

## laC로 Azure Container Apps 배포

**목차**

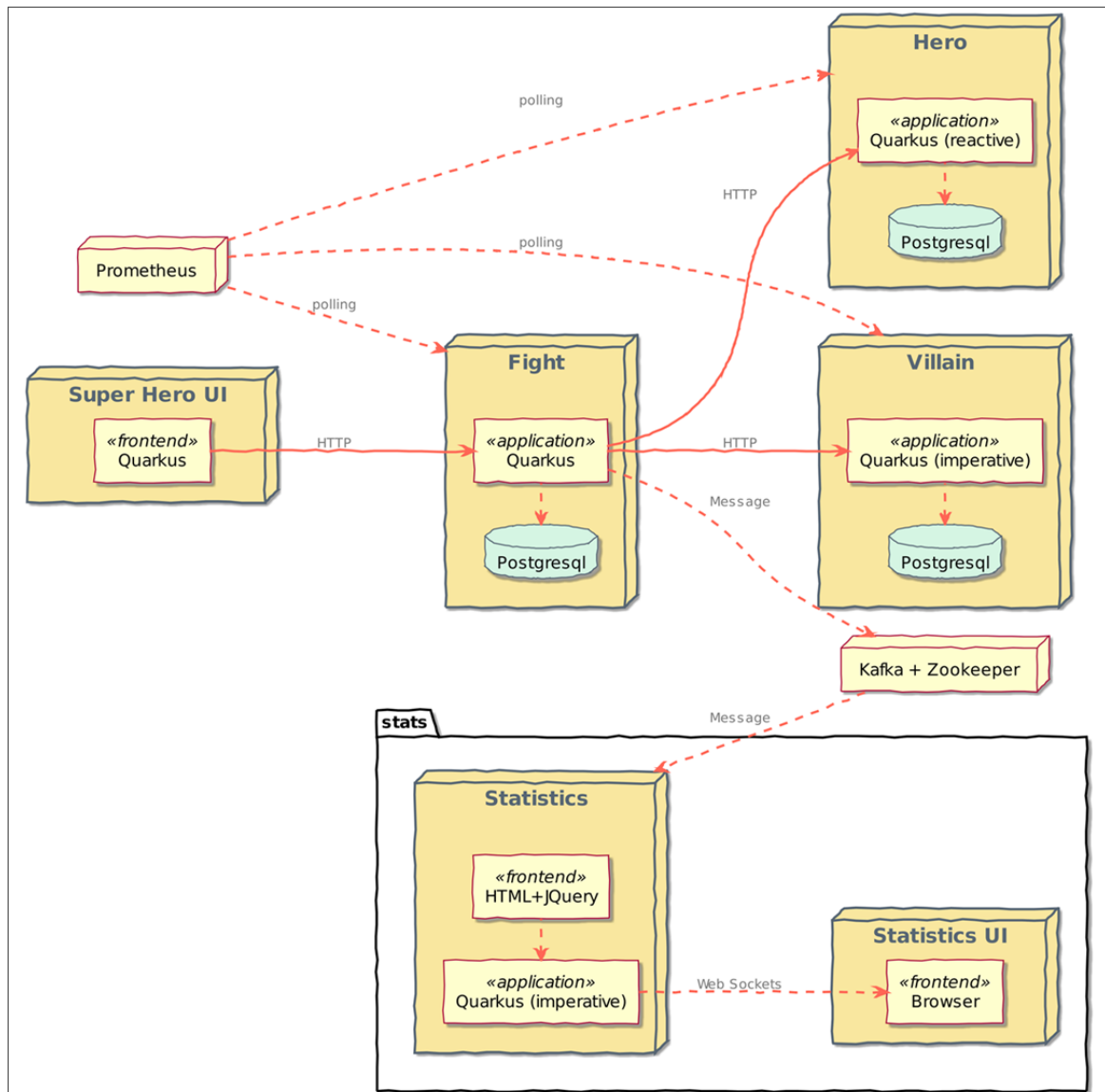
개요.....	2
EXERCISE 01. BICEP을 사용하여 AZURE CONTAINER APPS 배포.....	4
EXERCISE 02. LOG ANALYTICS로 로그 확인.....	12

## 개요

이 실습에서는 Azure Container Apps를 사용하는 실질적인 방법에 대해 알아봅니다. 이를 위해 일반적인 HTTP, 반응형 및 이벤트 기반 마이크로서비스를 혼합하여 전체 마이크로서비스 아키텍처를 배포하는데 필요한 모든 도구에 대해 알아봅니다. 또한 부하를 처리할 수 있도록 마이크로서비스를 모니터링하고 확장합니다.

이 실습은 슈퍼 히어로와 슈퍼 빌런이 대결하는 애플리케이션을 배포합니다. 이를 위해 여러 마이크로서비스를 컨테이너화하고 배포하며 이런 마이크로서비스는 REST를 통한 동기식, Kafka를 통한 비동기식으로 커뮤니케이션합니다. 실습에서는 5가지 구성 요소가 배포됩니다.

- **Super Hero UI**: 랜덤하게 슈퍼 히어로와 슈퍼 빌런을 선택한 후 싸우게 하는 Angular 애플리케이션입니다. Super Hero UI는 Quarkus를 통해 노출되며 Fight REST API를 호출합니다.
- **Villain REST API**: PostgreSQL 데이터베이스에 저장되어 있는 슈퍼 빌런에 대한 CRUD 작업을 노출하는 일반적인 HTTP 마이크로서비스입니다.
- **Hero REST API**: PostgreSQL 데이터베이스에 저장되어 있는 슈퍼 히어로에 대한 CRUD 작업을 노출하는 반응형 HTTP 마이크로서비스입니다.
- **Fight REST API**: 이 REST API는 Hero REST API와 Villain REST API를 호출하여 랜덤한 슈퍼 히어로와 슈퍼 빌런을 가져옵니다. 각 싸움은 PostgreSQL 데이터베이스에 저장됩니다. 이 마이크로서비스는 기존 방식(명령형) 또는 반응적 접근 방식을 모두 사용하여 개발할 수 있습니다. 슈퍼 히어로와 슈퍼 빌런에 대한 호출은 복원력 패턴(재시도, timeout, circuit-breaker)을 사용하여 보호됩니다.
- **Statistics**: 각 싸움은 Kafka를 통해 비동기식으로 Statistics 마이크로서비스에 전송됩니다. 또한 모든 통계를 표시하는 HTML + JQuery UI가 제공됩니다.



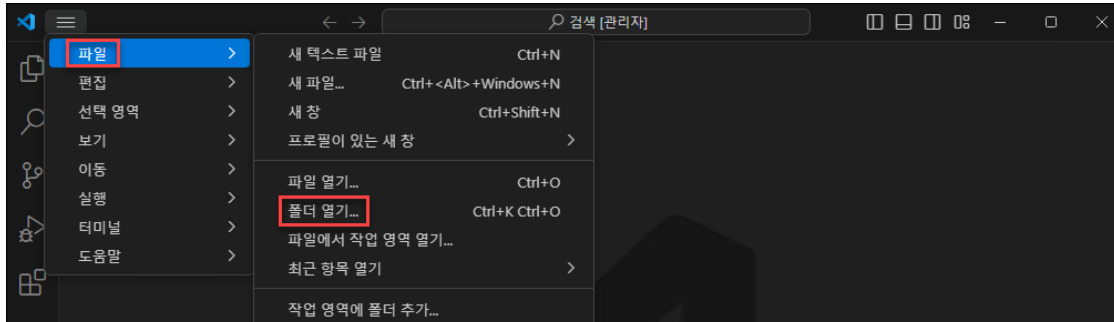
실습은 다음과 같은 순서로 진행됩니다.

- ① 필요한 도구 설치: 애플리케이션을 번들, 패키징 및 배포할 수 있는 모든 도구와 코드를 설치합니다.
- ② Azure Container Apps와 Quarkus: Azure Container Apps와 Quarkus에 대한 설명을 제공합니다.
- ③ 애플리케이션을 컨테이너로 빌드: 이 작업 단계는 선택적으로 테스트할 수 있습니다. 이미 만들어진 컨테이너 이미지를 바로 사용할 경우 이 작업 단계를 건너뛸 수 있습니다.
- ④ 로컬에서 애플리케이션 실행: Quarkus 마이크로서비스를 위한 Docker 이미지를 풀링한 다음 Docker Compose를 사용하여 로컬에서 실행하고 이 이미지를 다시 Azure Container Registry로 푸시합니다.
- ⑤ Azure Container Apps에서 애플리케이션 실행: Azure에 필요한 모든 인프라(PostgreSQL, Kafka 등)를 만들고 마이크로서비스를 Azure Container Apps에 배포합니다.

## EXERCISE 01. Bicep을 사용하여 Azure Container Apps 배포

이 작업에서는 앞서 진행했던 Azure CLI 기반의 작업을 Bicep 코드로 작성하여 배포합니다.

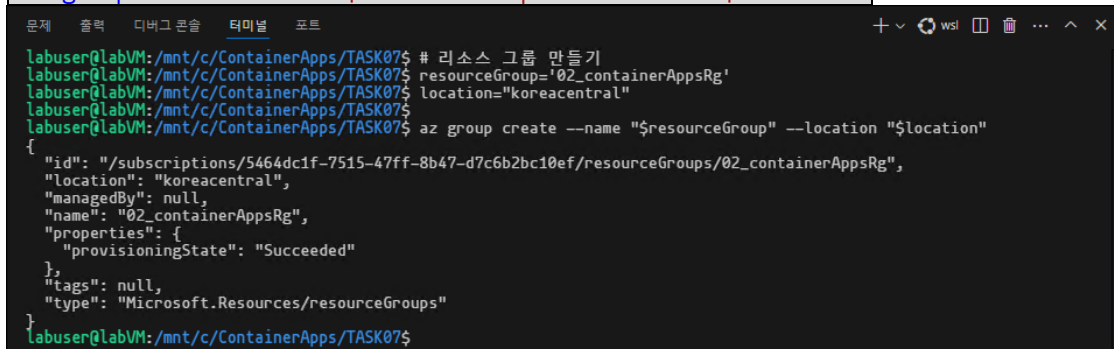
1. 실습 파일을 다운로드한 후 `C:\ContainerApps\TASK07` 폴더에 압축을 풉니다.
2. [Visual Studio Code]를 열고 메뉴에서 [파일 - 폴더 열기...]를 클릭한 후 "`C:\ContainerApps\TASK07`" 폴더를 엽니다.



3. [Visual Studio Code]의 메뉴에서 [터미널 - 새 터미널]을 열고 "Ubuntu (WSL)" 터미널을 엽니다. WSL 터미널에서 다음 명령을 실행하여 실습에 필요한 리소스 그룹을 만듭니다.

```
# 리소스 그룹 만들기
resourceGroup='02_containerAppsRg'
location='koreacentral'

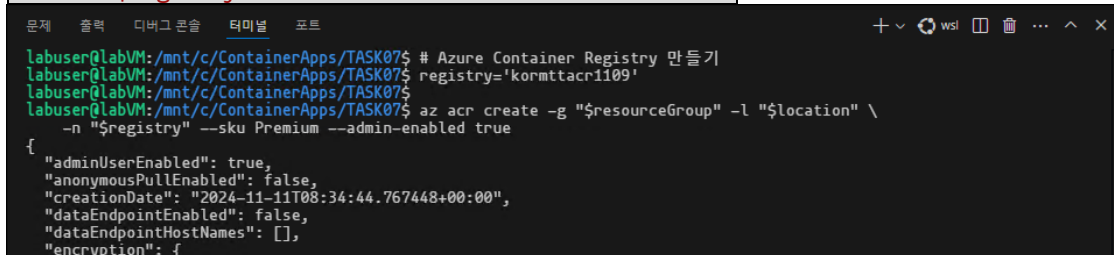
az group create --name "$resourceGroup" --location "$location"
```



4. WSL 터미널에서 다음 명령을 실행하여 Azure Container Registry를 만듭니다. 입력한 레지스트리 이름은 Bicep 템플릿 배포에서 사용할 것이기 때문에 메모장에 기록해 놓습니다.

```
# Azure Container Registry 만들기
registry=[UNIQUE_NAME]

az acr create -g "$resourceGroup" -l "$location" \
  -n "$registry" --sku Premium --admin-enabled true
```



5. WSL 터미널에서 다음 명령을 실행하여 Azure Container Registry 로그인 서버를 확인합니다.

```
# Azure Container Registry 로그인 서버 확인
registryUrl=$(az acr show -g "$resourceGroup" \
  -n "$registry" --query "loginServer" -o tsv)
```

```
echo $registryUrl

labuser@labVM:/mnt/c/ContainerApps/TASK07$ # Azure Container Registry 로그인 서버 확인
labuser@labVM:/mnt/c/ContainerApps/TASK07$ registryUrl=$(az acr show --g "$resourceGroup" \
  --n "$registry" --query "loginServer" -o tsv)
labuser@labVM:/mnt/c/ContainerApps/TASK07$
labuser@labVM:/mnt/c/ContainerApps/TASK07$ echo $registryUrl
kormttacr1109.azurecr.io
labuser@labVM:/mnt/c/ContainerApps/TASK07$
```

6. WSL 터미널에서 다음 명령을 실행하여 Azure Container Registry로 푸시할 이미지를 풀링합니다.

```
# 컨테이너 이미지 가져오기
docker pull 313mlclub/super-heroes-ui:1.0
docker pull 313mlclub/statistics-app:1.0
docker pull 313mlclub/fights-app:1.0
docker pull 313mlclub/villains-app:1.0
docker pull 313mlclub/heroes-app:1.0

labuser@labVM:/mnt/c/ContainerApps/TASK07$ # 컨테이너 이미지 가져오기
labuser@labVM:/mnt/c/ContainerApps/TASK07$ docker pull 313mlclub/super-heroes-ui:1.0
1.0: Pulling from 313mlclub/super-heroes-ui
3550ee360766: Already exists
e60614b44153: Already exists

labuser@labVM:/mnt/c/ContainerApps/TASK07$ docker pull 313mlclub/statistics-app:1.0
1.0: Pulling from 313mlclub/statistics-app
3550ee360766: Already exists
e60614b44153: Already exists

labuser@labVM:/mnt/c/ContainerApps/TASK07$ docker pull 313mlclub/fights-app:1.0
1.0: Pulling from 313mlclub/fights-app
3550ee360766: Already exists
e60614b44153: Already exists

labuser@labVM:/mnt/c/ContainerApps/TASK07$ docker pull 313mlclub/villains-app:1.0
1.0: Pulling from 313mlclub/villains-app
3550ee360766: Already exists
e60614b44153: Already exists

labuser@labVM:/mnt/c/ContainerApps/TASK07$ docker pull 313mlclub/heroes-app:1.0
1.0: Pulling from 313mlclub/heroes-app
3550ee360766: Already exists
e60614b44153: Already exists
7c5f6845c465: Pull complete
606533fcfb09: Pull complete
ef43e9dc9ec7: Pull complete
2bb082c8a850: Pull complete
Digest: sha256:1ce7a7e8be0b9b38adb9088ce500be35f3f4bf184bb67436a0cec6800b44b49f
Status: Downloaded newer image for 313mlclub/heroes-app:1.0
docker.io/313mlclub/heroes-app:1.0
labuser@labVM:/mnt/c/ContainerApps/TASK07$
```

7. WSL 터미널에서 다음 명령을 실행하여 풀링한 컨테이너 이미지를 확인합니다.

```
# 컨테이너 이미지 확인
docker images | grep 313mlclub

labuser@labVM:/mnt/c/ContainerApps/TASK07$ # 컨테이너 이미지 확인
labuser@labVM:/mnt/c/ContainerApps/TASK07$ docker images | grep 313mlclub
313mlclub/super-heroes-ui 1.0 65f8f1ec6410 17 months ago 418MB
313mlclub/statistics-app 1.0 e91064b8afb3 17 months ago 433MB
313mlclub/fights-app 1.0 f2d4c77b7313 17 months ago 454MB
313mlclub/villains-app 1.0 7af6b90a87b0 17 months ago 438MB
313mlclub/heroes-app 1.0 37d029d51873 17 months ago 436MB
labuser@labVM:/mnt/c/ContainerApps/TASK07$
```

8. WSL에서 다음 명령을 실행하여 풀링한 이미지에 태그를 지정하기 위한 변수를 선언합니다. 태그 이름은 Azure Container Registry의 로그인 이름으로 지정해야 합니다.

```
# 새 컨테이너 이미지 태그를 위한 변수 지정
imageTag="1.0"

heroesApp="heroes-app"
heroesImage="${registryUrl}/${heroesApp}:${imageTag}"

villainsApp="villains-app"
villainsImage="${registryUrl}/${villainsApp}:${imageTag}"

fightsApp="fights-app"
fightsImage="${registryUrl}/${fightsApp}:${imageTag}"
```

```
statisticsApp="statistics-app"
statisticsImage="${registryUrl}/${statisticsApp}:${imageTag}"

uiApp="super-heroes-ui"
uiImage="${registryUrl}/${uiApp}:${imageTag}"
```

```
문제 출력 디버그 콘솔 터미널 포트
labuser@labVM:/mnt/c/ContainerApps/TASK07$ # 새 컨테이너 이미지 태그를 위한 변수 지정
labuser@labVM:/mnt/c/ContainerApps/TASK07$ imageTag="1.0"
labuser@labVM:/mnt/c/ContainerApps/TASK07$
labuser@labVM:/mnt/c/ContainerApps/TASK07$ heroesApp="heroes-app"
labuser@labVM:/mnt/c/ContainerApps/TASK07$ heroesImage="${registryUrl}/${heroesApp}:${imageTag}"
labuser@labVM:/mnt/c/ContainerApps/TASK07$
labuser@labVM:/mnt/c/ContainerApps/TASK07$ villainsApp="villains-app"
labuser@labVM:/mnt/c/ContainerApps/TASK07$ villainsImage="${registryUrl}/${villainsApp}:${imageTag}"
labuser@labVM:/mnt/c/ContainerApps/TASK07$
labuser@labVM:/mnt/c/ContainerApps/TASK07$ fightsApp="fights-app"
labuser@labVM:/mnt/c/ContainerApps/TASK07$ fightsImage="${registryUrl}/${fightsApp}:${imageTag}"
labuser@labVM:/mnt/c/ContainerApps/TASK07$
labuser@labVM:/mnt/c/ContainerApps/TASK07$ statisticsApp="statistics-app"
labuser@labVM:/mnt/c/ContainerApps/TASK07$ statisticsImage="${registryUrl}/${statisticsApp}:${imageTag}"
labuser@labVM:/mnt/c/ContainerApps/TASK07$
labuser@labVM:/mnt/c/ContainerApps/TASK07$ uiApp="super-heroes-ui"
labuser@labVM:/mnt/c/ContainerApps/TASK07$ uiImage="${registryUrl}/${uiApp}:${imageTag}"
labuser@labVM:/mnt/c/ContainerApps/TASK07$
```

9. WSL 터미널에서 다음 명령을 실행하여 풀링한 이미지를 Azure Container Registry 로그인 서버 이름과 일치하도록 태그를 지정합니다.

```
# 새 이미지 태그 지정
docker tag 313mlclub/super-heroes-ui:1.0 $uiImage
docker tag 313mlclub/statistics-app:1.0 $statisticsImage
docker tag 313mlclub/fights-app:1.0 $fightsImage
docker tag 313mlclub/villains-app:1.0 $villainsImage
docker tag 313mlclub/heroes-app:1.0 $heroesImage
```

```
문제 출력 디버그 콘솔 터미널 포트
labuser@labVM:/mnt/c/ContainerApps/TASK07$ # 새 이미지 태그 지정
labuser@labVM:/mnt/c/ContainerApps/TASK07$ docker tag 313mlclub/super-heroes-ui:1.0 $uiImage
labuser@labVM:/mnt/c/ContainerApps/TASK07$ docker tag 313mlclub/statistics-app:1.0 $statisticsImage
labuser@labVM:/mnt/c/ContainerApps/TASK07$ docker tag 313mlclub/fights-app:1.0 $fightsImage
labuser@labVM:/mnt/c/ContainerApps/TASK07$ docker tag 313mlclub/villains-app:1.0 $villainsImage
labuser@labVM:/mnt/c/ContainerApps/TASK07$ docker tag 313mlclub/heroes-app:1.0 $heroesImage
labuser@labVM:/mnt/c/ContainerApps/TASK07$
```

10. WSL 터미널에서 다음 명령을 실행하여 새로 지정한 이미지 태그를 확인합니다. 태그 이름이 Azure Container Registry의 로그인 서버 이름과 일치하는지 확인합니다.

```
# 새 태그를 지정한 이미지 확인
docker images | grep $registryUrl
```

```
문제 출력 디버그 콘솔 터미널 포트
labuser@labVM:/mnt/c/ContainerApps/TASK07$ # 새 태그를 지정한 이미지 확인
labuser@labVM:/mnt/c/ContainerApps/TASK07$ docker images | grep $registryUrl
kormttacr1109.azurecr.io/super-heroes-ui 1.0 65f8f1ec6410 17 months ago 418MB
kormttacr1109.azurecr.io/statistics-app 1.0 e91064b8af63 17 months ago 433MB
kormttacr1109.azurecr.io/fights-app 1.0 f2d4c77b7313 17 months ago 454MB
kormttacr1109.azurecr.io/villains-app 1.0 7af6b90a87b0 17 months ago 438MB
kormttacr1109.azurecr.io/heroes-app 1.0 37d029d51873 17 months ago 436MB
labuser@labVM:/mnt/c/ContainerApps/TASK07$
```

11. WSL 터미널에서 다음 명령을 실행하여 Azure Container Registry에 로그인합니다. Azure Pass 구독을 사용하는 경우 구독 수준의 소유자 권한이 할당되어 있지 않고 "서비스 관리자" 클래식 관리자 역할만 할당되어 있는 경우 "Unable to get AAD authorization tokens with message" 경고가 발생할 수 있습니다.

```
# Azure Container Registry에 로그인
az acr login --name "$registry"
```

```
문제 출력 디버그 콘솔 터미널 포트
labuser@labVM:/mnt/c/ContainerApps/TASK07$ # Azure Container Registry에 로그인
labuser@labVM:/mnt/c/ContainerApps/TASK07$ az acr login --name "$registry"
Login Succeeded
labuser@labVM:/mnt/c/ContainerApps/TASK07$
```

12. WSL 터미널에서 다음 명령을 실행하여 이미지를 Azure Container Registry로 푸시합니다.

```
# Azure Container Registry에 이미지 푸시
docker push $uiImage
docker push $statisticsImage
```

```
docker push $fightersImage
docker push $villainsImage
docker push $heroesImage
```

```
labuser@labVM:/mnt/c/ContainerApps/TASK07$ # Azure Container Registry에 이미지 푸시
labuser@labVM:/mnt/c/ContainerApps/TASK07$ docker push $uiImage
The push refers to repository [kormttacr1109.azurecr.io/super-heroes-ui]
185df4f30abd: Pushed
1ec3c2cd9c91: Pushed

labuser@labVM:/mnt/c/ContainerApps/TASK07$ docker push $statisticsImage
The push refers to repository [kormttacr1109.azurecr.io/statistics-app]
94a0d7725a1d: Pushed
58b0d4d9c02f: Pushed

labuser@labVM:/mnt/c/ContainerApps/TASK07$ docker push $fightersImage
The push refers to repository [kormttacr1109.azurecr.io/fights-app]
9ef1d63d6acb: Pushed
e3d90521843c: Pushed

labuser@labVM:/mnt/c/ContainerApps/TASK07$ docker push $villainsImage
The push refers to repository [kormttacr1109.azurecr.io/villains-app]
e0c0398ec994: Pushed
482ac4b571b4: Pushed

labuser@labVM:/mnt/c/ContainerApps/TASK07$ docker push $heroesImage
The push refers to repository [kormttacr1109.azurecr.io/heroes-app]
e8a412b76425: Pushed
7fcfb0c72b02: Pushed
32e99e533641: Pushed
f9a4db71de33: Pushed
75d18d488805: Mounted from villains-app
e96a181d185e: Mounted from villains-app
1.0: digest: sha256:1ce7a7e8be0b9b38adb9088ce500be35f3f4bf184bb67436a0cec6800b44b49f size: 1582
labuser@labVM:/mnt/c/ContainerApps/TASK07$
```

13. WSL 터미널에서 다음 명령을 실행하여 Azure Container Registry로 푸시한 이미지 리포지토리를 확인합니다.

```
# Azure Container Registry에 푸시된 이미지 확인
az acr repository list -n "$registry" -o table
```

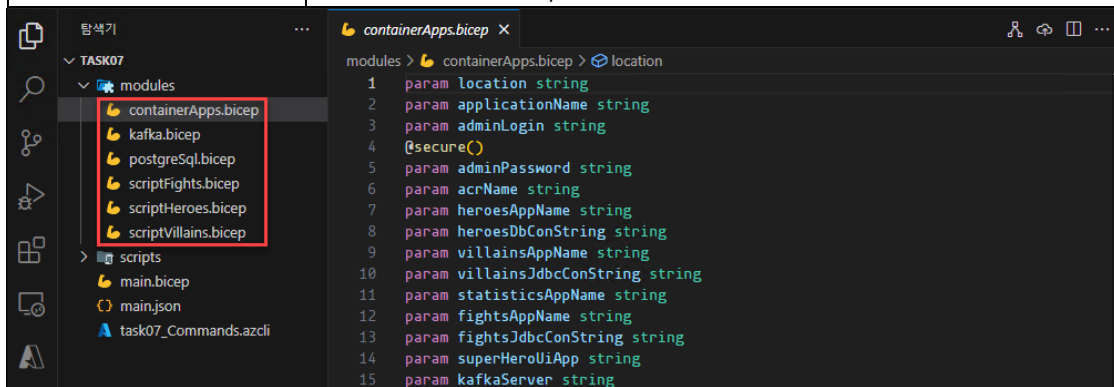
```
labuser@labVM:/mnt/c/ContainerApps/TASK07$ # Azure Container Registry에 푸시된 이미지 확인
labuser@labVM:/mnt/c/ContainerApps/TASK07$ az acr repository list -n "$registry" -o table
Result
-----
fighters-app
heroes-app
statistics-app
super-heroes-ui
villains-app
labuser@labVM:/mnt/c/ContainerApps/TASK07$
```

14. [Visual Studio Code]의 [탐색기]에서 "modules" 폴더로 이동합니다. 다음과 같은 Bicep 모듈 파일을 확인합니다.

Bicep 모듈	설명
containerApps.bicep	<ul style="list-style-type: none"> <li>5 개의 마이크로서비스 애플리케이션을 Azure Container Apps 로 배포하기 위한 Bicep 파일입니다.</li> <li>Container Apps 를 배포하기 위해 필요한 Log Analytics 작업 영역, Container Apps Environment 도 함께 프로비저닝됩니다.</li> <li>Container Apps 를 배포할 때 사용하는 이미지를 가져오기 위해 Azure CLI 로 배포한 Azure Container Registry 를 참조로 가져옵니다.</li> <li>각 Container Apps 는 컨테이너 구동을 위해 필요한 환경 변수가 설정되어 있습니다. 이러한 환경 변수는 매개 변수나 변수를 통해 처리됩니다.</li> </ul>
kafka.bicep	<ul style="list-style-type: none"> <li>Event Hub 네임스페이스를 만들고 Kafka 를 위한 토픽을 생성합니다.</li> <li>Event Hub 의 RootManageSharedAccessKey 는 Container Apps 에서 환경 변수로 처리되어야 하기 때문에 output 으로 이 값을 출력합니다.</li> </ul>
postgreSQL.bicep	<ul style="list-style-type: none"> <li>heroes 앱, villains 앱, fights 앱에서 사용할 PostgreSQL 을 배포합니다.</li> <li>Azure Database for PostgreSQL Flexible Server 를 배포하며 자식 개체로 방화벽 규칙과 데이터베이스를 함께 배포합니다.</li> <li>동일한 설정으로 3 개의 데이터베이스가 배포되어야 하기 때문에 이 모듈은 main.bicep 파일에서 loop 를 통해 처리됩니다.</li> </ul>
scriptFights.bicep	<ul style="list-style-type: none"> <li>fights 데이터베이스의 스키마와 테이블을 만들고 테스트에 필요한 데이터를 입력하기 위한 Bicep 배포 스크립트입니다.</li> </ul>

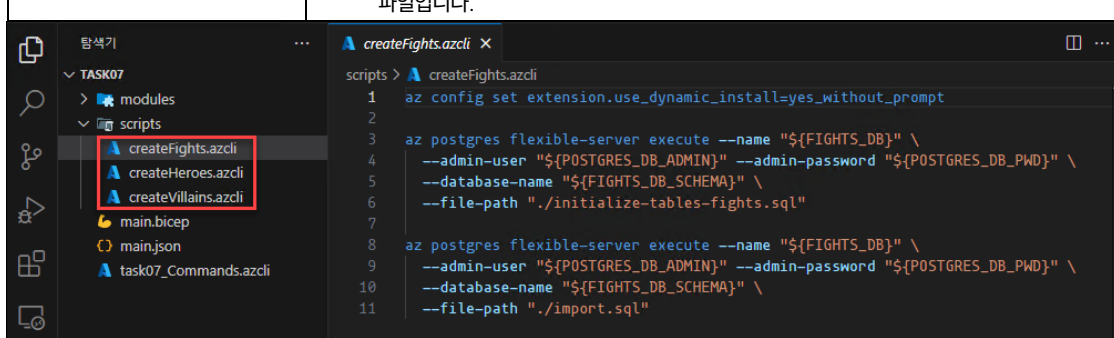


	<ul style="list-style-type: none"> <li>Bicep 배포 스크립트는 Azure Container Instance 를 사용하여 실행되며 "scripts\createFights.azcli" 파일을 실행합니다.</li> <li>데이터베이스 테이블을 만들고 초기 데이터를 입력하는 T-SQL 문은 GitHub 리포지토리에서 지원 스크립트로 가져옵니다.</li> </ul>
scriptHeroes.bicep	<ul style="list-style-type: none"> <li>heroes 데이터베이스의 스키마와 테이블을 만들고 테스트에 필요한 데이터를 입력하기 위한 Bicep 배포 스크립트입니다.</li> </ul>
scriptVillains.bicep	<ul style="list-style-type: none"> <li>villains 데이터베이스의 스키마와 테이블을 만들고 테스트에 필요한 데이터를 입력하기 위한 Bicep 배포 스크립트입니다.</li> </ul>



15. [Visual Studio Code]의 탐색기에서 "scripts" 폴더로 이동한 후 다음과 같은 배포 스크립트 파일을 확인합니다.

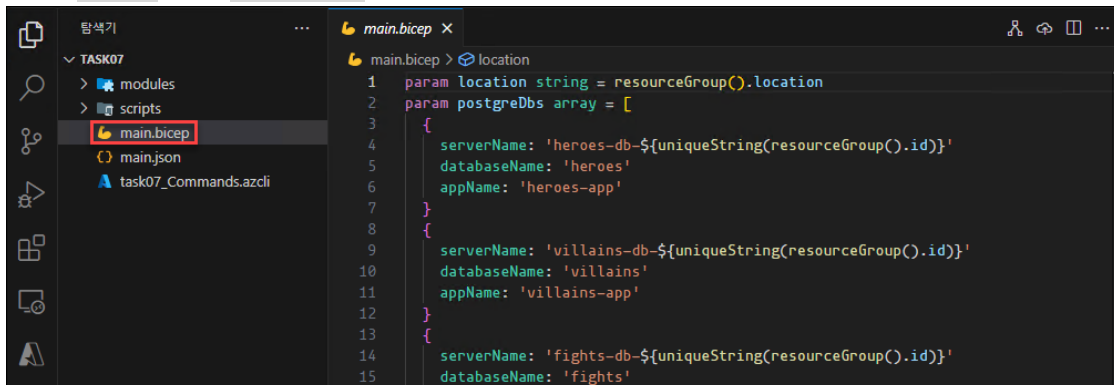
스크립트 파일	설명
createFights.azcli	<ul style="list-style-type: none"> <li>fightes 데이터베이스에 테이블을 만들고 초기 데이터를 입력하는 스크립트 파일입니다.</li> <li>스크립트에서 사용하는 매개 변수는 Bicep 파일의 배포 스크립트에서 environmentVariables 를 통해 처리됩니다.</li> <li>스크립트에서 사용하는 T-SQL 파일은 Bicep 파일에서 supportingScriptUri 를 통해 처리됩니다.</li> </ul>
createHeroes.azcli	<ul style="list-style-type: none"> <li>heroes 데이터베이스에 테이블을 만들고 초기 데이터를 입력하는 스크립트 파일입니다.</li> </ul>
createVillains.azcli	<ul style="list-style-type: none"> <li>villains 데이터베이스에 테이블을 만들고 초기 데이터를 입력하는 스크립트 파일입니다.</li> </ul>



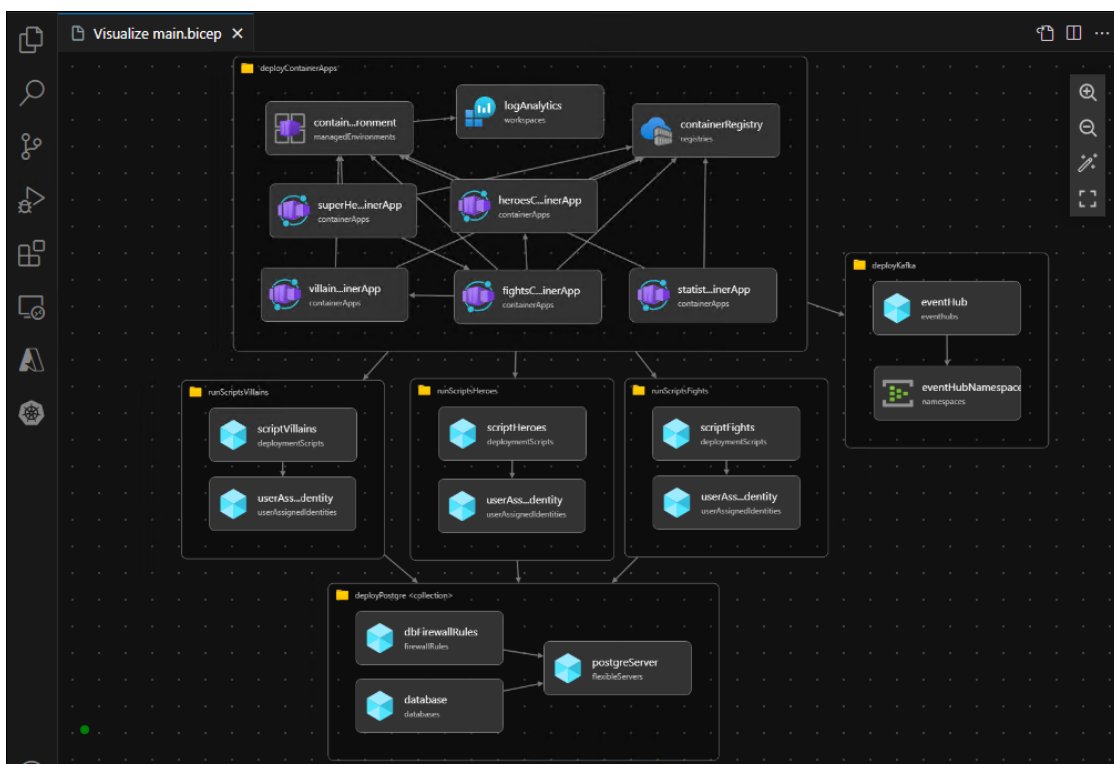
16. [Visual Studio Code]의 [탐색기]에서 "main.bicep" 파일을 열고 다음과 같은 내용을 검토합니다.

- 각 모듈 배포에 필요한 매개 변수와 Container Apps 배포에 필요한 변수가 선언되어 있습니다.
- 동일한 설정으로 배포되는 PostgreSQL 모듈을 배포하기 위해 for 구문이 사용됩니다.
- PostgreSQL이 배포된 후 테이블을 만들고 데이터를 추가하기 위해 Bicep 배포 스크립트 모듈이 사용됩니다.
- Event Hub를 배포하기 위해 모듈이 사용되며 이 모듈의 output이 ContainerApps 모듈의 매개 변수로 사용됩니다.
- Container Apps 배포를 위해 모듈이 사용되며 각 Container Apps 배포에 필요한 매개 변수는 main.bicep 파일의 매개 변수와 다른 모듈의 output을 통해 처리됩니다.

- `output` 영역에 PostgreSQL 서버 이름과 데이터베이스 이름이 출력됩니다.



17. [Visual Studio Code]의 Bicep Visualizer 아이콘을 클릭하면 아래와 같은 배포 시각화를 확인할 수 있습니다. Bicep Visualizer를 통해 각 모듈의 종속성 관계를 확인할 수 있습니다.



18. [Visual Studio Code]에서 WSL 터미널을 열고 다음 명령을 실행하여 Bicep 배포의 `what-if` 검사를 수행합니다.

- `adminPassword` 매개 변수에는 사용할 암호를 입력하고 `acrName`은 앞서 Azure CLI로 만들었던 Azure Container Registry의 이름을 입력합니다.
- 명령 실행 결과를 확인하고 아무런 오류가 출력되지 않는 것을 확인합니다.

```

# what-if 테스트
az deployment group what-if -g $resourceGroup -f main.bicep

```

```

labuser@labVM:/mnt/c/ContainerApps/TASK07$ # what-if 테스트
labuser@labVM:/mnt/c/ContainerApps/TASK07$ az deployment group what-if -g $resourceGroup -f main.bicep
Please provide securestring value for 'adminPassword' (? for help):
Please provide string value for 'acrName' (? for help): kormttacr1109
Note: The result may contain false positive predictions (noise).
You can help us improve the accuracy of the result by opening an issue here: https://aka.ms/WhatIfIssues

Resource and property changes are indicated with these symbols:
+ Create
* Ignore

The deployment will update the following scope:

Scope: /subscriptions/5464dc1f-7515-47ff-8b47-d7c6b2bc10ef/resourceGroups/02_containerAppsRg

+ Microsoft.DBforPostgreSQL/flexibleServers/fights-db-jxpty6lchpppg [2022-12-01]

    apiVersion: "2022-12-01"
    id: "/subscriptions/5464dc1f-7515-47ff-8b47-d7c6b2bc10ef/resourceGroups/02_containerAppsRg/providers/Microsoft.DBforPostgreSQL/flexibleServers/fights-db-jxpty6lchpppg"
    location: "koreacentral"
  
```

19. WSL 터미널에서 다음 명령을 실행하여 Azure Deployment Stack으로 전체 마이크로서비스를 배포합니다.

`adminPassword` 매개 변수와 `acrName` 매개 변수는 `what-if`에서 사용했던 값을 그대로 사용합니다. 전체 배포에 대략 10분 정도가 소요됩니다. 표시되는 경고는 Bicep 파일에서 암호를 보안 문자열로 처리하지 않았기 때문에 표시되는 경고이며 실습에서는 무시할 수 있습니다.

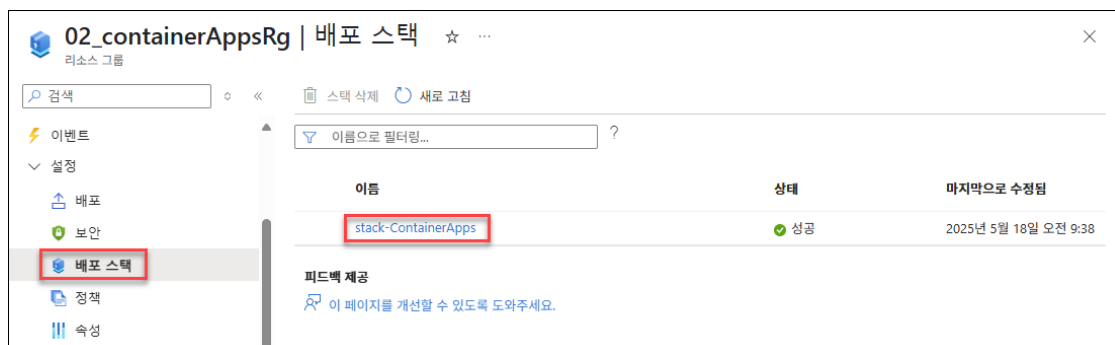
```

# 전체 마이크로서비스 배포
az deployment group create -n deployContainerApps -g $resourceGroup \
-f main.bicep
  
```

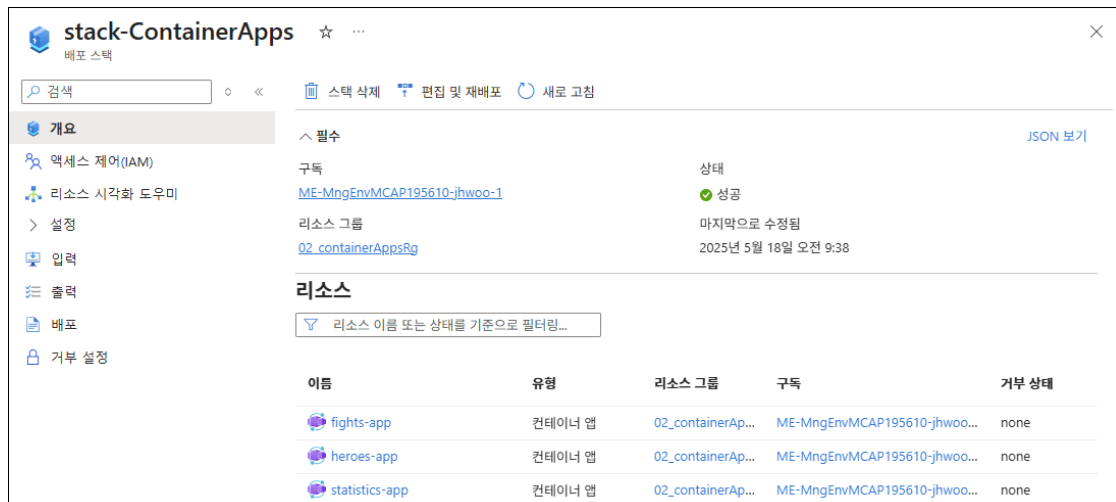
```

labuser@labVM:/mnt/c/02_containerApps$ # Deployment Stack으로 배포
labuser@labVM:/mnt/c/02_containerApps$ az stack group create --name stack-ContainerApps \
--resource-group $resourceGroup --template-file main.bicep \
--action-on-unmanage detachAll \
--deny-settings-mode none
/mnt/c/02_containerApps/modules/containerApps.bicep(47,20) : Warning use-secure-value-for-secure-inputs: Property 'sharedKey' expects a secure value, but the value provided may not be secure. [https://aka.ms/bicep/linter/use-secure-value-for-secure-inputs]
/mnt/c/02_containerApps/modules/containerApps.bicep(77,48) : Warning use-secure-value-for-secure-inputs: Property 'value' expects a secure value, but the value provided may not be secure. [https://aka.ms/bicep/linter/use-secure-value-for-secure-inputs]
/mnt/c/02_containerApps/modules/containerApps.bicep(146,18) : Warning use-secure-value-for-secure-inputs: Property 'value' expects a secure value, but the value provided may not be secure. [https://aka.ms/bicep/linter/use-secure-value-for-secure-inputs]
/mnt/c/02_containerApps/modules/containerApps.bicep(215,18) : Warning use-secure-value-for-secure-inputs: Property 'value' expects a secure value, but the value provided may not be secure. [https://aka.ms/bicep/linter/use-secure-value-for-secure-inputs]
/mnt/c/02_containerApps/modules/containerApps.bicep(280,18) : Warning use-secure-value-for-secure-inputs: Property 'value' expects a secure value, but the value provided may not be secure. [https://aka.ms/bicep/linter/use-secure-value-for-secure-inputs]
/mnt/c/02_containerApps/modules/containerApps.bicep(373,18) : Warning use-secure-value-for-secure-inputs: Property 'value' expects a secure value, but the value provided may not be secure. [https://aka.ms/bicep/linter/use-secure-value-for-secure-inputs]
Please provide securestring value for 'adminPassword' (? for help):
Please provide string value for 'acrName' (? for help): kormtt0516acr
Running ..
  
```

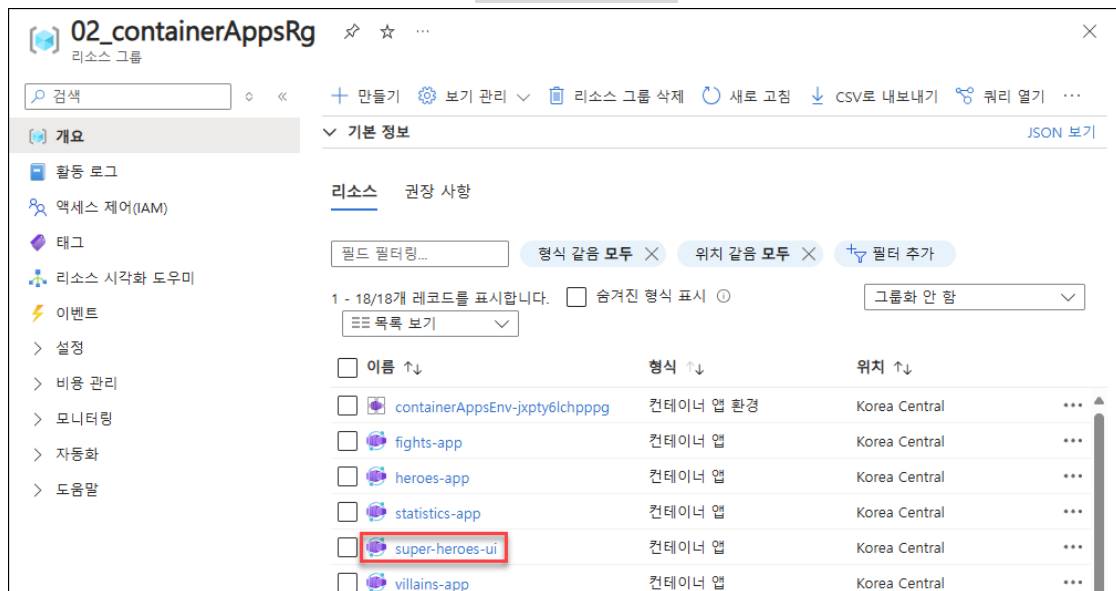
20. Azure 포털로 이동한 후 `[02_containerAppsRg 리소스 그룹]` 블레이드로 이동합니다. `[02_containerAppsRg 리소스 그룹]` 블레이드의 `[설정 - 배포 스택]`로 이동한 후 배포 스택 이름을 클릭합니다.



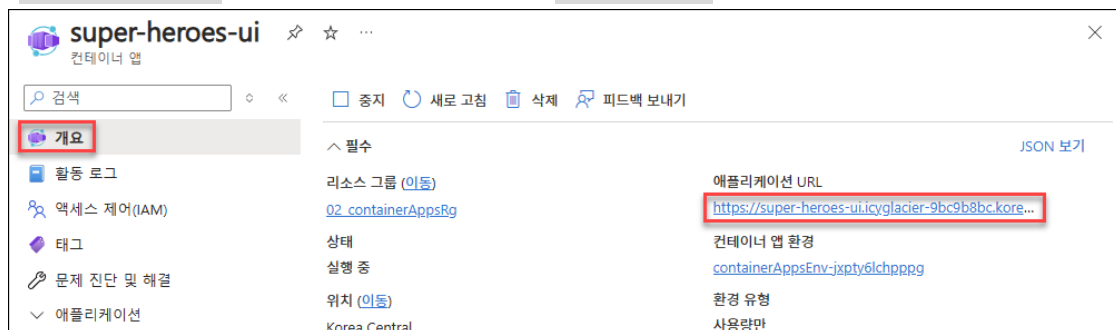
21. 아래와 같이 배포가 모두 성공적으로 완료된 것을 확인합니다. 각 배포를 클릭하여 Bicep에서 구성한 출력 내용이 표시되는지 검토합니다.



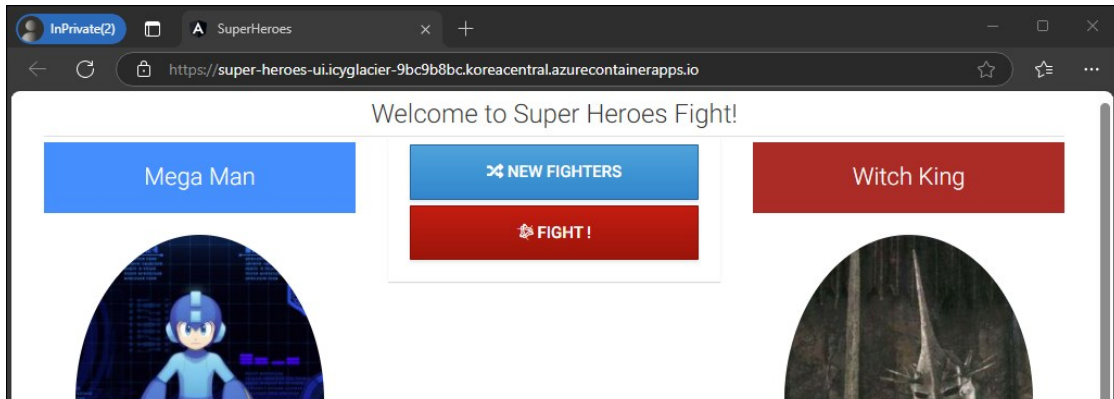
22. [02\_containerAppsRg 리소스 그룹] 블레이드의 [개요]로 이동한 후 컨테이너 앱 환경, 컨테이너 앱, 컨테이너 레지스트리, 배포 스크립트, 관리 ID, Log Analytics 작업 영역, Event Hub 네임스페이스, Azure Database for PostgreSQL 유연한 서버 리소스가 모두 배포된 것을 확인합니다. 테스트를 위해 **super-heroes-ui** 컨테이너 앱 리소스를 클릭합니다.



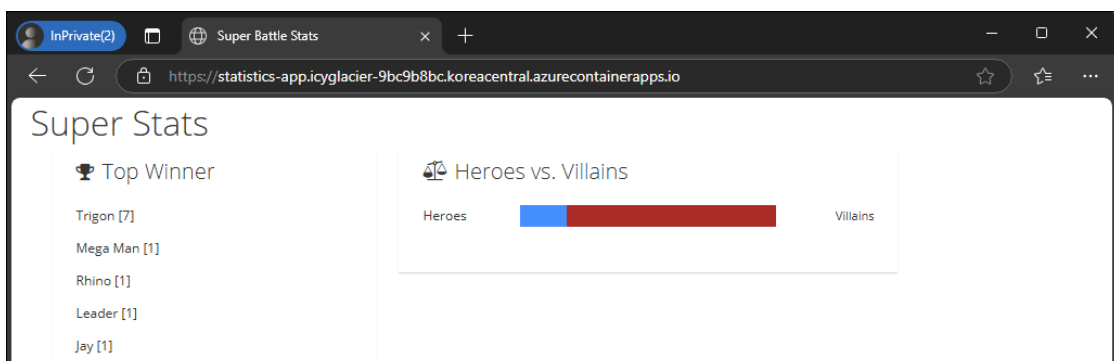
23. [super-heroes-ui 컨테이너 앱] 블레이드의 [개요]에서 "애플리케이션 URL"의 링크를 클릭합니다.



24. 아래와 같이 Super Heroes Fight 앱이 표시되는 것을 확인합니다. [NEW FIGHTERS]와 [FIGHT!] 버튼을 클릭하여 애플리케이션이 정상적으로 작동하는지 확인합니다.



25. 동일한 방법으로 `statistics-app` 컨테이너 앱의 "애플리케이션 URL"을 클릭하여 앱을 시작하고 통계 정보가 표시되는지 확인합니다.



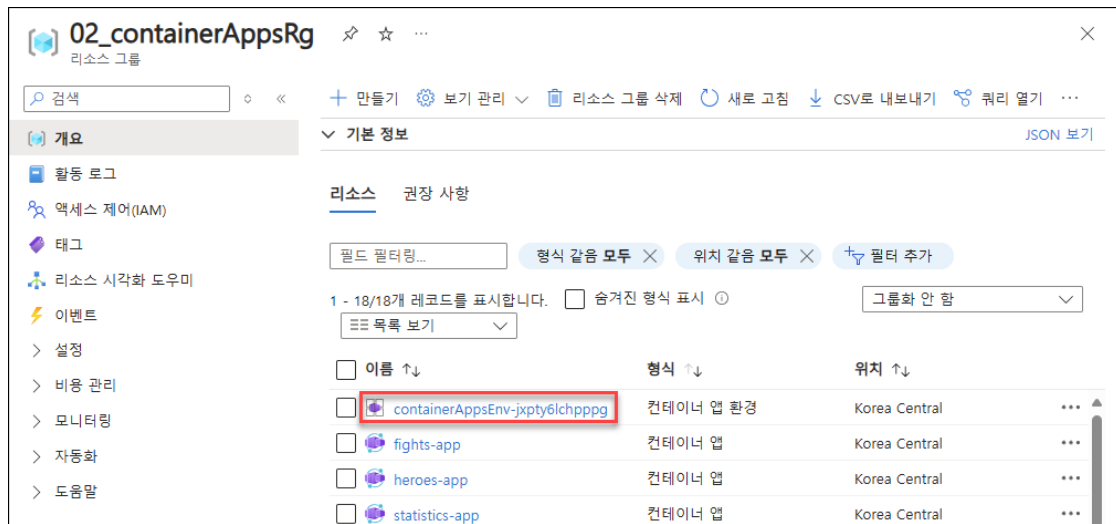
## EXERCISE 02. Log Analytics로 로그 확인

Azure Container Apps는 다음과 같은 두 가지 유형의 애플리케이션 로깅을 제공합니다.

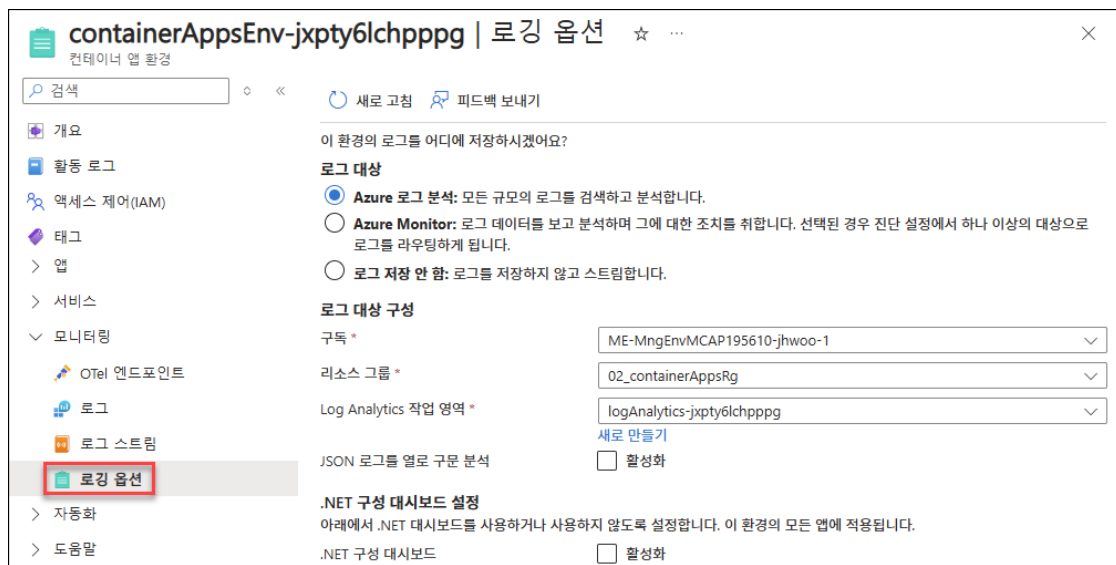
- 컨테이너 콘솔 로그: 컨테이너 콘솔의 로그 스트림입니다. 로그 스트림은 Azure 포털이나 CLI에서 거의 실시간으로 확인할 수 있습니다. Container Apps는 애플리케이션 컨테이너에서 `stdout` 및 `stderr` 출력 스트림을 캡처하고 이를 콘솔 로그로 보여줍니다.
- 시스템 로그: Azure Container Apps 서비스에서 생성된 로그입니다. 시스템 로그는 서비스 수준의 이벤트 상태를 보여주며 Dapr, 볼륨, 도메인, 인증, 리비전 등 다양한 정보를 보여줍니다.

이 작업에서는 여러가지 로깅 옵션을 검토하고 Container Apps에서 발생한 로그를 쿼리하는 방법에 대해 실습합니다.

1. `[02_containerAppsRg` 리소스 그룹] 블레이드에서 컨테이너 앱 환경 리소스를 클릭합니다.



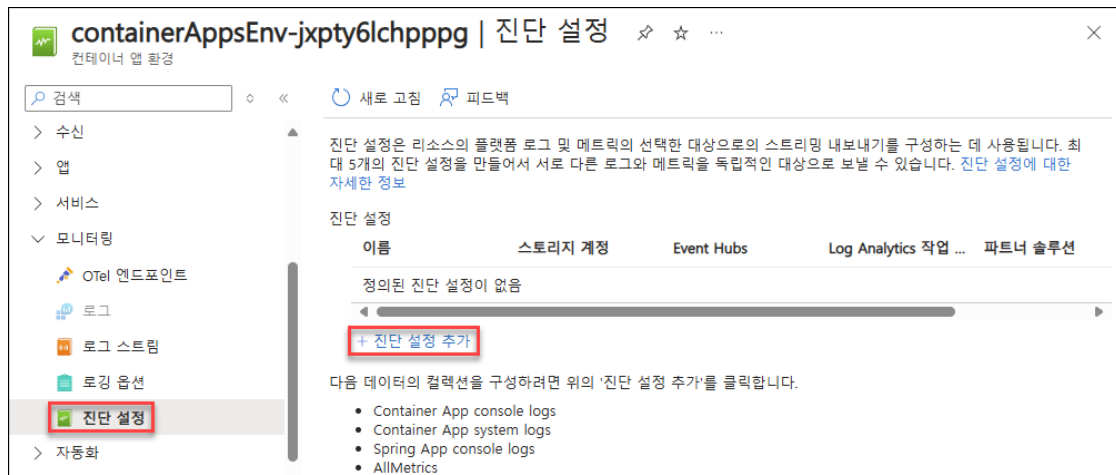
2. [컨테이너 앱 환경] 블레이드의 [모니터링 - 로깅 옵션]으로 이동합니다. 기본 설정으로 "Azure 로그 분석(Log Analytics)"이 선택되어 있는 것을 확인할 수 있습니다. 이 옵션에서는 Container Apps의 모든 로그가 Log Analytics 작업 영역으로 전송됩니다.



3. [컨테이너 앱 환경 | 로깅 옵션]에서 로깅 옵션을 "Azure Monitor"로 변경하고 [저장]을 클릭합니다.



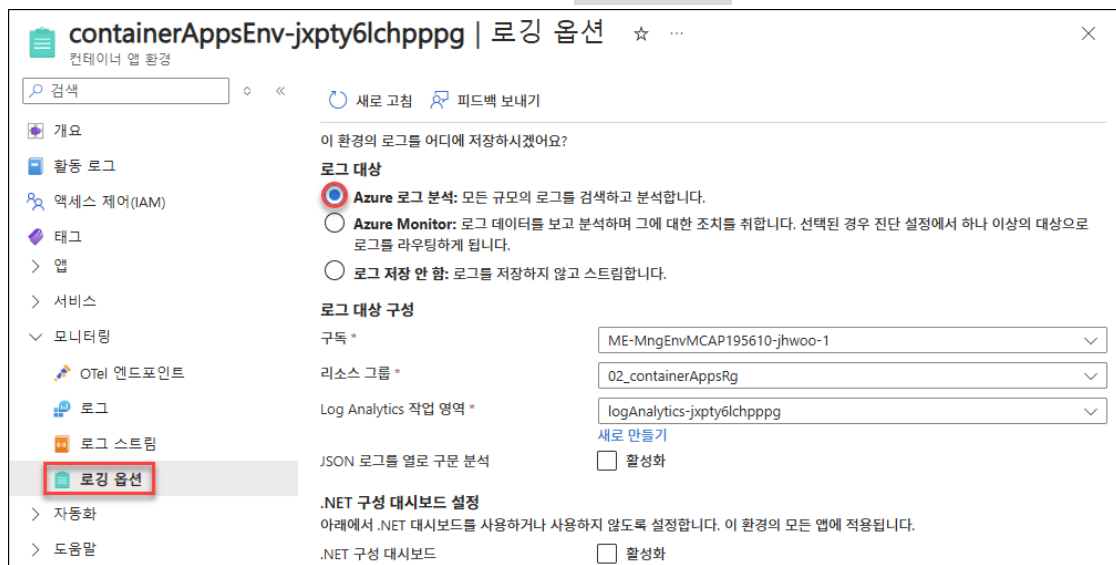
4. 로깅 옵션을 "Azure Monitor"로 변경하면 컨테이너 앱 환경에 "진단 설정" 메뉴가 추가됩니다. [모니터링 - 진단 설정]으로 이동한 후 "진단 설정 추가" 링크를 클릭합니다.



5. [진단 설정] 블레이드에서 아래와 같이 범주 그룹, 범주, 메트릭을 선택하고 이를 Log Analytics 뿐 아니라 스토리지 계정에 저장하거나 Event Hub를 통해 다른 모니터링 시스템으로 스트리밍하는 옵션이 제공됩니다. 이 실습에서는 기본 로깅 옵션을 사용할 것이기 때문에 [진단 설정] 블레이드에서 아무런 구성도 하지 않습니다.

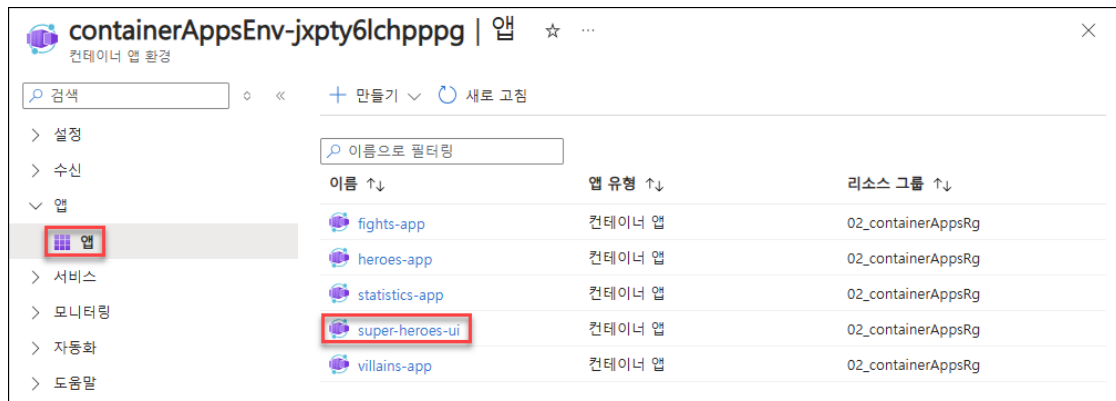


6. 다시 [컨테이너 앱 환경 | 로깅 옵션]으로 이동한 후 로깅 옵션을 "Azure 로그 분석"으로 선택하고 [저장]을 클릭합니다.

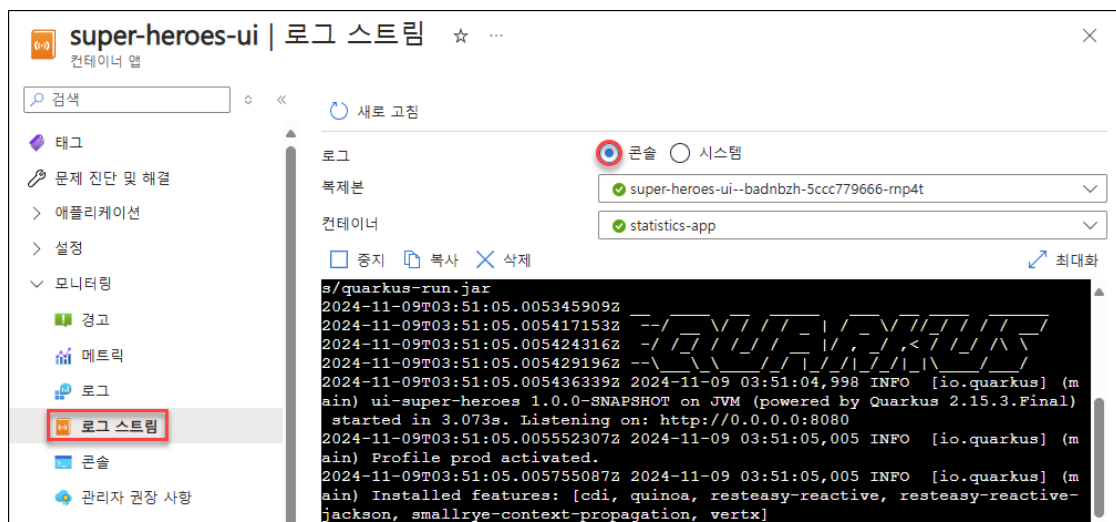


7. [containerAppsEnv 컨테이너 앱 환경] 블레이드의 [앱 - 앱]으로 이동한 후 super-heroes-ui 컨테이너 앱을 클릭합니다.

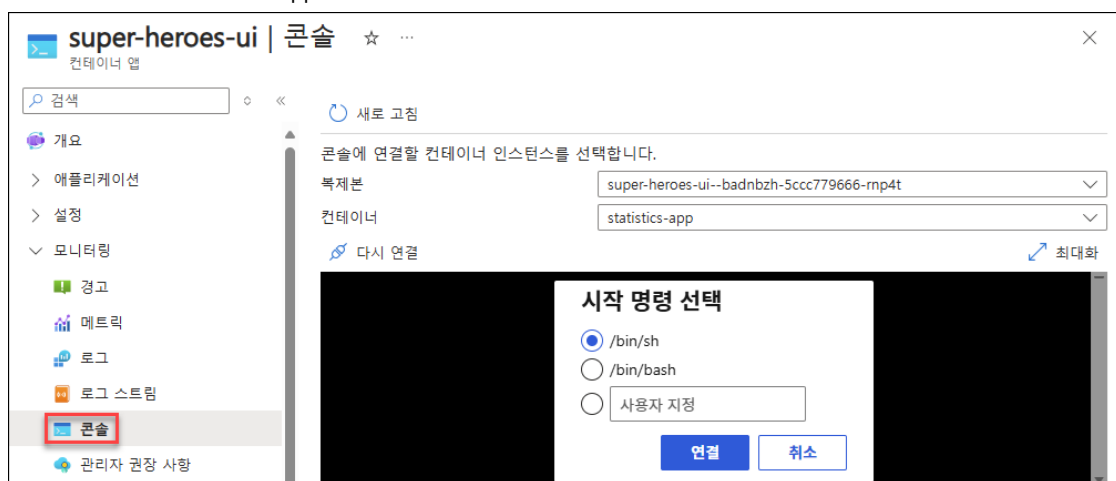




8. [super-heroes-ui 컨테이너 앱] 블레이드의 [모니터링 - 로그 스트림]으로 이동합니다. 로그를 "콘솔"로 선택하면 아래와 같이 Container App의 현재 콘솔 로그가 출력되는 것을 확인할 수 있습니다. 로그를 "시스템"으로 선택하면 Container App에서 실행 중인 모든 컨테이너의 시스템 로그를 볼 수 있습니다.

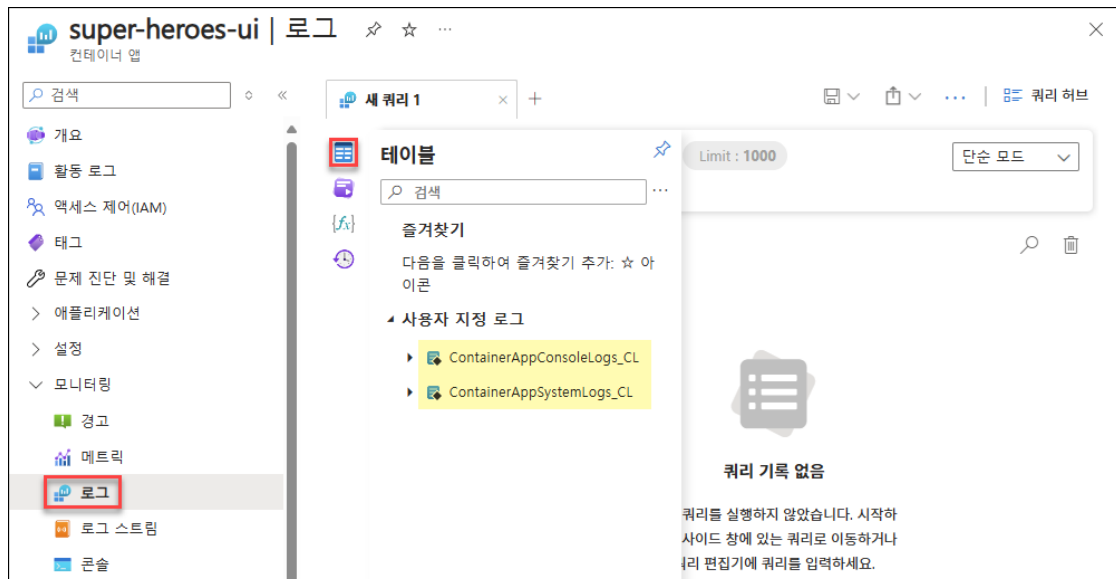


9. [super-heroes-ui 컨테이너 앱] 블레이드의 [모니터링 - 콘솔]로 이동합니다. 시작 명령을 "/bin/bash"로 선택한 후 [연결]을 클릭하면 Container Apps에서 실행 중인 컨테이너에 직접 연결할 수 있습니다.



10. [super-heroes-ui 컨테이너 앱] 블레이드의 [모니터링 - 로그]로 이동합니다. "스키마 및 필터" 영역의 [테이블] 탭에서 "사용자 지정 로그"를 확장하면 아래와 같이 ContainerAppConsoleLogs\_CL, ContainerAppSystemLogs\_CL 테이블이 표시되는 것을 확인할 수 있습니다. Container Apps의 로그가 Log Analytics에 기록될 때까지 몇 분의 시간이 소요될 수 있습니다.





11. Log Analytics의 쿼리 창에 다음과 같은 쿼리를 입력한 후 [실행]을 클릭합니다. `super-heroes-ui` Container App의 콘솔 로그가 출력되는 것을 확인할 수 있습니다. 이 로그를 통해 컨테이너가 정상적으로 실행되지 않는 경우 문제 해결을 할 수 있습니다.

```
// super-heroes-ui의 콘솔 로그 확인
ContainerAppConsoleLogs_CL
| where ContainerAppName_s == 'super-heroes-ui'
| project ContainerAppName_s, Log_s, TimeGenerated
```

