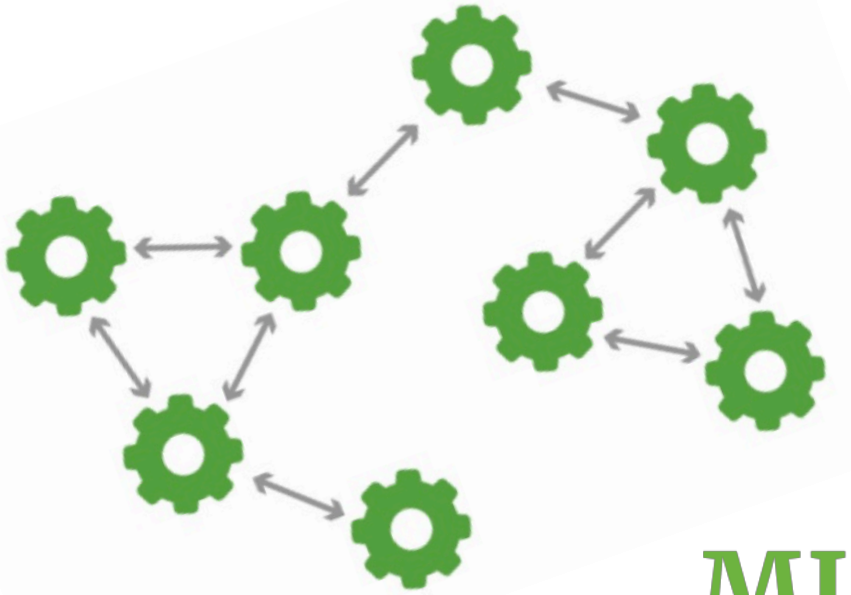
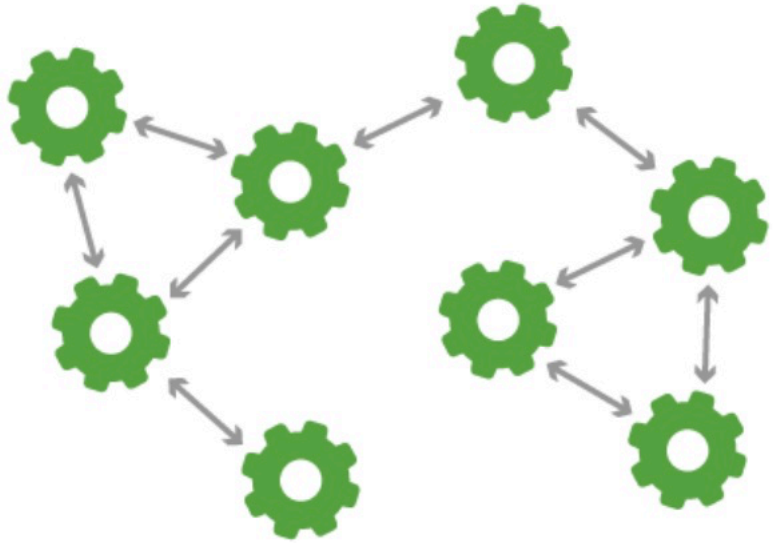




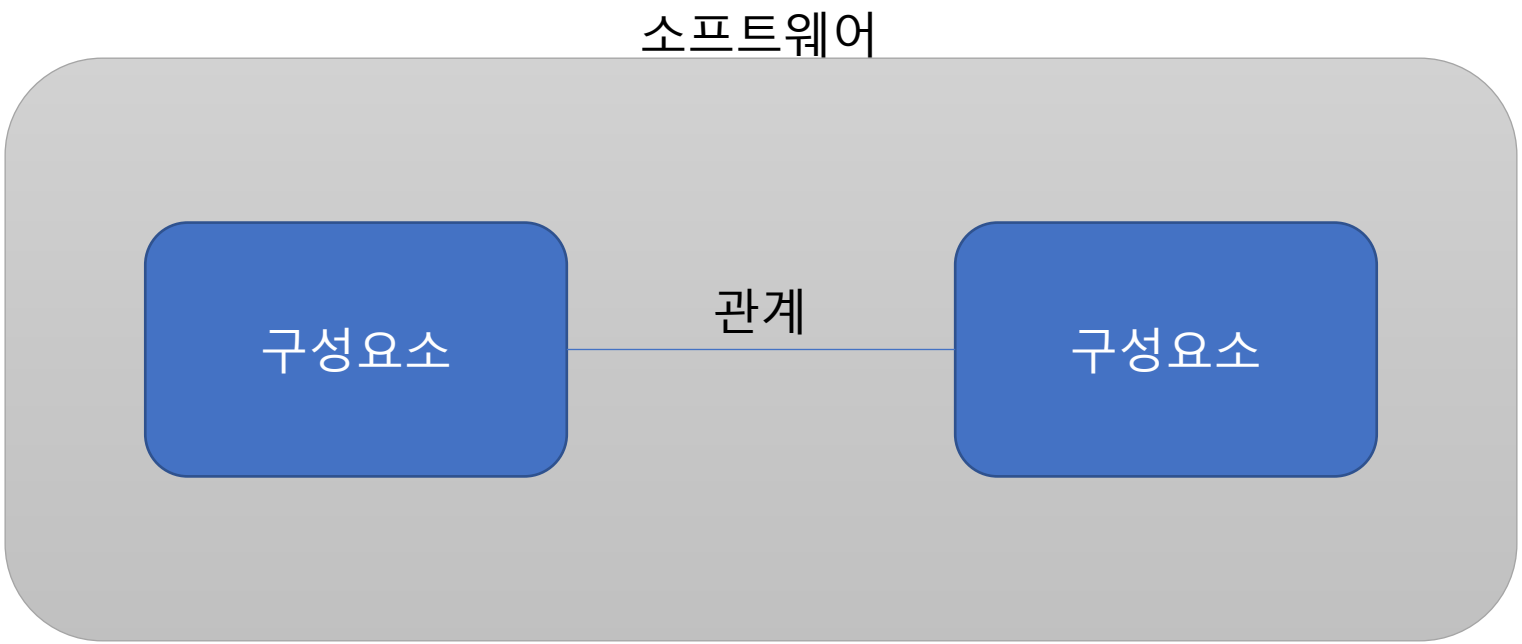
MICROSERVICES
with Spring Boot

목차

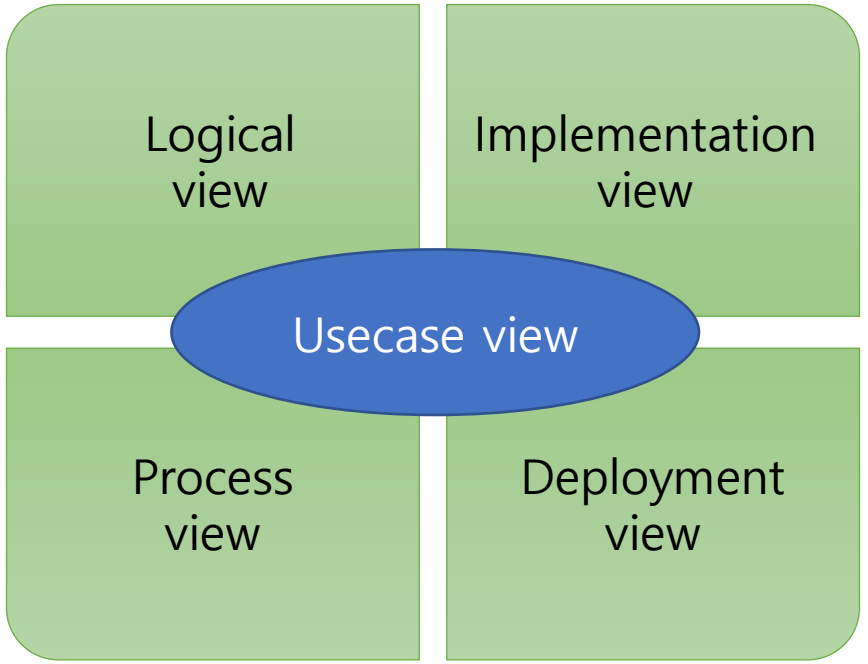
Software Architecture

Software Architecture

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



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

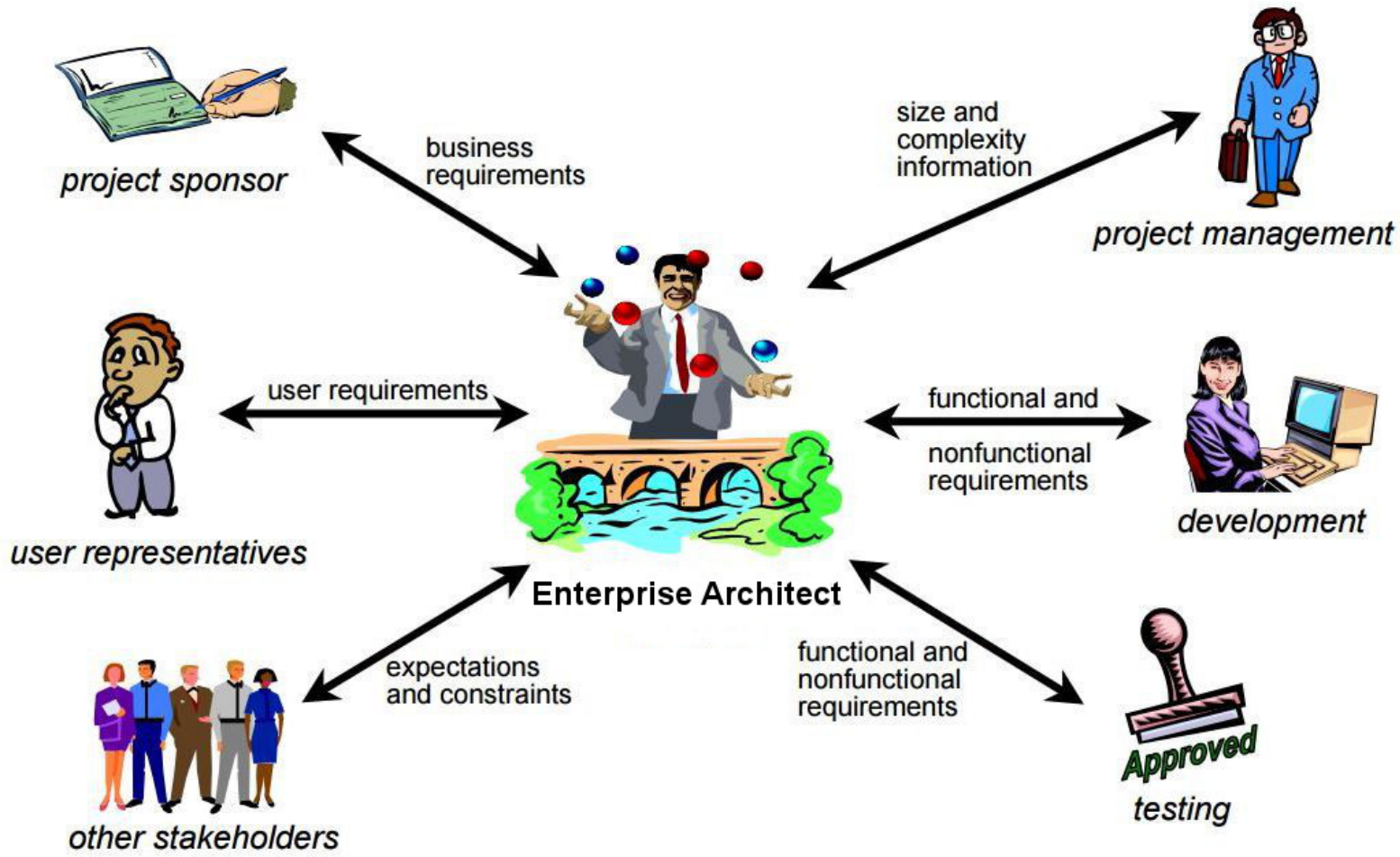


- Architecture의 역할

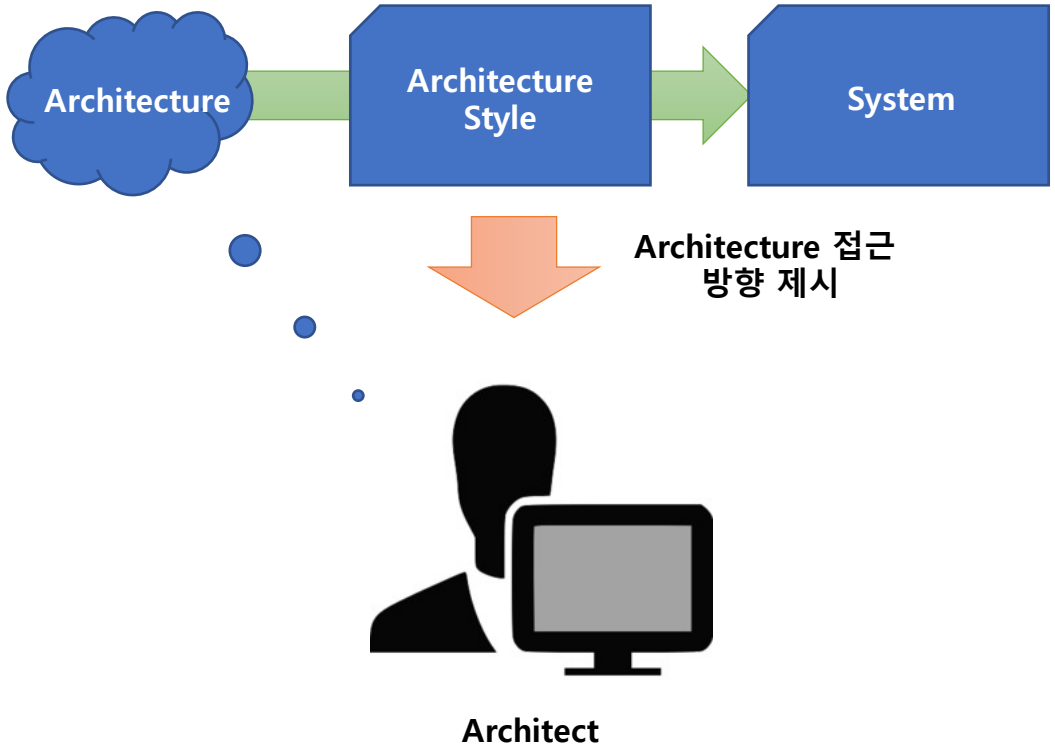
"Architecture is both the ***process*** and the product of ***planning***, ***designing***, and ***constructing*** buildings or any other structures. Architectural works, in the material form of buildings, are often perceived as cultural symbols and as works of art. ***Historical civilizations*** are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

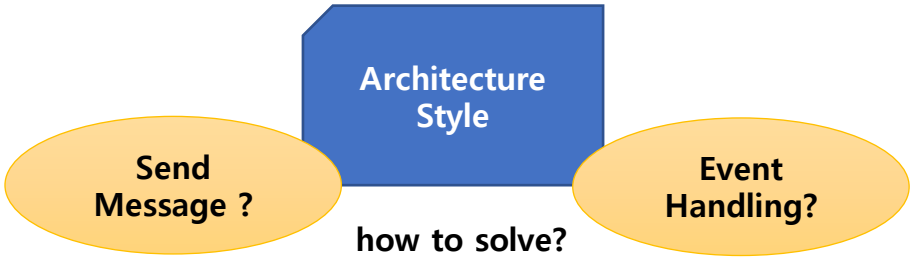
Software Architecture

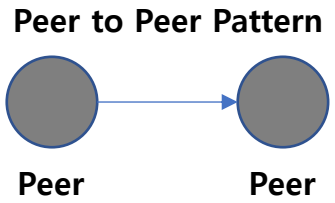
- Architecture의 역할
  - 무형의 지식 집합체(코드)
  - 기능적인 측면 보다는 안정적인 운영을 위한 전체적인 구조를 설계 ex) 오케스트라

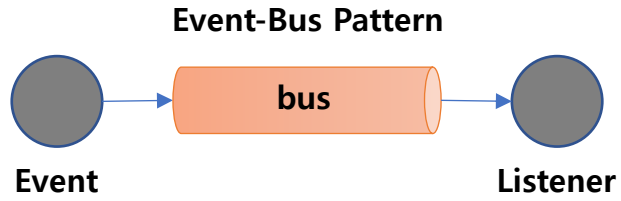


- Architecture 스타일과 패턴

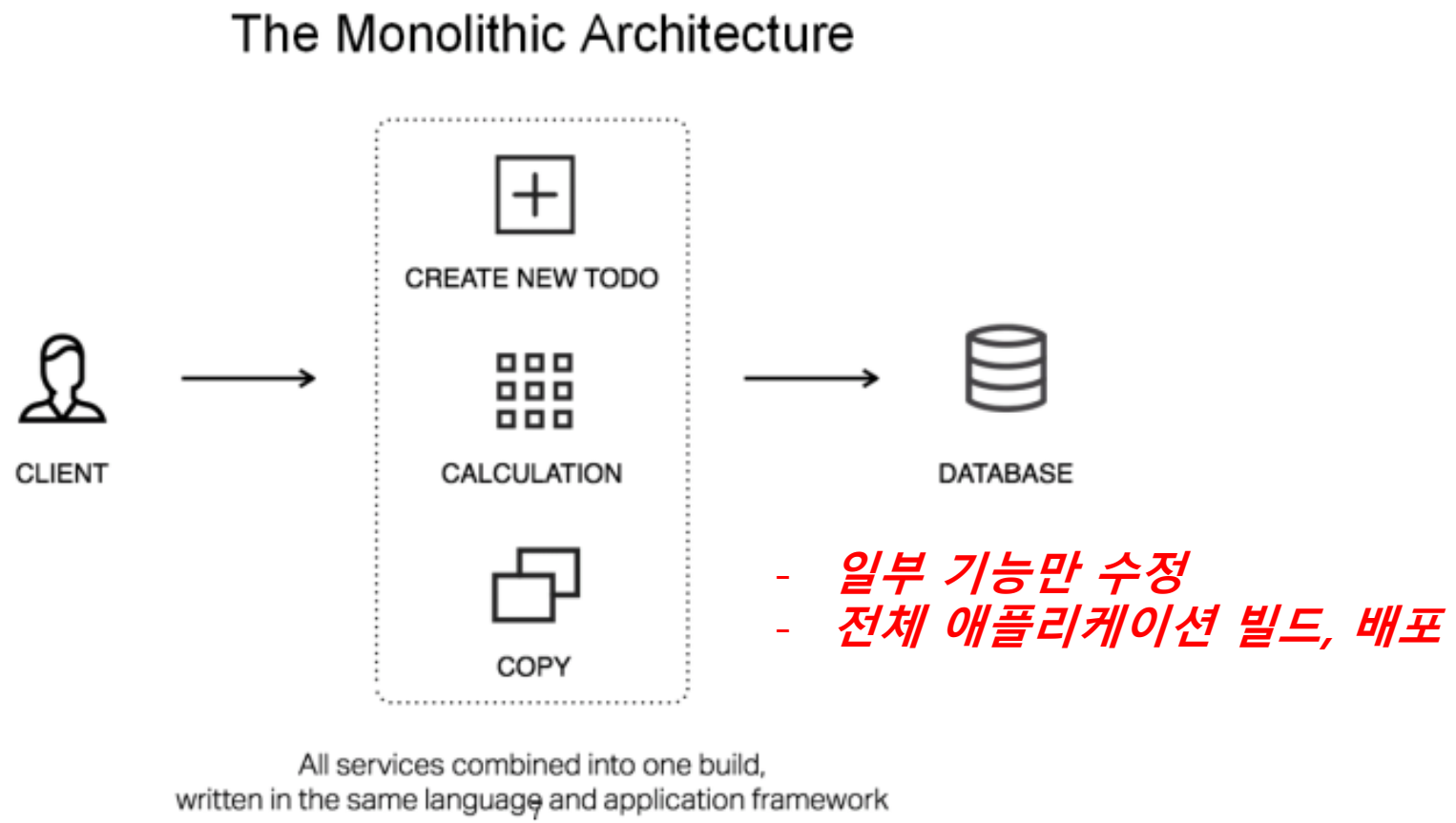








- Monolith Architecture
  - 모든 업무 로직이 하나의 애플리케이션 형태로 패키지 되어 서비스
  - 애플리케이션에서 사용하는 데이터가 한곳에 모여 참조되어 서비스되는 형태

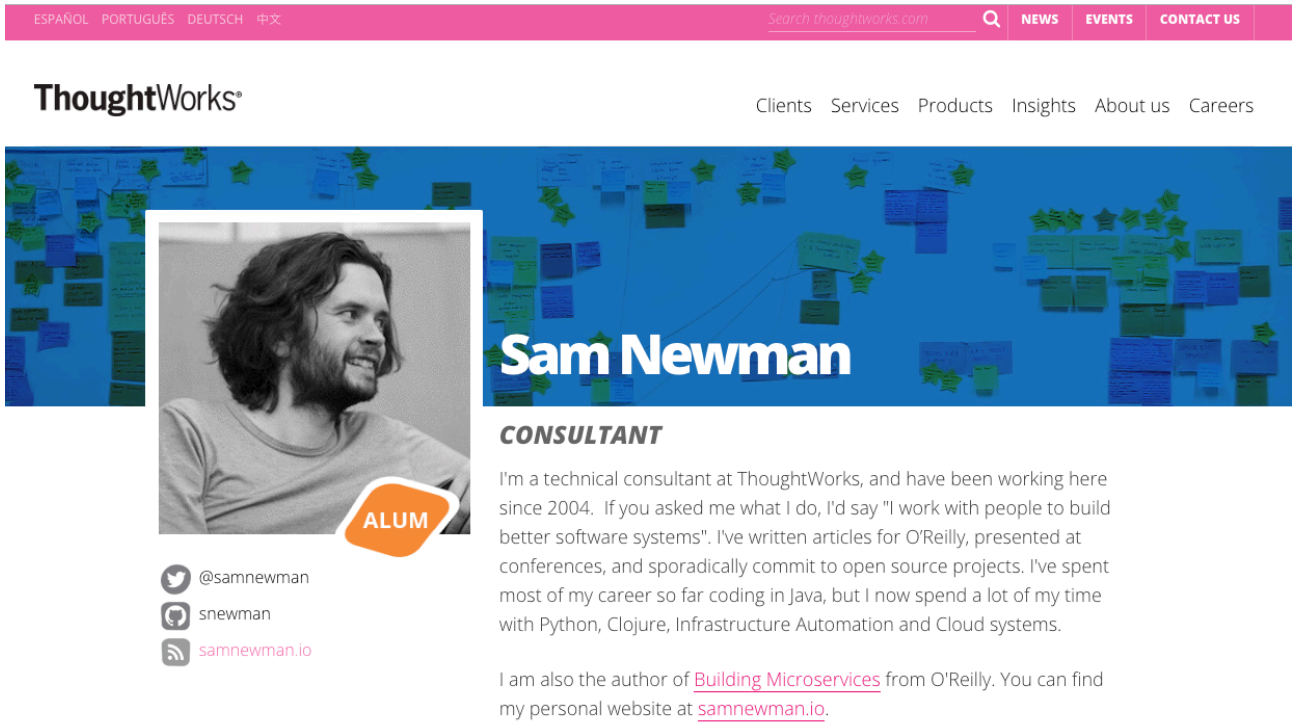


- *일부 기능만 수정*
- *전체 애플리케이션 빌드, 배포*

- What is the Microservice?

Small autonomous services that work together

- Sam Newman



- 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

- What is the Microservice?

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

- 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

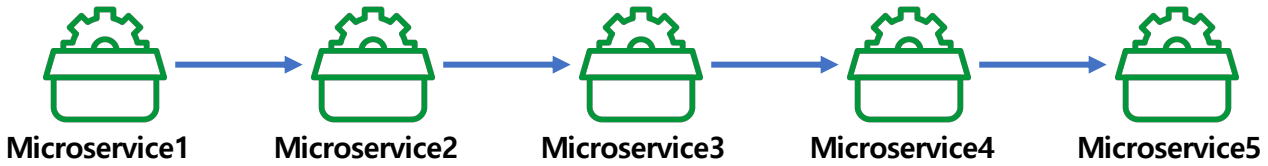
- James Lewis and Martin Fowler

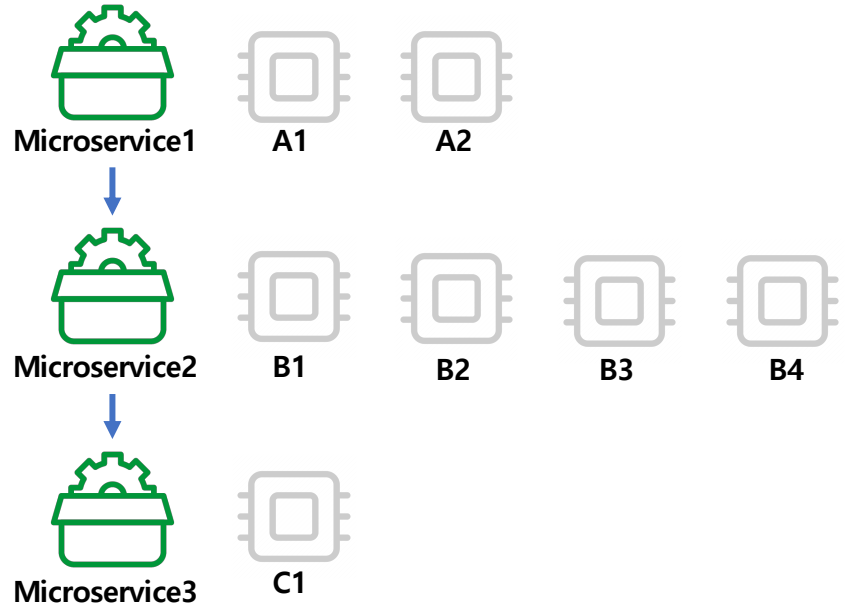
- What is the Microservice?

1) RESTful

2) Small Well Chosen Deployable Units

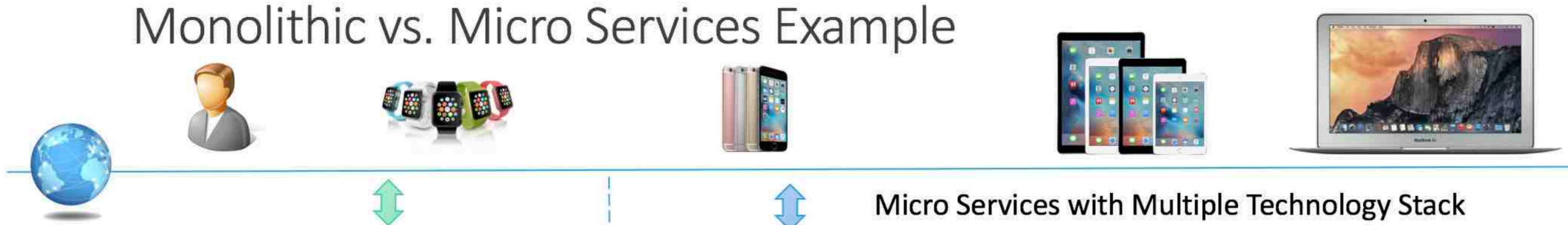
3) Cloud Enabled





- Monolith vs Microservice Architecture

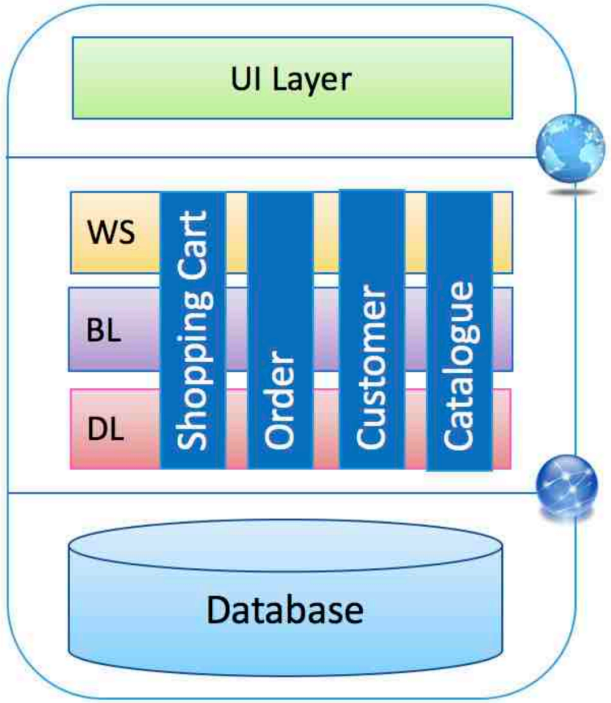
Monolithic vs. Micro Services Example



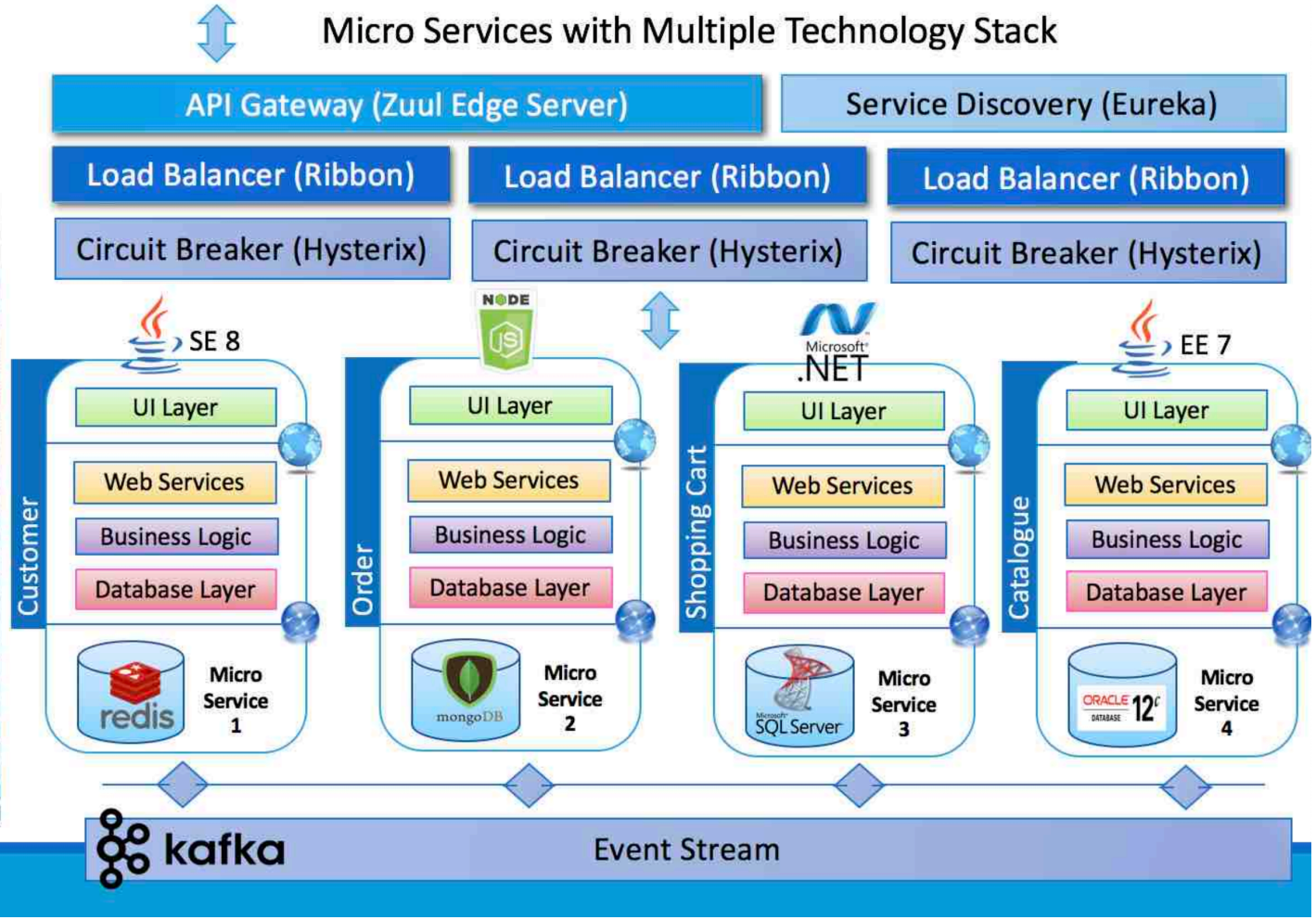
Existing aPaaS vendors creates Monolithic Apps.

This 3 tier model is obsolete now.

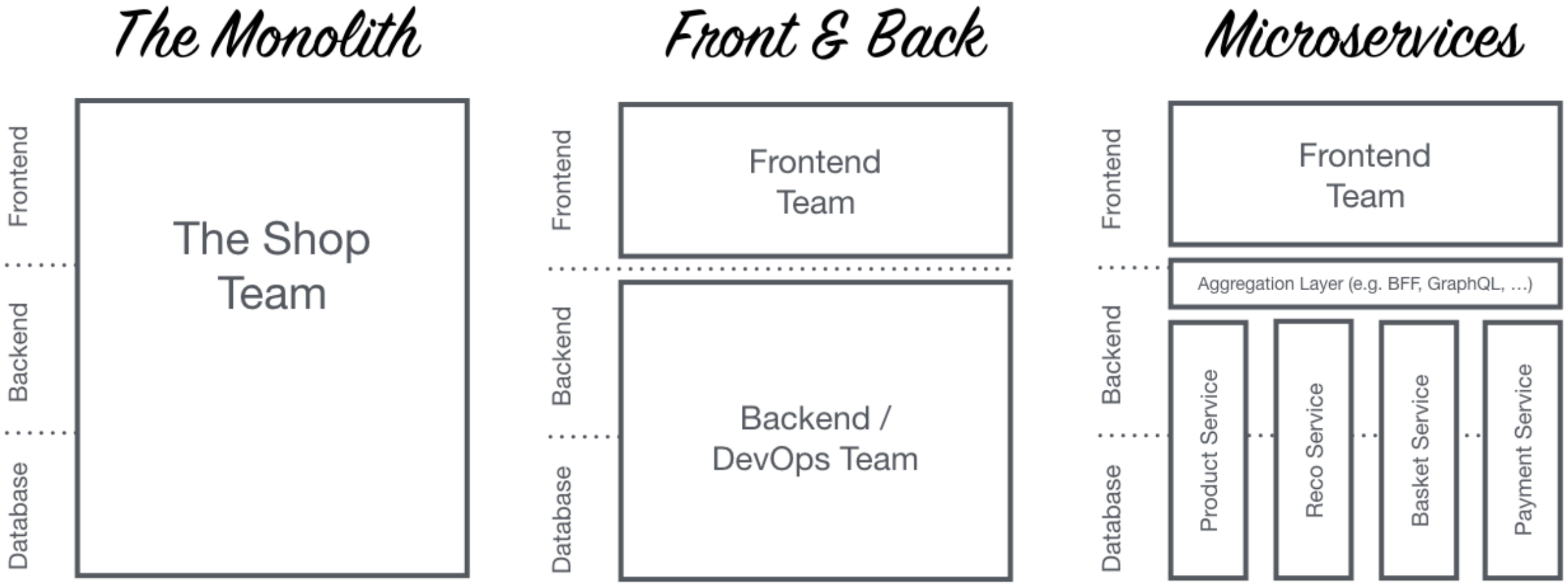
Source: Gartner Market Guide for Application Platforms Nov 23, 2016



Traditional Monolithic App using Single Technology Stack



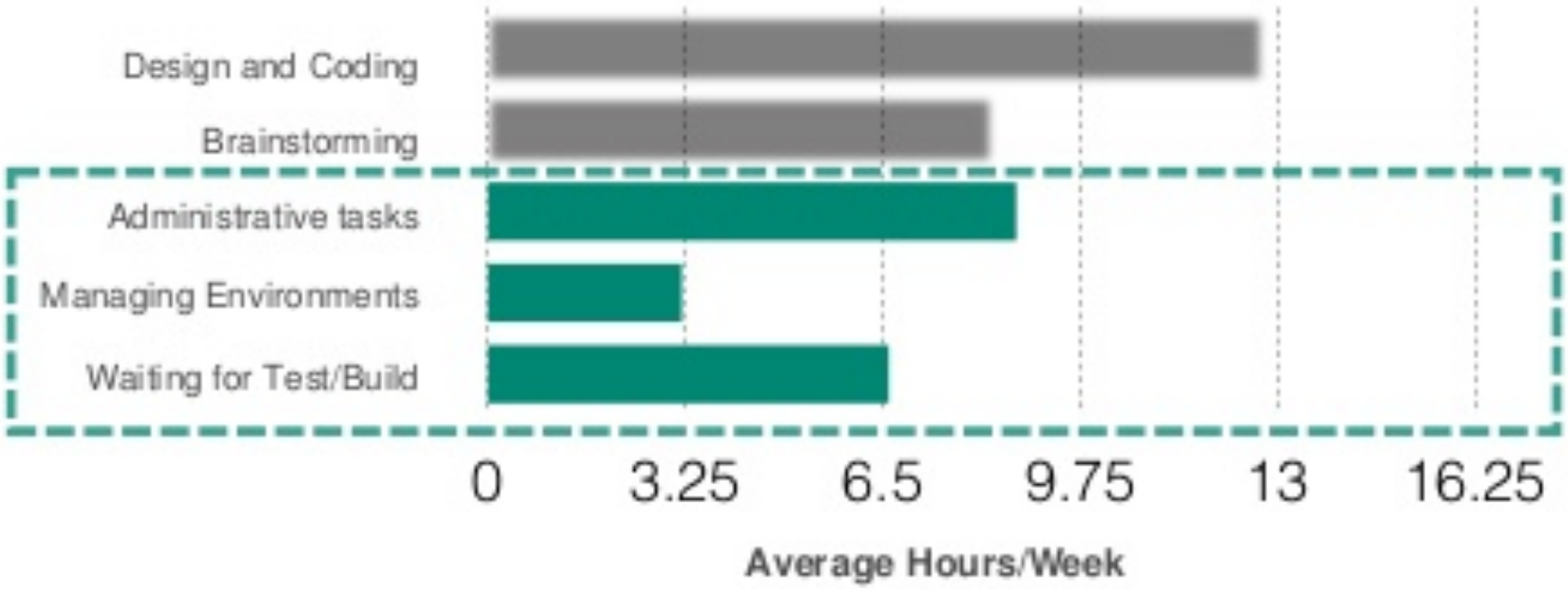
- Monolith vs Front & Back vs Microservice Architecture



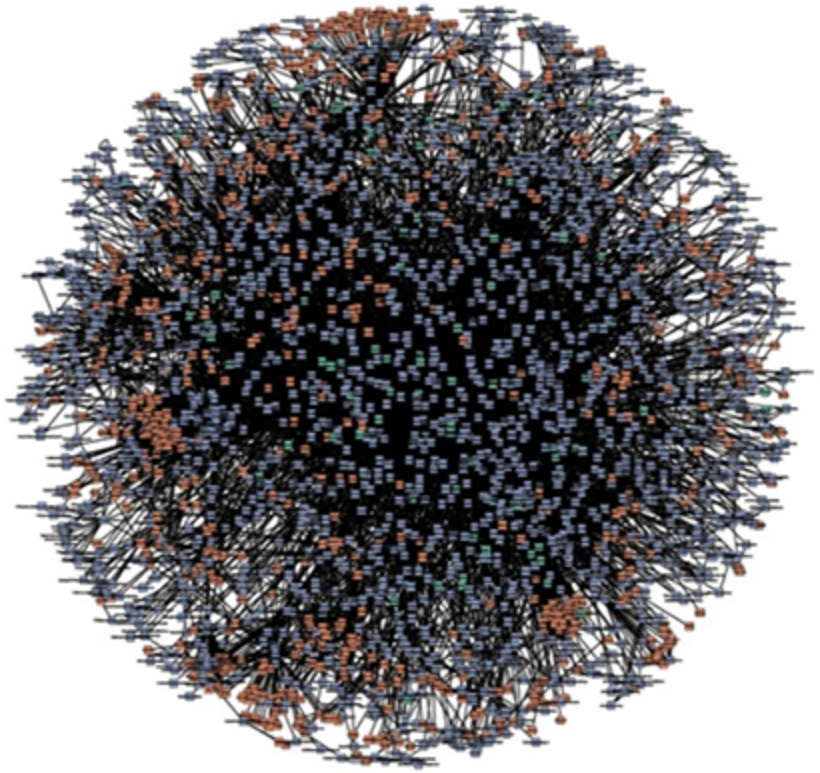
- How Regular is Regular for High Performing Cloud Native Enterprise?

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



Software Architecture



amazon.com

NETFLIX

- Microservice 특징

1) Challenges

2) Bounded Context

3) Configuration Management

4) Dynamic Scale Up And Scale Down

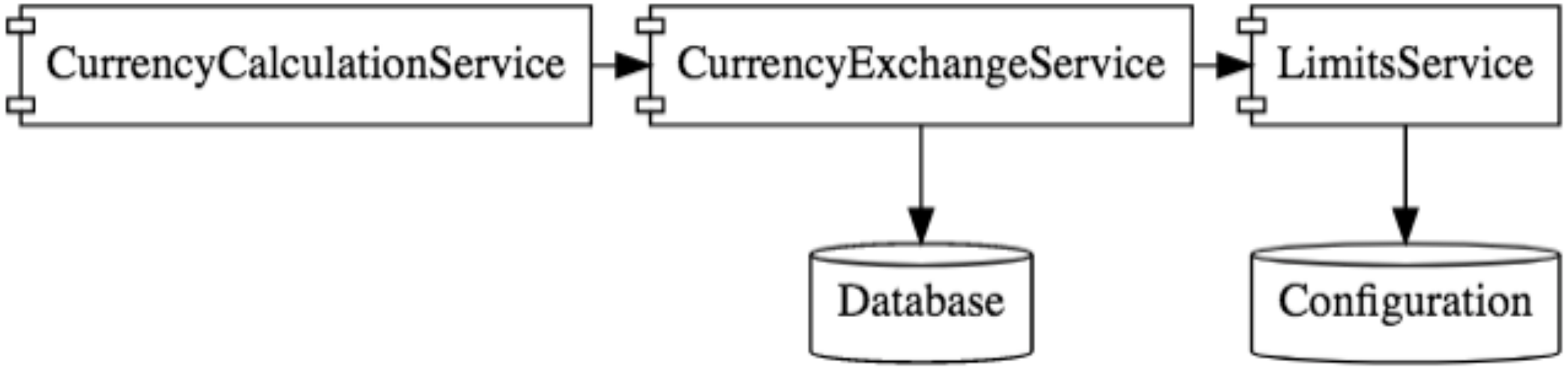
5) Visibility

- Microservice 특징

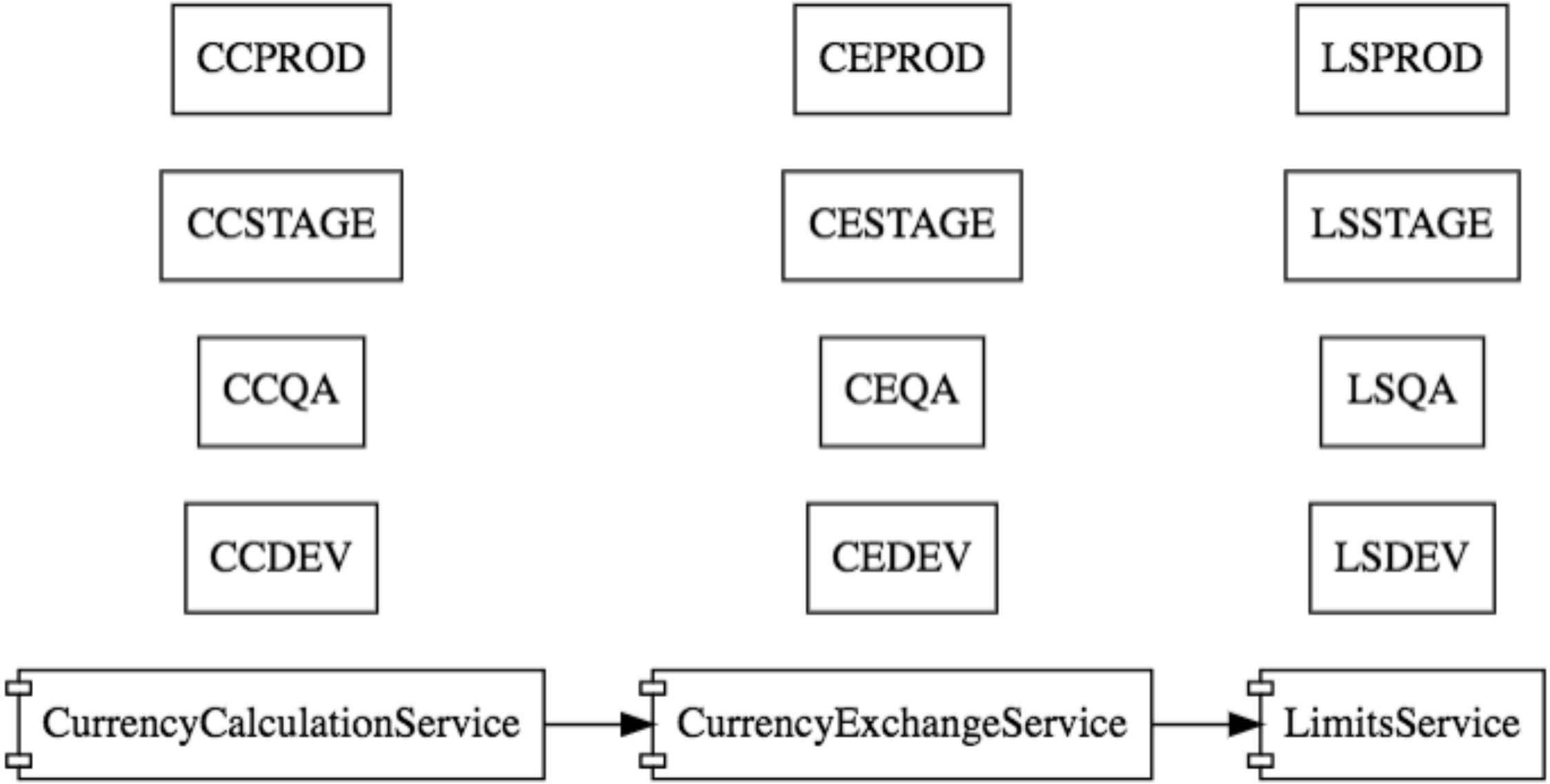
6) Solution

- 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

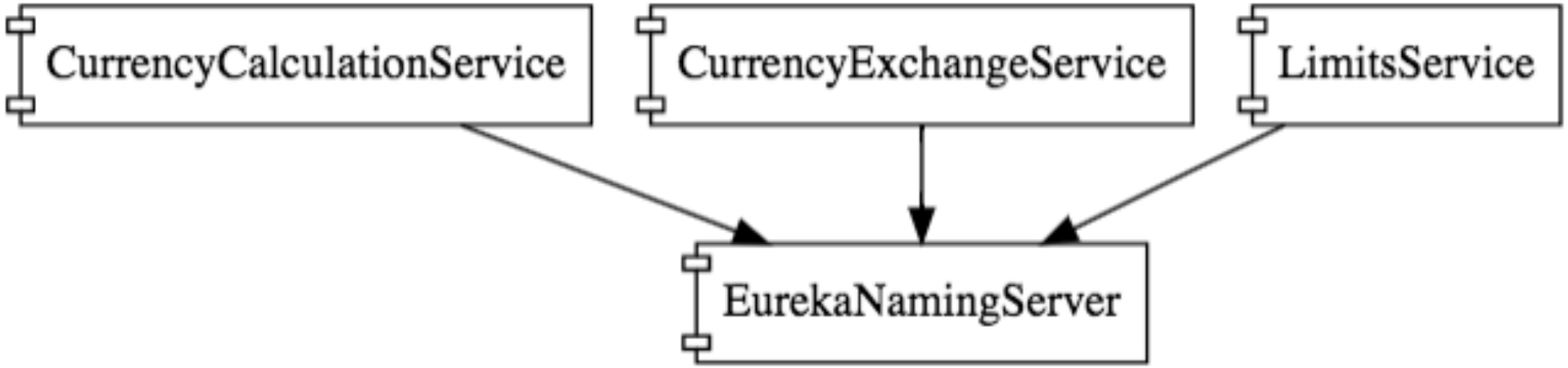
- Microservice



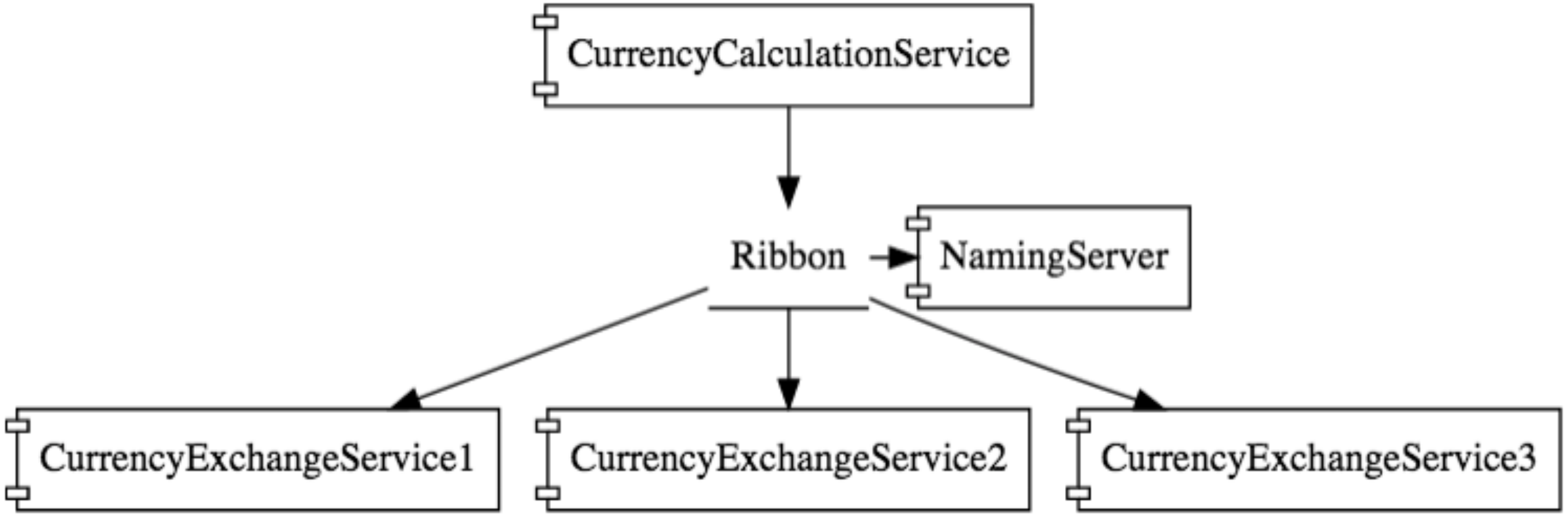
- Microservice Environments



- Microservice – Eureka Naming Server

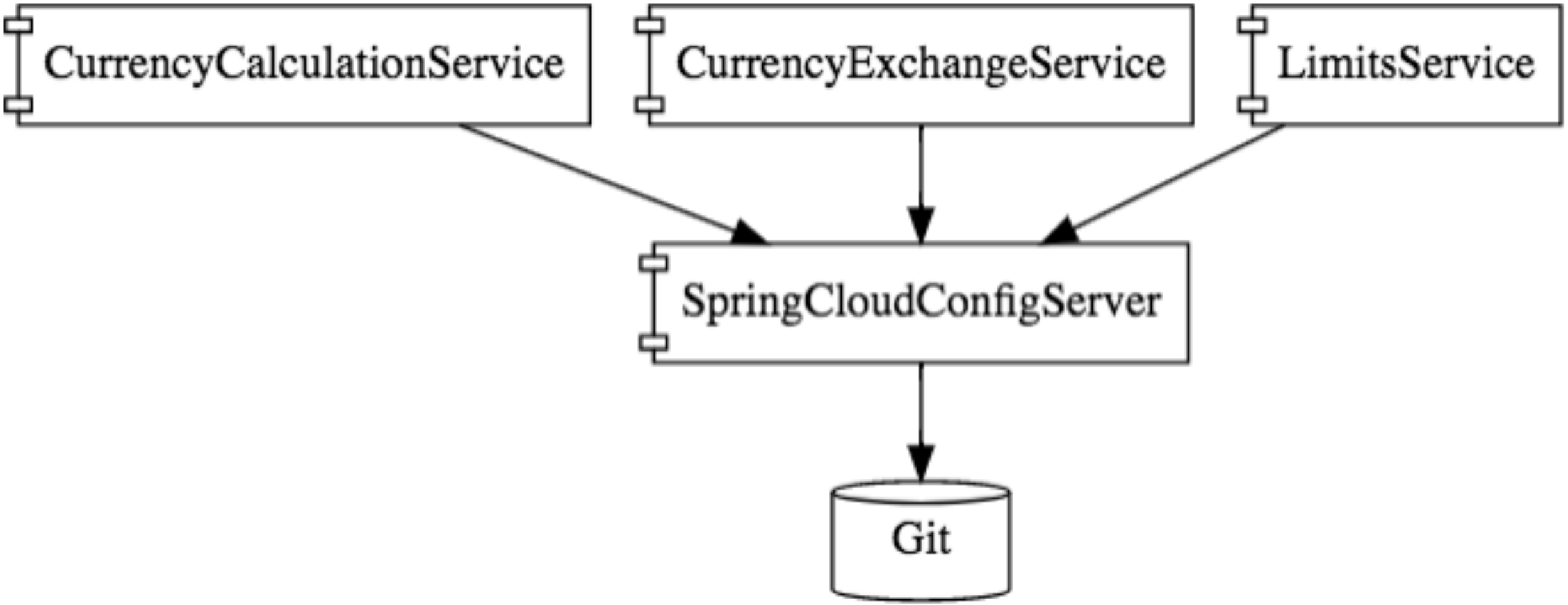


- Microservice – Ribbon Load Balancing



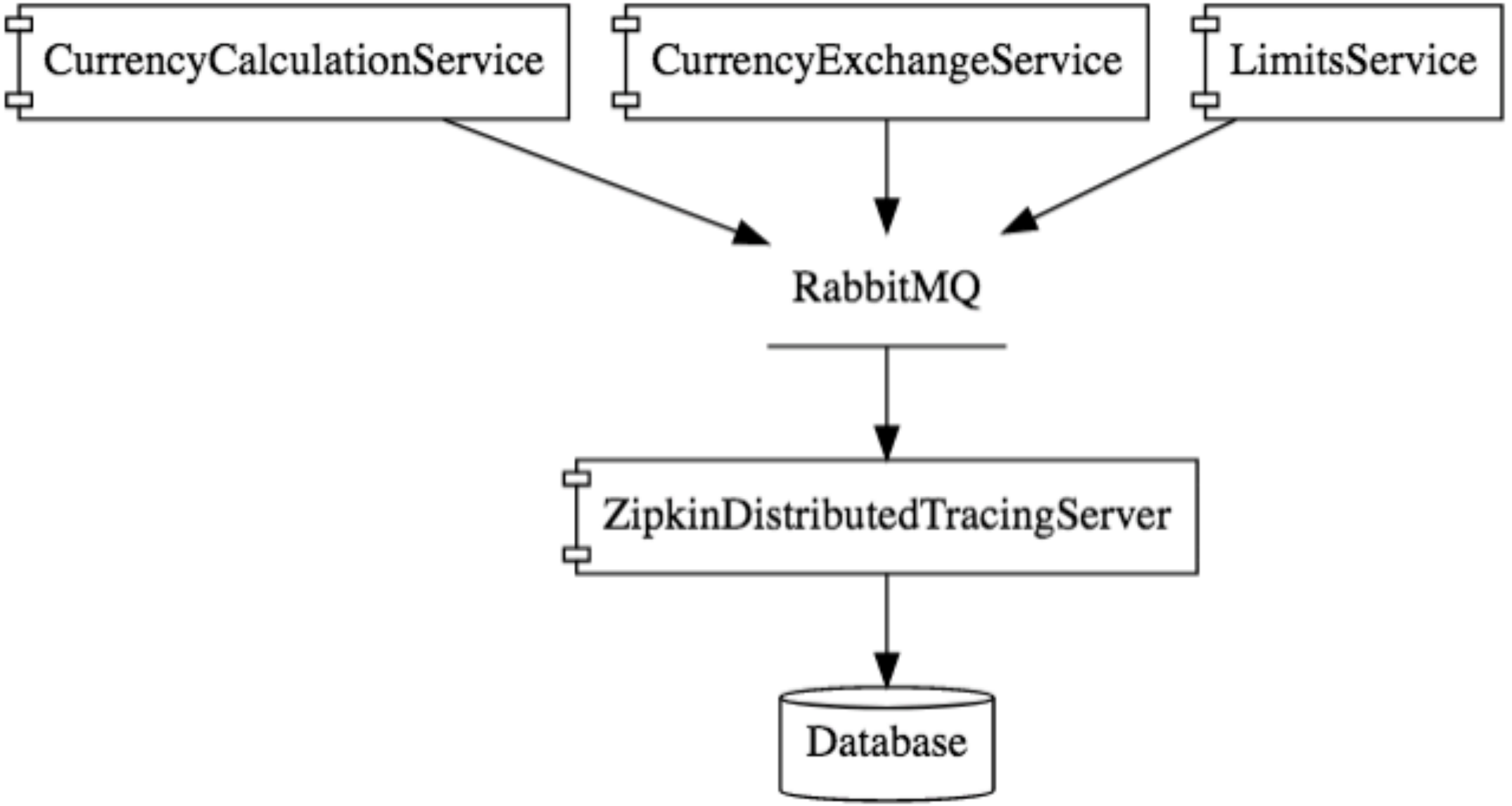
Software Architecture

- Microservice – Spring Cloud Config Server



Software Architecture

- Microservice – Zipkin Distributed Tracing



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

- Microservice – Web Services

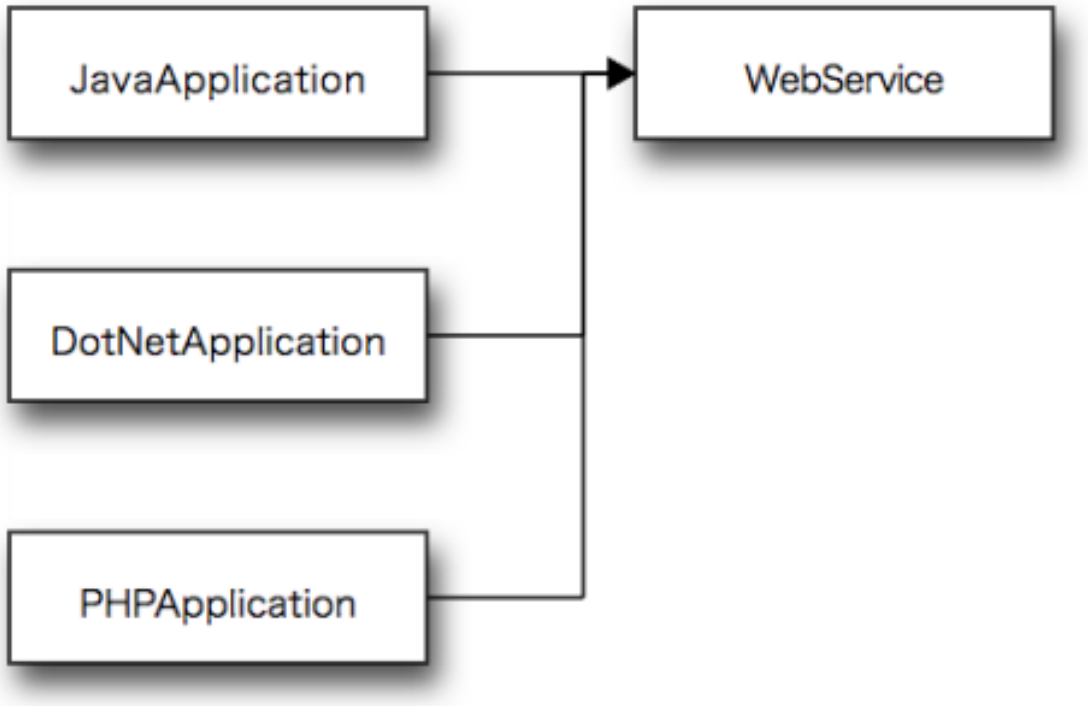
  - Service delivered over the web?
  - Is the Todo Management Application a Web Service?
  - Can I reuse the Business Layer by creating a JAR?
  - How can I make my Todo application consumable by other applications?

- 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

- Web Services

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



- Web Services
  - Data Format
    - XML

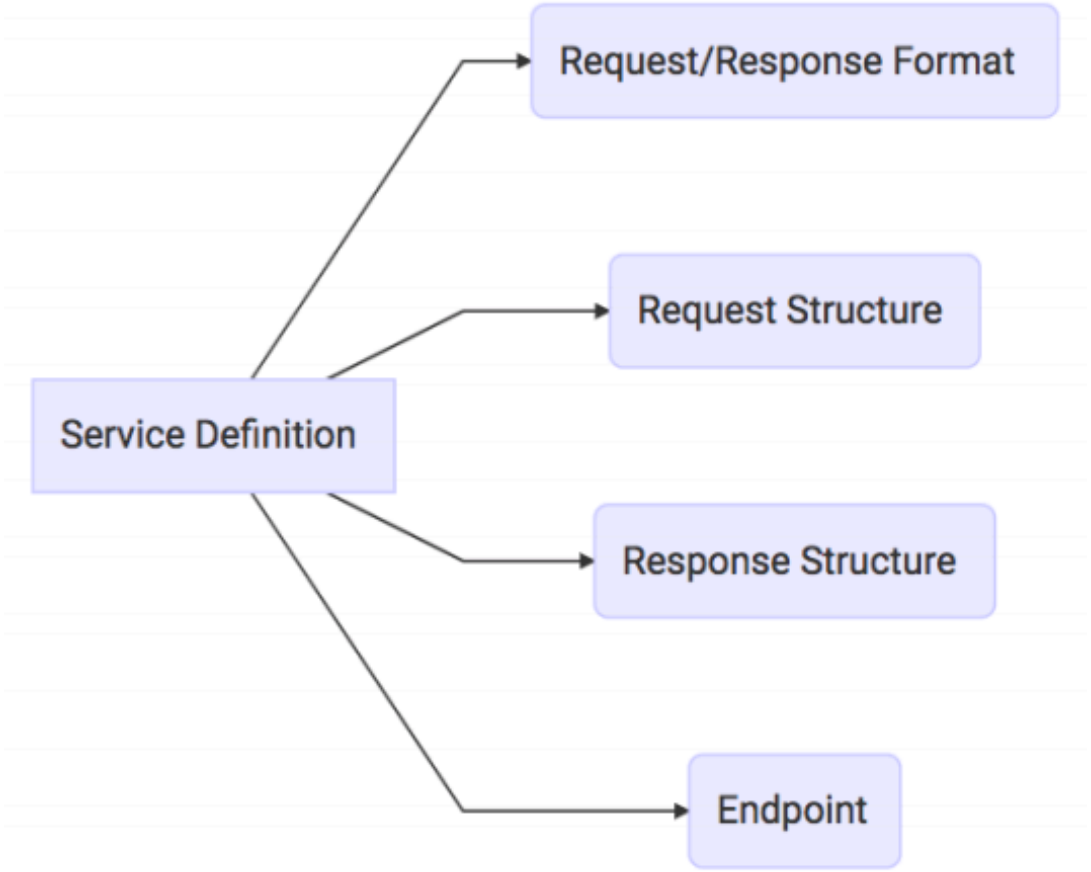
```
<getCourseDetailsRequest>
    <id>Course1</id>
</getCourseDetailsRequest>
```

    - 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"
  }
]
```

- 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)

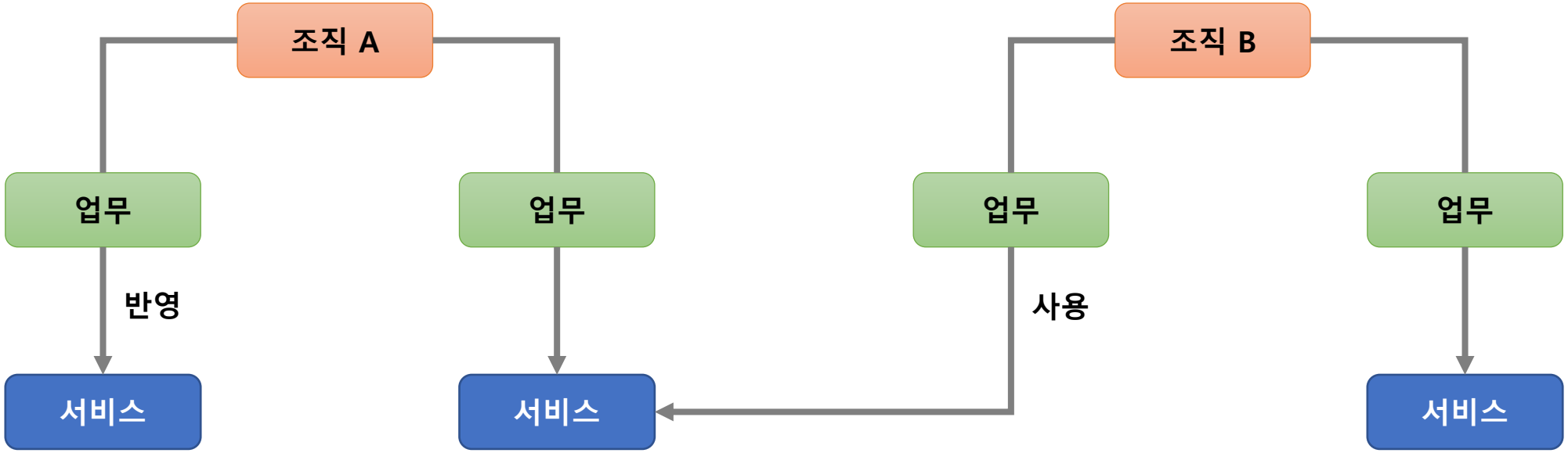


Service Oriented Architecture

Service Oriented Architecture

- SOA

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



- SOA 특징

1) 서비스 계약

2) 서비스 가용성

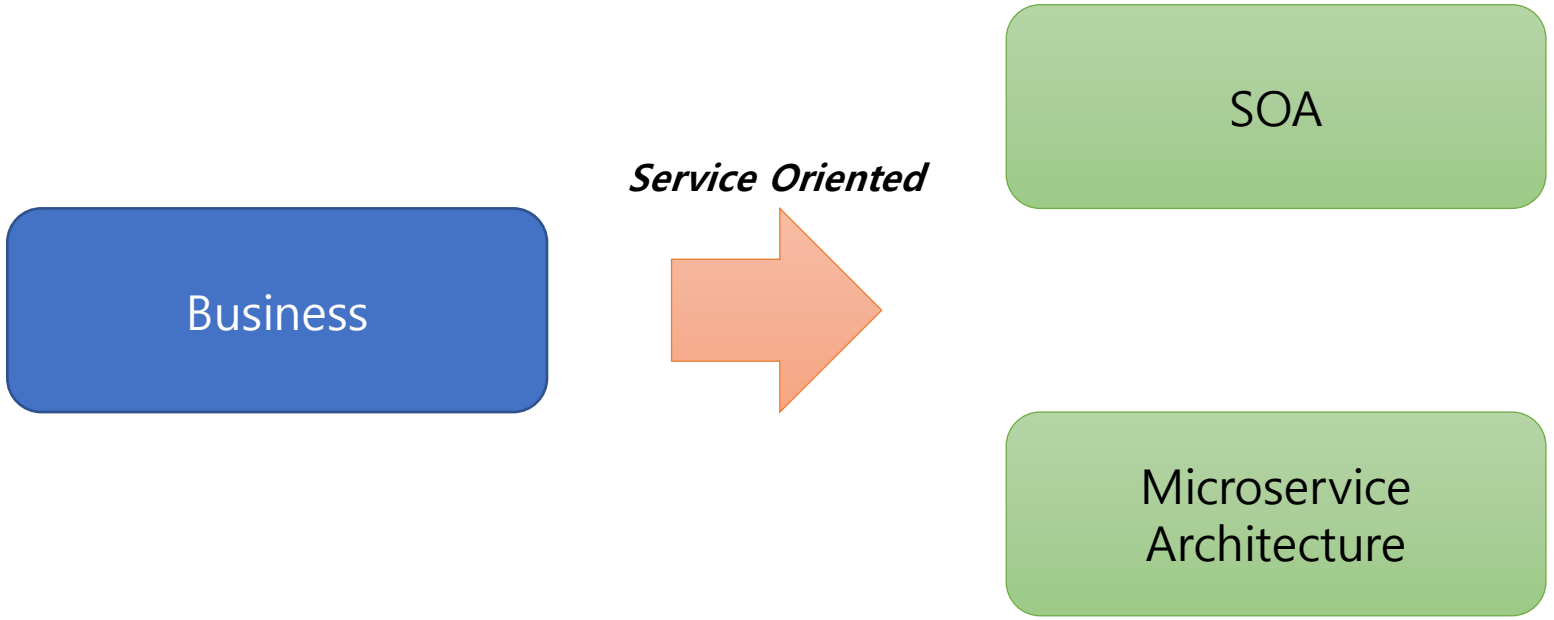
3) 서비스 권한

4) 트랜잭션

5) 서비스 관리

- 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

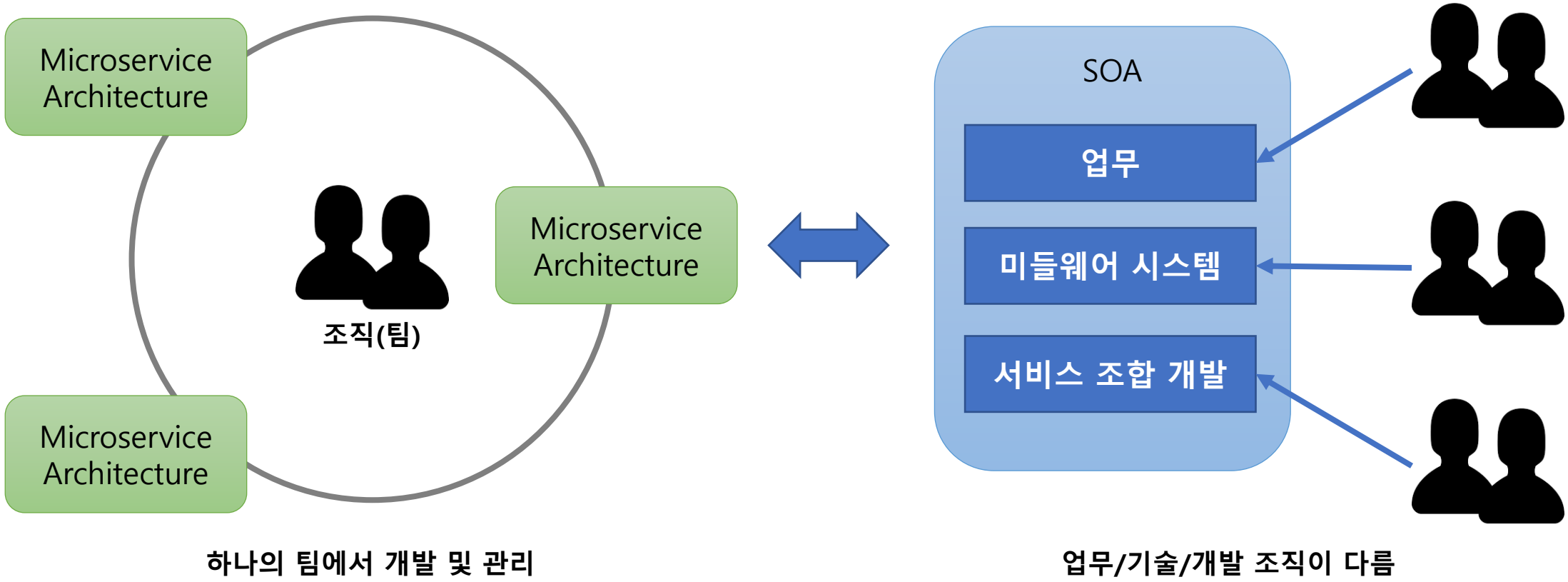
  - SOA
    - *비즈니스 측면에서의 서비스 재사용성*
    - *ESB(Enterprise Service Bus)라는 서비스 채널 이용 → 서비스 공유, 재사용*
  - MSA
    - *한 가지 작은 서비스에 집중*
    - *서비스 공유하지 않고 독립적 실행*

- SOA와 Microservice Architecture와의 차이점

  - 서비스의 상대적 크기와 관심사

- SOA와 Microservice Architecture와의 차이점

  - 서비스의 Ownership

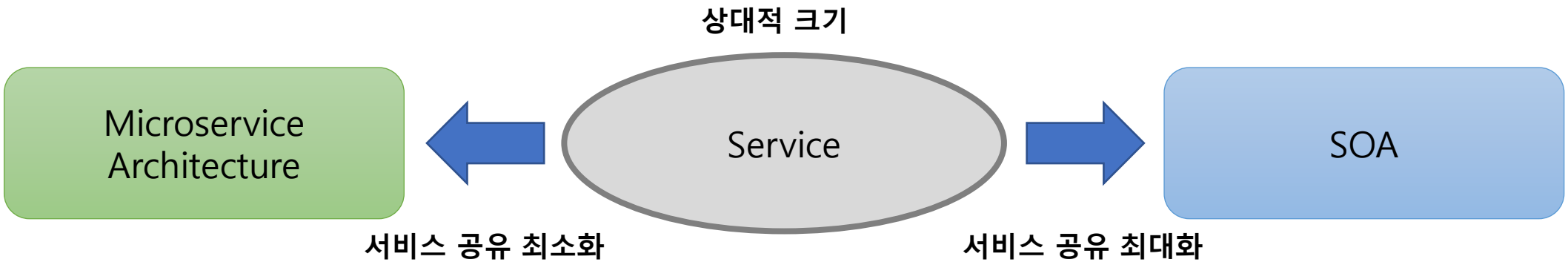


하나의 팀에서 개발 및 관리

업무/기술/개발 조직이 다름

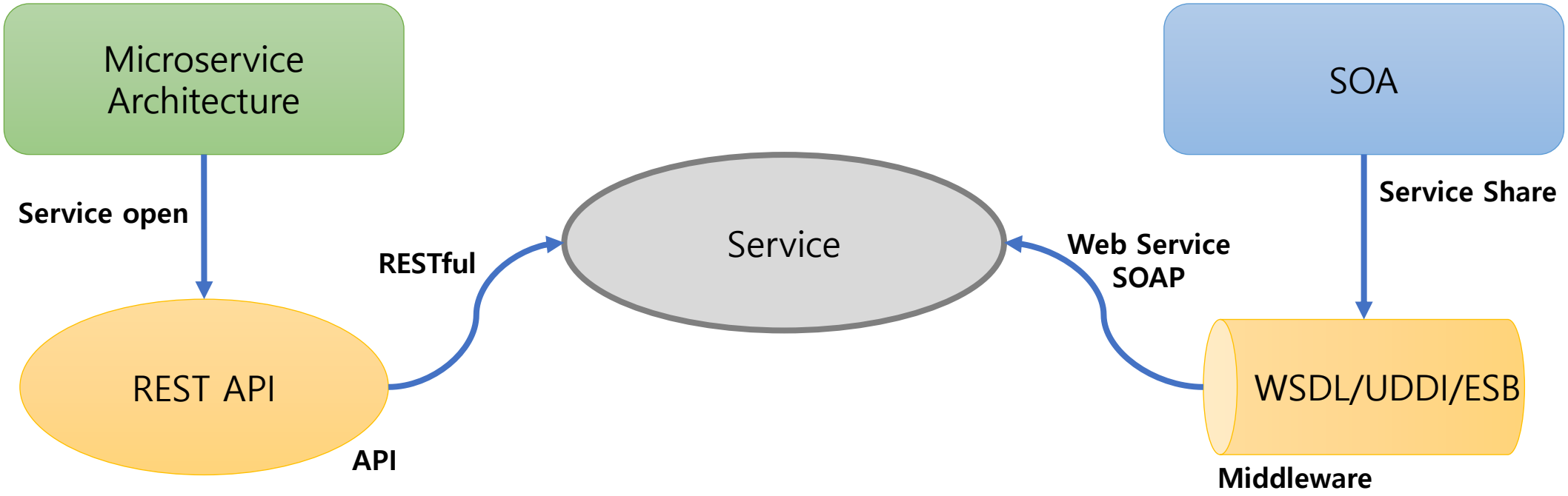
- SOA와 Microservice Architecture와의 차이점

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

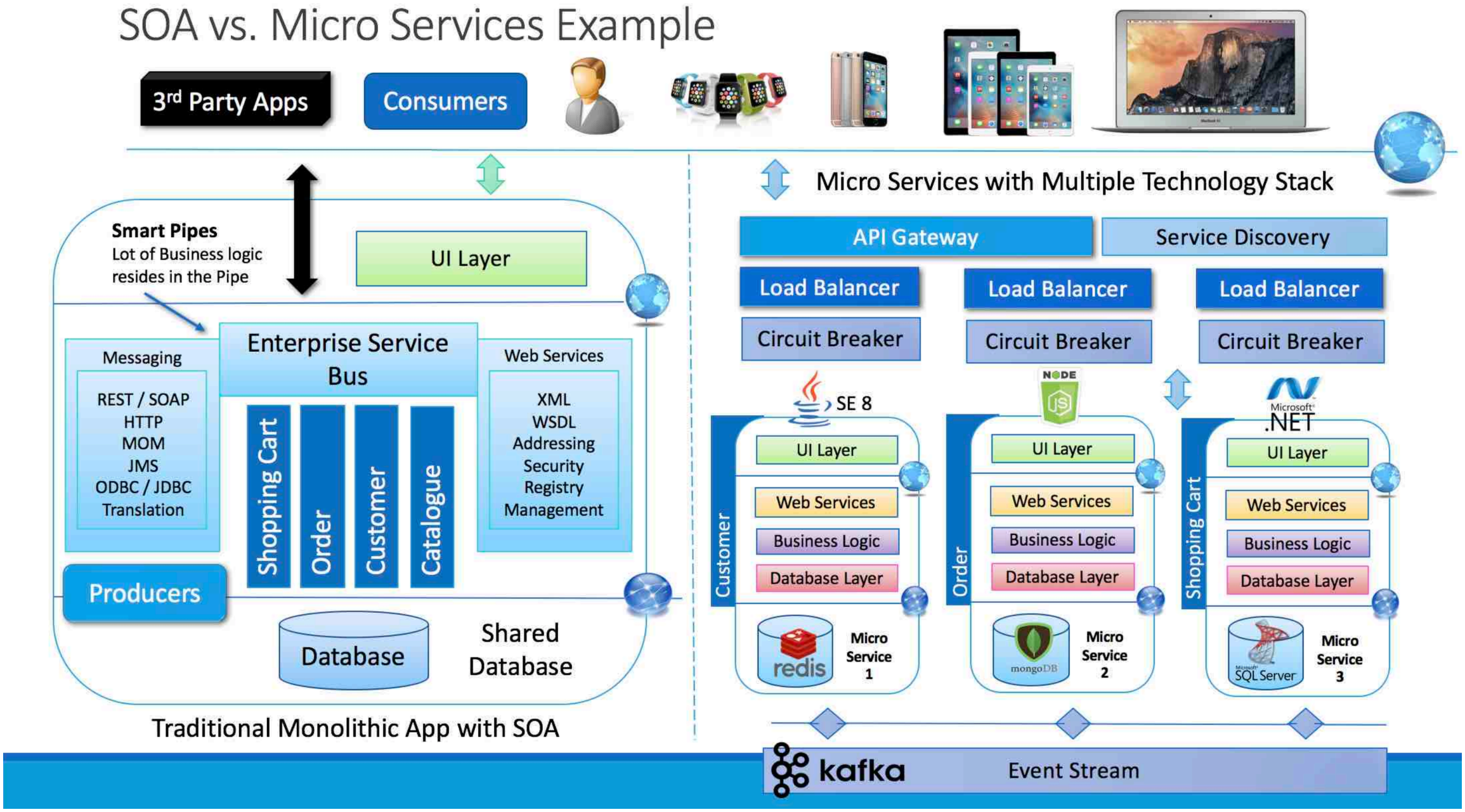


Service Oriented Architecture

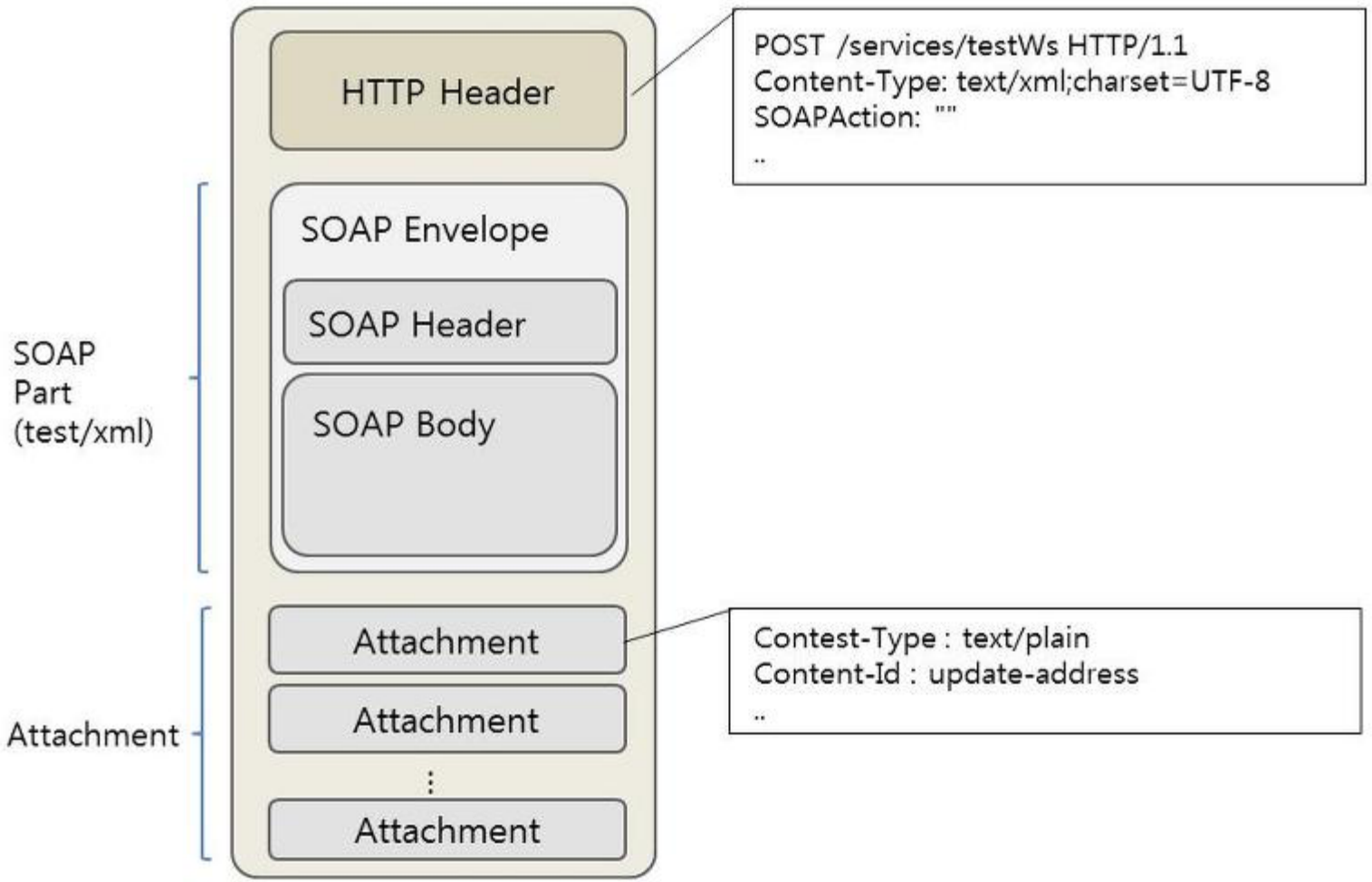
- SOA와 Microservice Architecture와의 차이점
  - 기술 방식
    - *SOA - 공통의 서비스를 ESB에 모아 사업 측면에서 공통 서비스 형식으로 서비스 제공*
    - *MSA - 각 독립된 서비스가 노출된 REST API를 사용*



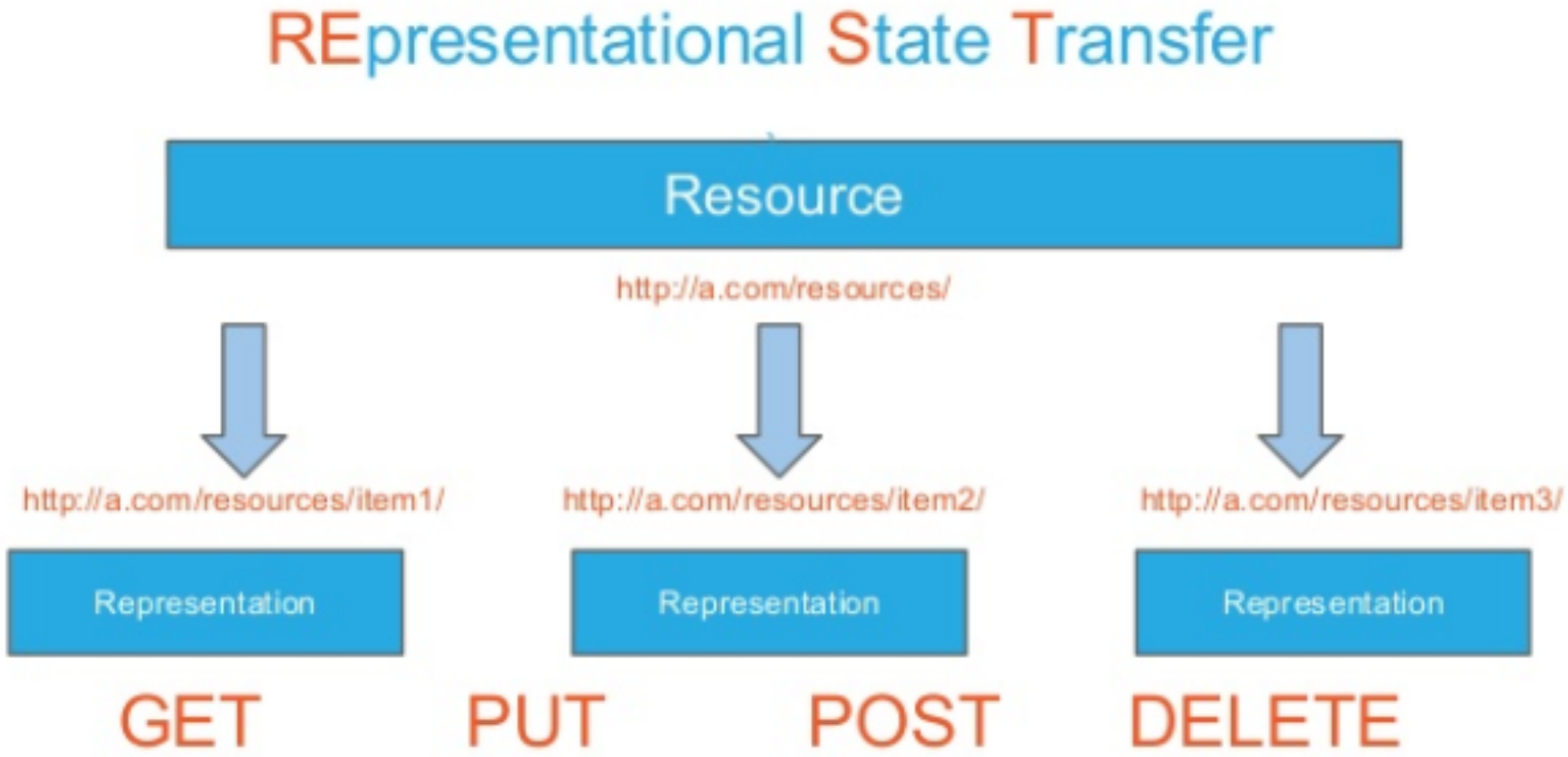
Service Oriented Architecture



- SOAP vs REST


HTTP Header
POST /services/testWs HTTP/1.1
Content-Type: text/xml;charset=UTF-8
SOAPAction: ""
..
SOAP Envelope
SOAP Header
SOAP Body
SOAP
Part
(test/xml)
Attachment
Attachment
Attachment
Attachment
Contest-Type : text/plain
Content-Id : update-address
..

- SOAP vs REST


REpresentational State Transfer
Resource
http://a.com/resources/
http://a.com/resources/item1/
http://a.com/resources/item2/
http://a.com/resources/item3/
Representation
Representation
Representation
GET
PUT
POST
DELETE

- REST

  - LEVEL 0
    - Expose Soap Web Services in Rest Style

      http://server/getPosts
      http://server/deletePosts
      http://server/doThis

- REST

  - LEVEL 1
    - Expose Resources with Proper URI

      http://server/accounts
      http://server/accounts/10

    Note: Improper Use of Http Methods

- REST

  - LEVEL 2
    - LEVEL1 + ***Http Methods***

      GET
      POST
      PUT
      DELETE

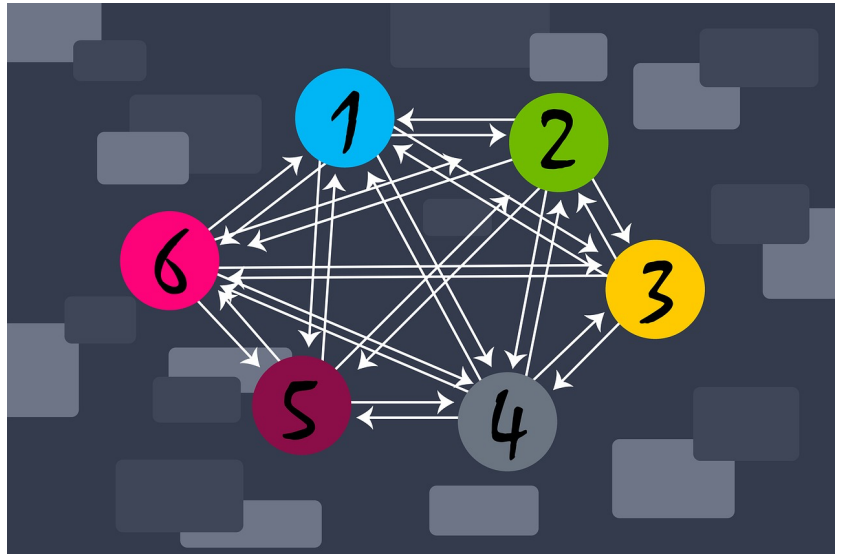
/books

GET	/books	Lists all the books in the database
DELETE	/books/{bookId}	Deletes a book based on their id
POST	/books	Creates a Book
PUT	/books/{bookId}	Method to update a book
GET	/books/{bookId}	Retrieves a book based on their id

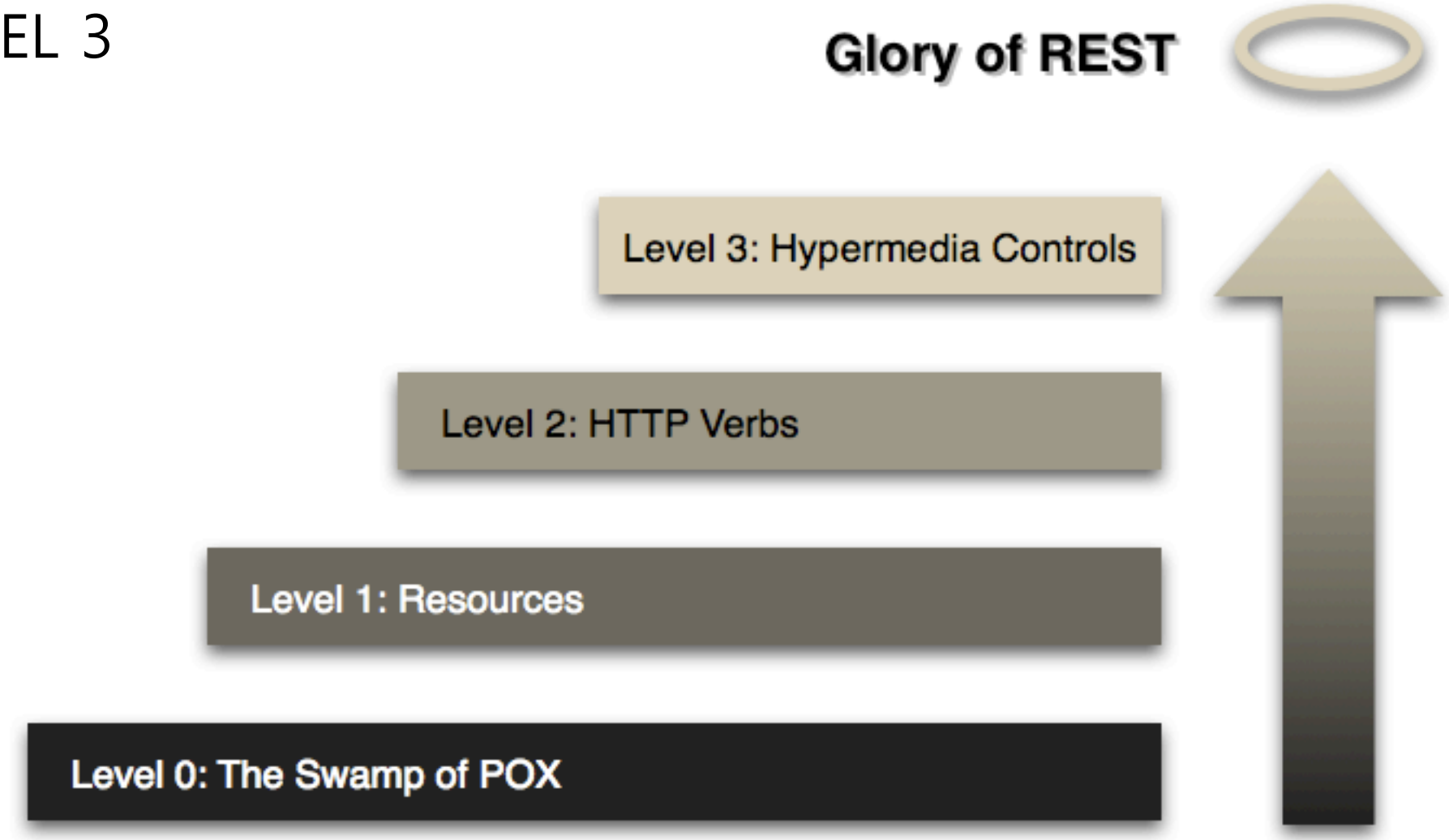
- REST

  - LEVEL 3
    - LEVEL2 + **HATEOAS**
    - DATA + Next Possible Actions
    - RESTful API를 사용하는 클라이언트가 전적으로 서버와 동적인 상호작용이 가능하도록 하는 것

    - Hypermedia as the Engine of Application State
    - It basically means that your application state should be Hypermedia driven.
    - Hypermedia ? – Basically hyperlinks to other resources.
    - Which also means that, once a client access a resource, it can use the hypermedia to navigate your resources.



- REST
  - LEVEL 3


Glory of REST
Level 3: Hypermedia Controls
Level 2: HTTP Verbs
Level 1: Resources
Level 0: The Swamp of POX

- REST

  - Best Practices in Restful Design
    - *Consumer First*
    - *Make Best Use of Http*
    - *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*

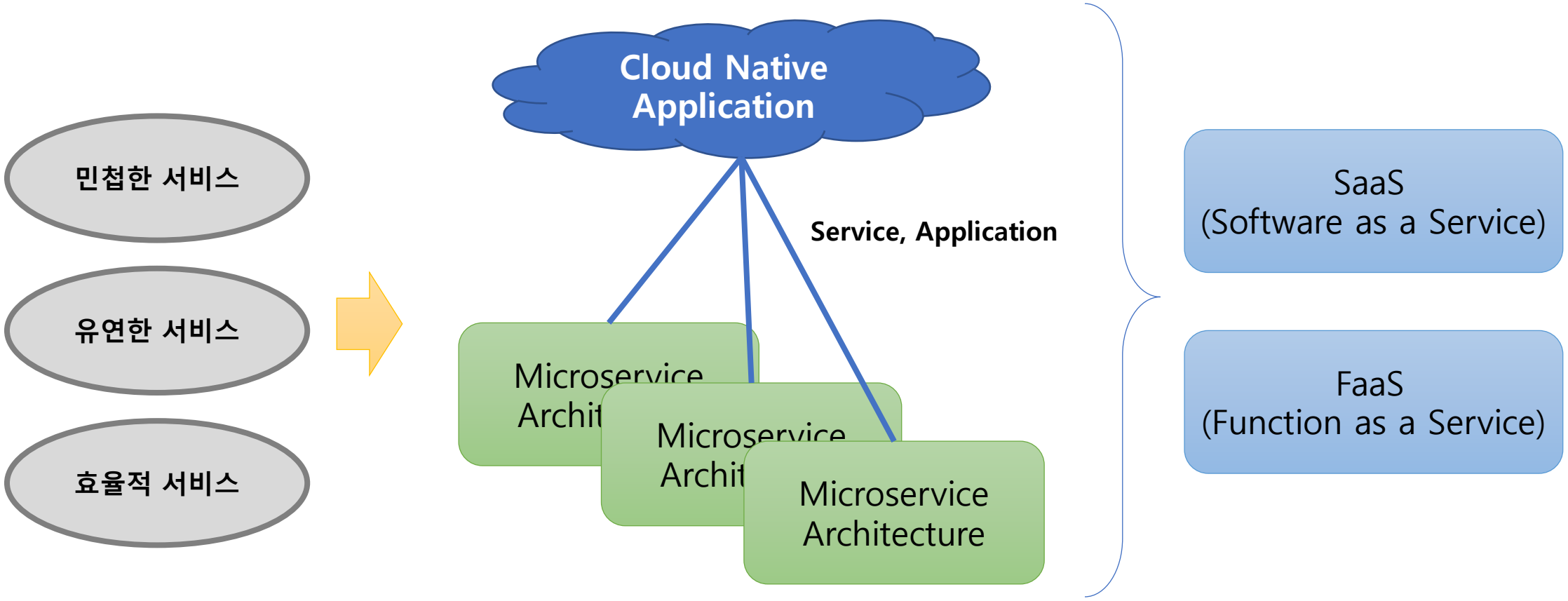
- REST

  - Response Status
    - 200 – SUCCESS
    - 201 – CREATED
    - 400 - BAD REQUEST
    - 401 – UNAUTHORIZED
    - 404 - RESOURCE NOT FOUND
    - 500 - SERVER ERROR

Cloud Native

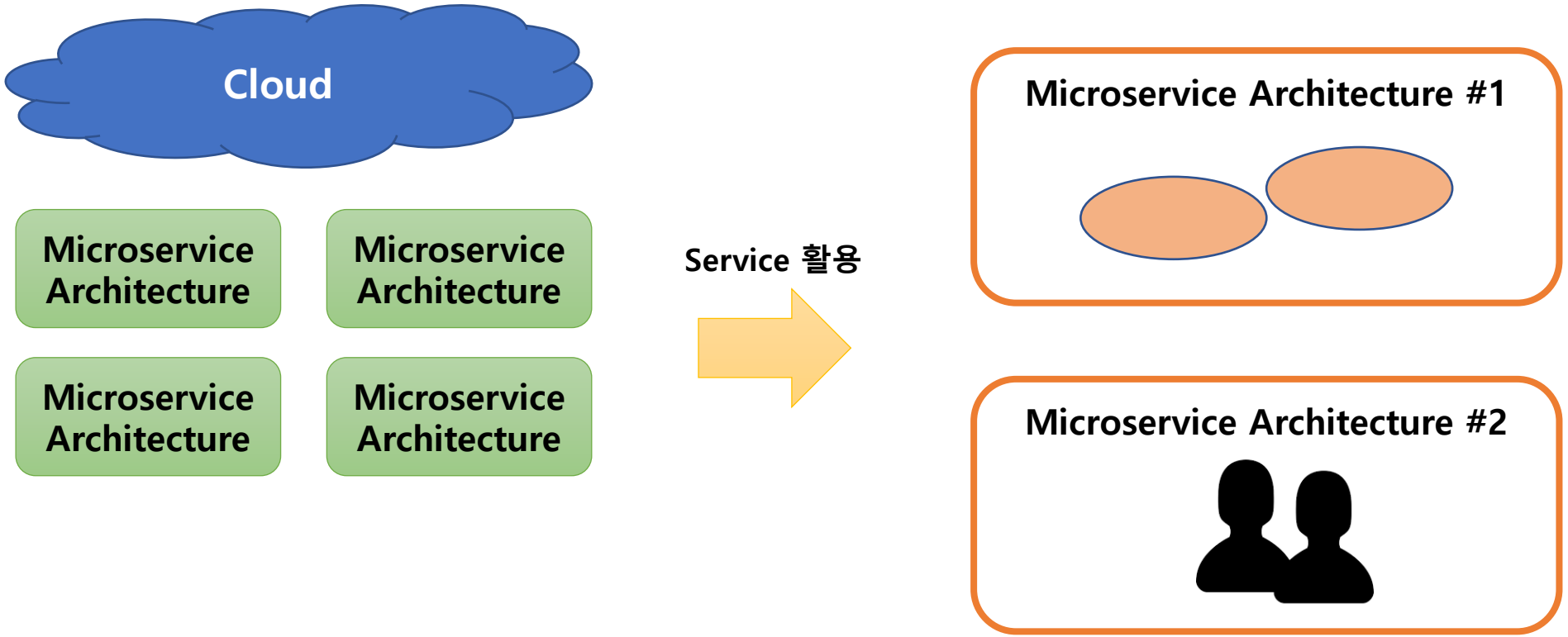
- Cloud Native Application

  - 클라우드 네이티브 환경에서 SaaS난 FaaS 형태로 서비스되는 애플리케이션



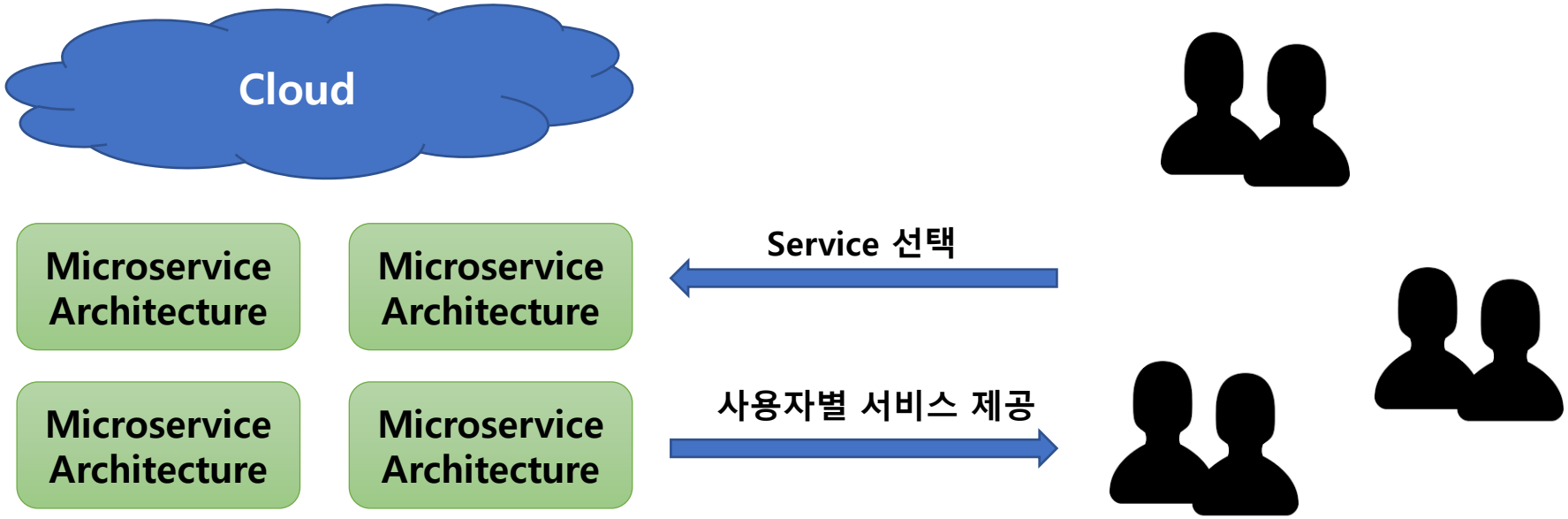
- Cloud Native Application

  - Microservice

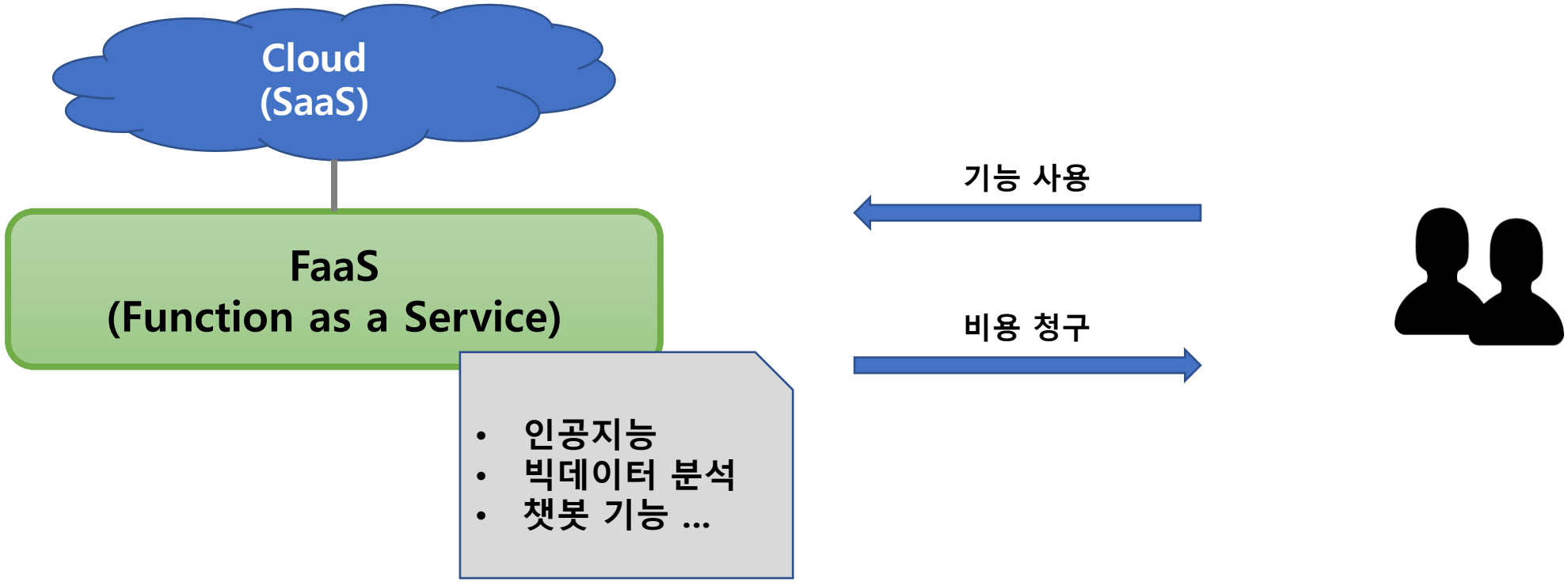


- Cloud Native Application

  - SaaS

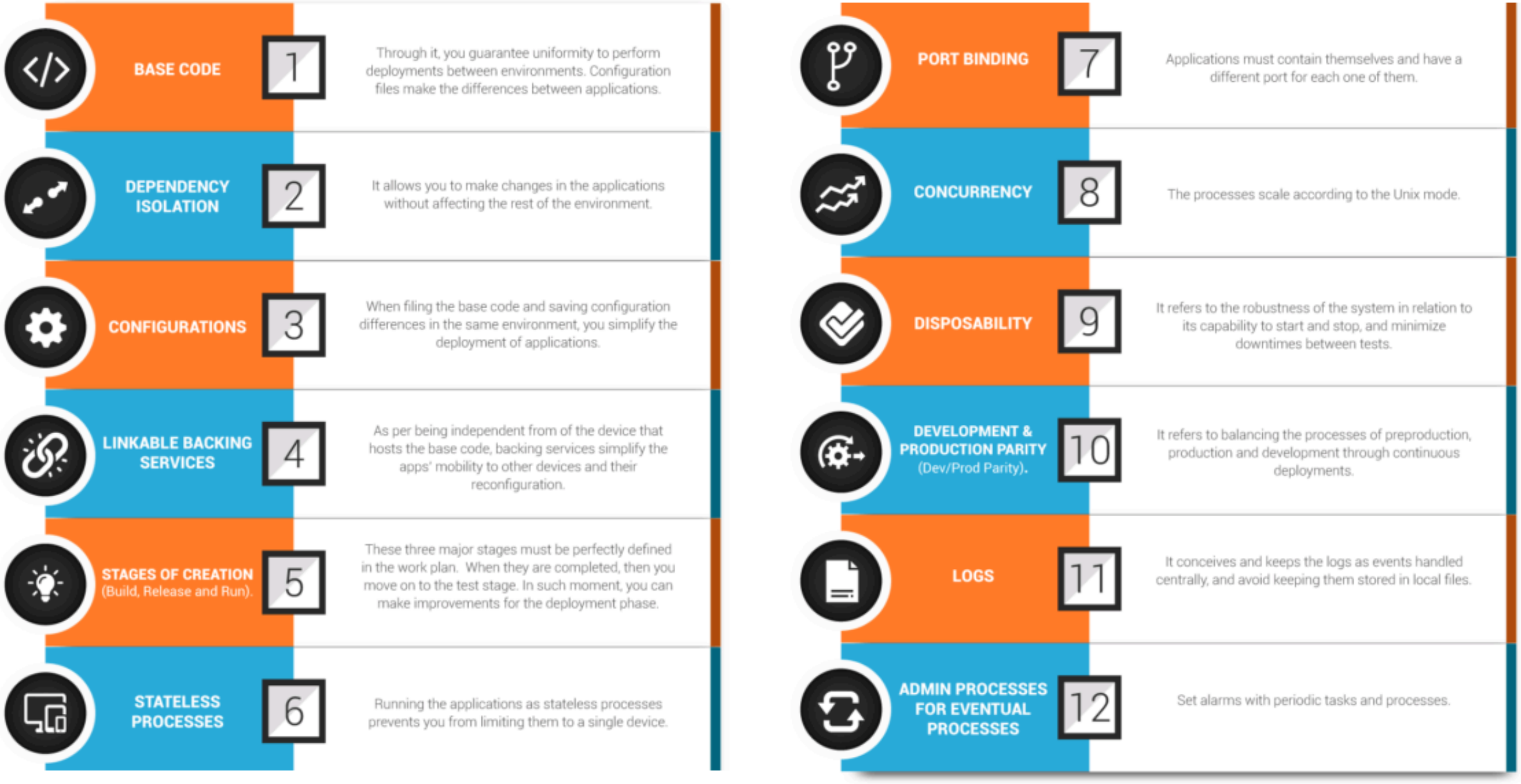


- Cloud Native Application
  - FaaS(Function as a Service)
    - *SW 수준보다 더 세분 된 기능 단위로 개발*
    - *필요할 때 즉시 사용*
    - *기능들의 조합을 통해서 새로운 서비스 생성*



Cloud Native

- Cloud Native Application - 12 Factors



- Cloud Native Application - 12 Factors

1. Codebase

- Single code base for an application to produce immutable releases for different environments.
- Git, Subversion, Mercurial, etc.

- Cloud Native Application - 12 Factors

2. Dependencies

- Applications can't assume server or application container will have everything they need
  - No reliance on specific system tools. i.e. Linux tool not available on Windows
- Dependencies
  - Packaged as jars (Java), RubyGems, CPAN (Perl)
  - Declared in a Manifest(pom.xml or gradle)

- Cloud Native Application - 12 Factors

3. Configuration

- Configuration
  - URLs, Credentials, properties, etc should be sperate from code
- Acid Test – could the codebase be made open source?
- Internal wiring (i.e. Spring configuration) considered part of codebase
  - Environment Variables recommended

- Cloud Native Application - 12 Factors

4. Backing Services

- Backing Services
  - DB, Message Queues, SMTP servers
- Should be possible to attach and detach backing services without re-deployment
- Application code assumes configuration of resource binding takes place external to application

- Cloud Native Application - 12 Factors

5. Build, Release, Run

- Single Codebase to **BUILD** to produce compiled artifact
- Then merged with external Configuration to produce **RELEASE**
- Then delivered to cloud environment(Dev, QA, Prod) and **RUN**

- Cloud Native Application - 12 Factors

6. Processes

- Single Application should be single, **stateless** process
- Processes should not store internal state (HTTP Sessions)
  - Shared Nothing
- Data needing to be shared should be persisted
  - Memory / local tmp storage considered volatile
  - Processes may intercommunicate via messaging / persistent storage

- Cloud Native Application - 12 Factors

7. Port binding

- Environment specific Port Binding without having to change any code
  - Port could be different among environments
- Let Cloud Provider decide and manage port assignment

- Cloud Native Application - 12 Factors

8. Concurrency

- Processes are first class citizens
- Disposable, Stateless, Shared-Nothing model
- Embrace Horizontal Scaling, not Vertical
- Must be able to span multiple machines, or containers

- Cloud Native Application - 12 Factors

9. Disposability

- Should be quick to start and stop
- Should exit gracefully / finish current requests.
- Or should be **idempotent / reentrant**
- Enhances scalability and fault tolerance

- Cloud Native Application - 12 Factors

10. Dev/prod parity

- Differences are from every angle
  - Kernel Parameter, DB drivers, JVM version, Firewall rule, security rule, etc
  - No more “It works on my machine”
- No More Snowflakes!

- Cloud Native Application - 12 Factors

11. Logs

- Logs are streams of aggregated, time-ordered events
  - Apps are not concerned with log management
  - Just write to STDOUT, STDERR
  - Separate log managers handle management
- Logging as a service
- Can be managed via tools like ELK, Splunk ...
  - Log indexing and analysis

- Cloud Native Application - 12 Factors

12. Admin processes

- Admin Processes / Management Tasks Run as One- Off Processes.
- DB Migrations, one time scripts, etc
- Use same environment, tools, language as application processes
- REPL

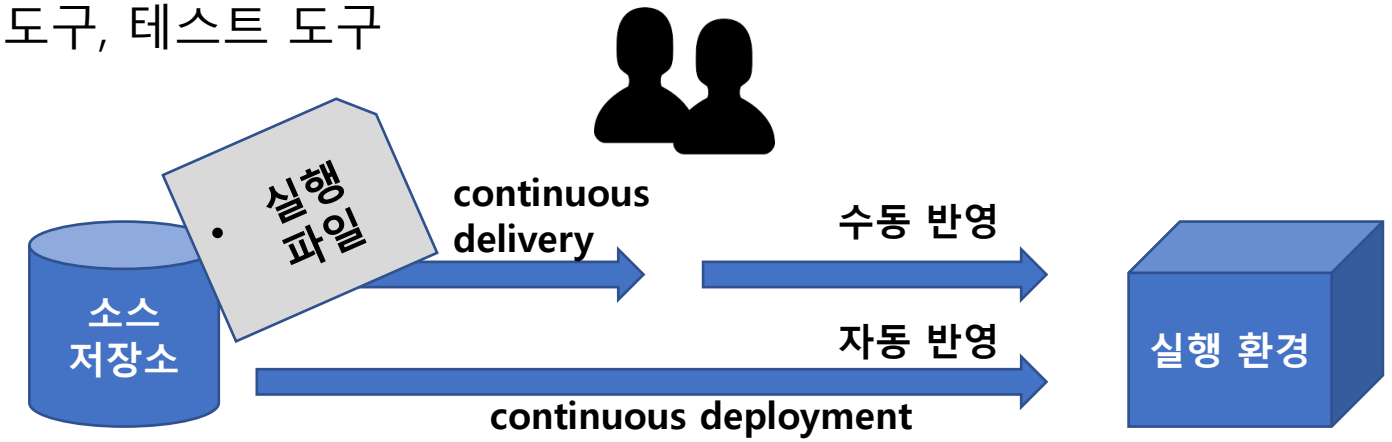
Cloud Native

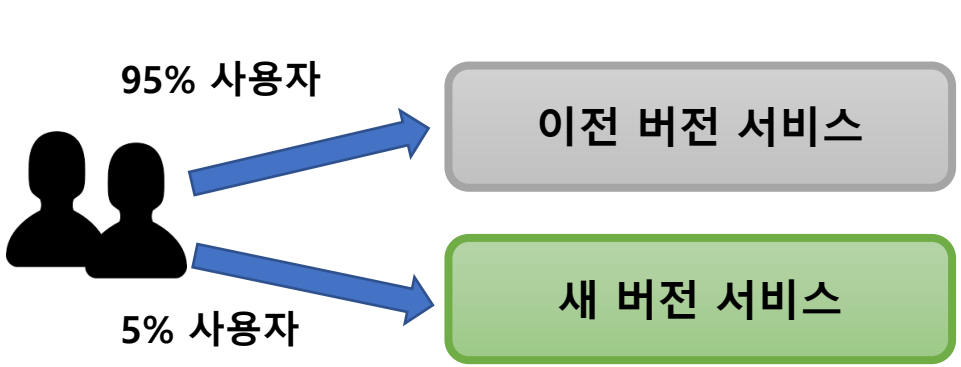
- Cloud Native Architecture
  - 확장 가능한 아키텍처
    - *시스템의 수평적 확정에 유연*
    - *확장된 서버로 시스템의 부하 분산, 가용성 보장*
    - *모니터링*
  - 탄력적 아키텍처
    - *서비스 생성 - 통합 - 배포, 비즈니스 환경 변화에 대응 시간 단축*
    - *분활 된 서비스 구조*
    - *무상태 통신 프로토콜*
  - 장애 격리 (Fault isolation)
    - *특정 서비스에 오류가 발생해도 다른 서비스에 영향 주지 않음*

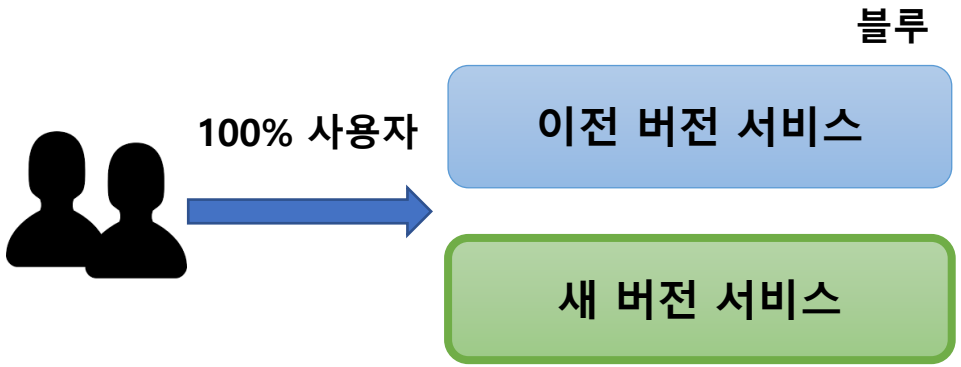
- Cloud Native Infra

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

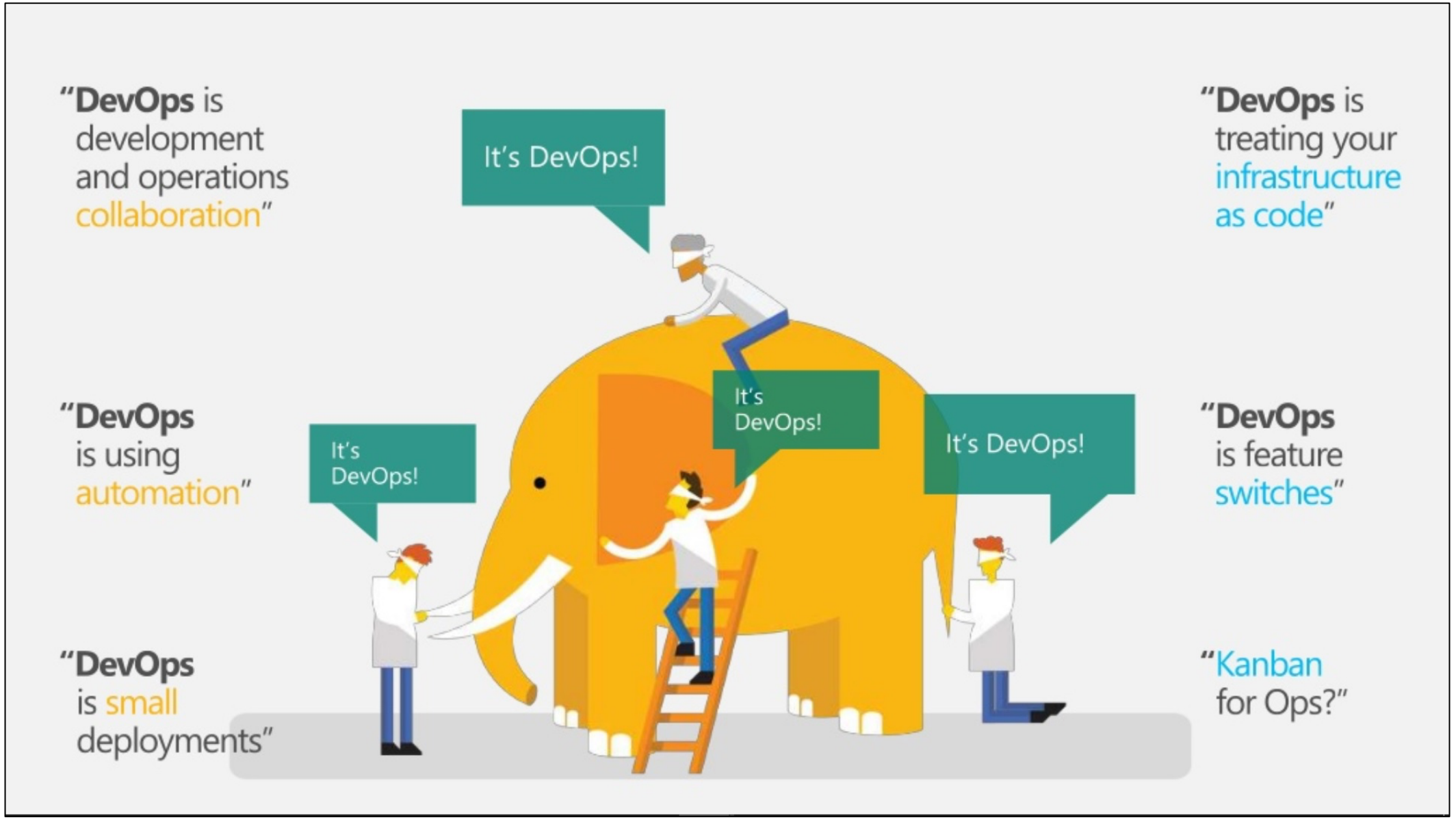
- 지속적 통합과 배포
  - CI(Continuous Integration)
    - 통합 서버, 소스 관리 (SCM), 빌드 도구, 테스트 도구
    - ex) Jenkins
  - 지속적 배포
    - Continuous Delivery
    - Continuous Deployment
    - Pipe line
  - 카나리 배포와 블루그린 배포







- DevOps


"DevOps is development and operations collaboration"
"DevOps is using automation"
"DevOps is small deployments"
It's DevOps!
It's DevOps!
It's DevOps!
It's DevOps!
"DevOps is treating your infrastructure as code"
"DevOps is feature switches"
"Kanban for Ops?"

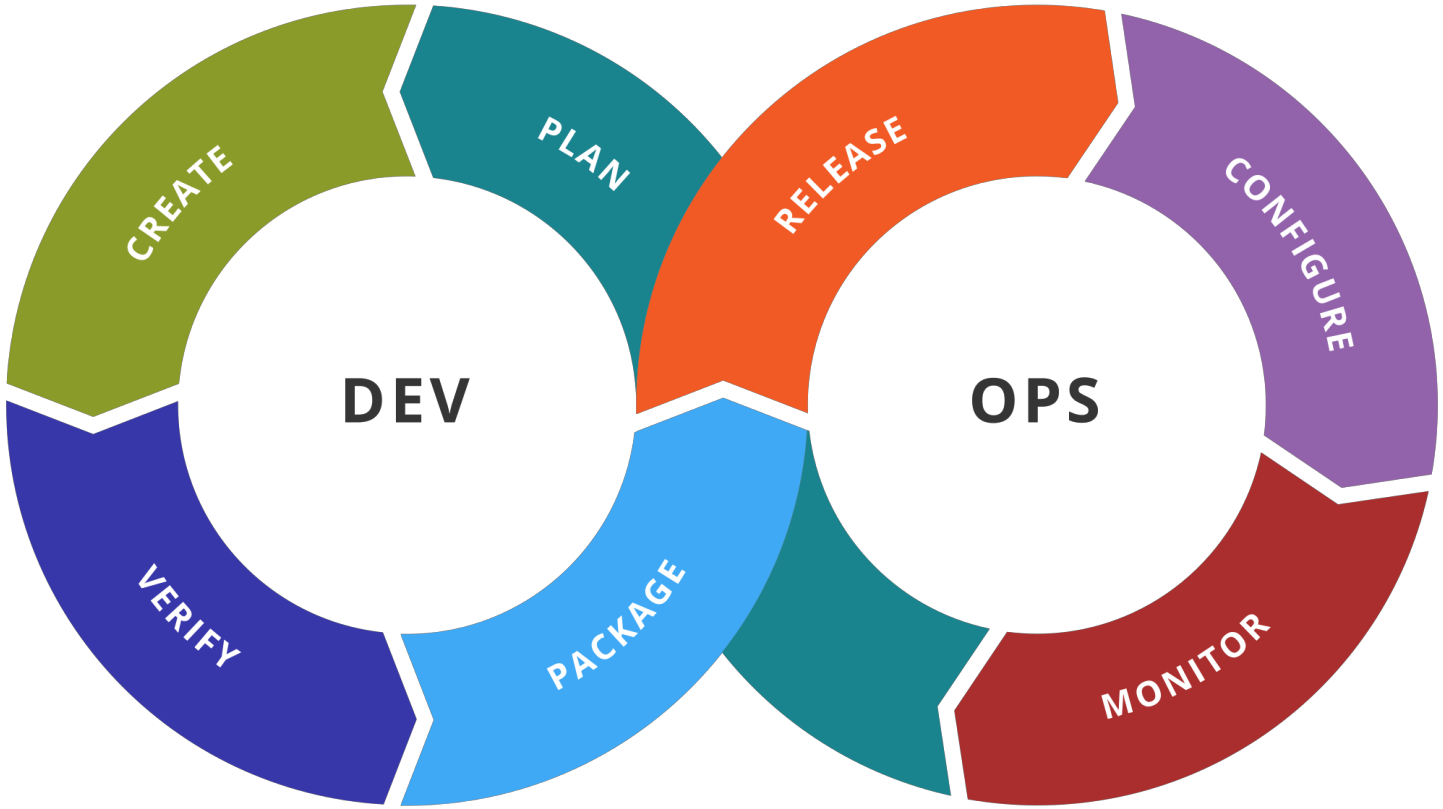
- DevOps
  - 2007~2008년 → IT 운영 및 SW 개발에 문제점 대두
    - 개발과 배포가 다른 조직에서 관리 (다른 목표를 가짐)
  - 협업 및 신뢰
  - 더 빠른 릴리스, 더 스마트한 업무 진행
  - 문제 해결 시간 단축
  - 계획되지 않은 업무 쉽게 관리

- 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

- 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***

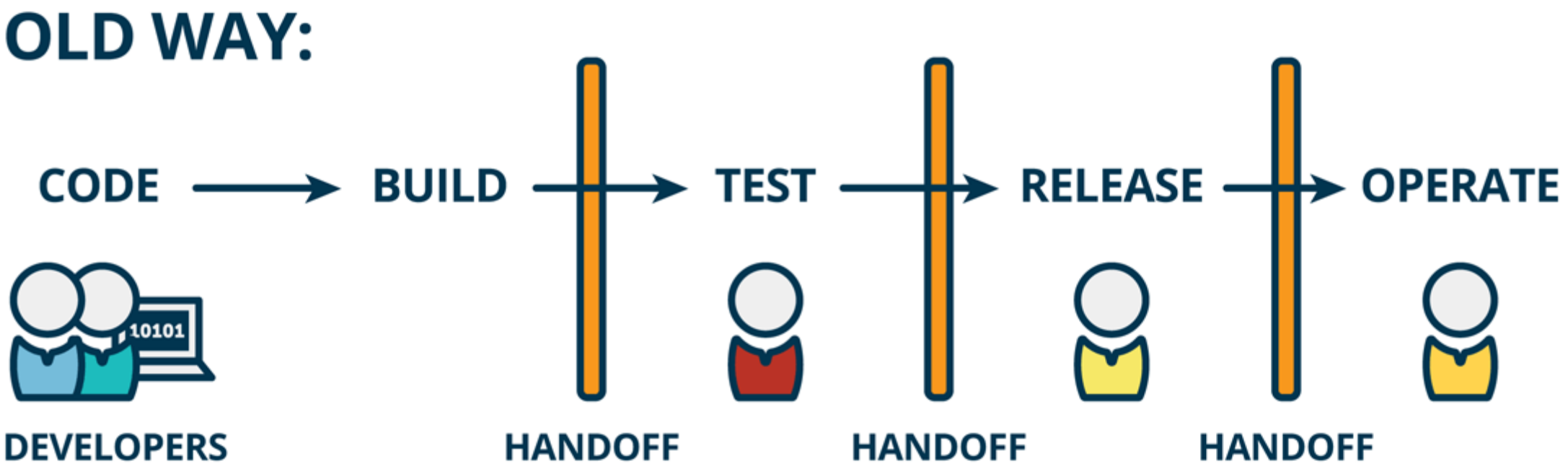


- DevOps

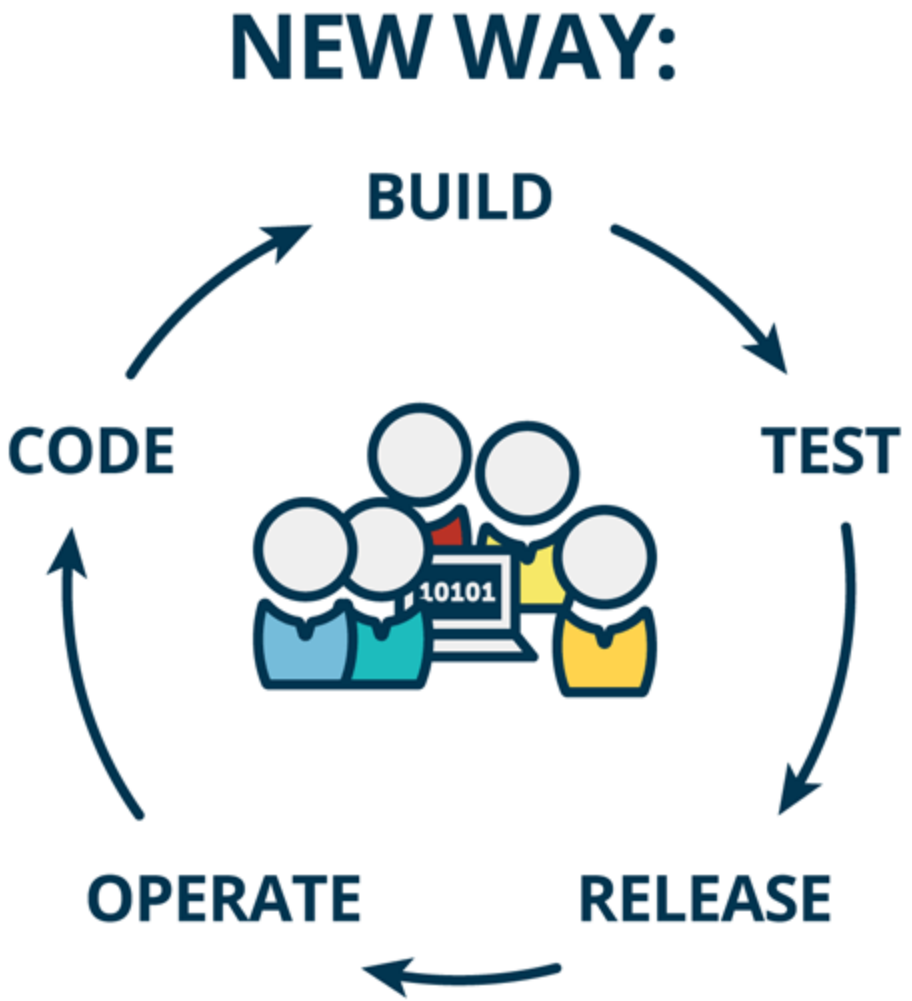
  - Microservice Team Structure

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

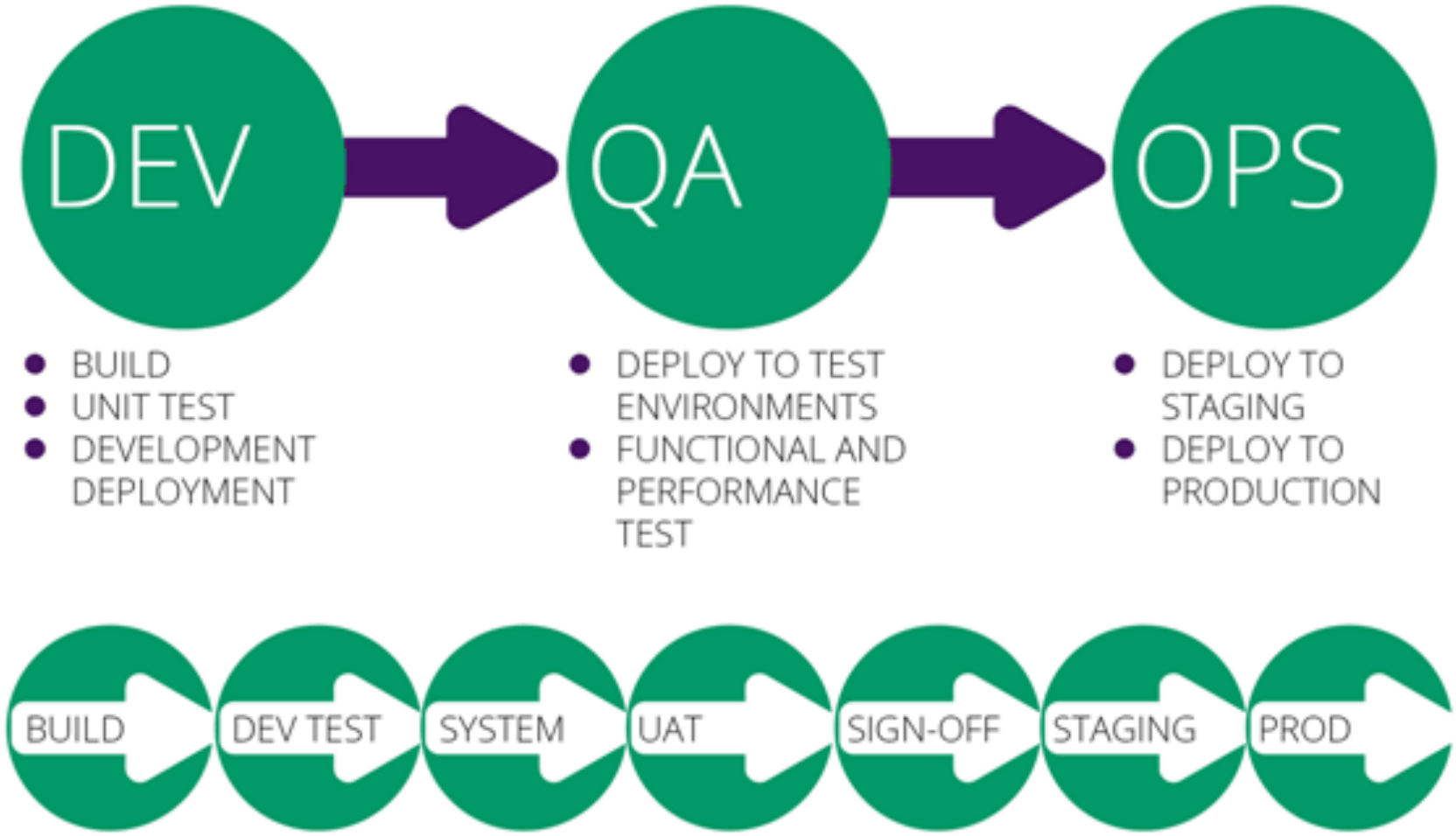
- DevOps


OLD WAY:
CODE
BUILD
TEST
RELEASE
OPERATE
10101
DEVELOPERS
HANDOFF
HANDOFF
HANDOFF

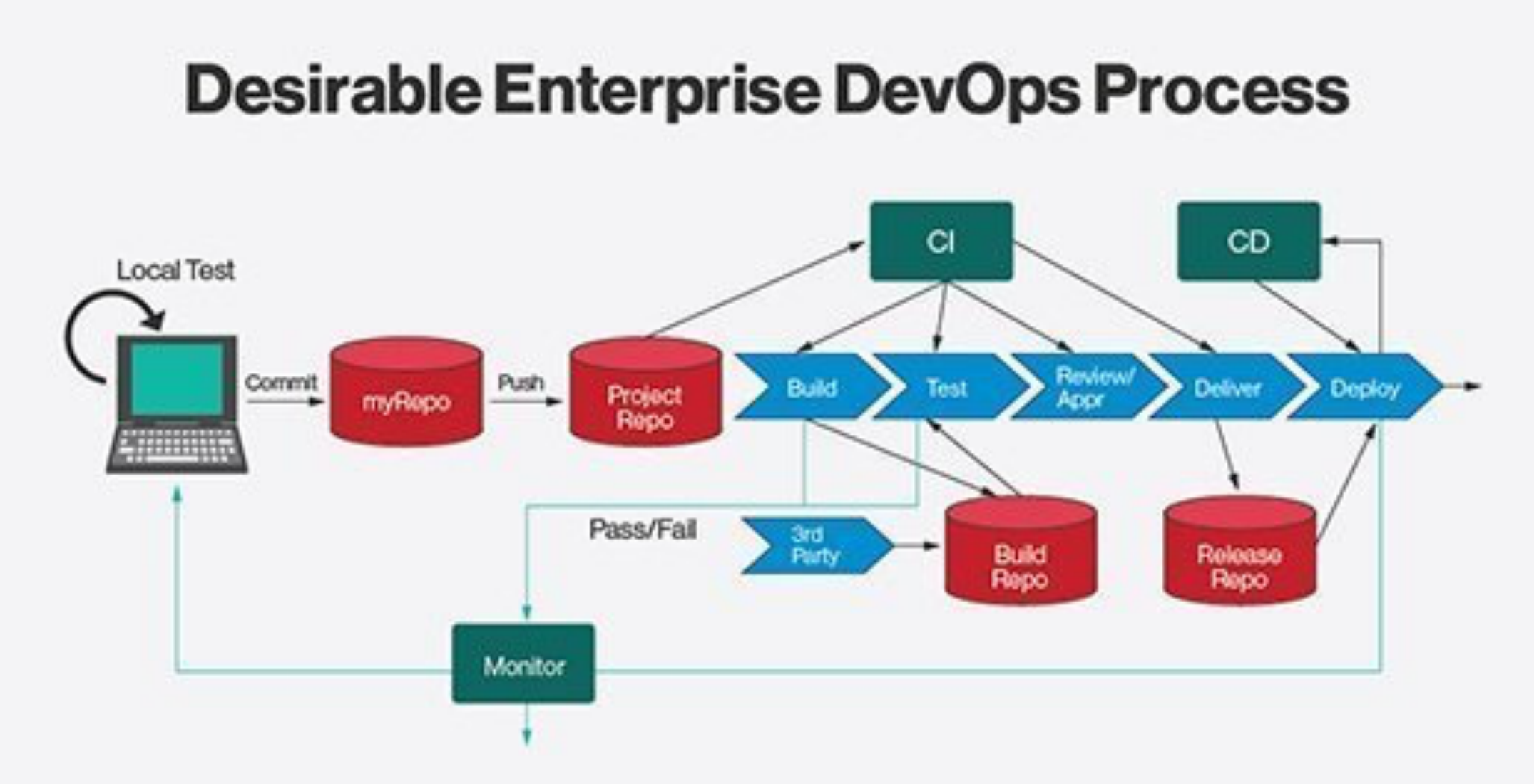
- DevOps



- DevOps


DEV
QA
OPS
BUILD
UNIT TEST
DEVELOPMENT DEPLOYMENT
DEPLOY TO TEST ENVIRONMENTS
FUNCTIONAL AND PERFORMANCE TEST
DEPLOY TO STAGING
DEPLOY TO PRODUCTION
BUILD
DEV TEST
SYSTEM
UAT
SIGN-OFF
STAGING
PROD

- DevOps
  - 자동화와 시각화

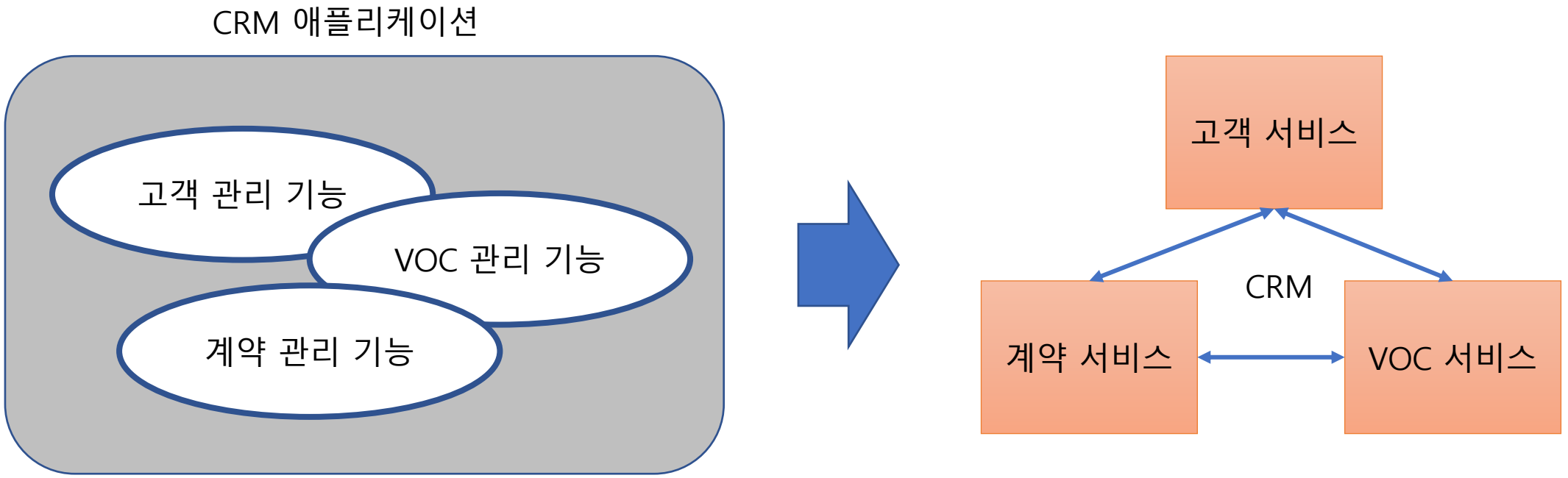


- Monitoring
  - ***Health check API*** - expose an endpoint that returns the health of the service
  - ***Log aggregation*** - log service activity and write logs into a centralized logging server, which provides searching and alerting
  - ***Distributed tracing*** - assign each external request a unique 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

What is the Microservice Architecture?

Microservice 특징

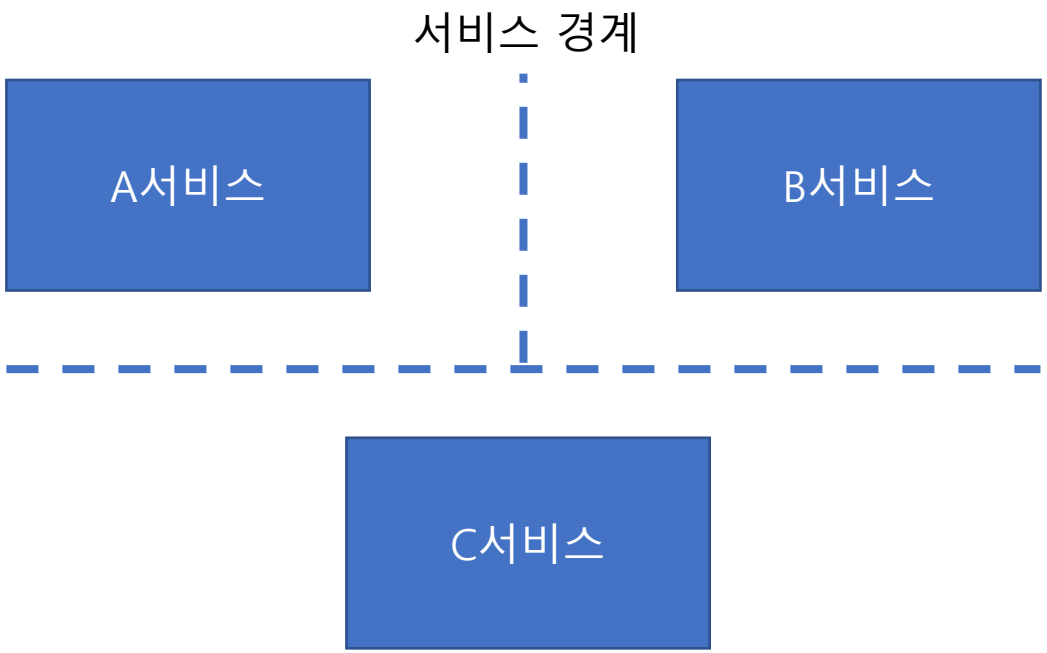
- 작은 서비스
  - ~~독립된 비즈니스~~ → **독립된 서비스**
  - Eric Evans "Domain Driven Design"
    - Bounded Context: 하나의 서비스에 담을 수 있는 기능들의 그룹, 서비스 수준의 경계
    - Ubiquitous Language: 같은 업무를 하는 사람들은 업무상 같은 단어와 용어를 사용



Microservice 특징

- 독립된 서비스
  - 독립적 실행, 다른 서비스 결합이 없는 서비스
  - 구현, 배포 실행, 장애에 대한 영향 받지 않음

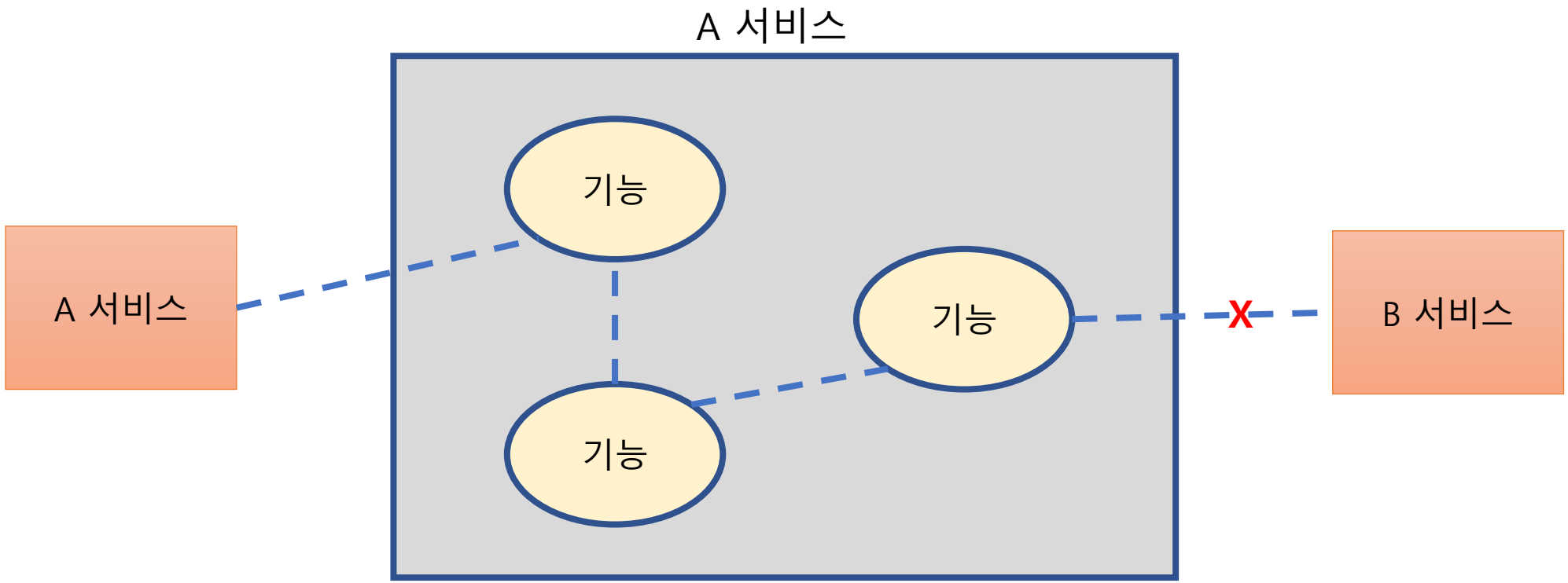
ex) "고객 관리 서비스" 중 고객 등록 기능에 문제가 생겨도, 이와 관계없이 예약 가능



Microservice 특징

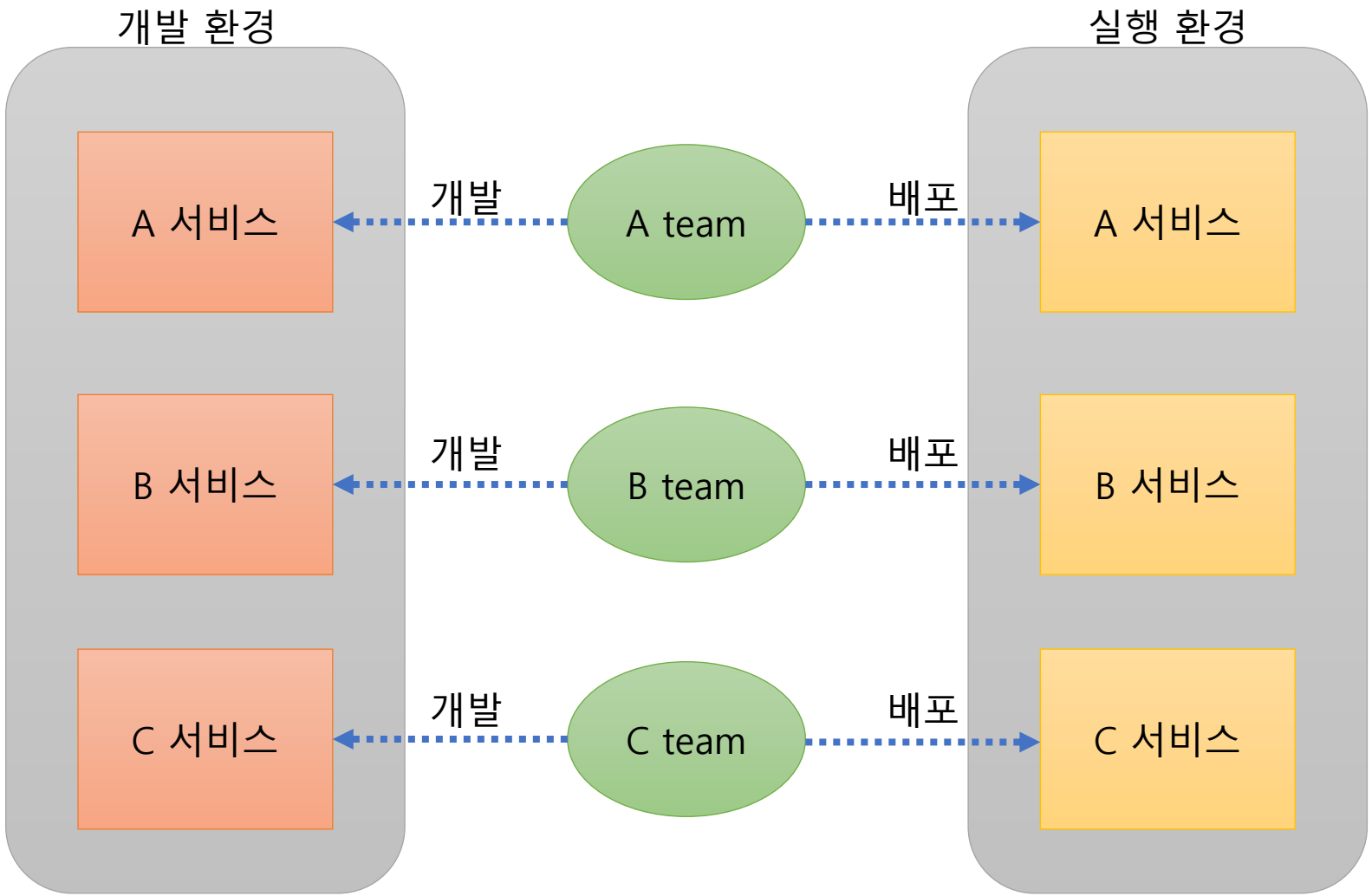
- 응집된 서비스
  - 하나의 서비스는 기능적으로 응집
  - 서비스의 역할이 한 가지 일을 위해 묶여야 함 → 단순 명확, 오류 최소화

    ex) 수강 신청 시스템



Microservice 특징

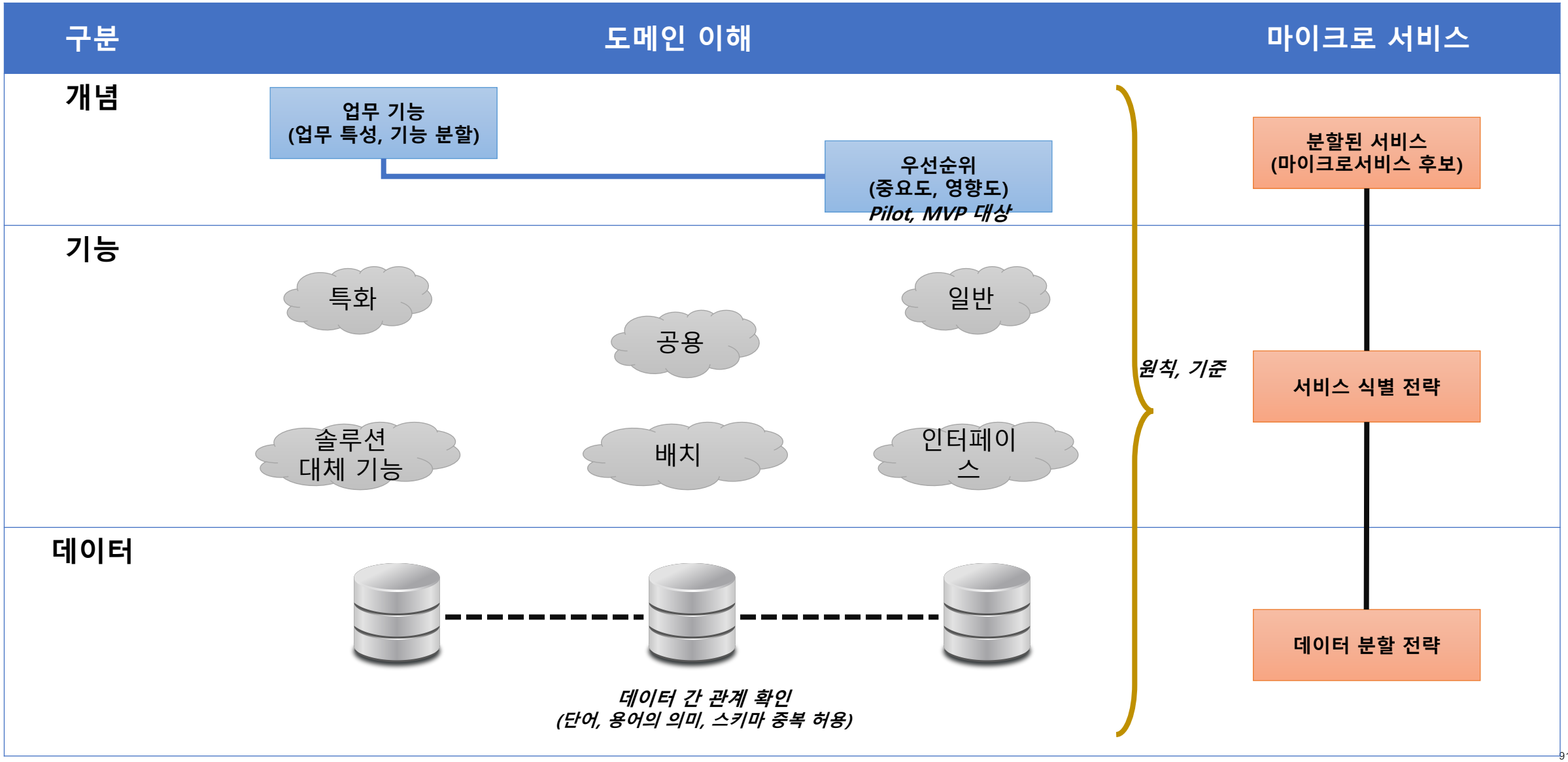
- 자율적 서비스
  - 서비스의 기획, 개발, 테스트, 배포 및 서비스의 운영까지 담당 조직이 독립적으로 의사결정 → Ownership



- Microservice 식별 전략

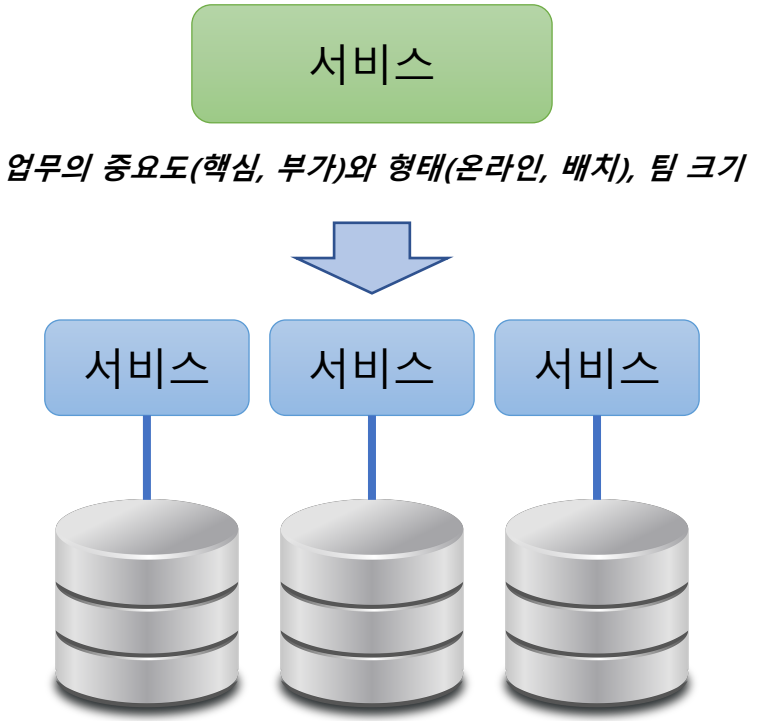
Microservice 기획

- Microservice 식별 전략

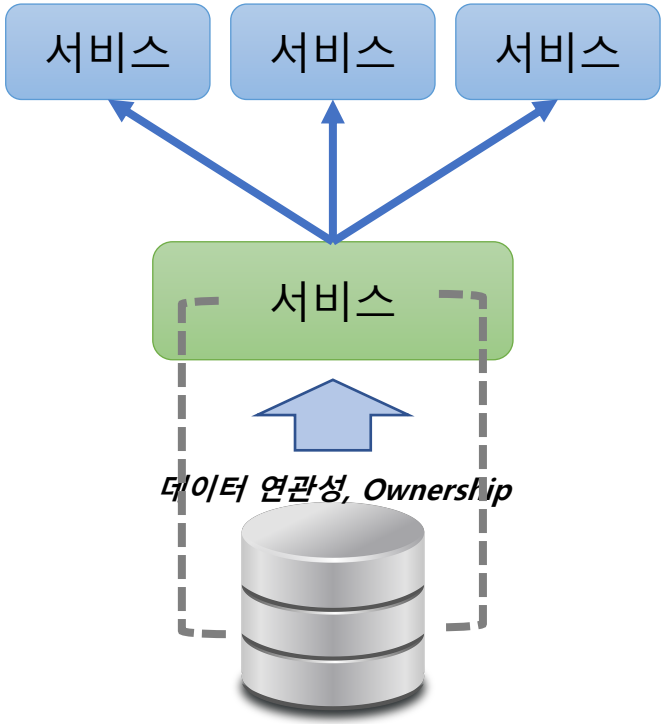


- Microservice 식별 전략

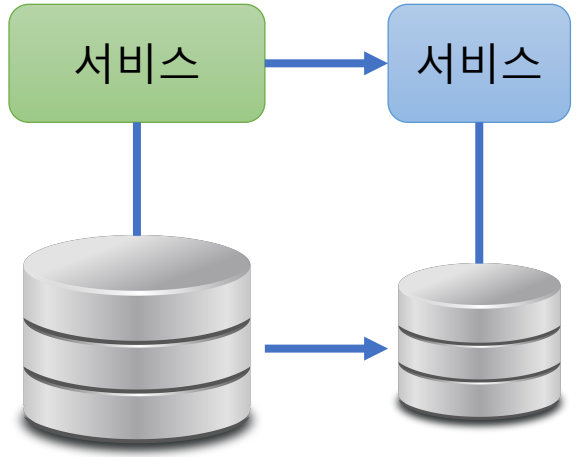
- Microservice 식별 전략



하향식 접근



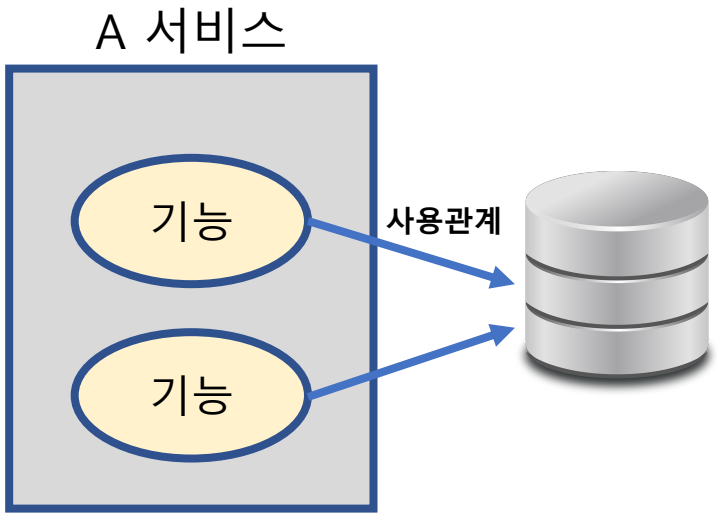
상향식 접근



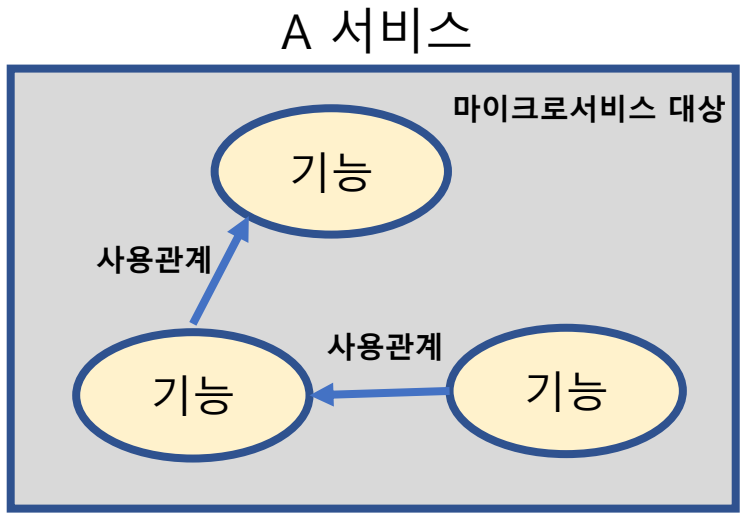
서비스 일부 분리

- Microservice 식별 전략

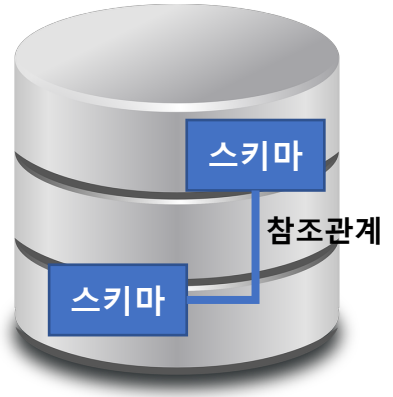
- Microservice 식별 전략



서비스와 데이터 관계

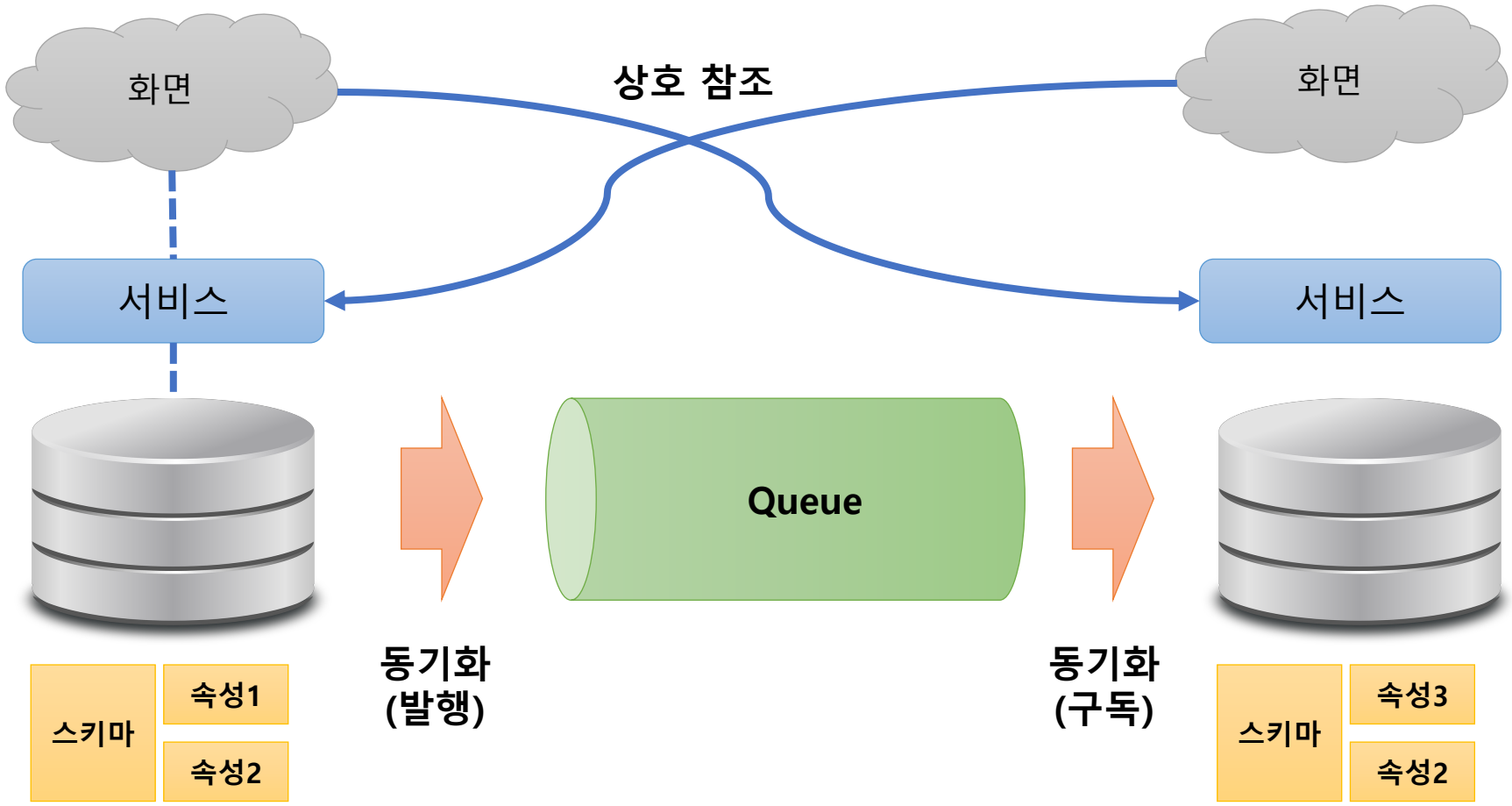


서비스 간의 종속성



데이터 간의 종속성

- Microservice 식별 전략

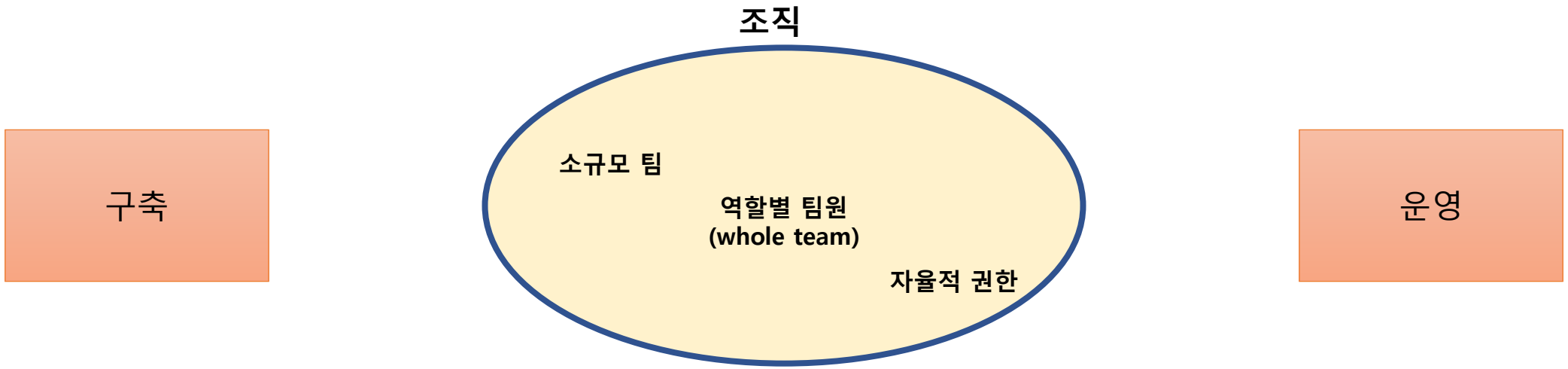


마이크로서비스 간 데이터 연계

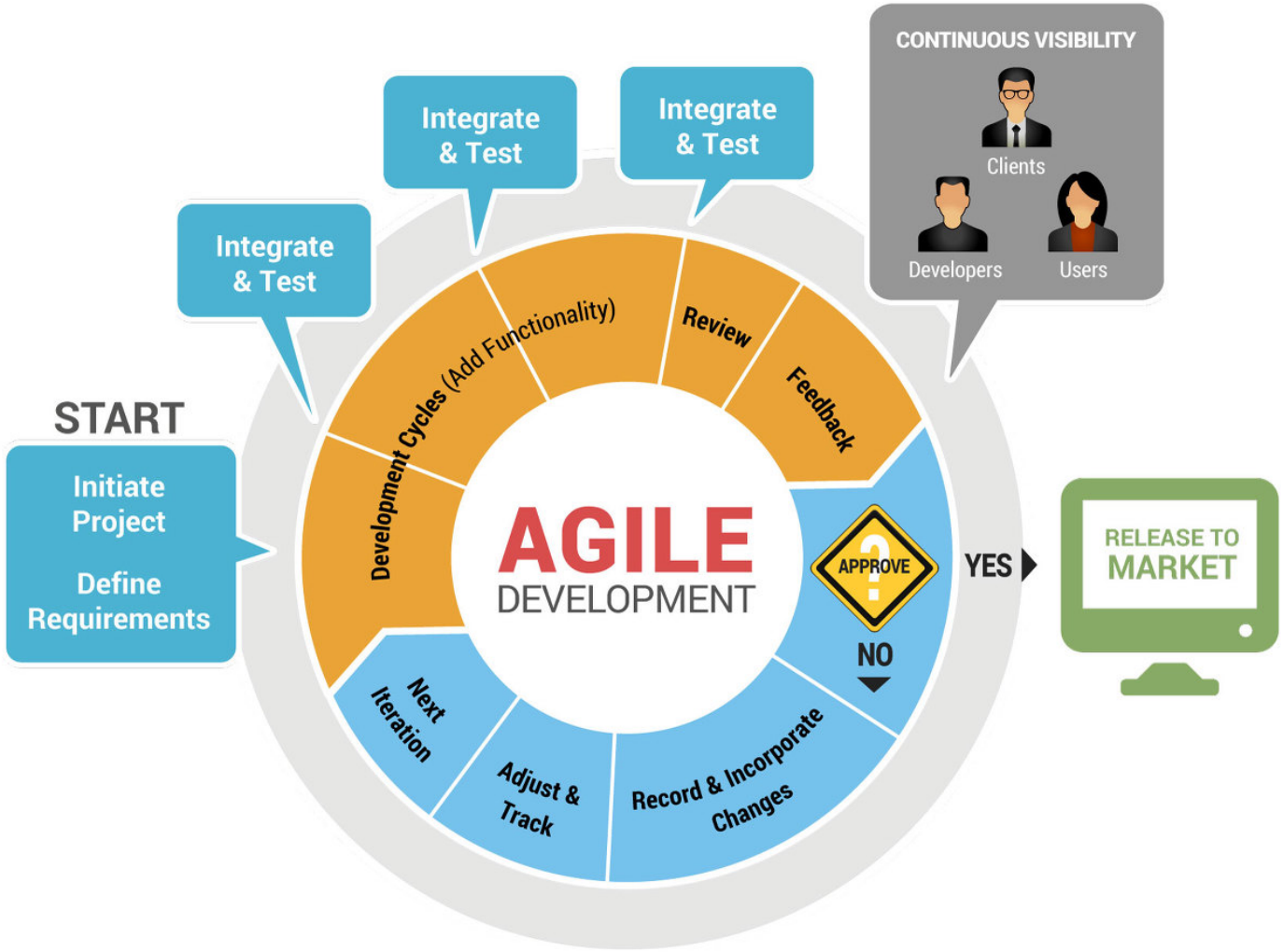
- Microservice 식별 전략

- Microservice 고려 사항

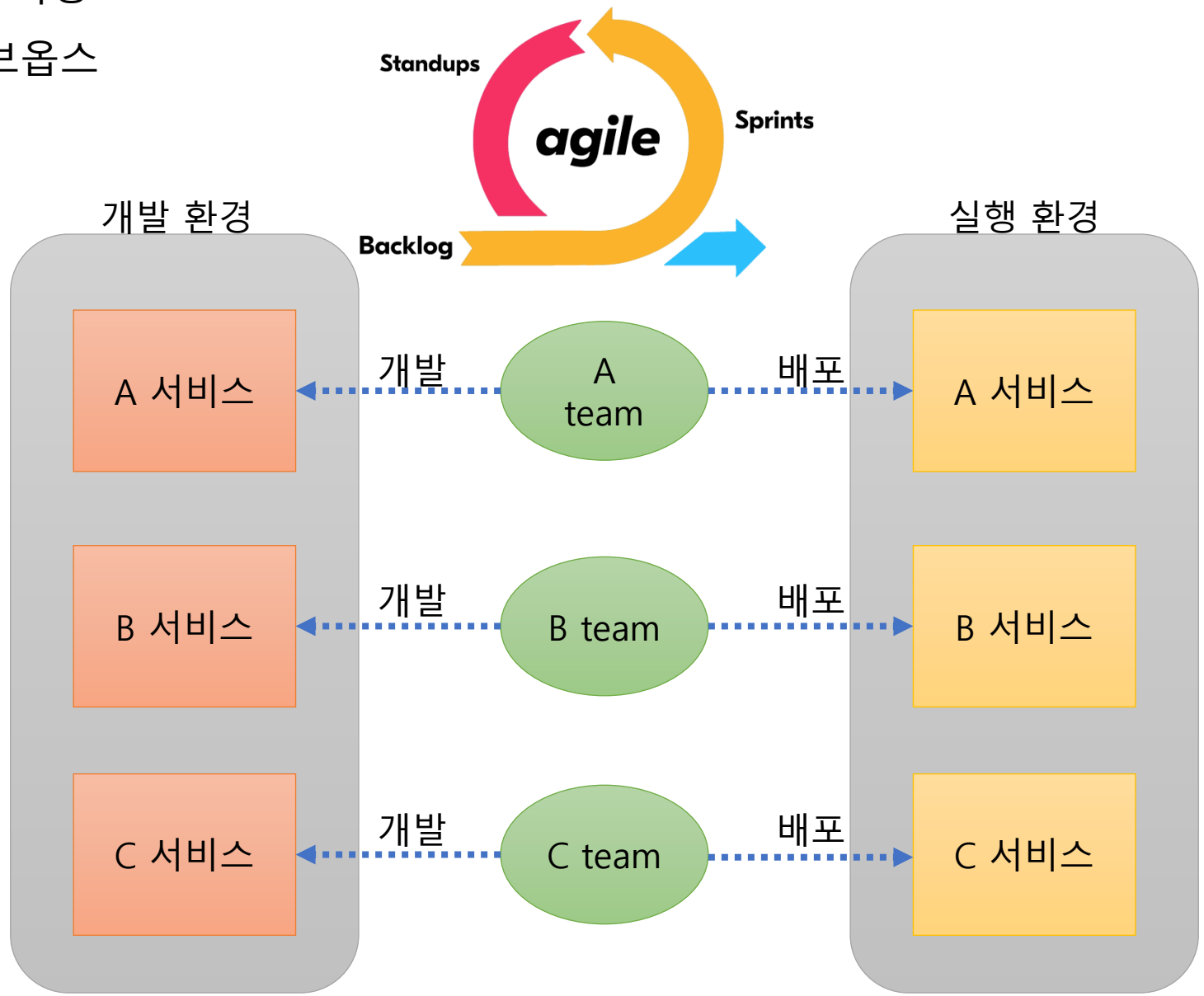
  1) 조직의 구성



- Microservice 고려 사항
  2) 애자일과 데브옵스

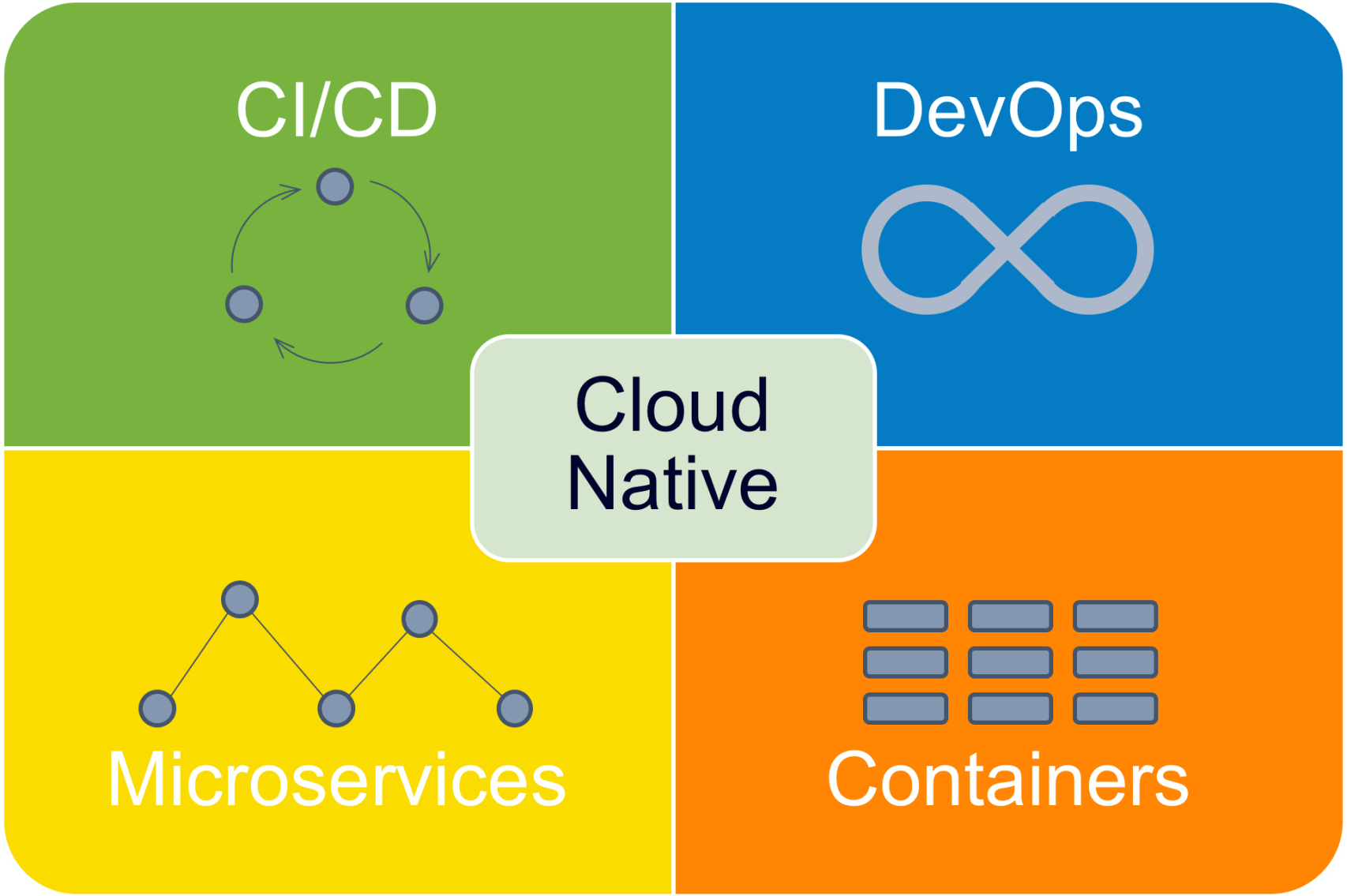


- Microservice 고려 사항
  2) 애자일과 데브옵스

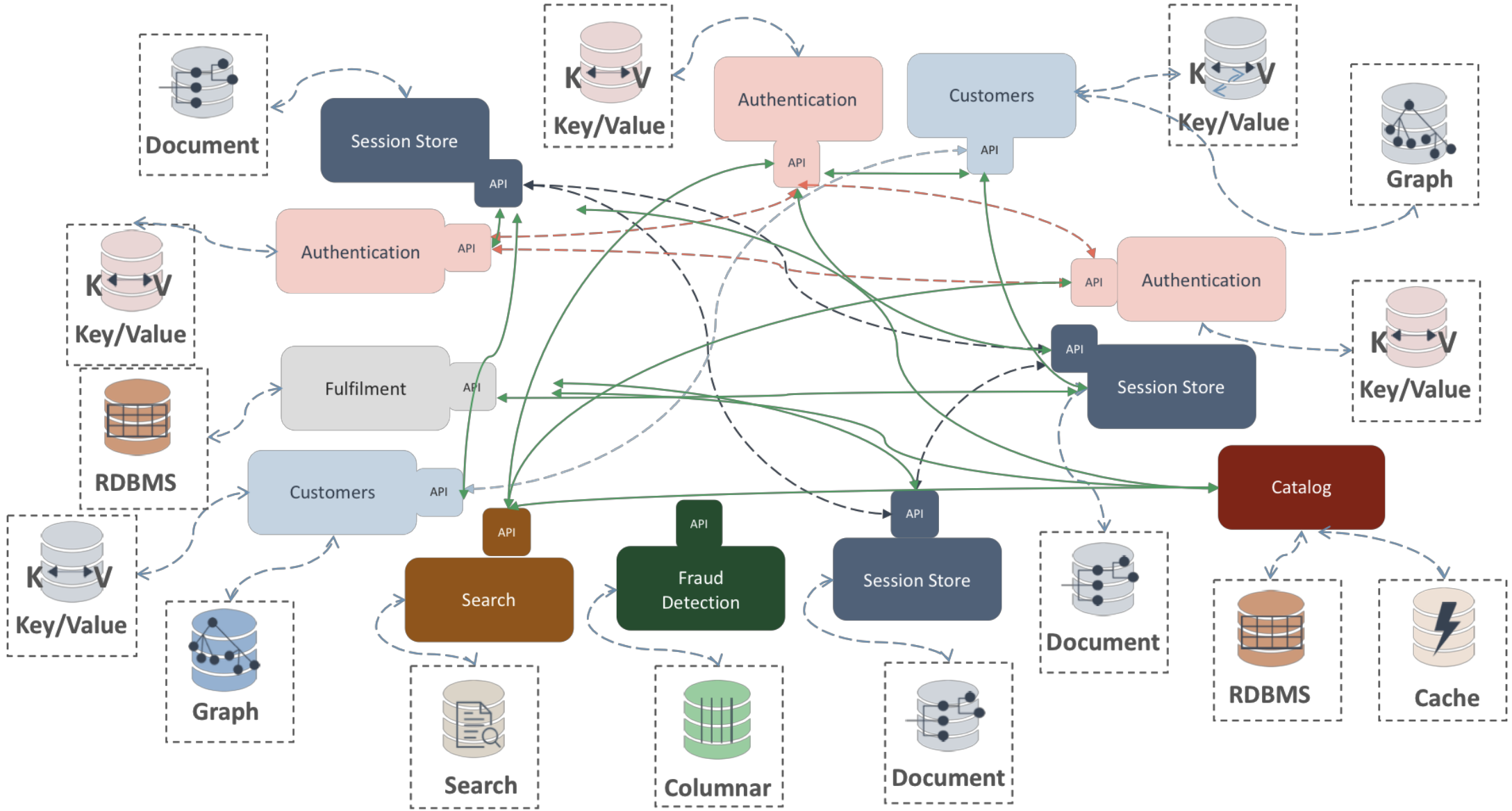


- Microservice 고려 사항

  3) 클라우드 네이티브 기술 환경

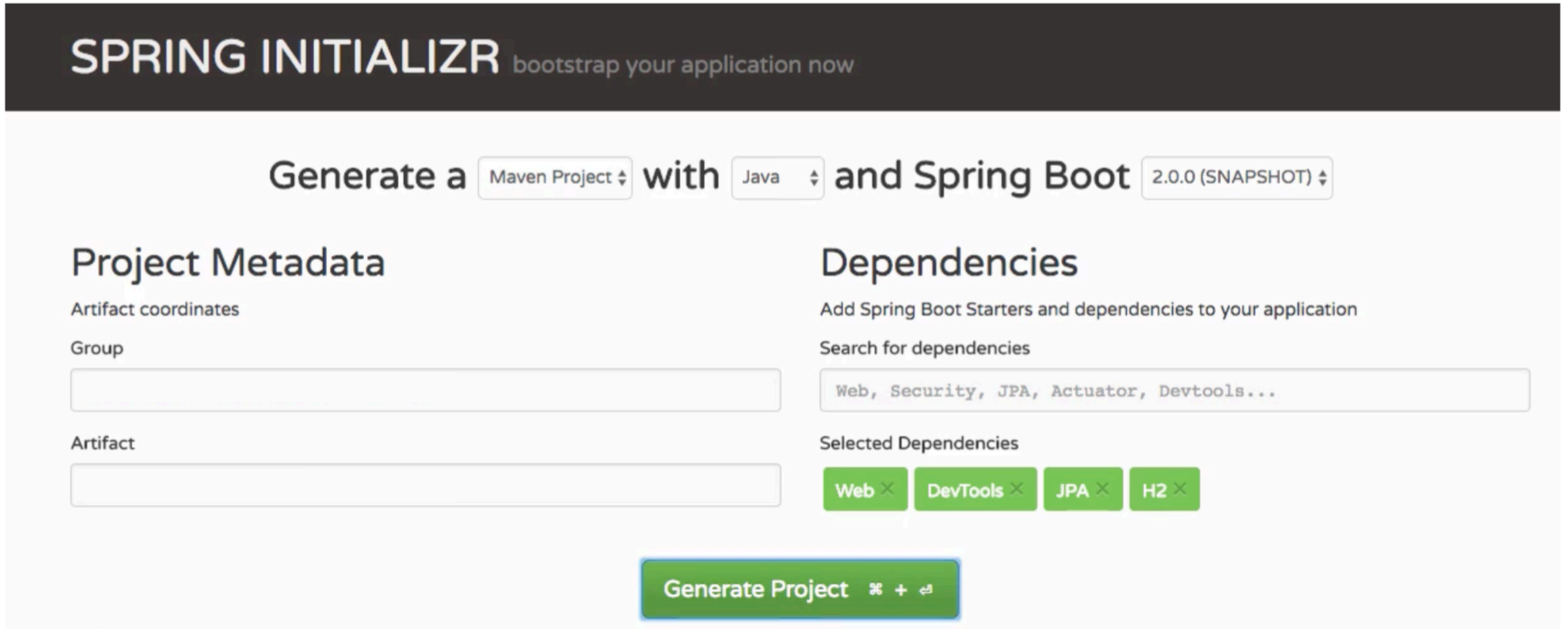


Microservices Architecture and Polyglot Persistence



Spring Boot

Spring Initializr http://start.spring.io/ is great tool to bootstrap your Spring Boot projects.



- User -> Posts
  - Retrieve all Users - GET /users
  - Create a User - POST /users
  - Retrieve one User - GET /users/{id} → /users/1
  - Delete a User - DELETE /users/{id} → /users/1

  - Retrieve all posts for a User - GET /users/{id}/posts
  - Create a posts for a User - POST /users/{id}/posts
  - Retrieve details of a post - GET /users/{id}/posts/{post_id}

Step 03 - Creating a Hello World Service

```
@RestController
public class HelloWorldController {

  @GetMapping(path = "/hello-world")
  public String helloWorld() {
    return "Hello World";
  } }
```

Step 04 - Enhancing the Hello World Service to return a Bean

```
@GetMapping(path = "/hello-world-bean")
public HelloWorldBean helloWorldBean() {
  return new HelloWorldBean("Hello  World");
}
```

/src/main/java/com/example/rest/webservices/restfulwebservices/HelloWorldBean.java 생성

Step 05 - Enhancing the Hello World Service with a Path Variable

/src/main/java/com/example/rest/webservices/restfulwebservices/HelloWorldController.java

```
//Controller
@RestController
public class HelloWorldController {

  @GetMapping(path = "/hello-world")
  public String helloWorld() {
    return "Hello World";
  }

  @GetMapping(path = "/hello-world-bean")
  public HelloWorldBean helloWorldBean() {
    return new HelloWorldBean("Hello World");
   }

  ///hello-world/path-variable/in28minutes
  @GetMapping(path = "/hello-world/path-variable/{name}")
  public HelloWorldBean
helloWorldPathVariable(@PathVariable String name) {
    return new HelloWorldBean(String.format("Hello World,
%s",  name));
  }


}
```

Step 05 - Enhancing the Hello World Service with a Path Variable

/src/main/resources/application.properties 수정

```
logging.level.org.springframework = info
```

Step 06 - Creating User Bean and User Service

/src/main/java/com/example/rest/webservices/restfulwebservices/user/User.java 생성

```
public class User {

  private Integer  id;

  private String

 name;

  private Date birthDate;
```

Step 06 - Creating User Bean and User Service

/src/main/java/com/example/rest/webservices/restfulwebservices/user/UserDaoService.java 생성

```
@Component
public class UserDaoService {
  private static List<User> users = new ArrayList<>();

  private static int usersCount =  3;

  static {
    users.add(new User(1, "Adam", new Date()));
    users.add(new User(2, "Eve", new Date()));
    users.add(new User(3, "Jack", new Date()));
  }

  public List<User> findAll() {
    return users;
   }

  public User save(User user)

 {
    if (user.getId() == null)
```

Step 07 - Implementing GET Methods for User Resource

/src/main/java/com/example/rest/webservices/restfulwebservices/user/UserResource.java 생성

```
public class UserResource {

  @Autowired
  private UserDaoService service;

  @GetMapping("/users")
  public List<User> retrieveAllUsers() {
    return service.findAll();
  }

  @GetMapping("/users/{id}")
  public User retrieveUser(@PathVariable int id) {
    return service.findOne(id);
  }

}
```

/src/main/resources/application.properties 수정

```
spring.jackson.serialization.write-dates-as-timestamps=false
```

Step 08 - Implementing POST Method to create User Resource

/src/main/java/com/example/rest/webservices/restfulwebservices/user/UserResource.java 수정

```
// input - details of user
// output - CREATED & Return the created URI
@PostMapping("/users")
public void createUser(@RequestBody User  user){
  User savedUser = service.save(user);
}
```

Step 09 - Enhancing POST Method to return correct HTTP Status Code and Location

/src/main/java/com/example/rest/webservices/restfulwebservices/user/UserResource.java 수정

```
  // input - details of user
  // output - CREATED & Return the created URI
  @PostMapping("/users")
  public ResponseEntity<Object> createUser(@RequestBody
User user) {
    User savedUser = service.save(user);
    // CREATED
    // /user/{id}     savedUser.getId()

    URI location = ServletUriComponentsBuilder
      .fromCurrentRequest()
      .path("/{id}")
      .buildAndExpand(savedUser.getId()).toUri();

    return  ResponseEntity.created(location).build();

  }
```

Step 10 - Implementing Generic Exception Handling for all Resources

- Implementing Exception Handling - 404 Resource Not Found

/src/main/java/com/example/rest/webservices/restfulwebservices/exception/CustomizedResponseEntityExceptionHandler.java 생성

```
@ControllerAdvice
@RestController
public class CustomizedResponseEntityExceptionHandler
extends ResponseEntityExceptionHandler  {

  @ExceptionHandler(Exception.class)
  public final ResponseEntity<Object>
handleAllExceptions(Exception ex, WebRequest request) {
    ErrorDetails errorDetails = new ErrorDetails(new
Date(), ex.getMessage(),
        request.getDescription(false));
    return new ResponseEntity(errorDetails,
HttpStatus.INTERNAL_SERVER_ERROR);
  }

   @ExceptionHandler(UserNotFoundException.class)
  public final ResponseEntity<Object>
handleUserNotFoundException(UserNotFoundException ex,
WebRequest request) {
```

Step 10 - Implementing Generic Exception Handling for all Resources

/src/main/java/com/example/rest/webservices/restfulwebservices/exception/ErrorDetails.java 생성

```
public class ErrorDetails {
  private Date timestamp;
  private String message;
  private String  details;

  public ErrorDetails(Date timestamp, String message,
String details) {
    super();
    this.timestamp = timestamp;
    this.message = message;
    this.details = details;
  }

  public Date getTimestamp() {
    return timestamp;
   }

  public String getMessage() {
    return message;
```

Step 10 - Implementing Generic Exception Handling for all Resources

```
@ResponseStatus(HttpStatus.NOT_FOUND) public class
UserNotFoundException extends RuntimeException {
  public UserNotFoundException(String message) {
    super(message);
  }
}
```

Exercise : User Post Resource and Exception Handling

Step 11 - Implementing DELETE Method to delete a User Resource

/src/main/java/com/example/rest/webservices/restfulwebservices/user/UserDaoService.java 수정

```
public User deleteById(int id) {
  Iterator<User> iterator = users.iterator();
  while (iterator.hasNext()) {
    User user = iterator.next();
    if (user.getId() == id) {
      iterator.remove();
      return user;
    }
  }
  return null;
}
```

Step 11 - Implementing DELETE Method to delete a User Resource

/src/main/java/com/example/rest/webservices/restfulwebservices/user/UserResource.java 수정

```
@GetMapping("/users/{id}")
public User retrieveUser(@PathVariable int id) {
  User user = service.findOne(id);

   if(user==null)
    throw new UserNotFoundException("id-"+ id);

  return user;
}
```

Step 11 - Implementing DELETE Method to delete a User Resource

/src/main/java/com/example/rest/webservices/restfulwebservices/user/UserResource.java 수정

```
@DeleteMapping("/users/{id}")
public void deleteUser(@PathVariable int id) {
  User user = service.deleteById(id);

  if(user==null)
    throw new UserNotFoundException("id-"+ id);
}
```

Step 11 - Implementing DELETE Method to delete a User Resource

/src/main/java/com/example/rest/webservices/restfulwebservices/user/UserResource.java 수정

```
  @PostMapping("/users")
  public ResponseEntity<Object> createUser(@RequestBody
User user)
 {
    User savedUser = service.save(user);
    // CREATED
    // /user/{id}     savedUser.getId()

    URI location = ServletUriComponentsBuilder
      .fromCurrentRequest()
      .path("/{id}")
      .buildAndExpand(savedUser.getId()).toUri();

    return ResponseEntity.created(location).build();

  }
```

/src/main/java/com/example/rest/webservices/restfulwebservices/exception/CustomizedResponseEntityExceptionHandler.java 수정

```
@Override
protected ResponseEntity<Object>
handleMethodArgumentNotValid(MethodArgumentNotValidExceptio
n ex,
    HttpHeaders headers, HttpStatus status, WebRequest
request) {
  ErrorDetails errorDetails = new ErrorDetails(new Date(),
"Validation Failed",
      ex.getBindingResult().toString());
  return new ResponseEntity(errorDetails,
HttpStatus.BAD_REQUEST);
}
```

Step 12 - Implementing Validations for RESTful Services

/src/main/java/com/example/rest/webservices/restfulwebservices/user/User.java 수정

```
  @Size(min=2, message="Name should have atleast 2
characters")
  private String name;
  @Past
  private Date birthDate;
```

/src/main/java/com/example/rest/webservices/restfulwebservices/user/UserResource.java 수정

```
public ResponseEntity<Object> createUser(@Valid
@RequestBody User user) {
```

/pom/xml 수정

```
<dependency>
    <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-hateoas</artifactId>
</dependency>
```

/src/main/java/com/example/rest/webservices/restfulwebservices/user/UserResource.java 수정

```
  @GetMapping("/users/{id}")
  public Resource<User> retrieveUser(@PathVariable int id)
{
    User user = service.findOne(id);

    if(user==null)
      throw new UserNotFoundException("id-"+ id);


    //"all-users", SERVER_PATH + "/users"
    //retrieveAllUsers
    Resource<User> resource = new  Resource<User>(user);

    ControllerLinkBuilder linkTo  =    linkTo(methodOn(this.getClass()).retrieveAllUsers());

       resource.add(linkTo.withRel("all-users"));

       //HATEOAS

       return resource;
    }
```

/src/main/java/com/example/rest/webservices/restfulwebservices/user/UserResource.java 수정

```
  //HATEOAS

  @PostMapping("/users")
  public ResponseEntity<Object> createUser(@Valid
@RequestBody User user) {
    User savedUser = service.save(user);
    // CREATED
    // /user/{id}     savedUser.getId()

    URI location = ServletUriComponentsBuilder
      .fromCurrentRequest()
      .path("/{id}")
      .buildAndExpand(savedUser.getId()).toUri();

    return ResponseEntity.created(location).build();

  }
```

/src/main/java/com/example/rest/webservices/restfulwebservices/RestfulWebServicesApplication.java 수정

```
  @Bean
  public LocaleResolver localeResolver() {
    SessionLocaleResolver localeResolver = new
SessionLocaleResolver();
    localeResolver.setDefaultLocale(Locale.US);
    return localeResolver;
  }

  @Bean
  public ResourceBundleMessageSource messageSource()  {
    ResourceBundleMessageSource messageSource = new
ResourceBundleMessageSource();
     messageSource.setBasename("messages");
    return messageSource;
  }
```

/src/main/java/com/example/rest/webservices/restfulwebservices/helloworld/HelloWorldController.java 수정

```
@Autowired private MessageSource messageSource;


@GetMapping(path = "/hello-world-internationalized")
public String helloWorldInternationalized(
    @RequestHeader(name="Accept-Language", required=false)
Locale locale) {
  return messageSource.getMessage("good.morning.message",
null, locale);
}
```

/src/main/resources/messages.properties 생성

```
good.morning.message=Good Morning.
```

/src/main/resources/messages_ko.properties 생성

```
good.morning.message=안녕하세요.
```

/src/main/resources/messages_ja.properties 생성

```
good.morning.message=おはようございます。
```

AcceptHeaderLocaleResolver 사용

```
@SpringBootApplication
public class RestfulWebServicesApplication {

  ....

  @Bean
  public LocaleResolver localeResolver() {
    AcceptHeaderLocaleResolver localeResolver = new
AcceptHeaderLocaleResolver();
    localeResolver.setDefaultLocale(Locale.US);
    return localeResolver;
  }
```

Step 15 - Simplifying Internationalization for RESTful Services

/src/main/java/com/example/rest/webservices/restfulwebservices/hellow orld/HelloWorldController.java

```
@GetMapping(path = "/hello-world-internationalized")
  public String helloWorldInternationalized() {
    return messageSource.getMessage("good.morning.message",
null,
                LocaleContextHolder.getLocale());
  }
```

application.properties에서 MessageSource configuration 사용

```
spring.messages.basename=messages
```

/pom.xml 수정

```
    <dependency>
     <groupId>com.fasterxml.jackson.dataformat</groupId>
     <artifactId>jackson-dataformat-xml</artifactId>
   </dependency>
```

Step 17 - Enhancing Swagger Documentation with Custom Annotations

/pom.xml 수정

```
    <dependency>
      <groupId>io.springfox</groupId>
      <artifactId>springfox-swagger2</artifactId>
      <version>2.4.0</version>
    </dependency>

    <dependency>
      <groupId>io.springfox</groupId>
      <artifactId>springfox-swagger-ui</artifactId>
      <version>2.4.0</version>
    </dependency>
```

Step 17 - Enhancing Swagger Documentation with Custom Annotations

/src/main/java/com/example/rest/webservices/restfulwebservices/SwaggerConfig.java 생성

```
@Configuration
@EnableSwagger2
public class SwaggerConfig {

  public static final Contact DEFAULT_CONTACT = new
Contact(
      "Ranga Karanam", "http://www.in28minutes.com",
"in28minutes@gmail.com");

  public static final ApiInfo DEFAULT_API_INFO = new
ApiInfo(
      "Awesome API Title", "Awesome API Description",
"1.0",
      "urn:tos", DEFAULT_CONTACT,
      "Apache 2.0",
"http://www.apache.org/licenses/LICENSE-2.0");

  private static final Set<String>
DEFAULT_PRODUCES_AND_CONSUMES =
      new HashSet<String>(Arrays.asList("application/json",
            "application/xml"));
```

Step 17 - Enhancing Swagger Documentation with Custom Annotations

/src/main/java/com/example/rest/webservices/restfulwebservices/UserApiDocumentationConfig.java 생성

```
@SwaggerDefinition(
        info = @Info(
                description = "Awesome Resources",
                version = "V12.0.12",
                title = "Awesome Resource API",
                contact = @Contact(
                   name = "Ranga Karanam",
                   email = "ranga.karanam@in28minutes.com",
                   url = "http://www.in28minutes.com"
                ),
                license = @License(
                   name = "Apache  2.0",
                   url =
"http://www.apache.org/licenses/LICENSE-2.0"
                )
        ),
        consumes = {"application/json", "application/xml"},
        produces = {"application/json", "application/xml"},
        schemes = {SwaggerDefinition.Scheme.HTTP,
SwaggerDefinition.Scheme.HTTPS},
        externalDocs = @ExternalDocs(value = "Read This For
Sure", url = "http://in28minutes.com")
```

Step 17 - Enhancing Swagger Documentation with Custom Annotations

/src/main/java/com/example/rest/webservices/restfulwebservices/user/User.java 수정

```
@ApiModel(description="All details about the user. ")
public class User  {

  private Integer id;

  @Size(min=2, message="Name should have atleast 2
characters")
  @ApiModelProperty(notes="Name should have atleast 2
characters")
  private String name;

  @Past
  @ApiModelProperty(notes="Birth date should be in the
past")
  private Date birthDate;
```

/src/main/java/com/example/rest/webservices/restfulwebservices/user/UserResource.java 수정

```
   @GetMapping("/users/{id}")
  public Resource<User> retrieveUser(@PathVariable int id)
{
    User user = service.findOne(id);

    if(user==null)
      throw new UserNotFoundException("id-"+ id);


    //"all-users", SERVER_PATH + "/users"
    //retrieveAllUsers
    Resource<User> resource = new Resource<User>(user);

    ControllerLinkBuilder linkTo =

linkTo(methodOn(this.getClass()).retrieveAllUsers());

    resource.add(linkTo.withRel("all-users"));

    //HATEOAS
```

/pom.xml 수정

```
    <dependency>
       <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>

     <dependency>
      <groupId>org.springframework.data</groupId>
      <artifactId>spring-data-rest-hal-browser</artifactId>
    </dependency>
```

/application.properties 수정

```
management.endpoints.web.exposure.include=*
```

Step 19 - Implementing Static/Dynamic Filtering for RESTful Service

/src/main/java/com/example/rest/webservices/restfulwebservices/UserApiDocumentationConfig.java 삭제
/src/main/java/com/example/rest/webservices/restfulwebservices/filtering/FilteringController.java 생성

```
@RestController public class FilteringController {

  // field1,field2
  @GetMapping("/filtering")
  public MappingJacksonValue retrieveSomeBean() {
    SomeBean someBean = new SomeBean("value1", "value2",
"value3");

    SimpleBeanPropertyFilter filter =
SimpleBeanPropertyFilter.filterOutAllExcept("field1", "field2");
    FilterProvider filters = new
SimpleFilterProvider().addFilter("SomeBeanFilter", filter);

    MappingJacksonValue mapping = new
MappingJacksonValue(someBean);

    mapping.setFilters(filters);

    return mapping;
  }
```

Step 19 - Implementing Static/Dynamic Filtering for RESTful Service

/src/main/java/com/example/rest/webservices/restfulwebservices/filtering/SomeBean.java 생성

```
 @JsonFilter("SomeBeanFilter")
public class SomeBean {

  private String field1;

  private String field2;

  private String field3;

  public SomeBean(String field1, String field2, String
field3) {
    super();
    this.field1 = field1;
    this.field2 =  field2;
    this.field3 = field3;
  }
```

Step 20 - Versioning RESTful Services - Header and Content Negotiation Approach

/src/main/java/com/example/rest/webservices/restfulwebservices/versioning/Name.java 생성

```
public class Name  {
  private String firstName;
  private String lastName;

  public Name() {
  }

  public Name(String firstName, String lastName) {
```

/src/main/java/com/example/rest/webservices/restfulwebservices/versioning/PersonV1.java 생성

```
 public class PersonV1 {
  private String name;

  public PersonV1() {
    super();
  }

  public PersonV1(String name)
```

/src/main/java/com/example/rest/webservices/restfulwebservices/versioning/PersonV2.java 생성

```
public class PersonV2 {
  private Name name;

  public PersonV2() {
    super();
  }

  public PersonV2(Name name)  {
    super();
    this.name = name;
  }
```

Step 20 - Versioning RESTful Services - Header and Content Negotiation Approach

/src/main/java/com/example/rest/webservices/restfulwebservices/versioning/PersonVersioningController.java 생성

```
 @RestController
public class PersonVersioningController {

  @GetMapping("v1/person")
  public PersonV1 personV1() {
    return new PersonV1("Bob Charlie");
  }

  @GetMapping("v2/person")
  public PersonV2 personV2() {
    return new PersonV2(new Name("Bob", "Charlie"));
   }

  @GetMapping(value = "/person/param", params =
"version=1")
  public PersonV1 paramV1()
```

/pom.xml 수정

```
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
```

/application.properties 수정

```
spring.security.filter.dispatcher-types=request
spring.security.user.name=username
spring.security.user.password=password
```

Step 22 - Updating POST and DELETE methods on User Resource to use JPA

/src/main/java/com/example/rest/webservices/restfulwebservices/user/User.java 수정

```
@ApiModel(description="All details about the user. ")

@Entity
public class User {

  @Id
  @GeneratedValue
  private Integer id;
```

Step 22 - Updating POST and DELETE methods on User Resource to use JPA

/src/main/java/com/example/rest/webservices/restfulwebservices/user/UserJPAResource.java 수정

```
@RestController
public class UserJPAResource {

  @Autowired
  private UserDaoService service;

  @Autowired
  private UserRepository  userRepository;

  @GetMapping("/jpa/users")
  public List<User> retrieveAllUsers() {
    return userRepository.findAll();
  }

  @GetMapping("/jpa/users/{id}")
  public Resource<User> retrieveUser(@PathVariable int id)
{
    Optional<User> user = userRepository.findById(id);

    if(!user.isPresent())
      throw new UserNotFoundException("id-"+  id);

    //"all-users", SERVER_PATH + "/users"
    //retrieveAllUsers
    Resource<User> resource = new
```

```
Resource<User>(user.get());

   ControllerLinkBuilder linkTo =


linkTo(methodOn(this.getClass()).retrieveAllUsers());

   resource.add(linkTo.withRel("all-users"));

   //HATEOAS

   return resource;
 }

 @DeleteMapping("/jpa/users/{id}")
 public void deleteUser(@PathVariable int id) {
   User user =  service.deleteById(id);

   if(user==null)
     throw new UserNotFoundException("id-"+ id);
 }
```

Step 22 - Updating POST and DELETE methods on User Resource to use JPA

/src/main/java/com/example/rest/webservices/restfulwebservices/user/UserRepository.java 생성

```
 @Repository
public interface UserRepository extends JpaRepository<User,
Integer>{


}
```

Step 22 - Updating POST and DELETE methods on User Resource to use JPA

/application.properties 수정

```
management.endpoints.web.exposure.include=* spring.jpa.show-sql=true
spring.h2.console.enabled=true
```

/src/main/resources/data.sql 생성

```
insert into user values(1, sysdate(), 'AB'); insert into user values(2,
sysdate(), 'Jill'); insert into user values(3, sysdate(), 'Jam');
```

Step 23 - Implementing a GET/POST service to create a Post for a User

/src/main/java/com/example/rest/webservices/restfulwebservices/user/Post.java 생성

```
@Entity
public class Post {

  @Id
  @GeneratedValue
  private Integer id;
  private String description;

  @ManyToOne(fetch=FetchType.LAZY)
  @JsonIgnore
  private User  user;

  public Integer getId() {
    return
```

Step 23 - Implementing a GET/POST service to create a Post for a User

/src/main/java/com/example/rest/webservices/restfulwebservices/user/PostRepository.java 생성

```
@Repository

public interface PostRepository extends JpaRepository<Post,
Integer>{


}
```

/src/main/java/com/example/rest/webservices/restfulwebservices/user/User.java 수정

```
  @OneToMany(mappedBy="user")
 private List<Post> posts;
```

Step 23 - Implementing a GET/POST service to create a Post for a User

/src/main/java/com/example/rest/webservices/restfulwebservices/user/UserJPAResource.java 수정

```
@RestController
public class UserJPAResource {

  @Autowired
  private UserRepository userRepository;

  @Autowired
  private PostRepository postRepository;

  @GetMapping("/jpa/users")
  public List<User> retrieveAllUsers() {
    return userRepository.findAll();
```

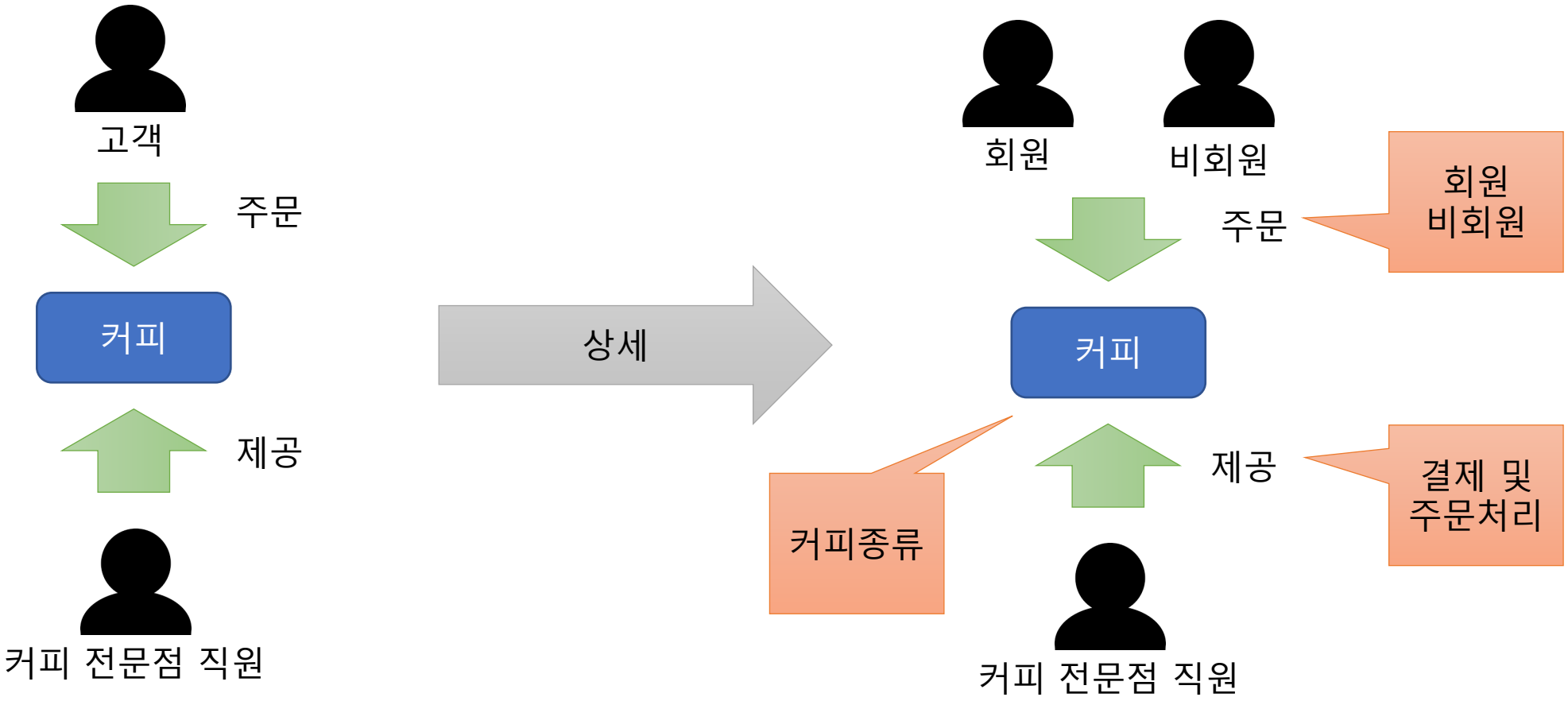
Step 23 - Implementing a GET/POST service to create a Post for a User

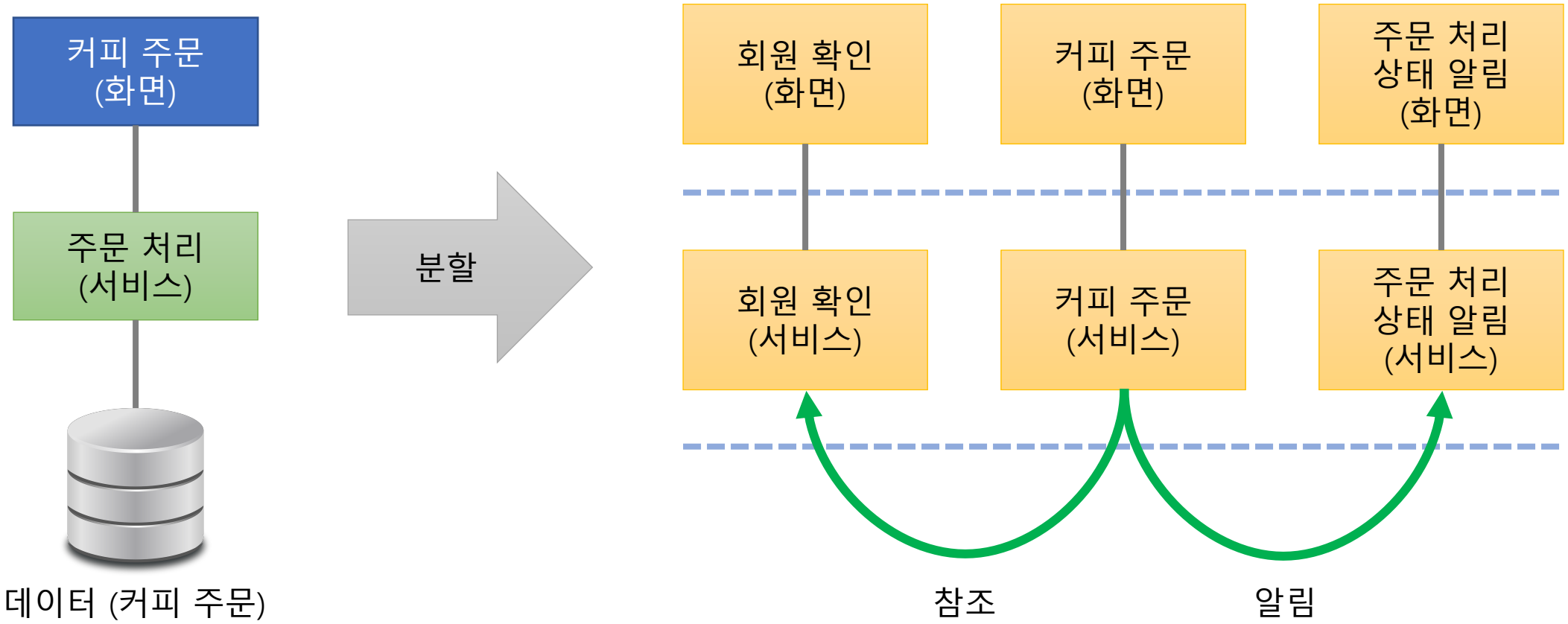
/src/main/resources/data.sql 수정

```
insert into user values(10001, sysdate(), 'AB');
insert into user values(10002, sysdate(), 'Jill');
insert into user values(10003, sysdate(), 'Jam');
insert into post values(11001, 'My First Post', 10001);
insert into post values(11002, 'My Second Post', 10001);
```

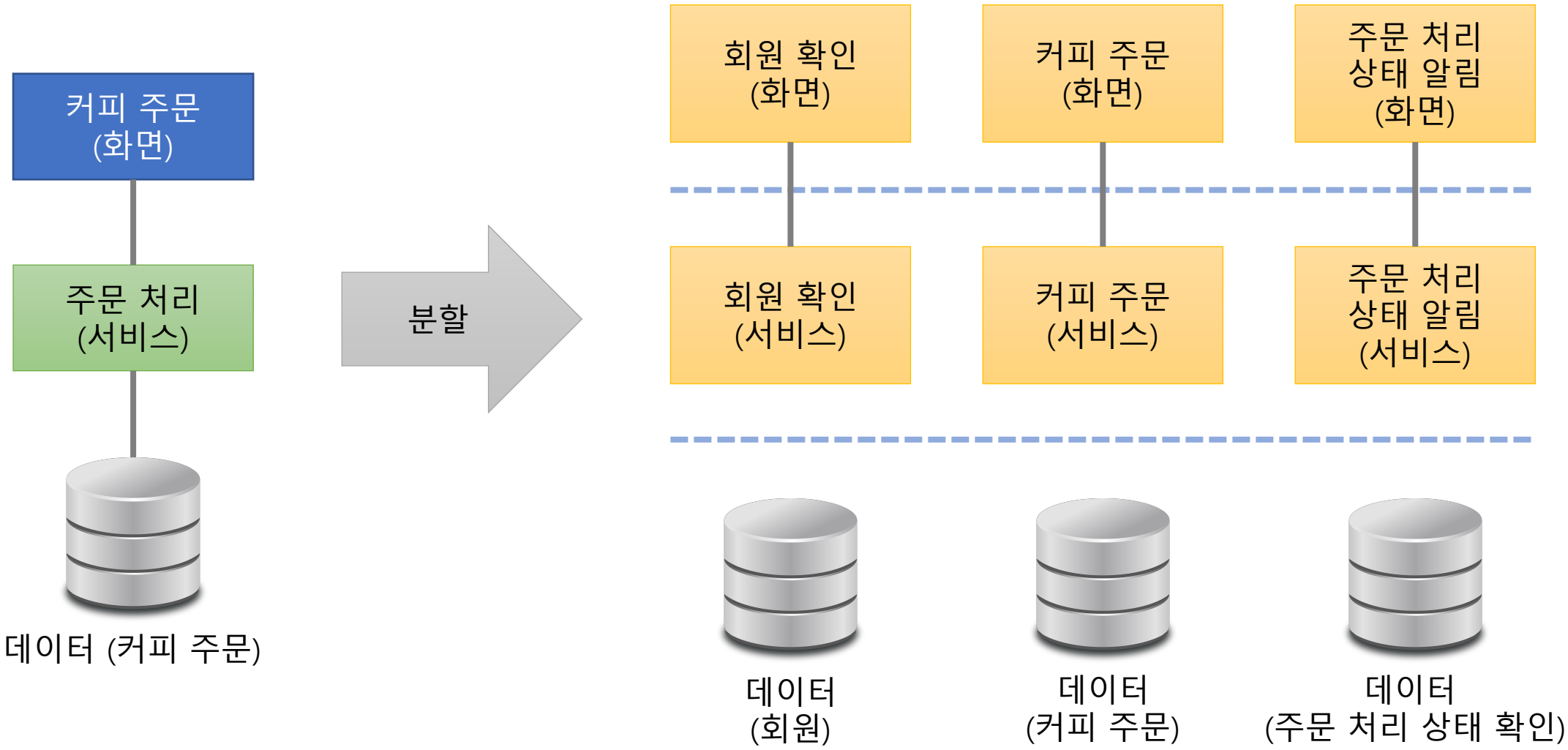
Microservice Design

Service Scenario

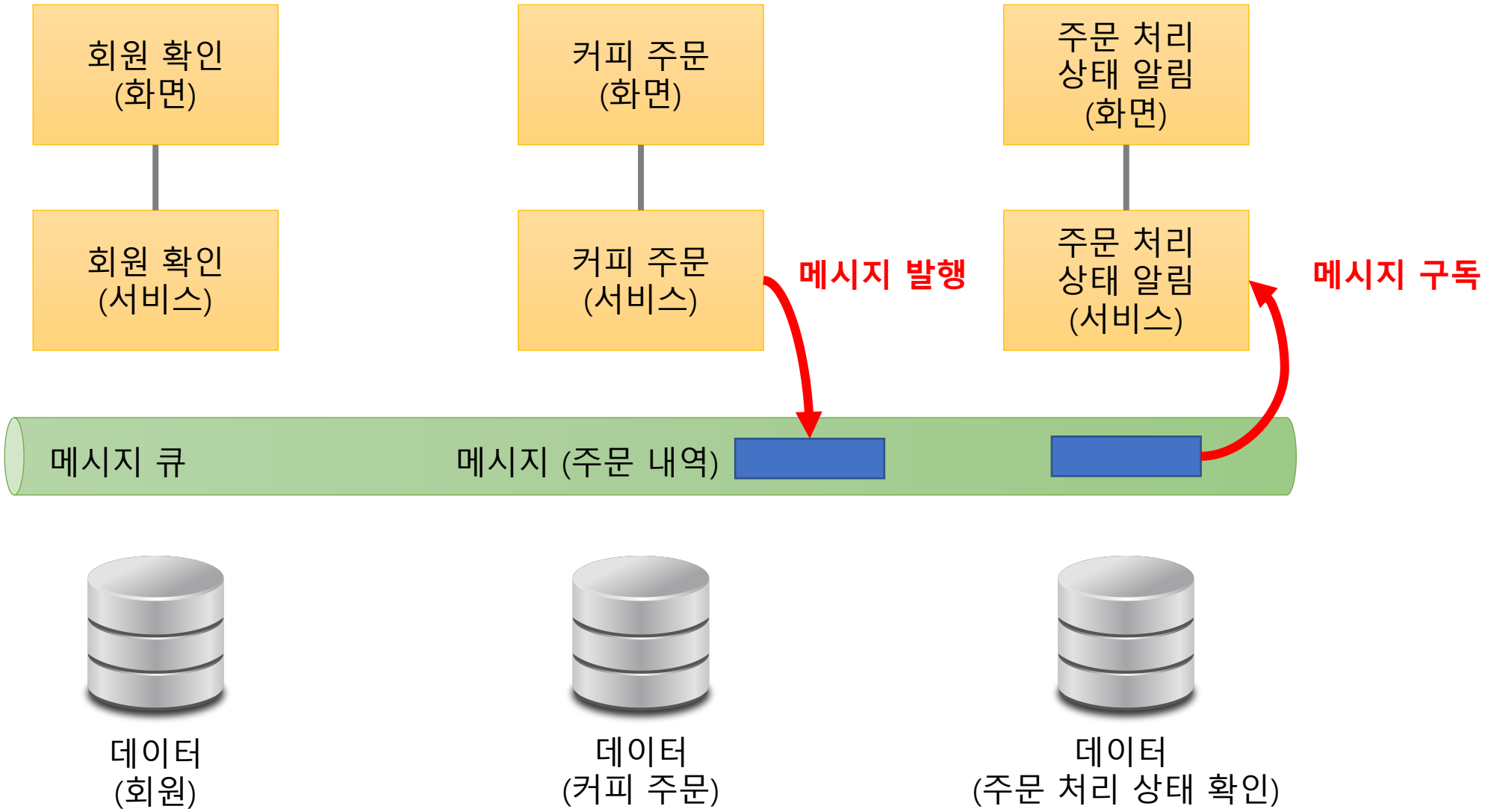



커피 주문
(화면)
주문 처리
(서비스)
데이터 (커피 주문)
분할
회원 확인
(화면)
커피 주문
(화면)
주문 처리
상태 알림
(화면)
회원 확인
(서비스)
커피 주문
(서비스)
주문 처리
상태 알림
(서비스)
참조
알림

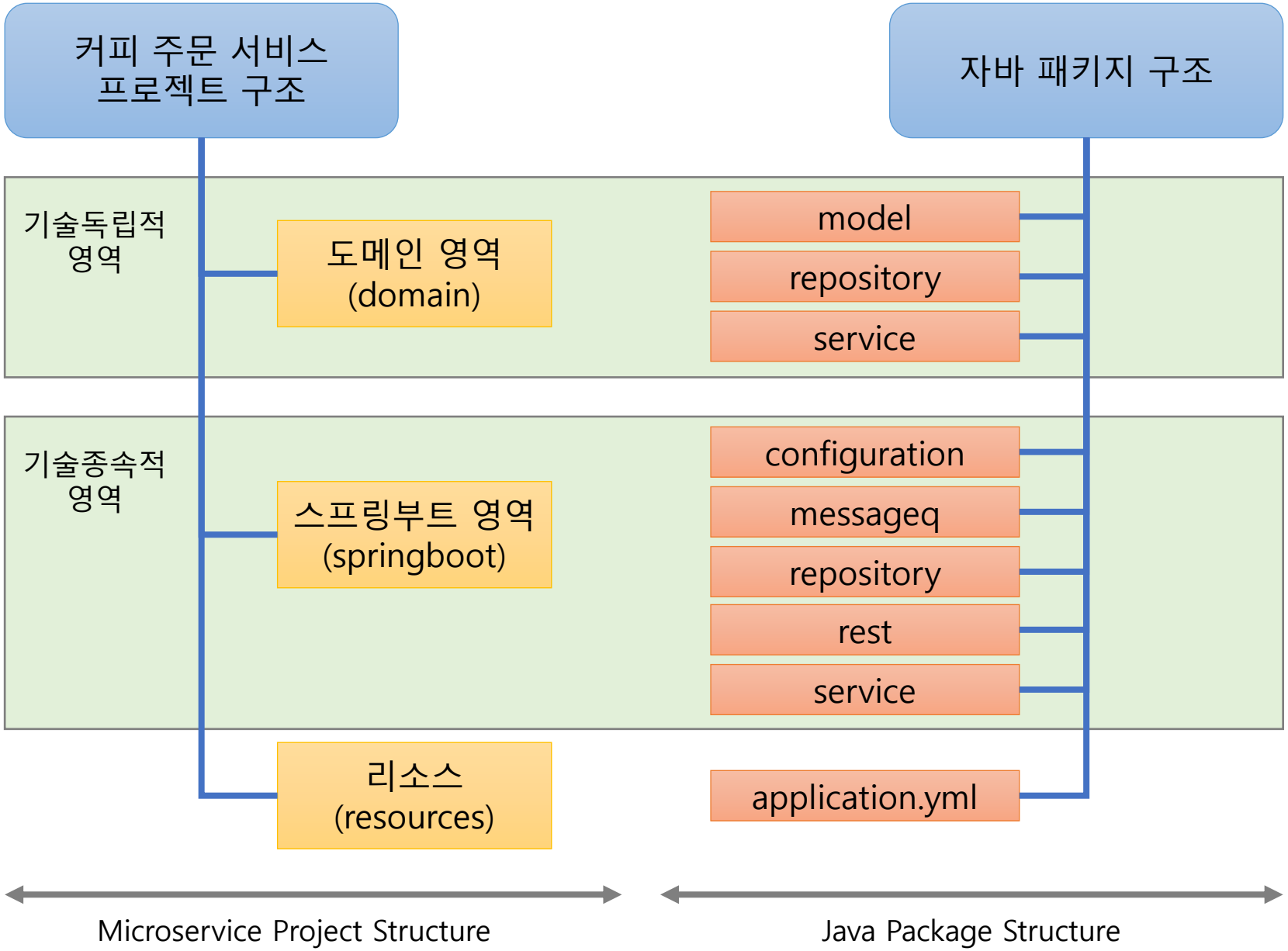
Data Design


커피 주문
(화면)
주문 처리
(서비스)
데이터 (커피 주문)
분할
회원 확인
(화면)
커피 주문
(화면)
주문 처리
상태 알림
(화면)
회원 확인
(서비스)
커피 주문
(서비스)
주문 처리
상태 알림
(서비스)
데이터
(회원)
데이터
(커피 주문)
데이터
(주문 처리 상태 확인)

Message Queue Design

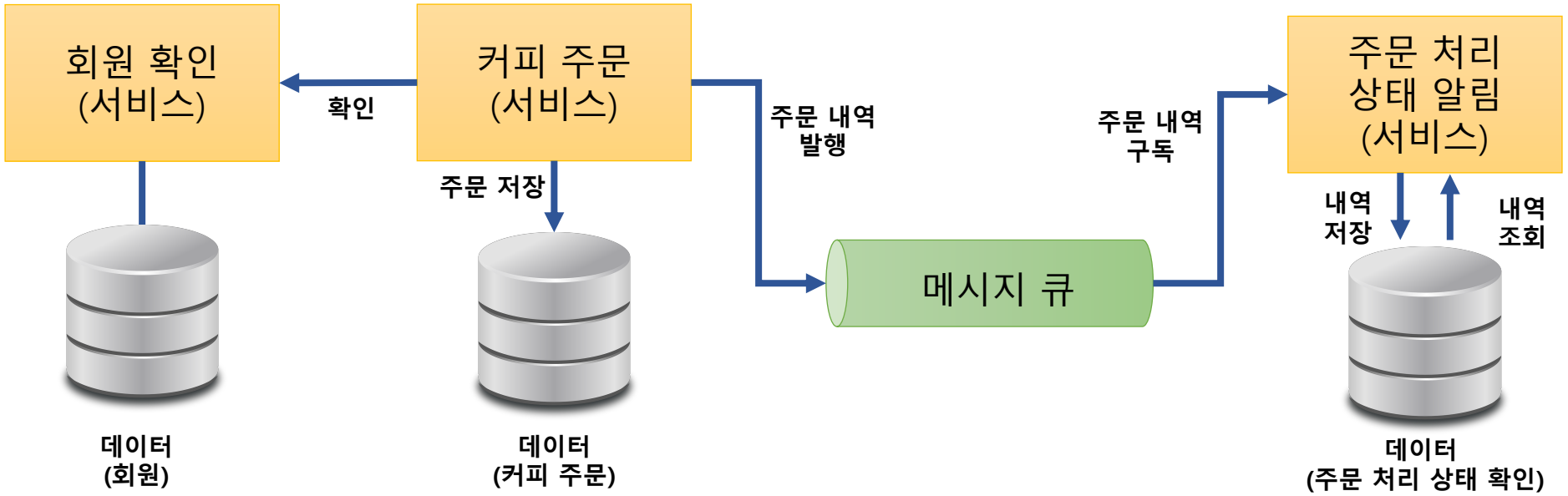


Microservice Package Structure



Project Structure

프로젝트 구분	프로젝트 명	프로젝트 설명
Root	msa-book	Microservice Root Project
Microservice	msa-service-coffee-member	회원 정보 서비스 프로젝트
	msa-service-coffee-order	커피 주문 서비스 프로젝트
	msa-service-coffee-status	주문 처리 상태 확인 서비스 프로젝트

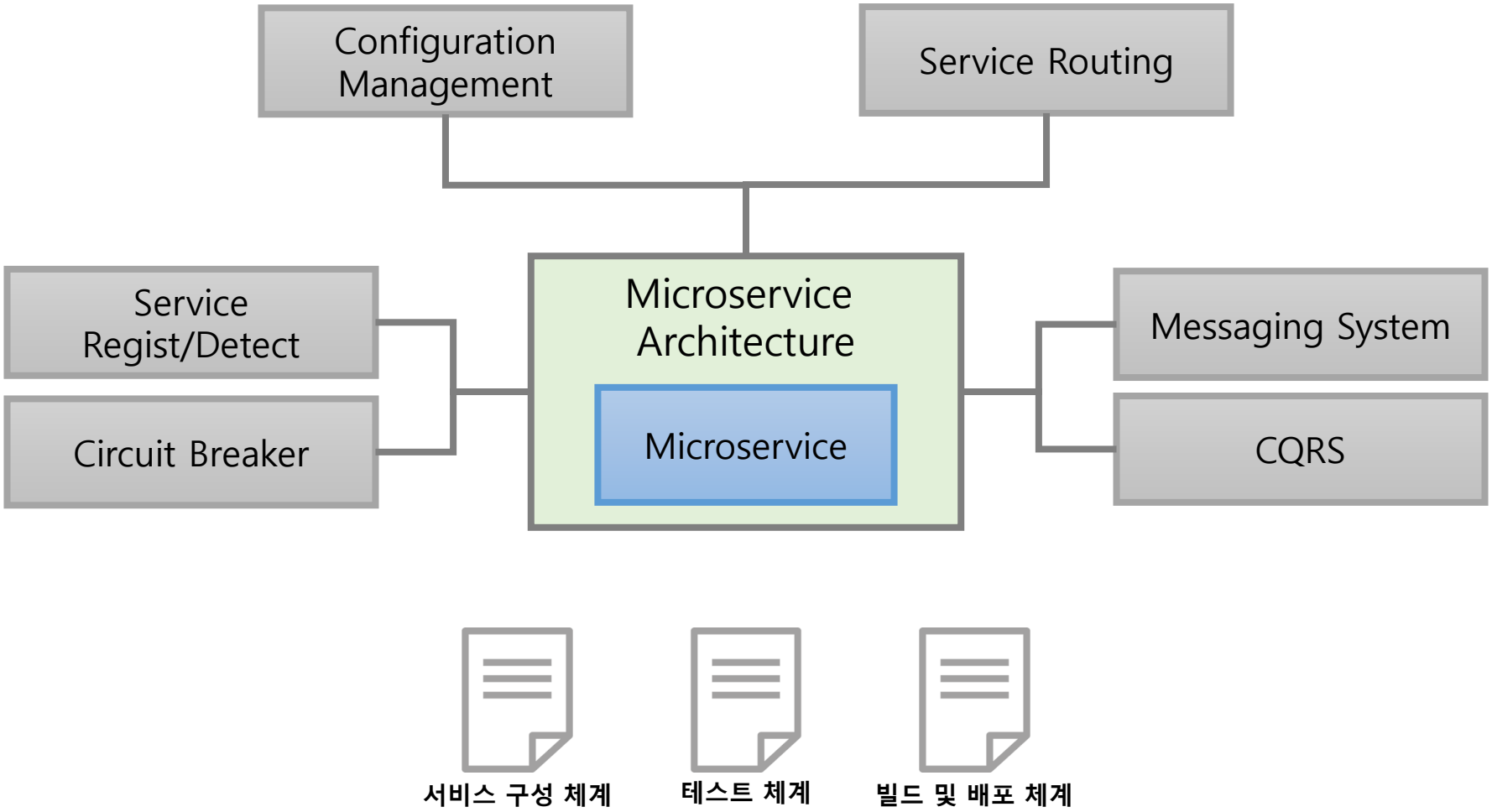


Project Structure

구분	회원 확인 MS	커피 주문 MS	주문 처리 상태 확인 MS
프로젝트 명	msa-service-coffee-member	msa-service-coffee-order	msa-service-coffee-status
개요	회원 가입 유무 확인	커피 주문	주문 내역 알림
주요 기능	회원 정보 관리	커피 주문 및 주문 내역 전송	주문 내역 수식 저장 및 주문 상태 확인 조회
설계	독립된 서비스로 조회	도메인과 기술 영역의 분리 마이크로서비스 간 연계	비동기 방식의 데이터 수신 동기화
패키지	springboot resources	domain springboot resources	springboot resources
데이터 제어	Mybatis	JPA	Mybatis
큐잉 시스템	-	Kafka 메시지 발행	Kafka 메시지 구독

Microservice Architecture Design

Microservice Architecture System

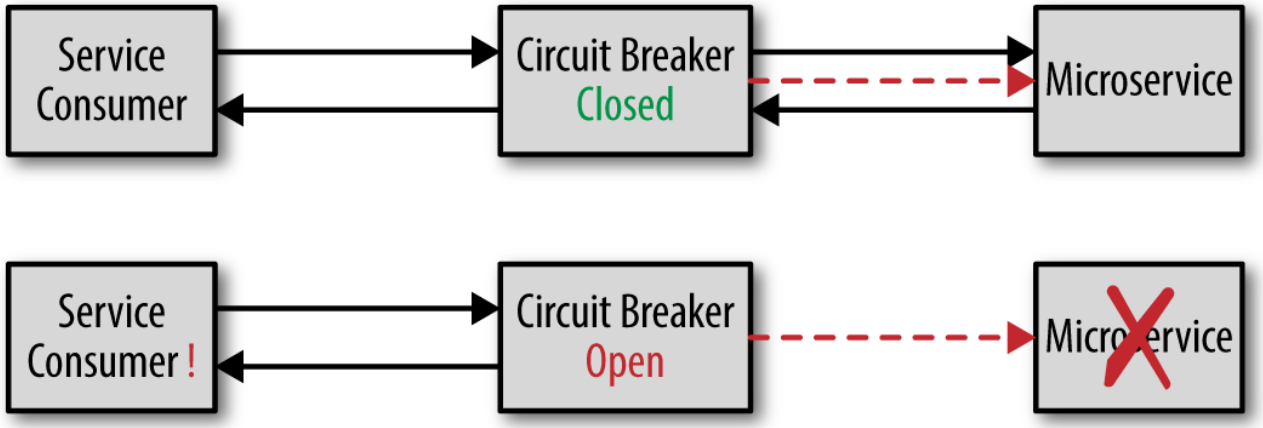


- 환경설정
  - 환경 설정 정보, 변수를 별도의 저장소에 관리
  - 개발 서버, 테스트 서버 등의 시스템 관련 설정 정보 분리
- 서비스 등록 및 감지
  - 마이크로서비스의 등록 삭제 감지
  - 서비스 게이트웨이에서 인지할 수 있도록 지원
- 서비스 게이트웨이
  - 마이크로서비스에 대한 요청을 받아 필요한 서비스로 연결

Microservice Architecture System

- Circuit Breaker
  - 특정 서비스가 정상적으로 동작하지 않을 경우 다른 기능으로 대체 수행 → 장애 회피
  - fallbackMethod
- 큐잉 시스템
  - 마이크로서비스 간 데이터 전달을 처리 (느슨한 결합)
  - Kafka
  - 두 개의 마이크로서비스가 비동기적으로 메시지를 송수신 → 서비스에 대한 부담 최소
- COQS와 이벤트 소싱
  - Command and Query Responsibility Segregation
  - 명령을 처리하는 책임, 조회하는 처리하는 책임 분리
  - 읽기 쓰기 저장소 분리
  - 애플리케이션이 실행될 때 발생되는 모든 이벤트 스트림을 별도 관리 (DB 등)
- Polyglot Programming and Polyglot Persistence
  - 서비스별로 목적과 특성에 맞는 언어와 기술 사용
  - 데이터의 성격과 목적에 맞는 데이터베이스 사용

- 회복성 패턴 - Circuit Breaker
  - 전기 회로의 차단기와 같은 역할 → 문제를 감지하면 모든 시스템과의 연결을 차단
  - 장애가 발생하는 서비스에 반복적인 호출이 되지 못하게 차단



- 회복성 패턴 - Circuit Breaker

```
compile('org.springframework.cloud:spring-cloud-starter-hystrix:1.4.5.RELEASE')
```

```
@EnableCircuitBreaker
@EnableEurekaClient
@SpringBootApplication
public class MicroServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(MicroServiceApplication.class, args);
    }
}
```

```
@HystrixCommand
@RequestMapping(value = "/coffeeOrderStatus", method = RequestMethod.POST)
public ResponseEntity<OrderStatusDVO> coffeeOrderStatus() {
    OrderStatusDVO orderStatusDVO = iCoffeeStatusMapper.selectCoffeeOrderStatus();
    return new ResponseEntity<OrderStatusDVO>(orderStatusDVO, HttpStatus.OK);
}
```

- 회복성 패턴 - Circuit Breaker

```
@HystrixCommand
@RequestMapping(value = "/coffeeOrderStatusWaiting", method = RequestMethod.POST)
public ResponseEntity<OrderStatusDVO> coffeeOrderStatusWaiting() {
    randomlyRunLong();

    OrderStatusDVO orderStatusDVO = iCoffeeStatusMapper.selectCoffeeOrderStatus();

    return new ResponseEntity<OrderStatusDVO>(orderStatusDVO, HttpStatus.OK);
}

private void randomlyRunLong() {
    Random rand = new Random();
    int randomNum = rand.nextInt((3 - 1) + 1) + 1;
    if (randomNum == 3)
        sleep();
}

private void sleep() {
    try {
        Thread.sleep(11000);
    } catch (InterruptedException ex) {
        ex.printStackTrace();
    }
}
```

- 회복성 패턴 - Circuit Breaker

POST http://localhost:9090/coffeeOrder/coffeeOrderWaiting Send

Params Authorization Headers (1) Body Pre-request Script Tests Cookies Code

none form-data x-www-form-urlencoded raw binary JSON (application/json)

```
{
    "id": "",
    "orderNumber": "1",
    "coffeeName": "espresso",
    "coffeeCount": "2",
    "customerName": "kevin"
}
```

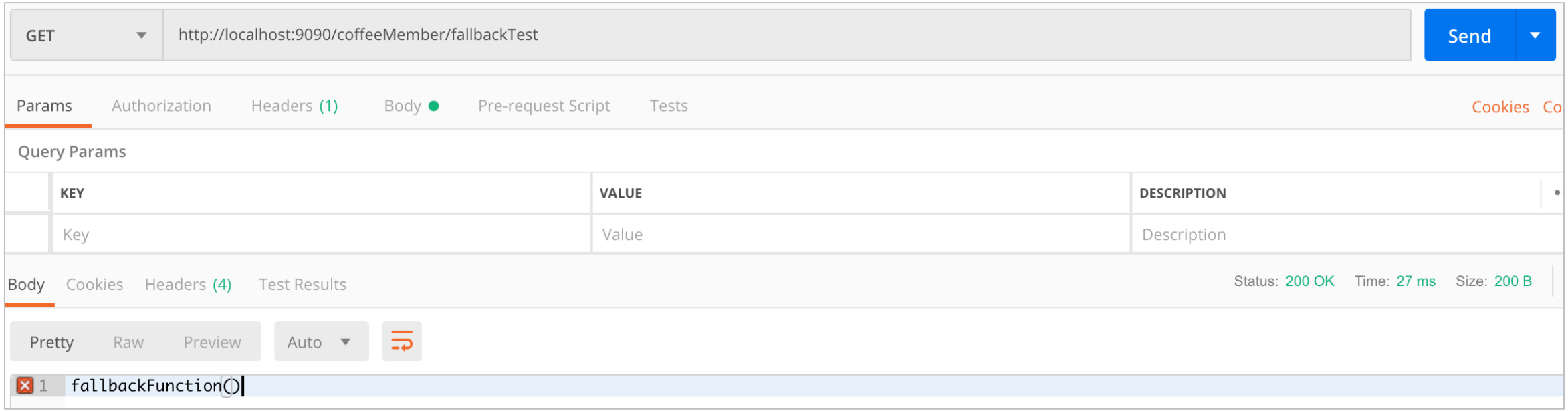
Body Cookies Headers (5) Test Results Status: 500 Internal Server Error Time: 247 ms Size: 373 B

Pretty Raw Preview JSON

```
{
    "timestamp": 1553813897656,
    "status": 500,
    "error": "Internal Server Error",
    "exception": "com.netflix.zuul.exception.ZuulException",
    "message": "GENERAL"
}
```

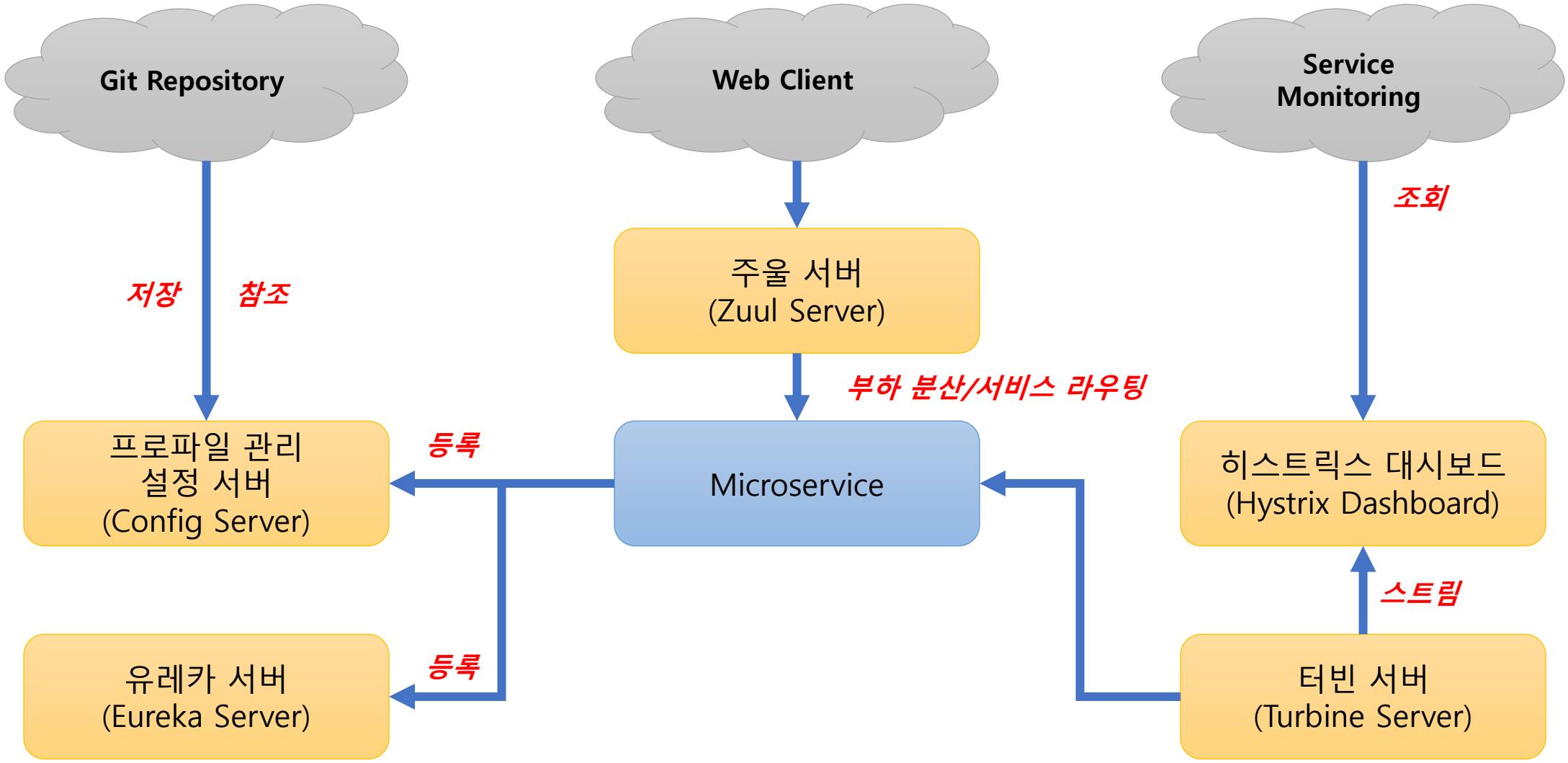
- 회복성 패턴 - fallback

```
@HystrixCommand(fallbackMethod = "fallbackFunction")
@RequestMapping(value = "/fallbackTest", method = RequestMethod.GET)
public String fallbackTest() throws Throwable{
    throw new Throwable("fallbackTest");
}
public String fallbackFunction(){
    return "fallbackFunction()";
}
```



Microservice Architecture Implementation

Spring Cloud Microservice Architecture



구성요소	설명
Git Repository	마이크로서비스 소스 관리 및 프로파일 관리
Config Server	Git 저장소에 등록된 프로파일 연계
Eureka Server	마이크로서비스 등록 및 발견
Zuul Server	마이크로서비스 부하 분산 및 서비스 라우팅
Microservice	커피 주문, 회원 확인, 주문 처리 상태 확인 서비스
Queuing System	마이크로서비스 간 메시지 발행 및 구독
Turbine Server	마이크로서비스의 스트림 데이터 수집
Hystrix Dashboard	마이크로서비스 스트림 데이터 모니터링 및 시각화

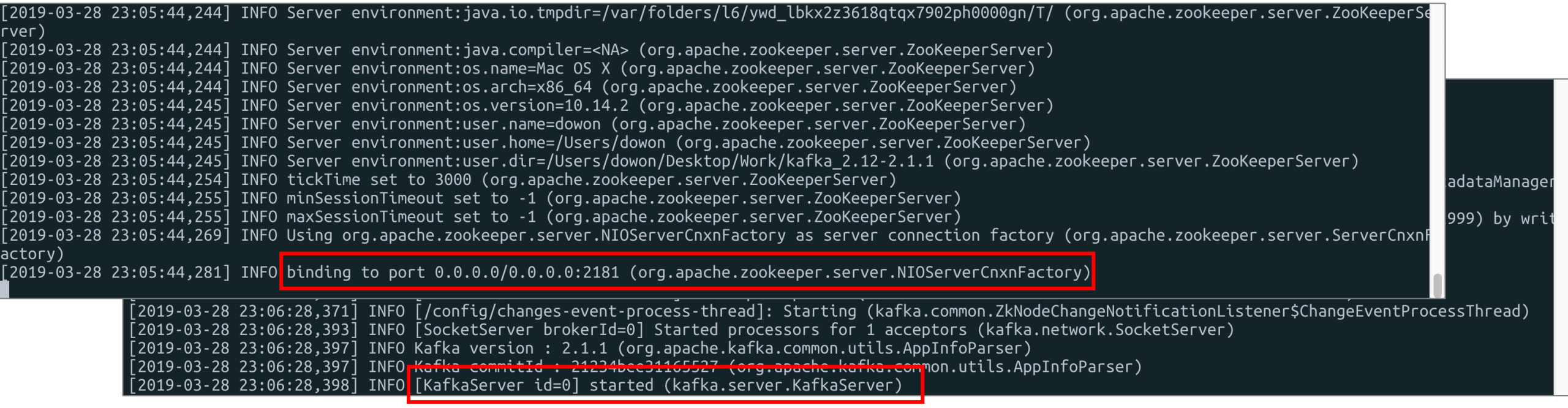
큐잉 시스템 구성

- Kafka 시스템
  - Zookeeper 설치 포함
  - https://kafka.apache.org/downloads
    > tar -xzf ***kafka_2.12-2.1.1.tgz***
    > cd ***kafka_2.12-2.1.1***

```
 dowon@DOWON-MacBook ~/Desktop/Work/kafka_2.12-2.1.1 pwd
/Users/dowon/Desktop/Work/kafka_2.12-2.1.1
 dowon@DOWON-MacBook ~/Desktop/Work/kafka_2.12-2.1.1 ll
total 72
-rw-r--r--@  1 dowon  staff    31K  2  9 03:30 LICENSE
-rw-r--r--@  1 dowon  staff   336B  2  9 03:30 NOTICE
drwxr-xr-x  33 dowon  staff   1.0K  3 26 10:48 bin
drwxr-xr-x  16 dowon  staff   512B  3 26 10:55 config
drwxr-xr-x  85 dowon  staff   2.7K  3 26 10:48 libs
drwxr-xr-x  70 dowon  staff   2.2K  3 28 08:40 logs
drwxr-xr-x   3 dowon  staff    96B  3 26 10:48 site-docs
 dowon@DOWON-MacBook ~/Desktop/Work/kafka_2.12-2.1.1
```

큐잉 시스템 구성

- Kafka 시스템
  - Zookeeper 기동 → Kafka 기동
    - ***bin/zookeeper-server-start.sh config/zookeeper.properties (LINUX, MACOS)***
    - ***bin/windows/zookeeper-server-start.bat config/zookeeper.properties (WINDOWS)***
    - ***bin/kafka-server-start.sh config/server.properties (LINUX, MACOS)***
    - ***bin/windows/kafka-server-start.bat config/server.properties (WINDOWS)***



큐잉 시스템 테스트

- Kafka 시스템
  - Topic 생성
    - ***bin/kafka-topics.sh --create --zookeeper localhost:2181***
      --replication-factor 1 --partitions 1 --topic test20190327
  - Consumer 등록
    - ***bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test20190327***
  - Producer로 메시지 전송
    - ***bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test20190327***

Configuration Server

- application.yml 파일 작성

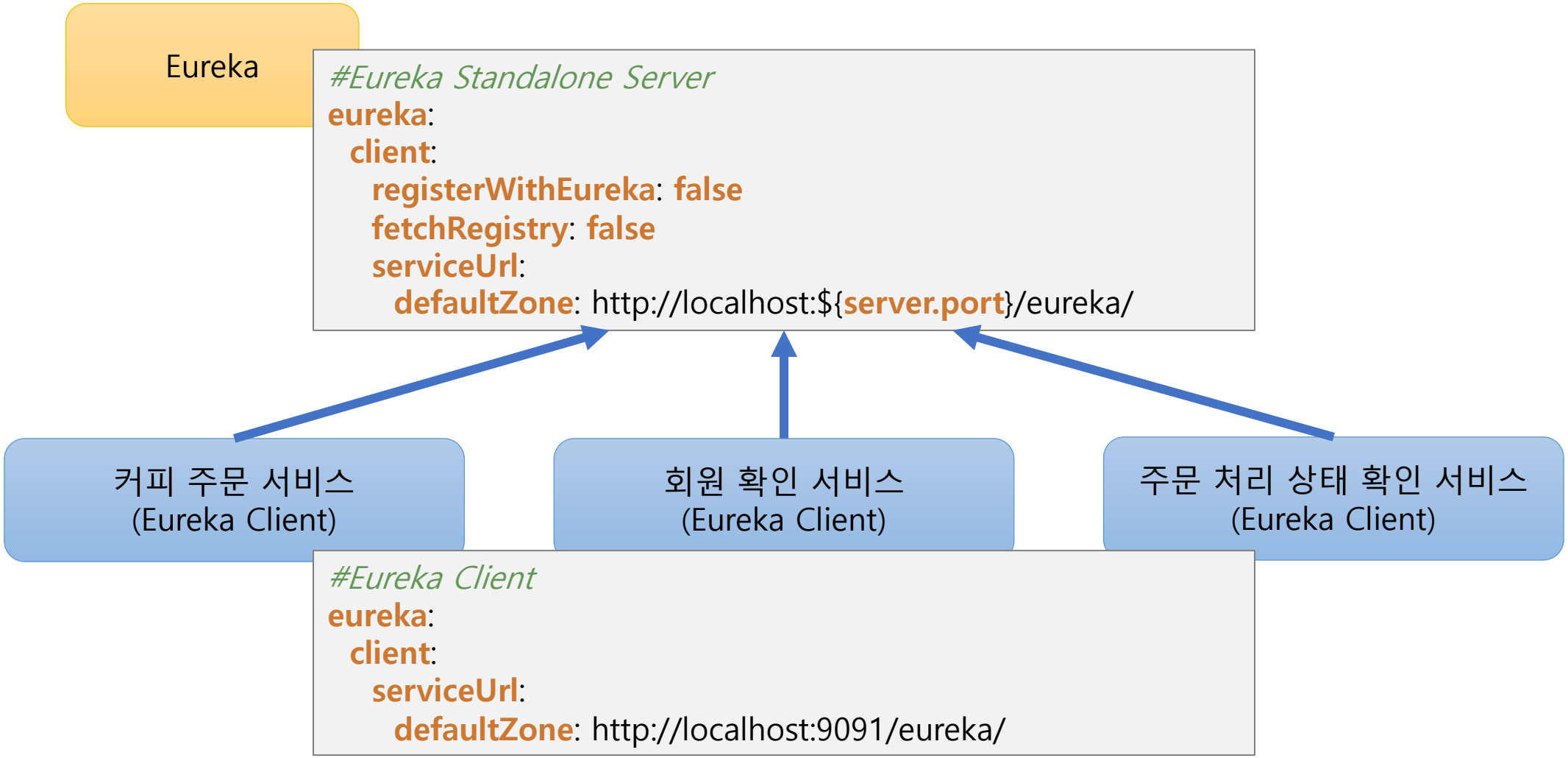
```yaml
server:
  port: 8888

spring:
  application:
    name: msa-architecture-config-server

  cloud:
    config:
      server:
        git:
          uri: https://github.com/joneconsulting/spring-microservice.git
```

- http://<설정 서버 주소>/프로파일명/refresh

Configuration Server

Eureka Server

- *마이크로서비스의 등록과 삭제에 관한 상태 정보를 동적으로 감지*
- *마이크로서비스가 자신의 정보를 유레카 서버에 등록*

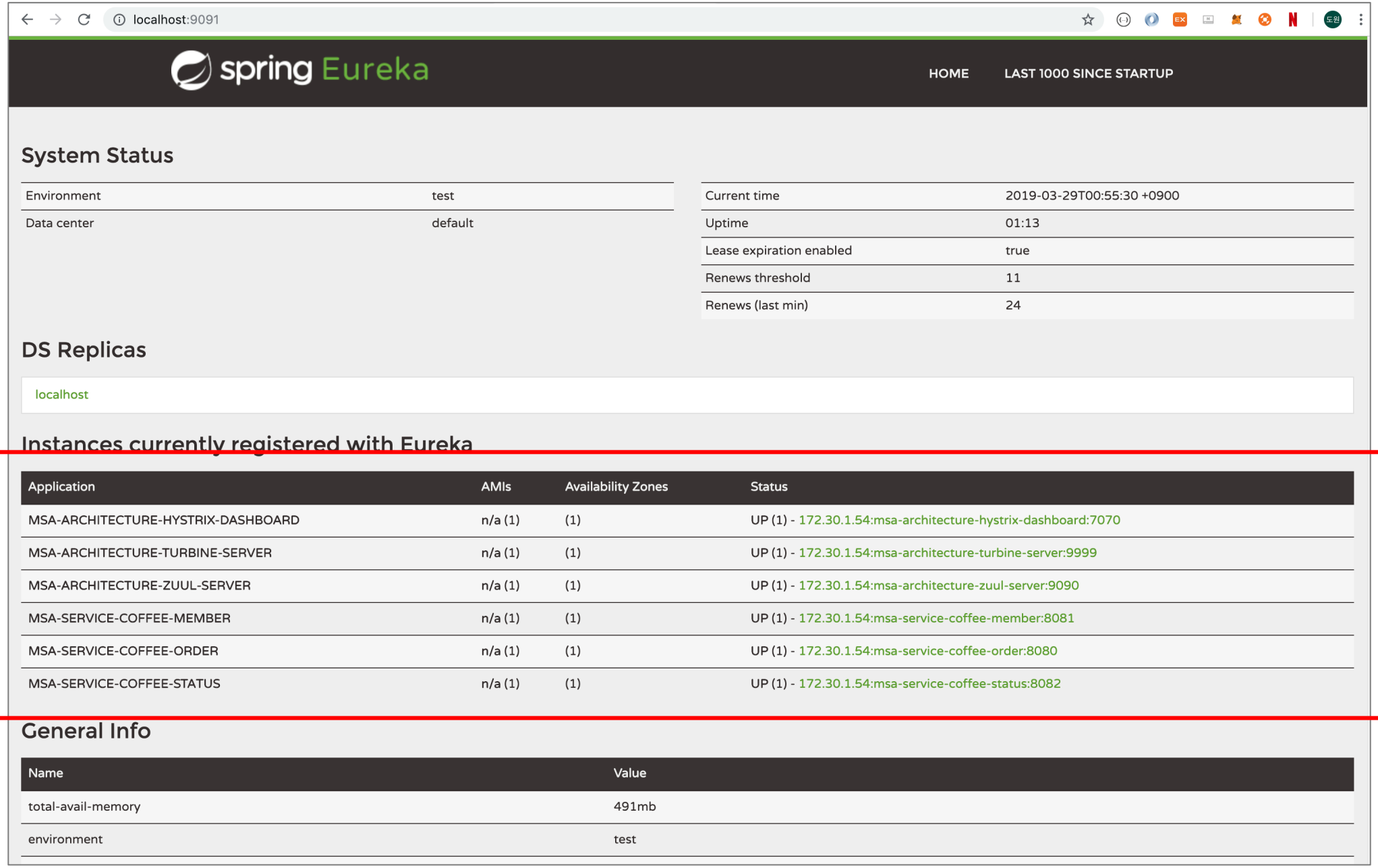


Eureka Server

```yaml
server:
  port: 9091
spring:
  application:
    name: msa-architecture-eureka-server
#Config Server
  cloud:
    config:
      uri: http://localhost:8888
      name: msa-architecture-config-server
#Eureka Standalone Server
eureka:
  client:
    registerWithEureka: false
    fetchRegistry: false
    serviceUrl:
      defaultZone: http://localhost:${server.port}/eureka/
```

```java
@EnableEurekaServer
@SpringBootApplication
public class MsaEurekaServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(MsaEurekaServerApplication.class, args);
    }
}
```

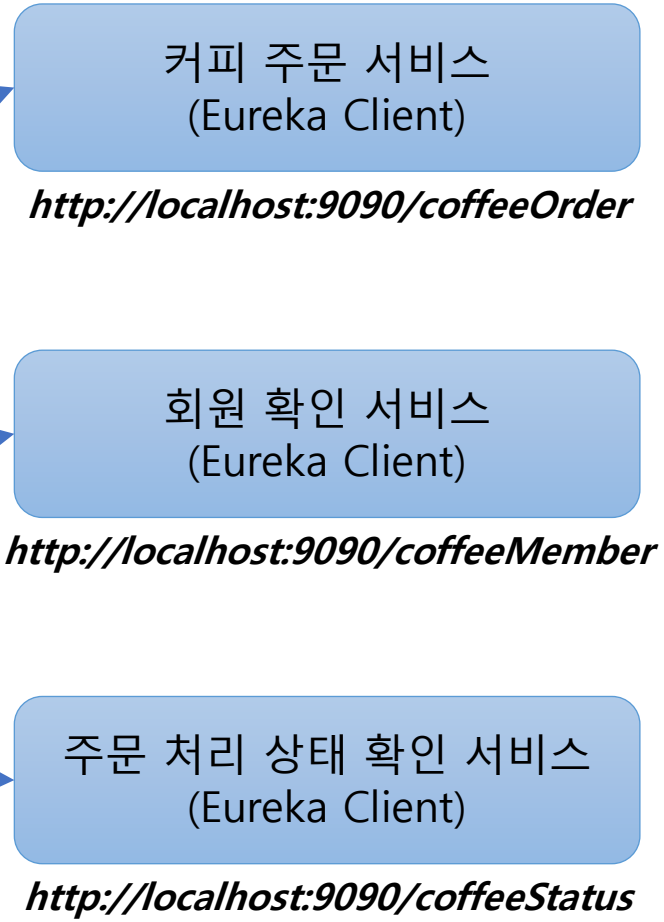
Eureka Server



Zuul Server

- *부하 분산 설정과 서비스 라우팅 기능 수행*
- *서비스 라우팅은 주울 서버에서 설정한 Context Path를 기준으로 MS를 라우팅*

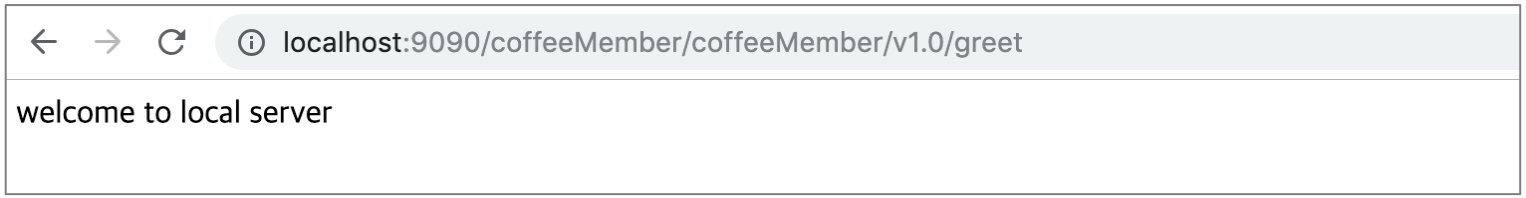
```
#Zuul Routing
zuul:
  routes:
    coffeeOrder: #routing id
      path: /coffeeOrder/** #zuul context root
      serviceId: msa-service-coffee-order #spring application name

    coffeeMember:
      path: /coffeeMember/**
      serviceId: msa-service-coffee-member

    coffeeStatus:
      path: /coffeeStatus/**
      serviceId: msa-service-coffee-status
```



Zuul Server

```yaml
server:
  port: 9090

spring:
  application:
    name: msa-architecture-zuul-server

#Config Server
  cloud:
    config:
      uri: http://localhost:8888
      name: msa-architecture-config-server

#Eureka Client
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:9091/eureka/
```

```java
@EnableZuulProxy
@EnableEurekaClient
@SpringBootApplication
public class MsaZuulApplication {

    public static void main(String[] args) {
        SpringApplication.run(MsaZuulApplication.class, args);
    }

    @Bean
    public SimpleFilter simpleFilter() {
        return new SimpleFilter();
    }
}
```



Turbine Server

- ***마이크로서비스에 설치된 히스트릭스 클라이언트 스트림을 통합***

  (마이크로서비스에서 생성되는 히스트릭스 클라이언트 스트림 메시지를 터빈 서버로 수집)

```
#Turbine Server
turbine:
  appConfig:
            msa-service-coffee-order,
            msa-service-coffee-member,
            msa-service-coffee-status
  clusterNameExpression: new String("default")
```

커피 주문
(coffeeOrder)

```
spring:
  application:
    name: msa-service-coffee-member
```

회원 확인
(coffeeMember)

```
spring:
  application:
    name: msa-service-coffee-order
```

주문 처리 상태 확인
(coffeeStatus)

```
spring:
  application:
    name: msa-service-coffee-status
```

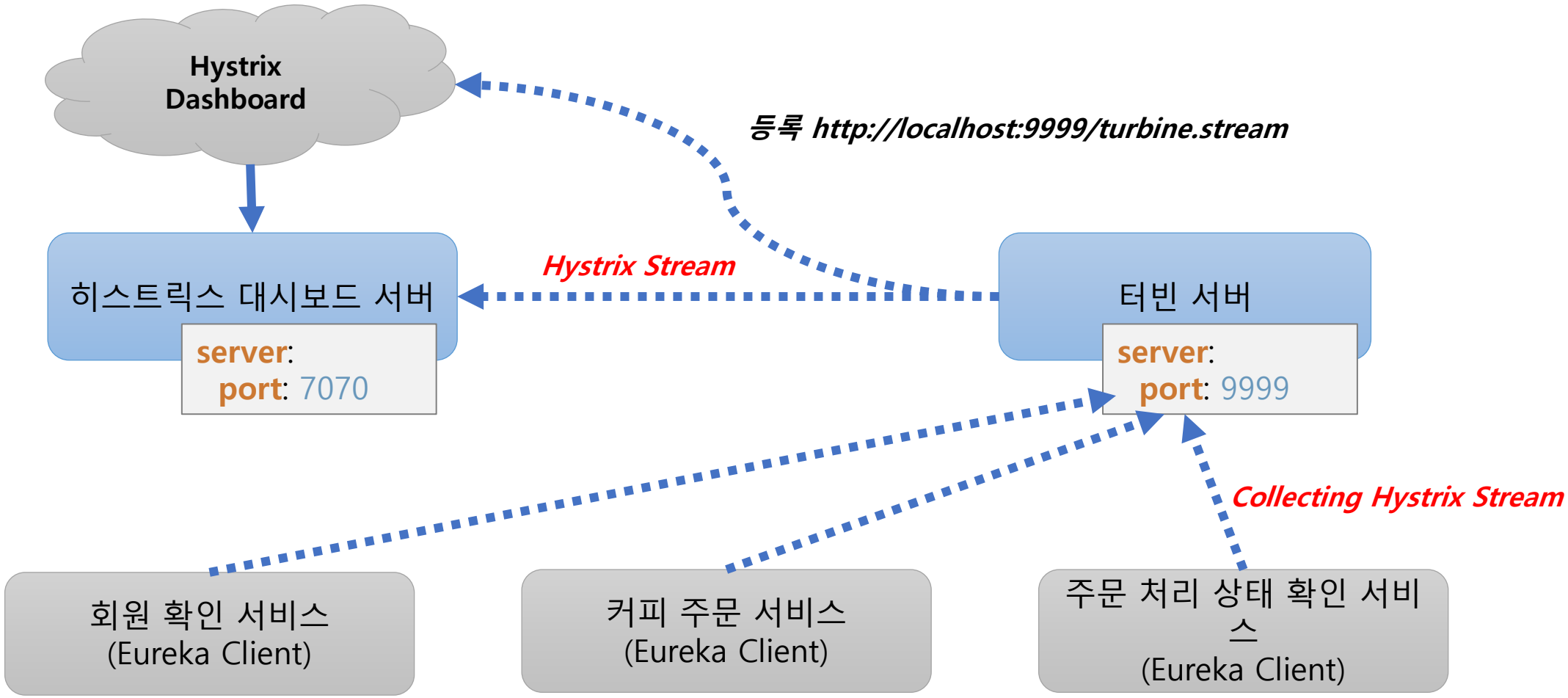
Turbine Server

```
@SpringBootApplication
@EnableEurekaClient
@EnableTurbine
public class MsaTurbineApplication {

    public static void main(String[] args) {
        SpringApplication.run(MsaTurbineApplication.class, args);
    }
}
```

```
@HystrixCommand
@RequestMapping(value = "/coffeeOrder", method = RequestMethod.POST)
public ResponseEntity<CoffeeOrderCVO> coffeeOrder(@RequestBody CoffeeOrderCVO coffeeOrderCVO) {

    //is member
    if(iMsaServiceCoffeeMember.coffeeMember(coffeeOrderCVO.getCustomerName())) {
        System.out.println(coffeeOrderCVO.getCustomerName() + " is a member!");
    }

    //coffee order
    coffeeOrderServiceImpl.coffeeOrder(coffeeOrderCVO);

    //kafka
    kafkaProducer.send("example-kafka-test", coffeeOrderCVO);

    return new ResponseEntity<CoffeeOrderCVO>(coffeeOrderCVO, HttpStatus.OK);
}
```

Hystrix Dashboard

- *히스트릭스 클라이언트에서 생성하는 스트림을 시각화하여 웹화면에 보여주는 대시보드*



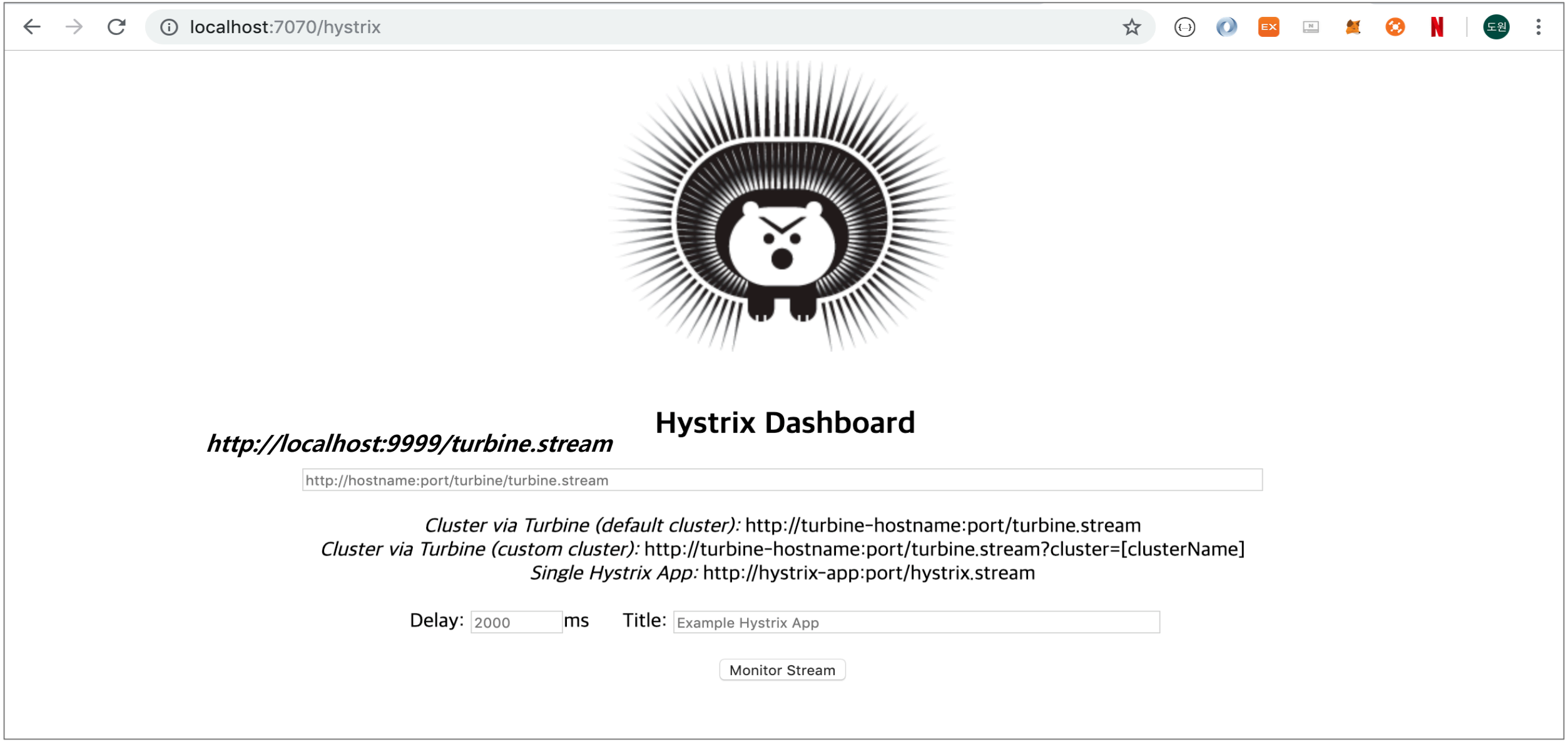
Hystrix Dashboard

- *히스트릭스 클라이언트에서 생성하는 스트림을 시각화하여 웹화면에 보여주는 대시보드*

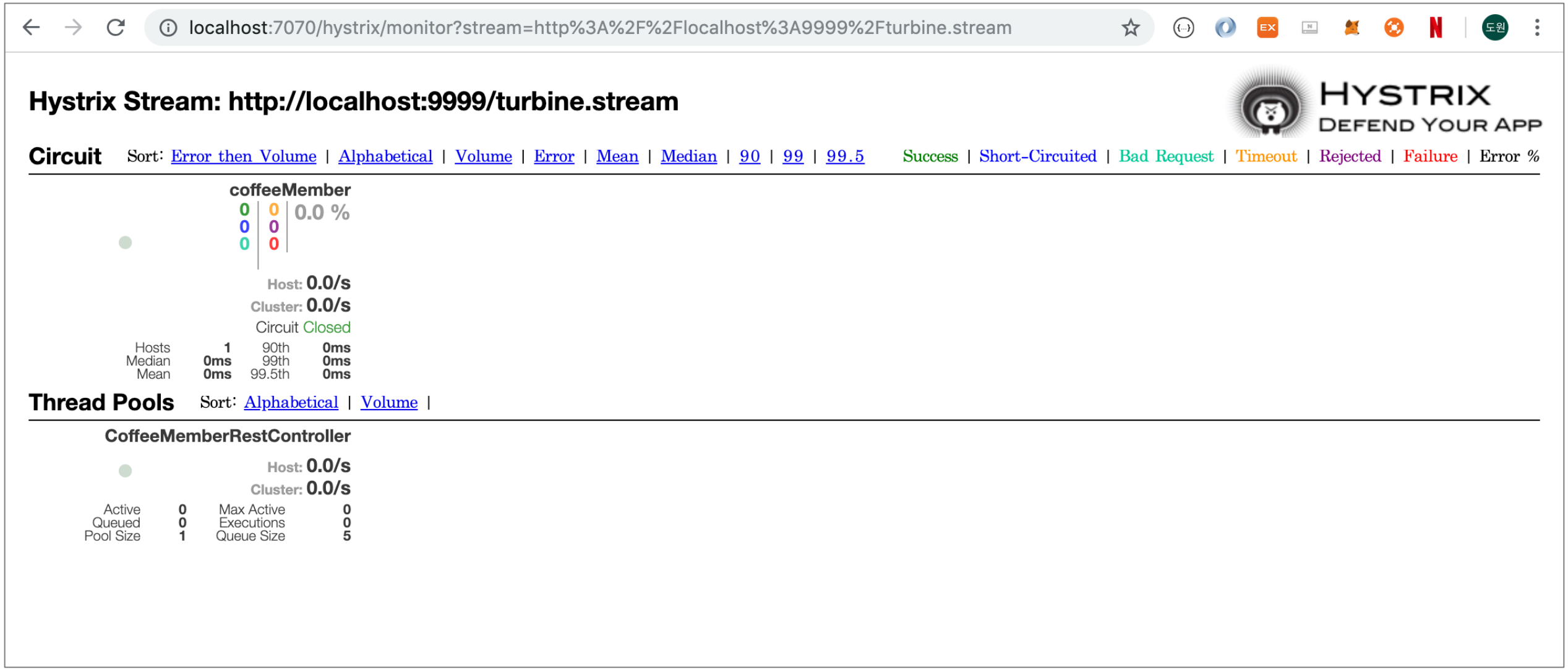
```yaml
server:
  port: 7070

spring:
  application:
    name: msa-architecture-hystrix-dashboard

#Config Server
  cloud:
    config:
      uri: http://localhost:8888
      name: msa-architecture-config-server

#Eureka Client
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:9091/eureka/
```

```java
@EnableHystrixDashboard
@EnableEurekaClient
@SpringBootApplication
public class MsaHystrixDashboardApplication {

    public static void main(String[] args) {
        SpringApplication.run(MsaHystrixDashboardApplication.class, args);
    }
}
```

Hystrix Dashboard



Hystrix Dashboard



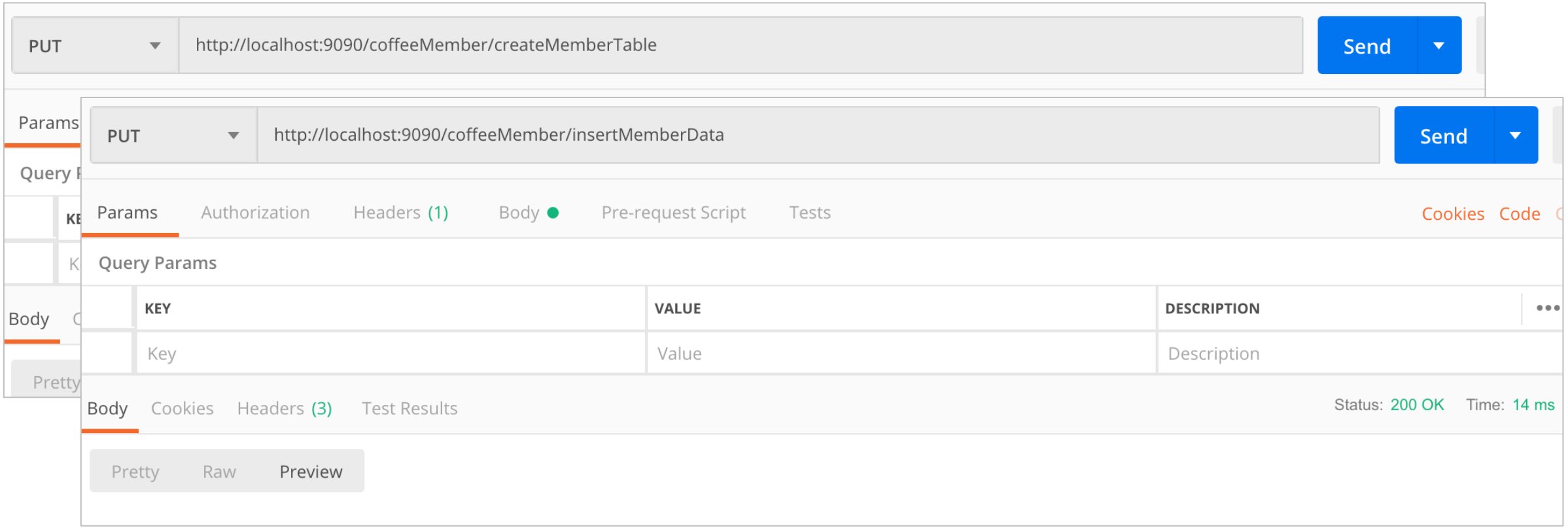
Microservice Architecture Demo

Startup EcoSystem

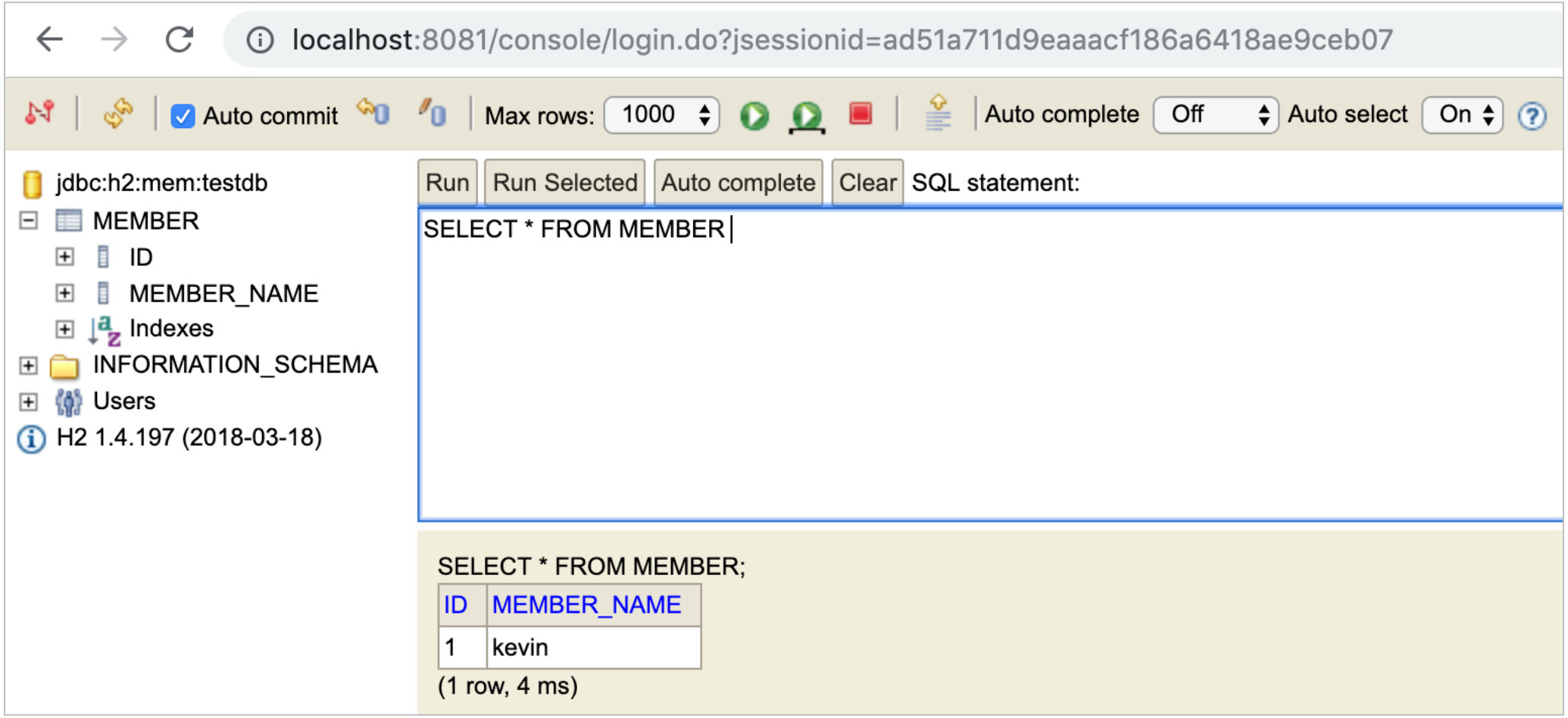
1) Zookeeper & Kafka Server
2) Config Server
3) Eureka Server
4) Zuul Server
5) Turbine Server
6) Hystrix Dashboard
7) Coffee Member Microservice
8) Coffee Order Microservice
9) Coffee Order Status Microservice

```
2019-03-28 23:46:59.018  INFO 22954 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer  : Tomcat started on port(s): 8080 (http)
2019-03-28 23:46:59.018  INFO 22954 --- [           main] .s.c.n.e.s.EurekaAutoServiceRegistration  : Updating port to 8080
2019-03-28 23:46:59.023  INFO 22954 --- [           main] com.example.msa.MicroServiceApplication   : Started MicroServiceApplication in 7.706 seconds (JVM runni
2019-03-28 23:46:59.573  INFO 22954 --- [on(7)-127.0.0.1] c.c.c.ConfigServicePropertySourceLocator : Fetching config from server at: http://localhost:8888

2019-03-28 23:46:04.931  INFO 22908 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8081 (http)
2019-03-28 23:46:04.932  INFO 22908 --- [           main] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 8081
2019-03-28 23:46:04.936  INFO 22908 --- [           main] com.example.msa.MicroServiceApplication  : Started MicroServiceApplication in 8.723 seconds (JVM runni
2019-03-28 23:46:05.501  INFO 22908 --- [on(2)-127.0.0.1] c.c.c.ConfigServicePropertySourceLocator : Fetching config from server at: http://localhost:8888

2019-03-28 23:47:28.802  INFO 22985 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8082 (http)
2019-03-28 23:47:28.803  INFO 22985 --- [           main] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 8082
2019-03-28 23:47:28.807  INFO 22985 --- [           main] com.example.msa.MicroServiceApplication  : Started MicroServiceApplication in 5.958 seconds (JVM runni
2019-03-28 23:47:28.872  INFO 22985 --- [ntainer#0-0-C-1] o.a.k.c.c.internals.AbstractCoordinator  : Discovered coordinator localhost:9092 (id: 2147483647 rack:
2019-03-28 23:47:28.875  INFO 22985 --- [ntainer#0-0-C-1] o.a.k.c.c.internals.ConsumerCoordinator  : Revoking previously assigned partitions [] for group consum
2019-03-28 23:47:28.876  INFO 22985 --- [ntainer#0-0-C-1] o.s.k.l.KafkaMessageListenerContainer    : partitions revoked:[]
2019-03-28 23:47:28.876  INFO 22985 --- [ntainer#0-0-C-1] o.a.k.c.c.internals.AbstractCoordinator  : (Re-)joining group consumerGroupId
2019-03-28 23:47:28.909  INFO 22985 --- [ntainer#0-0-C-1] o.a.k.c.c.internals.AbstractCoordinator  : Successfully joined group consumerGroupId with generation 3
2019-03-28 23:47:28.911  INFO 22985 --- [ntainer#0-0-C-1] o.a.k.c.c.internals.ConsumerCoordinator  : Setting newly assigned partitions [example-kafka-test-0] fo
2019-03-28 23:47:28.918  INFO 22985 --- [ntainer#0-0-C-1] o.s.k.l.KafkaMessageListenerContainer    : partitions assigned:[example-kafka-test-0]
2019-03-28 23:47:29.037  INFO 22985 --- [on(5)-127.0.0.1] c.c.c.ConfigServicePropertySourceLocator : Fetching config from server at: http://localhost:8888
```

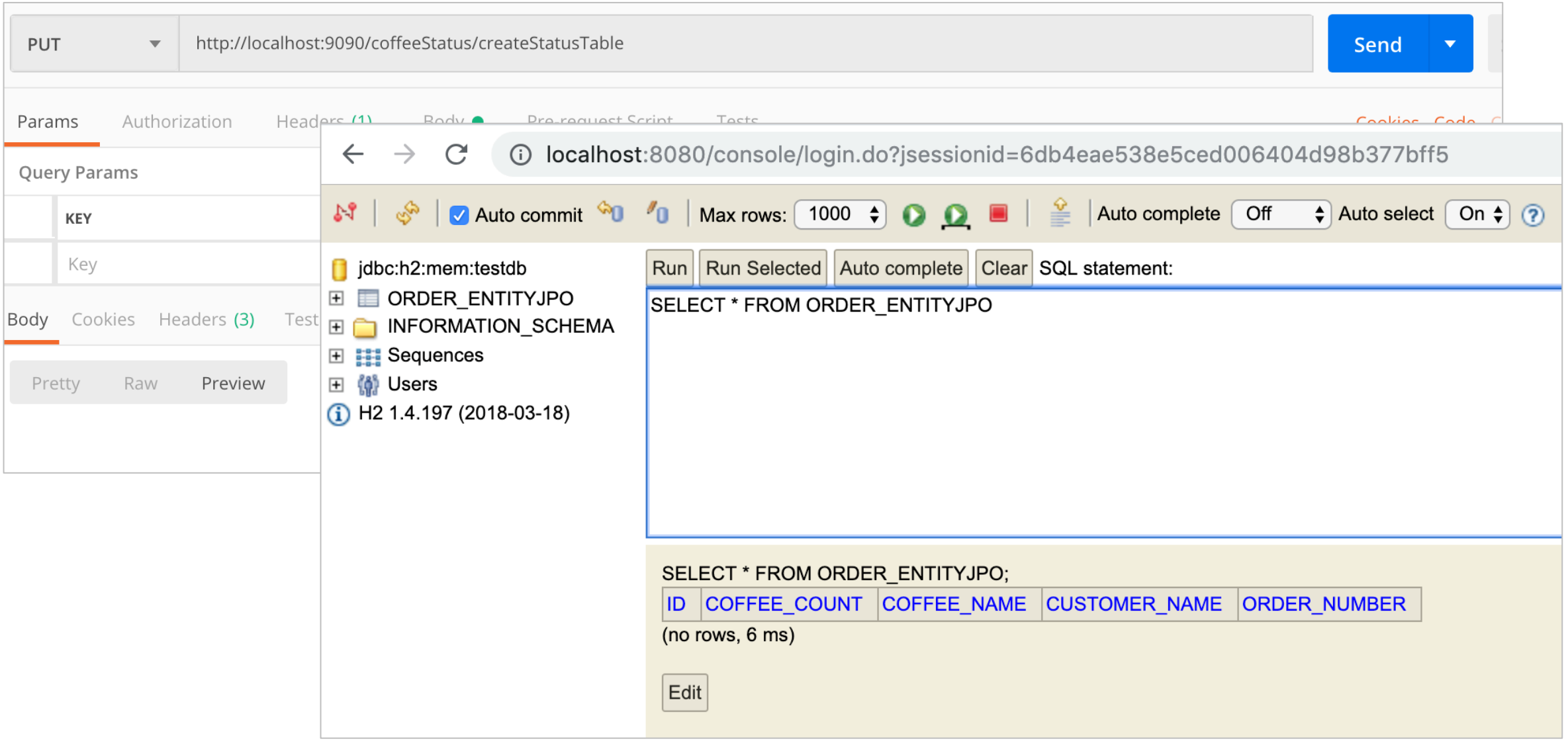
- 회원 테이블 및 데이터 생성



- 회원 데이터 확인


localhost:8081/console/login.do?jsessionid=ad51a711d9eaaacf186a6418ae9ceb07
Auto commit
Max rows: 1000
Auto complete Off
Auto select On
jdbc:h2:mem:testdb
MEMBER
ID
MEMBER_NAME
Indexes
INFORMATION_SCHEMA
Users
H2 1.4.197 (2018-03-18)
Run
Run Selected
Auto complete
Clear
SQL statement:
SELECT * FROM MEMBER
SELECT * FROM MEMBER;
ID
MEMBER_NAME
1
kevin
(1 row, 4 ms)

- 주문 처리 상태 확인 테이블 생성



Test

- 커피 주문

POST http://localhost:9090/coffeeOrder/coffeeOrder Send

Params Authorization Headers (1) Body Pre-request Script Tests Cookies Co

none form-data x-www-form-urlencoded raw binary JSON (application/json)

```
{
    "id": "",
    "orderNumber": "1",
    "coffeeName": "espresso",
    "coffeeCount": "2",
    "customerName": "kevin"
}
```

Body Cookies Headers (4) Test Results Status: 200 OK Time: 827 ms Size: 280 B

Pretty Raw Preview JSON

```
{
    "id": "",
    "orderNumber": "1",
    "coffeeName": "espresso",
    "coffeeCount": "2",
    "customerName": "kevin"
}
```

Test

- 커피 주문
  - 회원 확인 요청

```
2019-03-29 01:40:14.168  INFO 22954 --- [estController-2] c.n.l.DynamicServerListLoadBalancer      : DynamicServerListLoadBalancer for client msa-service-coffee
},Server stats: [[Server:172.30.1.54:8081;  Zone:defaultZone;   Total Requests:0;   Successive connection failure:0;    Total blackout seconds:0;   Last connecti
]}ServerList:org.springframework.cloud.netflix.ribbon.eureka.DomainExtractingServerList@5d06aa4d
kevin is a member!
2019-03-29 01:40:14.369  INFO 22954 --- [estController-2] o.a.k.clients.producer.ProducerConfig    : ProducerConfig values:
    acks = 1
    batch.size = 16384
    bootstrap.servers = [http://localhost:9092
    buffer.memory = 33554432
    client.id =
```

Test

- 커피 주문
  - 주문 정보를 메시지큐에 발행

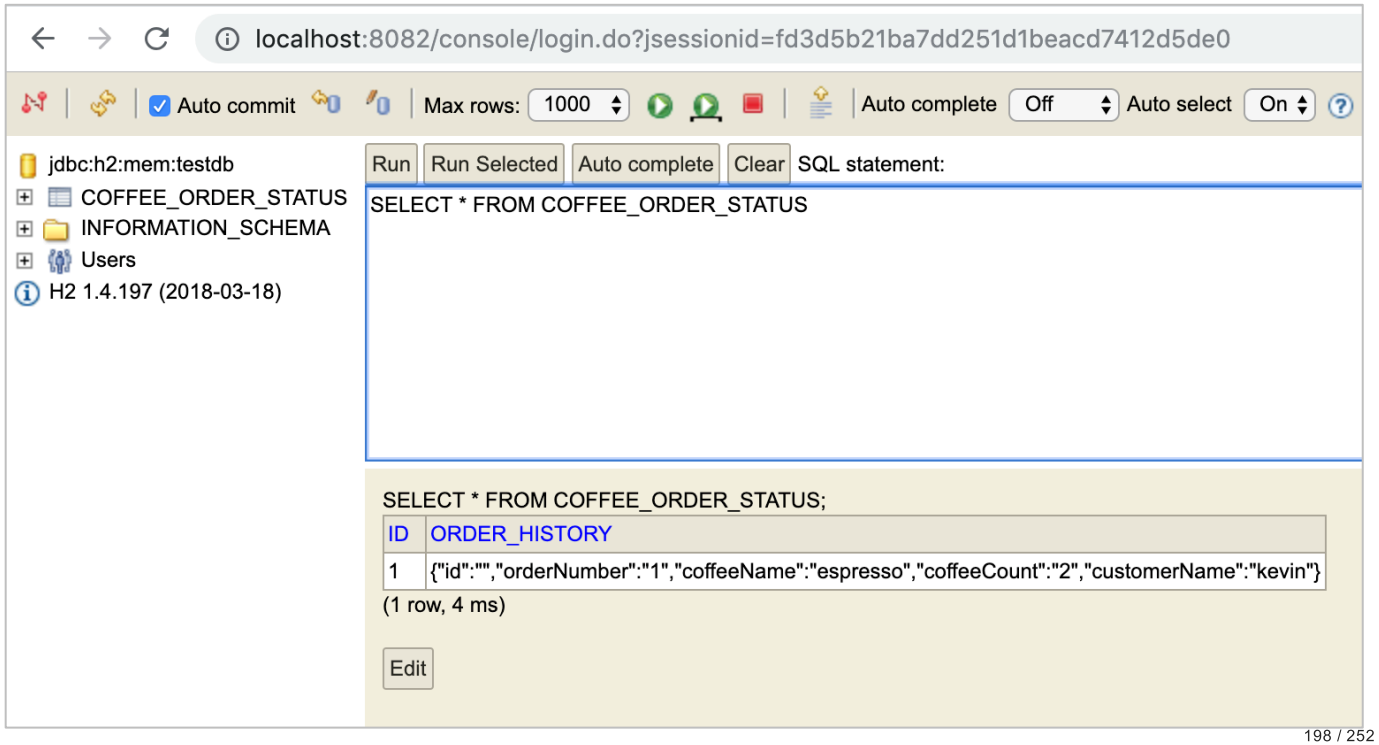
```
    ssl.truststore.location = null
    ssl.truststore.password = null
    ssl.truststore.type = JKS
    transaction.timeout.ms = 60000
    transactional.id = null
    value.serializer = class org.apache.kafka.common.serialization.StringSerializer

2019-03-29 01:40:14.406  INFO 22954 --- [estController-2] o.a.kafka.common.utils.AppInfoParser     : Kafka version : 0.11.0.0
2019-03-29 01:40:14.406  INFO 22954 --- [estController-2] o.a.kafka.common.utils.AppInfoParser     : Kafka commitId : cb8625948210849f
KafkaProducer send data from msa-service-coffee-order: CoffeeOrderCVO(id=, orderNumber=1, coffeeName=espresso, coffeeCount=2, customerName=kevin)
2019-03-29 01:40:15.142  INFO 22954 --- [erListUpdater-0] c.netflix.config.ChainedDynamicProperty  : Flipping property: msa-service-coffee-member.ribbon.ActiveC
2019-03-29 01:41:58.668  INFO 22954 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endpoints via configuration
```

Test

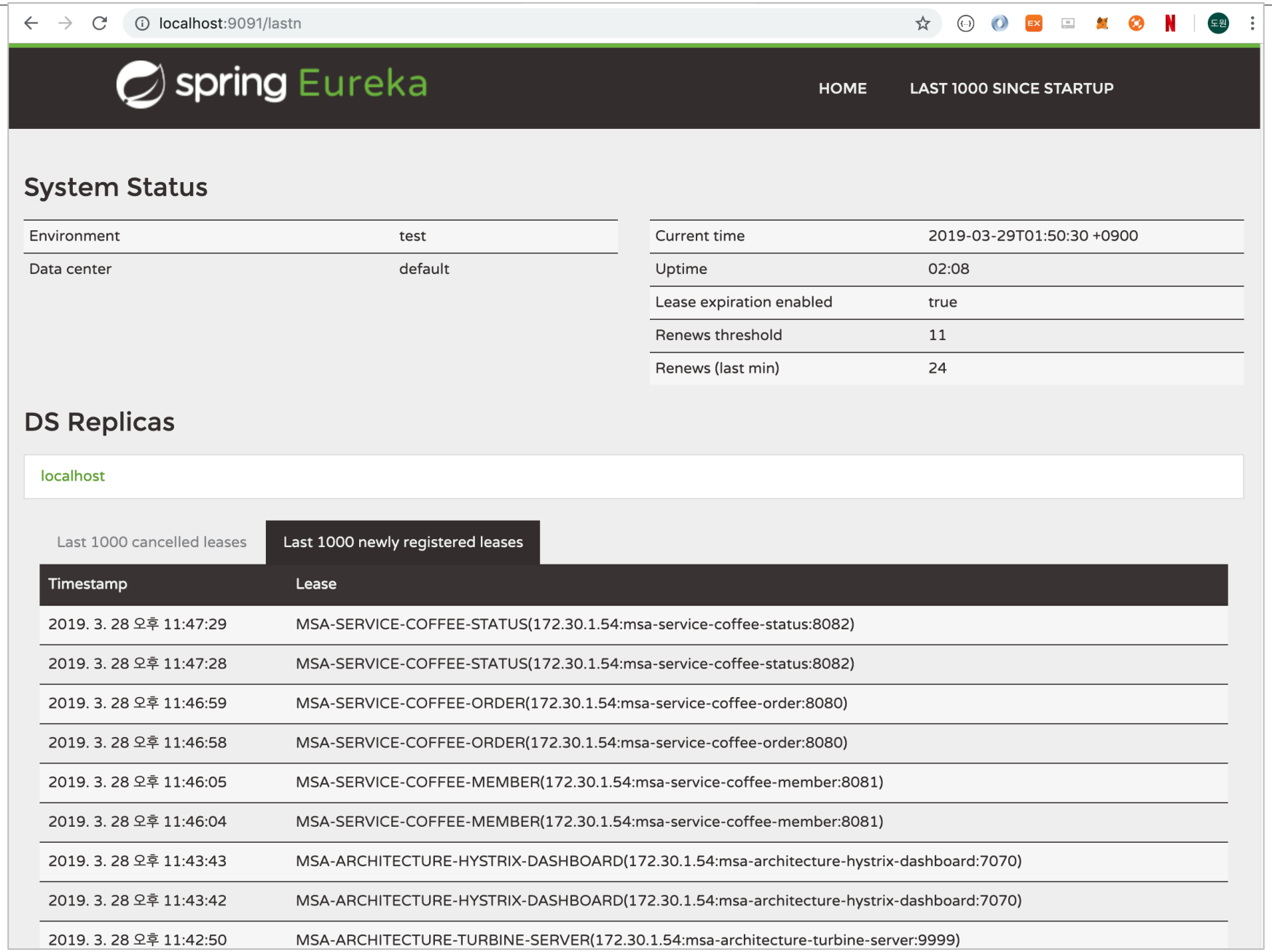
- 주문 처리 상태 확인
  - 커피 주문 마이크로서비스에서 발행한 커피 주문 내역 메시지를 구독

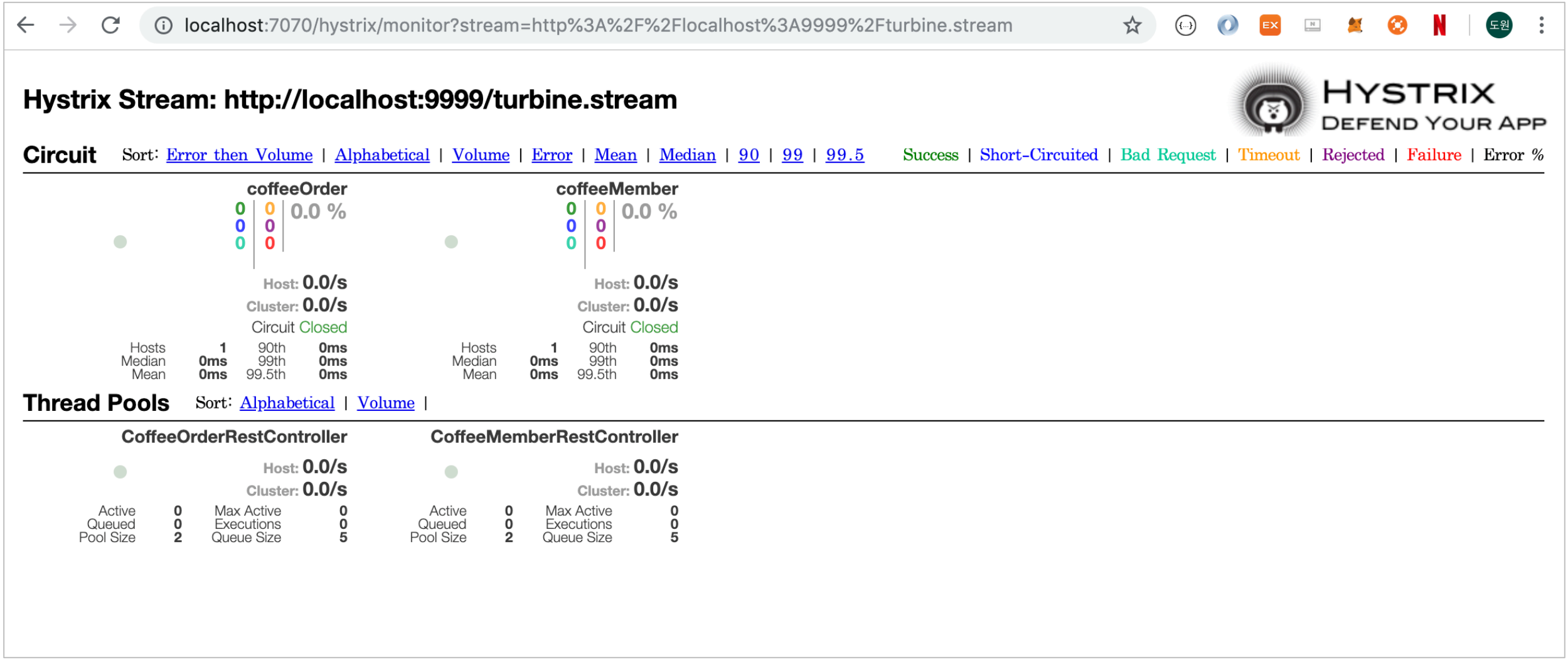
```
2019-03-29 01:32:28.430  INFO 22985 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endpoints via configuration
2019-03-29 01:37:28.423  INFO 22985 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endpoints via configuration
kafkaMessage : =====> {"id":"","orderNumber":"1","coffeeName":"espresso","coffeeCount":"2","customerName":"kevin"}
2019-03-29 01:42:28.417  INFO 22985 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endpoints via configuration
```

- 주문 정보 저장
  - 구독한 주문 정보를 ***"주문 처리 상태 확인"*** 마이크로서비스의 ***DB***에 저장



Inquiry EcoSystem




localhost:7070/hystrix/monitor?stream=http%3A%2F%2Flocalhost%3A9999%2Fturbine.stream
Hystrix Stream: http://localhost:9999/turbine.stream
HYSTRIX
DEFEND YOUR APP
Circuit
Sort: Error then Volume | Alphabetical | Volume | Error | Mean | Median | 90 | 99 | 99.5
Success | Short-Circuited | Bad Request | Timeout | Rejected | Failure | Error %
coffeeOrder
0 0 0.0 %
0 0
0 0
Host: 0.0/s
Cluster: 0.0/s
Circuit Closed
Hosts 1 90th 0ms
Median 0ms 99th 0ms
Mean 0ms 99.5th 0ms
coffeeMember
0 0 0.0 %
0 0
0 0
Host: 0.0/s
Cluster: 0.0/s
Circuit Closed
Hosts 1 90th 0ms
Median 0ms 99th 0ms
Mean 0ms 99.5th 0ms
Thread Pools
Sort: Alphabetical | Volume |
CoffeeOrderRestController
Host: 0.0/s
Cluster: 0.0/s
Active 0 Max Active 0
Queued 0 Executions 0
Pool Size 2 Queue Size 5
CoffeeMemberRestController
Host: 0.0/s
Cluster: 0.0/s
Active 0 Max Active 0
Queued 0 Executions 0
Pool Size 2 Queue Size 5

Security Test

- 회원 마이크로서비스
  - 특정 회원만 사용가능하도록 변경

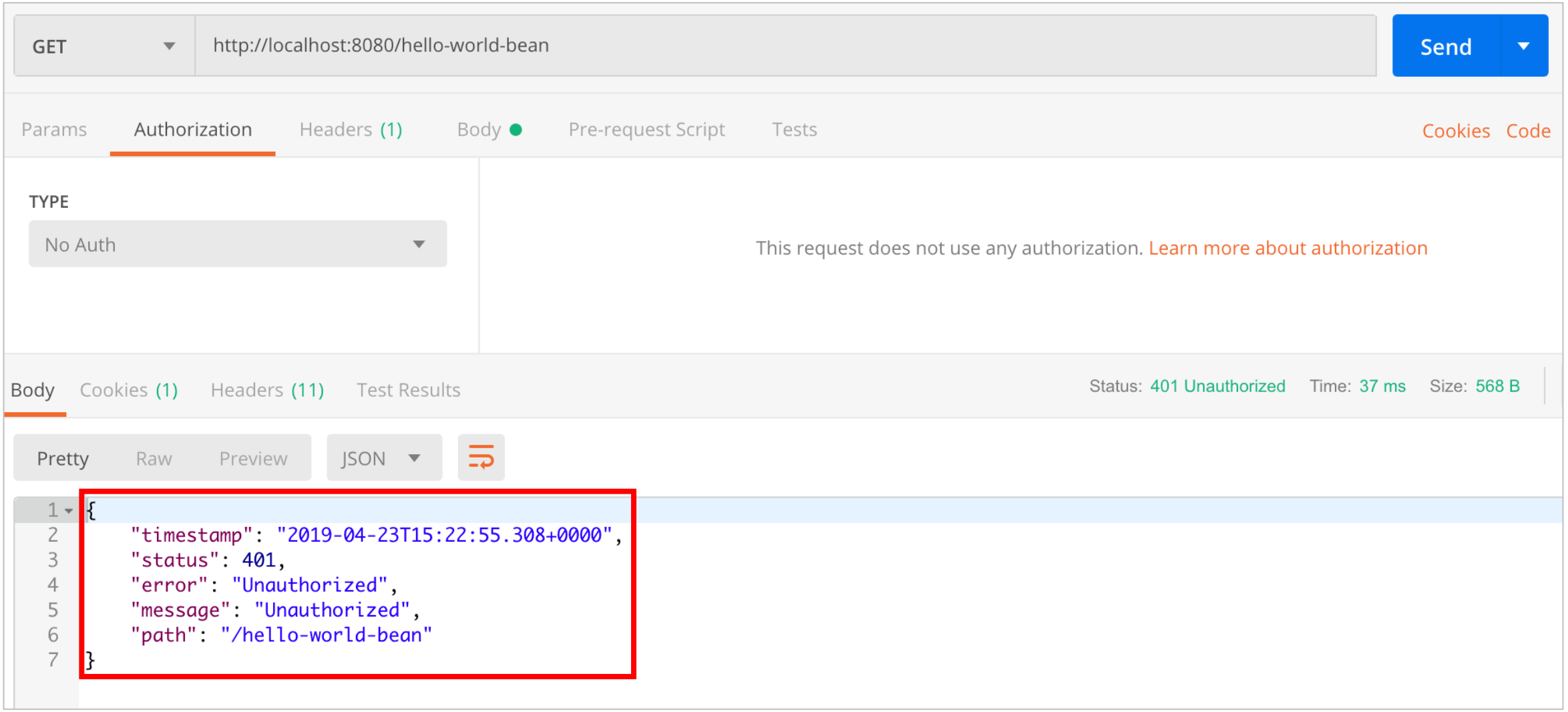
```
security:
  basic:
    enabled: false
  user:
    name: username
    password: password
```

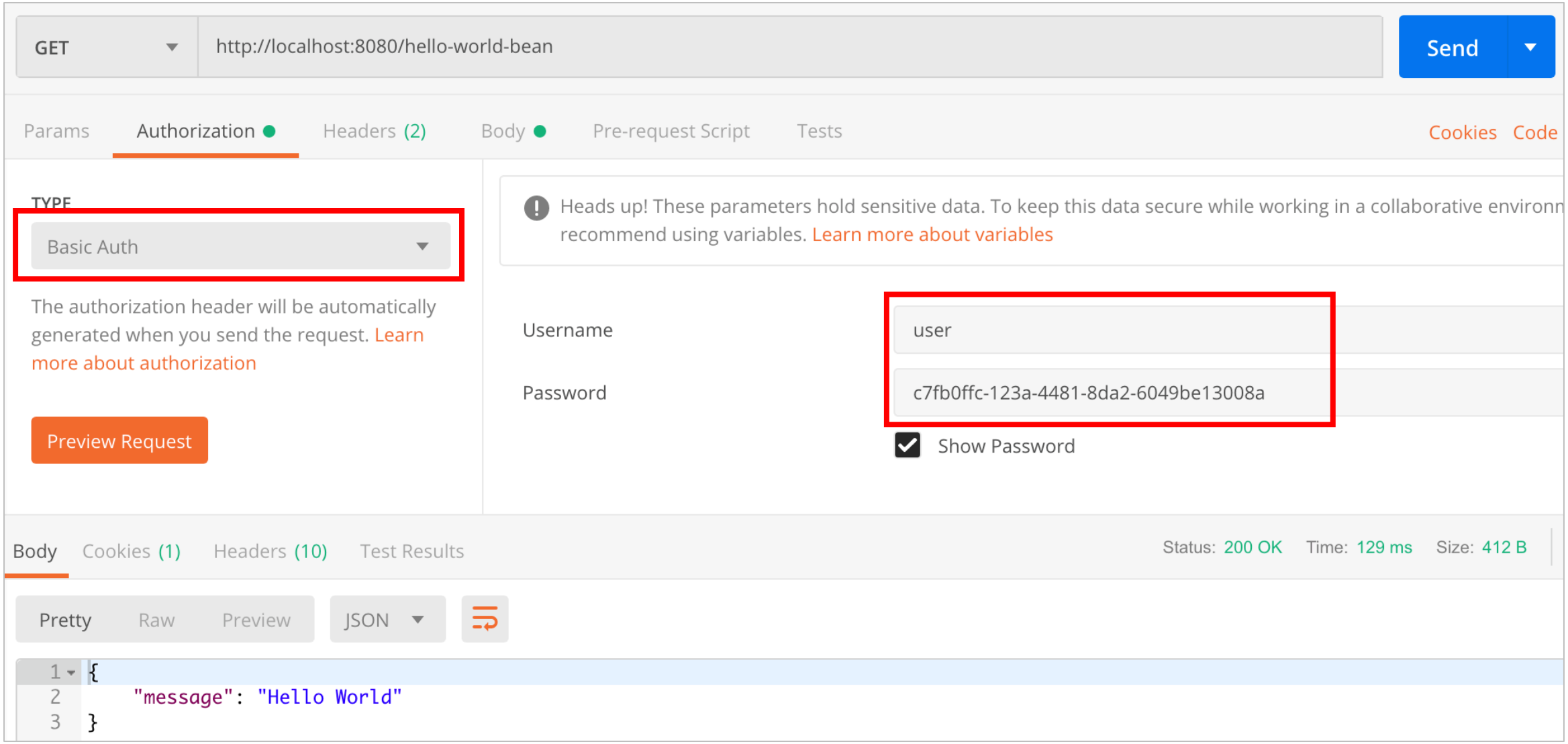
1) pom.xml 파일에 security 모듈 추가

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

```
Using generated security password: c7fb0ffc-123a-4481-8da2-6049be13008a
```

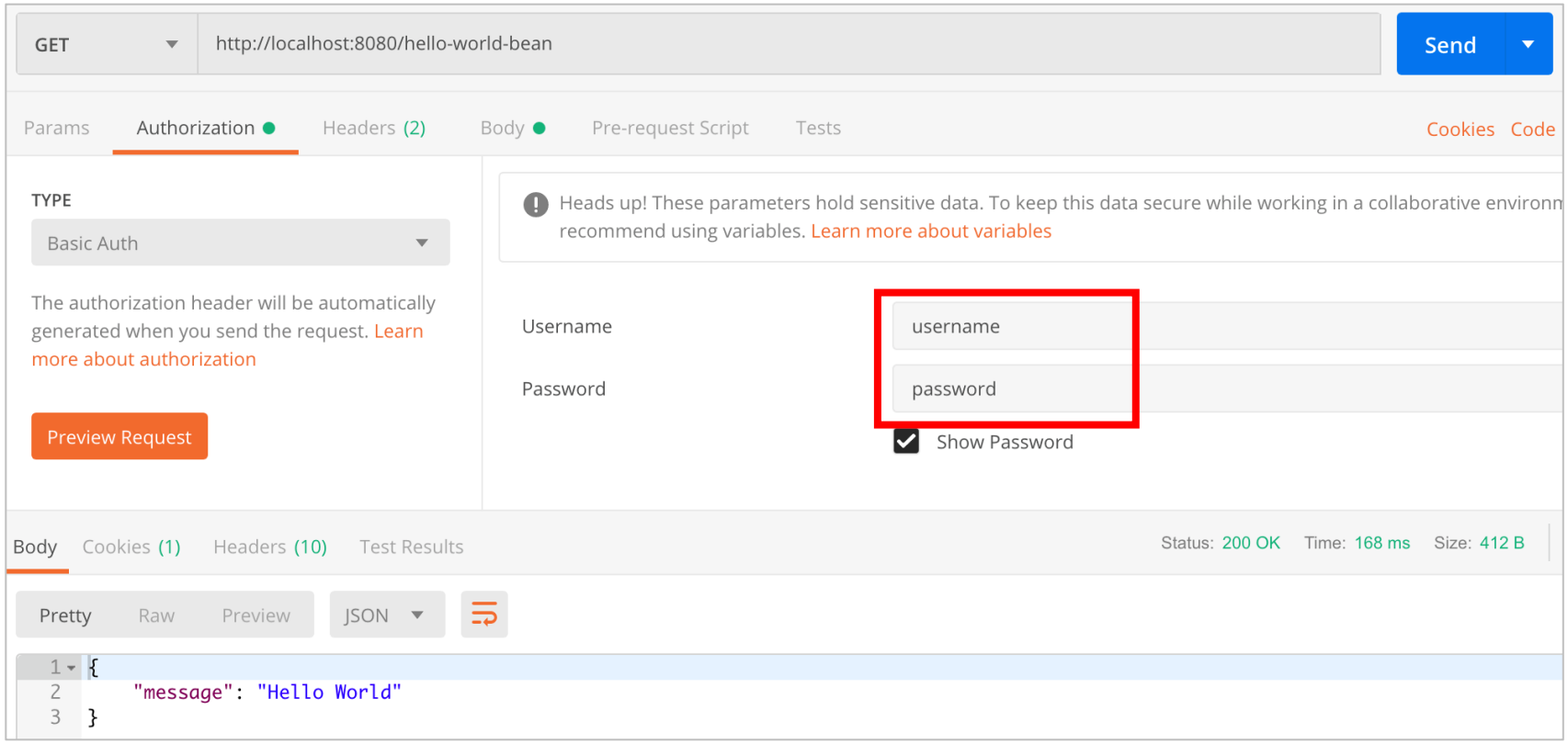
Spring Security


GET
http://localhost:8080/hello-world-bean
Send
Params
Authorization
Headers (1)
Body
Pre-request Script
Tests
Cookies
Code
TYPE
No Auth
This request does not use any authorization. Learn more about authorization
Body
Cookies (1)
Headers (11)
Test Results
Status: 401 Unauthorized
Time: 37 ms
Size: 568 B
Pretty
Raw
Preview
JSON
{
"timestamp": "2019-04-23T15:22:55.308+0000",
"status": 401,
"error": "Unauthorized",
"message": "Unauthorized",
"path": "/hello-world-bean"
}


GET
http://localhost:8080/hello-world-bean
Send
Params
Authorization
Headers (2)
Body
Pre-request Script
Tests
Cookies
Code
TYPE
Basic Auth
The authorization header will be automatically generated when you send the request. Learn more about authorization
Preview Request
Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environn
recommend using variables. Learn more about variables
Username
user
Password
c7fb0ffc-123a-4481-8da2-6049be13008a
Show Password
Body
Cookies (1)
Headers (10)
Test Results
Status: 200 OK
Time: 129 ms
Size: 412 B
Pretty
Raw
Preview
JSON
{
"message": "Hello World"
}

2) application.properties 파일로 보안 적용

```
spring.security.user.name=username
spring.security.user.password=password
```



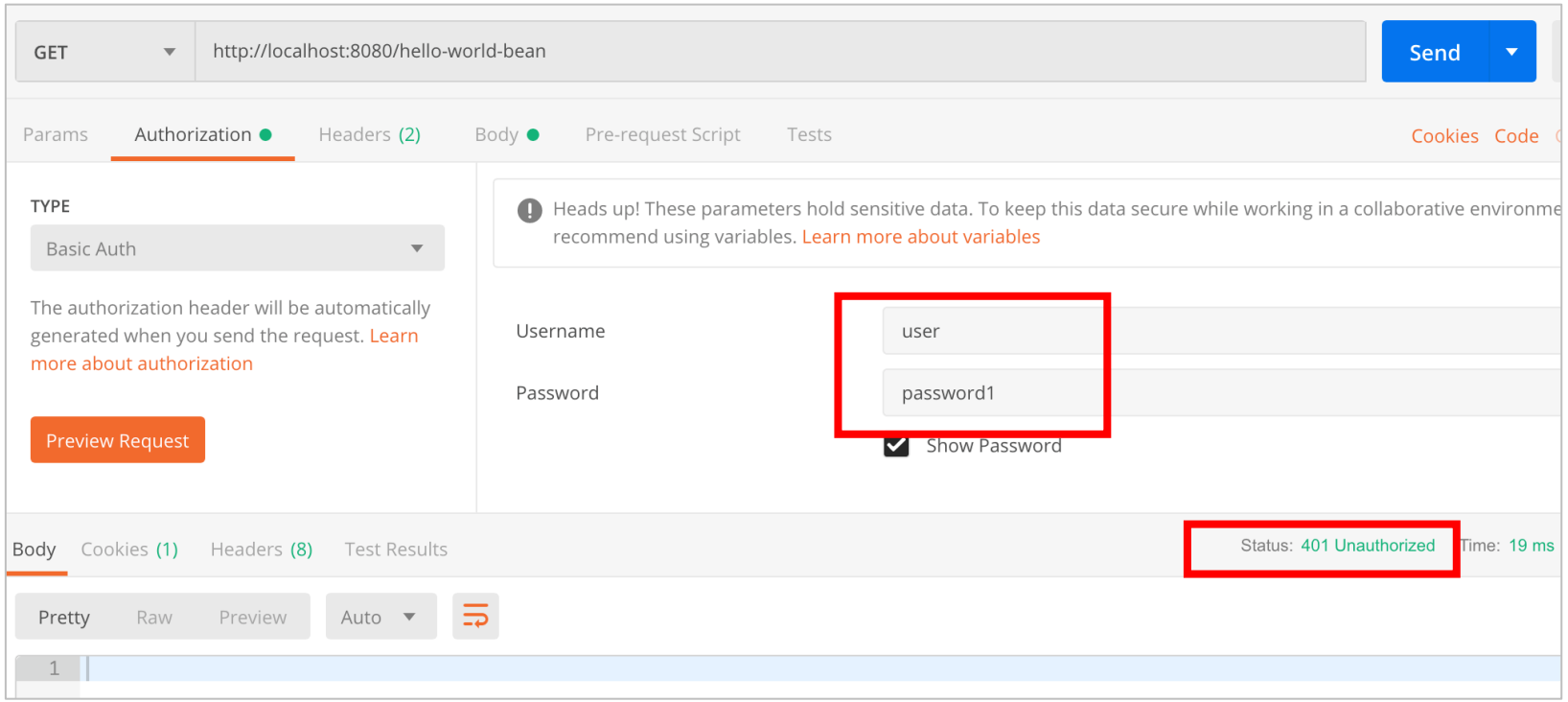
3) InMemory 보안

- InMemorySecurityConfiguration 클래스 생성

```
@Configuration
@EnableGlobalAuthentication
public class InMemorySecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Autowired
    public void confiureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication()
            .withUser("user").password("password1").roles("USER")
            .and()
            .withUser("admin").password("password1").roles("USER", "ADMIN");
    }

}
```

- @Configuration
  - 스프링 부트에서의 설정관련 클래스
- @EnableGlobalAuthentication
  - Application의 모든 빈에 보안 적용
- @Autowired/configuGlobal(AuthenticationManagerBuilder auth)
  - AuthenticationManagerBuilder 자동 연결
  - UserDetailsService와 인증 프로바이더를 연결

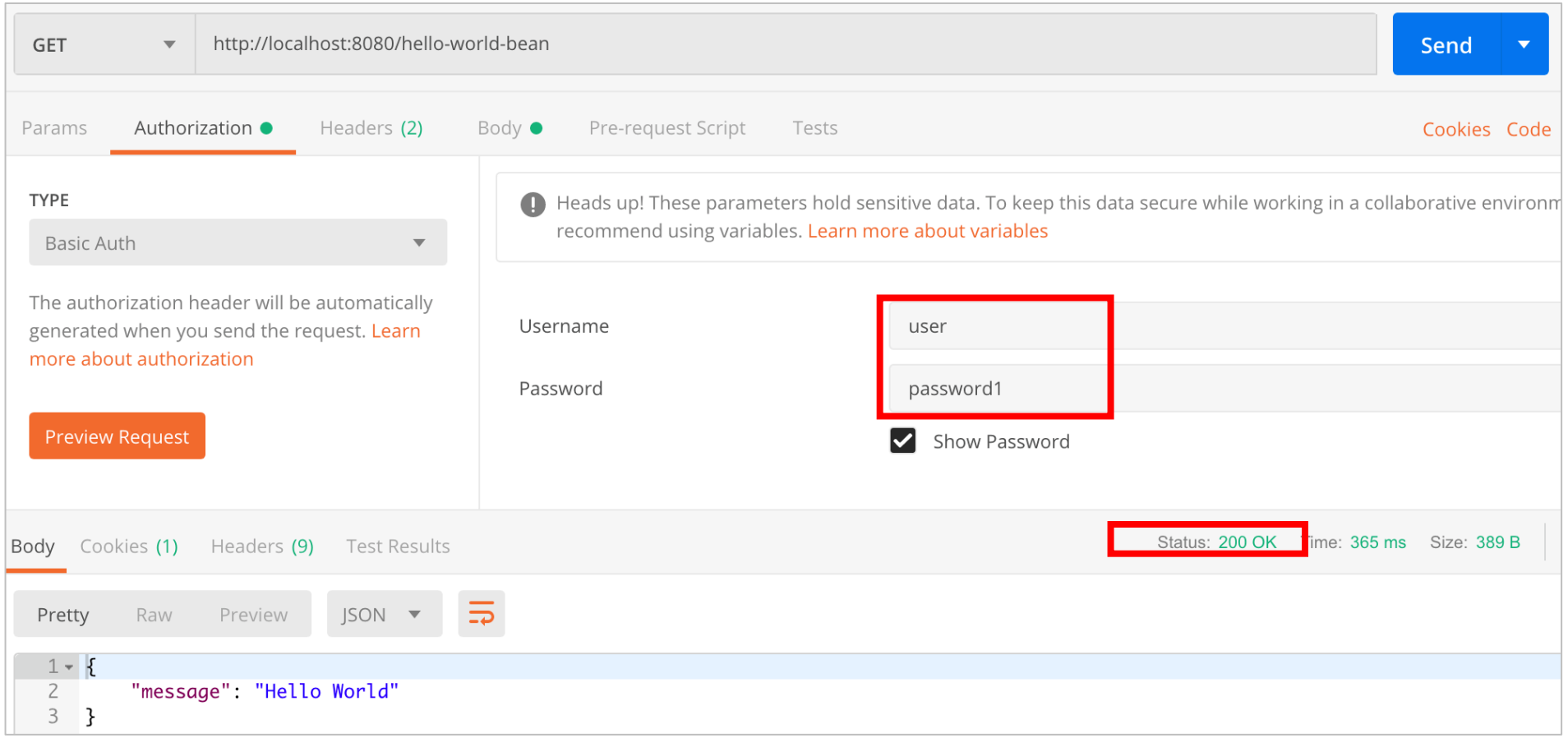
3) InMemory 보안




```
2019-04-24 00:35:00.870 ERROR 17262 --- [nio-8080-exec-3] o.a.c.c.C.[.[.[/].[dispatcherServlet]    : Servlet.service() for servlet [dispatcherServlet] in context with path []

java.lang.IllegalArgumentException: There is no PasswordEncoder mapped for the id "null"
        at org.springframework.security.crypto.password.DelegatingPasswordEncoder$UnmappedIdPasswordEncoder.matches(DelegatingPasswordEncoder.java:244) ~[spring-security-core
        at org.springframework.security.crypto.password.DelegatingPasswordEncoder.matches(DelegatingPasswordEncoder.java:198) ~[spring-security-core-5.1.5.RELEASE.jar:5.1.5.R
        at org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration$LazyPasswordEncoder.matches(AuthenticationConfiguration.jav
        at org.springframework.security.authentication.dao.DaoAuthenticationProvider.additionalAuthenticationChecks(DaoAuthenticationProvider.java:90) ~[spring-security-core-
        at org.springframework.security.authentication.dao.AbstractUserDetailsAuthenticationProvider.authenticate(AbstractUserDetailsAuthenticationProvider.java:166) ~[spring
        at org.springframework.security.authentication.ProviderManager.authenticate(ProviderManager.java:175) ~[spring-security-core-5.1.5.RELEASE.jar:5.1.5.RELEASE]
        at org.springframework.security.authentication.ProviderManager.authenticate(ProviderManager.java:200) ~[spring-security-core-5.1.5.RELEASE.jar:5.1.5.RELEASE]
```

3) InMemory 보안

- Password에 대한 Encryption 설정이 없기 때문에 오류
- ***{noop}*** 으로 처리



4) DB 보안

- JdbcSecurityConfiguration 클래스 생성

```
@Configuration
@EnableGlobalAuthentication
public class JdbcSecurityConfiguration extends GlobalAuthenticationConfigurerAdapter {

    @Bean
    public UserDetailsService userDetailsService(JdbcTemplate jdbcTemplate) {
        RowMapper<User> userRowMapper = (ResultSet rs, int i) ->
            new User(
                    rs.getString("ACCOUNT_NAME"),
                    rs.getString("PASSWORD"),
                    rs.getBoolean("ENABLED1"),
                    rs.getBoolean("ENABLED2"),
                    rs.getBoolean("ENABLED3"),
                    rs.getBoolean("ENABLED4"),
                    AuthorityUtils.createAuthorityList("ROLE_USER", "ROLE_ADMIN"));
            return username -> jdbcTemplate.queryForObject("SELECT * from ACCOUNT where ACCOUNT_NAME=?", userRowMapper, username);
    }
```

- GlobalAuthenticationConfigurationAdapter
  - SecurityConfiguration 인터페이스를 구현 init() 재정의
- init(AuthenticationManagerBuilder)
  - AuthenticationManagerBuilder 인스턴스는 UserDetailsService 인스턴스를 구성하여 ***인메모리, LDAP, 기타 JDBC 기반의 인증***을 구현

4) DB 보안

- 사용자 정보 저장을 위한 ACCOUNT 테이블 생성
- Account 클래스 생성

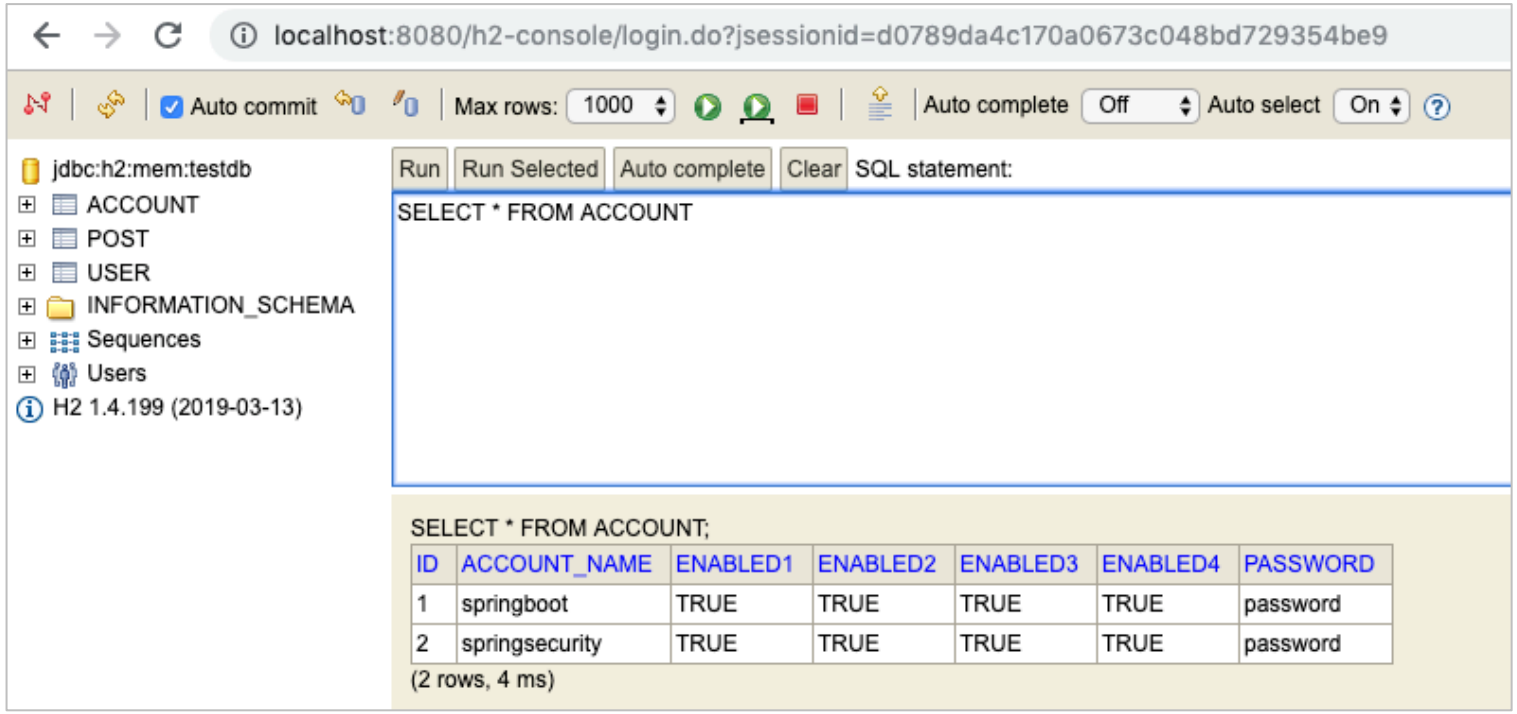
```java
@Entity
@Data
public class Account {
    @Id
    @GeneratedValue
    private Integer id;

    @Column(name="account_name")
    private String account_name;
    @Column(name="password")
    private String password;
    @Column(name="enabled1")
    private Boolean enabled1;
    @Column(name="enabled2")
    private Boolean enabled2;
    @Column(name="enabled3")
    private Boolean enabled3;
    @Column(name="enabled4")
    private Boolean enabled4;
```

```
Hibernate: create sequence hibernate_sequence start with 1 increment by 1
Hibernate: create table account (id integer not null, account_name varchar(255), enabled1 boolean, enabled2 boolean,
Hibernate: create table post (id integer not null, description varchar(255), user_id integer, primary key (id))
Hibernate: create table user (id integer not null, birth_date timestamp, name varchar(255), primary key (id))
```

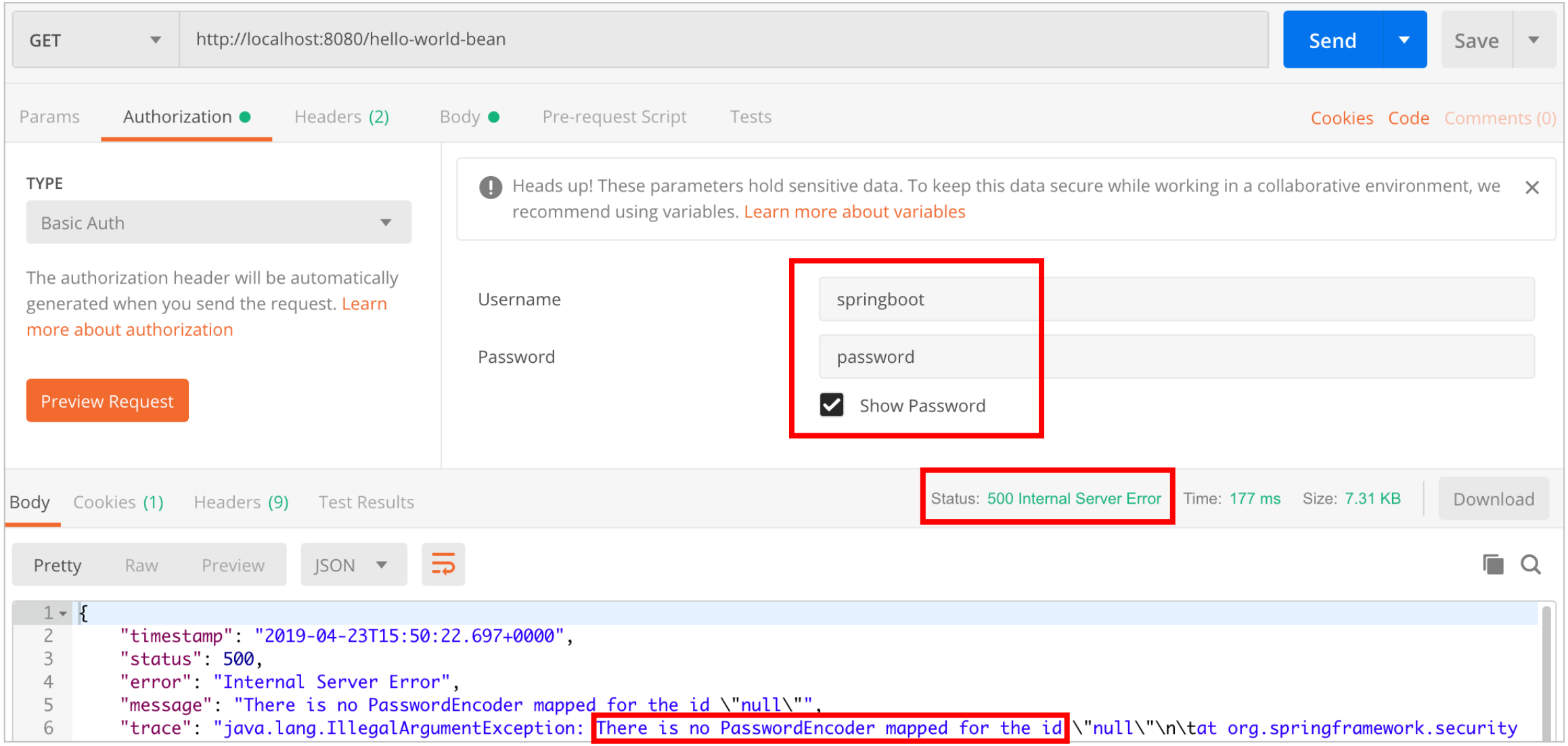
4) DB 보안

- 초기 데이터 저장

```
insert into account(id, account_name, password, enabled1, enabled2, enabled3, enabled4)
        values(1, 'springboot', 'password', true, true, true, true);
insert into account(id, account_name, password, enabled1, enabled2, enabled3, enabled4)
        values(2, 'springsecurity', 'password', true, true, true, true);
```



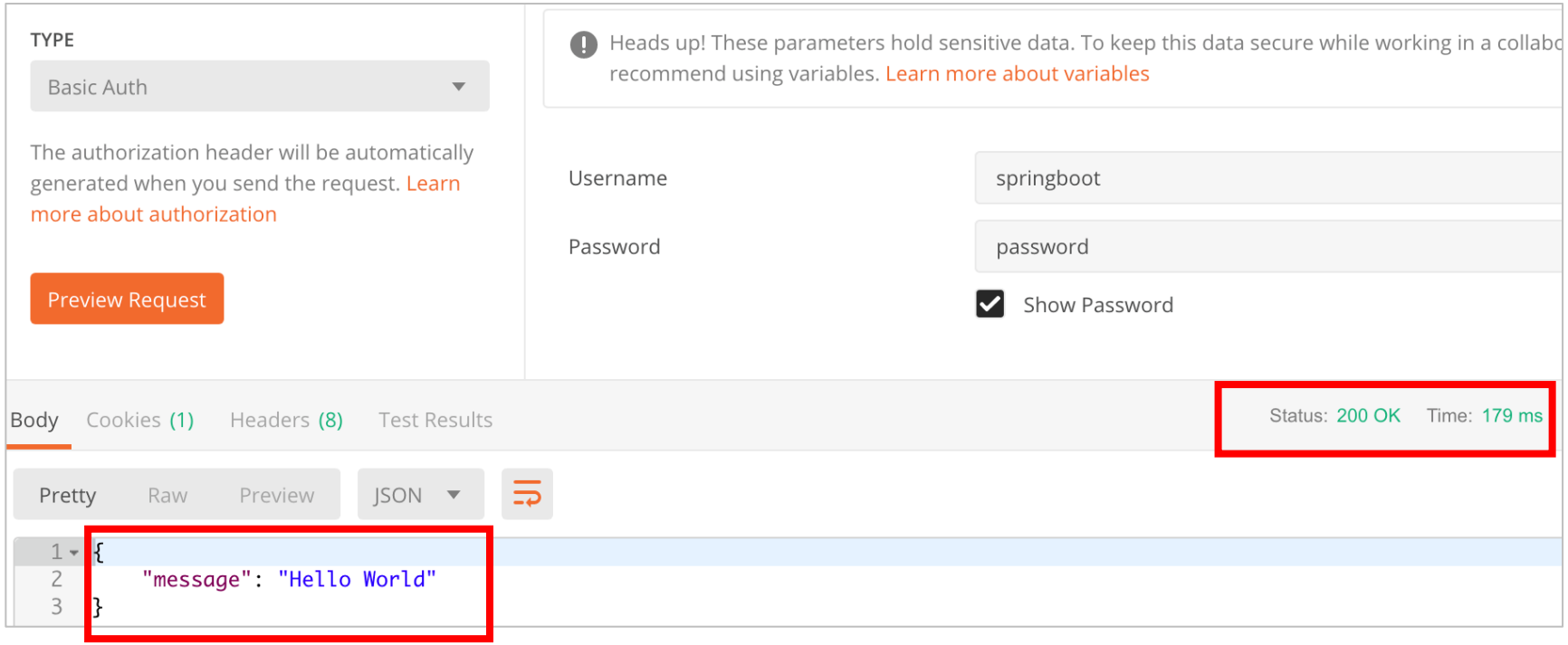
4) DB 보안


GET
http://localhost:8080/hello-world-bean
Send
Save
Params
Authorization
Headers (2)
Body
Pre-request Script
Tests
Cookies
Code
Comments (0)
TYPE
Basic Auth
The authorization header will be automatically generated when you send the request. Learn more about authorization
Preview Request
Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables
Username
springboot
Password
password
Show Password
Body
Cookies (1)
Headers (9)
Test Results
Status: 500 Internal Server Error
Time: 177 ms
Size: 7.31 KB
Download
Pretty
Raw
Preview
JSON
{
"timestamp": "2019-04-23T15:50:22.697+0000",
"status": 500,
"error": "Internal Server Error",
"message": "There is no PasswordEncoder mapped for the id \"null\"",
"trace": "java.lang.IllegalArgumentException: There is no PasswordEncoder mapped for the id \"null\"\n\tat org.springframework.security

4) DB 보안

- JdbcSecurityConfiguration 파일 수정
  - Password Encryption 사용하지 않도록 설정

```
@Bean
    public PasswordEncoder noOpPasswordEncoder(){
        return NoOpPasswordEncoder.getInstance();
    }
```



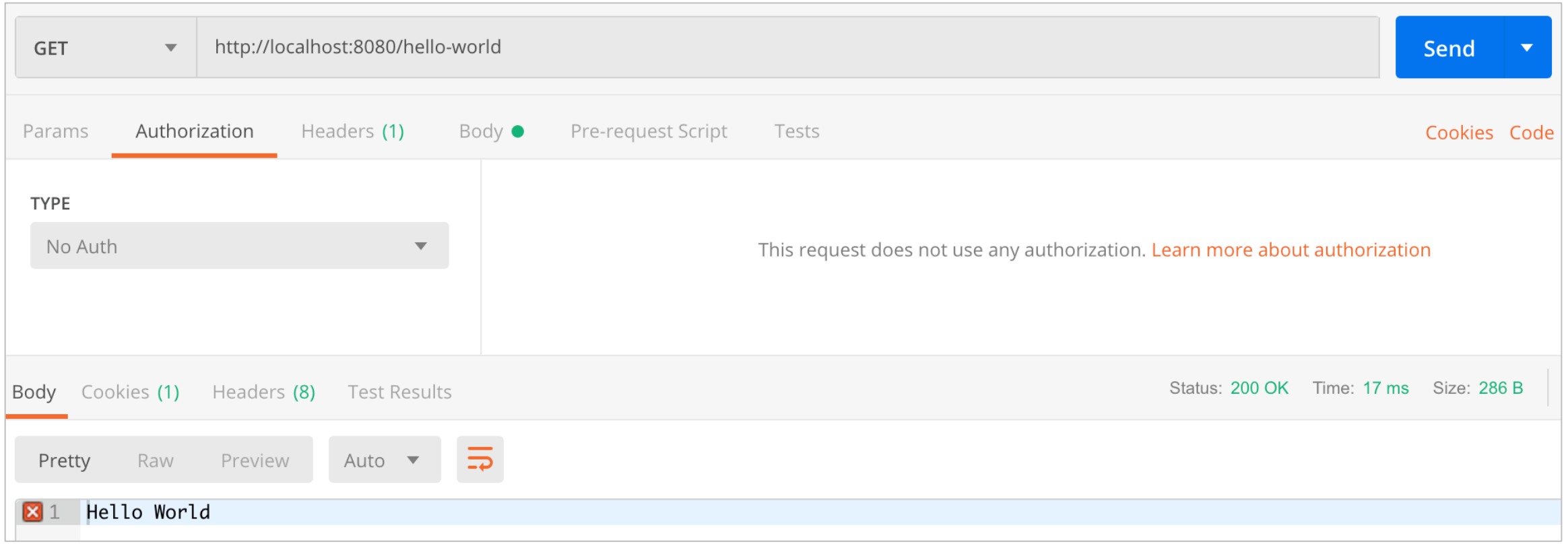
5) 리소스 보안

- 리소스 별로 권한 설정

```
@Configuration
public class WebSecurityConfiguration2 extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/hello-world/**").permitAll()
            .antMatchers("/h2-console/**").permitAll()
            .anyRequest().authenticated()
            .and()
            .csrf()
            .ignoringAntMatchers("/h2-console/**")
            .and()
            .headers()
            .frameOptions().disable()
            .and()
            .httpBasic();
    }
}
```

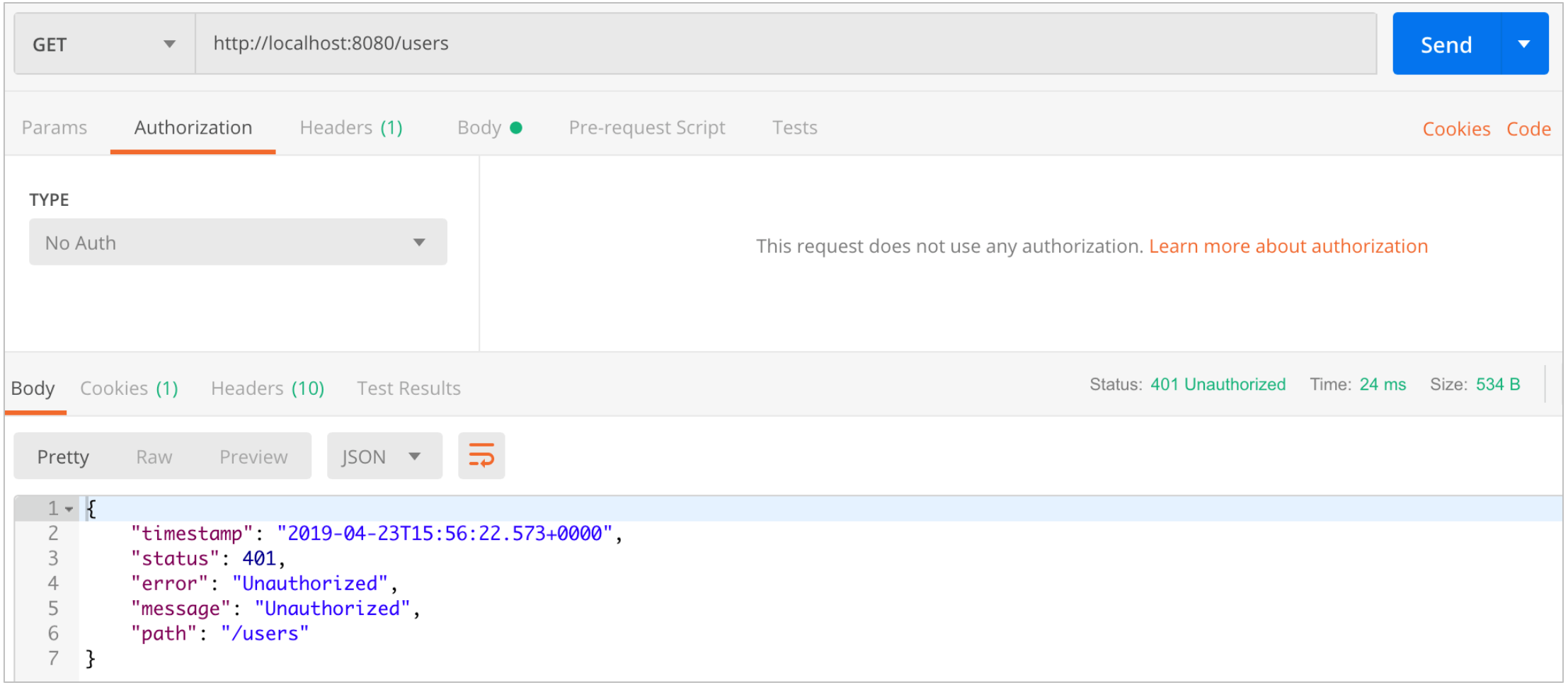
5) 리소스 보안

- /hello-world/** → ***PUBLIC 공개***



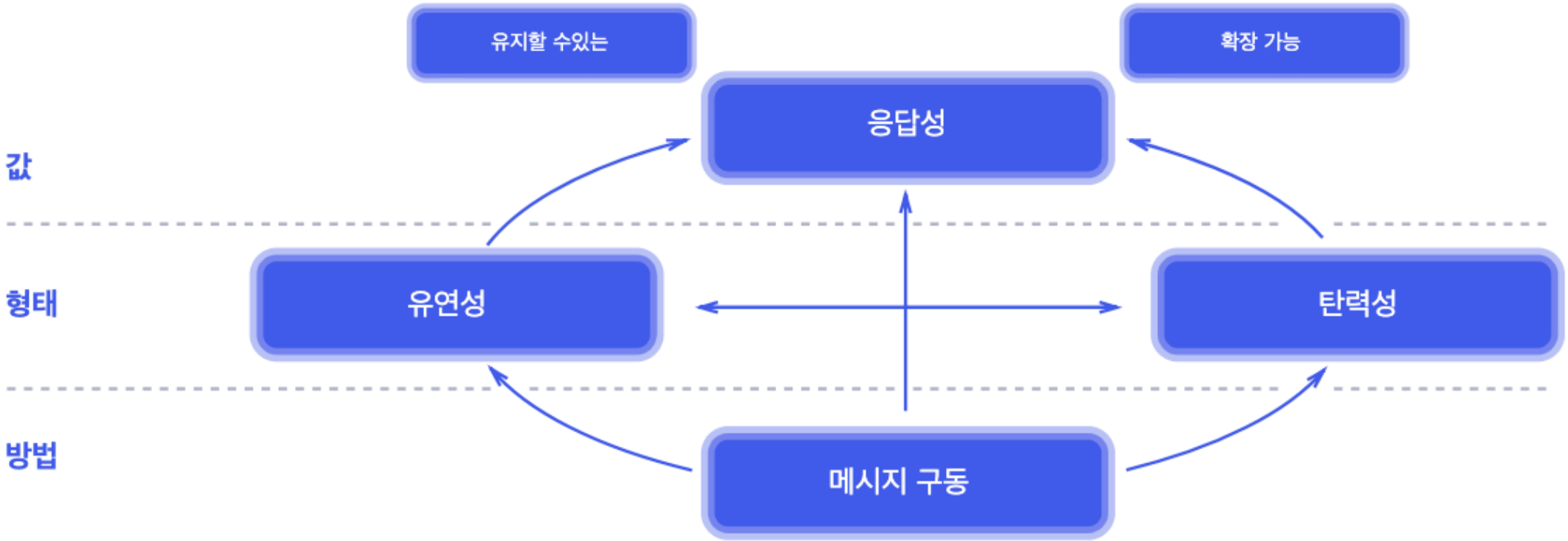
5) 리소스 보안

- Exclude /hello-world/** → ***PRIVATE 공개 (인증 필요)***



Reactive

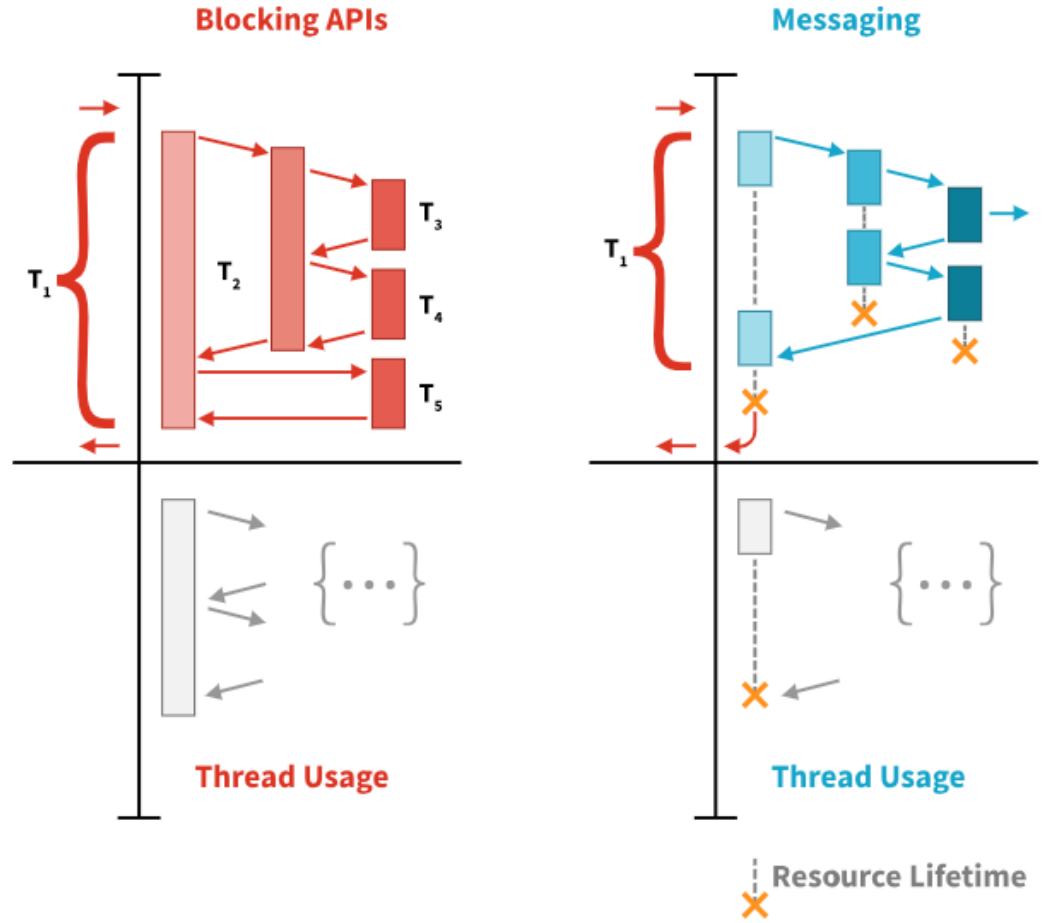
- 리액티브 시스템(Reactive Systems)
  - 응답이 잘 되고, 탄력적이며 유연하고 메시지 기반으로 동작하는 시스템



Spring Boot에서의 Reactive

- 비동기 데이터 스트림으로 프로그래밍하는 방식
- 기본적으로 변수, 속성, 캐시, 데이터 구조, 이벤트 등 모든 것을 스트림(Stream)으로 나타냄
  - 모든 데이터의 흐름을 시간 순서에 의해 전달되어지는 스트림으로 처리
  - 스트림은 시간 순서에 의해 전달되어진 값들의 Collection 혹은 List

- 하나의 문제를 각각 비동기와 논블로킹 방식으로 실행 될 수 있는 여러 단계로 분리
- 무한한 입력이나 출력을 생성할 수 있는 작업 흐름(workflow)를 만들기 위해 결합



Spring Boot에서의 Reactive

- 장점
  - 멀티 코어, 멀티 CPU 하드웨어에서 연산 자원 활용의 증가
  - 직렬화 지점을 감소 시켜 성능을 향상
  - 콜백 지옥에서 탈출
  - 스레드 관리 용이
  - backpressure. 과도한 사용과 무한한 자원 소비를 회피

- 단점
  - 멀티스레드와 함수형 프로그래밍에 대한 개념 이해 필수
  - 디버깅이 어려움

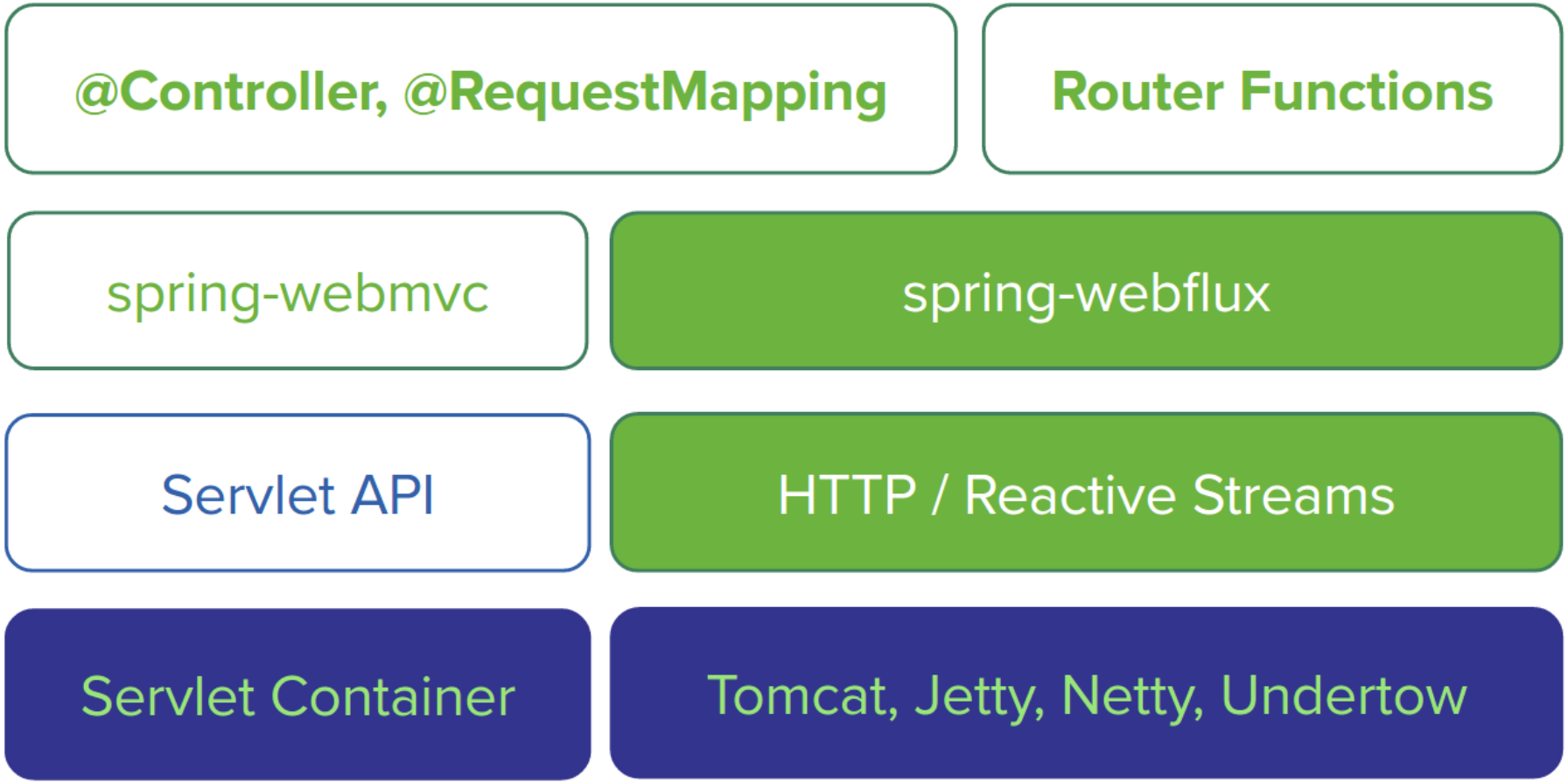
Spring Boot에서의 Reactive

- Reactive Programming
  - 비동기 데이터 스트림을 이용한 프로그래밍
  - 비동기 데이터 스트림에서 일어나는 변화를 지속적으로 관찰(Observe)하고 이에 따른 동작을 수행
- Functional Programming
  - 자료 처리를 수학적 함수의 계산으로 취급하고 상태와 가변 데이터를 멀리하는 프로그래밍 패러다임
  - 어떤 문제를 해결하기 위해서 그 과정이나 공식에 치중하기 보다는 이미 만들어진 함수의 응용을 강조
- Functional Reactive Programming
  - RP에 FP에서 제공하는 함수를 활용하는 Reactive Programming 패러다임

Spring 5 WebFlux

WebFlux

- Spring 5에서 새롭게 추가
- 기존 Spring Web Reactive → WebFlux
- 블로킹 런타임에서 리액티브 프로그래밍 할 수 있는 새로운 Web 프레임워크 (Spring MVC를 대체)



- WebFlux 프로그래밍 모델
  - @Controller
  - 함수형 모델
    - ***Router Functions, Handler Functions***
      - 람다 기반의 새로운 Controller의 구현 방법
      - ***Annotation 사용하지 않음***
      - 서블릿 기반이 아님

        → ***HttpServletRequest, HttpServletResponse 사용하지 않음***

        → ***ServerRequest, ServerResponse 를 사용***

- 스프링이 웹 요청을 처리하는 방식
  - 요청 매칭
    - 웹 요청을 어느 핸들러에게 보낼지 결정
    - URL 헤더
    - @RequestMapping
  - 요청 바인딩
    - 핸들러에게 전달할 웹 요청 준비
    - 웹 URL, 헤더, 쿠키, 바디
  - 핸들러 실행
    - 전달 받은 요청 정보를 이용해 로직을 수행 → 결과 반환 (Java Object)
  - 핸들러 결과 처리 (응답 생성)
    - 핸들러의 리턴 값으로 웹 응답 생성 → String or ResponseBody

- 스프링이 웹 요청을 처리하는 방식 – Spring MVC 방식

```java
@RestController
public class MyController {
    @GetMapping("/hello/{name}")
    public String hello(@PathVariable String name) {
        return "Hello " + name;
    }
}
```

요청 매핑

요청 바인딩

핸들러 실행

핸들러 결과 처리

- 스프링이 웹 요청을 처리하는 방식 – RouterFunction (함수형 스타일 매핑)

```java
@FunctionalInterface
public interface RouterFuntion<T extends ServerResponse> {
    Mono<HandlerFunction<T>> route(ServerRequest request);
}
```

웹 응답을 리턴

ServerResponse 타입으로 설정

- 스프링이 웹 요청을 처리하는 방식 – RouterFunction (함수형 스타일 매핑)
  - 요청 매칭 → RouterFunction (핸들러 결정)
  - 요청 바인딩
  - 핸들러 실행
  - 핸들러 결과 처리

  } 상태코드 생성, 리턴 타입 결정 등 결정

- WebFlux 함수형 Hello/{name} 작성
  - 함수 2개만 생성
    - HandlerFunction
    - RouterFunction에서 Path기준 매핑

- 스프링이 웹 요청을 처리하는 방식 – HandlerFunction (함수형 스타일 매핑)

```
HandlerFunction helloHandler = req -> {
    String name = req.pathVariable("name");
    Mono<String> result = Mono.just("Hello ", name);
    Mono<ServerResponse> res = ServerResponse
                                .ok()
                                .body(result, String.class);
    return res;
};
```

HandlerFunction은 람다식 → 파라미터 + 바디

- 스프링이 웹 요청을 처리하는 방식 – HandlerFunction (함수형 스타일 매핑)

```
HandlerFunction helloHandler = req -> {
    String name = req.pathVariable("name");
    Mono<String> result = Mono.just("Hello ", name);
    Mono<ServerResponse> res = ServerResponse
                                .ok()
                                .body(result, String.class);
    return res;
};
```

HandlerFunction은 람다식 → 파라미터 + 바디

Mono<T> handler(ServerRequest request);

- 스프링이 웹 요청을 처리하는 방식 – HandlerFunction (함수형 스타일 매핑)

```
HandlerFunction helloHandler = req -> {
 1)  String name = req.pathVariable("name");
 2)  Mono<String> result = Mono.just("Hello ", name);
 3)  Mono<ServerResponse> res = ServerResponse
                                    .ok()
                                    .body(result, String.class);
     return res;
};
```

- 웹 응답 생성 (ServerResponse)
- HTTP 응답 → 응답코드 + 헤더 + 바디
  - ServerResponse의 빌더 이용
- Mono에 담긴 ServerResponse 타입으로 리턴

- 스프링이 웹 요청을 처리하는 방식 – HandlerFunction (함수형 스타일 매핑)

```
HandlerFunction helloHandler = req ->
    ok().body(fromObject("Hello " + req.pathVariable("name")));
```

- ServerRequest에서 로직에 필요한 데이터 추출 & 바이딩

- 애플리케이션 로직을 적용하고 결과 값을 생성
- BodyInserters.fromObject()을 이용하여 Mono에 저장

- 로직의 결과 값을 바디에 담고 상태 코드를 추가 → 웹 응답(ServerResponse)으로 생성
- content-type이나 기타 헤더들은 스프링이 알아서 처리

함수형(람다식)으로 생성

- 스프링이 웹 요청을 처리하는 방식 – RouterFunction (함수형 스타일 매핑)

```
RouterFunction router = req ->
    RequestPredicates.path("/hello/{name}").test(req) ?
        Mono.just(HelloHandler) : Mono.empty();
```

- 웹 요청 정보 중에서 URL 경로 패턴 검사

- 조건에 맞으면 핸들러 함수를 Mono에 담아서 반환
- 함수이기 때문에 원하는 형태 반환 가능

RouterFunction()의 람다식

- 스프링이 웹 요청을 처리하는 방식 – HandlerFunction과 RouterFunction 조합

```
RouterFunction router = req ->
    RouterFunctions.route(RequestPredicates.path("/hello/{name}"),
        req -> ServerResponse.ok().body(fromObject("Hello " +
            req.pathVariable("name"))));
```

```
RouterFunction router = req ->
    RequestPredicates.path("/hello/{name}").test(req) ?
        Mono.just(HelloHandler) : Mono.empty();
```

- RouterFunction의 매핑 조건을 체크하는 로직만 추출

- 스프링이 웹 요청을 처리하는 방식 – HandlerFunction과 RouterFunction 조합

```
RouterFunction router = req ->
    RouterFunctions.route(RequestPredicates.path("/hello/{name}"),
        req -> ServerResponse.ok().body(fromObject("Hello " +
                req.pathVariable("name"))));
```

```
HandlerFunction helloHandler = req ->
    ok().body(fromObject("Hello " + req.pathVariable("name"));
```

- HandlerFunction은 그대로

- RuouterFunction의 등록
  - RouterFunction 타입의 ***@Bean***으로 만듬

```
@Bean
RouterFunction helloPathVarRouter() {
    return route(RequestPredicates.path("/hello/{name}"),
        req -> ok().body(fromObject("Hello " +
                                    req.pathVariable("name"))));
}
```

- 핸들러 내부 로직이 복잡하다면 분리

  → 핸들러 코드만 람다 식으로 선언

  → 메소드를 정의하고 메소드 참조로 가져옴

- RuouterFunction의 등록

```
HandlerFunction handler = req -> {
    String res = myService.hello(req.pathVariable("name"));
    return ok().body(fromObject(res);)
}

return route(path("/hello/{name}"), handler);
```

- Handler를 별도의 클래스로 생성

```
@Component
public class HelloHandler {
    @Autowired MyService myService;

    Mono<ServerResponse> hello(ServerRequest req) {
        String res = myService.hello(req.pathVariable("name"));
        return ok().body(fromObject(res));
    }
    ...
}
```

```
@Bean
RouterFunction helloRouter(@Autowired HelloHandler helloHandler) {
    return route(oath("/hello/{name}"), helloHandler::hello);
}
```

- 핸들러 메소드가 정의된 빈을 주입

- RouterFunction의 조합 (Chaining)
- RouterFunction의 매핑 조건의 중복
  - 타입 레벨 – 메소드 레벨의 @RequestMapping처럼 공통의 조건을 정의 가능
  - ***RouterFunction.nest()***

```
public RouterFunction<?> routingFunction() {
    return nest(pathPrefix("/person"),
        next(accept(APPLICATION_JSON),
            route(GET("/{id}"), handler::getPerson)
            .andRoute(method(HttpMethod.GET), hendler::listPerson)
        ).andRoute(POST("/").and(contentType(APPLICATION_JSON),
        handler::createPerson));
}
```

- WebFlux 함수형 스타일 장점
  - 모든 웹 처리 작업을 명시적인 코드로 작성
    - 메소드 시그니처 관례와 타입 체크가 불가능한 Annotation에 의존하는 @MVC 스타일 보다 명확
    - 정확한 타입 체크 가능
  - 함수 조합을 통한 편리한 구성, 추상화에 유리
  - 테스트 작성의 편리함
    - 핸들러 로직은 물론이고 요청 매핑과 리턴 값 처리까지 단위테스트로 작성 가능

- WebFlux 함수형 스타일 단점
  - 함수형 스타일의 코드 작성이 편하지 않으면 작성이 어려움

- @MVC WebFlux
  - Annotation 메소드 형식의 관례를 이용하는 @MVC 방식과 유사
  - 비동기 + 논블로킹 리액티브 스타일로 작성
- ServerRequest, ServerResponse
  - WebFlux의 기본 요청, 응답 인터페이스 사용
  - 함수형 WebFlux의 HandlerFunction을 메소드로 만들었을 때와 유사하게 작업
  - 매핑만 Annotation 방식 사용

- @MVC WebFlux

```
@RestController
public static class MyController {
    @RequestMapping("/hello/{name}")
    Mono<ServerResponse> hello(ServerRequest req) {
        return ok().body(fromObject(req.pathVariable("name")))
    }
}
```

  - 요청 매핑과 등록은 기존 @MVC 방식으로

- @MVC 요청 바인딩과 Mono/Flux 리턴 값
  - @RequestBody 바인딩 (JSON, XML)
    - T
    - Mono<T>
    - Flux<T>

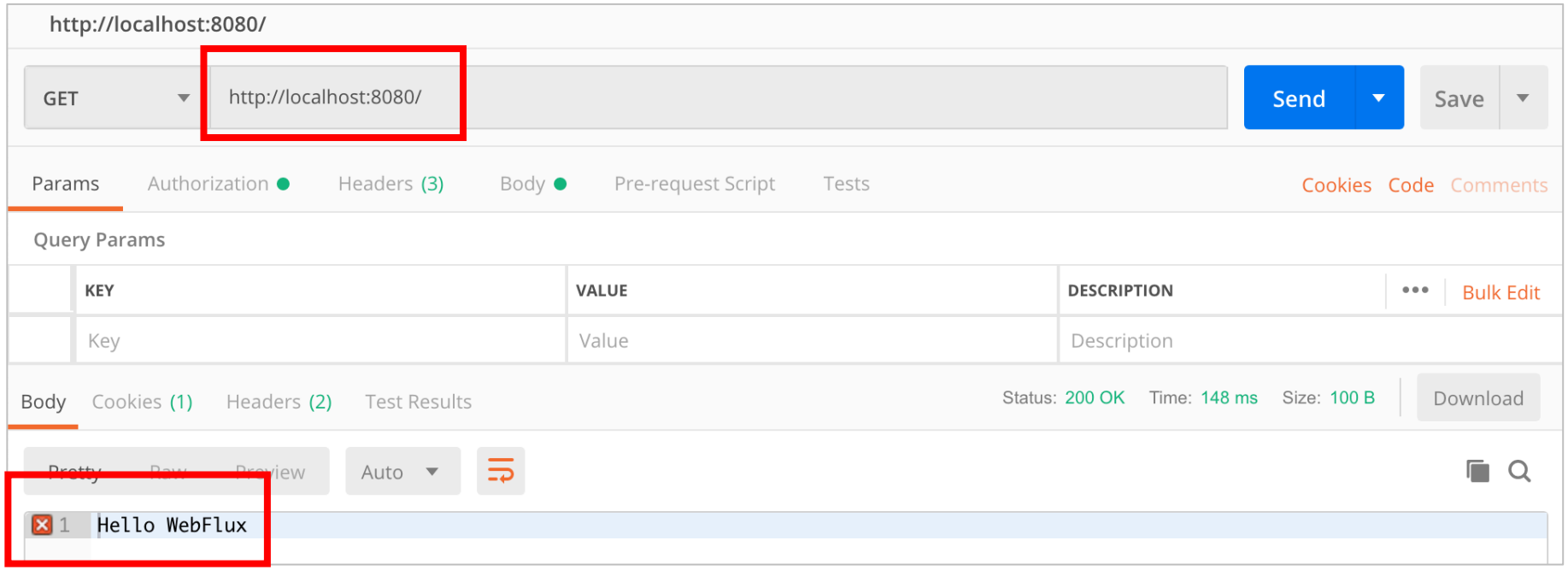
```
@GetMapping("/hello/{name}")
Mono<String> hello(@RequestBody User user) {
    return Mono.just("Hello " + user.getName());
}
```

- @MVC 요청 바인딩과 Mono/Flux 리턴 값
  - @ResponseBody 리턴 값 타입
    - T
    - Mono<T>
    - Flux<T>
    - Flux<ServerSideEvent>
    - void
    - Mono<Void>

WebFlux Example

- Router Functions에서 Routing 정의
  - ***WebFlux에서는 RouterFunction<ServerResponse>의 @Bean이 있으면 라우팅 정의로 취급***

```
@Bean
public RouterFunction<ServerResponse> routes() {
        return route(GET("/"), req -> ok().body(Flux.just("Hello", "WebFlux"), String.class));
}
```



WebFlux Example

- Router Functions에서 Routing 정의
  - ***PathVariable의 사용***
    - RequestPredicate (GET이나 POST 등의) 인수에 지금처럼 "{var}"과 같이 기술
    - ServerRequest의 ***.pathVariable()***로 취득

```
@Bean
public RouterFunction<ServerResponse> routes() {
        return RouterFunctions .route(GET("/hello/{name}"),
                req -> ok().body(Flux.just("Hello", req.pathVariable("name")), String.class));
}
```

- Router에서의 우선순위
  - PathVariable의 Routing과 "/hello/webflux"와 같은 고정 PATH의 경우에는 ***먼저 작성된*** 것이 우선 됨

```
RouterFunctions
        .route(GET("/hello/{name}"), req -> ...)
        .AndRroute(GET("/hello/webflux"), req -> ...);
```

```
RouterFunctions
        .route(GET("/hello/webflux"), req -> ...)
        .AndRroute(GET("/hello/{name}"), req -> ...);
```

WebFlux Example

- Routing 정의가 증가할 경우
  - RouterFunction<ServerResponse>를 반환하는 메서드를 만들어 다른 클래스에 정의
  - 원래 @Bean 정의하고있는 메소드 내에서 하나의 메소드를 호출

```
@Component
public class HelloRouter {
	private static final String PATH = "/hello";

	public RouterFunction<ServerResponse> route() {
		return RouterFunctions
				.route(GET(PATH), this::hello)
				.andRoute(GET(PATH + "/webflux"), this::helloWebflux)
				.andRoute(GET(PATH + "/{name}"), this::helloName);
	}

	private Mono<ServerResponse> hello(ServerRequest req) {
		return ok().body(Flux.just("Hello", "WebFlux"), String.class);
	}

	...
```

WebFlux Example

- Routing 정의가 증가할 경우
  - RouterFunction<ServerResponse>를 반환하는 메서드를 만들어 다른 클래스에 정의
  - 원래 @Bean 정의하고있는 메소드 내에서 하나의 메소드를 호출

```
@Component
public class HelloRouter {
	...

	private Mono<ServerResponse> helloName(ServerRequest req) {
		return ok().body(Flux.just("Hello", req.pathVariable("name")), String.class);
	}

	private Mono<ServerResponse> helloWebflux (ServerRequest req) {
		return ok().body(Flux.just("Hoge", "Hoge"), String.class);
	}
```

  - HelloRouter의 route() 호출

```
@Bean
public RouterFunction<ServerResponse> routes(HelloRouter helloRouter) {
    return helloRouter.route();
}
```

WebFlux Example

- Routing 정의가 증가할 경우
  - ***메소드 체인***으로 복수의 RouterFunction<ServerResponse>를 연결 가능

```
@Bean
public RouterFunction<ServerResponse> routes(HelloRouter helloRouter, StreamRouter streamRouter) {
    return helloRouter.route() .and(streamRouter.route());
}
```

  - 위 예제에서는 다음과 같은 routing 정보가 정의된다.
    - ***/hello***
    - ***/hello/webflux***
    - ***/hello/{name}***
    - ***/stream***

WebFlux Example

- RouterFunctions.nest()에 의한 정의
  - ***nest()*** 메소드를 사용하여 이전 코드를 변경 가능

```
public RouterFunction<ServerResponse> routes() {
    return nest(path("/hello"),
                route(GET("/"), this::hello)
                        .andRoute(GET("/hoge"), this::helloHoge)
                        .andRoute(GET("/{name}"), this::helloName));
}
```

WebFlux Example

- 무한 Stream
  - ServerResponse의 .body()로 설정
  - Flux라는 것은 Reactive Stream의 Publisher를 구현하고있는 클래스 → Reactive의 Stream을 실현 가능

ex) 엑세스 하면 카운트 된 숫자가 무한히 반복되어 반환

1) 무한 Stream 만들기
2) Stream에서 Flux 만들기
3) Flux을 body로 지정

```
@Bean
public RouterFunction<ServerResponse> routes() {
      Stream<Integer> stream = Stream.iterate(0, i -> i + 1);
      Flux<Integer> flux = Flux.fromStream(stream).delayElements(Duration.ofSeconds(1));

      return RouterFunctions
                .route(GET("/stream"),
                 req -> ok().contentType(MediaType.APPLICATION_STREAM_JSON).body(flux, Integer.class));
}
```

http://localhost:8080/stream 엑세스하면 숫자가 무한 반복됨