SWPP Practice Session #11

DB Optimization + Deployment pt.2

TOC

1. DB Optimization
2. Deployment pt.2

Database Optimization

Database optimization

- Why important?
  - Because the database hit is expensive
- Misusing your database queries makes your app slow
  - Scraping whole table
  - Redundant, repetitive queries
  - etc

Do not evaluate the whole `QuerySet`

- Django's `QuerySet` is lazy
- `QuerySet` is evaluated when
  - Iterating, Slicing
  - Pickling, Caching
  - `len()`, `repr()`, `list()`
- Evaluating large `QuerySet` will slow down your whole database actions

Setup

- Assuming:
  - You are in Linux or Mac environment
  - You can create and activate virtualenv as Python 3
  - You can install Django by `pip` (or already installed one)

- Before you start:
  - Install Django (`$ pip install django`)
  - Install Locust (`$ pip install locust`)
  - Clone the following repo. (You don't need to fork it.)
    - `git clone https://github.com/swpp22fall-practice-sessions/swpp-p11-optimization`
    - `cd swpp-p11-optimization`
    - `cd item && python manage.py migrate`

Example project

```python
from django.db import models

class Item(models.Model):
    name = modles.CharField(max_length=30)
    stock = models.PositiveIntegerField(default=0)
```

- Create a simple table and migrate (don't forget to add app to settings.py)
- … and create 10,000 items using django shell

```python
charset = string.ascii_
for i in range(10000):
    random_name = ''.jo
    random_stock = rand
    Item.objects.create
    print(f'Created {ra

print('Done!')
```

```
# django shell (python manage.py shell)

import random
from app.models import Item
import string

charset = string.ascii_letters + string.digits
for i in range(10000):
    rand_name = ''.join(random.choices(charset, k=20))
    rand_stock = random.randrange(1024)
    Item.objects.create(name=rand_name, stock=rand_stock)
    print(f'Created {rand_name} item')

print('Done'
```

How to use Django shell

$ python manage.py shell

```
import random
from app.models import Item
import string

charset = string.ascii_letters + string.digits
for i in range(10000):
    rand_name = ''.join(random.choices(charset, k=20))
    rand_stock = random.randrange(1024)
    Item.objects.create(name=rand_name, stock=rand_stock)
    print(f'Created {rand_name} item')

print('Done')
```

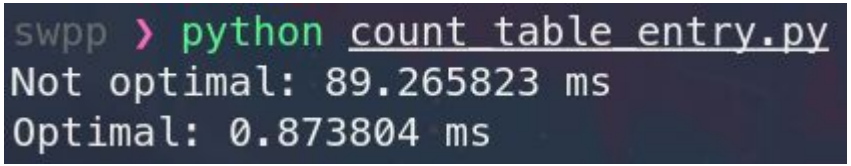
Test it

- Write a test script that counts the table entries
  - Calling `len()` and `.count()`

```
 8 from app.models import Item
 9 from time import time
10
11 start = time()
12 count = len(Item.objects.all())
13 print('Not optimal: {:3f} ms'.format((time() - start) * 1000))
14
15 start = time()
16 count = Item.objects.count()
17 print('Optimal: {:3f} ms'.format((time() - start) * 1000))
```

Test result

- `len()` is about 100x slower than `.count()`
  - Internally, len() will count after the `SELECT * FROM some_table query`, while count() will call `SELECT COUNT(*) some_table` query
  - This will become more serious when your database grows



Use appropriate field type

- Keep your database small
  - name is enough to be `CharField`, not `TextField`
- `DateTimeField` vs `DateField`
- `IntegerField` vs `SmallIntegerField` vs `BigIntegerField`
- https://docs.djangoproject.com/en/3.1/ref/models/fields/
- However, databases nowadays support high speed regardless of these types

Use `.values()` and `.values_list()`

- Don’t retrieve the fields that you do not need
  - **`values()`** and **`values_list()`** are used to retrieve a subset of data without the overhead of creating a model instance
  - Reduce the database overhead
  - Retrieve as Python `dict` or `list`, not model object itself

```
>>> from app.models import Item
>>>
>>> item = Item.objects.filter(id=1).values('name')[0]
>>>
>>> item
{'name': 'hERDKDozPIK3RH3C9bQM'}
>>>
```

Use DB index

- DB index can boost your query speed, such as `filter()`
- Consider adding DB index into the fields that you frequently use `filter()`, `exclude()`, `order_by()`, etc
  - Usually, the field use as primary key is indexed
- However, you'd better not overuse index, since indexes are updated every time you do inserts, updates, deletes and therefore involve extra costs

```
class Item(models.Model):
    name = models.CharField(max_length=30, db_index=True)
    stock = models.PositiveIntegerField(default=0)
```

Use Transaction

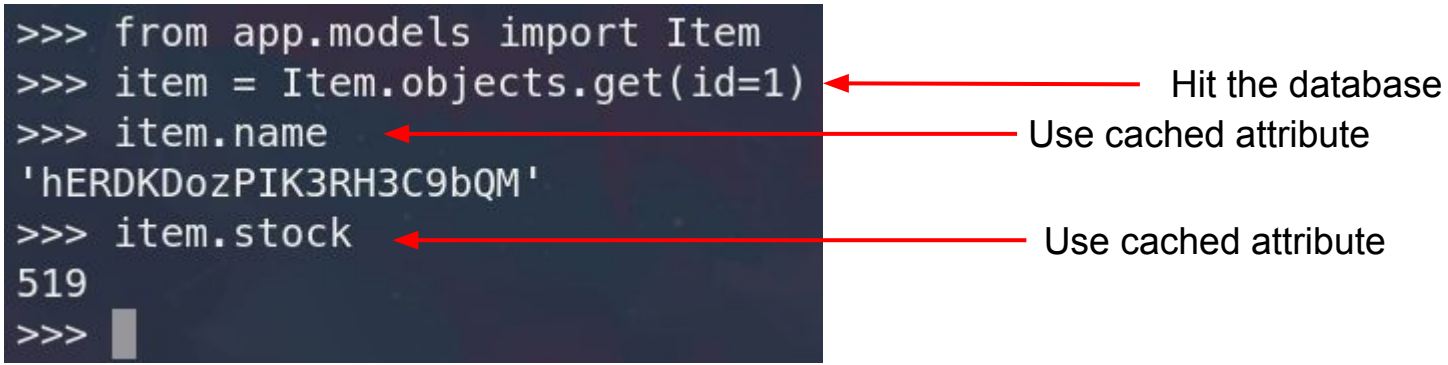
- Enclose several queries in a transaction
  - Commit at once
  - Rollback to initial state when exceptions are raised
- Transaction will help you to keep your database state integrity

```python
from django.db import transaction

@transaction.atomic
def viewfunc(request):
    # This code executes inside a transaction.
    do_stuff()
```

```python
from django.db import transaction

def viewfunc(request):
    # This code executes in autocommit mode (Django's default).
    do_stuff()

    with transaction.atomic():
        # This code executes inside a transaction.
        do_more_stuff()
```

Understand cached attributes

- Once your `QuerySet` or model object is evaluated, its attributes are cached
  - Same goes for related objects
    - `ForeignKey`, `ManyToManyFields`



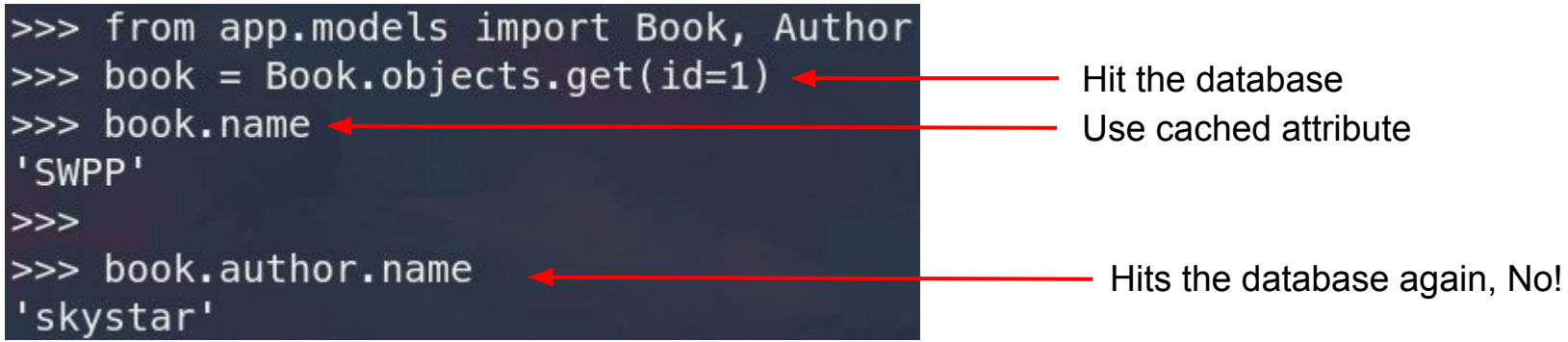
Retrieve related fields in one query

- Example tables

```
class Author(models.Model):
    name = models.CharField(max_length=30)


class Book(models.Model):
    name = models.CharField(max_length=30)
    author = models.ForeignKey(Author, on_delete=models.CASCADE)
```

Retrieve related fields in one query

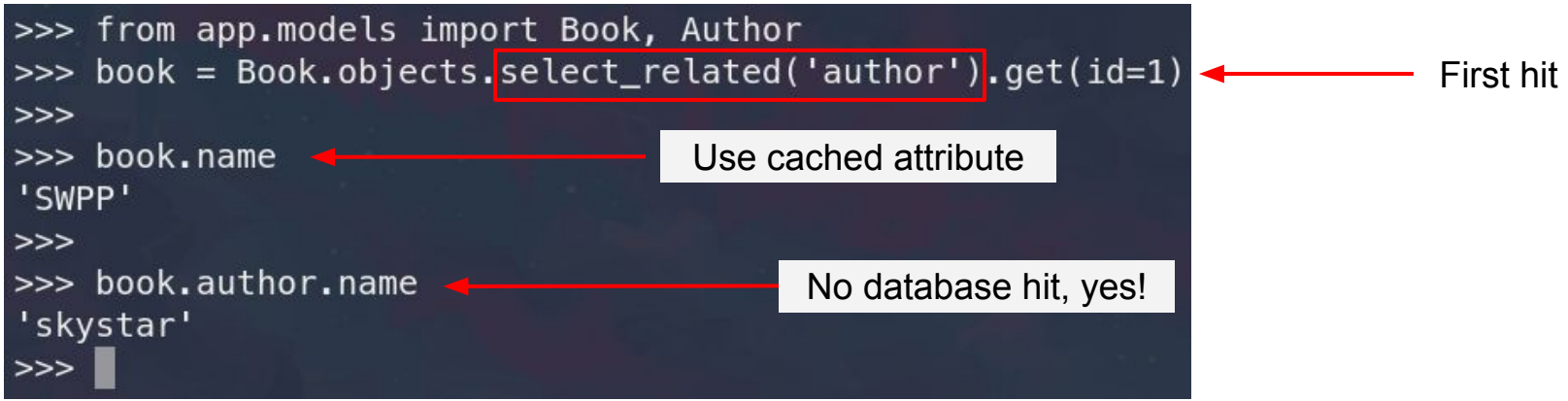
- Query a Book object and lookup its name and author's name



Any solution?

Use `select_related()`

- Prepopulate the related field in one query
  - Use `prefetch_related()` for the case of `ManyToManyFields`



Efficient!

Use ForeignKey value directly

- If you know the ForeignKey's primary key(id), you can use it directly
  - Append '_id' to the field name
  - More efficient due to reduced queries

```
>>> author = Author.objects.get(id=1)
>>>
>>> new_book = Book.objects.create(name='Hello', author=author)
>>>
```

vs

```
>>> new_book = Book.objects.create(name='World', author_id=1)
>>>
```

Use another database backend

- The default DB backend used by django is SQLite
- SQLite is easy to use and manage, but
  - Slow
  - Fail at large number of simultaneous connections
- Use battle-proven Database backends
  - PostgreSQL
  - MySQL
  - etc

Conclusion

- Profile and inspect raw SQL queries when needed
- Proper use of DB query will boost your whole application speed

- https://docs.djangoproject.com/en/2.1/topics/db/optimization/

Cache

Why cache?

- Database is basically file-based storage
  - Disk IO is desperately slow than DRAM access
- Solution?
  - Use memory-based cache to reduce access time
  - Use cache in frequently used, or transient data
    - Session
    - Game state
    - ...

Cache backends

- Local memory cache (default)
  - Fastest
- Database cache
  - If you have fast and well-indexed database server
  - Persistent
- External cache backends
  - Redis
  - Memcached
  - ...

Setting up cache backend

- CACHES setting in your root configuration file
- https://docs.djangoproject.com/en/3.1/topics/cache/

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
        'LOCATION': 'unique-snowflake',
    }
}
```

Cache the view

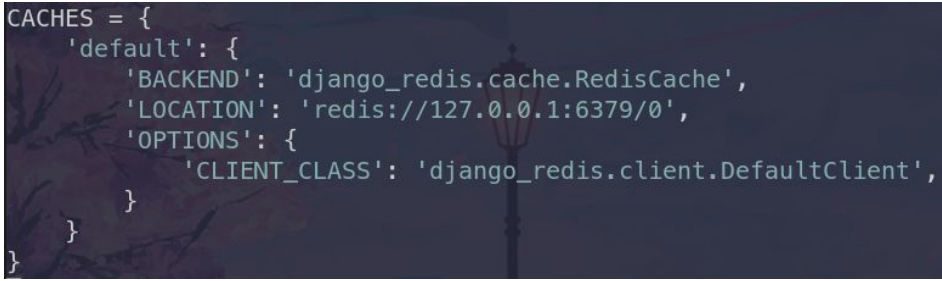
- Use `cache_page` decorator
  - This will cache your view per-url
  - Set timeout at decorator parameter

```python
from django.views.decorators.cache import cache_page

@cache_page(60 * 15)
def my_view(request):
    ...
```

will be cached for 15 mins

Use Redis as your cache backend

- Redis is a fast, scalable and versatile in-memory database
  - https://redis.io/topics/introduction
- You have to download Redis and run it on your server
  - `sudo apt-get install redis-server && redis-server` (for ubuntu)
  - `brew install redis` (for mac)
- No native support of Django, so use an adapter(https://github.com/niwinz/django-redis)
  - `$ python -m pip install django-redis`
- Set your cache backend as `django-redis`
- run redis: `$ redis-server`



Use cache as session engine

- Session data is frequently used
  - Storing and reading session data with database is not that good idea
- Use cache based session engine
  - Set at your root configuration file
- Redis also persist your data, so your session data will remain even after you restart the server
- https://docs.djangoproject.com/en/3.1/topics/http/sessions/#using-cached-sessions

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
SESSION_CACHE_ALIAS = 'default'
```

Accessing the raw cache

- Import `django.core.cache.cache`
- You can also set expire of cache entry
  - `cache.set(key, value, timeout=<timeout>)`
- Cleverly use to make fast your application
  - But be careful of simultaneous access, always (data races)

```
>>> from django.core.cache import cache
>>>
>>> cache.set('some_key', 'SWPP')
>>>
>>> cache.get('some_key')
'SWPP'
>>>
```

Load Test

Load Testing

- Figure out how many concurrent user a system can handle
- Attack your application and find bottlenecks

Locust

- Scalable user load testing tool
- https://github.com/locustio/locust
  - `$ pip install locust`

Write a locust file

- To represent a user, (or swarming locust), you can create a class inheriting `User` class
- Inside the class, you can define several options with predefined attributes (e.g. `wait_time`)
  - For more options and details, refer to this link
- A function annotated by `@task` defines the task performed by the user

locustfile.py

```python
from locust import HttpUser, between, task

class WebsiteUser(HttpUser):
    # wait_time determines the time gap between user task execution
    wait_time = between(3, 8)

    @task
    def my_task(self):
        self.client.get('/')
```

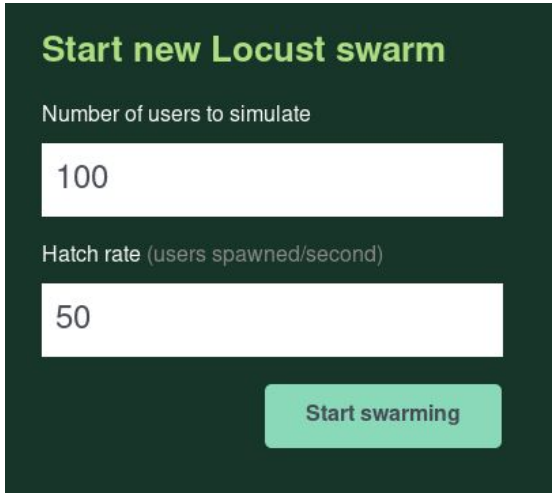
Example view function

- Create a new `Item` object for each request

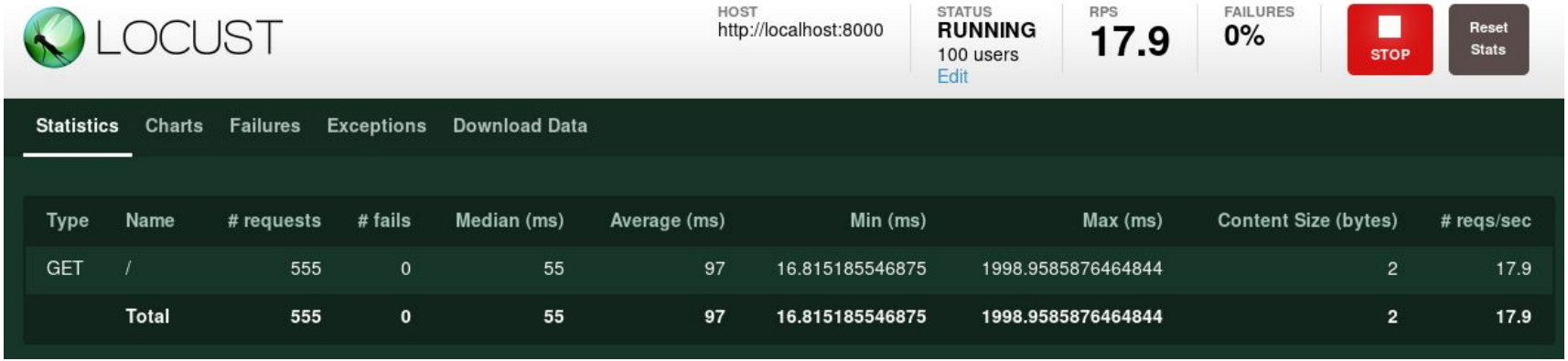
```
 8 def create_item(request):
 9     if request.method == 'GET':
10         charset = string.ascii_letters + string.digits
11         random_name = ''.join(random.choices(charset, k=20))
12
13         # create an item of random name
14         Item.objects.create(name=random_name)
15
16         return HttpResponse('OK')
17     else:
18         return HttpResponseNotAllowed(['GET'])
```

Run Locust

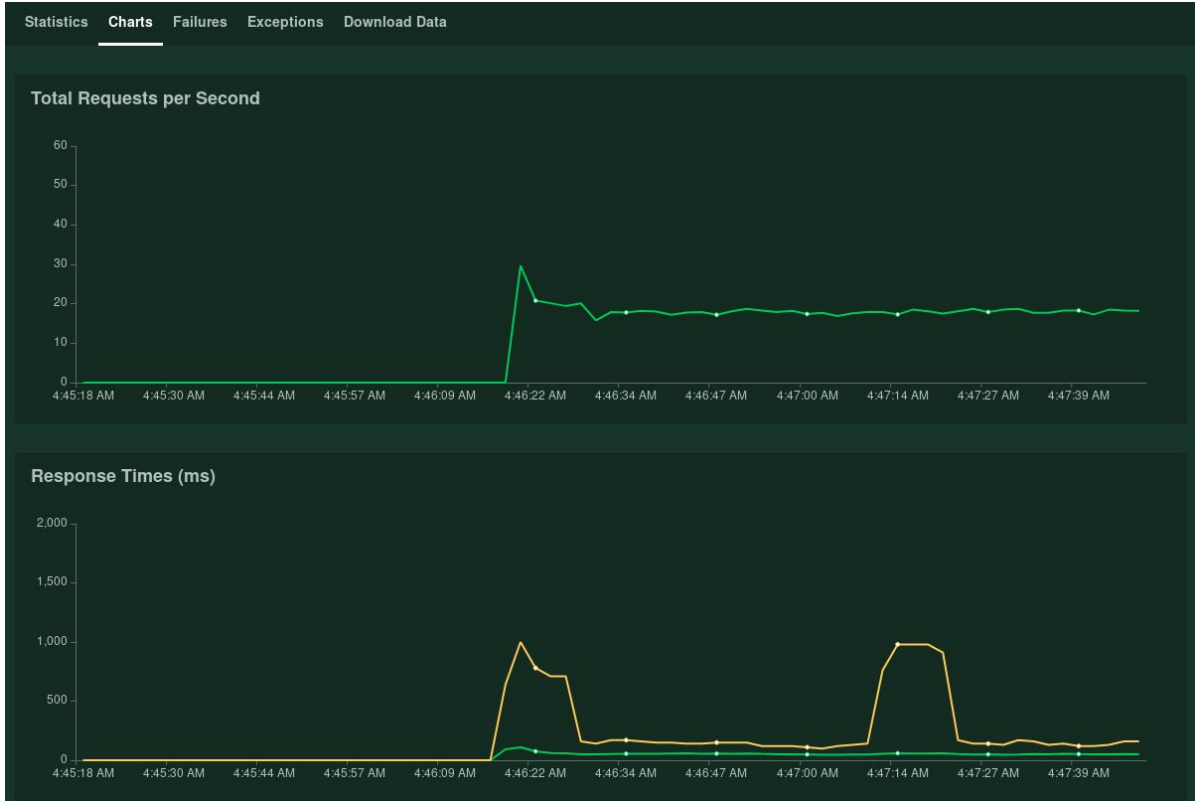
- `$ python manage.py runserver`
- `$ locust -f locustfile.py --host='http://localhost:8000'`
- Enter http://0.0.0.0:8089 on your browser to open the web-ui



Run Locust

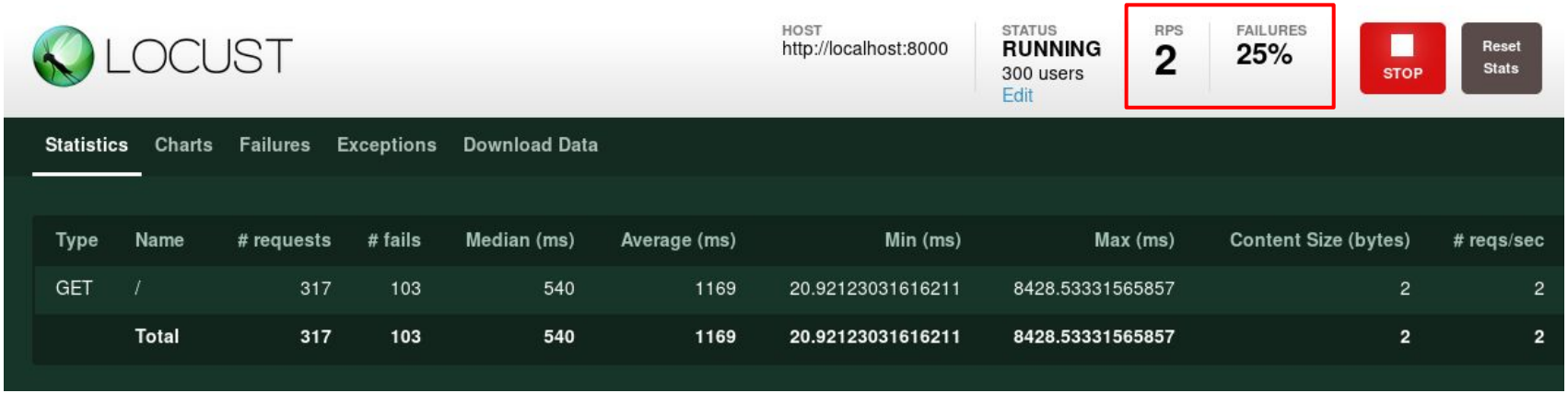

LOCUST
HOST
http://localhost:8000
STATUS
RUNNING
100 users
Edit
RPS
17.9
FAILURES
0%
STOP
Reset Stats
Statistics
Charts
Failures
Exceptions
Download Data
Type Name # requests # fails Median (ms) Average (ms) Min (ms) Max (ms) Content Size (bytes) # reqs/sec
GET / 555 0 55 97 16.815185546875 1998.9585876464844 2 17.9
Total 555 0 55 97 16.815185546875 1998.9585876464844 2 17.9

Run Locust


Statistics
Charts
Failures
Exceptions
Download Data
Total Requests per Second
60
50
40
30
20
10
0
4:45:18 AM
4:45:30 AM
4:45:44 AM
4:45:57 AM
4:46:09 AM
4:46:22 AM
4:46:34 AM
4:46:47 AM
4:47:00 AM
4:47:14 AM
4:47:27 AM
4:47:39 AM
Response Times (ms)
2,000
1,500
1,000
500
0
4:45:18 AM
4:45:30 AM
4:45:44 AM
4:45:57 AM
4:46:09 AM
4:46:22 AM
4:46:34 AM
4:46:47 AM
4:47:00 AM
4:47:14 AM
4:47:27 AM
4:47:39 AM

Attack your server with heavy load

- **How about 1000 concurrent users?**



What's going on?

Inspection

- `django.db.utils.OperationalError: database is locked`
- SQLite is not good at handling many concurrent requests
  - Just as its name implies… **Lite** :(

```
django.db.utils.OperationalError: database is locked
[23/Nov/2018 19:50:52] "GET / HTTP/1.1" 500 155054
```

  - Use another database backend
    - Try to change database backend from SQLite to PostgreSQL or MySQL
    - Check if the new database backend can handle 1000 concurrent users!

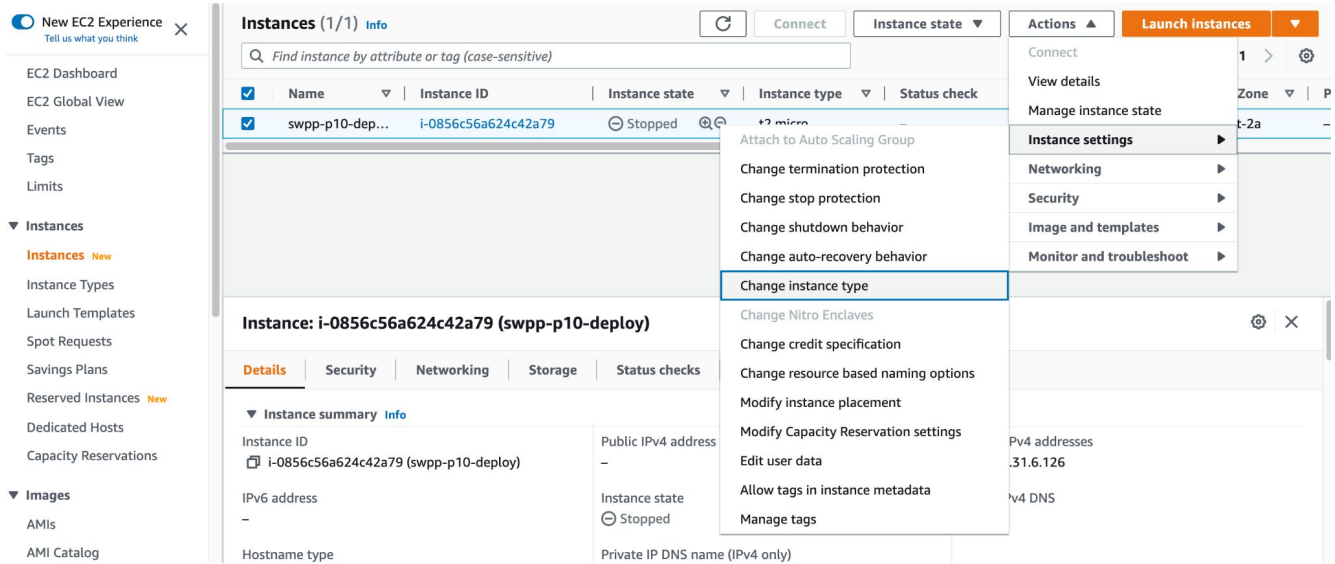
Setup Web Server

We will deploy our service on open domain, using Web Server & HTTPS

https://swpp2022sample.shop/heros

Setup

1. Let’s change to an instance with a higher capacity (t2.medium)

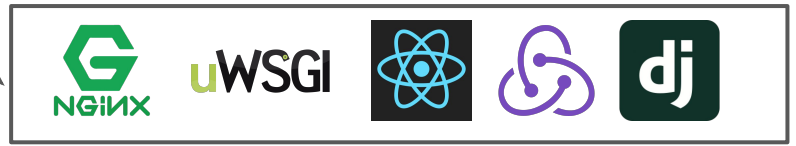


Setup

1. Start your ec2 instance
   - **We will develop everything inside your ec2 instance today**
   - Checkout this post. It will ease your ec2 execution.
     - You will be able to use VSCode inside ec2 instance
     - But you will have to change the public DNS part on every ec2 restart
2. Clone repo
   https://github.com/swpp22fall-practice-sessions/swpp-p11-deploy.pt2
3. Pull docker image
   - We installed `nginx` & `uwsgi`

```
$ sudo docker pull snuspl/swpp:practice11
```

contains

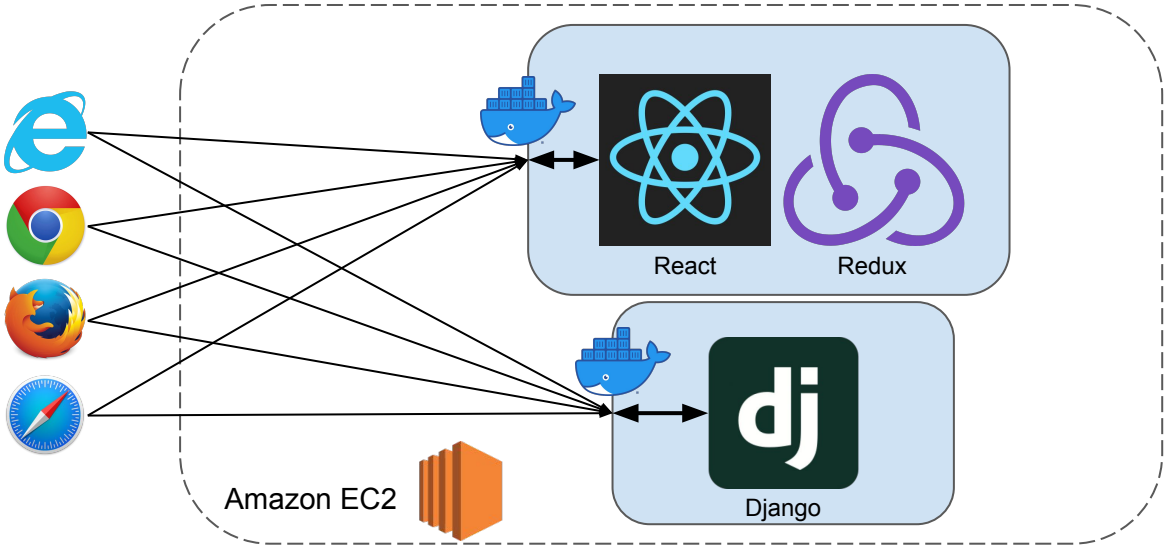


What is a web server?

- Communicator
  - Communicates with a web browser using the Hypertext Transfer Protocol (HTTP)
  - Directs request from one instance to another (proxy)
- Accelerator
  - Caching ⇒ Web acceleration of commonly requested contents
- Load balancer
  - Virtual hosting: Host a single website or multiple websites
  - Control loads of its instances
  - Limit the speed of response to different clients ⇒ Prevent resource domination

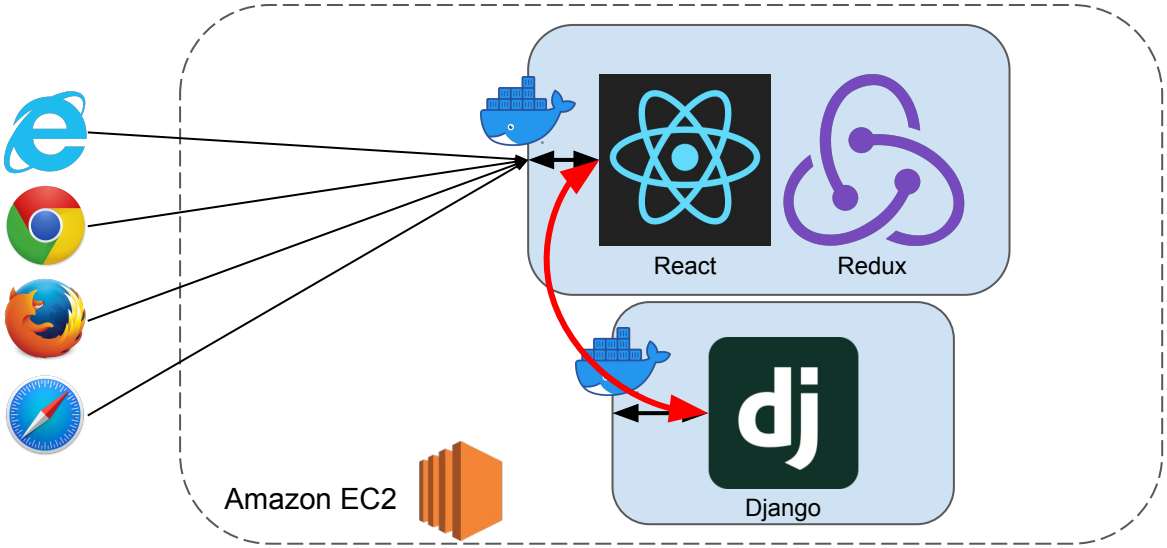
Why do we need to setup a web server?

1. Scalability
   - Think of a situation where many clients access the web application simultaneously.
   - Need an efficient request manager to handle multiple requests.



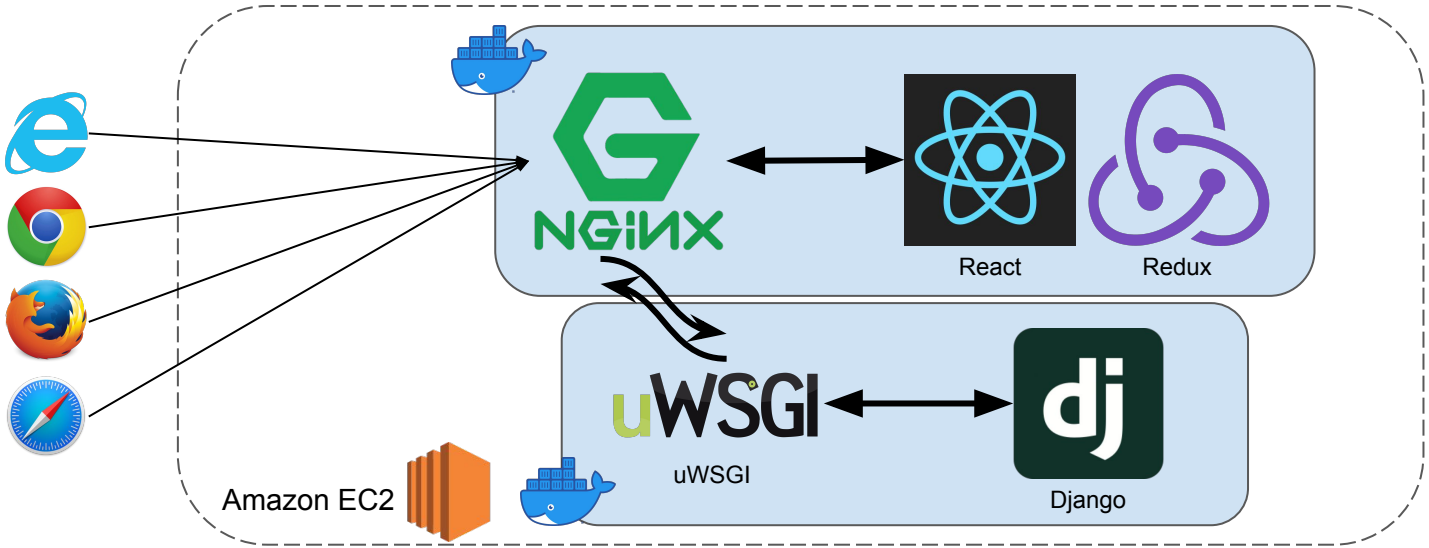
Why do we need to setup a web server?

2. Redirection
    - Need to redirect requests from one to another instance (e.g., from frontend to backend).



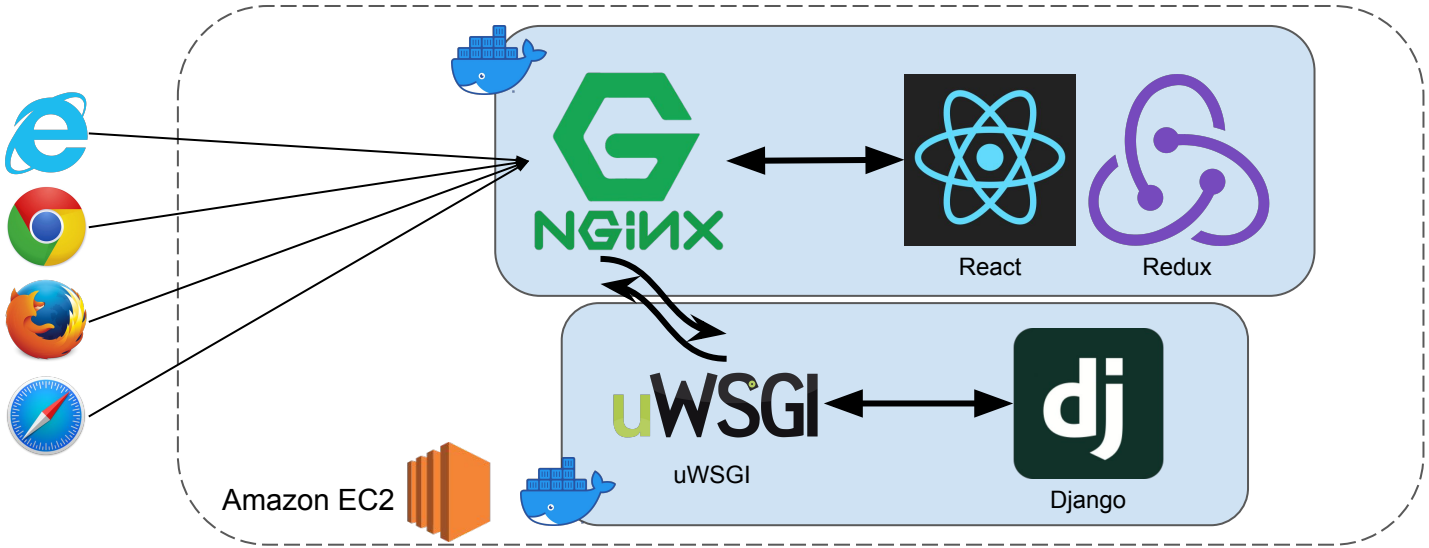
Why do we need to setup a web server?

- **NginX** is a web server software which can understand HTTP protocol.
  - Scalable: known to good at handling simultaneous requests.
  - Work as a proxy: can forward requests to other instances (e.g., backend).
  - Work as a load balancer: can balance the request loads if multiple web applications exist.



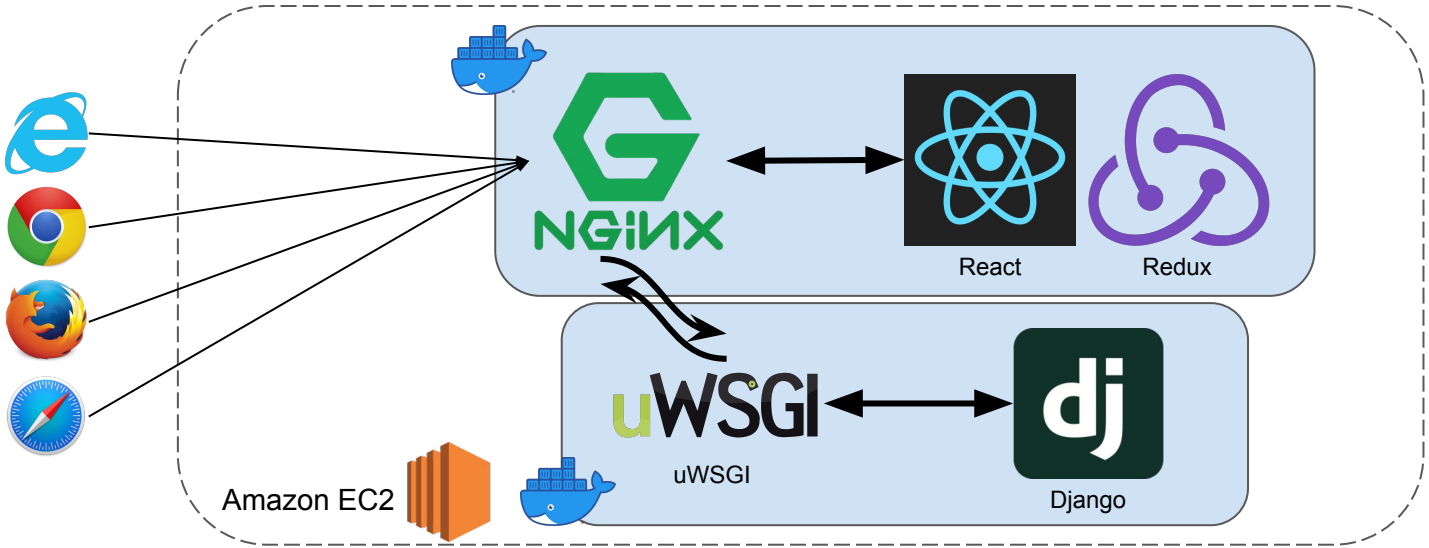
Why do we need to setup a web server?

- **uWSGI** == Web Server Gateway Interface.
  - describes how a web server (e.g., NginX) communicates with web applications (e.g., Django).
  - works as a Django wrapper
    - Logging, Monitoring, Load balancing, Resource Limiting …
    - Reference: https://uwsgi-docs.readthedocs.io/en/latest/WSGIquickstart.html

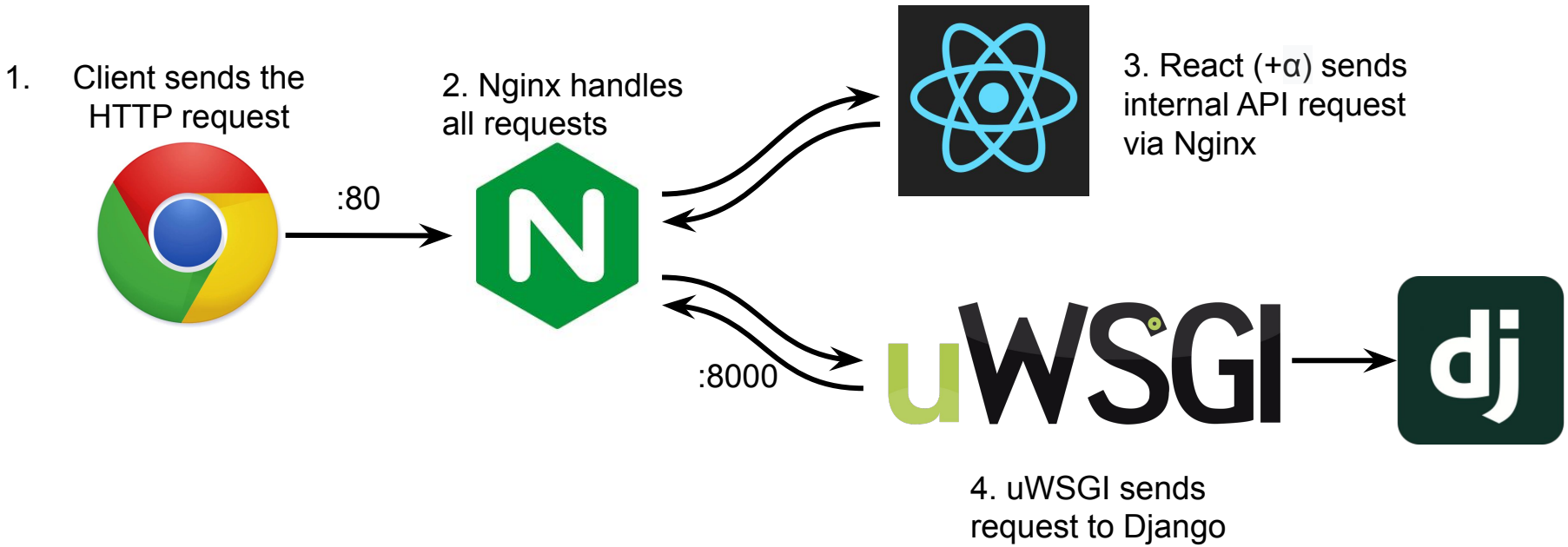


Why do we need to setup a web server?

- **NOTE:** We are not going to handle deployment issues (e.g., Load balancing, resource management) in detail.
  - We do not use NginX as a load balancer.
  - We use uWSGI merely as a Django wrapper. Refer to the document on how to limit resource, support



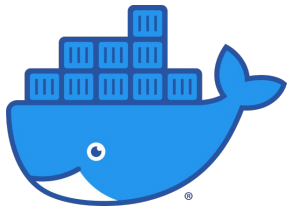
Deployment Layout



Backup - Running Docker container

```
docker run --rm -it \
    --ipc=host \
    --name "practice11" \
    -v ${PWD}:/home \
    snuspl/swpp:practice11 \
    /bin/bash
```

Deploying Backend with uWSGI & Docker



Let’s deploy the `hero` Django app with uWSGI

- `Hero` Django application in the repo.

- Django production check
  - Before you deploy your django application, you can check if your application is ready to be deployed with the following command(inside practice11 docker container):
  - `$ python manage.py check --deploy`

- `Refer to`
  - https://docs.djangoproject.com/en/3.2/howto/deployme nt/checklist/
  - https://blog.jun2.org/development/2019/07/23/django-security-options .html

Fix the errors before deploy

```
WARNINGS:
?: (security.W004) You have not set a value for the SECURE_HSTS_SECONDS setting. If your entire site is serve
d only over SSL, you may want to consider setting a value and enabling HTTP Strict Transport Security. Be sur
e to read the documentation first; enabling HSTS carelessly can cause serious, irreversible problems.
?: (security.W008) Your SECURE_SSL_REDIRECT setting is not set to True. Unless your site should be available
over both SSL and non-SSL connections, you may want to either set this setting True or configure a load balan
cer or reverse-proxy server to redirect all connections to HTTPS.
?: (security.W009) Your SECRET_KEY has less than 50 characters, less than 5 unique characters, or it's prefix
ed with 'django-insecure-' indicating that it was generated automatically by Django. Please generate a long a
nd random value, otherwise many of Django's security-critical features will be vulnerable to attack.
?: (security.W012) SESSION_COOKIE_SECURE is not set to True. Using a secure-only session cookie makes it more
 difficult for network traffic sniffers to hijack user sessions.
?: (security.W016) You have 'django.middleware.csrf.CsrfViewMiddleware' in your MIDDLEWARE, but you have not
set CSRF_COOKIE_SECURE to True. Using a secure-only CSRF cookie makes it more difficult for network traffic s
niffers to steal the CSRF token.
?: (security.W018) You should not have DEBUG set to True in deployment.
?: (security.W008) Your SECURE_SSL_REDIRECT setting is not set to True. Unless your site s
hould be available over both SSL and non-SSL connections, you may want to either set this
setting True or configure a load balancer or reverse-proxy server to redirect all connecti
ons to HTTPS.
```

Automation using Dockerfile

1. settings.py ENV binding
2. Default ENTRYPOINT bash script

settings.py ENV binding

Fix the errors from $ python manage.py check –deploy

```
# toh/settings.py
import os
…
# SECURITY WARNING: keep the secret key used in production secret!
secret_key_default = '${some_secret_key_over_50_letters}''
SECRET_KEY = os.environ.get('SECRET_KEY', secret_key_default)

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = os.environ.get('DEBUG', 'True') == 'True'                                    # default: True

ALLOWED_HOSTS = ['127.0.0.1','0.0.0.0', 'localhost', '${your_PUBLIC_DNS}', '${your_PUBLIC_IP}']

SECURE_HSTS_SECONDS = int(os.environ.get('SECURE_HSTS_SECONDS', 31536000))
SECURE_SSL_REDIRECT = os.environ.get('SECURE_SSL_REDIRECT', 'False') == 'True'                    # default: False
SESSION_COOKIE_SECURE = os.environ.get('SESSION_COOKIE_SECURE', 'False') == 'True'            # default: False
CSRF_COOKIE_SECURE = os.environ.get('CSRF_COOKIE_SECURE', 'False') == 'True'                    # default: False
SECURE_HSTS_INCLUDE_SUBDOMAINS = os.environ.get('SECURE_HSTS_INCLUDE_SUBDOMAINS', 'False') ==
'True'    # default: False
SECURE_HSTS_PRELOAD = os.environ.get('SECURE_HSTS_PRELOAD', 'False') == 'True'                # default: False
```

Automate settings.py ENV binding with Dockerfile

```
# backend/Dockerfile

FROM snuspl/swpp:practice11

VOLUME /app
WORKDIR /app

COPY . .

# Environment variables for django deployment
ENV DEBUG=False
ENV SECRET_KEY=${some_secret_key_over_50_letters}’
ENV SECURE_HSTS_SECONDS=31536000
# ENV SECURE_SSL_REDIRECT=True
ENV SESSION_COOKIE_SECURE=True
ENV CSRF_COOKIE_SECURE=True
ENV SECURE_HSTS_INCLUDE_SUBDOMAINS=True
ENV SECURE_HSTS_PRELOAD=True

ENTRYPOINT ./run_backend.sh
```

Let’s deploy the `hero` Django app with uWSGI

```
# backend/toh/uwsgi/uwsgi.ini

[uwsgi]
chdir = /app
module = toh.wsgi:application
socket = /app/toh.sock
enable-threads = true
master = true
vacuum = true
uid = root
gid = root
http = :8000
logto = /log/@(exec://date +%%Y-%%m-%%d).log
log-reopen = true
ignore-sigpipe = true
ignore-write-errors = true
disable-write-exception = true
post-buffering = 8192
processes = 1
threads = 2
no-orphans = 1
thunder-lock = true
```

- Writing uwsgi.ini configuration

Default ENTRYPOINT bash script with Dockerfile

```
# backend/run_backend.sh

#!/bin/bash

# TODO: Write automation script for launching BE app
python manage.py makemigrations
python manage.py migrate
mkdir -p /log # for `uwsgi` logging
uwsgi --ini uwsgi/uwsgi.ini
```

Automating migration on every docker container launch

Automating uwsgi server run on every docker container launch

Configuration file for uwsgi

Running Backend With Docker

- Build backend Docker image

```
sudo docker build -t backend .                    # inside backend/ directory
```

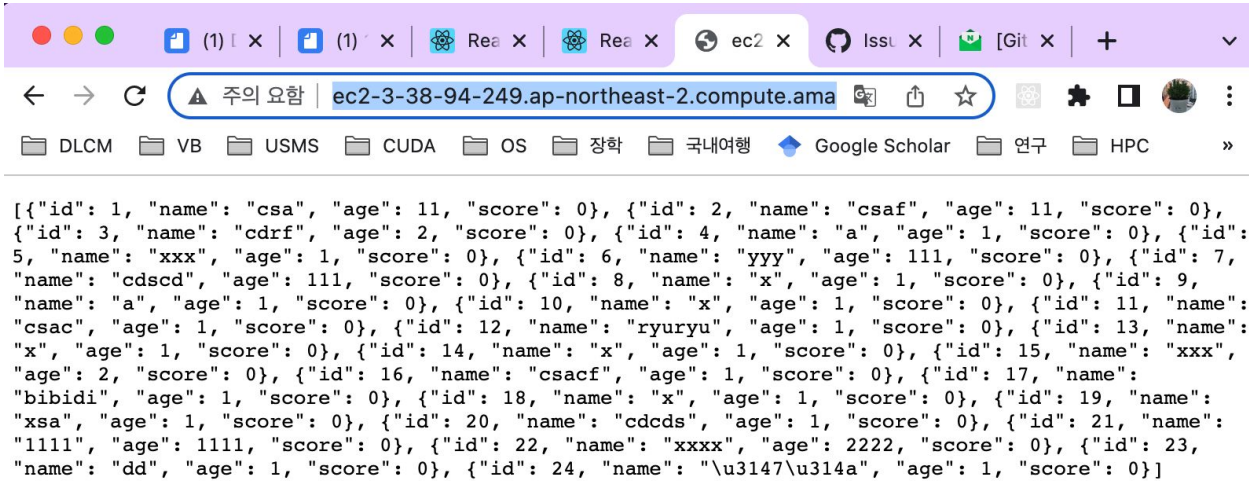
- Run backend image

```
sudo docker run -d –rm \
    --name "backend" \
    -p 8000:8000 \
    backend:latest
```

Run & Visit http://${your_PUBLIC_DNS}:8000/api/hero/ and check whether it works!

Running uWSGI With Docker

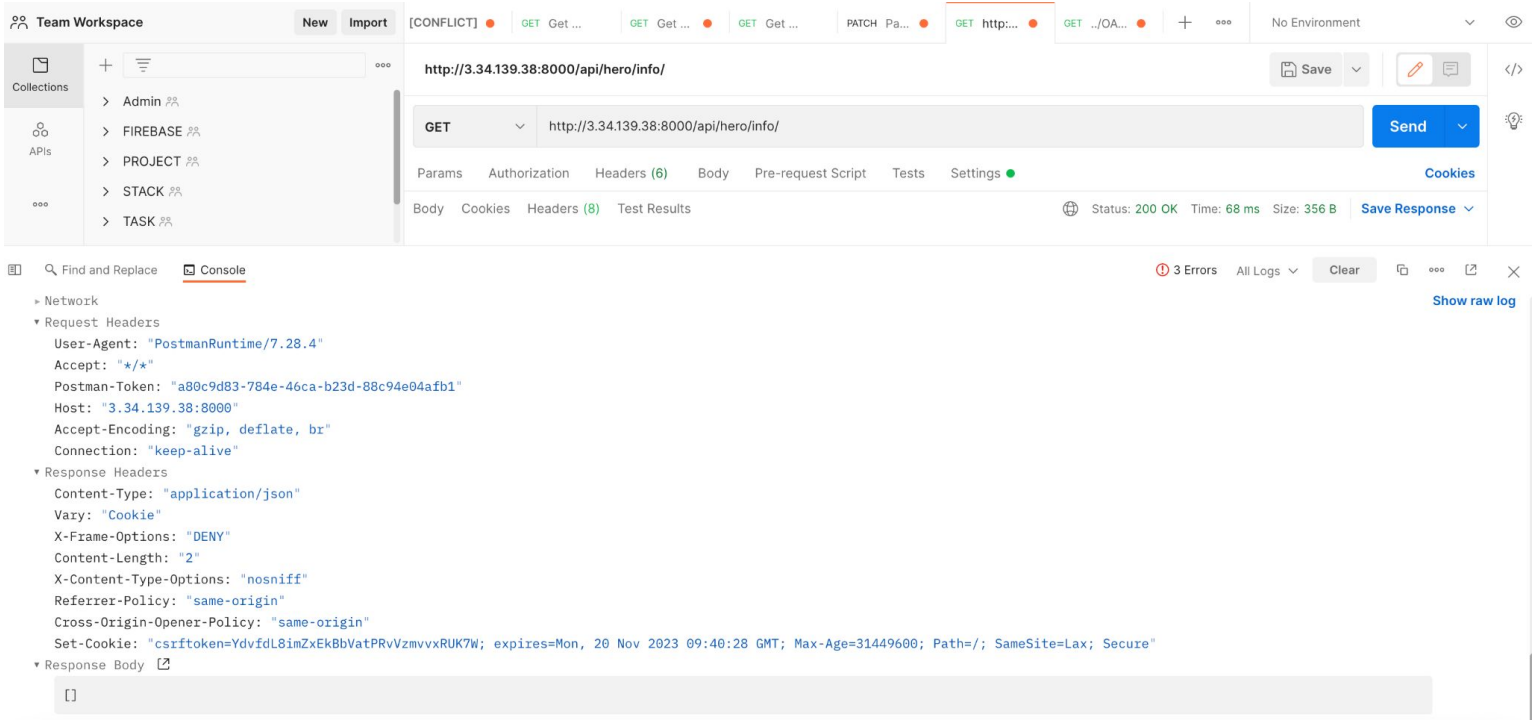
Run & Visit http://${your_PUBLIC_DNS_OR_IP}:8000/api/hero/info and check whether it works!



- Will show entries in your DB
- If it takes time, first try with Postman

Running uWSGI With Docker

You can also check http://${your_PUBLIC_DNS_OR_IP}:8000/api/hero/ with Postman



Mounting DB

- Below command will loss its DB updates on every container re-launch

```
sudo docker run -d –rm \
    --name "backend" \
    -p 8000:8000 \
    backend:latest
```

Because we save the DB data on pre-mounted volume

```
# backend/Dockerfile
FROM snuspl/swpp:practice11

VOLUME /app
WORKDIR /app
COPY . .
```

- To avoid this, add this option to the launch script

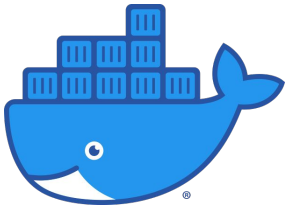
```
-v ${ABSOLUTE_DB_PATH}:${BACKEND_CONTAINER_DB_PATH}
```

Branch backend contains up to here

Code structure

- /backend
  - /run_backend.sh : Entrypoint for backend Docker container
- /frontend
- README.md
- run_docker_backend.sh : Backend Docker container launch script
- stop_docker_backend.sh : Backend Docker container terminate script

Deploying Frontend with Nginx & Docker



How to run NginX

- Don’t forget to install node modules
- Install nginx (already installed in the image)

```
$ sudo apt install nginx
```

- On the frontend directory, run the following command(inside practice11 Docker container):

```
$ npm run build --prod
```

  - This will automatically generate a new directory named ***build***
  - creates the production build files of a React application
- Run nginx

```
$ nginx -g ‘daemon off;’
```

We will automate this part (as we did in backend part)

Frontend - Configure Nginx

- Create a config file named `nginx.conf`
  (You will have to replace the bold characters to serve your project)

```
# frontend/nginx.conf
server {
  listen 3000;
  location / {
    root /usr/app/build;
    index index.html index.htm;
    try_files $uri $uri/ /index.html;
  }
  location /api/hero {
    proxy_pass http://{YOUR_DOMAIN OR IP}:8000/api/hero;
  }
}
```

meanwhile, port 80 is the standard for HTTP

Container path to `build` dir (generated by `npm run build`)

Proxy setting. replace the URL to AWS domain or IP.

Frontend - Configure Nginx

- Create a config file named `nginx.conf`
  (You will have to replace the bold characters to serve your project)

```
# frontend/nginx.conf
server {
  listen 3000;
  location / {
    root /usr/app/build;
    index index.html index.htm;
    try_files $uri $uri/ /index.html;
  }
  location /api/hero {
    proxy_pass http://{YOUR_DOMAIN OR IP}:8000/api/hero;
  }
}
```

```
export const fetchHeros = createAsyncThunk("hero/fetchHeros", async () => {
  const response = await axios.get<HeroType[]>("/api/hero/info/");
  return response.data;
});
```

```
urlpatterns = [
    path('api/hero/', include('hero.urls')),
    path('admin/', admin.site.urls),
]
```

Automatic Nginx configuring with Dockerfile

```
#frontend/Dockerfile
FROM snuspl/swpp:practice11

WORKDIR /usr/app

COPY package.json .
COPY src/ src/
COPY public/ public/
COPY run_frontend.sh .

RUN yarn install --silent
RUN npm run build --prod --silent

RUN apt-get update && apt-get install -y nginx
# should have made nginx configuration file to the frontend directory
COPY nginx.conf /etc/nginx/sites-available/nginx.conf
RUN rm /etc/nginx/sites-enabled/default
RUN ln -s /etc/nginx/sites-available/nginx.conf /etc/nginx/sites-enabled/nginx.conf
RUN mkdir -p /usr/share/nginx/html
RUN cp -r build/* /usr/share/nginx/html/

ENTRYPOINT ./run_frontend.sh
```

- On the frontend directory, run the following command(inside practice11 Docker container):

```
$ npm run build --prod
```

  - This will automatically generate a new directory named ***build***
  - creates the production build files of a React application
- Run nginx

```
$ nginx -g 'daemon off;'
```

We will automate this part (as we did in backend part)

Default ENTRYPOINT bash script with Dockerfile

```
#!/bin/bash

# TODO: Write automation script for launching FE app
nginx -g 'daemon off;'
```

Automating nginx server run on every docker container launch

Running Frontend With Docker

Build Docker image

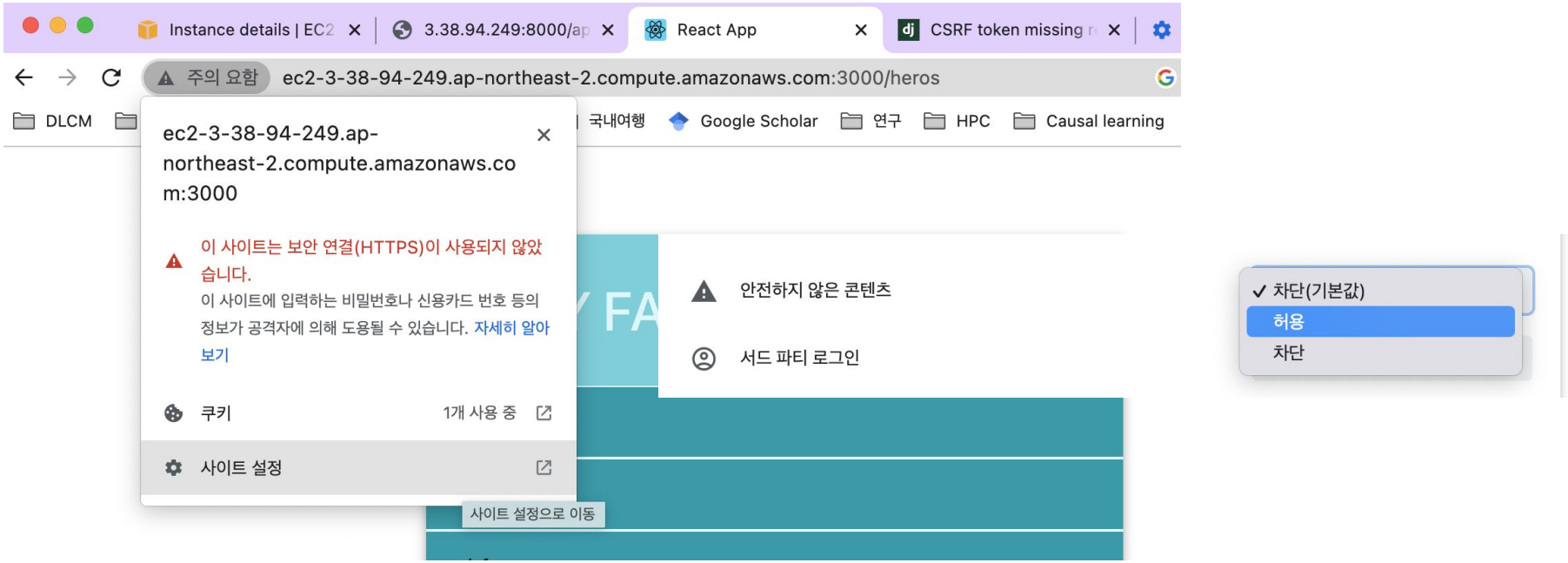
```
$ sudo docker build -t frontend .
```

Run the built Docker image

```
$ sudo docker run -d -p 3000:3000 --rm --name “frontend” frontend:latest
```

Visit http://ec2...:3000/heros and check whether it works

If your request stays pending…



Try this until we enable HTTPS

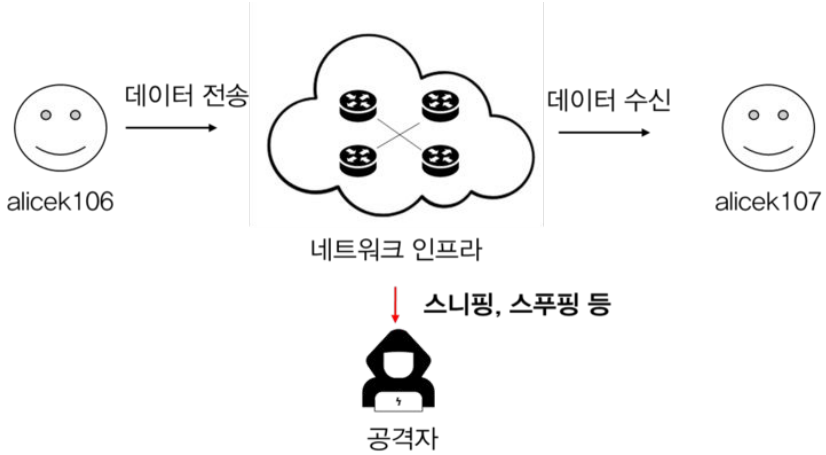
Branch frontend contains up to here

Code structure

- /backend
  - /run_backend.sh : Entrypoint for backend Docker container
- /frontend
  - /run_frontend.sh : Entrypoint for frontend Docker container
- README.md
- run_docker_backend.sh : Backend Docker container launch script
- stop_docker_backend.sh : Backend Docker container terminate script
- run_docker_frontend.sh : Frontend Docker container launch script
- stop_docker_frontend.sh : Frontend Docker container terminate script

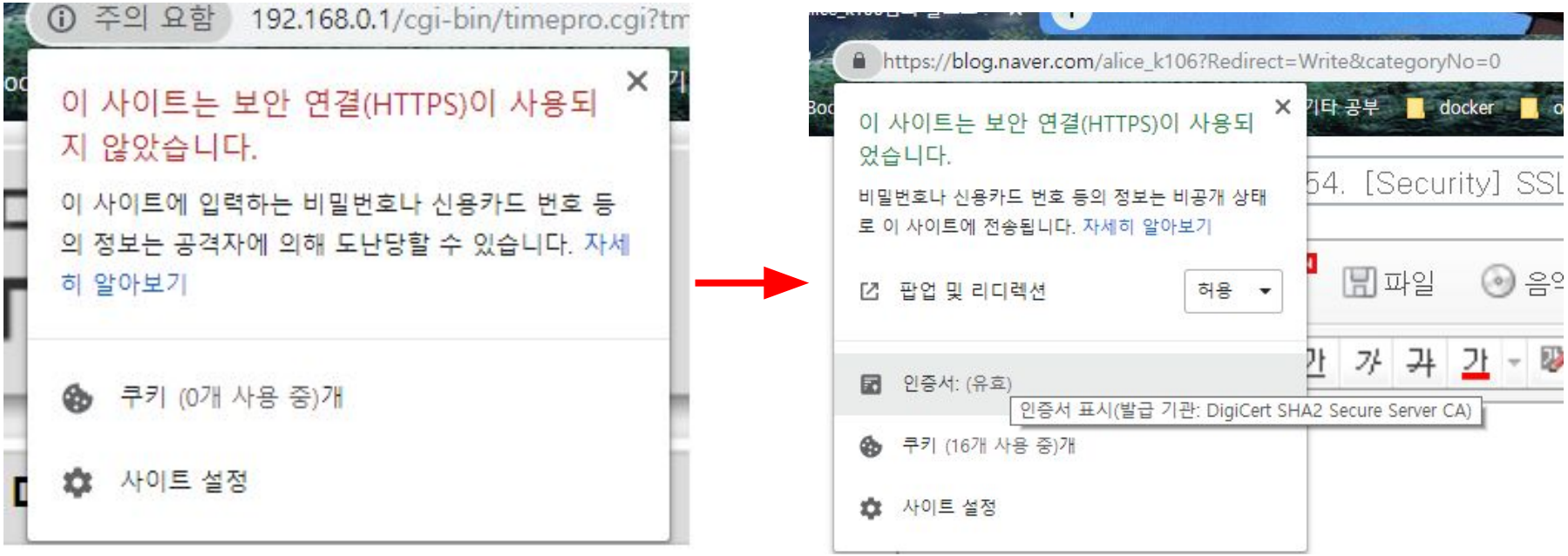
HTTPS (SSL)

What is HTTPS (SSL)



- HTTP: **H**yper**t**ext **T**ransfer **P**rotocol
  - Messages are not encrypted (not secure)
  - Vulnerable to packet sniffing
  - In other words, don’t use this
- HTTPS: **H**yper**t**ext **T**ransfer **P**rotocol over **S**SL
  - Messages are encrypted (secure)
  - Only the sender and receiver can know what is in the packet
- SSL: Secure Socket Layer (or TLS)
  - A protocol to provide communication security over a computer network

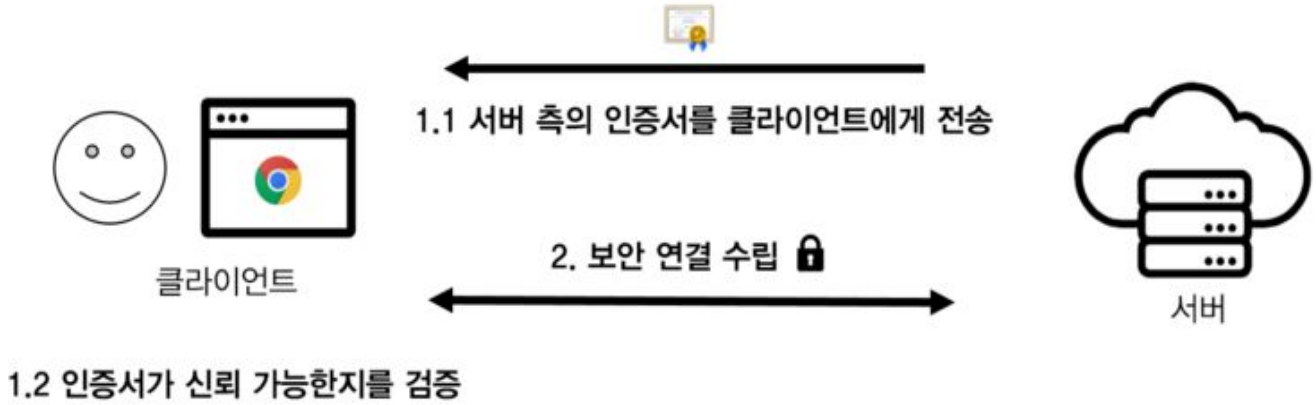
Our current deployed app does not use HTTPS



Secure connection between Server & Client

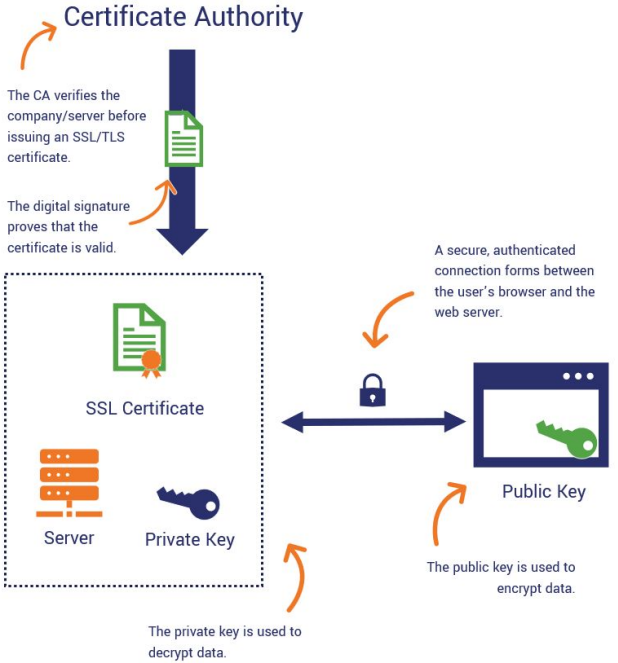
1. Client CA verifies Server SSL

2. Transfer encrypted data



Certificate Authority (CA)

- CA Manages SSL Certificates.
- With Certificates, CA verifies if the client is accessing a valid domain.
- In this practice, we will create our own domain and issue a SSL certificate.



Let's Encrypt and *Certbot*

- We will use *Let's Encrypt* and *Certbot*
- ***Let's Encrypt*** : a non-profit SSL CA
- ***Certbot*** : an automatic certification program for *Let's Encrypt*

How to get a certificate

1. Register your domain to the *domain registrars*.
    - *Domain registrars* are the maintainers of the registered domain names.
    - e.g. GoDaddy, Gabia, etc.
    - You're going to have to pay money for this :(
    - I have obtained my own domain from Gabia with ₩550 (for the first year).
        i. Please buy a domain for your team project later.
            - https://event.gabia.com/d_event_190201
            - Sorry, the page is in Korean :(

How to get a certificate

2. Install certbot

```
$ sudo add-apt-repository ppa:certbot/certbot
$ sudo apt-get update
$ sudo apt-get install -y python-certbot-nginx
```

How to get a certificate

3. Get your certificate

```
$ sudo certbot certonly --manual \
  --preferred-challenges dns \
  --server https://acme-v02.api.letsencrypt.org/directory \
  -d 'YOUR_DOMAIN_NAME_HERE'
```

```
ubuntu@ip-172-31-6-126:~/swpp-p11-deploy.pt2$ sudo certbot certonly --manual \
>   --preferred-challenges dns \
>   --server https://acme-v02.api.letsencrypt.org/directory \
>   -d 'swpp2022sample.shop'
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Plugins selected: Authenticator manual, Installer None
Enter email address (used for urgent renewal and security notices) (Enter 'c' to
cancel): gajagajago@snu.ac.kr
```

How to get a certificate

4. You have to set DNS txt record to verify that you are the owner of this domain

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Please deploy a DNS TXT record under the name
_acme-challenge.haeyooncho.shop with the following value:

N4E4en87EaBEWgXQHSwKB78jfUhJuQiIssHgpUFef3k

Before continuing, verify the record is deployed.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Tip!

1. ctrl + c to escape current certbot process
2. Register DNS TXT
3. Rerun certbot command

세팅 완료 되었습니다.

확인

DNS 관리

swpp2022sample.shop

• 레코드 개수 : 0개 • 최근 업데이트 : - • 네임서버 : ns.gabia.co.kr

이력

DNS 설정 레코드 수정

타입	호스트	값/위치	TTL	우선 순위	서비

DNS 레코드 수정

타입	호스트	값/위치	TTL	우선 순위	서비스	상태
TXT	_acme-challenge	xlGmUCdfSyz0U7ZSF6x6MzO4ZXlOC7RCJg95Auu_Tc4	600		DNS 설정	수정 삭제

+ 레코드 추가

저장 취소

도메인 연결

포워딩

레코드 종류	연결 서비스

```
Press Enter to Continue
Waiting for verification...
Cleaning up challenges

IMPORTANT NOTES:
 - Congratulations! Your certificate and chain have been saved at:
   /etc/letsencrypt/live/swpp2022sample.shop/fullchain.pem
   Your key file has been saved at:
   /etc/letsencrypt/live/swpp2022sample.shop/privkey.pem
   Your cert will expire on 2023-02-20. To obtain a new or tweaked
   version of this certificate in the future, simply run certbot
   again. To non-interactively renew *all* of your certificates, run
   "certbot renew"
 - If you like Certbot, please consider supporting our work by:

   Donating to ISRG / Let's Encrypt:   https://letsencrypt.org/donate
   Donating to EFF:                    https://eff.org/donate-le
```

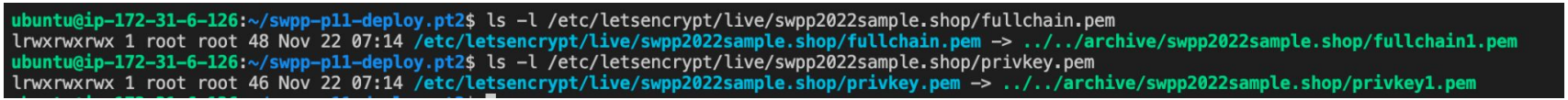
Add your server IP to the domain

DNS 설정 레코드 수정

타입 ?	호스트	값/위치	TTL	우선 순위	서비스
TXT	_acme-challenge	"zST0QJcxuDHiHebSF67yXwJWjf49R9S5aowl1Ts_kQg"	600		DNS 설정
A	@	3.83.138.221	600		DNS 설정

Mount ssl certificate to frontend container

You may have to `chmod -R 755` ssl certificate directory.
ssl_certificate and key is symbolic link. So we need to find its original file to mount by using `ls -l` command.



Change files accordingly…

run_docker_frontend.sh

```
#!/bin/bash

ENCRYPT_SSL_FULLCHAIN_PATH="/etc/letsencrypt/archive/swpp2022sample.shop/fullchain1.pem"
ENCRYPT_SSL_PRIVKEY_PATH="/etc/letsencrypt/archive/swpp2022sample.shop/privkey1.pem"

CONTAINER_SSL_FULLCHAIN_PATH="/usr/app/ssl/fullchain.pem"
CONTAINER_SSL_PRIVKEY_PATH="/usr/app/ssl/privkey.pem"

sudo docker run -it --rm \
    --name "frontend" \
    -p 443:443 \
    -v $ENCRYPT_SSL_FULLCHAIN_PATH/:$CONTAINER_SSL_FULLCHAIN_PATH \
    -v $ENCRYPT_SSL_PRIVKEY_PATH/:$CONTAINER_SSL_PRIVKEY_PATH \
    frontend:latest bash
```

Change files accordingly…

nginx.conf

```
# frontend/nginx.conf
server {
  server_name swpp2022sample.shop;
  listen 443 ssl http2;
  ssl on;
  ssl_certificate /usr/app/ssl/fullchain.pem;
  ssl_certificate_key /usr/app/ssl/privkey.pem;

  location / {
    root /usr/app/build;
    index index.html index.htm;
    try_files $uri $uri/ /index.html;
  }
  location /api/hero {
    proxy_pass http://ec2-3-38-94-249.ap-northeast-2.compute.amazonaws.com:8000/api/hero;
  }
}
```

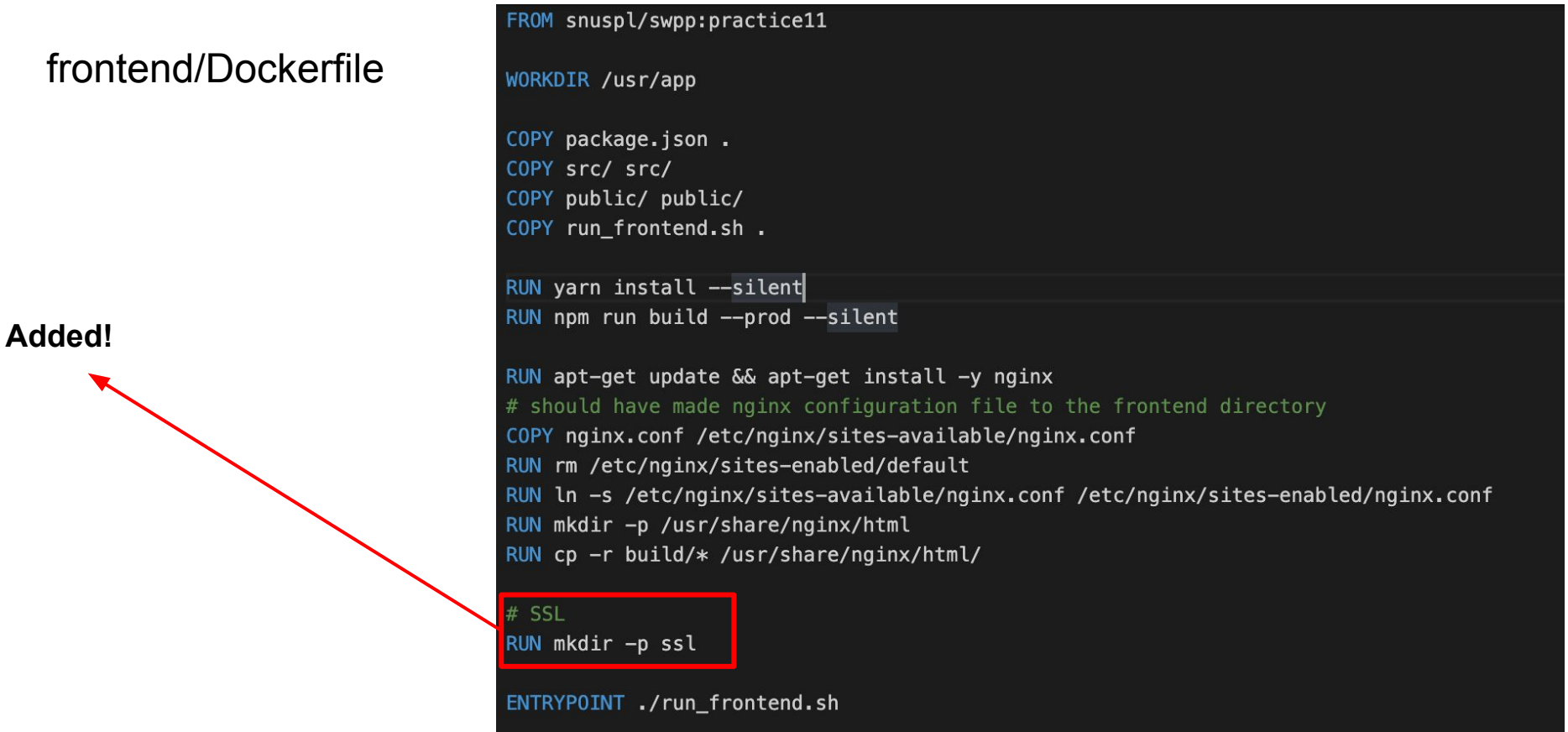
New port again :) Change the inbound rule!

```
proxy_pass http://{YOUR_DOMAIN OR IP}:8000/api/hero;
```

Change files accordingly…

frontend/Dockerfile

Added!

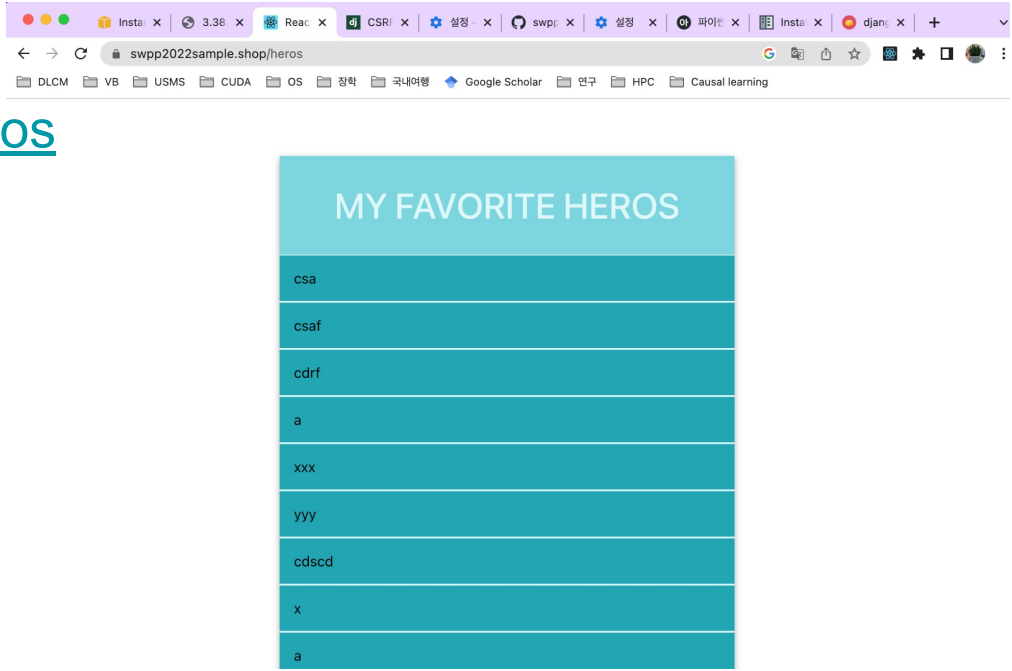


```
FROM snuspl/swpp:practice11

WORKDIR /usr/app

COPY package.json .
COPY src/ src/
COPY public/ public/
COPY run_frontend.sh .

RUN yarn install --silent
RUN npm run build --prod --silent

RUN apt-get update && apt-get install -y nginx
# should have made nginx configuration file to the frontend directory
COPY nginx.conf /etc/nginx/sites-available/nginx.conf
RUN rm /etc/nginx/sites-enabled/default
RUN ln -s /etc/nginx/sites-available/nginx.conf /etc/nginx/sites-enabled/nginx.conf
RUN mkdir -p /usr/share/nginx/html
RUN cp -r build/* /usr/share/nginx/html/

# SSL
RUN mkdir -p ssl

ENTRYPOINT ./run_frontend.sh
```

Finally!

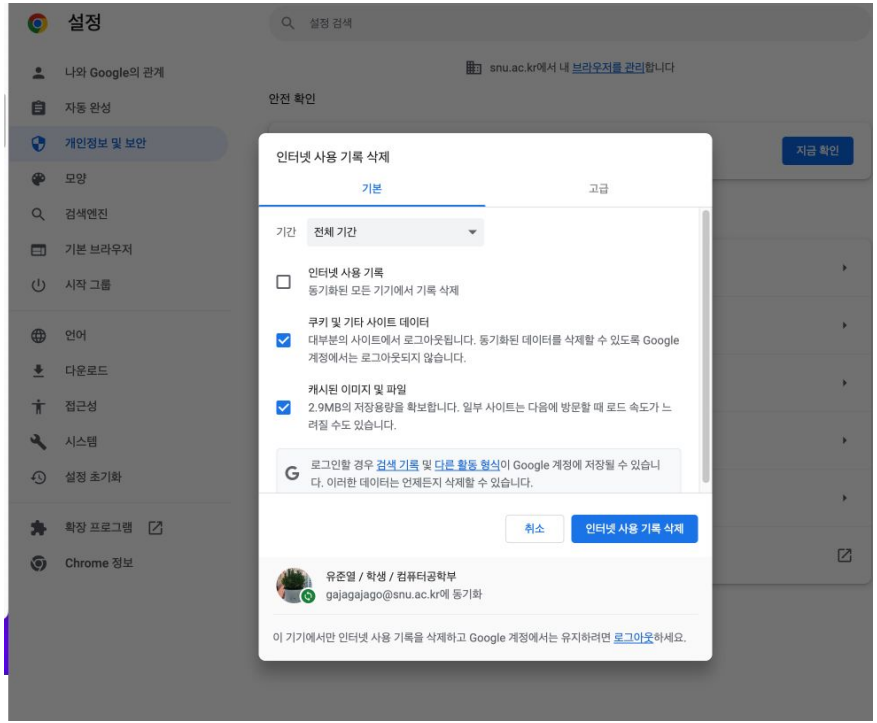
Branch https contains up to here

https://swpp2022sample.shop/heros



If there is some error…

- Try to relinquish the cache & retry!
- If CSRF token bugs you too much, we will allow disabling it for practice session 11 submission

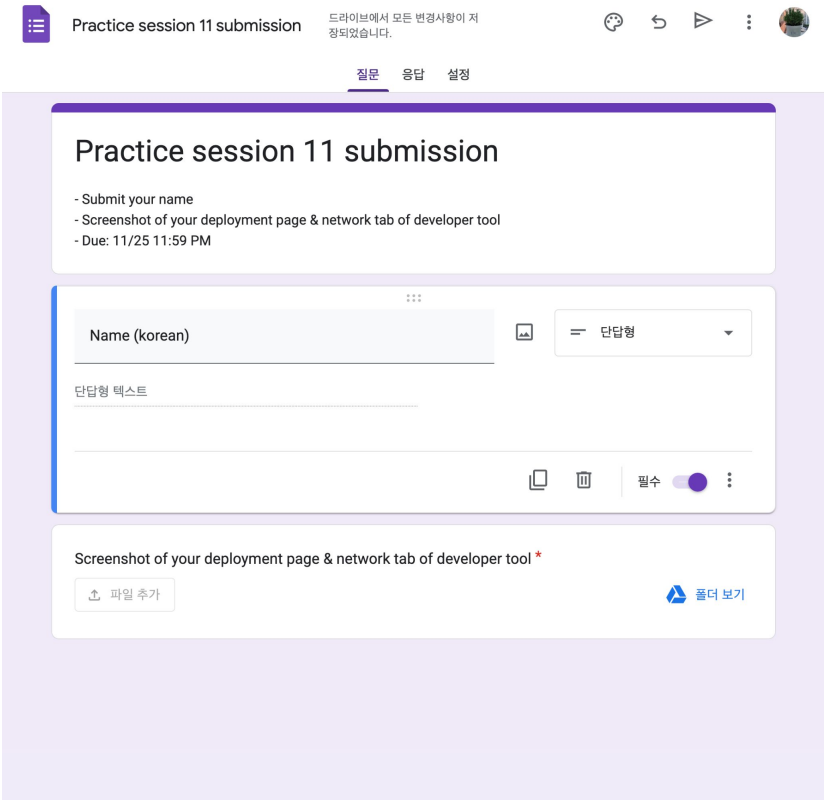


Practice Session #11 Submission

https://forms.gle/vUZw7rcSgsagZYyi7

Due 11/25 11:59 PM

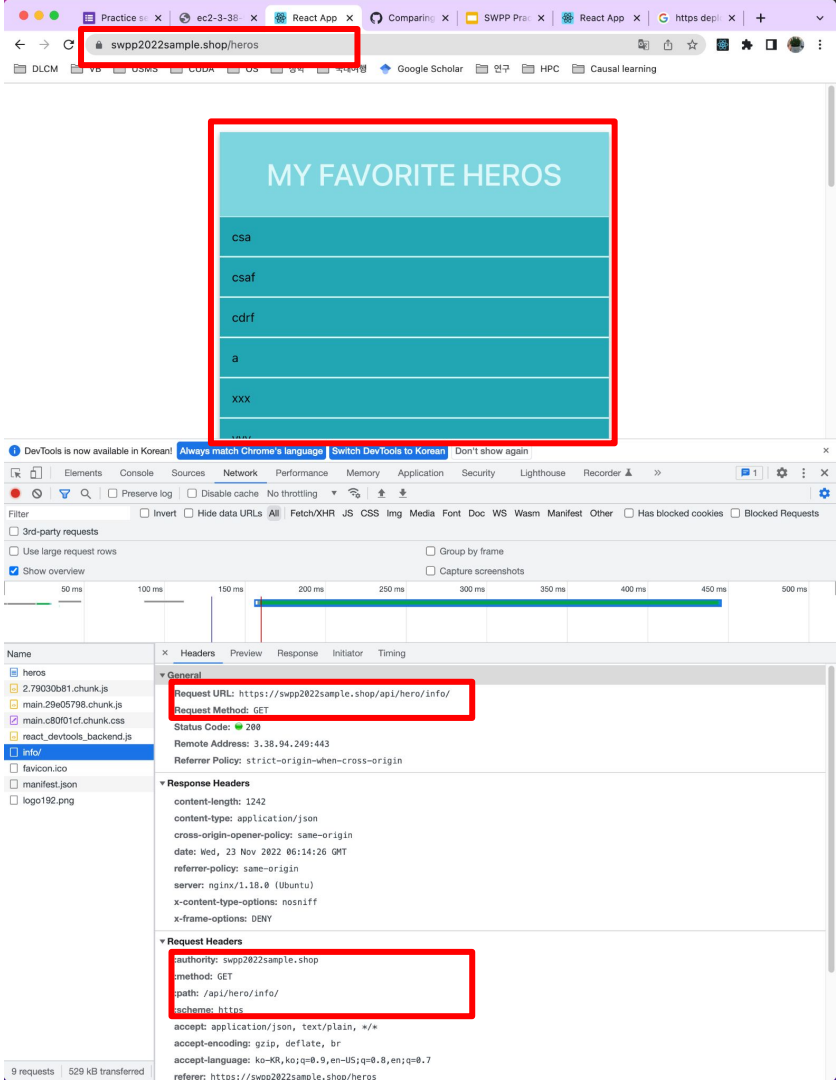
You may your instance & domain after submission. Not needed anymore!



Submission screenshot example

Things in red box are important

It must be identifiable on the screenshot



Certbot

You have to renew your certificate every 90 days

- Run `$ certbot renew`

Certbot (Docker)

Run certbot with Docker (replace ***.yourdomain.com** with your domain name)

```
$ docker run -it --rm --name certbot \
  -v '/etc/letsencrypt:/etc/letsencrypt' \
  -v '/var/lib/letsencrypt:/var/lib/letsencrypt' \
  certbot/certbot certonly -d '*.yourdomain.com' \
  --manual --preferred-challenges dns \
  --server https://acme-v02.api.letsencrypt.org/directory
```