

점진적인 레거시 웹 애플리케이션 개선 과정

(continual improve legacy web application with spring cloud netflix & vue.js)

박용권

우아한신선들



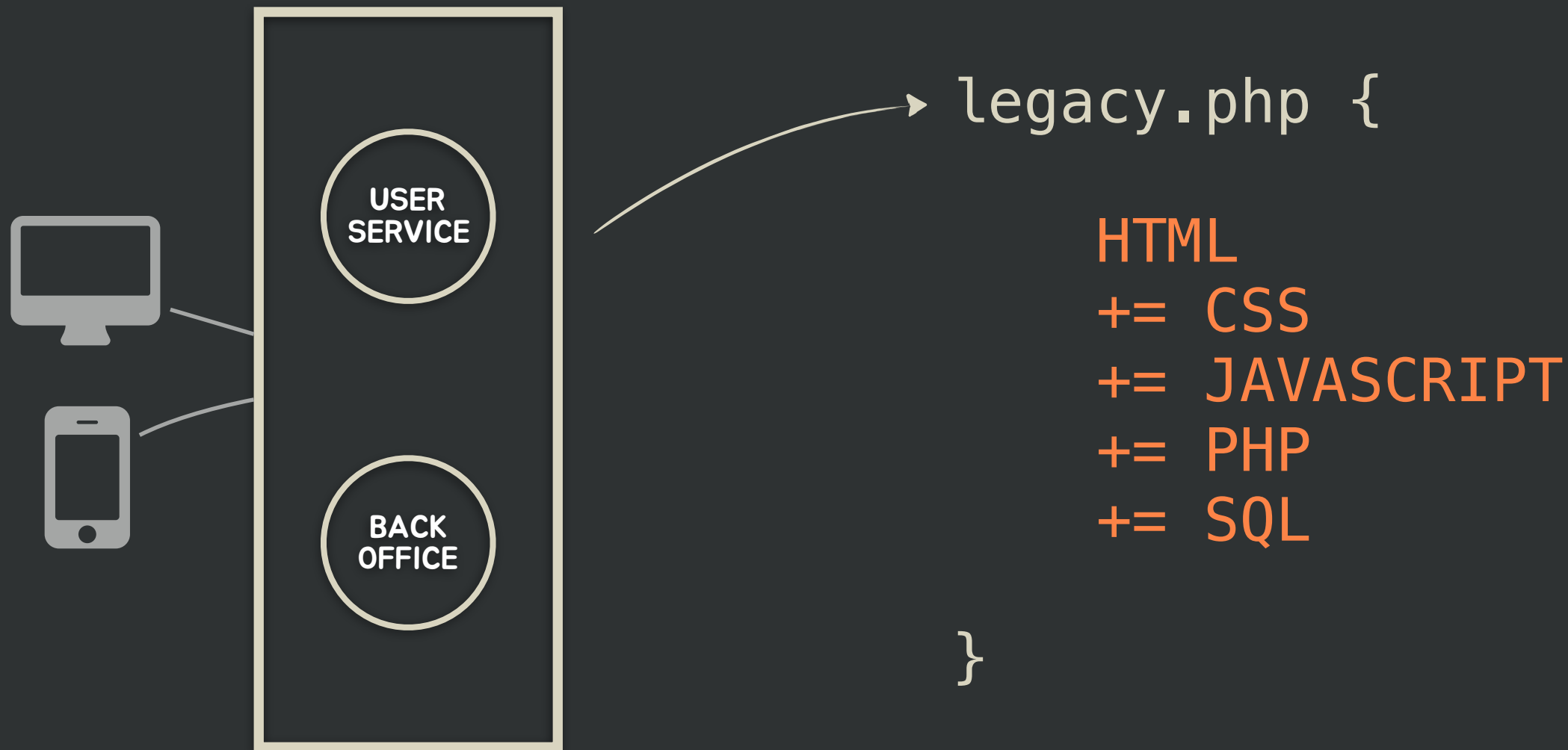
지난 1년 6개월간
배민찬 레거시 시스템을
개선하는 과정에서
주어졌던 고민거리
생각했던 아이디어
만들었던 결과물을 공유합니다.

◉ 2016년, 1만km 상공에서 배민찬 시스템 굽어보기



- ✓ 사용자 서비스와 백 오피스 모듈로 구성된 모놀리식 시스템
- ✓ 상용 온라인 커머스 솔루션을 사용자화(customizing)
- ✓ 다양한 비즈니스 정책을 흡수하며, 5년간 숙성
- ✓ 아마존 웹 서비스(Amazon Web Service)에서 운영 중

환상적인 하모니를 이루고 있는 레거시 코드



◉ 레거시 시스템을 개선하는 두 갈래 길

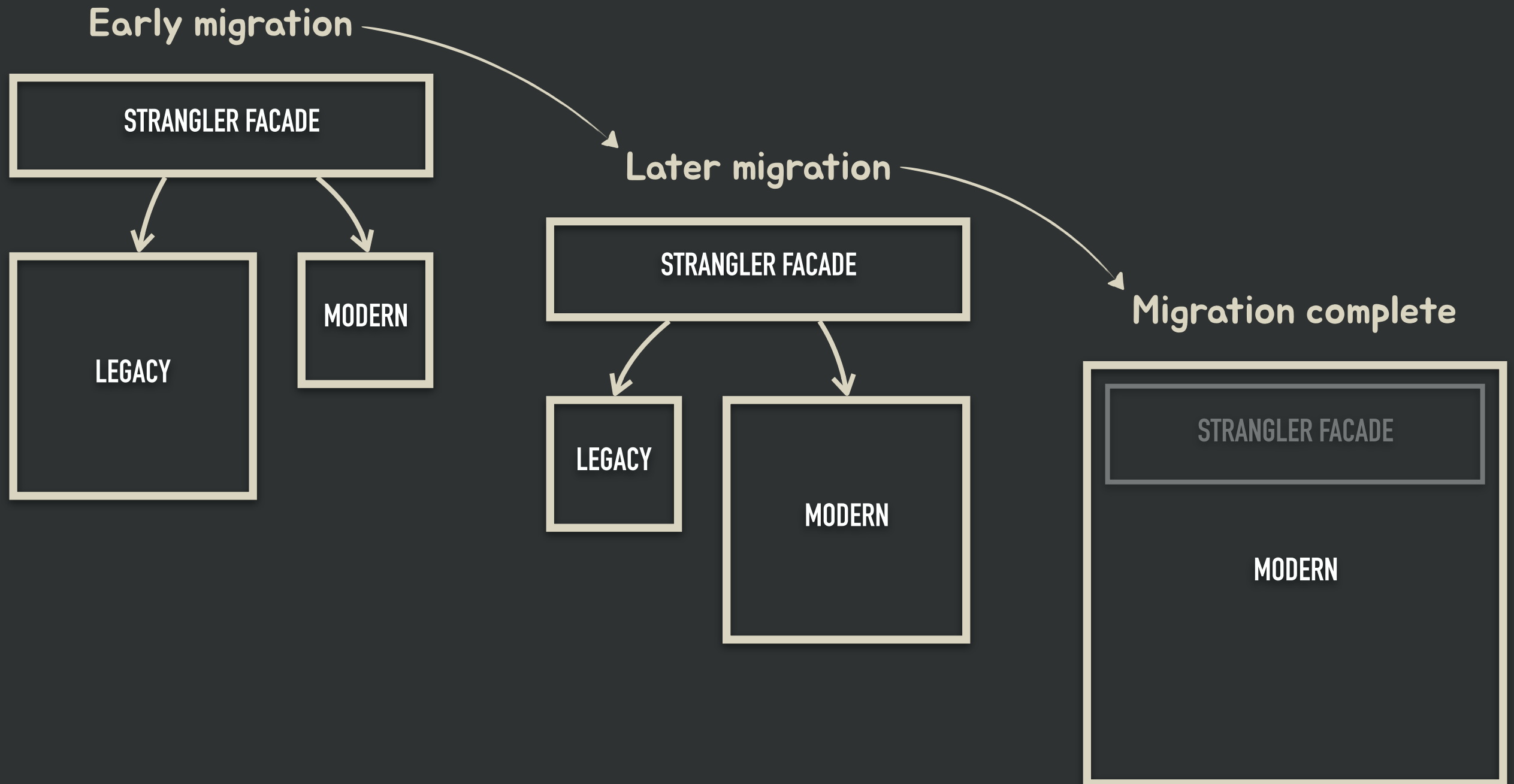
천 김에 왕까지!

한번에 하나씩! ◀

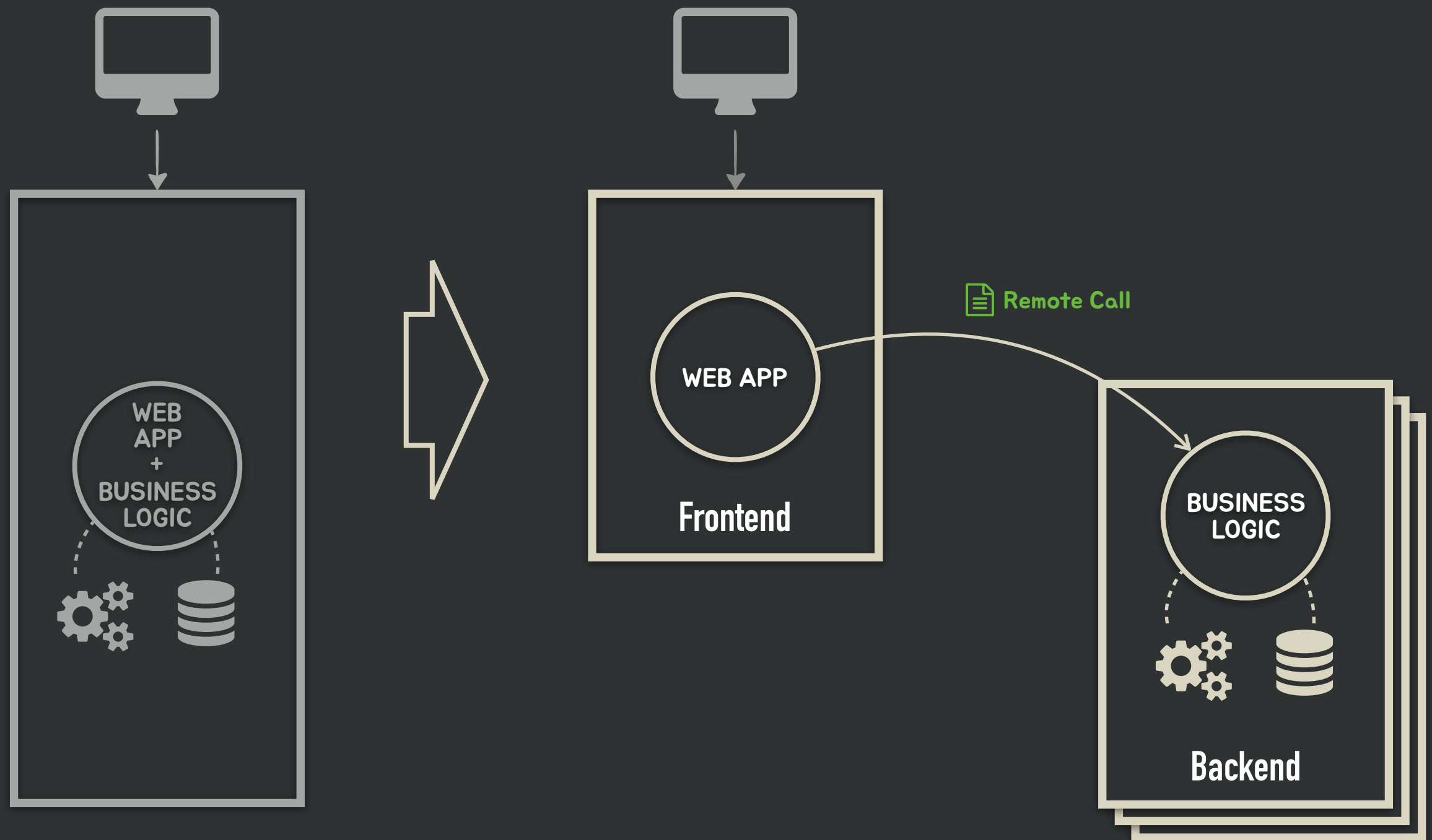
당시 배민찬 프로젝트 팀은 한참 재구축하던 시기였고, 완숙된 기술력을 잘 갖춘 팀이 아니었다. 또한 개선을 위한 면밀하게 분석하고, 설계하고, 충분히 조사할 비용(인력 및 시간)을 들일 여유가 있는 팀도 아니다. 그렇기에 무리하지 않게 한번에 하나씩, 점진적으로 바꿔나가기로 결정했다.

🌀 점진적 웹 개선을 위한 스트랭글러 패턴 (Strangler Pattern)

스트랭글러 패턴은 특정 기능을 새로운 응용 프로그램 및 서비스로 점진적으로 교체하여 레거시 시스템을 단계적으로 마이그레이션하는 것이다.



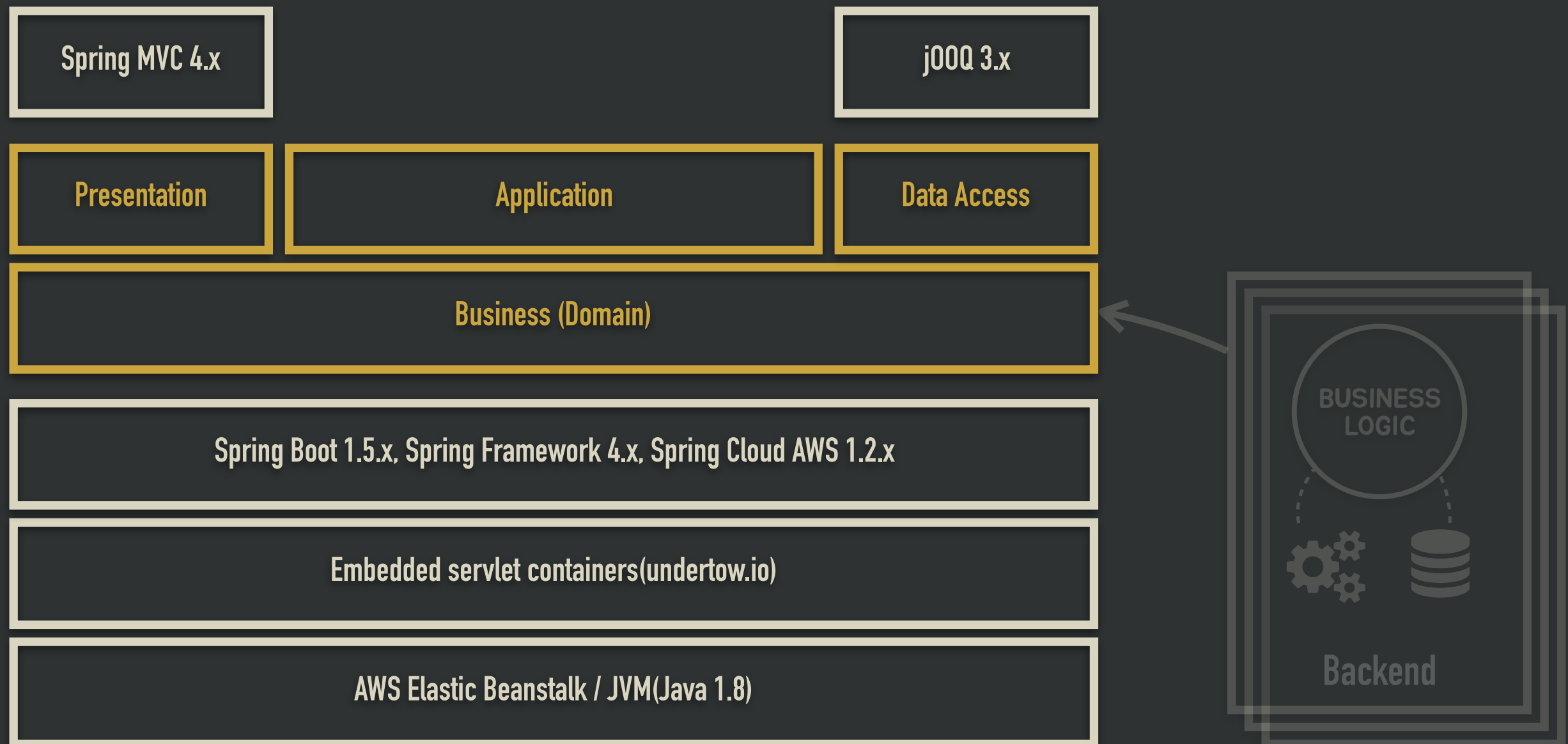
◉ 첫번째 결정, 프론트엔드와 백엔드 나누기



프리젠테이션 역할을 수행할 프론트엔드와 비즈니스 로직 및 데이터를 다루는 백엔드를 나누기로 한다.
그리고 백엔드는 역할에 따라 여러개를 구성한다.

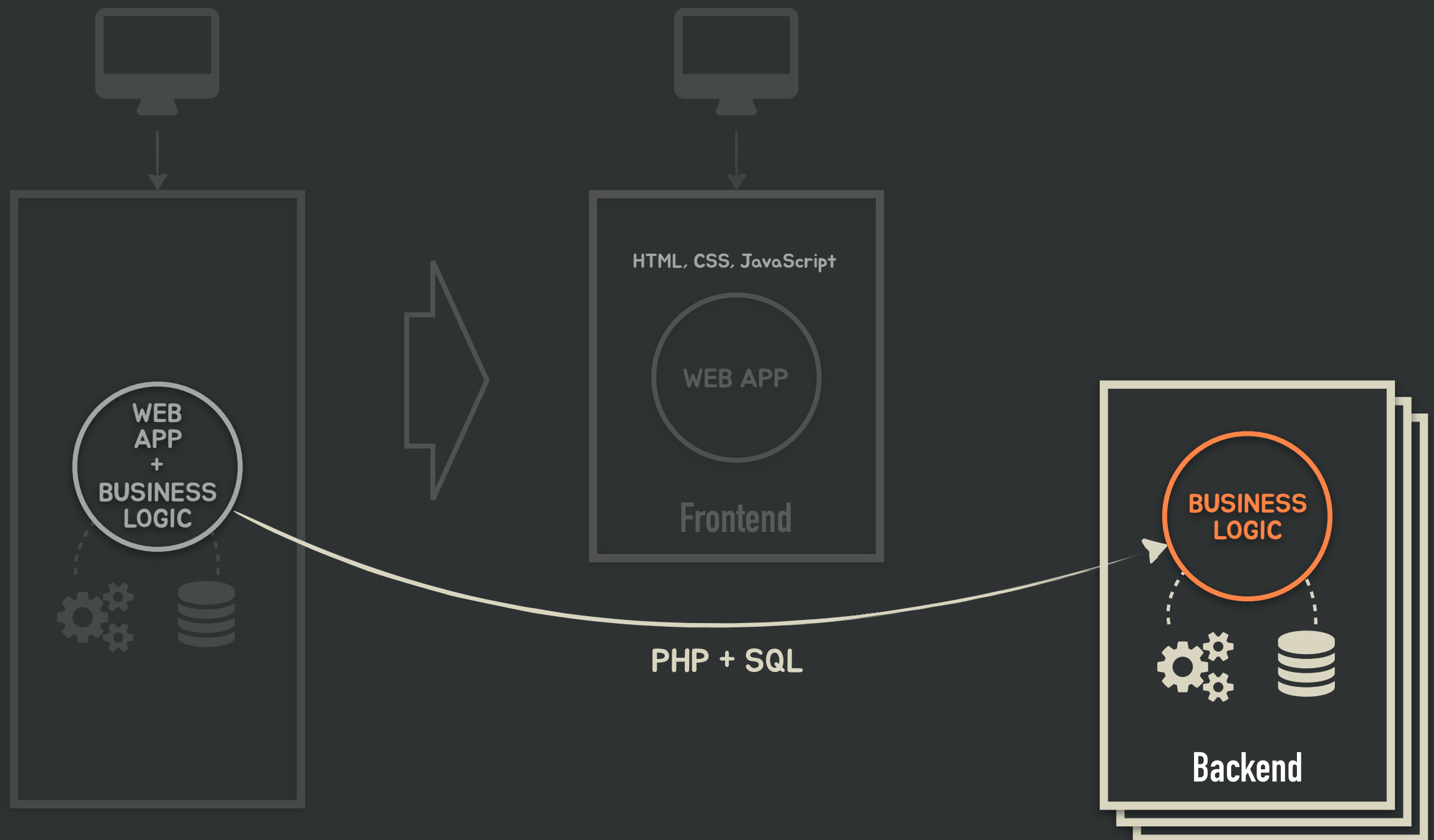
⋮ 점진적으로 기능을 백엔드(Backend)로 옮기기

복잡하게 얹히고 꼬인 레거시 시스템을 깨고,
푸드 커머스 플랫폼으로 **진화**하기 위한 발걸음



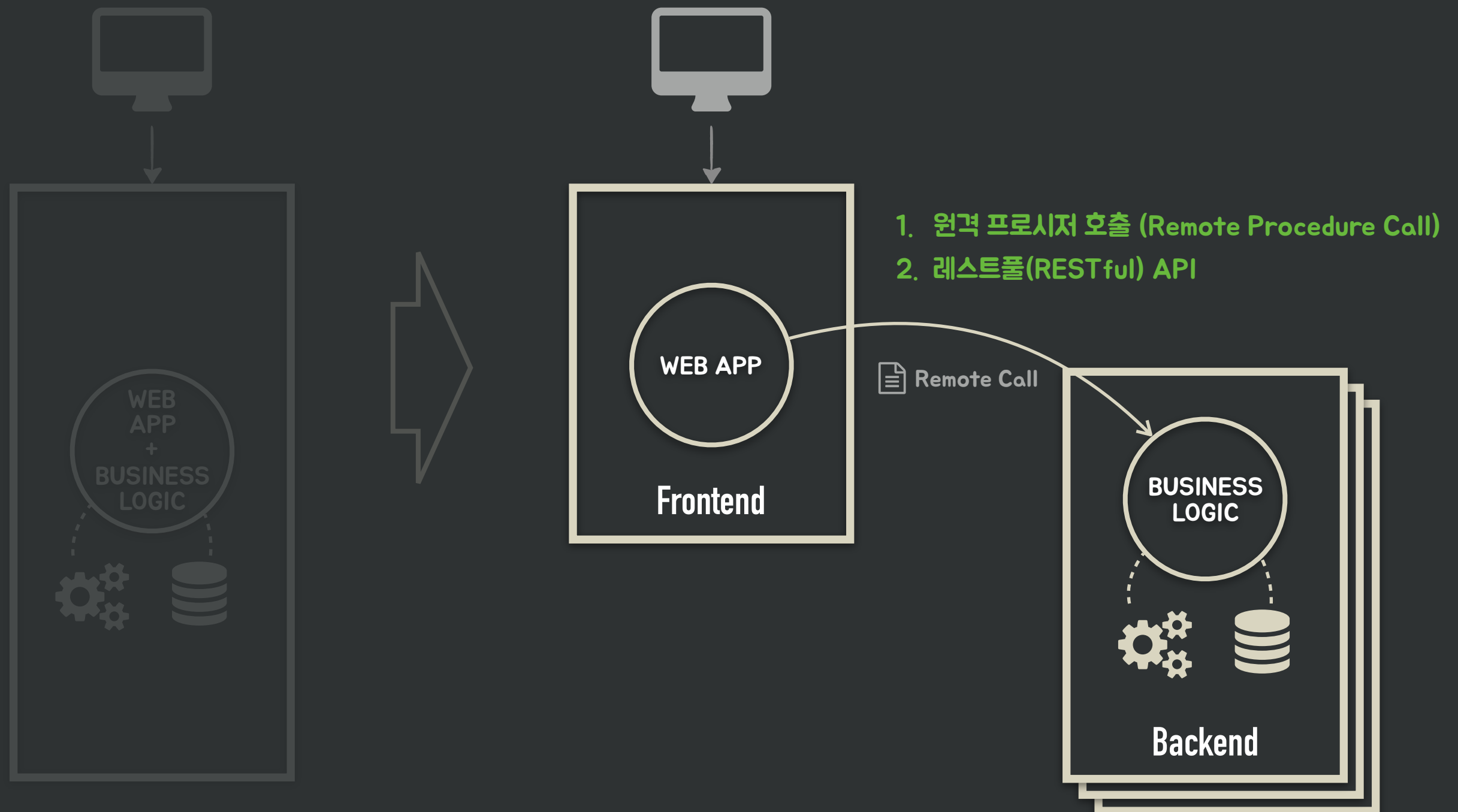
애플리케이션 구조 개선을 통해 사업의 요구사항을 보다 유연하게 받아 들일 수 있도록 한다.
플랫폼(PHP -> Java) 전환을 통해 유지보수성 / 견고함 / 보안 등을 얻어낸다.

◉ 점진적으로 기능을 백엔드(Backend)로 옮기기



레거시 시스템에서 기능(비즈니스 및 데이터를 처리하는 코드)을 점진적으로 새로운 서비스로 옮긴다.

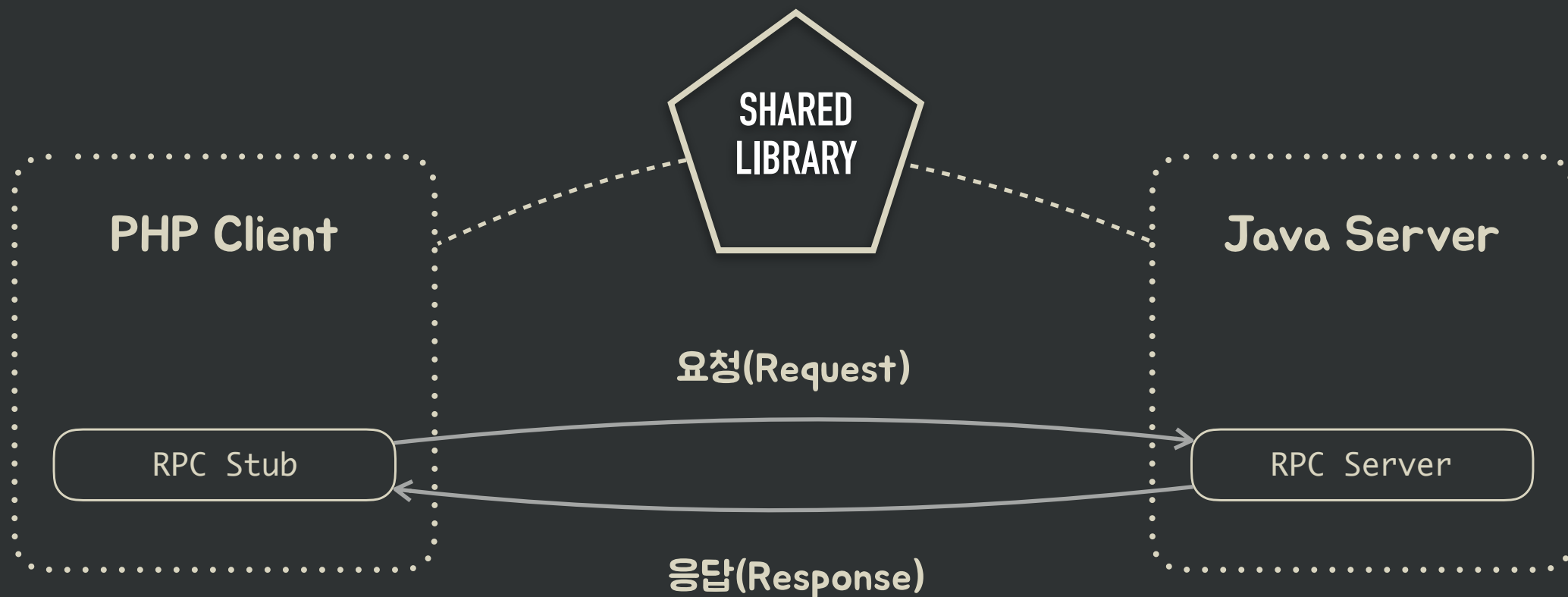
프론트엔드와 백엔드를 어떻게 통합할 것인가?



프론트엔드와 백엔드를 어떻게 통합할 것인가?

원격 프로시저 호출
(Remote Procedure Call)

레스트풀(RESTful) API



로컬 프로시저나 메소드를 호출하듯 원격 시스템에 서비스를 사용할 수 있는 방법으로 네트워크를 이용한다는 인지도 없이 원격 시스템에 서비스를 쉽게 통합하고, 사용할 수 있다. 하지만 두 시스템간 강결합이 일어난다. 오래된 기술 같지만 구글에서 공개한 gRPC의 경우 강력한 성능과 다양한 플랫폼을 지원하는 도구로 최근 큰 관심을 이끌고 있다.

⦿ 프론트엔드와 백엔드를 어떻게 통합할 것인가?

원격 프로시저 호출
(Remote Procedure Call)

레스트풀(RESTful) API

HTTP 프로토콜 기반

- 자원(RESOURCE) - URI
- 행위(Verb) - HTTP Method
- 표현(Representations)

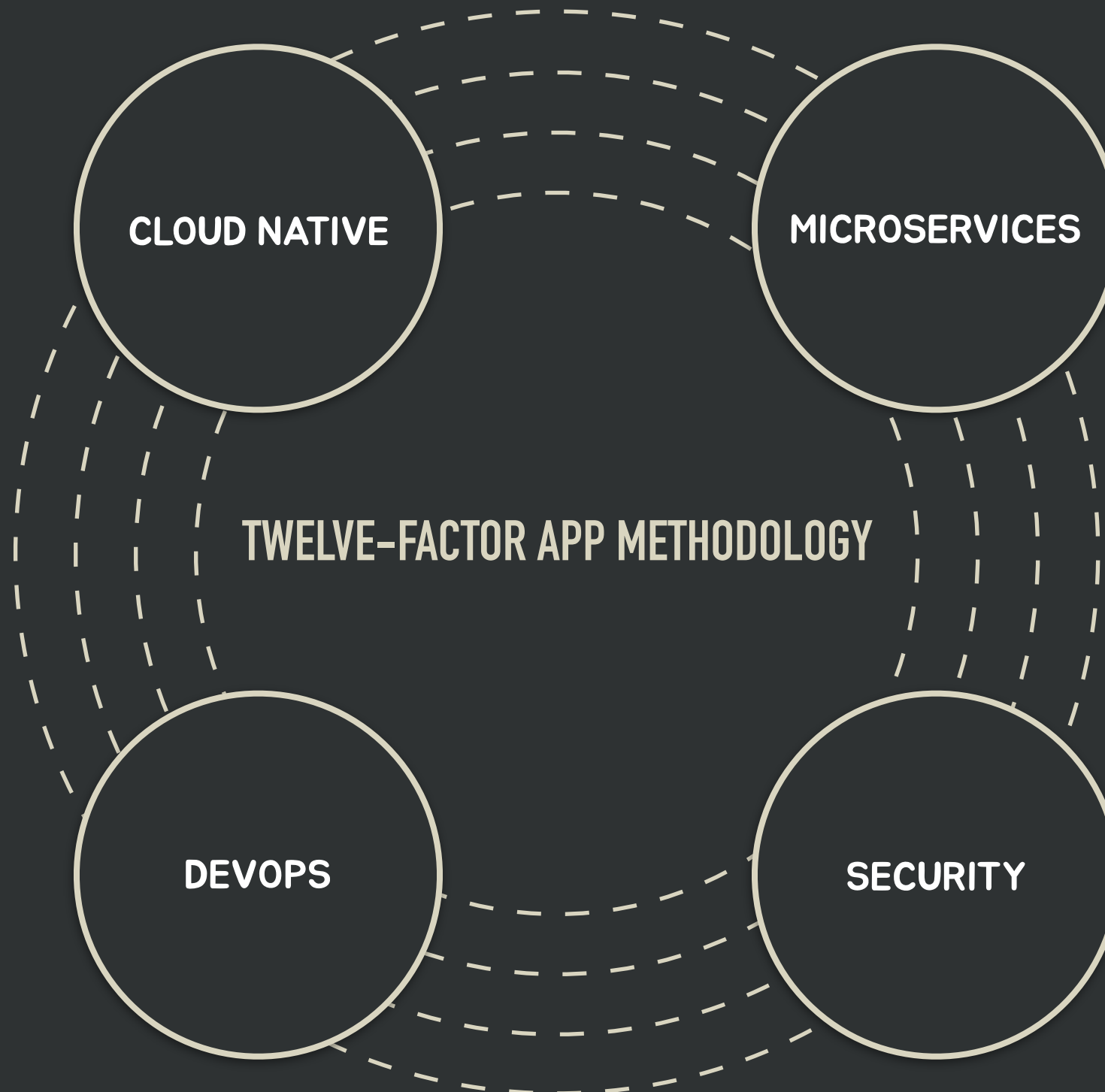


웹 기술이 가진 신뢰성, 확장성을 그대로 이어받아 분산 컴퓨팅에 적합한 API 만들수 있는 방법으로 자원(RESOURCE) - URI, 행위(Verb) - HTTP METHOD, 표현(Representations)으로 구성되어 있다. 서비스 제공자 장애나 네트워크 지연 등을 고려해야 한다.

플랫폼을 구축하기 위한 아키텍처 원칙

클라우드 컴퓨팅 환경에서 효율적으로 작동하는 시스템을 구축한다.

독립적인 역할을 수행하는 서비스들에 협업을 기반으로 시스템을 구축한다



신속한 서비스 제공, 비즈니스 가치 창출, 효율적인 비용-편익을 고려한다.

고객의 정보를 안전하게 지키기 위해, 외부 공격에 대비된 시스템을 구축한다.

🌌 참고: 12 요소 애플리케이션(The Twelve-Factor App)

허로쿠(Heroku) 클라우드 플랫폼을 만든 창시자들이 정립한 애플리케이션 개발 원칙 중 유익한 것을 모아 정리한 것으로, 클라우드 네이티브 애플리케이션을 만드는데 많은 역할을 수행한다.

핵심 사상

선언적 형식으로 설정을 자동화해서 프로젝트에 새로 참여하는 동료가 적응하는 데 필요한 시간과 비용을 최소화한다.
운영체제에 구애받지 않는 투명한 계약을 통해 다양한 실행 환경에서 작동할 수 있도록 이식성을 극대화한다.
현대적인 클라우드 플랫폼 기반 개발을 통해 서버와 시스템 관리에 대한 부담을 줄인다.
개발과 운영의 간극을 최소화해서 지속적 배포를 가능하게 하고 애자일성을 최대화한다.
도구, 아키텍처, 개발 관행을 크게 바꾸지 않아도 서비스 규모 수직적 확장이 가능하다.

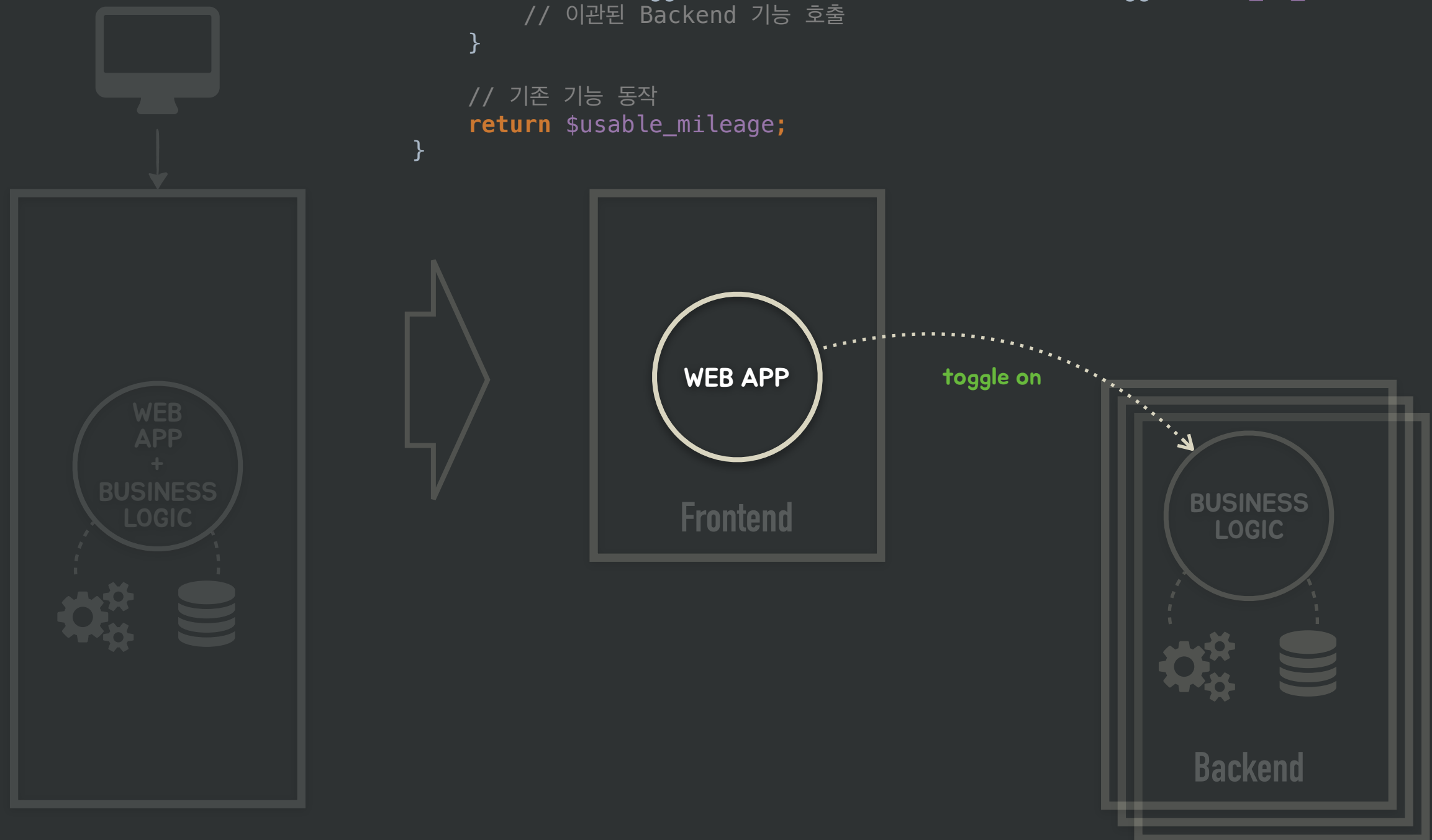
실천법

코드베이스(*Codebase*): 버전 관리되는 하나의 코드베이스가 여러 번 배포된다.
의존관계(*Dependencies*): 의존관계는 명시적으로 표시하고 격리한다.
설정(*Config*): 설정 정보는 실행 환경에 저장한다.
지원 서비스(*Backing services*): 지원 서비스는 필요에 따라 추가되는 자원으로 취급한다.
빌드, 릴리스, 실행(*Build, Release, Run*): 빌드와 릴리스, 실행 단계는 엄격하게 분리한다.
프로세스(*Processes*): 애플리케이션은 하나 이상의 무상태 프로세스로 실행한다.
포트 바인딩(*Port binding*): 서비스는 포트에 연결해서 외부에 공개한다.
동시성(*Concurrency*): 프로세스 모델을 통해 수평적으로 확장한다.
처분성(*Disposability*): 빠른 시작과 깔끔한 종료로 견고함을 극대화한다.
개발/운영 짝맞춤(*Dev/Prod parity*): 개발과 이관, 운영은 가능한 한 동일하게 유지한다.
로그(*Logs*): 로그는 이벤트 스트림으로 취급한다.
관리 프로세스(*Admin Process*): 관리(admin/management) 작업은 일회성 프로세스로 실행한다.

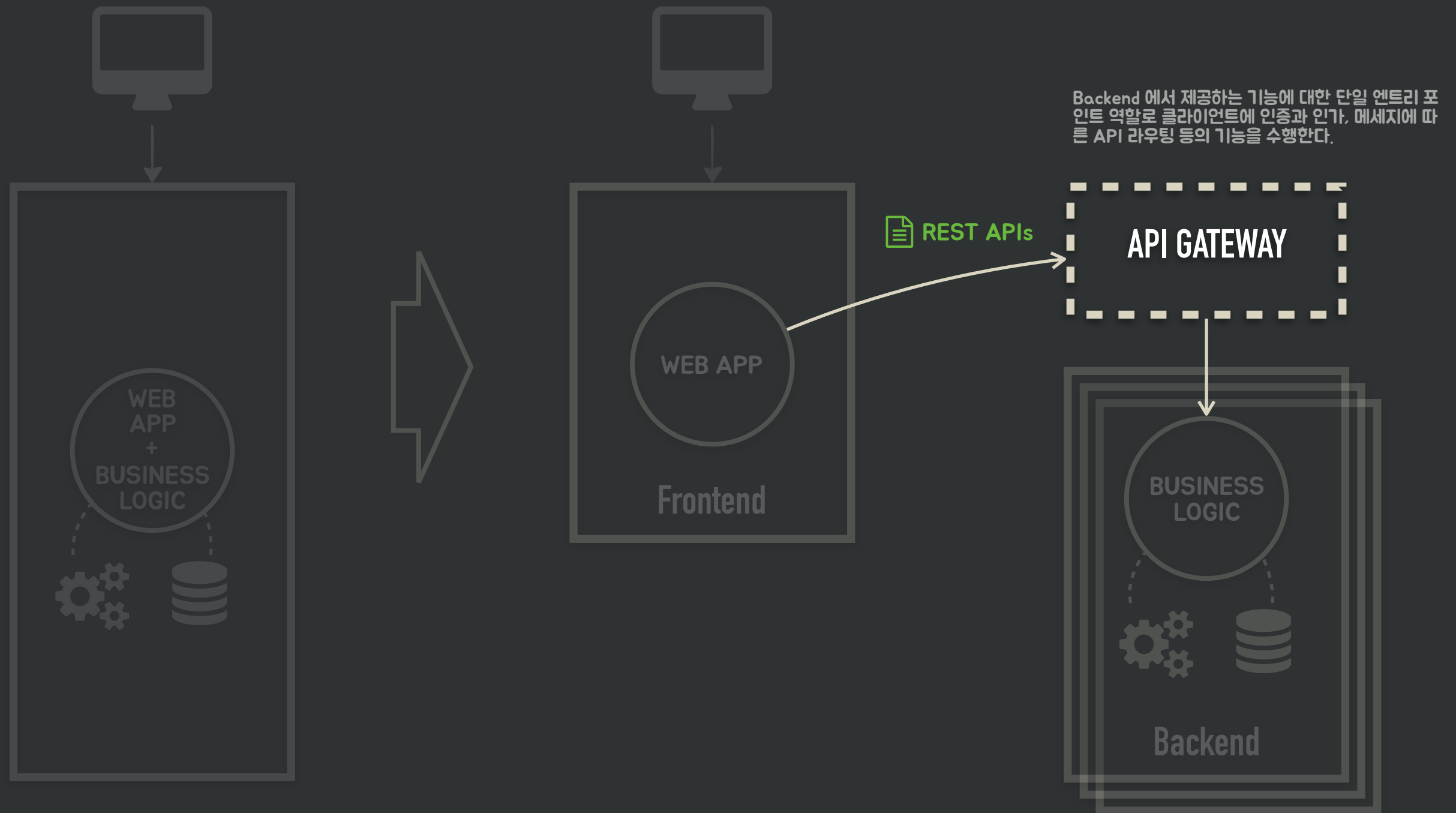
기능 토글(Feature Toggles)을 통한 안전장치 확보

기능 토글(Feature Toggle)은 변경한 기능이나 코드를 설정에 따라 켜고, 끄는 역할을 수행한다.
기능을 교체할 때 구 기능과 새 기능 모두 배포하고, 기능 토글을 사용해 전환함으로써 안정장치를 확보할 수 있다.


```
function getUsableMileage(){  
  if (FeatureToggle::create()->isEnabled(FeatureToggle::USE_BE_MILEAGE)) {  
    // 이관된 Backend 기능 호출  
  }  
  
  // 기존 기능 동작  
  return $usable_mileage;  
}
```



⋮ REST API 종단점을 위한 API Gateway 출현





스프링 클라우드 넷플릭스(Spring Cloud Netflix)



[DOCS](#) [GUIDES](#) [PROJECTS](#) [BLOG](#) [QUESTIONS](#) [Q](#)


PROJECTS : SPRING CLOUD

Spring Cloud Netflix



Spring Cloud Netflix provides Netflix OSS integrations for Spring Boot apps through autoconfiguration and binding to the Spring Environment and other Spring programming model idioms. With a few simple annotations you can quickly enable and configure the common patterns inside your application and build large distributed systems with battle-tested Netflix components. The patterns provided include Service Discovery (Eureka), Circuit Breaker (Hystrix), Intelligent Routing (Zuul) and Client Side Load Balancing (Ribbon)..

QUICK START



Features

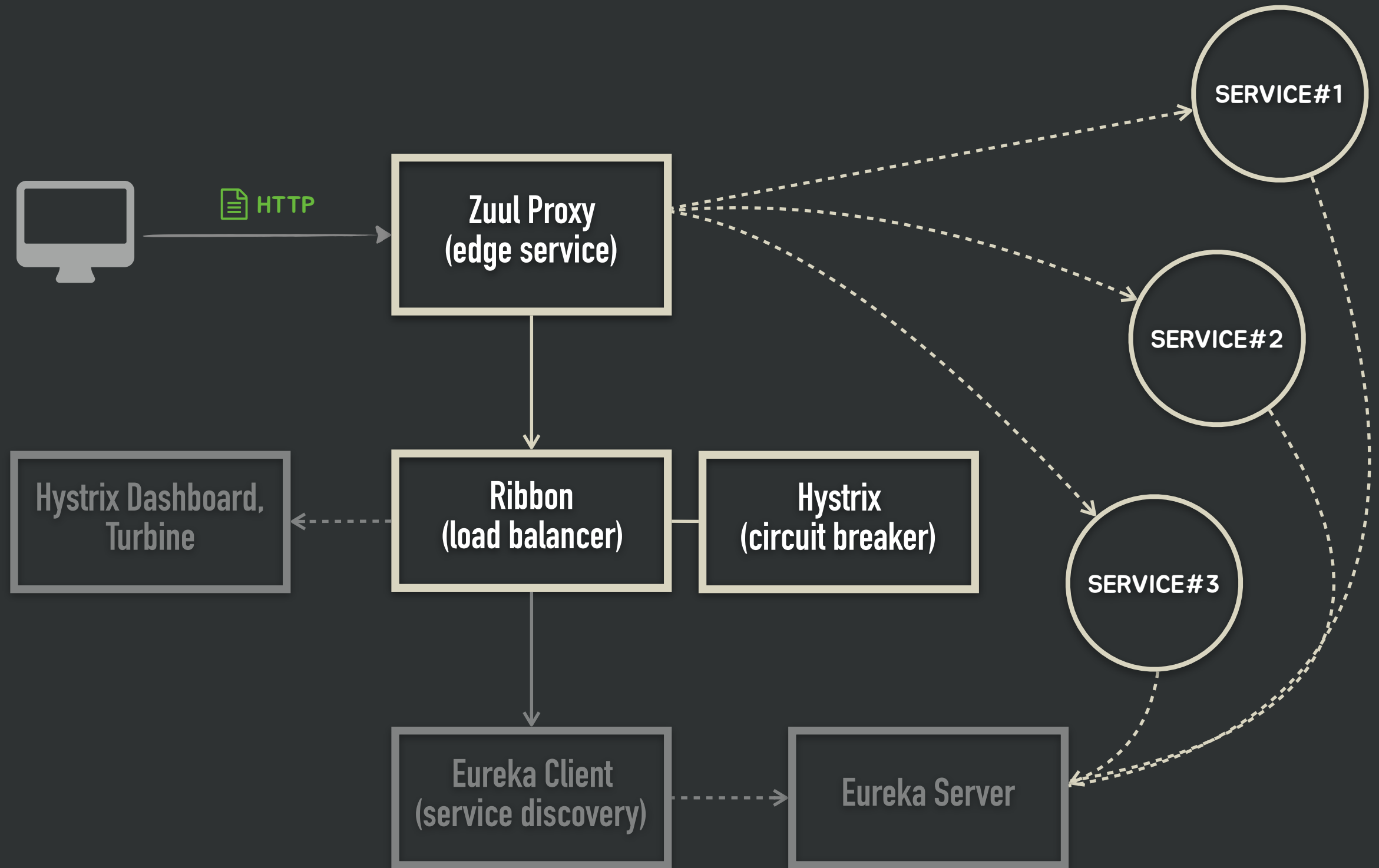
Spring Cloud Netflix features:

- Service Discovery: Eureka instances can be registered and clients can discover the instances using Spring-managed beans

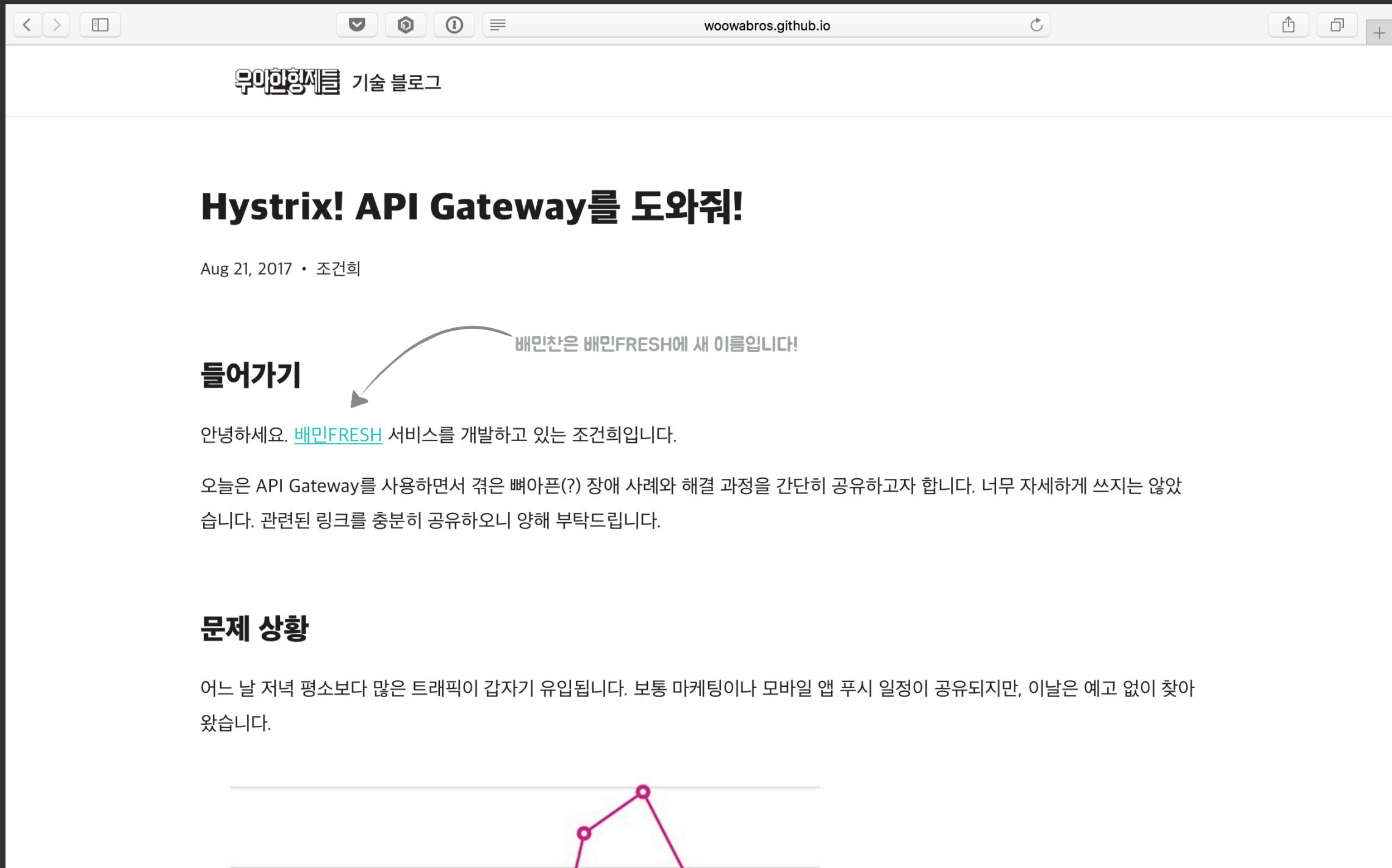
Spring Cloud Netflix	
RELEASE	DOCUMENTATION
2.0.2 CURRENT	Reference API

Fork me on GitHub

스프링 클라우드 넷플릭스(Spring Cloud Netflix)

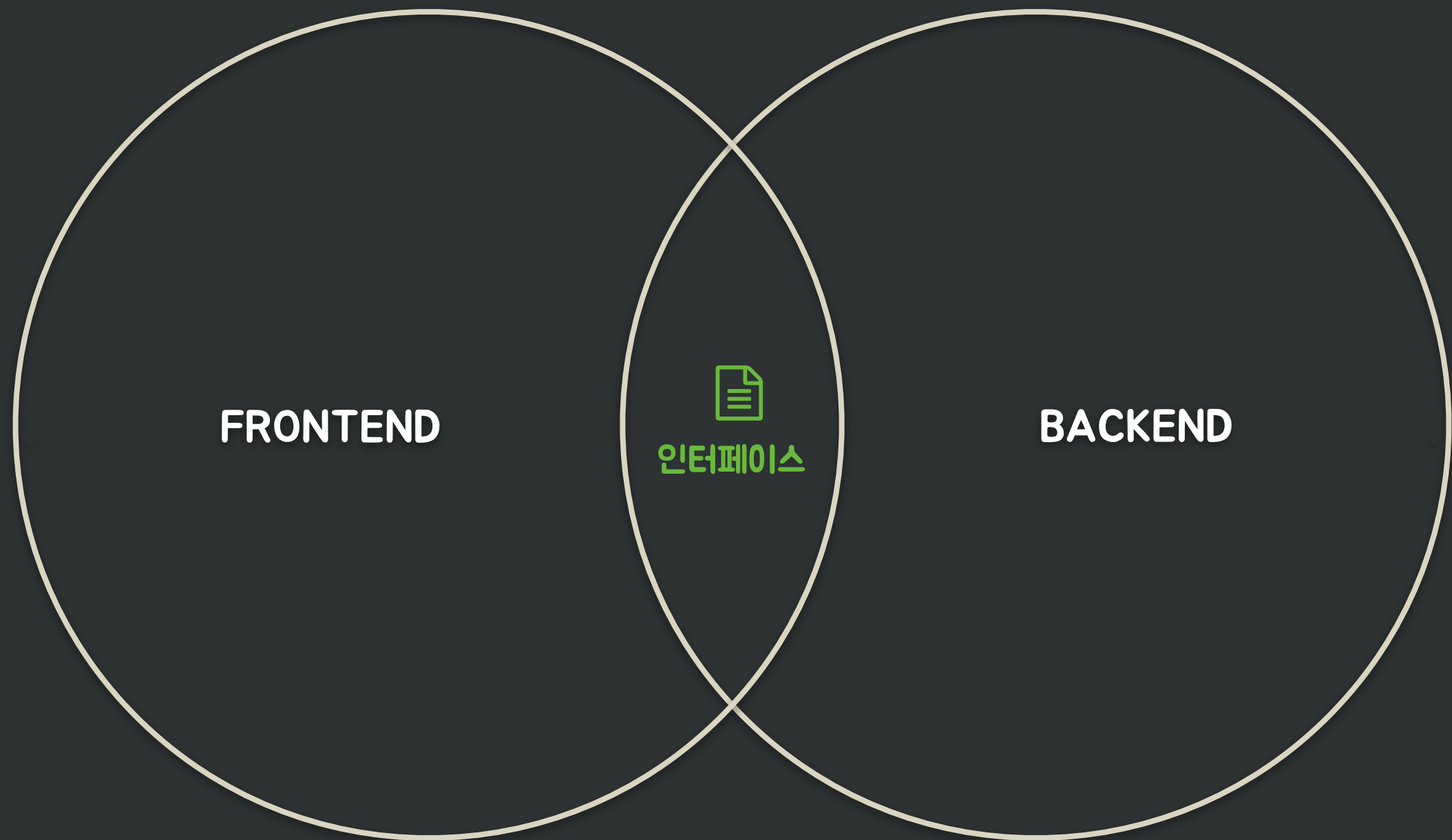


🌀 Hystrix! API Gateway를 도와줘!



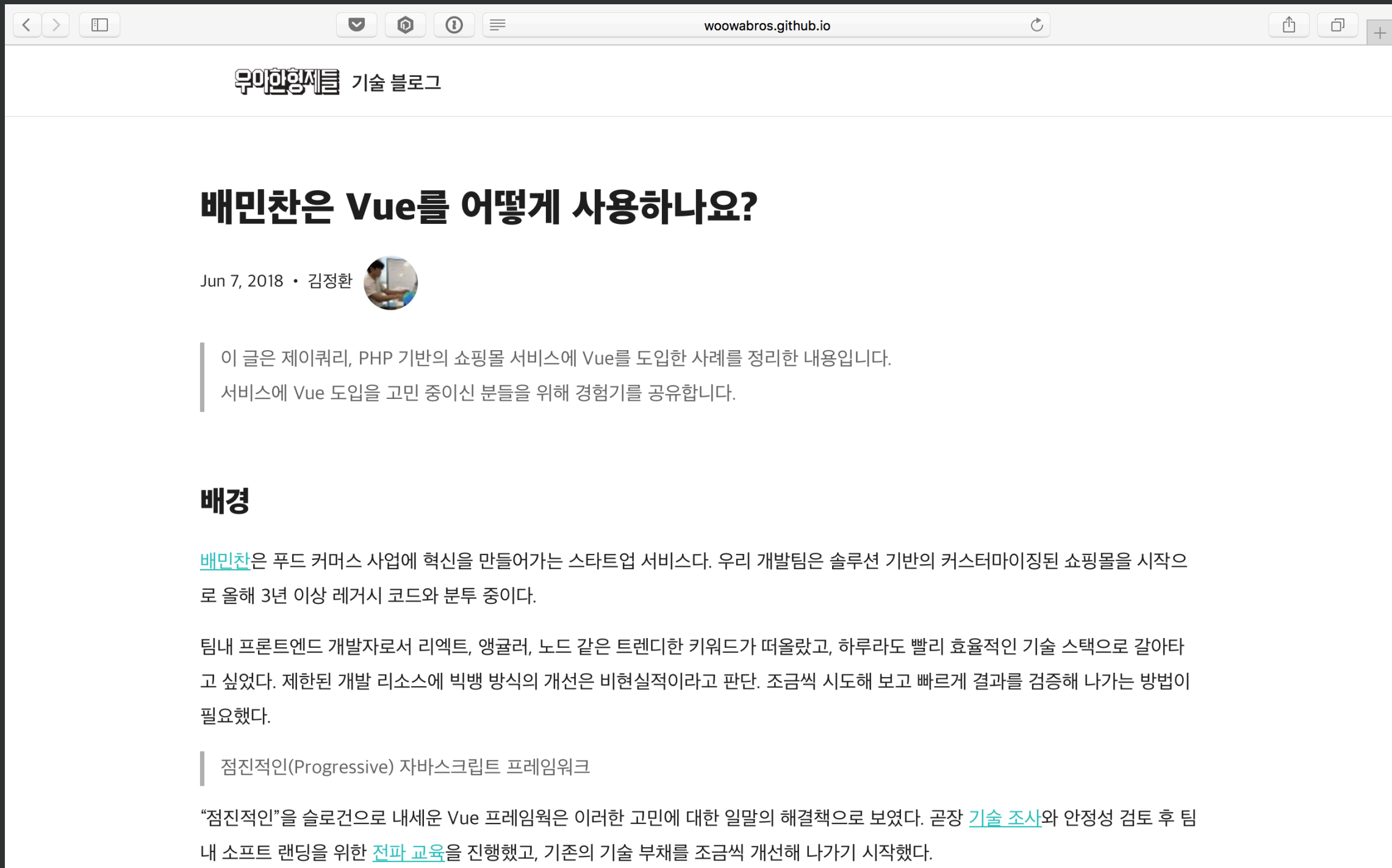
<http://woowabros.github.io/experience/2017/08/21/hystrix-tunning.html>

⋮ 양방향 영향없이 고유영역을 지속적으로 개선 가능



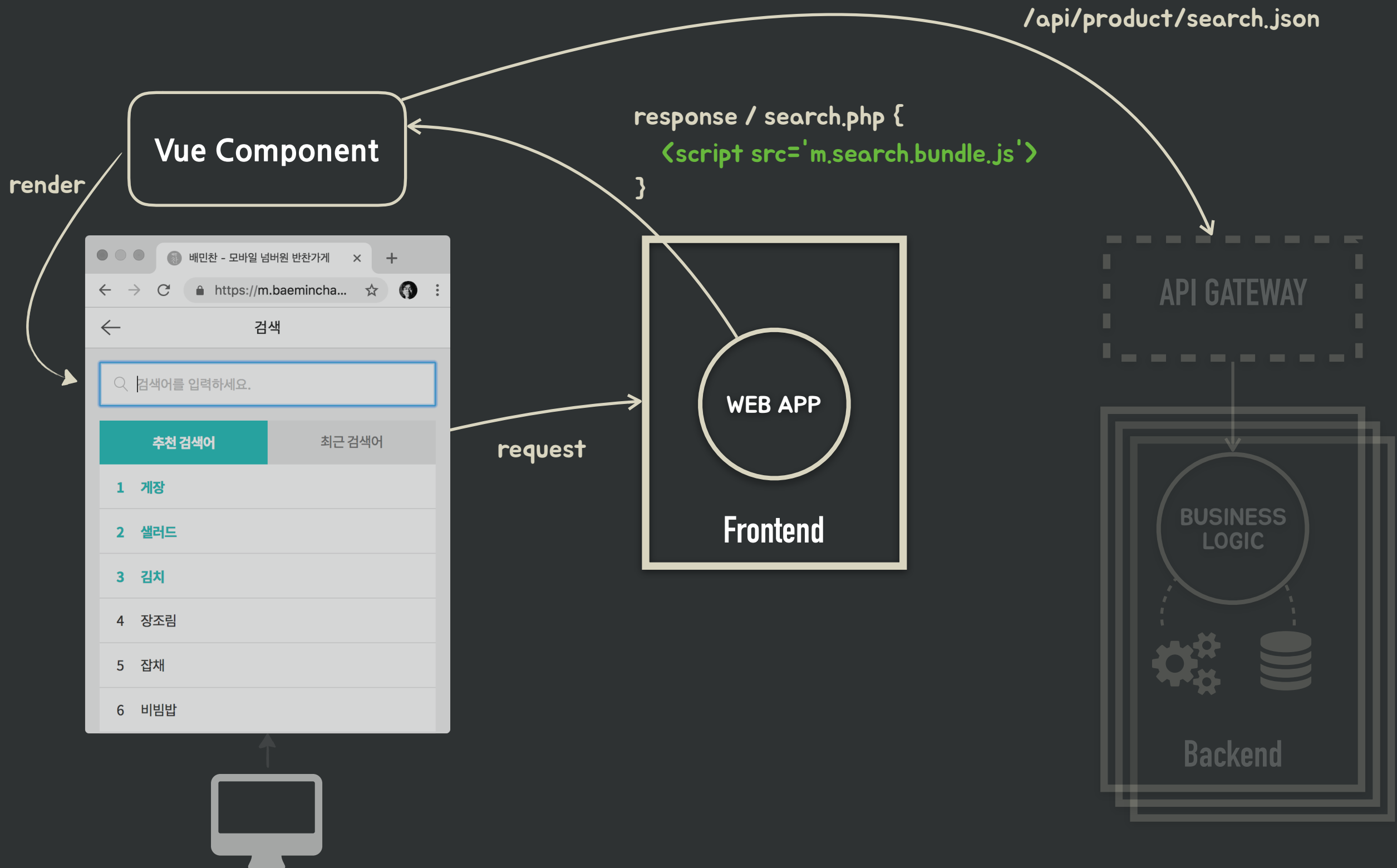
인터페이스만 잘 지킨다면, 서로에게 영향을 주지 않고 내부 구조 개선과 코드를 다듬는 리팩토링이 가능하다.

두번째 결정, 점진적인 자바스크립트 프레임워크

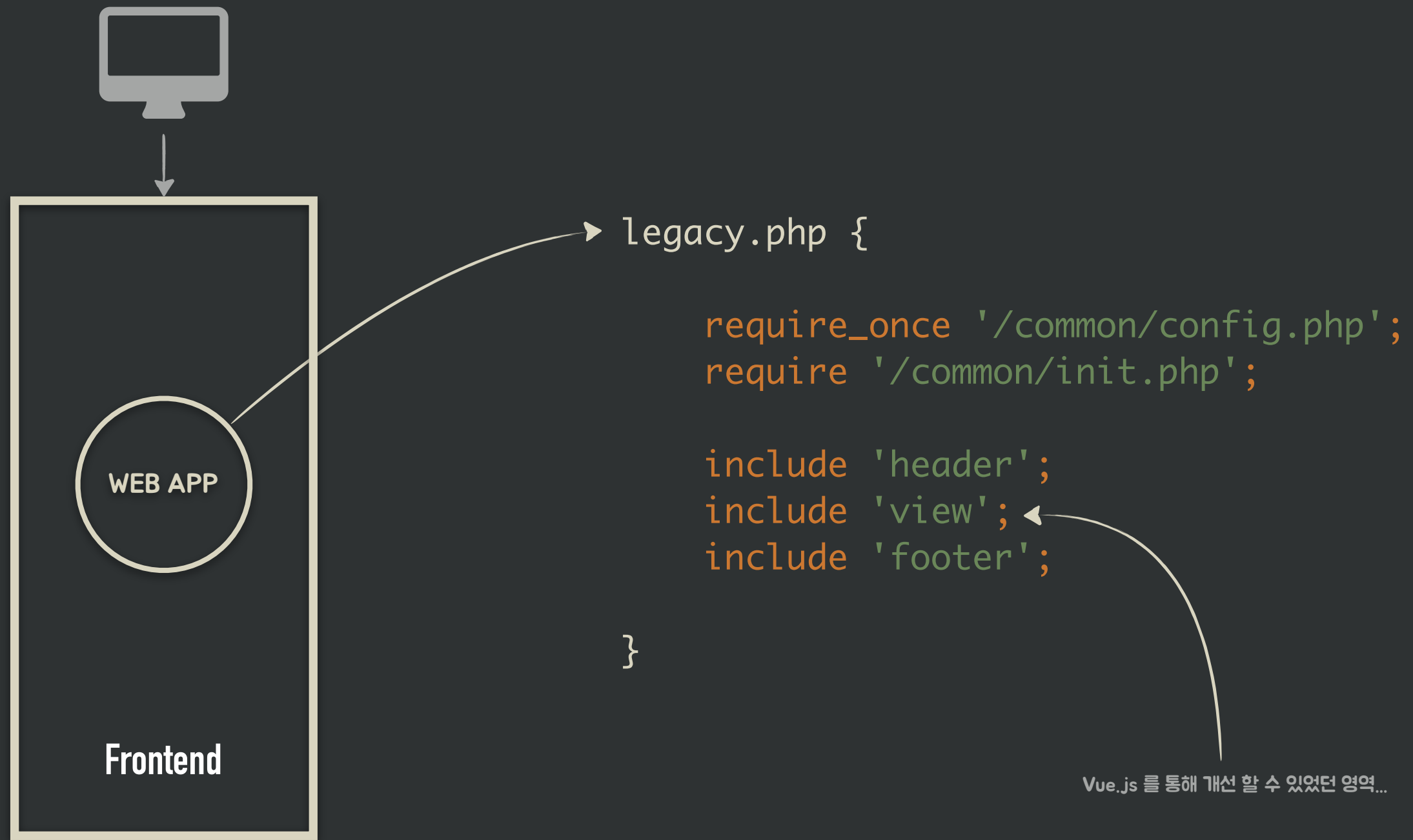


<http://woowabros.github.io/experience/2018/06/07/vue-story-of-baminchan.html>

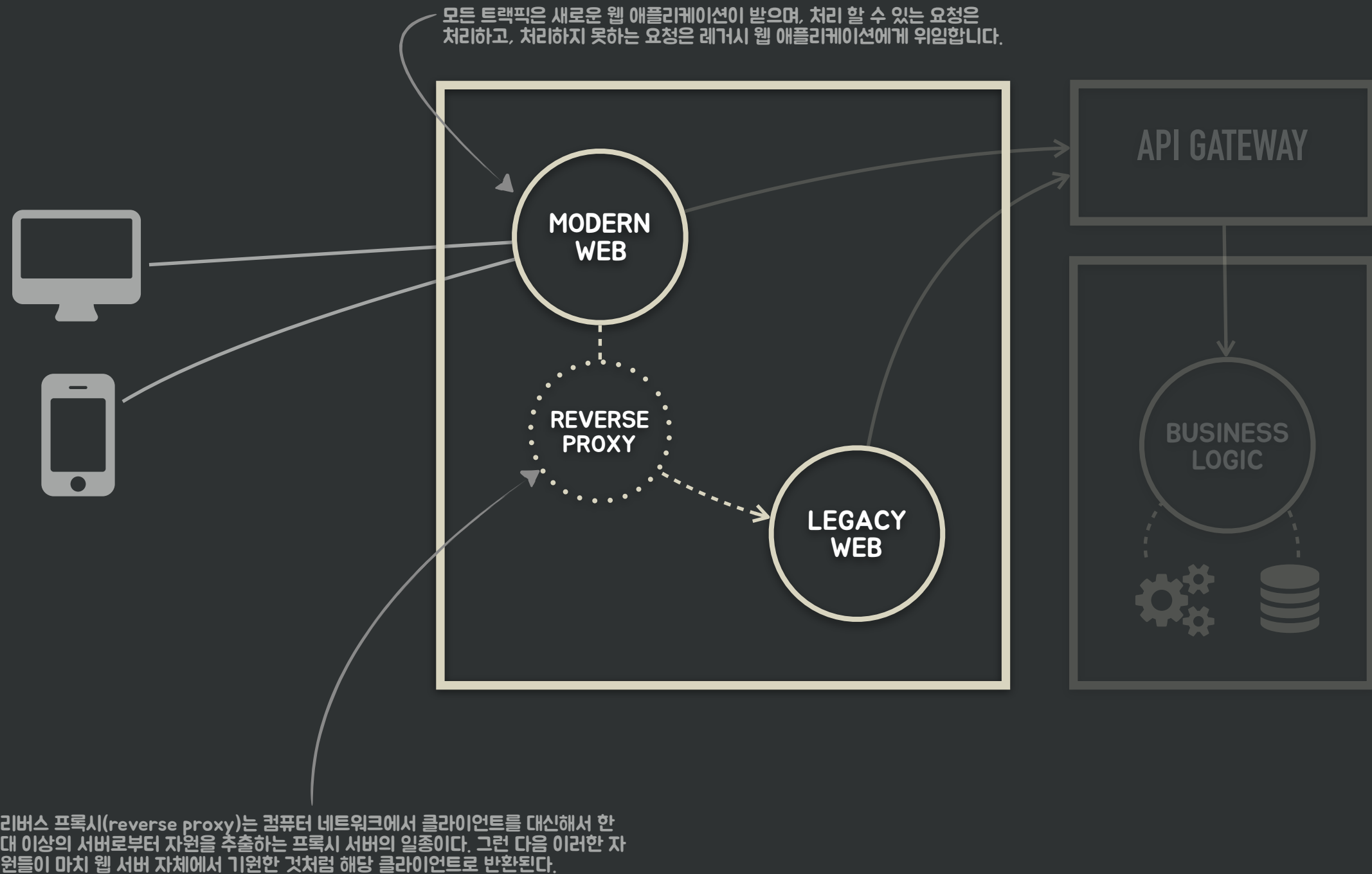
클라이언트 사이드 렌더링과 컴포넌트 기반 뷰 구현



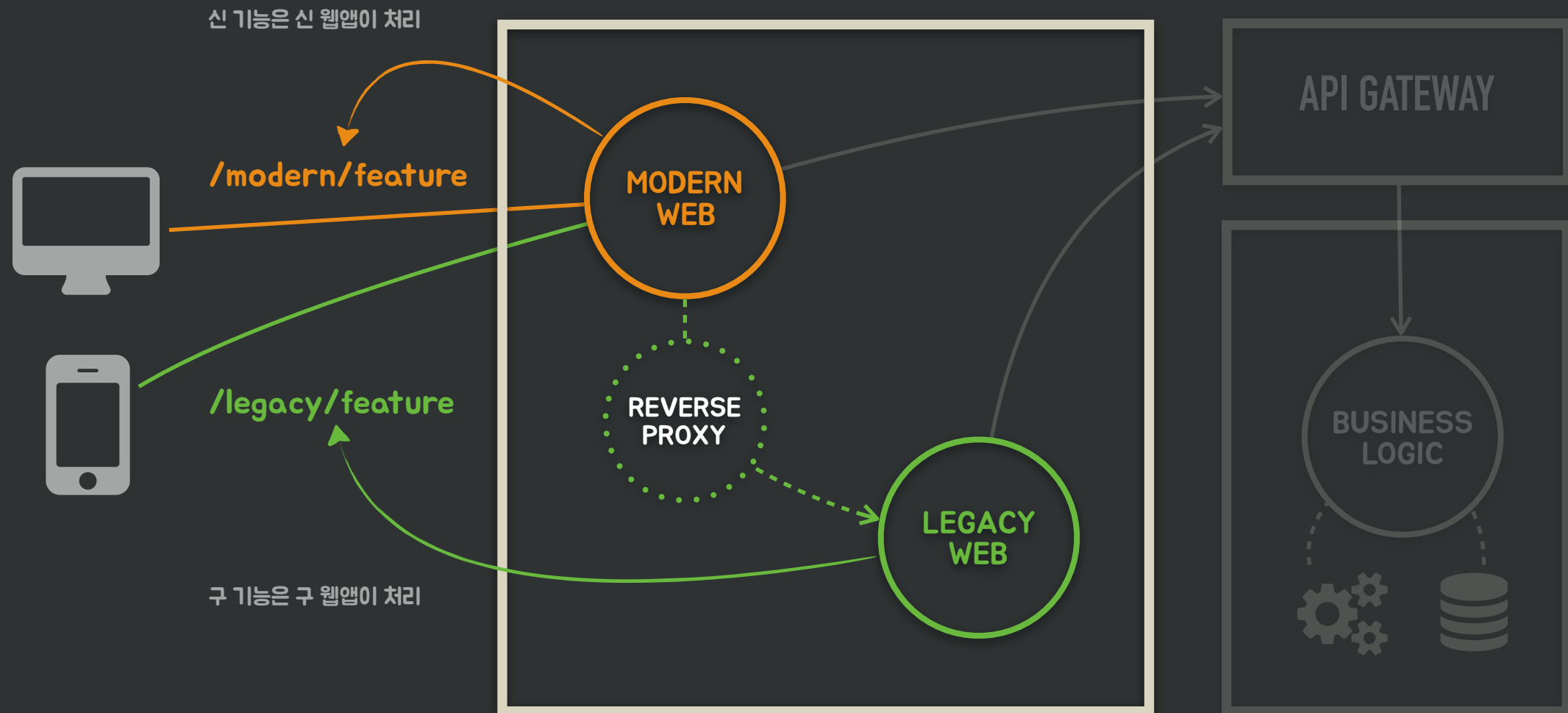
⋮ 아직 남아 있는 애플리케이션 구조적 문제



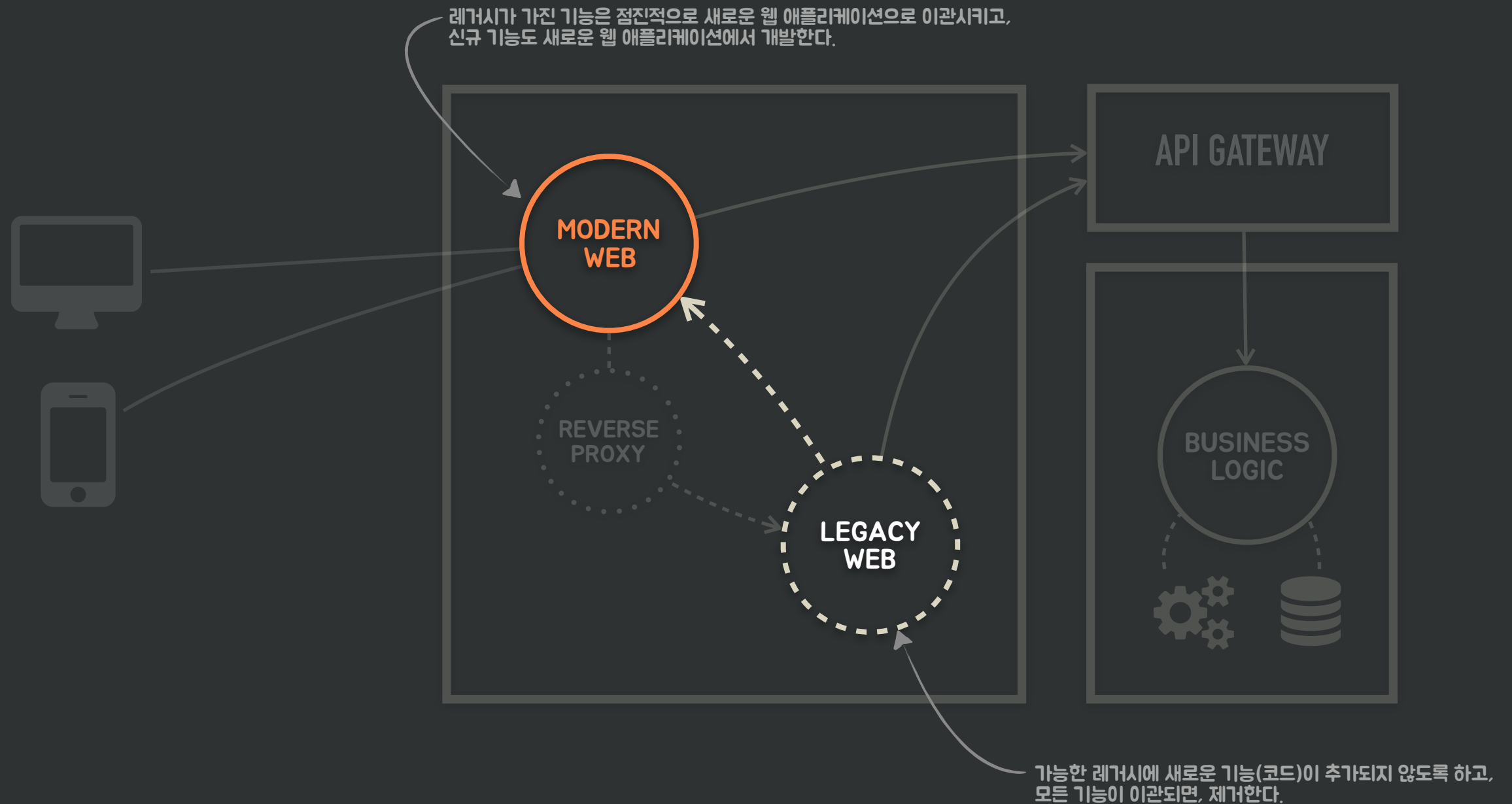
세번째 결정, 새로운 웹앱과 리버스 프록시(Reverse Proxy)



세번째 결정, 새로운 웹앱과 리버스 프록시 (Reverse Proxy)



◉ 점진적으로 기능을 새로운 웹앱으로 옮기기

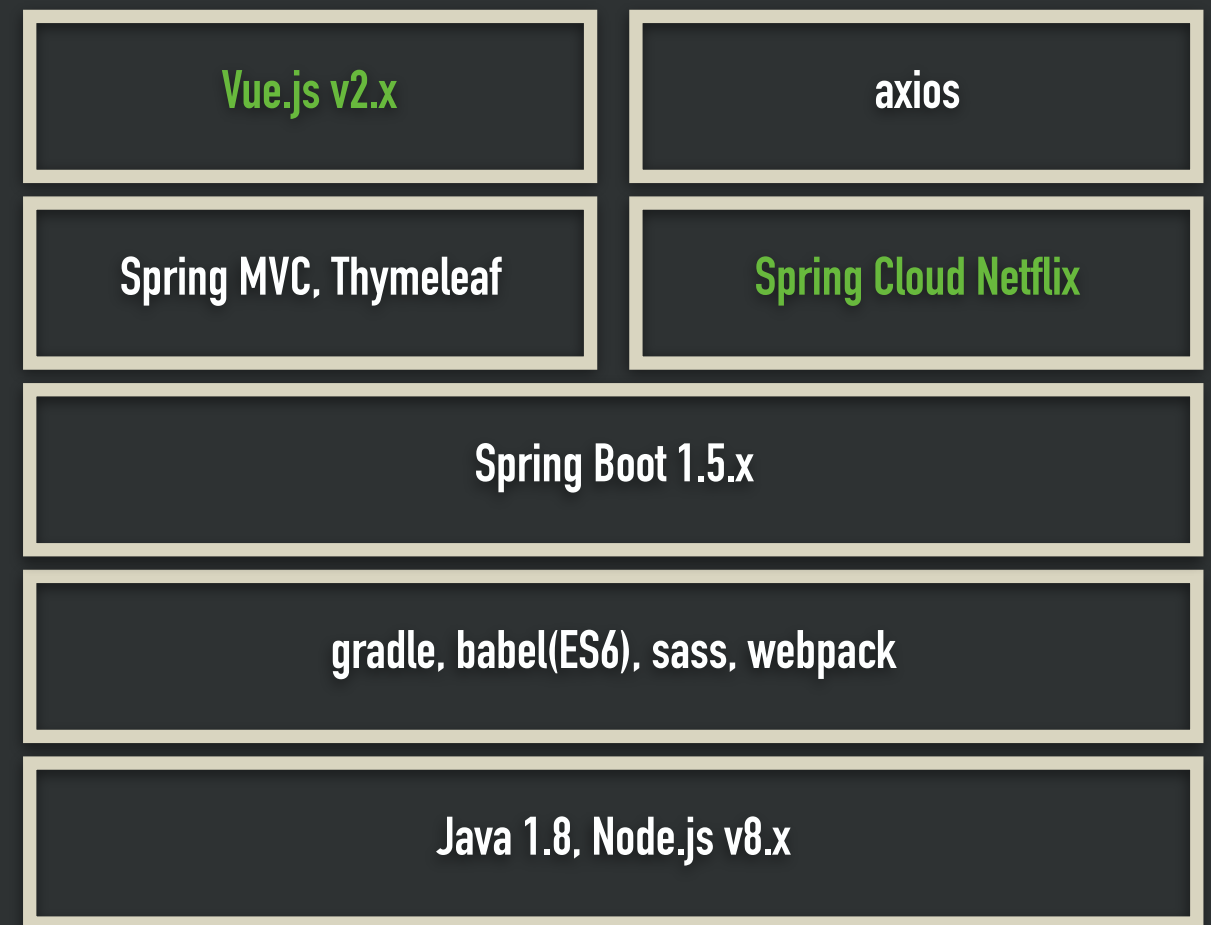


🌀 두가지 프로토타입(prototype) 결과물

Node.js 기반으로 쌓아 올린 애플리케이션 구조



Java 기반으로 쌓아 애플리케이션 구조

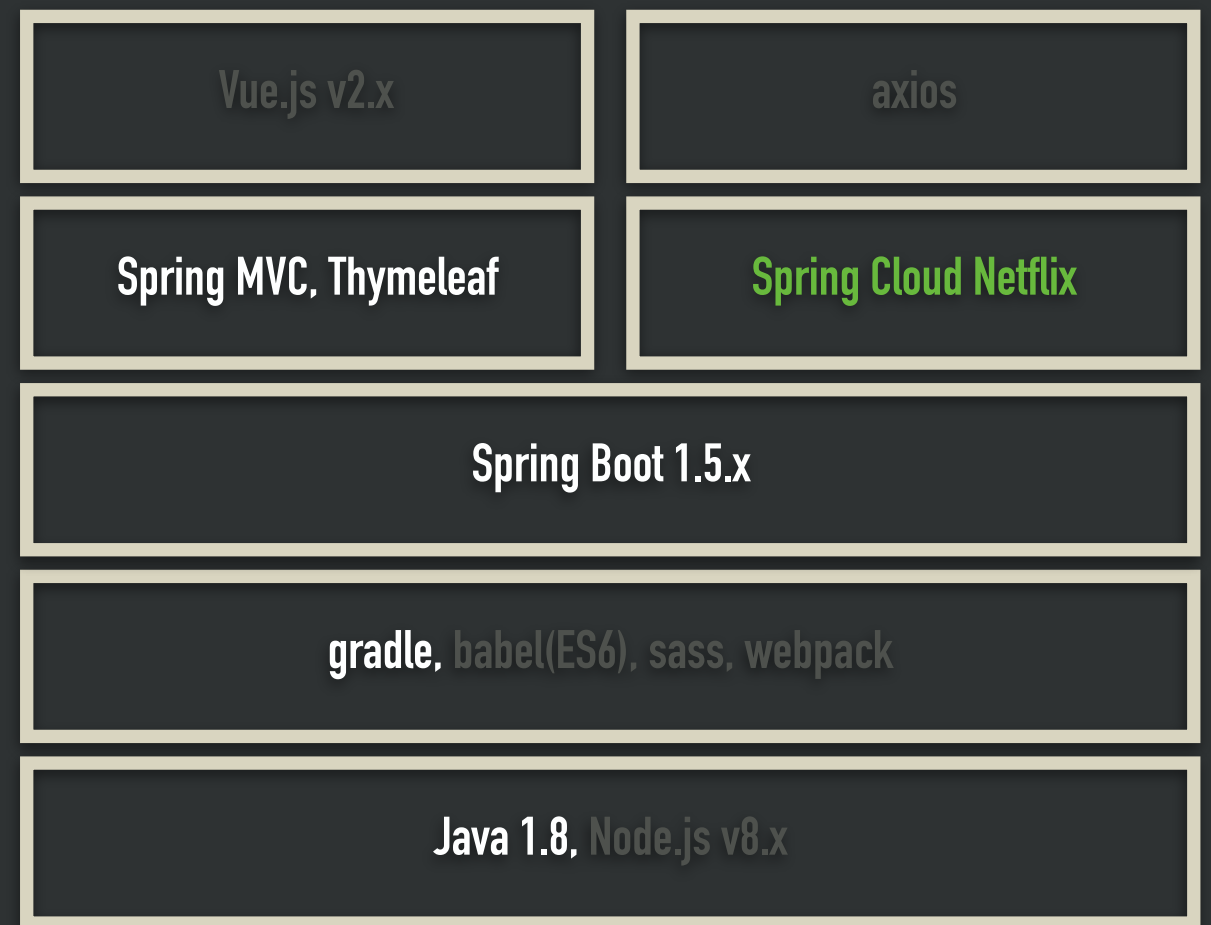


◉ 두가지 프로토타입(prototype) 결과물

Node.js 기반으로 쌓아 올린 애플리케이션 구조



Java 기반으로 쌓아 애플리케이션 구조



서버사이드(Server-side) 애플리케이션 프레임워크와 개발 환경

🌀 두가지 프로토타입 (prototype) 결과물

Node.js 기반으로 쌓아 올린 애플리케이션 구조



Java 기반으로 쌓아 애플리케이션 구조



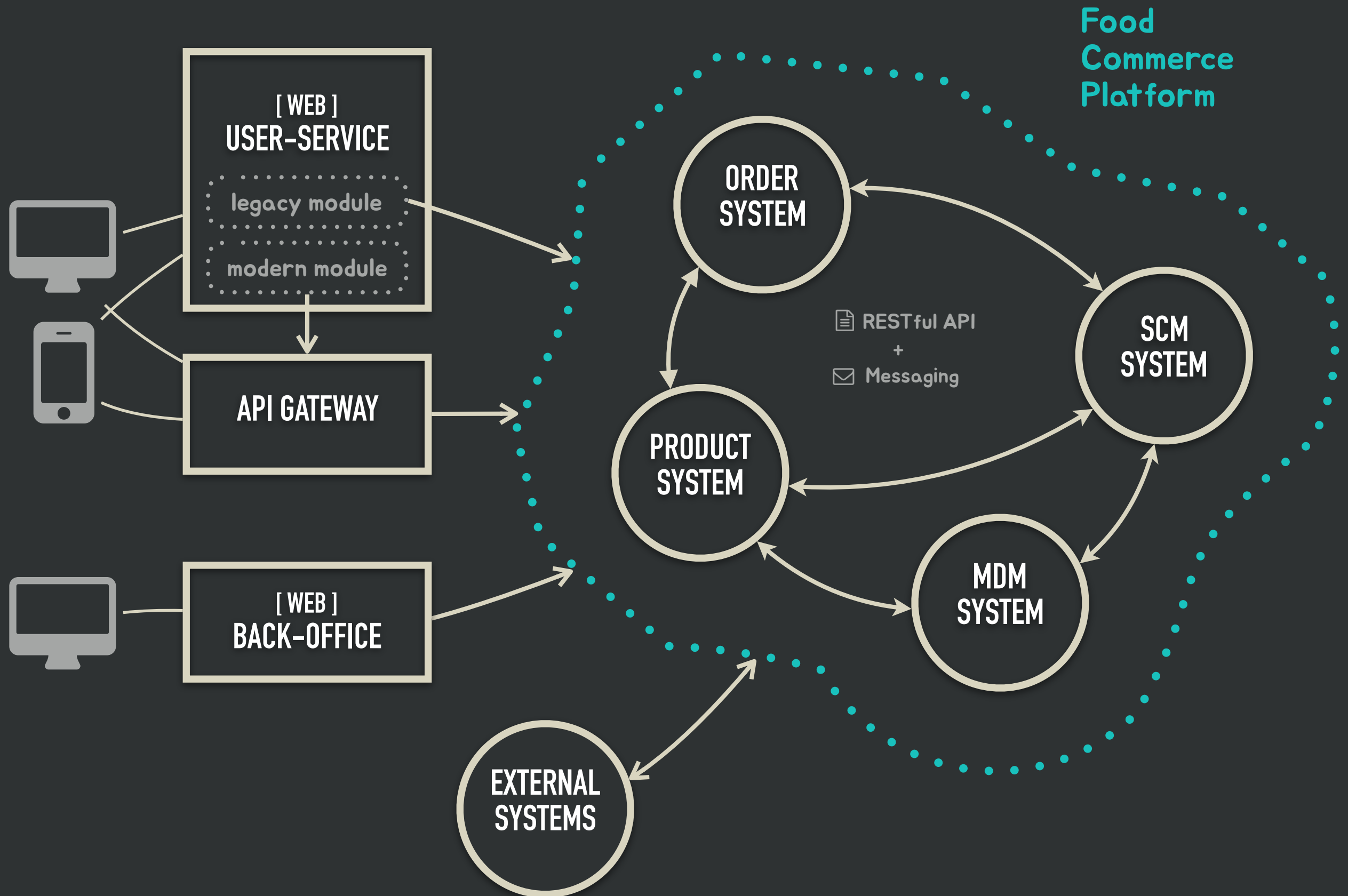
클라이언트사이드(Client-side) 애플리케이션 프레임워크와 개발 환경

◉ 개발자는 코드로 대화한다...구요?

DEMO

<https://github.com/arawn/continual-improve-legacy-web>

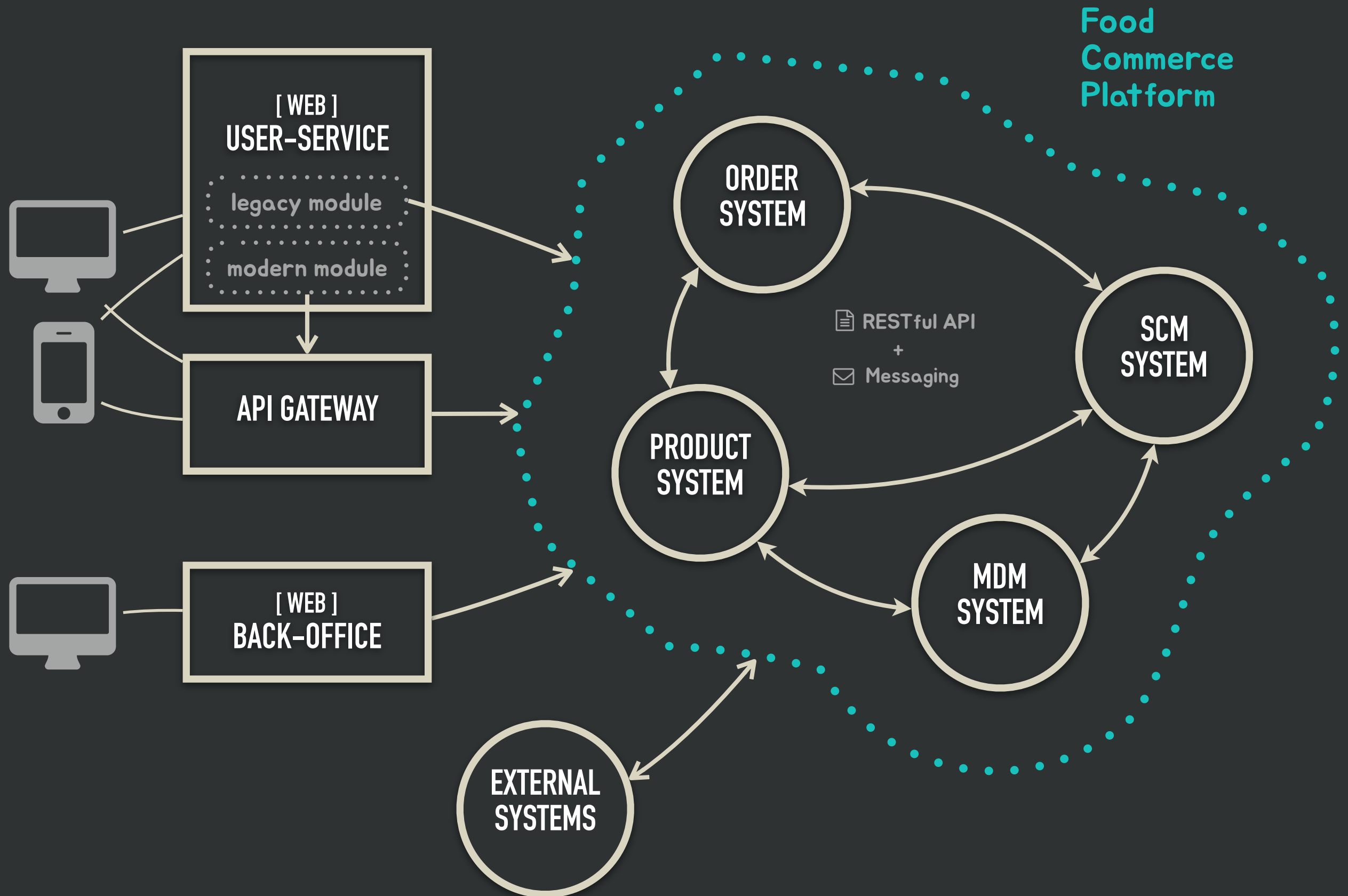
2018년, 1만km 상공에서 배민찬 시스템 굽어보기



◉ 쉬어 가는 곳



2018년, 1만km 상공에서 배민찬 시스템 굽어보기



◉ 서비스 운영환경에 따라 달라지는 환경설정

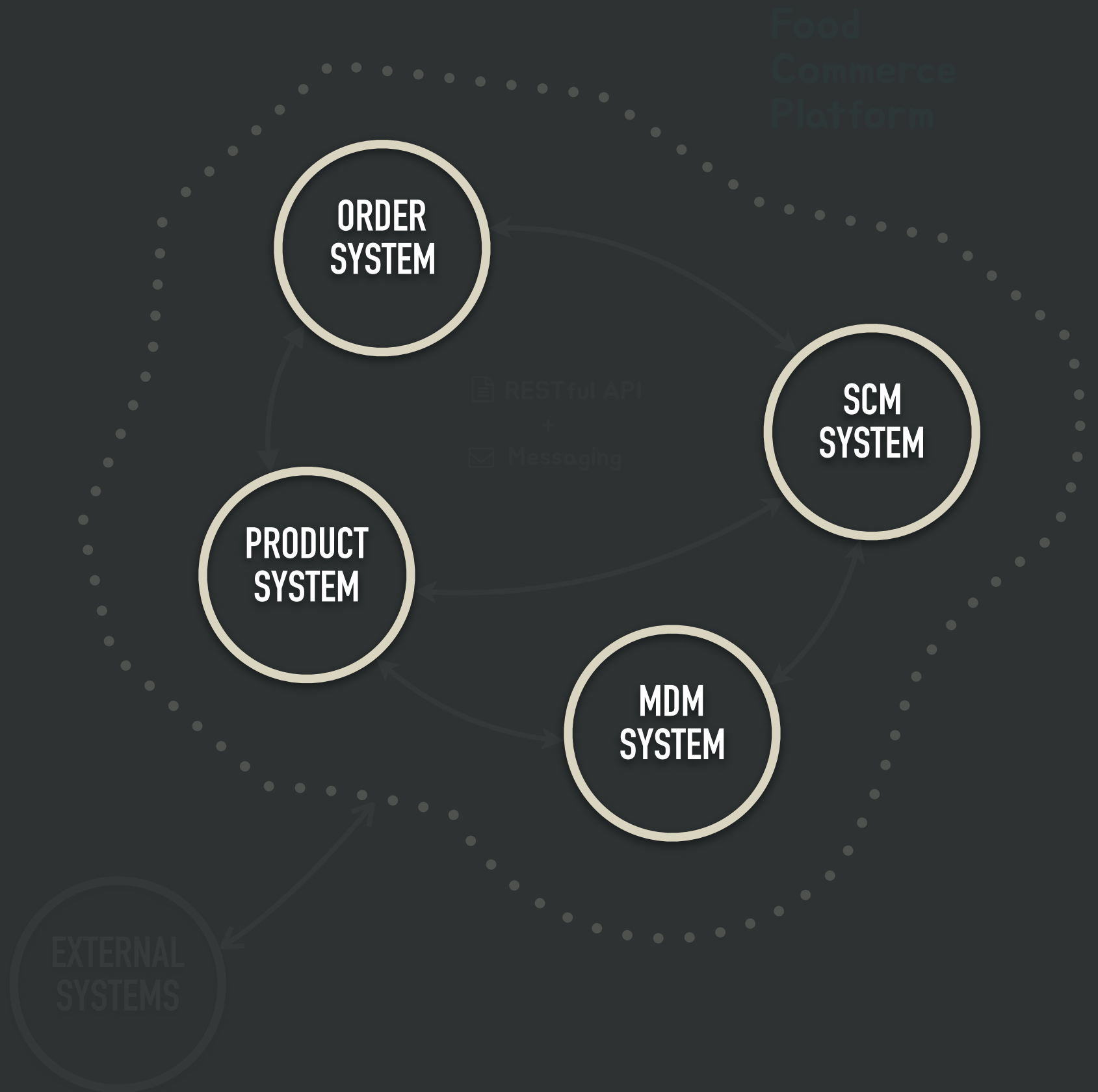
local.baeminchan.com

dev.baeminchan.com

test.baeminchan.com

staging.baeminchan.com

production.baeminchan.com



⦿ 환경에 저장된 설정 :12 요소 애플리케이션(The Twelve-Factor App)

애플리케이션 코드는 설정 정보(configuration)와 엄격하게 분리되어야 한다. 애플리케이션의 설정 정보는 실행 환경에서 읽어올 수 있어야 한다. 실행 환경에서 달라지는 값은 아래 그림과 같이 환경설정 정보로 분류하고, 그 환경에 저장돼야 하고, 애플리케이션에 포함하지 않아야 한다.

실행 환경

설정 정보

코드베이스



스프링 부트 프로파일(Spring Boot Profiles) 활용한 환경구성

```
> java -jar application.jar --spring.profiles.active=dev,ec2
```

```
└─ resources
    ├── application.yml           # 공통 설정
    ├── application-local.yml     # 로컬 환경설정 파일
    ├── application-dev.yml       # 개발서버 환경설정 파일
    ├── application-test.yml      # 테스트서버 환경설정 파일
    ├── application-staging.yml   # 스테이징서버 환경설정 파일
    ├── application-production.yml # 프로덕션서버 환경설정 파일
    └─ application-ec2.yml        # AWS EC2 환경설정 파일
```

애플리케이션이 기동하는데 필요한 환경변수를 프로파일별로 관리하고, 기동시 활성화할 프로파일을 지정한다.
이 환경설정 파일들은 형상관리 대상으로 보안에 민감한 값(데이터베이스 계정 정보 류)은 포함하지 않는다.

```
+ OS environment variables (AWS Elastic Beanstalk) {
```

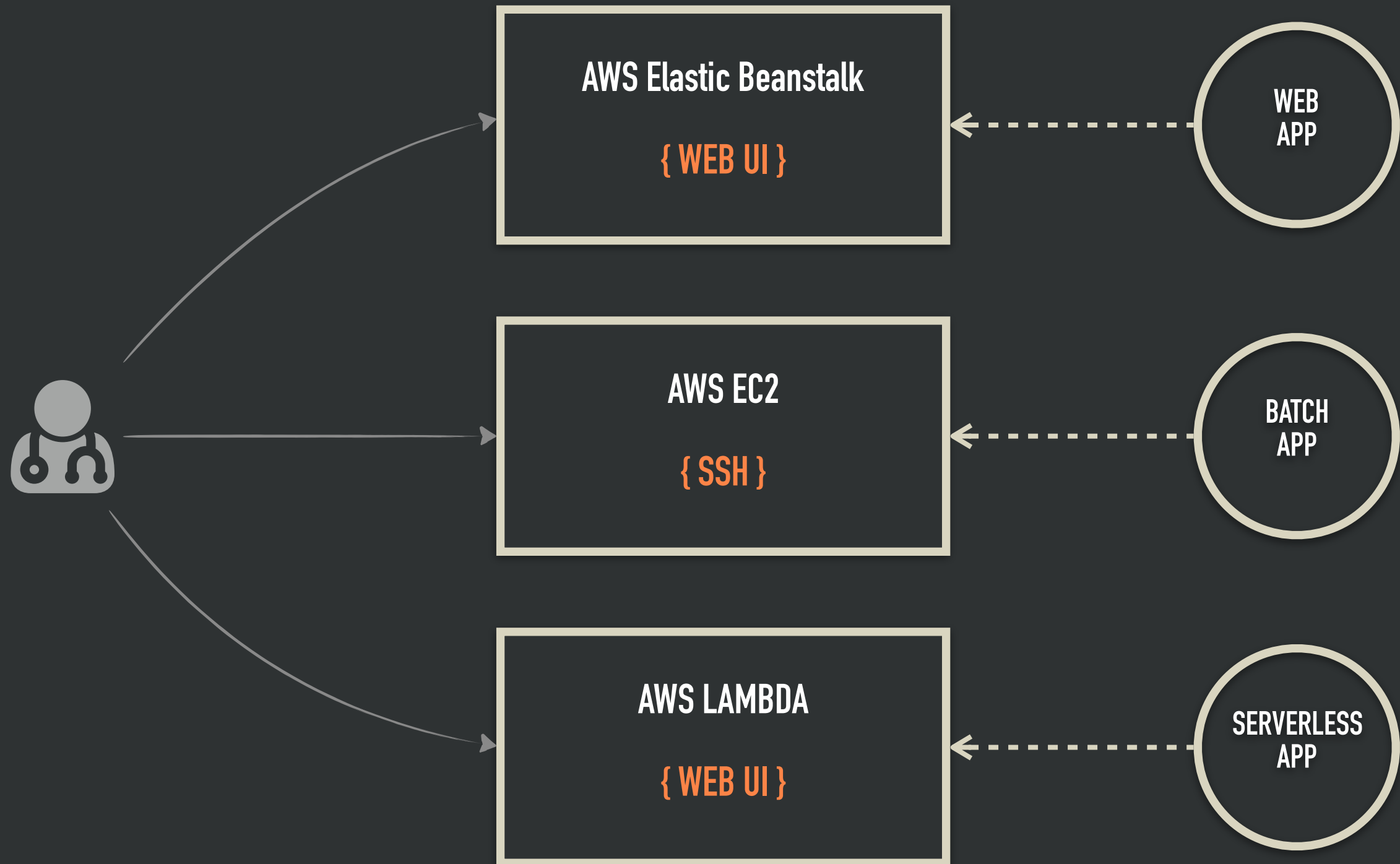
```
    export DB_USERNAME="order_service"
    export DB_PASSWORD="7%wo9;ttArRYMfYp"
```

```
}
```

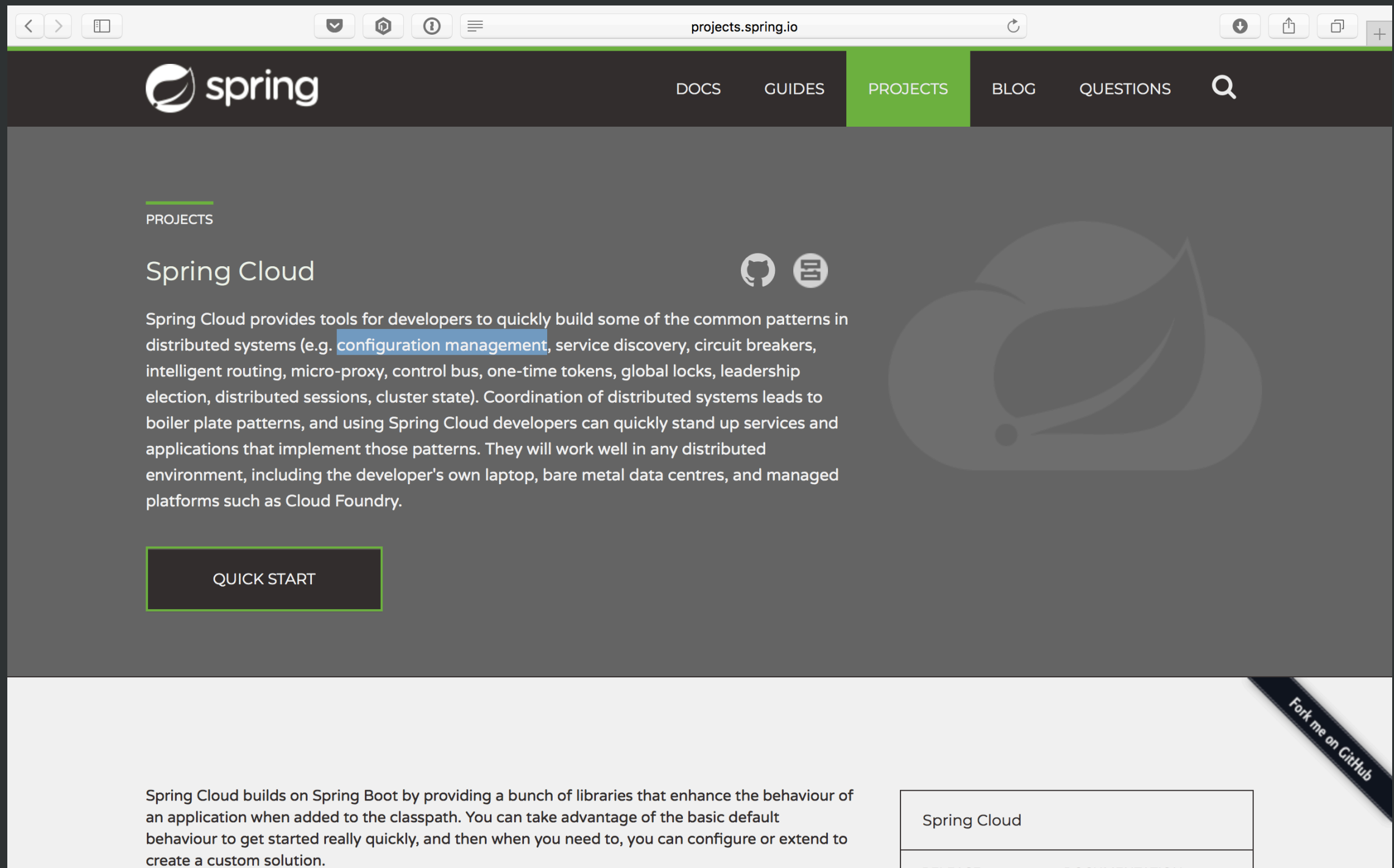
보안에 민감한 정보는 OS 환경변수를 통해 애플리케이션에 제공한다.
또한 스프링 부트에 환경구성 전략에 따라 OS 환경변수가 우선순위가 높으므로 필요에 따라 application*.yml 내부에 설정을 덮어씌울 수 있다.

실행 환경별 환경변수 관리에 문제점과 불편함

환경변수를 한곳에서 관리 할 수 없고, 여러곳에서 다양한 형태로 관리해야 하며, 필요에 따라 배포 또는 서버 재시작이 필요하다.



스프링 클라우드(Spring Cloud) 설정 정보 관리 지원



The screenshot shows the Spring Cloud project page on the projects.spring.io website. The browser's address bar displays 'projects.spring.io'. The website's navigation bar includes the Spring logo and links for DOCS, GUIDES, PROJECTS (which is highlighted), BLOG, and QUESTIONS, along with a search icon. On the left, a 'PROJECTS' section lists 'Spring Cloud'. The main content area features the Spring Cloud title, GitHub and Maven icons, and a detailed description of the framework's capabilities. A 'QUICK START' button is prominently displayed. At the bottom, a paragraph explains that Spring Cloud builds on Spring Boot. A table at the bottom right lists 'Spring Cloud' with links for 'RELEASE' and 'DOCUMENTATION'. A diagonal banner in the bottom right corner encourages users to 'Fork me on GitHub'.

spring

DOCS GUIDES **PROJECTS** BLOG QUESTIONS

PROJECTS

Spring Cloud

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

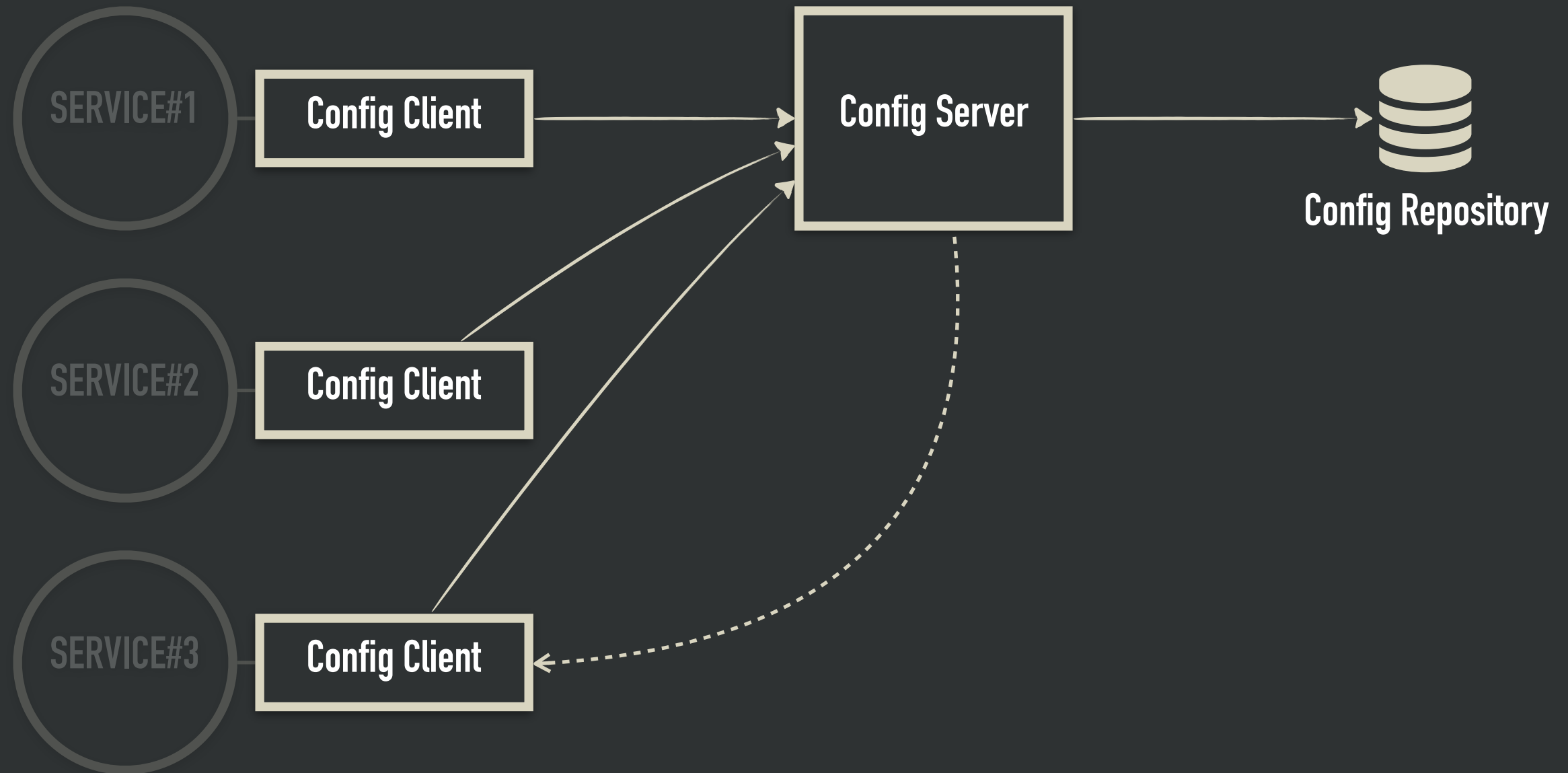
QUICK START

Spring Cloud builds on Spring Boot by providing a bunch of libraries that enhance the behaviour of an application when added to the classpath. You can take advantage of the basic default behaviour to get started really quickly, and then when you need to, you can configure or extend to create a custom solution.

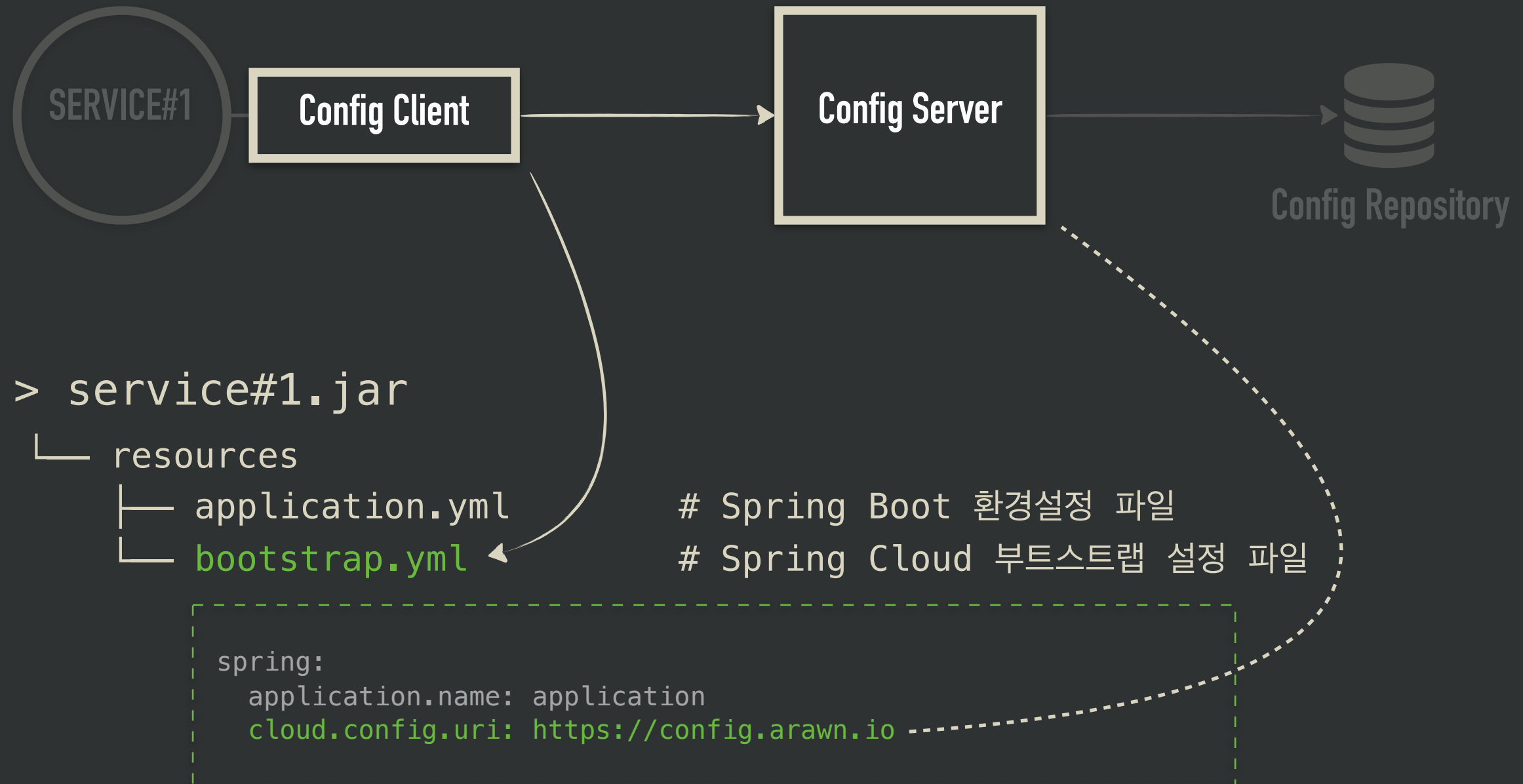
Spring Cloud
RELEASE DOCUMENTATION

Fork me on GitHub

스프링 클라우드(Spring Cloud) 설정 정보 관리 구조



스프링 클라우드(Spring Cloud) 설정 정보 관리 동작 방식



스프링 클라우드는 스프링 애플리케이션이 생성되기 전에 먼저 설정 서버로부터 필요한 환경설정 정보를 획득한다. 이때 사용되는 특별한 설정파일이 `bootstrap.yml` (또는 `properties`) 이다.

스프링 클라우드(Spring Cloud) 설정 정보 관리 프로젝트

Spring Cloud Config

Centralized external configuration management backed by a git repository. The configuration resources map directly to Spring `Environment` but could be used by non-Spring applications if desired.

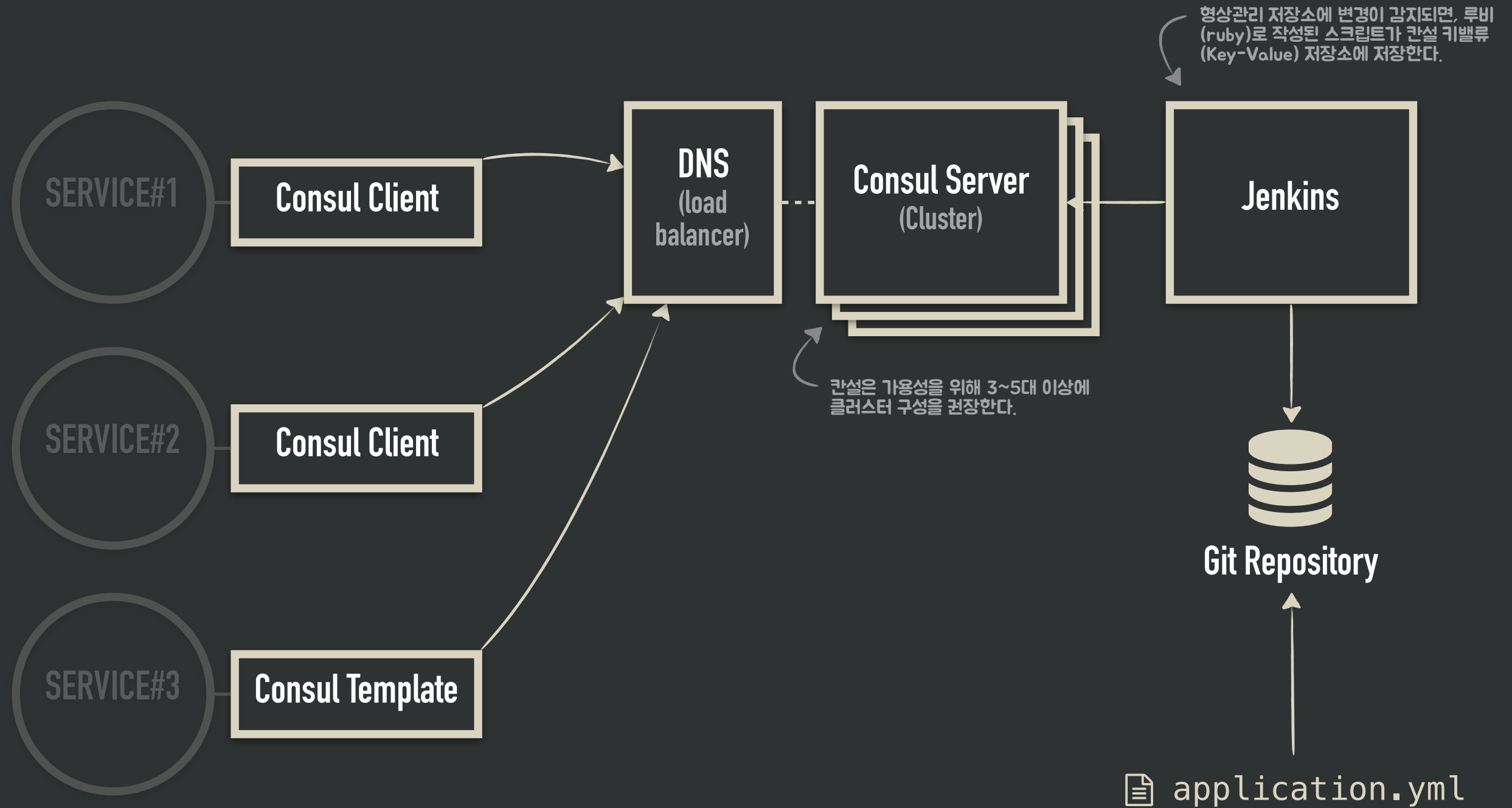
Spring Cloud Consul

Service discovery and configuration management with Hashicorp Consul.

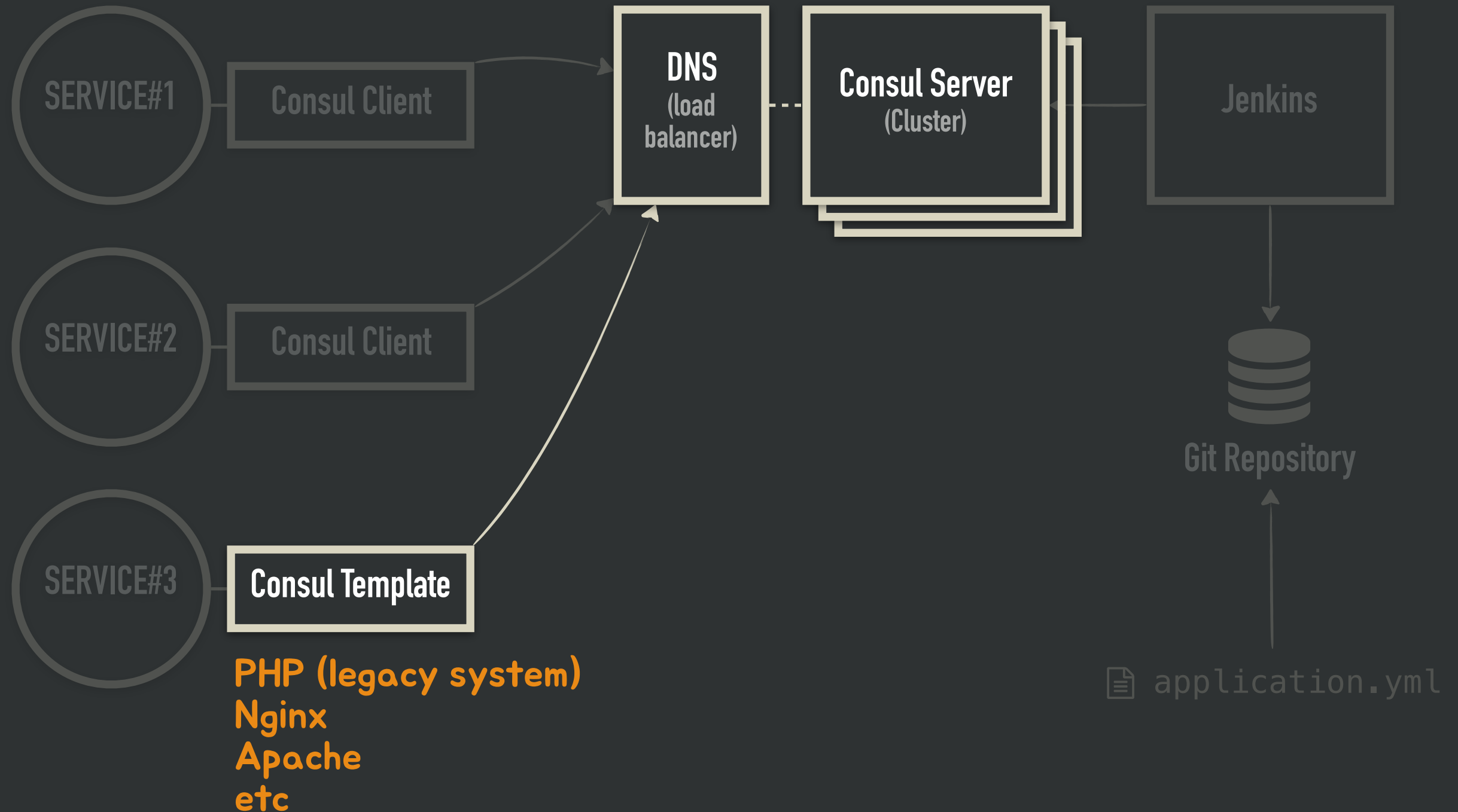
Spring Cloud Zookeeper

Service discovery and configuration management with Apache Zookeeper.

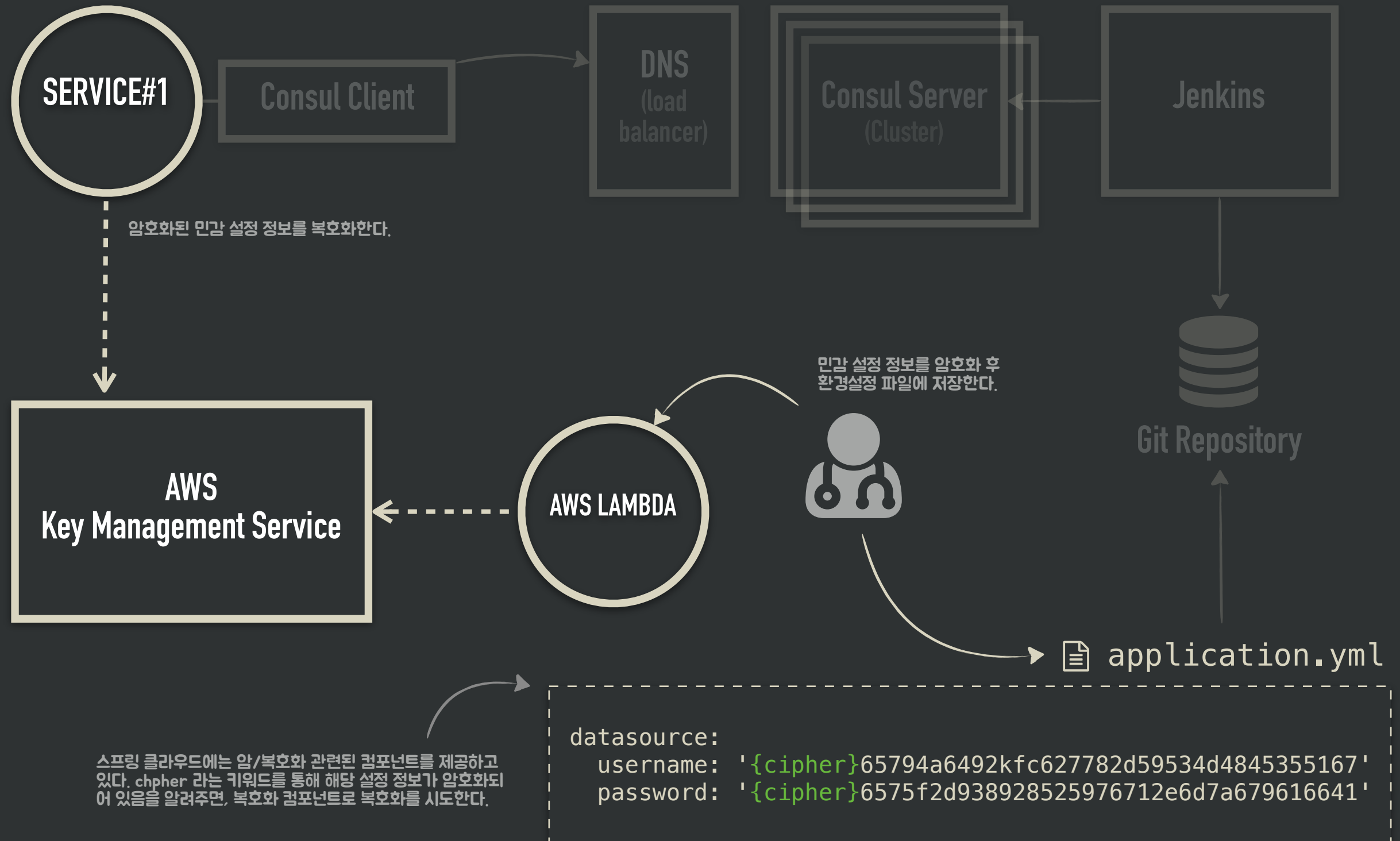
칸설(Consul)로 구성한 환경설정 외부화



:: 컨실 템플릿으로 여러 형태에 환경설정 외부화 구성



민감 설정 정보를 다루기 위한 암호/복호화 처리



과정은 끝나지 않았다-

완전히 뚜렷하게 구분되거나 또는 궁극이라고 말할 수 있는 마이크로서비스는 세상에 없다.
마이크로서비스는 일종의 여행이고, 날마다 성숙하고 있는 **진화 과정**이다.

도서, '스프링 마이크로서비스'에서 발췌...

이 과정은 가치있게 사용되는 시스템을 만드는 것

software maintenance is not 'keep it **working** like before'
It is 'keep it being **useful** in a changing world'

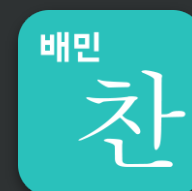
Jessica Keer, Atomist

<https://twitter.com/jessitron/status/728270662414934016>

소프트웨어 유지는 '이전과 같이 동작하게끔 유지하는 것'이 아니다
'변화하는 환경 속에서 항상 쓸모있도록 하는 것'이다

꾸
쑈

우아한신선들



🌀 참고자료

- ✓ [스트랭글러 패턴\(strangler patterns\)](#)
- ✓ [스트랭글러 애플리케이션\(strangler application\)](#)
- ✓ [Refactoring a Monolith into Microservices](#)
- ✓ [API 게이트웨이 패턴과 클라이언트](#)
- ✓ [Pattern: API Gateway / Backend for Front-End](#)
- ✓ [Netflix Zuul : Importance of Reverse Proxy in Microservices Architecture](#)
- ✓ [Cloud Native Java](#)
- ✓ [The Twelve-Factor app \(12 요소 애플리케이션\)](#)