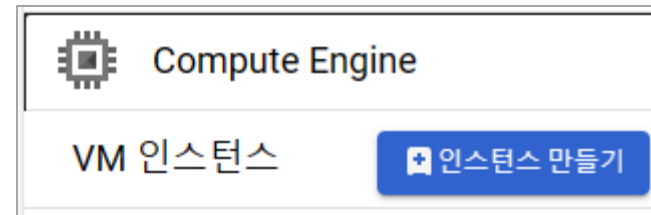
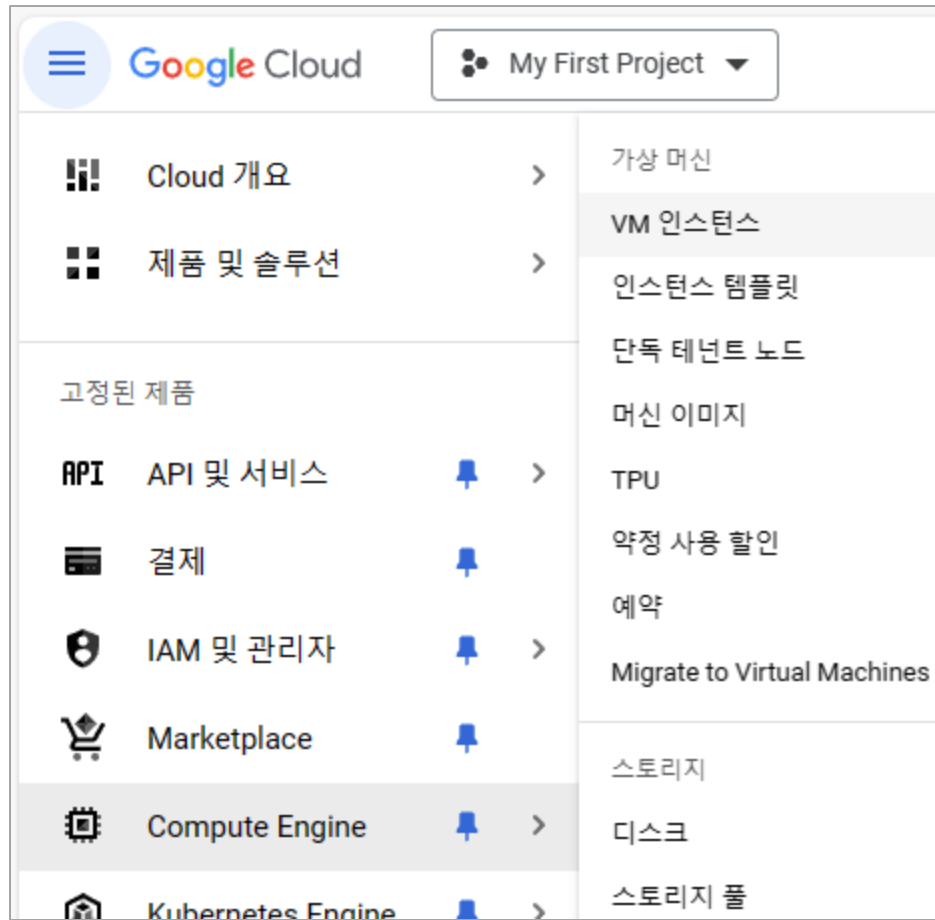


# K8S-Spark 인프라 구축

GCP 환경 구축하기: Base 이미지 만들기 - 1



기본 이미지용 VM 생성

- 이름: k8s-master
- 리전: us-west4(라스베이거스)
- 머신: E2 (2vcore, 4GB)
- 부팅디스크: Rockylinux
- 방화벽: HTTP 체크, HTTPS 체크
- 만들기 클릭

# K8S-Spark 인프라 구축

GCP 환경 구축하기: Base 이미지 만들기 - 2

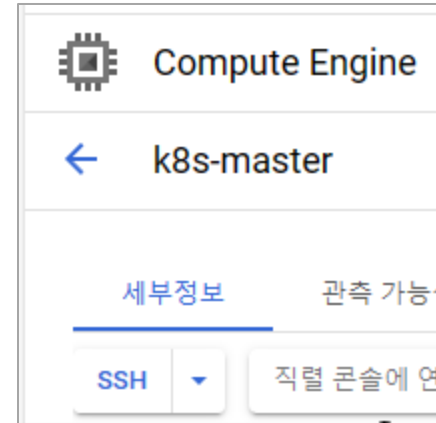
VSCode 접속을 위한 root 사용 권한 설정(root 로그인 허용 및 패스워드 인증 허용)

- SSH 연결(웹 터미널)
- root 사용 권한 설정

```
[jolajoayo_game@k8s-master ~]# sudo su
[root@k8s-master jolajoayo_game]# passwd
Changing password for user root.
New password: 자신의 암호 입력
Retype new password: 방금 입력한 암호 한 번 더 입력
passwd: all authentication tokens updated successfully.
[root@k8s-master jolajoayo_game]# vi /etc/ssh/sshd_config
```

vi 편집 화면에서 아래 문구를 찾아 yes로 바꾸고 저장  
PermitRootLogin yes  
PasswordAuthentication yes

```
[root@k8s-master jolajoayo_game]# systemctl restart sshd
```



아래 스크립트를 한 줄 씩 실행시키면  
좌측 vi 편집과 동일함

```
sed -i 's/^PermitRootLogin no/PermitRootLogin yes/' /etc/ssh/sshd_config
sed -i 's/^PasswordAuthentication no/PasswordAuthentication yes/' /etc/ssh/sshd_config
systemctl restart sshd
```

# K8S-Spark 인프라 구축

VSCode 터미널 접속 환경 설정 (아래 재설정 및 초기화 먼저 진행하세요!)

## 1. VSCode에서 VM 접속 설정 (재설정 및 초기화 →)

- ① VSCode Extension Remote-SSH 설치
- ② 검색창(Ctrl + Shift + p)에서 ssh open 검색
- ③ Open SSH Configuration File 선택
- ④ Config 파일이 열리면 아래 그림과 같이 입력

```
Host k8s-master
  HostName 192.168.56.30
  User root
Host k8s-node1
  HostName 192.168.56.31
  User root
Host k8s-node2
  HostName 192.168.56.32
  User root
Host helm-client
  HostName 192.168.56.29
  User root
```

## VSCode SSH 초기화

VM 재설치 등으로 SSH 설정  
정보 초기화 필요 시:

- 아래 폴더로 이동  
c:\user\<내계정>\.ssh\
- known\_hosts 파일 삭제
- 이 폴더나 파일이 없다면 이  
작업 skip!

## 2. VSCode에서 VM 접속

- ① 검색창(Ctrl + Shift + p)에서 ssh connect 검색
- ② Remote-SSH: Connect to Host 선택하고 " 계속 " 이나 패스워드를 입력하여 접속 진행

# K8S-Spark 인프라 구축

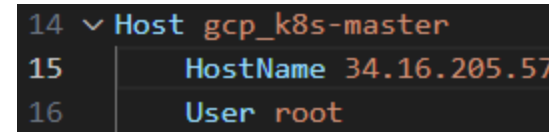
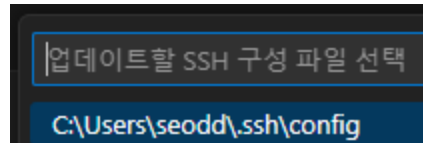
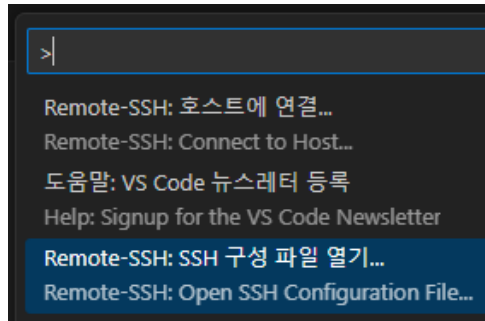
GCP 환경 구축하기: Base 이미지 만들기 - 3

VSCode로 k8s master 접속하기

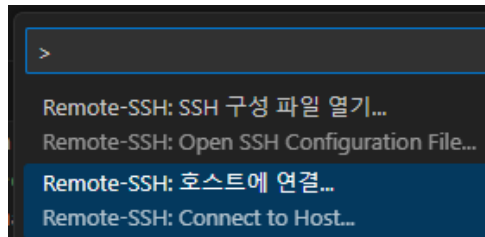
- VSCode에서 Ctrl + Shift + P
- Remote-SSH: SSH 구성 파일 열기... 선택  
C:\User\W[본인계정]\.ssh\config 선택

접속할 호스트 정보 추가 (Host 이름이 실제 이름과 같을 필요는 없음. IP만 같으면 됨)

config 파일 저장



- 호스트에 연결



# K8S-Spark 인프라 구축

GCP 환경 구축하기: Base 이미지 만들기 - 4

k8s master 서버에 공통 환경 적용

- [gcp\\_k8s\\_init.sh](#) 파일을 열고 안의 스크립트를 모두 복사한다.
- VSCode에서 파일 > 새 텍스트 파일 메뉴 클릭해서 새 창을 열고 스크립트를 붙여넣는다.
- 파일 > 저장 (or Ctrl+S)을 누르고 `/root/k8s/gcp_k8s_init.sh` 이름으로 저장한다.  
(새 폴더를 만들면서 저장할 것인지 물어보면 순순히 응할 것)
- 터미널 창에서 저장한 파일을 실행한다.

방금 저장한 파일은 실행 권한이 없으므로 실행 권한을 부여한다.

```
[root@k8s-master ~]# chmod +x ./k8s/gcp_k8s_init.sh
```

파일을 실행한다.

```
[root@k8s-master ~]# ./k8s/gcp_k8s_init.sh
```

# K8S-Spark 인프라 구축

GCP 환경 구축하기: Base 이미지 만들기 - 5

k8s master 서버 이용하여 Base 이미지 생성

- 아래 그림의 메뉴를 이용하여 새 머신 이미지를 만든다.

The screenshot shows the 'VM 인스턴스' (VM Instances) page in the Google Cloud Platform console. A table lists the 'k8s-master' instance in the 'us-west4-b' region. Below the table, a '관련 작업' (Related tasks) section contains several cards for backup, billing, monitoring, and patching. A context menu is open for the 'k8s-master' instance, showing options like '시작/재개' (Start/Restart), '중지' (Stop), '정지' (Suspend), '재설정' (Reset), '삭제' (Delete), '이 VM을 기반으로 그룹 만들기' (Create a group based on this VM), '미리보기' (Preview), '네트워크 세부정보 보기' (View network details), and '새 머신 이미지 만들기' (Create new machine image), which is highlighted.

상태	이름 ↑	영역	권장사항	다음에서 사용 중:	내부 IP	외부 IP	연결
<input checked="" type="checkbox"/>	<a href="#">k8s-master</a>	us-west4-b			10.182.0.7 ( <a href="#">nic0</a> )	34.16.205.57 ( <a href="#">nic0</a> )	SSH

관련 작업

- 백업 및 DR 살펴보기 **신규**  
VM을 백업하고 재해 복구를 설정합니다.
- 결제 보고서 보기  
Compute Engine 결제를 확인하고 관리합니다.
- VM 모니터링  
CPU 및 네트워크와 같은 측정항목 전반에서 이상점 VM을 확인합니다.
- VM 로그 살펴  
VM 인스턴스 로그를 다운로드할 수 있습니다.
- 방화벽 규칙 설정  
VM 인스턴스와 주고받는 트래픽을 제어합니다.
- 패치 관리  
패치 업데이트를 예약하고 VM 인스턴스의 패치 규정 준수를 확인합니다.
- VM 간 부하 분산  
트래픽 및 사용자 증가에 따라 애플리케이션에 부하 분산 설정

시작/재개  
중지  
정지  
재설정  
삭제  
이 VM을 기반으로 그룹 만들기  
**미리보기**  
네트워크 세부정보 보기  
새 머신 이미지 만들기

- 이름: img-k8s-base-v1
- 만들기 클릭

# K8S-Spark 인프라 구축

GCP 환경 구축하기: 머신 이미지로 k8s-node 서버 만들기

아래 그림의 메뉴를 이용하여 인스턴스 2대를 추가한다.

Compute Engine

가상 머신

VM 인스턴스

인스턴스 템플릿

단독 테넌트 노드

머신 이미지

TPU

약정 사용 할인

머신 이미지

+ 머신 이미지 만들기

새로고침

삭제

머신 이미지에는 VM의 속성, 메타데이터, 권한, 연결된 모든 디스크의 데이터가 포함됩니다. 머신 이미지를 사용하여 VM을 생성, 백업 또는 복원할 수 있습니다. [자세히 알아보기](#)

필터

속성 이름 또는 값 입력

<input type="checkbox"/>	상태	이름 ↑	소스 인스턴스	머신 유형	건축물	스토리지 위치	생성 시간	작업
<input type="checkbox"/>	✓	<a href="#">img-gcp-k8s</a>	k8s-master	e2-medium	—	us-west4	4월 1, 2024, 5:16:18 PM UTC+09:00	⋮
<input type="checkbox"/>	✓	<a href="#">img-k8s-base-v1</a>	k8s-master	e2-medium	—	us	4월 3, 2024, 1:17:28 AM UTC+09:00	⋮

+ 인스턴스 만들기

삭제

- 이름: k8s-node1, k8s-node2
- 방화벽: HTTP 체크, HTTPS 체크
- 만들기 클릭

# K8S-Spark 인프라 구축

GCP 환경 구축하기: 모든 서버에 VSCode 접속환경 세팅하기

VSCode의 SSH 구성파일에 추가한 2대의 서버 정보를 추가하고 저장한다.

```
14 ▾ Host gcp_k8s-master
15     |   HostName 34.16.205.57
16     |   User root
17 ▾ Host gcp_k8s-node1
18     |   HostName 34.125.227.203
19     |   User root
20 ▾ Host gcp_k8s-node2
21     |   HostName 34.16.200.161
22     |   User root
```

2대 모두 접속하여 터미널 창을 띄워놓자.



# K8S-Spark 인프라 구축

GCP 환경 구축하기: 모든 서버의 hosts 파일에 kube node 정보 추가

VSCode 파일탐색기에서 폴더열기 클릭

- 창에 /etc 입력 후 확인
- 탐색기에서 hosts 파일 찾아 클릭하여 파일 열기
- 파일 제일 뒤에 아래 정보 추가하고 저장(Ctrl+S) [IP는 자기 서버의 내부 IP로 변경하세요!]

```
10.182.0.7 k8s-master  
10.182.0.8 k8s-node1  
10.182.0.9 k8s-node2
```

# K8S-Spark 인프라 구축

GCP 환경 구축하기: k8s-master 서버 마무리

k8s master 서버에 단독 환경 적용

- 고정IP 적용(수정 > 네트워크 인터페이스 > 외부IPv4 주소 > 고정 외부IP 주소 예약: k8s-master-ip)  
→ IP 변경으로 기존 터미널 먹통되므로 ssh config 파일 수정하고 다시 연결하자
- gcp\_k8s\_master.sh 파일을 열고 안의 스크립트를 모두 복사한다.
- VSCode에서 파일 > 새 텍스트 파일 메뉴 클릭해서 새 창을 열고 스크립트를 붙여넣는다.
- 파일 > 저장 (or Ctrl+S)을 누르고 /root/k8s/gcp\_k8s\_master.sh 이름으로 저장한다.  
(새 폴더를 만들면서 저장할 것인지 물어보면 순순히 응할 것)
- 터미널 창에서 저장한 파일을 실행한다.

방금 저장한 파일은 실행 권한이 없으므로 실행 권한을 부여한다.

```
[root@k8s-master ~]# chmod +x ./k8s/gcp_k8s_master.sh
```

파일을 실행한다.

```
[root@k8s-master ~]# ./k8s/gcp_k8s_master.sh
```

여기까지 완료되면 kube 연결 슬라이드를  
참고하여 kube 설정을 마무리하자.

```
1  #!/bin/bash
2
3  # 클러스터 생성
4  echo '===== [8] kubeadm으로 클러스터 생성 ====='
5  echo '===== [8-1] 클러스터 초기화 (Pod Network 세팅) ====='
6  # kubeadm init --pod-network-cidr=20.96.0.0/12 --apiserver-advertise-address=10.102.0.2
7  kubeadm init --pod-network-cidr=20.96.0.0/12 --apiserver-advertise-address=GCP내부망주소 --apiserver-cert-extra-sans=GCP외부망주소
8  kubeadm token create --print-join-command > ~/join.sh
```



# K8S-Spark 인프라 구축

## Kube 환경 확인

### 1. Kube master – worker 노드간 연결

Master노드에서 아래 명령으로 토큰 생성 및 연결 명령어를 출력한다

```
[root@k8s-master ~]# cat ./join.sh
```

```
kubeadm join 192.168.56.30:6443 --token 7uruf6.n2w97dphqikk5m8f --discovery-token-ca-cert-hash  
sha256:b889a7b0767bde6e8d1dca4b326ef31669a7c6a4495ac18c7884d0dc41649e8f
```

Master노드에서 출력된 문자열을 copy 후 worker node 1,2에 접속하여 붙여넣고 실행시킨다.

```
[root@k8s-node2 ~]# kubeadm join 192.168.56.30:6443 --token 7uruf6.n2w97dphqikk5m8f --discovery-token-ca-cert-hash sha256:b889a7b0767bde6e8d1dca4b32  
6ef31669a7c6a4495ac18c7884d0dc41649e8f  
[preflight] Running pre-flight checks  
[preflight] Reading configuration from the cluster...
```

간혹 연결이 안 되는 경우가 있다.

이런 경우에는 토큰을 새로 생성하고 출력된 문자열을 복사하자.

```
[root@k8s-master ~]# kubeadm token create --print-join-command
```

### 2. kube 설치 확인

Master 노드에서 `kubectl get nodes` 명령어로 worker node가 잘 연결되어 있는지 확인

```
[root@k8s-master ~]# kubectl get nodes
```

```
[root@k8s-master ~]# kubectl get nodes  
NAME          STATUS    ROLES    AGE   VERSION  
k8s-master    Ready     control-plane  14d   v1.27.2  
k8s-node1     Ready     <none>      5m44s v1.27.2  
k8s-node2     Ready     <none>      5m27s v1.27.2
```

# K8S-Spark 인프라 구축

## GCP 환경 구축하기: 방화벽 설정

외부망에서 kube와 통신하기 위한 방화벽 설정

6443: for helm-client

30000: for dashboard

GCP 메뉴: VPC 네트워크 > 방화벽

- 방화벽 규칙 만들기 선택
- 이름: k8s-common
- 대상: 네트워크의 모든 인스턴스
- 소스 IPv4 범위: 0.0.0.0/0
- TCP 체크, 포트: 6443, 30000
- 만들기 선택

네트워크 보안

보안 헬프 문서

Cloud Armor

Cloud Armor 정책

Adaptive Protection

Cloud Armor 서비스 등급

Cloud IDS

IDS 대시보드

IDS 엔드포인트

IDS 위협

Cloud NGFW

대시보드

방화벽 정책

위협

방화벽 엔드포인트

공통 구성요소

보안 프로필

TLS 검사 정책

SSL 정책

클라이언트 인증

← 방화벽 규칙 만들기

방화벽 규칙은 인스턴스로 수신 또는 발신되는 트래픽을 제어합니다. 기본적으로 네트워크 외부에서 수신되는 트래픽은 차단됩니다. [자세히 알아보기](#)

이름 \*  
k8s-common

소문자, 숫자, 하이픈이 허용됩니다.

설명

로그

방화벽 로그를 사용 설정하면 대량의 로그가 생성되어 Logging 비용이 증가할 수 있습니다. [자세히 알아보기](#)

☐ 사용

☒ 사용 안함

네트워크 \*  
default

우선순위 \*  
1000

다른 방화벽 규칙의 우선순위 확인

우선순위 범위는 0~65535입니다.

트래픽 방향

☒ 수신

☐ 송신

일치 시 작업

☒ 허용

☐ 거부

대상  
네트워크의 모든 인스턴스

소스 필터  
IPv4 범위

소스 IPv4 범위 \*  
0.0.0.0/0

보조 소스 필터  
없음

대상 필터  
없음

프로토콜 및 포트

☐ 모두 허용

☒ 지정된 프로토콜 및 포트

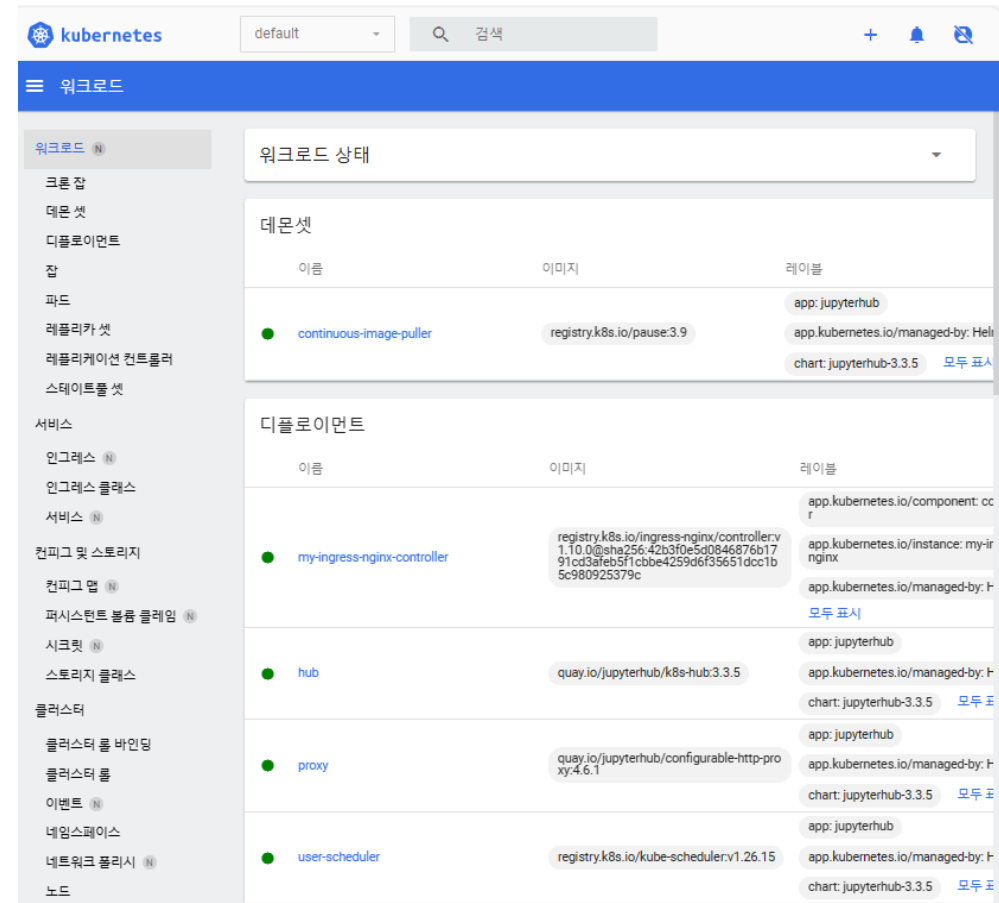
☒ TCP

포트  
6443,30000

# K8S-Spark 인프라 구축

Kube 환경 확인 - 대시보드 접속하기

- ① <https://34.125.136.103:30000> 접속 > 고급 > (안전하지 않음) 클릭
- ② 만약 로그인/토큰 팝업이 뜬다면 "생략" 클릭



# K8S-Spark 인프라 구축

Helm이란?



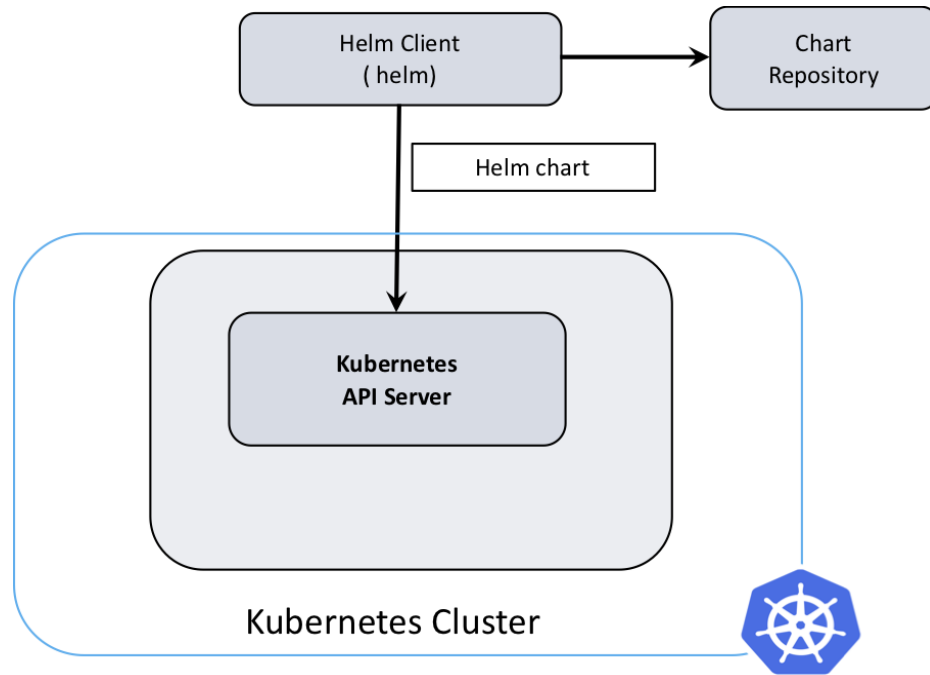
Helm은 Kubernetes 애플리케이션을 설치, 관리 및 업그레이드하는 데 사용되는 패키지 관리자입니다. Helm은 Kubernetes의 패키지 매니저로서, 복잡한 애플리케이션을 쉽게 배포할 수 있도록 도와줍니다. 다음은 Helm의 기능 및 장점입니다.

1. **패키지 관리:** Helm은 "차트"라고 하는 패키지 형식을 사용하여 Kubernetes 애플리케이션을 패키지화합니다. 이를 통해 애플리케이션을 쉽게 설치, 관리 및 업그레이드할 수 있습니다.
2. **재사용성:** Helm 차트는 재사용 가능한 템플릿으로 제공됩니다. 따라서 Helm을 사용하여 애플리케이션을 패키지화하면 해당 애플리케이션을 다른 환경 또는 프로젝트로 쉽게 이동할 수 있습니다.
3. **버전 관리:** Helm은 애플리케이션의 여러 버전을 관리할 수 있습니다. 이를 통해 특정 버전의 애플리케이션을 빠르게 롤백하거나 업그레이드할 수 있습니다.
4. **간편한 배포:** Helm을 사용하면 명령행 인터페이스를 통해 간단하게 애플리케이션을 배포할 수 있습니다. 사용자는 복잡한 Kubernetes 매니페스트를 직접 작성할 필요가 없습니다.
5. **커뮤니티 및 생태계:** Helm은 널리 사용되며, 커뮤니티에서 수많은 공식 및 커뮤니티 차트를 제공합니다. 따라서 Helm을 사용하여 다양한 유용한 애플리케이션을 쉽게 배포할 수 있습니다.

Helm은 Kubernetes 애플리케이션을 관리하는 데 매우 유용한 도구이며, Kubernetes의 애플리케이션 관리 과정을 단순화하고 자동화하는 데 큰 도움이 됩니다.

# K8S-Spark 인프라 구축

## Helm 아키텍처



### Helm 클라이언트를 통해서

- Helm chart Repository
- Kubernetes에 패키지 배포

### Helm chart

Kube 어플리케이션을 패키징하기 위한 템플릿 제공  
`values.yaml` 에서 설정 커스터마이징 가능

### Helm Client의 위치

일반적으로 kubernetes 외부에 위치

### Helm과 kube와의 통신

Kube API 통해서 통신하며 뒤에 나올 helm 환경  
설정에서 자세히 기술 예정

# K8S-Spark 인프라 구축

GCP Helm 환경 구축하기: Base 머신 이미지로 helm-client 만들기

(Quiz 1)

머신 이미지 메뉴에서 전에 만든 Base 이미지를 이용하여 인스턴스 1대를 추가한다.

- 이름: helm-client
- 방화벽: HTTP 체크, HTTPS 체크
- 만들기 클릭

(Quiz 2)

VSCoDe 접속 환경을 만든다

- SSH Config 파일에 정보 추가
- VSCoDe로 터미널 접속



# K8S-Spark 인프라 구축

GCP Helm 환경 구축하기: Base 머신 이미지로 helm-client 만들기

Helm 설치하기

- [gcp\\_helm.sh](#) 파일을 열고 안의 스크립트를 모두 복사한다.
- VSCode에서 파일 > 새 텍스트 파일 메뉴 클릭해서 새 창을 열고 스크립트를 붙여넣는다.
- 파일 > 저장 (or Ctrl+S)을 누르고 /root/helm/gcp\_helm.sh 이름으로 저장한다.  
(새 폴더를 만들면서 저장할 것인지 물어보면 순순히 응할 것)
- 터미널 창에서 저장한 파일을 실행한다.

방금 저장한 파일은 실행 권한이 없으므로 실행 권한을 부여한다.

```
[root@k8s-master ~]# chmod +x ./helm/gcp_helm.sh
```

파일을 실행한다.

```
[root@k8s-master ~]# ./helm/gcp_helm.sh
```

# K8S-Spark 인프라 구축

## helm 환경 설정

### 1. Helm 설치 확인

Helm 버전 정보 확인해보기

```
[root@helm-client ~]# helm version
```

### 2. Helm repo 관리 명령어

```
[root@helm-client ~]# helm repo list
```

(필요한 경우 실행) `[root@helm-client ~]# helm repo remove 리/포이름`

### 3. Helm repo 추가

K8s-spark repo를 설치하는 여러 방법

```
[root@helm-client ~]# helm repo add bitnami https://charts.bitnami.com/bitnami
```

```
[root@helm-client ~]# helm repo update
```

(필요한 경우 실행) `[root@helm-client ~]# helm pull bitnami/spark --version 5.1.2`

### 4. Helm repo 및 chart 조회

```
[root@helm-client ~]# helm search repo bitnami
```

```
[root@helm-client ~]# helm search repo bitnami/spark --versions
```

솔루션 설치의 사전 설정 정보 조회

# 기본 `values.yaml` 보기

```
[root@helm-client ~]# helm show values bitnami/spark
```

그런데 아래 링크에서 보는게 더 편하긴 하다.

<https://github.com/bitnami/charts/tree/main/bitnami>

### 5. (참고) Helm과 kube 클러스터간 연결 (gcp\_helm.sh에서 이미 실행함)

Kube master의 `~/.kube/config` 파일을 helm-client의 동일한 경로에 카피하면 됨

```
[root@helm-client ~]# mkdir .kube
```

```
[root@helm-client ~]# scp root@192.168.56.30:~/.kube/config .kube/config
```

```
[root@helm-client ~]# ls .kube
```

kube master 재 설치 등으로 혹시 호스트 키가 변경되었을 경우 아래 명령으로 호스트키 초기화 한다.

```
[root@helm-client ~]# ssh-keygen -R [호스트명 또는 IP 주소]
```

```
[root@k8s-master ~]# ls -all
total 48
dr-xr-x---.  4 root root 4096 Feb 29 14:28 .
dr-xr-xr-x. 17 root root  240 May 18  2023 ..
-rw-----.  1 root root 6411 May 18  2023 anaconda-ks.cfg
-rw-----.  1 root root  699 Mar  7 19:20 .bash_history
-rw-r--r--.  1 root root   18 Mar 15  2021 .bash_logout
-rw-r--r--.  1 root root  176 Mar 15  2021 .bash_profile
-rw-r--r--.  1 root root   267 Feb 27 09:37 .bashrc
-rw-r--r--.  1 root root   100 Mar 15  2021 .cshrc
-rw-r--r--.  1 root root   168 Feb 27 09:37 .join.sh
drwxr-xr-x.  3 root root   33 Feb 27 09:37 .kube
-rw-----.  1 root root 6171 May 18  2023 original-ks.cfg
drwx-----.  2 root root   25 Feb 29 14:29 .ssh
-rw-r--r--.  1 root root  129 Mar 15  2021 .tcshrc
```

```
[root@k8s-master ~]# ls .kube/ -all
total 12
drwxr-xr-x. 3 root root   33 Feb 27 09:37 .
dr-xr-x---. 4 root root 4096 Feb 29 14:28 ..
drwxr-xr-x. 4 root root   35 Feb 27 09:37 cache
-rw-----. 1 root root 5637 Feb 27 09:37 config
```

### 6. GCP의 경우 config 파일을 열고 내부IP를 외부IP로 수정한다.

# K8S-Spark 인프라 구축

## spark 환경 구축

### 1. Helm 이용한 Spark 설치

Helm-client 접속

1) Bitnami spark 설치 (ChatGPT 버전, repo를 add했을 때 사용 가능)

```
[root@helm-client ~]# helm install my-first-spark bitnami/spark
```

2) Bitnami spark 설치 (공식 문서 버전, repo add 없이 사용 가능)

```
[root@helm-client ~]# helm install my-first-spark oci://registry-1.docker.io/bitnamicharts/spark
```



당신

아래 명령에서 oci로 시작하는 건 뭐야?

```
helm install my-release oci://REGISTRY_NAME/REPOSITORY_NAME/spark
```



ChatGPT

`oci://REGISTRY\_NAME/REPOSITORY\_NAME/spark`에서 `oci`로 시작하는 부분은 OCI(Open Container Initiative) 이미지 형식을 나타내는 것입니다. OCI는 컨테이너 이미지의 표준을 정의한 것으로, 컨테이너 이미지와 관련된 규격을 통일하기 위해 만들어졌습니다. 이 표준을 사용함으로써 다양한 컨테이너 오케스트레이션 시스템 간에 이미지 호환성을 보장할 수 있습니다.

여기서 `oci://REGISTRY\_NAME/REPOSITORY\_NAME/spark`는 Helm 차트를 설치할 때 사용하는 컨테이너 이미지의 위치를 지정하는 것입니다. 이 구문은 OCI 호환 레지스트리에서 Helm 차트를 찾으라는 의미입니다. Helm 3.7.0 버전부터 OCI 기반 레지스트리를 사용하여 차트를 저장하고 공유할 수 있는 기능이 도입되었습니다.

# K8S-Spark 인프라 구축

```
[root@helm-client ~]# helm install my-first-spark oci://registry-1.docker.io/bitnamicharts/spark
Pulled: registry-1.docker.io/bitnamicharts/spark:9.0.0
Digest: sha256:00828dd9f646ecd5dd4747e54af25a4fbc33be3d989618c9850ecde1cd0cdb6a
NAME: my-first-spark
LAST DEPLOYED: Thu Mar 21 09:14:36 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: spark
CHART VERSION: 9.0.0
APP VERSION: 3.5.1
```

**\*\* Please be patient while the chart is being deployed \*\***

1. Get the Spark master WebUI URL by running these commands:

```
1 kubectl port-forward --namespace default svc/my-first-spark-master-svc 80:80
echo "Visit http://127.0.0.1:80 to use your application"
```

2. Submit an application to the cluster:

To submit an application to the cluster the spark-submit script must be used. That script can be obtained at <https://github.com/apache/spark/tree/master/bin>. Also you can use kubectl run.

```
2 export EXAMPLE_JAR=$(kubectl exec -ti --namespace default my-first-spark-worker-0 -- find examples/jars/ -name 'spark-example*W.jar' | tr -d 'W')

kubectl exec -ti --namespace default my-first-spark-worker-0 -- spark-submit --master spark://my-first-spark-master-svc:7077 W
--class org.apache.spark.examples.SparkPi W
$EXAMPLE_JAR 5
```

**\*\* IMPORTANT: When submit an application from outside the cluster service type should be set to the NodePort or LoadBalancer. \*\***

**\*\* IMPORTANT: When submit an application the --master parameter should be set to the service IP, if not, the application will not resolve the master. \*\***

WARNING: There are "resources" sections in the chart not set. Using "resourcesPreset" is not recommended for production. For production installations, please set the following values according to your workload needs:

- master.resources
- worker.resources

+info <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>

## spark 설치 결과

- ① WEB-UI 확인 방법
- ② 샘플 코드 실행 방법

**먼저 대시보드도 한 번 봐 주시다**

**<http://k8s-masterIP:30000>**

# K8S-Spark 인프라 구축

## spark 설치 결과

### ① WEB-UI 확인 방법

Get the Spark master WebUI URL by running these commands:

```
kubectl port-forward --namespace default svc/my-first-spark-master-svc 80:80  
echo "Visit http://127.0.0.1:80 to use your application"
```

port-forward 명령을 치면 포트 매핑 작업의 활성(Listening) 상태가 되어 해당 터미널을 더 이상 쓰지 못 한다.

```
[root@k8s-master ~]# kubectl port-forward --namespace default svc/my-first-spark-master-svc 80:80
```

```
[root@k8s-master ~]# kubectl port-forward --namespace default svc/my-first-spark-master-svc 80:80  
Forwarding from 127.0.0.1:80 -> 8080  
Forwarding from [::1]:80 -> 8080
```

WebUI URL 확인을 위해 새 터미널을 열어 확인한다. (80포트는 기본 http포트이므로 생략 가능)

```
[root@k8s-master ~]# curl 127.0.0.1
```

```
[root@k8s-master ~]# curl 127.0.0.1  
<!DOCTYPE html><html>  
  <head>  
    <meta http-equiv="Content-type"  
    l="stylesheet" href="/static/bootstrap.m  
    k_rel="stylesheet" href="/static/webui.d  
    c/sorttable.js"></script><script src="/s  
    ic/bootstrap.bundle.min.js"></script><sc  
    imeline-view.js"></script><script src="/
```

- 확인이 끝나면 활성 상태의 터미널은 중지시켜도 된다(Ctrl + C)
- 샘플코드 실행 확인 후 로컬PC의 브라우저에서 WebUI를 확인해보자.

# K8S-Spark 인프라 구축

## spark 설치 결과

### ② 샘플코드 확인 방법

Submit an application to the cluster.

To submit an application to the cluster the spark-submit script must be used. That script can be obtained at <https://github.com/apache/spark/tree/master/bin>. Also you can use kubectl run.

```
export EXAMPLE_JAR=$(kubectl exec -ti --namespace default my-first-spark-worker-0 -- find examples/jars/ -name 'spark-example*W.jar' | tr -d '\n')
```

```
kubectl exec -ti --namespace default my-first-spark-worker-0 -- spark-submit --master spark://my-first-spark-master-svc:7077 \W  
--class org.apache.spark.examples.SparkPi \W  
$EXAMPLE_JAR 5
```

일단 시키는대로 하자. (*EXAMPLE\_JAR* 변수에 특정 문자열을 할당하는 작업)

```
[root@k8s-master ~]# export EXAMPLE_JAR=$(kubectl exec -ti --namespace default my-first-spark-worker-0 -- find examples/jars/ -  
name 'spark-example*\.jar' | tr -d '\n')
```

과연 *EXAMPLE\_JAR* 변수에 들어있는 내용은? 두구두구두구...

```
[root@k8s-master ~]# echo $EXAMPLE_JAR
```

샘플 코드 실행

```
[root@k8s-master ~]# kubectl exec -ti --namespace default my-first-spark-worker-0 -- spark-submit --master spark://my-first-  
spark-master-svc:7077 --class org.apache.spark.examples.SparkPi $EXAMPLE_JAR 5
```

샘플코드는 원주율을 구하는 코드였으며 다음에 소스 및 작동 방식을 좀 더 자세히 볼 예정이다.

# K8S-Spark 인프라 구축

spark 샘플코드 소스 및 작동 방식 확인

확률적인 계산 방법으로 원주율을 구하는 파이썬 코드 (<https://github.com/apache/spark/blob/master/examples/src/main/python/pi.py>)

kubectl을 이용한 python 샘플 코드 실행 명령어

kubectl exec -ti --namespace default <소스파일이 있는 서버> -- spark-submit --master spark://my-first-spark-master-svc:7077 <소스파일의 경로>/<소스파일명>

[root@k8s-master ~]# kubectl exec -ti --namespace default my-first-spark-worker-0 -- spark-submit --master spark://my-first-spark-master-svc:7077 examples/src/main/python/pi.py 5

```
import sys
```

```
from random import random
```

```
from operator import add
```

```
from pyspark.sql import SparkSession
```

```
if __name__ == "__main__":
```

```
    spark = SparkSession.builder.appName("PythonPi").getOrCreate()
```

```
    partitions = int(sys.argv[1]) if len(sys.argv) > 1 else 2
```

```
    n = 100000 * partitions
```

```
    def f(_: int) -> float:
```

```
        x = random() * 2 - 1
```


```
        y = random() * 2 - 1
```

```
        return 1 if x**2 + y**2 <= 1 else 0
```

```
    count = spark.sparkContext.parallelize(range(1, n + 1), partitions).map(f).reduce(add)
```

```
    print("Pi is roughly %f" % (4.0 * count / n))
```

```
    spark.stop()
```

 **Spark Master** at spark://my-first-spark-master-0.my-first-spark-headless.default.svc.cluster.local:7077  
3.5.1

URL: spark://my-first-spark-master-0.my-first-spark-headless.default.svc.cluster.local:7077

Alive Workers: 2

Cores in use: 2 Total, 0 Used

Memory in use: 2.0 GiB Total, 0.0 B Used

Resources in use:

Applications: 0 Running, 2 Completed

Drivers: 0 Running, 0 Completed

Status: ALIVE

**Workers (2)**

Worker Id	Address	State	Cores	Memory	Resources
worker-20240324061059-20.111.156.117-44445	20.111.156.117:44445	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	
worker-20240324061116-20.109.131.42-43107	20.109.131.42:43107	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	

**Running Applications (0)**

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

**Completed Applications (2)**

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20240324111327-0001	PythonPi	2	1024.0 MiB		2024/03/24 11:13:27	spark	FINISHED	14 s
app-20240324105430-0000	Spark Pi	2	1024.0 MiB		2024/03/24 10:54:30	spark	FINISHED	11 s

# K8S-Spark 인프라 구축

Helm에 관하여

왜 Helm인가?

ChatGPT4에 K8S-Spark 질문하니 Helm을 사용하라고 알려줌



당신

K8S-Spark 구축하는 방법 중 가장 쉬운 방법 알려줘



ChatGPT

Kubernetes(K8s) 상에서 Apache Spark를 구축하는 가장 쉬운 방법은 Helm을 사용하는 것입니다. Helm은 Kubernetes 애플리케이션의 패키징과 배포를 간소화하는 도구입니다. Helm 차트를 사용하면 복잡한 Spark 클러스터 구성을 쉽게 관리하고 배포할 수 있습니다. 아래 단계별로 가장 간단한 방법을 소개합니다.

(흥미롭게도 같은 질문에 대해 ChatGPT 3.5는 helm에 대한 언급이 1도 없었음)



# K8S-Spark 인프라 구축

## ChatGPT 3.5와 4의 차이



당신

kube 기반에 helm으로 spark를 설치하는 방법 알려줘



ChatGPT

Apache Spark를 Kubernetes 클러스터에 Helm을 사용하여 설치하는 방법은 다음과 같습니다.

### 1. Helm 설치:

먼저 Helm을 설치합니다. Helm이 설치되어 있지 않은 경우 공식 Helm 설치 문서를 참조하여 설치합니다.

### 2. Helm 리포지터리 추가:

Apache Spark를 설치할 Helm 차트를 사용할 수 있는 Helm 리포지터리를 추가합니다. 일반적으로는 `stable` 또는 `incubator` 리포지터리를 사용합니다.

```
bash
```

Copy code

```
helm repo add stable https://charts.helm.sh/stable
```

### 3. Spark 설치용 설정 파일 작성:

Spark 설치에 필요한 설정 파일을 작성합니다. 예를 들어, Spark Worker 수, 메모리 할당량 등을 지정할 수 있습니다. 필요에 따라 `values.yaml` 파일을 작성하여 이러한 설정을 지정할 수 있습니다.

### 4. Helm을 사용하여 Spark 설치:

Helm을 사용하여 Spark를 설치합니다. 예를 들어, 다음 명령어를 사용하여 설치할 수 있습니다.

```
bash
```

Copy code

```
helm install my-spark-release stable/spark -f values.yaml
```

여기서 `my-spark-release`는 설치할 Spark 차트의 이름입니다. `values.yaml` 파일에 지정한 설정을 사용하여 설치가 진행됩니다.

### 5. 설치 확인:



ChatGPT

Kubernetes(K8s) 상에서 Apache Spark를 구축하는 방법은 여러 가지가 있지만, 가장 쉬운 방법 중 하나는 Helm 차트를 사용하는 것입니다. Helm은 Kubernetes 애플리케이션의 패키징과 배포를 쉽게 만들어주는 툴입니다. Apache Spark에 대한 Helm 차트를 사용하면, 복잡한 구성 작업 없이도 Spark 클러스터를 쉽게 배포할 수 있습니다.

## Helm을 사용하여 Spark를 Kubernetes에 배포하는 단계:

- Helm 설치:** Helm은 Kubernetes 클러스터를 관리하기 위한 툴이므로 먼저 Helm을 로컬 시스템에 설치해야 합니다. Helm 공식 웹사이트(<https://helm.sh/docs/intro/install/>)에서 설치 지침을 찾을 수 있습니다.
- Helm 리포지터리 추가:** 필요한 Helm 차트를 찾기 위해 Bitnami 또는 다른 공식 Spark Helm 차트를 호스팅하는 리포지토리를 추가합니다.

```
bash
```

Copy code

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo update
```

- Helm 차트를 사용하여 Spark 배포:** 이제 Kubernetes 클러스터에 Spark를 배포할 준비가 되었습니다. 다음 명령을 사용하여 Spark 클러스터를 배포할 수 있습니다.

```
bash
```

Copy code

```
helm install my-spark bitnami/spark
```

- 배포 확인:** 배포가 성공적으로 완료되었는지 확인하기 위해 다음 명령을 사용하여 Spark 파드와 서비스를 확인할 수 있습니다.

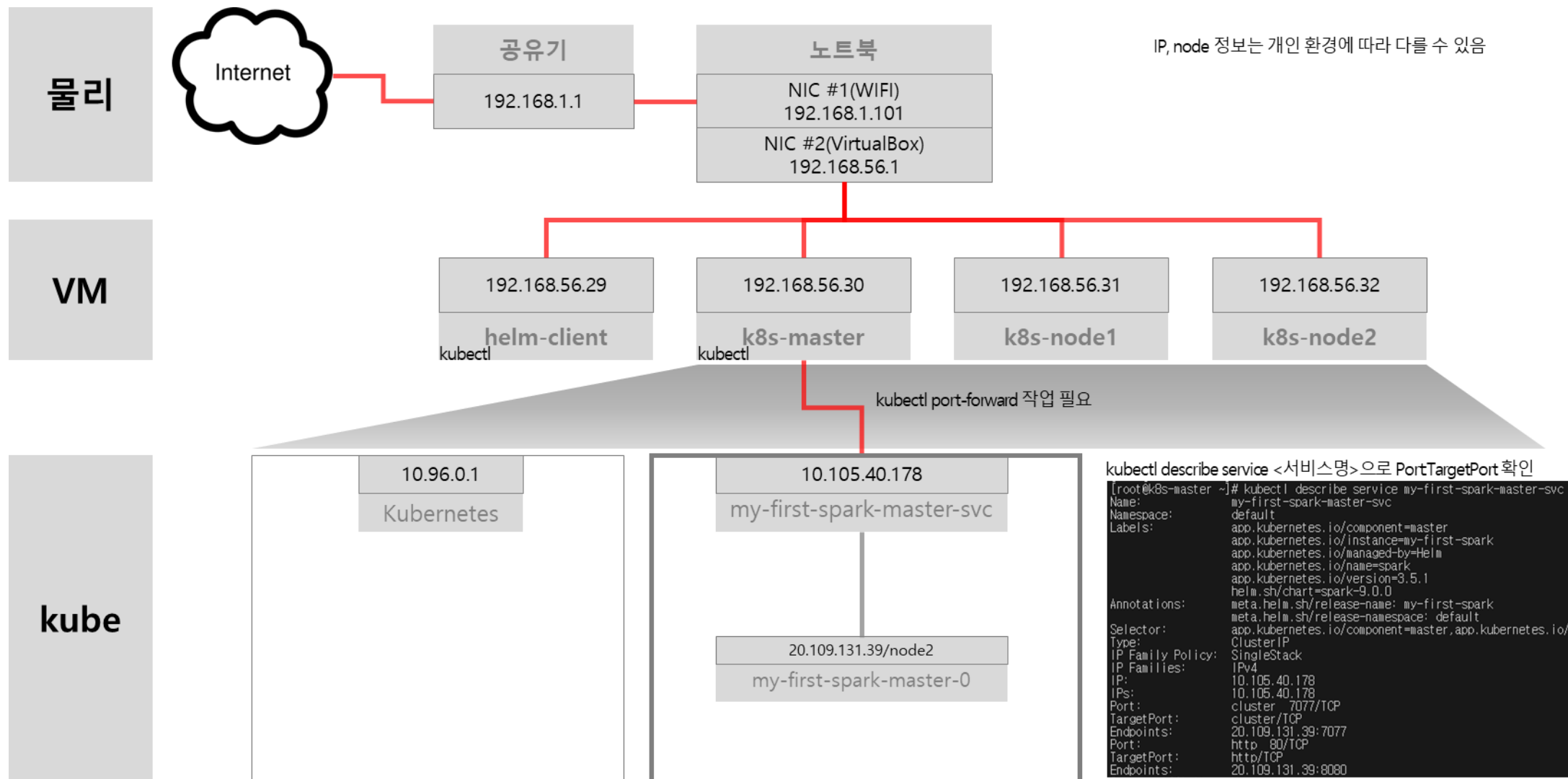
```
bash
```

Copy code

```
kubectl get pods
kubectl get svc
```

# K8S-Spark 인프라 구축

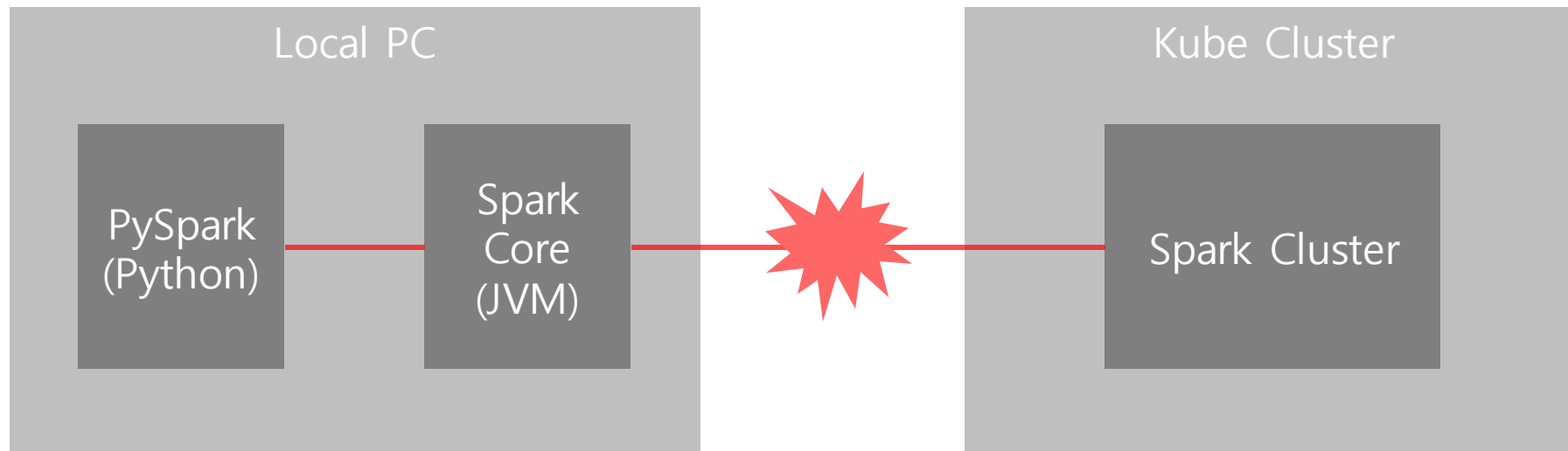
spark WebUI를 로컬PC 브라우저에서 확인하기 전에 네트워크 구성을 알아보자.



# K8S-Spark 인프라 구축

로컬PC VSCode의 주피터노트북 환경에서 바로 스파크 클러스터 연결하기

아래 그림과 같이 현재 로컬PC에서 Spark cluster로 통신할 수 있는 경로가 없다.  
이 통신 경로를 뚫어주기 위해 kube의 Service 설정을 수정해야 한다.

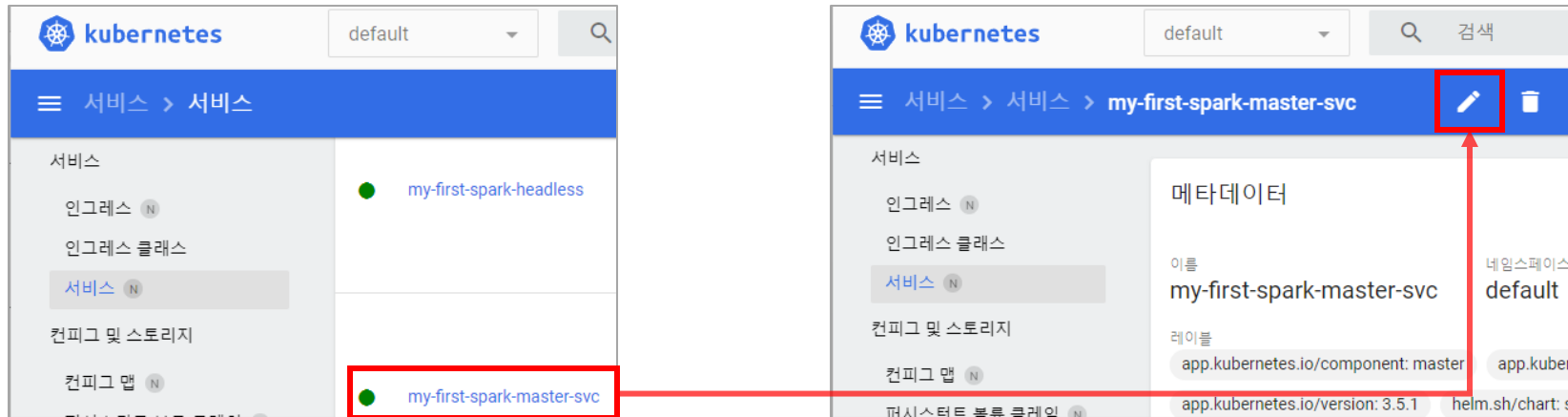


# K8S-Spark 인프라 구축

로컬PC VSCode의 주피터노트북 환경에서 바로 스파크 클러스터 연결하기

실습X

- ① kube 대시보드에서 서비스 > 서비스 > my-first-spark-master-svc를 선택 후 편집 버튼 클릭



# K8S-Spark 인프라 구축

## 로컬PC VSCode의 주피터노트북 환경에서 바로 스파크 클러스터 연결하기

- ② 리소스 yaml을 아래와 같이 수정 후 "업데이트" 클릭

리소스 편집

YAML

JSON

```

71 spec:
72   ports:
73     - name: cluster
74       protocol: TCP
75       port: 7077
76       targetPort: cluster
77       nodePort: 30077
78     - name: http
79       protocol: TCP
80       port: 80
81       targetPort: http
82       nodePort: 30078
83   selector:
84     app.kubernetes.io/component: master
85     app.kubernetes.io/instance: my-first-spark
86     app.kubernetes.io/name: spark
87   clusterIP: 10.97.148.129
88   clusterIPs:
89     - 10.97.148.129
90   type: NodePort
91   sessionAffinity: None
92   externalTrafficPolicy: Cluster

```

추가

추가

변경: ClusterIP → NodePort

이 액션은 다음 커맨드와 동일합니다. `kubectl apply -f`

업데이트 취소

**Spark Master at spark://my-first**

**URL:** spark://my-first-spark-master-0:my-first-spark-headless.default.svc.cluster.local

**Alive Workers:** 2

**Cores in use:** 2 Total, 0 Used

**Memory in use:** 2.0 GiB Total, 0.0 B Used

**Resources in use:**

**Applications:** 0 Running, 3 Completed

**Drivers:** 0 Running, 0 Completed

**Status:** ALIVE

▼ **Workers (2)**

Worker Id
-----------

<http://192.168.56.30:30078>

## 에서 WEB UI 확인

## 30077 포트는 Spark 실행 관련 포트

GCP/AWS에서는  
방화벽 규칙 추가 필요

# K8S-Spark 인프라 구축

GCP 환경 구축하기: 방화벽 설정

외부망에서 spark와 통신하기 위한 방화벽 설정

GCP 메뉴: VPC 네트워크 > 방화벽

- 방화벽 규칙 만들기 선택
- 이름: kube-spark
- 대상: 네트워크의 모든 인스턴스
- 소스 IPv4 범위: 0.0.0.0/0
- TCP 체크, 포트: 4040,30077,30078
- 만들기 선택

네트워크 보안

보안 웹 프록시

Cloud Armor

Cloud Armor 정책

Adaptive Protection

Cloud Armor 서비스 등급

Cloud IDS

IDS 대시보드

IDS 엔드포인트

IDS 위협

Cloud NGFW

대시보드

방화벽 정책

위협

방화벽 엔드포인트

공통 구성요소

보안 프로필

TLS 검사 정책

SSL 정책

클라이언트 인증

← 방화벽 규칙 세부정보

이미 수정 모드에 있음

kube-spark

설명

로그

방화벽 로그를 사용 설정하면 대량의 로그가 생성되어 Logging 비용이 증가할 수 있습니다.[자세히 알아보기](#)

☐ 사용

☐ 사용 안함

네트워크

default

우선순위 \*

1000

다른 방화벽 규칙의 우선순위 확인

우선순위 범위는 0~65535입니다.

방향

인그레스

일치 시 작업

허용

대상

네트워크의 모든 인스턴스

소스 필터

IPv4 범위

소스 IPv4 범위 \*

0.0.0.0/0

보조 소스 필터

없음

대상 필터

없음

프로토콜 및 포트

☐ 모두 허용

☒ 지정된 프로토콜 및 포트

☒ TCP

포트

4040,30077,30078

예: 20, 50~60

# K8S-Spark 인프라 구축

로컬PC VSCode의 주피터노트북 환경에서 바로 스파크 클러스터 연결하기

## ① Spark cluster 삭제하기

```
[root@k8s-master ~]# helm uninstall my-first-spark
```

## ② 더 다양한 옵션으로 Spark Cluster 설치하기

현재 최신 버전인 spark 3.5로 설치하려면 9.0.0 입력하면 되나

## ③ 다른 에코와의 궁합 정보가 가장 많이 알려져있는 Spark 3.4.1로 설치하기 위해 7.2.2로 설치한다

```
[root@k8s-master ~]# helm install my-first-spark bitnami/spark --version 7.2.2 -n default \
--set service.type=NodePort \
--set service.nodePorts.cluster=30077 \
--set service.nodePorts.http=30078 \
--set worker.replicaCount=3
```

더 다양한 옵션들은 아래 사이트에서 확인 가능하다.

<https://artifacthub.io/packages/helm/bitnami/spark>

# K8S-Spark 인프라 구축

실습X

로컬PC VSCode의 주피터노트북 환경에서 바로 스파크 클러스터 연결하기

- ① 탐색기를 열고 편한 곳에 jupyter 폴더를 만든다.
- ② 만들어진 폴더를 오른쪽클릭하고 VSCode에서 열기를 클릭한다.
- ③ 새 파일을 만들고 spark01.ipynb으로 저장한다.
  - 한 셀씩 실행시켜보자.
  - 마지막 셀 실행 중에 spark web ui를 새로고침하며 진행상황을 확인해보자.

```
%pip install pyspark
```

```
from pyspark.sql import SparkSession
```

```
spark = SparkSession.builder \
    .master("spark://34.125.136.103:30077") \
    .appName("MyApp2") \
    .config("spark.kubernetes.namespace", "default") \
    .getOrCreate()
```

```
spark
```

```
# Spark DataFrame 예제
df = spark.createDataFrame([{"Name": "John Doe", "Age": 30}, {"Name": "Jane Doe", "Age": 25}])
df.printSchema()
```

```
df.show()
```

```
spark.stop()
```



# K8S-Spark 인프라 구축

로컬PC VSCode의 주피터노트북 환경에서 바로 스파크 클러스터 연결하기  
여기까지 진행하면서 발생한 상황을 정리해보자.

① 외부망에서 Spark Cluster의 WebUI는 볼 수 있다.

그러나 worker node의 상황은 볼 수 없다.

② 외부망의 주피터 노트북에서 스파크 클러스터 세션 연결은 가능하다.

그러나 job은 실행되지 않는다.

- Executor가 Running 상태가 되었다가 바로 Exited되고 다음 Executor가 무한 반복하는 현상

③ 왜 이런 현상이 발생하는지 알아야 해결 가능할텐데 아직은 정확하게 이해하는 수준은 아니다.

아는 수준 내에서 설명을 해 보자면,

- 주피터 노트북에서 스파크에 접속하면 기본적으로 Client Mode로 실행된다.

이는 스파크 Executor를 지휘하는 Driver가 노트북이 실행되는 곳(로컬PC)에 뜬다는 것이다.

(여기까지는 주피터 노트북 셀에서 spark 명령으로 확인 가능함)

- Driver가 지휘해야 하는 Executor는 스파크 Worker node에서 작동한다.

그러나 Worker node는 외부에서 접근 불가능하다.

- 이를 해결하려면??

- 주피터 노트북을 꼭 로컬PC에서 실행시켜야 한다면 모든 Worker node를 외부망에 노출시킨다

(이건 여기저기 알아본 바 추천하지 않았음)

- 주피터 노트북과 스파크 클러스터가 같은 망에 위치하면 될 듯?

→ 이 환경을 만들어보자!

# K8S-Spark 인프라 구축

스파크 클러스터와 같은 망에서 주피터노트북 환경 세팅하고 스파크 클러스터 연결하기

(Quiz 3)

- ① GCP에 주피터용 VM을 만든다.
  - 베이스 이미지 이용해서 VM 생성
  - 이름: jupyter-notebook

(Quiz 4)

- ② VSCode에서 jupyter-notebook 터미널 연결 설정 및 접속
- ③ 주피터 노트북 환경을 세팅한다.

# K8S-Spark 인프라 구축

## GCP 주피터 노트북 환경 세팅하기

우리가 사용하는 Base 이미지에서 k8s-jupyter notebook 이런 체계로 설치하고 싶었는데 많이 복잡해져서 일단 conda 환경의 힘을 빌리기로 하였다.

### ① Conda 설치

Anaconda는 너무 거대해서 부담스러우니 Miniconda 설치하자.

```
[root@jupyter-notebook ~]# dnf install wget -y
[root@jupyter-notebook ~]# wget https://repo.anaconda.com/miniconda/Miniconda3-Latest-Linux-x86_64.sh
[root@jupyter-notebook ~]# chmod +x Miniconda3-Latest-Linux-x86_64.sh
[root@jupyter-notebook ~]# ./Miniconda3-Latest-Linux-x86_64.sh
```

- 설치 과정에서 약관 동의 필요(약관 전문 보게 하는데 스페이스 연타로 빨리 넘기고 yes)
- 이렇게 물어보면 그냥 엔터

```
Miniconda3 will now be installed into this location:
/root/miniconda3

- Press ENTER to confirm the location
- Press CTRL-C to abort the installation
- Or specify a different location below

[/root/miniconda3] >>>
```

- 마지막 질문은 yes 하세요. 오른쪽 그림처럼 no 하면 귀찮아집니다.

```
You can undo this by running 'conda init --reverse $SHELL'? [yes|no]
[no] >>> no
```

마지막 질문에서 no라고 한 경우 아래 명령 실행하여 base 가상환경 활성화

```
[root@jupyter-notebook ~]# eval "$(/root/miniconda3/bin/conda shell.bash hook)"
```

설치 완료 후 터미널 재시작하자.

yes라고 한 경우 자동으로 conda init을 실행하는 듯 하다. no 한 경우는 아래와 같이 conda init 실행한다.

```
[root@jupyter-notebook ~]# conda init
```

# K8S-Spark 인프라 구축

GCP 주피터 노트북 환경 세팅하기

현재 가상환경 내에 있다면 `conda deactivate`로 탈출하자.

## ② spark341 가상환경 생성

```
# 중간에 가상환경 들어가서 실행하는 부분 때문에 한 줄씩 실행해야 함
# spark 3.4.1 wants Python 3.9.19
conda create --name spark341 python=3.9.19 -y
conda activate spark341
# conda-forge에서 openjdk를 가져와 설치하라
conda install -c conda-forge openjdk=11.0.13 -y
pip install pyspark==3.4.1
# 주피터 노트북에서 커널로 선택할 수 있도록 설정(이걸 할 필요가 있나?)
conda install -n spark341 ipykernel --update-deps --force-reinstall -y
```

## ③ 주피터 노트북 설치

```
(spark341) [root@jupyter-notebook ~]# conda install jupyter -y
(spark341) [root@jupyter-notebook ~]# jupyter notebook --generate-config # 한 번만 실행하면 되는 듯
주피터 노트북은 root 권한으로 실행되는 것을 싫어하므로 root로 실행하는 옵션을 준다. (아래 둘 중 하나 실행)
(spark341) [root@jupyter-notebook ~]# jupyter notebook --allow-root
(spark341) [root@jupyter-notebook ~]# jupyter notebook --ip='0.0.0.0' --port=8888 --no-browser --allow-root
아래 그림의 둘 중 하나 링크 클릭하여 브라우저에서 확인한다.
```

```
[C 2024-04-06 15:46:44.629 ServerApp]
To access the server, open this file in a browser:
file:///root/.local/share/jupyter/runtime/jpserver-8899-open.html
Or copy and paste one of these URLs:
http://localhost:8888/tree?token=32854a55d699f9f085f5a041b8c2c999b25d3252682089f0
http://127.0.0.1:8888/tree?token=32854a55d699f9f085f5a041b8c2c999b25d3252682089f0
```

# K8S-Spark 인프라 구축

## GCP 주피터 노트북 환경 세팅하기

운이 좋았던 것인지 앞장에서 나온 localhost 링크가 정상 작동하는 이유는 VSCode를 이용했기 때문이었다.  
GCP에서 제공하는 브라우저 SSH 통해서 주피터 노트북 실행시키면 아래와 같은 오류가 나온다.

```
[I 2024-04-06 17:47:55.904 ServerApp] Jupyter Server 2.10.0 is running at:
[I 2024-04-06 17:47:55.904 ServerApp] http://localhost:8888/tree?token=fee2d51df2f20c3cd0a6f3abbae86179837adf764e4a1a9d
[I 2024-04-06 17:47:55.904 ServerApp] http://127.0.0.1:8888/tree?token=fee2d51df2f20c3cd0a6f3abbae86179837adf764e4a1a9d
[I 2024-04-06 17:47:55.904 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[W 2024-04-06 17:47:55.908 ServerApp] No web browser found: Error('could not locate runnable browser').
[C 2024-04-06 17:47:55.909 ServerApp]

To access the server, open this file in a browser:
file:///root/.local/share/jupyter/runtime/jpserver-15268-open.html
Or copy and paste one of these URLs:
http://localhost:8888/tree?token=fee2d51df2f20c3cd0a6f3abbae86179837adf764e4a1a9d
http://127.0.0.1:8888/tree?token=fee2d51df2f20c3cd0a6f3abbae86179837adf764e4a1a9d
```

그리고 localhost:8888로 시작하는 링크로 접속해보아도 연결이 되지 않는다.

VSCode를 사용할 땐 8888 포트포워딩이 자동으로 실행되는 것을 power shell 명령을 통해서 확인할 수 있었다.

```
Get-NetTCPConnection | Where-Object {$_.State -eq "Listen"}
```

이 경우 다음 명령으로 주피터 노트북을 실행시키면 VM의 외부IP 통해서 접속 가능하다.

```
(myenv) [root@jupyter-notebook ~]# jupyter notebook --ip='0.0.0.0' --port=8888 --no-browser --allow-root
```

# K8S-Spark 인프라 구축

GCP 주피터 노트북 접속 및 코드 테스트

주피터 노트북 접속 후 우상단 New 버튼 클릭하여 새로운 노트북을 띄운다.  
아래 셀 코드들을 하나하나 실행해보자.

```
%pip install pyspark # 가상환경에서 pyspark3.4.1을 설치했기 때문에 이 줄은 필요없다.
```

```
from pyspark.sql import SparkSession
```

```
spark = SparkSession.builder \
    .master("spark://34.125.136.103:30077") \
    .appName("MyApp2") \
    .config("spark.kubernetes.namespace", "default") \
    .getOrCreate()
```

```
spark
```

```
# Spark DataFrame 예제
df = spark.createDataFrame([{"Name": "John Doe", "Age": 30}, {"Name": "Jane Doe", "Age": 25}])
df.printSchema()
```

```
df.show()
```

```
spark.stop() # 이 줄은 다음 슬라이드 실습 중에 실행합니다
```

그렇다. PySpark는 Java가 필요하다. Java를 설치하자.

# K8S-Spark 인프라 구축

로컬 VSCode에서 GCP 주피터 노트북 접속 및 코드 테스트

VSCode에서 spark01.ipynb파일을 새로 만들고 오른쪽 상단 파이썬 커널을 클릭하여 다른 커널을 선택한다.

기존 주피터 서버를 선택하고 `http://주피터서버IP:8888/tree?token=토큰ID` 를 입력 후 커널 선택

아래 코드로 정상 작동 확인

```
%pip install pyspark
```

```
from pyspark.sql import SparkSession
```

```
spark = SparkSession.builder \
    .master("spark://34.125.136.103:30077") \
    .appName("MyApp2") \
    .config("spark.kubernetes.namespace", "default") \
    .getOrCreate()
```

```
spark
```

```
# Spark DataFrame 예제
df = spark.createDataFrame([{"Name": "John Doe", "Age": 30}, {"Name": "Jane Doe", "Age": 25}])
df.printSchema()
```

```
df.show()
```

```
spark.stop()
```

# K8S-Spark 데이터 분석

Spark에서 S3 파일 읽고 쓰기

전제조건

- AWS IAM 생성 및 accessKeys.csv 파일 다운로드 상태

주피터 노트북 서버에서 새 파일을 만들고 아래와 같이 입력

```
[default]
```

```
aws_access_key_id = AKIAZI2LB26L37BLLQP4
```

```
aws_secret_access_key = crBhJRQiBeTxxQd+dlPPOFzO8sop68j4JHJPFux2
```

/root/.aws/credentials 라는 파일 명으로 저장

(이 파일은 실행 파일이 아니므로 chmod +x 할 필요 없음)

# S3 연동



# K8S-Spark 데이터 분석

Spark에서 S3 파일 읽고 쓰기

# S3 연동

```
from pyspark.sql import SparkSession
from pyspark.conf import SparkConf

# Create a Spark session with your AWS Credentials
conf = (
    SparkConf()
    .setAppName("MY_APP") # replace with your desired name
    .set("spark.jars.packages", "io.delta:delta-core_2.12:2.3.0,org.apache.hadoop:hadoop-aws:3.3.2")
    .set("spark.sql.catalog.spark_catalog", "org.apache.spark.sql.delta.catalog.DeltaCatalog")
    .set("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension")
    .set("spark.sql.shuffle.partitions", "4") # default is 200 partitions which is too many for local
    .set("spark.hadoop.fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem")
    .set("com.amazonaws.services.s3.enableV4", "true")
    .set("spark.hadoop.fs.s3a.endpoint", "s3.amazonaws.com")
    .set("spark.hadoop.fs.s3a.aws.credentials.provider", "com.amazonaws.auth.profile.ProfileCredentialsProvider")
    .setMaster("spark://34.125.136.103:30077") # replace the * with your desired number of cores. * for use all.
)

spark = SparkSession.builder.config(conf=conf).getOrCreate()

df = spark.read.format('json').load('s3a://jolaajoayo-spark-0001/spark2-sql/airports/airport-codes.csv.json')
df.show(5, truncate=False)
spark.stop()
```

# K8S-Spark 데이터 분석

Spark에서 S3 파일 읽고 쓰기

# S3 연동

```
from pyspark.sql import SparkSession
from pyspark.conf import SparkConf

# Create a Spark session with your AWS Credentials
conf = (
    SparkConf()
    .setAppName("MY_APP") # replace with your desired name
    .set("spark.jars.packages", "io.delta:delta-core_2.12:2.3.0,org.apache.hadoop:hadoop-aws:3.3.2")
    .set("spark.sql.catalog.spark_catalog", "org.apache.spark.sql.delta.catalog.DeltaCatalog")
    .set("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension")
    .set("spark.sql.shuffle.partitions", "4") # default is 200 partitions which is too many for local
    .set("spark.hadoop.fss3a.impl", "org.apache.hadoop.fss3a.S3AFileSystem")
    .set("com.amazonaws.services.s3.enableV4", "true")
    .set("spark.hadoop.fss3a.endpoint", "s3.amazonaws.com")
    .set("spark.hadoop.fss3a.aws.credentials.provider", "com.amazonaws.auth.profile.ProfileCredentialsProvider")
    .setMaster("spark://34.125.136.103:30077") # replace the * with your desired number of cores. * for use all.
)

spark = SparkSession.builder.config(conf=conf).getOrCreate()

df = spark.read.format("json").load('s3a://jolajoayo-spark-0001/spark2-sql/airports/airport-codes.csv.json')
df.show(5, truncate=False)
spark.stop()
```

*kafka* 연동

# K8S-Spark 데이터 분석

프로듀서 테스트

사용할 kafka 서버([hosts](#))들을 hosts에 추가한다.  
주피터 서버에서 파일 열기(Ctrl+O), `/etc/hosts` 선택

아래 파일들의 스크립트를 복사하여 jupyter 서버에 만든다.  
[book\\_data\\_pb2.py](#), [kakaoBookProducer.py](#), [keywords.py](#)

단! kakao~ 파일에서 44번째 줄을 아래와 같이 수정한다.  
`topic = "book-sdh"`

저장 경로는 `/root/producer` 디렉토리 아래에 같은 파일 명으로 만든다.  
만들어진 파일들에 실행 권한을 부여한다.

*# producer 폴더의 모든 파일에 실행권한 부여*

```
[root@jupyter-notebook ~]# chmod -R +x ./producer
```

*# 실행에 필요한 모듈 설치*

```
[root@jupyter-notebook ~]# pip install confluent_kafka google protobuf
```

*# producer 실행*

```
[root@jupyter-notebook ~]# python ./producer/kakaoBookProducer.py
```

# kafka

# 연동

# K8S-Spark 데이터 분석

## 컨슈머 테스트

아래 파일의 스크립트를 복사하여 jupyter 서버에 만든다.

저장 경로는 /root/consumer 디렉토리 아래에 같은 파일 명으로 만든다.

[sparkConsumer.py](#)

단! kakao~ 파일에서 44번째 줄을 아래와 같이 수정한다.

topic = "book-sdh"

만들어진 파일들에 실행 권한을 부여한다.

*# producer 폴더의 모든 파일에 실행권한 부여*

`[root@jupyter-notebook ~]# chmod -R +x ./producer`

*# 실행에 필요한 모듈 설치*

`[root@jupyter-notebook ~]# pip install confluent_kafka google protobuf`

*# producer 실행*

`[root@jupyter-notebook ~]# python ./producer/kakaoBookProducer.py`

# kafka

# 연동

## K8S-Spark 데이터 분석

# Create a Spark session with your AWS Credentials

```
conf = (  
    SparkConf()  
        .setAppName("MY_APP") # replace with your desired name  
        .set("spark.jars.packages", "  
            "io.delta:delta-core_2.12:2.3.0"₩  
            ",org.apache.hadoop:hadoop-aws:3.3.2"₩  
            ",org.apache.commons:commons-pool2:2.11.1"₩  
            ",org.apache.kafka:kafka-clients:3.4.0"₩  
            ",org.apache.spark:spark-protobuf_2.13:3.4.1"₩  
            ",org.apache.spark:spark-sql-kafka-0-10_2.13:3.4.1"₩  
            ",org.apache.spark:spark-token-provider-kafka-0-10_2.13:3.4.1"₩  
        )  
        .set("spark.sql.catalog.spark_catalog","org.apache.spark.sql.delta.catalog.DeltaCatalog")  
        .set("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension")  
        .set("spark.sql.shuffle.partitions", "4") # default is 200 partitions which is too many for local  
        .set("spark.hadoop.fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem")  
        .set("com.amazonaws.services.s3.enableV4", "true")  
        .set("spark.hadoop.fs.s3a.endpoint", "s3.amazonaws.com")  
        .set("spark.hadoop.fs.s3a.aws.credentials.provider", "com.amazonaws.auth.profile.ProfileCredentialsProvider")  
        .setMaster("local[*]") # replace the * with your desired number of cores. * for use all.  
    )
```

# *S3/kafka*

# 연동

# K8S-Spark 데이터 분석

한거:

로컬 카프카 설치 - 실행

공유기 포트포워딩 설정

로컬 방화벽 인바운드 9092, 29092

GCP 방화벽 아웃바운드 9092, 29092 (이건 의미 없는 듯)

*S3/kafka*

연동

## K8S-Spark 인프라 구축

이런 생각한 사람 저 뿐인가요?

"주피터 노트북만 사용하려고 VM을 쓰는 것은 에바잖아?!"

```
conda install pyspark
```

이건 VSCode에서도 주피터 노트북을 쓰고 싶을 때

```
export SPARK_HOME=/root/miniconda3/envs/myenv/lib/python3.11/site-packages/pyspark
```

```
export PYSPARK_DRIVER_PYTHON=jupyter
```

```
export PYSPARK_DRIVER_PYTHON_OPTS='notebook --ip=0.0.0.0 --port=8888 --no-browser --allow-root'
```

```
pyspark --packages graphframes:graphframes:0.8.3-spark3.5-s_2.13
```

아니면 그냥 브라우저에서 사용할 때

```
export SPARK_HOME=/root/miniconda3/envs/myenv/lib/python3.11/site-packages/pyspark
```

```
export PYSPARK_DRIVER_PYTHON=jupyter
```

```
export PYSPARK_DRIVER_PYTHON_OPTS='notebook'
```

```
pyspark --packages graphframes:graphframes:0.8.3-spark3.5-s_2.13
```

*jupyter on*

*k8s*



# K8S-Spark 인프라 구축



GCP 주피터 노트북 접속 및 코드 테스트 – SPARK\_HOME 설정하기

나중에 kafka 나 S3 접속에 사용할 jar 파일 설치를 위해 SPARK\_HOME을 설정해야 한다.

주피터 노트북 내에서 아래 코드 실행하여 pyspark 설치 위치를 찾자

```
import pyspark
print(pyspark.__file__)
```

(출력결과 예시) /root/miniconda3/envs/myenv/lib/python3.11/site-packages/pyspark

결과를 복사하고 다시 터미널에서 아래 스크립트를 실행하여

SPARK\_HOME을 만들고 PATH에 등록한다.

```
[root@jupyter-notebook ~]# echo 'export SPARK_HOME=/root/miniconda3/envs/myenv/lib/python3.11/site-packages/pyspark' >> ~/.bashrc
```

```
[root@jupyter-notebook ~]# echo 'export PATH=$SPARK_HOME /bin:$PATH' >> ~/.bashrc
```

```
[root@jupyter-notebook ~]# source ~/.bashrc
```

자동으로 base 가상환경에 들어갔다면 conda deactivate로 가상환경에서 빠져나온다.

다시 myenv 가상환경에 들어가서 주피터 노트북을 실행한다.

```
[root@jupyter-notebook ~]# conda activate myenv
```

```
[root@jupyter-notebook ~]# jupyter notebook --ip='0.0.0.0' --port=8888 --no-browser --allow-root
```

## K8S-Spark 인프라 구축

이런 생각한 사람 저 뿐인가요?

"주피터 노트북만 사용하려고 VM을 쓰는 것은 에바잖아?!"

그래서 이미 구축한 kube에 주피터 노트북용 pod를 하나 만들고 거기서 스파크 클러스터에 접근하는 환경 세팅

연구해보자!

<https://qkqhxda1.tistory.com/1138>

K8S-Spark 인프라 구축

기/에/아

K8S-Spark 인프라 구축

질/취/선