

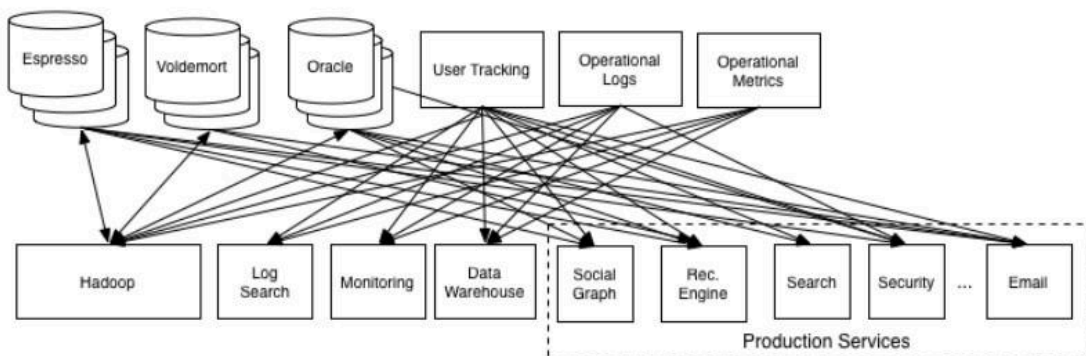
# KAFKA

## 1. KAFKA란?

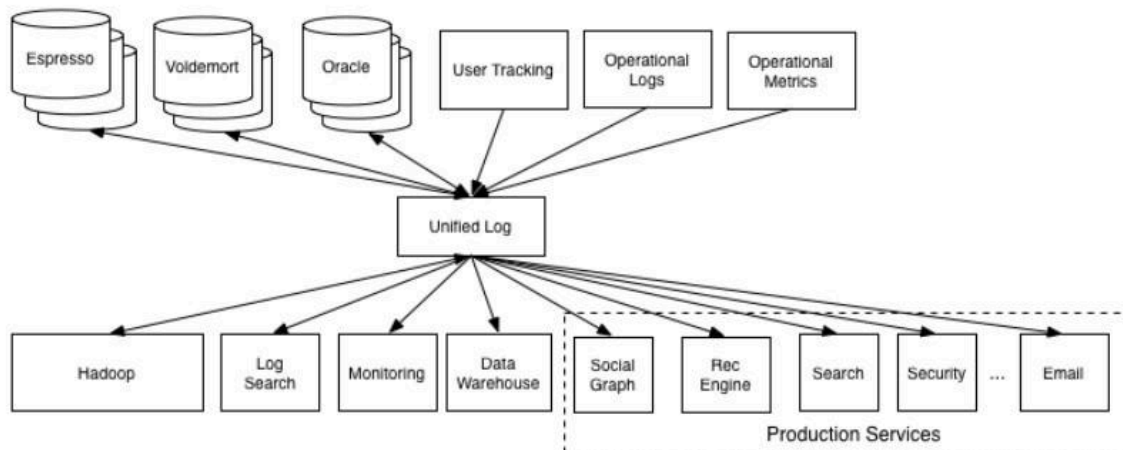
아파치 카프카는 메시지 발행/구독 시스템이다. ‘분산 커밋 로그’ 혹은 ‘분산 스트리밍 플랫폼’이라고 불리기도 한다. 카프카에 저장된 데이터는 순서를 유지한채로 지속성 있게 보관되며 결정적으로 읽을 수 있다. 또한 확장시 성능을 향상시키고 실패가 발생하더라도 데이터 사용에는 문제가 없도록 시스템 안에서 데이터를 분산시켜 저장할 수 있다.

### 1-1 카프카의 탄생

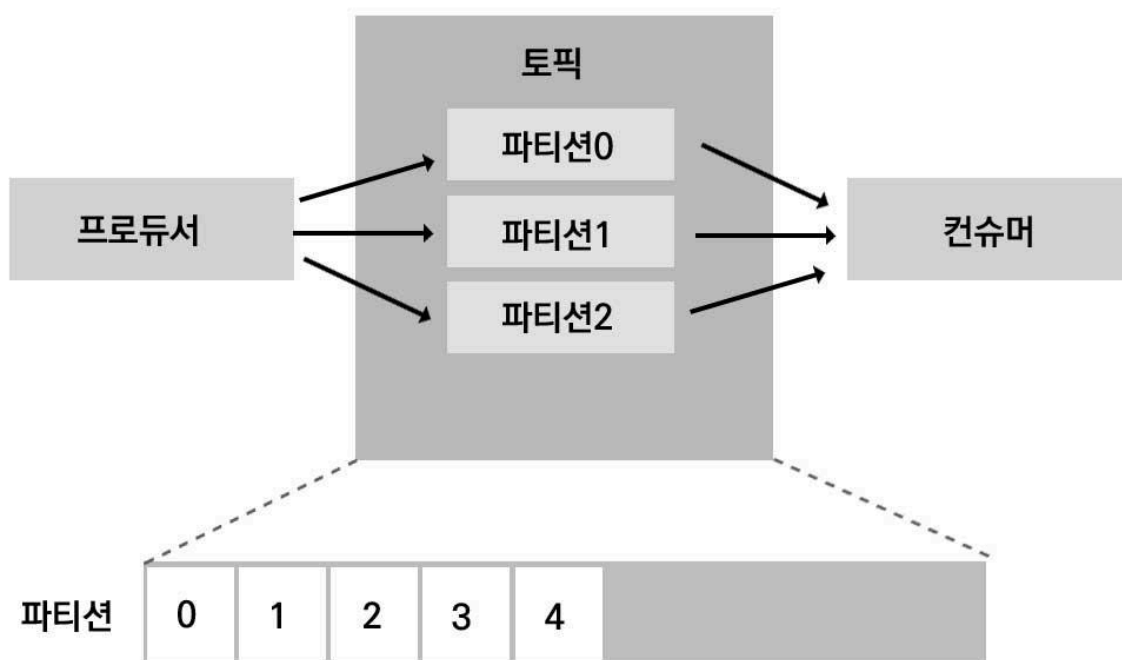
2011년 구인/구직 및 동종업계의 동향을 살펴볼 수 있는 소셜 네트워크 사이트인 ‘링크드인(LinkedIn)’에서는 데이터를 생성하고 적재하기 위해서는 데이터를 생성하는 소스 애플리케이션과 데이터가 최종 적재되는 타깃 애플리케이션을 연결해야 한다. 시간이 지날수록 아키텍처는 거대해졌고 소스 애플리케이션과 타깃 애플리케이션의 개수가 점점 많아지면서 문제가 생겼다. 데이터를 전송하는 라인이 기하급수적으로 복잡해지기 시작했다.



카프카는 각각의 애플리케이션끼리 연결하여 데이터를 처리하는 것이 아니라 한 곳에 모아 처리할 수 있도록 중앙집중화했다. 카프카를 통해 웹사이트, 애플리케이션, 센서 등에서 취합한 데이터 스트림을 한 곳에서 실시간으로 관리할 수 있게 된 것이다.



카프카 내부에 데이터가 저장되는 파티션의 동작은 **FIFO(First In First Out)** 방식의 큐 자료구조와 유사하다. 큐에 데이터를 보내는 것이 ‘프로듀서’이고 큐에서 데이터를 가져가는 것이 ‘컨슈머’다.



카프카 프로듀서, 컨슈머, 토픽 모식도

## 1-2 카프카의 특징

### 1) 높은 처리량

카프카는 프로듀서가 브로커로 데이터를 보낼 때와 컨슈머가 브로커로부터 데이터를 받을 때 모두 묶어서 전송함.

### 2) 확장성

카프카는 가변적인 환경에서 안정적으로 확장 가능하도록 설계됨

### 3) 영속성

영속성이란 데이터를 생성한 프로그램이 종료되더라도 사라지지 않은 데이터의 특성을 뜻한다. 카프카는 다른 메시징 플랫폼과 다르게 전송받은 데이터를 메모리에 저장하지 않고 파일 시스템을 저장함

### 4) 고가용성

3개 이상의 서버들로 운영되는 카프카 클러스터는 일부 서버에 장애가 발생하더라도 무중단으로 안전하고 지속적으로 데이터를 처리할 수 있음.

## 1-3 카프카 주요 개념

### 1) 토픽과 파티션

카프카에 저장되는 메시지는 토픽(**topic**) 단위로 분류된다. 토픽과 저장 비슷한 개념으로는 데이터베이스의 테이블이나 파일시스템의 폴더가 있을 것이다. 토픽은 다시 여러 개의 파티션(**partition**)으로 나뉘어진다

## 2) 프로듀서와 컨슈머

프로듀서는 새로운 메시지를 생성한다. 기본적으로 프로듀서는 메시지를 쓸 때 토픽에 속한 파티션들 사이에 고르게 나눠서 쓰도록 되어 있다.

컨슈머는 메시지를 읽는다. 컨슈머는 1개 이상의 토픽을 구독해서 여기에 저장된 메시지들을 각 파티션에 쓰여진 순서대로 읽어 온다.

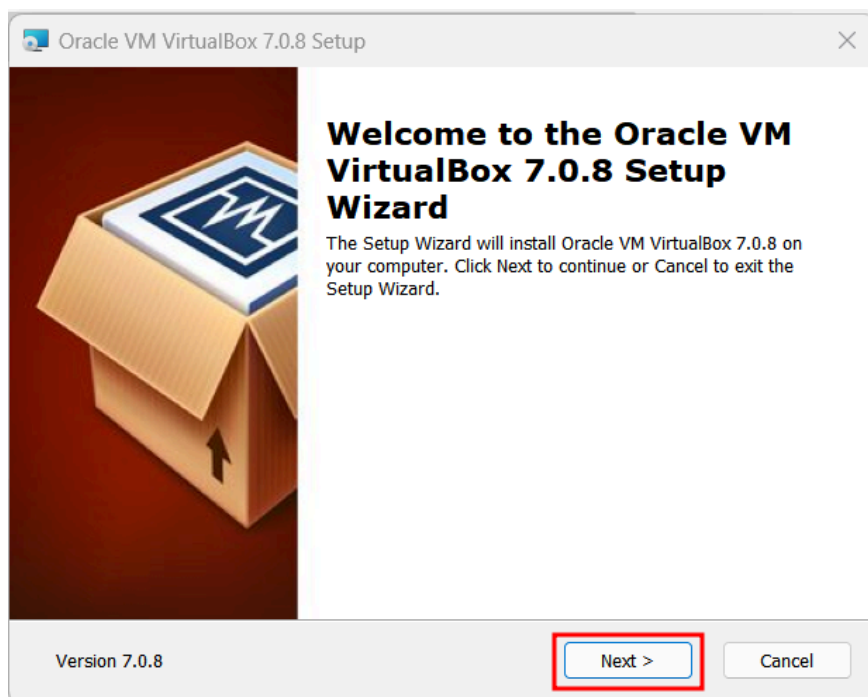
## 3) 카프카 브로커

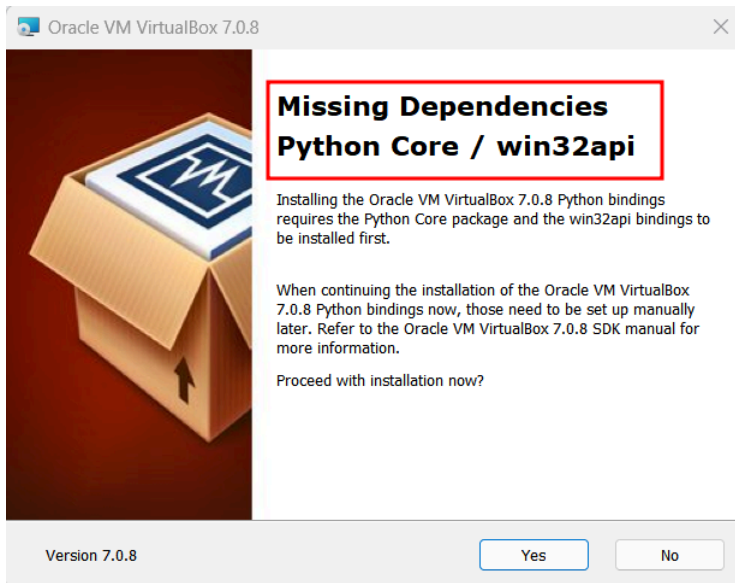
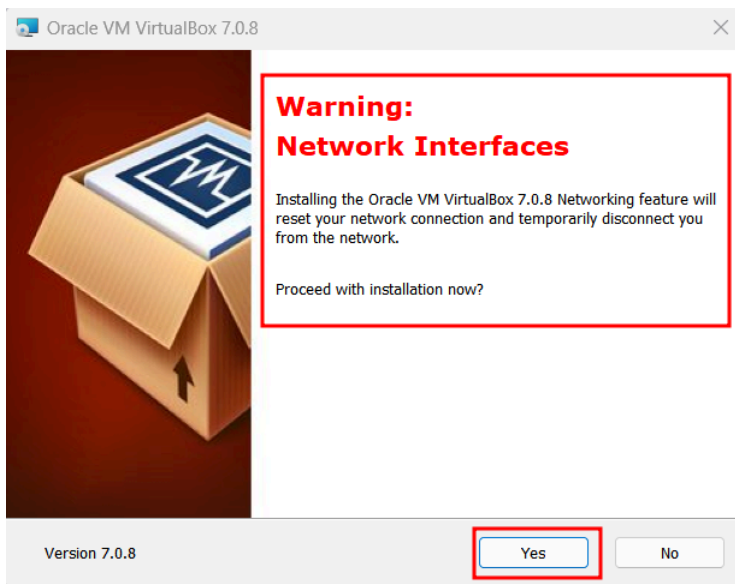
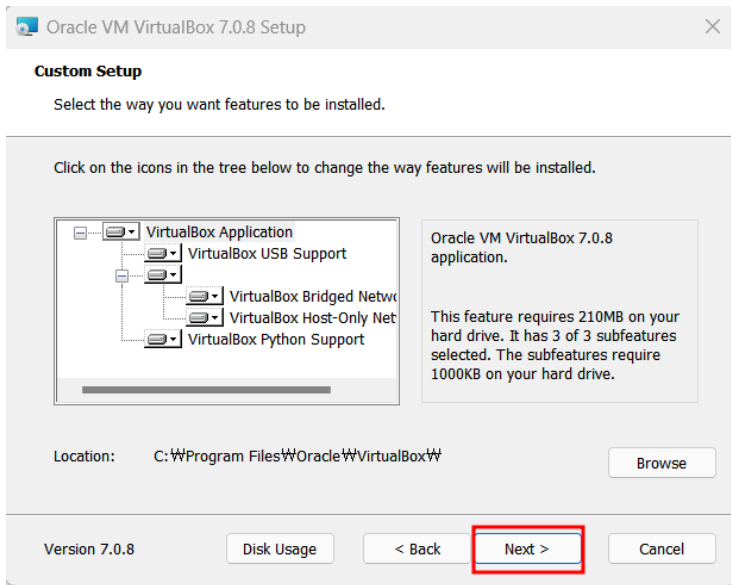
하나의 카프카 서버를 브로커라고 부른다. 브로커는 프로듀서로부터 메시지를 전달받아 오프셋을 할당한 뒤 디스크 저장소에 쓴다. 브로커는 컨슈머의 파티션 읽기 요청 역시 처리하고 발생한 메시지를 보내준다. 카프카 클러스터로 묶인 브로커들은 프로듀서가 보낸 데이터를 안전하게 분산 저장하고 복제하는 역할을 수행한다.

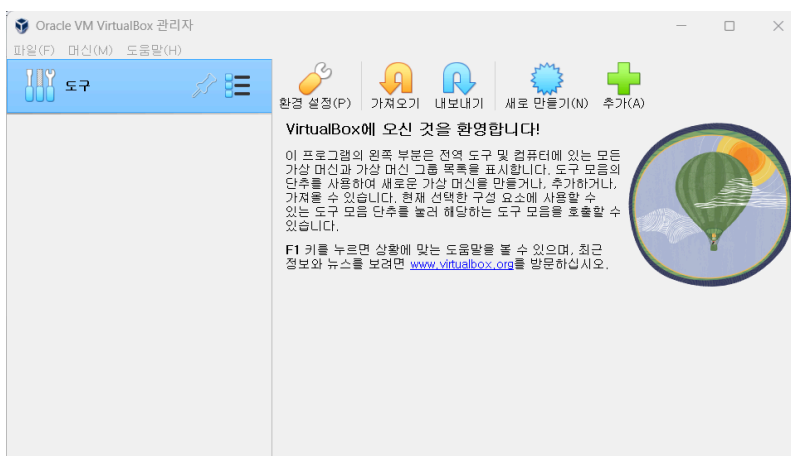
## 2. 카프카 설치하기

### 2-1 VirtualBox 설치하기

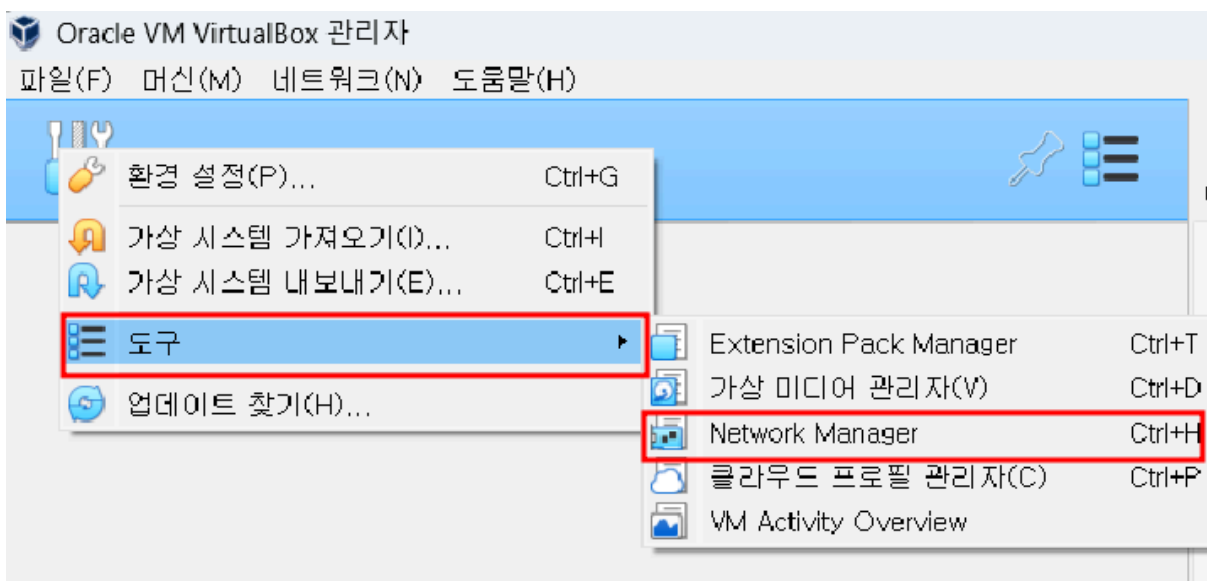
\* VirtualBox가 설치되지 않으면 VC\_redist.x64.exe 를 우선 설치한 후 설치할 것

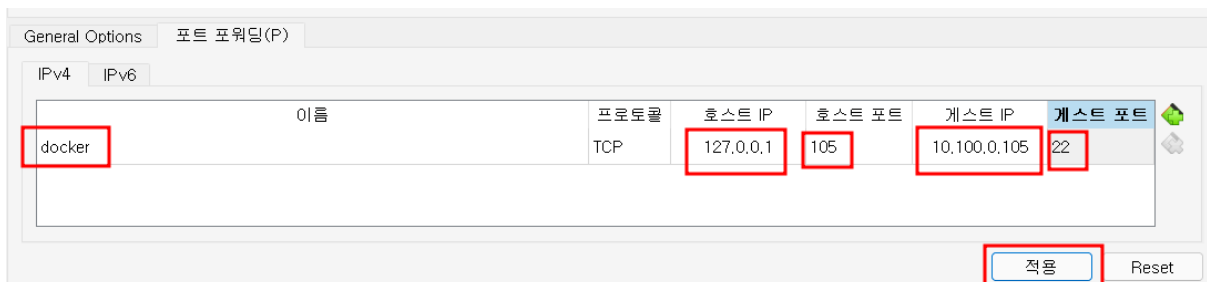
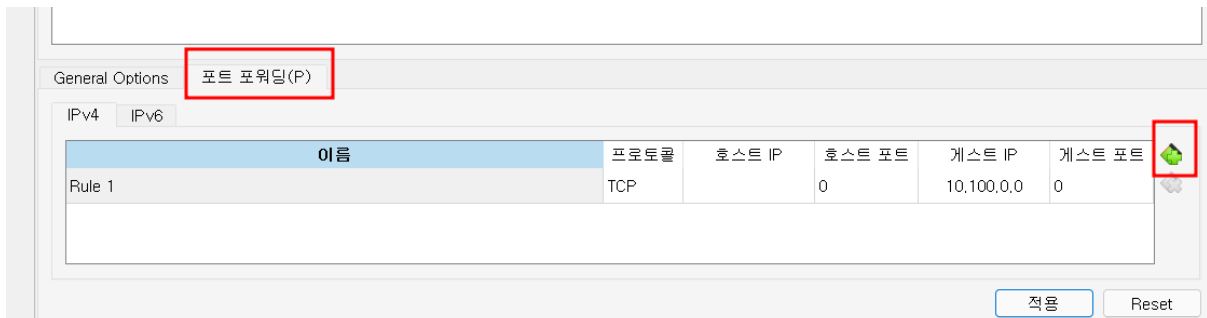
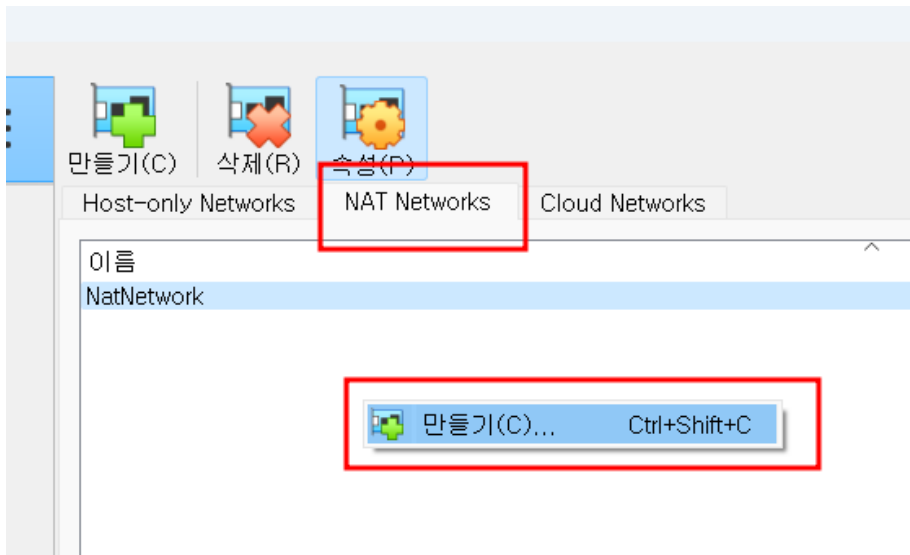






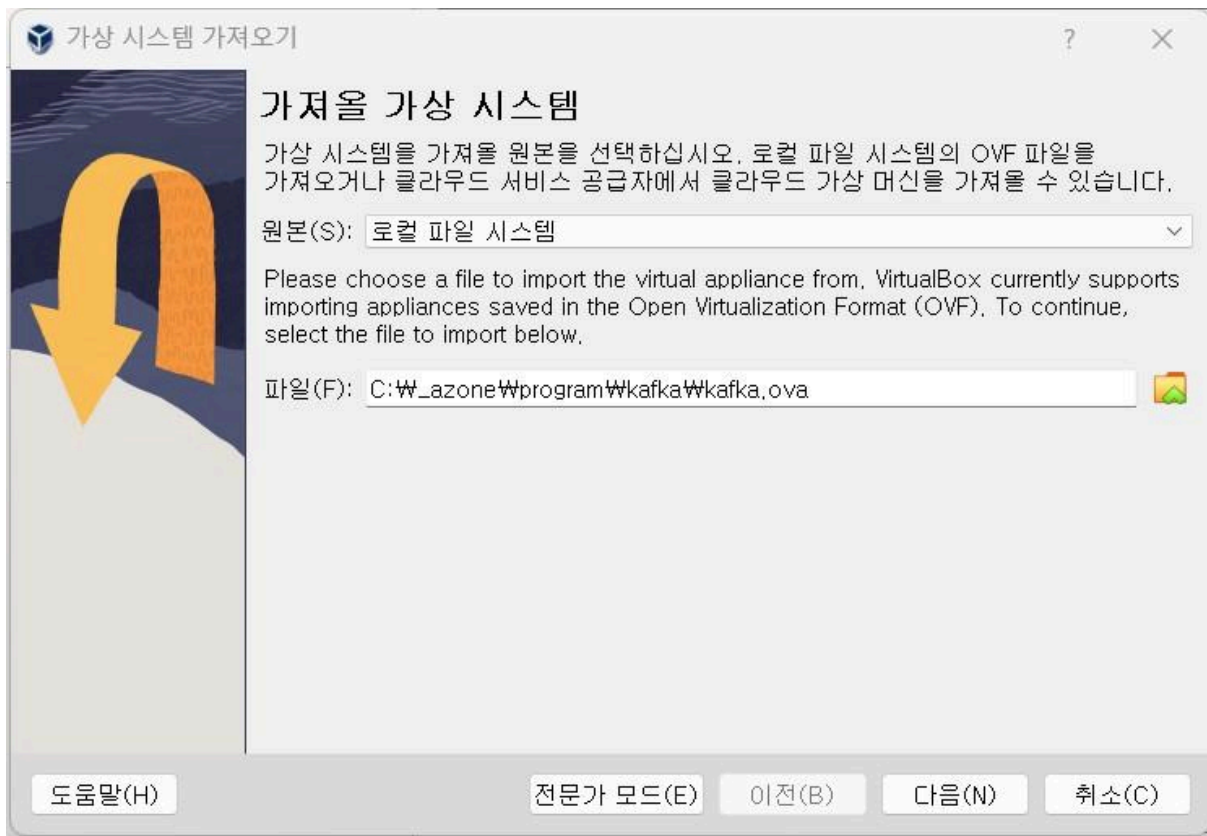
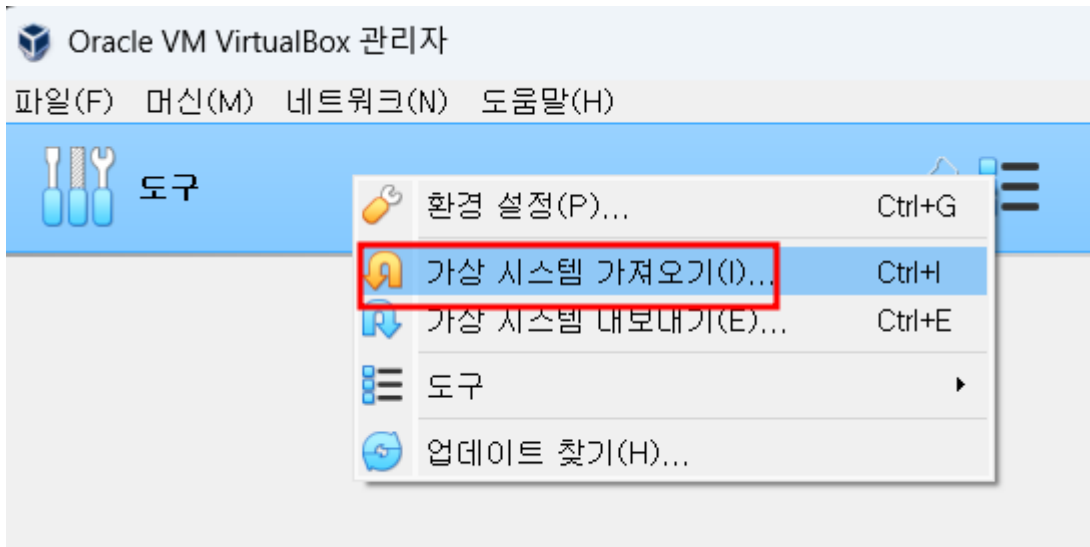
## Virtual Box에 ubuntu 이미지 설치하기

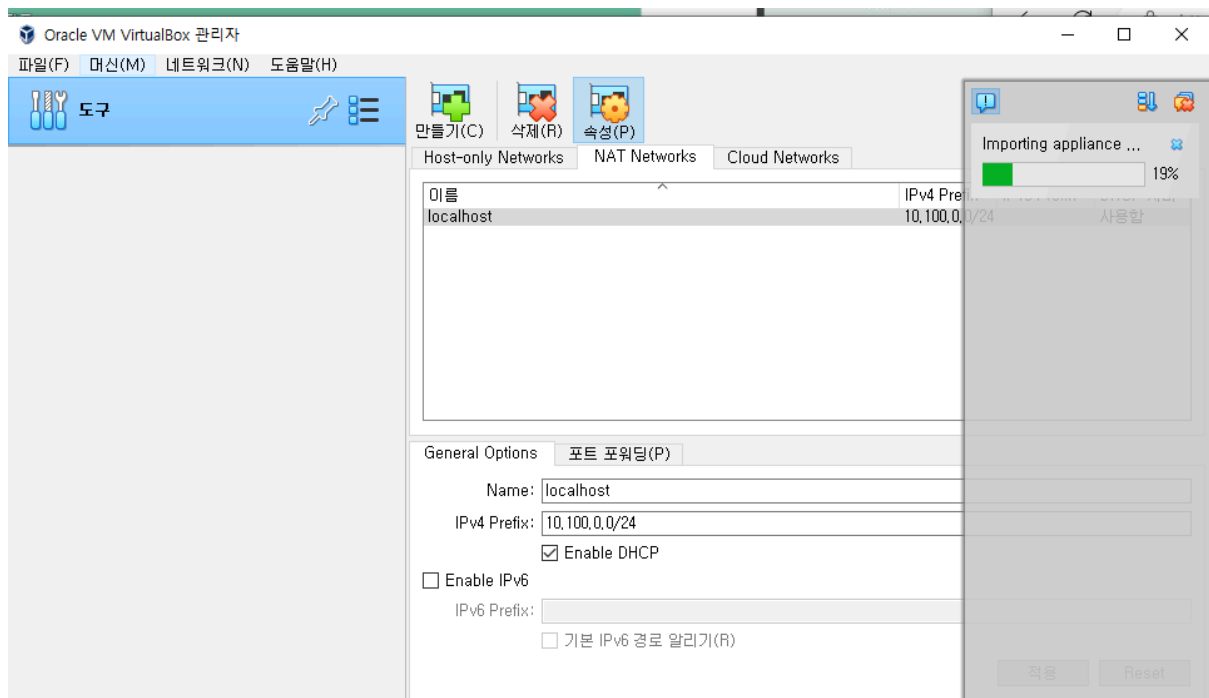




\* 호스트 IP는 사용하는 PC의 IP 주소 사용

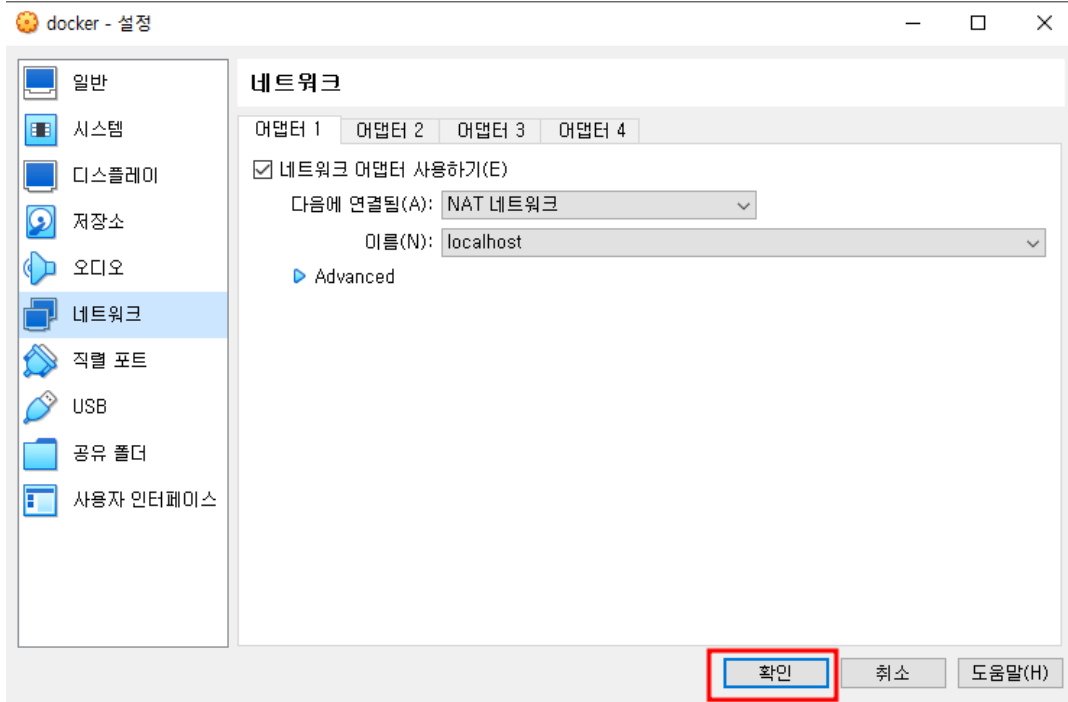
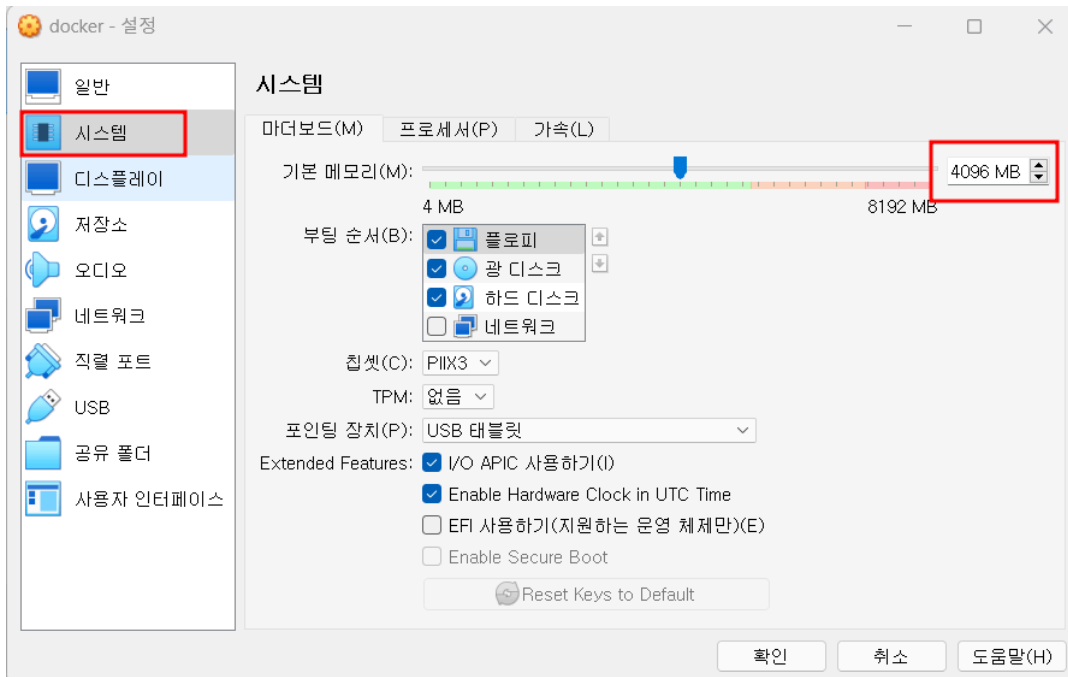
이름	프로토콜	호스트 IP	호스트 포트	게스트 IP	게스트 포트
kafka	TCP	192.168.10.68	105	10.100.0.105	22
kafka2	TCP	192.168.10.68	9092	10.100.0.105	9092







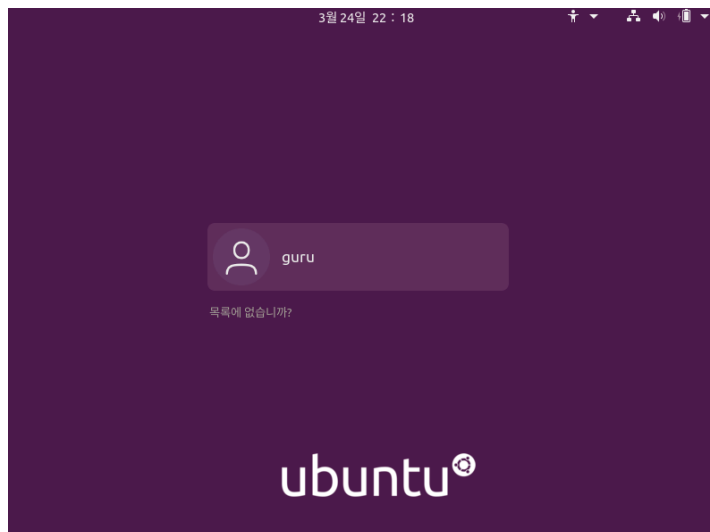




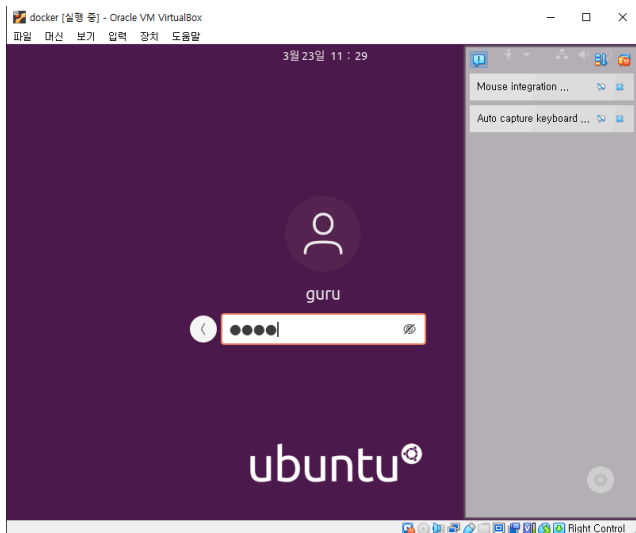
(주의)설치한 리눅스가 실행이 되지 않는 현상(exit code -1073741819 (0xc0000005))이 발생하면 Fasoo DRM Client for Kyobo Book Wix 가 설치되어 있는지 확인 하고 설치되어 있으면 제거하고 실행



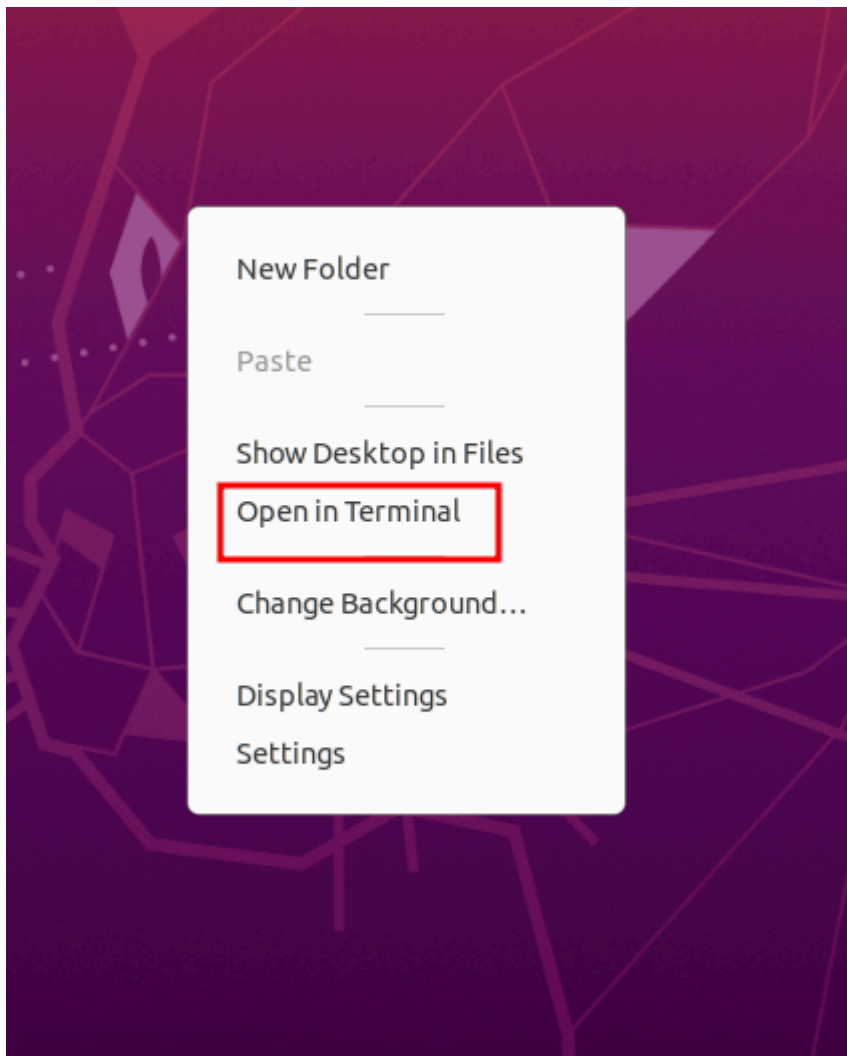
리눅스가 구동되면 일반계정 아이디 guru로 로그인

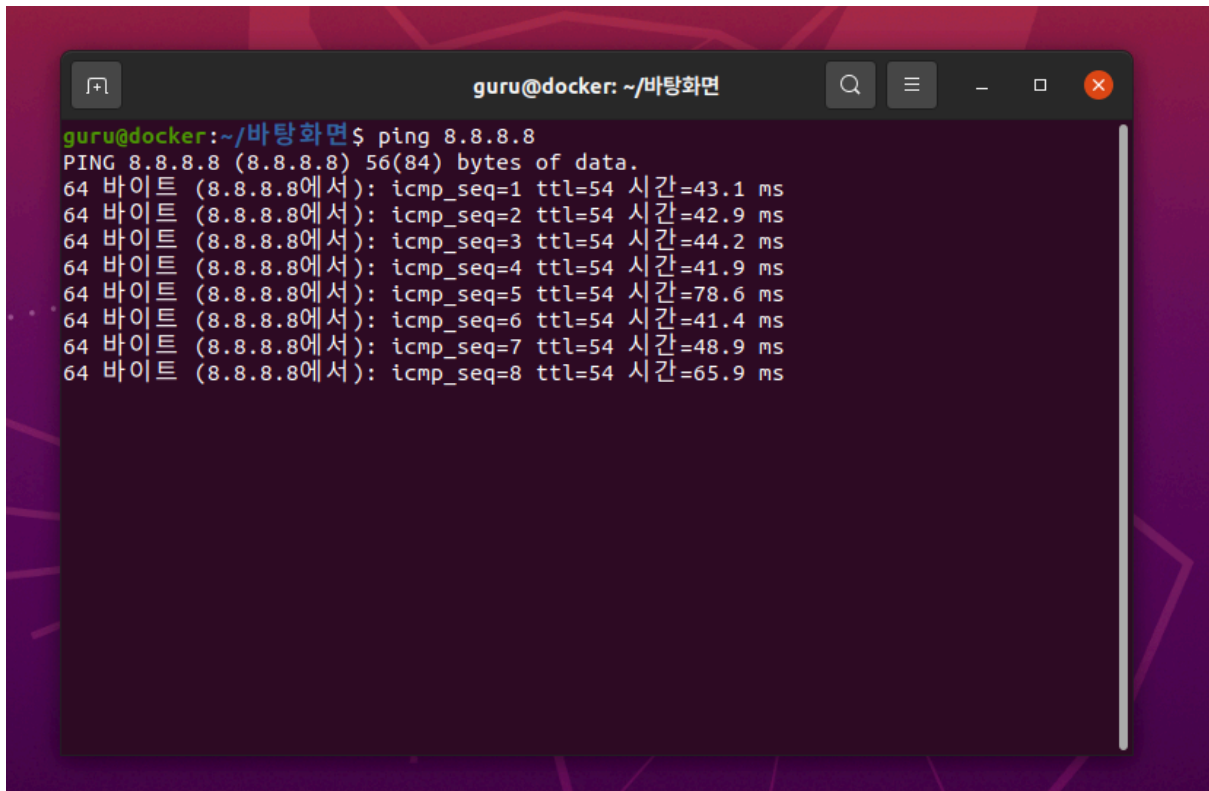


guru를 클릭하고 비밀번호 work를 입력



화면에서 우클릭한 후 Open in Terminal를 클릭한다

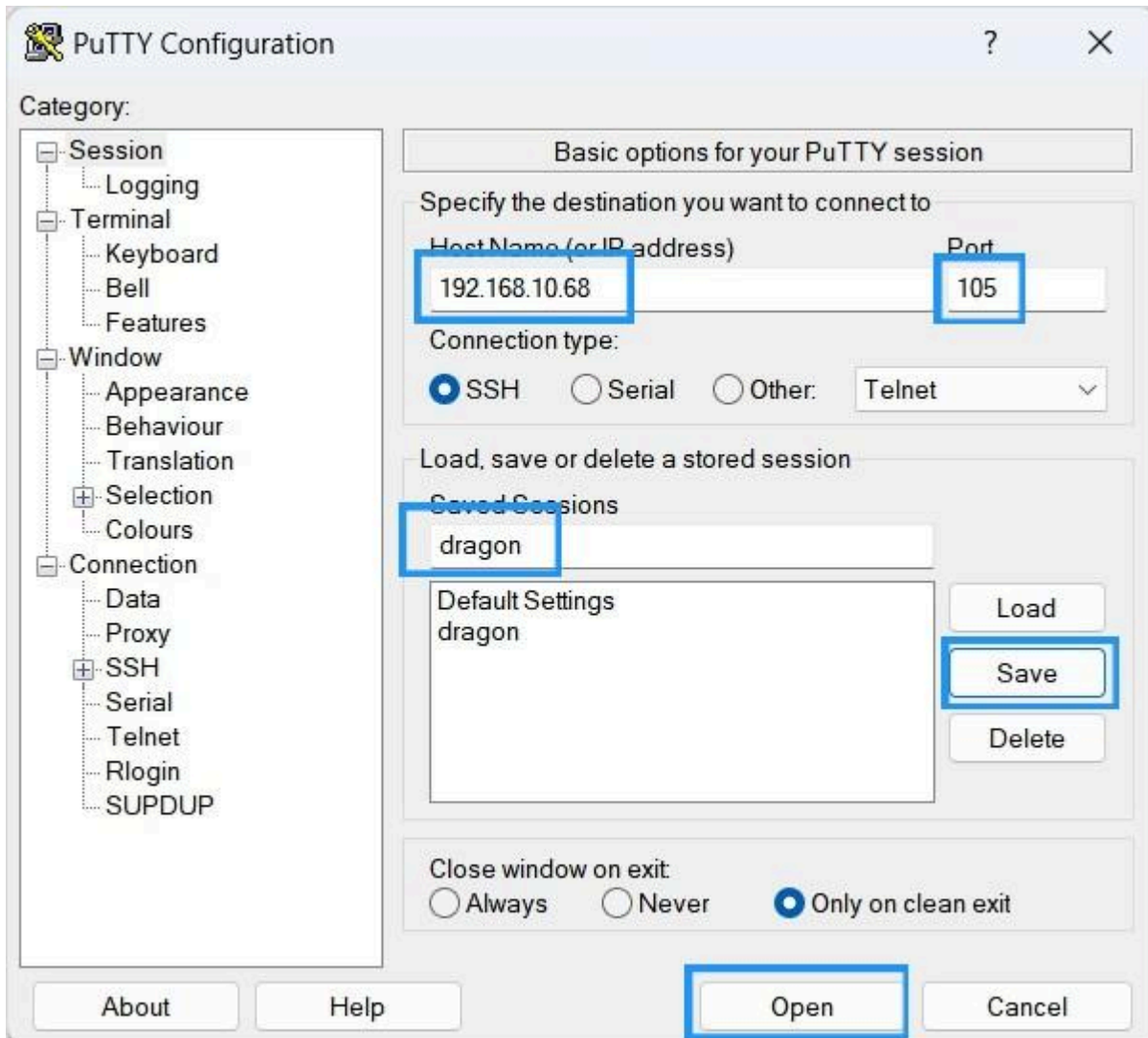




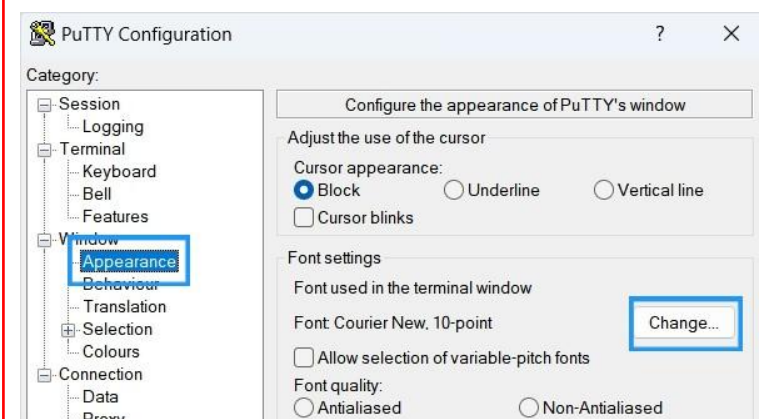
A terminal window titled "guru@docker: ~/바탕화면" with standard window controls. The terminal shows the execution of a ping command to 8.8.8.8. The output displays the size of the data packet (56(84) bytes), followed by eight lines of ping results. Each line shows the size of the received data (64 바이트), the source (8.8.8.8에서), the ICMP sequence number (icmp\_seq=1 to 8), the TTL (54), and the response time in milliseconds (시간=...). The response times are: 43.1 ms, 42.9 ms, 44.2 ms, 41.9 ms, 78.6 ms, 41.4 ms, 48.9 ms, and 65.9 ms.

```
guru@docker:~/바탕화면$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 바이트 (8.8.8.8에서): icmp_seq=1 ttl=54 시간=43.1 ms
64 바이트 (8.8.8.8에서): icmp_seq=2 ttl=54 시간=42.9 ms
64 바이트 (8.8.8.8에서): icmp_seq=3 ttl=54 시간=44.2 ms
64 바이트 (8.8.8.8에서): icmp_seq=4 ttl=54 시간=41.9 ms
64 바이트 (8.8.8.8에서): icmp_seq=5 ttl=54 시간=78.6 ms
64 바이트 (8.8.8.8에서): icmp_seq=6 ttl=54 시간=41.4 ms
64 바이트 (8.8.8.8에서): icmp_seq=7 ttl=54 시간=48.9 ms
64 바이트 (8.8.8.8에서): icmp_seq=8 ttl=54 시간=65.9 ms
```

Putty로 접속하기



\* 글자 크기 변경



리눅스에 로그인

```
guru@kafka: ~  
login as: guru  
guru@192.168.137.10's password:  
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-79-generic x86_64)  
  
* Documentation:  https://help.ubuntu.com  
* Management:    https://landscape.canonical.com  
* Support:        https://ubuntu.com/advantage  
  
* Introducing Expanded Security Maintenance for Applications.  
  Receive updates to over 25,000 software packages with your  
  Ubuntu Pro subscription. Free for personal use.  
  
  https://ubuntu.com/pro  
  
Expanded Security Maintenance for Applications is not enabled.  
  
219 updates can be applied immediately.  
174 of these updates are standard security updates.  
추가 업데이트를 확인하려면 apt list --upgradable 을 실행하세요 .  
  
Enable ESM Apps to receive additional future security updates.  
See https://ubuntu.com/esm or run: sudo pro status  
  
The list of available updates is more than a week old.  
To check for new updates run: sudo apt update  
Your Hardware Enablement Stack (HWE) is supported until April 2025.  
Last login: Fri Aug 25 09:04:52 2023 from 10.100.0.2  
guru@kafka:~$
```

ifconfig를 이용해서 ip 주소 확인하기

```
Last login: Fri Aug 25 09:04:52 2023 from 10.100.0.2  
guru@kafka:~$ ifconfig  
  
명령어 'ifconfig' 을(를) 찾을 수 없습니다. 그러나 다음을 통해 설치할 수 있습니다  
:  
  
sudo apt install net-tools  
  
guru@kafka:~$
```

ifconfig를 사용할 수 없을 경우 우선 net-tools를 설치 한 후 사용

```
guru@kafka:~$ sudo apt install net-tools
```

아래와 같이 오류가 날 경우

E: dpkg가 중단되었습니다. 수동으로 'dpkg --configure -a' 명령을 실행해 문제점을 바로잡으십시오.

```
guru@kafka:~$ -su
```

```
Password:password
```

```
root@kafka:~# dpkg --configure -a
```

```
root@kafka:~# apt install net-tools
```



```
guru@kafka:~$ ifconfig
```

```
Processing triggers for man-db (2.9.1-1) ...
root@kafka:~# ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.137.10 netmask 255.255.255.0 broadcast 192.168.137.255
    inet6 fe80::df5:cb9:2241:3dc8 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:14:3c:17 txqueuelen 1000 (Ethernet)
    RX packets 23209 bytes 34272386 (34.2 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 4864 bytes 363930 (363.9 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 169 bytes 14462 (14.4 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 169 bytes 14462 (14.4 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@kafka:~#
```

## 2-2 인스턴스에 자바 설치

```
guru@kafka:~$ sudo apt-get update
guru@kafka:~$ sudo apt-get install openjdk-8-jdk
```

JDK 버전 확인

```
guru@kafka:~$ java -version
```

## 2-3 카프카 설치

1) 카프카 바이너리 패키지 다운로드

```
guru@kafka:~$ wget https://archive.apache.org/dist/kafka/2.5.0/kafka_2.12-2.5.0.tgz
guru@kafka:~$ tar xvf kafka_2.12-2.5.0.tgz
guru@kafka:~$ ll
guru@kafka:~$ cd kafka_2.12-2.5.0
```

2) 카프카 브로커 힙 메모리 설정



~/bashrc 파일은 bash 셸이 실행될 때마다 반복적으로 구동되어 적용되는 파일

```
guru@kafka:~$ vi ~/.bashrc → vi 편집기로 .bashrc 파일을 연다
```

.bashrc 파일 편집

```
→ vi 화면에서 i 입력 -> insert 모드로 전환됨
# .bashrc

#Source global definitions
if [ - /etc/bashrc ]; then
    . /etc/bashrc
fi
export KAFKA_HEAP_OPTS="-Xmx400m -Xms400m" 입력

→ esc 클릭하고 :wq (입력한 내용 저장하고 vi 빠져나옴) 입력
```

source 명령어는 스크립트 파일을 수정한 후 수정된 값을 바로 적용하기 위해 사용

```
guru@kafka:~$ source ~/.bashrc
guru@kafka:~$ echo $KAFKA_HEAP_OPTS
-Xmx400m -Xms400m
```

## 2-4 카프카 실행

### 1) 카프카 브로커 실행 옵션 설정

advertised.listeners는 카프카 클라이언트 또는 커맨드 라인 툴을 브로커와 연결할 때 사용된다. 현재 접속하고 있는 인스턴스의 퍼블릭 IP와 카프카 기본 포트인 9092를 PLAINTEXT://와 함께 붙여넣고 advertised.listeners를 주석에서 해제한다.

```
root@kafka:~# cd kafka_2.12-2.5.0
root@kafka:~# vi config/server.properties
```

```
advertised.listeners=PLAINTEXT://192.168.0.2:9092 <-- 활성화
```

```
→ esc 클릭 :wq 입력
```

### 2) 주키퍼 실행

분산 코디네이션 서비스를 제공하는 주키퍼는 카프카의 클러스터 설정 리더 정보, 컨트롤러 정보를 담고 있어 카프카를 실행하는 데에 필요한 필수 애플리케이션이다. -daemon 옵션과 주키퍼 설정 경로인 config/zookeeper.properties와 함께 주키퍼 시작 스크립트인 bin/zookeeper-server-start.sh를 실행하면 주키퍼를 백그라운드에서 실행할 수 있다.

```
guru@kafka:~/kafka_2.12-2.5.0$ bin/zookeeper-server-start.sh -daemon
config/zookeeper.properties
```

jps는 JVM 프로세스 상태를 보는 도구로서 JVM 위에서 동작하는 주키퍼의 프로세스를 확인할 수 있다.

```
guru@kafka:~/kafka_2.12-2.5.0$ jps -vm
```

### 3) 카프카 브로커 실행 및 로그 확인

-daemon 옵션과 함께 카프카 브로커를 백그라운드 모드로 실행할 수 있다.

```
guru@kafka:~/kafka_2.12-2.5.0$ bin/kafka-server-start.sh -daemon
config/server.properties
```

주키퍼와 브로커 프로세스의 동작 여부 확인

```
guru@kafka:~/kafka_2.12-2.5.0$ jps -m
```

tail 명령어를 통해 로그를 확인하여 카프카 브로커가 정상 동작하는지 확인 가능

```
guru@kafka:~/kafka_2.12-2.5.0$ tail -f logs/server.log
```

#### \* 브로커 중지 및 재실행

브로커 중지하기

```
guru@kafka:~/kafka_2.12-2.5.0$ $ bin/kafka-server-stop.sh
```

브로커 재실행하기

```
guru@kafka:~/kafka_2.12-2.5.0$ bin/kafka-server-stop.sh && bin/kafka-server-start.sh
```

### 3. 로컬 컴퓨터에서 카프카와 통신 확인

#### 3-1 윈도우 환경에서 리눅스를 설치하고 카프카 설치하기

윈도우즈10부터는 WSL(Linux용 Windows 하위 시스템)과 리눅스 배포판(Ubuntu,openSUSE 등)을 설치하여 셸 환경에서 명령어를 실행할 수 있다.

<https://learn.microsoft.com/ko-kr/windows/wsl/install>

기본적으로 설치된 Linux 배포는 Ubuntu입니다.

필터링

## 필수 조건

아래 명령을 사용하려면 Windows 10 버전 2004 이상(빌드 19041 이상) 또는 Windows 11을 실행해야 합니다. 이전 버전을 사용 중인 경우 [수동 설치 페이지](#)를 참조하세요.

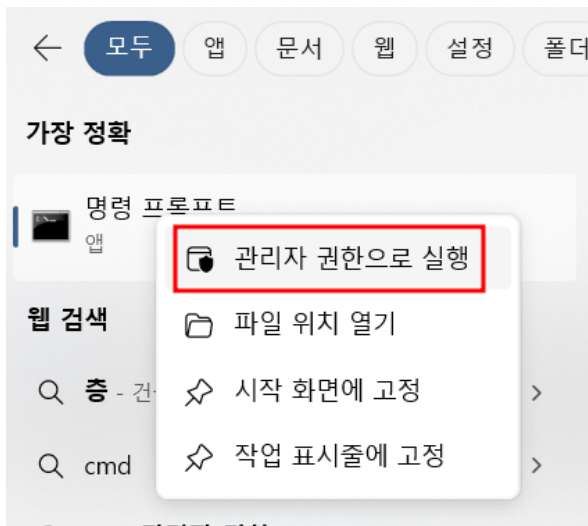
## WSL 설치 명령

이제 단일 명령으로 WSL을 실행하는 데 필요한 모든 항목을 설치할 수 있습니다. **관리자** 모드에서 PowerShell 또는 Windows 명령 프롬프트를 마우스 오른쪽 단추로 클릭하고 "관리자 권한으로 실행"을 선택하여 열고 `wsl --install` 명령을 입력한 다음 컴퓨터를 다시 시작합니다.

PowerShell

복사

```
wsl --install
```



(주의) Ubuntu 재설치 접속 오류가 발생할 경우 해결 방법

'C:\Users\drago\AppData\Local\Packages\CanonicalGroupLimited.Ubuntu\_79rhkp1fndgsc\LocalState\ext4.vhdx' 디스크를 WSL2에 연결하지 못했습니다. 지정된 파일을 찾을 수 없습니다. ← 와 같은 오류 발생시

```
wsl --unregister ubuntu
```

```
wsl --install
```

```
관리자: 명령 프롬프트 - wsl --install
Microsoft Windows [Version 10.0.22621.3296]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>wsl --install
설치 중: Ubuntu
[= 3.0%
```

username : dragon  
New password : 1234

```
dragon@DESKTOP-3HH4O5N: ~
Microsoft Windows [Version 10.0.22621.3296]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>wsl --unregister ubuntu
등록 취소 중입니다.
작업을 완료했습니다.

C:\Windows\System32>wsl --install
Ubuntu0이(가) 이미 설치되어 있습니다.
Ubuntu을(를) 시작하는 중...
Installing, this may take a few minutes...
Please create a default UNIX user account. The username does not need
For more information visit: https://aka.ms/wslusers
Enter new UNIX username: dragon
New password:
Retype new password:
passwd: password updated successfully
Installation successful!
To run a command as administrator (user "root"), use "sudo <command>"
See "man sudo_root" for details.

Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.146.1-microsoft-standard

* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage

This message is shown once a day. To disable it please create the
/home/dragon/.hushlogin file.
dragon@DESKTOP-3HH4O5N:~$
```

root 비밀번호 설정  
sudo passwd root  
[sudo] password for dragon:1234  
New password:1234

Retype password:1234

```
root@DESKTOP-3HH4O5N: ~ × + v
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

dragon@DESKTOP-3HH4O5N:~$ sudo passwd root
[sudo] password for dragon:
New password:
Retype new password:
passwd: password updated successfully
dragon@DESKTOP-3HH4O5N:~$ su -
Password:
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.146.1-microsoft-standard-

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:       https://ubuntu.com/advantage
```

root 계정 종료

```
~# exit
```

패키지 업데이트 및 업그레이드

```
$ sudo apt update && sudo apt upgrade
```

로컬 컴퓨터에 설치한 리눅스에 접속

### 3-2 자바 설치

```
$ sudo apt install openjdk-8-jdk-headless
$ java -version
```

### 3-3 카프카 설치

카프카 바이너리 패키지 다운로드

```
$ wget https://archive.apache.org/dist/kafka/2.5.0/kafka_2.12-2.5.0.tgz
$ tar xvf kafka_2.12-2.5.0.tgz
$ ll
$ cd kafka_2.12-2.5.0
```

### 3-5 접속 확인

```
$ bin/kafka-broker-api-versions.sh --bootstrap-server 192.168.10.68:9092
```

```
DESKTOP-3HH4O5N.:9092 (id: 0 rack: null) -> (  
  Produce(0): 0 to 8 [usable: 8],  
  Fetch(1): 0 to 11 [usable: 11],  
  ListOffsets(2): 0 to 5 [usable: 5],  
  Metadata(3): 0 to 9 [usable: 9],  
  LeaderAndIsr(4): 0 to 4 [usable: 4],  
  StopReplica(5): 0 to 2 [usable: 2],  
  UpdateMetadata(6): 0 to 6 [usable: 6],  
  ControlledShutdown(7): 0 to 3 [usable: 3],  
  OffsetCommit(8): 0 to 8 [usable: 8],  
  OffsetFetch(9): 0 to 7 [usable: 7],  
  FindCoordinator(10): 0 to 3 [usable: 3],  
  JoinGroup(11): 0 to 7 [usable: 7],  
  Heartbeat(12): 0 to 4 [usable: 4],  
  LeaveGroup(13): 0 to 4 [usable: 4],  
  SyncGroup(14): 0 to 5 [usable: 5],  
  DescribeGroups(15): 0 to 5 [usable: 5],  
  ListGroups(16): 0 to 3 [usable: 3],  
  SaslHandshake(17): 0 to 1 [usable: 1],  
  ApiVersions(18): 0 to 3 [usable: 3],  
  CreateTopics(19): 0 to 5 [usable: 5],  
  DeleteTopics(20): 0 to 4 [usable: 4],  
  DeleteRecords(21): 0 to 1 [usable: 1],  
  InitProducerId(22): 0 to 3 [usable: 3],  
  OffsetForLeaderEpoch(23): 0 to 3 [usable: 3],  
  AddPartitionsToTxn(24): 0 to 1 [usable: 1],  
  AddOffsetsToTxn(25): 0 to 1 [usable: 1],  
  EndTxn(26): 0 to 1 [usable: 1],  
  WriteTxnMarkers(27): 0 [usable: 0],  
  TxnOffsetCommit(28): 0 to 3 [usable: 3],  
  DescribeAcls(29): 0 to 2 [usable: 2],  
  CreateAcls(30): 0 to 2 [usable: 2],  
  DeleteAcls(31): 0 to 2 [usable: 2],  
  DescribeConfigs(32): 0 to 2 [usable: 2],  
  AlterConfigs(33): 0 to 1 [usable: 1],  
  AlterReplicaLogDirs(34): 0 to 1 [usable: 1],  
  DescribeLogDirs(35): 0 to 1 [usable: 1],  
  SaslAuthenticate(36): 0 to 2 [usable: 2],  
  CreatePartitions(37): 0 to 2 [usable: 2],  
  CreateDelegationToken(38): 0 to 2 [usable: 2],  
  RenewDelegationToken(39): 0 to 2 [usable: 2],  
  ExpireDelegationToken(40): 0 to 2 [usable: 2],  
  DescribeDelegationToken(41): 0 to 2 [usable: 2],  
)
```

```
DeleteGroups(42): 0 to 2 [usable: 2],
ElectLeaders(43): 0 to 2 [usable: 2],
IncrementalAlterConfigs(44): 0 to 1 [usable: 1],
AlterPartitionReassignments(45): 0 [usable: 0],
ListPartitionReassignments(46): 0 [usable: 0],
OffsetDelete(47): 0 [usable: 0]
```

)

## 4. 카프카에서 데이터 처리

### 4-1 토픽 생성 - 기본값 지정

```
①      ②      ③
$ bin/kafka-topics.sh --create --bootstrap-server 192.168.10.68:9092 --topic
hello.kafka
Created topic hello.kafka.
```

- ① --create 옵션으로 토픽을 생성하는 명령어라는 것을 명시한다.
- ② --bootstrap-server 에는 토픽을 생성할 카프카 클러스터를 구성하는 브로커들의 IP와 port를 적는다. 여기서는 1개의 카프카 브로커와 통신하므로 my-kafka;9092만 입력한다.
- ③ --topic 에서 토픽 이름 작성한다. 토픽 이름은 내부 데이터가 무엇이 있는지 유추가 가능할 정도로 자세히 적은 것을 추천한다. 카프카를 운영하다 보면 다양한 데이터 처리를 위해 수많은 토픽이 만들어지는데 이때 토픽 이름이 불명확하여 출처를 알 수 없으면 유지보수가 곤란해지기 때문이다.

토픽 생성 - 파티션 개수, 복제 개수, 토픽 데이터 유지 기간 옵션 지정

```
①
$ bin/kafka-topics.sh --create --bootstrap-server 192.168.10.68:9092 --partitions 3
--replication-factor 1 --config retention.ms=172800000 --topic hello.kafka.2
②      ③
Created topic hello.kafka.2.
```

- ① --partitions 는 파티션 개수를 지정할 수 있다. 파티션 최소 개수는 1개이다. 만약 이 옵션을 사용하지 않으면 카프카 브로커 설정파일(config/server.properties)에 있는 num.partitions 옵션값을 따라 생성된다.
- ② --replication-factor 에는 토픽의 파티션을 복제할 복제 개수를 적는다. 1은 복제를 하지 않고 사용한다는 의미이다. 2면 1개의 복제본을 사용하겠다는 의미이다. 파티션의 데이터는 각 브로커마다 저장된다. 한 개의 브로커에 장애가 발생하더라도 나머지 한 개 브로커에 저장된 데이터를 사용하여 안전하게 데이터 처리를 지속적으로 할 수 있다. 복제 개수의 최소 설정은 1이고 최대 설정은 통신하는 카프카 클러스터의 브로커 개수이다. 현재 실습하고 있는 실습용 카프카의 브로커 개수는 1개이므로 설정할 수 있는 최대 복제 설정은 1이다. 실제 업무환경에서는 3개 이상의 카프카 브로커로 운영하는 것이 일반적으로 2 또는 3으로 복제 개수를 설정하여 사용한다. 만약 이 설정을 명시적으로

하지 않으면 카프카 브로커 설정에 있는 `default.replication.factor` 옵션값에 따라서 생성한다.

③ `--config` 를 통해 `kafka-topics.sh` 명령에 포함되지 않은 추가적인 설정을 할 수 있다. `retention.ms` 는 토픽의 데이터를 유지하는 기간을 뜻한다. `172800000ms` 는 2일을 `ms`(밀리세컨드)단위로 나타낸 것이다. 2일이 지난 토픽의 데이터는 삭제된다.

#### 4-2 토픽 리스트 조회

```
$ bin/kafka-topics.sh --bootstrap-server 192.168.10.68:9092 --list
hello.kafka
hello.kafka.2
```

토픽 상세 조회

```
$ bin/kafka-topics.sh --bootstrap-server 192.168.10.68:9092 --describe --topic
hello.kafka.2
```

```
Topic: hello.kafka.2 PartitionCount: 3 ReplicationFactor: 1 Configs:
segment.bytes=1073741824,retention.ms=172800000
```

```
Topic: hello.kafka.2 Partition: 0 Leader: 0 Replicas: 0 Isr: 0
Topic: hello.kafka.2 Partition: 1 Leader: 0 Replicas: 0 Isr: 0
Topic: hello.kafka.2 Partition: 2 Leader: 0 Replicas: 0 Isr: 0
```

파티션 개수가 몇 개인지, 복제된 파티션이 위치한 브로커의 번호, 기타 토픽을 구성하는 설정들을 출력한다. 토픽이 가진 파티션의 리더가 현재 어느 브로커에 존재하는지도 같이 확인할 수 있다. `hello.kafka.2` 토픽의 정보를 보면 3개 파티션이 모두 **Leader**가 0으로 표시되어 있는데 0번 브로커에 위치하고 있음을 뜻한다.

#### 4-3 토픽 옵션 수정

```
$ bin/kafka-topics.sh --bootstrap-server 192.168.10.68:9092 --topic hello.kafka --alter
--partitions 4
```

① `--alter` 옵션과 `--partitions` 옵션을 함께 사용하여 파티션 개수를 변경할 수 있다. 토픽의 파티션을 늘릴 수 있지만 줄일 수는 없다.

수정 내용 확인하기

```
$ bin/kafka-topics.sh --bootstrap-server 192.168.10.68:9092 --topic hello.kafka
--describe
```

```
Topic: hello.kafka PartitionCount: 4 ReplicationFactor: 1 Configs:
segment.bytes=1073741824
```

```
Topic: hello.kafka Partition: 0 Leader: 0 Replicas: 0 Isr: 0
```



Topic: hello.kafka	Partition: 1	Leader: 0	Replicas: 0	Isr: 0
Topic: hello.kafka	Partition: 2	Leader: 0	Replicas: 0	Isr: 0
Topic: hello.kafka	Partition: 3	Leader: 0	Replicas: 0	Isr: 0

토픽 옵션 수정

```
$ bin/kafka-configs.sh --bootstrap-server 192.168.10.68:9092 --entity-type topics
--entity-name hello.kafka --alter --add-config retention.ms=86400000
```

①

Completed updating config for topic hello.kafka.

① retention.ms 를 수정하기 위해 kafka-configs.sh와 --alter, --add-config 옵션을 사용했다. --add-config 옵션을 사용하면 이미 존재하는 설정값은 변경하고 존재하지 않는 설정값은 신규로 추가한다.

수정 내용 확인하기

```
$ bin/kafka-configs.sh --bootstrap-server 192.168.10.68:9092 --entity-type topics
--entity-name hello.kafka --describe
```

Dynamic configs for topic hello.kafka are:

retention.ms=86400000 sensitive=false

synonyms={DYNAMIC\_TOPIC\_CONFIG:retention.ms=86400000}

#### 4-4 토픽에 데이터 넣기

kafka-console-producer.sh 명령어를 이용해서 생성된 hello.kafka 토픽에 데이터를 넣는다. 토픽에 넣은 데이터는 '레코드(record)'라고 부르며 메시지 키(key)와 메시지 값(value)로 이루어져 있다. 메시지 키 없이 메시지 값만 보내본다. 메시지 키는 자바의 null로 기본 설정되어 브로커로 전송된다.

```
$ bin/kafka-console-producer.sh --bootstrap-server 192.168.10.68:9092 --topic
hello.kafka
```

>hello

>kafka

>0

>1

>2

>3

>4

>5

← 빠져나오기 위해 Ctrl + c

메시지와 키를 가지는 레코드 전송하기

```
$ bin/kafka-console-producer.sh --bootstrap-server 192.168.10.68:9092 --topic hello.kafka --property "parse.key=true" --property "key.separator="
```

①

②

```
>key1:no1
>key2:no2
>key3:no3
```

① `parse.key`를 `true`로 두면 레코드를 전송할 때 메시지 키를 추가할 수 있다.  
② 메시지 키와 메시지 값을 구분하는 구분자를 선언한다. `key.separator`를 선언하지 않으면 기본 설정은 `Tab delimiter(\t)`이다. 그러므로 `key.separator`를 선언하지 않고 메시지를 보내려면 메시지 키를 작성하고 탭 키를 누른 뒤 메시지 값을 작성하고 엔터를 누른다. 여기서는 명시적으로 확인하기 위해 콜론(:)을 구분자로 선언했다. 만약 `key.separator`로 사용하는 구분자를 넣지 않고 엔터를 누르면 `KafkaException`과 함께 종료된다.

#### 4-5 토픽으로 전송한 데이터 읽기

`kafka-console-consumer.sh` 를 이용해서 `hello.kafka` 토픽으로 전송한 데이터 읽기

```
$ bin/kafka-console-consumer.sh --bootstrap-server 192.168.10.68:9092 --topic hello.kafka --from-beginning
```

```
no2
0
3
4
hello
1
2
5
kafka
no3
no1
```

#### 4-6 메시지 키와 메시지 값을 확인하고 싶다면 `--property` 옵션 사용

```
$ bin/kafka-console-consumer.sh --bootstrap-server 192.168.10.68:9092 --topic hello.kafka --property print.key=true --property key.separator="-" --group hello-group --from-beginning
```

```
$ bin/kafka-console-consumer.sh --bootstrap-server my-kafka:9092 \
--topic hello.kafka \
--property print.key=true \
--property key.separator="-" \
--group hello-group \
--from-beginning
```

①

②

③

```
key1-no1
null-4
null-5
null-0
null-1
null-2
null-3
null-hello
null-kafka
key2-no2
key3-no3
```

1 메시지 키를 확인하기 위해 **print.key**를 **true**로 설정했다. **print.key**를 설정하지 않으면 기본 설정값이 **false**이기 때문에 메시지 키를 확인할 수 없다.

2 메시지 키 값을 구분하기 위해 **key.separator**를 설정했다. **key.separator**를 설정하지 않으면 **tab delimiter(t)**가 기본값으로 설정되어 출력된다. 여기서는 대시(-)로 설정했으므로 <메시지 키>-<메시지 값> 형태로 출력된다.

3 **--group** 옵션을 통해 신규 컨슈머 그룹(**consumer group**)을 생성했다. 컨슈머 그룹은 1개 이상 컨슈머로 이루어져 있다. 이 컨슈머 그룹을 통해 가져간 토픽의 메시지는 가져간 메시지에 대해 커밋(**commit**)을 한다. 커밋이란 컨슈머가 특정 레코드까지 처리를 완료했다고 레코드의 오프셋 번호를 카프카 브로커에 저장하는 것이다. 커밋 정보는 **\_\_consumer\_offsets** 이름의 내부 토픽에 저장된다.

#### 4-7 생성된 컨슈머 그룹의 리스트 확인

**hello-group** 이름의 컨슈머 그룹으로 생성된 컨슈머로 **hello.kafka** 토픽의 데이터를 가져갔다. 컨슈머 그룹은 따로 생성하는 명령을 날리지 않고 컨슈머를 동작할 때 컨슈머 그룹 이름을 지정하면 새로 생성된다. 생성된 컨슈머 그룹의 리스트는 **kafka-consumer-groups.sh** 명령어로 확인할 수 있다.

```
$ bin/kafka-consumer-groups.sh --bootstrap-server 192.168.10.68:9092 --list
hello-group
```

컨슈머 그룹을 통해 현재 컨슈머 그룹이 몇 개나 생성되었는지, 어떤 이름의 컨슈머 그룹이 존재하는지 확인할 수 있다. 이렇게 확인한 컨슈머 그룹 이름을 토대로 컨슈머 그룹이 어떤 토픽의 데이터를 가져가는지 확인할 때 쓰인다.

kafka-consumer-groups.sh

```
$ bin/kafka-consumer-groups.sh --bootstrap-server 192.168.10.68:9092 --group
hello-group --describe
```

Consumer group 'hello-group' has no active members.

GROUP CONSUMER-ID CLIENT-ID	TOPIC HOST	PARTITION	CURRENT-OFFSET	LOG-END-OFFSET	LAG
hello-group -	hello.kafka	3	7	7	0
hello-group -	hello.kafka	2	8	8	0
hello-group -	hello.kafka	1	12	12	0
hello-group	hello.kafka	0	9	9	0

kafka-verifiable-producer,consumer.sh

kafka-verifiable로 시작하는 2개의 스크립트를 사용하면 String 타입 메시지 값을 코드 없이 주고받을 수 있다. 카프카 클러스터 설치가 완료된 이후에 토픽에 데이터를 전송하여 간단한 네트워크 통신 테스트를 할 때 유용하다.

**\$ bin/kafka-verifiable-producer.sh --bootstrap-server 192.168.10.68:9092  
--max-messages 10 --topic verify-test**

\$ bin/kafka-verifiable-producer.sh --bootstrap-server my-kafka:9092 \  
--max-messages 10 \  
--topic verify-test

①  
②  
③

```
{ "timestamp": 1712669057484, "name": "startup_complete" }
{ "timestamp": 1712669057963, "name": "producer_send_success", "key": null, "value": "0", "offset": 10, "topic": "verify-test", "partition": 0 }
{ "timestamp": 1712669057969, "name": "producer_send_success", "key": null, "value": "1", "offset": 11, "topic": "verify-test", "partition": 0 }
{ "timestamp": 1712669057969, "name": "producer_send_success", "key": null, "value": "2", "offset": 12, "topic": "verify-test", "partition": 0 }
{ "timestamp": 1712669057970, "name": "producer_send_success", "key": null, "value": "3", "offset": 13, "topic": "verify-test", "partition": 0 }
{ "timestamp": 1712669057970, "name": "producer_send_success", "key": null, "value": "4", "offset": 14, "topic": "verify-test", "partition": 0 }
{ "timestamp": 1712669057971, "name": "producer_send_success", "key": null, "value": "5", "offset": 15, "topic": "verify-test", "partition": 0 }
{ "timestamp": 1712669057971, "name": "producer_send_success", "key": null, "value": "6", "offset": 16, "topic": "verify-test", "partition": 0 }
{ "timestamp": 1712669057971, "name": "producer_send_success", "key": null, "value": "7", "offset": 17, "topic": "verify-test", "partition": 0 }
{ "timestamp": 1712669057972, "name": "producer_send_success", "key": null, "value": "8", "offset": 18, "topic": "verify-test", "partition": 0 }
{ "timestamp": 1712669057974, "name": "producer_send_success", "key": null, "value": "9", "offset": 19, "topic": "verify-test", "partition": 0 }
{ "timestamp": 1712669057998, "name": "shutdown_complete" }
```

```
{"timestamp":1712669058000,"name":"tool_data","sent":10,"acked":10,"target_throughput":-1,"avg_throughput":19.193857965451055}
```

6

- 1 통신하고자 하는 클러스터 호스트와 포트를 `--bootstrap-server` 옵션으로 입력한다.
- 2 `--max-messages`는 `kafka-verifiable-producer.sh`로 보낸 데이터 개수를 지정한다. 만약 `-1`을 옵션값으로 입력하면 `kafka-verifiable-producer.sh`가 종료될 때까지 계속 데이터를 토픽으로 보낸다.
- 3 데이터를 받을 대상 토픽을 입력한다.
- 4 최초 실행 시점이 `startup_complete`와 함께 출력된다.
- 5 메시지별로 보낸 시간과 메시지 키, 메시지 값, 토픽, 저장된 파티션, 저장된 오프셋 번호가 출력된다.
- 6 10개 데이터가 모두 전송된 이후 통계값이 출력된다. 평균 처리량을 확인할 수 있다.

전송한 데이터는 `kafka-verifiable-consumer.sh`로 확인할 수 있다.

```
$ bin/kafka-verifiable-consumer.sh --bootstrap-server 192.168.10.68:9092 --topic verify-test --group-id test-group
```

```
$ bin/kafka-verifiable-consumer.sh --bootstrap-server my-kafka:9092 \
--topic verify-test \
--group-id test-group
```

①  
②  
③

```
{"timestamp":1712669299434,"name":"startup_complete"}
{"timestamp":1712669300419,"name":"partitions_assigned","partitions":[{"topic":"verify-test","partition":0}]}
{"timestamp":1712669300671,"name":"records_consumed","count":20,"partitions":[{"topic":"verify-test","partition":0,"count":20,"minOffset":0,"maxOffset":19}]}
{"timestamp":1712669300704,"name":"offsets_committed","offsets":[{"topic":"verify-test","partition":0,"offset":20}], "success":true}
```

- 1 통신하고자 하는 클러스터 호스트와 포트를 `--bootstrap-server` 옵션으로 입력한다.
- 2 데이터를 가져오고자 하는 토픽인 `verify-test`를 `--topic` 옵션값으로 입력한다.
3. 컨슈머 그룹을 지정한다. `test-group`을 `--group-id` 옵션값으로 입력한다.

#### 4-8 적재된 토픽의 데이터 지우기

`kafa-delete-records.sh` 는 이미 적재된 토픽의 데이터 중 가장 오래된 데이터(가장 낮은 숫자의 오프셋)부터 특정 시점의 오프셋까지 삭제할 수 있다. 예를 들어 `test` 토픽의 0번 파티션에 0부터 100까지 데이터가 들어 있다고 가정하자. `test` 토픽의 0번 파티션에 저장된 데이터 중 0부터 30 오프셋 데이터까지 지우고 싶다면 다음과 같이 입력할 수 있다.

```
$ vi delete-topic.json
{"partitions":[{"topic":"test","partition":0,"offset":50}], "version":1}
```

```
$ bin/kafka-delete-records.sh --bootstrap-server my-kafka:9092 --offset-json-file
```

```
delete-topic.json
```

```
Executing records delete operation  
Records delete operation completed:  
partition: test-0 low_watermark: 50
```

## 5. 카프카 클라이언트

카프카 클러스터에 명령을 내리거나 데이터를 송수신하기 위해 카프카 클라이언트 라이브러리는 카프카 프로듀서, 컨슈머, 어드민 클라이언트를 제공하는 카프카 클라이언트를 사용하여 애플리케이션을 개발한다.

### 5-1 토픽 생성

```
$ bin/kafka-topics.sh --bootstrap-server 192.168.10.68:9092 --create --topic test  
--partitions 3  
Created topic test.
```

### 5-2 카프카 프로듀서 프로젝트 생성

메시지 키를 가지지 않는 데이터를 전송하는 프로듀서  
SimpleProducer

필수 옵션

bootstrap.servers	프로듀서가 데이터를 전송할 대상 카프카 클러스터에 속한 브로커의 호스트 이름:포트를 1개 이상 작성한다. 2개 이상 브로커 정보를 입력하여 일부 브로커에 이슈가 발생하더라도 접속하는 데에 이슈가 없도록 설정 가능하다.
key.serializer	레코드의 메시지 키를 직렬화하는 클래스를 지정한다.
value.serializer	레코드의 메시지 값을 직렬화하는 클래스를 지정한다.

선택 옵션

group.id	컨슈머 그룹 아이디를 지정한다.
auto.offset.reset	컨슈머 그룹이 특정 파티션을 읽을 때 저장된 컨슈머 오프셋이 없는 경우 어느 오프셋부터 읽을지 선택하는 옵션이다
enable.auto.commit	자동 커밋으로 할지 수동 커밋으로 할지 선택한다. 기본값은 false이다.
auto.commit.interval.ms	자동 커밋(enable.auto.commit=true)일 경우 오프셋 커밋 간격을 지정한다. 기본값은 5000(5초)이다.
max.poll.records	poll() 메서드를 통해 반환되는 레코드 개수를 지정한다.

	기본값은 500이다.
session.timeout.ms	컨슈머가 브로커와 연결이 끊기는 최대 시간이다. 기본값은 1000(10초)
heartbeat.interval.ms	하트비트를 전송하는 시간 간격이다. 기본값은 3000(3초)
max.poll.interval.ms	poll()메서드를 호출하는 간격의 최대 시간을 지정한다. 기본값은 300000(5분)
isolation.level	트랜잭션 프로듀서가 레코드를 트랜잭션 단위로 보낼 경우 사용한다.

### 5-3 SimpleProducer 클래스 작성 및 실행

```
package kr.kafka;

import java.util.Properties;

import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.serialization.StringSerializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class SimpleProducer {
    private final static Logger logger =
    LoggerFactory.getLogger(SimpleProducer.class);
    /*
    프로듀서는 생성한 레코드를 전송하기 위해 전송하고자 하는 토픽을 알고 있어야 한다.
    토픽을 지정하지 않고서는 데이터를 전송할 수 없기 때문이다. 토픽 이름은
    ProducerRecord 인스턴스를 생성할 때 사용된다.
    */
    private final static String TOPIC_NAME = "test";
    /*
    전송하고자 하는 카프카 클러스터 서버의 host와 ip를 지정한다. 여기서는 실습용 카프카
    브로커가 1대이므로 1개의 host와 ip명을 지정한다.
    */
    private final static String BOOTSTRAP_SERVERS = "192.168.10.68:9092";

    public static void main(String[] args) {
    /*
    Properties에는 KafkaProducer 인스턴스를 생성하기 위한 프로듀서 옵션들을 key/value
    값으로 선언한다. 필수 옵션을 반드시 선언해야 하며, 선택 옵션은 선언하지 않아도 된다.
    선택 옵션을 선언하지 않으면 기본 옵션값으로 설정되어 동작한다. 그러므로 선택 옵션을
    선언하지 않더라도 선택 옵션에 무엇이 있는지, 옵션값은 무엇인지에 대해서는 알고
    있어야 한다.
    */
        Properties configs = new Properties();
        configs.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
        BOOTSTRAP_SERVERS);
    /*
```

메시지 키, 메시지 값을 직렬화하기 위한 직렬화 클래스를 선언한다. 여기서는 `String` 객체를 전송하기 위해 `String`을 직렬화하는 클래스인 카프카 라이브러리의 `StringSerializer`를 사용한다.

```
*/
        configs.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
StringSerializer.class.getName());
        configs.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
StringSerializer.class.getName());
/*
Properties를 KafkaProducer의 생성 파라미터로 추가하여 인스턴스를 생성한다. producer
인스턴스는 ProducerRecord를 전송할 때 사용된다.
```

```
*/
        KafkaProducer<String, String> producer = new KafkaProducer<String,
String>(configs);
```

```
/*
메시지 값을 선언한다.
```

```
*/
        String messageValue = "testMessage";
```

```
/*
카프카 브로커로 데이터를 보내기 위해 ProducerRecord를 생성한다. 메시지 키는 따로
선언하지 않았으므로 null로 설정되어 전송된다. ProducerRecord를 생성할 때 생성자에
2개의 제네릭 값이 들어가는데, 이 값은 메시지 키와 메시지 값의 타입을 뜻한다.
```

```
*/
        ProducerRecord<String, String> record = new
ProducerRecord<>(TOPIC_NAME, messageValue);
```

```
/*
프로듀서에서 send()는 즉각적인 전송을 뜻하는 것이 아니라, 파라미터로 들어간
record를 프로듀서 내부에 가지고 있다가 배치 형태로 묶어서 브로커에 전송한다. 이러한
전송 방식을 '배치 전송'이라고 부른다. 배치 전송을 통해 카프카는 타 메시지 플랫폼과
차별화된 전송 속도를 가지게 되었다.
```

```
*/
        producer.send(record);
        logger.info("{} ", record);
```

```
/*
flush()를 통해 프로듀서 내부 버퍼에 가지고 있던 레코드 배치를 브로커로 전송한다.
```

```
*/
        producer.flush();
        producer.close();
```

```
    }
}
```

```
[main] INFO kr.kafka.SimpleProducer -
ProducerRecord(topic=test, partition=null,
headers=RecordHeaders(headers = [], isReadOnly = true),
key=null, value=testMessage, timestamp=null)
[main] INFO org.apache.kafka.clients.producer.KafkaProducer -
[Producer clientId=producer-1] Closing the Kafka producer
with timeoutMillis = 9223372036854775807 ms.
```

#### 5-4 토픽에 있는 모든 레코드 확인



```
$ bin/kafka-console-consumer.sh --bootstrap-server 192.168.10.68:9092 --topic test
--from-beginning
testMessage
```

## 5-5 메시지 키를 가지는 데이터를 전송하는 프로듀서

SimpleProducer2 클래스 작성 및 실행

```
package kr.kafka;

import java.util.Properties;

import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.serialization.StringSerializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class SimpleProducer2 {
    private final static Logger logger =
        LoggerFactory.getLogger(SimpleProducer2.class);
    private final static String TOPIC_NAME = "test";
    private final static String BOOTSTRAP_SERVERS = ":9092";

    public static void main(String[] args) {
        Properties configs = new Properties();
        configs.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
            BOOTSTRAP_SERVERS);
        configs.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
            StringSerializer.class.getName());
        configs.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
            StringSerializer.class.getName());

        KafkaProducer<String, String> producer = new KafkaProducer<String,
            String>(configs);

        //메시지 키를 가진 데이터를 전송하는 프로듀서
        ProducerRecord<String, String> record = new
        ProducerRecord<>(TOPIC_NAME, "Pangyo", "23");
        producer.send(record);
        logger.info("{} ", record);

        producer.flush();
        producer.close();
    }
}
```

```
[main] INFO kr.kafka.SimpleProducer2 -
ProducerRecord(topic=test, partition=null,
```

```
headers=RecordHeaders(headers = [], isReadOnly = true),
key=Pangyo, value=23, timestamp=null)
[main] INFO org.apache.kafka.clients.producer.KafkaProducer -
[Producer clientId=producer-1] Closing the Kafka producer
with timeoutMillis = 9223372036854775807 ms.
```

**5-6** 메시지 키와 메시지 값이 **key.separator** 설정값을 기준으로 나뉘어 한 줄로 출력

```
$ bin/kafka-console-consumer.sh --bootstrap-server 192.168.10.68:9092 --topic test
--property print.key=true --property key.separator="-" --from-beginning
null-testMessage
Pangyo-23
```

**5-7** 카프카 컨슈머 프로젝트 생성

SimpleConsumer 클래스 작성 및 실행

```
package kr.kafka;

import java.time.Duration;
import java.util.Arrays;
import java.util.Properties;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.apache.kafka.common.serialization.StringSerializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class SimpleConsumer {
    private final static Logger logger =
LoggerFactory.getLogger(SimpleConsumer.class);
    private final static String TOPIC_NAME = "test";
    private final static String BOOTSTRAP_SERVERS = "192.168.10.68:9092";
/*
컨슈머 그룹 이름 선언한다. 컨슈머 그룹을 통해 컨슈머의 목적을 구분할 수 있다. 예를
들어, email 발송 처리를 하는 애플리케이션이라면 email-application-group으로 지정하여
동일한 역할을 하는 컨슈머를 묶어 관리할 수 있다. 컨슈머 그룹을 기준으로 컨슈머
오프셋을 관리하기 때문에 subscribe() 메서드를 사용하여 토픽을 구독하는 경우에는
컨슈머 그룹을 선언해야 한다. 컨슈머가 중단되거나 재시작되더라도 컨슈머 그룹의
컨슈머 오프셋을 기준으로 이후 데이터 처리를 하기 때문이다. 컨슈머 그룹을 선언하지
않으면 어떤 그룹에도 속하지 않는 컨슈머로 동작하게 된다.
*/
    private final static String GROUP_ID = "test-group";
```

```

        public static void main(String[] args) {
            Properties configs = new Properties();
            configs.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
BOOTSTRAP_SERVERS);
            configs.put(ConsumerConfig.GROUP_ID_CONFIG, GROUP_ID);
/*
프로듀서가 직렬화하여 전송한 데이터를 역직렬화하기 위해 역직렬화 클래스를
지정한다.
*/
            configs.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
            configs.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());

            KafkaConsumer<String, String> consumer = new KafkaConsumer<String,
String>(configs);
/*
컨슈머에게 토픽을 할당하기 위해 subscribe() 메서드를 사용한다. 이 메서드는 Collection
타입의 String 값들을 받는데, 1개 이상의 토픽 이름을 받을 수 있다.
*/
            consumer.subscribe(Arrays.asList(TOPIC_NAME));
/*
컨슈머는 poll() 메서드를 호출하여 데이터를 가져와서 처리한다. 지속적으로 데이터를
처리하기 위해서 반복 호출을 해야 한다.
*/
            while(true) {
/*
컨슈머는 poll()메서드를 통해 ConsumerRecord 리스트를 반환한다. poll() 메서드는
Duration 타입을 인자로 받는다. 이 인자 값은 브로커로부터 데이터를 가져올 때 컨슈머
버퍼에 데이터를 기다리기 위한 타임아웃 간격을 뜻한다.
*/
                ConsumerRecords<String, String> records =
consumer.poll(Duration.ofSeconds(1));
/*
for loop를 통해 poll() 메서드가 반환한 ConsumerRecord 데이터들을 순차적으로
처리한다.
*/
                for(ConsumerRecord<String,String> record : records) {
                    logger.info("{} ",record);
                }
            }
        }
    }
}

```

카프카 컨슈머 애플리케이션이 실행되면서 카프카 라이브러리 로그가 출력됨과 동시에 컨슈머가 **test** 토픽을 구독하면서 컨슈머는 브로커로부터 **polling**을 시작한다.

**test**에 데이터를 넣어 주기 위해 **kafka-console-producer** 명령으로 데이터를 넣는다

```
$ bin/kafka-console-producer.sh --bootstrap-server 192.168.10.68:9092 --topic test
> testMessage
```

출력 화면에 polling을 통해 가져온 정보가 로그로 남았다.

```
[main] INFO kr.kafka.consumer.SimpleConsumer -
ConsumerRecord(topic = test, partition = 2, leaderEpoch = 0,
offset = 0, CreateTime = 1714898369689, serialized key size =
-1, serialized value size = 11, headers =
RecordHeaders(headers = [], isReadOnly = false), key = null,
value = testMessage)
```

## 5-8 어드민 API

카프카 클라이언트에서는 내부 옵션들을 설정하거나 조회하기 위해 **AdminClient** 클래스를 제공한다.

KafkaAdminClient 메서드명	설명
describeCluster(DescribeClusterOptions options)	브로커의 정보 조회
listTopics(ListTopicsOptions options)	토픽 리스트 조회
listConsumerGroups(ListConsumerGroupsOptions options)	컨슈머 그룹 조회
createTopics(Collection<NewTopic> newTopics, CreateTopicsOptions options)	파티션 개수 변경
createAcls(Collection<AclBinding> acls, CreateAclsOptions options)	접근 제어 규칙 생성

브로커 정보 조회

```
package kr.kafka.admin;

import org.apache.kafka.clients.admin.*;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.Node;
import org.apache.kafka.common.config.ConfigResource;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Collections;
import java.util.Map;
import java.util.Optional;
import java.util.Properties;

public class KafkaAdminClient {
    private final static Logger logger = LoggerFactory.getLogger(KafkaAdminClient.class);
```

```

private final static String BOOTSTRAP_SERVERS = "192.168.10.68:9092";

public static void main(String[] args) throws Exception {

    Properties configs = new Properties();
    configs.put(ProducerConfig.BootstrapServersConfig,
BOOTSTRAP_SERVERS);
    AdminClient admin = AdminClient.create(configs);
    logger.info("== Get broker information");
    for (Node node : admin.describeCluster().nodes().get()) {
        logger.info("node : {}", node);
        ConfigResource cr = new ConfigResource(ConfigResource.Type.BROKER,
node.idString());
        DescribeConfigsResult describeConfigs =
admin.describeConfigs(Collections.singleton(cr));
        describeConfigs.all().get().forEach((broker, config) -> {
            config.entries().forEach(configEntry -> logger.info(configEntry.name() + " = " +
configEntry.value()));
        });
    }

    logger.info("== Get default num.partitions");
    for (Node node : admin.describeCluster().nodes().get()) {
        ConfigResource cr = new ConfigResource(ConfigResource.Type.BROKER,
node.idString());
        DescribeConfigsResult describeConfigs =
admin.describeConfigs(Collections.singleton(cr));
        Config config = describeConfigs.all().get().get(cr);
        Optional<ConfigEntry> optionalConfigEntry = config.entries().stream().filter(v ->
v.name().equals("num.partitions")).findFirst();
        ConfigEntry numPartitionConfig =
optionalConfigEntry.orElseThrow(Exception::new);
        logger.info("{} ", numPartitionConfig.value());
    }

    logger.info("== Topic list");
    for (TopicListing topicListing : admin.listTopics().listings().get()) {
        logger.info("{} ", topicListing.toString());
    }

    logger.info("== test topic information");
    Map<String, TopicDescription> topicInformation =
admin.describeTopics(Collections.singletonList("test")).all().get();
    logger.info("{} ", topicInformation);

    logger.info("== Consumer group list");
    ListConsumerGroupsResult listConsumerGroups = admin.listConsumerGroups();
    listConsumerGroups.all().get().forEach(v -> {
        logger.info("{} ", v);
    });

    admin.close();
}
}

```

## 출력 로그 확인

토픽 설정을 조회하는 어드민 **API**의 결과는 파티션 개수, 파티션의 위치, 리더 파티션의 위치 등을 확인할 수 있다.

```
[main] INFO kr.kafka.admin.KafkaAdminClient - == test topic information
[main] INFO kr.kafka.admin.KafkaAdminClient -
{test=(name=test, internal=false, partitions=(partition=0, leader=192.168.0.2:9092 (id: 0 rack: null), replicas=192.168.0.2:9092 (id: 0 rack: null), isr=192.168.0.2:9092 (id: 0 rack: null)), (partition=1, leader=192.168.0.2:9092 (id: 0 rack: null), replicas=192.168.0.2:9092 (id: 0 rack: null), isr=192.168.0.2:9092 (id: 0 rack: null)), (partition=2, leader=192.168.0.2:9092 (id: 0 rack: null), replicas=192.168.0.2:9092 (id: 0 rack: null), isr=192.168.0.2:9092 (id: 0 rack: null))), authorizedOperations=null)}
[main] INFO kr.kafka.admin.KafkaAdminClient - == Consumer group list
[main] INFO kr.kafka.admin.KafkaAdminClient -
(groupId='console-consumer-82926', isSimpleConsumerGroup=false)
```

## 6 카프카 실전 프로젝트

### 6-1 웹 페이지 이벤트 적재 파이프라인 생성

웹 페이지에서 생성되는 이벤트들을 분석하기 위해 엘라스틱서치에 적재하는 파이프라인을 만드는 프로젝트

아키텍처를 구현하기 위해 필요한 작업 리스트

- 1) 엘라스틱서치, 키바나, 로그스태시 설치
- 2) 토픽 생성
- 3) 이벤트 수집 웹 페이지 개발
- 4) REST API 프로듀서 애플리케이션 개발
- 5) 로그스태시를 이용해서 데이터를 엘라스틱서치에 적재

### 6-2 엘라스틱서치, 키바나, 로그스태시 설치

엘라스틱서치(7.7.0), 키바나(7.7.0), 로그스태시(8.0.0)를 window 버전으로 내려받아 압축을 풀고 실행

로그스태시 설정

C:\elasticsearch\logstash-8.0.0\bin\logstash.conf 생성

```

input {
  kafka {
    bootstrap_servers => "192.168.10.68:9092"
    topics => "select-color"
    consumer_threads => 3
  }
}

output {
  elasticsearch {
    hosts => ["localhost:9200"]
    index => "kafka-elastic-test2"
  }
}

```

C:\elasticsearch\logstash-8.0.0\config\pipeline.yml 편집

```

- pipeline.id: kafka-test-logs
  path.config: "logstash.conf"

```

### 6-3 토픽 생성

```

$ bin/kafka-topics.sh --create --bootstrap-server 192.168.10.68:9092
--replication-factor 1 --partitions 3 --topic select-color
Created topic select-color.

```

### 6-4 이벤트 수집 웹 페이지 개발

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Favorite color select webpage</title>
  <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <script>
    function selectedColor(color_name) {
      if($("#user_name").val().length == 0){
        alert("이름을 입력하세요");
      }else{
        var url =
'http://localhost:8080/api/select?color='+color_name+'&user='
+ $("#user_name").val();
        $.get(url)
      }
    }
  </script>

```

```

</head>
<body>
  <div style="width:100%; height:50%;">
    <h1>이름을 입력하세요.</h1>
    <label for="user_name">Name:</label>
    <input type="text" id="user_name">
    <h1>마음에 드는 색상을 고르세요.</h1>
    <ul>
      <li><button type="button"
onclick="selectedColor('red')">red</button></li>
      <li><button type="button"
onclick="selectedColor('yellow')">yellow</button></li>
      <li><button type="button"
onclick="selectedColor('green')">green</button></li>
      <li><button type="button"
onclick="selectedColor('blue')">blue</button></li>
    </ul>
  </div>
</body>
</html>

```

## 6-5 REST API 프로듀서 애플리케이션 개발

```

package kr.kafka;

import com.google.gson.Gson;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.kafka.support.SendResult;
import org.springframework.util.concurrent.ListenableFutureCallback;
import org.springframework.web.bind.annotation.*;

import java.text.SimpleDateFormat;
import java.util.Date;

@RestController
@CrossOrigin(origins = "*", allowedHeaders = "*")
public class ProduceController {

    private final Logger logger = LoggerFactory.getLogger(ProduceController.class);

    private final KafkaTemplate<String, String> kafkaTemplate;

    public ProduceController(KafkaTemplate<String, String> kafkaTemplate) {
        this.kafkaTemplate = kafkaTemplate;
    }

    @GetMapping("/api/select")
    public void selectColor(

```



```

        @RequestHeader("user-agent") String userAgentName,
        @RequestParam(value = "color")String colorName,
        @RequestParam(value = "user")String userName) {
    SimpleDateFormat sdfDate = new
SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSSZZ");
    Date now = new Date();
    Gson gson = new Gson();
    UserEventVO userEventVO = new UserEventVO(sdfDate.format(now),
userAgentName, colorName,userName);
    String jsonColorLog = gson.toJson(userEventVO);
    kafkaTemplate.send("select-color", jsonColorLog).addCallback(new
ListenableFutureCallback<SendResult<String, String>>() {
        @Override
        public void onSuccess(SendResult<String, String> result) {
            logger.info("->");
            logger.info(result.toString());
        }

        @Override
        public void onFailure(Throwable ex) {
            logger.error(ex.getMessage(), ex);
        }
    });
}
}
}

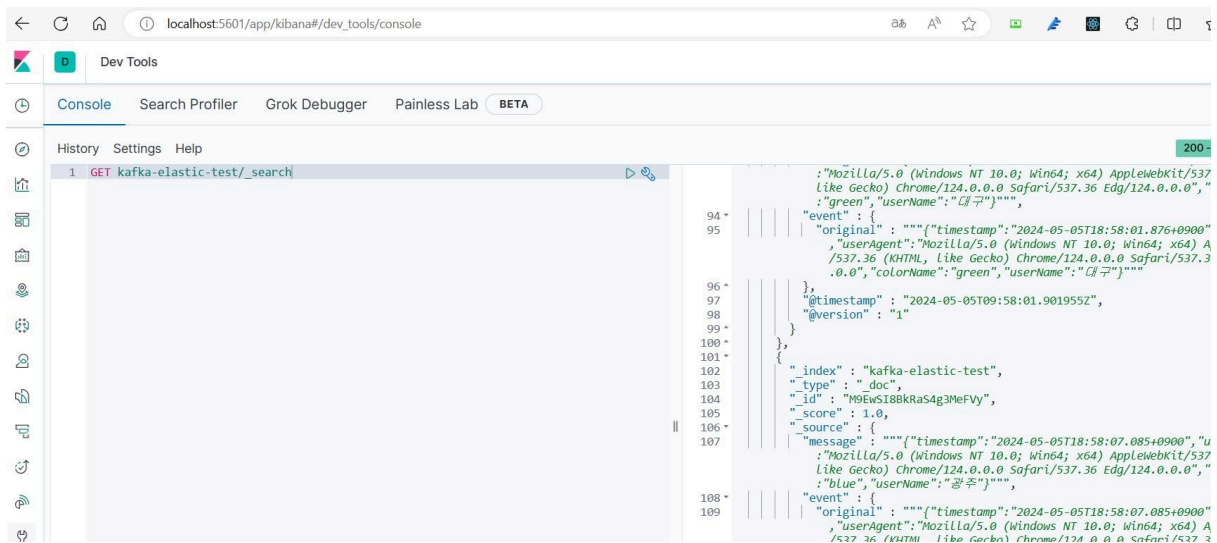
```

## 6-6 로그스태시를 이용해서 데이터를 엘라스틱서치에 적재

키바나를 통해 엘라스틱서치에 적재된 데이터 확인

[http://localhost:5601/app/kibana#/dev\\_tools/console](http://localhost:5601/app/kibana#/dev_tools/console) 로 접속

GET kafka-elastic-test/\_search



## 6-7 엘라스틱 서치에 저장된 데이터를 웹어플리케이션에서 확인

```

@GetMapping("/elastic/search")
@ResponseBody
public String get() throws IOException {

    RestClient restClient = RestClient.builder(
        new HttpHost("localhost", 9200, "http"),
        new HttpHost("localhost", 9201,
"http")).build();
    Request request = new Request("GET",
"/kafka-elastic-test/_search");
    Response response = restClient.performRequest(request);
    restClient.close();

    return EntityUtils.toString(response.getEntity());
}

```

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>엘라스틱 서치에 저장된 데이터 확인하기</title>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script type="text/javascript">
$(function(){
    function selectList(){
        $.ajax({
            url: '/elastic/search',
            dataType: 'json',
            success: function(param) {
                //table의 내부 내용물 제거 (초기화)
                $('#output').empty();

                $(param.hits.hits).each(function(index,item) {
                    let output = '';
                    output += '<tr>';
                    output += '<td>' + new
Date(JSON.parse(item._source.message).timestamp).toLocaleStri
ng() + '</td>';

                    output += '<td>' +
JSON.parse(item._source.message).userName + '</td>';
                    output += '<td>' +
JSON.parse(item._source.message).colorName + '</td>';

                    $('#output').append(output);
                });
            }
        });
    }
});

```

```
        },
        error: function() {
            alert('네트워크 오류 발생!');
        }
    });
}
selectList();
});
</script>
</head>
<body>
    <table id="output" border="1"></table>
</body>
</html>
```