

R로 API 서버를 만드는 4가지 방법

(은 삽질기)

<https://mrchypark.github.io/apiR>

[pdf버전] [문의하기] [의견 및 오류 신고]

박찬엽

2017년 10월 28일

발표자 소개

박찬엽



- 서울도시가스 선행연구팀 연구원
 - 챗봇 엔진 개발 및 서버 구축
- 패스트 캠퍼스 데이터 분석 R 강의
 - 데이터 분석을 위한 중급 R 프로그래밍
- R 네이버 뉴스 크롤러 N2H4 관리자
 - ForkonLP 프로젝트
- **KAKAO@알코홀릭** R 질문방
- **FACEBOOK@mrchypark**
- **GITHUB@mrchypark**

R로 API 서버를 만드는 4가지 방법 (은 삽질기)

자매품

R로 웹 데이터를 가져오는 4가지 방법
(은 크롤링)

의사 결정을 위한 데이터 분석 프로젝트의 단계

데이터 확보 - 전처리 - 모델링 - 시각화 - 문서화

머신러닝 프로젝트의 단계

데이터 확보 - 전처리 - 모델링 - **제품화**

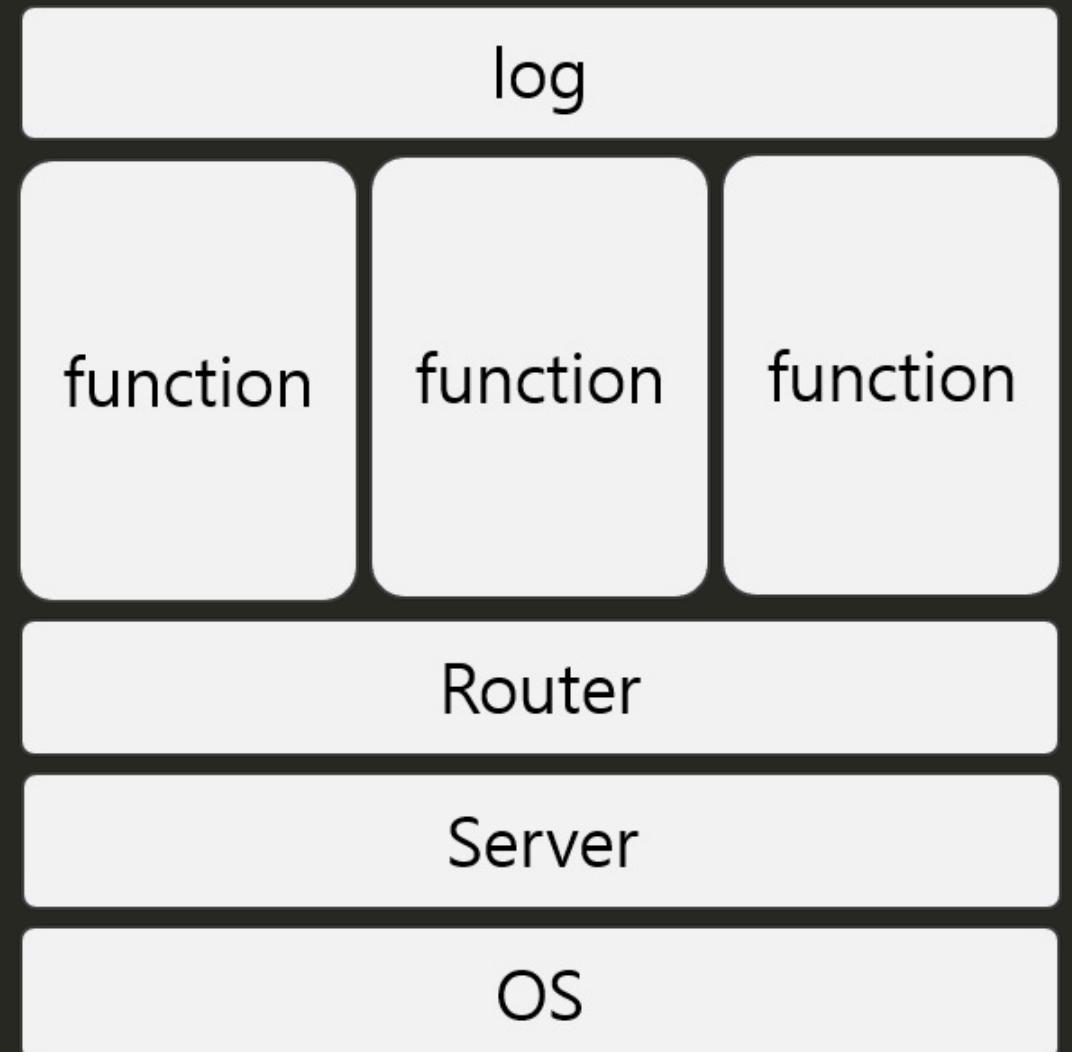
제품화

머신러닝 프로젝트의 결과물을 서비스에 활용하는 것 또한 많은 노력이 필요함

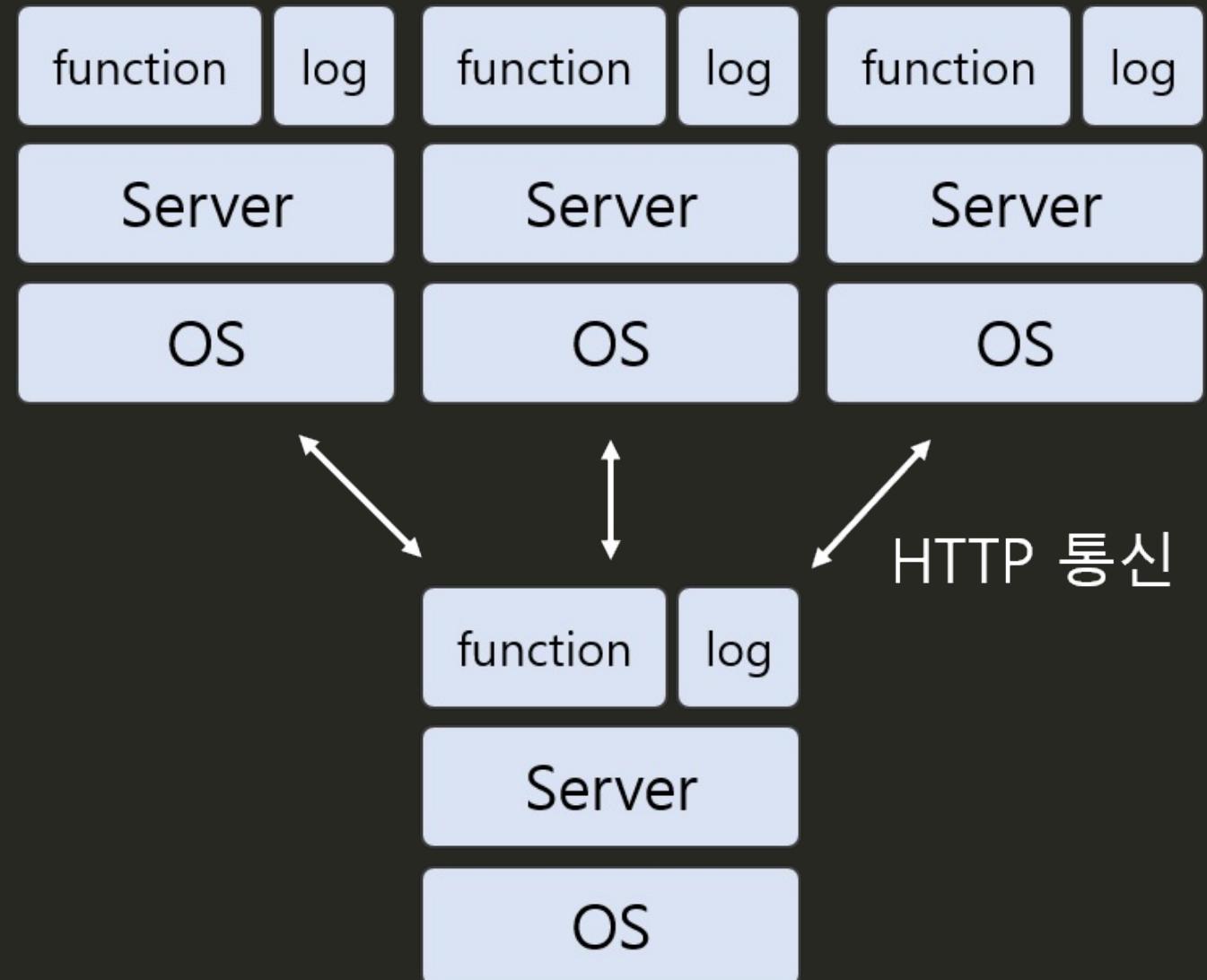


제품화하는데 노력이 덜 드는 방법은 없을까

Monolithic Architecture



Micro Service Architecture



외부 요청

외부 요청

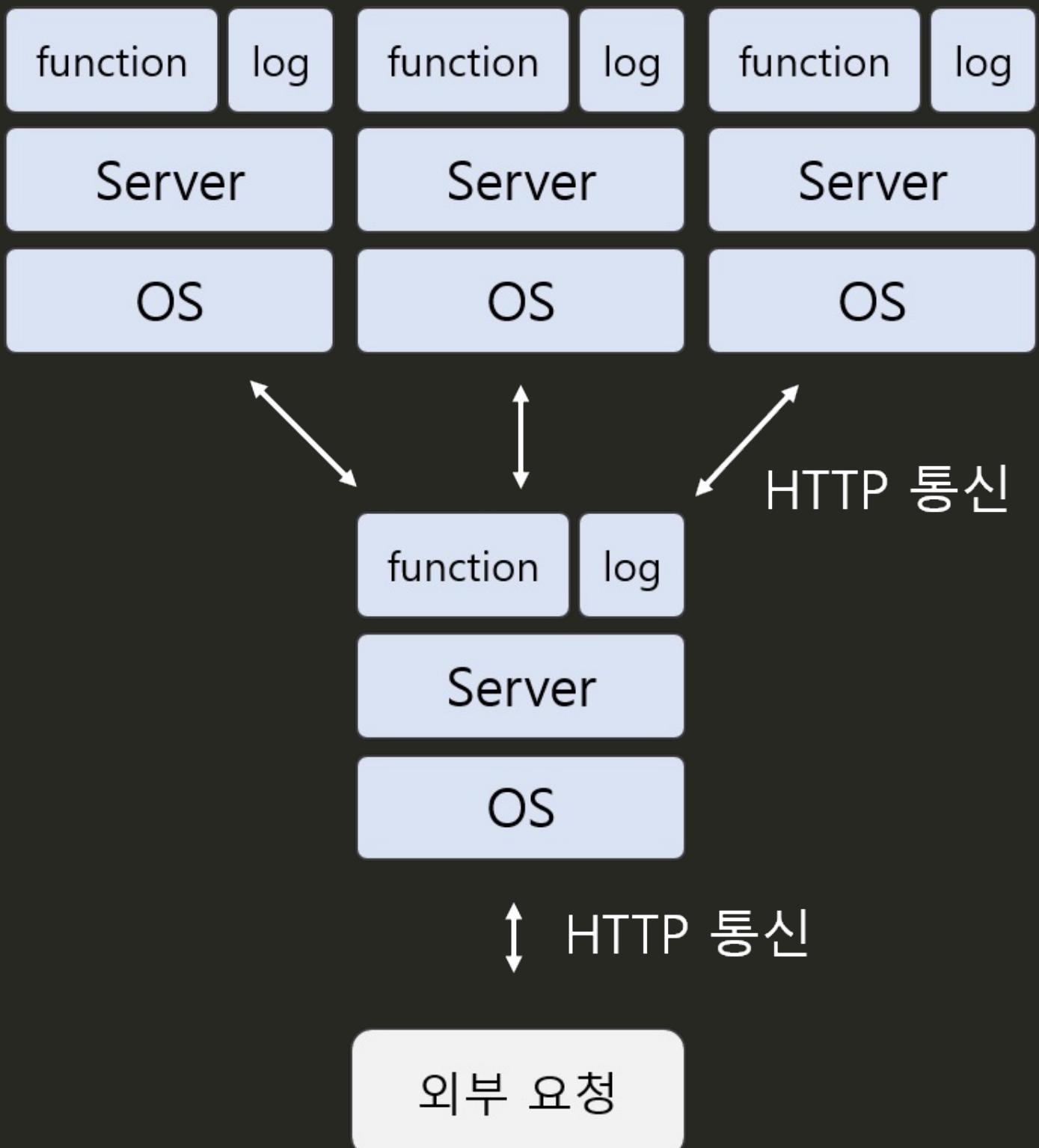
マイ크로 서비스형 설계

장점

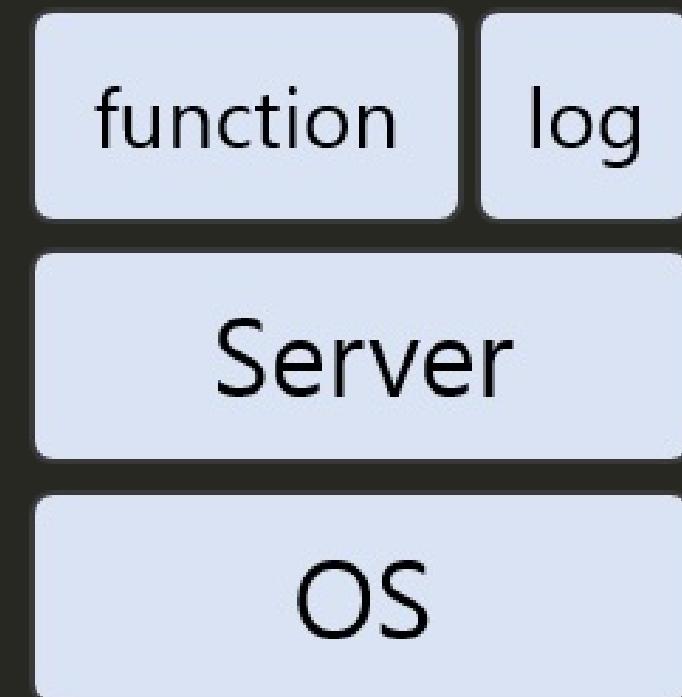
1. 기능 단위로 완결되어 개발 단위가 작음
2. 개별 서비스간의 의존성이 적어 유연함
3. 단일 모듈 장애시 전체 서비스의 영향이 적음

단점

1. 개별 서비스의 테스트 뿐만 아니라 전체 테스트가 필요
2. 서비스간 인터페이스 정의와 관리 이슈가 발생
3. 전체 서비스의 로그 관리를 위한 솔루션 필요



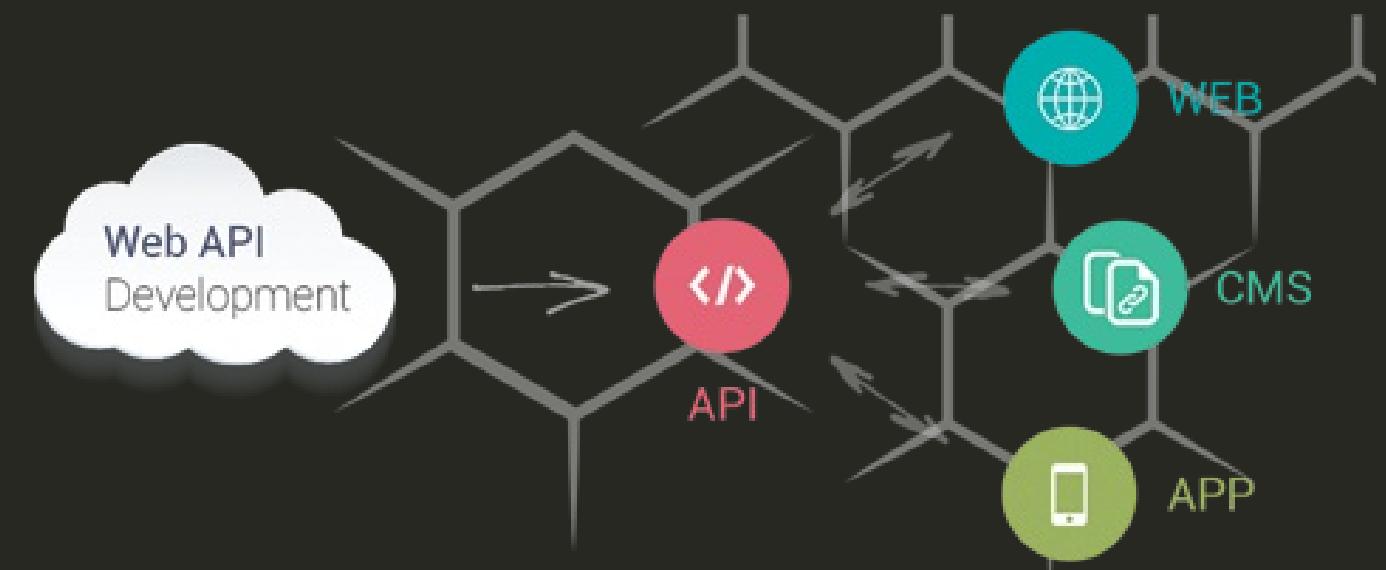
그럼 이거만 만들 줄 알면...!



그럼 작은 API 서버를 R로 만들어 보자

API 서버

http 표준으로 요청을 받아서 처리하여 응답하는 서버



http/1.1 명세

HTTP 메소드 ◆	RFC ◆	요청에 Body가 있음 ◆	응답에 Body가 있음 ◆	안전 ◆	멱등(Idempotent) ◆	캐시 가능 ◆
GET	RFC 7231 ↗	아니오	예	예	예	예
HEAD	RFC 7231 ↗	아니오	아니오	예	예	예
POST	RFC 7231 ↗	예	예	아니오	아니오	예
PUT	RFC 7231 ↗	예	예	아니오	예	아니오
DELETE	RFC 7231 ↗	아니오	예	아니오	예	아니오
CONNECT	RFC 7231 ↗	예	예	아니오	아니오	아니오
OPTIONS	RFC 7231 ↗	선택 사항	예	예	예	아니오
TRACE	RFC 7231 ↗	아니오	예	예	예	아니오
PATCH	RFC 5789 ↗	예	예	아니오	아니오	예

아 됐고!



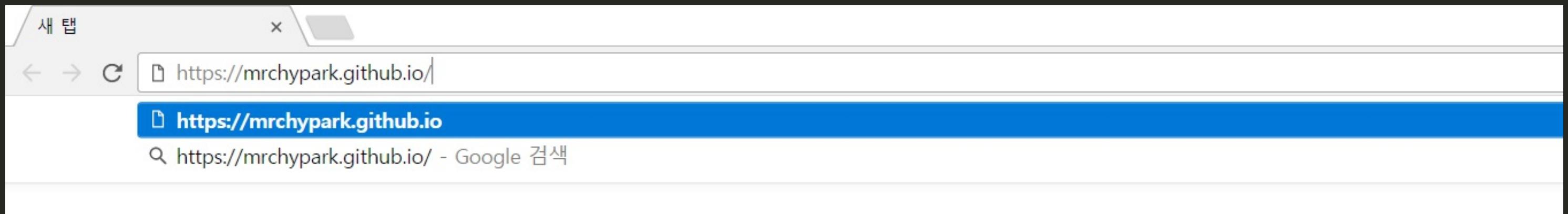
두 가지만 알면 됩니다.

GET vs POST

심지어 POST만 알아도 됨.

GET 이란

브라우저에서 주소를 입력하고 엔터를 치는 것!



그럼 서버는

요청하면 주기로 정해진 데이터(ex>이미지 파일)를 제공

The screenshot shows a browser window displaying a news article from JTBC. On the left, there's a sidebar with various news categories. The main content area shows a headline about a camera being left on a movie set. Below the headline is a large blue banner with the JTBC logo and the text 'JTBC 뉴스룸'. At the bottom of the screen, there's a video player showing a thumbnail and a progress bar indicating the video has played 1 second out of 4 seconds.

The right side of the image shows the Network tab of the browser's developer tools. The 'Font' tab is selected. Above the table, a timeline shows several requests starting around 1000ms and continuing at regular intervals. The table below lists the requests:

Name	Status	Type	Initiator	Size	Time	Waterfall
read.nhn?mode=LSD&... 200	docu...	Other	127 KB	48 ms		
common.css 200	styles...	read.nhn?m...	(from...	16 ms		
news.jindo.js 200	script	read.nhn?m...	(from...	0 ms		
news.service.js 200	script	read.nhn?m...	(from...	0 ms		
dic.tooltip.js 200	script	read.nhn?m...	(from...	0 ms		
nil.news.js 200	script	read.nhn?m...	(from...	0 ms		
snb_h_entertain.png 200	png	read.nhn?m...	(from...	0 ms		
snb_h_sports.png 200	png	read.nhn?m...	(from...	0 ms		
snb_h_newsstand.png 200	png	read.nhn?m...	(from...	0 ms		
snb_h_weather.png 200	png	read.nhn?m...	(from...	0 ms		

At the bottom of the developer tools interface, the following statistics are displayed: 124 requests | 3.9 MB transferred | Finish: 4.01 s | DOMContentLoaded: 818 ms | Load: 987 ms

우리 GET 서버가 할 일

서버에게 GET 요청이 오면 필요한 데이터를 전달해주기.

"안녕하세요!!"

우리 GET 서버가 할 일

서버에게 GET 요청이 오면 필요한 데이터를 전달해주기.

"안녕하세요!!"

jug를 소개합니다!

```
if (!requireNamespace("jug")) {  
  install.packages("jug")}  
library(jug)  
  
jug() %>%  
  get("/", function(req, res, err){  
    res$json(enc2utf8("안녕하세요!!"))  
    return(res)  
}) %>%  
  simple_error_handler_json() %>%  
  serve_it()
```

Serving the jug at <http://127.0.0.1:8080>

R로 GET 요청하기

R에서 http 표준의 요청을 처리해 주는 유용한 패키지는 `httr`입니다.

```
if (!requireNamespace("httr")) {  
  install.packages("httr")}  
library(httr)  
  
target<-"http://127.0.0.1:8080/"  
content(GET(url=target))  
  
## [1] "안녕하세요!!"
```

GET 서버를 만들었다!

R로 api 서버를 만드는 첫 번째 방법!

jug 패키지를 사용

이제 POST 서버를 만들어 보자

더 알아야 할 것

요청(request) vs 응답(response) + 헤더(header)¹

1. 헤더(header): http://www.w3ii.com/ko/http/http_header_fields.html

GET과 POST의 차이점

요청 헤더에 body라는 이름으로 값을 보내면 POST이고 없으면 GET

GET과 POST의 차이점

요청 헤더에 body라는 이름으로 값을 보내면 POST이고 없으면 GET

R 함수로 예를 들면

```
# 입력없이 결과를 주는 GET  
R.Version()
```

```
## $platform  
## [1] "x86_64-w64-mingw32"  
##  
## $arch  
## [1] "x86_64"  
##  
## $os  
## [1] "mingw32"  
##  
## $system  
## [1] "x86_64, mingw32"  
##  
## $status  
## [1] ""  
##  
## $major  
## [1] "3"  
##
```

```
# a, b 입력으로 결과를 주는 POST  
a<-1;b<-2  
sum(a,b)
```

```
## [1] 3
```

우리 POST 서버가 할 일

서버에게 POST 요청이 한글 문장을 "sent"라는 이름으로 같이 보내면 띄어쓰기 개수를 세서 줌.

```
# 입력  
"{sent='R 로 API 서버를 만드는 4가지 방법(은 삽질기)'}"  
  
# 예상 결과  
"7"
```

stringr with jug

stringr은 정규표현식을 활용한 글자 처리를 도와주는 패키지

```
if (!requireNamespace("jug")) {  
  install.packages("jug")}  
library(jug)  
  
if (!requireNamespace("stringr")) {  
  install.packages("stringr")}  
library(stringr)  
  
count_ws<-function(sent){  
  stringr::str_count(sent, "[[:space:]]")  
}  
  
jug() %>%  
  post("/", decorate(count_ws)) %>%  
  simple_error_handler_json() %>%  
  serve_it()
```

R로 POST 요청하기

httr로 POST 요청을 해보겠습니다.

```
library(httr)  
  
url<- "http://127.0.0.1:8080"  
body<- list(sent="R로 API 서버를 만드는 4가지 방법(은 삽질기)")  
content(POST(url, body=body), "text")
```

```
## No encoding supplied: defaulting to UTF-8.  
## [1] "7"
```

POST 서버를 만들었다!

R의 서버 개발 패키지들

`httpuv, Rserve, jug, plumber, fiery`

이렇게 많은 패키지가 있는데...

오픈소스 사용시 중요한 점.

사용자층이 넓을 것, 설명서가 충실할 것, 버전이 1.X.X 이상일 것 등등

reticulate 를 소개합니다!

python 패키지를 R에서 사용할 수 있게 해주는 패키지

Flask

python의 가볍고 유명한 웹 서버 개발 프레임워크

github 스타 수

plumber: 453

Rserve: 161

jug: 119

fiery: 114

httpuv: 98

Flask: 30,605

R로 api 서버를 만드는 두 번째 방법!

python의 **flask**를 이용하기

lumiamitie님께서 초기 삽질을 대신해 주셨습니다!

<http://lumiamitie.github.io/r/flask-on-r/>

COMMENTS

댓글 한 건 lumiamitie

4 mrchypark ▾

추천 공유

인기순 ▾



토론 참여하기



mrchypark • 5달 전

하아 미리 해주셔서 정말 감사합니다ㅠ_ㅠ 덕분에 빨리 해볼 수 있게 됬어요!

^ | v . 수정 . 답글 . 공유 ,

GET 서버 예시

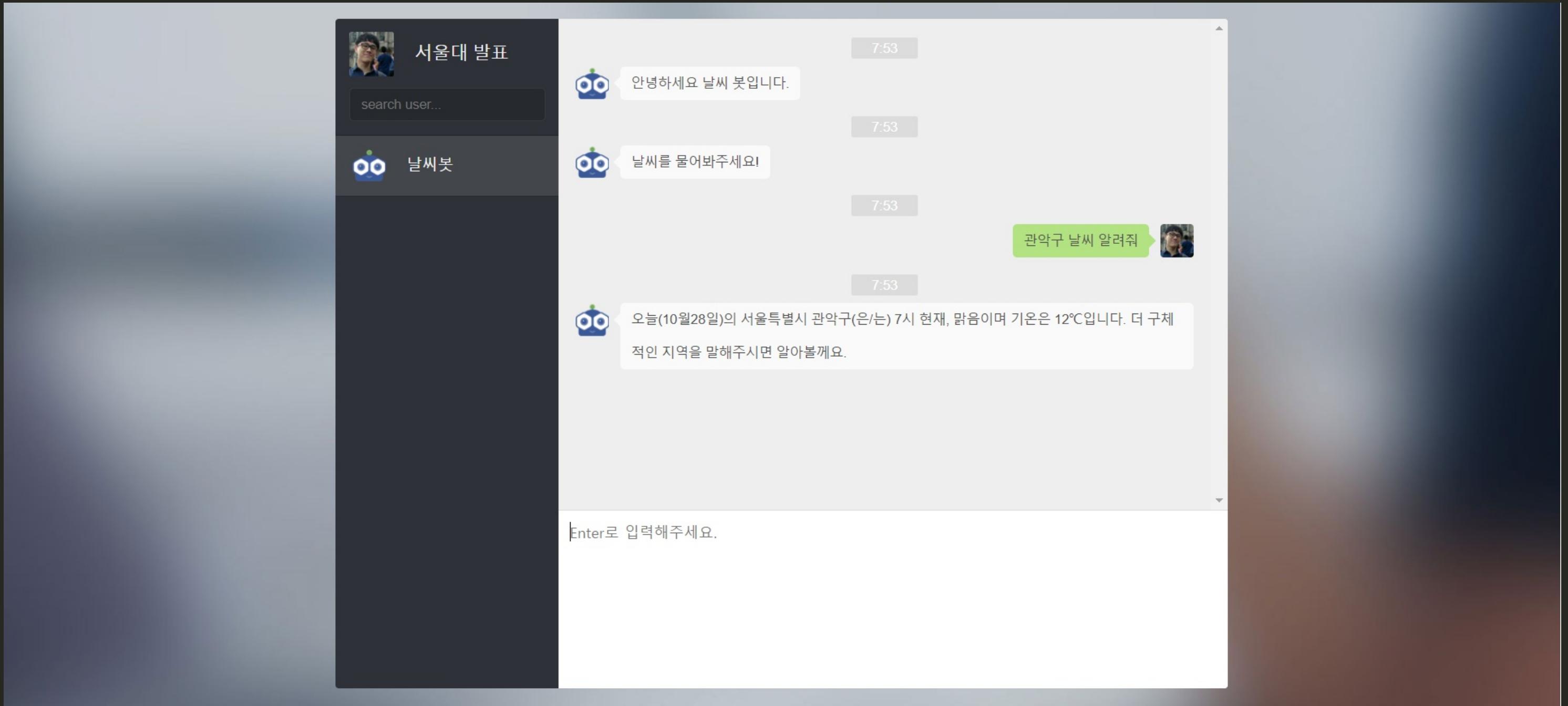
```
library('reticulate')

flask = import('flask')
app = flask$Flask('__main__')

app$route('/'){
  index = function() {return('Hello R user Conference!')}
}

app$run()
```

시연



flask on r

장점

- flask의 풍부한 자료를 활용



단점

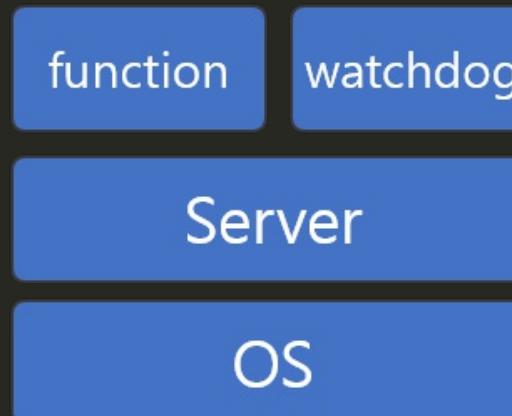
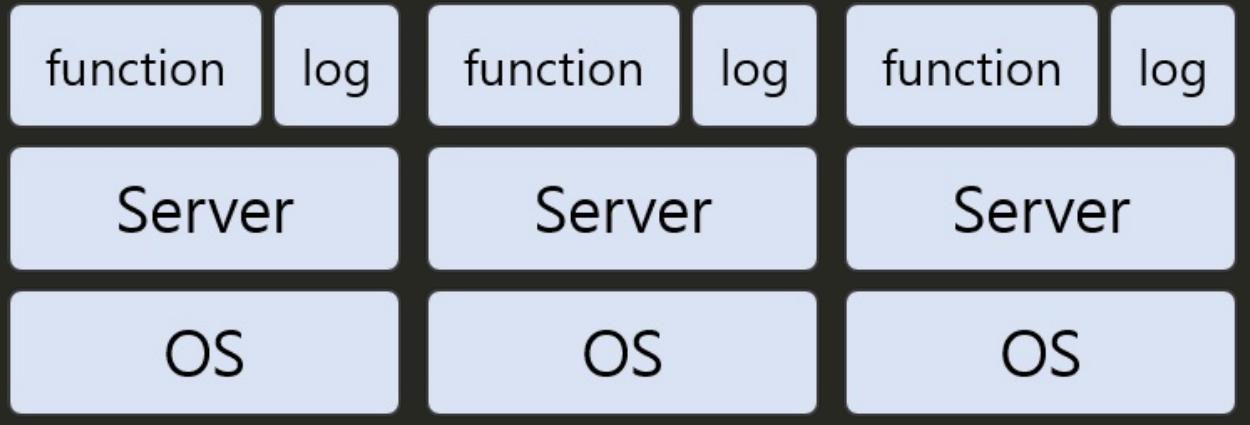
- python 도 알아야 함



더 좋은 방법은 없을까

Micro Service Architecture

Function as a Service



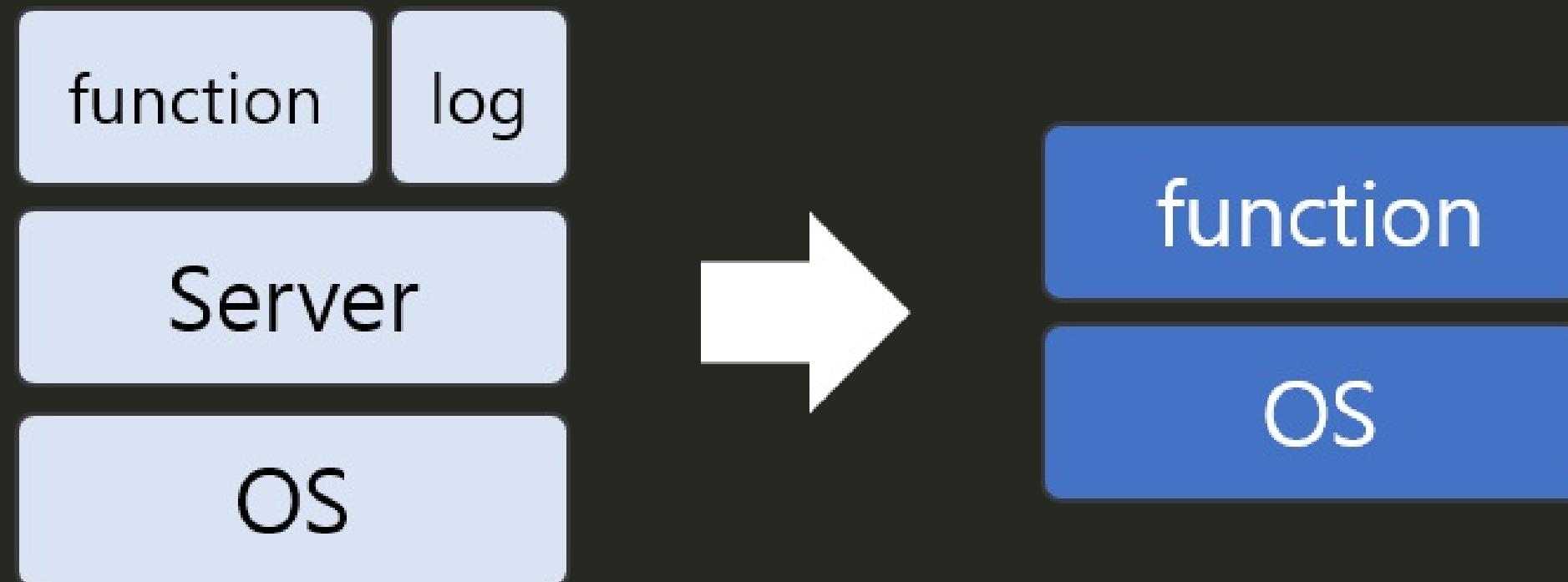
↔ HTTP 통신

↔ HTTP 통신

외부 요청

외부 요청

서버도 몰라도 됨



R로 api 서버를 만드는 세 번째 방법!



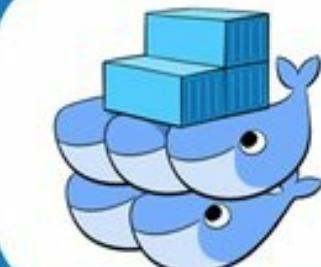
Functions as a Service

API Gateway

Function Watchdog



Prometheus



Swarm



Kubernetes



표준 입출력으로 함수 작성

```
## 표준 입력을 받아 line 객체로 저장
f <- file("stdin")
open(f)
line<-readLines(f, n=1, warn = FALSE)

## 데이터를 처리하여 result 객체로 저장
result<-paste0("Hi ", line)

## 표준 출력으로 결과 전달
write(result, stderr())
```

친절한 예시

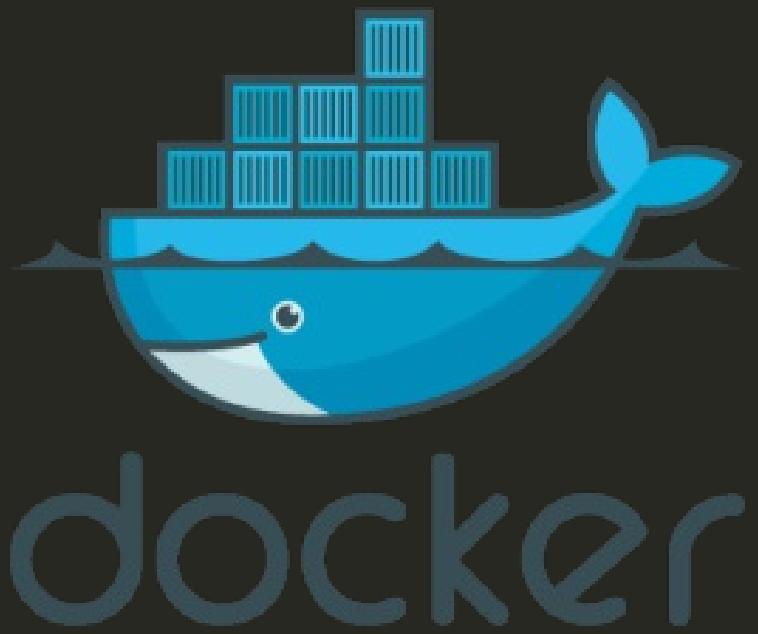
The screenshot shows a GitHub repository page for 'openfaas / faas'. The repository has 238 stars and 373 forks. The 'Code' tab is selected, showing the file 'handler.R' under the branch 'master'. The commit history shows a single commit by 'mrchypark' adding an R sample function and README, with the commit ID 'fdb67e3' and date 'on 1 Sep'. The code editor displays the following R script:

```
1 #!/usr/bin/env Rscript
2
3 f <- file("stdin")
4 open(f)
5 line<-readLines(f, n=1, warn = FALSE)
6
7 write(paste0("Hi ", line), stderr())
```

서버는 몰라도

docker를 좀 알면 편합니다.

```
FROM artemklevtsov/r-alpine:latest
ADD https://github.com/openfaas/faas/releases/download/0.6.1/fwatchdog /usr/bin
RUN chmod +x /usr/bin/fwatchdog
WORKDIR /root/
COPY handler.R .
ENV fprocess="Rscript handler.R"
HEALTHCHECK --interval=1s CMD [ -e /tmp/.lock ] || exit 1
CMD ["fwatchdog"]
```



는 시장입니다.

R로 api 서버를 만드는 마지막 방법!

저한테 연락하시면 됩니다.



끝!

<https://mrchypark.github.io/apiR>

[pdf버전] [문의하기] [의견 및 오류 신고]