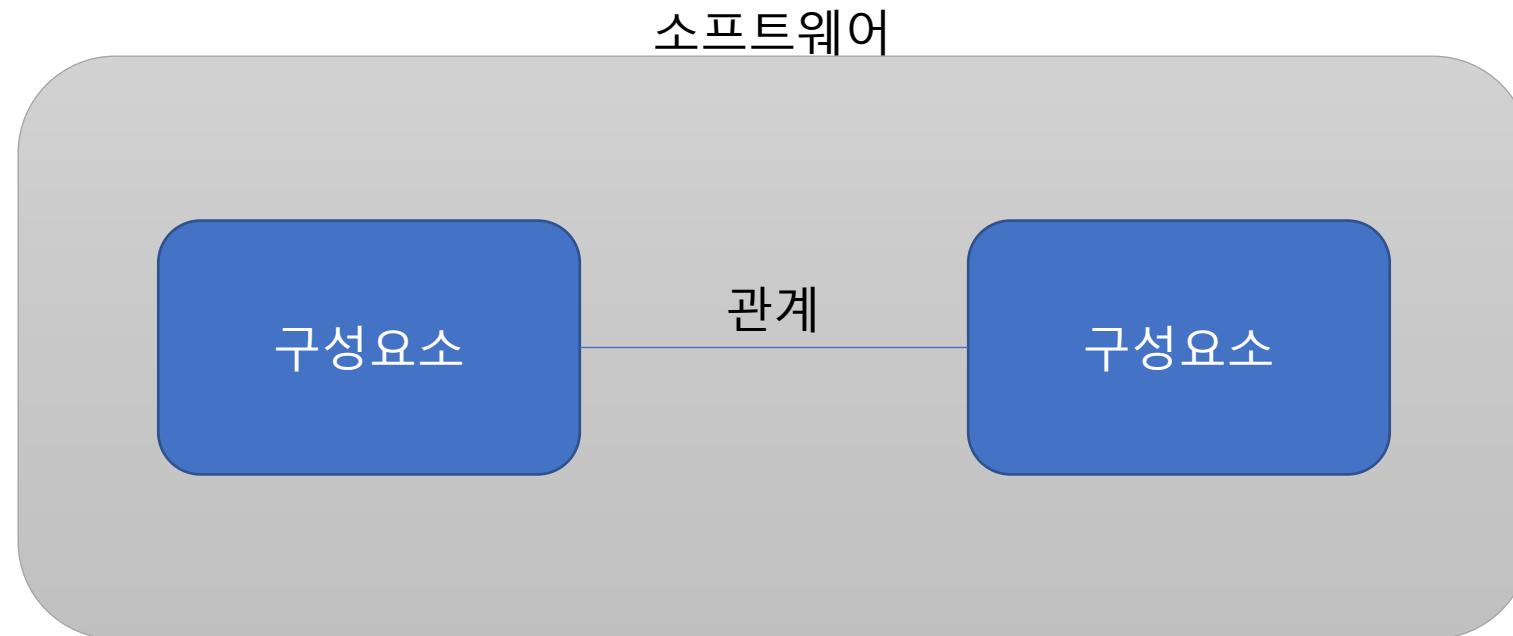




Software Architecture

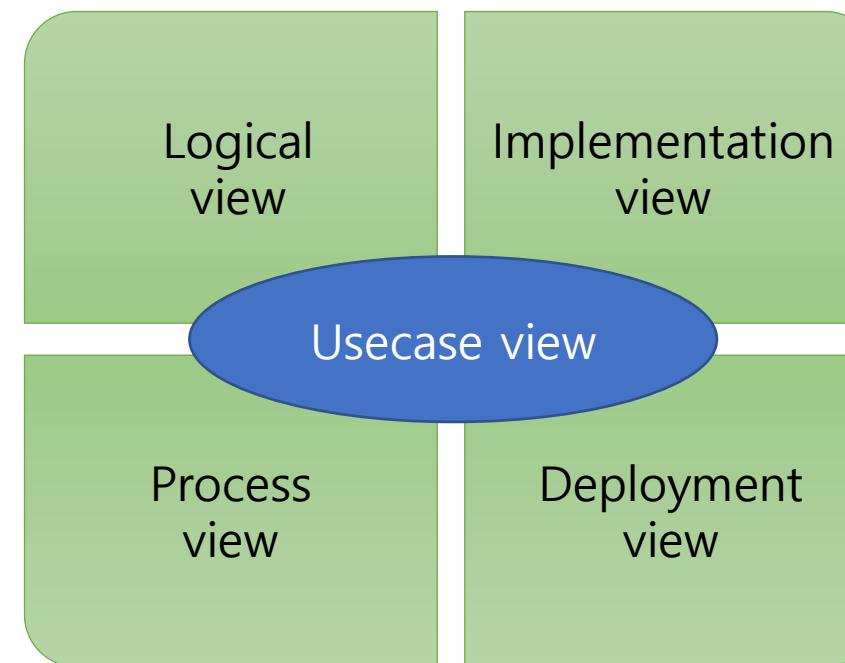
Software Architecture

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



Software Architecture

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



Software Architecture

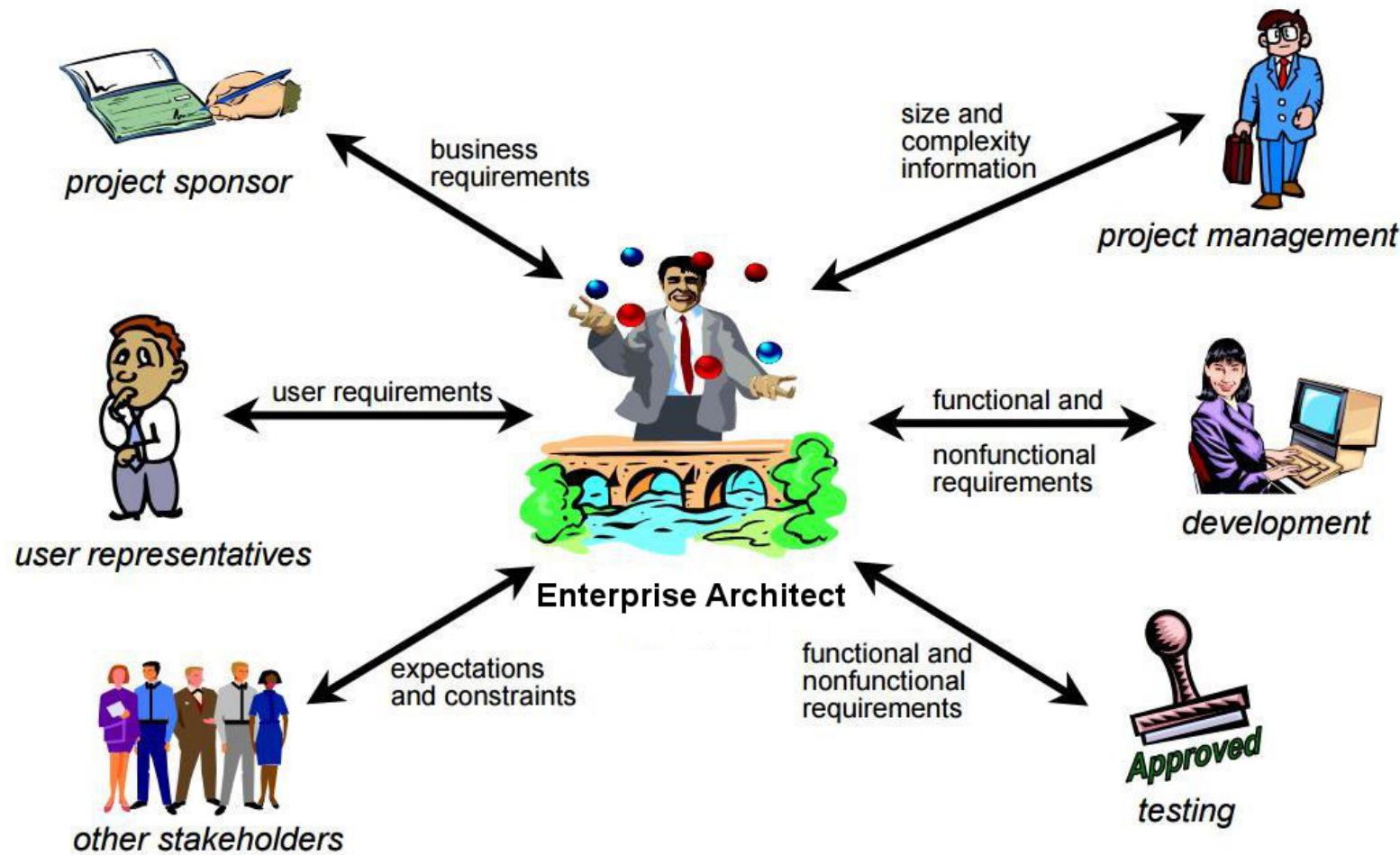
- Architecture의 역할

"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**."



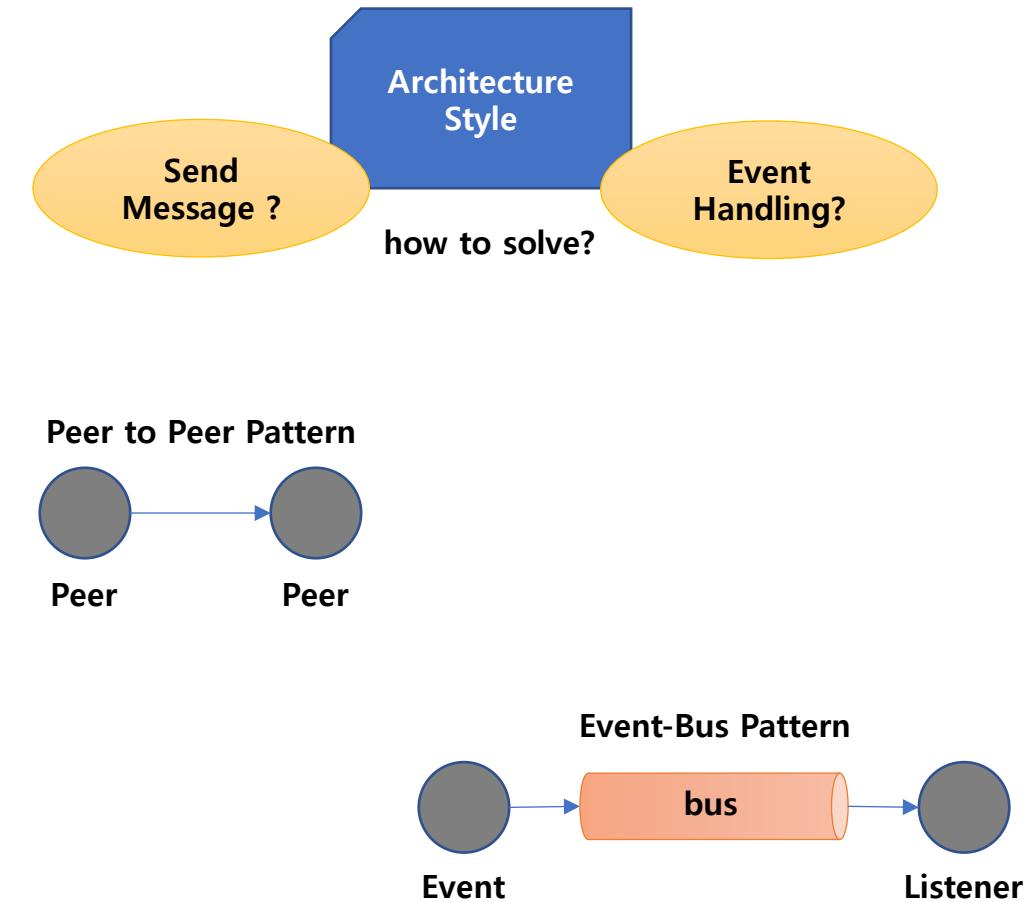
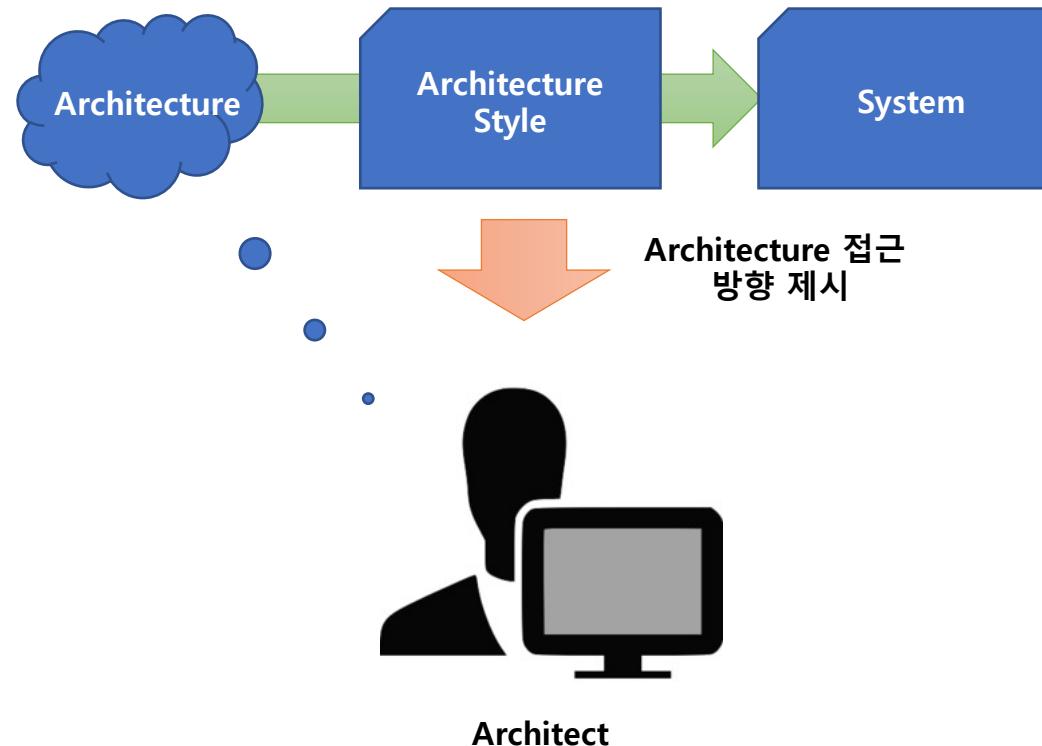
Software Architecture

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



Software Architecture

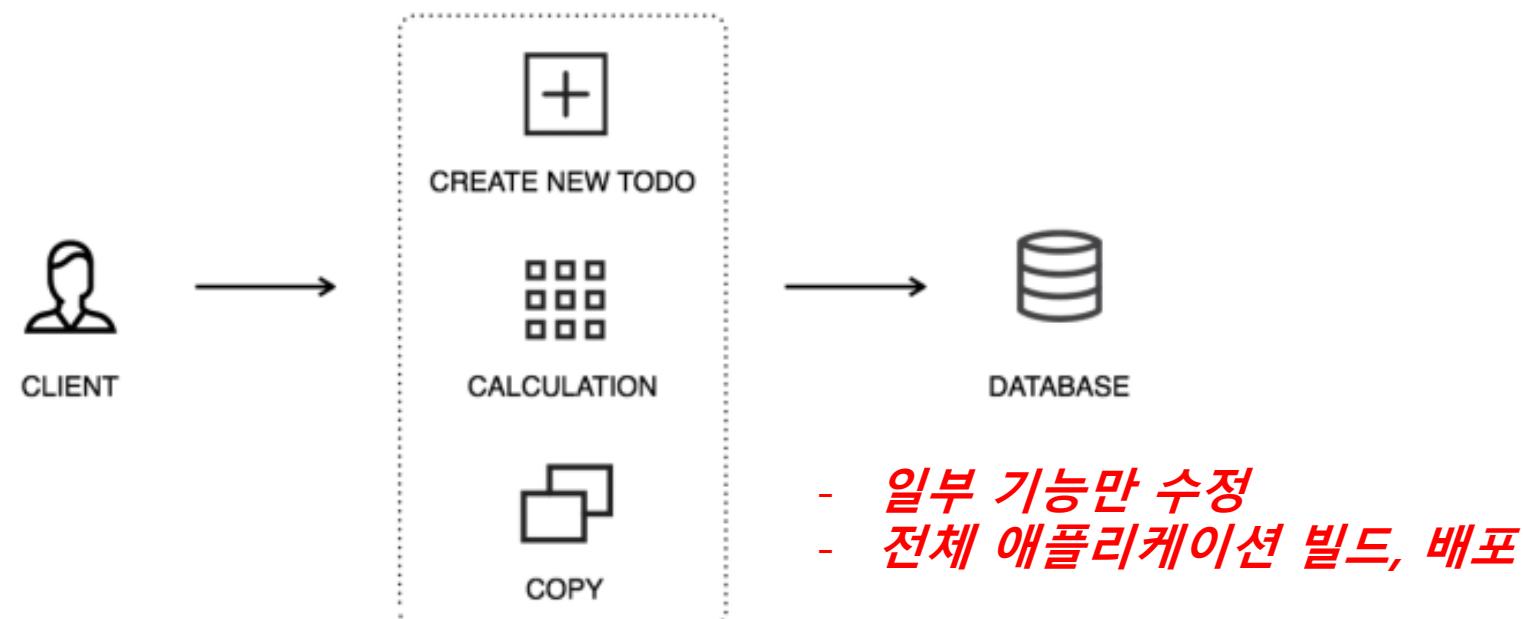
- Architecture 스타일과 패턴



Software Architecture

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

The Monolithic Architecture



All services combined into one build,
written in the same language and application framework

Software Architecture

- What is the Microservice?

Small autonomous services that work together

- *Sam Newman*

The screenshot shows the homepage of ThoughtWorks' website. At the top, there's a navigation bar with links for ESPAÑOL, PORTUGUÉS, DEUTSCH, 中文, a search bar, and buttons for NEWS, EVENTS, and CONTACT US. The main header reads "ThoughtWorks®". Below it, a large banner features a portrait of Sam Newman, a man with long hair and a beard, smiling. The background of the banner is a blue wall covered in numerous sticky notes and diagrams, representing a typical Agile or DevOps workspace. The name "Sam Newman" is prominently displayed in white text over the banner. To the right of the banner, there are links for Clients, Services, Products, Insights, About us, and Careers. On the left side of the banner, there's a small photo of Sam Newman and an orange button labeled "ALUM". Below the banner, social media links for Twitter (@samnewman), GitHub (snewman), and a personal website (samnewman.io) are listed. A bio text describes Sam as a technical consultant at ThoughtWorks since 2004, mentioning his work with people to build better software systems, contributions to O'Reilly articles and conferences, and his current focus on Python, Clojure, Infrastructure Automation, and Cloud systems. It also notes his authorship of "Building Microservices" and directs readers to his personal website.

ESPAÑOL PORTUGUÉS DEUTSCH 中文

Search thoughtworks.com

NEWS EVENTS CONTACT US

ThoughtWorks®

Clients Services Products Insights About us Careers

Sam Newman

CONSULTANT

I'm a technical consultant at ThoughtWorks, and have been working here since 2004. If you asked me what I do, I'd say "I work with people to build better software systems". I've written articles for O'Reilly, presented at conferences, and sporadically commit to open source projects. I've spent most of my career so far coding in Java, but I now spend a lot of my time with Python, Clojure, Infrastructure Automation and Cloud systems.

I am also the author of [Building Microservices](#) from O'Reilly. You can find my personal website at samnewman.io.

ALUM

@samnewman

snewman

samnewman.io

Software Architecture

- 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

Software Architecture

- What is the Microservice?

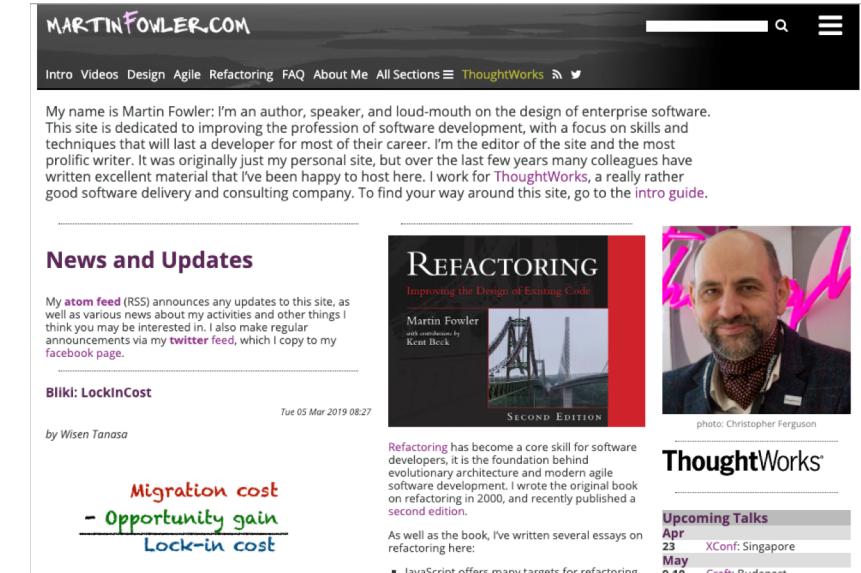
These services are built around ***business capabilities*** and ***independently deployable*** by fully ***automated deployment*** machinery...contd

Software Architecture

- 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*



The screenshot shows the homepage of Martin Fowler's website, martinfowler.com. The header includes navigation links for Intro, Videos, Design, Agile, Refactoring, FAQ, About Me, All Sections, ThoughtWorks, and social media icons for RSS, Twitter, and Facebook. The main content area features a bio about Martin Fowler, a section for News and Updates (with a recent post by Wisen Tanasa), a sidebar for his book "Refactoring: Improving the Design of Existing Code" (Second Edition), and a sidebar for ThoughtWorks. The footer contains sections for Upcoming Talks, a calendar for April and May, and a note about JavaScript refactoring.

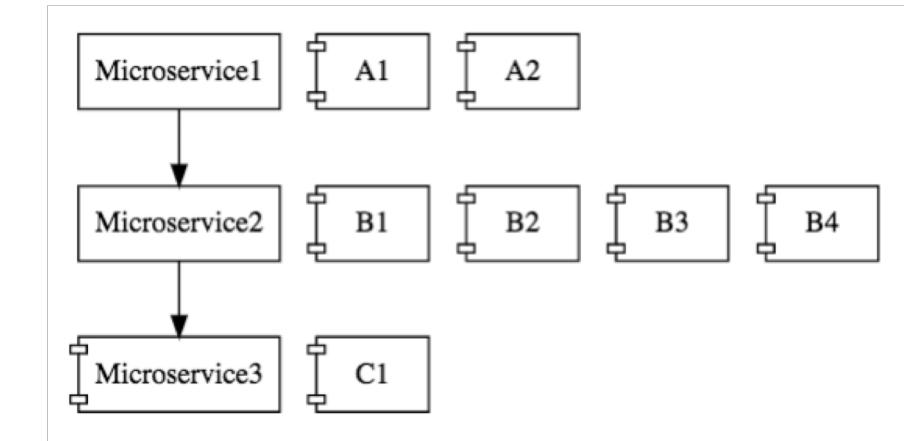
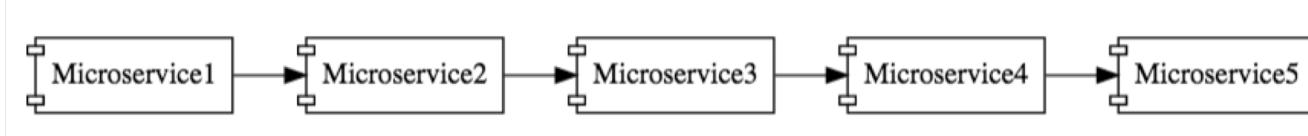
Software Architecture

- What is the Microservice?

1) RESTful

2) Small Well Chosen Deployable Units

3) Cloud Enabled



Software Architecture

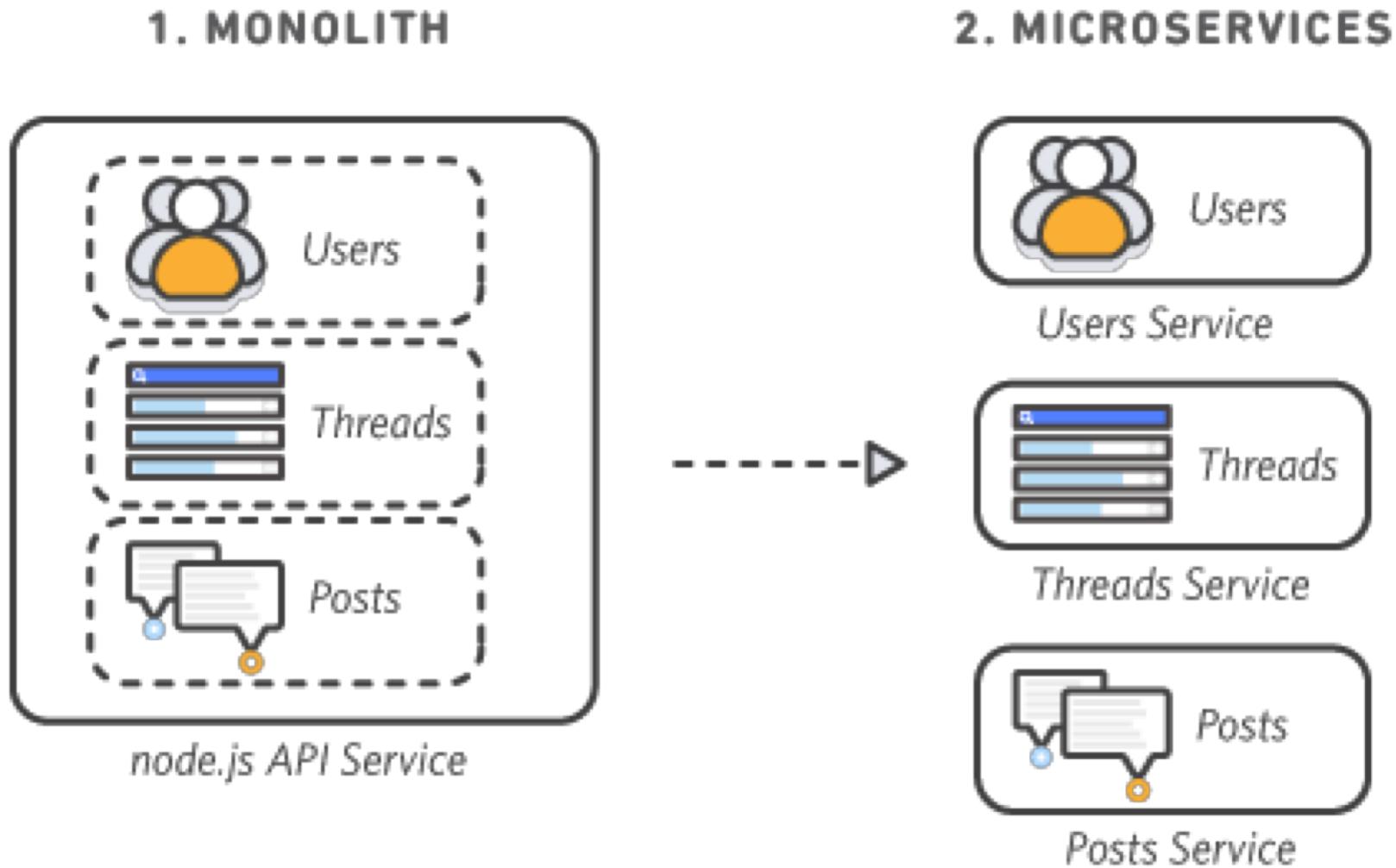
- Monolith vs Microservice Architecture



Monolith vs Microservices

Software Architecture

- Monolith vs Microservice Architecture



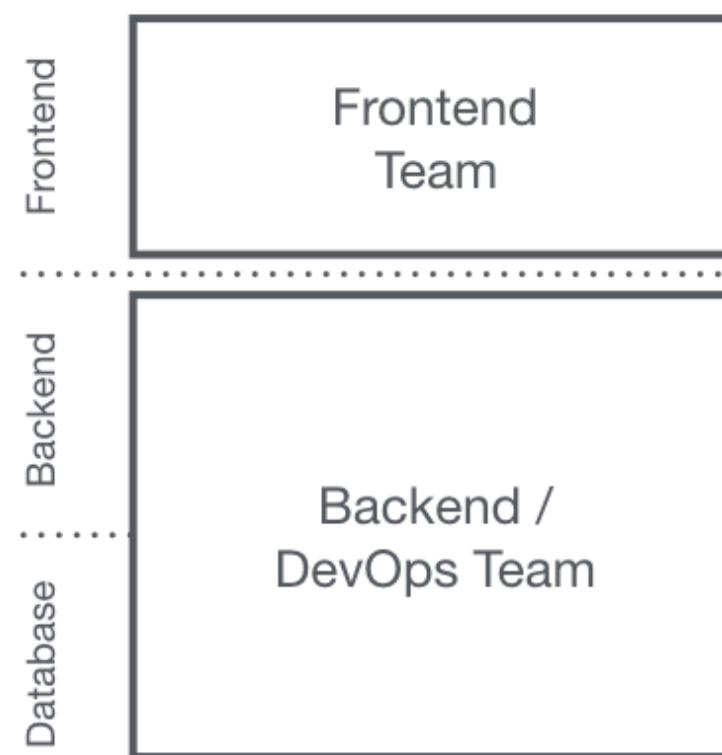
Software Architecture

- Monolith vs Front & Back vs Microservice Architecture

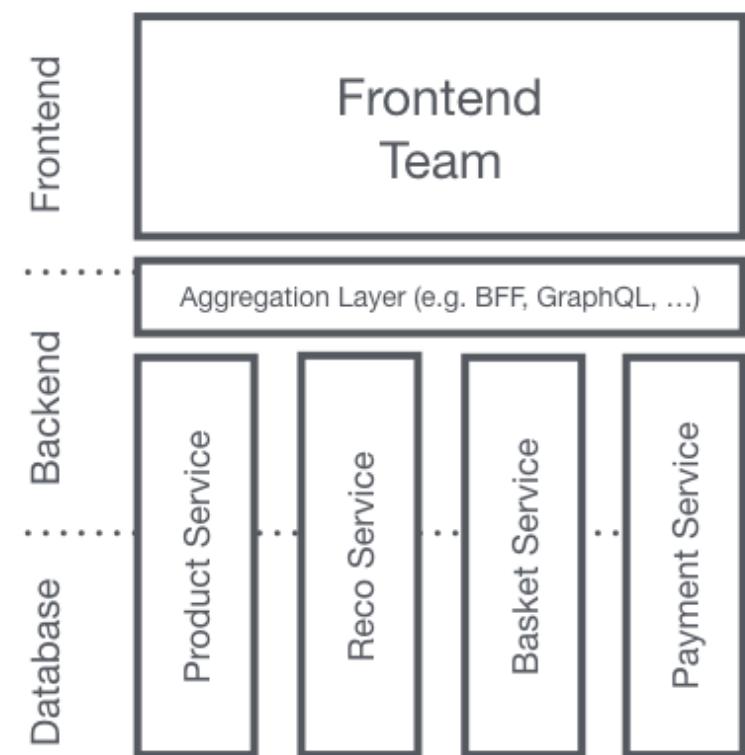
The Monolith



Front & Back



Microservices



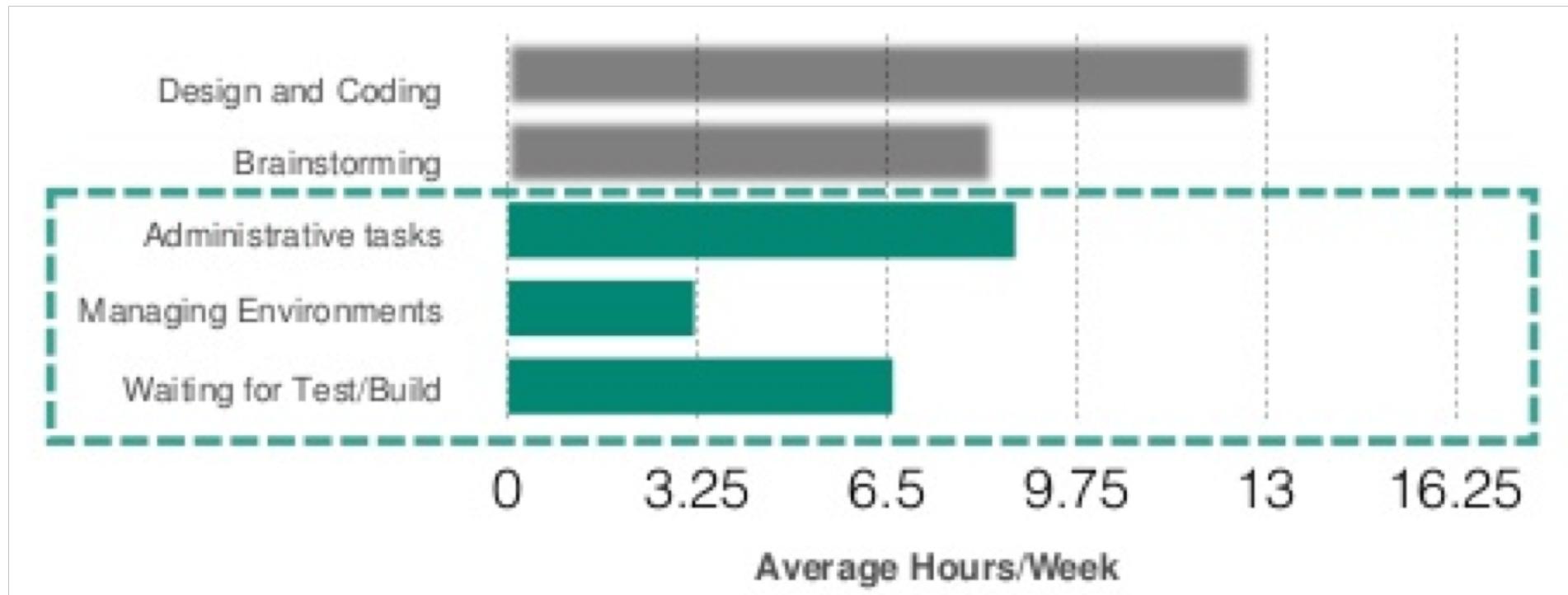
Software Architecture

- How Regular is Regular for High Performing Cloud Native Enterprise?

Company	Deploy Frequency	Deploy Lead Time
Amazon	23,000 / day	MINUTES
Google	5,500 / day	MINUTES
Netflix	500 / day	MINUTES
Facebook	1 / day	HOURS
Twitter	3 / week	HOURS
Typical Enterprise	Once every 9+ months	MONTHS or QUARTERS

Software Architecture

- Software developers spend too much time ***NOT writing software***



Software Architecture

- Microservice 특징

- 1) *Challenges*
- 2) *Bounded Context*
- 3) *Configuration Management*
- 4) *Dynamic Scale Up And Scale Down*
- 5) *Visibility*

Software Architecture

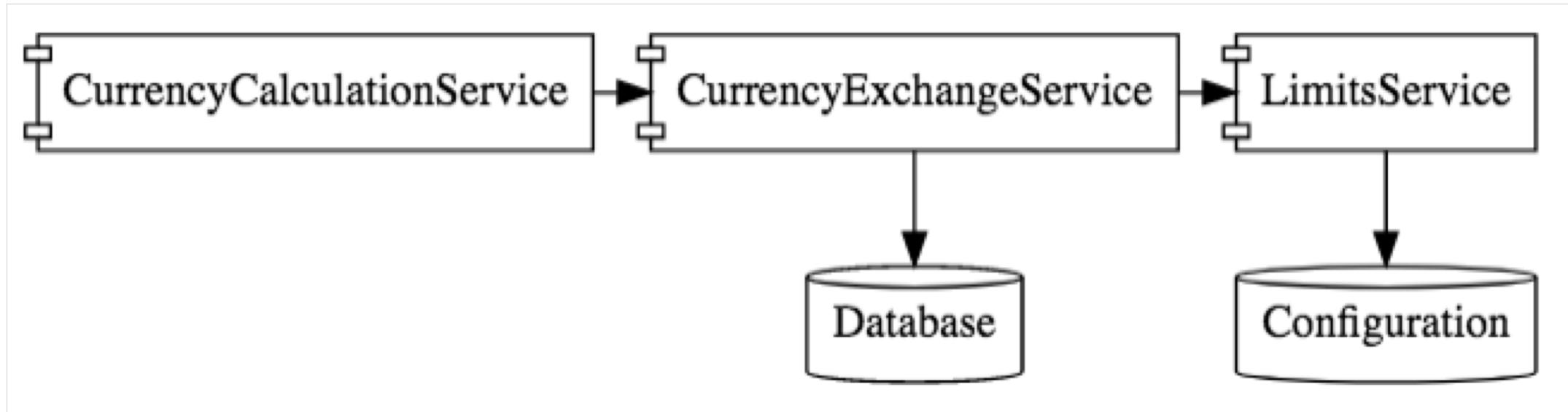
- Microservice 특징

6) *Solution*

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

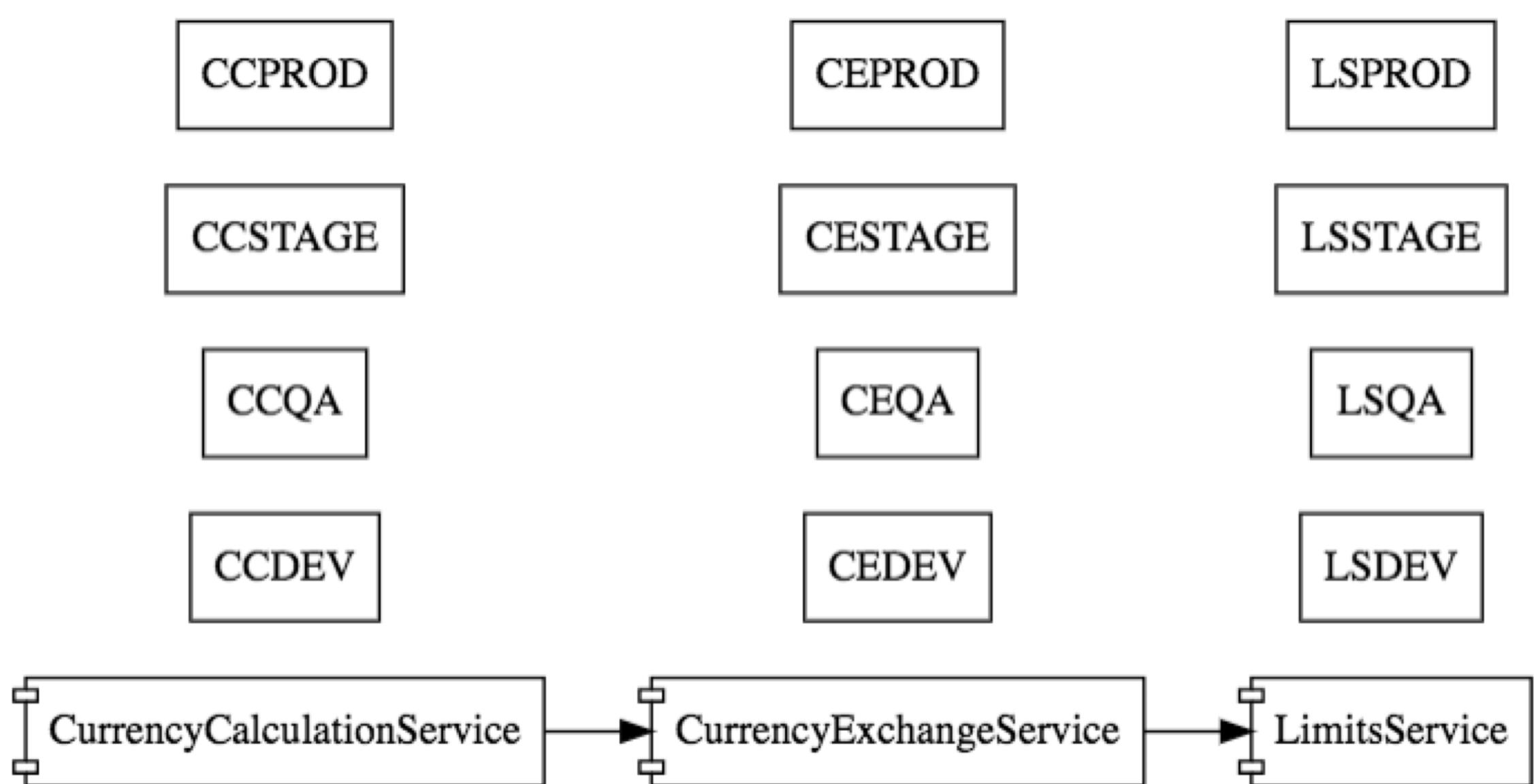
Software Architecture

- Microservice



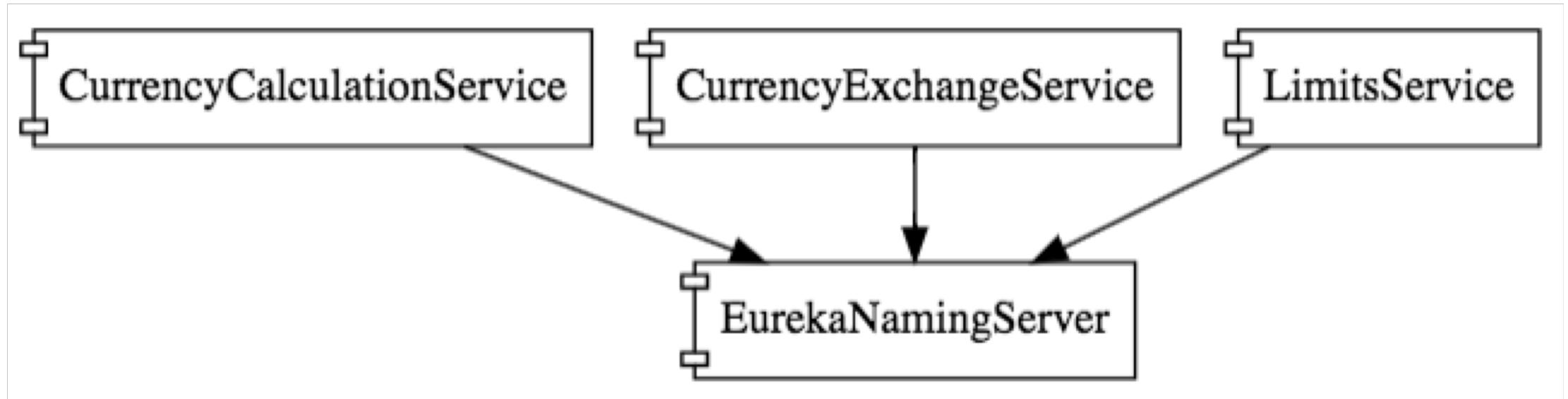
Software Architecture

- Microservice Environments



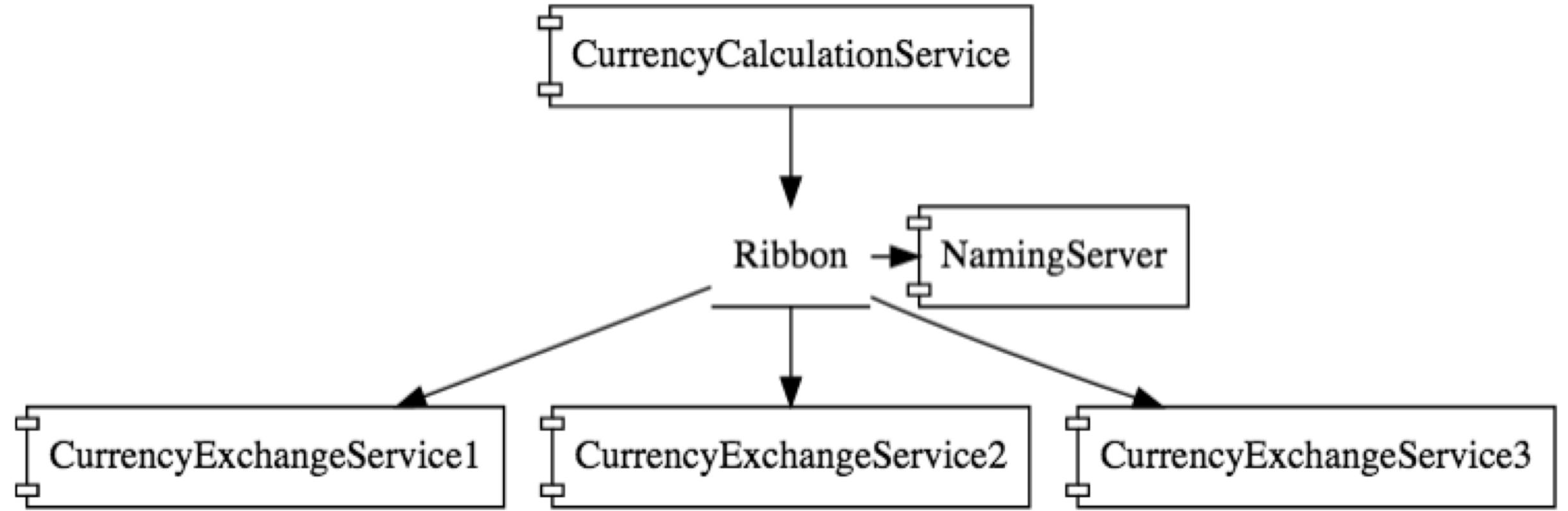
Software Architecture

- Microservice – Eureka Naming Server



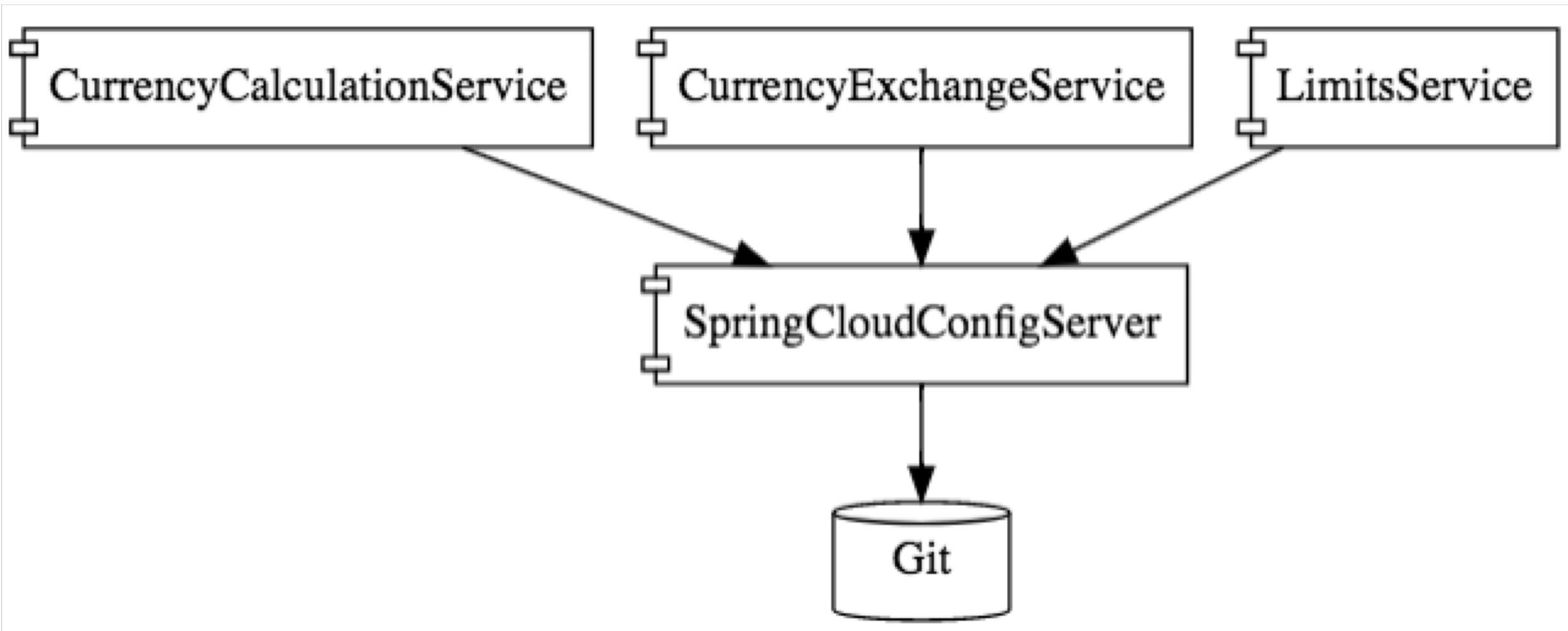
Software Architecture

- Microservice – Ribbon Load Balancing



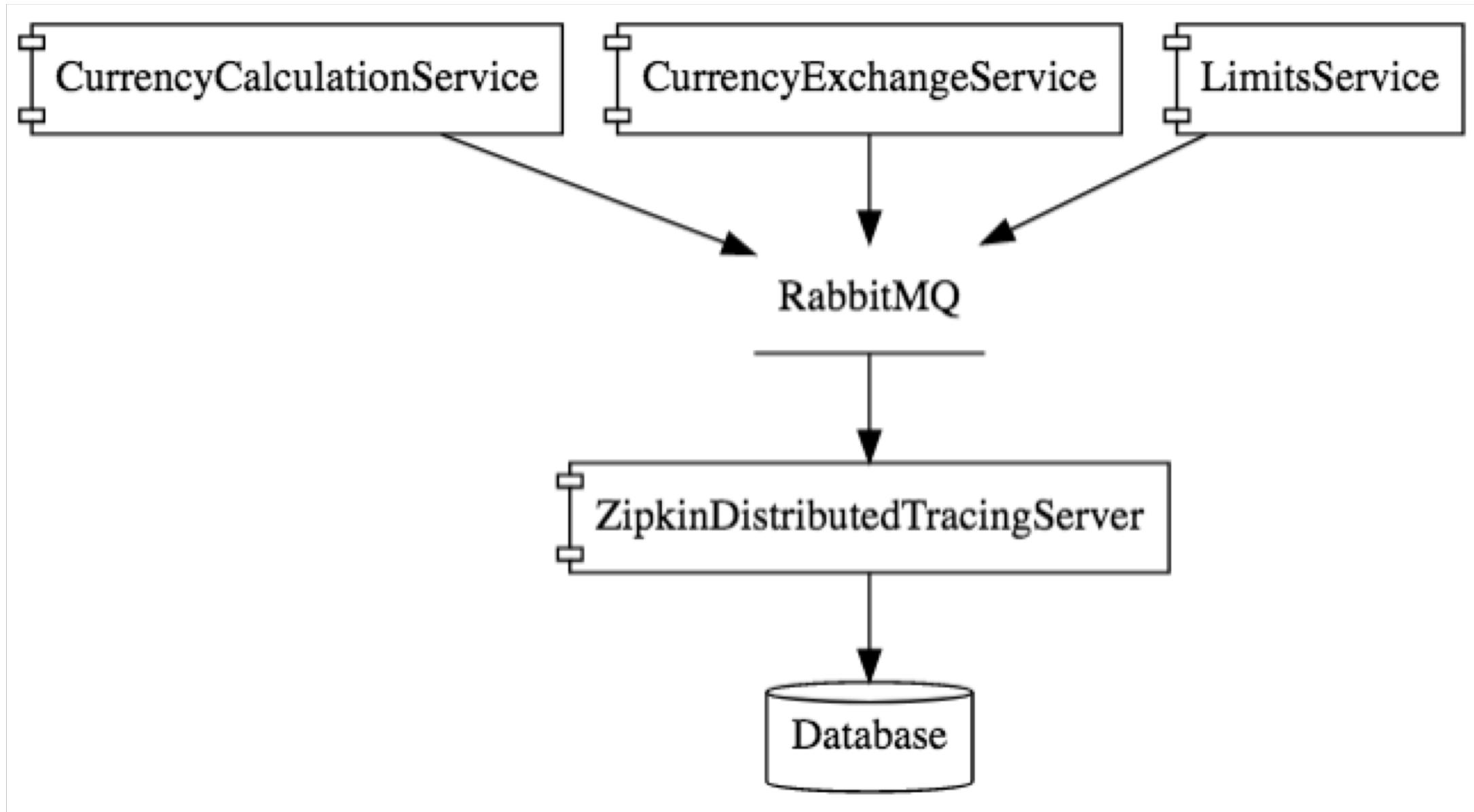
Software Architecture

- Microservice – Spring Cloud Config Server



Software Architecture

- Microservice – Zipkin Distributed Tracing



Software Architecture

- Microservice – API Gateways
 - Authentication, Authorization and Security
 - Rate Limits
 - Fault Tolerance
 - Service Aggregation

Software Architecture

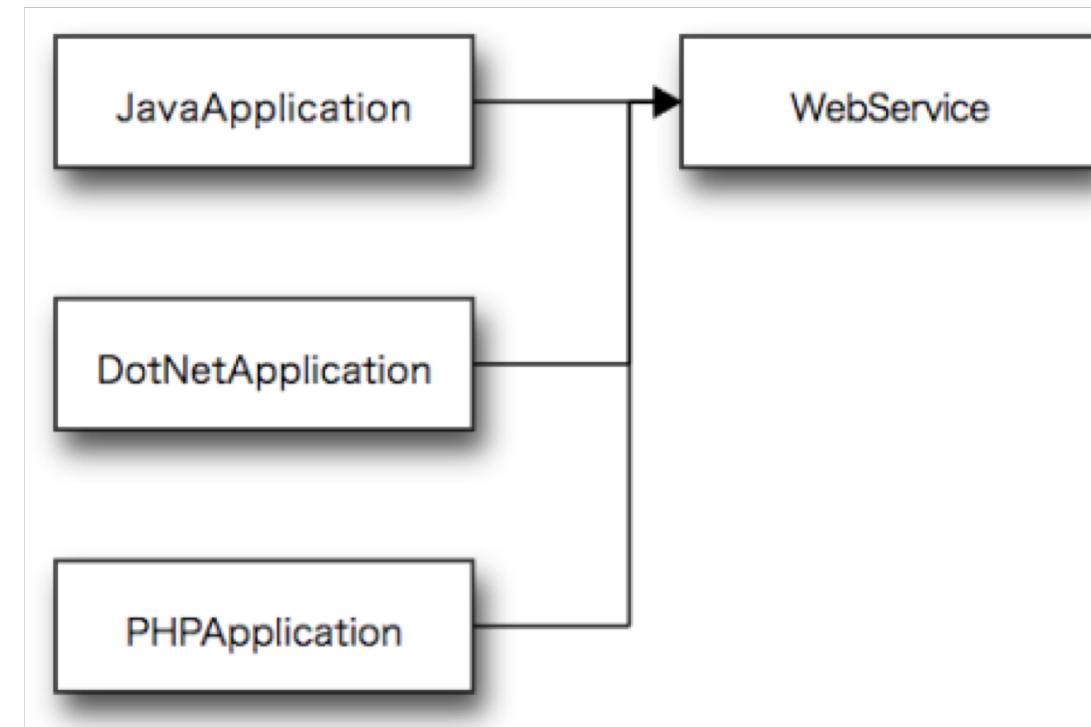
- Microservice – Web Services
 - Service delivered over the web?
 - Is the Todo Management Application a Web Service?
 - Can I reuse the Business Layer by creating a JAR?
 - How can I make my Todo application consumable by other applications?

Software Architecture

- Web Services – 3 KEYS
 - Designed for machine-to-machine (or application-to-application) interaction
 - Should be interoperable - Not platform dependent
 - Should allow communication over a network

Software Architecture

- Web Services
 - How does data exchange between applications take place?
 - How can we make web services platform independent?



Software Architecture

- Web Services
 - Data Format
 - XML
 - JSON

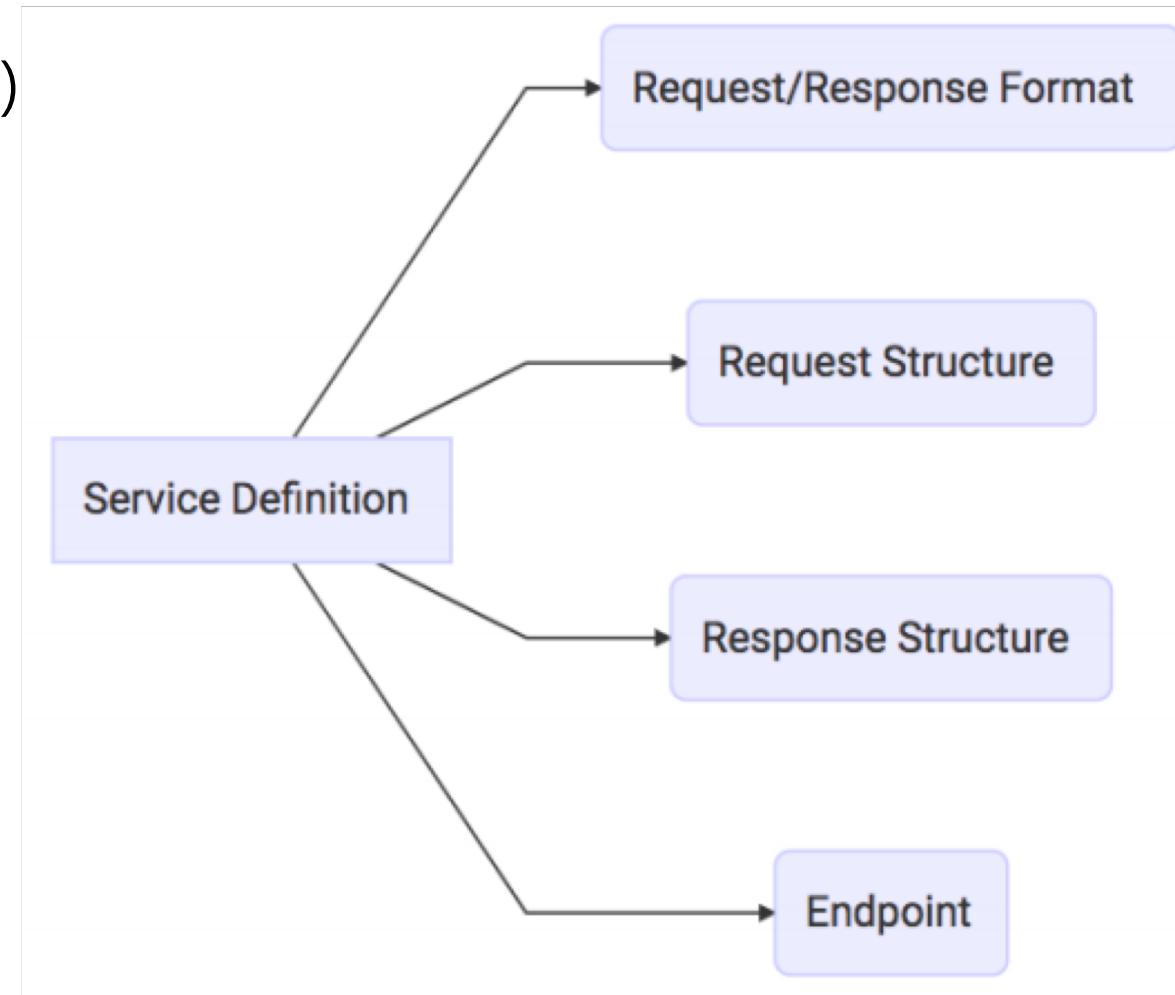
```
<getCourseDetailsRequest>
  <id>Course1</id>
</getCourseDetailsRequest>
```

- JSON

```
[
  {
    "id": 1,
    "name": "Even",
    "birthDate": "2017-07-10T07:52:48.270+0000"
  },
  {
    "id": 2,
    "name": "Abe",
    "birthDate": "2017-07-10T07:52:48.270+0000"
  }
]
```

Software Architecture

- Web Services
 - How does the Application A know the format of Request and Response?
 - How does Application A and Web Service convert its internal data to (XML or JSON)

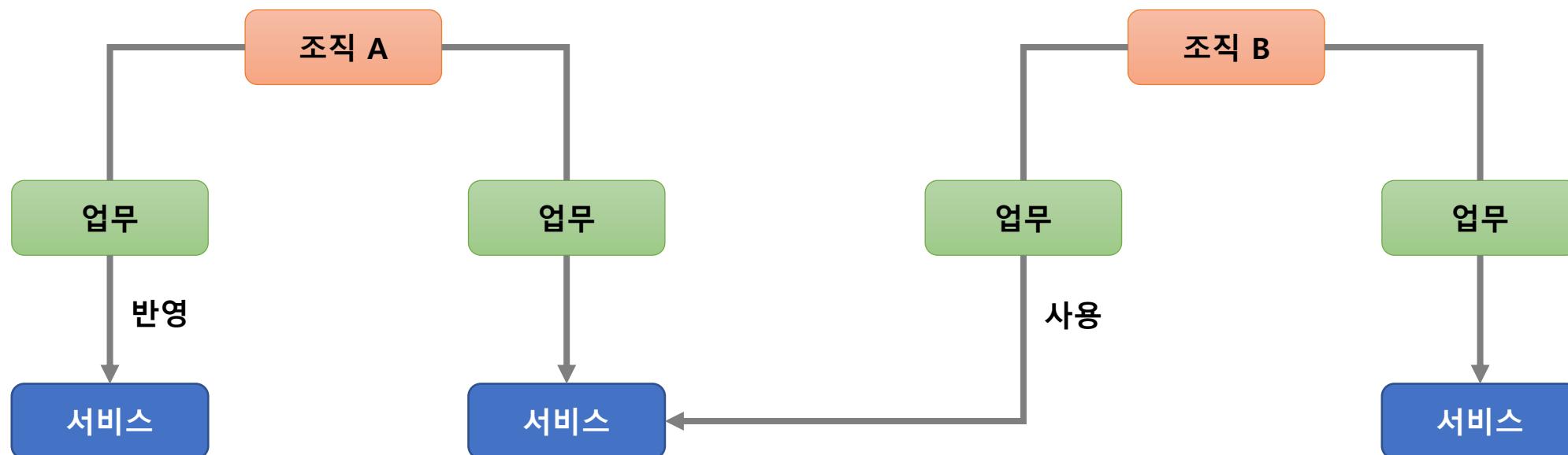




Service Oriented Architecture

Service Oriented Architecture

- SOA
 - 업무 처리 단위를 각각의 서비스로 반영
 - 데이터 중심 → 서비스 중심 설계
 - 서비스를 통해 비즈니스 환경 변화와 업무 변화에 대응



Service Oriented Architecture

- SOA 특징

1) 서비스 계약

2) 서비스 가용성

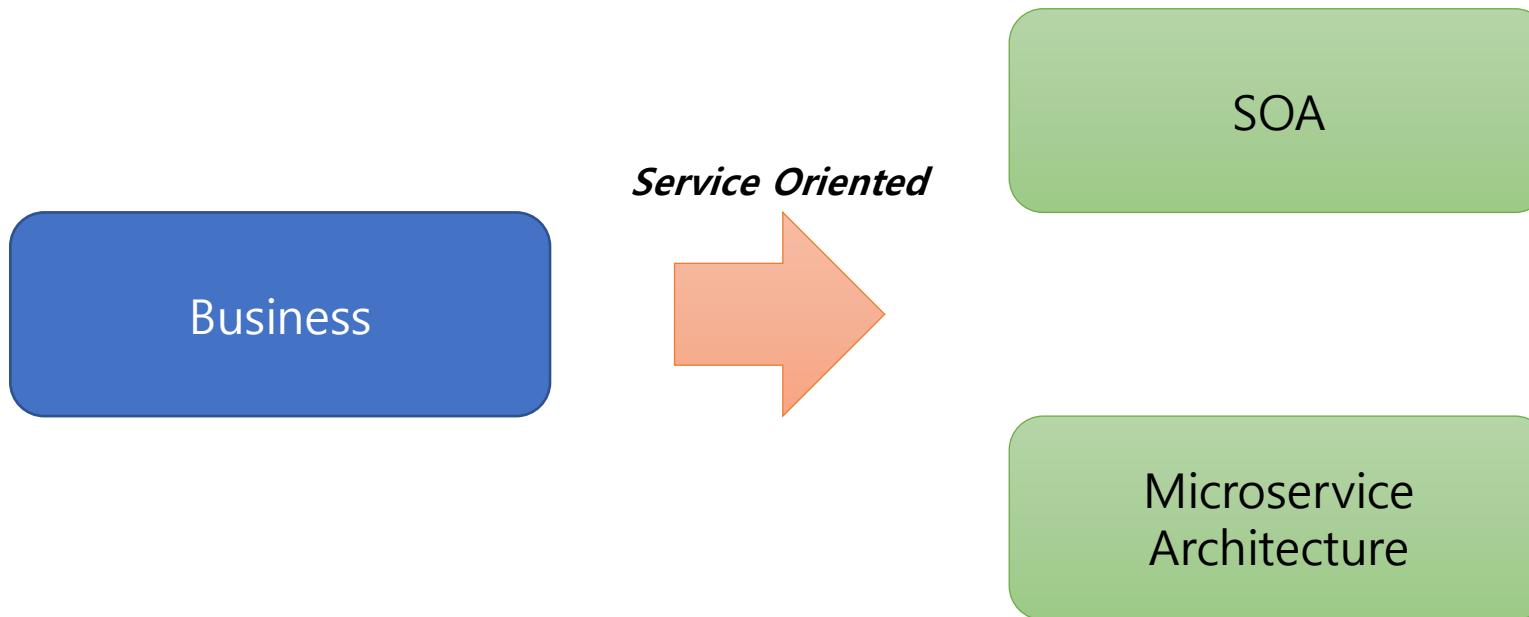
3) 서비스 권한

4) 트랜잭션

5) 서비스 관리

Service Oriented Architecture

- SOA와 Microservice Architecture와의 차이점
 - 공통점: 비즈니스 변화 대응을 위한 서비스 중심의 아키텍쳐
 - 서비스의 상대적 크기와 관심사, *Ownership*, 기술 구조



Service Oriented Architecture

- SOA와 Microservice Architecture와의 차이점
 - SOA
 - 비즈니스 측면에서의 서비스 재사용성
 - *ESB(Enterprise Service Bus)*라는 서비스 채널 이용 → 서비스 공유, 재사용
 - MSA
 - 한 가지 작은 서비스에 집중
 - 서비스 공유하지 않고 독립적 실행

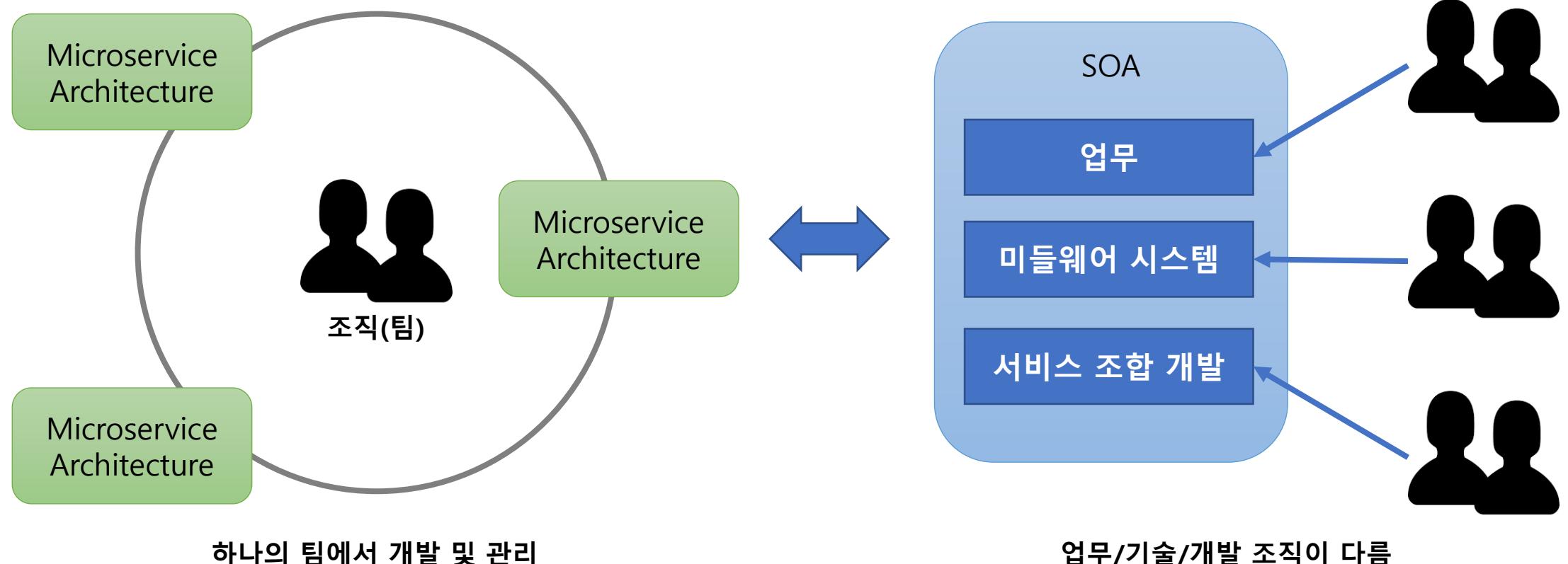
Service Oriented Architecture

- SOA와 Microservice Architecture와의 차이점
 - 서비스의 상대적 크기와 관심사



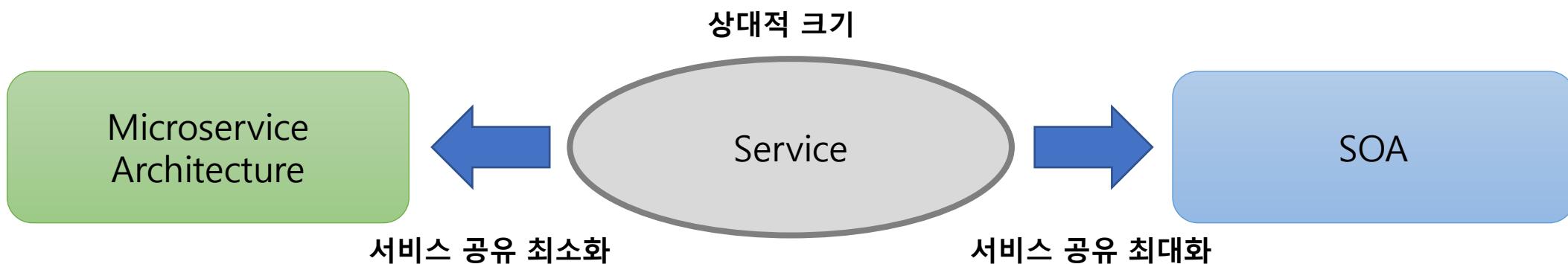
Service Oriented Architecture

- SOA와 Microservice Architecture와의 차이점
 - 서비스의 Ownership



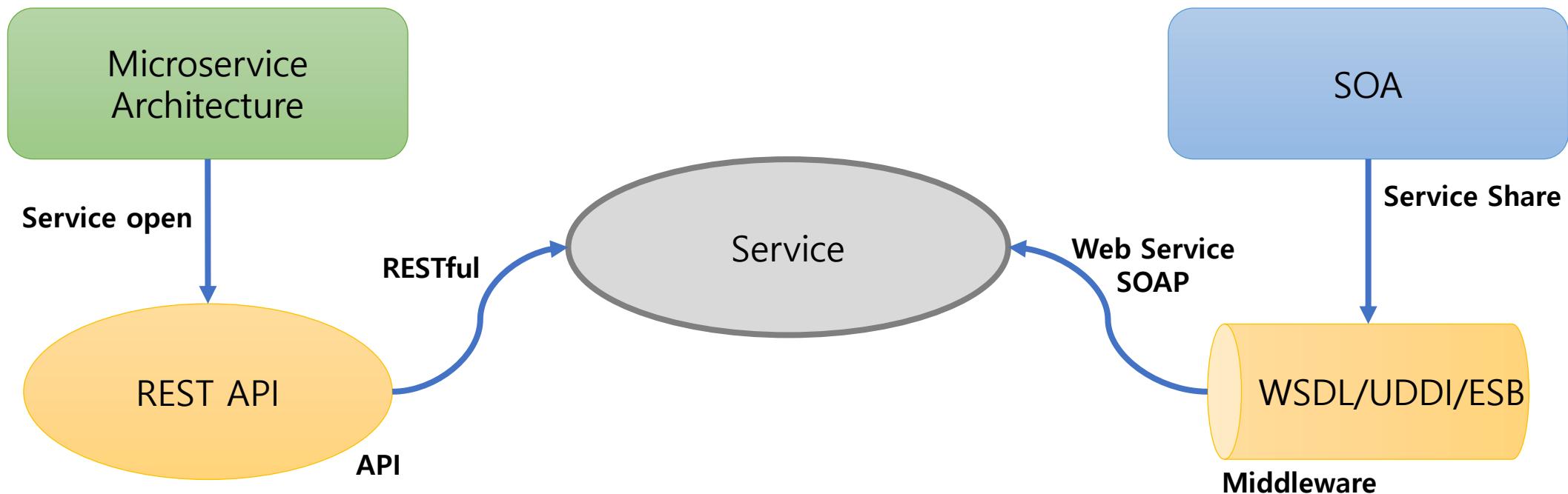
Service Oriented Architecture

- SOA와 Microservice Architecture와의 차이점
 - 서비스의 공유 지향점
 - *SOA* - 재사용을 통한 비용 절감
 - *MSA* - 서비스 간의 결합도를 낮추어 변화에 능동적으로 대응



Service Oriented Architecture

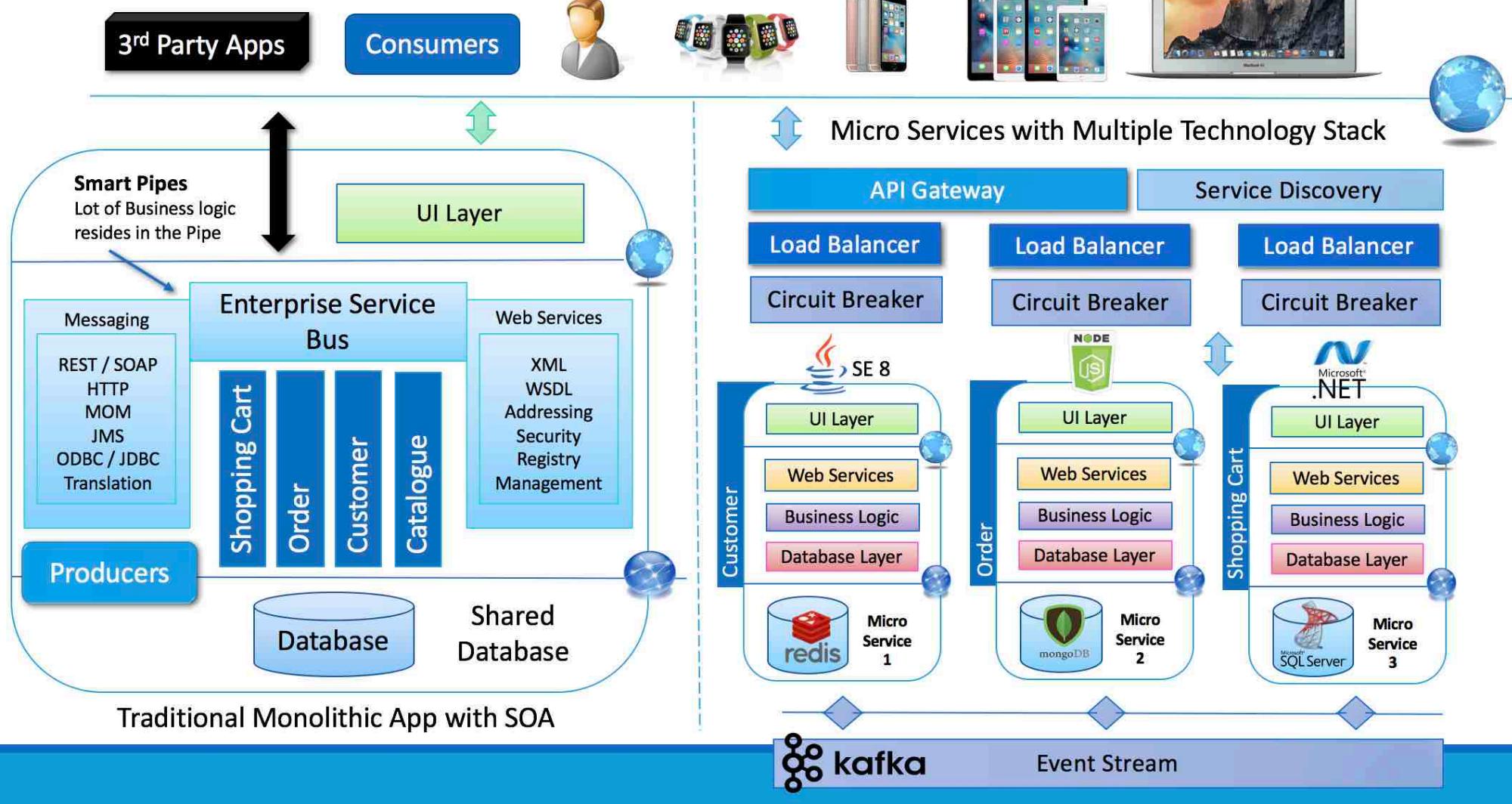
- SOA와 Microservice Architecture와의 차이점
 - 기술 방식
 - *SOA* - 공통의 서비스를 ESB에 모아 사업 측면에서 공통 서비스 형식으로 서비스 제공
 - *MSA* - 각 독립된 서비스가 노출된 REST API를 사용



Service Oriented Architecture

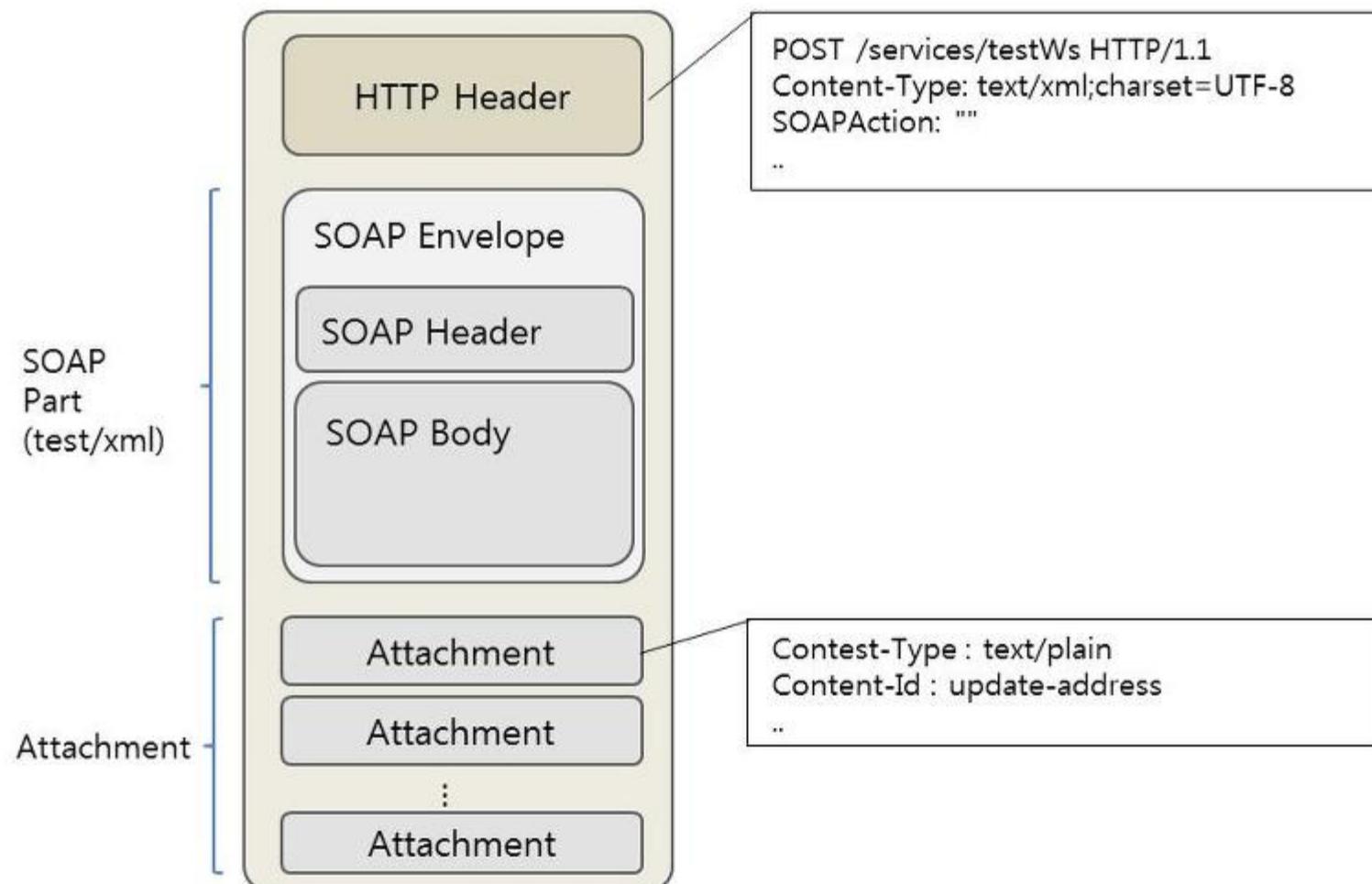
- SOA와 Microservice Architecture와의 차이점

SOA vs. Micro Services Example



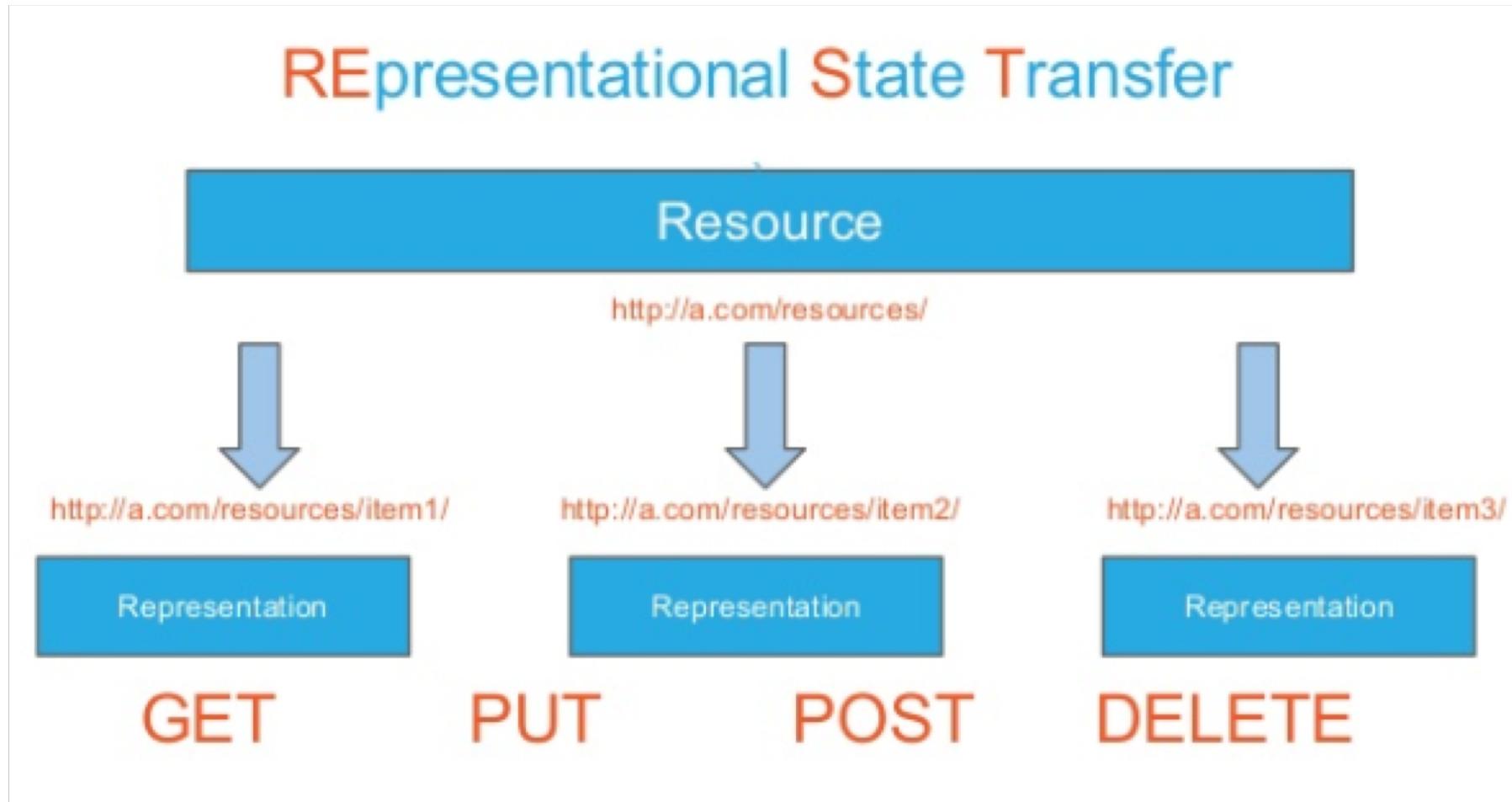
Service Oriented Architecture

- SOAP vs REST



Service Oriented Architecture

- SOAP vs REST

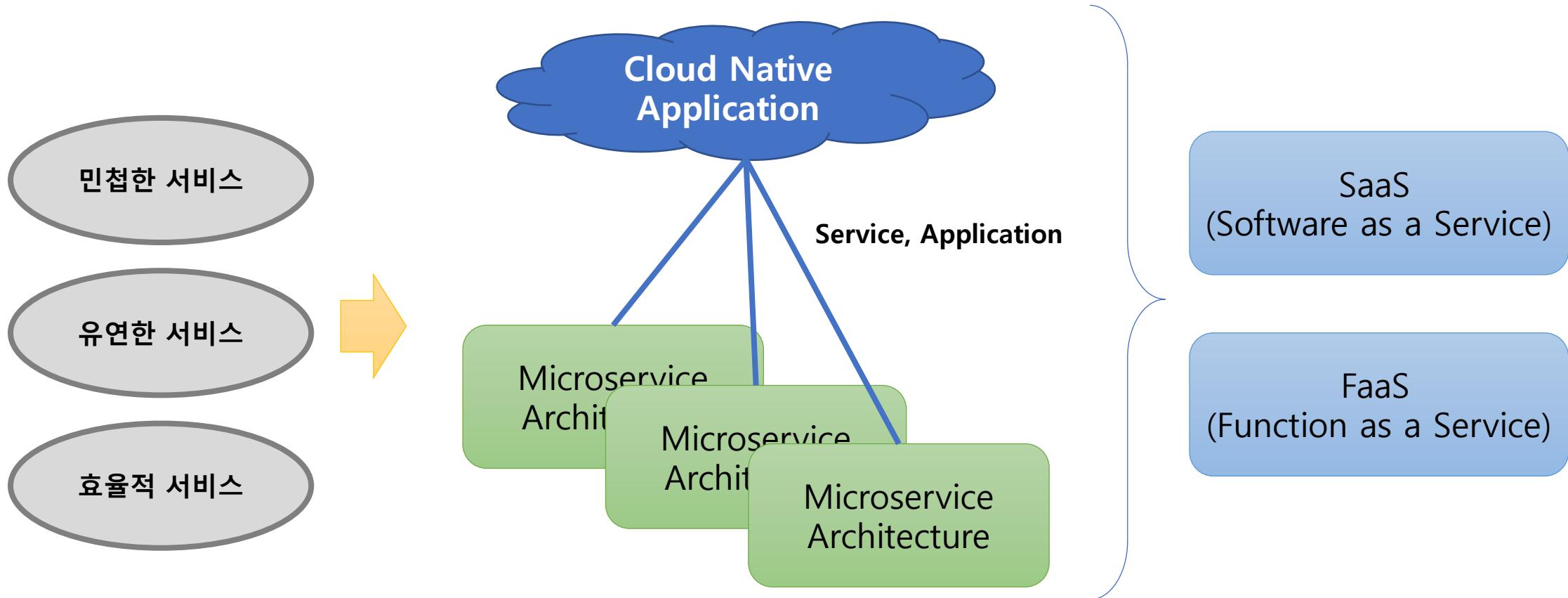




Cloud Native

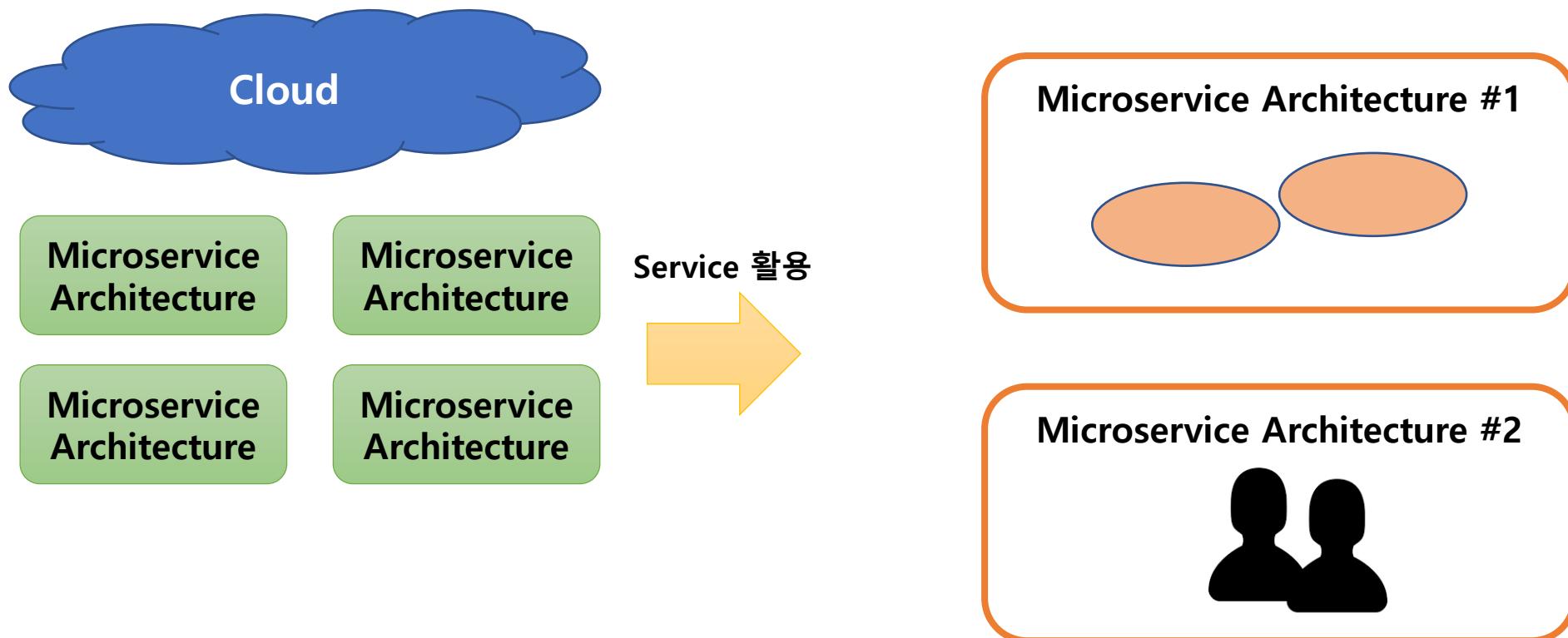
Cloud Native

- Cloud Native Application
 - 클라우드 네이티브 환경에서 SaaS난 FaaS 형태로 서비스되는 애플리케이션



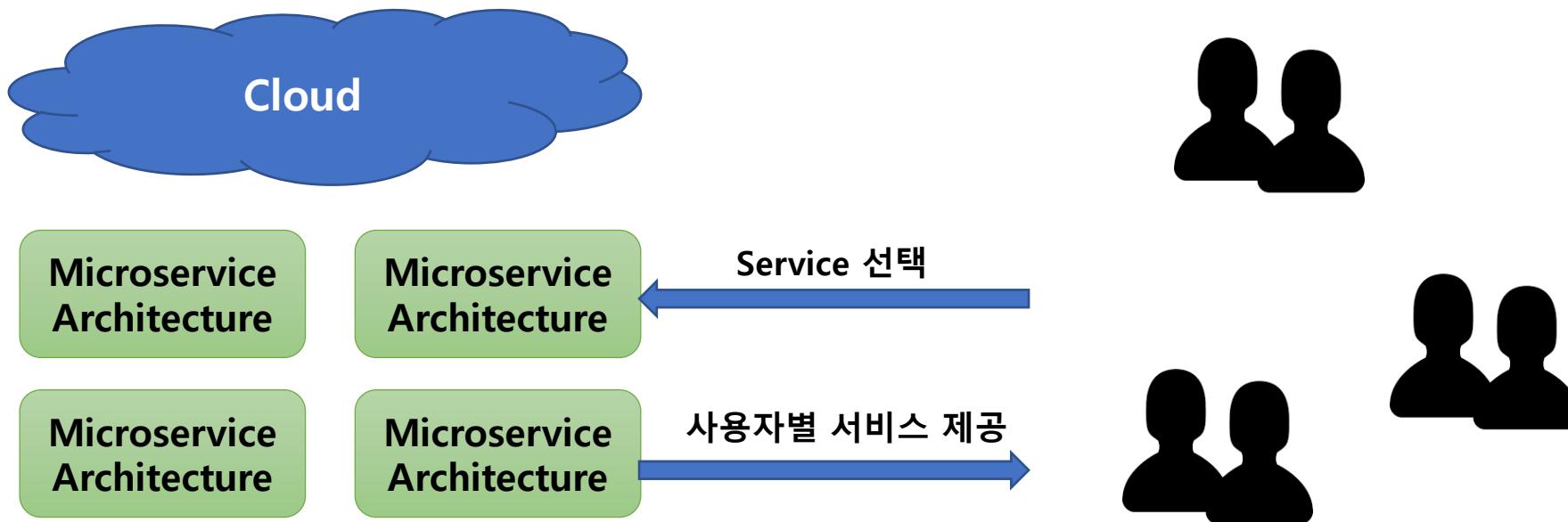
Cloud Native

- Cloud Native Application
 - Microservice



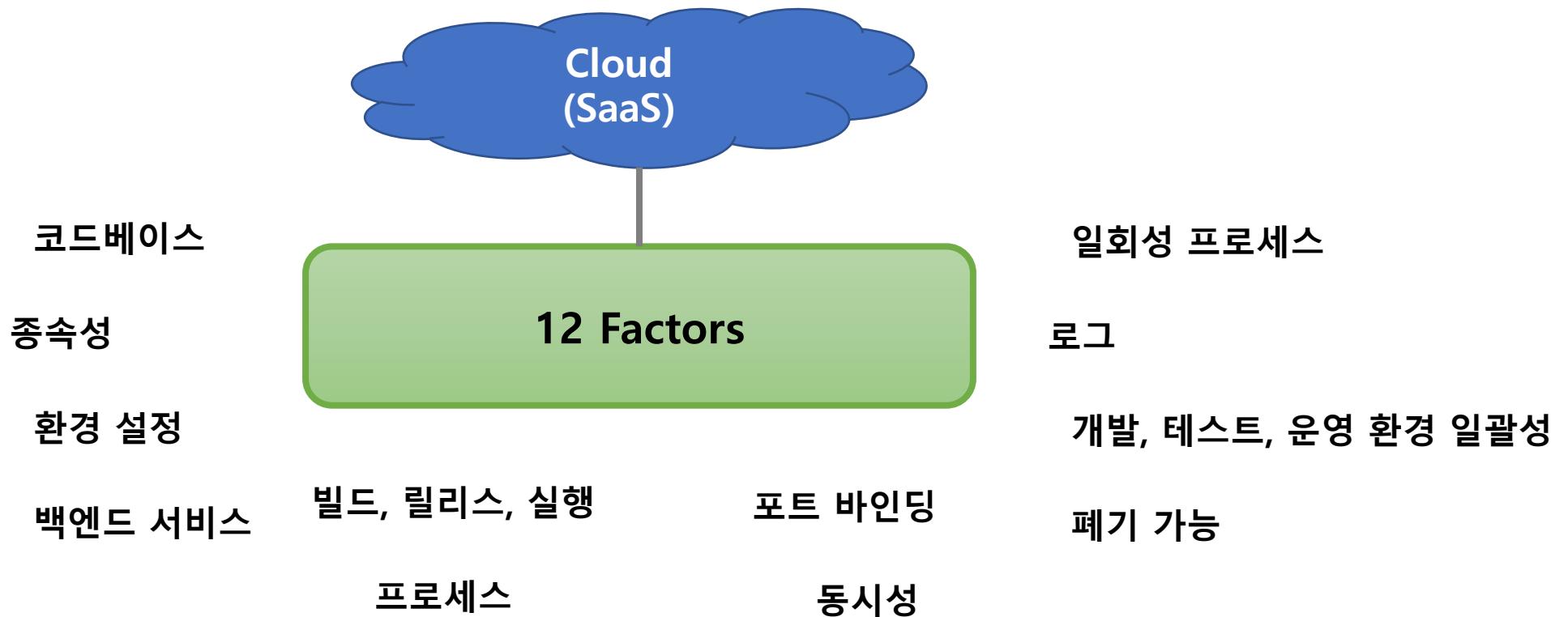
Cloud Native

- Cloud Native Application
 - SaaS



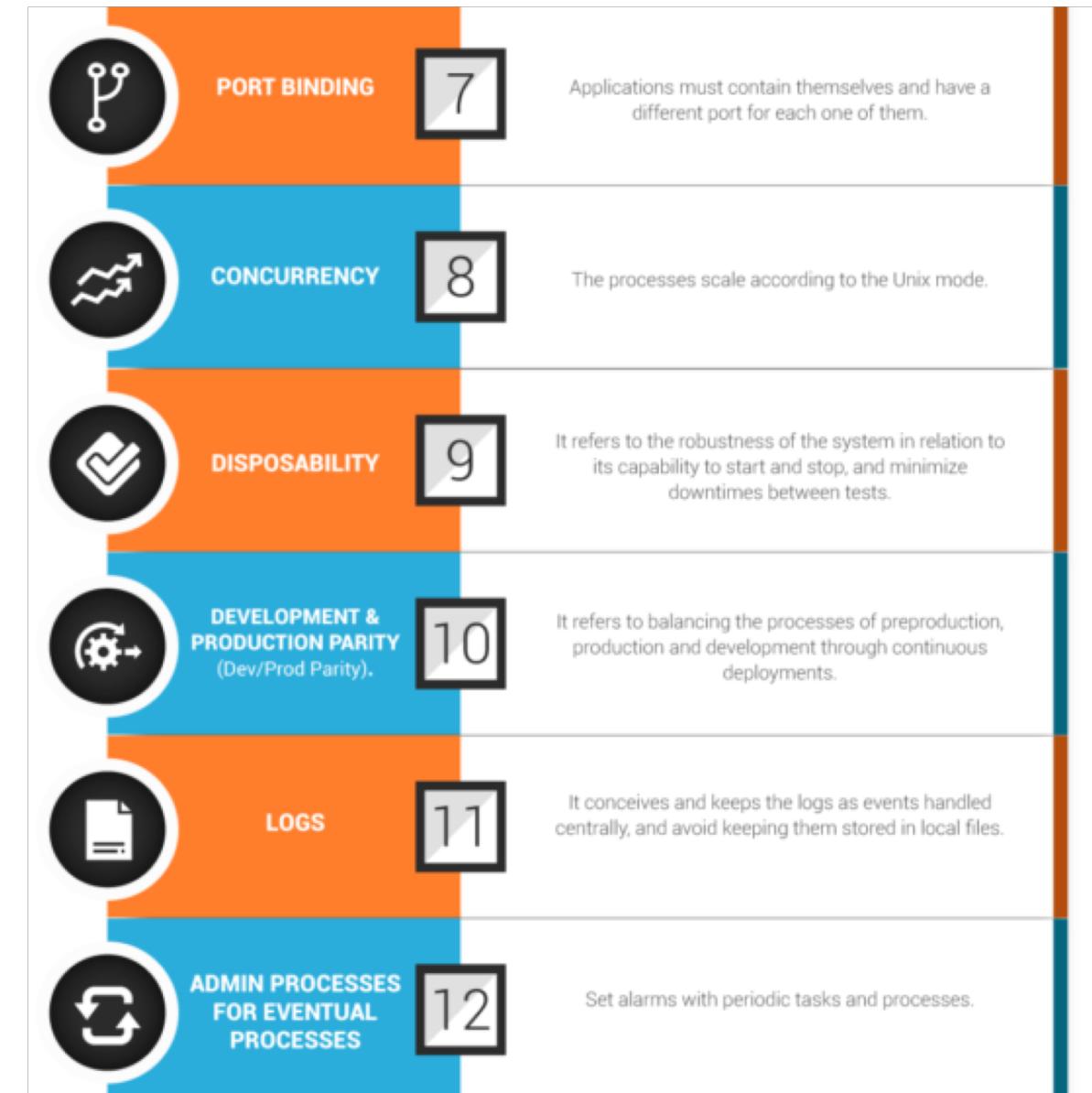
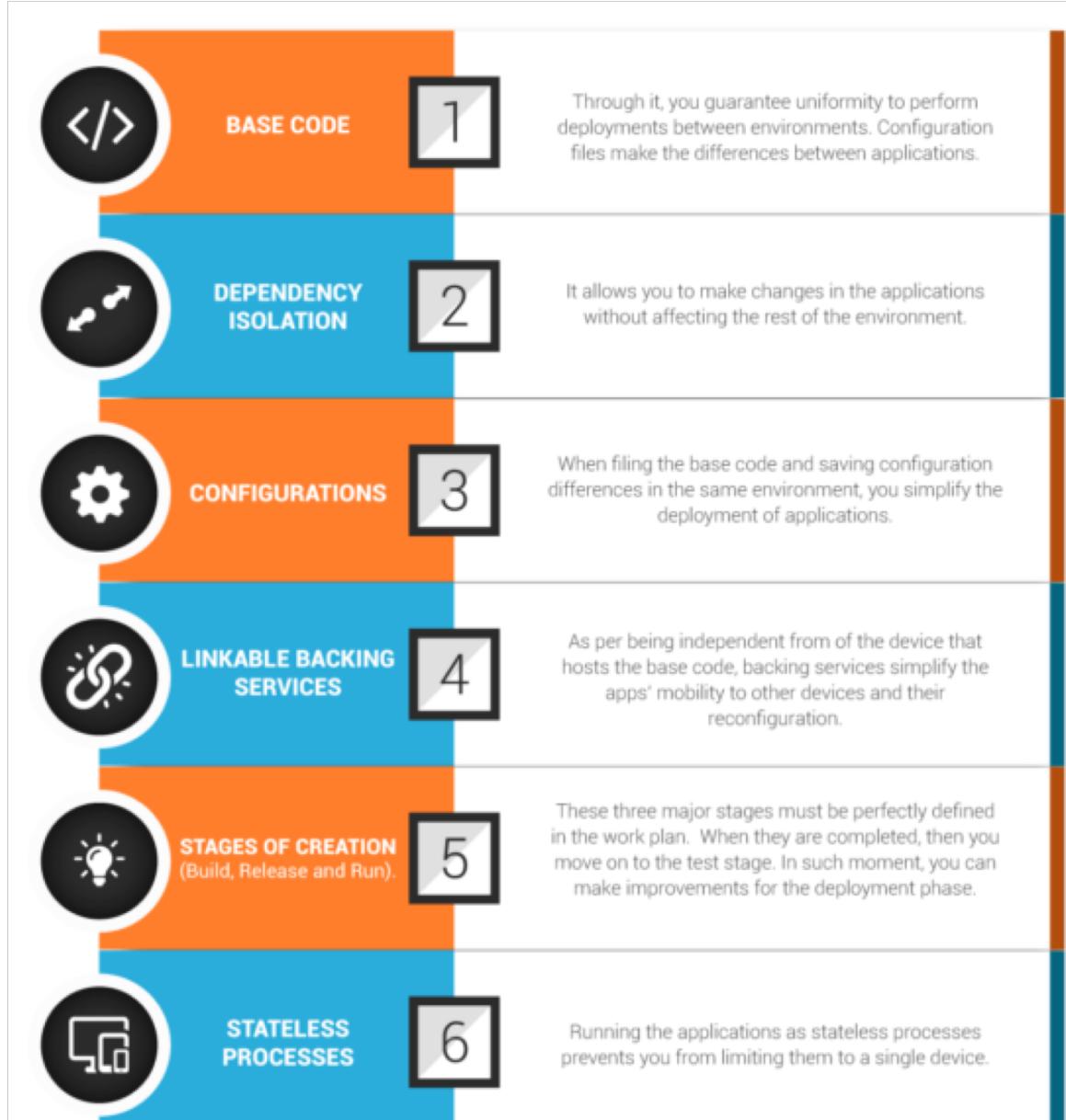
Cloud Native

- Cloud Native Application
 - 12 Factors



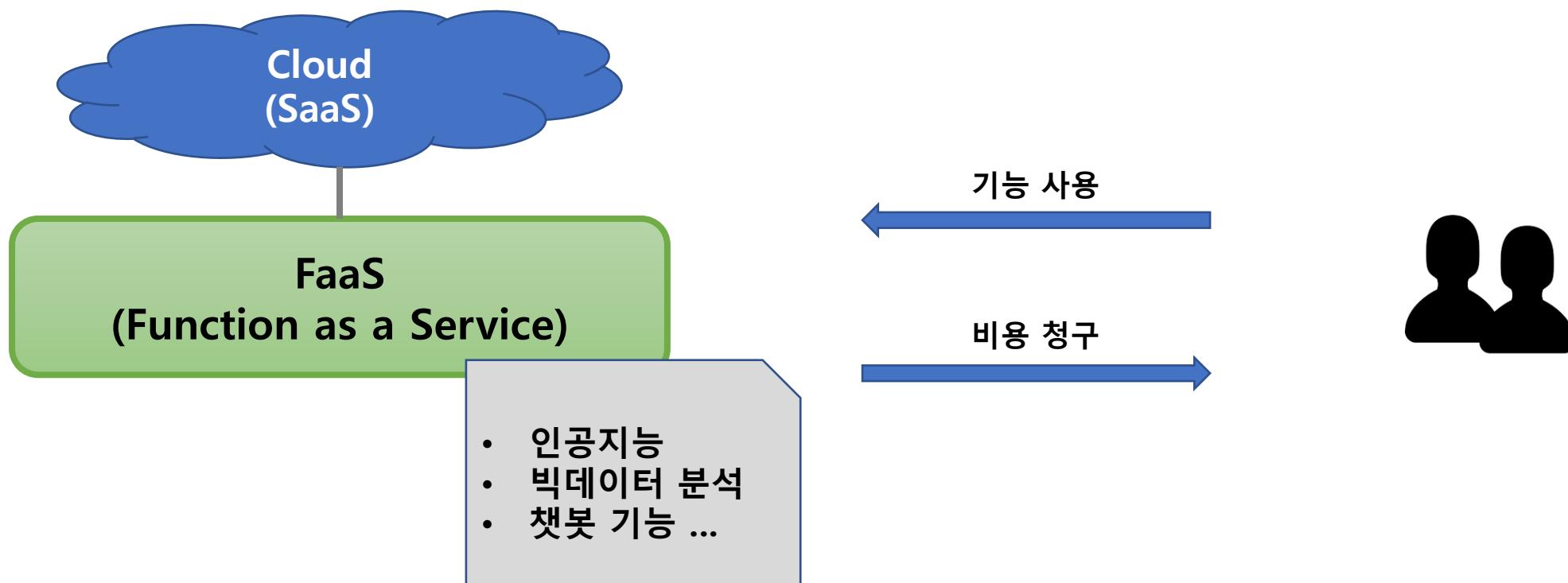
Cloud Native

- Cloud Native Application - 12 Factors



Cloud Native

- Cloud Native Application
 - FaaS(Function as a Service)
 - SW 수준보다 더 세분 된 기능 단위로 개발
 - 필요할 때 즉시 사용
 - 기능들의 조합을 통해서 새로운 서비스 생성



Cloud Native

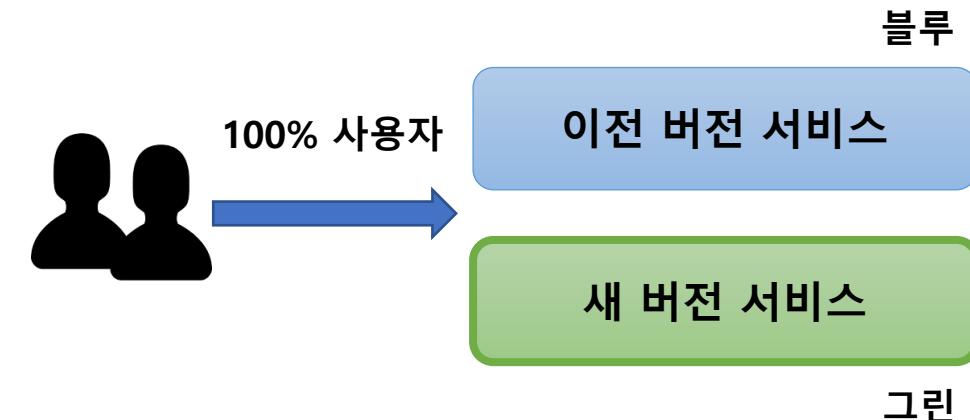
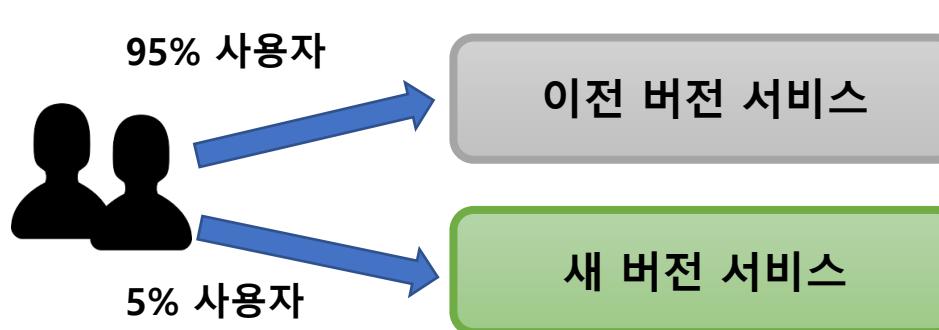
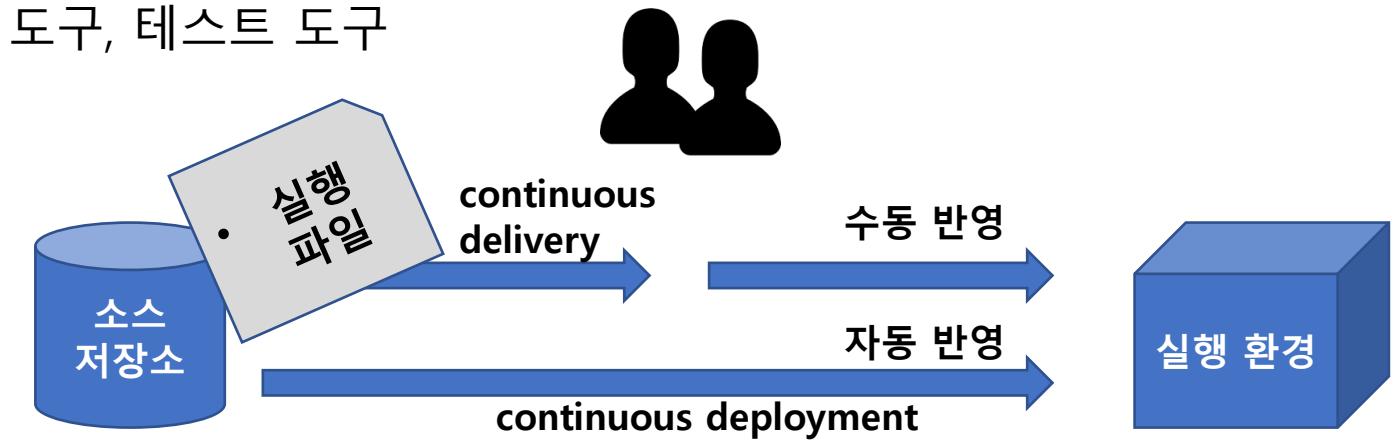
- Cloud Native Architecture
 - 확장 가능한 아키텍처
 - 시스템의 수평적 확장에 유연
 - 확장된 서버로 시스템의 부하 분산, 가용성 보장
 - 모니터링
 - 탄력적 아키텍처
 - 서비스 생성 - 통합 - 배포, 비즈니스 환경 변화에 대응 시간 단축
 - 분할 된 서비스 구조
 - 무상태 통신 프로토콜
 - 장애 격리 (Fault isolation)
 - 특정 서비스에 오류가 발생해도 다른 서비스에 영향 주지 않음

Cloud Native

- Cloud Native Infra
 - 컨테이너 기반 패키지
 - 시스템 또는, 서비스 애플리케이션 단위의 패키지
 - 동적 관리
 - 서비스의 추가와 삭제 자동으로 감지
 - 변경된 서비스 요청을 라우팅

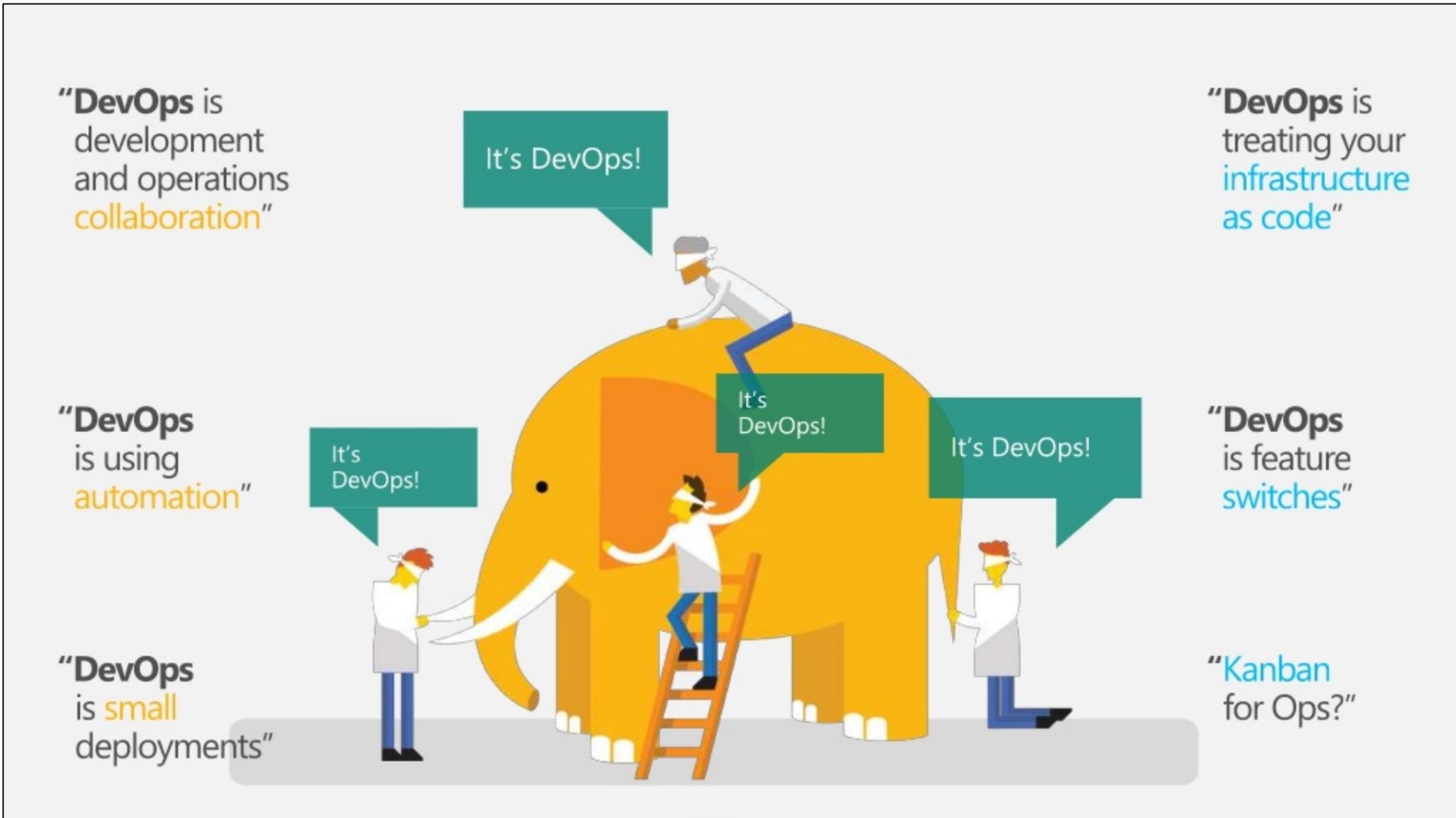
Cloud Native

- 지속적 통합과 배포
 - CI(Continuous Integration)
 - 통합 서버, 소스 관리 (SCM), 빌드 도구, 테스트 도구
 - ex) Jenkins
 - 지속적 배포
 - Continuous Delivery
 - Continuous Deployment
 - Pipe line
 - 카나리 배포와 블루그린 배포



Cloud Native

- DevOps

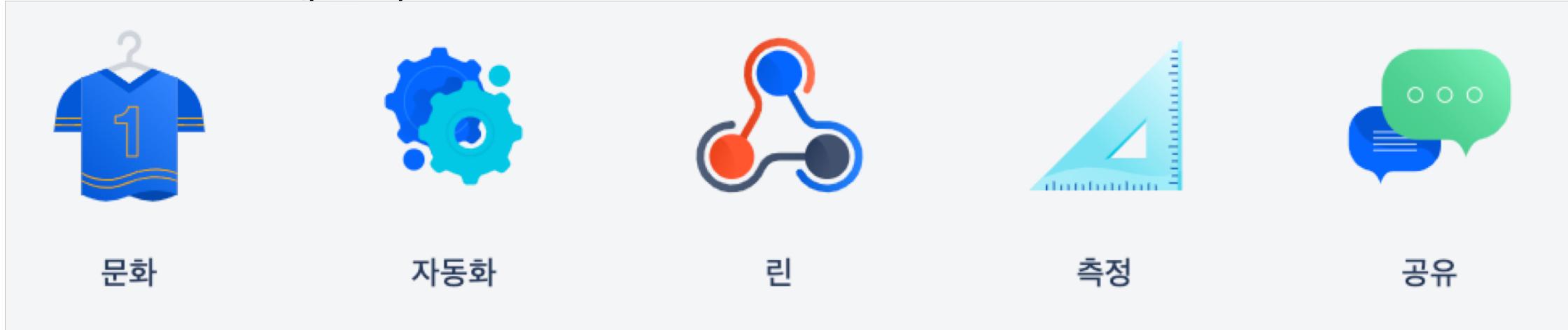


Cloud Native

- DevOps
 - 2007~2008년 → IT 운영 및 SW 개발에 문제점 대두
 - 개발과 배포가 다른 조직에서 관리 (다른 목표를 가짐)
 - 협업 및 신뢰
 - 더 빠른 릴리스, 더 스마트한 업무 진행
 - 문제 해결 시간 단축
 - 계획되지 않은 업무 쉽게 관리

Cloud Native

- DevOps
 - CALMS 프레임워크

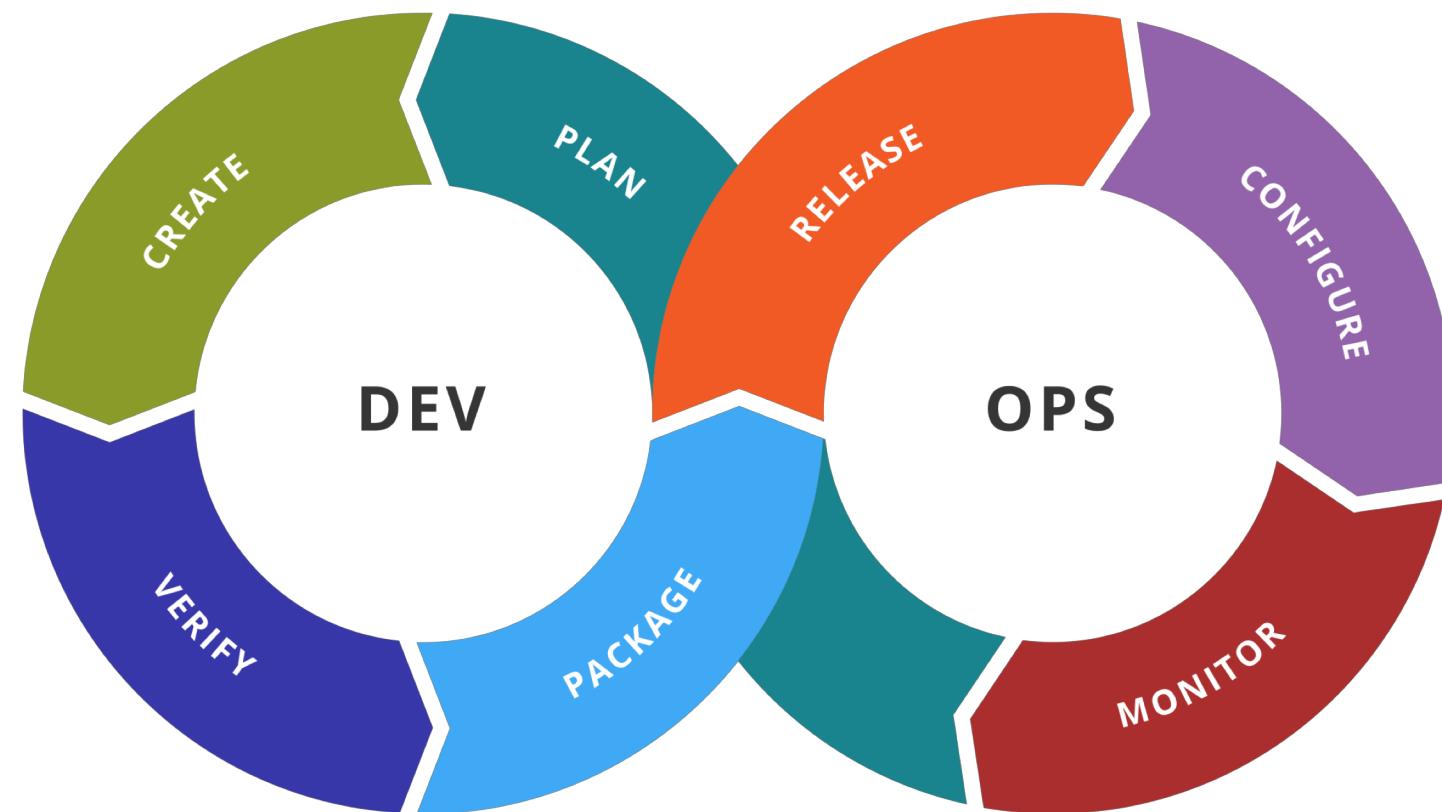


“DevOps를 활용하는 팀은 30배 더 자주 배포할 수 있고, 실패는 60배 줄일 수 있으며, 160배 빠르게 복구할 수 있습니다.”

Puppet Labs 2016 DevOps 상태 보고서

Cloud Native

- DevOps
 - 팀의 구성
 - 애플리케이션과 서비스의 개발에서 배포 운영까지 빠르게 제공하기 위한 조직의 협업 문화
 - 하나의 파이프라인으로 소스 코드의 배포가 필요할 때 즉시 반영
 - **서비스의 기획, 설계, 개발, 배포 및 운영 → 한 팀에서 수행**



Cloud Native

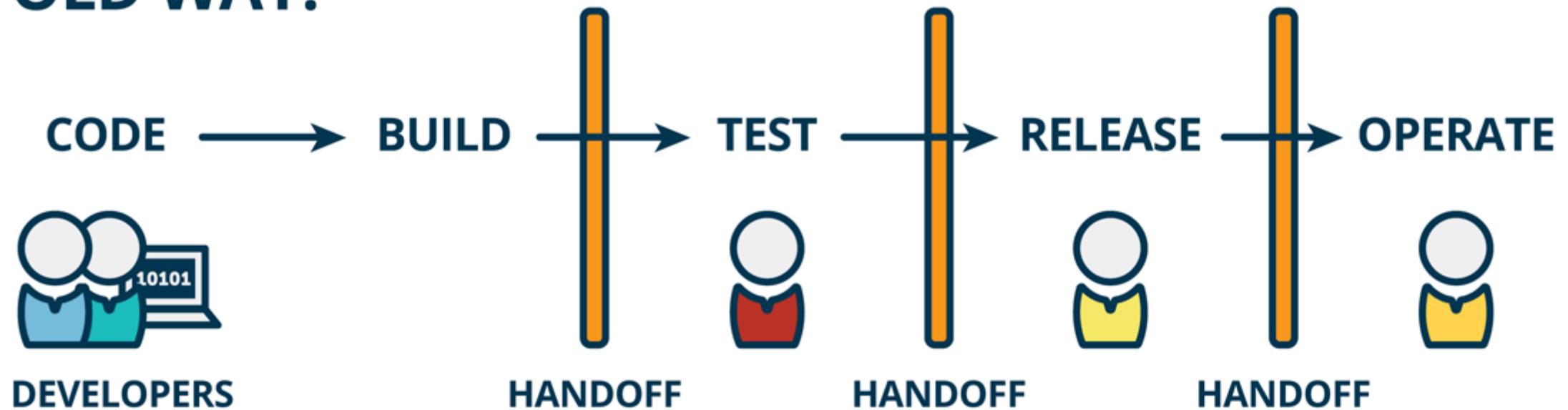
- DevOps
 - Microservice Team Structure
 - Two Pizza team
 - Teams communicating through API contracts
 - Develop, test and deploy each service independently
 - Consumer Driven Contract



Cloud Native

- DevOps

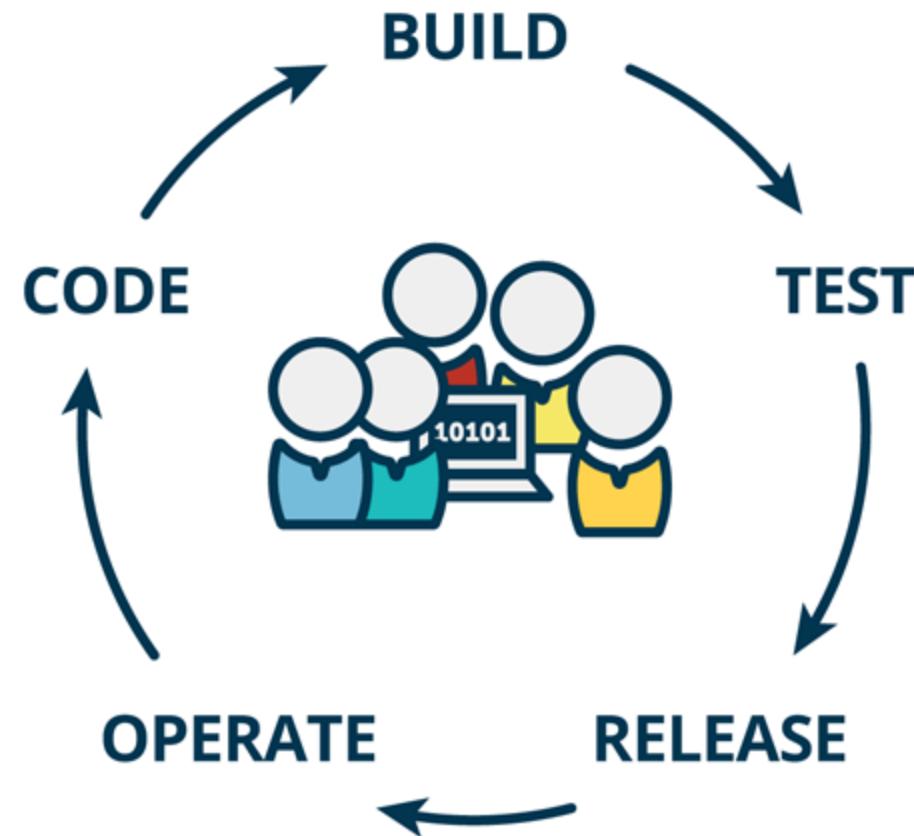
OLD WAY:



Cloud Native

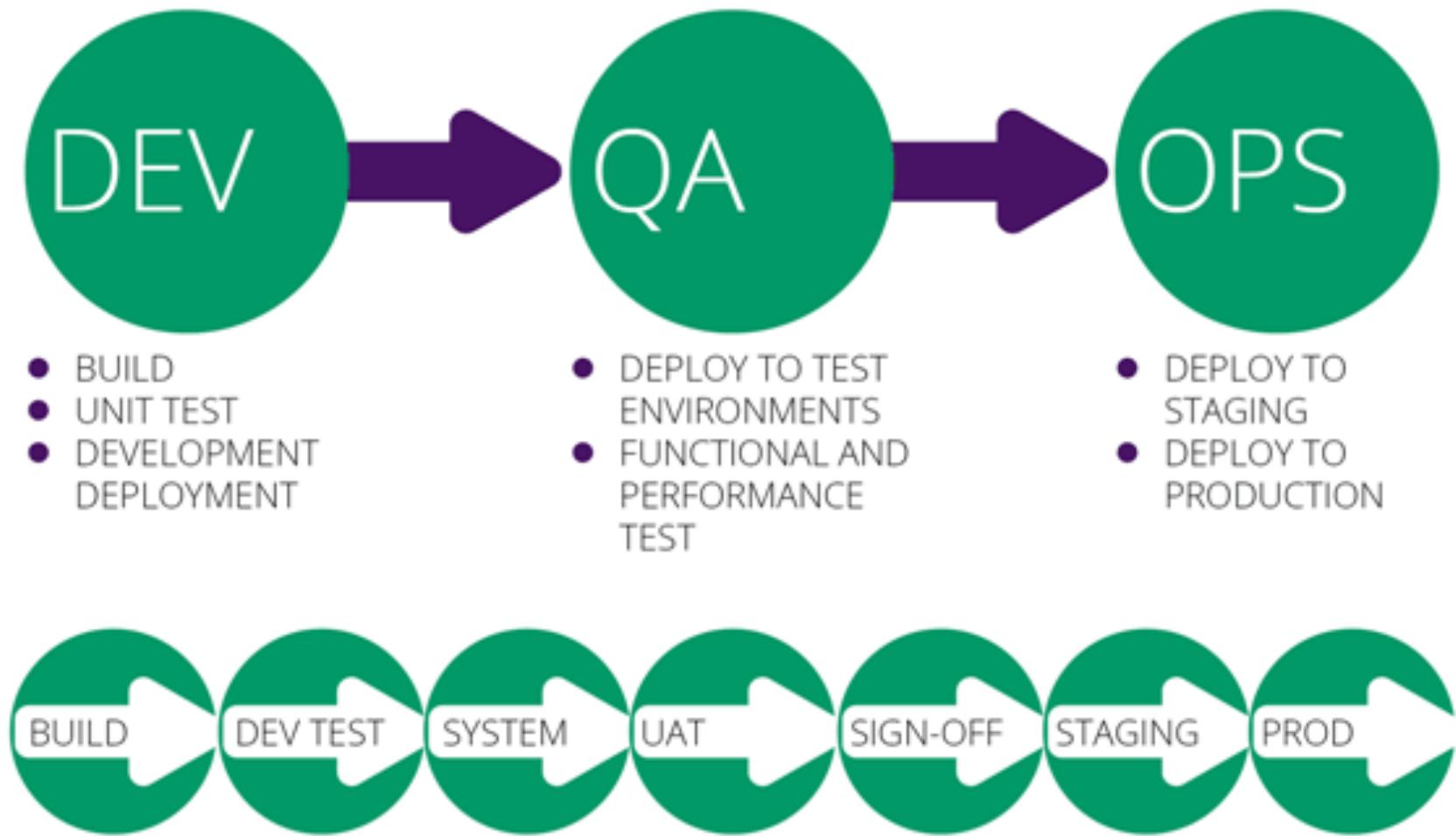
- DevOps

NEW WAY:



Cloud Native

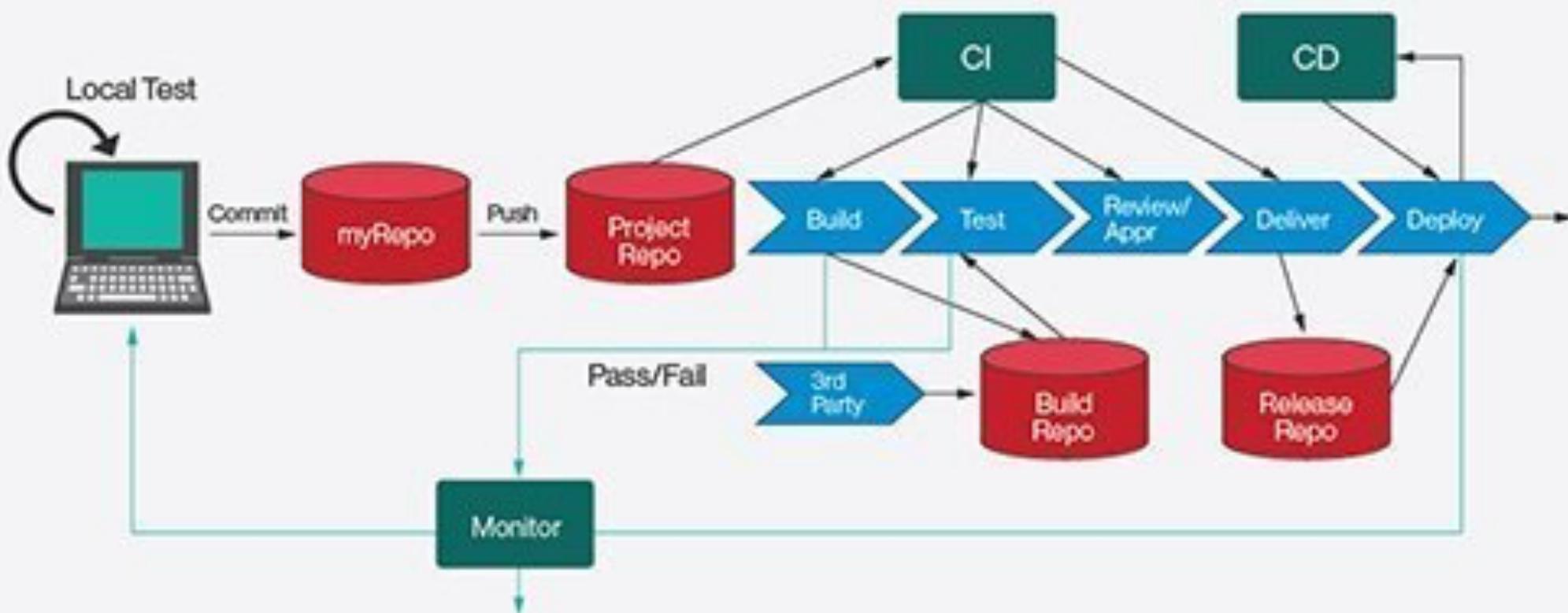
- DevOps



Cloud Native

- DevOps
 - 자동화와 시각화

Desirable Enterprise DevOps Process



Cloud Native

- Monitoring
 - ***Health check API*** - expose an endpoint that returns the health of the service
 - ***Log aggregation*** - log service activity and write logs into a centralized logging server, which provides searching and alerting
 - ***Distributed tracing*** - assign each external request a unique id and trace requests as they flow between services
 - ***Exception tracking*** - report exceptions to an exception tracking service, which deduplicates exceptions, alerts developers, and tracks the resolution of each exception
 - ***Application metrics*** - service maintain metrics, such as counters and gauges, and expose them to a metrics server
 - ***Audit logging*** - log user actions