to /user/1
- Use nouns for resources
- For exceptions
  - define a consistent approach

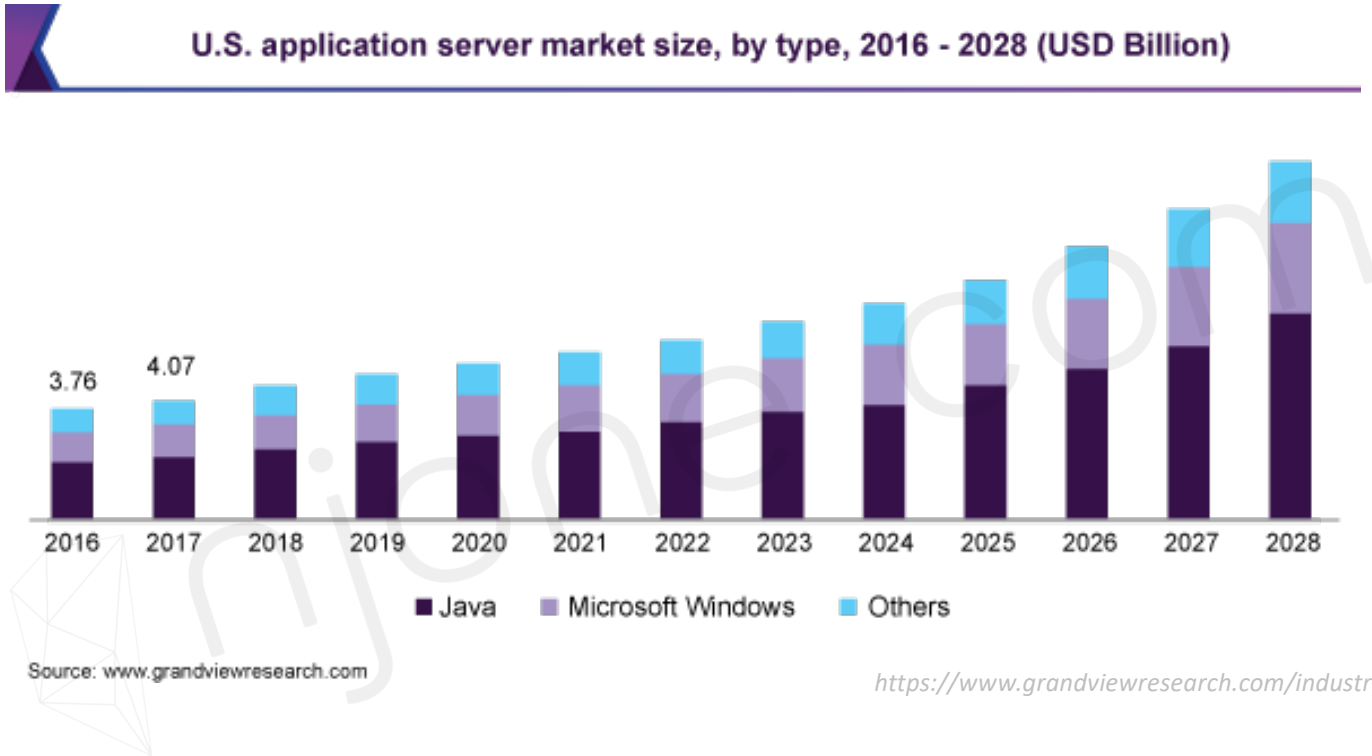
    /search

    PUT /gists/{id}/star

    DELETE /gists/{id}/star

njone company

“The global application server market size was valued at USD 15.84 billion in 2020 and is expected to expand at a compound annual growth rate (CAGR) of 13.2% from 2021 to 2028.”



https://www.grandviewresearch.com/industry-analysis/application-server-market

Spring Cloud

https://spring.io/projects/spring-cloud

Spring Cloud 2022.0.2

OVERVIEW LEARN SAMPLES

Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems (e.g. configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, one-time tokens, global locks, leadership election, distributed sessions, cluster state). Coordination of distributed systems leads to boiler plate patterns, and using Spring Cloud developers can quickly stand up services and applications that implement those patterns. They will work well in any distributed environment, including the developer's own laptop, bare metal data centres, and managed platforms such as Cloud Foundry.

njone company

Spring Cloud

- Spring Boot + Spring Cloud

Release Train	Release Train
2022.0.x aka Kilburn	3.0.x
2021.0.x aka Jubilee	2.6.x, 2.7.x (Starting with 2021.0.3)
2020.0.x aka Ilford	2.4.x, 2.5.x (Starting with 2020.0.3)
Hoxton	2.2.x, 2.3.x (Starting with SR5)
Greenwich	2.1.x
Finchley	2.0.x
Edgware	1.5.x
Dalston	1.5.x

Spring Cloud Dalston, Edgware, Finchley, and Greenwich have all reached end of life status and are no longer supported.

njone company

Spring Cloud

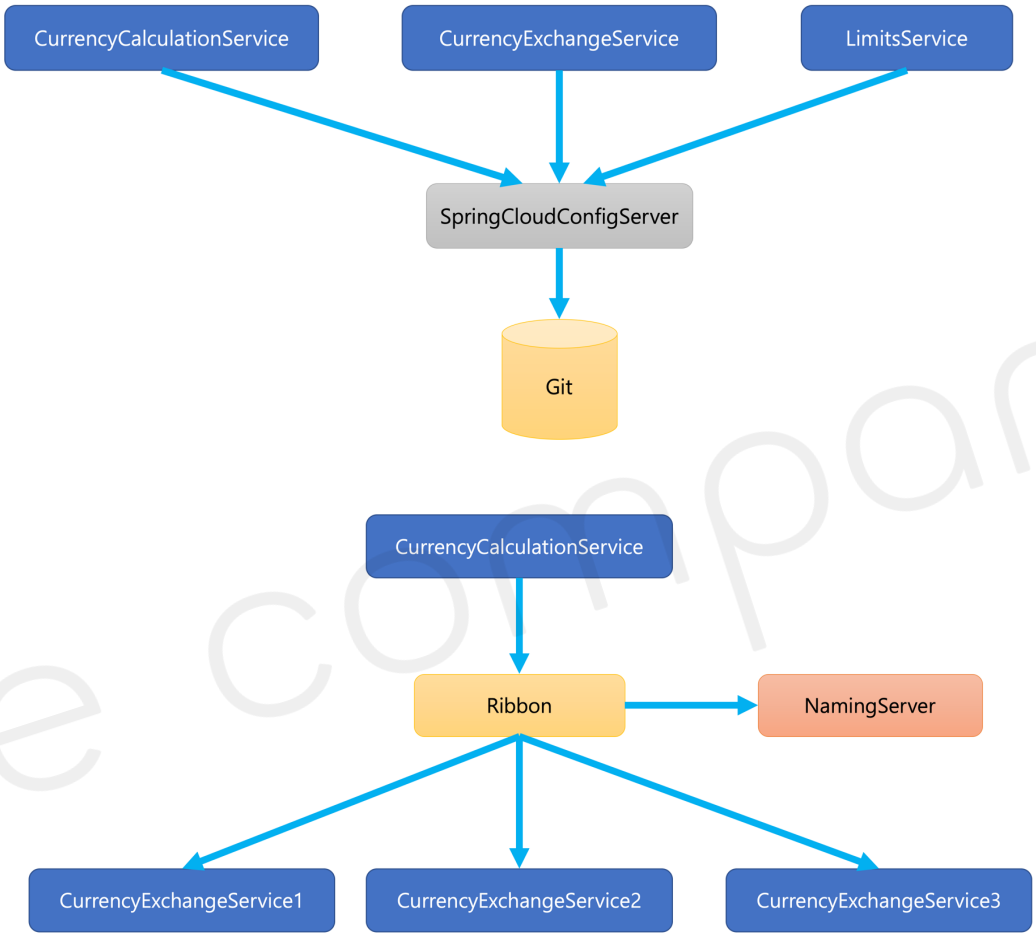
- Main Projects
  - Spring Cloud Config
  - Spring Cloud Netflix
  - Spring Cloud Bus
  - Spring Cloud Cloudfoundry
  - Spring Cloud Open Service Broker
  - Spring Cloud Cluster
  - Spring Cloud Consul
  - Spring Cloud Security
  - Spring Cloud Sleuth
  - Spring Cloud Data Flow
  - Spring Cloud Stream
  - Spring Cloud Stream App Starters
  - Spring Cloud Task
  - Spring Cloud Task App Starters
  - Spring Cloud Zookeeper
  - Spring Cloud Connectors
  - Spring Cloud Starters
  - Spring Cloud CLI
  - Spring Cloud Contract
  - Spring Cloud Gateway
  - Spring Cloud OpenFeign
  - Spring Cloud Pipelines
  - Spring Cloud Function

njone company

Spring Cloud

njone company

- Centralized configuration management
  - Spring Cloud Config Server
- Location transparency
  - Naming Server (Eureka)
- Load Distribution (Load Balancing)
  - Ribbon (Client Side)
  - Spring Cloud Gateway
- Easier REST Clients
  - FeignClient
- Visibility and monitoring
  - Zipkin Distributed Tracing
  - Netflix API gateway
- Fault Tolerance
  - Hystrix



njone company

Spring Cloud vs Kubernetes

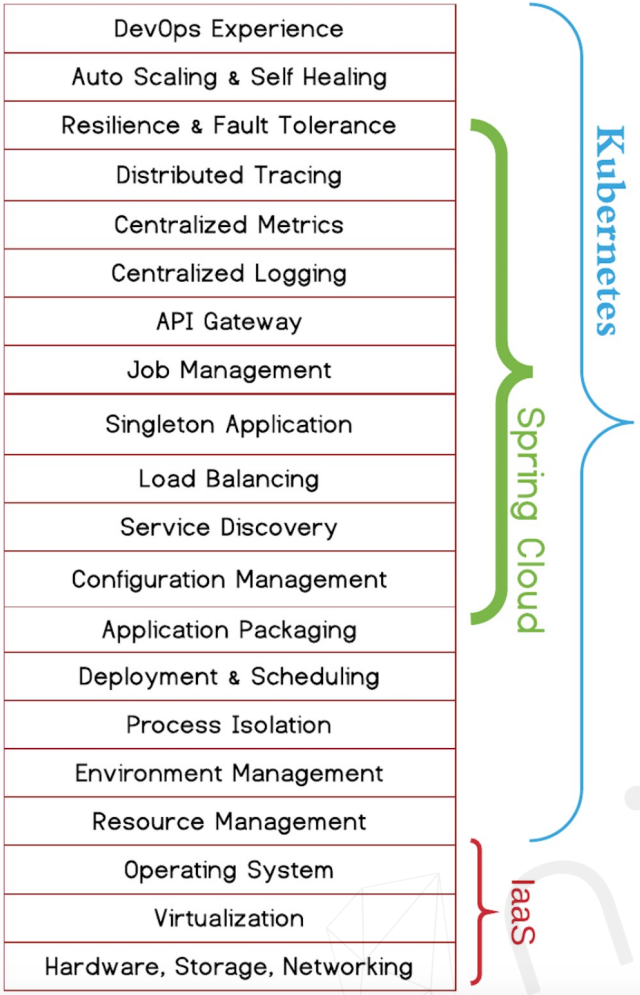
njone company

Strengths	Spring Cloud	Weaknesses
Unified programing model with Spring framework		Main functionality limited to Java platform
Feature rich collection of Java libraries		Lot's of responsibility for Java devs and the application stack
Well integrated Java libraries		Does not cover full microservices life cycle
Strengths	**Kubernetes**	**Weaknesses**
Polyglot and generic platform based on containers		A generic platform with coarse grained primitives
Covers full microservices life cycle		Operations focused platform
Cutting edge technology and large community		Actively developed and rapidly changing

https://dzone.com/articles/deploying-microservices-spring-cloud-vs-kubernetes

Spring Cloud vs Kubernetes

njone company



Capability	Spring Cloud with Kubernetes
DevOps Experience	Self service, multi-environment capabilities
Auto Scaling & Self Healing	Pod/Cluster Autoscaler, HealthIndicator, Scheduler
Resilience & Fault Tolerance	HealthIndicator, Hystrix, HealthCheck, Process Check
Distributed Tracing	Zipkin
Centralized Metrics	Heapster, Prometheus, Grafana
Centralized Logging	EFK
Job Management	Spring Batch, Schedulled Job
Load Balancing	Ribbon, Service
Service Discovery	Service
Configuration Management	Externalized Configurations, ConfigMap, Secret
Servie Logic	Apache Camel, Spring Framework
Application Packaging	Spring Boot maven plugin
Deployment & Scheduling	Deployment strategy, A/B, Canary, Scheduler strategy
Process Isolation	Docker, Pods
Environment Management	Namespaces, Authorizations
Resource Management	CPU and memory limits, Namespace resource quotas
IaaS	GCE, Azure, CenturyLink, VMware, Openstack

njone company

Spring Cloud App 개발을 위한 SW 설치

- IntelliJ IDEA Ultimate
  - https://www.jetbrains.com/ko-kr/idea
- Git
  - https://git-scm.com
- Visual Studio Code
  - https://code.visualstudio.com
- Postman
  - https://www.postman.com

POSTMAN

njone company