

02. 파드(Pod)를 활용해 서버 띄워보기

1. 파드(Pod)란?

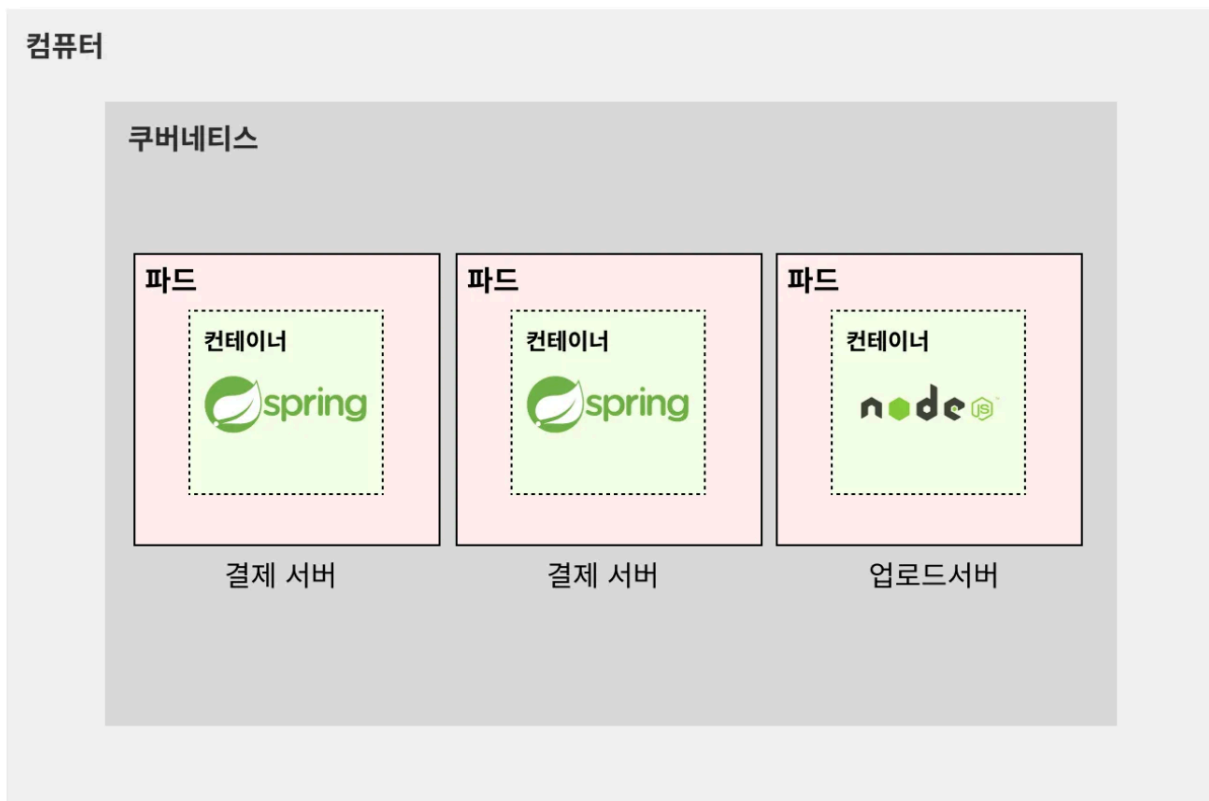
✓ 파드(Pod)란?

도커에서는 하나의 프로그램을 실행시키는 단위를 컨테이너라고 주로 불렀다.
쿠버네티스에서는 하나의 프로그램을 실행시키는 단위를 파드(Pod)라고 부른다. 따라서 파드(Pod)는 일반적으로 쿠버네티스에서 하나의 프로그램을 실행시키는 단위라고 기억해두면 이해하기 편하다.

- 쿠버네티스에서 가장 작은 단위
- 일반적으로 하나의 파드가 하나의 컨테이너를 가진다.

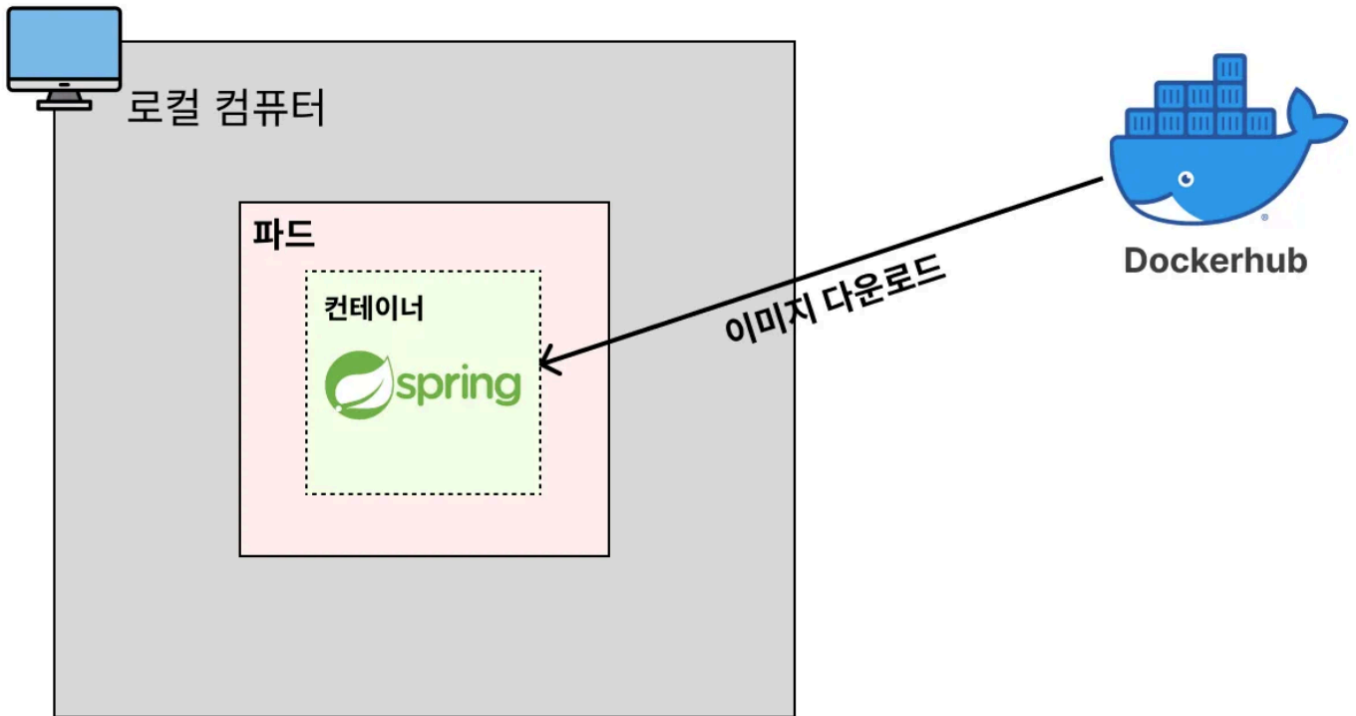
(예외적으로 하나의 파드가 여러 개의 컨테이너를 가지는 경우도 있다.)

** 컨테이너 : 'Docker의 컨테이너'를 뜻한다.




- 2개의 결제 서버가 띄워져있다. = 2개의 결제 서버 파드(Pod)가 띄워져있다.
- 1개의 결제 서버가 죽었다. = 1개의 결제 서버 파드(Pod)가 죽었다.
- 업로드 서버를 하나 띄우자. = 업로드 서버 하나를 파드(Pod)로 띄우자.

- ✓ 쿠버네티스도 도커처럼 이미지를 기반으로 파드(Pod)를 띄워 실행시킨다.



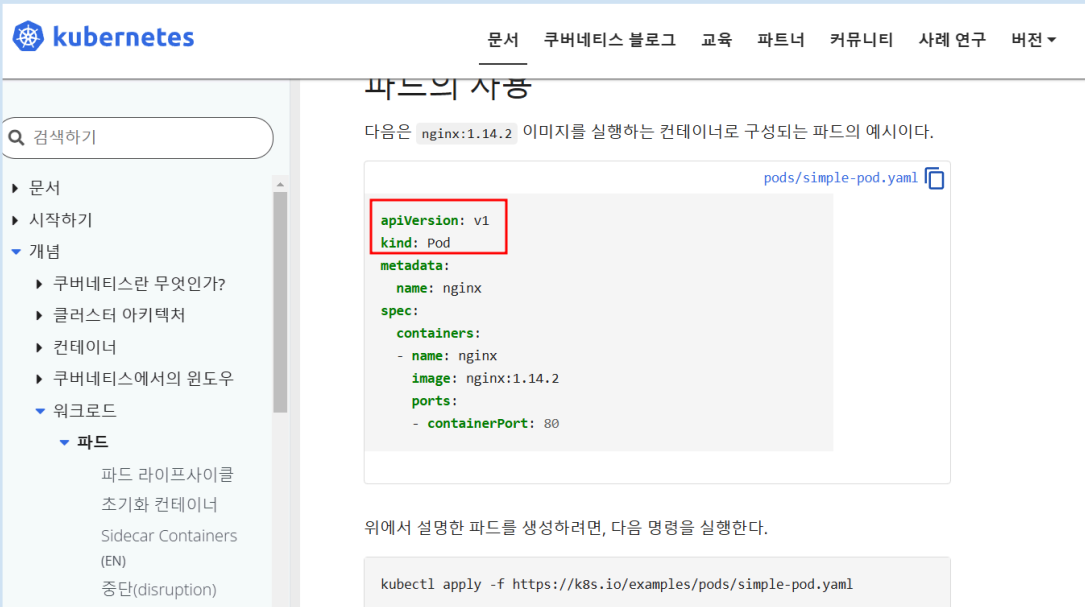
2. [예제] 웹 서버(Nginx)를 파드(Pod)로 띄워보기

✓ 웹 서버(Nginx)를 파드(Pod)로 띄워보기

 파드(Pod)를 생성할 때 CLI를 활용하는 방법이 있고, **yaml** 파일을 활용하는 방법이 있다. 실제 현업에서는 **yaml** 파일을 활용하는 경우가 많다. 따라서 **yaml** 파일을 활용해서 파드(Pod)를 생성해볼 것이다.

1. yaml 파일 생성하기
2. nginx-pod.yaml

```
apiVersion: v1 # Pod를 생성할 때는 v1이라고 기재한다. (공식 문서)
kind: Pod # Pod를 생성한다고 명시
metadata:
  name: nginx-pod # Pod에 이름 붙이는 기능
spec:
  containers:
    - name: nginx-container # 생성할 컨테이너의 이름
      image: nginx # 컨테이너를 생성할 때 사용할 Docker 이미지
      ports:
        - containerPort: 80 # 해당 컨테이너가 어떤 포트를 사용하는 지 명시적으로 표현
```



문서 쿠버네티스 블로그 교육 파트너 커뮤니티 사례 연구 버전 ▾

파드의 사용

다음은 `nginx:1.14.2` 이미지를 실행하는 컨테이너로 구성되는 파드의 예시이다.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - name: nginx
      image: nginx:1.14.2
      ports:
        - containerPort: 80
```

위에서 설명한 파드를 생성하려면, 다음 명령을 실행한다.

```
kubectl apply -f https://k8s.io/examples/pods/simple-pod.yaml
```

- 주의) **YAML** 문법상 들여쓰기를 할 때 **Tab**을 사용하면 안 되고 반드시 띄어쓰기를 활용해야 한다.
- **spec.containers.ports.containerPort** : 실제 작동에는 영향을 미치지 않는다. 단순히 컨테이너가 어떤 포트를 사용하는 지 명시적으로 나타내기 위한 문서화용이다. (Dockerfile의 **EXPOSE**와 비슷한 역할이다.)

3. yaml 파일을 기반으로 파드(Pod) 생성하기

```
$ kubectl apply -f nginx-pod.yaml # yaml 파일에 적혀져있는 리소스(파드)를 생성
```

```
PS C:\Users\AIclass> cd C:\DevData\Kubernetes\kube-practice
PS C:\DevData\Kubernetes\kube-practice> kubectl apply -f nginx-pod.yaml
pod/nginx-pod created
PS C:\DevData\Kubernetes\kube-practice>
```

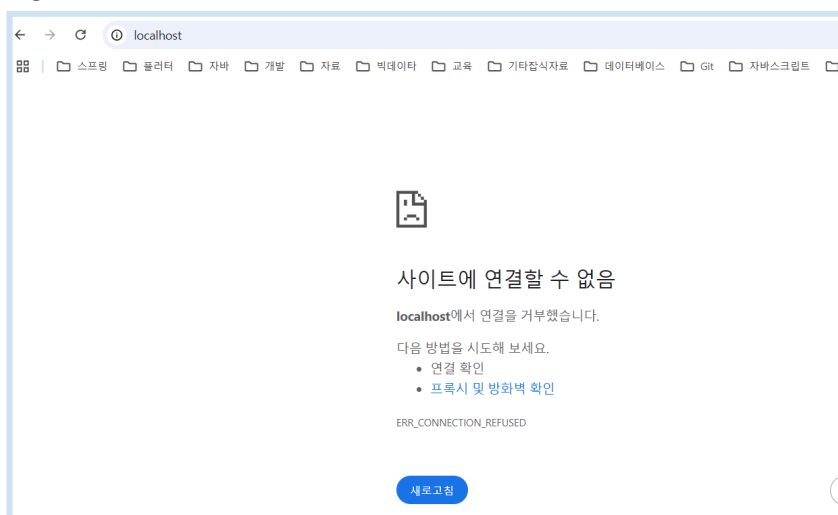
4. 파드(Pod)가 잘 생성 됐는지 확인

```
$ kubectl get pods # 파드(Pod) 조회
```

```
PS C:\DevData\Kubernetes\kube-practice> kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
nginx-pod     1/1     Running   0           97s
PS C:\DevData\Kubernetes\kube-practice>
```

- **NAME** : Pod의 이름
- **READY** : (파드 내 준비 완료된 컨테이너 수)/(파드 내 총 컨테이너 수)
- **STATUS** : 파드의 상태 (**Running** : 정상적으로 실행 중)
- **RESTARTS** : 해당 파드의 컨테이너가 재시작된 횟수
- **AGE** : 파드가 생성되어 실행된 시간

5. Nginx에 정상적으로 접속이 되는 지 확인하기

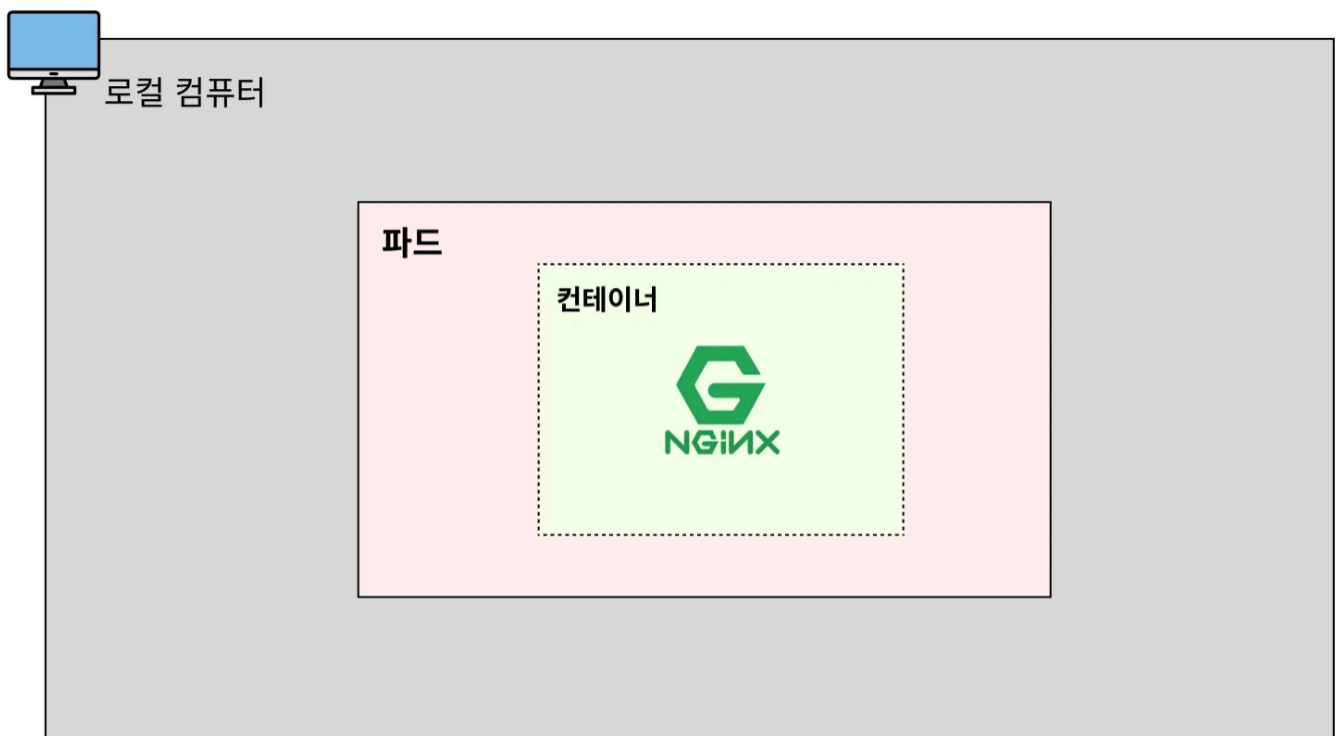
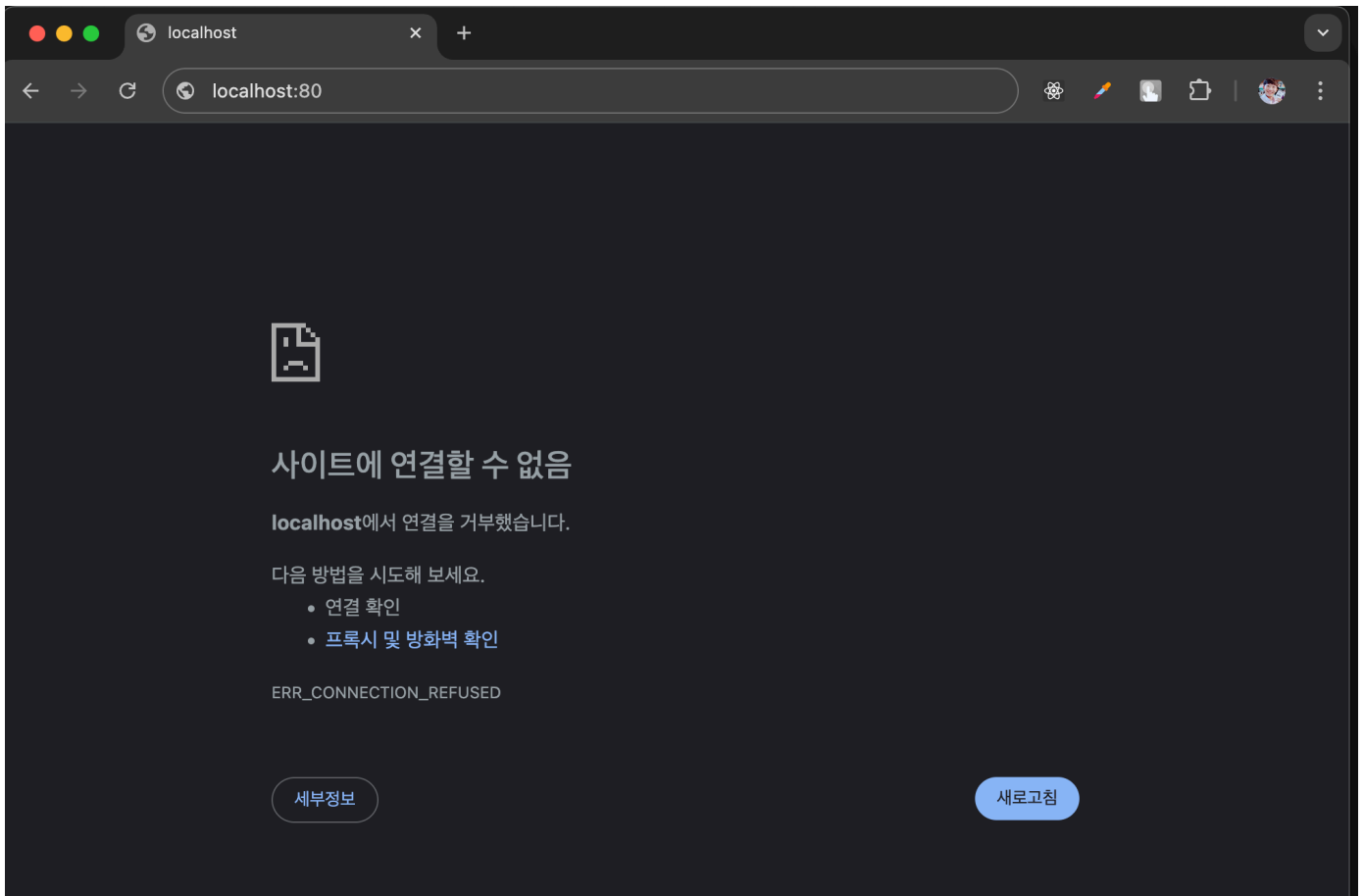


분명 파드(Pod)로 Nginx를 띄웠는데 접속이 안 된다. 왜 그럴까?

★ 쿠버네티스에서는 위에서 작성한 **yaml** 파일을 보고 **매니페스트 파일(Manifest File)**이라고 부른다. 이 **매니페스트 파일**은 쿠버네티스에서 다양한 리소스(파드, 서비스, 볼륨 등)를 생성하고 관리하기 위해 사용하는 파일이라고 기억하자. 이 용어는 자주 사용되니 반드시 기억해두자. (Docker로 치면 Dockerfile과 같은 역할을 하는 파일이다.)

3. 파드(Pod)로 띄운 프로그램에 접속이 안 되는 이유

✓ 파드(Pod)로 띄운 프로그램에 접속이 안 되는 이유



- 도커에 대해서 공부했을 때는 컨테이너 내부와 컨테이너 외부의 네트워크가 서로 독립적으로 분리되어 있다. 하지만 쿠버네티스에서는 파드(Pod) 내부의 네트워크를 컨테이너가 공유해서 같이 사용한다.
- 파드(Pod)의 네트워크는 로컬 컴퓨터의 네트워크와는 독립적으로 분리되어 있다. 이 때문에 파드(Pod)로 띄운 Nginx에 아무리 요청을 보내도 응답이 없던 것이다.

따라서 Nginx가 띄우는 웹 페이지에 접근하려면 2가지 방법이 있다.

1. 파드(Pod) 내부로 들어가서 접근하기
2. 파드(Pod)의 내부 네트워크를 외부에서도 접속할 수 있도록 포트 포워딩(= 포트 연결시키기) 활용하기

하나씩 알아보자.

✅ 파드(Pod) 내부로 들어가서 Nginx로 요청보내기

파드(Pod) 내부로 접속해 Nginx로 요청을 보냈을 때, Nginx가 띄운 웹 페이지를 잘 응답하는 지 확인해보자.

```
# kubectl exec -it [파드명] -- bash
# 도커에서 컨테이너로 접속하는 명령어(docker exec -it [컨테이너 ID]
bash)와 비슷하다.
$ kubectl exec -it nginx-pod -- bash # nginx-pod 내부 환경으로 접속

# ---Pod 내부---
$ curl localhost:80 # Nginx로 요청보내기
```

[curl로 요청 보낸 결과]

```
PS C:\DevData\Kubernetes\kubernetes-practice> kubectl exec -it nginx-pod -- bash
root@nginx-pod:/# curl localhost:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

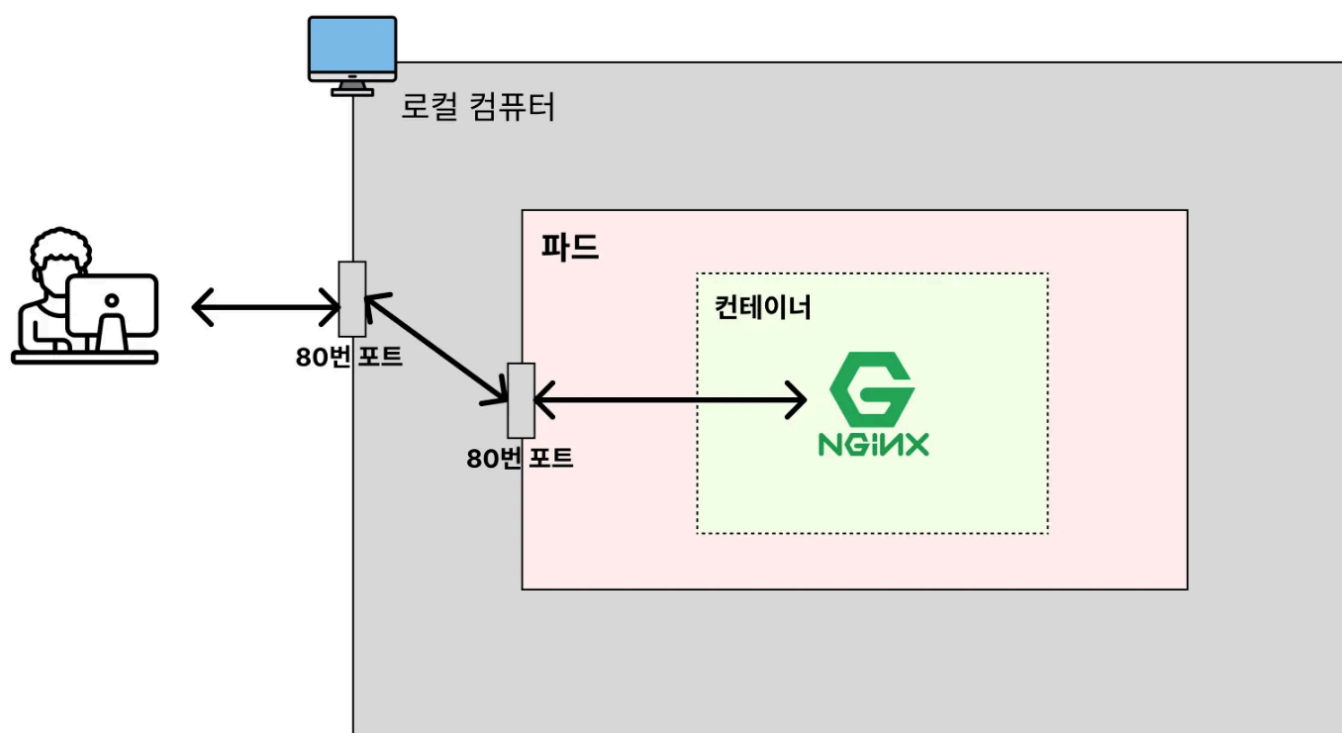
<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
root@nginx-pod:/#
```

쿠버네티스에서는 파드(Pod) 내부의 네트워크를 컨테이너가 공유해서 같이 사용하기 때문에, 파드로 접속해서 Nginx로 요청을 보냈을 때 정상적으로 응답이 날라온 것이다.



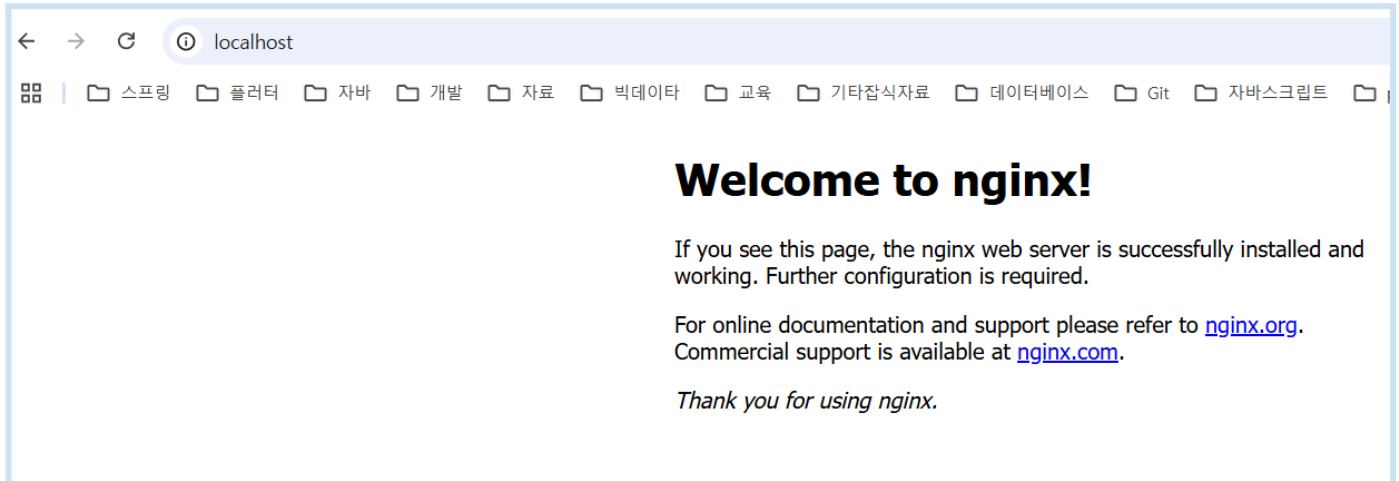
✅ 포트 포워딩을 활용해 Nginx로 요청 보내기



아래 명령어를 통해 포트 포워딩을 활용해 로컬 환경에서도 Nginx로 접속할 수 있게 만들어보자.

```
# kubectl port-forward pod/[파드명] [로컬에서의 포트]/[파드에서의 포트]
$ kubectl port-forward pod/nginx-pod 80:80
```

[크롬으로 접속한 결과]



✅ 파드 삭제하기

```
# kubectl delete pod [파드명]
$ kubectl delete pod nginx-pod # nginx-pod라는 파드 삭제

$ kubectl get pods # 파드가 잘 삭제 됐는지 확인
```

```
PS C:\DevData\Kubernetes\kube-practice> kubectl delete pod nginx-pod
pod "nginx-pod" deleted
PS C:\DevData\Kubernetes\kube-practice> kubectl get pods
No resources found in default namespace.
PS C:\DevData\Kubernetes\kube-practice>
```


4. [예제] 백엔드(Spring Boot) 서버를 파드(Pod)로 띄워보기

✓ 백엔드(Spring Boot) 서버를 파드(Pod)로 띄워보기

1. Spring Boot 프로젝트 셋팅

start.spring.io

의존성 추가 web, devtools

2. 간단한 코드 작성

AppController

@RestController

```
public class AppController {  
    @GetMapping("/")  
    public String home() {  
        return "Hello, World!";  
    }  
}
```

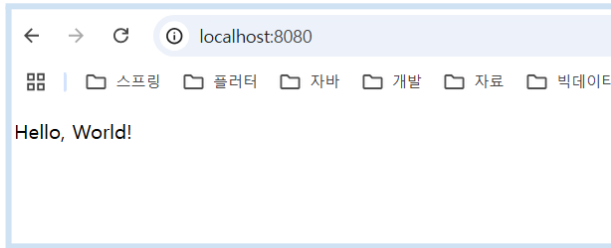
3. 프로젝트 실행시켜보기

The screenshot shows an IDE (likely IntelliJ IDEA) with a Spring Boot project. The Project Explorer on the left shows the project structure, including the 'demo' project with 'src/main/java' and 'src/test/java' directories. The 'AppController.java' file is open in the editor, showing the code for the 'AppController' class. The 'Console' window at the bottom displays the application logs, indicating that the application has started successfully and is running on port 8080.

```
package com.example.demo;  
  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class AppController {  
    @GetMapping("/")  
    public String home() {  
        return "Hello, World!";  
    }  
}
```

Console logs:

```
2024-12-17T09:20:06.733+09:00 INFO 16164 --- [demo] [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]  
2024-12-17T09:20:06.733+09:00 INFO 16164 --- [demo] [ restartedMain] w.s.c.ServletWebServerApplicationContext  
2024-12-17T09:20:07.122+09:00 INFO 16164 --- [demo] [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer  
2024-12-17T09:20:07.137+09:00 INFO 16164 --- [demo] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServ  
2024-12-17T09:20:13.6+09:00 INFO 16164 --- [demo] [nio-8080-exec-1] com.example.demo.DemoApplication  
2024-12-17T09:20:13.6+09:00 INFO 16164 --- [demo] [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]  
2024-12-17T09:20:13.8+09:00 INFO 16164 --- [demo] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet
```



4. Dockerfile 작성하기

Dockerfile

```
FROM openjdk:17-jdk

COPY build/libs/*SNAPSHOT.jar app.jar

ENTRYPOINT ["java", "-jar", "/app.jar"]
```

5. Spring Boot 프로젝트 빌드하기

```
$ ./gradlew clean build
```

6. Dockerfile을 바탕으로 이미지 빌드하기

```
$ docker build -t spring-server .
```

```
PS C:\WDevData\W\Kubernetes\Wdemo> ./gradlew clean build
Starting a Gradle Daemon, 3 incompatible and 1 stopped Daemons could not be reused, use --status for details
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended

BUILD SUCCESSFUL in 48s
8 actionable tasks: 7 executed, 1 up-to-date
PS C:\WDevData\W\Kubernetes\Wdemo> docker build -t spring-server .
[+] Building 4.4s (7/7) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile              0.1s
=> => transferring dockerfile: 142B                             0.0s
=> [internal] load metadata for docker.io/library/openjdk:17-jdk 1.6s
=> [internal] load .dockerignore                                0.0s
=> => transferring context: 2B                                    0.0s
=> [internal] load build context                                0.8s
=> => transferring context: 20.66MB                               0.8s
=> CACHED [1/2] FROM docker.io/library/openjdk:17-jdk@sha256:528707081fdb9562eb819128a9f85a 0.1s
=> => resolve docker.io/library/openjdk:17-jdk@sha256:528707081fdb9562eb819128a9f85ae7fe000 0.1s
=> [2/2] COPY build/libs/*SNAPSHOT.jar app.jar                 0.2s
=> exporting to image                                           1.4s
=> => exporting layers                                           1.2s
=> => exporting manifest sha256:7d423d08b876b4fce66395c2f64b255bf9cbe51df87feb4e1515d5a3bd4 0.0s
=> => exporting config sha256:d88e9dc7b72e995e389e5033d8c52bb48379c65548246bf6c3e6d3cd3f86f 0.0s
=> => exporting attestation manifest sha256:1c2ca15786efd5637469599e8582b954d28ba594d02ce0b 0.0s
=> => exporting manifest list sha256:532da9f3ee5546050a318324e43591cad010dbcdc14c34a03d4ed6 0.0s
=> => naming to docker.io/library/spring-server:latest         0.0s
=> => unpacking to docker.io/library/spring-server:latest      0.2s
PS C:\WDevData\W\Kubernetes\Wdemo>
```

7. 이미지가 잘 생성 됐는지 확인하기

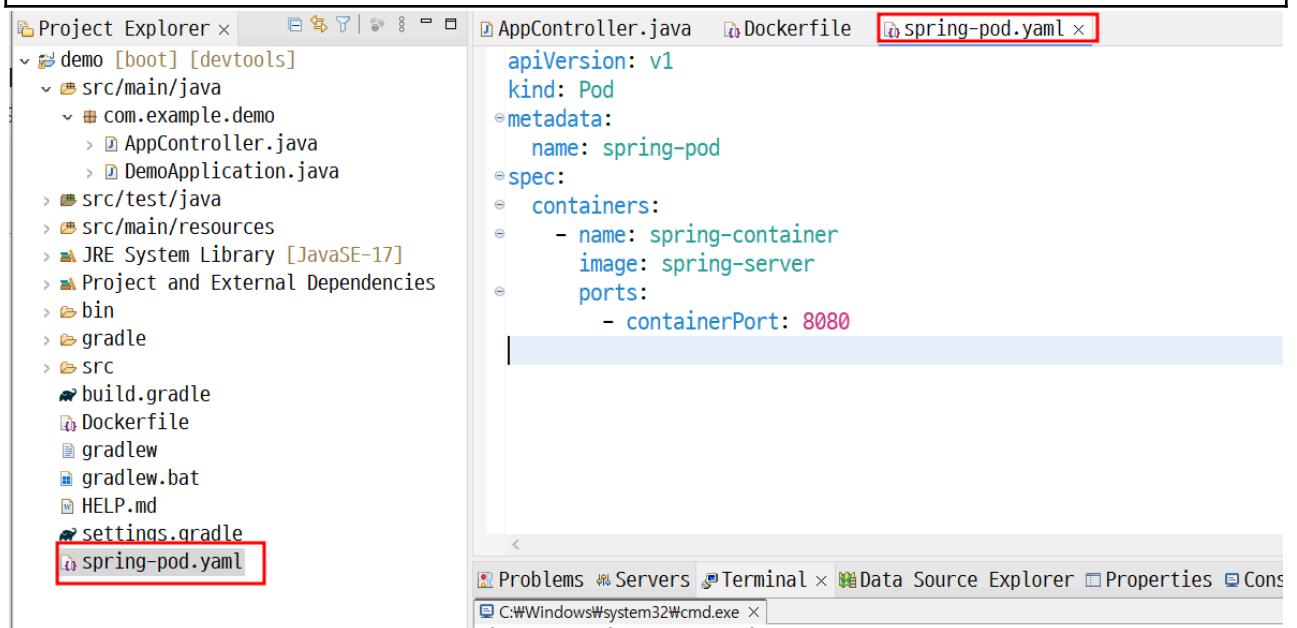
```
$ docker image ls
```

```
PS C:\WDevData\WKubernetes\Wdemo> docker image ls
REPOSITORY              IMAGE ID      CREATED        TAG      SIZE
spring-server           532da9f3ee55 18 minutes ago latest    766MB
```

8. 매니페스트 파일 작성하기

spring-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: spring-pod
spec:
  containers:
    - name: spring-container
      image: spring-server
      ports:
        - containerPort: 8080
```



9. 매니페스트 파일을 기반으로 파드(Pod) 생성하기

```
$ kubectl apply -f spring-pod.yaml
```

10. 파드(Pod)가 잘 생성 됐는지 확인

```
$ kubectl get pods
```

```
PS C:\DevData\Kubernetes\demo> kubectl apply -f spring-pod.yaml
pod/spring-pod created
PS C:\DevData\Kubernetes\demo> kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
spring-pod    0/1     ErrImagePull  0          13s
PS C:\DevData\Kubernetes\demo>
```

STATUS를 보면 **ErrImagePull**라고 떠있다. 이미지를 **Pull** 받아오는 과정 중에 문제가 생긴 것이다. **\$ docker image ls** 명령어를 입력해서 이미지를 확인해봤더니 **spring-server** 이미지가 정상적으로 있다. 그런데 왜 **ErrImagePull**라는 에러가 났을까?

5. 이미지가 없다고 에러가 뜨는 이유 (이미지 풀 정책)

✓ 이미지가 없다고 에러가 뜨는 이유

이전에 Spring Boot 프로젝트를 이미지로 빌드해서 파드로 띄웠다. 하지만 ErrImagePull라는 에러가 발생했다. 이 문제는 **이미지 풀 정책(Image Pull Policy)** 때문에 발생한 것이다. 이미지 풀 정책이 뭔지 알아보자.

```
PS C:\DevData\Kubernetes\demo> kubectl apply -f spring-pod.yaml
pod/spring-pod created
PS C:\DevData\Kubernetes\demo> kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
spring-pod    0/1     ErrImagePull   0          13s
PS C:\DevData\Kubernetes\demo>
```

✓ 이미지 풀 정책 (Image Pull Policy)

이미지 풀 정책(Image Pull Policy)이란 쿠버네티스가 yaml 파일을 읽어들이고 파드를 생성할 때, 이미지를 어떻게 Pull을 받아올 건지에 대한 정책을 의미한다. 어떤 정책들이 있는 지 알아보자.

1. Always

- 로컬에서 이미지를 가져오지 않고, 무조건 레지스트리(= Dockerhub, ECR과 같은 원격 이미지 저장소)에서 가져온다.

2. IfNotPresent

- 로컬에서 이미지를 먼저 가져온다. 만약 로컬에 이미지가 없는 경우에만 레지스트리에서 가져온다.

3. Never

- 로컬에서만 이미지를 가져온다.

✓ 매니페스트 파일에 이미지 풀 정책 설정하는 방법

```
apiVersion: v1
kind: Pod
metadata:
  name: spring-pod
spec:
  containers:
    - name: spring-container
      image: spring-server
      ports:
        - containerPort: 8080
      imagePullPolicy: Always
```

✓ 기존 매니페스트 파일 다시 살펴보기

spring-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: spring-pod
spec:
  containers:
    - name: spring-container
      image: spring-server
      ports:
        - containerPort: 8080
```

위의 매니페스트 파일에서는 이미지 풀 정책을 따로 설정하지 않았다. 이럴 때는 아래와 같이 작동한다.

- 이미지의 태그가 **latest**이거나 명시되지 않은 경우 : **imagePullPolicy**는 **Always**로 설정됨
- 이미지의 태그가 **latest**가 아닌 경우 : **imagePullPolicy**는 **IfNotPresent**로 설정됨

따라서 기존 매니페스트 파일은 **imagePullPolicy**가 **Always**로 작동했던 것이다. 즉, 로컬에서 이미지를 가져오지 않고 레지스트리에서 가져오려고 했다. 하지만 **spring-server**라는 이미지는 **Dockerhub**에 올린 적이 없기 때문에 이미지를 못 받아온 것이다. 따라서 아래와 같이 에러가 났다.

```
PS C:\DevData\Kubernetes\demo> kubectl apply -f spring-pod.yaml
pod/spring-pod created
PS C:\DevData\Kubernetes\demo> kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
spring-pod    0/1     ErrImagePull   0          13s
PS C:\DevData\Kubernetes\demo>
```

그러면 어떻게 해야 로컬에 있는 이미지를 가져올 수 있을까?

spring-pod.yaml

```
apiVersion: v1
kind: Pod
```

```

metadata:
  name: spring-pod
spec:
  containers:
    - name: spring-container
      image: spring-server
      ports:
        - containerPort: 8080
      imagePullPolicy: IfNotPresent

```

위와 같이 정책을 설정해주어야 로컬에서 이미지를 가져오게 된다.

```

apiVersion: v1
kind: Pod
metadata:
  name: spring-pod
spec:
  containers:
    - name: spring-container
      image: spring-server
      ports:
        - containerPort: 8080
      imagePullPolicy: IfNotPresent

```

기존 파드를 삭제하고 다시 생성해보자.

```

$ kubectl delete pod spring-pod
$ kubectl apply -f spring-pod.yaml
$ kubectl get pods

```

```

PS C:\DevData\Kubernetes\demo> kubectl apply -f spring-pod.yaml
pod/spring-pod created
PS C:\DevData\Kubernetes\demo> kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
spring-pod    1/1     Running   0           3m32s
PS C:\DevData\Kubernetes\demo>

```

Spring Boot 서버에 요청을 보내서 잘 응답하는 지도 알아보자.

방법 1. 파드 내부로 들어가서 요청보내기

```

$ kubectl exec -it spring-pod -- bash
$ curl localhost:8080

```

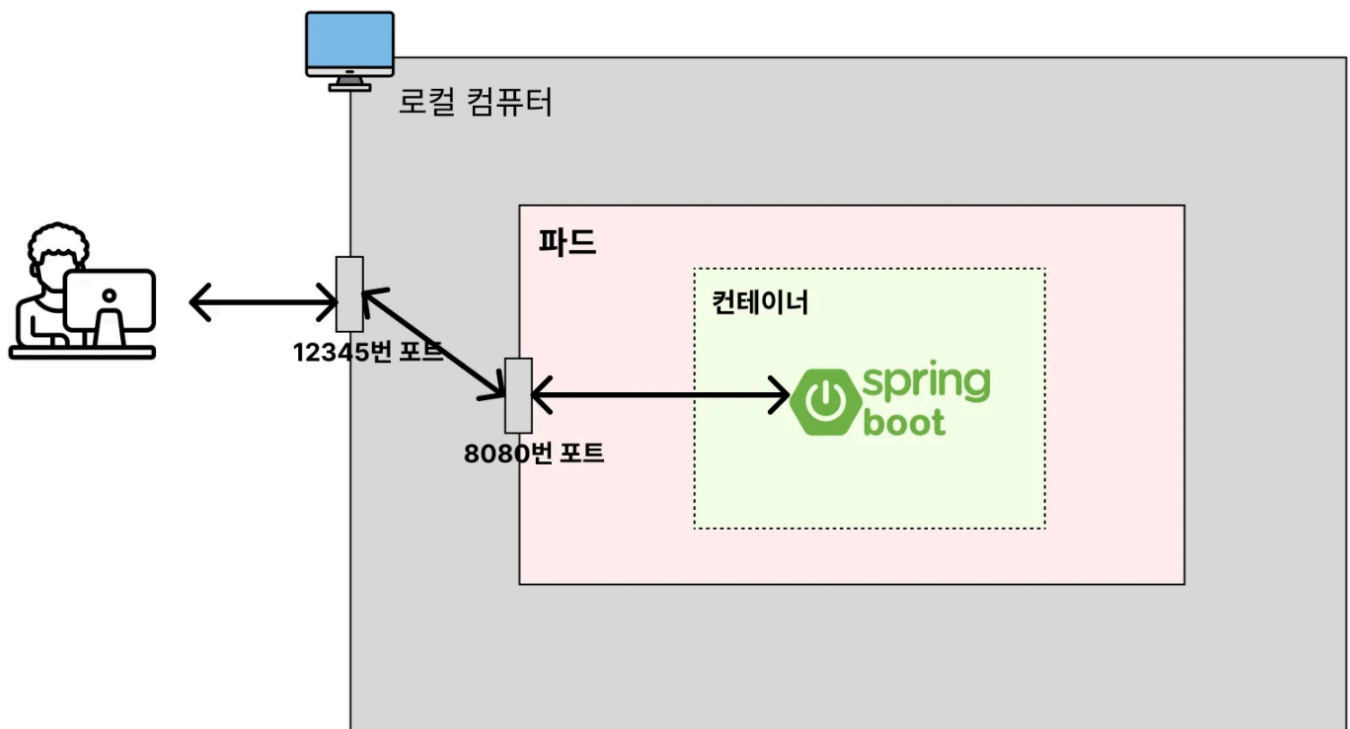
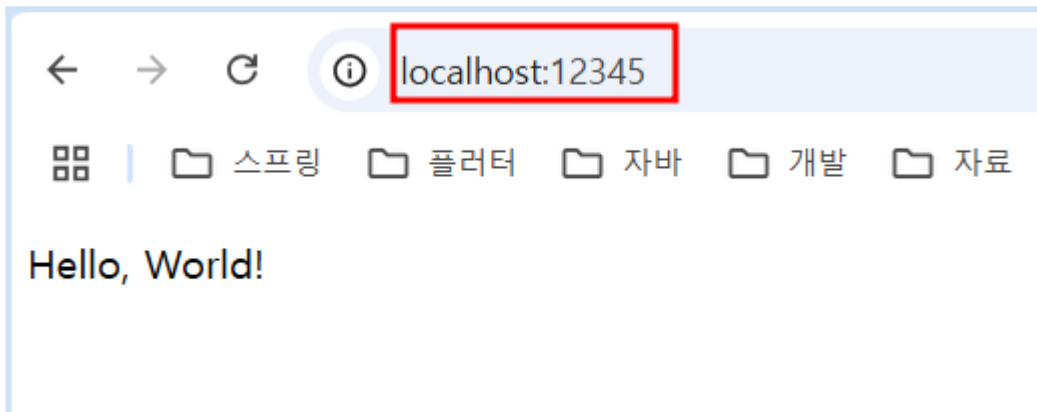
```

PS C:\DevData\Kubernetes\demo> kubectl exec -it spring-pod -- bash
bash-4.4# curl localhost:8080
Hello, World!bash-4.4#

```

방법 2. 포트 포워딩 활용하기

```
# 포트  
$ kubectl port-forward pod/spring-pod 12345:8080
```



✓ 파드 삭제하기

```
$ kubectl delete pod spring-pod
```

```
PS C:\Users\AIclass> kubectl get pods  
No resources found in default namespace.  
PS C:\Users\AIclass>
```



6. [예제] 백엔드(Nest.js) 서버를 파드(Pod)로 띄워보기

✓ 백엔드(Nest.js) 서버를 파드(Pod)로 띄워보기

1. Nest.js 프로젝트 만들기

```
# Nest CLI 설치
$ npm i -g @nestjs/cli

# nest new {프로젝트명}
$ nest new nest-server
```

 선택 C:\Windows\system32\cmd.exe

```
C:\Users\Alclass>cd C:\DevData\Kubernetes\nest
C:\DevData\Kubernetes\nest>npm i -g @nestjs/cli
added 4 packages, and changed 253 packages in 33s
53 packages are looking for funding
  run `npm fund` for details
C:\DevData\Kubernetes\nest>nest new nest-server
$ We will scaffold your app in a few seconds..

? Which package manager would you ❤️ to use? npm
CREATE nest-server/.eslintrc.js (688 bytes)
CREATE nest-server/.prettierrc (54 bytes)
CREATE nest-server/nest-cli.json (179 bytes)
CREATE nest-server/package.json (2016 bytes)
CREATE nest-server/README.md (5304 bytes)
CREATE nest-server/tsconfig.build.json (101 bytes)
CREATE nest-server/tsconfig.json (567 bytes)
CREATE nest-server/src/app.controller.ts (286 bytes)
CREATE nest-server/src/app.module.ts (259 bytes)
CREATE nest-server/src/app.service.ts (150 bytes)
CREATE nest-server/src/main.ts (236 bytes)
CREATE nest-server/src/app.controller.spec.ts (639 bytes)
CREATE nest-server/test/jest-e2e.json (192 bytes)
CREATE nest-server/test/app.e2e-spec.ts (654 bytes)

√ Installation in progress... 📦

💎 Successfully created project nest-server
💎 Get started with the following commands:

$ cd nest-server
$ npm run start

Failed to execute command: git init
Git repository has not been initialized

Thanks for installing Nest 💎
Please consider donating to our open collective
to help us maintain this package.

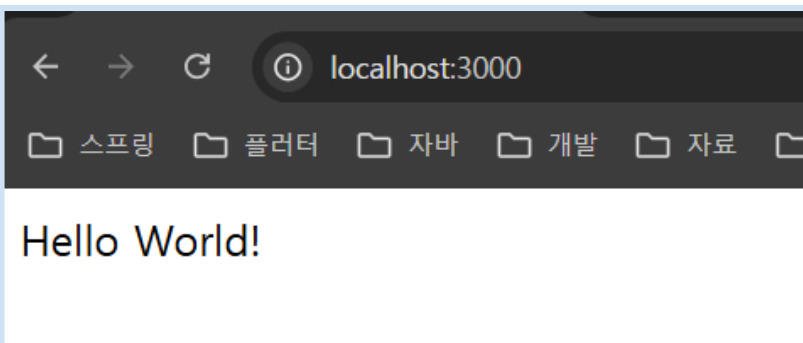
💎 Donate: https://opencollective.com/nest
```

2. 프로젝트 실행시켜보기

```
$ npm i
$ npm run start
```

```
C:\DevData\Kubernetes\nest>cd C:\DevData\Kubernetes\nest\nest-server
C:\DevData\Kubernetes\nest\nest-server>npm i
up to date, audited 698 packages in 2s
116 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
C:\DevData\Kubernetes\nest\nest-server>npm run start
> nest-server@0.0.1 start
> nest start

[Nest] 12112 - 2024. 12. 17. 오후 2:00:41 LOG [NestFactory] Starting Nest application...
[Nest] 12112 - 2024. 12. 17. 오후 2:00:41 LOG [InstanceLoader] AppModule dependencies initialized +21ms
[Nest] 12112 - 2024. 12. 17. 오후 2:00:41 LOG [RoutesResolver] AppController {/}: +13ms
[Nest] 12112 - 2024. 12. 17. 오후 2:00:41 LOG [RouterExplorer] Mapped {/, GET} route +3ms
[Nest] 12112 - 2024. 12. 17. 오후 2:00:41 LOG [NestApplication] Nest application successfully started +4ms
```



3. Dockerfile 작성하기

Dockerfile

```
FROM node

WORKDIR /app

COPY . .

RUN npm install

RUN npm run build

EXPOSE 3000

ENTRYPOINT [ "node", "dist/main.js" ]
```

4. .dockerignore 작성하기

.dockerignore

```
node_modules
```

dist	2024-12-17 오후 2:00	파일 폴더	
node_modules	2024-12-17 오후 1:59	파일 폴더	
src	2024-12-17 오후 1:55	파일 폴더	
test	2024-12-17 오후 1:55	파일 폴더	
.dockerignore	2024-12-17 오후 2:04	DOCKERIGNORE ...	1KB
.eslintrc.js	2024-12-17 오후 1:55	JavaScript 파일	1KB
.gitignore	2024-12-17 오후 1:56	Git Ignore 원본 파...	1KB
.prettierrc	2024-12-17 오후 1:55	PRETTIERRC 파일	1KB
Dockerfile	2024-12-17 오후 2:03	파일	1KB
nest-cli.json	2024-12-17 오후 1:55	Adobe After Effec...	1KB
package.json	2024-12-17 오후 1:55	Adobe After Effec...	2KB
package-lock.json	2024-12-17 오후 1:59	Adobe After Effec...	335KB
README.md	2024-12-17 오후 1:55	Markdown 원본 ...	6KB
tsconfig.build.json	2024-12-17 오후 1:55	Adobe After Effec...	1KB
tsconfig.json	2024-12-17 오후 1:55	Adobe After Effec...	1KB

5. Dockerfile을 바탕으로 이미지 빌드하기

```
$ docker build -t nest-server .
```

```
PS C:\DevData\Kubernetes\nest\nest-server> docker build -t nest-server .
[+] Building 38.4s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 172B
=> [internal] load metadata for docker.io/library/node:latest
=> [internal] load .dockerignore
=> => transferring context: 52B
=> [1/5] FROM docker.io/library/node:latest@sha256:0b50ca11d81b5ed2622ff8770f040cdd4bd93a2561208c01c0c5db98bd65d551
=> => resolve docker.io/library/node:latest@sha256:0b50ca11d81b5ed2622ff8770f040cdd4bd93a2561208c01c0c5db98bd65d551
=> [internal] load build context
=> => transferring context: 485.34kB
=> CACHED [2/5] WORKDIR /app
=> [3/5] COPY .
=> [4/5] RUN npm install
=> [5/5] RUN npm run build
=> exporting to image
=> exporting layers
=> => exporting manifest sha256:ebf92199737281f80710906f2d1cd857d3b68429878a3c496ae59f1f1977d205
=> => exporting config sha256:12d042c3eb55ab6894c0556962519c2adebc8665fd3373f6dce928045667ef49
=> => exporting attestation manifest sha256:9e47738ecf0a4fb92665122a942749cbf0cd11240d79885aa302aa1a753039c8
=> => exporting manifest list sha256:1eadd27ca850cb0c5af16fedcd11efcd52abb7caacbb3c3da33c866270652
=> => naming to docker.io/library/nest-server:latest
=> => unpacking to docker.io/library/nest-server:latest
PS C:\DevData\Kubernetes\nest\nest-server>
```

6. 이미지가 잘 생성 됐는지 확인하기

```
$ docker image ls
```

```
PS C:\DevData\Kubernetes\nest\nest-server> docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
nest-server         latest             1eadd27ca85        About a minute ago
1.87GB
spring-server       latest             532da9f3ee55       5 hours ago
766MB
docker/desktop-kubernetes   kubernetes-v1.30.5-cni-v1.4.0-critools-v1.29.0-cri-dockerd-v0.3.11-1-debian   7a7b02256c8d       2 months ago
625MB
registry.k8s.io/kube-apiserver   v1.30.5           7746ea55ad74       3 months ago
153MB
registry.k8s.io/kube-scheduler   v1.30.5           62c91756a3c9       3 months ago
84.6MB
registry.k8s.io/kube-controller-manager   v1.30.5           bbd15d267294       3 months ago
146MB
```

7. 매니페스트 파일 작성하기

nest-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: nest-pod
spec:
  containers:
    - name: nest-container
      image: nest-server
      imagePullPolicy: IfNotPresent
```

8. 매니페스트 파일을 기반으로 파드(Pod) 생성하기

```
$ kubectl apply -f nest-pod.yaml
```

9. 파드(Pod)가 잘 생성 됐는지 확인

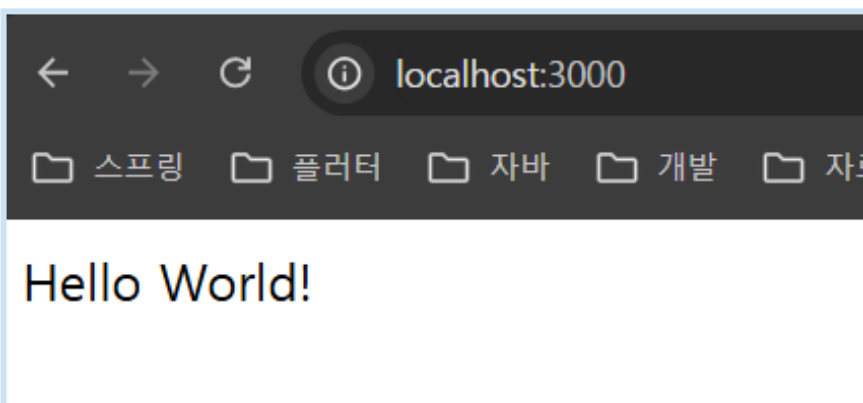
```
$ kubectl get pods
```

```
PS C:\DevData\Kubernetes\nest\nest-server> kubectl apply -f nest-pod.yaml
pod/nest-pod created
PS C:\DevData\Kubernetes\nest\nest-server> kubectl get pods
NAME        READY   STATUS    RESTARTS   AGE
nest-pod    1/1     Running   0           8s
PS C:\DevData\Kubernetes\nest\nest-server> █
```

10. 포트 포워딩으로 Nest.js 서버가 실행 됐는지 확인

```
$ kubectl port-forward nest-pod 3000:3000
```

```
PS C:\DevData\Kubernetes\nest\nest-server> kubectl port-forward nest-pod 3000:3000
Forwarding from 127.0.0.1:3000 -> 3000
Forwarding from [::1]:3000 -> 3000
Handling connection for 3000
Handling connection for 3000
█
```



11. 파드 삭제하기

```
$ kubectl delete pod nest-pod
```

```
PS C:\DevData\Kubernetes\nest\nest-server> kubectl delete pod nest-pod  
pod "nest-pod" deleted  
PS C:\DevData\Kubernetes\nest\nest-server> kubectl get pods  
No resources found in default namespace.
```

7. [예제] 프론트엔드(HTML, CSS, Nginx) 서버를 파드(Pod)로 띄워보기

✓ 프론트엔드(HTML, CSS, Nginx) 서버를 파드(Pod)로 띄워보기

1. HTML, CSS 파일 만들기

index.html

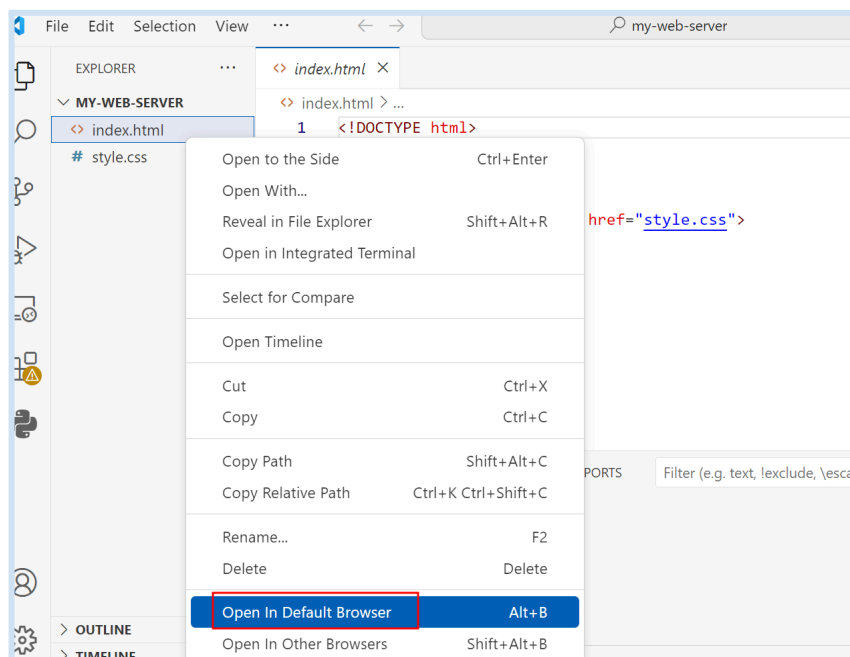
```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>My Web Page</h1>
</body>
</html>
```

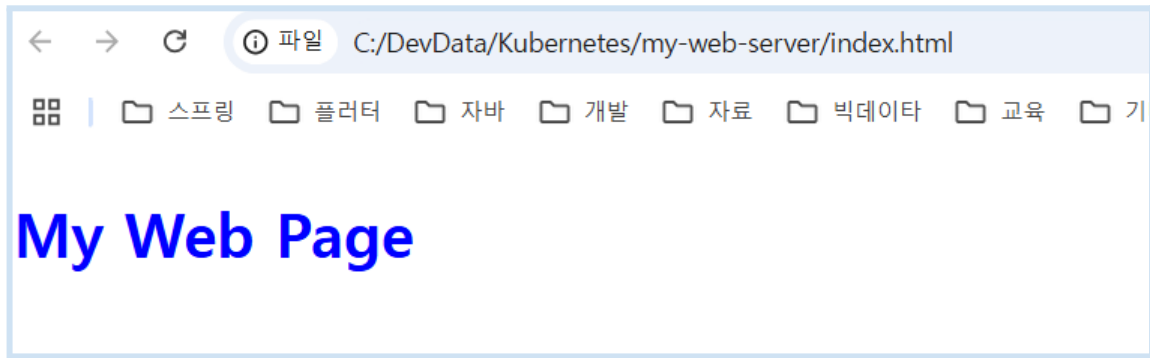
- 주의) Nginx의 기본 설정에 의하면 메인 페이지(첫 페이지)의 파일명을 **index.html**이라고 지어야 한다.

style.css

```
* {
  color: blue;
}
```

2. 실행시켜보기





3. Dockerfile 작성하기

[Official build of Nginx.](#)

Dockerfile

```
FROM nginx
COPY ./ /usr/share/nginx/html
```

4. Dockerfile을 바탕으로 이미지 빌드하기

```
$ docker build -t my-web-server .
```

```
PS C:\DevData\Kubernetes\my-web-server> docker build -t my-web-server .
[+] Building 3.7s (7/7) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile              0.1s
=> => transferring dockerfile: 79B                               0.0s
=> [internal] load metadata for docker.io/library/nginx:latest  2.0s
=> [internal] load .dockerignore                                0.1s
=> => transferring context: 2B                                    0.0s
=> [internal] load build context                                0.2s
=> => transferring context: 350B                                  0.0s
=> CACHED [1/2] FROM docker.io/library/nginx:latest@sha256:fb197595ebe76b9c0c14ab68159fd3c08bd067ec623 0.1s
=> => resolve docker.io/library/nginx:latest@sha256:fb197595ebe76b9c0c14ab68159fd3c08bd067ec6230058354 0.1s
=> [2/2] COPY ./ /usr/share/nginx/html                          0.2s
=> exporting to image                                           0.7s
=> => exporting layers                                           0.3s
=> => exporting manifest sha256:5bc91f5d6b3e1b962be00a171faab0ce79b6b25b867bd11b7f99f7e519b5298a 0.0s
=> => exporting config sha256:c549acb412ce752bf5cf33c0e3608e0a56327393403b2ec530d265aebdcca8a 0.0s
=> => exporting attestation manifest sha256:21d68dea24e785927b10b39ea34b996c93e91f55e4c8b2586f5bf05293 0.1s
=> => exporting manifest list sha256:86910fd7bdb2cba1fab98ae7f517eed50bff96b8fefd6949a78e466b1364ba48 0.0s
=> => naming to docker.io/library/my-web-server:latest          0.0s
=> => unpacking to docker.io/library/my-web-server:latest       0.1s
PS C:\DevData\Kubernetes\my-web-server>
```

5. 이미지가 잘 생성 됐는지 확인하기

```
$ docker image ls
```

```
PS C:\DevData\Kubernetes\my-web-server> docker image ls
REPOSITORY          IMAGE ID      CREATED        TAG      SIZE
my-web-server       86910fd7bdb2  50 seconds ago latest    278MB
nest-server         1eadddd27ca85 4 hours ago   latest    1.87GB
spring-server       532da9f3ee55 8 hours ago   latest    766MB
```

6. 매니페스트 파일 작성하기

web-server-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: web-server-pod
spec:
  containers:
    - name: web-server-container
      image: my-web-server
      ports:
        - containerPort: 80
      imagePullPolicy: IfNotPresent
```

7. 매니페스트 파일을 기반으로 파드(Pod) 생성하기

```
$ kubectl apply -f web-server-pod.yaml
```

8. 파드(Pod)가 잘 생성 됐는지 확인

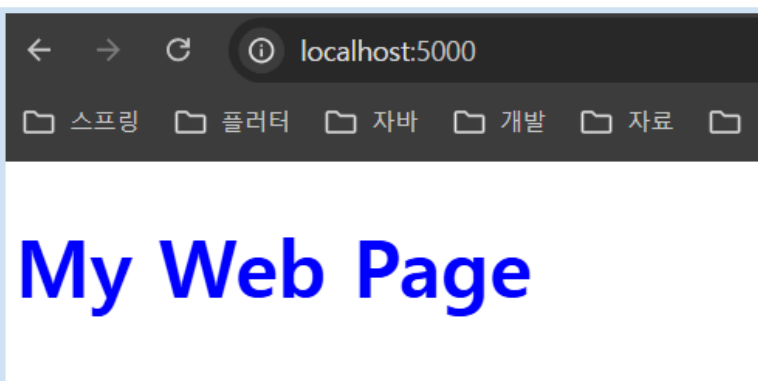
```
$ kubectl get pods
```

```
PS C:\DevData\Kubernetes\my-web-server> kubectl apply -f web-server-pod.yaml
pod/web-server-pod created
PS C:\DevData\Kubernetes\my-web-server> kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
web-server-pod      1/1     Running   0          9s
```

9. 포트 포워딩으로 웹 서버가 실행 됐는지 확인

```
$ kubectl port-forward web-server-pod 5000:80
```

```
PS C:\DevData\Kubernetes\my-web-server> kubectl port-forward web-server-pod 5000:80
Forwarding from 127.0.0.1:5000 -> 80
Forwarding from [::1]:5000 -> 80
```



10. 파드 삭제하기

```
$ kubectl delete pod web-server-pod
```

```
PS C:\DevData\Kubernetes\my-web-server> kubectl delete pod web-server-pod
pod "web-server-pod" deleted
PS C:\DevData\Kubernetes\my-web-server> kubectl get pods
No resources found in default namespace.
PS C:\DevData\Kubernetes\my-web-server>
```

8. [예제] 프론트엔드(Next.js) 서버를 파드(Pod)로 띄워보기

✓ 프론트엔드(Next.js) 서버를 파드(Pod)로 띄워보기

1. Next.js 프로젝트 만들기

```
$ npx create-next-app
```

```
C:\DevData\Kubernetes\my-app>npx create-next-app
Need to install the following packages:
create-next-app@15.1.0
Ok to proceed? (y) y

✔ What is your project named? ... my-app
✔ Would you like to use TypeScript? ... No / Yes
✔ Would you like to use ESLint? ... No / Yes
✔ Would you like to use Tailwind CSS? ... No / Yes
✔ Would you like your code inside a `src/` directory? ... No / Yes
✔ Would you like to use App Router? (recommended) ... No / Yes
✔ Would you like to use Turbopack for `next dev`? ... No / Yes
✔ Would you like to customize the import alias (`@/*` by default)? ... No / Yes
Creating a new Next.js app in C:\DevData\Kubernetes\my-app\my-app.

Using npm.

Initializing project with template: app-tw
```

2. 프로젝트 실행시켜보기

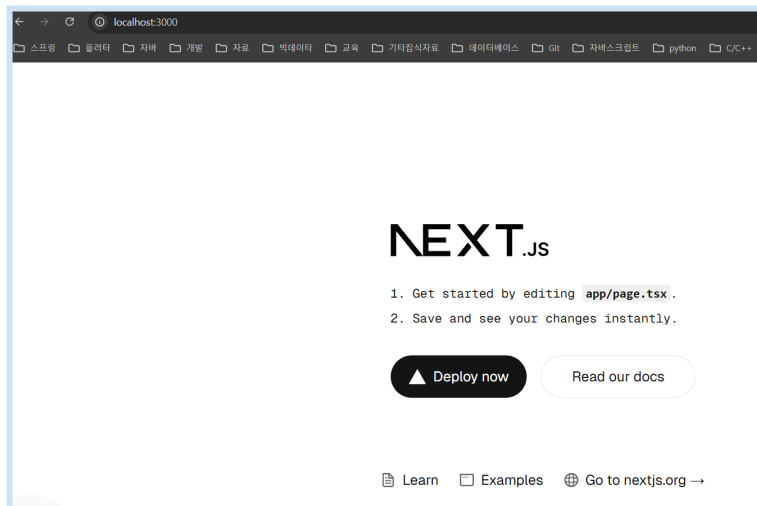
```
$ npm run dev
```

```
C:\DevData\Kubernetes\my-app>npm run dev

> my-app@0.1.0 dev
> next dev --turbo

  ▲ Next.js 15.1.0 (Turbopack)
  - Local:      http://localhost:3000
  - Network:    http://172.21.64.1:3000

  ✓ Starting...
  ✓ Ready in 1624ms
  ○ Compiling / ...
  ✓ Compiled / in 3.3s
  GET / 200 in 3825ms
  ✓ Compiled /favicon.ico in 397ms
  GET /favicon.ico?favicon.45db1c09.ico 200 in 569ms
일괄 작업을 끝내시겠습니까 (Y/N)?
y
```



3. Dockerfile 작성하기

Dockerfile

```
FROM node:20-alpine

WORKDIR /app

COPY . .

RUN npm install

RUN npm run build

EXPOSE 3000

ENTRYPOINT [ "npm", "run", "start" ]
```

4. .dockerignore 작성하기

.dockerignore

```
node_modules
```

5. Dockerfile을 바탕으로 이미지 빌드하기

```
$ docker build -t next-server .
```

6. 이미지가 잘 생성 됐는지 확인하기

```
$ docker image ls
```

```
PS C:\DevData\Kubernetes\my-app> docker image ls
REPOSITORY          IMAGE ID      CREATED        SIZE
next-server         d4bfcc8c3dd6 About a minute ago 1.36GB
my-web-server       86910fd7bdb2 25 minutes ago 278MB
nest-server         1eadd27ca85 4 hours ago    1.87GB
```

7. 매니페스트 파일 작성하기

next-pod.yaml

```
apiVersion: v1
```

```
kind: Pod
metadata:
  name: next-pod
spec:
  containers:
    - name: next-container
      image: next-server
      imagePullPolicy: IfNotPresent
      ports:
        - containerPort: 3000
```

8. 매니페스트 파일을 기반으로 파드(Pod) 생성하기

```
$ kubectl apply -f next-pod.yaml
```

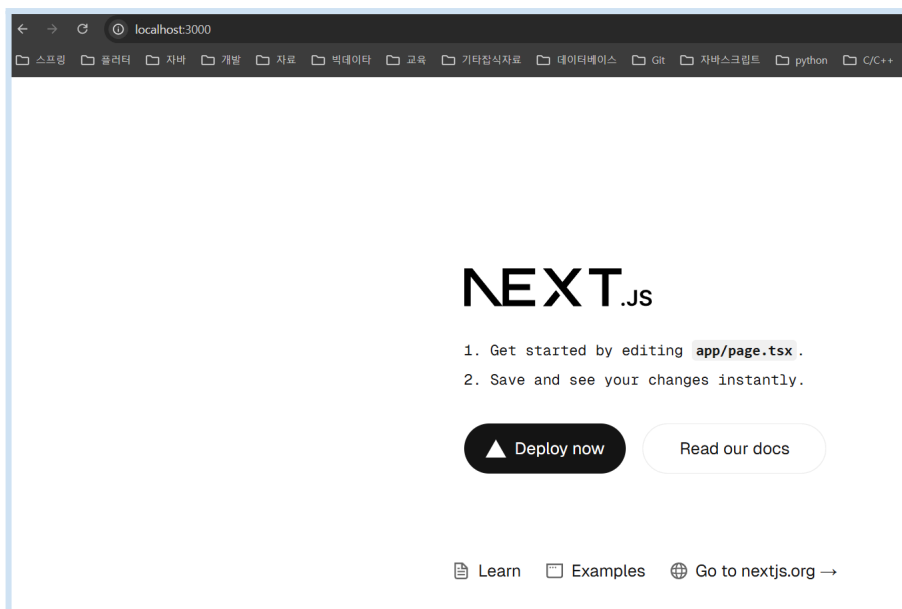
9. 파드(Pod)가 잘 생성 됐는지 확인

```
$ kubectl get pods
```

```
PS C:\DevData\Kubernetes\my-app> kubectl apply -f next-pod.yaml
pod/next-pod created
PS C:\DevData\Kubernetes\my-app> kubectl get pods
NAME       READY   STATUS    RESTARTS   AGE
next-pod   1/1     Running   0           6s
```

10. 포트 포워딩으로 Nest.js 서버가 실행 됐는지 확인

```
$ kubectl port-forward next-pod 3000:3000
```



11. 파드 삭제하기

```
$ kubectl delete pod next-pod
```

```
PS C:\DevData\Kubernetes\my-app> kubectl delete pod next-pod
pod "next-pod" deleted
PS C:\DevData\Kubernetes\my-app> kubectl get pods
No resources found in default namespace.
PS C:\DevData\Kubernetes\my-app>
```

9. [예제] 백엔드(Spring Boot) 서버 3개 띄워보기

✅ 백엔드(Spring Boot) 서버 3개 띄워보기

👤 실제 서비스를 운영하다보면 트래픽이 증가해서 서버가 버벅거리는 경우가 생긴다. 이 때는 서버를 수평적 확장(서버의 개수를 늘리는 방식)을 통해 해결한다. 이런 상황을 가정해 백엔드 서버인 Spring Boot 서버를 3대로 늘려보자. 위의 예제 demo사용하기

1. Spring Boot 프로젝트 셋팅

start.spring.io

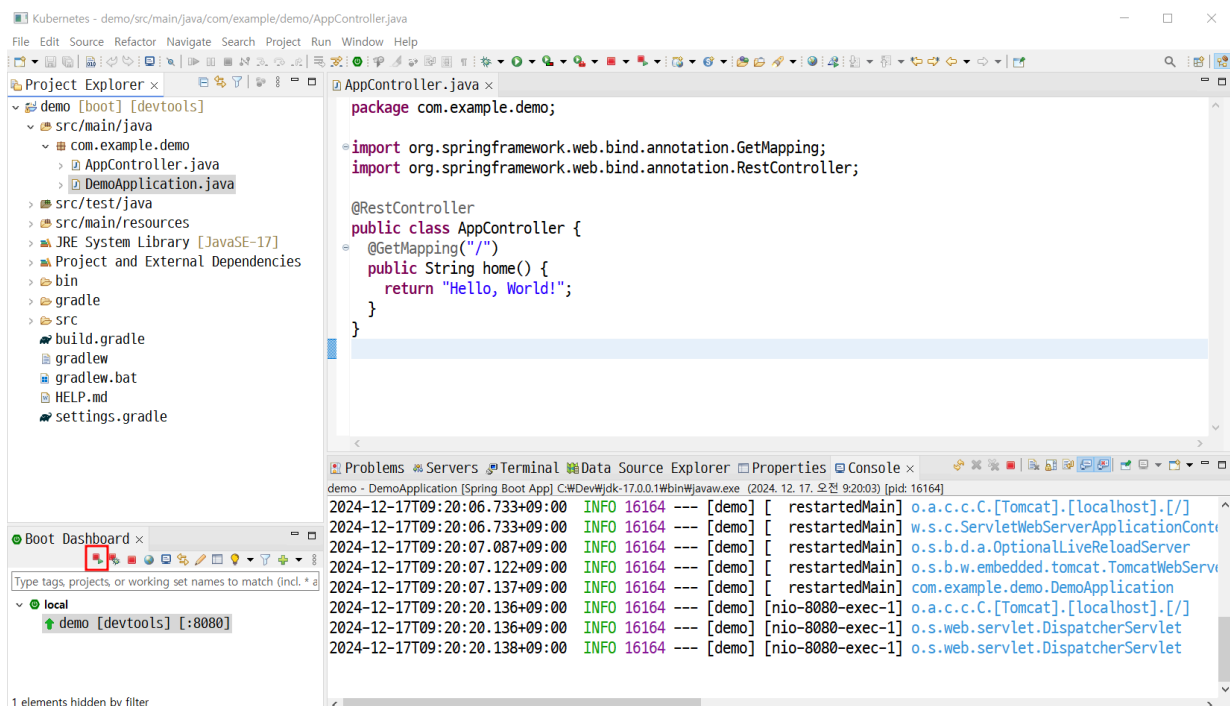
의존성 추가 web, devtools

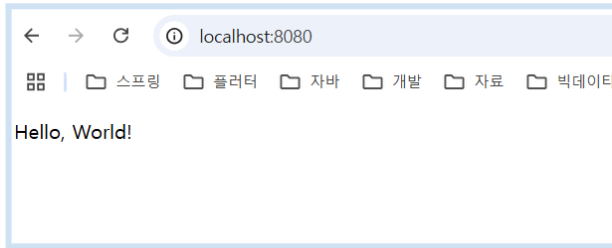
2. 간단한 코드 작성

AppController

```
@RestController
public class AppController {
    @GetMapping("/")
    public String home() {
        System.out.println("Hello, World!"); // 추후 디버깅용
        return "Hello, World!";
    }
}
```

3. 프로젝트 실행시켜보기





4. Dockerfile 작성하기

Dockerfile

```
FROM openjdk:17-jdk

COPY build/libs/*SNAPSHOT.jar app.jar

ENTRYPOINT ["java", "-jar", "/app.jar"]
```

5. Spring Boot 프로젝트 빌드하기

```
$ ./gradlew clean build
```

6. Dockerfile을 바탕으로 이미지 빌드하기

```
$ docker build -t spring-server .
```

7. 이미지가 잘 생성 됐는지 확인하기

```
$ docker image ls
```

REPOSITORY	IMAGE ID
spring-server	3cb9024572da
next-server	d4bfc8c3ddb
my-web-server	86910fd7bdb2
nest-server	1eadd27ca85
docker/desktop-kubernetes	7a7b02256c8d

8. 매니페스트 파일 작성하기

spring-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
```

```
name: spring-pod-1
spec:
  containers:
    - name: spring-container
      image: spring-server
      imagePullPolicy: IfNotPresent
      ports:
        - containerPort: 8080
```

```
apiVersion: v1
kind: Pod
metadata:
  name: spring-pod-2
spec:
  containers:
    - name: spring-container
      image: spring-server
      imagePullPolicy: IfNotPresent
      ports:
        - containerPort: 8080
```

```
apiVersion: v1
kind: Pod
metadata:
  name: spring-pod-3
spec:
  containers:
    - name: spring-container
      image: spring-server
      imagePullPolicy: IfNotPresent
      ports:
        - containerPort: 8080
```

9. 매니페스트 파일을 기반으로 파드(Pod) 생성하기

```
$ kubectl apply -f spring-pod.yaml
```

10. 파드(Pod)가 잘 생성 됐는지 확인


```
$ kubectl get pods
```

```
PS C:\DevData\Kubernetes\demo> kubectl apply -f spring-pod.yaml
pod/spring-pod-1 created
pod/spring-pod-2 created
pod/spring-pod-3 created
PS C:\DevData\Kubernetes\demo> kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
spring-pod-1  1/1     Running   0           7s
spring-pod-2  1/1     Running   0           7s
spring-pod-3  1/1     Running   0           7s
PS C:\DevData\Kubernetes\demo>
```

3개의 **Spring Boot** 서버를 띄우는 데 성공했다. 서버가 3개여서 망정이지 만약 100개의 서버를 띄워야 한다면 불편하지 않을까? 그리고 서비스를 운영하다보면 시간, 계절, 이벤트 등에 따라 트래픽은 시시각각 변한다. 그럴 때마다 트래픽에 맞게 서버의 대수를 바꿔야 한다면 아주 불편할 것이다.

이런 불편함을 해결해주는 쿠버네티스의 기능이 **디플로이먼트(Deployment)**이다. 다음 강의에서 디플로이먼트에 대해 자세히 알아보자.

10. [보충] 파드(Pod) 디버깅 하는 방법

👤 개발을 하다보면 에러를 디버깅하고 해결하는 데에만 대부분의 시간을 쓴다. 따라서 어떤 기술을 익힐 때 반드시 에러를 디버깅 할 수 있는 방법을 정리해두어야 한다.

✅ 파드(Pod)가 정상적으로 실행되지 않았을 때

1. 매니페스트 파일 생성하기

nginx-pod.yaml

```
apiVersion: v1 # Pod를 생성할 때는 v1이라고 기재한다. (공식 문서)
kind: Pod # Pod를 생성한다고 명시
metadata:
  name: nginx-pod # Pod에 이름 붙이는 기능
spec:
  containers:
    - name: nginx-container # 생성할 컨테이너의 이름
      image: nginx:1.26.4 # 컨테이너를 생성할 때 사용할 Docker 이미지
      ports:
        - containerPort: 80 # 해당 컨테이너가 어떤 포트를 사용하는 지 명시적으로 표현
```

2. 파드 생성하기

```
$ kubectl apply -f nginx-pod.yaml
$ kubectl get pods # 파드가 잘 생성됐는지 파드 조회해보기
```

```
PS C:\DevData\Kubernetes\kube-practice> kubectl apply -f nginx-pod.yaml
pod/nginx-pod created
PS C:\DevData\Kubernetes\kube-practice> kubectl get pods
NAME        READY   STATUS    RESTARTS   AGE
nginx-pod   0/1     ErrImagePull  0          11s
PS C:\DevData\Kubernetes\kube-practice>
```

파드를 관리하고 생성하다보면 위와 같이 파드 생성에 실패하는 경우가 종종 생긴다. 위의 출력값을 보면 **STATUS**가 **ErrImagePull**인걸 보고 에러가 발생했음을 짐작할 수 있다. 하지만 구체적인 에러 메시지가 아니기에 **STATUS**만 보고 문제점을 단번에 알아차리기 어려운 경우가 종종 있다. 어떻게 에러 메시지를 구체적으로 확인하는 지 알아보자.

3. 에러 메시지 확인하기

```
# kubectl describe pods [파드명]
$ kubectl describe pods nginx-pod # nginx-pod 파드의 세부 정보 조회
```

node.kubernetes.io/unreacheable-NoExecute-op-Exists: 10m 300s

Events:	Type	Reason	Age	From	Message
	Normal	Scheduled	72s	default-scheduler	Successfully assigned default/nginx-pod to docker-desktop
	Normal	Pulling	29s (x3 over 71s)	kubelet	Pulling image "nginx:1.26.4"
	Warning	Failed	27s (x3 over 69s)	kubelet	Failed to pull image "nginx:1.26.4": Error response from daemon: failed to resolve reference "docker.io/library/nginx:1.26.4": docker
	Warning	Failed	27s (x3 over 69s)	kubelet	Error: ErrImagePull
	Normal	BackOff	12s (x3 over 68s)	kubelet	Back-off pulling image "nginx:1.26.4"
	Warning	Failed	12s (x3 over 68s)	kubelet	Error: ImagePullBackOff

✓ 파드(Pod)의 로그를 확인하고 싶을 때

1. 매니페스트 파일 수정하기

nginx-pod.yaml

```

apiVersion: v1 # Pod를 생성할 때는 v1이라고 기재한다. (공식 문서)
kind: Pod # Pod를 생성한다고 명시
metadata:
  name: nginx-pod # Pod에 이름 붙이는 기능
spec:
  containers:
    - name: nginx-container # 생성할 컨테이너의 이름
      image: nginx:1.26.2 # 컨테이너를 생성할 때 사용할 Docker 이미지
      ports:
        - containerPort: 80 # 해당 컨테이너가 어떤 포트를 사용하는 지 명시적으로 표현
  
```

2. 변경사항 적용시키기

```
$ kubectl apply -f nginx-pod.yaml
```

```

PS C:\DevData\Kubernetes\k8s-practice> kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
nginx-pod      1/1     Running   0           7m49s
spring-pod-1  1/1     Running   0           23s
spring-pod-2  1/1     Running   0           23s
spring-pod-3  1/1     Running   0           23s
PS C:\DevData\Kubernetes\k8s-practice>
  
```

3. 파드의 로그 확인하기

```

# kubectl logs [파드명]
$ kubectl logs nginx-pod # 파드 로그 확인하기
  
```

```

PS C:\DevData\Kubernetes\k8s-practice> kubectl logs nginx-pod
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/12/18 00:45:17 [notice] 1#1: using the "epoll" event method
2024/12/18 00:45:17 [notice] 1#1: nginx/1.26.2
2024/12/18 00:45:17 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/12/18 00:45:17 [notice] 1#1: OS: Linux 5.15.167.4-microsoft-standard-WSL2
2024/12/18 00:45:17 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/12/18 00:45:17 [notice] 1#1: start worker processes
2024/12/18 00:45:17 [notice] 1#1: start worker process 30
2024/12/18 00:45:17 [notice] 1#1: start worker process 31
2024/12/18 00:45:17 [notice] 1#1: start worker process 32
2024/12/18 00:45:17 [notice] 1#1: start worker process 33
  
```

✓ 파드(Pod)에 접속하고 싶을 때

```
# kubectl exec -it [파드명] -- bash
$ kubectl exec -it nginx-pod -- bash

# kubectl exec -it [파드명] -- sh
$ kubectl exec -it nginx-pod -- sh
```

- 도커에서 컨테이너로 접속하는 명령어(**docker exec -it [컨테이너 ID] bash**)와 비슷하다.
- 컨테이너 종류에 따라 컨테이너 내부에 **bash**가 설치되어 있을 수도 있고, **sh**가 설치되어 있을 수도 있다. 만약 **bash**가 설치되어 있지 않는데 **\$ kubectl exec -it nginx-pod -- bash** 명령어를 입력하면 에러가 뜨면서 컨테이너로 접속이 안 된다. 그럴 때는 **\$ kubectl exec -it nginx-pod -- sh**으로 접속을 시도해보자.

11. [요약] 지금까지 나온 명령어 정리

✓ 파드 조회

```
$ kubectl get pods
```

✓ 파드 포트 포워딩

```
# kubectl port-forward pod/[파드명] [로컬에서의 포트]/[파드에서의 포트]
$ kubectl port-forward pod/nginx-pod 80:80
```

✓ 파드 삭제

```
# kubectl delete pod [파드명]
$ kubectl delete pod nginx-pod # nginx-pod라는 파드 삭제
```

✓ 파드 디버깅

1. 파드 세부 정보 조회하기

```
# kubectl describe pods [파드명]
$ kubectl describe pods nginx-pod # nginx-pod 파드의 세부 정보
조회
```

2. 파드 로그 확인하기

```
# kubectl logs [파드명]
$ kubectl logs nginx-pod # 파드 로그 확인하기
```

3. 파드 내부로 접속하기

```
# kubectl exec -it [파드명] -- bash
$ kubectl exec -it nginx-pod -- bash

# kubectl exec -it [파드명] -- sh
$ kubectl exec -it nginx-pod -- sh
```

✓ 매니페스트 파일에 적혀져있는 리소스(파드 등) 생성

```
# kubectl apply -f [파일명]  
$ kubectl apply -f nginx-pod.yaml
```