Table of Contents

LINUX시스템 과목 기말대체과제

2023학년도 LINUX시스템 과목 기말대체과제용 Gitbook입니다.

Gitbook?

협업을 위한 문서 작성 플랫폼으로 업무 가이드라인, API 명세서 작성 등의 기본 템플릿을 제공한다. 뭣보다 마크다운으로 간편하게 책과 같은 형태로 문서를 만들어낼 수 있어 개발문서 작성에 많이 이용한다.

PDF Export

Gitbook CLI를 거의 방치하고 수익을 내기 위해 SaaS로 전환해 최신 버전의 노드에 대응되지 않는다. 따라서 node 16.18.0 버전을 이용해야 한다.

```
npm install -g gitbook
# 전자책 도구 calibre 설치 및 패키지에 포함된 ebook-convert CLI 도구 /usr/local/bin에 심볼릭 링크 생성
ln -s /Applications/calibre.app/Contents/MacOS/ebook-convert /usr/local/bin/
gitbook pdf
```

리눅스 디스크 관리

파티션 생성하기

- 단순 파티션 만들기
  - `fdisk {disk_path}`
    - 디스크 파티션 기능을 제공하는 커맨드라인 유틸리티로, 여러 커맨드를 이용해 디스크를 관리할 수 있다. 예: `fdisk /dev/sdb`
    - 커맨드
      - n: 새로운 파티션 분할
      - p: 설정 내용 확인
      - w: 설정 내용 저장
- 파일 시스템 생성
  - `mkfs -t {file_system} {partition_device}`
    - 예: `mkfs -t ext4 /dev/sdb1`
  - `mkfs.{file_system} {partition_device}`
    - 예: `mkfs.ext4 /dev/sdb1`
- 파일 시스템 마운트하기
  - `mount {partition_device} {path_to_mount}`
- 파일 시스템 마운트 해제하기
  - `umount {partition_device}`
- 자동 마운트 세팅하기
  - `/etc/fstab` 파일의 마지막 줄에 다음 내용 추가

`{partition_device} {path_to_mount} {file_system} {option} {dump} {pass}`

공간 할당

1. 옵션 부여

   디스크가 꽉 찼을 때 시스템 전체가 가동되지 않는 문제를 해결하기 위해 쿼터 개념을 이용해 각 사용자가 사용할 수 있는 파일의 용량을 제한하게 된다. 아래 옵션들을 `/etc/fstab` 파일의 해당 파티션 부분에 추가한다.

   - `usrjquota=aquota.user`
   - `jqfmt=vfsv0`

   추가로, 설정을 재부팅 없이 반영하기 위해 `mount --options remount {path_to_mount}` 명령을 이용해 다시 마운트한다.

2. 쿼터 DB 생성
   - quota 패키지 설치: `apt -y install quota`

   ```
   cd {path_to_mount}
   quotaoff -avug
   quotacheck -augmn
   rm -f aquota.*
   quotacheck -augmn
   touch aquota.user aquota.group
   chmod 600 aquota.*
   quotacheck -augmn
   quotaon -avug
   ```

3. 사용자별로 공간 할당
   - `edquota -u {user_name}` 명령으로 사용자에 대해 파티션별 할당량 편집
   - `quota` 명령을 이용해 사용자에게 할당된 디스크 공간 확인

        - grace 열을 이용해 할당된 공간이 언제까지 이용 가능한지 확인 가능
    - `repquota {partition_device}` 명령으로 사용자별 현재 사용량 확인
    - `edquota -p {original_user} {target_user}`

디스크 관련 용어

- Filesystem
    - 사용자별 쿼터를 할당하는 파일 시스템 ex) `/dev/sdb1`
- blocks: 현재 사용자가 사용하는 블록 크기. (kb단위)
- soft/hard: 소프트/하드 사용 한도. 0이면 제한이 없음을 의미.
- inodes: inode의 개수를 의미.

Terraform을 이용해 선언적으로 프로젝트별 인프라 관리하기

필요성

AWS를 이용해 프로젝트를 배포하려 할 때, AWS의 설정을 직접 웹 GUI를 이용해 세팅해주는 작업은 서버를 확장하거나 하는 등의 상황에서 불편하고 불필요한 실수를 만들어낼 수 있다. Terraform과 같은 IaC 도구를 이용하면 인프라를 선언적으로, 또 자동으로 관리할 수 있다.


테라폼을 이용한 인스턴스 배포

클라우드 서비스에 구애받지 않음

AWS뿐 아니라 Azure, GCP 등 클라운드 서비스들에 대한 구현을 제공하므로 어떤 클라우드 서비스를 이용하느냐에 구애받지 않는다.

방법

AWS 계정 생성

당연히 AWS를 이용하므로 AWS 계정을 생성해야 한다.

AWS API 엑세스 키 만들기

검색을 통해 **IAM 웹 콘솔**로 이동해 좌측 **사용자** 탭에서 사용자를 생성한다.

사용자에게 **보안 자격 증명 탭**에서 엑세스 키를 생성한다.

AWS CLI 설치 및 구성

아래 명령을 이용해 AWS CLI를 설치할 수 있다.

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

- `aws configure`

  명령을 이용하면 액세스키, 시크릿키, 기본 리전, 기본 출력 형식을 정의해 셋업할 수 있다.

- `cat ~/.aws/credentials` 명령을 이용해 설정 확인

테라폼 설치

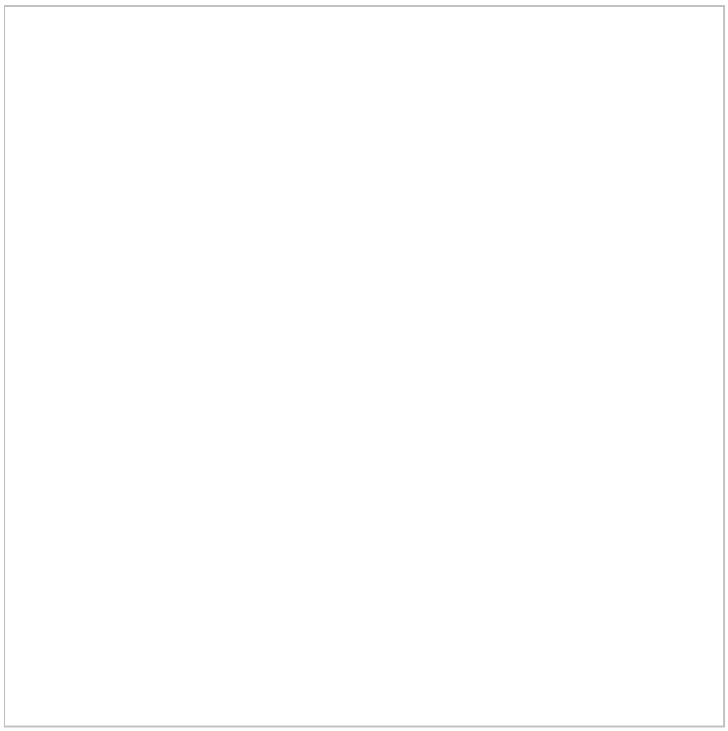
```
wget https://releases.hashicorp.com/terraform/0.14.8/terraform_0.14.8_linux_amd64.zip
ls -al
sudo apt install yum
sudo yum install -y unzip
unzip terraform_0.14.8_linux_amd64.zip
cp terraform /usr/bin/
./terraform -version
```

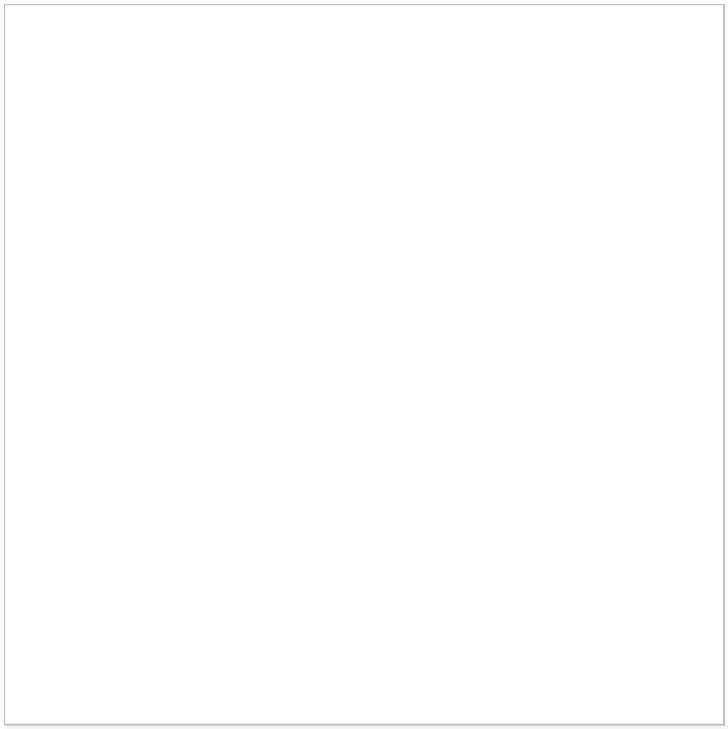
테라폼을 이용해 인스턴스 생성/배포/제거

```
nano main.tf # 테라폼 설정 파일 셋업
terraform init
terraform apply
terraform destroy
```

AWS EC2 Instance 배포를 위한 설정 파일 구성

- `ami` : Amazon Machine Image. 생성할 인스턴스의 이미지 ID를 지정해준다.
- `instance_type` : 인스턴스 종류
- `tags` : 인스턴스에 부여할 태그





```
provider "aws" {
  region = "ap-northeast-2"
}

resource "aws_instance" "test_instance" {
  ami           = "ami-086cae3329a3f7d75"
  instance_type = "t2.micro"
  tags = {
    Name = "Test"
  }
}
```

저장소

웹서버와 함께 이미지, 동영상 등의 파일을 저장하는 경우 AWS S3와 같은 오브젝트 스토리지를 이용하는 것이 효율적

개념

버킷

- 저장공간을 구분하는 단위 (디렉토리와 유사)
- 고유한 값을 ID로 가짐

파일 객체 구성 정보

- Owner
- Key, Value
- Version ID
- Metadata

보안 설정

- 기본적으로 버킷은 비공개, 사용 목적에 따라 변경 가능
- 버킷 단위로 보안 설정
- 엑세스 록 설정

IPFS

파일, 애플리케이션 등 데이터를 보관하는 분산 시스템.

IPFS는 공짜다. 아니, 사실은 아니다.

원래 스토리지에 무언가를 저장하고 호스팅한다는 것은 하드웨어의 전원이 항시 들어오게 해야 한다는 것이고, 분산 시스템이라고 해서 예외는 아니다.

저장 증명 방식으로서 스토리지에 기여하는 노드들에게 인센티브를 지급한다.

RAID

개념

- 여러 개의 디스크를 하나의 디스크 처럼 관리할 수 있는 기능

하드웨어 RAID

- 하드웨어 레벨에서 제조 업체가 여러 개의 디스크를 연결한 장비를 만들어 공급하는 것 혹은 연결하는 것
- 안정적이고 제조 업체의 기술지원을 받을 수 있지만 가격이 비싸고 제조 업체마다 조작 방법 상이
- 고가의 경우 SA-SCSI 디스크로, 중저가의 경우 SATA 디스크로 만듦

소프트웨어 RAID

- 운영체제 안에서 구현되어 디스크를 관리한다
- 하드웨어 RAID와 비교하면 신뢰성, 속도가 낮지만 적은 비용으로 안전하게 데이터 저장 가능

RAID 레벨

RAID 구성 방식

- 단순 볼륨 (레이드가 아님)
- **Linear RAID**
  - 앞 디스크에 데이터를 모두 저장한 후 다음 디스크에 저장
- **RAID 0**
  - 모든 디스크를 병렬적으로 사용 (분산해서 저장)

Linear RAID와 RAID 0 비교

RAID 0은 여러 개의 디스크에 동시로 저장하는 방법인 스트라이핑 방식으로, 12바이트를 저장하는 경우 Linear RAID는 한 곳에 12바이트를 순차적으로 쓰는 반면에 RAID 0는 4바이트씩 써서 속도가 더 빠르다.

Linear RAID는 한 디스크 당 공간 효율성이 좋지만 속도가 느려 안정성에 초점을 맞출 때 적합하고,

RAID 0는 속도가 빠른 대신에 디스크가 하나만 고장나도 전체 데이터의 무결성을 보장하지 못해 속도에 초점을 맞출 때 적합하다.

- **RAID 1**

  미러링 개념으로, 똑같은 데이터를 각 디스크 저장한다. 공간 효율성이 50%에 달하고, 중요 데이터를 저장하기에 적당한 방식이다. 데이터를 이중화하므로 안정성이 매우 높다.

- RAID 2
- RAID 3
- RAID 4
- **RAID 5**

  네트워크에서 오류 처리를 하는 것 처럼 데이터 중간에 패리티 비트를 삽입하고, 디스크에 오류가 발생하면 패리티 비트를 이용해 데이터를 복구할 수 있다. 예를 들어, 짝수 패리티의 경우 00111의 이진 데이터는 1이 홀수개이므로 1을 추가해 짝수개가 되게 한다.

최소 세개 이상의 디스크가 필요하며, 디스크 중 하나가 고장나도 복원이 가능하다.

어느 정도의 결성을 허용하고 저장 공간의 효율성도 뛰어나다는 것이 장점이다.

- RAID 6

  RAID 5에서 패리티를 두개 이용해 처리하게 된다.

관련 커맨드

- 디스크 리스트 조회: `ls -l /dev/sd*`
- 파티션 생성: `fdisk {disk_path}` (RAID로 묶을 파티션들 모두 생성)
  - 파일 시스템으로 `fd` 부여
- RAID 관리 패키지 설치: `apt -y install mdadm`
- 파티션 상태 확인: `fdisk -l {disk_path}`
- RAID 구축: `mdadm --create {target_path} --level={raid_level} --raid-devices={device_number} {partition_paths}`
- 파일 시스템 생성/마운트
  - 파일 시스템 생성
    - `mkfs -t {file_system} {partition_device}`
      - 예: `mkfs -t ext4 /dev/sdb1`
    - `mkfs.{file_system} {partition_device}`
      - 예: `mkfs.ext4 /dev/sdb1`
  - 파일 시스템 마운트
    - `mount {partition_device} {path_to_mount}`
- RAID 확인: `mdadm --detail {partition_device}`
- RAID 구성 해제: `mdadm --stop /dev/md*`
- RAID 버그 방지: `/etc/mdadm/mdadm.conf` 파일에 `mdadm --detail --scan` 명령의 결과 에서 name을 제거하고 삽입

웹 서버

nginx

nginx는 아파치와 함께 웹서버 중 하나로 이미지, 동영상, 자바스크립트, HTML 등 다양한 문서를 제공하는 서버 시스템으로, HTTP 통신 프로토콜을 통해 리소스를 전달한다.

```
sudo apt update
sudo apt install nginx
sudo ufw allow 'Nginx HTTP'
sudo systemctl status nginx # 상태 확인
```

프록시로서의 nginx

클라이언트가 웹서버에 요청을 보내면 중간에서 프록시 서버가 요청을 가로챈다. 방문하고자 하는 웹사이트에 직접적으로 방문하는 것을 방지

리버스 프록시

여러 개의 서버 앞에 두고, 특정 서버가 과부화되지 않게 로드 밸런싱을 수행하는 서버로, 웹서버의 IP 주소를 노출시킬 필요가 없다는 것도 장점이다. 특히, 나는 하나의 80 포트를 listening 하는 nginx 서버에서 서브도메인 별로 다른 웹서버/서비스를 이용하게 하고 싶어서 리버스 프록시를 이용했다.

systemctl 명령

stop/start/restart/reload/disable/enable 명령

nginx configuration

`/etc/nginx/sites-available` 디렉토리에 사이트들을 정의할 수 있다. configuration 파일은 다음과 같은 요소들을 포함할 수 있다.

```
server {
    listen 80;
    listen [::]:80;
    server_name dweb;

    root /var/dweb/
    index index.html;

    location /songdo {
        proxy_pass http://172.0.0.1:3000;
        proxy_http_version 1.1;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

- `server` 블록
  - `listen` : 어떤 URI/포트의 요청을 이 사이트로 넘길 것인지를 선언한다.
  - `server_name` : Host Header가 담고 있는 필드값과 일치한 도메인명을 가진 요청에 대해서만 허용하게 할 수 있다. wild card 와 정규 표현식 모두 이용 가능하다.
  - `root` : static file이 있는 파일 시스템의 경로이다.
  - `location` 블록: 경로에 따라 요청을 어떻게 처리할 지 결정할 수 있다. static file을 보여주거나 프록시 서버로 요청을 전송

할 수 있다. 아래 예제와 같이 설정하면 `/songdo` path로 요청하는 경우 localhost의 3000번 포트로 연결해주는 것이다.

이렇게 설정 파일을 생성하고 `/etc/nginx/sites-enabled` 를 타겟으로 심볼릭 링크를 생성한다. 이로서 설정 파일을 활성화 한 것이다. 아래는 심볼릭 링크 생성을 포함해 nginx를 재시작하는 예제이다.

```
ln -s /etc/nginx/sites-available/{conf_file} /etc/nginx/sites-enabled/
```

``` # server_names_hash_bucket_size 64; (# 제거) # include /etc/nginx/conf.d/*.conf; (# 추가) include /etc/nginx/sites-enabled/{conf_file}; ```

이렇게 설정하면 이 사이트의 루트로 접속하면 스태틱 파일을, /songdo 로 접속하면 백엔드 서버 등 다른 웹서버로 연결을 포워딩하게 된다. 이렇게 세팅한 후 리액트 프로젝트를 세팅하고 스태틱 페이지로 빌드해도 되고, express 등 서버를 구동시켜 놓을 수도 있다.
```

Github Action을 이용한 CI/CD와 프론트엔드 호스팅

CI/CD 파이프라인은 개발자가 새로운 버전을 배포할 때 직접 아래와 같은 과정을 거치지 않고 진행하게 해준다.

```
git fetch && git pull
npm i --save
npm run build
sudo systemctl restart nginx
```

CI

Continous Integration의 약자로, VCS 저장소에 커밋이 푸시되면 이 코드를 빌드해 오류가 없는지 확인한 후 프로젝트 구동에 필요한 의존성들과 함께 도커 컨테이너로 만들고, 이를 도커 허브같은 공개적 컨테이너 이미지 저장소나 ghcr.io같은 비공개 처리 가능한 컨테이너 이미지 저장소에 푸시하게 된다.

CD

Continous Devliery의 약자로, CI 과정에서 컨테이너 이미지 저장소에 푸시된 내용을 watchtower를 이용해 변화를 감지하거나 CI 과정 후 자동으로 ssh 접속해서 컨테이너 이미지 저장소의 내용을 pull 해와 컨테이너를 구동하게 된다.

이러한 과정을 통해 개발자들은 배포에 있어서 굉장한 편리함을 가질 수 있게 되었다.

https://github.com/KimWash/ipfs-hosting

Docker

Docker란

도커는 애플리케이션을 컨테이너로 사용할 수 있도록 만들어진 프로젝트로, 컨테이너 개념은 호스트 OS에 영향을 미치지 않고 애플리케이션이 구동될 수 있도록 한다.

- Docker 엔진
  - 도커의 메인 프로젝트로, 컨테이너와 관리의 주체이다.
  - 컨테이너를 제어하고, 다양한 기능들을 제공한다.

기존 가상화 기술과의 차이점?

VMWare나 VirtualBox와 같은 가상화 기술을 이용하게 되면 하이퍼파이저를 이용해 OS를 밑바닥부터 모두 구현하게 된다. 이렇게 하게 되면 하이퍼바이저로 인해 성능 손실이 발생하게 되며, 게스트 OS 전체를 포함하게 되므로 이미지 사이즈가 비대해지게 된다.

반면에 도커 컨테이너의 경우 리눅스의 chroot, namespace, cgroup 기능을 활용해 프로세스 단위의 격리된 환경을 구축하게 된다. 애플리케이션 구동에 필요한 라이브러리 및 바이너리만 존재하므로 이미지 크기가 감소하게 된다. 참고로, 커널은 호스트의 것을 그대로 공유하게 된다.

장점

호스트OS에 영향을 받을 수 있어도 영향을 주지는 못하므로 호스트에 서비스 운영에 필요한 패키지를 직접 설치하지 않아도 되고, 라이브러리 설치 등의 의존성 문제가 없어지게 돼 배포하는 상황에서 문제가 될만한 시나리오를 원천적으로 생기지 않게 해준다.

또한, 애플리케이션 간 독립성/확장성을 보장해주는데, 애플리케이션을 여러 모듈로 나눠 컨테이너 형태로 구동해 MSA를 구축할 수 있게 된다.

개념

이미지

- 컨테이너를 생성할 때 사용하는 요소
- VM의 iso 개념과 유사
- 이름 구성: {저장소이름}/{이미지이름}:{태그}
  - ex) `KimWash/my-service:latest`
- 저장소 이름/태그 생략 가능

컨테이너

- 도커 이미지로부터 컨테이너를 생성 가능
- 호스트로부터 분리되어 컨테이너 내의 작업이 호스트에 영향을 끼치지 않음

도커 컨테이너 생성 및 쉘 접속법

```
docker run -it ubuntu:22.04 # ubuntu 22.04 컨테이너 이미지 다운로드 후 실행
```

혹은 이미지를 따로 받고 컨테이너 생성 후 따로 쉘에 접속할 수도 있다.

```
docker pull centos:7
docker images
```

```
docker create -it --name mycentos centos:7
docker start mycentos
docker attach mycentos
```

도커 관련 커맨드

- `docker ps` : 현재 실행중인 컨테이너
  - `-a` : 실행중이지 않은 것 포함
- `docker rm` : 컨테이너 삭제 (실행중인 것 삭제 불가)
- `docker stop` : 컨테이너 중지
- `docker container prune` : 컨테이너 일괄 삭제
- `docker run {image_name}`
  - `-i` : 컨테이너와 상호 입출력
  - `-t` : bash 쉘 이용
  - `-p {outbound}:{inbound}` : outbound 포트를 컨테이너 내부에서 inbound 포트로 포워딩
  - `--name {container_name}` : 컨테이너 이름 지정
  - `-e {ENV_NAME}={ENV_VALUE}` : 컨테이너에 주입할 환경변수 지정
  - `-d` : 백그라운드
  - `-v {host_path_to_mount} {container_path_to_mount}` : 호스트의 특정 폴더를 컨테이너의 특정 폴더에 마운트 시킨다. 파일시스템을 공유할 때 이용한다.

도커 볼륨

도커 이미지는 read-only로, 컨테이너 계층에서 파일시스템에 변동이 생기면 컨테이너 삭제 시 초기화된다. 이를 방지하기 위해 호스트의 파일 시스템에 마운트 포인트를 두고 컨테이너에서 접근 가능하게 해주어야 한다.

- `docker volume create --name {volume_name}` : 도커에서 통합적으로 관리하는 볼륨 생성. 호스트쪽의 마운트 포인트를 생성하게 된다.
  - `docker run {image_name} -v {volume_name}:{container_path_to_mount}` 명령으로 이용 가능하다.

도커 네트워크 드라이버

- 컨테이너가 외부와 통신 가능하도록 지원하며, 아래와 같은 종류들을 가진다.
  - Bridge
  - Host
  - None
  - Container
  - Overlay
- 드라이버 생성

  `docker network create --driver {type} {driver_name}`

  - 해당 드라이버를 이용한 컨테이너 생성

    `docker run --name {container_name} --net {driver_name} {image_name}`

- 드라이버 연결 및 해제

  `docker network disconnect {driver_name}`

  `docker network connect {driver_name} {container_name}`

- 호스트의 네트워크와 같은 환경 이용하기 (Host)

  `docker run --name {container_name} --net host {image_name}`

Container Network

- 다른 컨테이너의 내부 IP나 MAC주소와 같은 네트워크 네임스페이스를 공유할 수 있다.

  `--net container:{container_id}`

컨테이너 로깅

표준 출력/에러 스트림의 로그가 별도의 파일에 저장되며, `docker log {container_id_or_name}` 명령을 이용해 확인 가능하다.

Docker Hub

Github와 같이 도커 컨테이너 이미지를 저장하는 저장소로, 도커에서 기본적으로 제공하므로 아래 명령어를 이용해 이용할 수 있다.

- `docker search {image_name}` : 이미지를 검색한다.
- `docker commit {container_name} {image_name}:{tag}`
  - `-a {author}` : 이미지 만든 사람
  - `-m {message}` : 커밋 메시지
- `docker push {author}/{image_name}:{tag}` : 커밋 내용 저장소에 푸시
- `docker pull {author}/{image_name}:{tag}` : Docker hub 저장소에서 이미지 다운로드
- `docker inspect {image_name}:{tag}` : 이미지의 레이어를 해시값과 함께 확인할 수 있다.
- `docker history {image_name}:{tag}` : 이미지의 히스토리를 확인할 수 있다.
- `docker rmi {image_name}:{tag}` : 이미지 삭제

Dockerfile

필요성

도커 이미지 생성 과정을 선언형으로 관리해 애플리케이션의 빌드 및 배포의 자동화를 가능하게 한다.

예제 Dockerfile

```
FROM ubuntu:22.04
LABEL maintainer="INU LINUX"

RUN apt-get update
RUN apt-get install apache2 -y
ADD test.html /var/www/html
WORKDIR /var/www/html
RUN ["/bin/bash", "-c", "echo hello >> test2.html"]
EXPOSE 80
CMD apachectl -DFOREGROUND
```

- `FROM` : 대상이 되는 원본 이미지
- `LABEL` : 이미지에 붙일 레이블
- `RUN` : 컨테이너 빌드 전 실행할 명령
- `ADD {filename} {path}` : 컨테이너 빌드 전 추가할 파일
- `WORKDIR {path}` : 컨테이너 상에서 작업 디렉토리로 전환
- `EXPOSE {port}` : 특정 포트를 노출시킬 수 있다.
- `CMD` : 컨테이너를 생성하여 최초로 실행할 때 실행할 명령 (런타임)

Multi-Stage Builds를 이용해 이미지 경량화하기

하나의 Dockerfile로 빌드 이미지와 실행 이미지를 분리해 빌드 이미지가 라이브러리 의존성에 의해 비대해지는 것을 방지할 수 있다.

```
FROM golang:alpine:3.18
ADD main.go /root
WORKDIR /root
RUN go build -o /root/mainApp /root/main.go

FROM alpine:latest
WORKDIR /root
COPY --from=0 /root/mainApp
CMD ["./mainApp"]
```

Docker Compose

여러 컨테이너가 유기적으로 동작하는 서비스를 위해 (예를 들면 DB가 필요한 서비스들) 만들어진 솔루션

- 설치

```
curl -L https://github.com/docker/compose/releases/download/v2.5.0/
docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose
docker-compose --version
```

- Docker Compose는 yaml 파일을 이용해 컨테이너를 생성한다.

```
version: "3.8"
services:
  mongo:
    image: mongo:5.0
    container_name: mongo
    environment:
docker-compose.yml
      - MONGO_INITDB_ROOT_USERNAME=dweb
      - MONGO_INITDB_ROOT_PASSWORD=1234
    restart: unless-stopped
    ports:
      - "27017:27017"
    volumes:
- ./database/dbs:/data/db
- ./database/dev.archive:/Databases/dev.archive - ./database/production:/Databases/production
  mongo-express:
    image: mongo-express
    container_name: mexpress
    environment:
- ME_CONFIG_MONGODB_SERVER=mongo
- ME_CONFIG_MONGODB_PORT=27017
- ME_CONFIG_MONGODB_AUTH_USERNAME=dweb - ME_CONFIG_MONGODB_AUTH_PASSWORD=1234 - ME_CONFIG_BASICAUTH_USERNAME=dw
eb
- ME_CONFIG_BASICAUTH_PASSWORD=1234
    links:
      - mongo
    restart: unless-stopped
    ports:
      - "8001:8081"
networks:
  default:
    name: mongo-express-network
```

Dockerfile 관련 커맨드

- `docker build -t {image_name}:{tag} {target_path}` : 이미지 빌드
- `docker-compose ps` : 현재 실행중인 도커 컴포즈 컨테이너로 생성된 컨테이너들의 상태를 확인할 수 있다.
  - `-p {project_name}` : 특정 프로젝트의 컨테이너로 한정한다.
- `docker-compose restart` : 서비스 컨테이너 재시작
- `docker-compose up` : 프로젝트의 컨테이너 모두 시작
  - `-p {project_name}` : 프로젝트 이름을 명시하면서 시작
  - `-d` : 백그라운드