

# W A # L E

S T U D I O



# 와플스튜디오 Backend Seminar

Instructor:

강지혁 @Jhvictor4

세미나 5차시

TA: 유진님, 원식님, 도현님

Contributor 변다빈 @bdv111

서울대학교 앱/웹개발 동아리 와플스튜디오

# Seminar 5

서버 프로그래밍의 아주 기본적인 내용들은 대략적으로 훑어본 것 같습니다.

앞으로도 알아보아야 할 내용은 무궁무진

↳ 이후의 WaffleStudio 활동에서 각자의 관심에 맞춰 발전해 나가자

오늘 세미나 내용

1. 배포 자동화

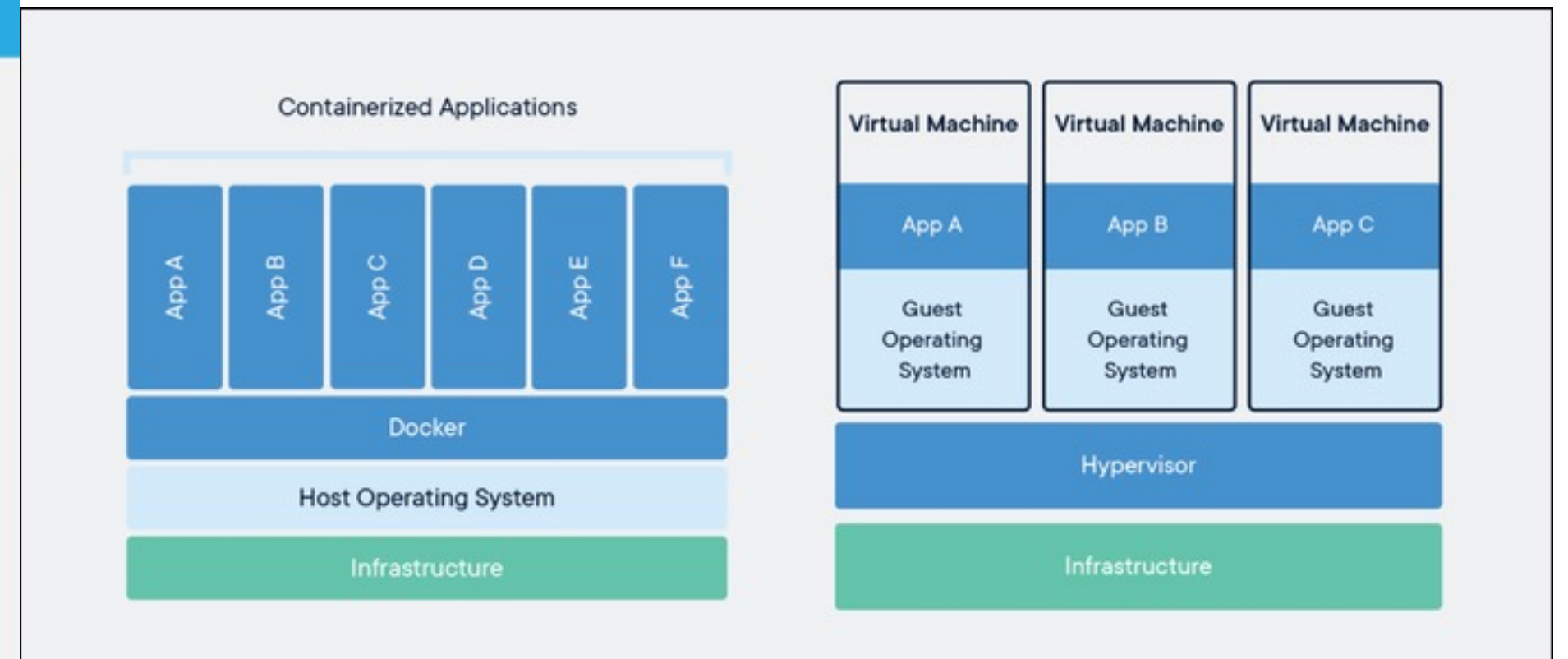
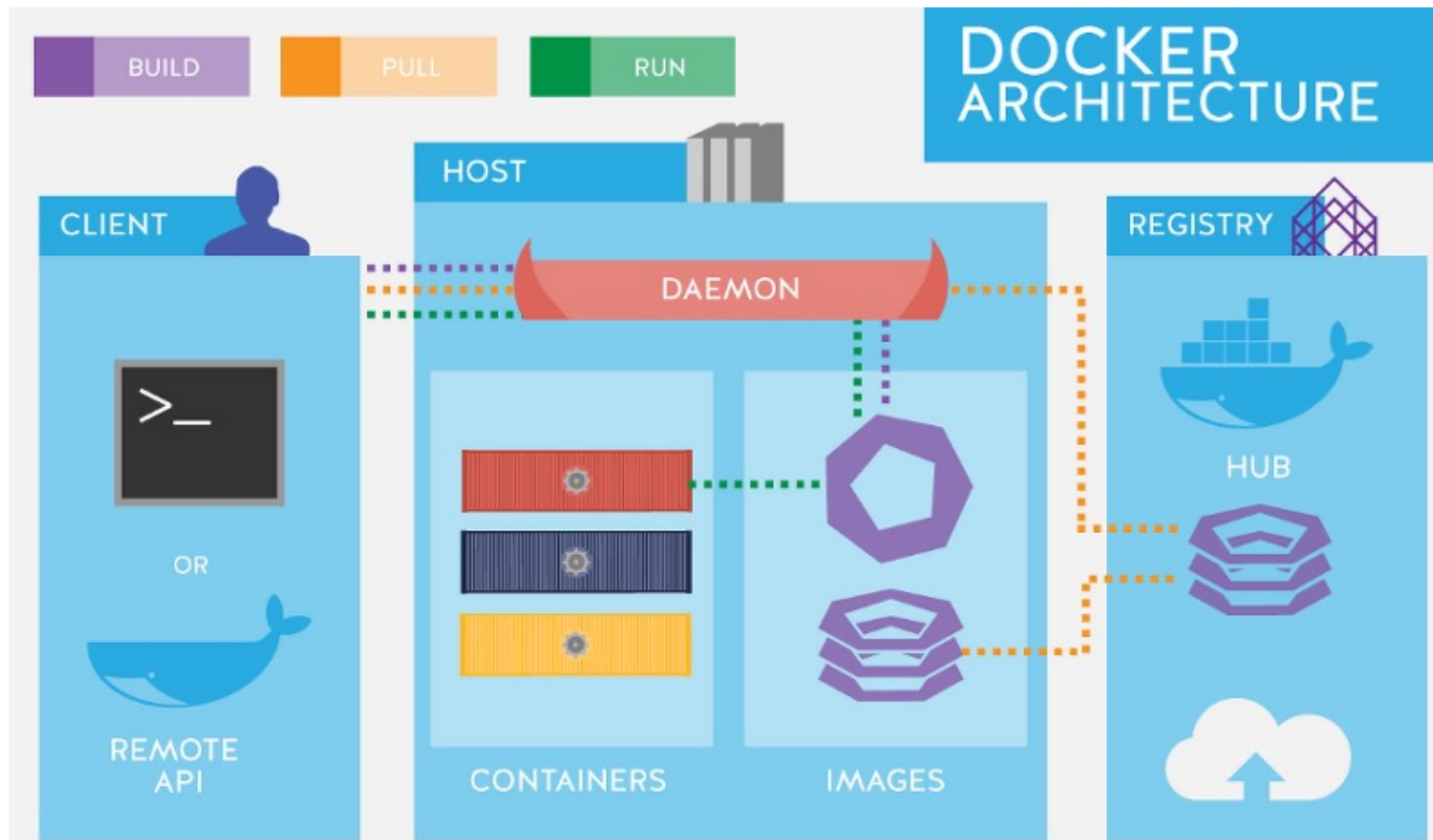
2. 협업에 대한 고민들

3. 실제로 서버를 운영하면서 겪는 일들

4. 세미나 마무리

# Container & Docker

가볍고, 빠르고, 효율적으로 서버 관리를 할 수 있게 된 중요한 기술



# Dockerfile, Docker-Compose

.yaml 파일을 통해 선언적으로 (declaratively) 배포 형태를 지정해줄 수 있음

⇒ 서버 환경에 대한 정보를 템플릿 형태로 공유할 수 있게 됨

⇒ Django + Nginx + Gunicorn 조합의 이미지를 긁어와서, 구성할 서버 환경에 맞게 값만 살짝 바꿔 주기만 해도 배포가 가능해짐

⇒ (예시 링크)

Docker는 익숙해지면 아주 좋음!

~~세미나장도 아직 안익숙합니다~~

# GitHub Actions & CI / CD

앞선 가상화 기술들 (특히 docker)

-> 다양한 task, job들을 빠르게 띄워볼 수 있게 됨

-> Github Actions 출시

: **.yaml** 에 작업 내용 서술

→ 컨테이너에서 실행 가능

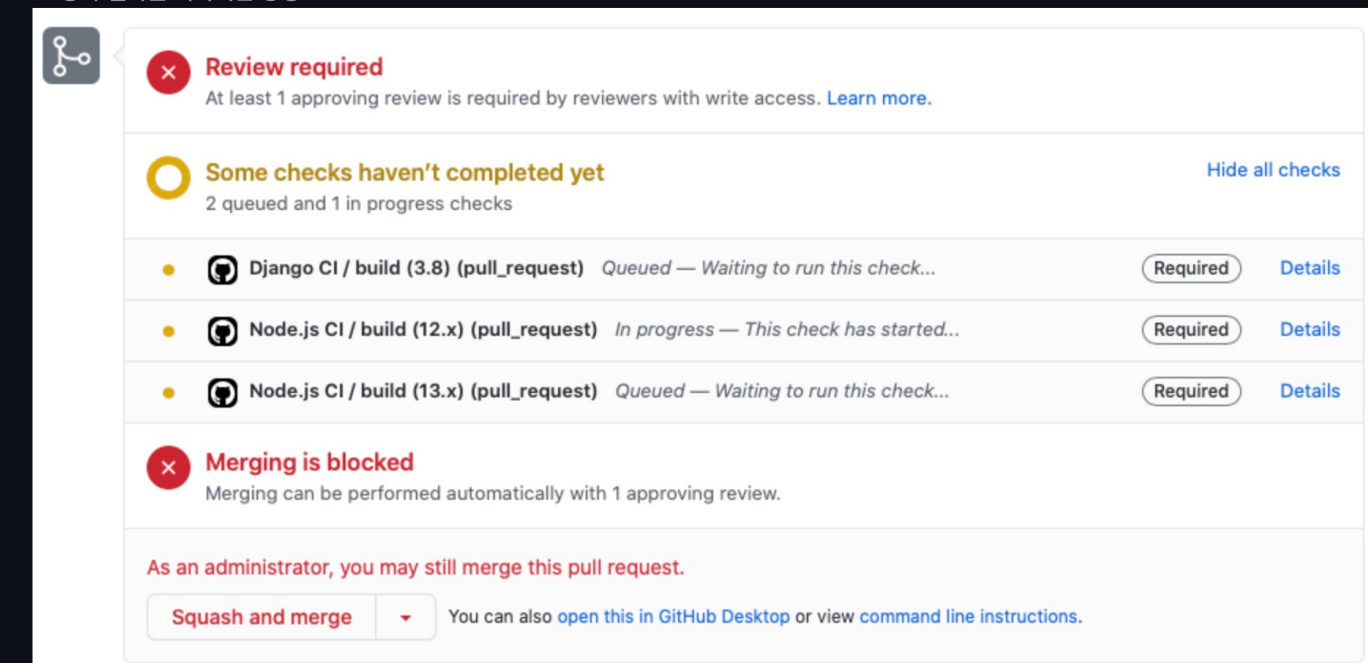
*Continuous Integration,*

*(+ Continuous Deployment)*

## 7 : GitHub Actions로 Django test에 대한 CI 진행하기

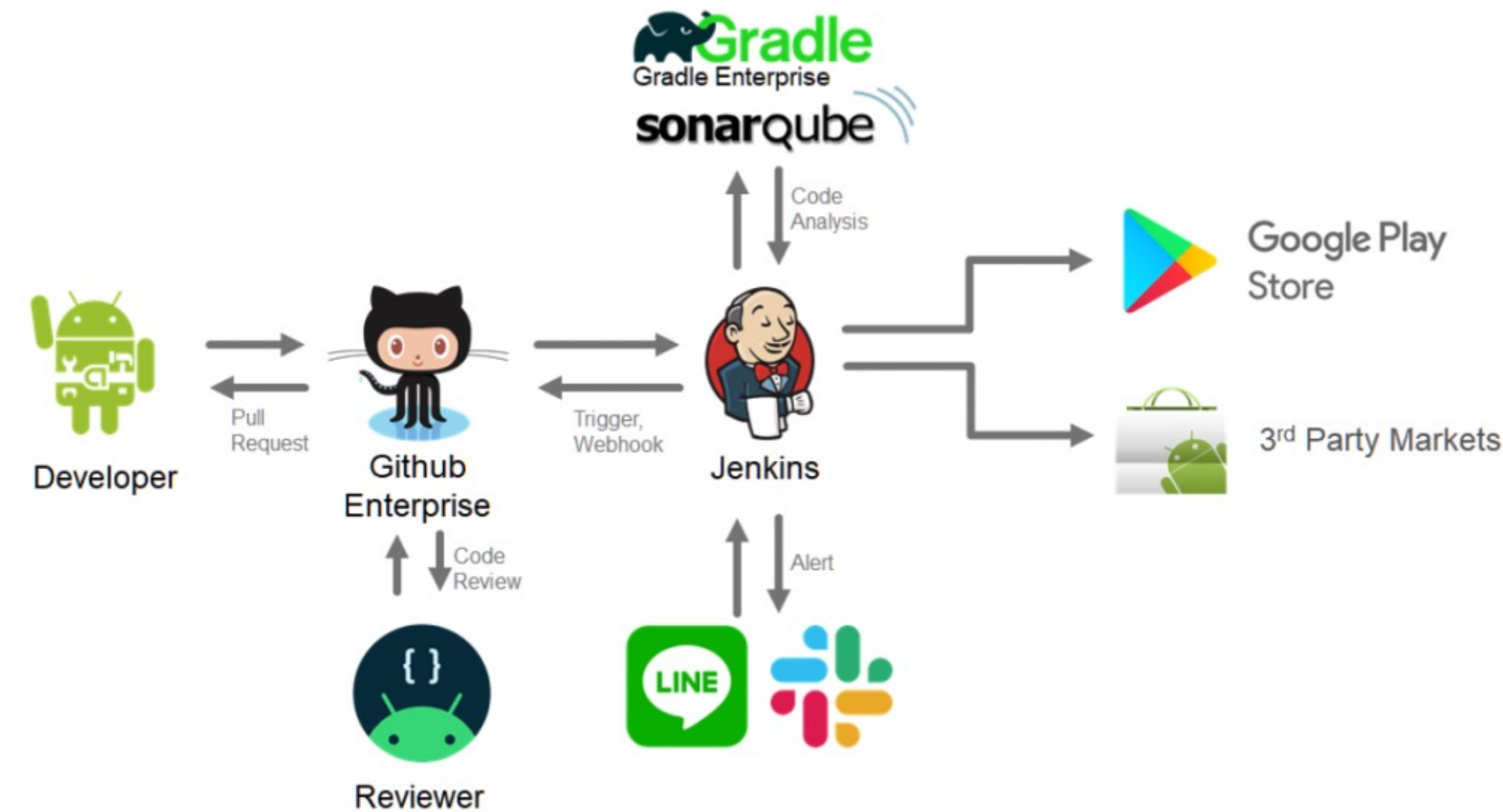
- CI(Continuous Integration)는 지속적인 통합으로, 개발자를 위한 자동화 프로세스라고 할 수 있습니다. CI를 성공적으로 구현할 경우 애플리케이션에 대한 변경 사항이 자동으로 테스트되어 repository에 통합되므로 여러 개발자가 동시에 작업을 할 경우 서로 충돌할 수 있는 문제를 좀 더 확실하게 방지할 수 있습니다. 또한, 애초에 개발자 스스로가 branch에 commit, push할 때 test를 철저히 하여 문제의 소지가 없게 해야하지만, 게을러서 또는 실수로 충분한 검토 없이 branch에 push를 하여 Pull Request를 만들고, 이것이 실제 서비스 배포를 진행하는 branch에 merge되어 버릴 수 있습니다. GitHub에 CI 도구를 적용시키면 PR이 open되었을 때, 각 branch에 push가 발생할 때마다 자동으로 설정한 스크립트 등에 따라 test, build 등이 진행되어 이러한 문제를 어느 정도 방지 할 수 있습니다.

- CI 동작 결과를 기다리는 상황





# CI / CD & DevOps



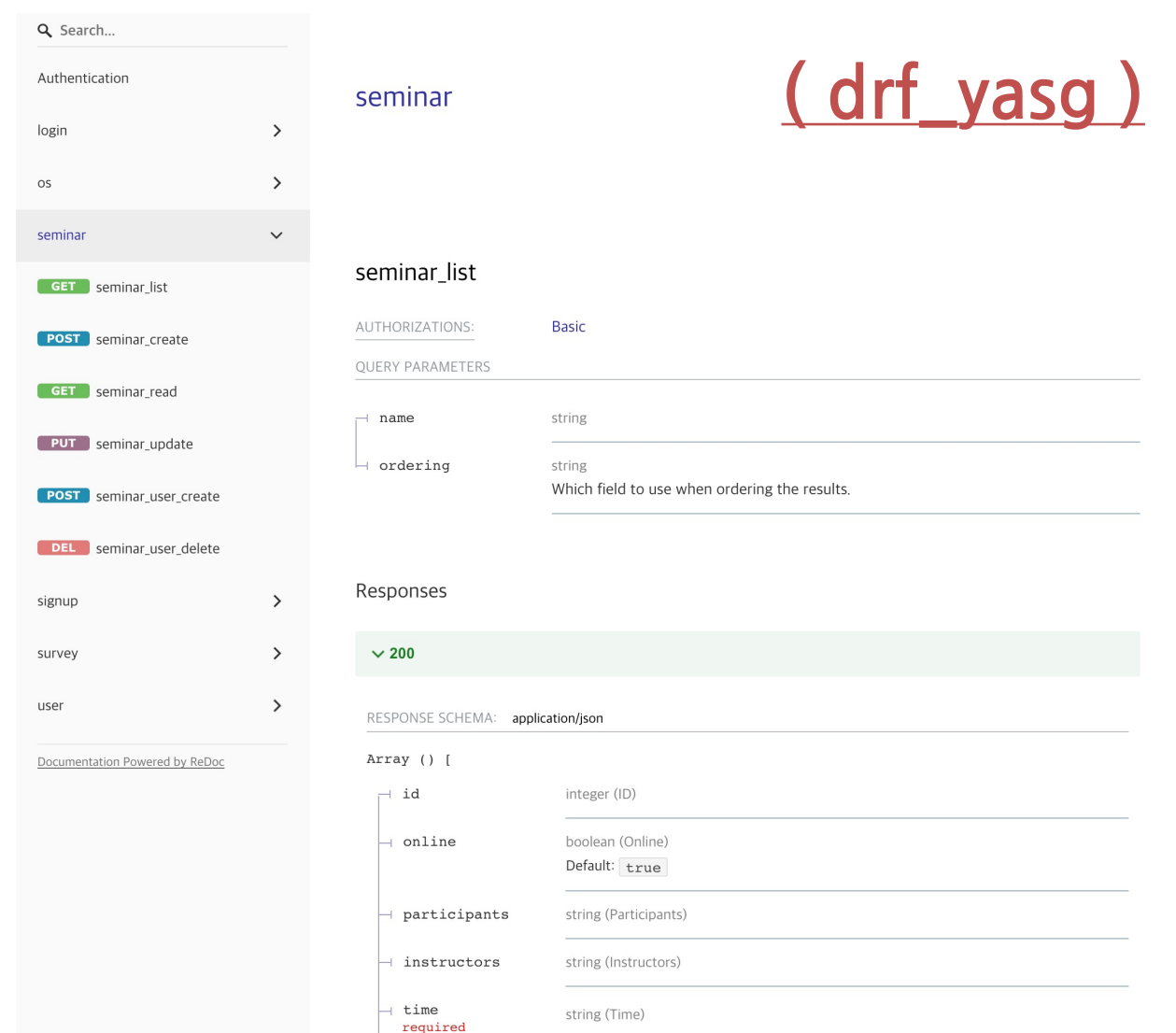
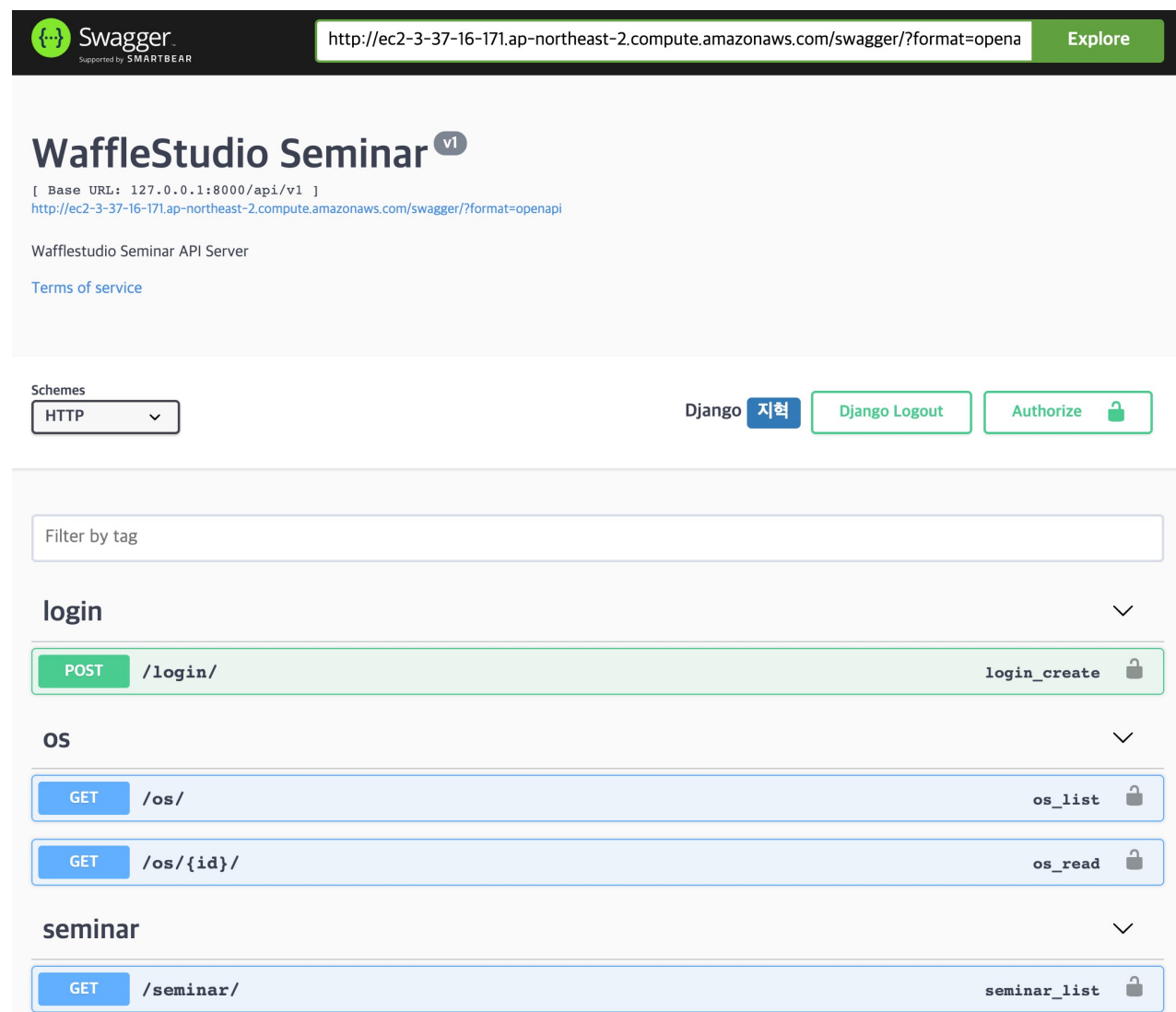
kubernetes

# 협업

## API

-> 클라이언트와 충분히 소통 후에, 처음부터 잘 설계해두자!

안드로이드 과제 4 에서 여러분과 함께 만든 Seminar 서버를 사용합니다.





# 협업

## API 전달 과정에서 고민해볼 점들

- Nullable ?
  - 서버에서 잘 정해주지 않으면, 클라이언트가 고생 ( null check pollution )
  - 귀찮아도 잘 정리해주자
- Exception Handling
  - 에러 메시지는 어떻게 관리할 것이고, 어떻게 띄워줄 것인가?
  - “알아서 대응해 놓는” 것이 좋은 서버 개발이라고 생각합니다 ☺

```
let title: string;

if(product != null) {
    if(product.id != null) {
        if(product.title != null) {
            title = product.title;
        } else {
            title = "N/A";
        }
    } else {
        title = "N/A"
    }
} else {
    title = "N/A"
}
```

# 협업

## DRF에서 Exception handling 해보기

- Custom Error
  - 쉽다! `rest_framework.exceptions` 보고 그대로 따라하기
- Custom Error Handlers

```
REST_FRAMEWORK = {  
  
    'EXCEPTION_HANDLER': 'common.exception_handler.exception_handler',  
  
    # 모든 API > 기본적으로 인증이 필요하게 됨
```

- DRF 기본 예러가 프로덕션에 적합할까?
- 스마트폰에 저거 다 담을 수 있을까?

예러 처리 그냥 클라이언트 시킬까?

```
HTTP 400 Bad Request  
Allow: GET, PUT, HEAD, OPTIONS  
Content-Type: application/json  
Vary: Accept  
  
{  
  "time": [  
    "Time has wrong format. Use one of these formats instead: hh:mm."  
  ],  
  "name": [  
    "This field may not be blank."  
  ],  
  "capacity": [  
    "A valid integer is required."  
  ],  
  "count": [  
    "A valid integer is required."  
  ]  
}
```

# 실제 서비스로 나아가기

서버 개발을 하다 보면 마주치는 다양한 상황들

- 백오피스 개발
- 개발 환경 나눠서 사용하기
- 동시성 이슈 처리하기

(등등..)

앞으로의 개발 과정에서 맞닥뜨릴 수 있는 상황들을 미리 훑어보자

# 실제 서비스로 나아가기

## 백오피스

- 서버는 API 개발 말고도 많은 일을 해야 함 + 서비스 운영에는 개발팀만 있는 것이 아님

⇒ 다른 팀에서 데이터를 수정해야 하거나 (ex. 거래 날짜 변경)

⇒ 기존 데이터를 조회하고 싶다면?

⇒ 1. SQL 배워서 DB에서 직접 꺼내 보기


⇒ 2. Django 배워서 ORM 꺼내 보기

⇒ 3. 그때그때 서버 팀에 문의하기

..?

# 실제 서비스로 나아가기

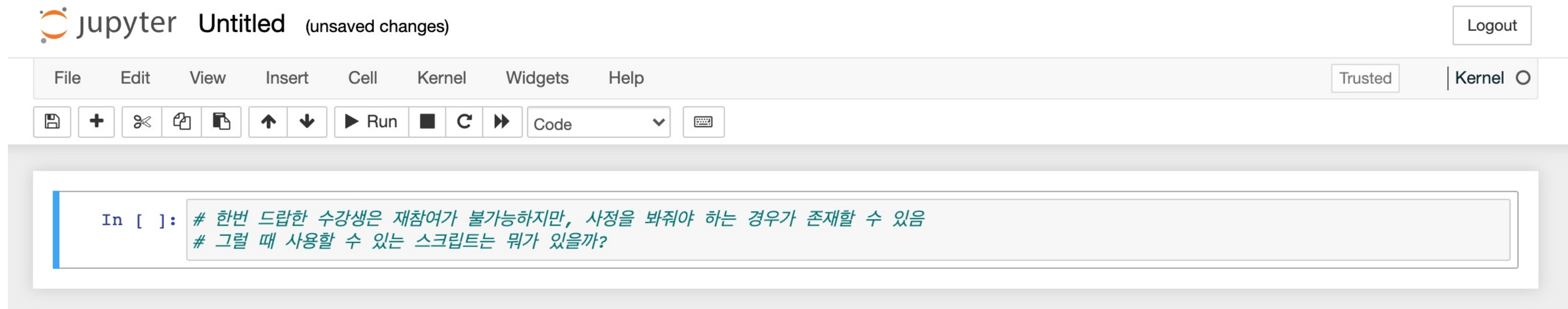
## 백오피스

- 개발을 몰라도, 도메인 데이터를 편하게 찾아보고 싶다
- Django  & Python 조합은 강력한 백 오피스 시스템을 갖출 수 있음
  - 1. 내장 Admin 기능 (batteries\_packed!)
    - Django Admin
    - DRF의 AdminRenderer
  - 2. Django Shell
    - 유동적인 스크립트 작성 기능

# 실제 서비스로 나아가기

## 스크립트 환경 만들기

### Django + Jupyter Notebook





# 실제 서비스로 나아가기

## 다양한 서버 환경에 대응하기

- test, dev, prod .. 의 환경은 다를 수밖에 없다.
  - DJANGO\_SETTINGS\_MODULE 통해 설정 파일 분기
- 인스턴스(ex. EC2)가 여러 개 돌아가고 있을 수 있다.
  - 동시성 이슈 유의하기
- DB가 여러 대 돌아가고 있을 수 있다.

# 실제 서비스로 나아가기

## 동시성 이슈 처리하기

### 1. select\_for\_update

- DB에 Transaction Lock 걸어줌
- 속도 포기, 일관성 유지 (ACID !)
- Deadlock 유의하며 사용하자

### 2. DB Constraint (unique\_together 옵션)

- DB에 Constraint 걸어줌
- 동시에 값에 접근할 경우 IntegrityError를 내는 식 (transacion rollback)
- 에러 핸들링을 잘 하자

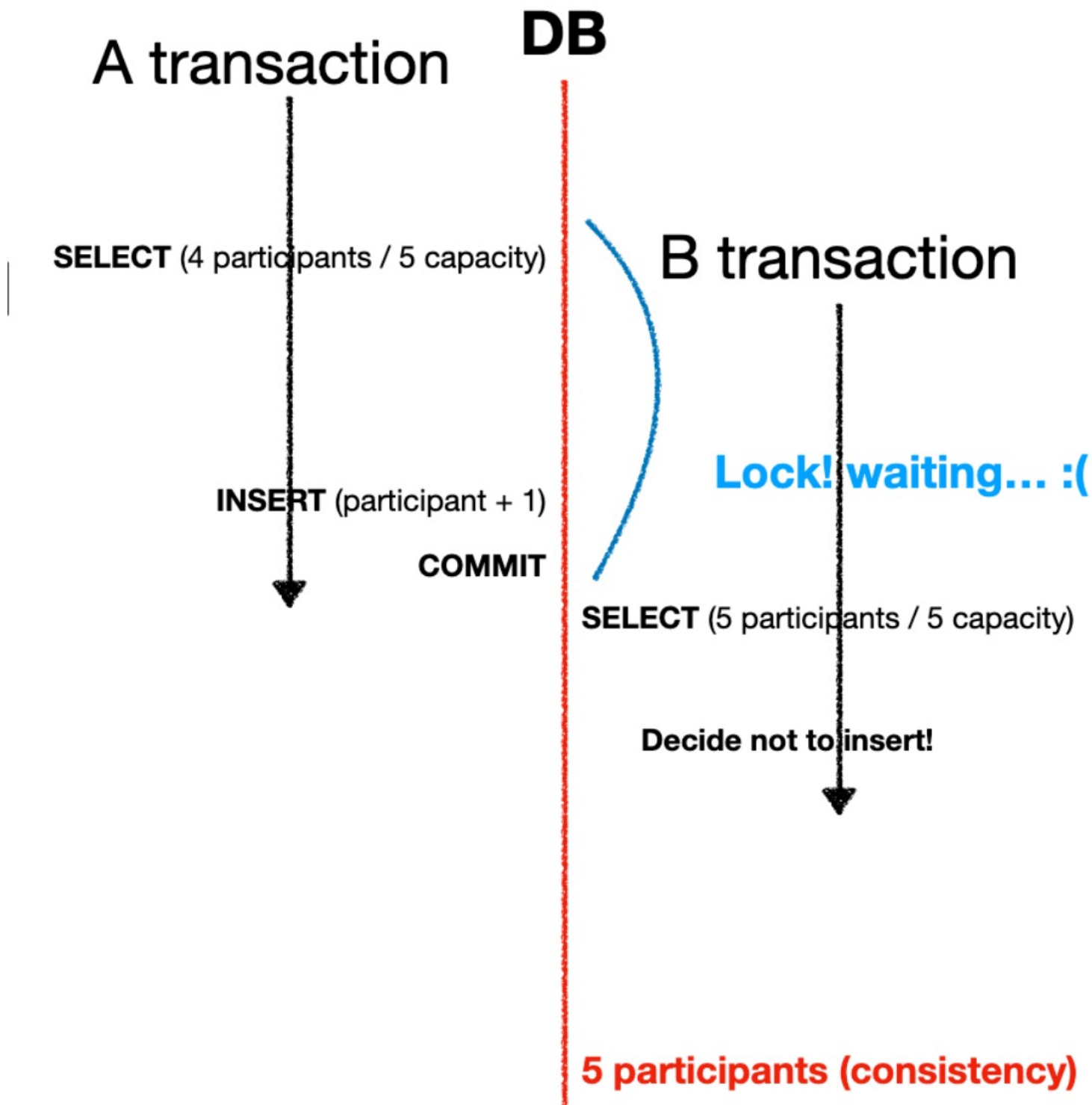
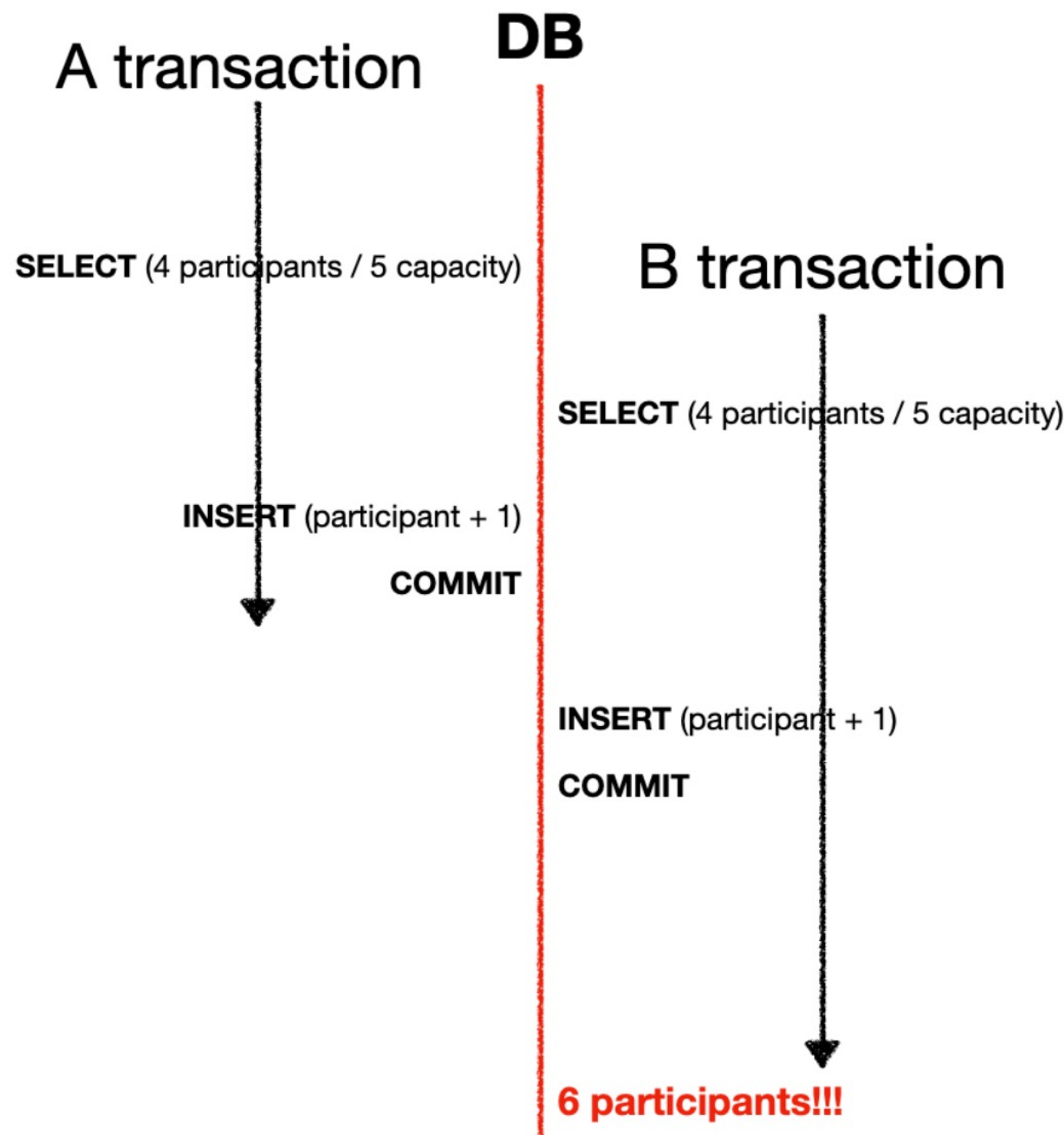
<https://victorydntmd.tistory.com/129>

#### ACID (revisited) DB transaction의 원칙

- Atomicity (원자성)
  - 다 되거나 다 안 되거나, 둘 중에 하나여야 함
- Consistency (일관성)
  - 미리 정해둔 규칙에 맞게 데이터를 저장, 변경해야 함
- Isolation (고립성)
  - 복수 개의 transaction이 실행될 때 서로 연산이 끼어들면 안 되며, 중간 과정이 이용되어도 안 됨
- Durability (영구성)
  - 일단 commit했으면 어떠한 문제가 발생해도 영구적으로 반영된 상태여야 함

# 실제 서비스로 나아가기

select\_for\_update 실행 시 오른쪽 그림처럼, 동시에 접근하려는 쿼리들은 Lock 걸림

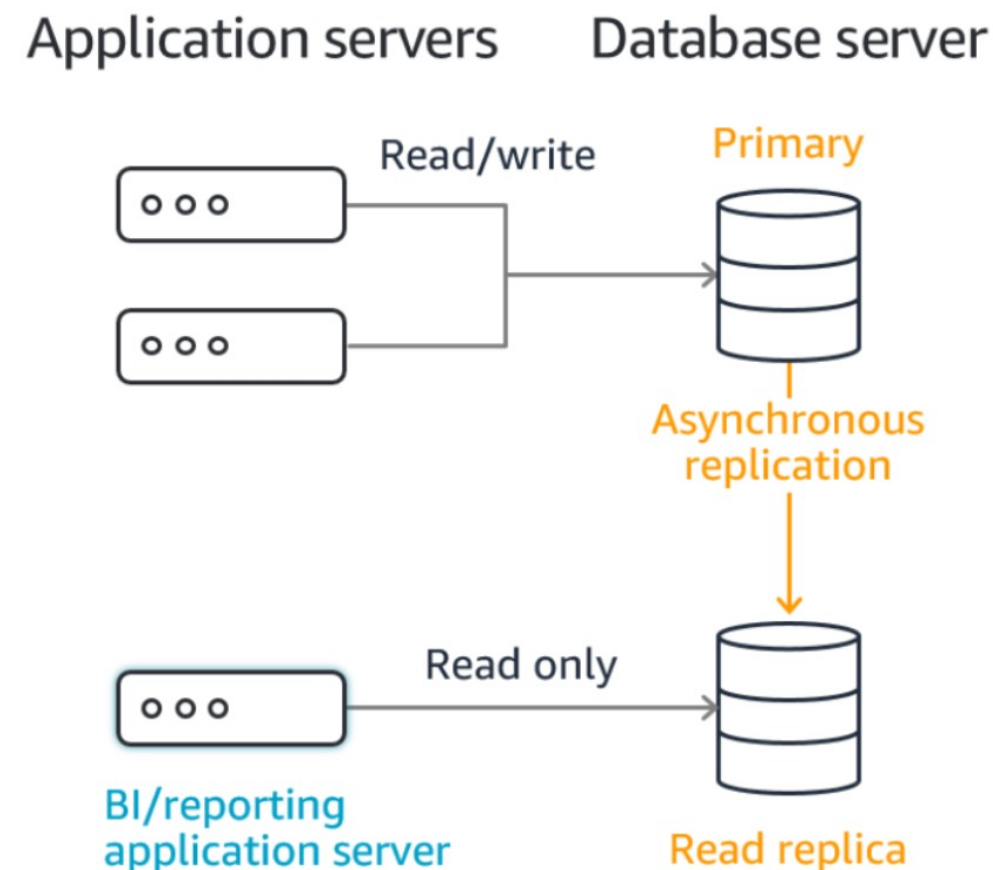


# 실제 서비스로 나아가기

DB가 여러 개!?

- 아예 다른 DB를 여러 개 사용
  - MSA 같은 아키텍처의 경우, 하나의 서비스지만 다른 서비스들처럼 움직임
    - 소셜 커머스 앱 : 채팅 서비스 ⇄ 결제 서비스

- Master - Slave DB
  - 쿼리의 95%는 READ
  - 요청 분산, 백업 기능 등등 ..
  - Replica lag 발생



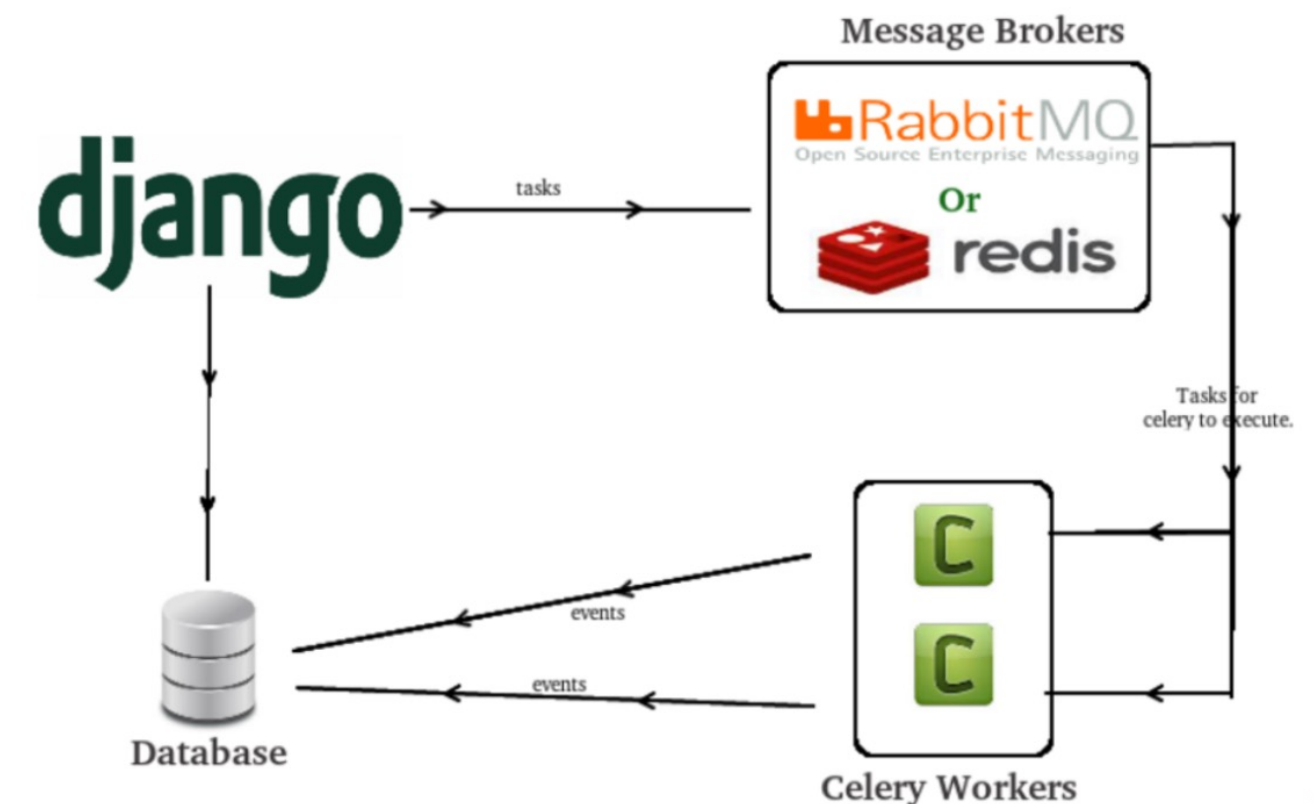
# Celery : Distributed Task Queue

## Synchronous 서버의 단점

- 요청이 들어오면, 끝나서 반환할 때까지 붙들고 있음
  - 한참 걸리는 일 (ex. 3분짜리 파일 다운로드)을 100명한테 요청 받았다고 치면,,
  - 다음 사람은,, 그 다음 사람은 ,, 100번째 사람은??

비현실적으로 느려진다.

- CELERY TO THE RESCUE
- 요청이 들어오면, Task Queue에 넣어둬م
- Celery Worker들이 Queue를 바라보고 있음
  - Queue에 담겨있는 Task 하나씩 소비





# Celery : Distributed Task Queue

Django + Redis + Celery 조합이 널리 쓰임

## 1. Django

: 요청을 받아서, Message Broker (Queue)로 토스

## 2. Celery Worker (숫자 조절 가능)

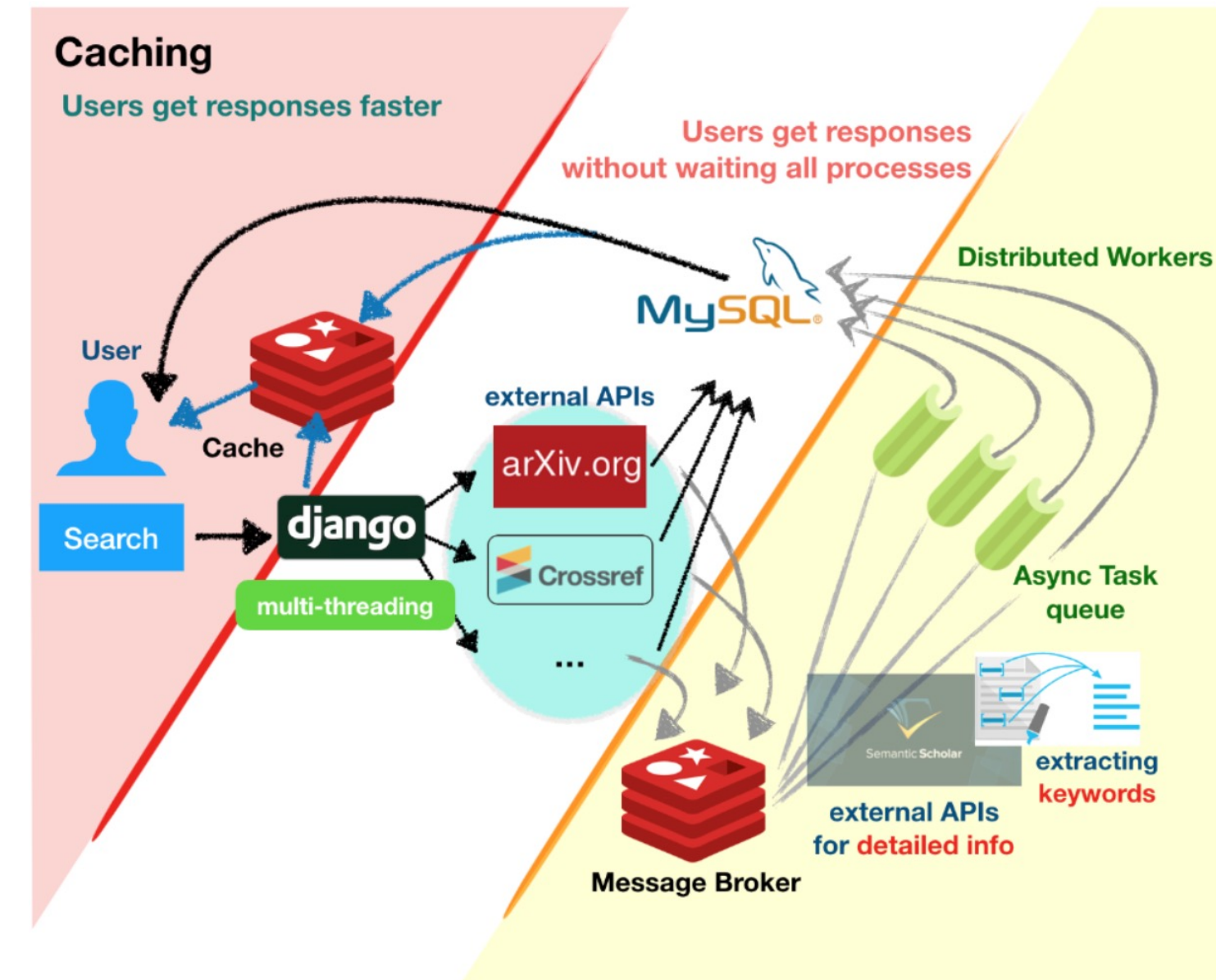
: Broker에 들어온 요청 분산 실행

=> 이후 실행이 끝나면 Result Backend에 값 저장

그럼 장고는 요청 결과를 어떻게 알까?

Task 실행은 AsyncResult 객체 반환

: 성공 / 실패 여부, 성공했을 경우 반환된 값 확인 가능





# Celery : Distributed Task Queue

또 다른 기능, Cron Job

- 매일 특정 시간에 앱 접속 데이터를 다운로드 받아서 분석하고 싶다... 등등
- Scheduling은 아주 흔한 개념 (Cron = Unix scheduler)
- <https://crontab.guru/> 에서 cron schedule 형식 확인 가능
- 애도 쓰기 쉽다. 셀러리 깔고, 세팅에 적어주면 끝 =>
- 여러분이 장고에서 쓸 거의 대부분의 패키지들
  - 설정만 바뀌어도 적용 가능
  - 뭘 쓰든, 패키지 공식 문서를 열심히 숙지만 하면 됨 (readthedocs.io 같은...)
  - 그래서, 장고는 대단히 쉽고 매력적인 프레임워크

```
CELERY_BEAT_SCHEDULE = {  
    'celery_cron': {  
        'task': 'celery_cron',  
        'schedule': crontab(minute='*'),  
    },  
}
```

# 마지막

장고 세미나에서 지금까지 배운 것들

- 이론적인 내용은 절대 Django-Specific한 것들이 아니고, 서버 개발을 할 때 무조건 만날 것들임
- ORM 정도를 제외하면, 사용 방법도 크게 다르지 않음

다른 서버 프레임워크들

- Spring, Node.js, Flask ...
- 여러 가지 프레임워크를 접해보는 것도 서버 개발 이해도를 높일 수 있는 좋은 방법

# 마지막

화이팅 !!

과제 마감 시간만 잘 지키면, 대부분 통과하실 것으로 예상됩니다.

꼭 완수하시고, 와플스튜디오에서 다같이 재미있게 개발 경험 만들어나가면 좋을 것 같습니다.

**Toy project, 그리고 이후 Wafflestudio 활동에서  
재밌는 것들을 함께 많이 만들어봅시다!** 🦄🌍🚀

# Q&A

WA#LE  
STUDIO

WA#LE  
STUDIO