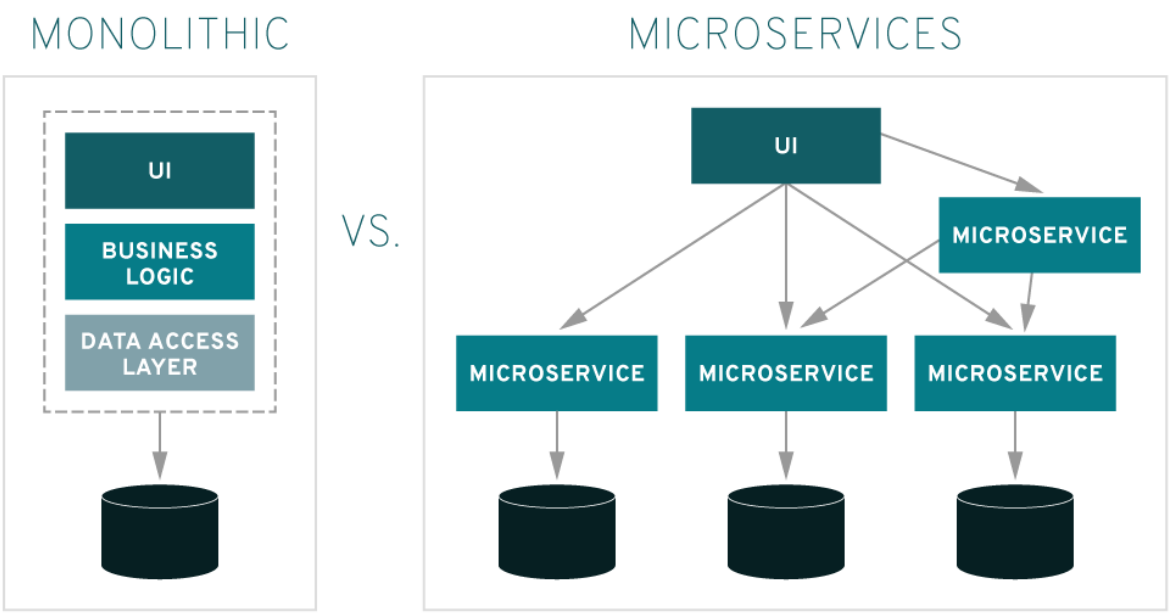
마이크로 서비스 아키텍처



마이크로서비스란 개발 어플리케이션을 하나로 묶어서 개발하지 않고 서비스들을 경량화하여 작은 단위로 개발하여 필요한 기능을 호출할 수 있도록 하는 구조를 말한다. 즉 독립적인 서비스들의 모음이 경량화 API를 통해 통신하는 애플리케이션 아키텍처의 한 유형이다.

더욱 효율적인 애플리케이션 개발 방식

마이크로서비스 아키텍처는 하나의 애플리케이션 내에 있는 각 핵심 기능이 독립적으로 존재할 수 있도록 소프트웨어를 구축하는 클라우드 네이티브 접근 방식이다.

애플리케이션의 요소들이 이러한 방식으로 분리되면 개발 팀과 운영 팀이 서로 방해가 되지 않으면서 협력할 수 있다. 따라서 동일한 애플리케이션 개발에 더 많은 개발자들이 동시 참여할 수 있으므로 개발에 소요되는 시간을 단축할 수 있다.

모놀리식 아키텍처와 마이크로서비스 아키텍처 비교

애플리케이션 빌드의 전통적인 방식은 모놀리식 위주였다. 모놀리식 아키텍처에서는 애플리케이션 내의 모든 기능과 서비스가 함께 맞물려 단일 유닛으로 운영된다. 애플리케이션이 어떤 방식으로든 추가되거나 개선되면 아키텍처는 더 복잡해진다. 그 결과 전체 애플리케이션을 분리하지 않고는 애플리케이션 내의 단일 기능을 최적화하기가 더 어려워지고, 만약 애플리케이션 내의 하나의 프로세스가 확장되어야 할 때는 전체 애플리케이션도 확장되어야 한다.

마이크로서비스 아키텍처에서 애플리케이션은 애플리케이션 내의 각 핵심 기능이 독립적으로 작동하도록 빌드된다. 따라서 개발 팀은 애플리케이션 전체를 중단하지 않고 변화하는 비즈니스 요구 사항을 충족하기 위한 새로운 구성 요소를 구축하고 업데이트할 수 있다.

서비스 지향 아키텍처와 마이크로서비스 아키텍처 비교

마이크로서비스 아키텍처는 서비스 지향 아키텍처(SOA)가 진화한 형태다. 이 두 가지 방식은 크고 복잡한 애플리케이션을 작업하기 쉬운 작은 구성 요소들로 세분화한다는 점에서 유사하다. 이러한 유사성 때문에 SOA와 마이크로서비스 아키텍처는 혼동하기 쉽다. 이 둘의 가장 큰 차이점은 범위에 있다. SOA가 전사적인 아키텍처 접근 방식이라면 마이크로서비스는 애플리케이션 개발팀의 구현 전략이다.

마이크로서비스 아키텍처의 장점

마이크로서비스는 분산형 개발을 통해 팀의 역량과 일상적인 업무 능력을 향상시킨다. 또한, 여러 마이크로서비스를 동시에 개발하는 것도 가능하다. 따라서 동일한 애플리케이션 개발에 더 많은 개발자들이 동시 참여할 수 있으므로 개발에 소요되는 시간을 단축할 수 있다.

더 빠른 출시

개발 주기가 단축되기 때문에 마이크로서비스 아키텍처는 더욱 민첩한 배포와 업데이트를 지원한다.

높은 확장성

특정 서비스에 대한 수요가 증가하면 필요에 따라 여러 서버와 인프라에 배포할 수 있다.

뛰어난 복구 능력

이러한 독립적인 서비스들은 제대로 구축되기만 한다면 서로에게 영향을 주지 않는다. 즉, 모놀리식 애플리케이션 모델과는 달리 한 부분에 장애가 발생하더라도 전체 애플리케이션이 중단되지 않는다.

손쉬운 배포

마이크로서비스 기반 애플리케이션은 전통적인 모놀리식 애플리케이션에 비해 더욱 모듈화되고 규모가 작아졌기 때문에 배포에 따르는 우려 사항들이 사라졌다. 이를 위해서는 더 많은 협업이 필요하며 서비스 메쉬 레이어가 협업에 도움이 될 수 있다. 그리고 이를 통해 몇 배로 성과를 거둘 수 있다.

편리한 액세스

하나의 큰 애플리케이션을 작은 부분들로 세분화했기 때문에 개발자들이 각 부분을 파악하고 업데이트하며 개선하기가 용이해졌다.

마이크로서비스와 관련된 잠재적 과제

마이크로서비스의 유연성 때문에 서둘러 새로운 변경 사항들을 배포하려고 하다 보면 새로운 패턴들이 생성된다. 소프트웨어 엔지니어링에서 "패턴"은 작동한다고 알려진 모든 알고리즘 솔루션을 가리킨다. 반대로 "안티패턴"이란 문제를 해결하려는 의도에서 발생하지만 장기적으로는 더 많은 문제를 발생시킬 수 있는 일반적인 실수를 말한다.

문화와 프로세스 이외에도 복잡성과 효율성은 마이크로서비스 기반 아키텍처의 두 가지 주요 과제다. 마이크로서비스 아키텍처로 작업할 때는 이러한 일반적인 안티패턴에 주의해야 한다.

확장: 소프트웨어 라이프사이클 개발 프로세스 내의 기능을 확장하는 것은 특히 초기에 문제가 될 수 있다. 초기 설정 과정에서는 시간을 들여 서비스들 간의 종속성을 파악하고 역호환성을 손상시킬 수 있는 잠재적 트리거를 인지하는 것이 중요하다. 그리고 배포 시기가 되면 마이크로서비스가 수동으로 배포하기에는 지나치게 복잡해졌기 때문에 자동화에 투자하는 것이 매우 중요하다.

로깅: 분산 시스템에서는 모든 내용을 한 곳에 모을 수 있는 중앙집중식 로그가 필요하다. 그렇지 않으면 확장 시에 이를 관리하기가 불가능해진다.

모니터링: 문제의 근원을 정확히 집어내려면 시스템을 중앙에서 파악하는 것이 중요하다.

디버깅: 로컬 통합 개발 환경(IDE)을 통한 원격 디버깅은 선택 사항이 아니다. 이 방식으로는 수십, 수백 개의 서비스를 관리할 수 없다. 안타깝게도 현재로서는 디버그 방법에 대한 정답이 없다.

연결성: 중앙집중식 또는 통합형 여부에 관계없이 서비스 검색을 고려해야 한다.

마이크로서비스를 지원하는 툴과 기술

컨테이너와 도커
컨테이너는 애플리케이션 코드와 이를 실행하는 데 필요한 모든 파일이 함께 패키징된 소프트웨어의 단위를 말한다. 이러한 구조를 통해 전체 기능을 유지하면서 컨테이너화된 애플리케이션을 환경 간에 쉽게 이동할 수 있다.

도커는 컨테이너 생성 및 관리 플랫폼으로, 마이크로서비스 애플리케이션의 관리, 확장, 배포를 자동화하기에 적합하다.

API
API(애플리케이션 프로그래밍 인터페이스)는 다른 애플리케이션과의 통신을 담당하는 애플리케이션의 일부를 말한다. 마이크로서비스 아키텍처의 인프라에서 API는 마이크로서비스 내의 여러 서비스가 정보를 공유하고 하나로 작동할 수 있게 하는 중요한 역할을 한다.

이벤트 스트리밍
이벤트는 마이크로서비스 서비스 내에서 발생하는 모든 것으로 정의될 수 있다. 예를 들어, 누군가 온라인 장바구니에 무언가를 추가하거나 삭제하는 것이 이벤트다.

이벤트는 이벤트 스트림으로 형성되고, 이벤트 스트림은 시스템의 행동 변화를 반영한다. 이벤트를 모니터링하면 조직은 데이터와 사용자 행동에 관한 유용한 결론에 도달할 수 있다. 이벤트 스트림 프로세싱을 적용하면 즉각적인 조치를 취할 수 있다. 이벤트 스트림 프로세싱은 실시간으로 운영 워크로드에 직접 사용될 수 있다. 오늘날 기업들은 이벤트 스트리밍을 사기 분석에서 머신 유지 관리까지 모든 것에 적용하고 있다.

서버리스 컴퓨팅
서버리스 컴퓨팅은 개발자는 애플리케이션을 빌드, 실행할 수 있고 클라우드 공급업체는 서버 인프라의 프로비저닝, 유지 관리, 확장을 담당하는 클라우드 네이티브 개발 모델이다. 개발자는 배포를 위해 코드를 컨테이너에 패키징하기만 하면 된다. 서버리스의 경우 애플리케이션이 기본 인프라에서 추상화되므로 조직이 더 빠르게 혁신할 수 있다.

<docker swarm 으로 msa 환경 구축>

1. ec2 인스턴스 3개 생성(모두 10GB)
   인바운드 규칙에서 다음 포트 개방
   2377/tcp : 클러스터 관리에 사용되는 포트
   7946/tcp, 7946/udp : 노드 간 통신에 사용
   4789/udp: 클러스터에서 사용되는 Ingress 오버레이 네트워크 트래픽에 사용
   8081, 8082, 3000, 9000: 앱에서 사용

2. 3노드에 도커 설치
ubuntu@ip-172-31-7-221:~$ sudo apt update
ubuntu@ip-172-31-7-221:~$ sudo apt install docker.io
ubuntu@ip-172-31-7-221:~$ sudo apt install docker-compose

3. 매니저 노드로 사용할 노드에서 swarm 시작
ubuntu@ip-172-31-2-139:~$ sudo docker swarm init
Swarm initialized: current node (yappjhwj8omo32bzfvsjb45p4) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-3sk5xmhz54g6yukmihjnba397ddaeb8mjwrvpj6w5a7r0hp1gb-7ex7wkq9hi4pb6ddksw1hzd8z 172.31.2.139:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

4. 노드 목록 확인

ubuntu@ip-172-31-2-139:~$ sudo docker node ls

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS ENGINE VERSION

yappjhwj8omo32bzfvsjb45p4 * ip-172-31-2-139 Ready Active Leader 24.0.7

=> 현재 swam을 시작한 노드 하나만 클러스터에 포함됨. leader로 지정된 것을 확인(매니저 노드)

<노드 추가>

5. 나머지 2노드도 클러스트에 조인

3번 작업 실행 결과 보면 다른 노드를 이 클러스터에 조인하려면 실행하라고 명령어 제시함 이를 복사하여 두 노드에서 각각 실행

$docker swarm join --token SWMTKN-1-3sk5xmhz54g6yukmihjnba397ddaeb8mjwrvpj6w5a7r0hp1gb-7ex7wkq9hi4pb6ddksw1hzd8z 172.31.2.139:2377

ubuntu@ip-172-31-8-106:~$ sudo docker swarm join --token SWMTKN-1-3sk5xmhz54g6yukmihjnba397ddaeb8mjwrvpj6w5a7r0hp1gb-7ex7wkq9hi4pb6ddksw1hzd8z 172.31.2.139:2docker swarm join --token SWMTKN-1-3sk5xmhz54g6yukmihjnba397ddaeb8mjwrvpj6w5a7r0hp1gb-7ex7wkq9hi4pb6ddksw1hzd8z 172.31.2.139:2377

This node joined a swarm as a worker.

ubuntu@ip-172-31-12-243:~$ sudo docker swarm join --token SWMTKN-1-3sk5xmhz54g6yukmihjnba397ddaeb8mjwrvpj6w5a7r0hp1gb-7ex7wkq9hi4pb6ddksw1hzd8z 172.31.2.139:2377docker swarm join --token SWMTKN-1-3sk5xmhz54g6yukmihjnba397ddaeb8mjwrvpj6w5a7r0hp1gb-7ex7wkq9hi4pb6ddksw1hzd8z 172.31.2.139:2377

This node joined a swarm as a worker.

6. 다시 클러스터 노드 목록 확인.

ubuntu@ip-172-31-2-139:~$ sudo docker node ls

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS ENGINE VERSION

yappjhwj8omo32bzfvsjb45p4 * ip-172-31-2-139 Ready Active Leader

24.0.7
3y1gckdqdc6h4xbbhvl4817gy ip-172-31-8-106 Ready Active
24.0.7
xfs70ipkxb17fyz4i2083464m ip-172-31-12-243 Ready Active
24.0.7

=> 노드가 3개, 리더는 하나

7. 클러스터의 상태 확인. 이 명령은 매니저 노드에서만 실행가능. worker노드는 안됨
ubuntu@ip-172-31-2-139:~$ sudo docker node inspect self --pretty
ID: yappjhwj8omo32bzfvsjb45p4
Hostname: ip-172-31-2-139
Joined at: 2024-11-02 05:26:07.597052756 +0000 utc
Status:
 State: Ready
 Availability: Active
 Address: 172.31.2.139
Manager Status:
 Address: 172.31.2.139:2377
 Raft Status: Reachable
 Leader: Yes
Platform:
 Operating System: linux
 Architecture: x86_64
Resources:
 CPUs: 1
 Memory: 957.4MiB
Plugins:
 Log: awslogs, fluentd, gcplogs, gelf, journald, json-file, local, logentries, splunk, syslog
 Network: bridge, host, ipvlan, macvlan, null, overlay
 Volume: local
Engine Version: 24.0.7

<롤 변경>
- MANAGER STATUS: 매니저 노드들의 상태를 보여줌. 상태에는 다음 3가지가 있고, 이 상태값은 매니저 노드들만 갖는다.
  Leader : 스웜 클러스터의 관리와 오케스트레이션을 관리하는 노드
  Reachable : 매니저 노드로 다른 매니저 노드들과 정상적으로 통신 가능한 상태. 만약 Leader 노드에 장애가 발생하면, 이 상태값을 가진 매니저 노드는 새로운 Leader 노드로 선출 가 능

하다.

Unavailable : 매니저 노드로 Leader를 포함한 다른 매니저 노드들과 통신이 불가능한 상태. 이런 경우엔 노드를 다시 복구하거나, 다른 워커 노드를 매니저 노드로 변경하거나, 새 매니저 노드를 추가하는 작업이 필요하다.

- Docker Swarm 클러스터는 1개 이상의 매니저 노드를 가질 수 있다. 리더(Leader)로 선별된 매니저 노드가 전체 클러스터를 실질적으로 관리한다.
클러스터의 모든 변경 사항은 리더 노드를 통해 전파되고 나머지 노드들은 리더 노드와 동기화된 상태를 유지한다.

8. 매니저 노드 추가

promote 명령으로 특정 노드를 매니저 노드로 변경

리더 노드에 문제가 발생하면 다른 매니저 노드가 리더가 됨

클러스터 제어 명령은 매니저 노드에서만 가능

```
ubuntu@ip-172-31-2-139:~$ sudo docker node promote ip-172-31-8-106
Node ip-172-31-8-106 promoted to a manager in the swarm.
```

9. 다시 노드 목록을 확인하면 promote한 노드의 manager status가 reachable로 바뀐것 확인

```
ubuntu@ip-172-31-2-139:~$ sudo docker node ls
ID                            HOSTNAME                STATUS    AVAILABILITY     MANAGER STATUS     ENGINE VERSION
yappjhwj8omo32bzfvsjb45p4 *   ip-172-31-2-139         Ready     Active           Leader
24.0.7
3y1gckdqdc6h4xbbhvl4817gy     ip-172-31-8-106         Ready     Active           Reachable
24.0.7
xfs70ipkxb17fyz4i2083464m     ip-172-31-12-243        Ready     Active
24.0.7
```

10. 매니저에서 워커로 노드 변환 - demote

```
ubuntu@ip-172-31-2-139:~$ sudo docker node demote ip-172-31-8-106
Manager ip-172-31-8-106 demoted in the swarm.
ubuntu@ip-172-31-2-139:~$ sudo docker node ls
ID                            HOSTNAME                STATUS    AVAILABILITY     MANAGER STATUS     ENGINE VERSION
yappjhwj8omo32bzfvsjb45p4 *   ip-172-31-2-139         Ready     Active           Leader
24.0.7
3y1gckdqdc6h4xbbhvl4817gy     ip-172-31-8-106         Ready     Active
24.0.7
xfs70ipkxb17fyz4i2083464m     ip-172-31-12-243        Ready     Active
```

24.0.7

11. [update --role manager/worker] 명령으로도 롤을 변경할 수 있다.

```
ubuntu@ip-172-31-2-139:~$ sudo docker node update --role manager ip-172-31-12-243
ip-172-31-12-243
ubuntu@ip-172-31-2-139:~$ sudo docker node ls
ID                            HOSTNAME              STATUS    AVAILABILITY     MANAGER
STATUS    ENGINE VERSION
yappjhwj8omo32bzfvsjb45p4 *   ip-172-31-2-139       Ready     Active           Leader
24.0.7
3y1gckdqdc6h4xbbhvl4817gy     ip-172-31-8-106       Ready     Active
24.0.7
xfs70ipkxb17fyz4i2083464m     ip-172-31-12-243      Ready     Active           Reachable
24.0.7
ubuntu@ip-172-31-2-139:~$ sudo docker node update --role worker ip-172-31-12-243
ip-172-31-12-243
ubuntu@ip-172-31-2-139:~$ sudo docker node ls
ID                            HOSTNAME              STATUS    AVAILABILITY     MANAGER
STATUS    ENGINE VERSION
yappjhwj8omo32bzfvsjb45p4 *   ip-172-31-2-139       Ready     Active           Leader
24.0.7
3y1gckdqdc6h4xbbhvl4817gy     ip-172-31-8-106       Ready     Active
24.0.7
xfs70ipkxb17fyz4i2083464m     ip-172-31-12-243      Ready     Active
24.0.7
```

<노드 상태 변경>

- AVAILABILITY : 노드 상태 속성. 다음 3가지가 있고 매니저, 워커 모두 상태를 갖는다.
  Active : 정상적으로 태스크를 할당받을 수 있는 상태
  Pause : 이 상태의 노드에는 스케줄러가 새로운 태스크를 할당을 하지 않는다. 그러나 해당 노드에서 돌아가는 태스크는 구동 상태를 그대로 유지.
  Drain : 이 상태의 노드에는 스케줄러가 새로운 태스크를 할당을 하지 않는다. 해당 노드에서 돌아가던 태스크들은 모두 종료되며 가용 상태(Active)인 다른 노드로 다시 스케줄링 된다.

12. 리더 노드를 drain 상태로 변경

```
ubuntu@ip-172-31-2-139:~$ sudo docker node update --availability drain ip-172-31-2-139
ip-172-31-2-139
ubuntu@ip-172-31-2-139:~$ sudo docker node ls
ID                            HOSTNAME              STATUS    AVAILABILITY     MANAGER
```

STATUS ENGINE VERSION
yappjhwj8omo32bzfvsjb45p4 * ip-172-31-2-139 Ready Drain Leader
24.0.7
3y1gckdqdc6h4xbbhvl4817gy ip-172-31-8-106 Ready Active
24.0.7
xfs70ipkxb17fyz4i2083464m ip-172-31-12-243 Ready Active Reachable
24.0.7

=> 매니저 노드는 기본적으로 워커 노드와 똑같이 태스크를 할당 받아 실행할 수 있다. 그러나 drain 상태의 매니저 노드는 새로운 태스크 할당을 받지 않고, 기존에 할당 받았던 태스크도 다른 노드로 옮긴다. 결국 해당 노드는 클러스터 관리 기능만 가능. 단, 서비스(Service)로 구성하지 않고 docker run 또는 docker compose up 명령으로 직접 구동시킨 컨테이너들은 Drain 상태에서도 종료되지 않고 그대로 남는다. 이들은 스웜 모드(Swarm Mode)에서 관리되지 않으며 해당 호스트에 개별적으로 남아있게 된다.

< 노드 라벨링 >
노드에는 필요에 따라 라벨을 지정할 수 있다. 라벨은 키=값 또는 키 형태로 부여. 이후에 컨테이너 기반 애플리케이션 구동을 위해 서비스를 클러스터에 배포할 때, 노드에 부여된 라벨 정보를 이용하여 스케줄링 조건을 지정할 수 있다. 예를 들어 redis 서비스를 배포한다고 가정한다면, type=redis라는 라벨이 붙은 노드에만 해당 컨테이너가 돌아가도록 규칙을 만들 수 있다.

13. 노드에 라벨 추가. 세 노드에 공통 라벨로 my-cluster, 각 노드별 타입 값을 추가한다.
ubuntu@ip-172-31-2-139:~$ sudo docker node update --label-add my-cluster --label-add type=auth ip-172-31-2-139
ip-172-31-2-139
ubuntu@ip-172-31-2-139:~$ sudo docker node update --label-add my-cluster --label-add type=mems ip-172-31-8-106
ip-172-31-8-106
ubuntu@ip-172-31-2-139:~$ sudo docker node update --label-add my-cluster --label-add type=board ip-172-31-12-243
ip-172-31-12-243

14. 추가된 라벨 확인
ubuntu@ip-172-31-2-139:~$ sudo docker node inspect self --pretty
ID: yappjhwj8omo32bzfvsjb45p4
Labels:
 - my-cluster
 - type=auth
Hostname: ip-172-31-2-139

...
ubuntu@ip-172-31-2-139:~$ sudo docker node inspect ip-172-31-8-106 --pretty
ID: 3y1gckdqdc6h4xbbhvl4817gy
Labels:
 - my-cluster
 - type=mems
Hostname: ip-172-31-8-106
...
ubuntu@ip-172-31-2-139:~$ sudo docker node inspect ip-172-31-12-243 --pretty
ID: xfs70ipkxb17fyz4i2083464m
Labels:
 - my-cluster
 - type=board
Hostname: ip-172-31-12-243
...

15. 추가한 라벨 삭제
ubuntu@ip-172-31-2-139:~$ sudo docker node update --label-rm my-cluster ip-172-31-12-243
ip-172-31-12-243
ubuntu@ip-172-31-2-139:~$ sudo docker node inspect ip-172-31-12-243 --pretty
ID: xfs70ipkxb17fyz4i2083464m
Labels:
 - type=board
Hostname: ip-172-31-12-243
...

<노드 제거>
매니저 노드라면 워커로 전환
상태를 drain으로 전환하여 태스크를 다른 노드로 전환
삭제할 노드에서 leave로 cluster에서 연결 끊음
매니저 노드에서 노드 삭제

- 삭제하려는 노드가 매니저라서 워커로 전환
ubuntu@ip-172-31-2-139:~$ sudo docker node demote ip-172-31-12-243
Manager ip-172-31-12-243 demoted in the swarm.
ubuntu@ip-172-31-2-139:~$ sudo docker node ls
ID   HOSTNAME   STATUS   AVAILABILITY   MANAGER STATUS   ENGINE VERSION
yappjhwj8omo32bzfvsjb45p4 *   ip-172-31-2-139   Ready   Active   Leader

24.0.7
3y1gckdqdc6h4xbbhvl4817gy ip-172-31-8-106 Ready Active
24.0.7
xfs70ipkxb17fyz4i2083464m ip-172-31-12-243 Ready Active
24.0.7

- 삭제할 노드 상태를 drain으로 전환
ubuntu@ip-172-31-2-139:~$ sudo docker node update --availability drain ip-172-31-12-243
ip-172-31-12-243
ubuntu@ip-172-31-2-139:~$ sudo docker node ls
ID HOSTNAME STATUS AVAILABILITY MANAGER
STATUS ENGINE VERSION
yappjhwj8omo32bzfvsjb45p4 * ip-172-31-2-139 Ready Active Leader
24.0.7
3y1gckdqdc6h4xbbhvl4817gy ip-172-31-8-106 Ready Active
24.0.7
xfs70ipkxb17fyz4i2083464m ip-172-31-12-243 Ready Drain
24.0.7

- 삭제할 노드로 이동하여 cluster연결 끊음
ubuntu@ip-172-31-12-243:~$ sudo docker swarm leave
Node left the swarm.

- 매니저 노드에서 노드 삭제
ubuntu@ip-172-31-2-139:~$ sudo docker node rm ip-172-31-12-243
ip-172-31-12-243

- 클러스터 노드 목록 확인(1개 삭제됨)
ubuntu@ip-172-31-2-139:~$ sudo docker node ls
ID HOSTNAME STATUS AVAILABILITY MANAGER
STATUS ENGINE VERSION
yappjhwj8omo32bzfvsjb45p4 * ip-172-31-2-139 Ready Active Leader
24.0.7
3y1gckdqdc6h4xbbhvl4817gy ip-172-31-8-106 Ready Active
24.0.7

==========
<예제>
auth: 인증센터

mem: 사용자 추가 정보
react: front app

1. pc에서 먼저 빌드
세 프로젝트 모두 각 루트에 Dockerfile 작성
<auth 프로젝트>

```
FROM eclipse-temurin:21.0.2_13-jdk-jammy AS builder
#WORKDIR /doc-app1
COPY .mvn/ .mvn
COPY mvnw pom.xml ./
RUN ./mvnw dependency:go-offline
COPY ./src ./src
RUN ./mvnw -Dmaven.test.skip=true clean install

FROM eclipse-temurin:21.0.2_13-jre-jammy AS final
#WORKDIR /doc-app1
COPY --from=builder target/*.jar auth.jar
ENTRYPOINT ["java", "-jar", "auth.jar"]
```

<mem 프로젝트>

```
FROM eclipse-temurin:21.0.2_13-jdk-jammy AS builder
#WORKDIR /doc-app1
COPY .mvn/ .mvn
COPY mvnw pom.xml ./
RUN ./mvnw dependency:go-offline
COPY ./src ./src
RUN ./mvnw -Dmaven.test.skip=true clean install

FROM eclipse-temurin:21.0.2_13-jre-jammy AS final
#WORKDIR /doc-app1
COPY --from=builder target/*.jar mem.jar
ENTRYPOINT ["java", "-jar", "mem.jar"]
```

<react 프로젝트>

```
from node:20
workdir /src
copy package.json .
```

```
run npm install
run npm install react-router-dom
run npm install axios
copy . .
expose 3000
cmd ["npm", "start"]
```

2. 프로젝트들과 동등한 위치(이클립스 웍스페이스)에 폴더 만들고 그 안에 컴포즈(docker-compose.yml) 파일 작성

<docker-compose.yml>

```
version: '3'

services:
  database:
    container_name: mysql_db
    image: mysql:8.0
    restart: always
    environment:
      MYSQL_DATABASE: world
      MYSQL_ROOT_HOST: '%'
      MYSQL_ROOT_PASSWORD: 1234
      TZ: 'Asia/Seoul'
    ports:
     - "3306:3306"
    volumes:
     - ./mysql/conf.d:/etc/mysql/conf.d # MySQL 설정 파일 위치
    command:
     - "mysqld"
     - "--character-set-server=utf8mb4"
     - "--collation-server=utf8mb4_unicode_ci"
    networks:
     - test_network

  myapp1:
     container_name: auth
     depends_on:
          - database
     restart: on-failure
```

```
        environment:
            SPRING_DATASOURCE_URL:
jdbc:mysql://mysql_db:3306/world?useSSL=false&allowPublicKeyRetrieval=true
            SPRING_DATASOURCE_USERNAME: "root"
            SPRING_DATASOURCE_PASSWORD: "1234"
        build:
            context: ../auth/
            dockerfile: Dockerfile
        ports:
            - "8081:8081"
        networks:
            - test_network

    myapp2:
        container_name: mem
        depends_on:
           - database
        restart: on-failure
        environment:
              SPRING_DATASOURCE_URL:
jdbc:mysql://mysql_db:3306/world?useSSL=false&allowPublicKeyRetrieval=true
              SPRING_DATASOURCE_USERNAME: "root"
              SPRING_DATASOURCE_PASSWORD: "1234"
        build:
          context: ../mem/
          dockerfile: Dockerfile
        ports:
          - "8082:8082"
        networks:
          - test_network

networks:
    test_network:
```

3. docker-compose up 명령 실행
   빌드 결과 확인 및 실행해 본다.

4. 생성된 이미지들을 도커 허브에 푸시한다.
- app1 푸시

C:₩Users₩kingn>docker tag myapp-myapp1 kingnuna/msa_app1:v1
C:₩Users₩kingn>docker push kingnuna/msa_app1:v1

- app2 푸시
C:₩Users₩kingn>docker tag myapp-myapp2 kingnuna/msa_app2:v1
C:₩Users₩kingn>docker push kingnuna/msa_app2:v1

- react 앱 빌드 및 푸시
C:₩Users₩kingn₩Desktop₩react₩my-app1>docker build -t reactapp .
C:₩Users₩kingn₩Desktop₩react₩my-app1>docker tag reactapp kingnuna/msa_front:v1
C:₩Users₩kingn₩Desktop₩react₩my-app1>docker push kingnuna/msa_front:v1

5. 컴포즈 파일을 서비스 stack 파일로 변경하여 aws에 올린다.
< service_stack.yml>

```
version: '3'

services:
  database:
    container_name: mysql_db
    image: mysql:8.0
    restart: always
    environment:
      MYSQL_DATABASE: world
      MYSQL_ROOT_HOST: '%'
      MYSQL_ROOT_PASSWORD: 1234
      TZ: 'Asia/Seoul'
    ports:
     - "3306:3306"
    volumes:
     - ./:/etc/mysql/conf.d # MySQL 설정 파일 위치
       #- ./mysql/conf.d:/etc/mysql/conf.d # MySQL 설정 파일 위치
    command:
     - "mysqld"
     - "--character-set-server=utf8mb4"
     - "--collation-server=utf8mb4_unicode_ci"
    deploy:
     mode: global
     placement:
       constraints: [node.labels.type == mems]
```

```
    networks:
     - test_network

  myapp1:
    container_name: auth
    depends_on:
        - database
    restart: on-failure
    environment:
        SPRING_DATASOURCE_URL:
jdbc:mysql://mysql_db:3306/world?useSSL=false&allowPublicKeyRetrieval=true
        SPRING_DATASOURCE_USERNAME: "root"
        SPRING_DATASOURCE_PASSWORD: "1234"
    image: kingnuna/msa_app1:v1
    ports:
        - "8081:8081"
     deploy:
        mode: global
        placement:
          constraints: [node.labels.type == auth]
    networks:
        - test_network

  myapp2:
    container_name: mem
    depends_on:
      - database
    restart: on-failure
    environment:
         SPRING_DATASOURCE_URL:
jdbc:mysql://mysql_db:3306/world?useSSL=false&allowPublicKeyRetrieval=true
         SPRING_DATASOURCE_USERNAME: "root"
         SPRING_DATASOURCE_PASSWORD: "1234"
    image: kingnuna/msa_app3:v1
    deploy:
         mode: global
         placement:
           constraints: [node.labels.type == mems]
    ports:
```

```
      - "8082:8082"
    networks:
      - test_network

  myapp3:
    container_name: react
    image: kingnuna/msa_front:v2
    deploy:
      mode: global
      placement:
        constraints: [node.labels.type == react]
    ports:
     - "3000:3000"
    networks:
     - test_network

networks:
  test_network:
```

C:₩Users₩kingn>scp -i "C:₩Users₩kingn₩Downloads₩ubuntuservkey.pem" "C:₩Users₩kingn₩eclipse-workspace₩service_stack.yml" ubuntu@13.125.228.149:~

6. aws에 배포

ubuntu@ip-172-31-2-139:~$ sudo docker stack deploy -c service_stack.yml

- 스택 목록 확인

ubuntu@ip-172-31-2-139:~$ sudo docker stack ps myweb

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS

zn1sms1ei6v9 myweb_database.1 mysql:8.0 ip-172-31-8-106 Ready Rejected 3 seconds ago "invalid mount config for type..."

o4j1iz391853 ₩_ myweb_database.1 mysql:8.0 ip-172-31-8-106 Shutdown Rejected 8 seconds ago "invalid mount config for type..."

5b8mxnjxaoi5 ₩_ myweb_database.1 mysql:8.0 ip-172-31-8-106 Shutdown Rejected 13 seconds ago "invalid mount config for type..."

r8kxyojhj4qt ₩_ myweb_database.1 mysql:8.0 ip-172-31-8-106 Shutdown Rejected 18 seconds ago "invalid mount config for type..."

c8ovwhmzse2i ₩_ myweb_database.1 mysql:8.0 ip-172-31-8-106

Shutdown Rejected 23 seconds ago "invalid mount config for type..."
wvodj19kn0lg myweb_myapp1.1 kingnuna/msa_app1:v1 ip-172-31-2-139
Running Running less than a second ago
eihpe0j9rjng ₩_ myweb_myapp1.1 kingnuna/msa_app1:v1 ip-172-31-2-139
Shutdown Failed 6 seconds ago "task: non-zero exit (1)"
um4zhl02ksbt ₩_ myweb_myapp1.1 kingnuna/msa_app1:v1 ip-172-31-2-139
Shutdown Failed 25 seconds ago "task: non-zero exit (1)"
uczq0i5slmby ₩_ myweb_myapp1.1 kingnuna/msa_app1:v1 ip-172-31-2-139
Shutdown Failed 44 seconds ago "task: non-zero exit (1)"
hsti1pflfvtb ₩_ myweb_myapp1.1 kingnuna/msa_app1:v1 ip-172-31-2-139 Shutdown
Failed about a minute ago "task: non-zero exit (1)"
oqa3ngbhtrns myweb_myapp2.1 kingnuna/msa_app2:v1 ip-172-31-8-106
Running Running 2 seconds ago
sb56znsobhhd ₩_ myweb_myapp2.1 kingnuna/msa_app2:v1 ip-172-31-8-106
Shutdown Failed 8 seconds ago "task: non-zero exit (1)"
knjbeo6zfgxv ₩_ myweb_myapp2.1 kingnuna/msa_app2:v1 ip-172-31-8-106
Shutdown Failed 29 seconds ago "task: non-zero exit (1)"
xr4wxo1r5ps3 ₩_ myweb_myapp2.1 kingnuna/msa_app2:v1 ip-172-31-8-106
Shutdown Failed 49 seconds ago "task: non-zero exit (1)"
wwmbdk2ui2t8 myweb_myapp3.1 kingnuna/msa_app2:v1 ip-172-31-12-243
Running Running 2 seconds ago
xckwt711c2u1 ₩_ myweb_myapp3.1 kingnuna/msa_app2:v1 ip-172-31-12-243
Shutdown Failed 8 seconds ago "task: non-zero exit (1)"
c2e4o8444dcg ₩_ myweb_myapp3.1 kingnuna/msa_app2:v1 ip-172-31-12-243
Shutdown Failed 22 seconds ago "task: non-zero exit (1)"
qwfqkgid98ur ₩_ myweb_myapp3.1 kingnuna/msa_app2:v1 ip-172-31-12-243
Shutdown Failed 35 seconds ago "task: non-zero exit (1)"
kab5qq23mqv6 ₩_ myweb_myapp3.1 kingnuna/msa_app2:v1 ip-172-31-12-243
Shutdown Failed 47 seconds ago "task: non-zero exit (1)"

<stack 파일이 아닌 직접 서비스 생성>

7. 스택 삭제

```
ubuntu@ip-172-31-2-139:~$ sudo docker stack rm myweb
Removing service myweb_database
Removing service myweb_myapp1
Removing service myweb_myapp2
Removing service myweb_myapp3
Removing network myweb_test_network
```

8. 서비스로 실행할 이미지 다운로드

ubuntu@ip-172-31-2-139:~$ sudo docker pull kingnuna/msa_app1:v1

9. 서비스로 실행

- 서비스 생성

ubuntu@ip-172-31-2-139:~$ sudo docker service create --name auth --mode global --constraint node.labels.type==auth --publish 8081:8081 kingnuna/msa_app1:v1

--name: 서비스 이름

--mode: 배포 모드 지정. global(1개), replicated(여러 개 배포. replicas로 개수 지정)

--constraint: 배포 조건 지정. (node.labels.type==auth는 노드 라벨이 type=auth인 노드)

--publish: 배포 포트 지정. 외부포트:내부포트

- 서비스 목록 확인

ubuntu@ip-172-31-2-139:~$ sudo docker service ls

- 지정한 서비스 내의 태스크 목록

ubuntu@ip-172-31-2-139:~$ sudo docker service ps <서비스명>

ubuntu@ip-172-31-2-139:~$ sudo docker service ps auth

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS

vtco3hm2dlql auth.yappjhwj8omo32bzfvsjb45p4 kingnuna/msa_app1:v1 ip-172-31-2-139 Running Running 32 seconds ago

84qjzca6puwg ₩_ auth.yappjhwj8omo32bzfvsjb45p4 kingnuna/msa_app1:v1 ip-172-31-2-139 Shutdown Failed 40 seconds ago "task: non-zero exit (1)"

tgth1tqasm1r ₩_ auth.yappjhwj8omo32bzfvsjb45p4 kingnuna/msa_app1:v1 ip-172-31-2-139 Shutdown Failed about a minute ago "task: non-zero exit (1)"

05457jwn60wy ₩_ auth.yappjhwj8omo32bzfvsjb45p4 kingnuna/msa_app1:v1 ip-172-31-2-139 Shutdown Failed 3 minutes ago "task: non-zero exit (1)"

zg2s89ee3q6b ₩_ auth.yappjhwj8omo32bzfvsjb45p4 kingnuna/msa_app1:v1 ip-172-31-2-139 Shutdown Failed 4 minutes ago "task: non-zero exit (1)"

- 서비스 삭제

ubuntu@ip-172-31-2-139:~$ sudo docker service rm <서비스명>

10. 모니터링

리더 노드에서 다음 명령 실행한다.

ubuntu@ip-172-31-2-139:~$ sudo docker container run -d --restart always -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock --name portainer portainer/portainer

웹 브라우저로 [리더노드ip:9000] 접속하면 전체 네트워크 상태를 확인할 수 있다.