

CT 활용하기#1

#1 docker Image를 K-ECP의 Container Platform에 등록시키기

본 가이드에서는 사용자의 로컬 PC에서 docker파일을 생성하고 해당 파일을 build하여 docker image를 생성시키고 K-ECP의 Container Platform에 등록시키는 과정을 안내합니다.

관련 안내서

- [Container 시작하기](#)
- [Container Terminal 시작하기](#)

목차

[전제 조건](#)

1단계: docker 개발환경 구축

2단계: docker image K-ECP에 배포

3단계: K-ECP Container실행

[다음단계](#)

전제 조건

- 사용자의 로컬 PC에 docker가 설치되어 있어야 합니다. [URL: <https://www.docker.com/products/docker-desktop/>]

1단계: docker 개발환경 구축

 **안내:** 본 가이드에서는 Nginx를 사용하여 HTML기반 Website기동을 시행합니다.

1. docker 파일을 구성할 디렉토리 생성(본 가이드에서는 디렉토리명을 d2ocp로 설정)
2. docker 파일을 구성할 HTML 페이지와 dockerfile 생성

 **안내:** 1. 에서 생성한 디렉토리 안에 2개의 파일 생성

- index.html

```
<html>
<head>
  <title>Docker Image to K-ECP Container</title>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>

<body>
  <h1>DI20CP Application</h1>
  <h2>Docker Image(DI)로 K-ECP 컨테이너 실행하기</h2>
</body>
</html>
```


- Dockerfile

```
FROM nginx:latest
COPY index.html /usr/share/nginx/html
```

3. 디렉토리 안에 `index.html` 과 `dockerfile` 이 있는지 확인

4. 내부 Repository에 Docker Image 빌드

 **안내:** 도커파일을 구성할 파일들이 있는 디렉토리에서 명령어를 실행해야 합니다. (본 가이드의 경우 디렉토리명은 d2ocp입니다.)

 **안내:** `[DockerImage_name]` 은 사용자가 원하는 도커이미지의 이름을 작성하면 됩니다. `[tag]` 는 해당 도커 이미지의 버전관리를 위해 사용됩니다.

```
docker build -t [DockerImage_name]:[tag] .
```

```
[+] Building 0.1s (7/7) FINISHED
docker:default
=> [internal] load build definition from dockerfile
0.0s
=> => transferring dockerfile: 152B
0.0s
=> [internal] load .dockerignore
0.0s
=> => transferring context: 2B
0.0s
=> [internal] load metadata for docker.io/library/nginx:latest
0.0s
=> [internal] load build context
0.0s
=> => transferring context: 91B
0.0s
=> [1/2] FROM docker.io/library/nginx:latest
0.0s
=> CACHED [2/2] COPY index.html /usr/share/nginx/html
0.0s
=> exporting to image
0.0s
=> => exporting layers
0.0s
=> => writing image
sha256:db9f6ac1f6e0a92b584ba5121dacf0df1b59520a0403ac5f2b1ef99ce14bef2e
0.0s
=> => naming to docker.io/library/d2ocp
0.0s
```

3. 내부 docker Repository에서 이미지 확인

```
docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
[DockerImage_name]	[tag]	db9f6ac1f6e0	10 seconds ago	187MB

4. 로컬에서 docker image로 컨테이너가 실행되는지 확인

```
docker run --rm -p 7878:80 [DockerImage_name]:[tag]
```

- 로컬 브라우저(<http://localhost:7878>) 혹은 터미널 curl 명령어를 통해 기동 확인

```
curl http://localhost:7878
```

```
<html>  
<head>  
    <title>Docker Image to K-ECP Container</title>  
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  
</head>  
  
<body>  
    <h1>DI20CP Application</h1>  
    <h2>Docker Image(DI)?? K-ECP ???=? ?? ????_?</h2>  
</body>  
</html>
```

5. 해당 docker image를 파일로 저장

 **안내:** [DockerImage.tar] 는 도커 이미지를 압축할 압축파일명을 작성하면 됩니다.

```
docker save -o [DockerImage.tar] [DockerImage_name]:[tag]
```

- 해당 save 명령어를 실행한 작업공간에 [DockerImage_name.tar]이름의 tar 압축파일이 생성되었는지 확인

2단계: docker image K-ECP에 배포

⚠ **주의사항:** nginx 등 os 권한상승이 필요할 경우 사전에 K-ECP 운영팀에 문의해 주시기 바랍니다.

1. [Container Terminal 시작하기](#)를 통해서 CT 접속
2. [DockerImage_name.tar] 압축파일 CT로 전송
 - [DockerImage_name.tar] 파일이 있는 로컬 PC의 디렉토리에서 터미널 명령어 실행(본 가이드에서는 /home/kecpuser 에 파일을 전송)

```
scp -P 10040 [DockerImage_name.tar] kecpuser@[CT_IP]:/hoem/kecpuser
```

💡tip: `/home/kecpuser` 의 경우 사용자가 원하는 경로를 작성하여 해당 경로에 압축파일을 전송할 수 있습니다.

3. CT에서 [DockerImage_name.tar] 파일확인

- [DockerImage_name.tar]가 있는 경로로 이동

```
cd /home/kecpuser
```

- 해당 경로에 압축파일이 있는지 확인

```
ls
```

```
[DockerImage_name.tar]
```

4. CT에 docker image 파일 업로드

```
podman load -i [DockerImage_name.tar]
```

```
Getting image source signatures
Copying blob 1998c5cd2230 done
Copying blob 434c6a715c30 done
Copying blob 24839d45ca45 done
Copying blob f36897eea34d done
Copying blob b821d93f6666 done
Copying blob 9fdfd12bc85b done
Copying blob 3c9d04c9ebd5 done
Copying blob 9099fc4eae86 done
Copying config af59c66793 done
Writing manifest to image destination
Storing signatures
Loaded image(s): localhost/[DockerImage_name]:[tag]
```

5. CT에 docker image 업로드 확인

```
podman images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
localhost/[DockerImage_name]	[tag]	af59c6679339	4 hours ago	191 MB

6. CT에서 podman을 통해 K-ECP의 OSP 레지스트리 로그인

🔔 **안내:** [CT_USER_ID]의 경우 CT서비스에서 생성한 USER ID 입니다.

```
podman login -u [CT_USER_ID] -p $(oc whoami -t) default-route-openshift-image-registry.apps.ocp4.kdnecp.com --tls-verify=false
```

7. 현재 podman image에 등록된 docker image의 tag 수정

```
podman tag [DockerImage_name]:[tag] default-route-openshift-image-registry.apps.ocp4.kdnecp.com/[project명]/[DockerImage_name]:[tag]
```

- podman image에 tag가 수정되었는지 확인

```
podman images
```

```
REPOSITORY
TAG          IMAGE ID      CREATED      SIZE
localhost/[DockerImage_name]
[tag]        af59c6679339 4 hours ago  191 MB
default-route-openshift-image-registry.apps.ocp4.kdnecp.com/ssg-test-del/[DockerImage_name] [tag]        af59c6679339 4 hours ago  191 MB
```

8. K-ECP의 OSP 레지스트리에 tag가 수정된 docker image push

```
podman push default-route-openshift-image-registry.apps.ocp4.kdnecp.com/[project명]/[DockerImage_name]:[tag] --tls-verify=false
```

```
Getting image source signatures
Copying blob 434c6a715c30 skipped: already exists
Copying blob f36897eea34d skipped: already exists
Copying blob 1998c5cd2230 skipped: already exists
Copying blob 9fdfd12bc85b skipped: already exists
Copying blob b821d93f6666 skipped: already exists
Copying blob 24839d45ca45 skipped: already exists
Copying blob 9099fc4eae86 skipped: already exists
Copying blob 3c9d04c9ebd5 [-----] 0.0b / 0.0b
Copying config af59c66793 [=====] 7.1KiB / 7.1KiB
Writing manifest to image destination
Storing signatures
```


⚠ **주의사항:** podman push 명령어 실행시 오류 발생시 같은 명령어를 재실행 해주시기 바랍니다.

9. 해당 project의 image stream에 docker image가 등록되었는지 확인

```
oc get is
```

NAME	IMAGE REPOSITORY
[DockerImage_name]	default-route-openshift-image-registry.apps.ocp4.kdnecp.com/ssg-test-del/[DockerImage_name]
TAGS	[tag]
UPDATED	2 minutes ago

3단계: K-ECP OSP에서 Container 실행

 **안내:** 이전 단계에서 K-ECP OSP에 등록한 docker image를 기반으로 Conatiner를 실행합니다.

1. 이미지 기반으로 Conatiner 실행

```
oc new-app [DockerImage_name] --name=[App명 (Container 명)]
```

```
--> Found image af59c66 (5 hours old) in image stream "ssg-test-del/[DockerImage_name]" under tag "[tag]" for "[DockerImage_name]"
```

```
--> Creating resources ...
```

```
deployment.apps "[App명 (Container 명)]" created
```

```
service "[App명 (Container 명)]" created
```

```
--> Success
```

Application is not exposed. You can expose services to the outside world by executing one or more of the commands below:

```
'oc expose service/[App명 (Container 명)]'
```

Run 'oc status' to view your app.

2. 생성된 pod 및 service 확인

```
oc get pod
```

NAME	READY	STATUS	RESTARTS	AGE
[App명 (Container 명)]-6476b476d6-s6sqx	1/1	Running	0	2m38s

```
oc get service
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
[App명 (Container 명)]	ClusterIP	172.30.235.243	<none>	80/TCP	3m15s

3. 생성된 service 외부 노출

```
oc expose service/[App명 (Container 명)]
```

```
route.route.openshift.io/[App명 (Container 명)] exposed
```

4. service 외부 노출 확인

```
oc get route
```

NAME	HOST/PORT	PATH	SERVICES
PORT	TERMINATION	WILDCARD	
[App명 (Container 명)]	di2ocp-ssg-test-del.apps.ocp4.kdnecp.com		[App명
(Container 명)]	80-tcp	None	

5. HOST/PORT에 표시된 URL을 통해 브라우저에서 HTML기반 Website 기동 확인

```
http://di2ocp-app-ssg-test-del.apps.ocp4.kdnecp.com/
```

다음단계

- CT 활용하기 #2 를 통해 실행중인 Container를 관리할 수 있습니다.