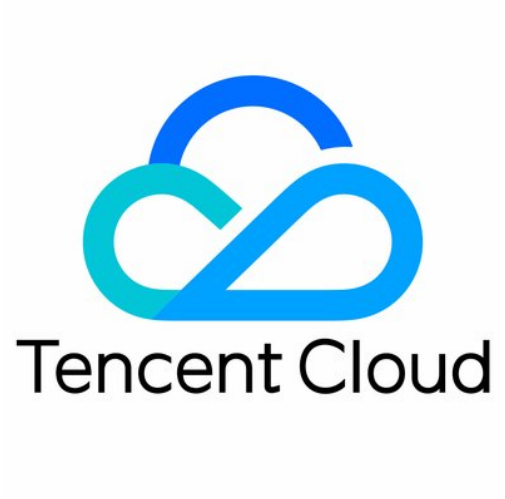
Tencent Kubernetes Engine

모범 사례

제품 문서



목록:

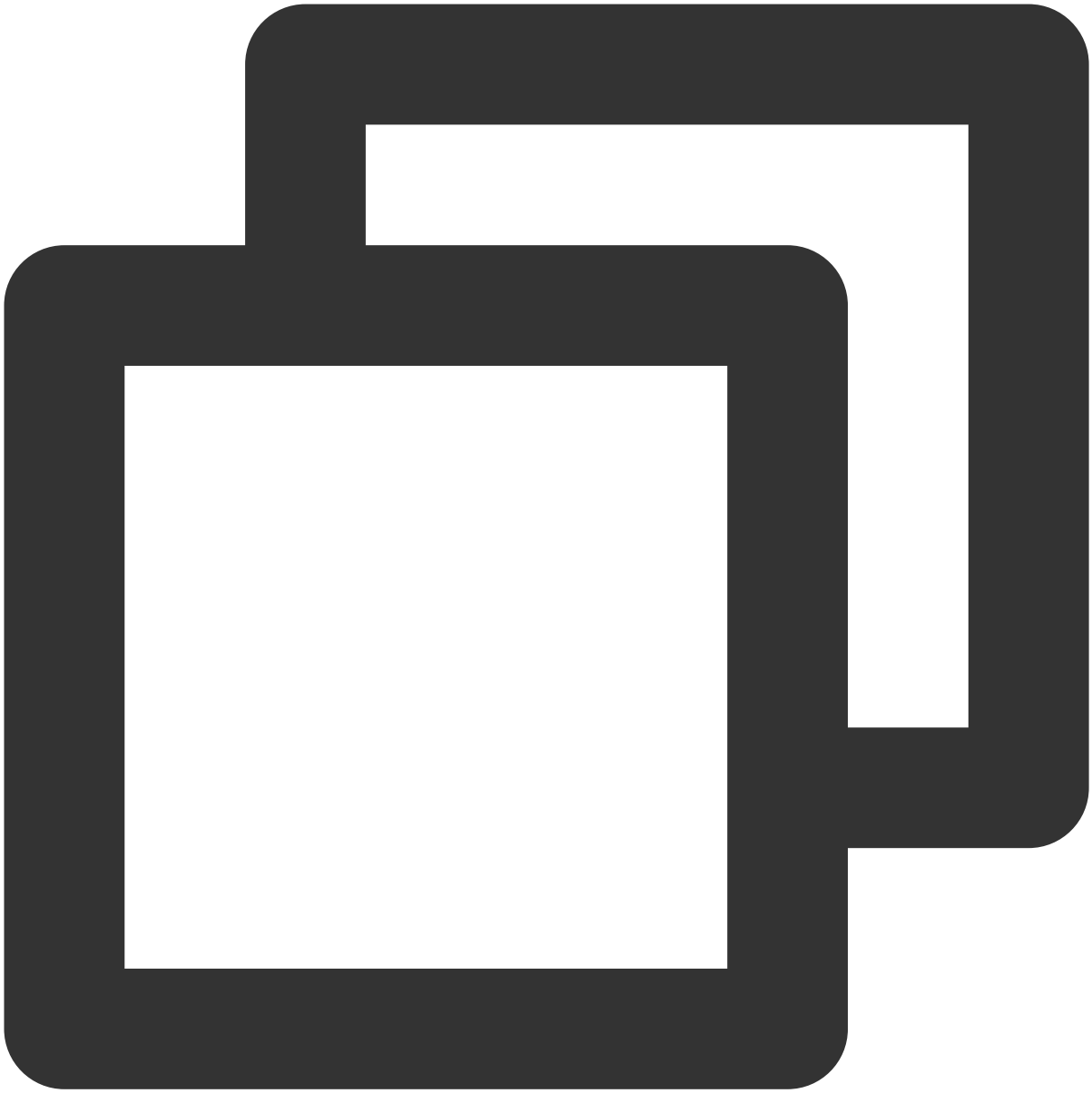
모범 사례
Serverless 클러스터
EIP를 통해 공중망 액세스

최종 업데이트 날짜: : 2023-04-26 18:43:22

현재 TKE Serverless를 사용하면 Pod에 EIP를 바인딩할 수 있습니다. template annotation에서 EIP를 선언하기만 하면 됩니다. 자세한 내용은 Annotation을 참고하십시오.
EIP와 관련된 네 가지 Annotation이 있습니다.

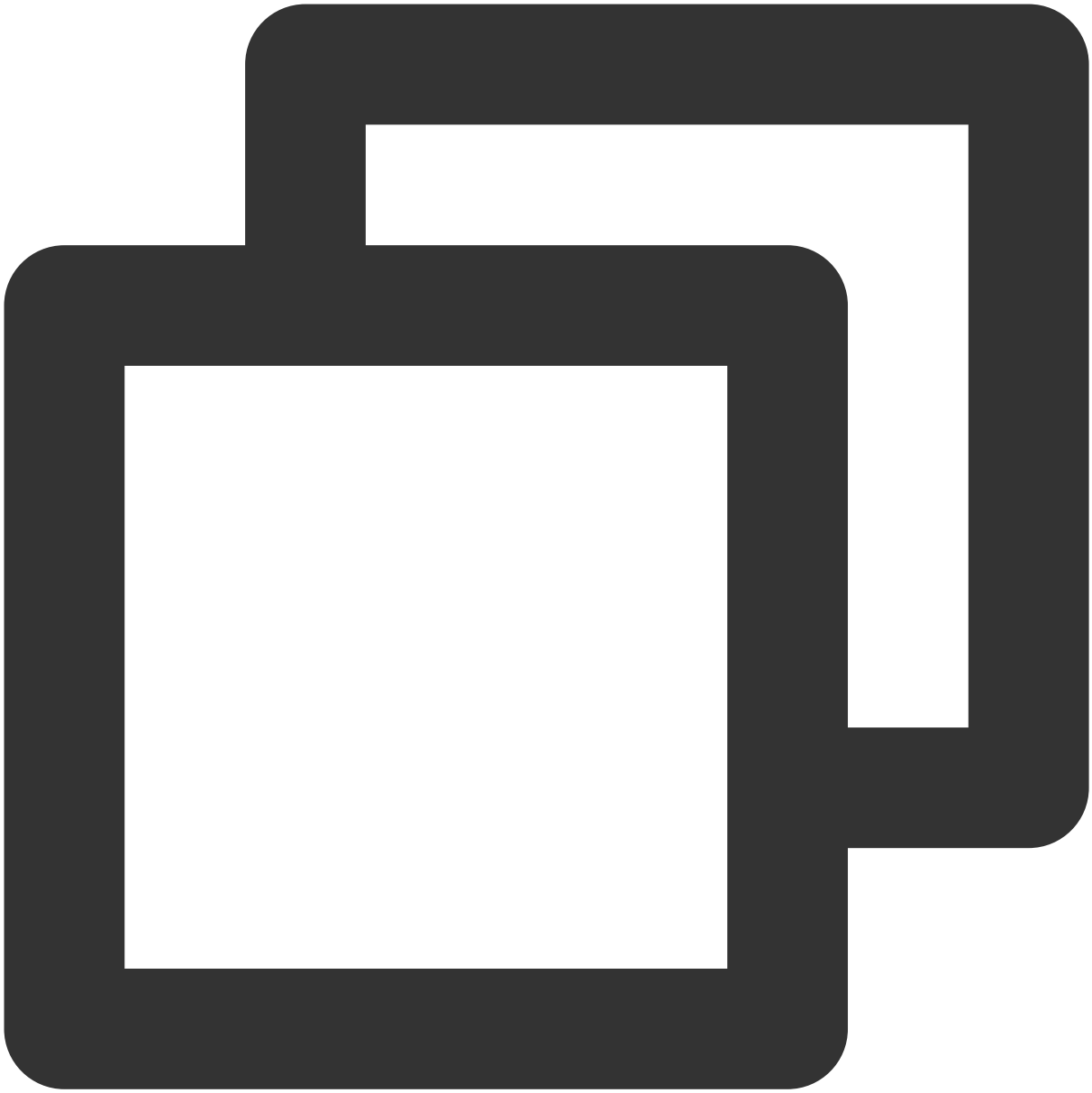
Annotation Key	Annotation Value 및 설명	필수 입력
eks.tke.cloud.tencent.com/eip-attributes	Workload의 Pod가 EIP에 바인딩되어야 함을 나타냅니다. 값이 `""` 이면 기본 EIP 구성으로 바인딩이 생성됩니다. `""` 에 EIP의 TencentCloud API 매개변수 json 문자열을 입력하여 구성을 사용자 정의할 수 있습니다.	EIP를 바인딩해야 하는 경우, 필수 항목입니다.
eks.tke.cloud.tencent.com/eip-claim-delete-policy	Pod가 삭제된 후 EIP 회수 여부를 나타냅니다. Never는 회수하지 않음을 나타냅니다. 기본값은 회수 입니다.	No
eks.tke.cloud.tencent.com/eip-id-list	기존 EIP가 사용되며 statefulset만 지원됨을 나타냅니다. Pod가 종료된 후에는 해당 EIP가 기본적으로 회수되지 않습니다. statefulset pod의 수는 이 Annotation에 지정된 eipId 값의 수를 초과할 수 없습니다.	No

1. 공중망 액세스를 위해 EIP를 Workload 또는 Pod에 바인딩하려는 경우 가장 간단한 방법은 다음과 같이 해당 Workload 또는 Pod의 `annotation` 아래에 `eks.tke.cloud.tencent.com/eip-attributes: ""` 플래그를 추가하는 것입니다.



```
metadata:
  name: tf-cnn
  annotations:
    eks.tke.cloud.tencent.com/eip-attributes: ""  #EIP가 필요하며 기본 구성 사용
```

2. 관련 이벤트를 보려면 다음 명령을 실행하십시오.



```
kubectl describe pod [name]
```

아래와 같이 EIP와 관련된 두 개의 새로운 이벤트가 있음을 알 수 있습니다. 실행 성공을 나타냅니다.

```
Events:
  Type    Reason          Age   From               Message
  ----    ------          ----  ----               -------
  Normal  Scheduled       106s  default-scheduler  Successfully assigned default/tf-cnn to eklet-subnet-6rjbxwwb
  Normal  AllocatedEip    95s   eklet              Successfully allocate eip eip-[redacted], ip 43.[redacted]
  Normal  Starting        81s   eklet              Starting pod sandbox eks-j0i3y99u
  Normal  Starting        66s   eklet              Sync endpoints
  Normal  Pulling         64s   eklet              Pulling image "hkccr.ccs.tencentyun.com/carltk/[redacted]:latest"
  Normal  Pulled          64s   eklet              Successfully pulled image "hkccr.ccs.tencentyun.com/carltk/[redacted]
  Normal  Created         64s   eklet              Created container tf-cnn
  Normal  Started         63s   eklet              Started container tf-cnn
  Normal  AssociatedEip   50s   eklet              Successfully associate eip eip-[redacted]
```

3. log 파일을 보면 아래와 같이 데이터셋이 정상적으로 다운로드되는 것을 확인할 수 있습니다.

```
I0803 07:56:37.621758 140120123275072 dataset_builder.py:400] Generating dataset mnist (/root/tensorflow_datasets/mnist/3.0.1)
I0803 07:56:38.315572 140120123275072 dataset_builder.py:433] Dataset mnist is hosted on GCS. It will automatically be downloade
local data directory. If you'd instead prefer to read directly from our public
GCS bucket (recommended if you're running on GCP), you can instead pass
`try_gcs=True` to `tfds.load` or set `data_dir=gs://tfds-data/datasets`.

Downloading and preparing dataset 11.06 MiB (download: 11.06 MiB, generated: 21.00 MiB, total: 32.06 MiB) to /root/tensorflow_da
Dl Completed...: 100%|          | 4/4 [00:02<00:00,  1.92 file/s]
I0803 07:56:40.842971 140120123275072 dataset_info.py:358] Load dataset info from /root/tensorflow_datasets/mnist/3.0.1.incomple
```

참고

하루에 신청할 수 있는 EIP의 수가 제한되어 있기 때문에 EIP는 매일 여러 번 실행해야 하는 작업에는 적합하지 않습니다.

Serverless 클러스터에서 딥러닝 사용
딥러닝 컨테이너 이미지 빌드

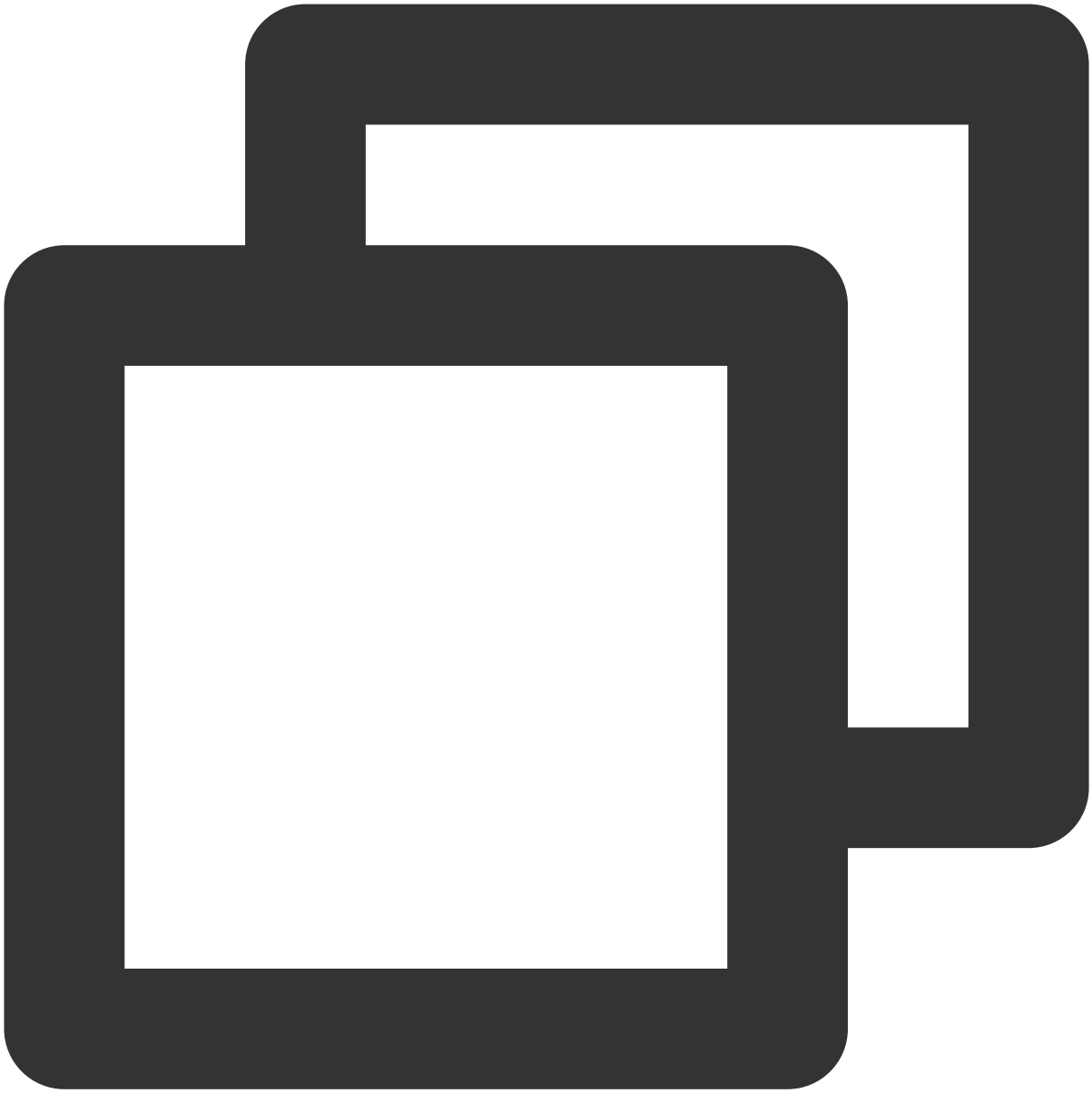
최종 업데이트 날짜: : 2023-04-28 15:30:11

작업 시나리오

본 문서 시리즈는 직접 TensorFlow 배포에서 후속 Kubeflow 배포까지 TKE Serverless 클러스터에서 딥 러닝을 배포하는 방법을 설명하고, 컨테이너 기반 딥 러닝을 구현하기 위한 포괄적인 체계를 제공하기 위한 것입니다. 본 문서는 딥 러닝을 배포하는 더 쉽고 빠른 방법을 제공하는 딥 러닝 컨테이너 이미지를 생성하는 방법에 중점을 둡니다.
공용 이미지는 본 문서의 딥 러닝 배포 요구 사항을 충족할 수 없습니다. 따라서 자체 구축 이미지를 사용합니다.
딥 러닝 프레임워크 TensorFlow-gpu 외에도 이 이미지에는 GPU 기반 교육에 필요한 cuda 및 cudnn도 포함되어 있으며 CV, NLP 및 RS와 같은 분야를 위한 SOTA 모델을 포함하여 공식 TensorFlow 딥 러닝 모델을 통합합니다. 모델에 대한 자세한 내용은 Model Garden for TensorFlow를 참고하십시오.

작업 단계

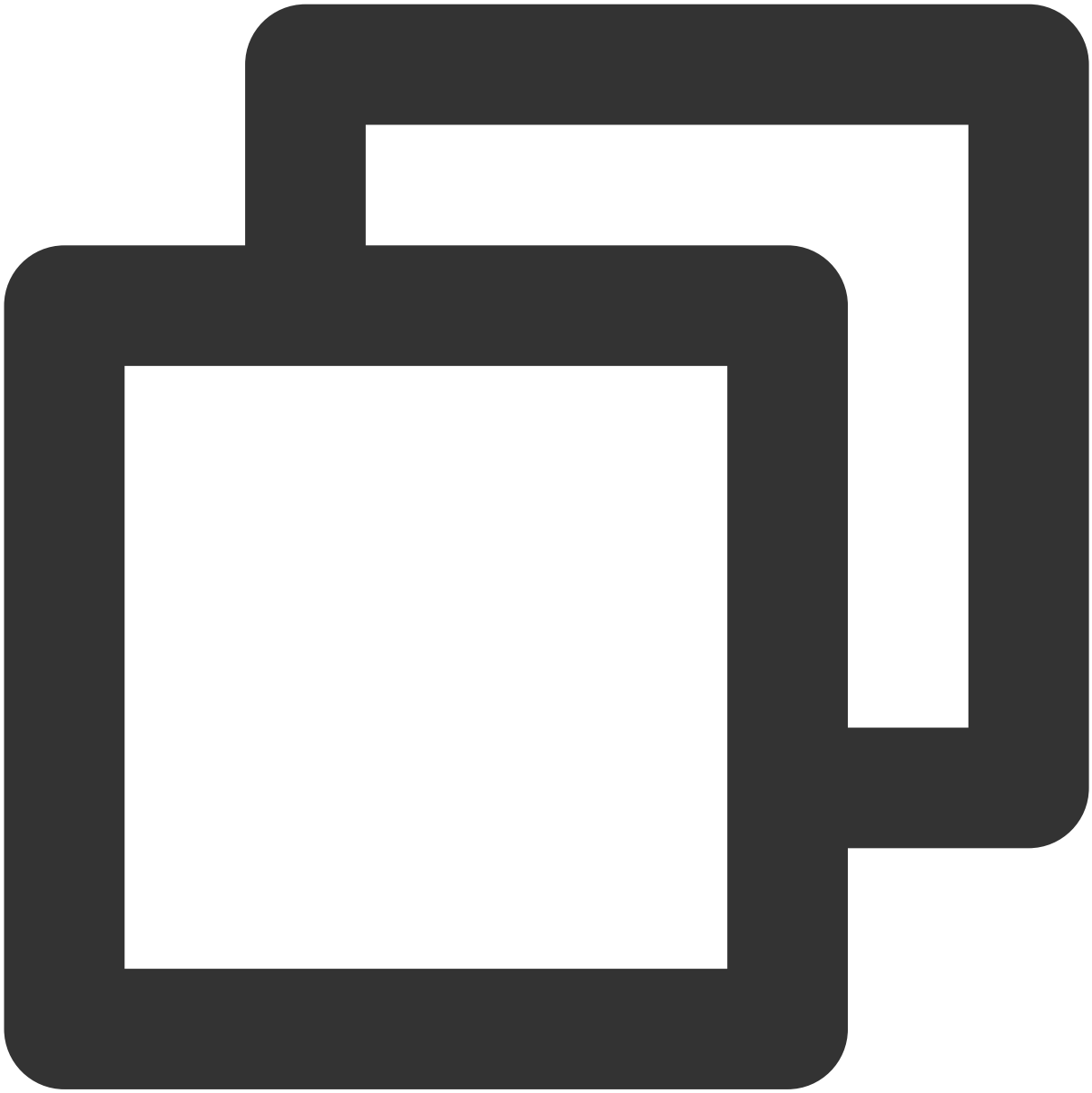
1. 이 예시에서는 Docker 컨테이너를 사용하여 이미지를 생성합니다. 다음과 같이 Dockerfile을 준비합니다.



```
FROM nvidia/cuda:11.3.1-cudnn8-runtime-ubuntu20.04
RUN apt-get update -y \\
 && apt-get install -y python3 \\
    python3-pip \\
    git \\
 && git clone git://github.com/tensorflow/models.git \\
 && apt-get --purge remove -y git \\    #불필요한 컴포넌트 즉시 제거(선택 사항)
 && rm -rf /var/lib/apt/lists/*         #apt를 통해 설치할 패키지 삭제(선택 사항)
 && mkdir /tf /tf/models /tf/data       #마운트 지점으로 사용할 수 있는 스토리지 모델 및 데이
ENV PYTHONPATH $PYTHONPATH:/models
ENV LD_LIBRARY_PATH $LD_LIBRARY_PATH:/usr/local/cuda-11.3/lib64:/usr/lib/x86_64-lin
```

```
RUN pip3 install --user -r models/official/requirements.txt \\
 && pip3 install tensorflow
```

2. 다음 명령을 실행하여 배포합니다.



```
docker build -t [name]:[tag] .
```

설명

Python, TensorFlow, cuda, cudnn 및 모델 라이브러리와 같은 필수 컴포넌트를 설치하는 단계는 이 문서에서 자세히 설명하지 않습니다.

관련 설명

이미지

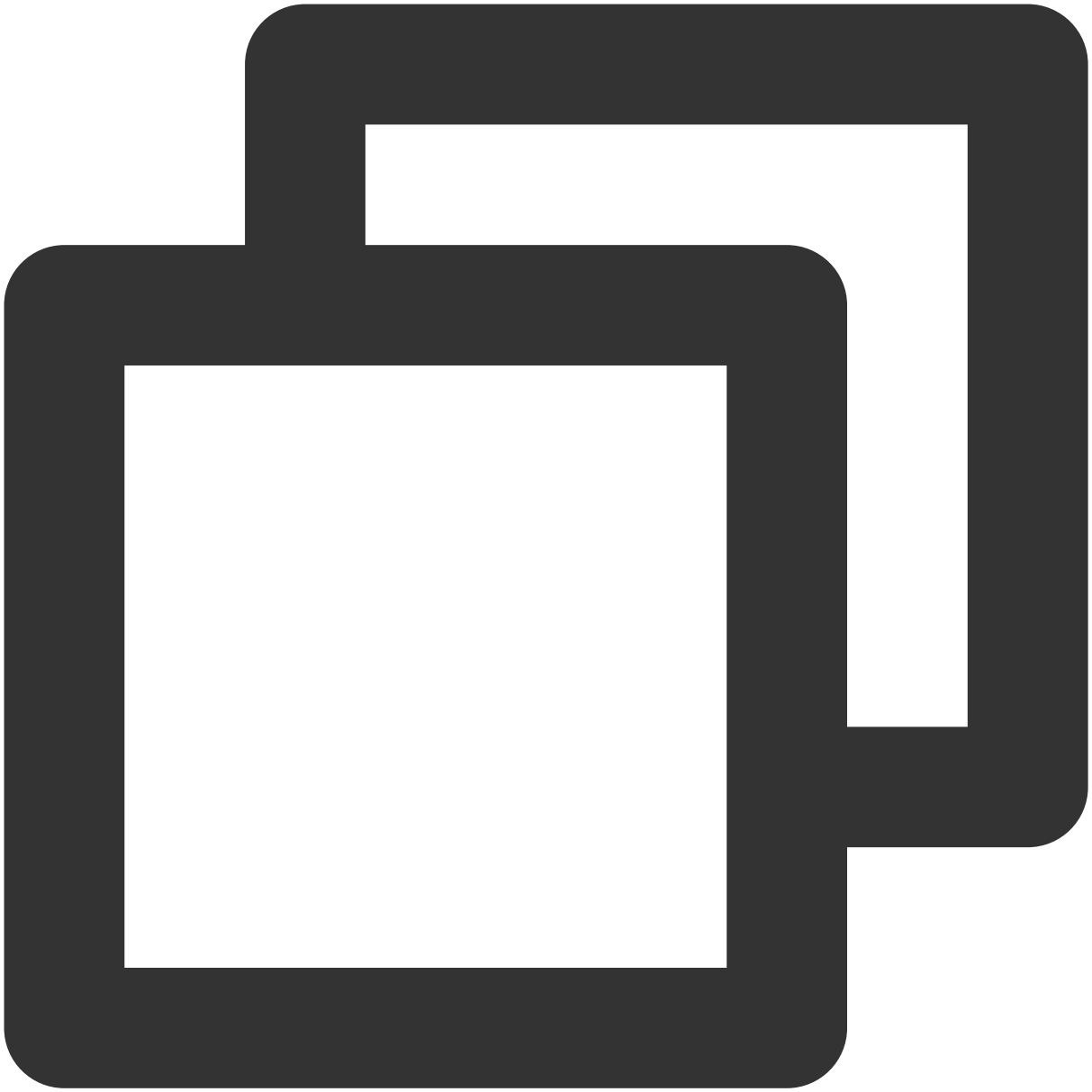
기본 이미지 nvidia/cuda의 경우 CUDA 컨테이너 이미지는 CUDA 지원 플랫폼 및 아키텍처에 사용하기 쉬운 배포판을 제공합니다. 여기에서는 cuda 11.3.1 및 cudnn 8이 선택됩니다. 버전에 대한 자세한 내용은 Supported tags를 참고하십시오.

환경 변수

이 문서의 모범 사례를 구현하기 전에 `LD_LIBRARY_PATH` 환경 변수에 특별한 주의를 기울여야 합니다. `LD_LIBRARY_PATH` 는 일반적으로 libcudart.so.[version], ibcusolver.so.[version], libcudnn.so.[version]과 같은 libxxxx.so 형식으로 동적 링크 라이브러리의 설치 경로를 나열하며, 이 예시에서는 cuda 및 cudnn을 연결하는 데 사용됩니다. `ll` 명령을 실행하여 아래와 같이 경로를 볼 수 있습니다.

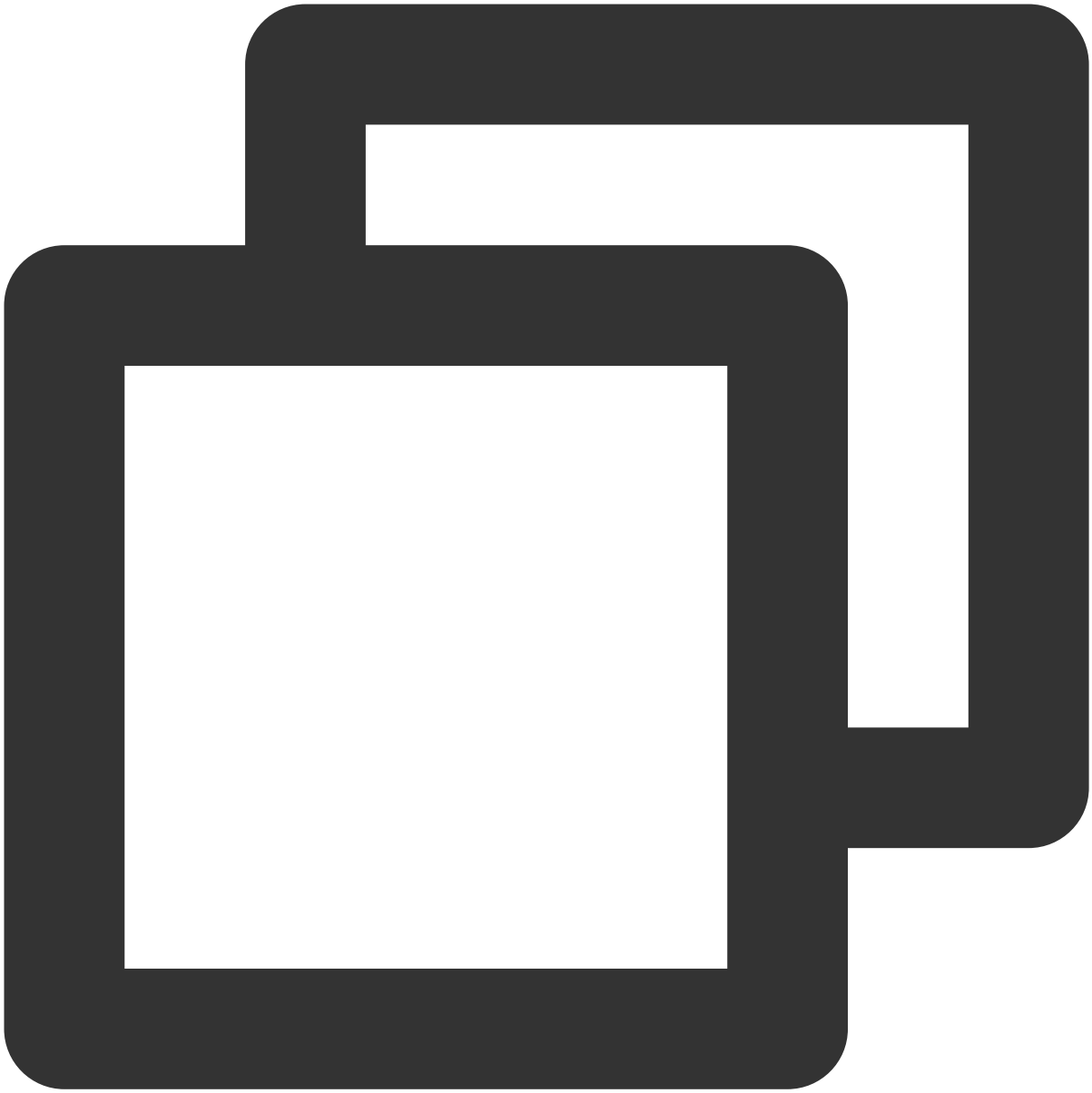
```
root@5a949761c669:/usr/local/cuda-11.3/lib64# ll
total 1534336
drwxr-xr-x 1 root root      4096 Jul  2 03:57 ./
drwxr-xr-x 1 root root      4096 Jul  2 03:57 ../
lrwxrwxrwx 1 root root        16 May  4 02:30 libOpenCL.so.1 -> libOpenCL.so.1.0
lrwxrwxrwx 1 root root        18 May  4 02:30 libOpenCL.so.1.0 -> libOpenCL.so.1.0.0
-rw-r--r-- 1 root root     30856 May  4 02:30 libOpenCL.so.1.0.0
lrwxrwxrwx 1 root root        23 May 13 23:26 libcublas.so.11 -> libcublas.so.11.5.1
-rw-r--r-- 1 root root 121866104 May 13 23:26 libcublas.so.11.5.1.109
lrwxrwxrwx 1 root root        25 May 13 23:26 libcublasLt.so.11 -> libcublasLt.so.11
-rw-r--r-- 1 root root 263770264 May 13 23:26 libcublasLt.so.11.5.1.109
lrwxrwxrwx 1 root root        21 May  4 02:30 libcudart.so.11.0 -> libcudart.so.11.3
-rw-r--r-- 1 root root    619192 May  4 02:30 libcudart.so.11.3.109
lrwxrwxrwx 1 root root        22 May 13 23:30 libcufft.so.10 -> libcufft.so.10.4.2.1
-rw-r--r-- 1 root root 190417864 May 13 23:30 libcufft.so.10.4.2.109
lrwxrwxrwx 1 root root        23 May 13 23:30 libcufftw.so.10 -> libcufftw.so.10.4.2
-rw-r--r-- 1 root root    631888 May 13 23:30 libcufftw.so.10.4.2.109
```

공식 이미지의 Dockerfile 소스 코드를 기반으로 다음 명령어를 실행합니다.



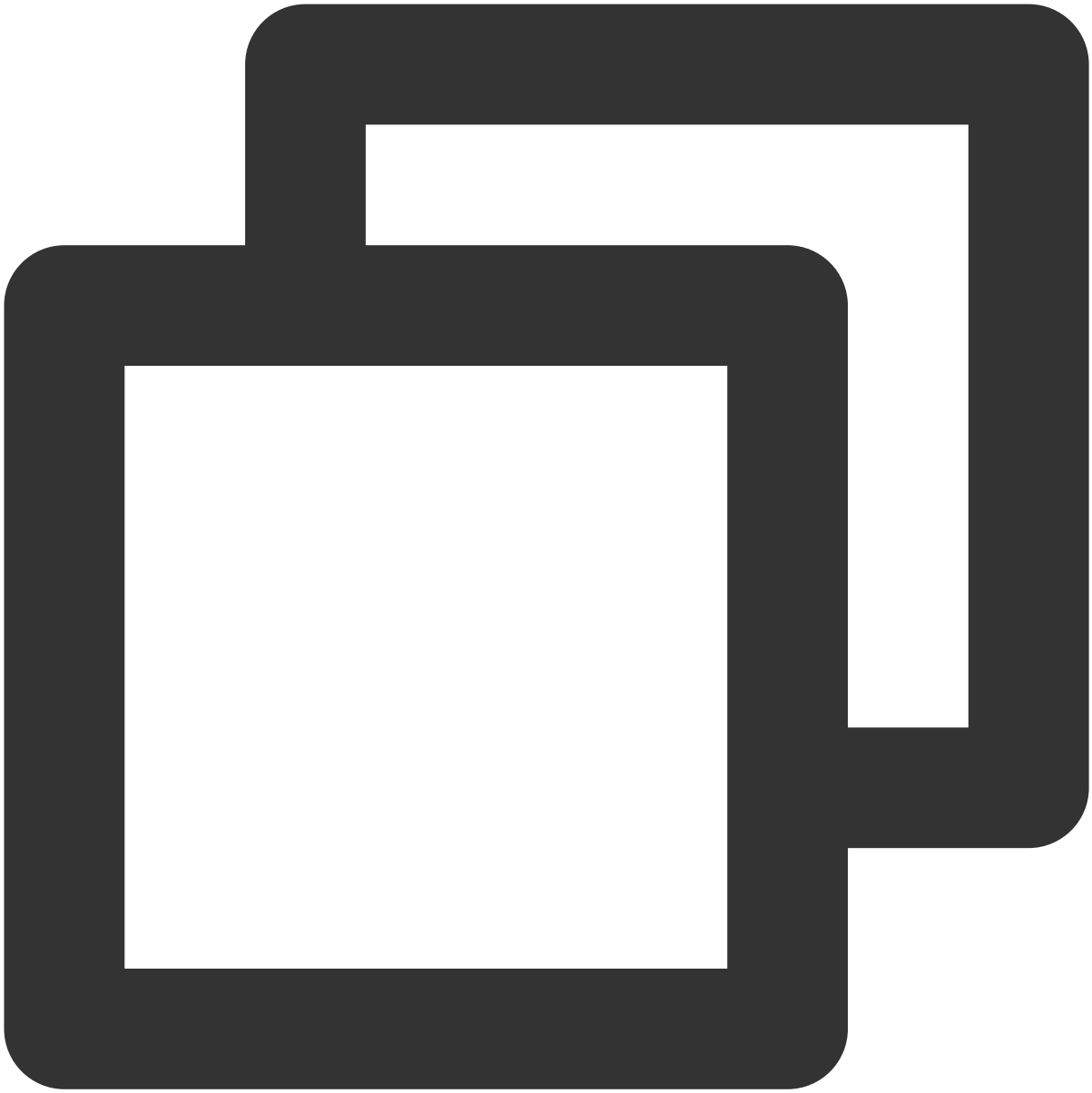
```
ENV LD_LIBRARY_PATH /usr/local/nvidia/lib:/usr/local/nvidia/lib64
```

여기서 `/usr/local/nvidia/lib` 는 cuda 경로의 소프트 링크를 가리키며 cuda를 위해 준비됩니다. 그러나 cudnn이 포함된 태그에서는 cudnn만 설치되고 cudnn에 대해 `LD_LIBRARY_PATH` 가 지정되지 않아 Warning이 리포트되고 GPU 리소스를 사용할 수 없게 될 수 있습니다. 오류는 아래와 같습니다.



```
Could not load dynamic library 'libcudnn.so.8'; dlerror: libcudnn.so.8: cannot open
Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned
```

이러한 오류가 리포트되면 cudnn 경로를 수동으로 추가할 수 있습니다. 여기에서 다음 명령을 실행하여 이미지를 실행하고 libcudnn.so의 경로를 볼 수 있습니다.



```
docker run -it nvidia/cuda:[tag] /bin/bash
```

소스 코드에서 볼 수 있듯이 cudnn은 기본적으로 `apt-get install` 명령으로 `/usr/lib` 에 설치됩니다. 이 예시에서 libcudnn.so.8의 실제 경로는 `/usr/lib/x86_64-linux-gnu#` 아래에 있으며 콜론 뒤 끝에 추가됩니다.
실제 경로는 태그 및 시스템에 따라 다를 수 있습니다. 소스 코드의 경로와 실제로 표시되는 경로가 우선합니다.

후속 작업

후속 작업에 대해서는 TKE Serverless에서 딥 러닝 실행 문서를 참고하십시오.

FAQ

구현 과정 중 문제가 발생하면 문제 해결을 위해 FAQ를 참고하십시오.

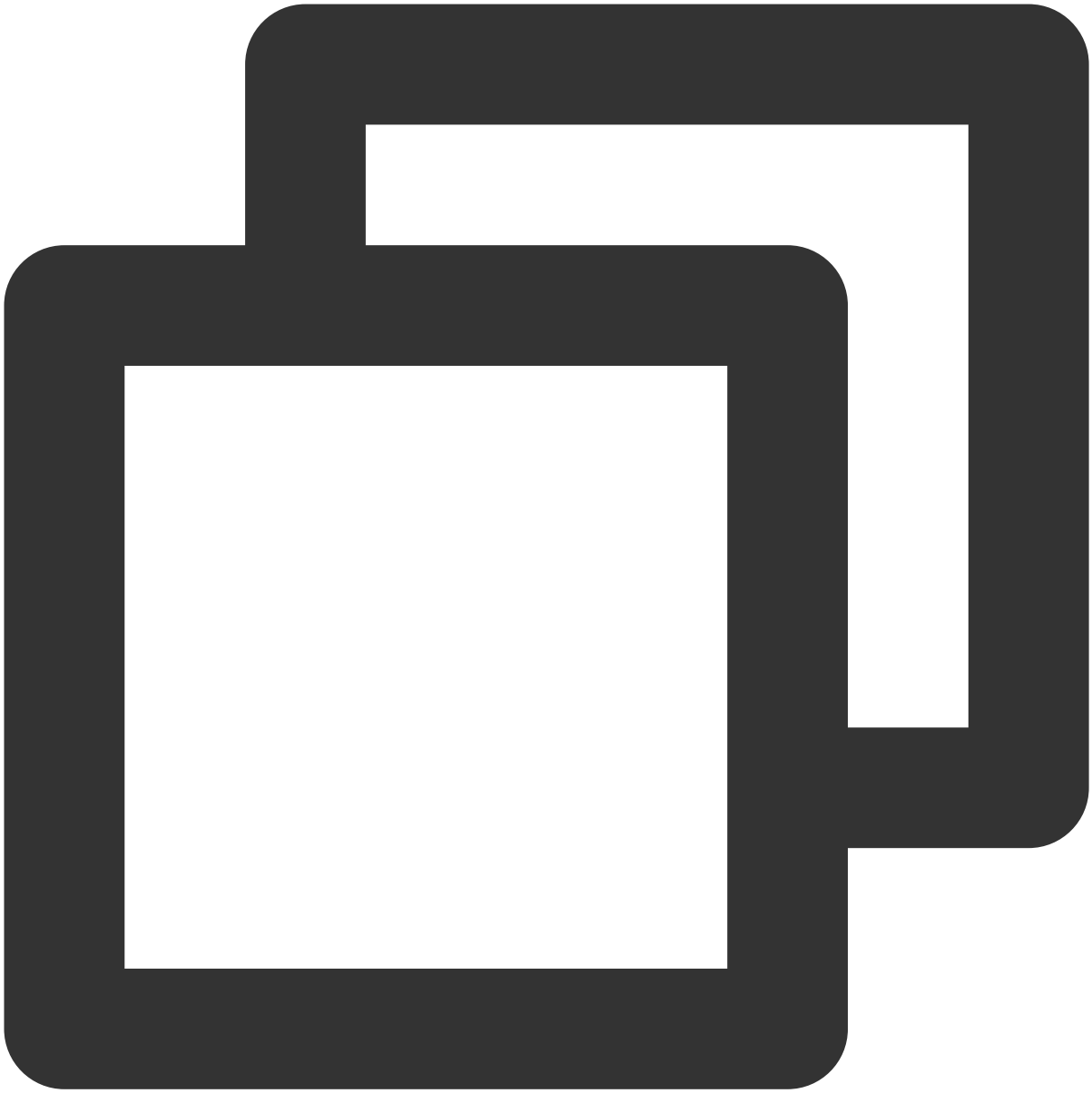
FAQ
공중망 액세스

최종 업데이트 날짜: : 2023-04-28 15:30:11

본문은 딥러닝 컨테이너 이미지 구축 및 TKE Serverless에서 딥러닝 실행 시 발생할 수 있는 질문에 대한 답변을 제공합니다.

컨테이너는 공중망에 어떻게 액세스합니까?

작업 중에 교육 데이터 세트를 다운로드해야 할 수 있으므로 공중망에 대한 액세스가 필요할 수 있습니다. 그러나 초기 상태의 컨테이너는 공중망에 액세스할 수 없으며 데이터 세트 다운로드로 직접 명령을 실행하면 다음 오류가 보고됩니다.



```
W tensorflow/core/platform/cloud/google_auth_provider.cc:184] All attempts to get a
E tensorflow/core/platform/cloud/curl_http_request.cc:614] The transmission  of req
```

상기 문제에 대해 두 가지 공중망 액세스 방법이 제공됩니다.

NAT Gateway 사용: VPC의 많은 Pod가 공중망으로 통신해야 하는 시나리오에 적합합니다. NAT Gateway를 통한 인터넷 액세스를 따라 구성하십시오.

주의사항

생성된 NAT Gateway 및 라우팅 테이블은 TKE Serverless 클러스터와 동일한 리전 및 VPC에 있어야 합니다.

EIP 사용: 하나 또는 몇 개의 Pod가 공중망으로 상호 연결되어야 하는 시나리오에 적합합니다. EIP 사용하여 공중망 액세스에 따라 작업하십시오.

네트워크
DNS
TKE DNS 모범 사례

최종 업데이트 날짜: : 2023-04-28 15:30:11

개요

DNS는 Kubernetes 클러스터에서 서비스 액세스의 첫 번째 단계이므로 안정성과 성능이 매우 중요합니다. DNS를 더 나은 방식으로 구성하고 사용하는 방법에는 여러 측면이 포함됩니다. 본 문서는 DNS의 모범 사례를 소개합니다.

가장 적합한 CoreDNS 버전 선택

다음 표에는 다양한 버전의 TKE 클러스터에 배포된 기본 CoreDNS 버전이 나열되어 있습니다.

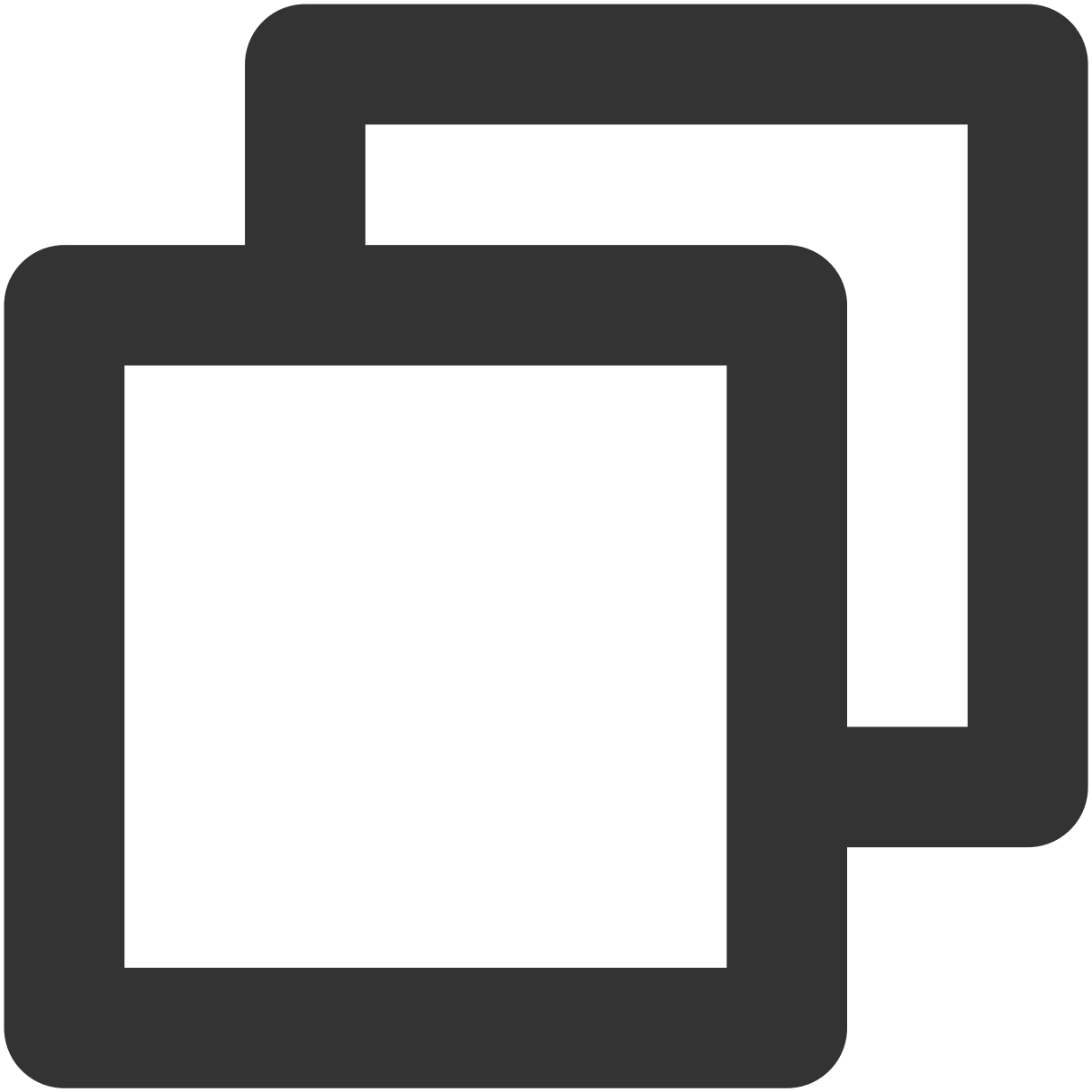
TKE Version	CoreDNS version
v1.22	v1.8.4
v1.20	v1.8.4
v1.18	v1.7.0
v1.16	v1.6.2
v1.14	v1.6.2

기존 이유로 CoreDNS v1.6.2는 v1.18 이상의 클러스터에 계속 배포될 수 있습니다. 현재 CoreDNS 버전이 요구 사항을 충족하지 않는 경우 다음과 같이 수동으로 업그레이드할 수 있습니다.
v1.7.0으로 업그레이드
v1.8.4로 업그레이드

적절한 수의 CoreDNS 복제본 구성

1. TKE의 기본 CoreDNS 복제본 수는 2개이며 podAntiAffinity는 서로 다른 노드에 두 개의 복제본을 배포하도록 구성되었습니다.

2. 클러스터에 80개가 넘는 노드가 있는 경우 TKE 클러스터에서 NodeLocal DNSCache 사용에 따라 NodeLocal DNSCache를 설치하는 것이 좋습니다.
3. 일반적으로 DNS에 대한 비즈니스 액세스의 QPS, 노드 수 또는 총 CPU 코어 수를 기반으로 CoreDNS 복제본 수를 결정할 수 있습니다. NodeLocal DNSCache를 설치한 후 최대 10개의 CoreDNS 복제본을 사용하는 것이 좋습니다. 다음과 같이 복제본 수를 구성할 수 있습니다.
복제본 수 = min (max (ceil (QPS/10000), ceil (클러스터 노드 수/8)), 10)
예시:
클러스터에 10개의 노드가 있고 DNS 서비스 요청의 QPS가 22000인 경우 복제본 수를 3으로 구성합니다
클러스터에 30개의 노드가 있고 DNS 서비스 요청의 QPS가 15000인 경우 복제본 수를 4로 구성합니다
클러스터에 100개의 노드가 있고 DNS 서비스 요청의 QPS가 50000인 경우 복제본 수를 10으로 구성합니다 (NodeLocal DNSCache가 배포됨)
4. 콘솔에 DNSAutoScaler 애드온을 설치하여 CoreDNS 복제본 수를 자동으로 조정할 수 있습니다(원활한 업그레이드는 사전에 구성해야 함). 다음은 기본 구성입니다.



```
data:
  ladder: |-
 {
   "coresToReplicas":
   [
     [ 1, 1 ],
     [ 128, 3 ],
     [ 512,4 ],
   ],
   "nodesToReplicas":
   [
```

```
      [ 1, 1 ],
      [ 2, 2 ]
    ]
  }
```

NodeLocal DNSCache 사용

NodeLocal DNSCache를 TKE 클러스터에 배포하면 서비스 검색 안정성 및 성능을 개선할 수 있습니다. 클러스터 노드에서 DaemonSet으로 DNS 캐시 에이전트를 실행하여 클러스터 DNS 성능을 향상시킵니다.
NodeLocal DNSCache 및 TKE 클러스터에서 NodeLocal DNSCache를 배포하는 방법에 대한 자세한 내용은 TKE 클러스터에서 NodeLocal DNS Cache 사용을 참고하십시오.

CoreDNS 원활한 업그레이드 구성

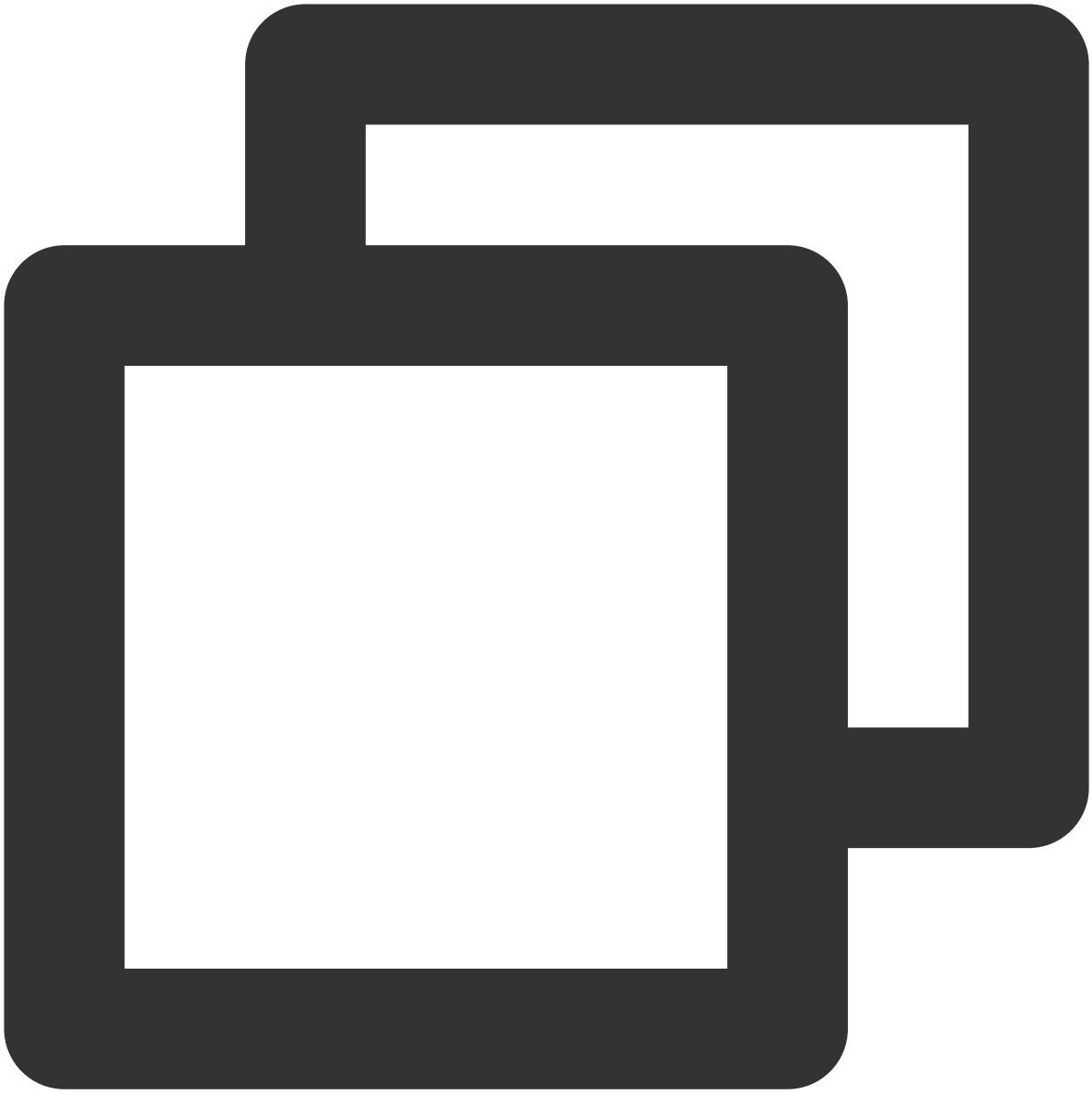
노드를 다시 시작하거나 CoreDNS를 업그레이드하는 동안 일정 기간 동안 일부 CoreDNS 복제본을 사용하지 못할 수 있습니다. 다음 항목을 구성하여 DNS 서비스 가용성을 최대화하고 원활한 업그레이드를 구현할 수 있습니다.

kube-proxy는 iptables 모드, 구성할 필요 없음

iptables 모드에서 kube-proxy는 iptables 규칙을 동기화한 후 남아 있는 conntrack 항목을 즉시 정리합니다. 세션 유지 문제가 없으며 구성이 필요하지 않습니다.

kube-proxy는 IPVS 모드. IPVS UDP 프로토콜의 세션 지속성 제한 시간 구성

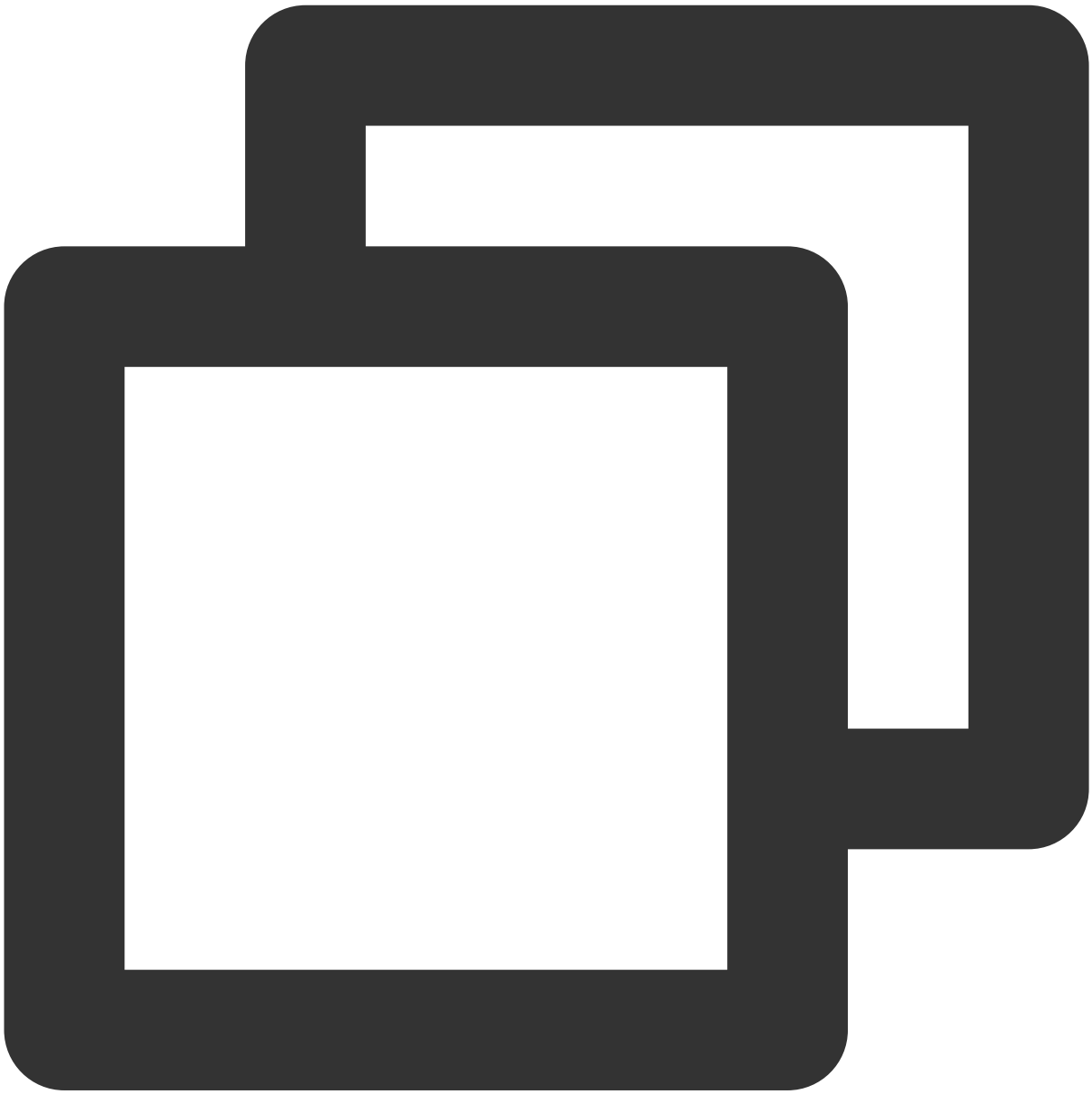
IPVS 모드에서는 비즈니스 자체에 UDP 서비스가 없는 경우 IPVS UDP 프로토콜의 세션 유지 시간 제한을 줄여 서비스 이용 불가 시간을 최소화할 수 있습니다.
1. 클러스터가 v1.18 이상인 경우 kube-proxy는 `--ipvs-udp-timeout` 매개변수에 기본값 0s를 제공하거나 시스템 기본값 300s를 사용할 수 있습니다. `--ipvs-udp-timeout=10s` 를 구성하는 것이 좋습니다. 다음과 같이 kube-proxy DaemonSet를 구성합니다.



```
spec:
  containers:
  - args:
    - --kubeconfig=/var/lib/kube-proxy/config
    - --hostname-override=$(NODE_NAME)
    - --v=2
    - --proxy-mode=ipvs
    - --ipvs-scheduler=rr
    - --nodeport-addresses=$(HOST_IP)/32
    - --ipvs-udp-timeout=10s
    command:
```

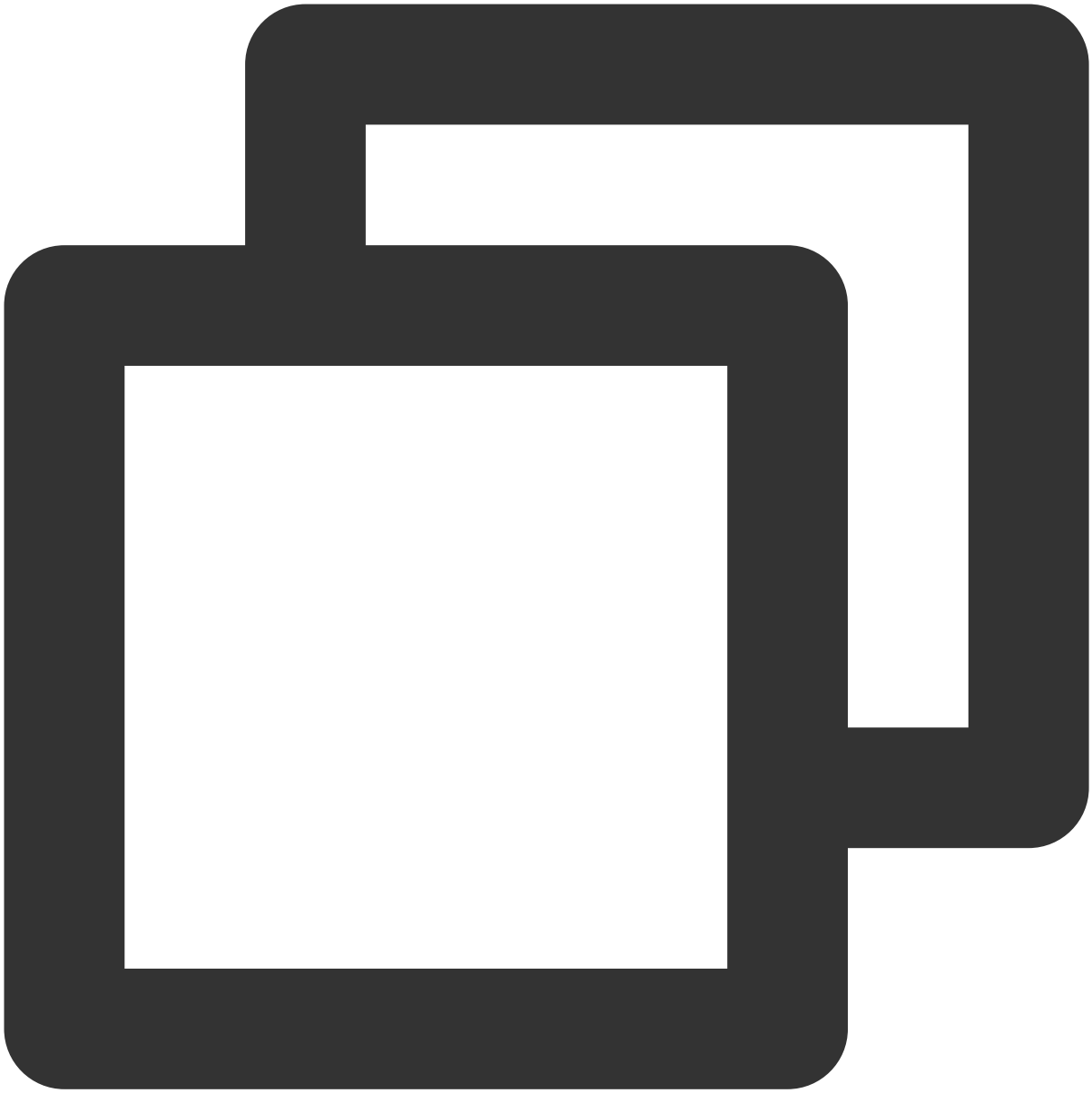
```
      - kube-proxy
      name: kube-proxy
```

2. 클러스터가 v1.16 이하인 경우 kube-proxy는 이 매개변수를 지원하지 않으며 `ipvsadm` 툴을 사용하여 다음과 같이 노드의 정보를 일괄 수정할 수 있습니다.



```
yum install -y ipvsadm
ipvsadm --set 900 120 10
```

3. 구성을 완료한 후 다음과 같이 결과를 확인합니다.



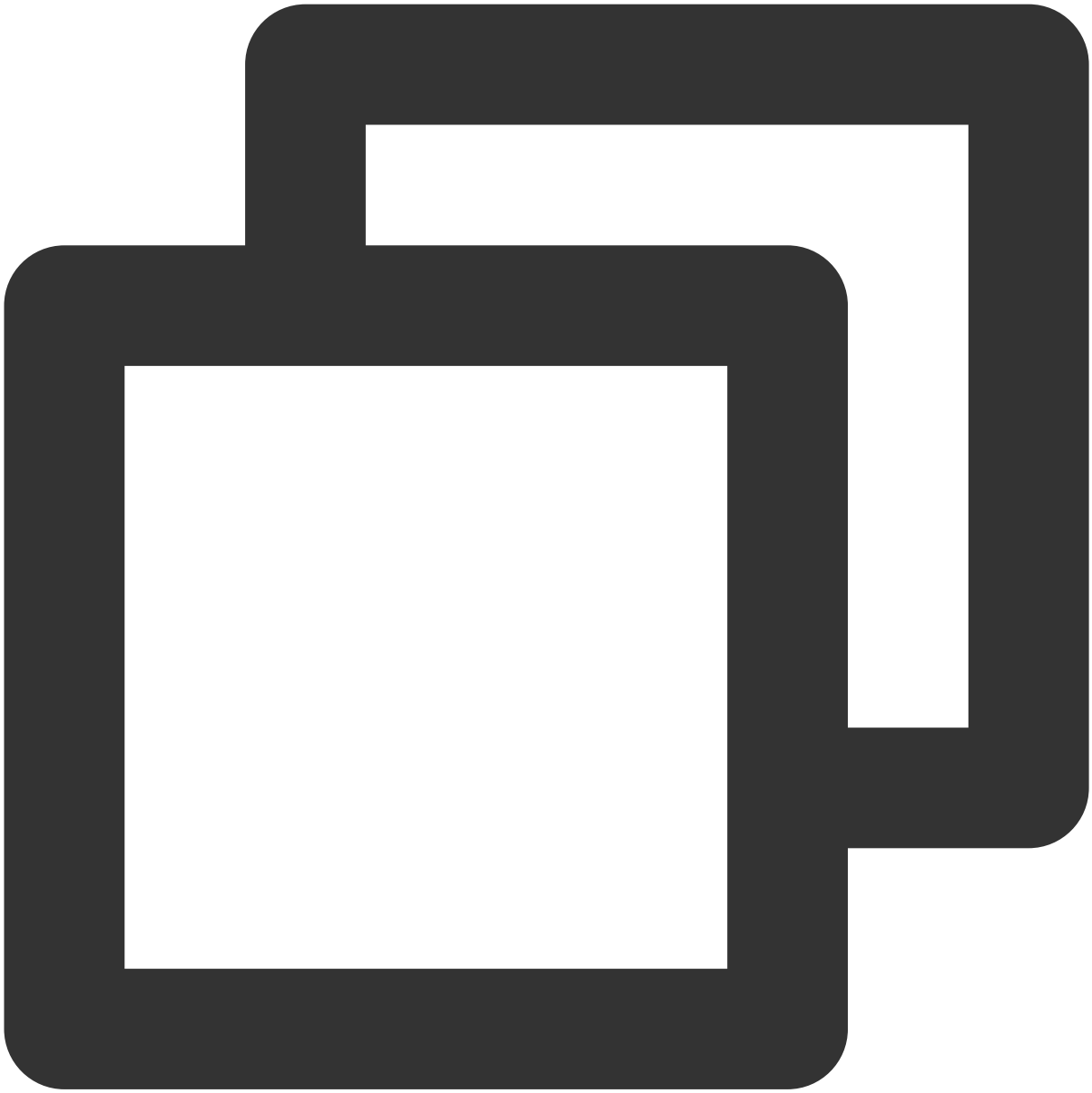
```
ipvsadm -L --timeout
Timeout (tcp tcpfin udp): 900 120 10
```

주의사항

구성을 완료한 후 후속 단계를 진행하기 전에 5min 동안 기다려야 합니다. 귀하의 비즈니스가 UDP 서비스를 사용하는 경우 Submit Ticket하십시오.

CoreDNS에 대한 정상 종료 구성

이미 종료 신호를 받은 복제본이 일정 시간 동안 서비스를 계속 제공하도록 lameduck을 구성할 수 있습니다. 다음과 같이 CoreDNS ConfigMap을 구성합니다(아래는 CoreDNS 1.6.2 구성의 일부일 뿐입니다. 다른 버전의 구성은 CoreDNS 수동 업그레이드를 참고하십시오).

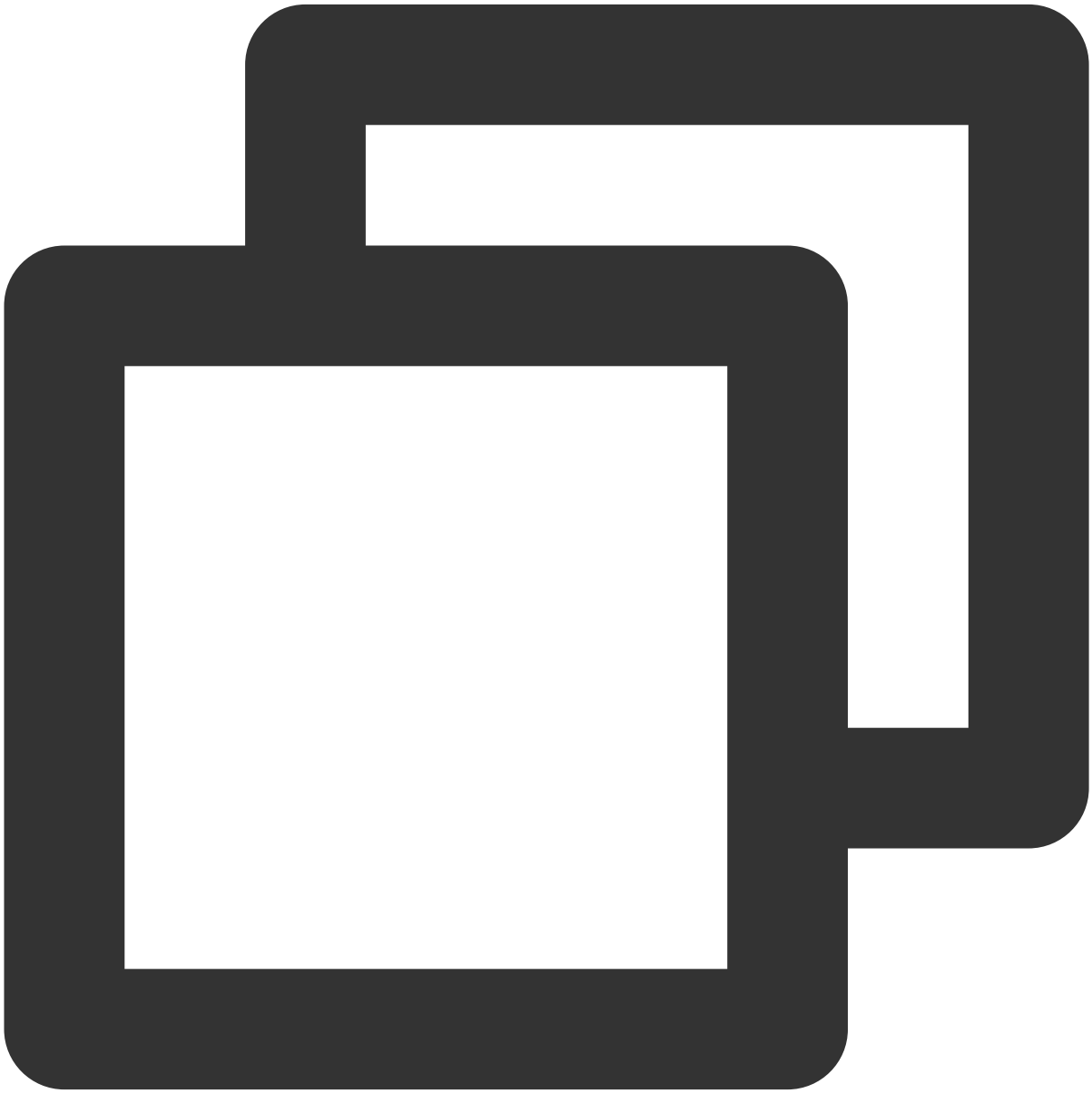


```
        .:53 {
            health {
                lameduck 30s
            }
            kubernetes cluster.local. in-addr.arpa ip6.arpa {
                pods insecure
                upstream
```

```
                fallthrough in-addr.arpa ip6.arpa
            }
        }
```

CoreDNS 서비스 준비 확인 구성

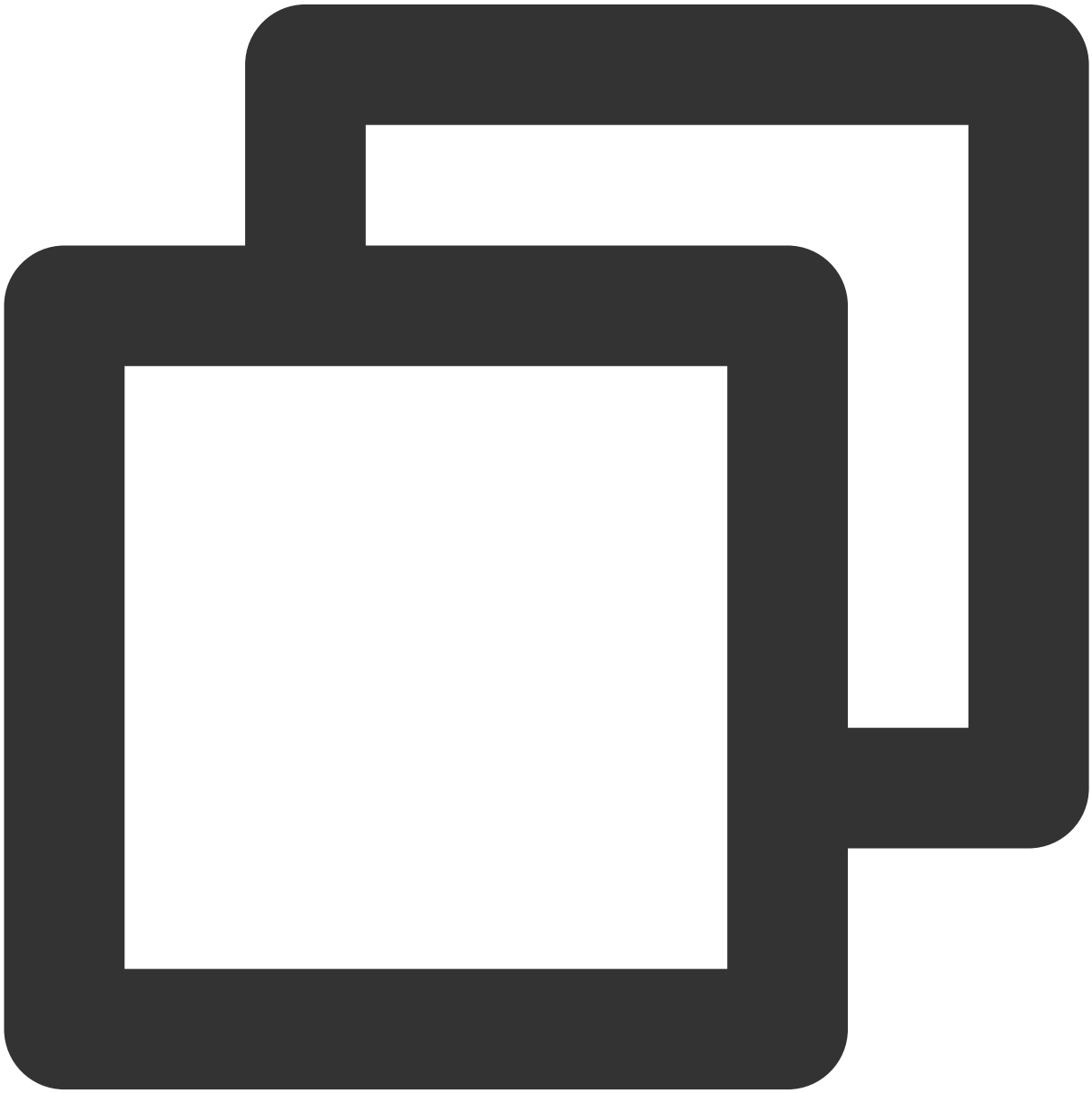
새 복제본이 시작된 후 서비스 준비 상태를 확인하고 DNS 서비스의 백엔드 목록에 추가해야 합니다.
1. ready 플러그인을 열고 CoreDNS configmap을 다음과 같이 구성합니다(아래는 CoreDNS 1.6.2 구성의 일부일 뿐입니다. 다른 버전의 구성은 CoreDNS 수동 업그레이드를 참고하십시오).



```
        .:53 {
```

```
            ready
            kubernetes cluster.local. in-addr.arpa ip6.arpa {
                pods insecure
                upstream
                fallthrough in-addr.arpa ip6.arpa
            }
        }
```

2. CoreDNS에 대한 ReadinessProbe 구성 추가:

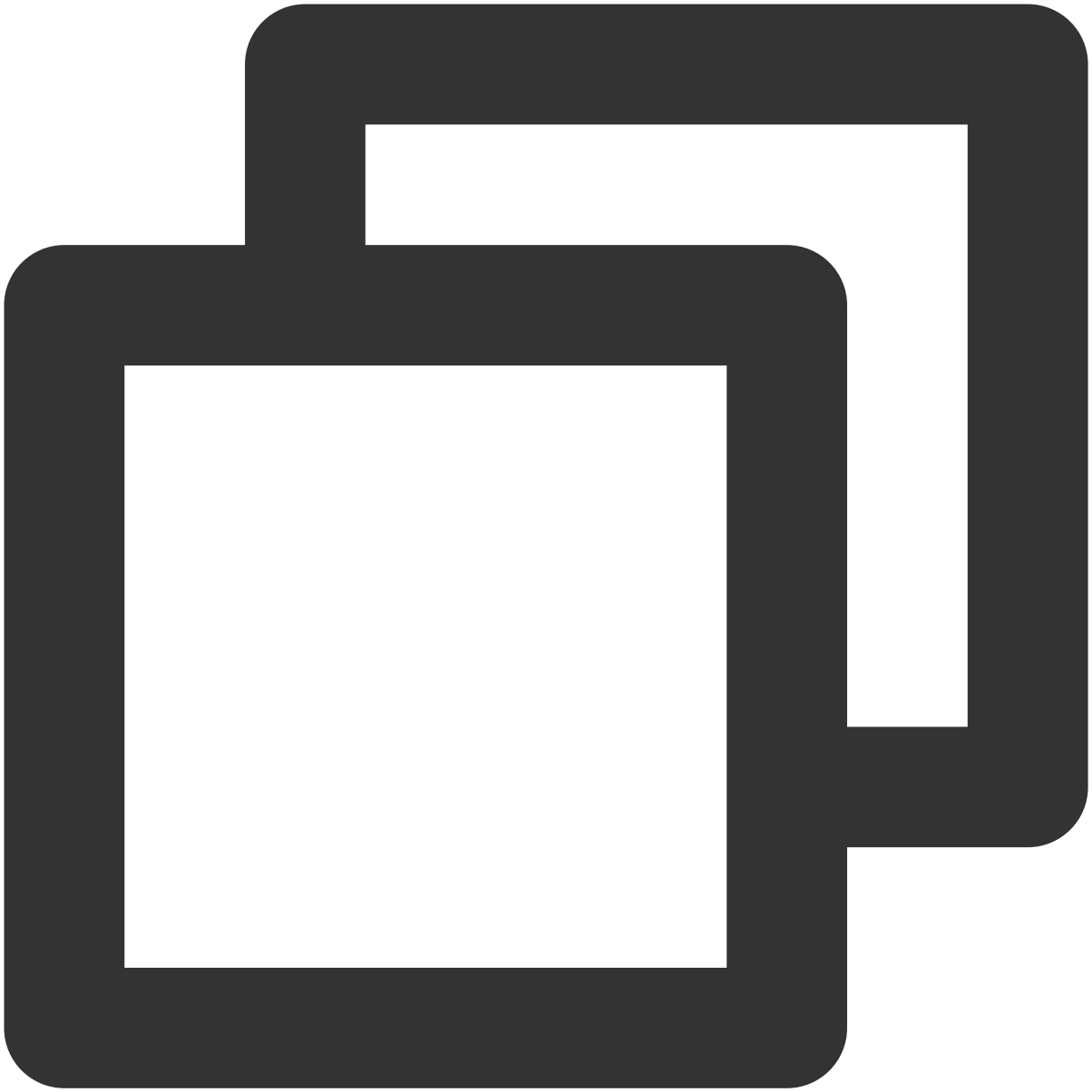


```
readinessProbe:
  failureThreshold: 5
```

```
    httpGet:
      path: /ready
      port: 8181
      scheme: HTTP
    initialDelaySeconds: 30
    periodSeconds: 10
    successThreshold: 1
    timeoutSeconds: 5
```

UDP를 통해 업스트림 DNS에 액세스하도록 CoreDNS 구성

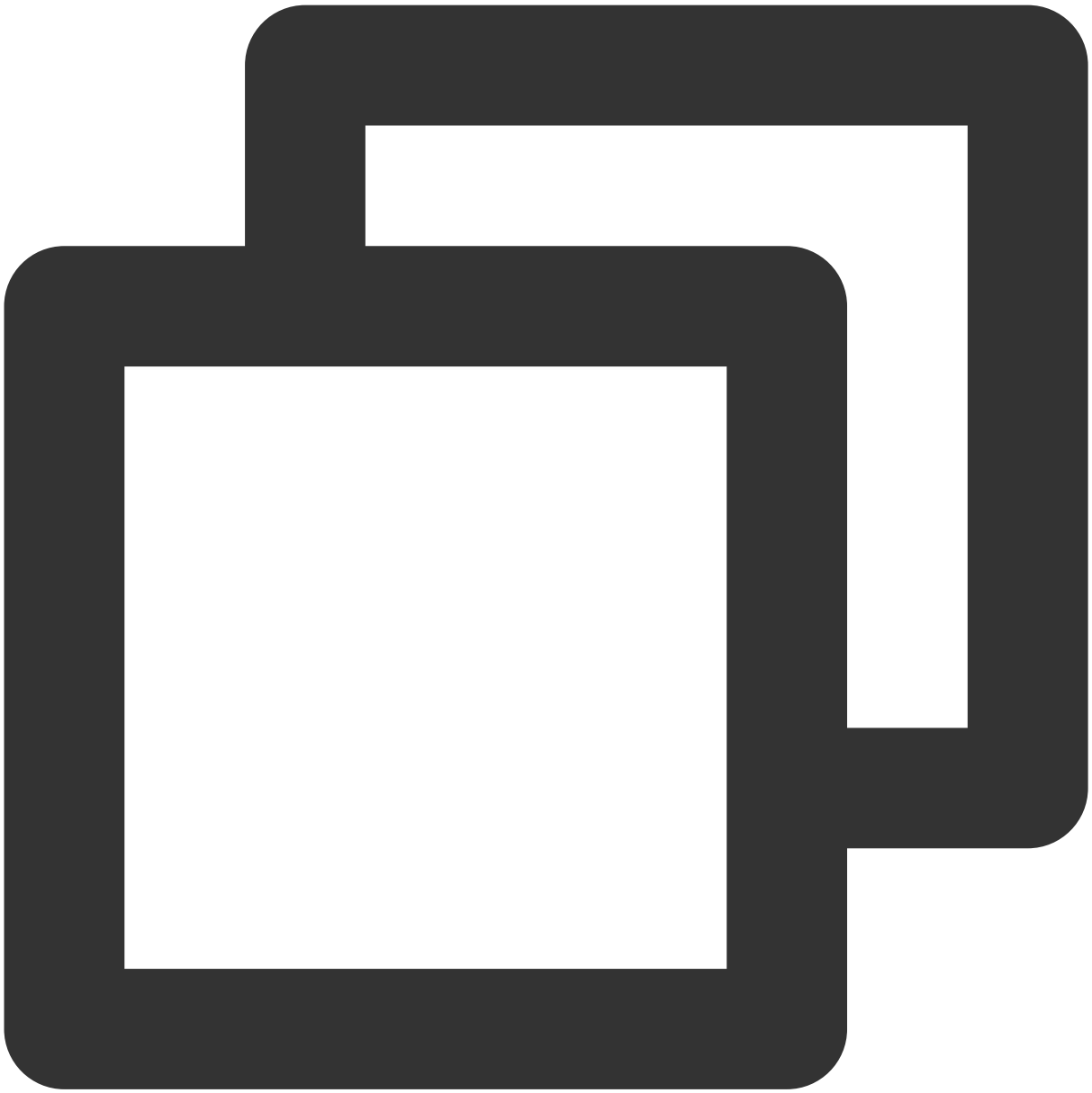
CoreDNS가 DNS Server와 통신해야 하는 경우 기본적으로 클라이언트 요청 프로토콜(UDP 또는 TCP)을 사용합니다. 그러나 TKE에서 CoreDNS의 업스트림 서비스는 기본적으로 TCP에 대한 제한된 지원을 제공하는 VPC의 DNS 서비스입니다. 따라서 다음과 같이 UDP를 사용하여 구성하는 것이 좋습니다(특히 NodeLocal DNSCache가 설치된 경우).



```
.:53 {
    forward . /etc/resolv.conf {
        prefer_udp
    }
}
```

HINFO 요청을 필터링하도록 CoreDNS 구성

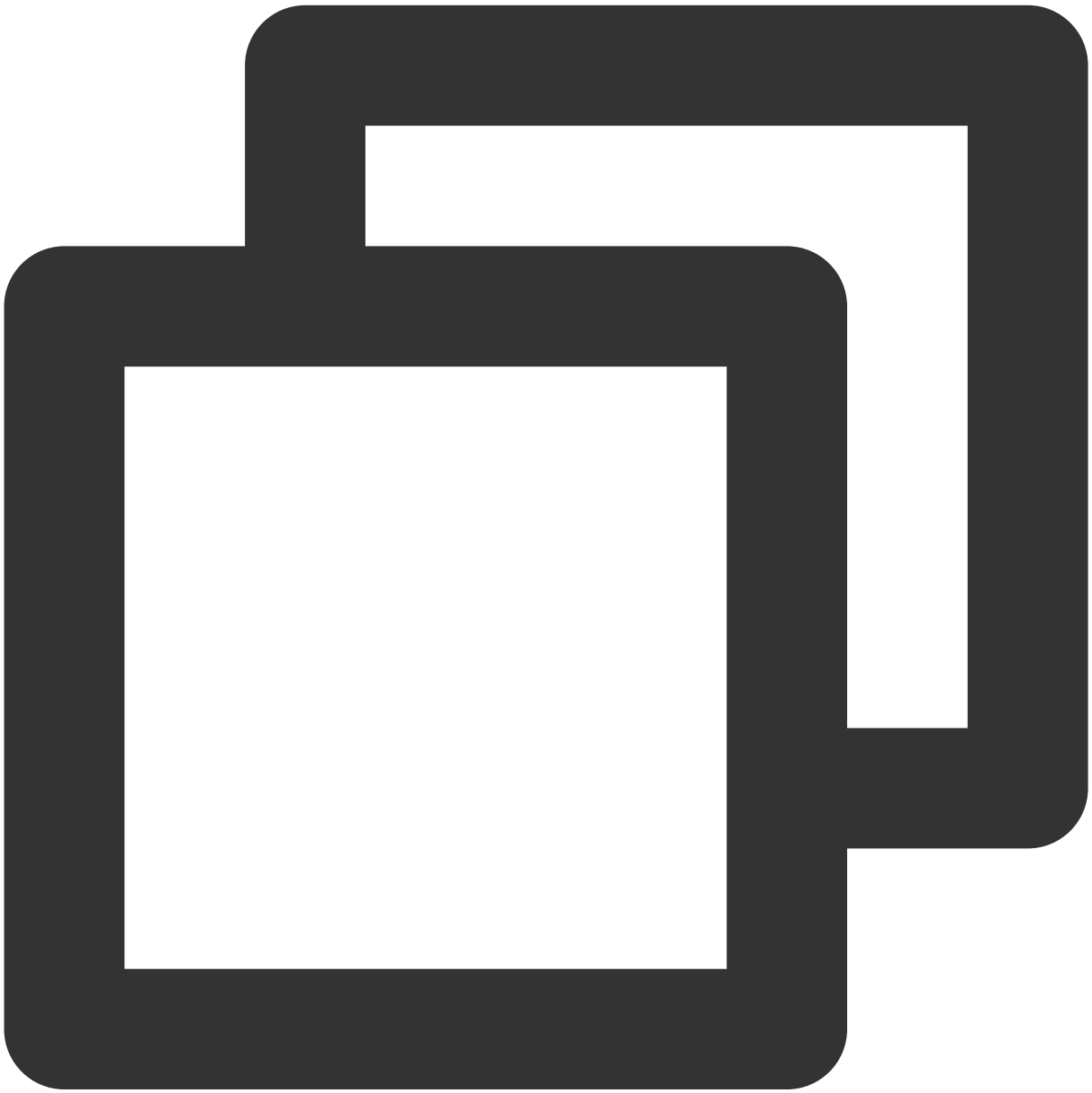
VPC의 DNS 서비스는 HINFO 유형의 DNS 요청을 지원하지 않으므로 CoreDNS 측(특히 NodeLocal DNSCache가 설치된 경우)에서 이러한 요청을 필터링하도록 다음과 같이 구성하는 것이 좋습니다.



```
.:53 {
    template ANY HINFO . {
        rcode NXDOMAIN
    }
}
```

IPv6 AAAA 레코드 쿼리에 대해 도메인 이름이 존재하지 않음을 반환하도록 CoreDNS 구성

비즈니스에서 IPv6 도메인 이름을 확인할 필요가 없는 경우 통신 비용을 줄이기 위해 다음과 같이 구성할 수 있습니다.



```
.:53 {
    template ANY AAAA {
        rcode NXDOMAIN
    }
```

```
        }
```

주의사항

IPv4/IPv6 이중 스택 클러스터에서는 이 구성을 사용하지 마십시오.

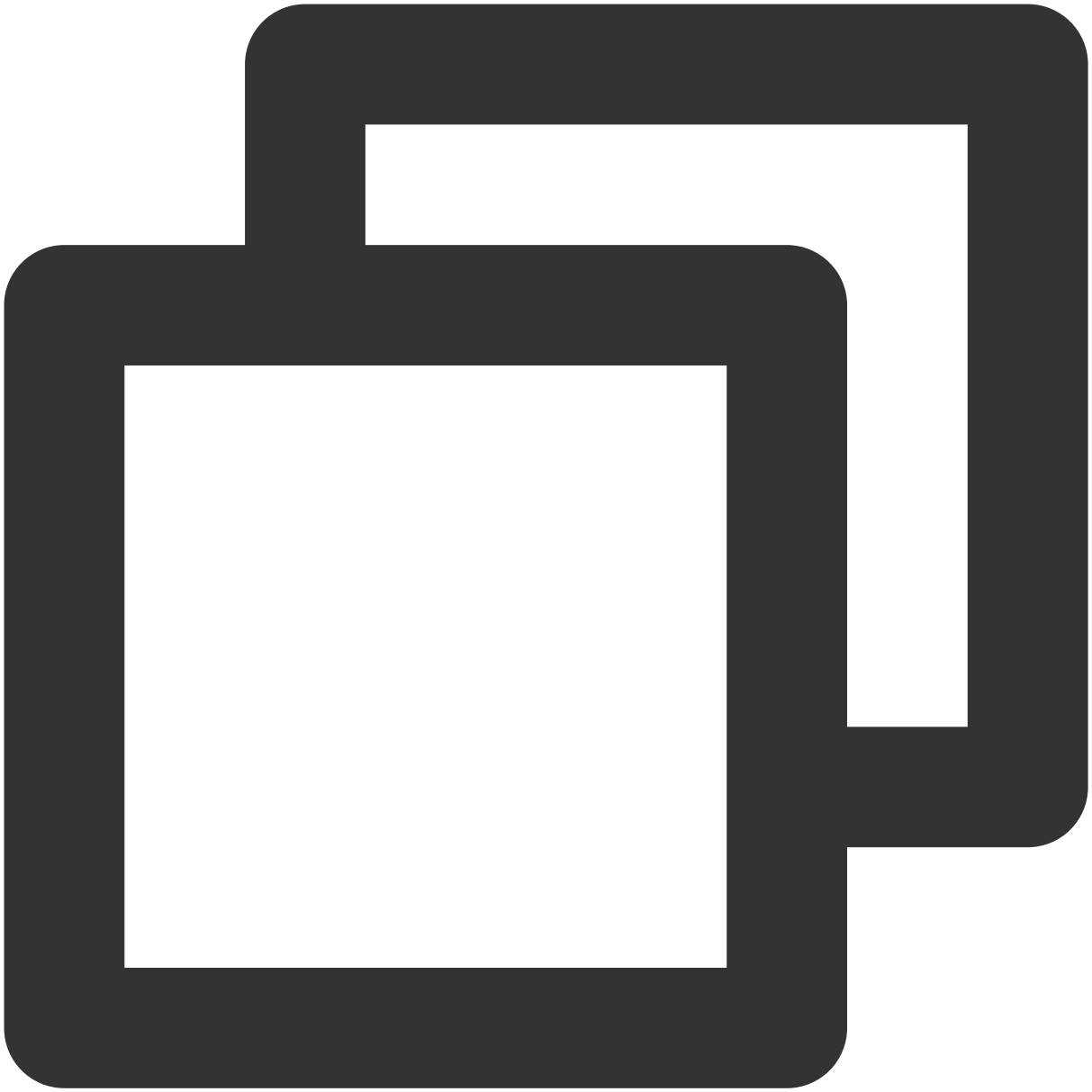
사용자 지정 도메인 이름 확인 구성

자세한 내용은 TKE에서 사용자 지정 도메인 이름 확인 구현을 참고하십시오

수동 업그레이드

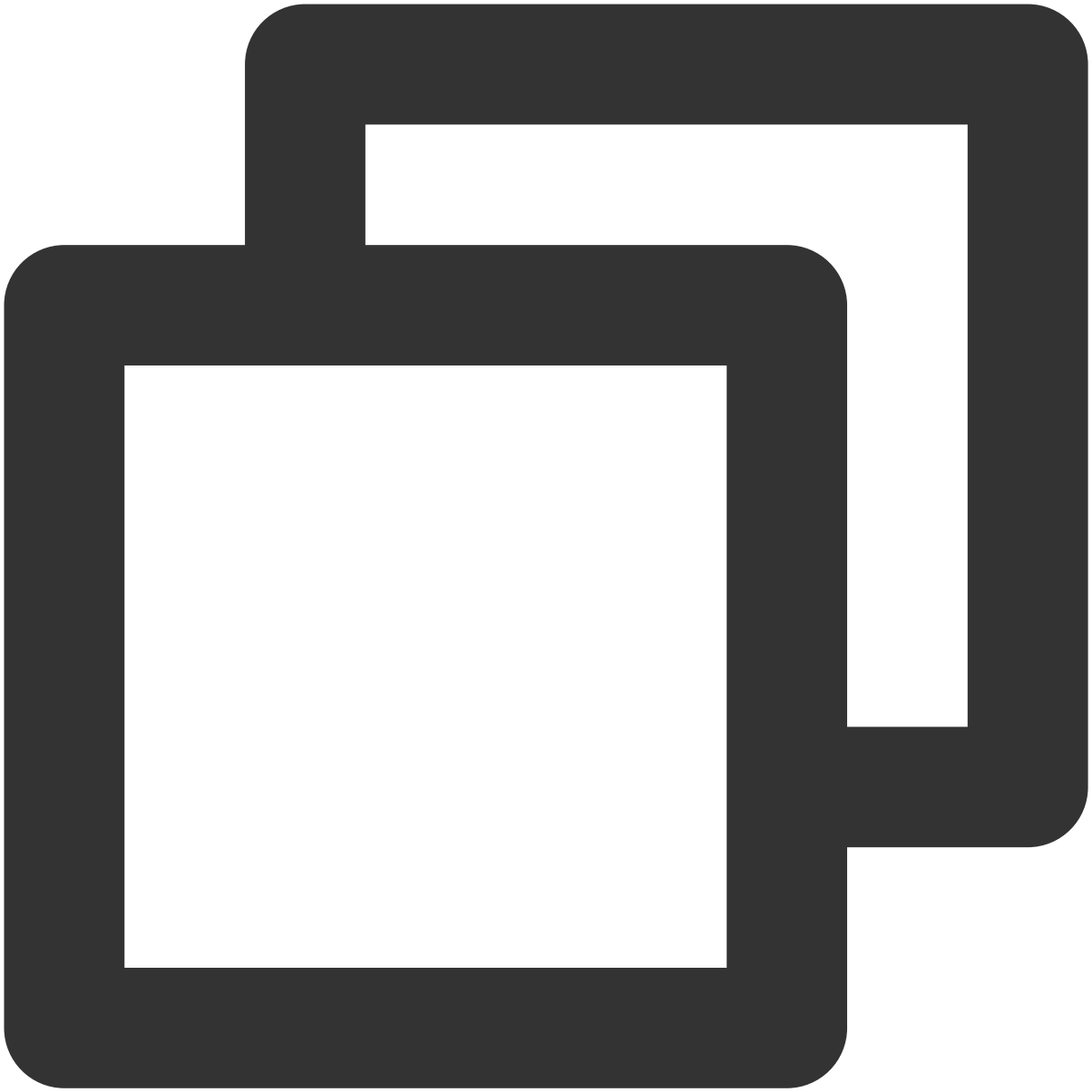
v1.7.0으로 업그레이드

1. coredns configmap 편집



```
kubectl edit cm coredns -n kube-system
```

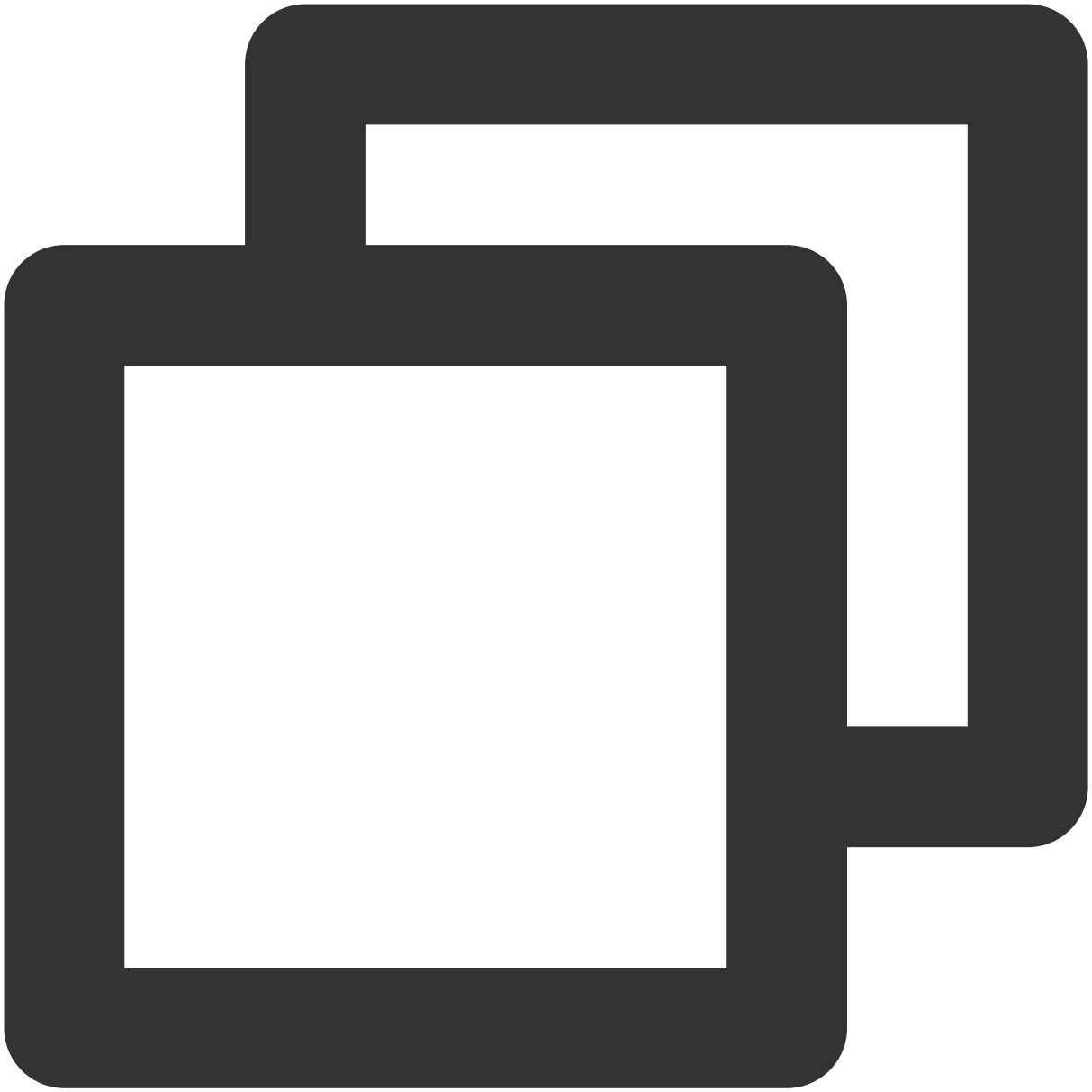
다음과 같이 내용을 수정합니다.



```
.:53 {
    template ANY HINFO . {
        rcode NXDOMAIN
    }
    errors
    health {
        lameduck 30s
    }
    ready
    kubernetes cluster.local. in-addr.arpa ip6.arpa {
        pods insecure
```

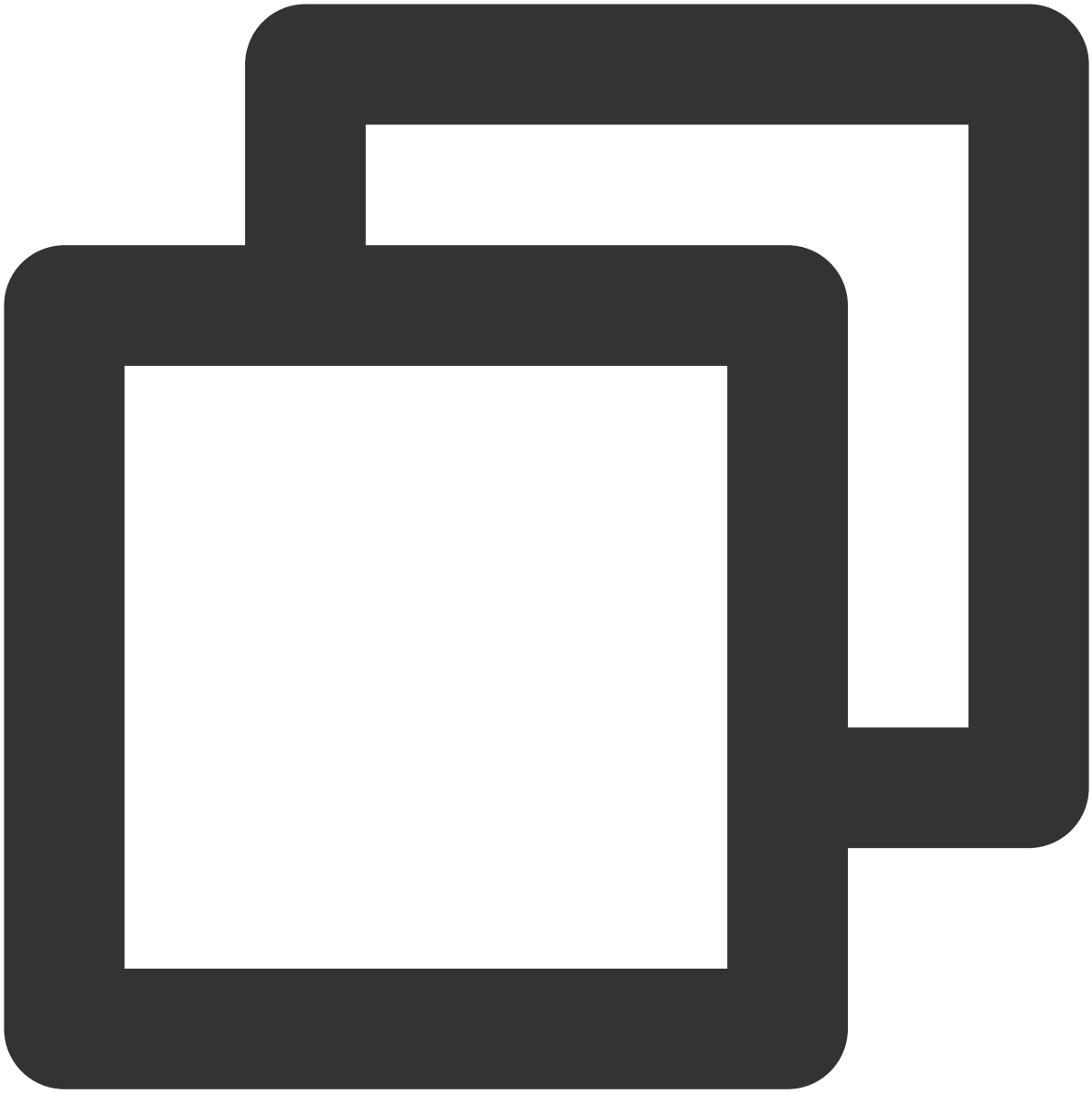
```
            fallthrough in-addr.arpa ip6.arpa
        }
        prometheus :9153
        forward . /etc/resolv.conf {
            prefer_udp
        }
        cache 30
        reload
        loadbalance
    }
```

2. coredns deployment 편집



```
kubectl edit deployment coredns -n kube-system
```

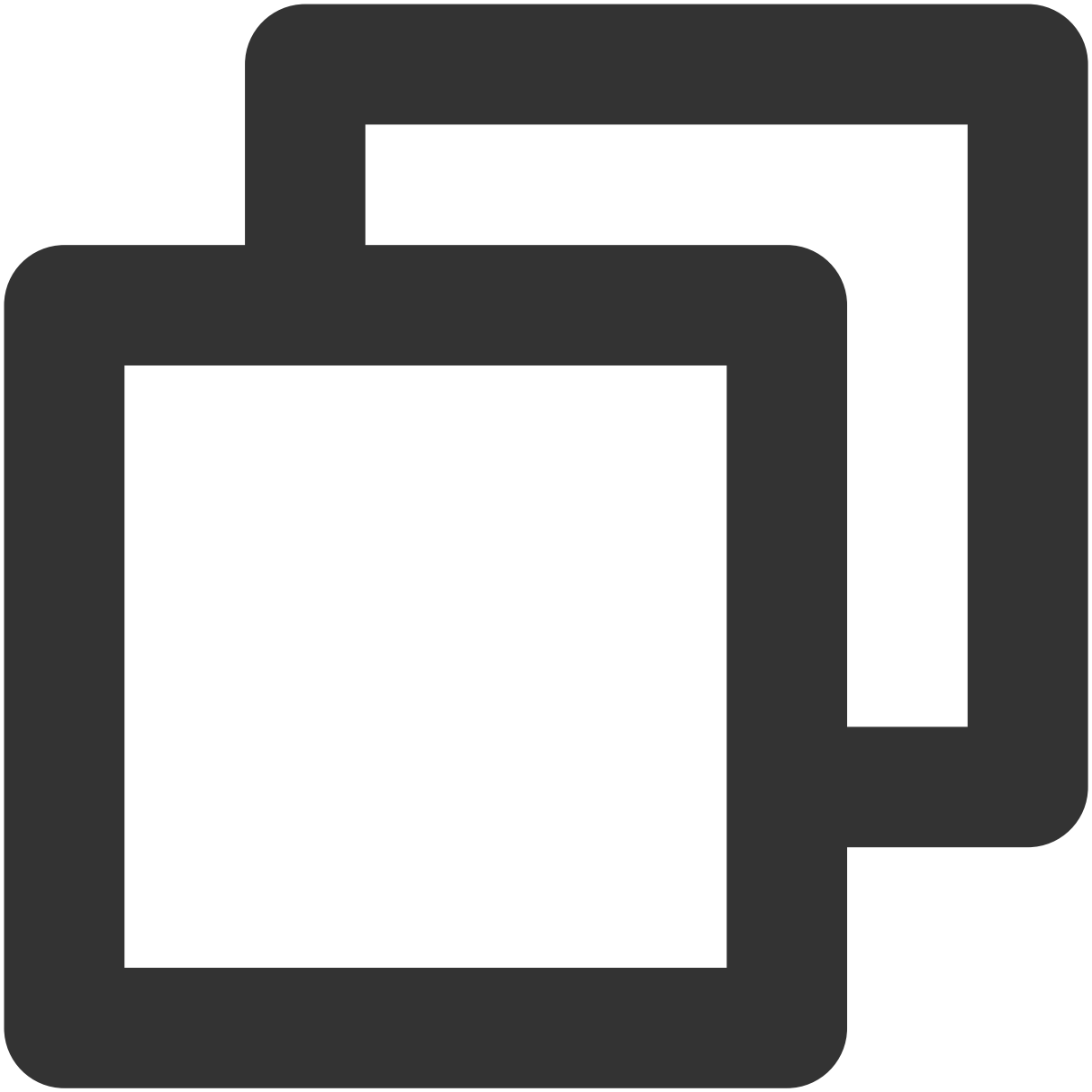
다음과 같이 이미지를 바꿉니다



```
image: ccr.ccs.tencentyun.com/tkeimages/coredns:1.7.0
```

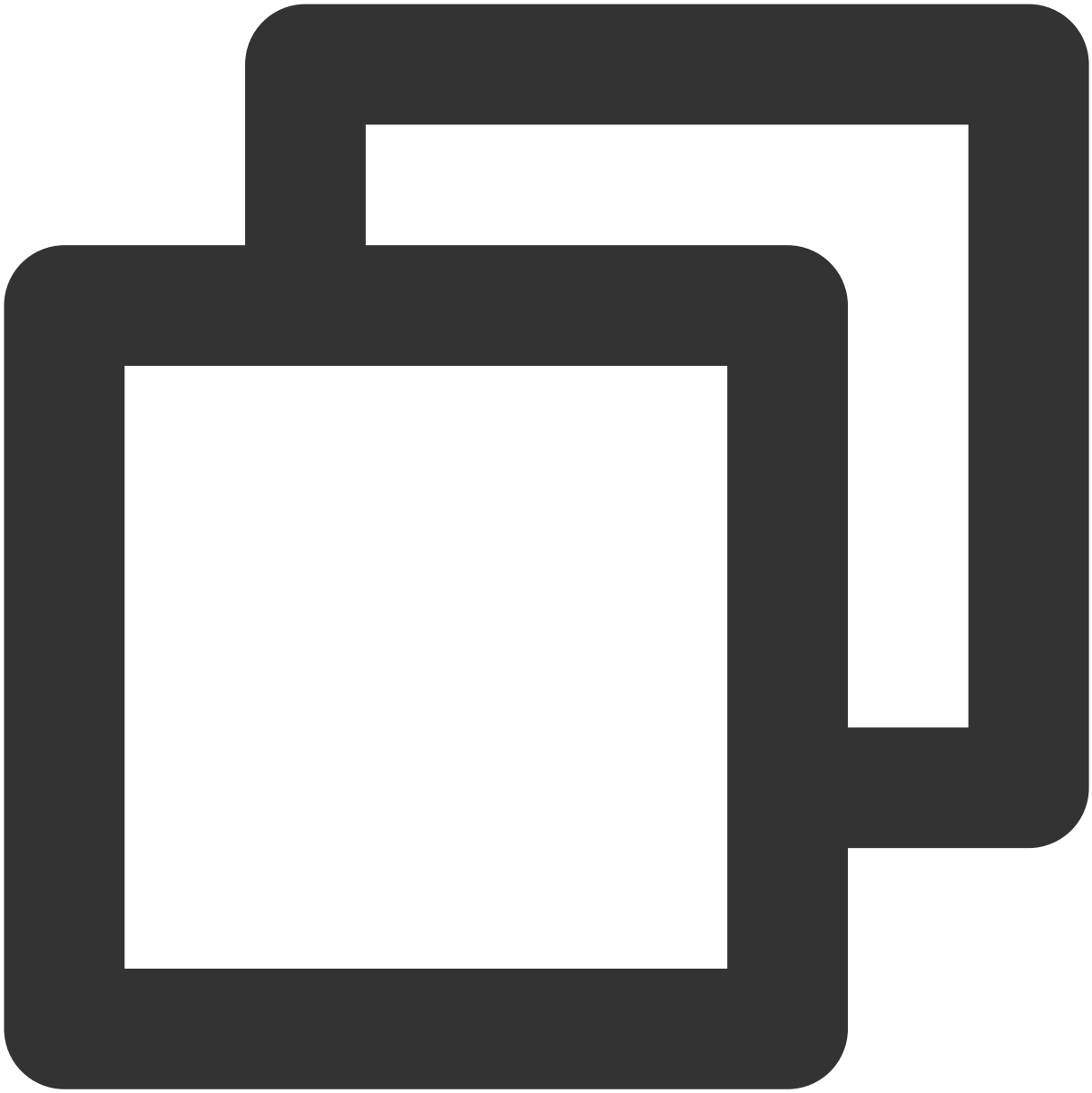
v1.8.4로 업그레이드

1. coredns clusterrole 편집



```
kubectl edit clusterrole system:coredns
```

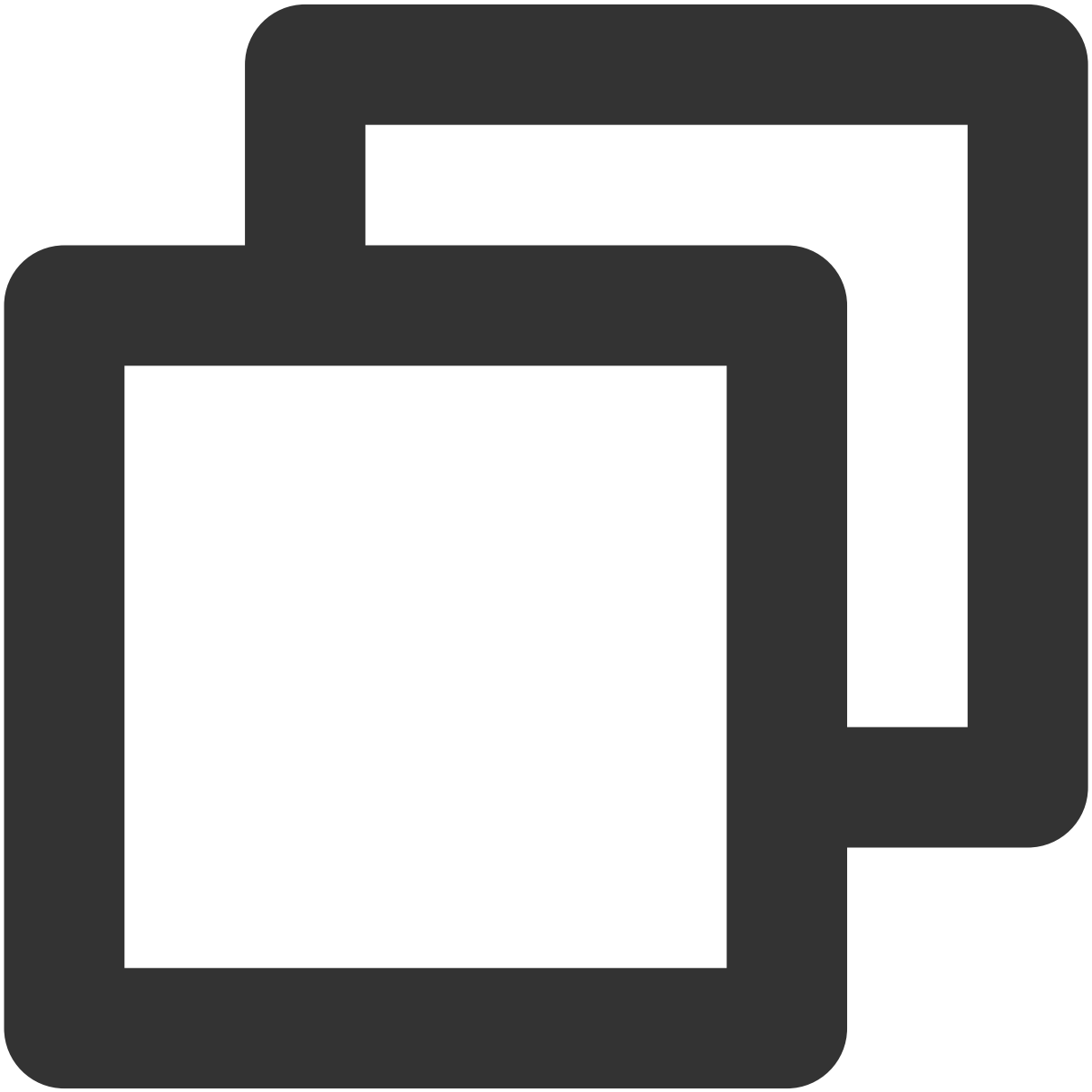
다음과 같이 내용을 수정합니다.



```
rules:
- apiGroups:
  - '*'
  resources:
  - endpoints
  - services
  - pods
  - namespaces
  verbs:
  - list
  - watch
```

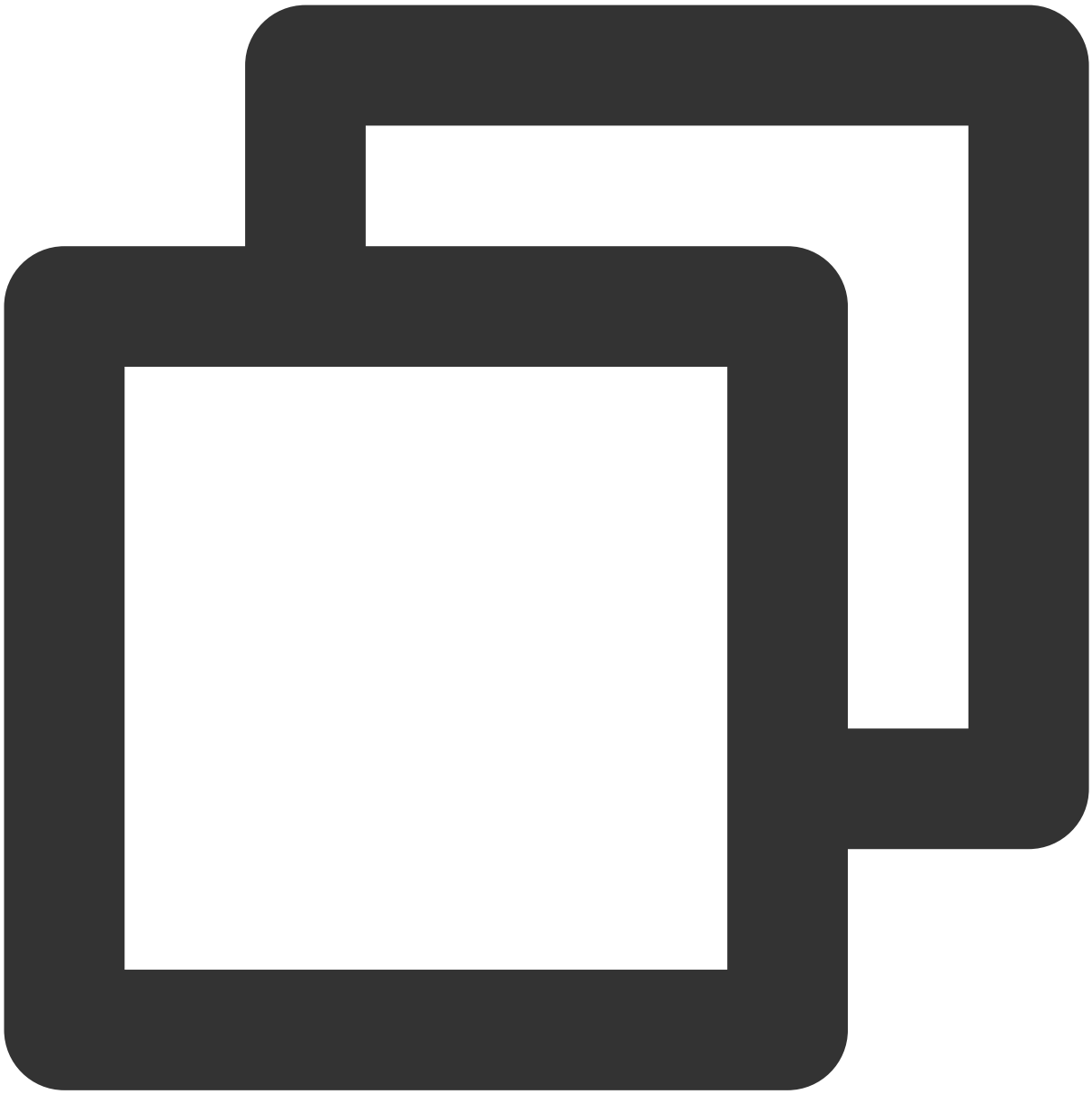
```
- apiGroups:
  - discovery.k8s.io
  resources:
  - endpointslices
  verbs:
  - list
  - watch
```

2. coredns configmap 편집



```
kubectl edit cm coredns -n kube-system
```

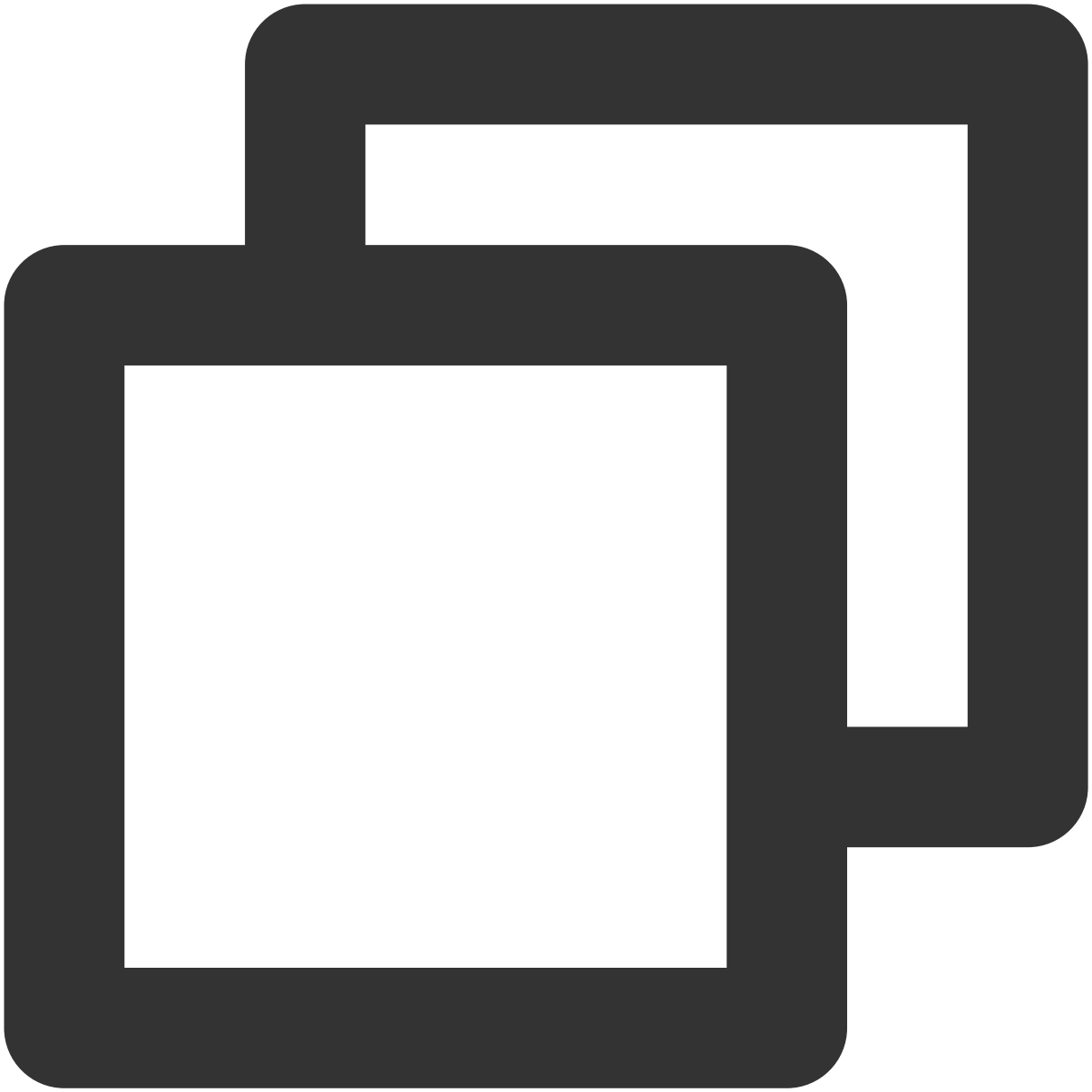
다음과 같이 내용을 수정합니다.



```
.:53 {
    template ANY HINFO . {
        rcode NXDOMAIN
    }
    errors
    health {
        lameduck 30s
    }
    ready
```

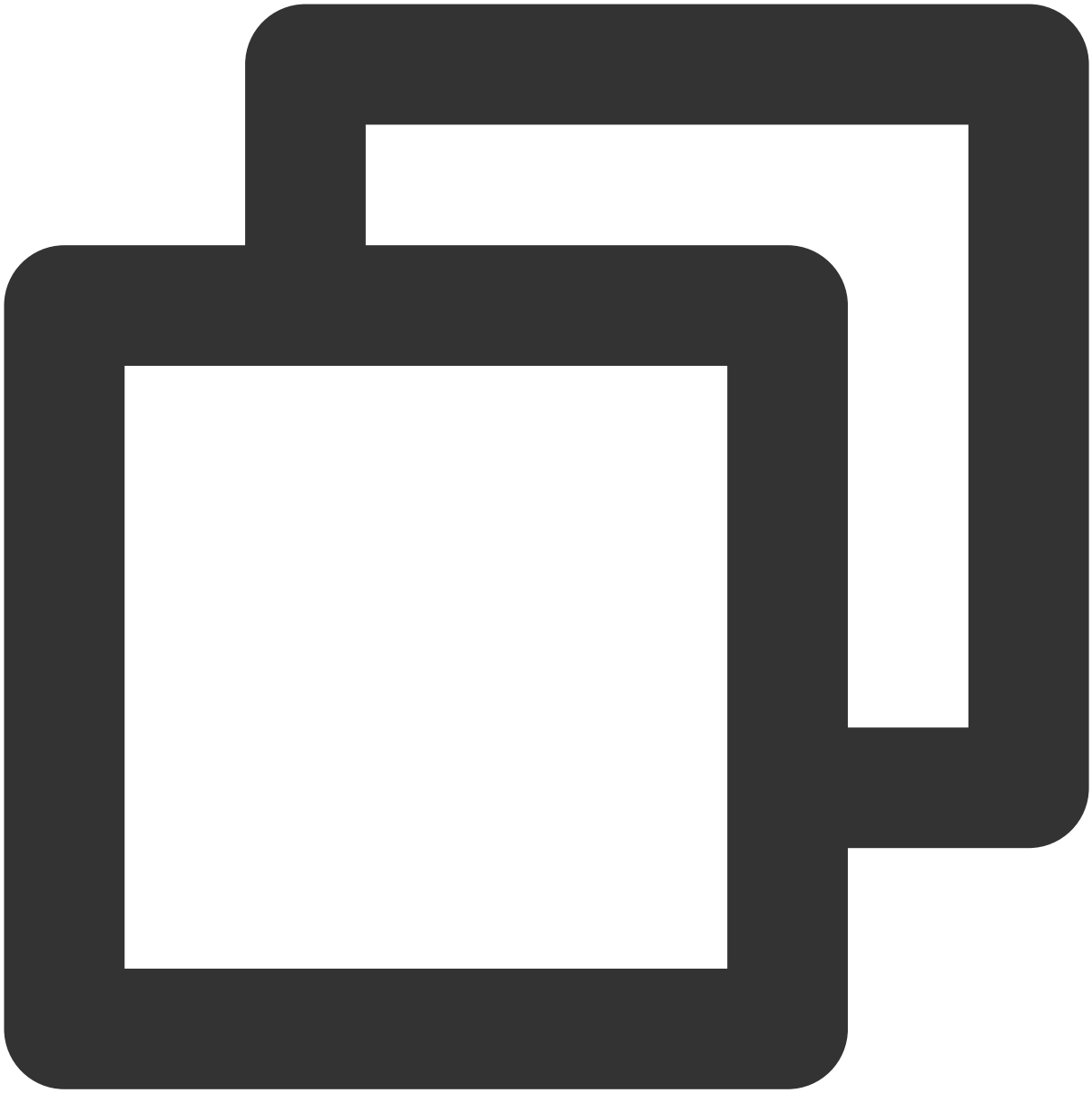
```
        kubernetes cluster.local. in-addr.arpa ip6.arpa {
            pods insecure
            fallthrough in-addr.arpa ip6.arpa
        }
        prometheus :9153
        forward . /etc/resolv.conf {
            prefer_udp
        }
        cache 30
        reload
        loadbalance
    }
```

3. coredns deployment 편집



```
kubectl edit deployment coredns -n kube-system
```

다음과 같이 이미지를 바꿉니다

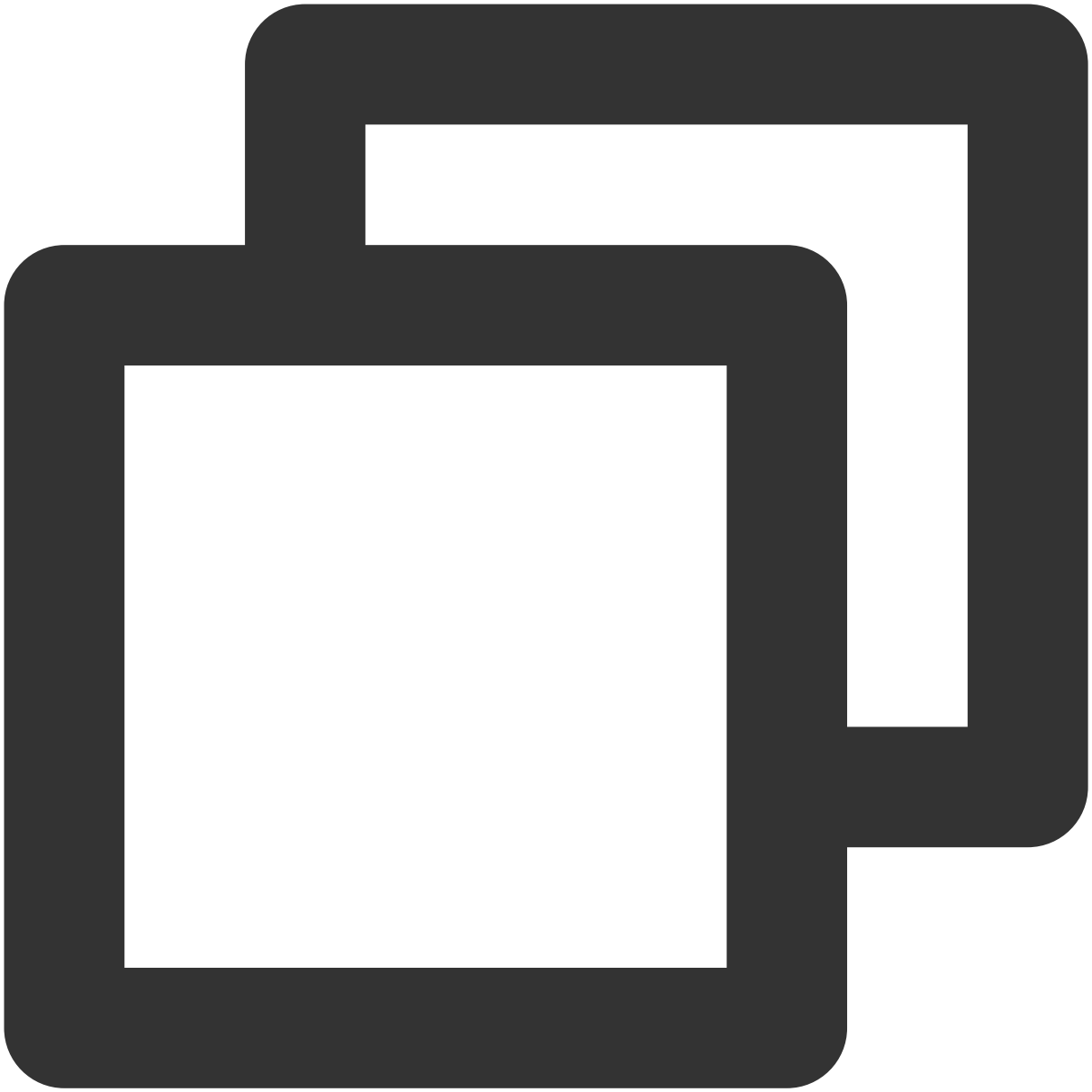


```
image: ccr.ccs.tencentyun.com/tkeimages/coredns:1.8.4
```

비즈니스 구성에 대한 제안

DNS 서비스의 모범 사례 외에도 비즈니스 측면에서 적절한 최적화 구성을 수행하여 DNS 사용자 경험을 개선할 수 있습니다.

1. 기본적으로 Kubernetes 클러스터의 도메인 이름은 일반적으로 여러 확인 요청 후에 확인할 수 있습니다. pod에서 `/etc/resolv.conf` 를 보면 `ndots` 의 기본값이 5임을 알 수 있습니다. 예를 들어, debug 네임스페이스의 `kubernetes.default.svc.cluster.local` service가 쿼리되는 경우:
도메인 이름에 `.` 이 4개 있으므로 시스템에서 쿼리에 `kubernetes.default.svc.cluster.local.debug.svc.cluster.local` 을 사용하기 위해 첫 번째 search를 추가하려고 시도하지만 도메인 이름을 찾을 수 없습니다.
시스템은 쿼리에 `kubernetes.default.svc.cluster.local.svc.cluster.local` 을 계속 사용하지만 여전히 도메인 이름을 찾을 수 없습니다.
시스템은 쿼리에 `kubernetes.default.svc.cluster.local.cluster.local` l을 계속 사용하지만 여전히 도메인 이름을 찾을 수 없습니다.
시스템은 확장을 추가하지 않고 `kubernetes.default.svc.cluster.local` 을 사용하려고 시도합니다. 쿼리가 성공하고 응답하는 ClusterIP가 반환됩니다.
2. 상기 간단한 service 도메인 이름은 네 가지 확인 후 성공적으로 레졸루션할 수 있으며 클러스터에는 쓸모없는 DNS 요청이 많이 있습니다. 따라서 쿼리 수를 줄이기 위해 비즈니스에 대해 구성된 액세스 유형을 기반으로 적절한 ndots 값을 설정해야 합니다.



```
spec:
  dnsConfig:
    options:
    - name: ndots
      value: "2"
  containers:
  - image: nginx
    imagePullPolicy: IfNotPresent
    name: diagnosis
```

3. 또한 비즈니스에서 서비스에 액세스할 수 있도록 도메인 이름 구성을 최적화할 수 있습니다.

Pod는 `<service-name>` 을 통해 현재 네임스페이스의 Service에 액세스해야 합니다.
Pod는 `<service-name>.<namespace-name>` 을 통해 다른 네임스페이스의 Service에 액세스해야 합니다.
Pod는 쓸데없는 검색을 줄이기 위해 끝에 `.` 이 추가된 FQDN(정규화된 도메인 이름)을 통해 외부 도메인 이름에 액세스해야 합니다.

관련 내용

구성 설명

errors

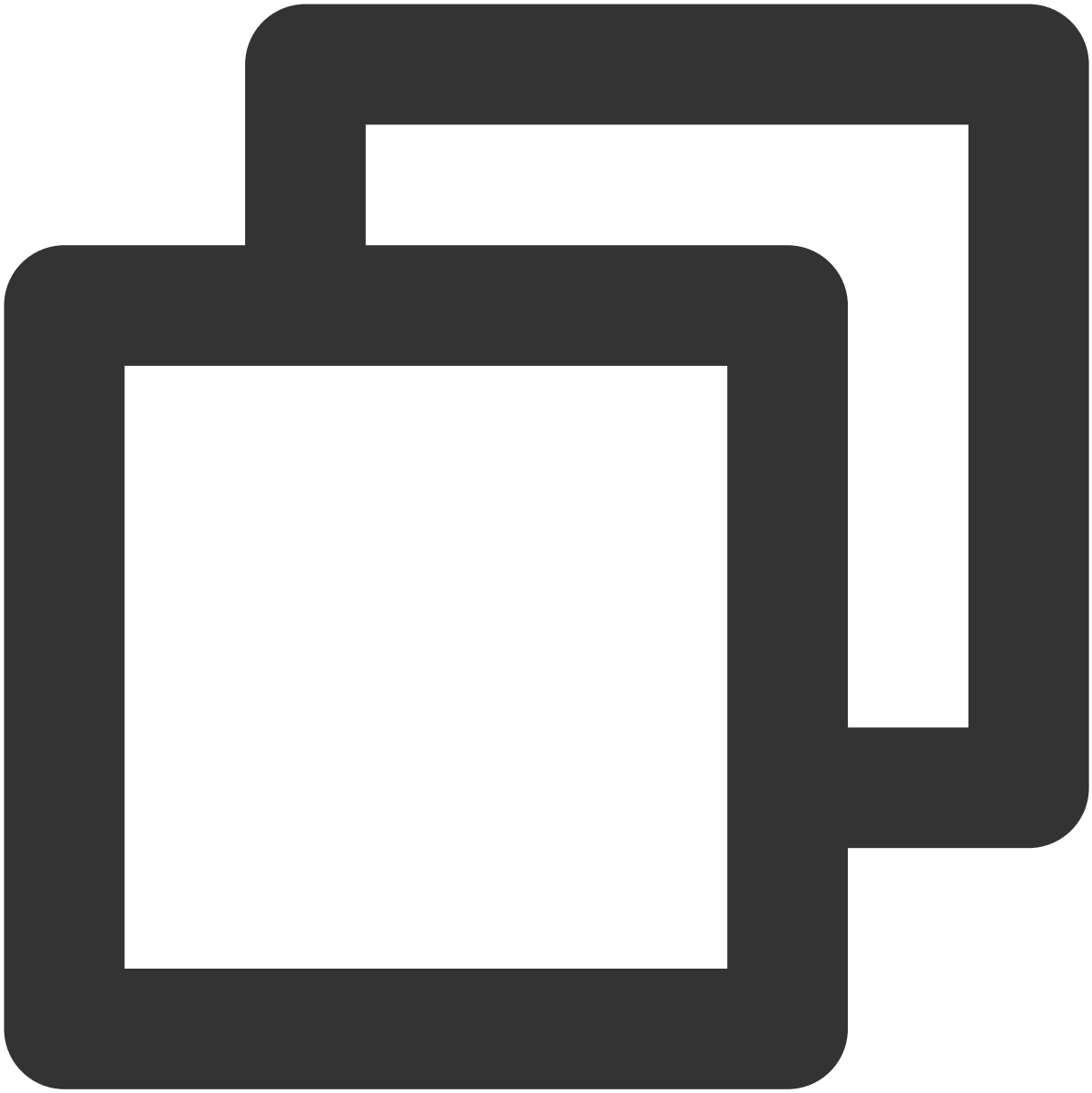
오류 메시지를 출력합니다.

health

상태를 리포트하고 `livenessProbe` 와 같은 상태 확인 구성에 사용됩니다. 기본적으로 포트 8080에서 수신하고 `http://localhost:8080/health` 경로를 사용합니다.

주의사항

여러 Server 블록이 있는 경우 health를 한 번만 구성하거나 다른 포트에 대해 구성할 수 있습니다.



```
com {
  whoami
  health :8080
}

net {
  erratic
  health :8081
}
```

lameduck

정상적인 종료 기간을 구성하는 데 사용됩니다. hook는 CoreDNS가 종료 신호를 수신할 때 sleep 모드를 실행하여 서비스가 특정 시간 동안 계속 실행될 수 있도록 하는 구현 방법입니다.

ready

플러그인 상태를 리포트하고 `readinessProbe` 와 같은 서비스 준비 확인 구성에 사용됩니다. 기본적으로 포트 8181에서 수신하고 `http://localhost:8181/ready` 경로를 사용합니다.

kubernetes

클러스터에서 서비스를 레졸루션할 수 있는 Kubernetes 플러그인입니다.

prometheus

모니터링 데이터를 가져오는 데 사용되는 metrics 데이터 API입니다. 해당 경로는 `http://localhost:9153/metrics` 입니다.

forward(proxy)

처리되지 않은 요청을 업스트림 DNS 서버로 포워딩하고 기본적으로 호스트의 `/etc/resolv.conf` 구성을 사용합니다.

forward aaa bbb의 구성에 따라 업스트림 udns 서버 목록 [aaa,bbb]는 내부적으로 유지됩니다.

요청이 도착하면 사전 설정된 정책(random|round_robin|sequential, 여기서 random이 기본 정책)에 따라 요청을 포워딩하기 위해 [aaa,bbb] 목록에서 업스트림 udns 서버가 선택됩니다. 포워딩에 실패하면 포워딩을 위해 다른 서버가 선택되고 정상적인 상태가 될 때까지 실패한 서버에 대해 정기적인 상태 확인이 수행됩니다.

서버가 연속해서 여러 번(기본적으로 두 번) 상태 검사에 실패하면 해당 udns 상태가 down으로 설정되고 후속 서버 선택에서 down 상태인 udns는 건너뜁니다.

모든 udns가 down된 경우 시스템은 포워딩할 udns를 무작위로 선택합니다.

따라서 coredns는 여러 upstream 서버 간에 지능적으로 전환할 수 있습니다. forward 목록에 사용 가능한 udns 서버가 있는 한 요청이 성공할 수 있습니다.

cache

DNS 캐시입니다.

reload

Corefile을 핫로딩합니다. ConfigMap이 수정된 후 2분 안에 새 구성을 다시 로딩합니다.

loadbalance

응답의 레코드 순서를 무작위로 지정하여 DNS 기반의 로드 밸런싱 기능을 제공합니다.

CoreDNS의 리소스 사용량

메모리

CPU

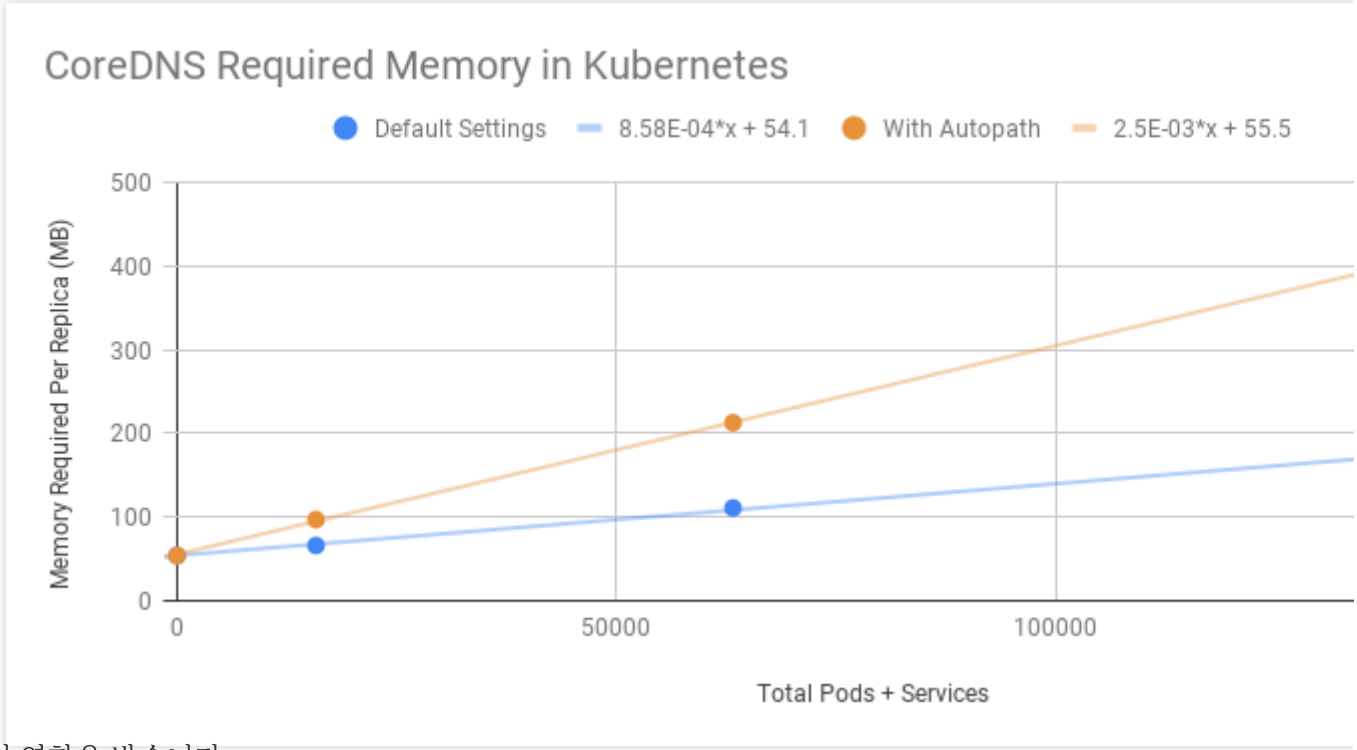
클러스터의 Pod 및 Service 수에 따라 달라집니다.

활성화된 캐시 크기의 영향을 받습니다.

QPS의 영향을 받습니다.

다음은 CoreDNS의 공식 데이터입니다.

MB required (default settings) = (Pods + Services) / 1000 + 54



QPS의 영향을 받습니다.

다음은 CoreDNS의 공식 데이터입니다.

단일 복제 CoreDNS, 실행 중인 노드 사양: 2 vCPUs, 7.5 GB memory

Query Type	QPS	Avg Latency (ms)	Memory Delta (MB)
external	6733	12.02	+5
internal	33669	2.608	+5

TKE에서 사용자 지정 도메인 이름 레졸루션 구현

최종 업데이트 날짜: : 2023-04-26 19:23:11

작업 시나리오

TKE 또는 Serverless TKE를 사용하는 경우 다음 시나리오에서 사용자 지정 내부 도메인 이름을 레졸루션해야 할 수 있습니다.
외부 중앙 집중식 스토리지 서비스를 구축하고 클러스터의 모니터링 또는 로그 수집 데이터를 고정된 내부 도메인 이름을 통해 외부 스토리지 서비스로 보내야 합니다.
기존 서비스의 컨테이너화 중에 일부 서비스의 코드는 고정된 도메인 이름으로 다른 내부 서비스를 호출하도록 구성되며 구성을 수정할 수 없습니다. 즉, Kubernetes의 Service 이름을 호출에 사용할 수 없습니다.

솔루션

이 문서에서는 클러스터에서 사용자 지정 도메인 이름 레졸루션을 사용하기 위한 다음 세 가지 솔루션에 대해 설명합니다.

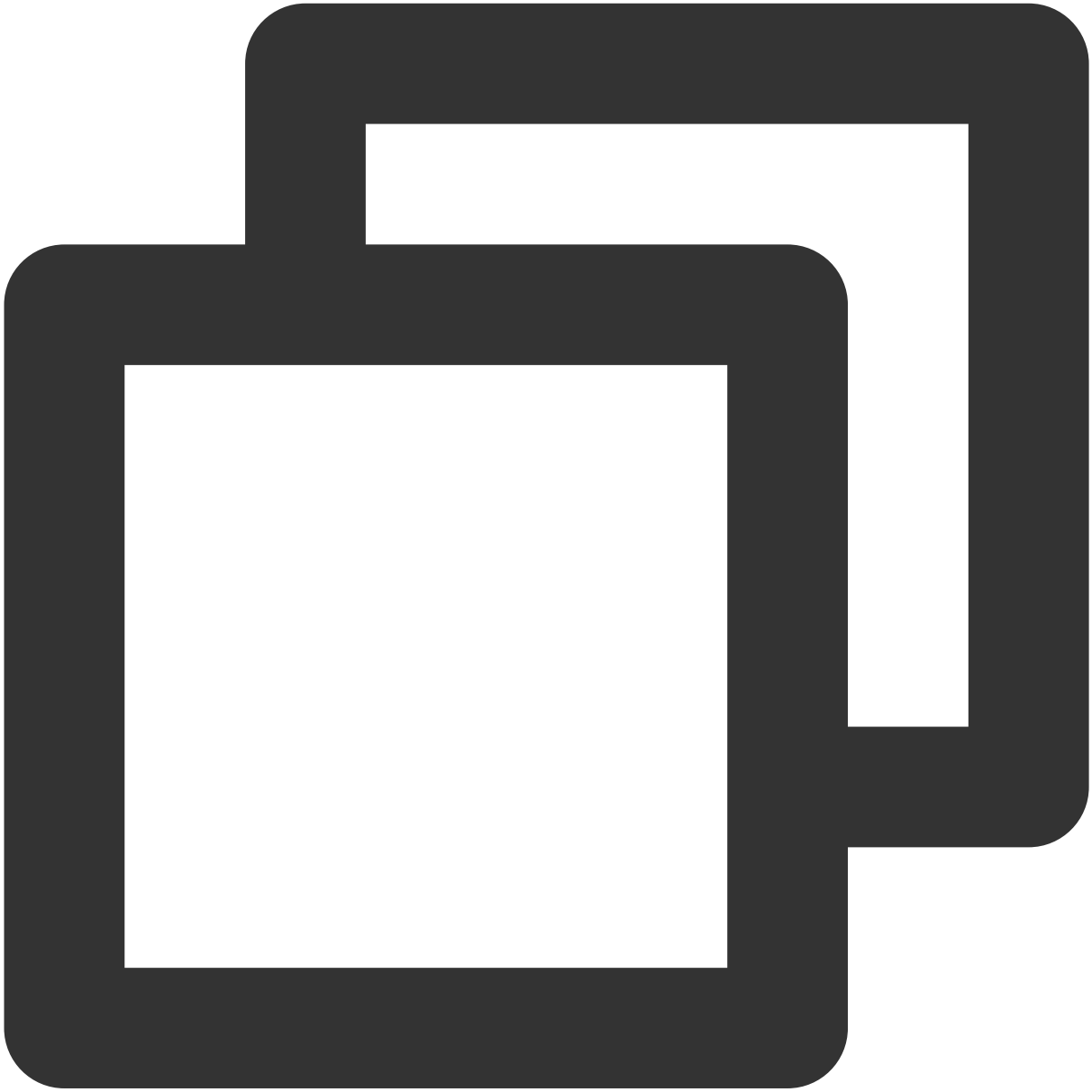
솔루션	장점
솔루션1: CoreDNS Hosts 플러그인을 사용하여 임의 도메인 이름 레졸루션 구성	이 솔루션은 간단하고 직관적입니다. 임의의 레졸루션 레코드를 추가할 수 있습니다.
솔루션2: CoreDNS Rewrite 플러그인을 사용하여 도메인 이름을 클러스터의 서비스에 매핑	레졸루션 레코드의 IP 주소를 미리 알 필요는 없지만 레졸루션 레코드로 매핑된 IP 주소는 클러스터에 배포되어야 합니다.
솔루션3: CoreDNS Forward 플러그인을 사용하여 외부 DNS를 업스트림 DNS로 설정	많은 수의 레졸루션 레코드를 관리할 수 있습니다. 모든 레코드는 외부 DNS에서 관리되므로 레코드를 추가하거나 삭제할 때 CoreDNS 구성을 수정할 필요가 없습니다.

설명
솔루션1과 솔루션2에서는 레졸루션 레코드를 추가할 때마다 CoreDNS 구성 파일을 수정해야 합니다(다시 시작할 필요 없음). 실제 필요에 따라 솔루션을 선택하십시오.

예시

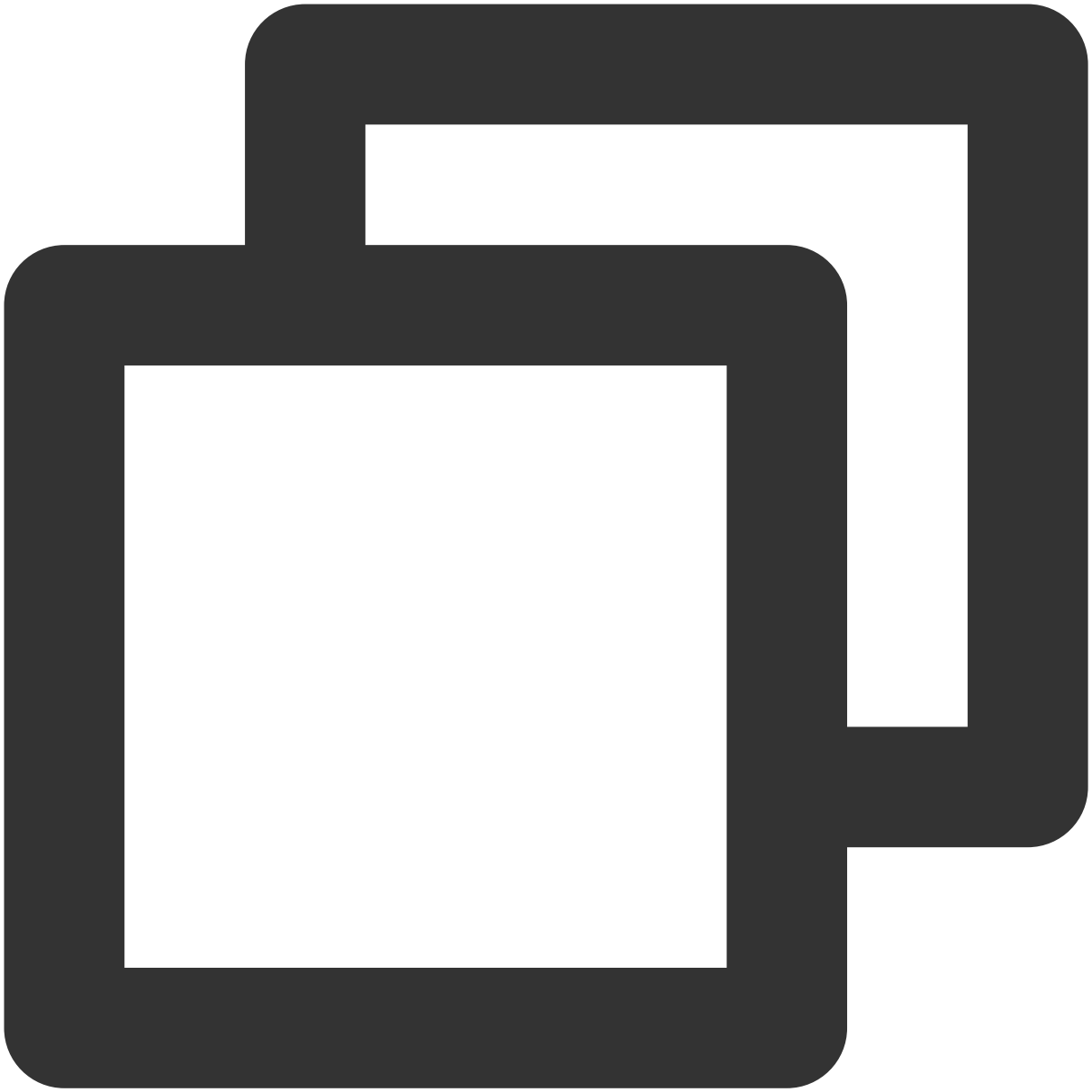
솔루션1: CoreDNS Hosts 플러그인을 사용하여 임의의 도메인 이름 레졸루션 구성

1. 다음 명령을 실행하여 아래와 같이 CoreDNS의 configmap을 수정합니다.



```
kubectl edit configmap coredns -n kube-system
```

2. 아래와 같이 hosts에 도메인 이름을 추가하도록 hosts 구성을 수정합니다.

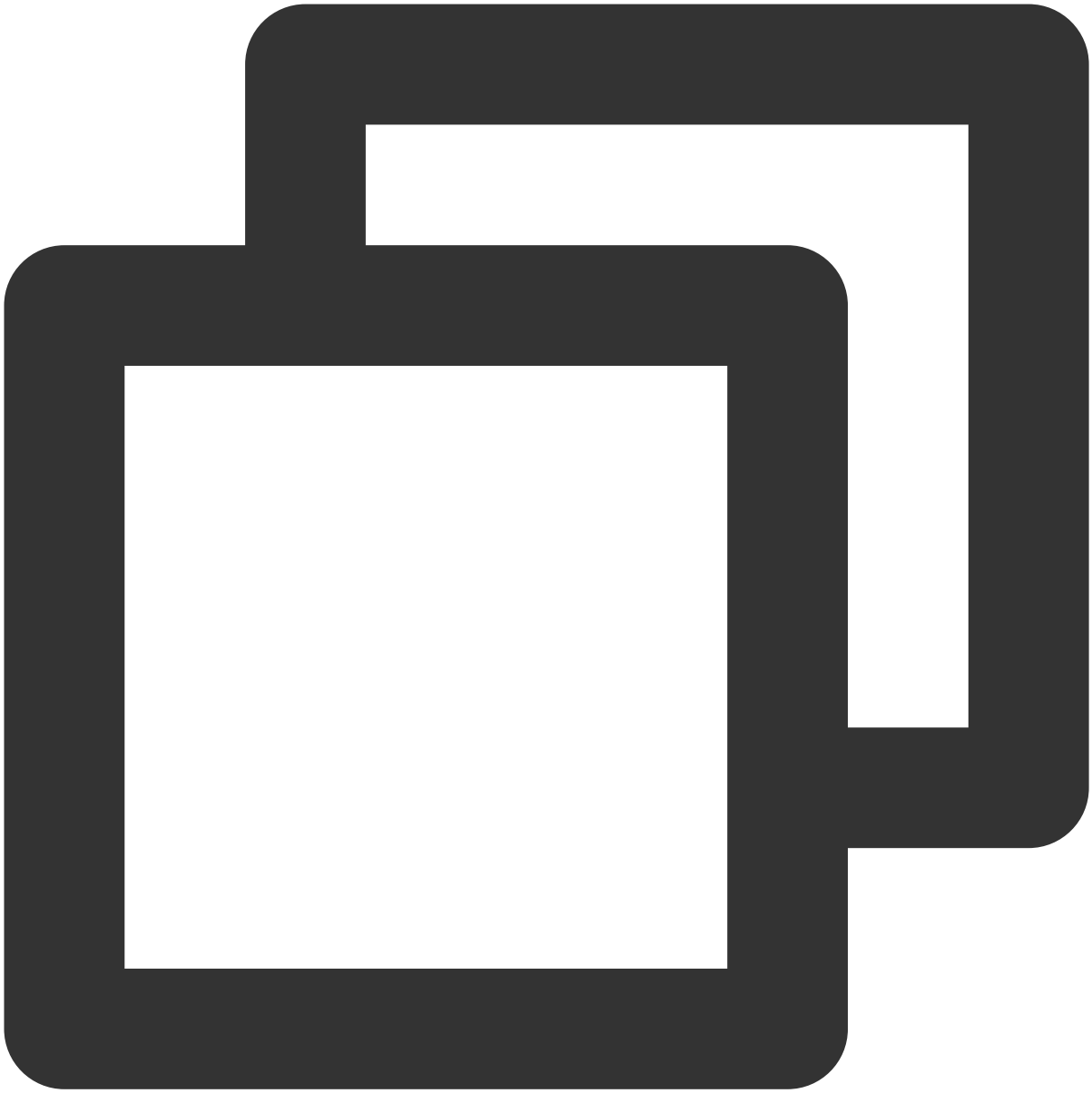


```
hosts {
      192.168.1.6     harbor.example.com
      192.168.1.8     es.example.com
      fallthrough
}
```

설명

`harbor.example.com` 을 192.168.1.6에 매핑하고 `es.example.com` 을 192.168.1.8에 매핑합니다.

전체 구성 예시는 다음과 같습니다.

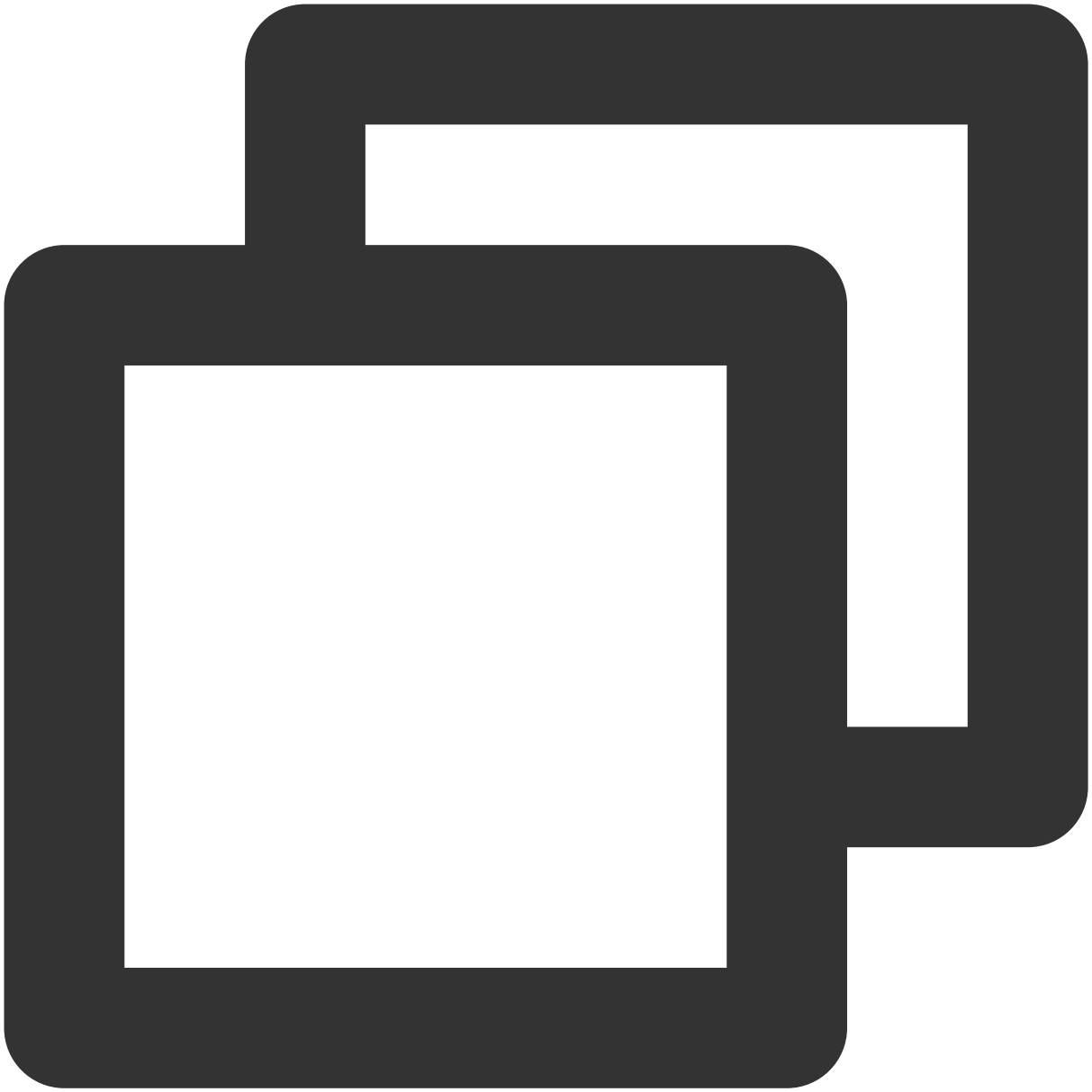


```
apiVersion: v1
data:
      Corefile: |2-
        .:53 {
            errors
            health
            kubernetes cluster.local. in-addr.arpa ip6.arpa {
                pods insecure
                upstream
                fallthrough in-addr.arpa ip6.arpa
            }
```

```
            hosts {
                192.168.1.6     harbor.example.com
                192.168.1.8     es.example.com
                fallthrough
            }
            prometheus :9153
            forward . /etc/resolv.conf
            cache 30
            reload
            loadbalance
        }
kind: ConfigMap
metadata:
      labels:
        addonmanager.kubernetes.io/mode: EnsureExists
      name: coredns
      namespace: kube-system
```

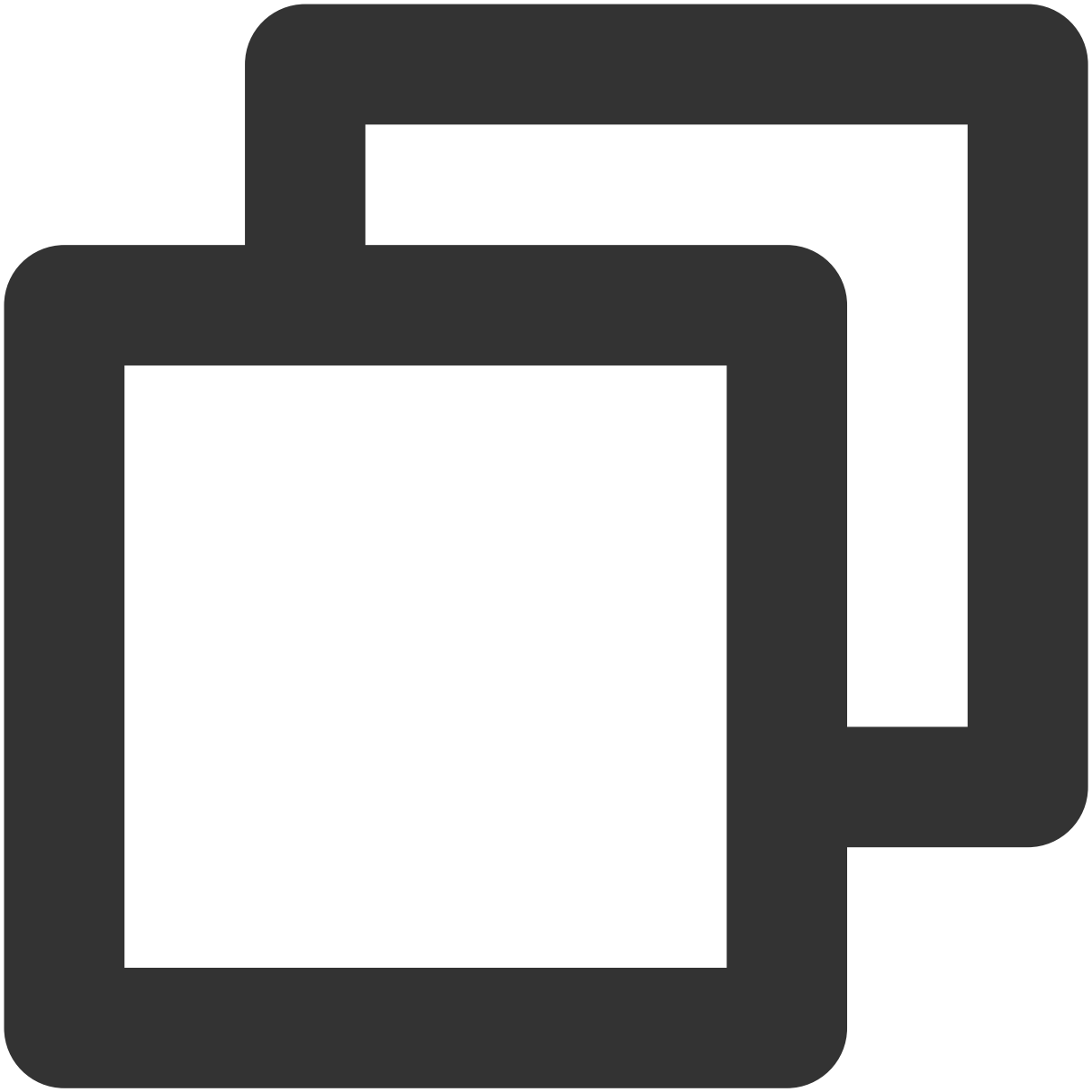
솔루션2: CoreDNS Rewrite 플러그인을 사용하여 도메인 이름을 클러스터의 서비스에 매핑

클러스터에 사용자 지정 도메인 이름이 있는 서비스를 배포하려면 CoreDNS의 Rewrite 플러그인을 사용하여 지정된 도메인 이름을 Service의 ClusterIP로 확인할 수 있습니다.
1. 다음 명령을 실행하여 아래와 같이 CoreDNS의 configmap을 수정합니다.



```
kubectl edit configmap coredns -n kube-system
```

2. 다음 명령을 실행하여 아래와 같이 Rewrite 구성을 추가합니다.

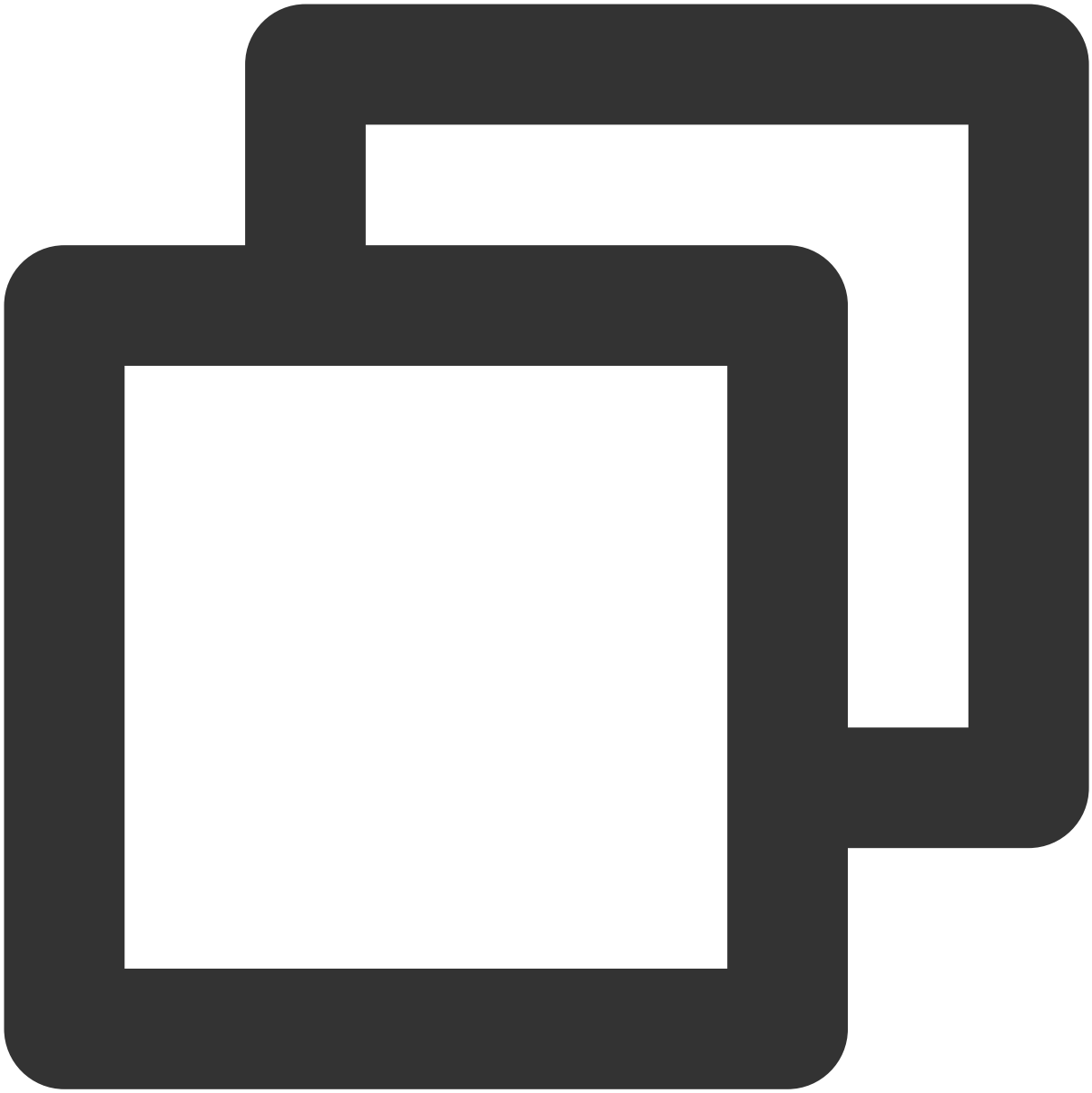


```
rewrite name es.example.com es.logging.svc.cluster.local
```

설명

`es.example.com` 을 `logging` 네임스페이스 아래에 배포된 `es` 서비스에 매핑합니다. 캐리지 리턴으로 여러 도메인 이름을 구분하십시오.

전체 구성 예시는 다음과 같습니다.

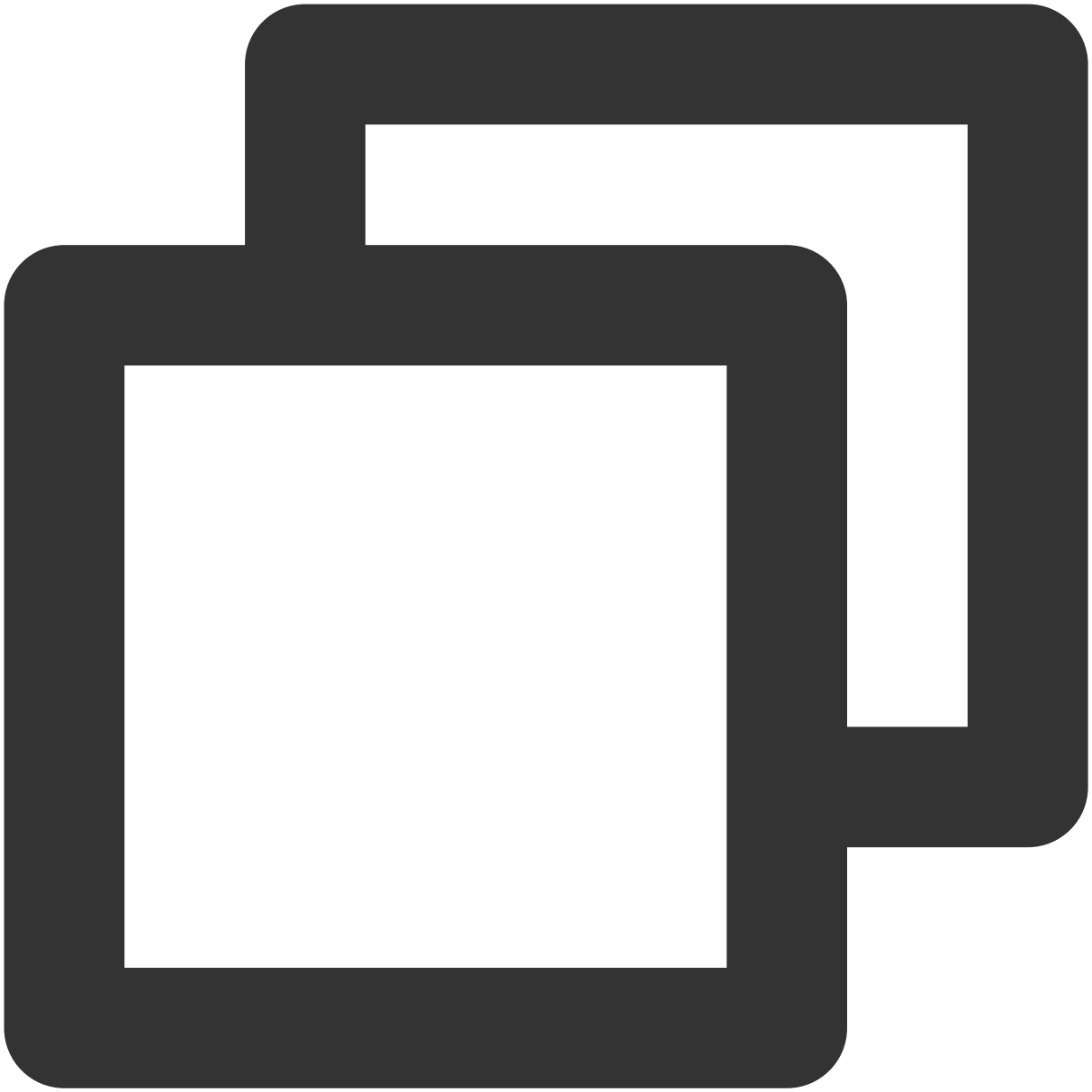


```
apiVersion: v1
data:
      Corefile: |2-
        .:53 {
            errors
            health
            kubernetes cluster.local. in-addr.arpa ip6.arpa {
                pods insecure
                upstream
                fallthrough in-addr.arpa ip6.arpa
            }
```

```
            rewrite name es.example.com es.logging.svc.cluster.local
            prometheus :9153
            forward . /etc/resolv.conf
            cache 30
            reload
            loadbalance
        }
kind: ConfigMap
metadata:
      labels:
        addonmanager.kubernetes.io/mode: EnsureExists
      name: coredns
      namespace: kube-system
```

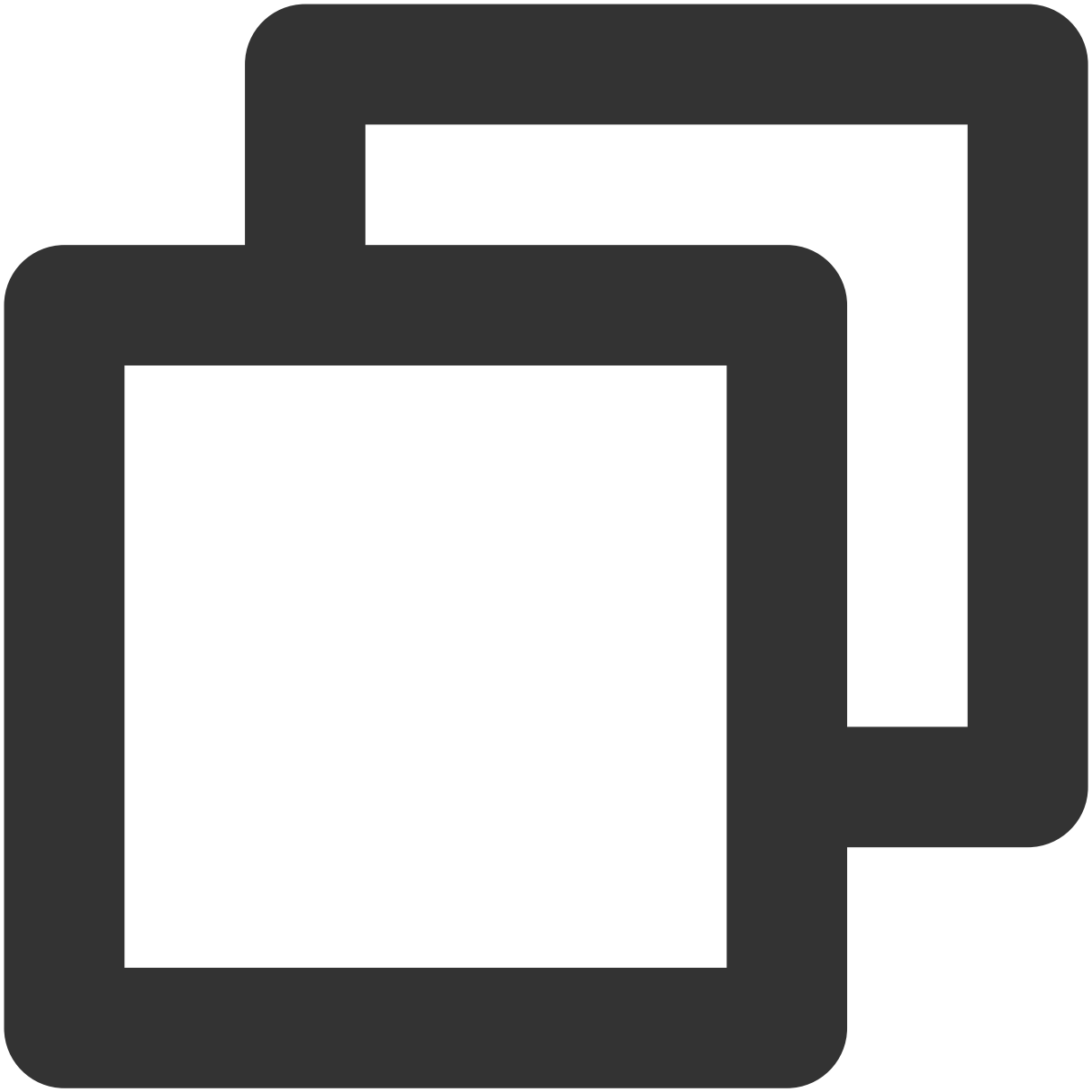
솔루션3: CoreDNS Forward 플러그인을 사용하여 외부 DNS를 업스트림 DNS로 설정

1. forward 구성을 확인합니다. forward의 기본 설정은 다음과 같습니다. 즉, 클러스터에 없는 도메인 이름은 CoreDNS가 위치한 노드의 `/etc/resolv.conf` 파일에 구성된 nameserver로 레졸루션됩니다.



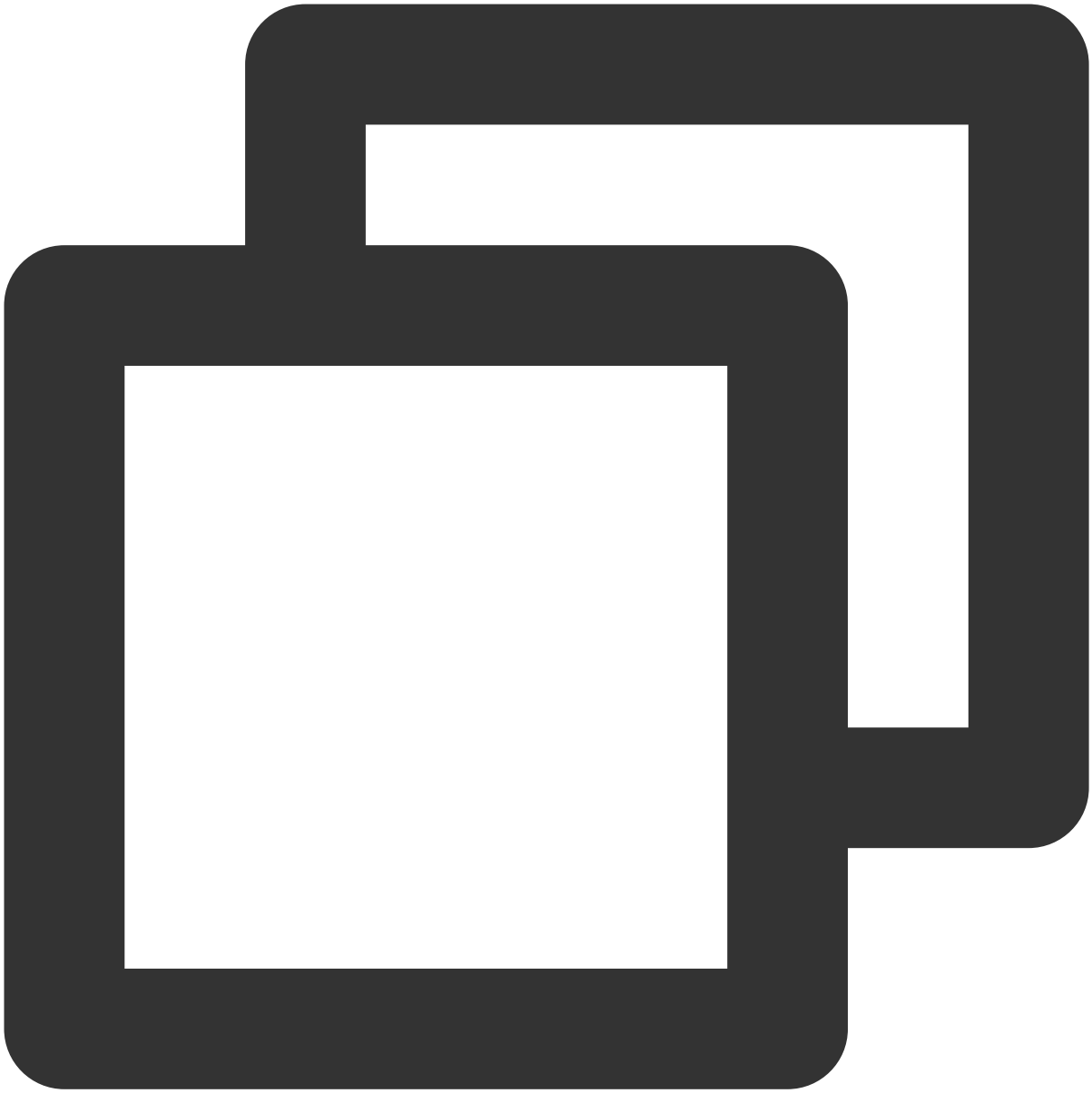
```
forward . /etc/resolv.conf
```

2. 다음과 같이 forward를 구성하고 `/etc/resolv.conf` 를 명시적으로 외부 DNS 서버 주소로 바꿉니다.



```
forward . 10.10.10.10
```

전체 구성 예시는 다음과 같습니다.



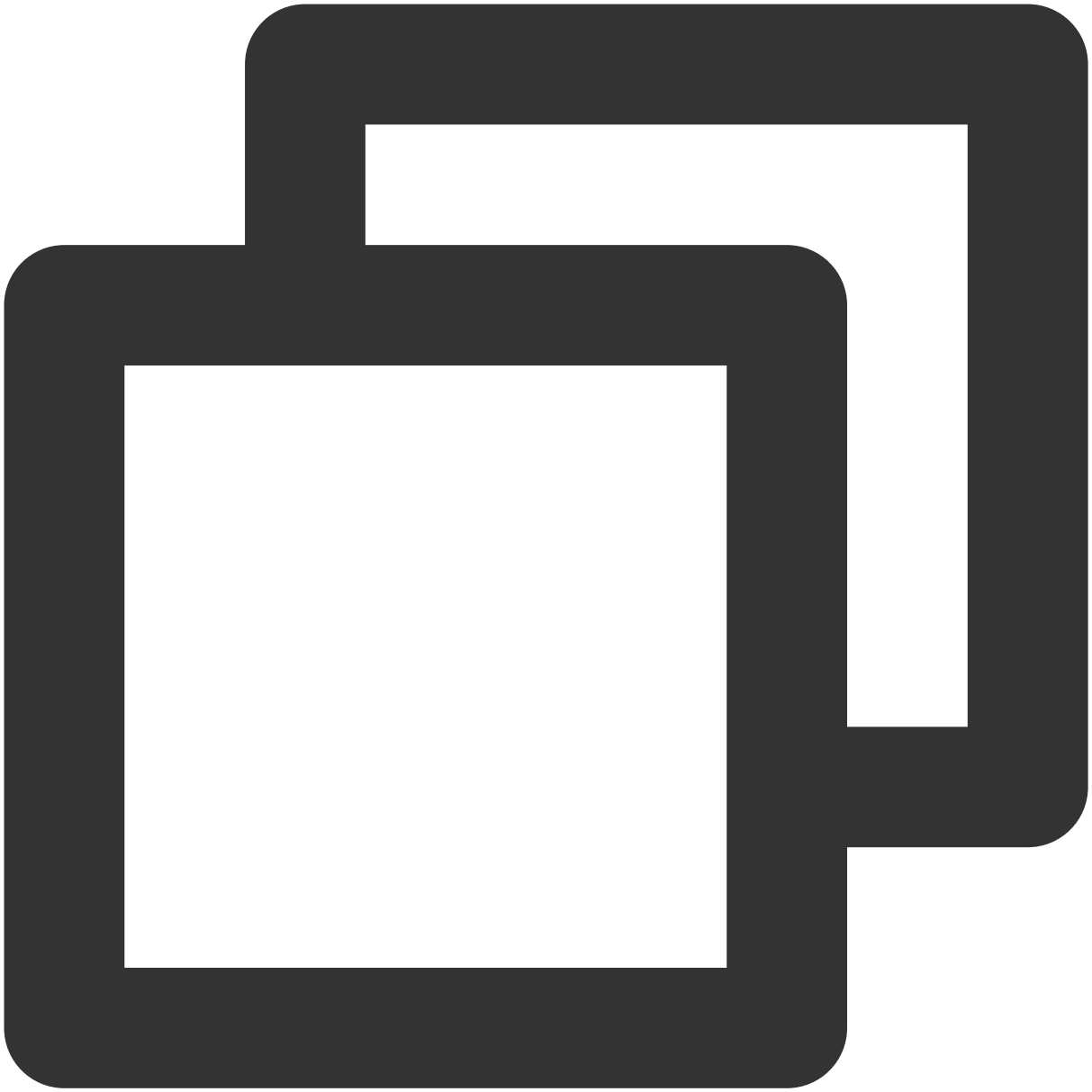
```
apiVersion: v1
data:
   Corefile: |2-
     .:53 {
         errors
         health
         kubernetes cluster.local. in-addr.arpa ip6.arpa {
             pods insecure
             upstream
             fallthrough in-addr.arpa ip6.arpa
         }
```

```
        prometheus :9153
        forward . 10.10.10.10
        cache 30
        reload
        loadbalance
    }
kind: ConfigMap
metadata:
  labels:
    addonmanager.kubernetes.io/mode: EnsureExists
  name: coredns
  namespace: kube-system
```

3. 외부 DNS에 대한 사용자 지정 도메인 이름의 레졸루션 레코드를 구성합니다. 외부 DNS의 업스트림 노드의 `/etc/resolv.conf` 에 nameserver를 추가하는 것을 권장합니다. 일부 서비스는 Tencent Cloud 내부 DNS 레졸루션에 의존하기 때문에 자체 구축 DNS의 업스트림으로 설정하지 않으면 일부 서비스가 제대로 작동하지 않을 수 있습니다. 이 문서는 아래와 같이 구성 파일을 수정하고 forwarders에 업스트림 DNS 주소를 쓰는 예시로 BIND 9를 사용합니다.

참고

외부 DNS Server와 요청 소스가 동일한 Region에 있지 않으면 리전 간 액세스를 지원하지 않는 일부 Tencent 도메인 이름이 무효화될 수 있습니다.



```
options {
     forwarders {
             183.60.83.19;
             183.60.82.98;
     };
     ...
```

참고 문서

CoreDNS Hosts
CoreDNS Rewrite
CoreDNS Forward

Nginx Ingress 모범 사례

최종 업데이트 날짜: : 2023-04-28 15:30:11

작업 시나리오

TKE는 Nginx-ingress 애드온 설치를 지원하고 이를 사용하여 Ingress 트래픽에 액세스합니다. Nginx-ingress에 대한 자세한 내용은 Nginx-ingress 설명을 참고하십시오. 본 문서는 Nginx-ingress 애드온의 모범 사례를 설명합니다.

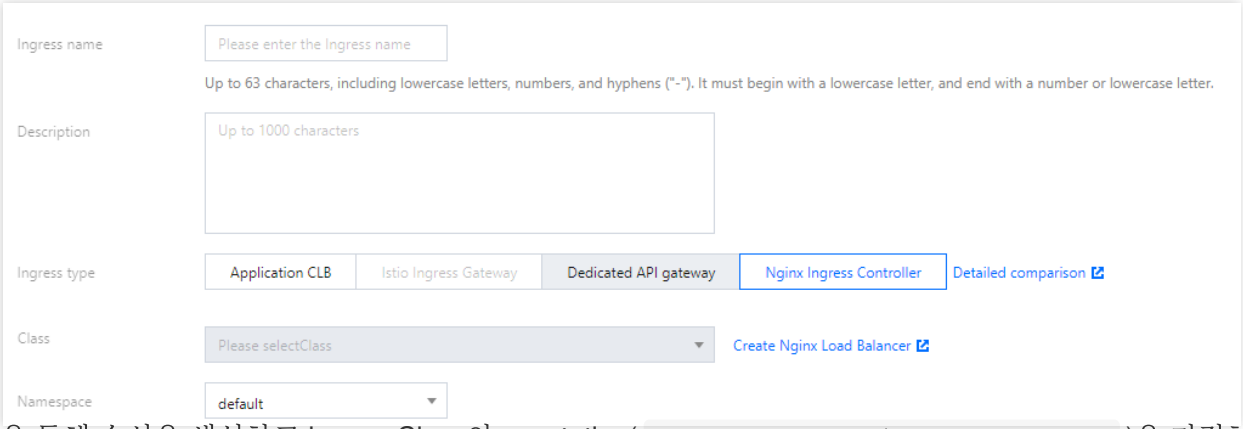
전제 조건

Nginx-ingress 애드온을 설치합니다.

작업 단계

클러스터에 대한 여러 Nginx Ingress 트래픽 항목 열기

Nginx-ingress 애드온이 설치되면 `kube-system` 아래에 Nginx-ingress operator 애드온이 있습니다. 이 애드온을 사용하여 여러 Nginx Ingress 인스턴스를 생성할 수 있습니다. 각 Nginx Ingress 인스턴스는 서로 다른 IngressClass를 사용하고 서로 다른 CLB를 트래픽 항목으로 사용하므로 서로 다른 Ingress가 서로 다른 트래픽 항목에 바인딩될 수 있습니다. 실제 요구 사항에 따라 클러스터에 대해 여러 Nginx Ingress 인스턴스를 생성할 수 있습니다.
1. TKE 콘솔에 로그인하고 왼쪽 사이드바에서 **클러스터**를 클릭합니다.
2. 클러스터 관리 페이지에서 대상 클러스터의 ID를 클릭하여 클러스터 세부 정보 페이지로 이동합니다.
3. 왼쪽 사이드바에서 **애드온 관리**를 클릭하여 애드온 목록 페이지로 이동합니다.
4. 설치된 Nginx-ingress 애드온을 클릭하여 세부 정보 페이지로 이동합니다.
5. **Nginx Ingress 인스턴스 추가**를 클릭하여 필요에 따라 Nginx Ingress 인스턴스를 구성하고 각 인스턴스에 대해 다른 IngressClass 이름을 지정합니다.
설명
Nginx Ingress 인스턴스 설치에 대한 자세한 내용은 Nginx-ingress 인스턴스 설치를 참고하십시오.
6. Ingress를 생성할 때 특정 IngressClass를 지정하여 Ingress를 특정 Nginx Ingress 인스턴스에 바인딩할 수 있습니다. 콘솔 또는 YAML을 통해 Ingress를 생성할 수 있습니다.
콘솔을 통해 Ingress 생성
YAML을 통해 Ingress 생성
콘솔에서 Ingress를 생성하는 방법에 대한 자세한 내용은 Ingress 생성을 참고하십시오.
Ingress유형: **Nginx 로드 밸런서**를 선택합니다.
Class: 이전 단계에서 생성한 Nginx Ingress 인스턴스를 선택합니다.



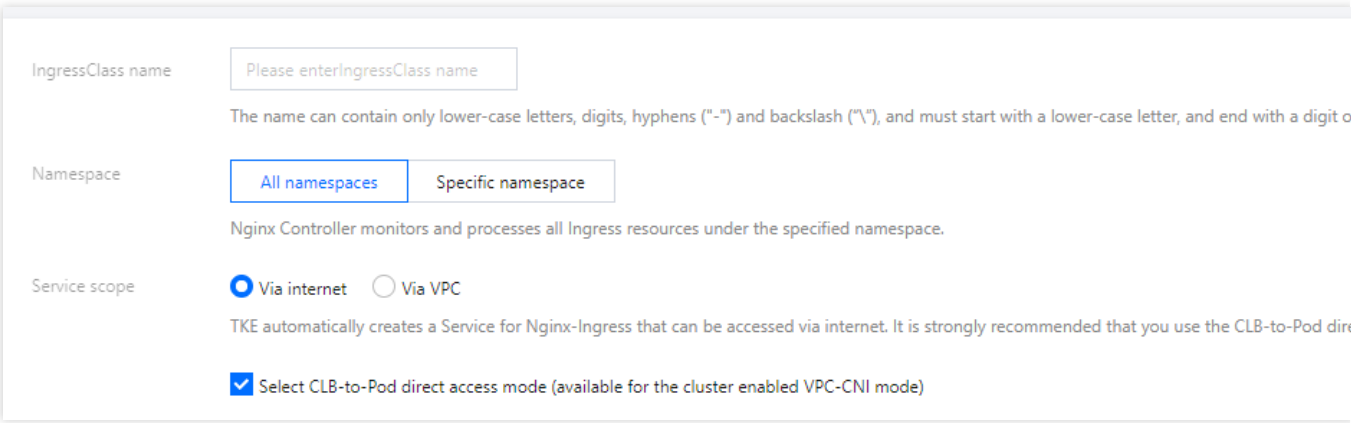
YAML을 통해 수신을 생성하고 ingressClass의 annotation(`kubernetes.io/ingress.class`)을 지정하는 방법에 대한 자세한 내용은 Ingress 생성을 참고하십시오.

```
1 apiVersion: networking.k8s.io/v1beta1
2 kind: Ingress
3 metadata:
4   annotations:
5     ingress.cloud.tencent.com/direct-access: "false"
6     kubernetes.io/ingress.class: nginx-external
```

성능 최적화

CLB-to-Pod 다이렉트 액세스 모드

클러스터 네트워크 모드가 Global Router인 경우 CLB-to-Pod 다이렉트 액세스 모드는 기본적으로 활성화되지 않습니다. 다음 지침에 따라 CLB-to-Pod 다이렉트 액세스 모드를 활성화하는 것이 좋습니다.
1. 클러스터에 대해 VPC-CNI 모드를 활성화합니다.
2. Nginx Ingress 인스턴스를 생성할 때 **CLB-to-Pod 다이렉트 액세스 모드 선택**을 선택하여 트래픽이 NodePort를 우회하고 Pod에 직접 도달하여 성능을 향상시킬 수 있습니다.

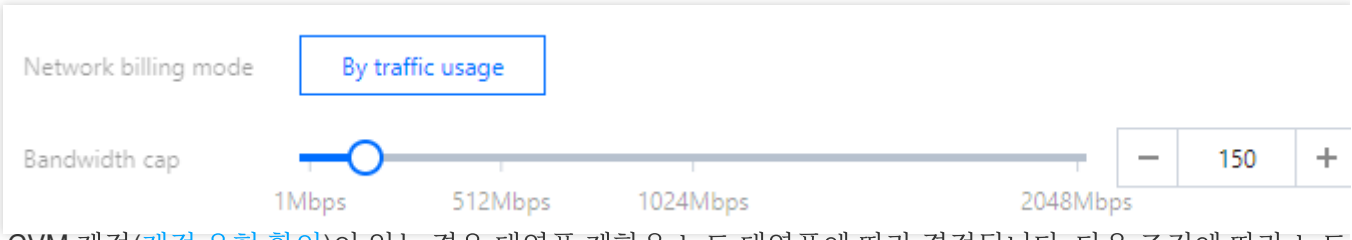


설명

Nginx Ingress 인스턴스 설치에 대한 자세한 내용은 Nginx-ingress 인스턴스 설치를 참고하십시오.

LB 대역폭 제한 조정

트래픽 항목으로 LB에 더 높은 동시성 또는 처리량이 필요한 경우 Nginx Ingress 인스턴스를 생성할 때 실제 요구 사항을 기반으로 대역폭 제한을 설정하고 Nginx Ingress에 더 높은 대역폭을 할당할 수 있습니다.



Bill-by-CVM 계정(계정 유형 확인)이 있는 경우 대역폭 제한은 노드 대역폭에 따라 결정됩니다. 다음 조건에 따라 노드 대역폭 제한을 조정할 수 있습니다.

CLB-to-Pod 다이렉트 액세스 모드가 활성화된 경우 총 LB 대역폭은 Nginx Ingress 인스턴스 Pod가 위치한 노드의 대역폭 합계입니다. Nginx Ingress 인스턴스를 배포하기 위해 공중망 대역폭이 높은 일부 노드를 계획하는 것이 좋습니다(노드 풀을 DaemonSet으로 지정하여 배포).

CLB-to-Pod 다이렉트 액세스 모드가 활성화되지 않은 경우 LB의 총 대역폭은 모든 노드의 공중망 대역폭의 합계입니다.

Nginx-ingress 매개변수 최적화

Nginx Ingress 인스턴스는 기본적으로 커널 매개변수와 Nginx Ingress 구성을 최적화할 수 있습니다. 자세한 내용은 Nginx Ingress 높은 동시성 사례를 참고하십시오. 다음 지침을 참고하여 사용자 정의할 수 있습니다.

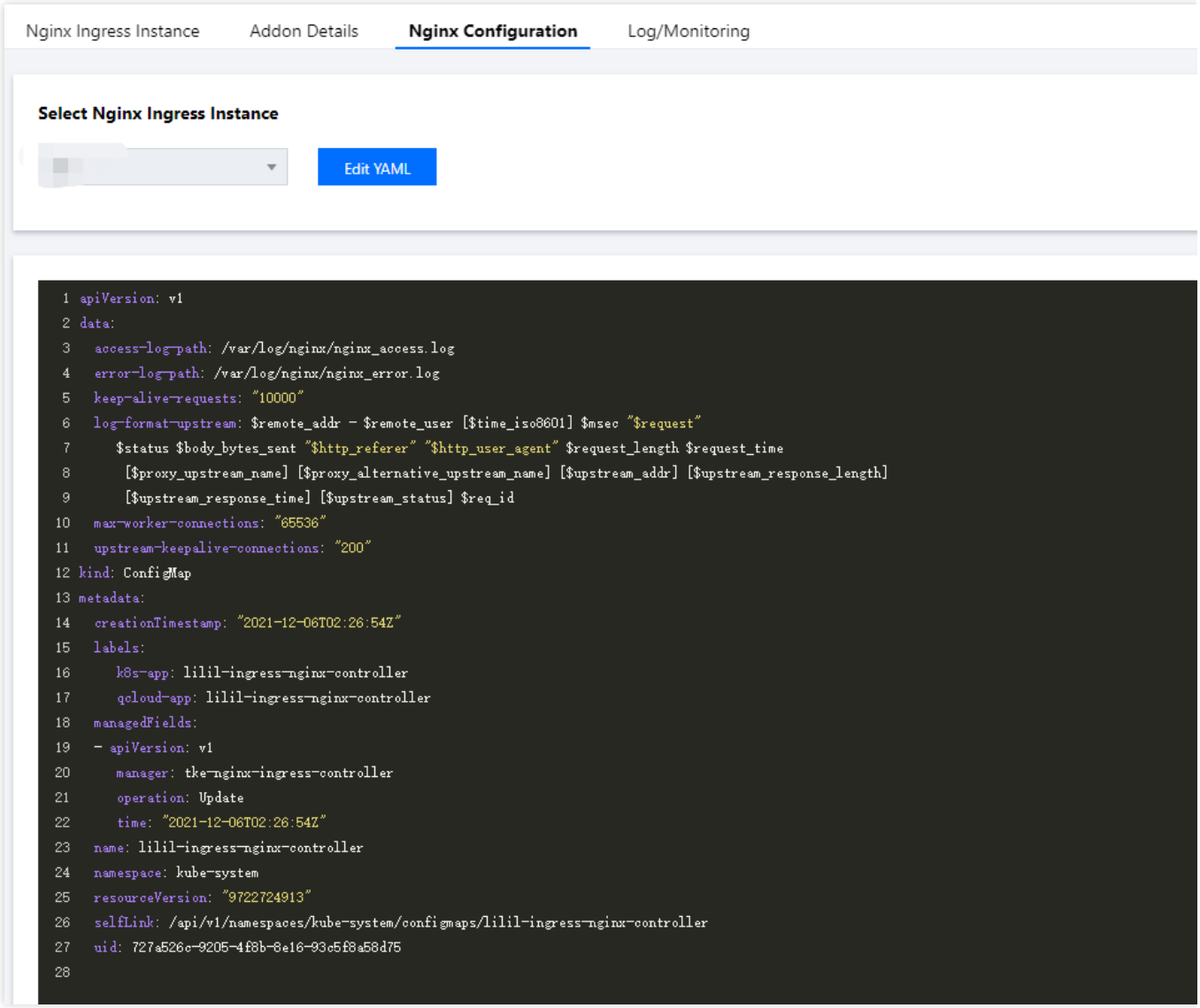
커널 매개변수 수정

Nginx Ingress 구성 수정

nginx-ingress-conntroller의 배포된 Daemonset 또는 Deployment(인스턴스 배포 옵션에 따라 다름)를 편집하고 아래와 같이 initContainers를 수정합니다(콘솔에서 kube-system 아래의 리소스는 수정할 수 없습니다. Kubectl을 사용하여 수정하십시오).

```
initContainers:
- command:
  - sh
  - -c
  - |-
    sysctl -w net.core.somaxconn=65535
    sysctl -w net.ipv4.ip_local_port_range="1024 65535"
    sysctl -w net.ipv4.tcp_tw_reuse=1
    sysctl -w fs.file-max=1048576
```

Nginx 구성 탭에서 수정할 인스턴스를 선택하고 **YAML 편집**을 클릭한 다음 아래와 같이 Nginx Ingress 인스턴스의 ConfigMap 구성을 수정합니다.



설명

ConfigMap 구성에 대한 자세한 내용은 공식 문서를 참고하십시오.

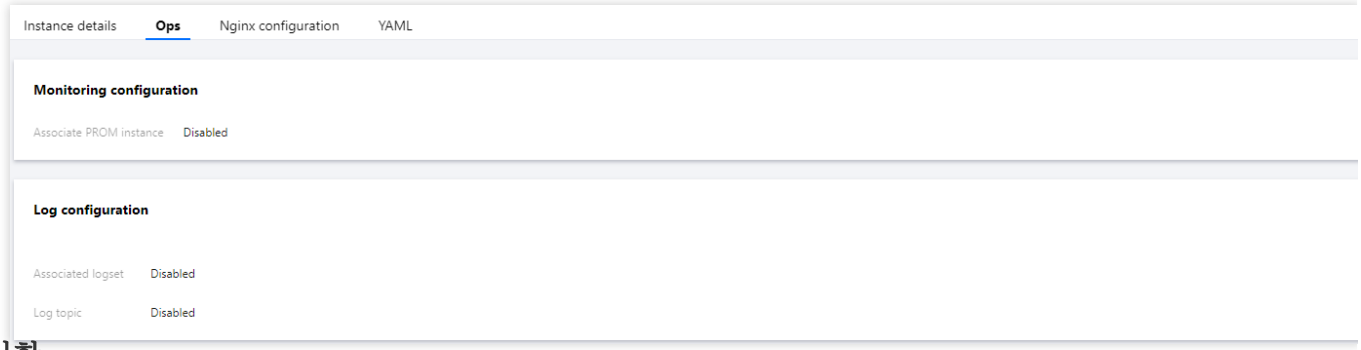
Nginx Ingress의 관찰 가능성 개선

로그 활성화

설명:

로그 구성은 CLS에 의존합니다. 사용 설정 방법은 Nginx-ingress 로그 구성을 참고하십시오.

Nginx Ingress 인스턴스를 생성한 후 인스턴스 세부 정보의 **유지보수** 기능 엔트리에서 인스턴스의 **로그** 구성을 활성화할 수 있습니다. 이는 아래와 같이 인스턴스의 상태 지표를 확인하고 문제를 해결하는 데 편리합니다.



주의사항:

v0.49.3 버전의 인스턴스에서 로그 수집의 검색 구성 파일은 LogConfig CRD 리소스 객체에서 찾을 수 있습니다. 이 리소스 객체를 수정한 후 해당 로그 수집 기능을 닫고/다시 열면 LogConfig 리소스 객체의 구성이 재설정됩니다. 이에 따라 리소스 객체의 데이터를 적시에 백업해야 합니다. Nginx Ingress 인스턴스 자체의 삭제 및 Nginx Ingress 애드온의 업그레이드는 해당 검색 구성 파일에 영향을 미치지 않습니다.

로그를 사용자 지정하려면 문서에 따라 구성하십시오.

로그 검색 및 로그 대시보드

로그 구성을 활성화한 후 Nginx Ingress 목록 페이지의 인스턴스 오른쪽에 있는 **작업**에서 **자세히**를 클릭하고 CLS에서 액세스 로그 확인 또는 액세스 로그 대시보드 보기를 선택할 수 있습니다.

CLS에서 액세스 로그 확인을 클릭하여 CLS로 이동하고 **검색 및 분석**에서 인스턴스에 해당하는 로그셋 및 항목을 선택하여 Nginx Ingress의 액세스 및 오류 로그를 확인합니다.

액세스 로그 대시보드 보기를 클릭하여 Nginx 인그레스 로그 데이터를 기반으로 통계를 표시하는 대시보드로 이동합니다.

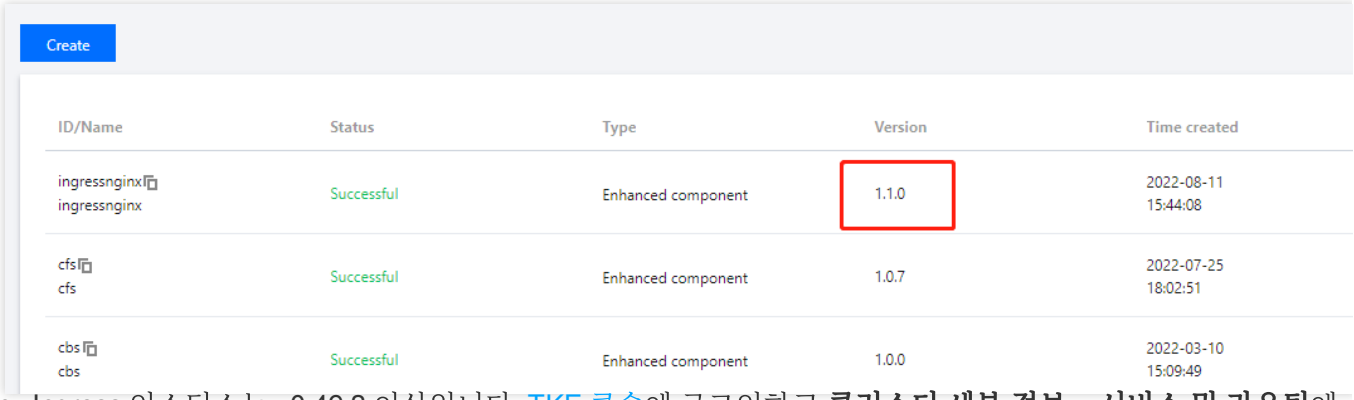
로그
사용자 지정 NginxIngress 로그

최종 업데이트 날짜: : 2023-04-28 11:08:19

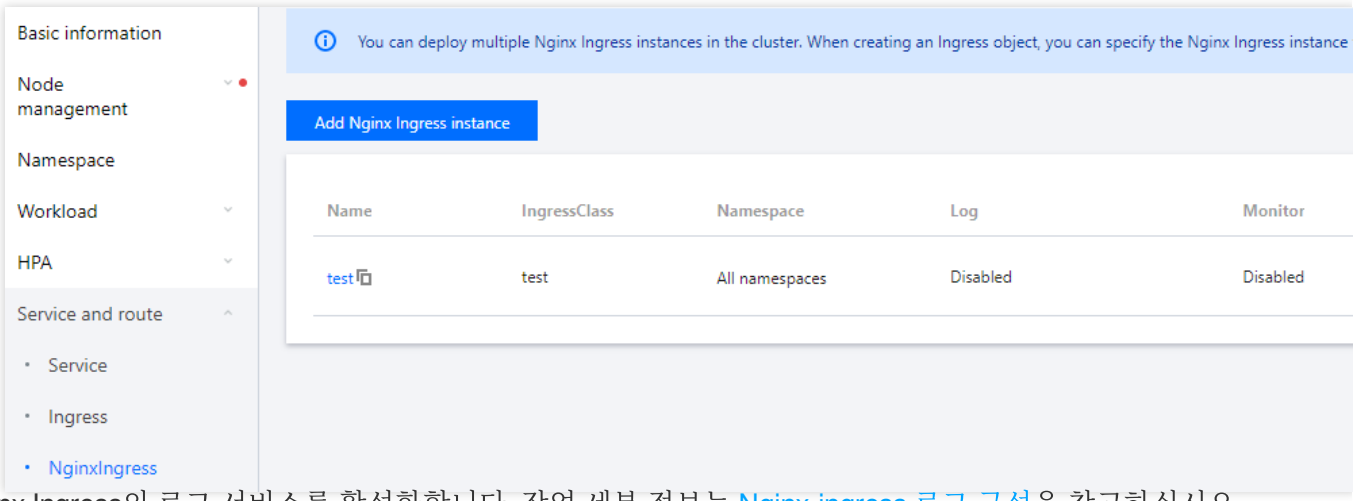
TKE는 CLS 통합을 통해 제품화된 완전한 능력을 제공하여 Nginx-ingress 로그 수집 및 소비 기능을 구현합니다. 자세한 내용은 Nginx-ingress 로그 구성을 참고하십시오. 기본 로그 인덱스가 요구 사항을 충족하지 않는 경우 인덱스를 사용자 지정할 수 있습니다. 이 문서는 Nginx Ingress의 로그 인덱스를 업데이트하는 방법을 설명합니다.

전제 조건

1. Nginx Ingress는 v1.1.0 이상입니다. TKE 콘솔에 로그인하고 **클러스터 세부 정보 > 애드온 관리**를 선택하면 Nginx Ingress 애드온의 버전을 볼 수 있습니다.
주의사항:
이 기능은 Nginx Ingress 버전 v1.1.0 이상에서만 지원됩니다. 예를 들어 버전 v1.0.0과 같은 v1.1.0 미만의 버전에서는 로그 인덱스 수정이 애드온에 의해 롤백되어 덮어씌워질 수 있습니다.



2. Nginx Ingress 인스턴스는 v0.49.3 이상입니다. TKE 콘솔에 로그인하고 **클러스터 세부 정보 > 서비스 및 라우팅**에서 **NginxIngress**를 선택한 후 대상 인스턴스 오른쪽에 있는 **YAML 보기**를 클릭합니다. YAML 파일에서 `ccr.ccs.tencentyun.com/paas/nginx-ingress-controller` 이미지는 v0.49.3 이상이어야 합니다.



3. Nginx Ingress의 로그 서비스를 활성화합니다. 작업 세부 정보는 Nginx-ingress 로그 구성을 참고하십시오.

작업 단계

주의사항

로그 구조를 수정하려면 로그 출력, 로그 수집, 로그 인덱싱으로 구성된 Nginx Ingress의 로그 스트림을 이해해야 합니다. 여기에서 로그 출력 또는 수집이 누락되거나 잘못 구성된 경우 로그 수정이 실패합니다.

1단계: Nginx Ingress 인스턴스의 로그 출력 형식 수정

Nginx Ingress 인스턴스의 로그 구성은 프라이머리 구성 ConfigMap `인스턴스 이름-ingress-nginx-controller` 에 있으며 여기에서 `log-format-upstream` Key를 수정해야 합니다.

```
apiVersion: v1
data:
  access-log-path: /var/log/nginx/nginx_access.log
  allow-snippet-annotations: "false"
  error-log-path: /var/log/nginx/nginx_error.log
  keep-alive-requests: "10000"
  log-format-upstream: $remote_addr - $remote_user [$time_iso8601] $msec "$request"
    $status $body_bytes_sent "$http_referer" "$http_user_agent" $request_length $request_time
    [$proxy_upstream_name] [$proxy_alternative_upstream_name] [$upstream_addr] [$upstream_response_
    [$upstream_response_time] [$upstream_status] $req_id $service_name $namespace
  max-worker-connections: "65536"
  upstream-keepalive-connections: "200"
kind: ConfigMap
metadata:
  creationTimestamp: "2022-07-22T02:56:35Z"
  labels:
    k8s-app:       s-ingress-nginx-controller
    qcloud-app:          -ingress-nginx-controller
  managedFields:
  - apiVersion: v1
    fieldsType: FieldsV1
    fieldsV1:
      f:data:
        .: {}
        f:access-log-path: {}
        f:allow-snippet-annotations: {}
        f:error-log-path: {}
        f:keep-alive-requests: {}
        f:max-worker-connections: {}
```

예시

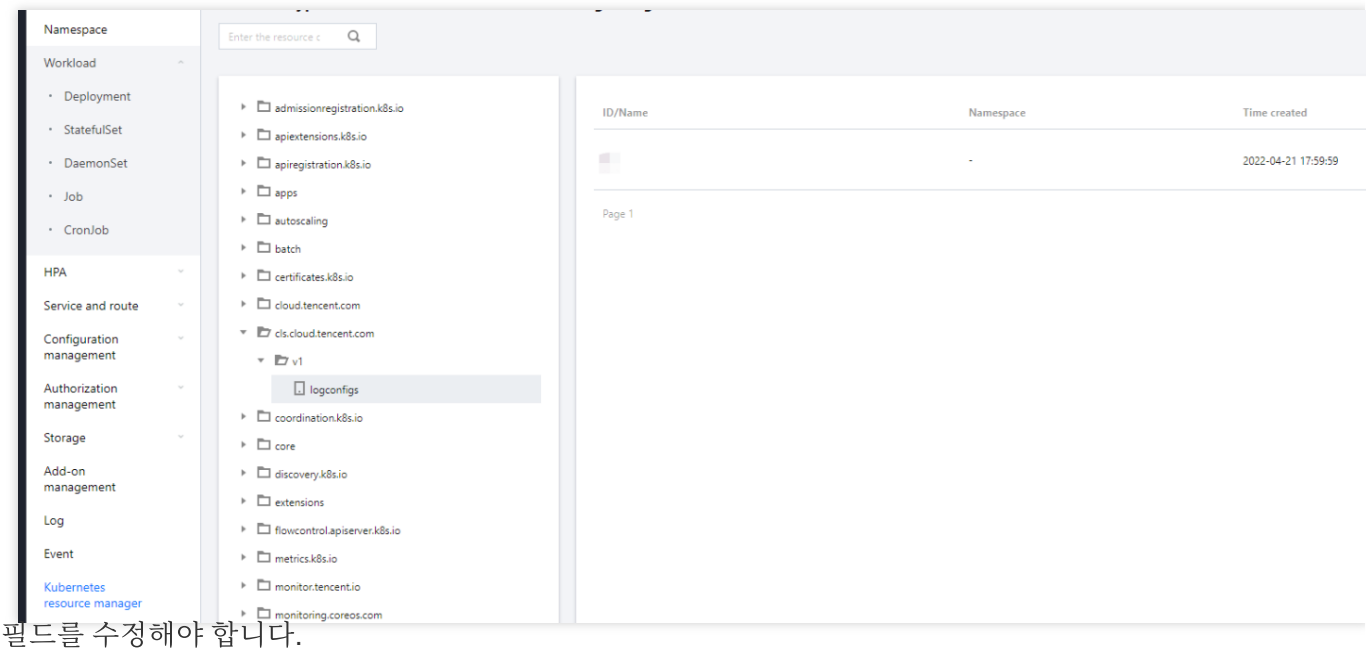
두 개의 연속 문자열 `$namespace` 및 `$service_name`을 로그 끝에 추가합니다. 추가 위치는 아래 그림과 같습니다.

```
apiVersion: v1
data:
  access-log-path: /var/log/nginx/nginx_access.log
  allow-snippet-annotations: "false"
  error-log-path: /var/log/nginx/nginx_error.log
  keep-alive-requests: "10000"
  log-format-upstream: $remote_addr - $remote_user [$time_iso8601] $msec "$request"
    $status $body_bytes_sent "$http_referer" "$http_user_agent" $request_length $request_time
    [$proxy_upstream_name] [$proxy_alternative_upstream_name] [$upstream_addr] [$upstream_response_le
    [$upstream_response_time] [$upstream_status] $req_id $service_name $namespace
  max-worker-connections: "65536"
  upstream-keepalive-connections: "200"
kind: ConfigMap
```

Nginx Ingress의 로그 필드에 대한 자세한 내용은 문서를 참고하십시오.

2단계: 클러스터 로그 수집 및 Agent 리포트 형식 수정

클러스터 로그 수집 규칙은 logconfigs.cls.cloud.tencent.com 유형의 리소스 객체에 있습니다. TKE 콘솔에 로그인하고 **클러스터 세부 정보 > Kubernetes 리소스 관리자**를 선택한 후, `인스턴스 이름-ingress-nginx-controller` 리소스 객체를 찾고 **YAML 편집**을 클릭하여 수정할 수 있습니다.



다음 필드를 수정해야 합니다.

beginningRegex: 로그 시작의 정규식

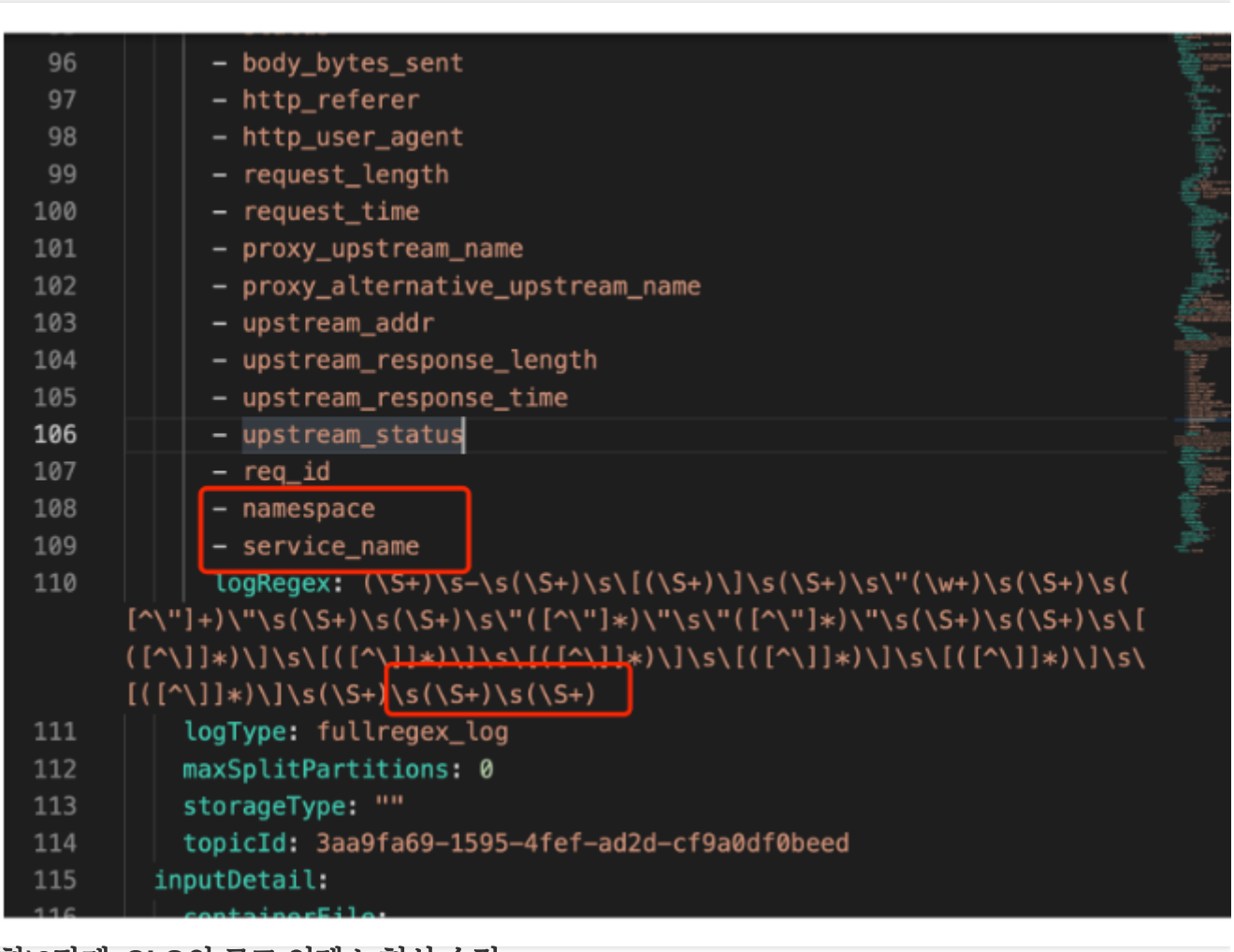
keys: 로그 필드

logRegex: 로그 종료의 정규식

정규식은 Nginx 로그 행 형식과 일치합니다. 기존 Nginx 로그 형식에 필드를 추가하고, keys 끝에 선언하고, 정규식 구문 분석 결과를 각각 beginningRegex 및 logRegex 끝에 추가하는 것이 좋습니다.

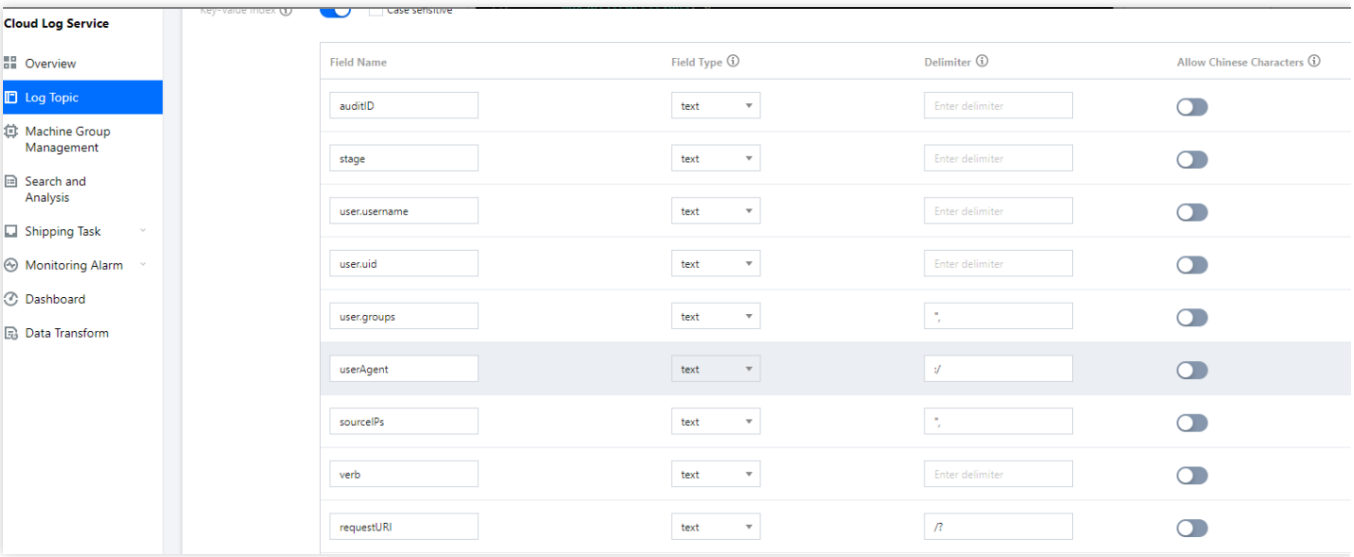
예시

1단계에서 키 두 개를 keys 끝에 추가하고 정규식 문자열을 각각 beginningRegex 및 logRegex 끝에 추가합니다.



(선택 사항)3단계: CLS의 로그 인덱스 형식 수정

필드를 검색하려면 인덱스 구성에 따라 CLS 콘솔의 해당 로그 항목에 새 필드의 인덱스를 추가해야 합니다. 이렇게 하면 수집된 모든 로그를 인덱스로 검색할 수 있습니다.



초기 설정 복원

로그 규칙 수정은 복잡하고 정규식을 포함하므로 잘못된 단계로 인해 로그 수집 실패가 발생할 수 있습니다. 로그 수집 오류가 발생할 경우 로그 수집 기능을 비활성화했다가 다시 로그 수집 활성화하여 초기 로그 수집 기능으로 복원할 것을 권장합니다.

모니터링
Prometheus를 사용하여 MySQL 및 MariaDB 모니터링

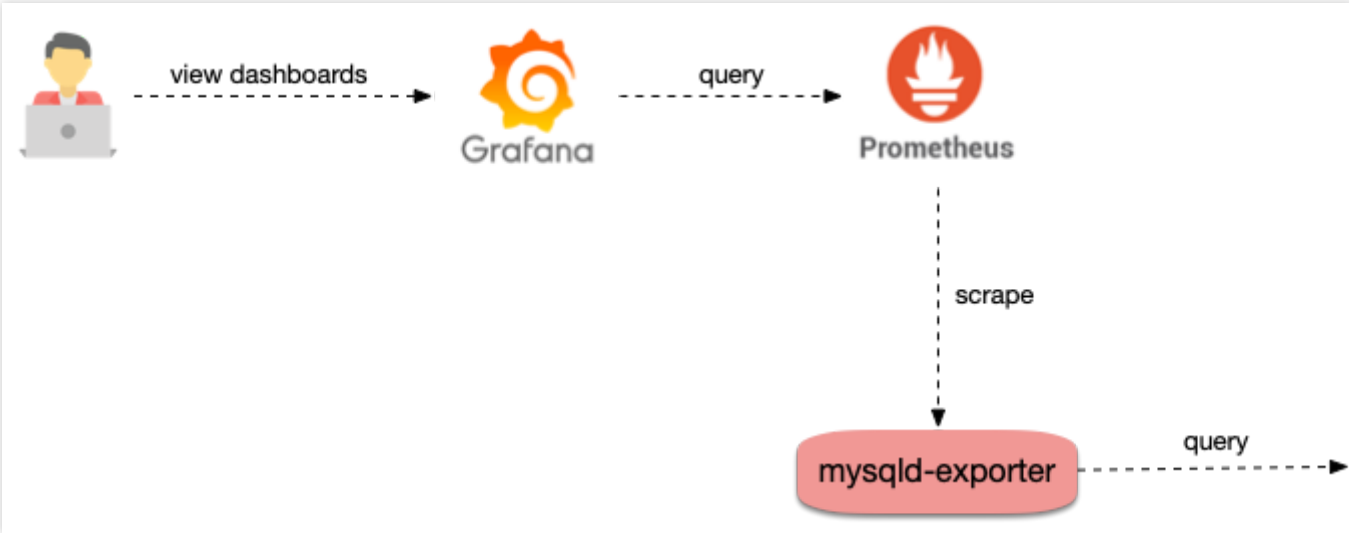
최종 업데이트 날짜: : 2023-04-26 18:43:22

작업 시나리오

MySQL은 일반적인 관계형 데이터베이스 관리 시스템입니다. MariaDB는 MySQL의 브랜치 버전으로서 MySQL과 호환되어 점점 인기를 얻고 있습니다. Kubernetes 환경에서 Prometheus를 사용하여 오픈 소스 mysqld-exporter를 사용하여 MySQL 및 MariaDB 데이터베이스를 모니터링할 수 있습니다. 본문은 Prometheus를 사용하여 MySQL 및 MariaDB를 모니터링하는 방법을 설명합니다.

mysqld-exporter 소개

mysqld-exporter는 MySQL 또는 MariaDB에서 데이터베이스 상태 데이터를 읽고 이를 Prometheus 메트릭 형식으로 변환한 다음 HTTP 인터페이스로 엽니다. 이 경우 Prometheus는 이러한 메트릭을 수집하고 모니터링할 수 있습니다. 다음 이미지와 같습니다.



작업 단계

mysqld-exporter 배포

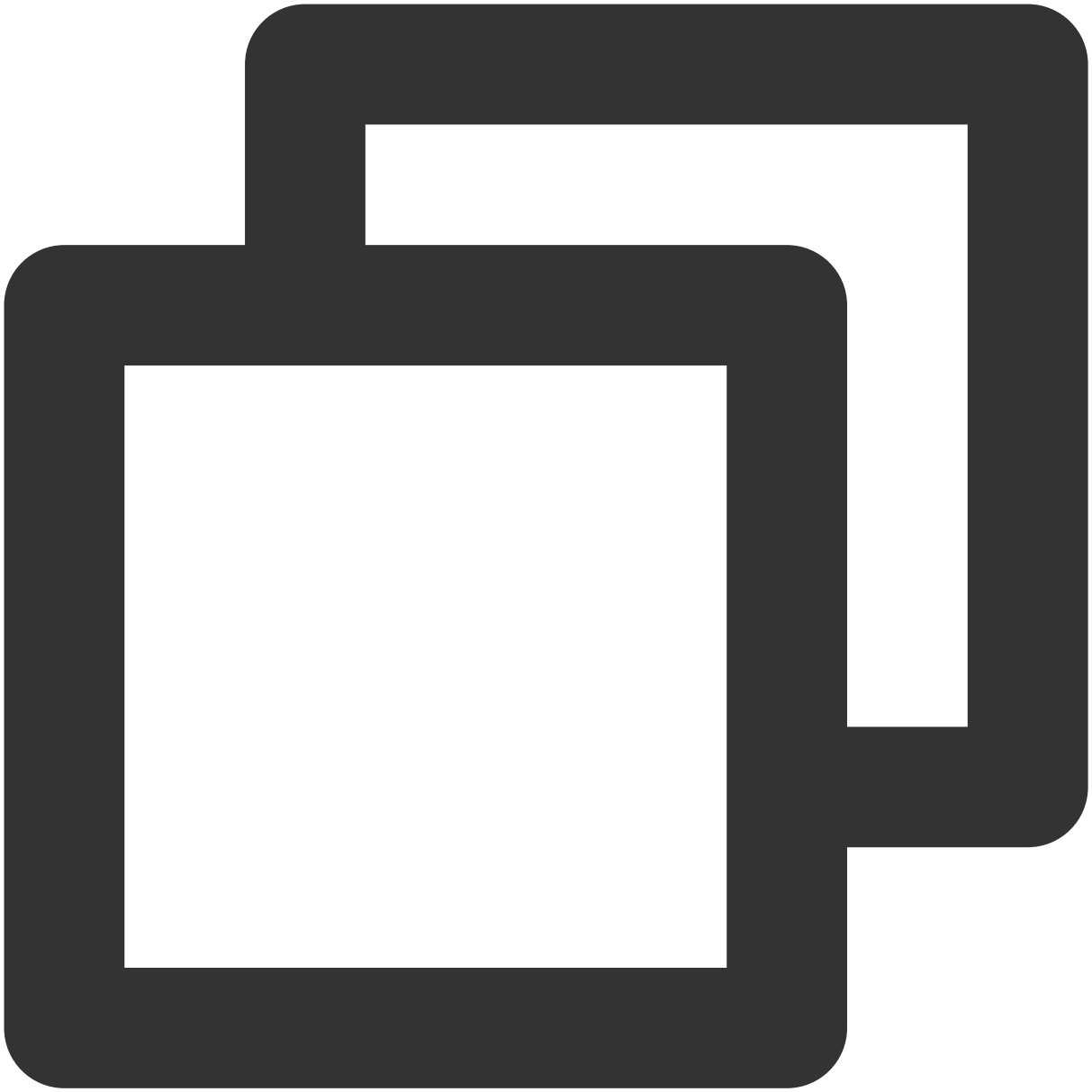
참고

mysqld-exporter를 배포하기 전에 MySQL 또는 MariaDB가 클러스터, 클러스터 외부 또는 사용된 클라우드 서비스에 배포되었는지 확인하십시오.

MySQL 배포

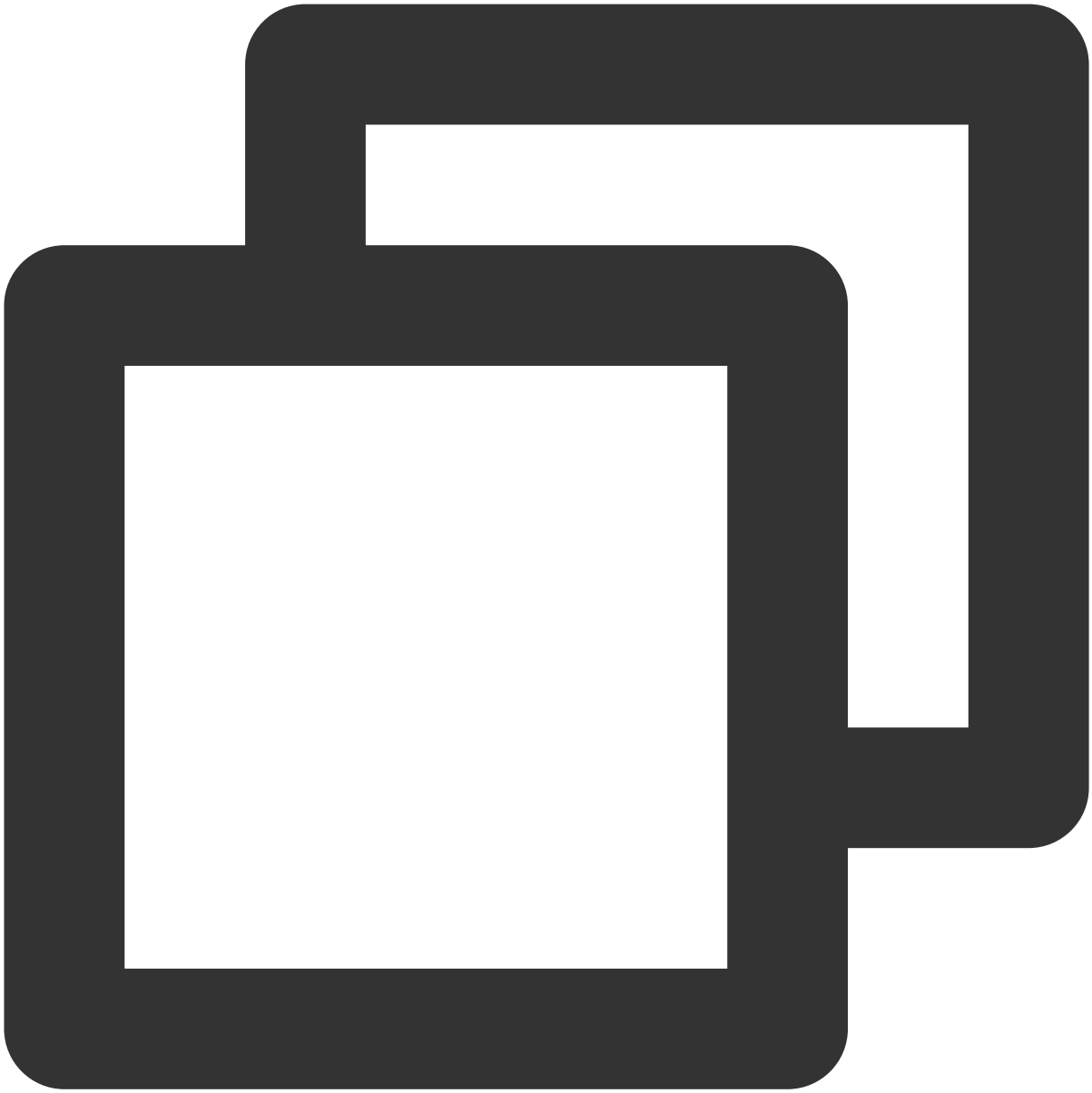
다음 예시는 마켓플레이스에서 MySQL을 클러스터에 배포하는 방법을 보여줍니다.

1. TKE 콘솔에 로그인하고 왼쪽 사이드바에서 **마켓플레이스**를 클릭합니다.
2. **마켓플레이스** 페이지에서 **MySQL**을 검색하고 클릭합니다.
3. **애플리케이션 세부 정보** 페이지에서 **애플리케이션 생성**을 클릭합니다.
4. **애플리케이션 생성** 페이지에서 필요한 정보를 입력하고 **생성**을 클릭합니다.
5. 애플리케이션 생성 후 왼쪽 사이드바에서 **애플리케이션**을 선택하고, **애플리케이션** 페이지에서 애플리케이션 세부 정보를 봅니다.
6. 다음 명령을 실행하여 MySQL이 제대로 실행되는지 확인합니다.



```
$ kubectl get pods
NAME                     READY   STATUS        RESTARTS   AGE
mysql-698b898bf7-4dc5k   1/1     Running       0          11s
```

7. 다음 명령을 실행하여 root 비밀번호를 가져옵니다.

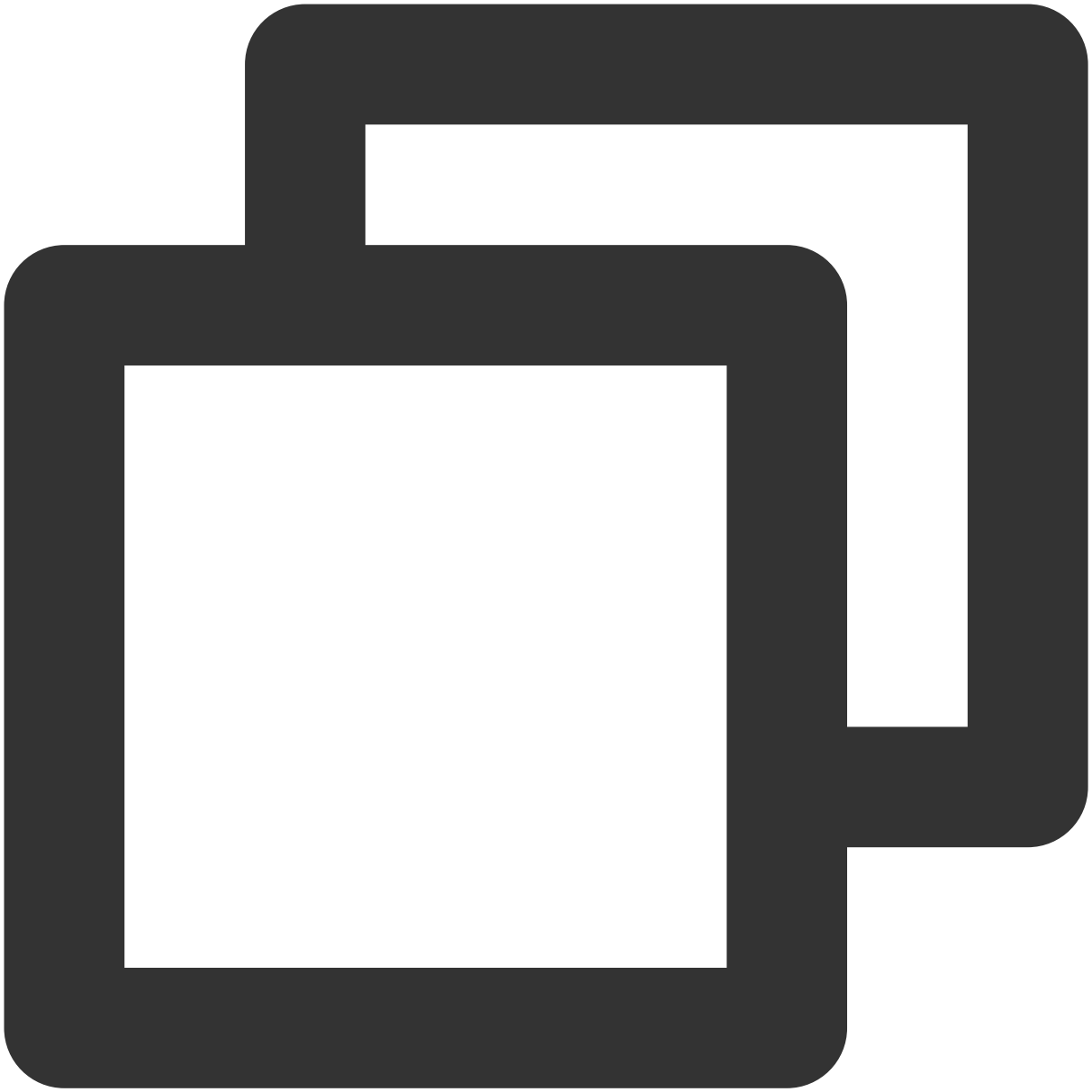


```
$ kubectl get secret -o jsonpath={.data.mysql-root-password} mysql | base64 -d
6ZAj33yLBo
```

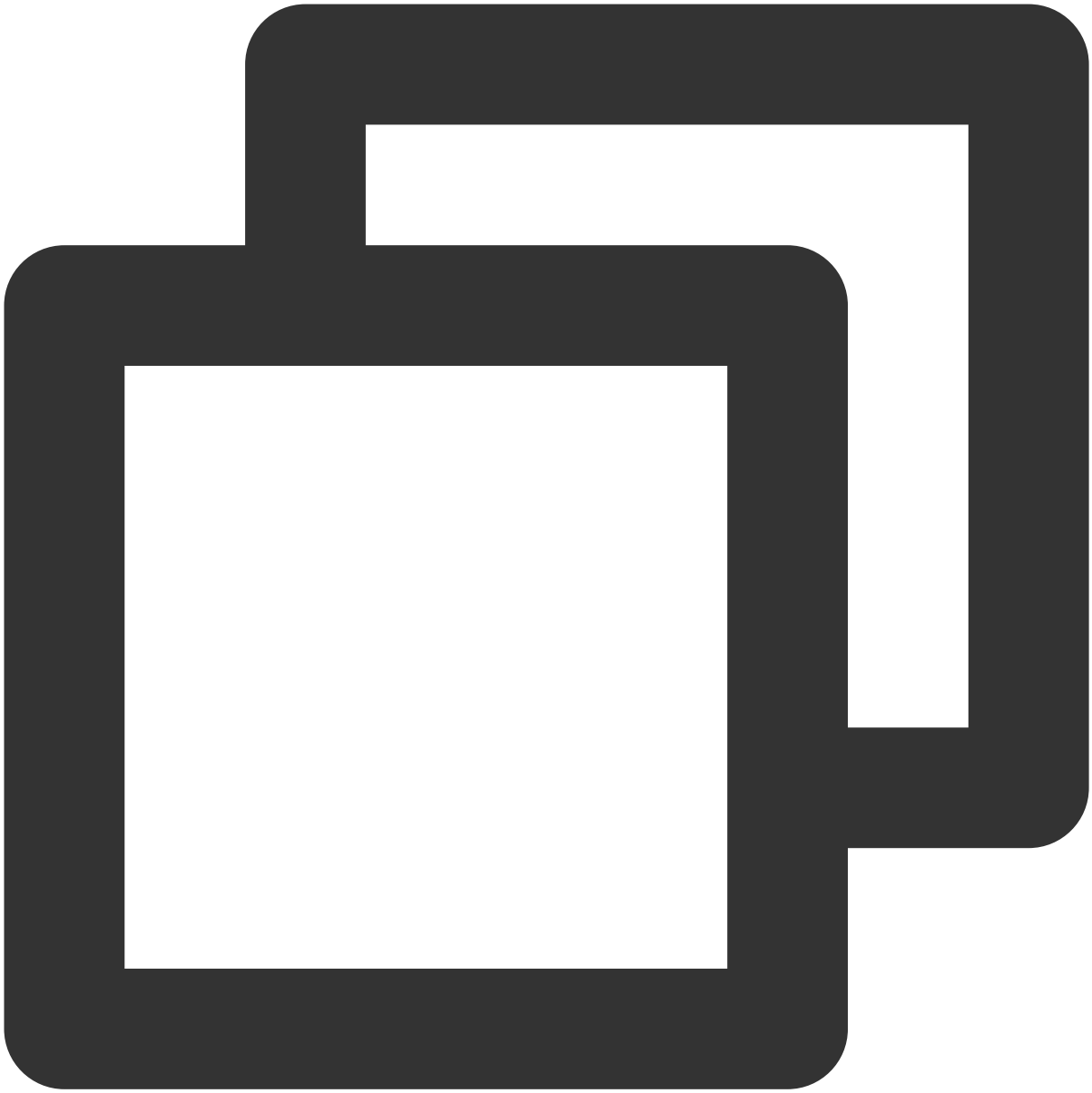
mysqld-exporter 배포

MySQL 배포 후 다음과 같이 mysqld-exporter를 배포합니다.

1. 다음 명령을 순서대로 실행하여 mysqld-exporter 계정을 생성하고 MySQL에 로그인합니다.

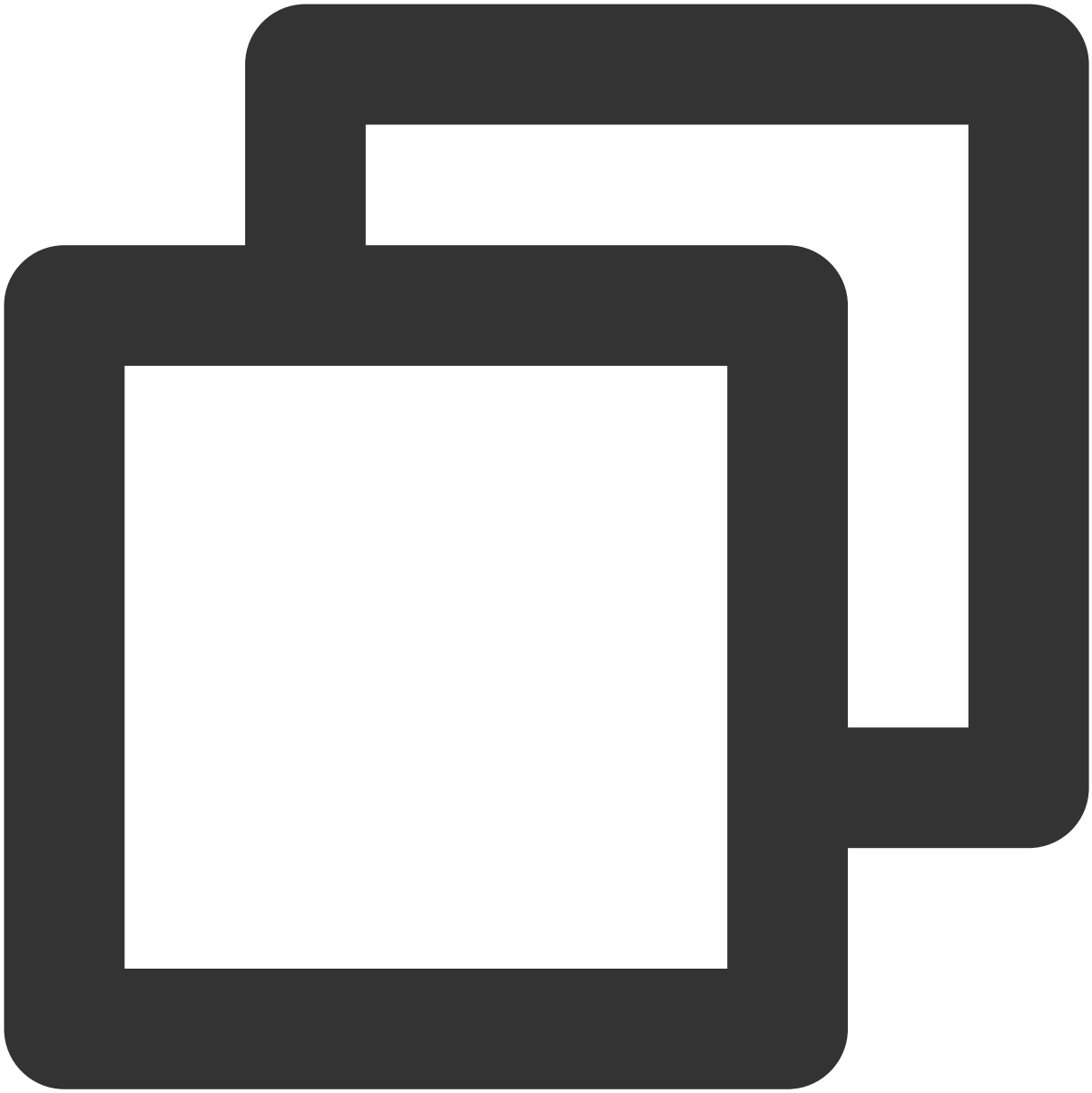


```
$ kubectl exec -it mysql-698b898bf7-4dc5k bash
```



```
$ mysql -uroot -p6ZAj33yLBo
```

2. 다음 명령을 실행하여 계정을 생성합니다. `mysqld-exporter/123456` 이 예시로 사용됩니다.

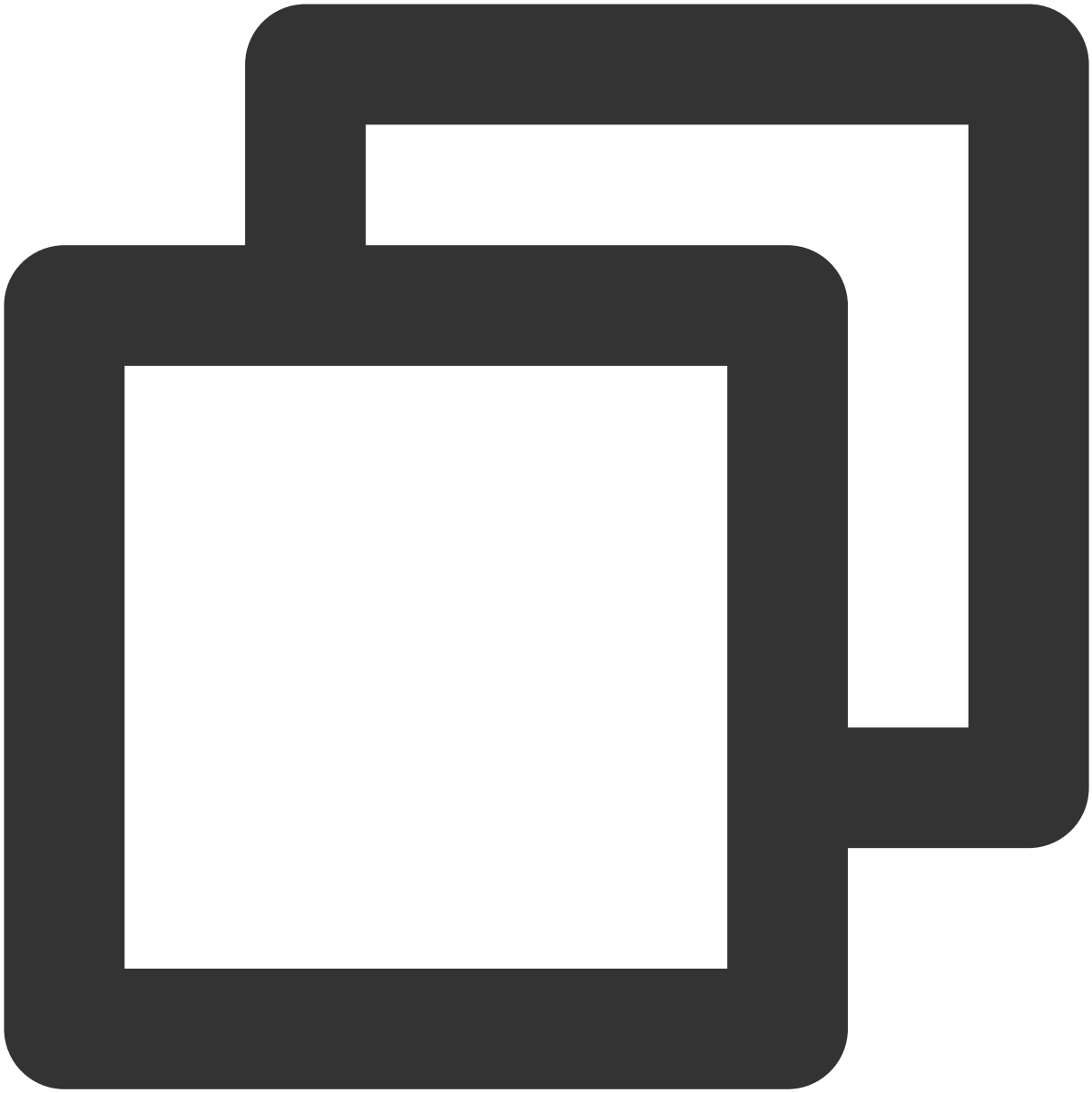


```
CREATE USER 'mysqld-exporter' IDENTIFIED BY '123456' WITH MAX_USER_CONNECTIONS 3;
GRANT PROCESS, REPLICATION CLIENT, REPLICATION SLAVE, SELECT ON *.* TO 'mysqld-expo
flush privileges;
```

3. yaml 파일을 사용하여 mysqld-exporter를 배포합니다. 예시는 다음과 같습니다.

참고

DATA_SOURCE_NAME의 계정, 비밀번호, MySQL 연결 주소를 실제 주소로 교체합니다.



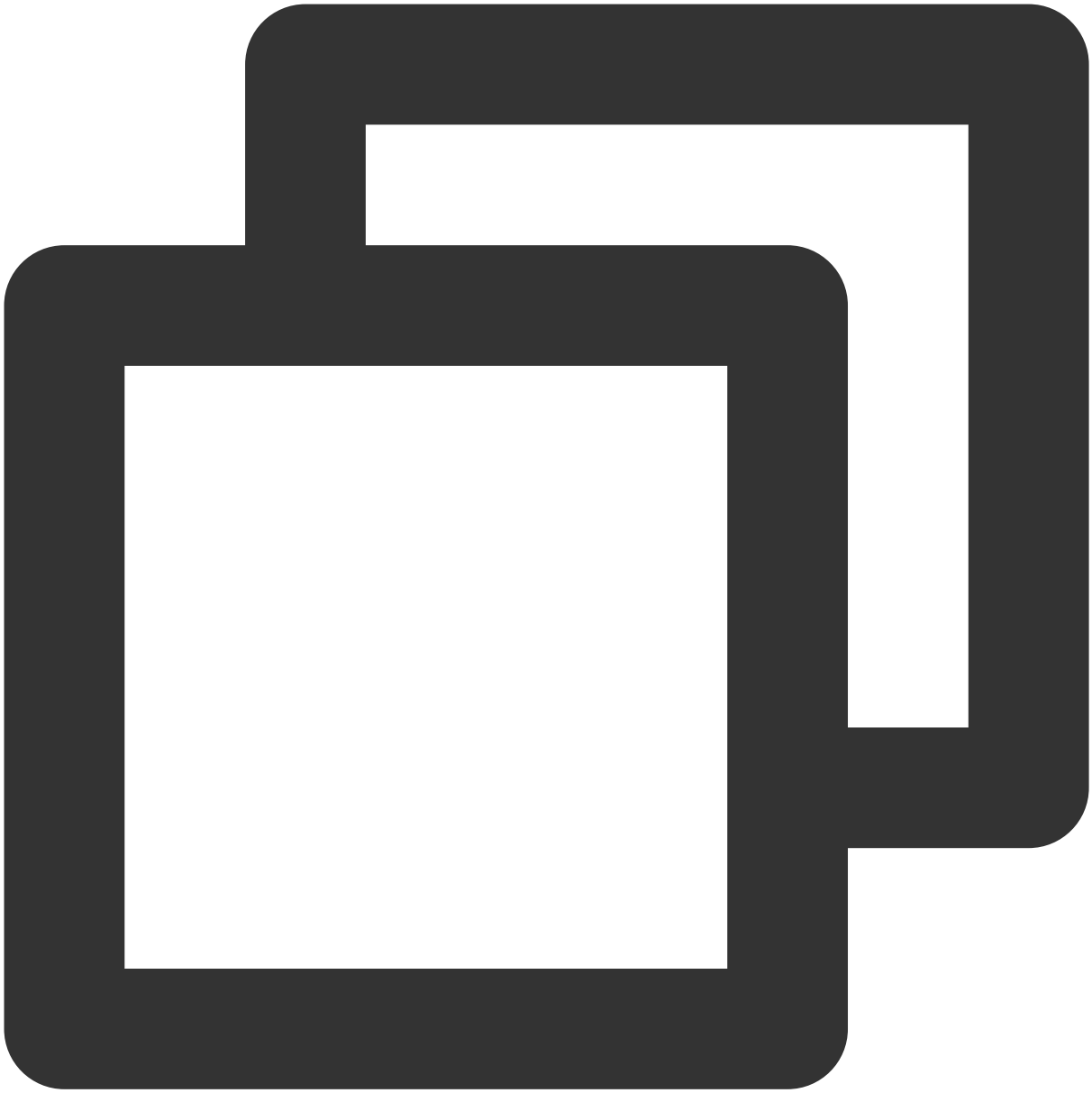
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysqld-exporter
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysqld-exporter
  template:
    metadata:
```

```
      labels:
        app: mysqld-exporter
    spec:
      containers:
      - name: mysqld-exporter
        image: prom/mysqld-exporter:v0.12.1
        args:
        - --collect.info_schema.tables
        - --collect.info_schema.innodb_tablespaces
        - --collect.info_schema.innodb_metrics
        - --collect.global_status
        - --collect.global_variables
        - --collect.slave_status
        - --collect.info_schema.processlist
        - --collect.perf_schema.tablelocks
        - --collect.perf_schema.eventsstatements
        - --collect.perf_schema.eventsstatementssum
        - --collect.perf_schema.eventswaits
        - --collect.auto_increment.columns
        - --collect.binlog_size
        - --collect.perf_schema.tableiowaits
        - --collect.perf_schema.indexiowaits
        - --collect.info_schema.userstats
        - --collect.info_schema.clientstats
        - --collect.info_schema.tablestats
        - --collect.info_schema.schemastats
        - --collect.perf_schema.file_events
        - --collect.perf_schema.file_instances
        - --collect.perf_schema.replication_group_member_stats
        - --collect.perf_schema.replication_applier_status_by_worker
        - --collect.slave_hosts
        - --collect.info_schema.innodb_cmp
        - --collect.info_schema.innodb_cmpmem
        - --collect.info_schema.query_response_time
        - --collect.engine_tokudb_status
        - --collect.engine_innodb_status
        ports:
        - containerPort: 9104
          protocol: TCP
        env:
        - name: DATA_SOURCE_NAME
          value: "mysqld-exporter:123456@(mysql.default.svc.cluster.local:3306)/"
--
apiVersion: v1
kind: Service
metadata:
  name: mysqld-exporter
```

```
    labels:
      app: mysqld-exporter
  spec:
    type: ClusterIP
    ports:
    - port: 9104
      protocol: TCP
      name: http
    selector:
      app: mysqld-exporter
```

모니터링 데이터 수집 구성

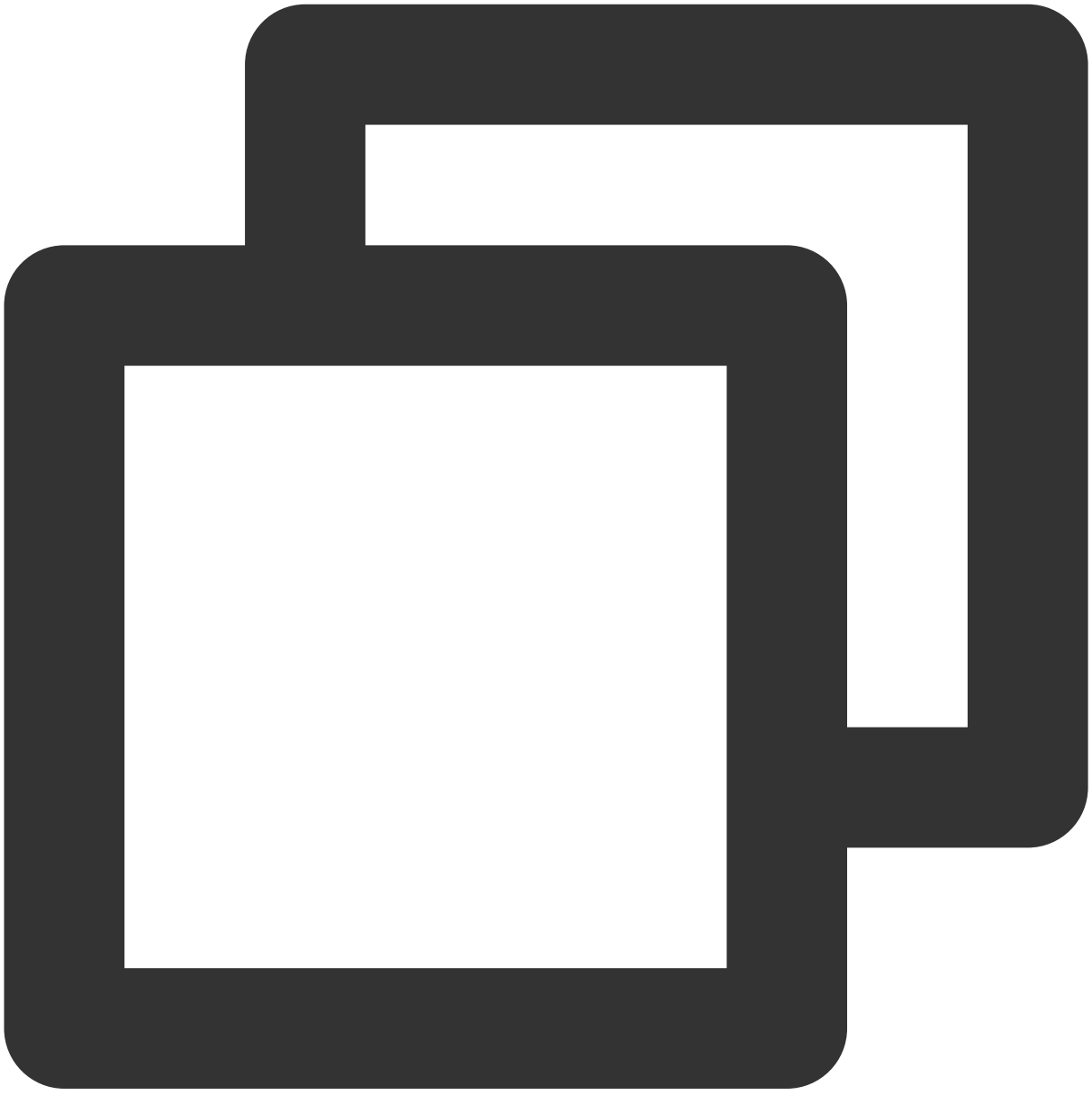
mysqld-exporter 배포 후 mysqld-exporter에서 노출된 데이터를 수집할 수 있도록 모니터링 데이터 수집을 구성합니다. 다음 예시는 ServiceMonitor 정의를 보여줍니다(수집 규칙을 구성하려면 클러스터가 ServiceMonitor 정의를 지원해야 함).



```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: mysqld-exporter
spec:
  endpoints:
    interval: 5s
    targetPort: 9104
  namespaceSelector:
    matchNames:
    - default
```

```
    selector:
      matchLabels:
        app: mysqld-exporter
```

다음 예시는 네이티브 Prometheus 구성을 보여줍니다.

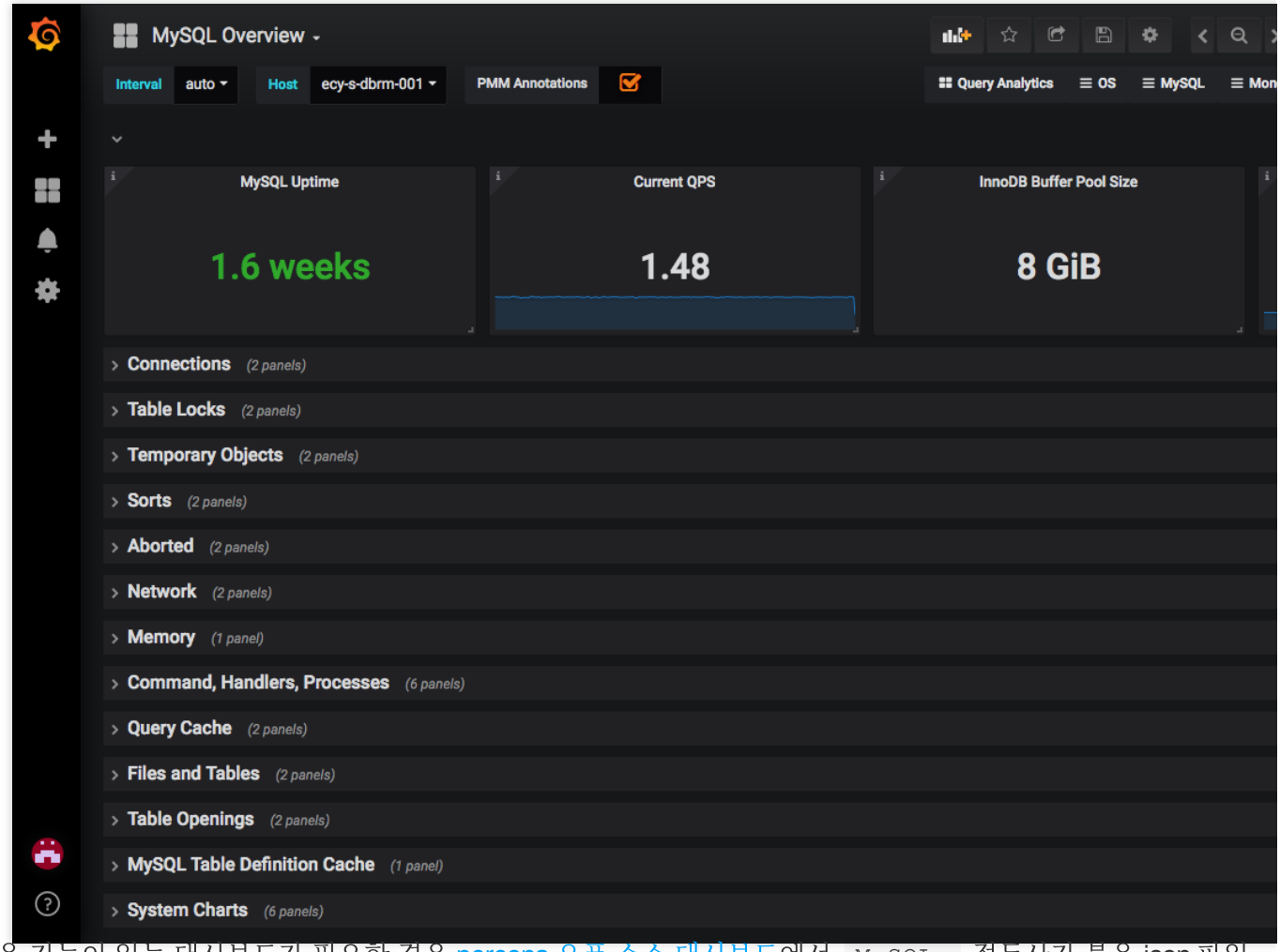


```
    - job_name: mysqld-exporter
      scrape_interval: 5s
      kubernetes_sd_configs:
      - role: endpoints
        namespaces:
          names:
```

```
        - default
    relabel_configs:
    - action: keep
      source_labels:
      - __meta_kubernetes_service_label_app_kubernetes_io_name
      regex: mysqld-exporter
    - action: keep
      source_labels:
      - __meta_kubernetes_endpoint_port_name
      regex: http
```

모니터링 대시보드 추가

데이터가 수집되면 Grafana용 모니터링 대시보드를 추가하여 데이터를 표시합니다.
MySQL 또는 MariaDB 개요 정보만 보려면 아래 이미지와 같이 grafana.com 대시보드를 가져옵니다.



더 많은 기능이 있는 대시보드가 필요한 경우 percona 오픈 소스 대시보드에서 `MySQL_` 접두사가 붙은 json 파일을 가져옵니다.

유지보수
클러스터 감사로 문제 진단

최종 업데이트 날짜: : 2023-04-28 15:30:11

사용 사례

오작동, 애플리케이션 bug 또는 악성 프로그램의 apiserver API 호출 시 클러스터 리소스가 삭제되거나 수정될 수 있습니다. 클러스터 감사 기능을 사용하여 apiserver API 호출 로그를 유지할 수 있습니다. 이러한 방식으로 감사 로그를 검색하고 분석하여 문제의 원인을 찾을 수 있습니다. 본 문서는 문제 해결을 위해 클러스터 감사 기능을 사용하는 방법을 설명합니다.

주의사항

본문은 TKE 클러스터에만 적용됩니다.

전제 조건

TKE 콘솔에서 클러스터 감사 기능을 활성화했습니다. 자세한 내용은 클러스터 감사 활성화를 참고하십시오.

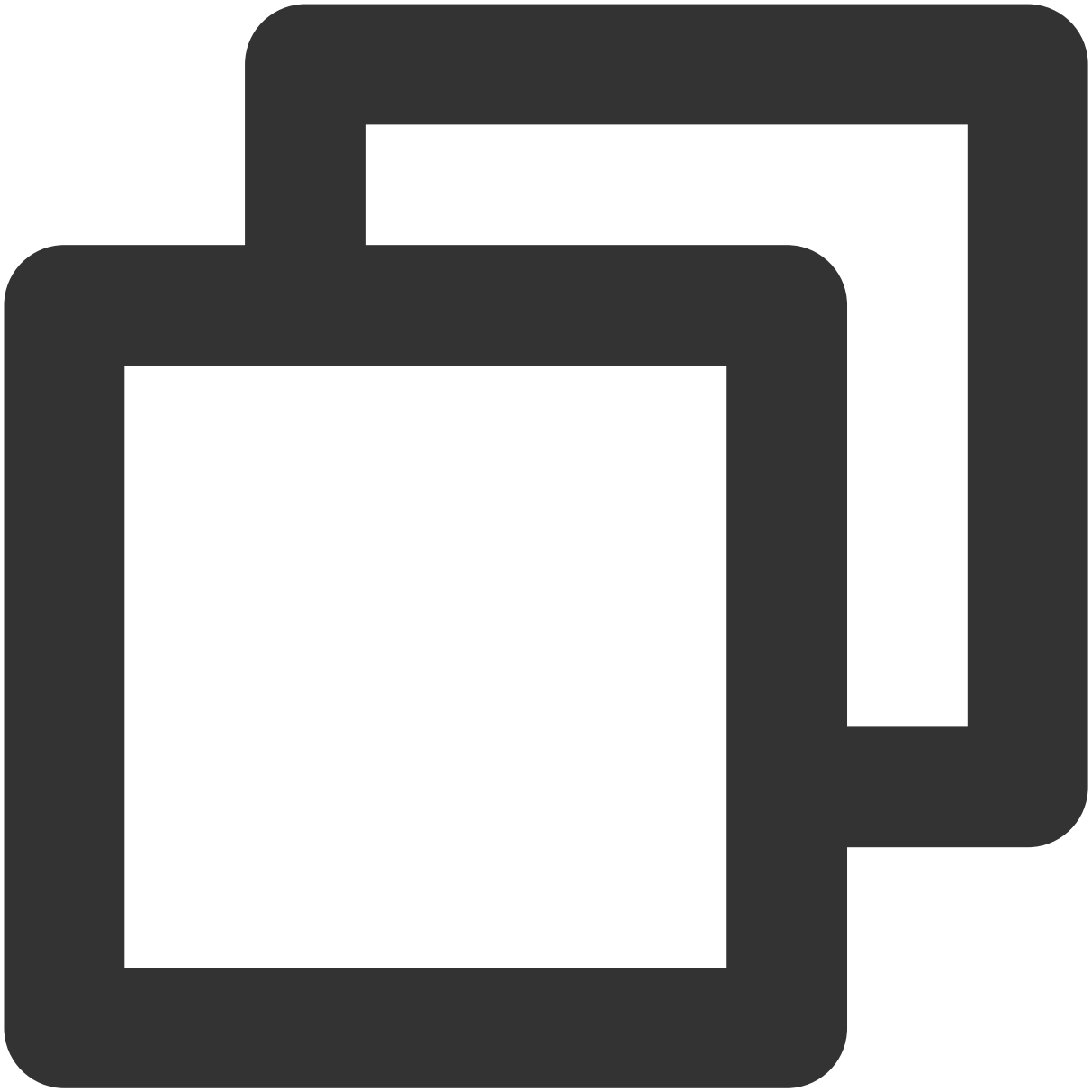
사용 예시

분석 결과 가져오기

1. CLS 콘솔에 로그인하고 왼쪽 사이드바에서 **검색 및 분석**을 선택합니다.
2. **검색 및 분석** 페이지에서 검색할 로그셋 및 로그 테마와 시간 범위를 선택합니다.
3. 분석문을 입력하고 **검색 및 분석**을 클릭하면 분석 결과를 얻을 수 있습니다.

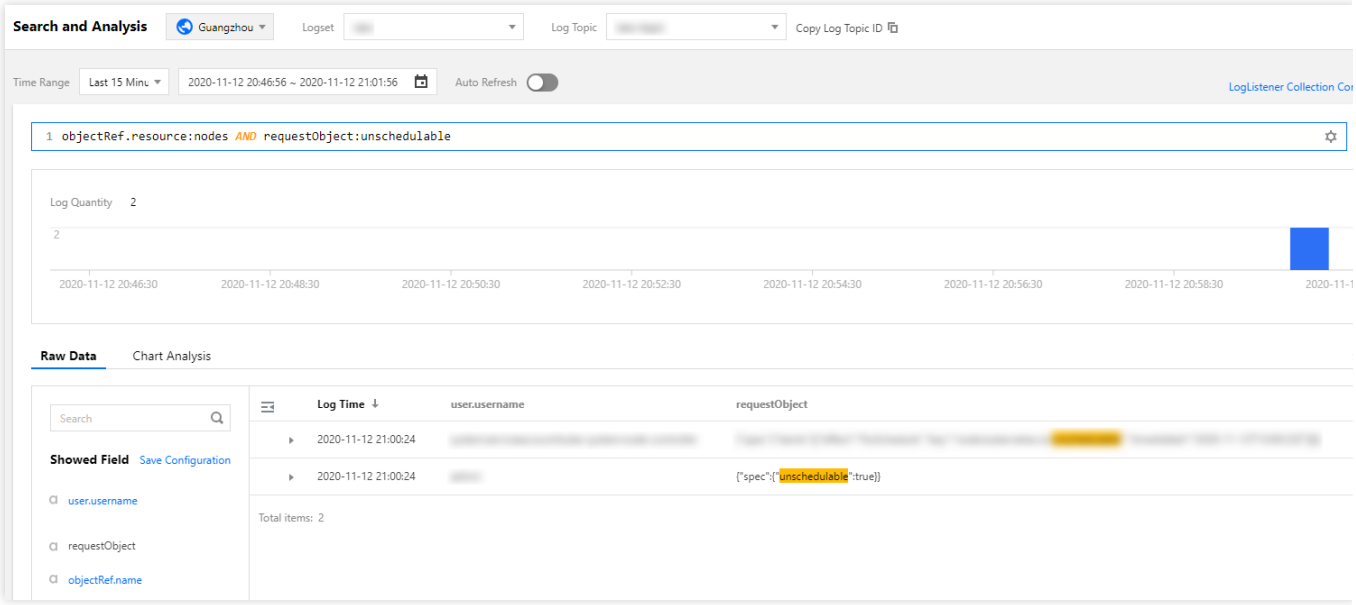
예시1: 노드를 차단한 작업자 쿼리

노드를 차단한 작업자를 조회하려면 다음 명령을 실행합니다.



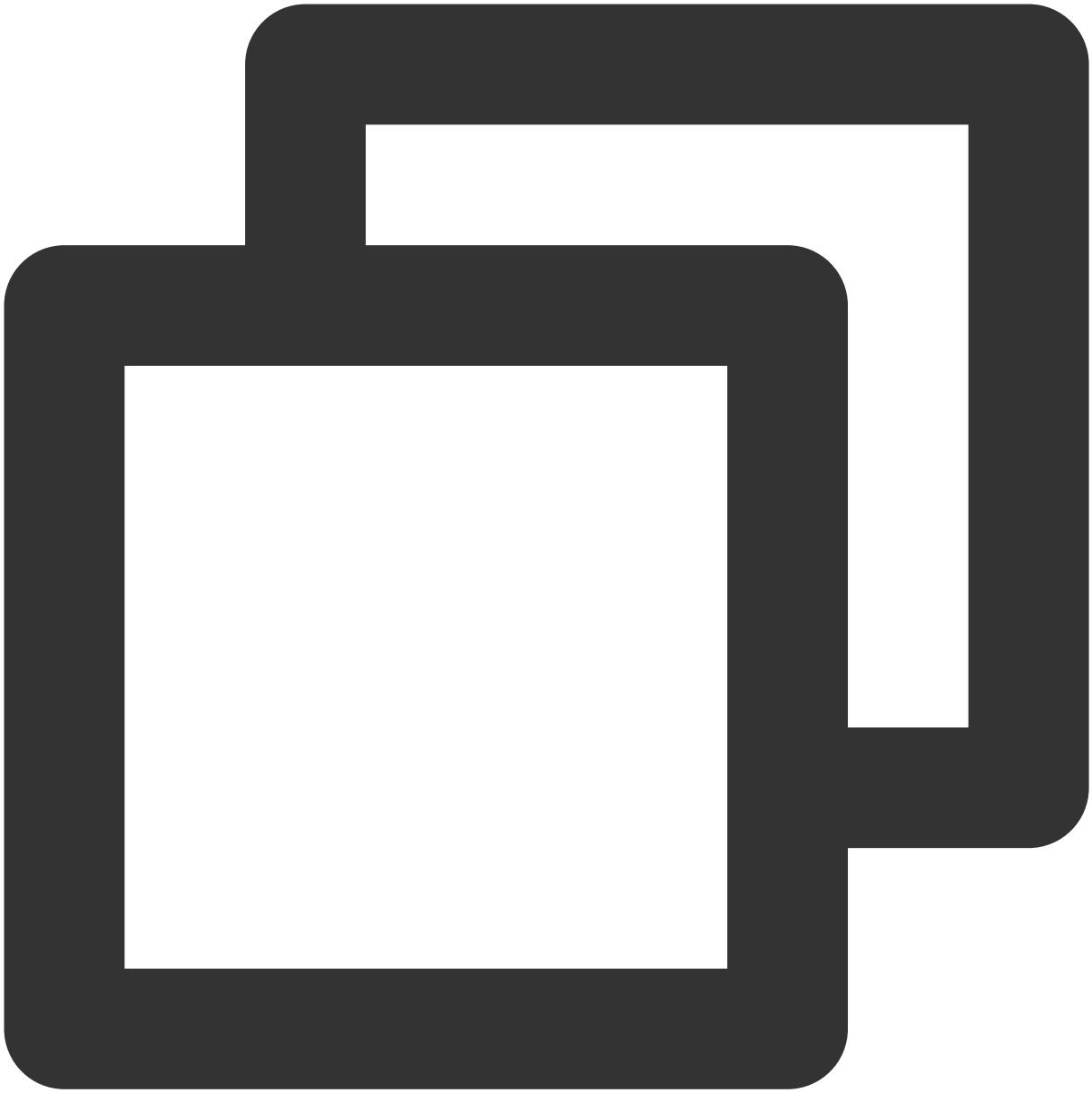
```
objectRef.resource:nodes AND requestObject:unschedulable
```

검색 및 분석 페이지에서 레이아웃을 **기본 구성**으로 선택하면 쿼리 결과가 아래 이미지와 같이 표시됩니다.



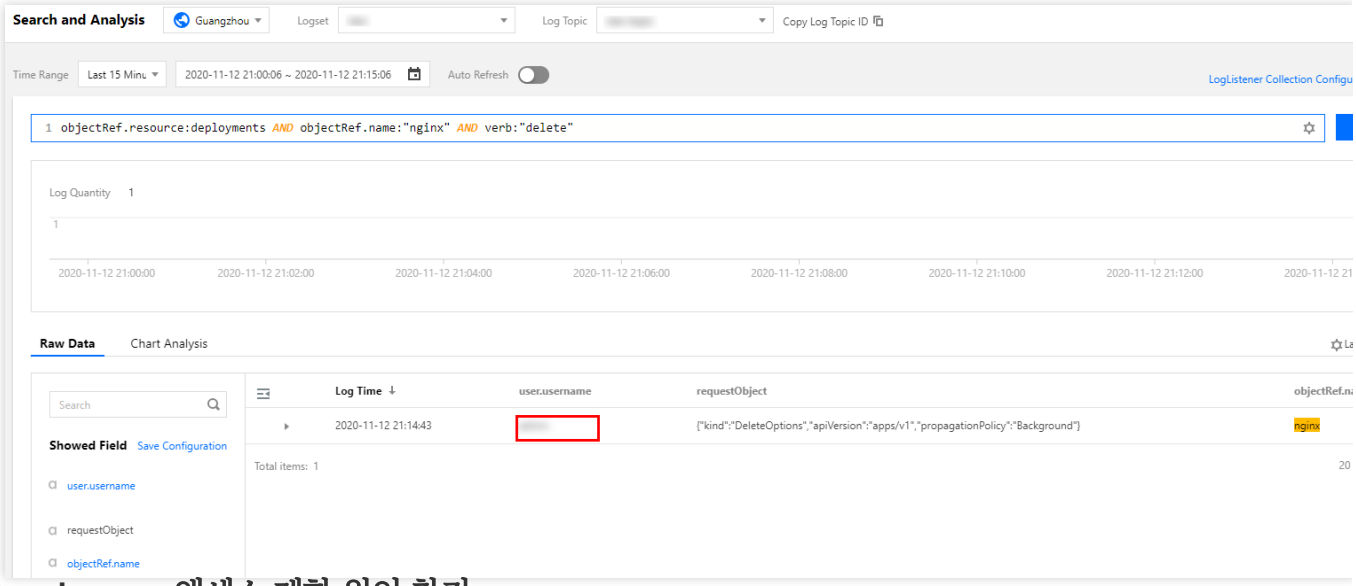
예시2: 워크로드를 삭제한 작업자 쿼리

워크로드를 삭제한 작업자를 조회하려면 다음 명령을 실행합니다.



```
objectRef.resource:deployments AND objectRef.name:"nginx" AND verb:"delete"
```

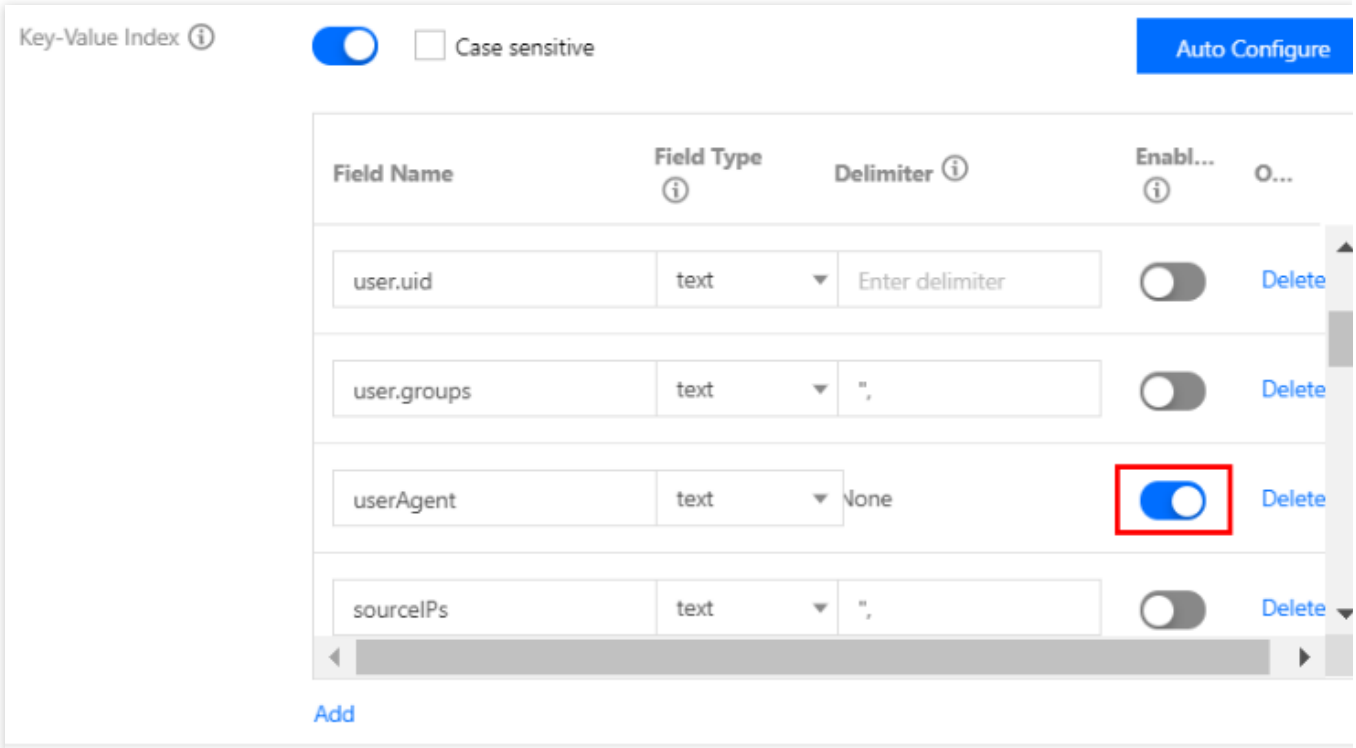
검색 결과를 기반으로 서브 계정에 대한 자세한 정보를 얻을 수 있습니다.



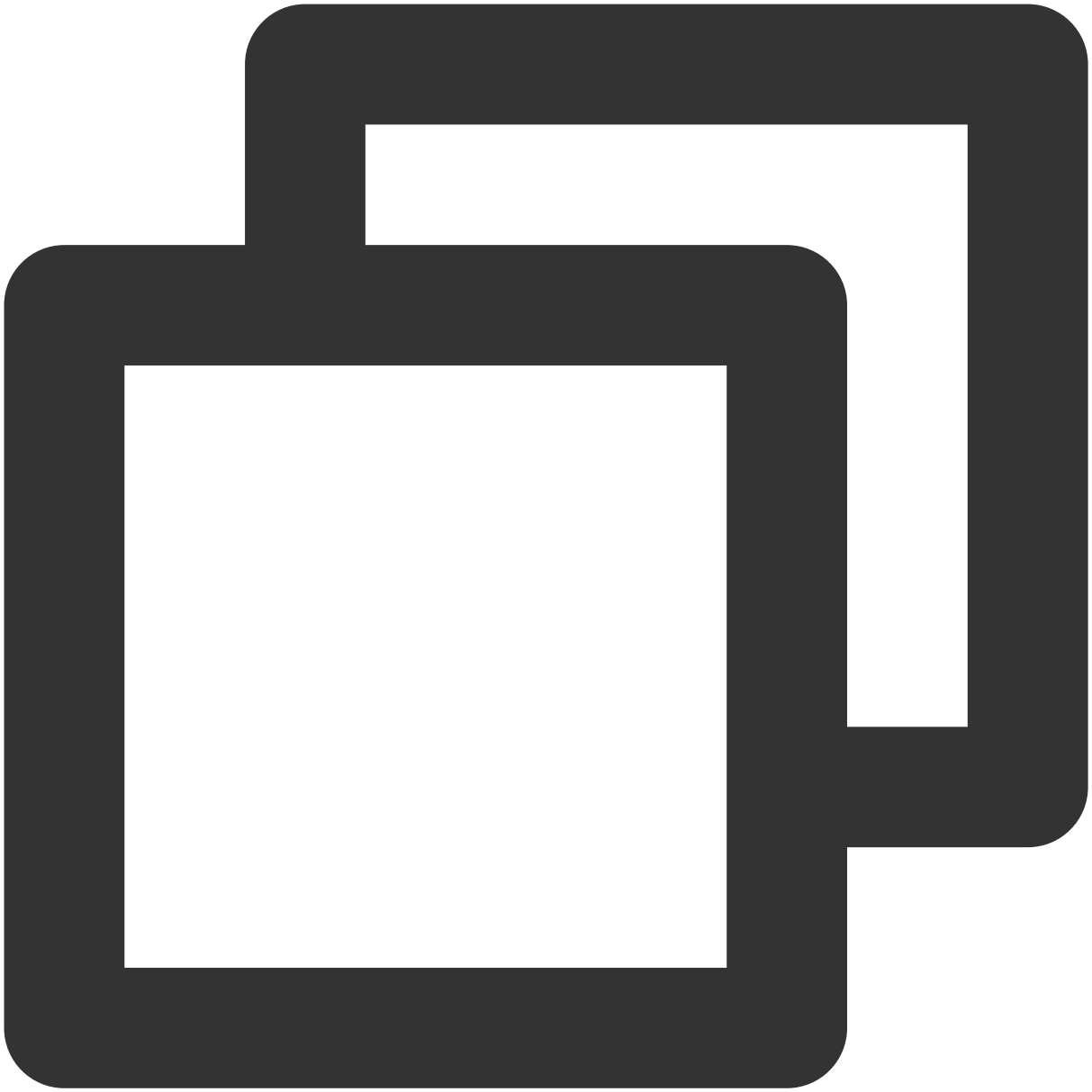
예시3: apiserver 액세스 제한 원인 찾기

악성 프로그램이나 bug로 인한 잦은 apiserver 액세스로 인해 apiserver/etcd가 과부하되는 것을 방지하기 위해, apiserver는 기본적으로 액세스 제한 메커니즘을 활성화합니다. 액세스 제한에 도달하면 감사 로그를 통해 많은 수의 요청을 보낸 클라이언트를 식별할 수 있습니다.

1. userAgent 기반으로 요청을 보내는 클라이언트를 분석하려면 '키-값 인덱스' 창에서 로그 테마를 수정하고 userAgent 필드를 기반으로 통계를 수집하면 됩니다.

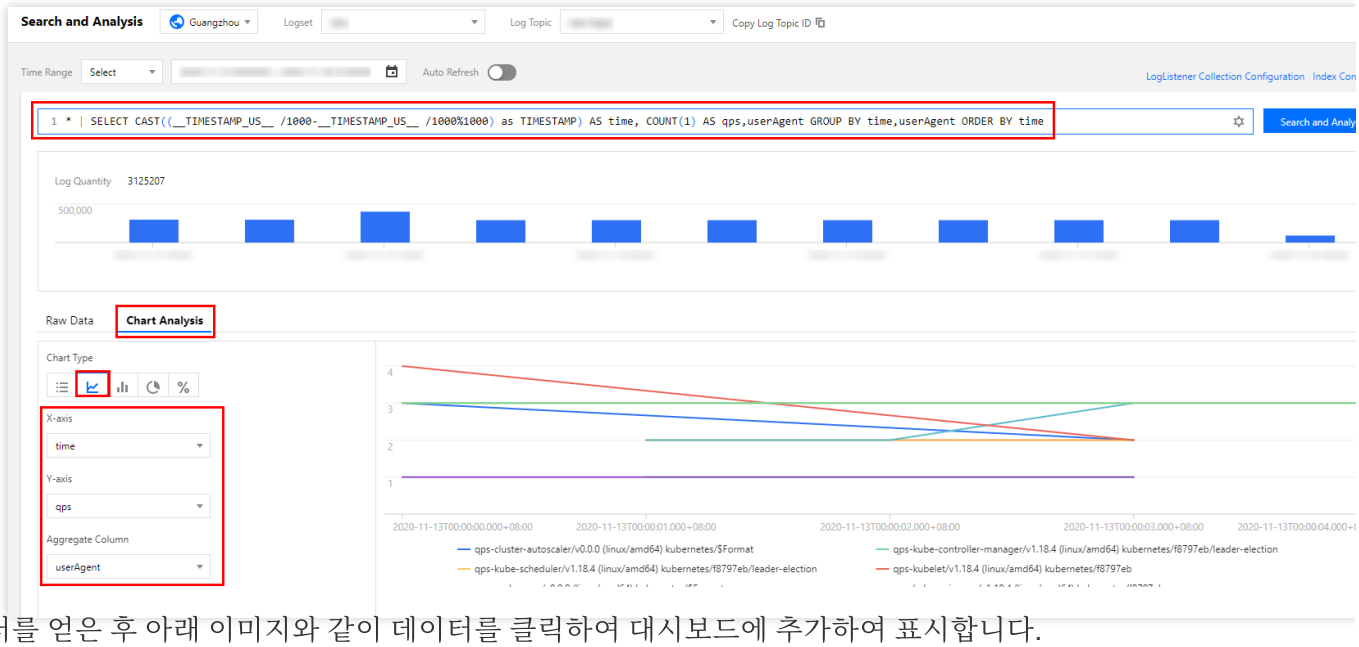


2. 다음 명령을 실행하여 각 클라이언트에서 apiserver로 QPS 통계를 수집합니다.

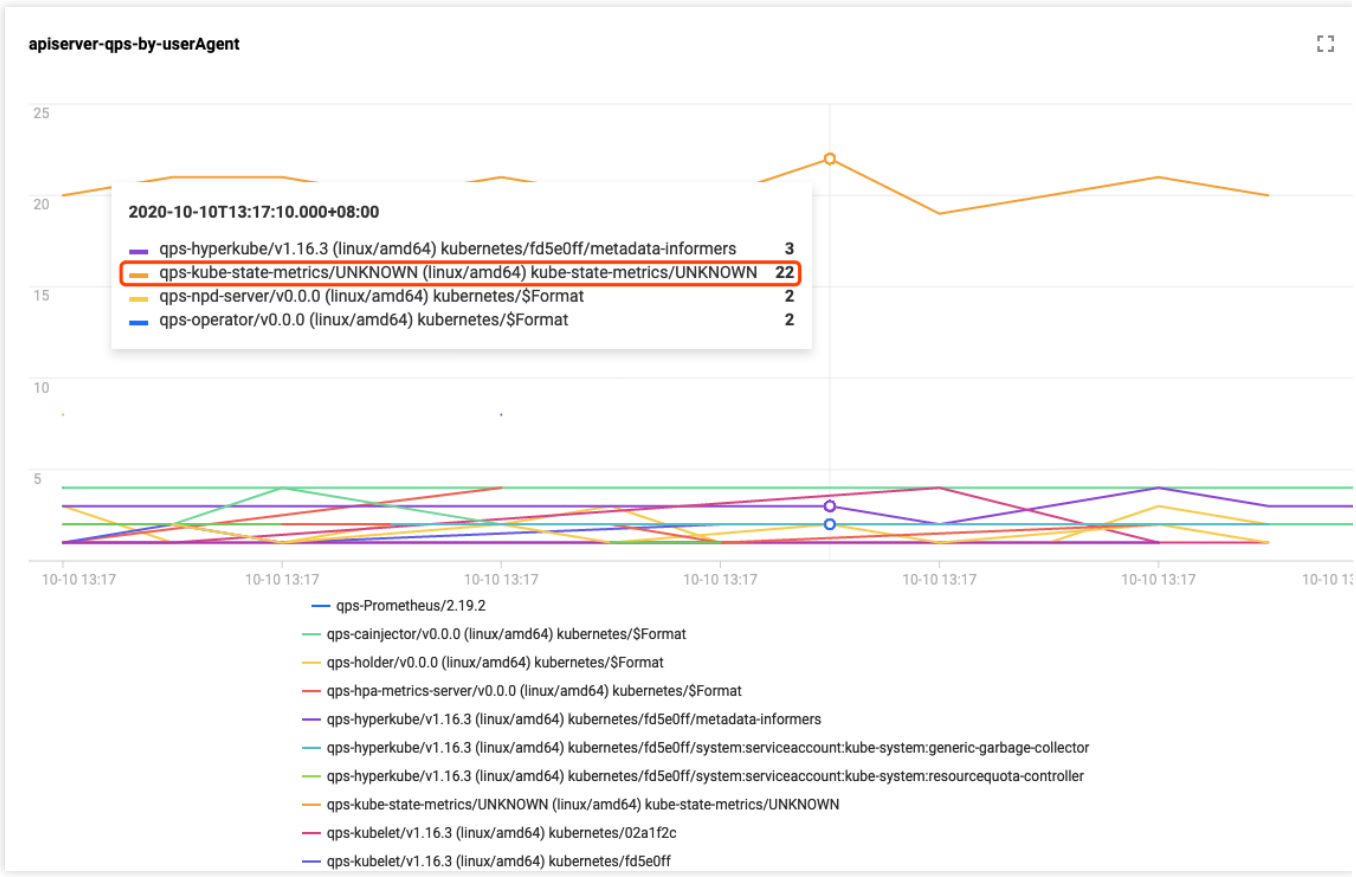


```
* | SELECT histogram( cast(__TIMESTAMP__ as timestamp),interval 1 minute) AS time,
```

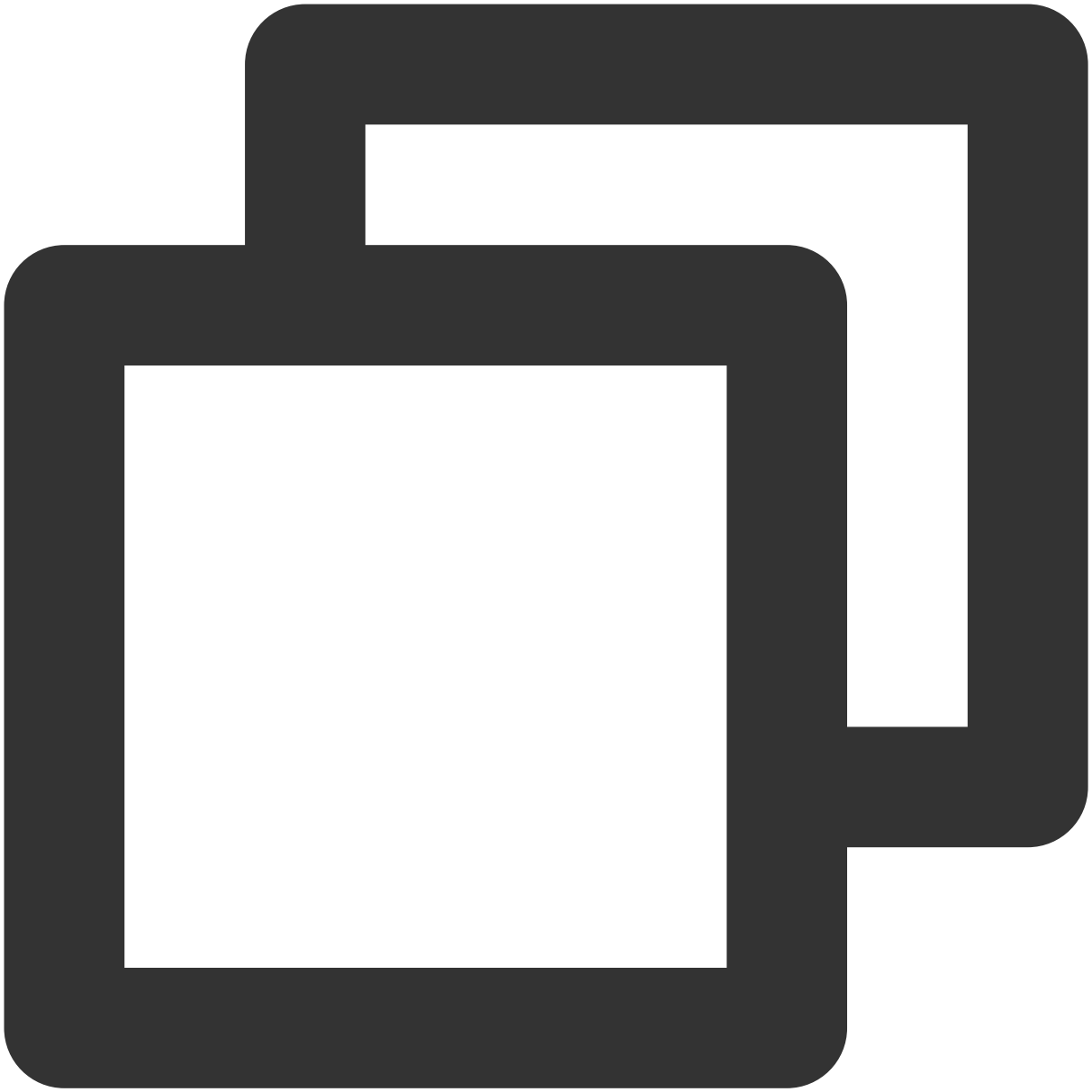
3. 통계 차트로 전환하여 시퀀스 다이어그램을 선택하면 다음 이미지와 같이 기본 정보, 좌표 축 등을 설정할 수 있습니다.



데이터를 얻은 후 아래 이미지와 같이 데이터를 클릭하여 대시보드에 추가하여 표시합니다.

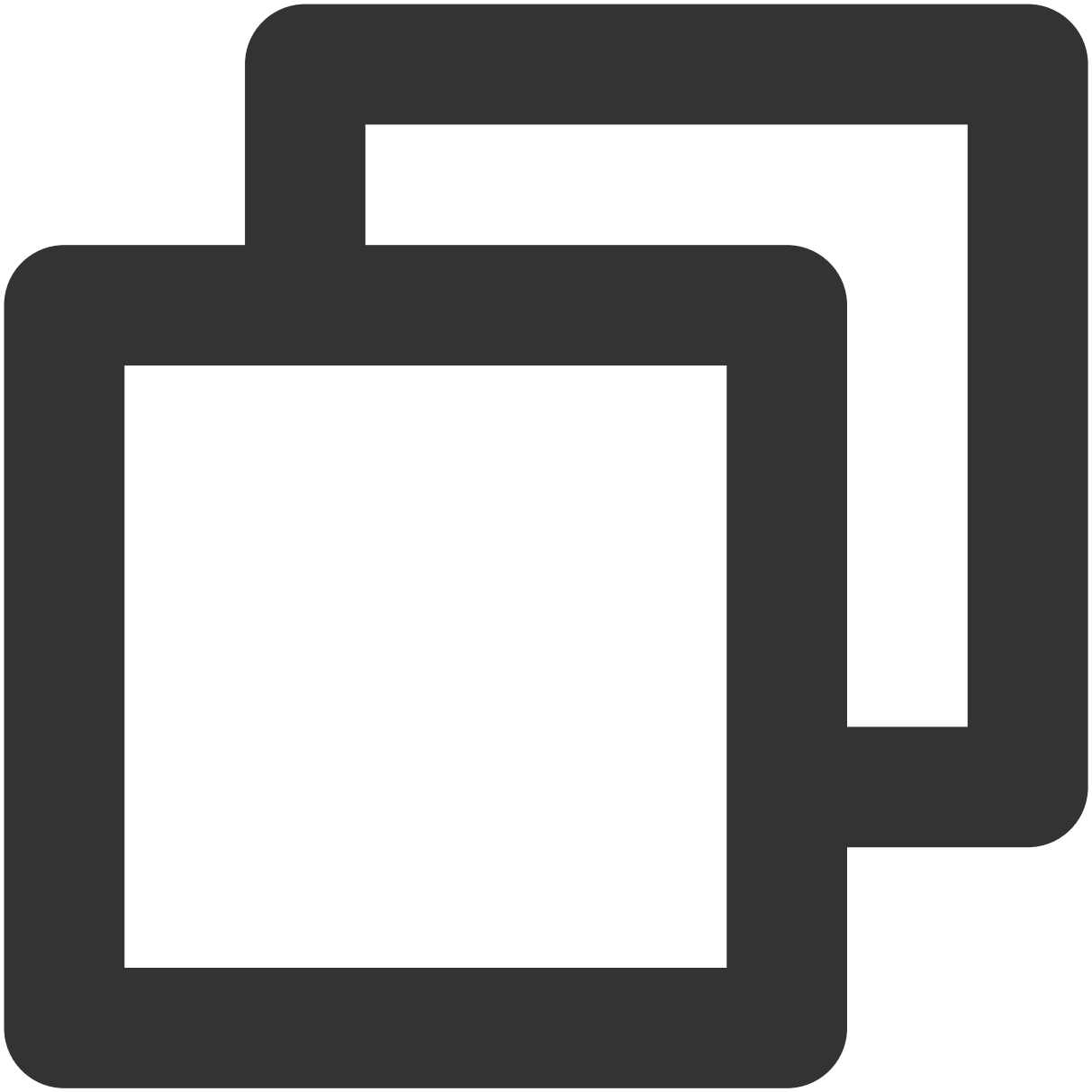


이미지는 kube-state-metrics 클라이언트에서 apiserver로의 요청 빈도가 다른 클라이언트보다 훨씬 높다는 것을 보여줍니다. 로그에 따르면 kube-state-metrics는 RBAC 권한 문제로 인해 apiserver에 요청을 계속 보내며, 결과적으로 apiserver 액세스 제한이 트리거됩니다. 로그는 다음과 같습니다.



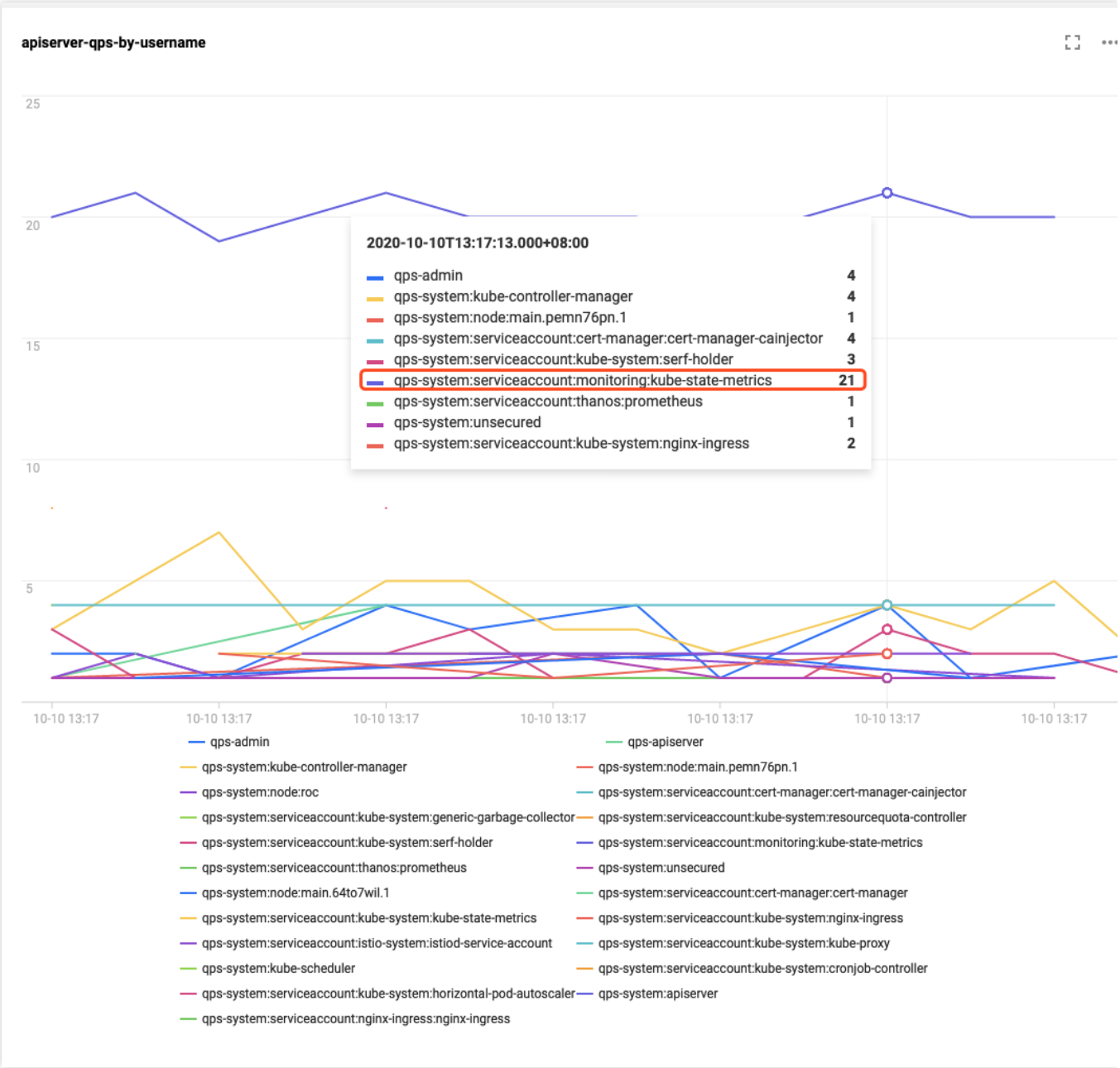
```
I1009 13:13:09.760767       1 request.go:538] Throttling request took 1.393921018s,
E1009 13:13:09.766106       1 reflector.go:156] pkg/mod/k8s.io/client-go@v0.0.0-201
```

user.username과 같은 다른 필드를 사용하여 데이터를 수집할 클라이언트를 구분하려면 필요에 따라 SQL 문을 수정할 수 있습니다. SQL문의 예시는 다음과 같습니다.



```
* | SELECT histogram( cast(__TIMESTAMP__ as timestamp),interval 1 minute)  AS time,
```

다음 이미지는 표시 결과를 보여줍니다.



관련 문서

TKE 클러스터 감사 기능 및 기본 작업에 대한 자세한 내용은 클러스터 감사를 참고하십시오.
클러스터 감사 데이터는 CLS에 저장됩니다. CLS 콘솔에서 감사 결과를 검색하고 분석하려면 검색 구문에 대한 구문 및 규칙을 참고하십시오.
감사 데이터를 분석하기 위해서는 CLS에서 지원하는 SQL 문이 필요합니다. 자세한 내용은 개요를 참고하십시오.

DevOps
TKE 기반 Jenkins 공중망 아키텍처 애플리케이션 구축 및 배포
1단계: TKE 클러스터 및 Jenkins 구성

최종 업데이트 날짜: : 2023-04-28 11:08:19

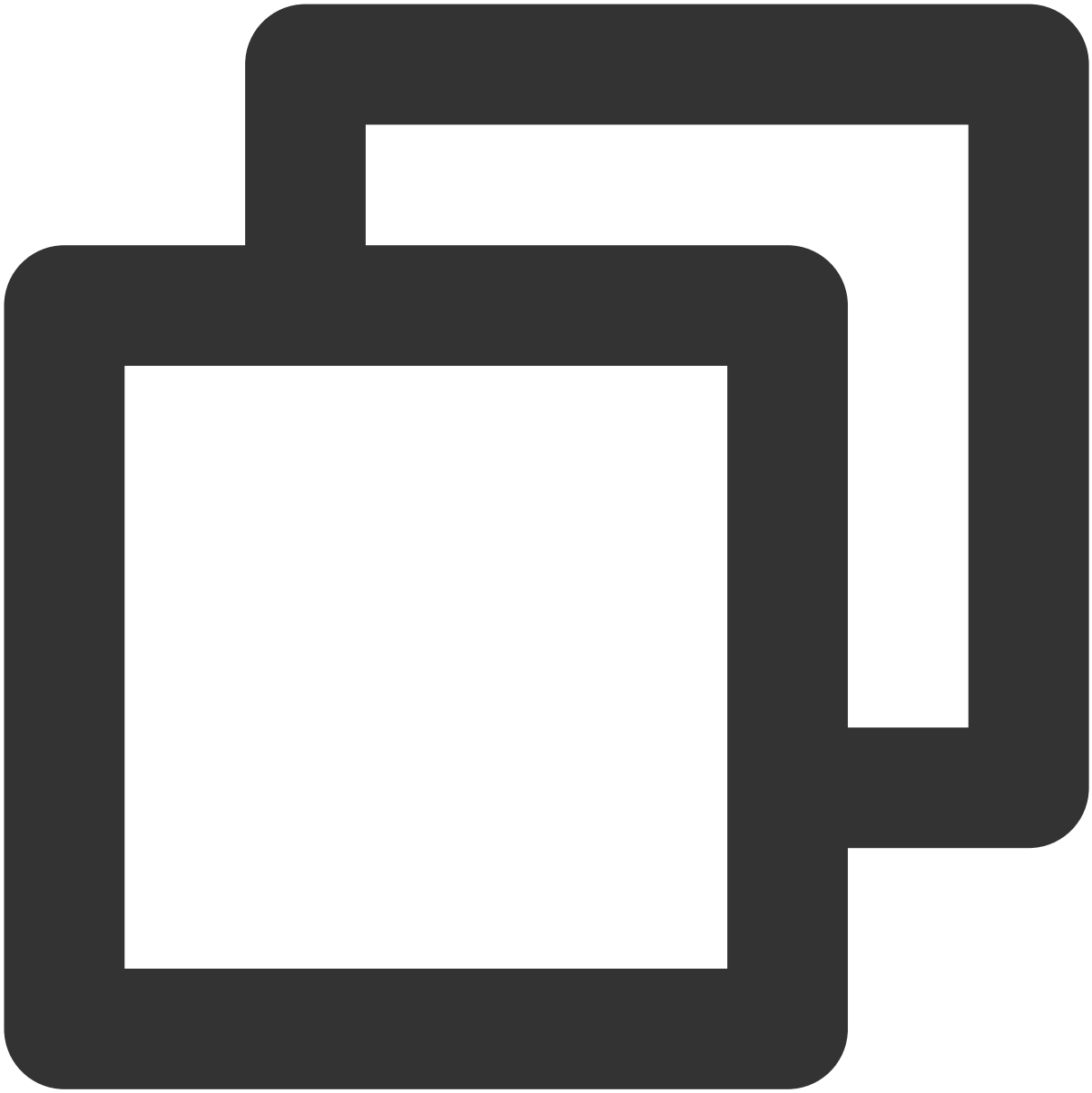
TKE 클러스터 구성

본문은 TKE에서 RBAC 인증 ServiceAccount 사용자 지정 방법과 Jenkins 구성 중에 필요한 클러스터 액세스 주소, token 및 클러스터 CA 인증서 정보를 가져오는 방법을 설명합니다.

클러스터 자격 증명 가져오기

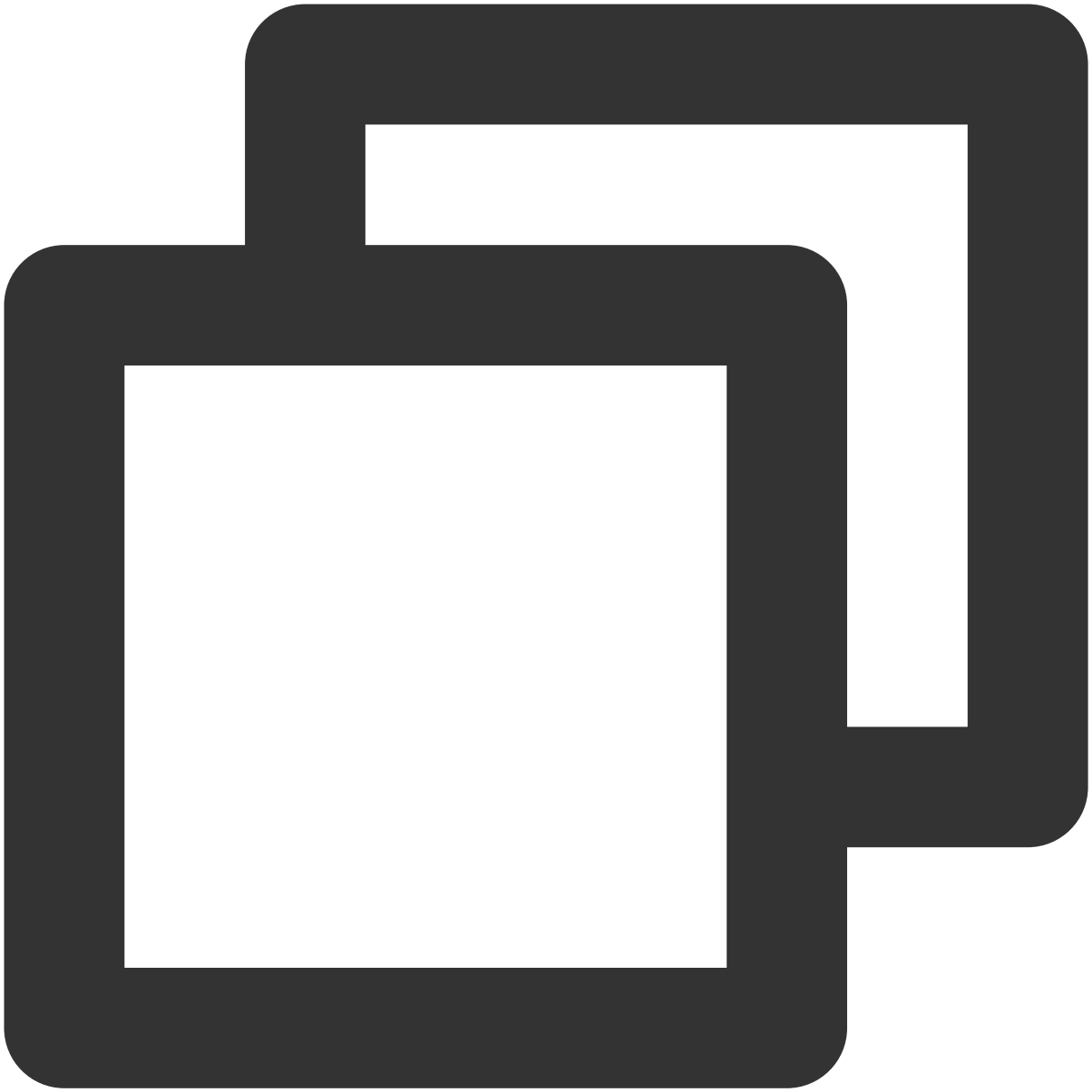
설명

현재 클러스터에서 사설망 액세스를 활성화해야 합니다. 자세한 내용은 Service 콘솔 작업 가이드를 참고하십시오.
1. 다음 Shell 스크립트를 사용하여 ServiceAccount 유형의 테스트 네임스페이스 ci 및 테스트 사용자 jenkins를 생성하고 아래와 같이 클러스터 액세스 자격 증명(token)을 가져옵니다.



```
# 테스트 네임스페이스 ci 생성
kubectl create namespace ci
# 테스트 ServiceAccount 계정 생성
kubectl create sa jenkins -n ci
# ServiceAccount 계정에 의해 자동으로 생성된 Secret token 가져오기
kubectl get secret $(kubectl get sa jenkins -n ci -o jsonpath={.secrets[0].name}) -
```

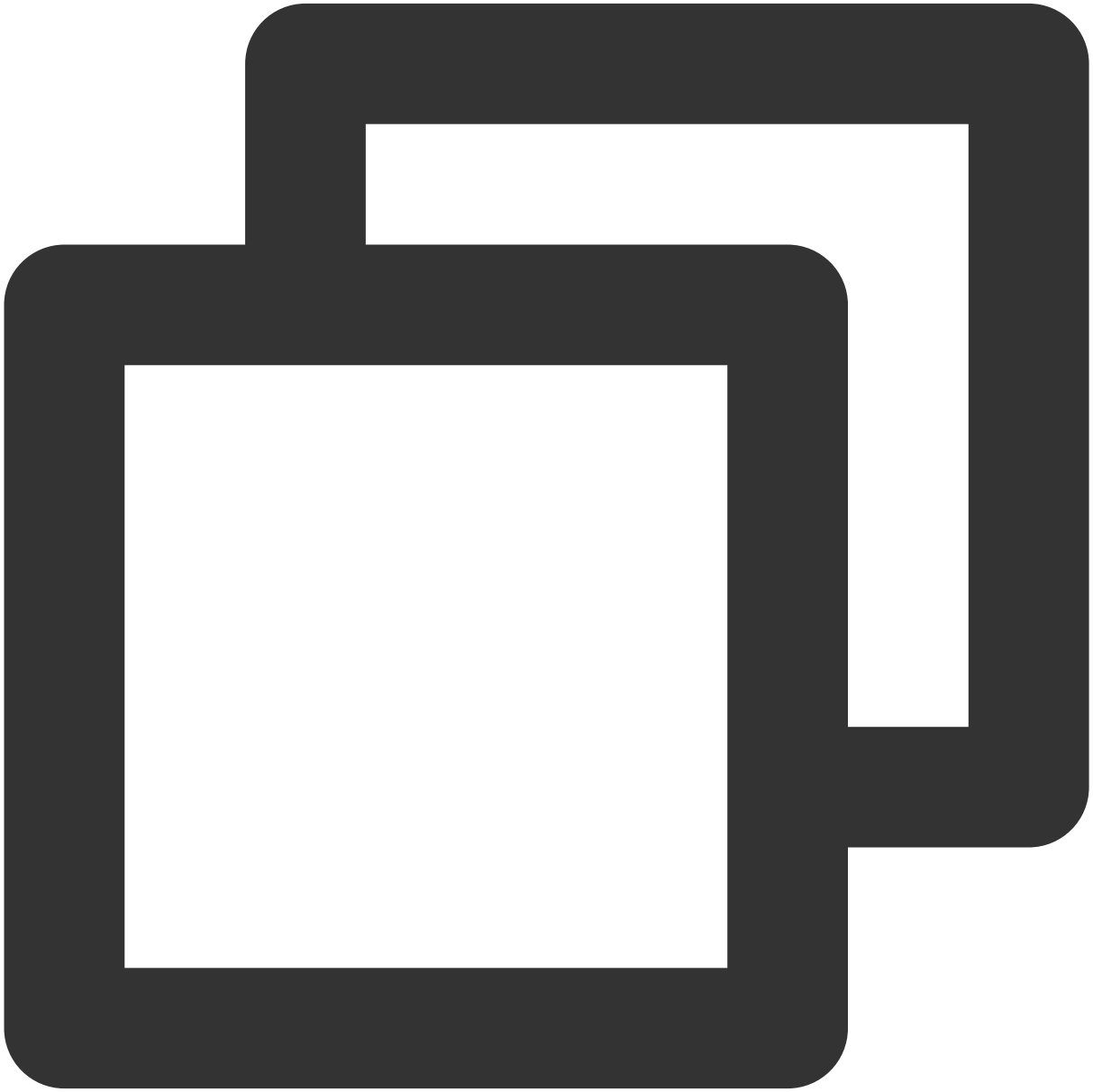
2. 다음과 같이 ci 테스트 네임스페이스에 Role 권한 객체 리소스 파일 jenkins-role.yaml을 생성합니다.



```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: jenkins
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["create","delete","get","list","patch","update","watch"]
- apiGroups: [""]
  resources: ["pods/exec"]
  verbs: ["create","delete","get","list","patch","update","watch"]
```

```
- apiGroups: [""]
  resources: ["pods/log"]
  verbs: ["get","list","watch"]
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get"]
```

3. RoleBinding 객체 리소스 파일 jenkins-rolebinding.yaml을 생성합니다. 다음 권한 바인딩은 ServiceAccount 유형의 jenkins 사용자가 아래와 같이 ci 네임스페이스에서 jenkins(Role 유형) 권한을 가지고 있음을 나타냅니다.

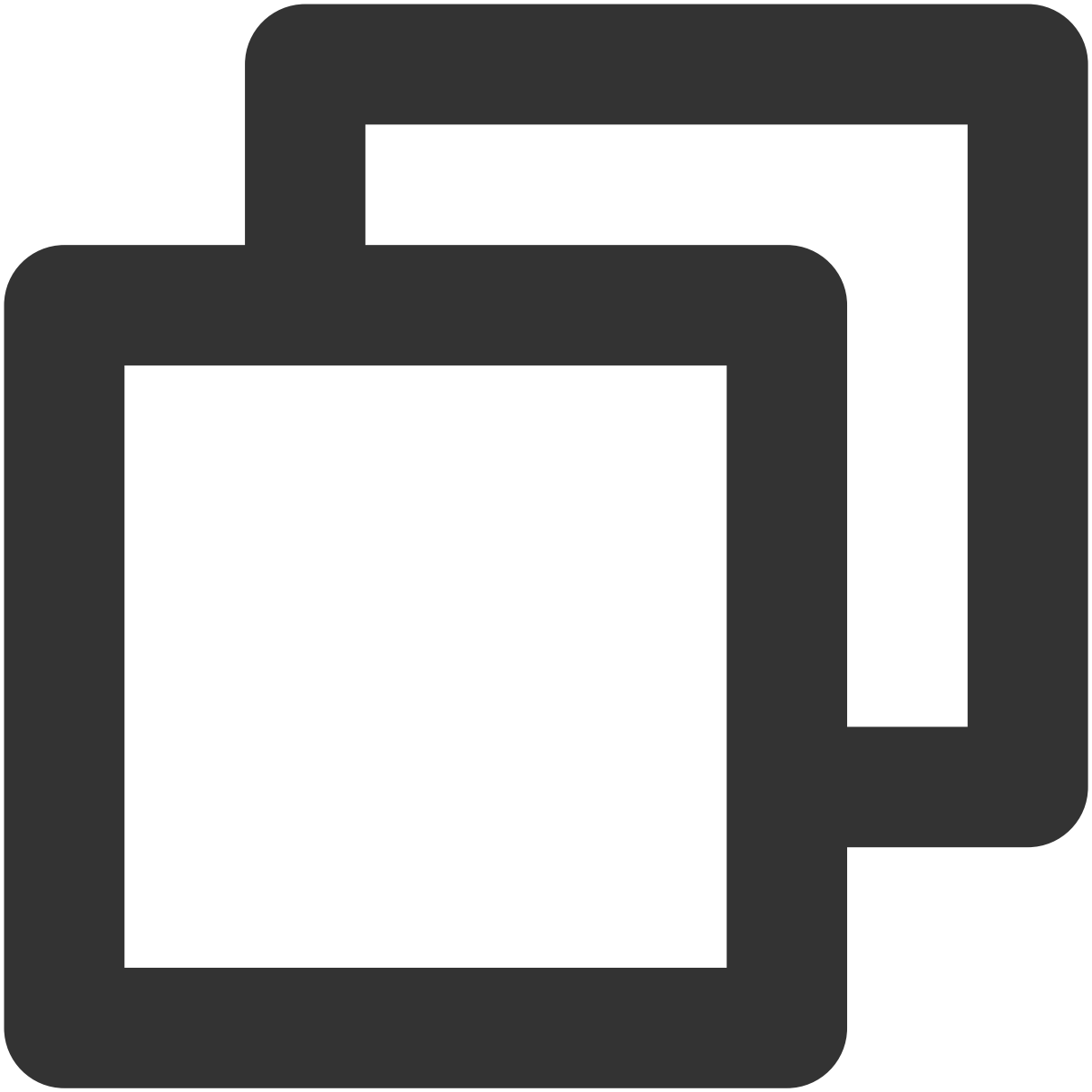


```
apiVersion: rbac.authorization.k8s.io/v1beta1
```

```
kind: RoleBinding
metadata:
  name: jenkins
  namespace: ci
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: jenkins
subjects:
- kind: ServiceAccount
  name: jenkins
```

클러스터 CA 인증서 가져오기

1. 표준 로그인 방법을 사용하여 Linux 인스턴스에 로그인(권장)에 설명된 대로 클러스터의 node에 로그인합니다.
2. 클러스터 CA 인증서를 보려면 다음 명령을 실행하십시오.

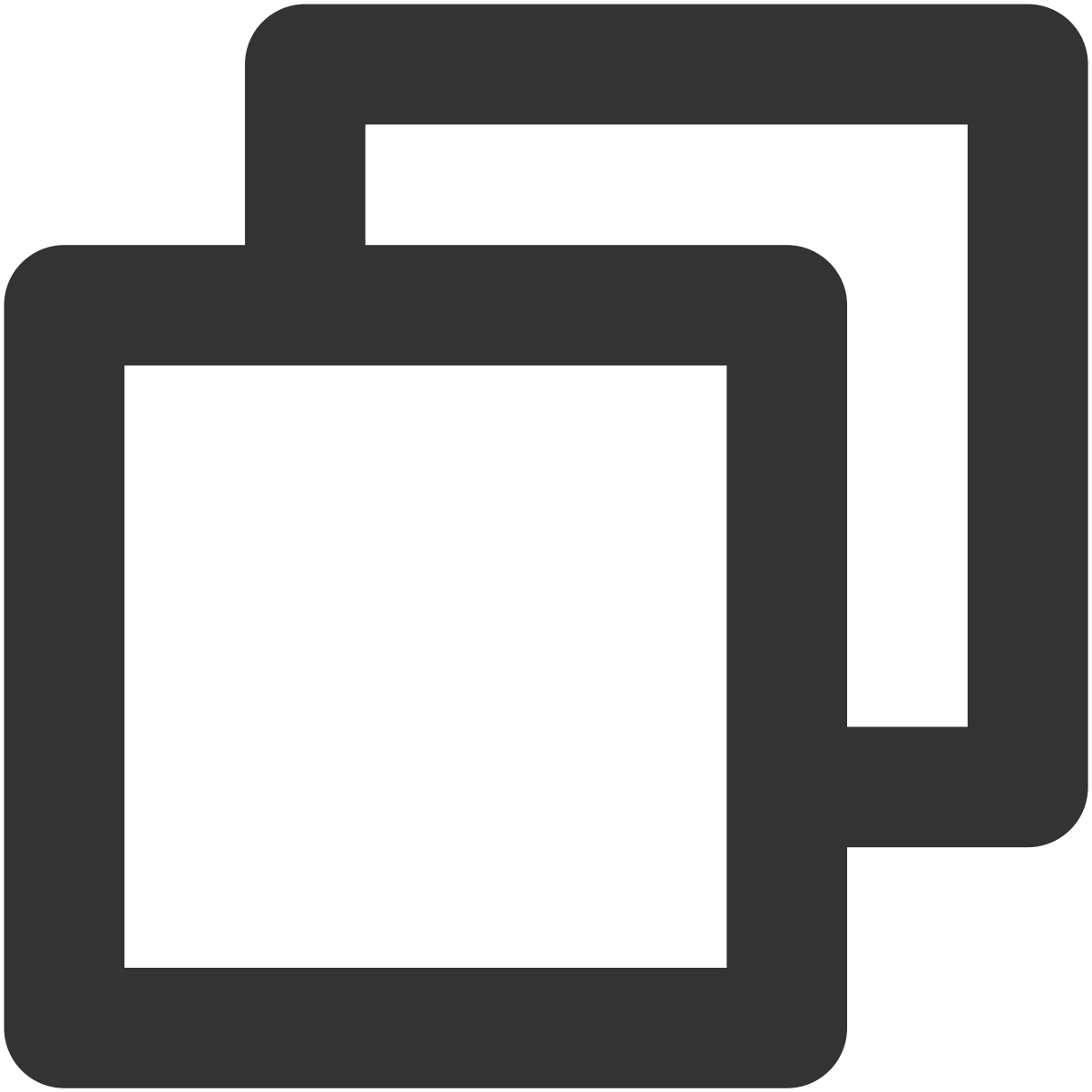


```
cat /etc/kubernetes/cluster-ca.crt
```

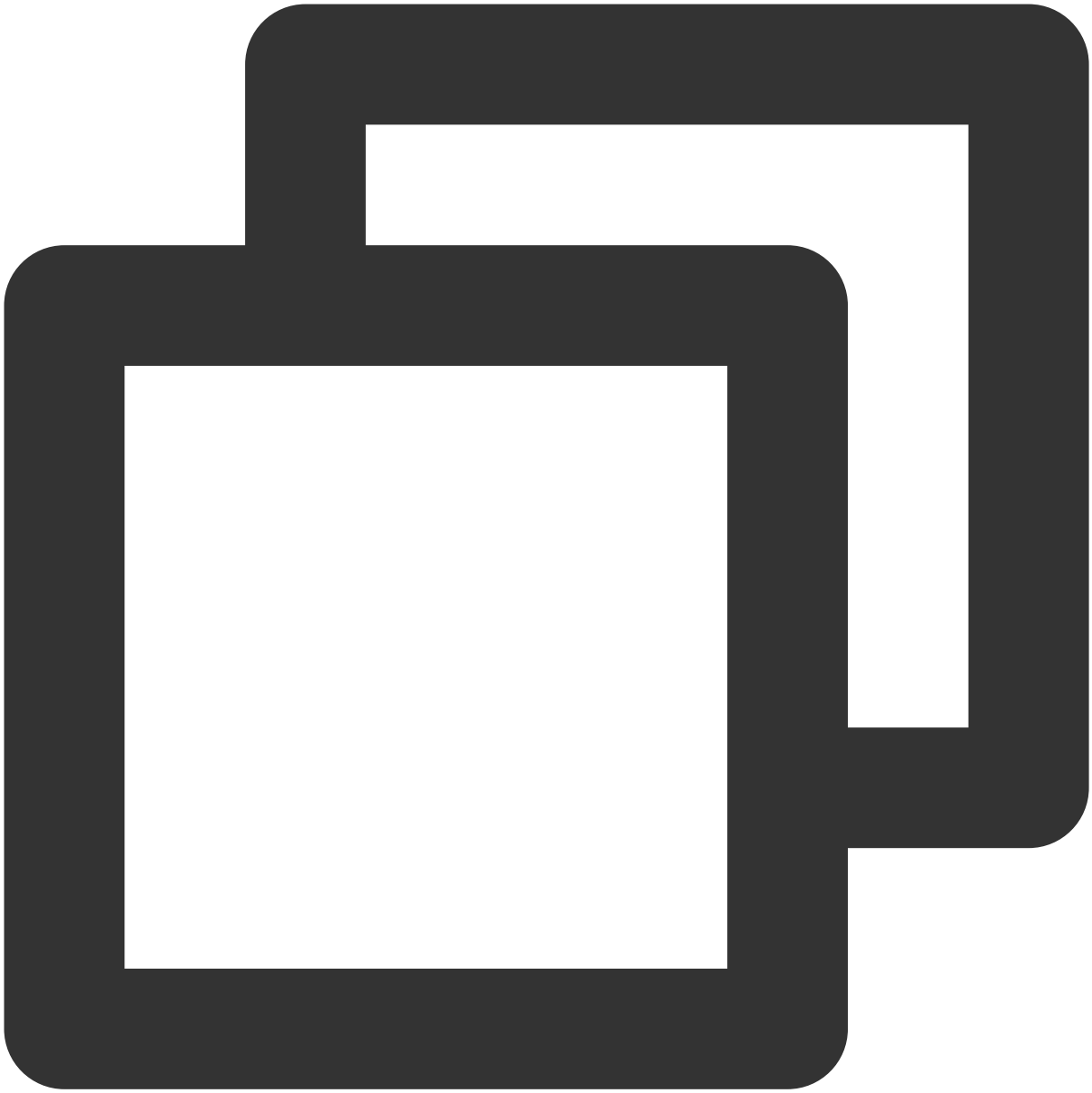
3. 다음 이미지와 같이 반환된 인증서 정보를 기록하고 저장합니다.

docker.sock 인증

TKE 클러스터의 각 node에는 `docker.sock` 파일이 있습니다. slave pod는 `docker build` 를 실행할 때 이 파일에 연결합니다. 그 전에 각 노드에 로그인하고 다음 명령을 실행하여 `docker build` 를 인증해야 합니다.



```
chmod 666 /var/run/docker.sock
```



```
ls -l /var/run/docker.sock
```

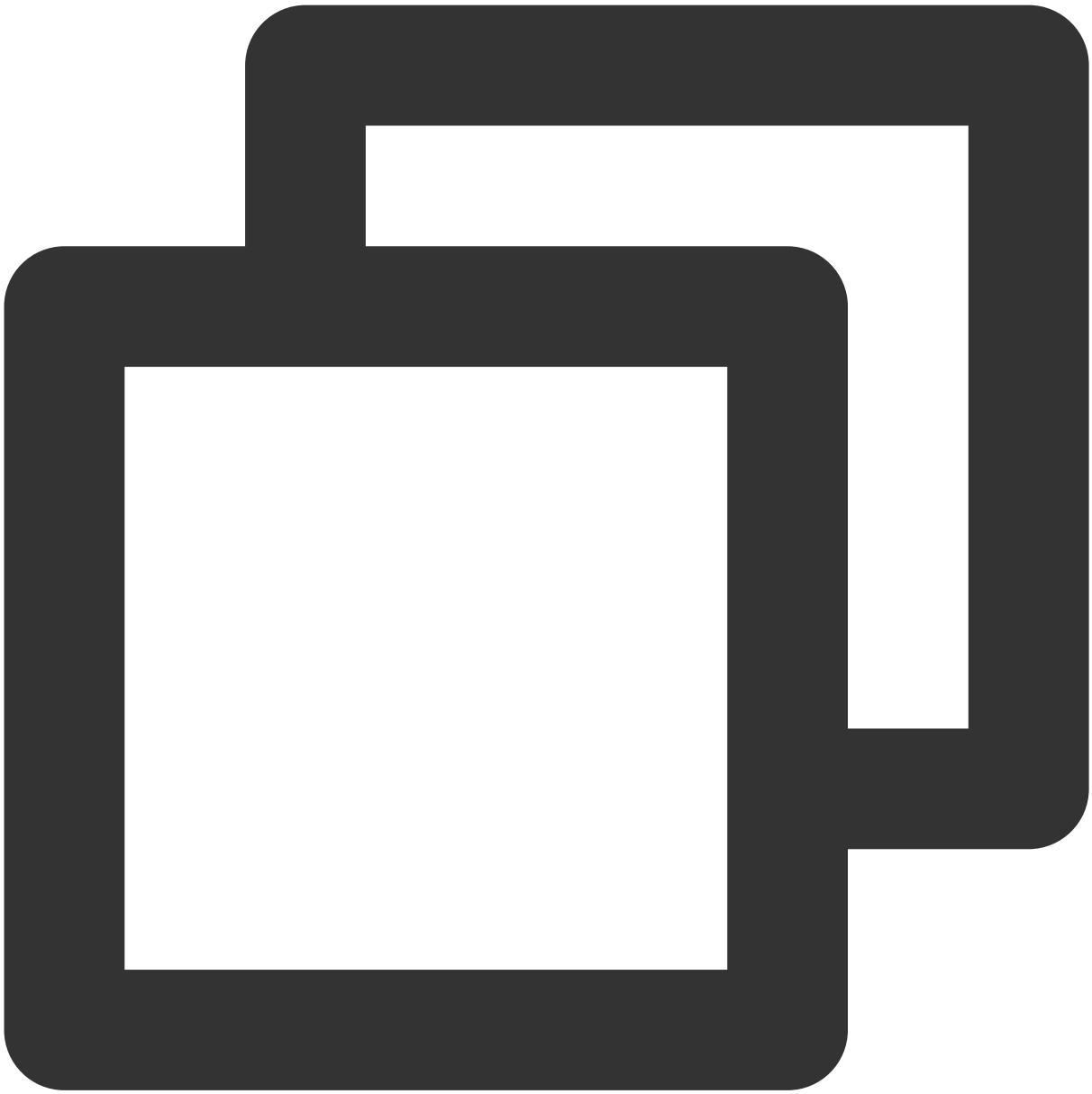
Jenkins 구성

설명

다른 Jenkins 버전에서 사용하는 UI에는 차이가 있습니다. 비즈니스 요구 사항에 따라 선택할 수 있습니다.

TKE 사설망 액세스 주소 추가

1. 표준 로그인 방법을 사용하여 Linux 인스턴스에 로그인(권장)에 설명된 대로 Jenkins Master 노드에 로그인합니다.
2. 다음 명령을 실행하여 액세스 도메인 이름을 구성합니다.

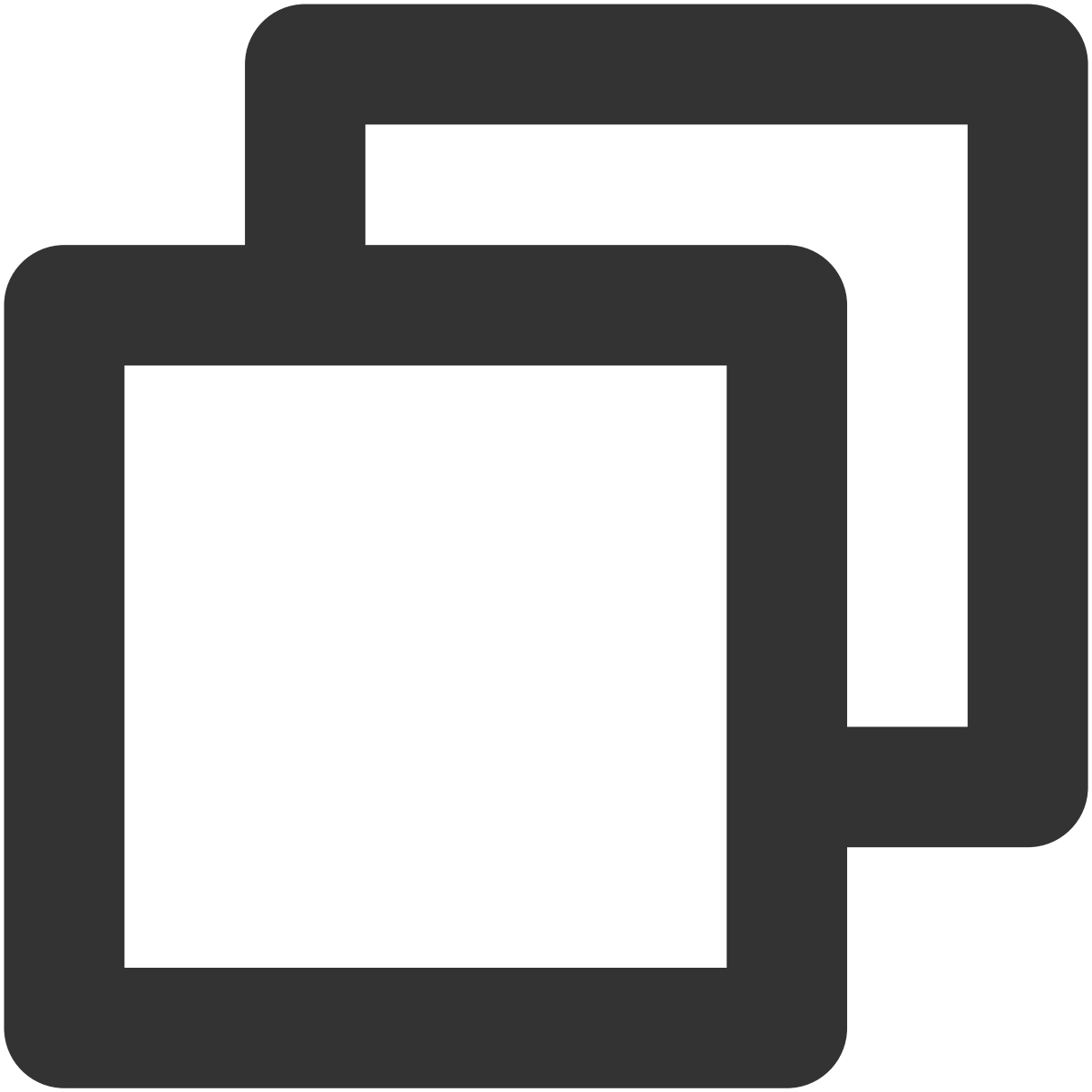


```
sudo sed -i '$a 10.x.x.x cls-ixxxelli.ccs.tencent-cloud.com' /etc/hosts
```

설명

이 명령은 클러스터에서 사설망 액세스가 활성화된 후 클러스터의 기본 정보 페이지에 있는 '클러스터 APIServer'에서 가져올 수 있습니다. 자세한 내용은 클러스터 자격 증명 가져오기를 참고하십시오.

3. 다음 명령을 실행하여 구성이 성공했는지 쿼리합니다.



```
cat /etc/hosts
```

다음 이미지와 같은 결과가 나타나면 구성에 성공한 것입니다.

```
[root@VM_0_7_centos ~]# sudo sed -i '$a 10 [redacted].ccs.tencent-clou
[root@VM_0_7_centos ~]# cat /etc/hosts
127.0.0.1 VM_0_7_centos VM_0_7_centos
127.0.0.1 localhost.localdomain localhost
127.0.0.1 localhost4.localdomain4 localhost4

::1 VM_0_7_centos VM_0_7_centos
::1 localhost.localdomain localhost
::1 localhost6.localdomain6 localhost6

10. [redacted].ccs.tencent-cloud.com
```

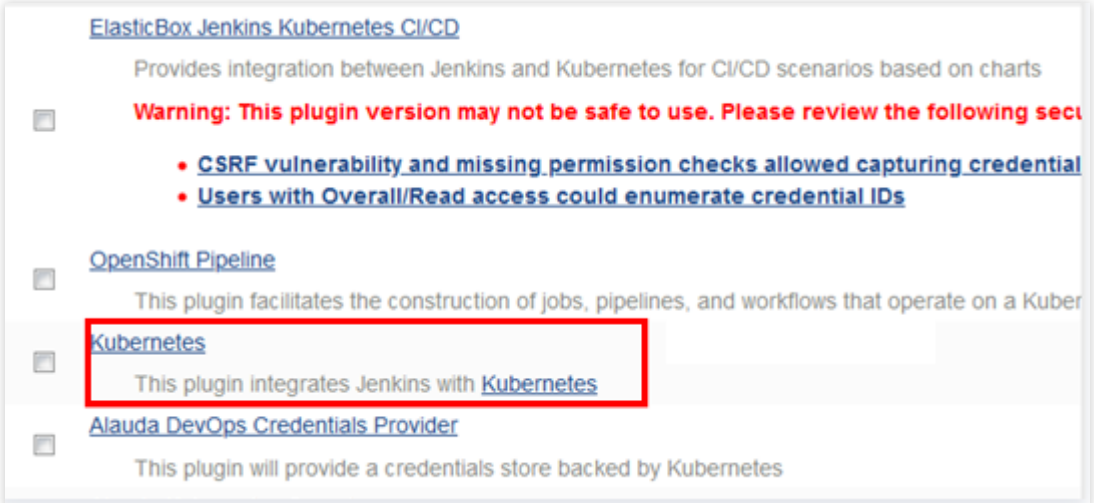
Jenkins 설치에 필요한 플러그인

1. Jenkins 백엔드에 로그인하고 왼쪽 사이드바에서 **Jenkins 관리**를 클릭합니다.
2. 'Jenkins 관리' 패널에서 **플러그인 관리**를 클릭합니다.
3. 플러그인 관리 페이지에서 **선택 가능한 플러그인**을 선택하고 Locale, Kubernetes, Git Parameter 및 Extended Choice Parameter를 선택합니다.

Locale: 중국어 플러그인. 이 플러그인을 설치하면 Jenkins 인터페이스가 중국어 버전으로 기본 설정될 수 있습니다.

Kubernetes: Kubernetes-plugin.

Git Parameter 및 **Extended Choice Parameter**: 패키지 빌드 중에 매개변수를 전달하는 데 사용됩니다. 다음 이미지와 같이 Kubernetes 플러그인을 예로 듭니다.



4. 상기 플러그인을 선택한 후 **재시작하지 않고 설치**를 클릭하고 Jenkins를 다시 시작하면 됩니다.

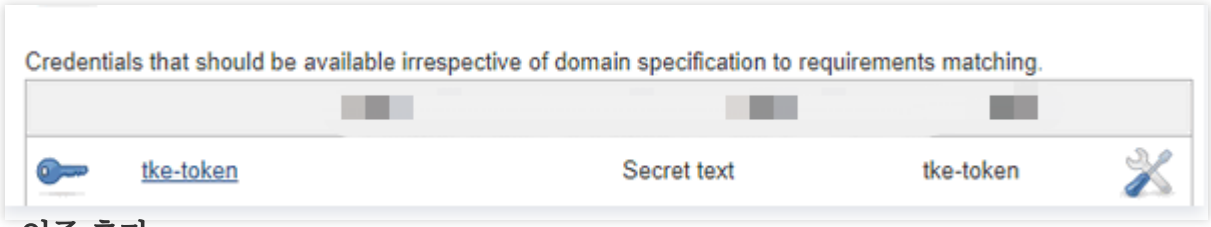
jnlp 포트 활성화

1. Jenkins 백엔드에 로그인하고 왼쪽 사이드바에서 **시스템 관리**를 클릭합니다.
2. 'Jenkins 관리' 패널에서 **전역 보안 구성**을 클릭합니다.
3. 인바운드 에이전트용 TCP 포트에서 고정을 선택하고 '고정 포트 50000'을 입력합니다.
4. 다른 구성 항목을 기본값으로 유지하고 페이지 하단에서 **저장**을 클릭합니다.

TKE 클러스터 token 추가

1. Jenkins 백엔드에 로그인하고 왼쪽 사이드바에서 **자격 증명** > **시스템**을 클릭합니다.

2. '시스템' 패널에서 **전역 자격 증명(unrestricted)**을 선택합니다.
3. '전역 자격 증명(unrestricted)' 페이지의 왼쪽 사이드바에서 **자격 증명 추가**를 클릭하고 다음과 같이 기본 자격 증명 정보를 구성합니다.
유형: **Secret text**를 선택합니다.
범위: 기본 옵션 **전역(Jenkins, nodes, items, all child items, etc)**을 사용합니다.
Secret: 클러스터 자격 증명 가져오기에서 얻은 ServiceAccount jenkins의 **Token**을 입력합니다.
ID: 기본값으로 비워 둡니다.
설명: 자격 증명 이름 및 설명 정보로 표시되는 자격 증명에 대한 정보를 완료합니다. 이 문서는 `tke-token` 을 예로 사용합니다.
4. 아래와 같이 성공적으로 추가된 후 자격 증명 목록에 표시되는 자격 증명을 추가하려면 **확인**을 클릭합니다.



gitlab 인증 추가

1. '전역 자격 증명(unrestricted)' 페이지의 왼쪽 사이드바에서 **자격 증명 추가**를 클릭하고 다음과 같이 기본 자격 증명 정보를 구성합니다.
유형: **Username with password**를 선택합니다.
범위: 기본 옵션 **전역(Jenkins, nodes, items, all child items, etc)**을 사용합니다.
사용자 이름: gitlab 사용자 이름.
비밀번호: gitlab 로그인 비밀번호.
ID: 기본값으로 비워 둡니다.
설명: 자격 증명 이름 및 설명 정보로 표시되는 자격 증명에 대한 정보를 완료합니다. 이 문서에서는 `gitlab-password` 를 예로 사용합니다.
2. **확인**을 클릭하면 추가가 완료됩니다.

slave pod 템플릿 구성

1. Jenkins 백엔드에 로그인하고 왼쪽 사이드바에서 **Jenkins 관리**를 클릭합니다.
2. 'Jenkins 관리' 패널에서 **시스템 구성**을 클릭합니다.
3. 아래와 같이 '시스템 구성' 패널 하단의 '클라우드' 섹션에서 **새 클라우드 추가** > **Kubernetes**를 선택합니다.
4. **Kubernetes Cloud details...**를 클릭하여 Kubernetes에 대한 다음 기본 정보를 구성합니다.
다음은 주요 매개변수 구성입니다. 다른 매개변수는 기본값을 유지하십시오.
이름: 사용자 지정 이름입니다. 이 문서에서는 `kubernetes` 를 예로 사용합니다.
Kubernetes URL: TKE 클러스터 액세스 주소입니다. 클러스터 자격 증명 가져오기를 참고하십시오.
Kubernetes 서버 인증서 Key: 클러스터 CA 인증서를 얻으려면 클러스터 CA 인증서 가져오기를 참고하십시오.

자격 증명: TKE 클러스터 token 추가 단계에서 만든 `tke-token` 자격 증명을 선택한 다음 **연결 테스트**를 클릭합니다. 연결에 성공하면 Connection test succeessful 프롬프트가 표시됩니다.
Jenkins URL: `http://10.x.x.x:8080` 과 같은 Jenkins 사설망 주소를 입력합니다.
5. **Pod Templates** > **Pod 템플릿 추가** > **Pod Templates details...**를 선택하고 Pod 템플릿의 기본 정보를 구성합니다.
다음은 주요 매개변수에 대한 설명입니다. 다른 매개변수는 기본값을 유지하십시오.
이름: 사용자 지정 이름을 입력합니다. 이 문서에서는 `jnlp-agent` 를 예로 사용합니다.
레이블: 태그 이름을 정의합니다. 태그를 기반으로 빌드할 Pod를 선택할 수 있습니다. 이 문서에서는 `jnlp-agent` 를 예로 사용합니다.
사용법: **가능한 한 이 노드 사용**을 선택합니다.
6. '컨테이너 드롭다운 목록'에서 **컨테이너 추가** > **Container Template**을 선택하고 다음 컨테이너 정보를 구성합니다.
이름: 사용자 지정 컨테이너 이름을 입력합니다. 이 문서에서는 `jnlp-agent` 를 예로 사용합니다.
Docker 이미지: 이미지 주소 `jenkins/jnlp-slave:alpine` 을 입력합니다.
작업 디렉터리: 기본값으로 유지합니다. shell 스크립트를 빌드하고 패키징하는 데 사용할 작업 디렉터리를 기록하십시오.
다른 옵션은 기본값으로 둡니다.
7. '볼륨'에서 다음 단계를 완료하여 볼륨을 추가하고 slave pod에 대한 docker 명령을 구성합니다.
7.1 **볼륨 추가** > **Host Path Volume**을 선택합니다. 호스트 및 마운트 경로 모두에 대해 `/usr/bin/docker` 를 입력합니다.
7.2 **볼륨 추가** > **Host Path Volume**을 선택합니다. 호스트 및 마운트 경로 모두에 대해 `/var/run/docker.sock` 을 입력합니다.
7.3 페이지 하단의 **저장**을 클릭하여 slave pod 템플릿 구성을 완료합니다.

다음 단계 작업

2단계: Slave pod 빌딩 구성으로 이동하여 작업을 생성하고 작업 매개 변수를 구성합니다.

2단계: Slave pod 빌드 구성

최종 업데이트 날짜: : 2023-04-28 11:08:19

이 단계에서는 작업을 생성하고 작업 매개변수를 구성하여 Jenkins에서 slave pod를 빌드하는 방법을 설명합니다.

설명

각기 다른 Jenkins 버전에서 사용하는 UI에는 차이가 있습니다. 비즈니스 요구 사항에 따라 선택할 수 있습니다.

작업 생성

1. Jenkins 백엔드에 로그인하고 **새 작업** 또는 **작업 생성**을 클릭합니다.
2. 작업 생성 페이지에서 작업의 기본 정보를 구성합니다.

작업 이름 입력: 사용자 지정 이름을 입력합니다. 이 문서에서는 `test` 를 예로 사용합니다.

유형: **자유형 소프트웨어 프로젝트 빌드**를 선택합니다.

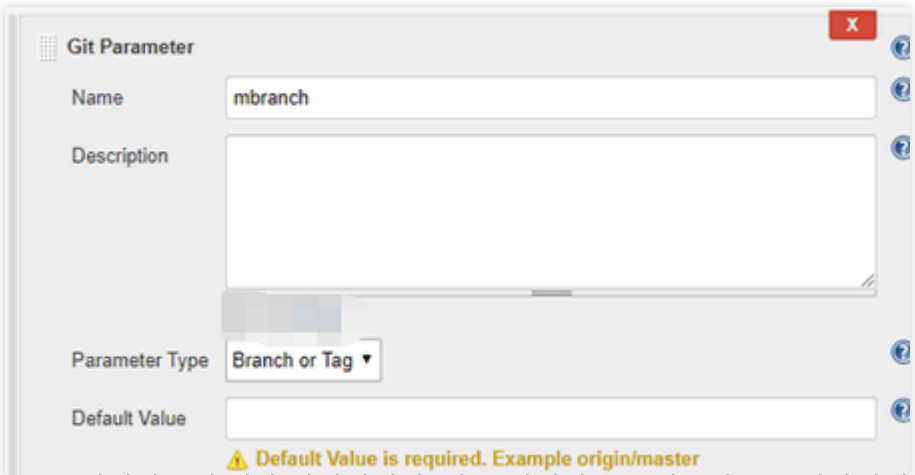
3. **확인**을 클릭하여 작업 매개변수 구성 페이지로 이동합니다.
4. 작업 매개변수 구성 페이지에서 기본 정보를 구성합니다.

설명: 사용자 지정 작업 정보를 입력합니다. 이 문서는 slave pod test를 예로 사용합니다.

빌드 프로세스 매개변수화: 이 옵션을 선택하고 **매개변수 추가** > **Git Parameter**를 선택합니다.

작업 매개변수 구성

1. 다음과 같이 'Git Parameter' 패널에서 다음 매개변수를 구성합니다.

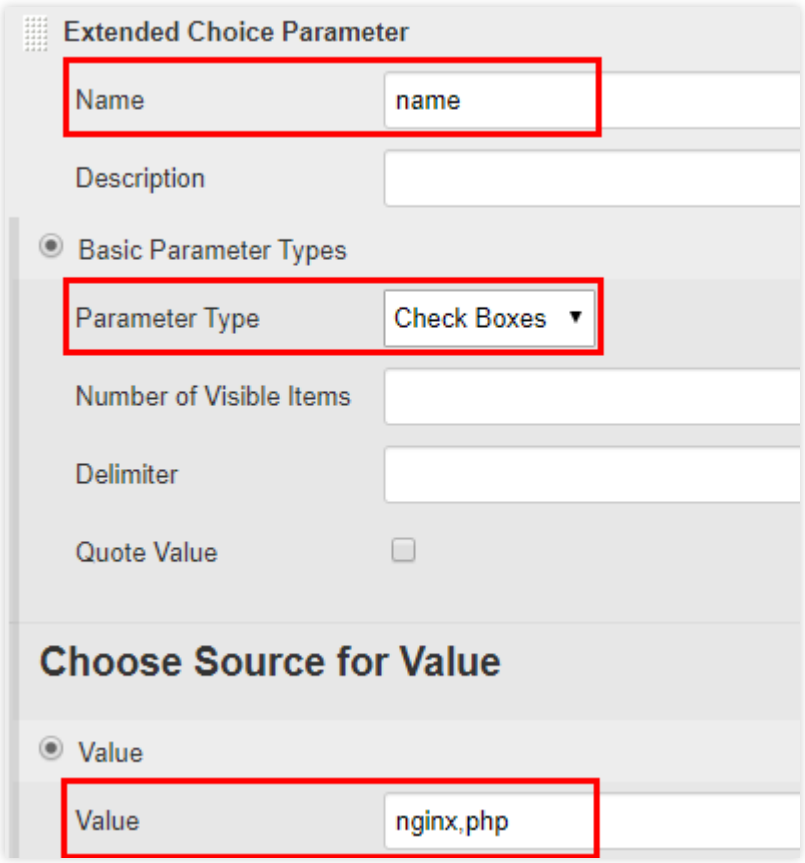


다음은 주요 매개변수에 대한 설명입니다. 다른 매개변수는 기본값을 유지하십시오.

Name: `mbranch` 를 입력하면 브랜치를 일치시켜 얻을 수 있습니다.

Parameter Type: **Branch or Tag**를 선택합니다.

2. **매개변수 추가** > **Extended Choice Parameter**를 선택합니다. 표시되는 'Extended Choice Parameter' 패널에서 다음 이미지와 같이 다음 매개변수를 구성합니다.



다음은 주요 매개변수에 대한 설명입니다. 다른 매개변수는 기본값을 유지하십시오.

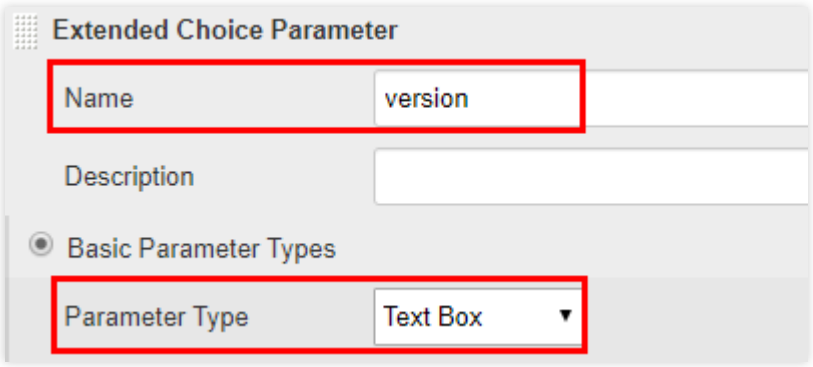
Name: 이미지 이름을 얻는 데 사용할 수 있는 `name` 을 입력합니다.

Basic Parameter Types: 이 옵션을 선택합니다.

Parameter Type: **Check Boxes**를 선택합니다.

Value: 이 옵션을 선택하고 사용자 지정 이미지 이름을 입력합니다. 이 값은 `name` 변수에 전달됩니다. 이 문서는 `nginx,php` 를 예로 사용합니다.

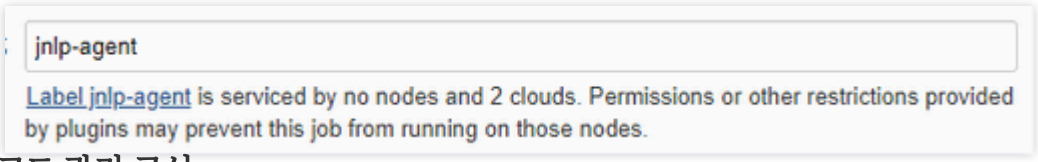
3. **매개변수 추가** > **Extended Choice Parameter**를 선택합니다. 'Extended Choice Parameters' 패널에서 다음 이미지와 같이 다음 매개변수를 구성합니다.



다음은 주요 매개변수에 대한 설명입니다. 다른 매개변수는 기본값을 유지하십시오.

Name: 이미지 태그 변수를 얻기 위해 사용할 수 있는 `version` 을 입력합니다.

Basic Parameter Types: 이 옵션을 선택합니다.
Parameter Type: 텍스트 형식의 이미지 값을 가져와 `version` 변수에 전달하려면 **Text Box**를 선택합니다.
4. **프로젝트의 실행 노드 제한**을 선택합니다. 태그 표현식의 경우 slave pod 템플릿 구성 단계에서 설정한 Pod 태그 `jnlp-agent` 를 입력합니다.

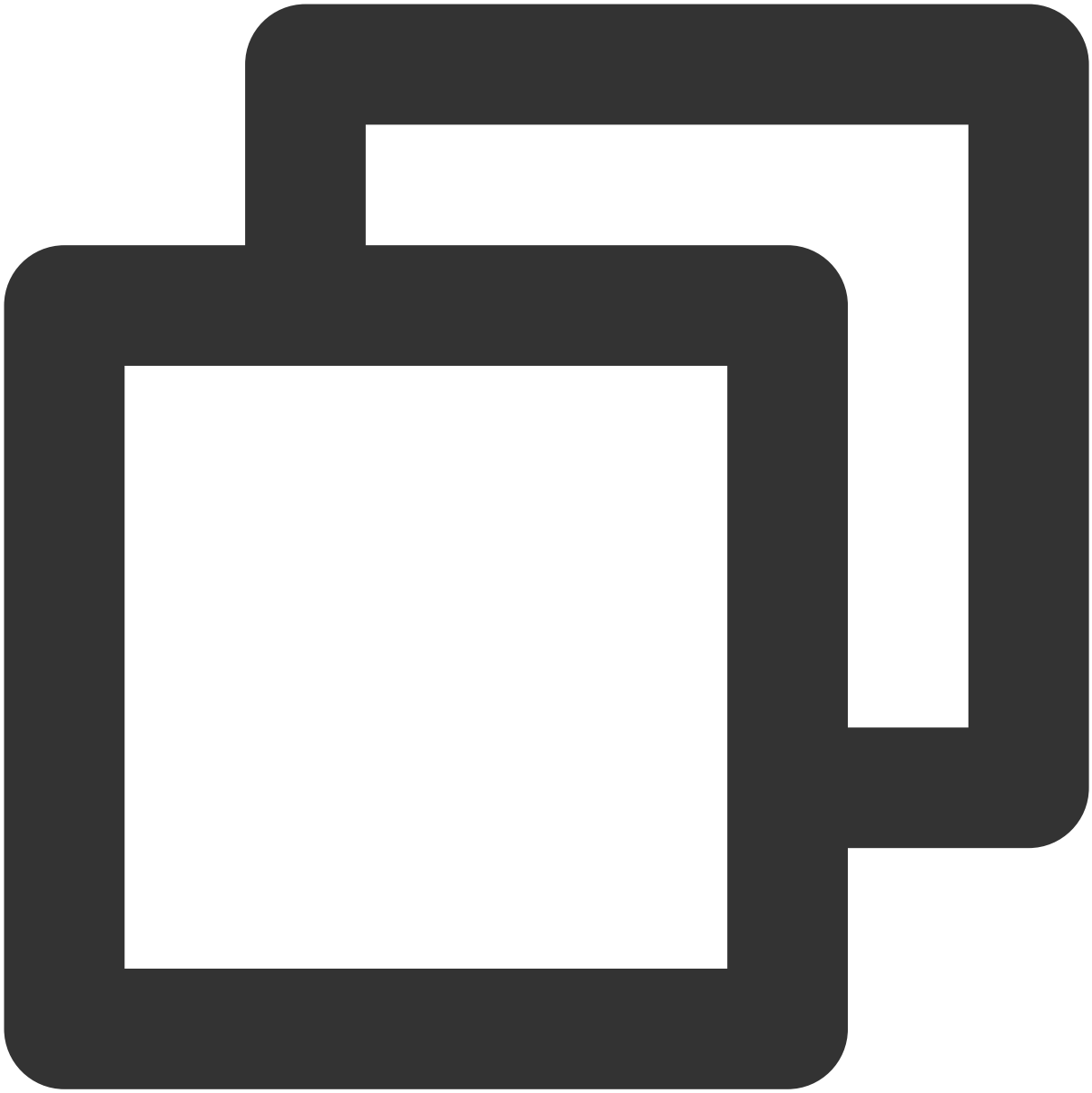


소스 코드 관리 구성

'소스 코드 관리' 영역에서 **Git**을 선택하고 다음 정보를 구성합니다.
Repositories:
Repository URL: `https://gitlab.com/user-name/demo.git` 과 같은 gitlab 주소를 입력합니다.
Credentials: gitlab 인증 추가 단계에서 생성한 인증 자격 증명을 선택합니다.
Branches to build:
지정된 브랜치(비어 있는 경우 any): 브랜치를 동적으로 획득하기 위해 사용되는 `$mbranch` 를 입력하고 그 값은 Git Parameter에서 정의한 `mbranch` 의 값에 해당합니다.

Shell 패키징 스크립트 구성

1. '빌드' 영역에서 **빌드 단계 추가** > **shell 실행**을 선택합니다.
2. 다음 스크립트 내용을 복사하여 '명령' 입력 상자에 붙여넣습니다. 그 다음 **저장**을 클릭합니다.
주의사항
이 스크립트에서 gitlab 주소, TKE 이미지 주소, 이미지 리포지토리의 사용자 이름 및 비밀번호와 같은 정보는 예시일 뿐입니다. 실제 사례에서는 필요에 따라 교체하십시오.
Docker build의 소스 코드를 기반으로 패키지를 빌드해야 합니다. 또한 작업 디렉터리 `/home/Jenkins/agent` 는 '컨테이너 목록'에 있는 Container Template의 작업 디렉터리와 일치해야 합니다.



```
echo " gitlab 주소: https://gitlab.com/[user]/[project-name]].git"
echo "선택한 브랜치(이미지):" $mbranch, "브랜치(이미지) 버전 설정: "$version
echo " TKE 이미지 주소: hkccr.ccs.tencentyun.com/[namespace]/[ImageName]"

echo "1. TKE 이미지 리포지토리에 로그인"
docker login --username=[username] -p [password] hkccr.ccs.tencentyun.com

echo "2. Docker build 의 소스 코드를 기반으로 패키지 빌드:"
cd /home/Jenkins/agent/workspace/[project-name] && docker build -t $name:$version

echo "3. Docker 이미지를 TKE 리포지토리에 업로드:"
```

```
docker tag $name:$version hkccr.ccs.tencentyun.com/[namespace]/[ImageName]:$name-$
docker push hkccr.ccs.tencentyun.com/[namespace]/[ImageName]:$name-$version
```

스크립트는 다음 기능을 제공합니다.

선택한 브랜치, 이미지 이름, 이미지 태그를 가져옵니다.

TKE 이미지 리포지토리의 코드와 결합 및 빌드된 docker 이미지를 게시합니다.

다음 단계 작업

이제 성공적으로 slave pod를 구축했습니다. 빌드 테스트로 이동하여 이미지를 게시하고 확인합니다.

TKE에 Jenkins 배포

최종 업데이트 날짜: : 2023-04-28 11:08:19

작업 시나리오

많은 DevOps 요구 사항은 Jenkins를 통해 구현해야 합니다. 본문은 TKE에서 Jenkins를 배포하는 방법을 설명합니다.

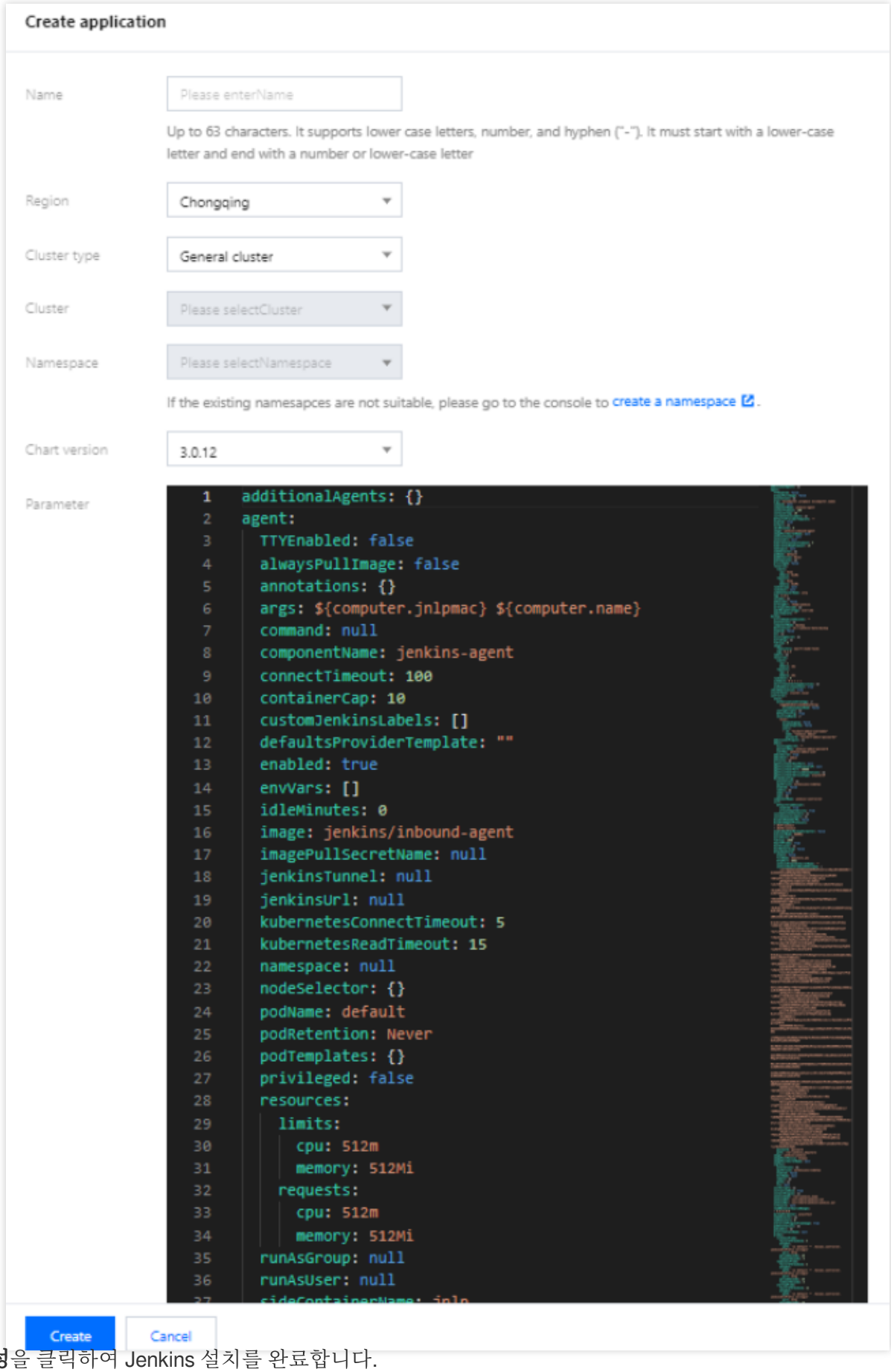
전제 조건

TKE 클러스터를 생성합니다.

작업 단계

Jenkins 설치

1. TKE 콘솔에 로그인하고 왼쪽 사이드바에서 마켓플레이스를 선택합니다.
2. **마켓플레이스** 페이지에서 Jenkins를 검색하고 Jenkins 애플리케이션 페이지를 입력합니다.
3. **애플리케이션 생성**을 클릭하고 필요에 따라 '매개변수'에서 values.yaml을 구성합니다.



4. **생성**을 클릭하여 Jenkins 설치를 완료합니다.

Jenkins UI 노출

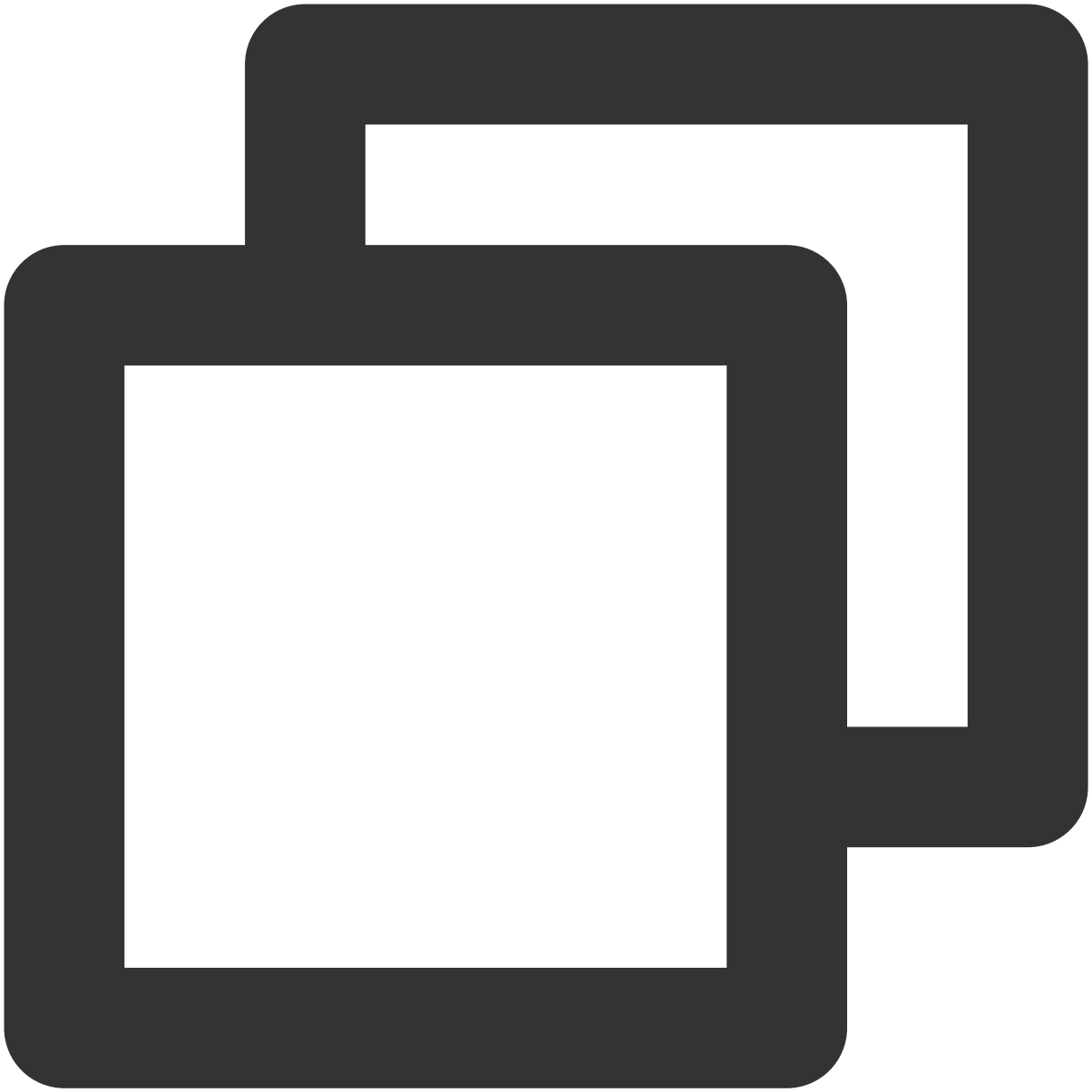
기본적으로 클러스터 외부에서 Jenkins UI에 액세스할 수 없습니다. Jenkins UI에 액세스하려면 Ingress를 사용할 수 있습니다. TKE는 CLB 유형 Ingress 및 Nginx 유형 Ingress를 제공합니다. 문서를 참고하여 선택하시기 바랍니다.

설명

다음 예시에서는 Jenkins 2.263 버전을 사용하며 Jenkins 버전에 따라 사용되는 UI에 차이가 있습니다. 비즈니스 요구 사항에 따라 선택할 수 있습니다.

Jenkins에 로그인

Jenkins UI에서 초기 사용자 이름과 비밀번호를 입력하여 Jenkins 백엔드에 로그인합니다. 사용자 이름은 admin이며 비밀번호는 다음 명령을 실행하여 가져올 수 있습니다.



```
kubectl -n devops get secret jenkins -o jsonpath='{.data.jenkins-admin-password}' |
```

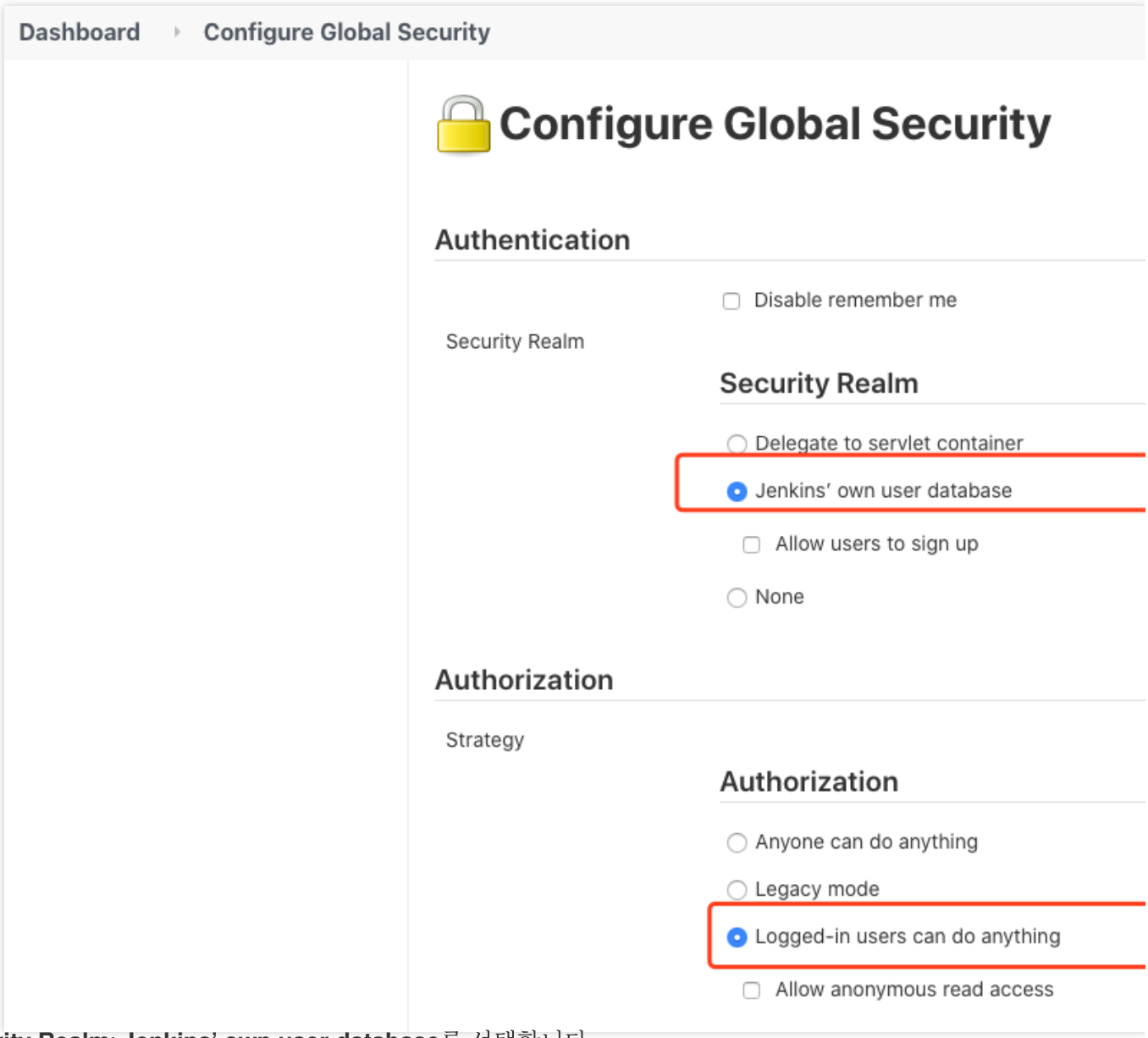
주의사항

상기 명령을 실행할 때 텍스트를 실제 네임스페이스로 바꿉니다.

사용자 생성

Jenkins를 일반 사용자로 관리하는 것을 권장합니다. 일반 사용자를 생성하기 전에 인증 및 권한 정책을 설정해야 합니다.

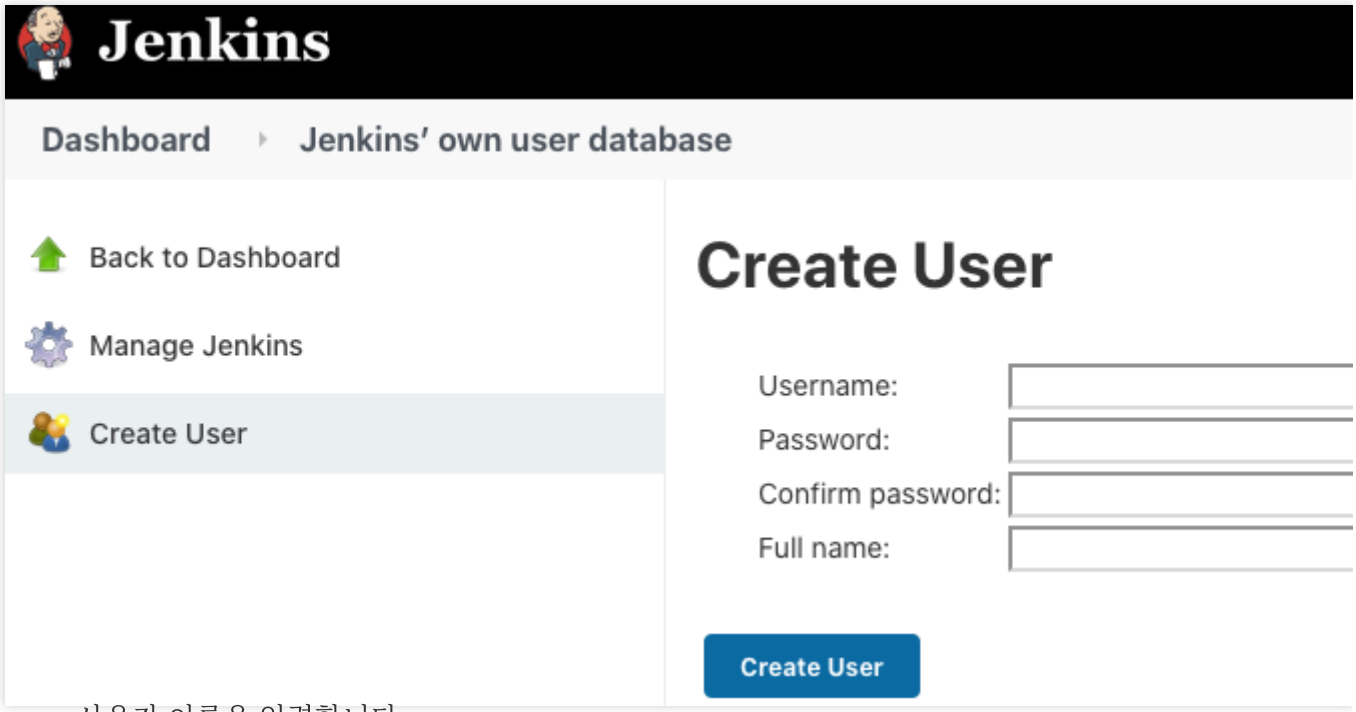
1. Jenkins 백엔드에 로그인하고 **Dashboard** > **Manage Jenkins** > **Security** > **Configure Global Security**를 클릭하여 아래와 같이 인증 및 권한 부여 정책 페이지로 들어갑니다.



Security Realm: **Jenkins' own user database**를 선택합니다.

Authorization: **Logged-in users can do anything**을 선택합니다.

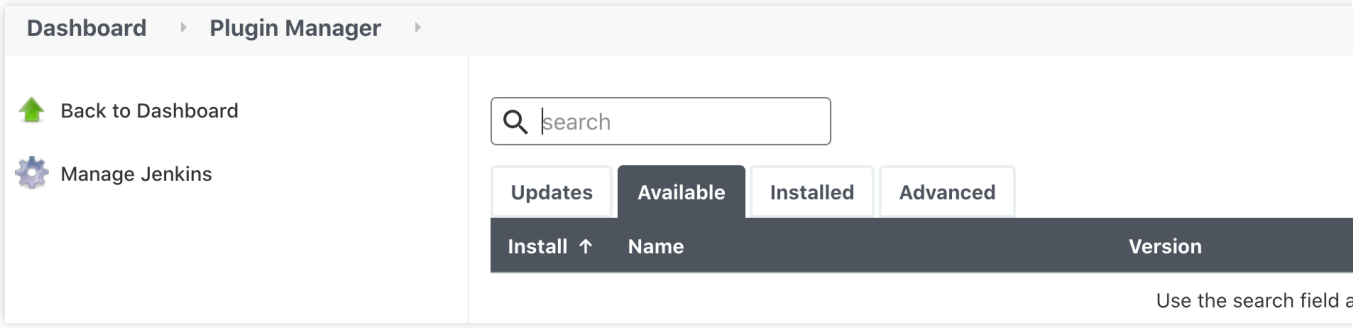
2. **Dashboard** > **Manage Jenkins** > **Security** > **Manage Users** > **Create User**를 클릭하고 아래와 같이 프롬프트에 따라 사용자를 생성합니다.



Username: 사용자 이름을 입력합니다.
Password: 비밀번호를 입력합니다.
Confirm password: 비밀번호를 확인합니다.
Full name: 사용자 이름 전체를 입력합니다.
3. **Create User**를 클릭하면 사용자 생성이 완료됩니다.

플러그 인 설치

Jenkins 백엔드에 로그인하고 **Dashboard** > **Manage Jenkins** > **System Configuration** > **Manage Plugins**를 클릭하여 플러그인 관리 페이지로 들어갑니다.

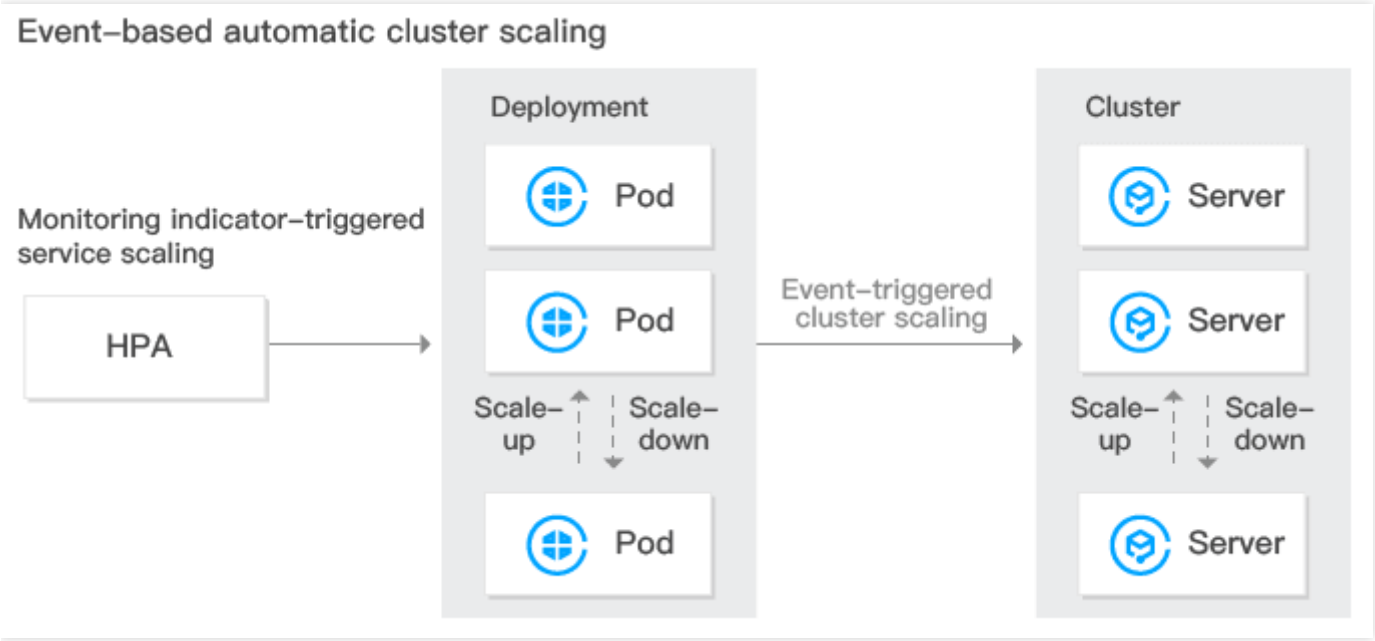


다음과 같이 일반적으로 사용되는 플러그인을 설치할 수 있습니다.
kubernetes
pipeline
git
gitlab
github

탄력적 스케일링
클러스터 오토 스케일링 사례

최종 업데이트 날짜: : 2022-09-22 17:00:41

Tencent Kubernetes Engine(TKE)은 클러스터 및 서비스 수준에서 탄력적인 확장성을 제공합니다. CPU, 메모리, 대역폭을 포함한 컨테이너의 메트릭을 모니터링하고 오토 스케일링을 수행할 수 있습니다. 동시에 컨테이너에 리소스가 충분하지 않거나 필요한 것보다 많은 리소스가 있는 경우 클러스터를 자동으로 스케일링할 수 있습니다. 아래 이미지를 참고하십시오.



클러스터 오토 스케일링 기능

TKE를 사용하면 사용자가 클러스터에 대한 오토 스케일링을 활성화하여 컴퓨팅 리소스를 효율적으로 관리할 수 있습니다. 사용자는 필요에 따라 스케일링 정책을 설정할 수 있습니다. Cluster 오토 스케일링에는 다음과 같은 기능이 있습니다.

- Cluster 오토 스케일링은 프로젝트 부하 상황에 따라 실시간으로 CVM(Cloud Virtual Machines)을 동적으로 자동 생성 및 해제하여 사용자가 최적의 인스턴스 수로 프로젝트 상황에 대처할 수 있도록 도와줍니다. 전체 프로세스에서 사람의 개입이 필요하지 않으므로 사용자가 수동 배포에서 벗어날 수 있습니다.
- Cluster 오토 스케일링은 사용자가 최적의 노드 리소스 양으로 프로젝트 상황을 처리하는 데 도움이 될 수 있습니다. 더 많은 요구 사항이 있는 경우 컨테이너 클러스터에 CVM을 원활하고 자동으로 추가합니다. 요구 사항이 적으면 불필요한 CVM을 자동으로 제거하여 장치 활용도를 높이고 배포 및 인스턴스 비용을 줄입니다.

클러스터 오토 스케일링 기능 설명

Kubernetes cluster autoscaling의 기본 기능

- 여러 스케일링 그룹 설정을 지원합니다.
- 스케일 인 및 스케일 아웃 정책 설정을 지원합니다. 자세한 내용은 Cluster Autoscaler를 참고하십시오.

고급 TKE 스케일링 그룹 기능

- 스케일링 그룹을 생성하는 동안 사용자 지정 모델 사용을 지원합니다(권장).
- 스케일링 그룹을 생성하는 동안 클러스터의 노드를 템플릿으로 사용할 수 있습니다.
- 스케일링 그룹에 스팟 인스턴스 추가를 지원합니다(권장).
- 모델이 품절되면 적절한 스케일링 그룹을 자동으로 매칭 지원합니다.
- 가용존에서 스케일링 그룹 설정을 지원합니다.

클러스터 오토 스케일링 제한 사항

- 클러스터 오토 스케일링으로 추가할 수 있는 노드 수는 VPC, 컨테이너 네트워크, TKE 클러스터 노드 할당량, 구매할 수 있는 CVM 할당량에 따라 제한됩니다.
- 노드 확장 가능 여부는 사용하려는 모델 지속 사용 가능 여부에 따라 다릅니다. 모델이 매진되면 노드를 확장할 수 없습니다. 여러 스케일링 그룹을 구성하는 것이 좋습니다.
- 워크로드에서 컨테이너의 request 값을 설정해야 합니다. request 값으로 클러스터의 리소스가 충분한지 평가하여 자동 확장을 트리거할지 여부를 결정할 수 있습니다.
- 노드의 메트릭 기반 오토 스케일링 모니터링을 활성화하지 않는 것이 좋습니다.
- 조정 그룹을 삭제하면 해당 그룹의 CVM 인스턴스도 종료됩니다. 이 때 주의하시기 바랍니다.

클러스터 스케일링 그룹 설정

- 여러 스케일링 그룹 설정(권장)
  클러스터에 여러 개의 스케일링 그룹이 있는 경우 오토 스케일링 구성 요소는 선택한 스케일링 알고리즘에 따라 스케일 아웃할 스케일링 그룹을 선택합니다. 구성 요소는 매번 하나의 스케일링 그룹만 선택합니다. CVM 모델 품절 등의 사유로 타깃 스케일링 그룹을 스케일 아웃하지 못한 경우, 스케일링 그룹은 일정 시간 휴면 상태가 됩니다. 동시에 두 번째 일치하는 스케일링 그룹이 스케일 아웃을 위해 선택됩니다.

- Random: 스케일 아웃을 위한 임의의 스케일링 그룹을 선택합니다.

- Most-Pods: pending 중인 Pod 및 스케일링 그룹에 대해 선택한 모델을 기반으로 가장 많은 Pod를 스케줄링할 수 있는 스케일링 그룹을 선택합니다.

- Least-waste: pending 중인 Pod 및 확장 그룹에 대해 선택한 모델을 기반으로 Pod 스케줄링 후 남은 리소스를 최소화할 수 있는 스케일링 그룹을 선택합니다.

모델 품절로 인한 스케일링 실패를 방지하기 위해 클러스터에 다른 모델로 여러 개의 스케일링 그룹을 구성하는 것이 좋습니다. 동시에 스팟 인스턴스와 일반 인스턴스의 조합을 사용하여 비용을 절감할 수 있습니다.

- 단일 스케일링 그룹 구성
  클러스터 확장에 하나의 특정 모델만 사용하려는 경우 확장 그룹을 여러 서브넷 및 가용존으로 구성하는 것이 좋습니다.

TKE에서 사용자 지정 지표를 사용하여 오토 스케일링 진행

최종 업데이트 날짜: : 2023-04-27 18:15:01

작업 시나리오

Custom Metrics API를 기반으로 TKE는 대부분의 HPA 시나리오를 다룰 수 있는 CPU, 메모리, 디스크, 네트워크 및 GPU 관련 메트릭을 포함하여 오토 스케일링을 위한 많은 메트릭을 지원합니다. 자세한 메트릭은 오토 스케일링 메트릭을 참고하십시오. 서비스 단일 복제본 QPS 수를 기반으로 하는 오토 스케일링과 같은 복잡한 시나리오의 경우 prometheus-adapter를 설치하여 오토 스케일링을 구현할 수 있습니다. Kubernetes는 메트릭을 기반으로 오토 스케일링을 수행할 수 있도록 HPA용 Custom Metrics API 및 External Metrics API를 제공하므로, 사용자가 필요에 따라 오토 스케일링을 사용자 지정할 수 있습니다. prometheus-adapter는 위의 두 API를 지원합니다. 실제 환경에서 Custom Metrics API는 대부분의 시나리오를 충족할 수 있습니다. 이 문서는 Custom Metrics API를 통해 오토 스케일링을 위해 사용자 지정 메트릭을 사용하는 방법을 설명합니다.

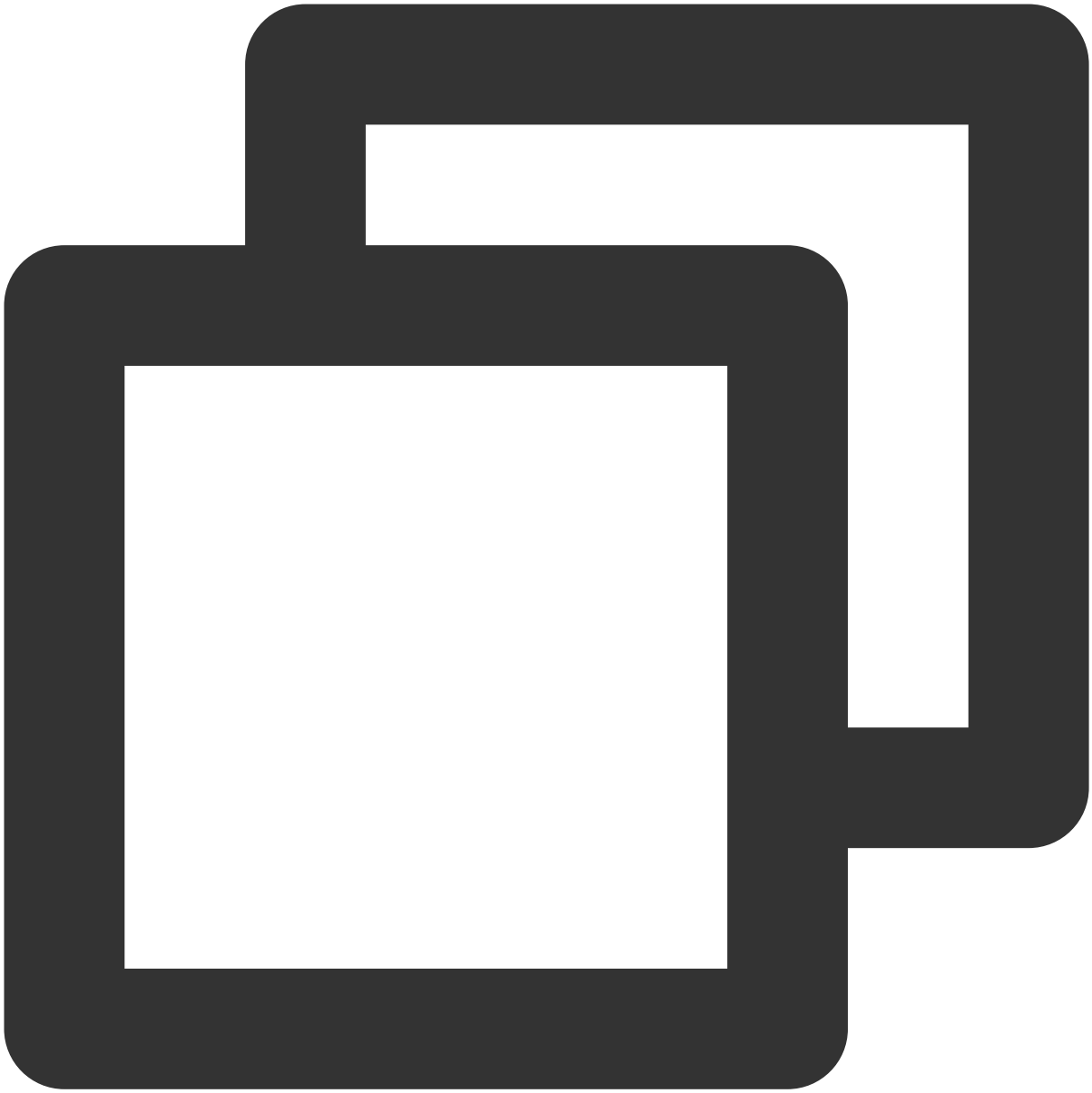
전제 조건

v1.12 이상의 TKE 클러스터를 생성합니다. 자세한 내용은 클러스터 생성을 참고하십시오.
Prometheus 인스턴스를 배포하고 해당하는 사용자 정의 메트릭을 수집합니다.
Helm을 설치합니다.

작업 단계

모니터링 메트릭 열기

본문은 `httpserver_requests_total` 메트릭을 열고 HTTP 요청을 기록하는 Golang 서비스 애플리케이션을 예로 듭니다. 이 메트릭은 아래와 같이 서비스 애플리케이션의 QPS 값을 계산하는 데 사용할 수 있습니다.

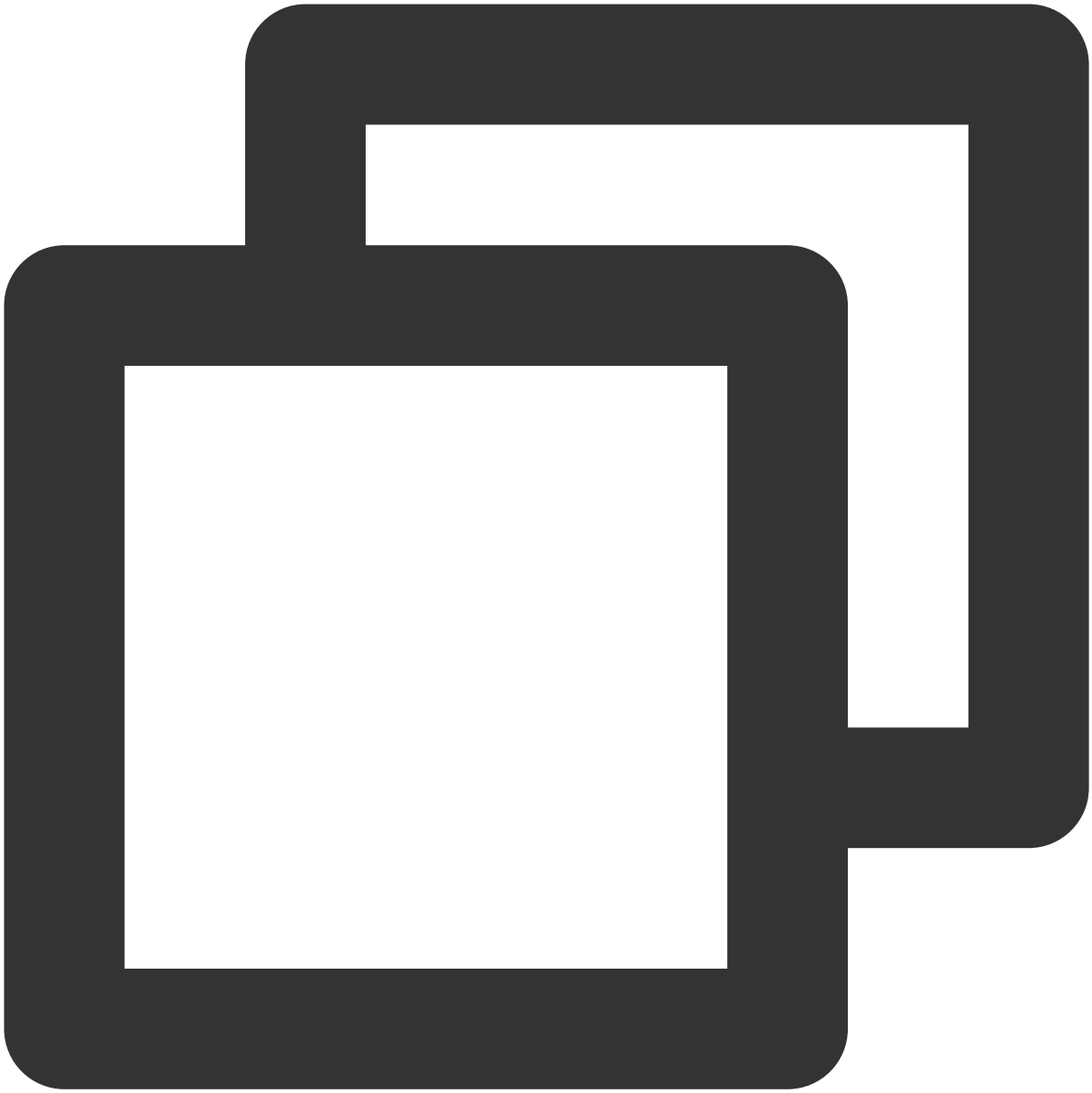


```
package main

import (
        "github.com/prometheus/client_golang/prometheus"
        "github.com/prometheus/client_golang/prometheus/promhttp"
        "net/http"
        "strconv"
)

var (
HTTPRequests = prometheus.NewCounterVec(
```

```
        prometheus.CounterOpts{
            Name: "httpserver_requests_total",
            Help: "Number of the http requests received since the server started",
        },
        []string{"status"},
    )
)

func init() {
    prometheus.MustRegister(HTTPRequests)
}

func main(){
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        path := r.URL.Path
        code := 200
        switch path {
        case "/test":
            w.WriteHeader(200)
            w.Write([]byte("OK"))
        case "/metrics":
            promhttp.Handler().ServeHTTP(w, r)
        default:
            w.WriteHeader(404)
            w.Write([]byte("Not Found"))
        }
        HTTPRequests.WithLabelValues(strconv.Itoa(code)).Inc()
    })
    http.ListenAndServe(":80", nil)
}
```

서비스 애플리케이션 배포

Deployment를 사용하면 아래와 같이 서비스 응용 프로그램을 TKE 클러스터에 컨테이너화하고 배포할 수 있습니다.



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: httpserver
  namespace: httpserver
spec:
  replicas: 1
  selector:
    matchLabels:
      app: httpserver
  template:
```

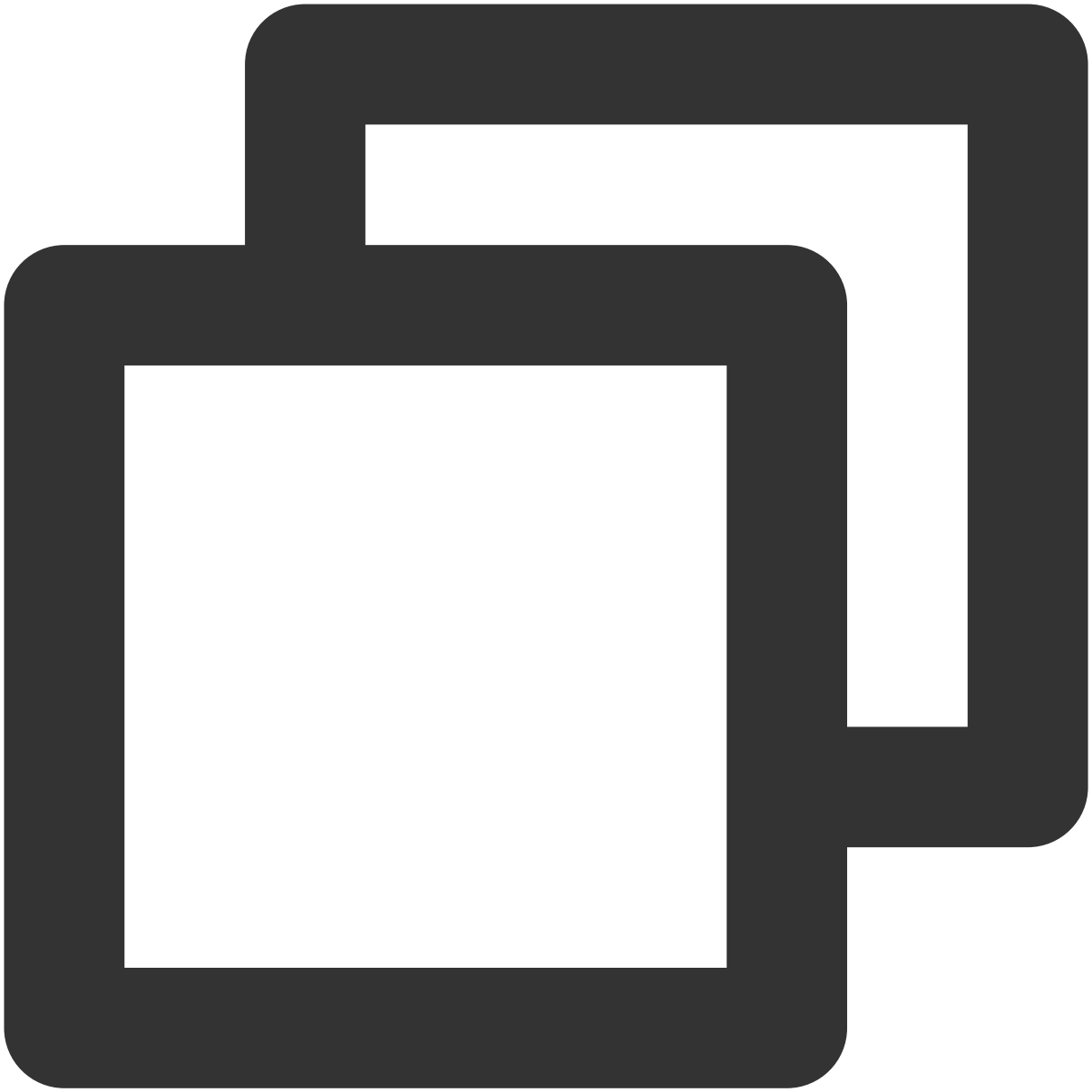
```
    metadata:
      labels:
        app: httpserver
    spec:
      containers:
      - name: httpserver
        image: registry.imroc.cc/test/httpserver:custom-metrics
        imagePullPolicy: Always

---

apiVersion: v1
kind: Service
metadata:
  name: httpserver
  namespace: httpserver
  labels:
    app: httpserver
  annotations:
    prometheus.io/scrape: "true"
    prometheus.io/path: "/metrics"
    prometheus.io/port: "http"
spec:
  type: ClusterIP
  ports:
  - port: 80
    protocol: TCP
    name: http
  selector:
    app: httpserver
```

Prometheus 인스턴스를 통한 서비스 모니터링 메트릭 수집

Promtheus 인스턴스 수집 규칙 또는 ServiceMonitor를 통해 서비스에서 연 모니터링 메트릭을 수집하도록 Promtheus 인스턴스를 구성할 수 있습니다.

방법1: Promtheus 인스턴스 수집 규칙 구성

아래와 같이 Promtheus 인스턴스 수집 규칙의 구성 파일에 다음 수집 규칙을 추가합니다.

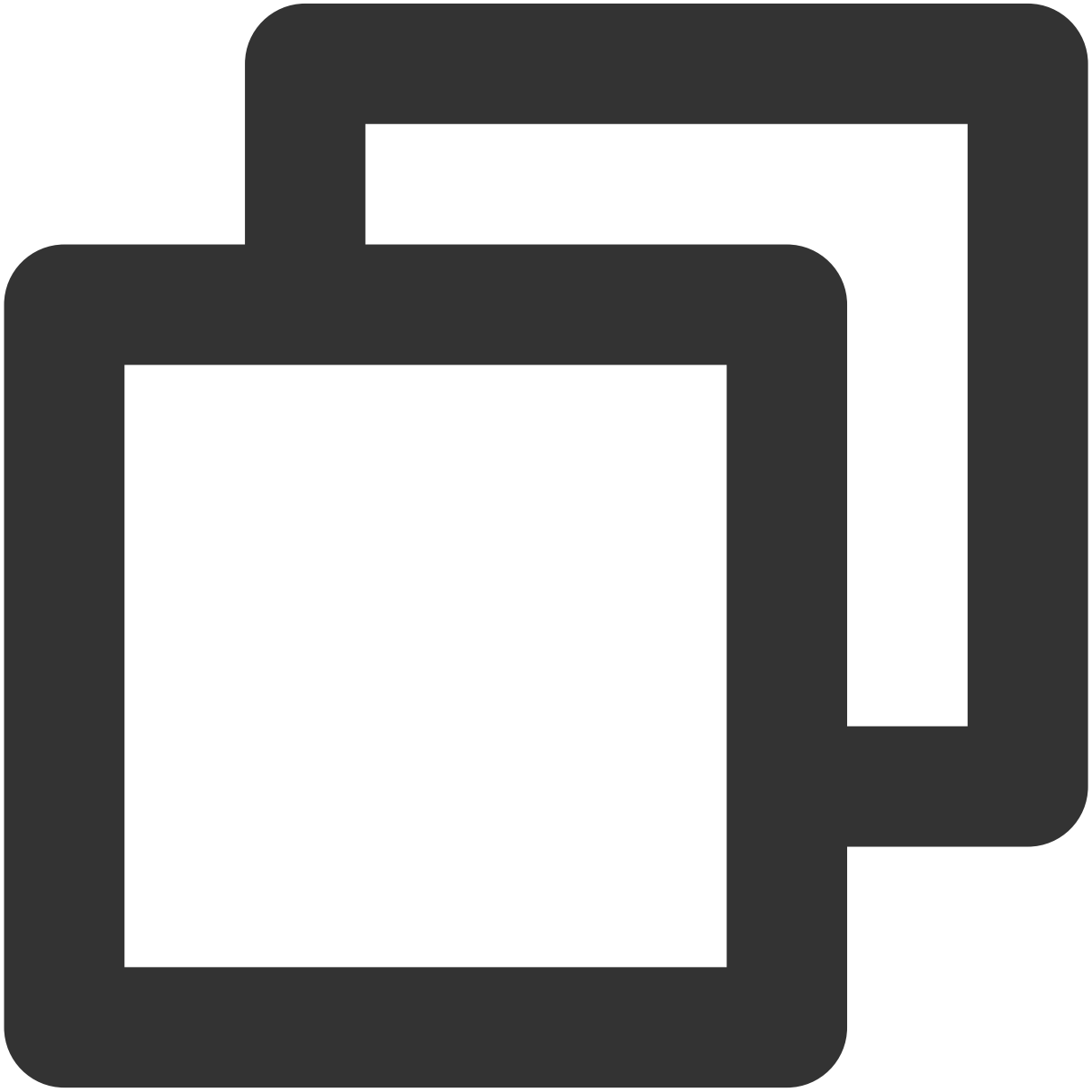


```
- job_name: httpserver
    scrape_interval: 5s
    kubernetes_sd_configs:
    - role: endpoints
        namespaces:
            names:
            - httpserver
    relabel_configs:
    - action: keep
        source_labels:
        - __meta_kubernetes_service_label_app
```

```
            regex: httpserver
        - action: keep
            source_labels:
            - __meta_kubernetes_endpoint_port_name
            regex: http
```

방법2: ServiceMonitor 구성

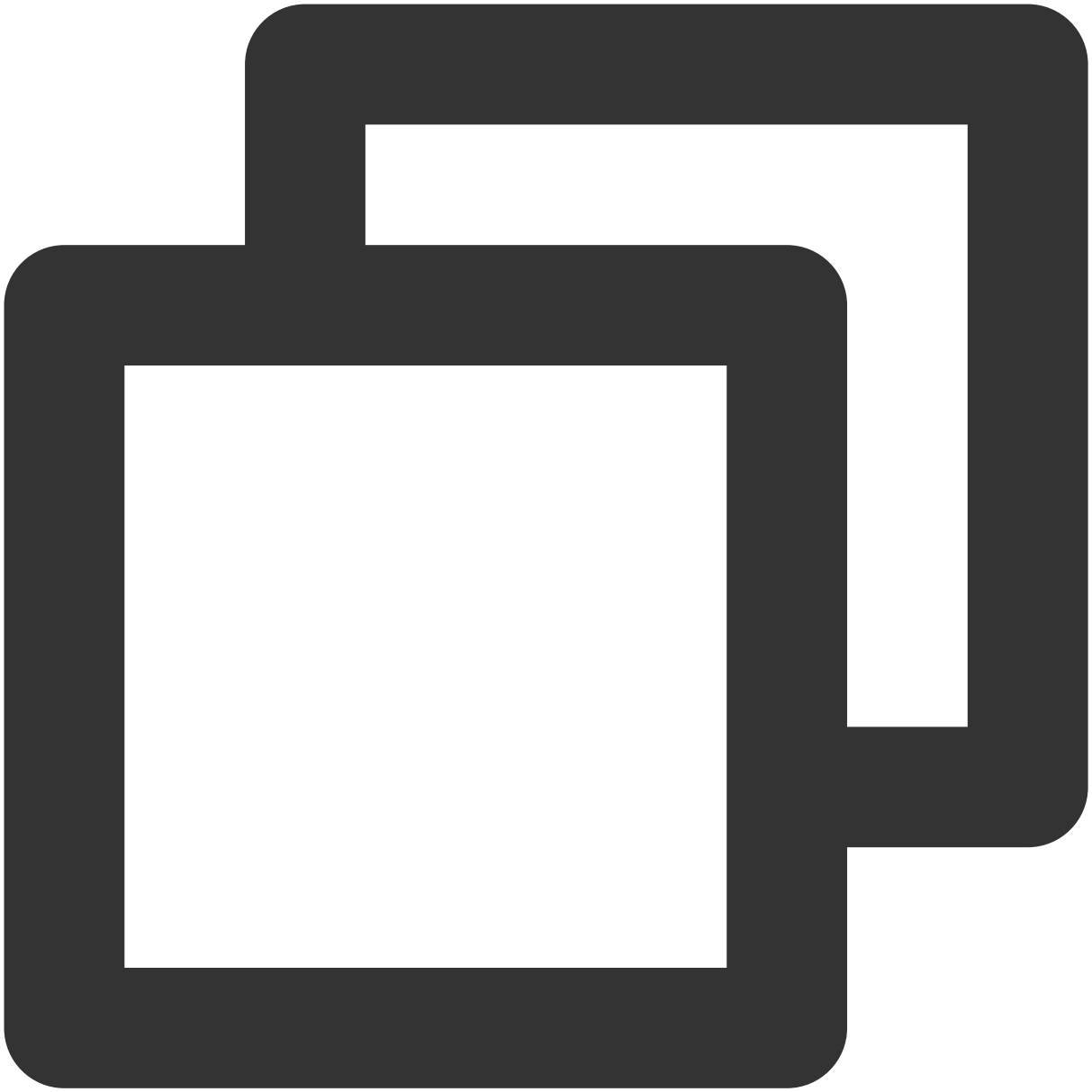
prometheus-operator가 설치된 경우 아래와 같이 ServiceMonitor의 CRD 객체를 생성하여 Prometheus 인스턴스를 구성할 수 있습니다.



```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: httpserver
spec:
  endpoints:
  - port: http
    interval: 5s
  namespaceSelector:
    matchNames:
    - httpserver
  selector:
    matchLabels:
      app: httpserver
```

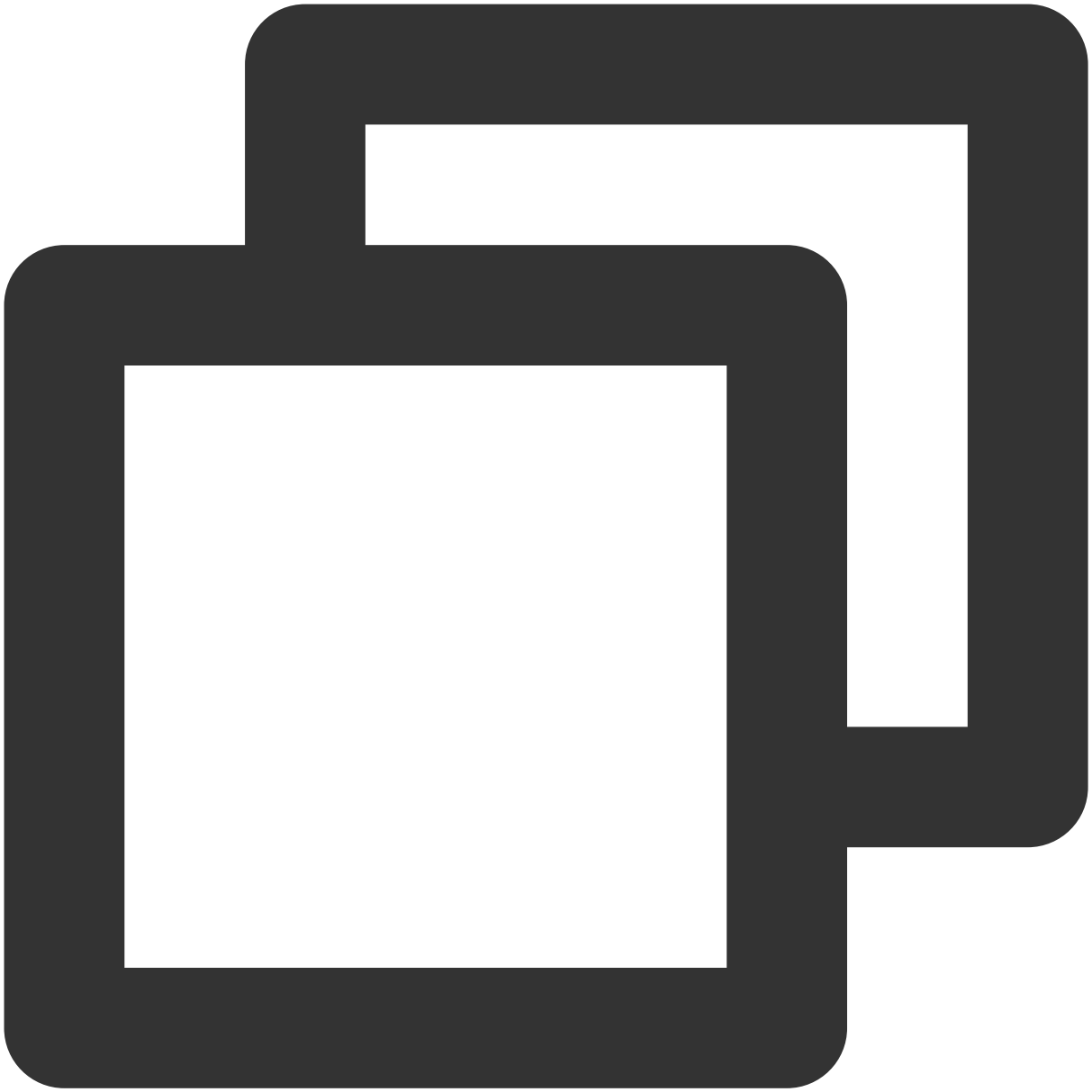
prometheus-adapter 설치

1. Helm을 사용하여 prometheus-adapter를 설치합니다. 설치하기 전에 사용자 정의 메트릭을 확인하고 구성하십시오. 상기 모니터링 메트릭 열기의 예시에 따르면 HTTP 요청을 기록하기 위해 서비스에서 `httpserver_requests_total` 메트릭을 사용하므로 아래와 같이 다음 PromQL을 통해 각 서비스 Pod의 QPS를 계산할 수 있습니다.



```
sum(rate(http_requests_total[2m])) by (pod)
```

2. prometheus-adapter의 구성으로 변환합니다. 다음과 같이 `values.yaml` 을 생성합니다.



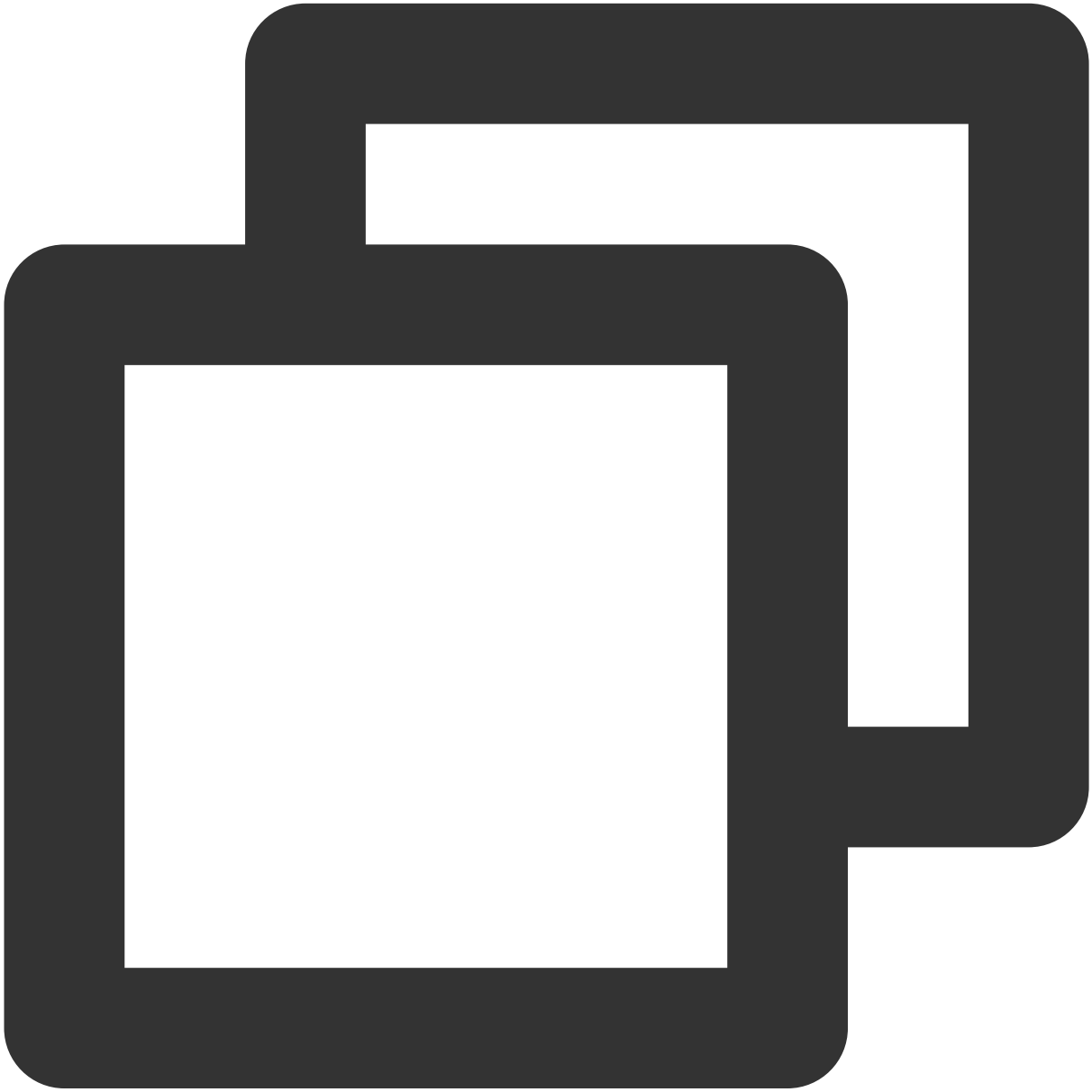
```
rules:
   default: false
   custom:
   - seriesQuery: 'httpserver_requests_total'
     resources:
       template: <<.Resource>>
     name:
       matches: "httpserver_requests_total"
       as: "httpserver_requests_qps" # PromQL에서 계산한 QPS 메트릭
     metricsQuery: sum(rate(<<.Series>>{<<.LabelMatchers>>}[1m])) by (<<.GroupBy>>)
prometheus:
```

```
    url: http://prometheus.monitoring.svc.cluster.local # Prometheus API 주소 교체(포트
    port: 9090
```

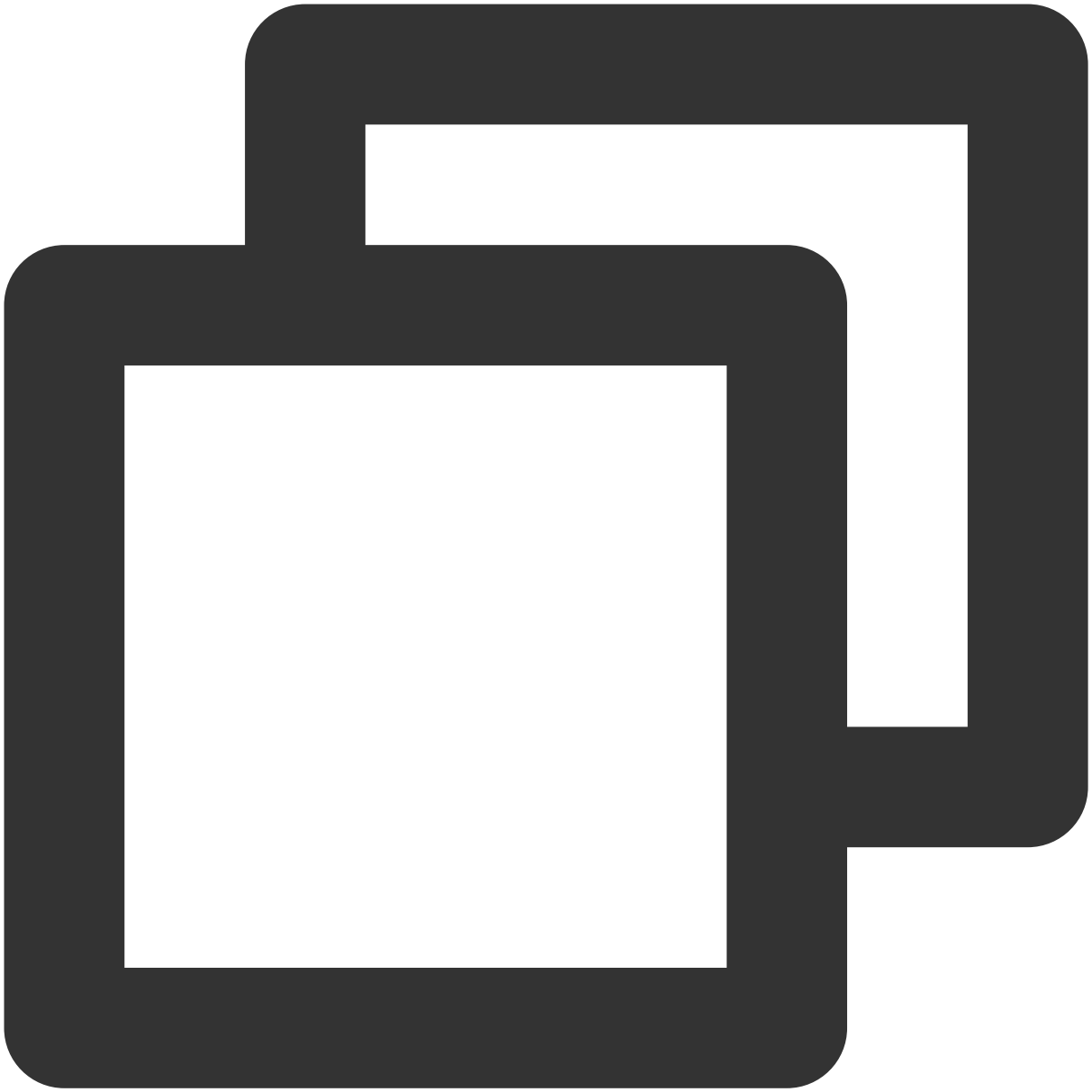
3. 다음 Helm 명령을 실행하여 아래와 같이 prometheus-adapter를 설치합니다.

주의사항

설치하기 전에 다음 명령을 사용하여 TKE의 등록된 Custom Metrics API를 삭제해야 합니다.



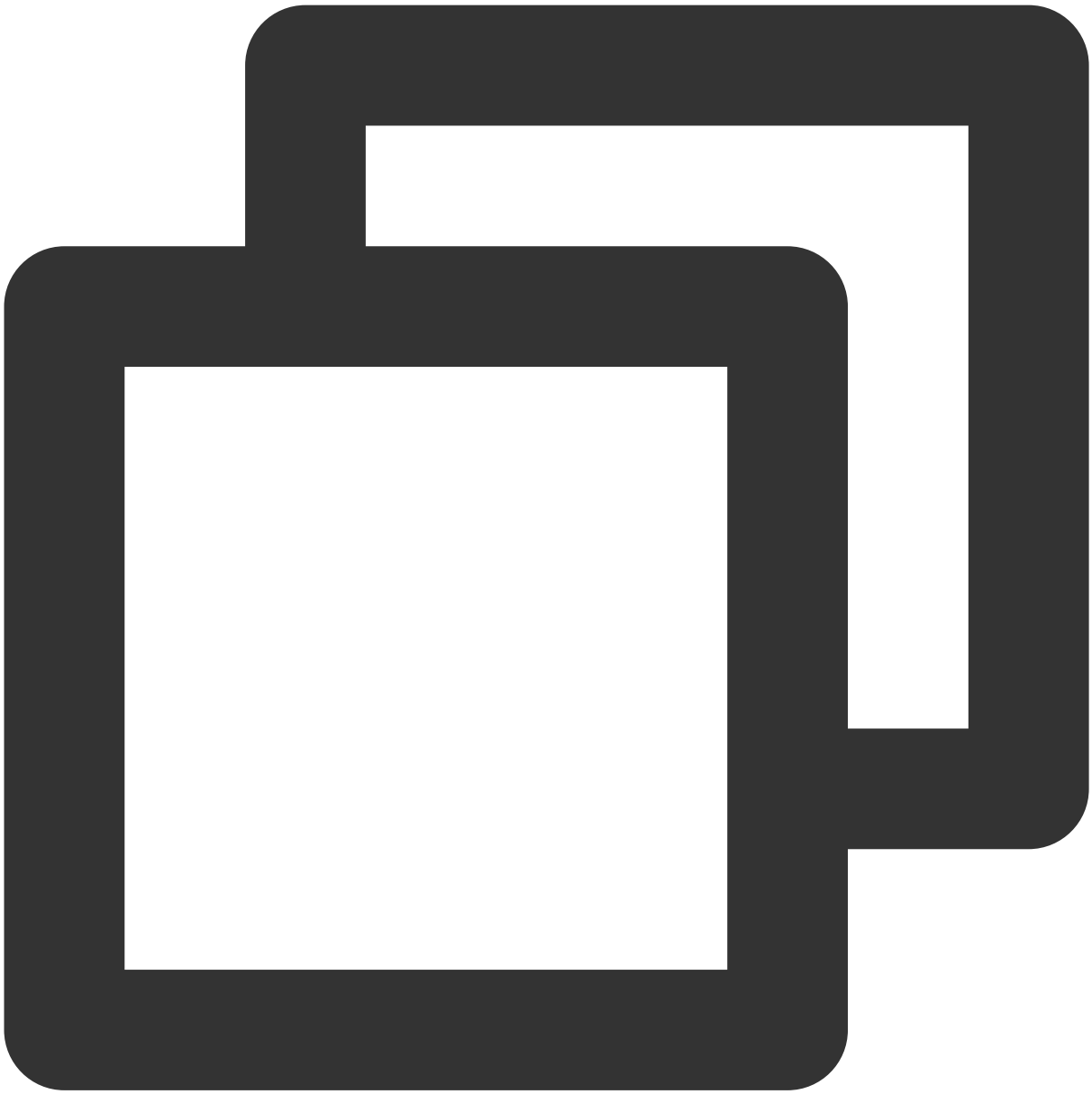
```
kubectl delete apiservice v1beta1.custom.metrics.k8s.io
```



```
helm repo add prometheus-community https://prometheus-community.github.io/helm-char
helm repo update
# Helm 3
helm install prometheus-adapter prometheus-community/prometheus-adapter -f values.y
# Helm 2
# helm install --name prometheus-adapter prometheus-community/prometheus-adapter -f
```

테스트 및 검증

올바르게 설치된 경우, 다음 명령을 실행하여 아래와 같이 Custom Metrics API가 반환하는 구성된 QPS 관련 메트릭을 볼 수 있습니다.



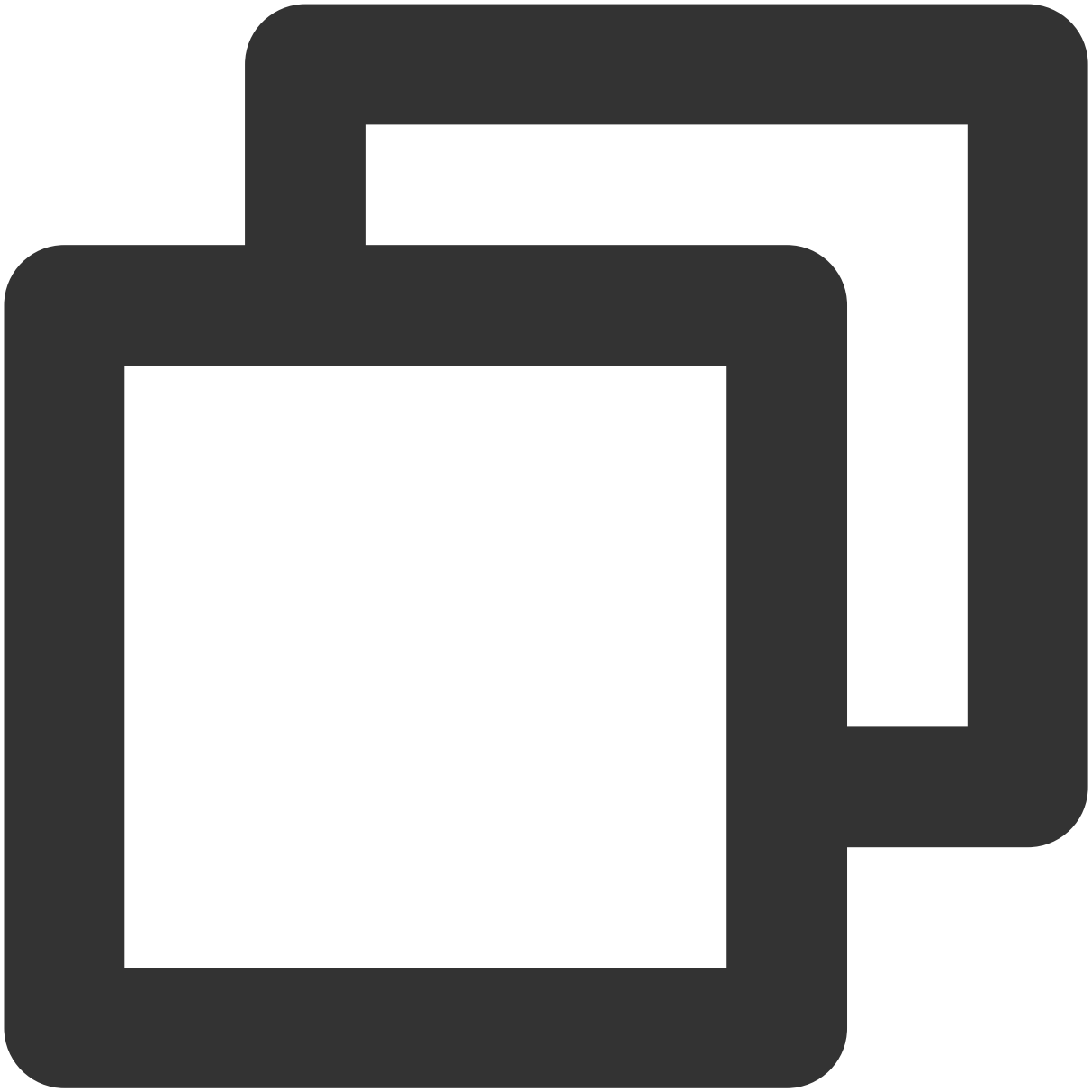
```
$ kubectl get --raw /apis/custom.metrics.k8s.io/v1beta1
{
  "kind": "APIResourceList",
  "apiVersion": "v1",
  "groupVersion": "custom.metrics.k8s.io/v1beta1",
  "resources": [
    {
      "name": "jobs.batch/httpserver_requests_qps",
```

```
      "singularName": "",
      "namespaced": true,
      "kind": "MetricValueList",
      "verbs": [
        "get"
      ]
    },
    {
      "name": "pods/httpserver_requests_qps",
      "singularName": "",
      "namespaced": true,
      "kind": "MetricValueList",
      "verbs": [
        "get"
      ]
    },
    {
      "name": "namespaces/httpserver_requests_qps",
      "singularName": "",
      "namespaced": false,
      "kind": "MetricValueList",
      "verbs": [
        "get"
      ]
    }
  ]
}
```

다음 명령어를 실행하여 아래와 같이 Pod의 QPS 값을 확인합니다.

설명

다음 예시에서 값은 500m이며 이는 QPS 값이 0.5 요청/초임을 의미합니다.

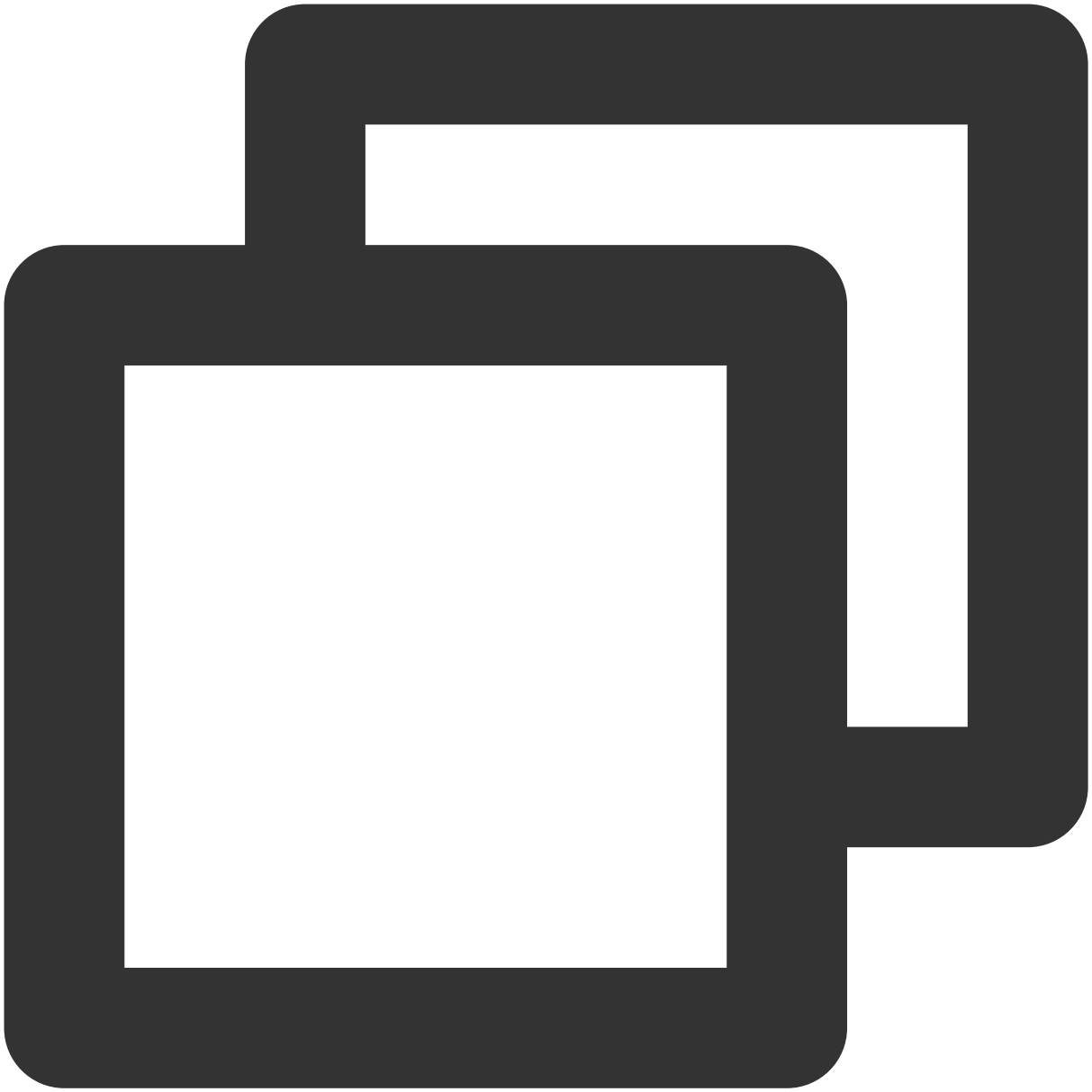


```
$ kubectl get --raw /apis/custom.metrics.k8s.io/v1beta1/namespaces/httpserver/pods/
{
  "kind": "MetricValueList",
  "apiVersion": "custom.metrics.k8s.io/v1beta1",
  "metadata": {
    "selfLink": "/apis/custom.metrics.k8s.io/v1beta1/namespaces/httpserver/pods/%2A
  },
  "items": [
    {
      "describedObject": {
        "kind": "Pod",
```

```
        "namespace": "httpserver",
        "name": "httpserver-6f94475d45-7rln9",
        "apiVersion": "/v1"
      },
      "metricName": "httpserver_requests_qps",
      "timestamp": "2020-11-17T09:14:36Z",
      "value": "500m",
      "selector": null
    }
  ]
}
```


HPA 테스트

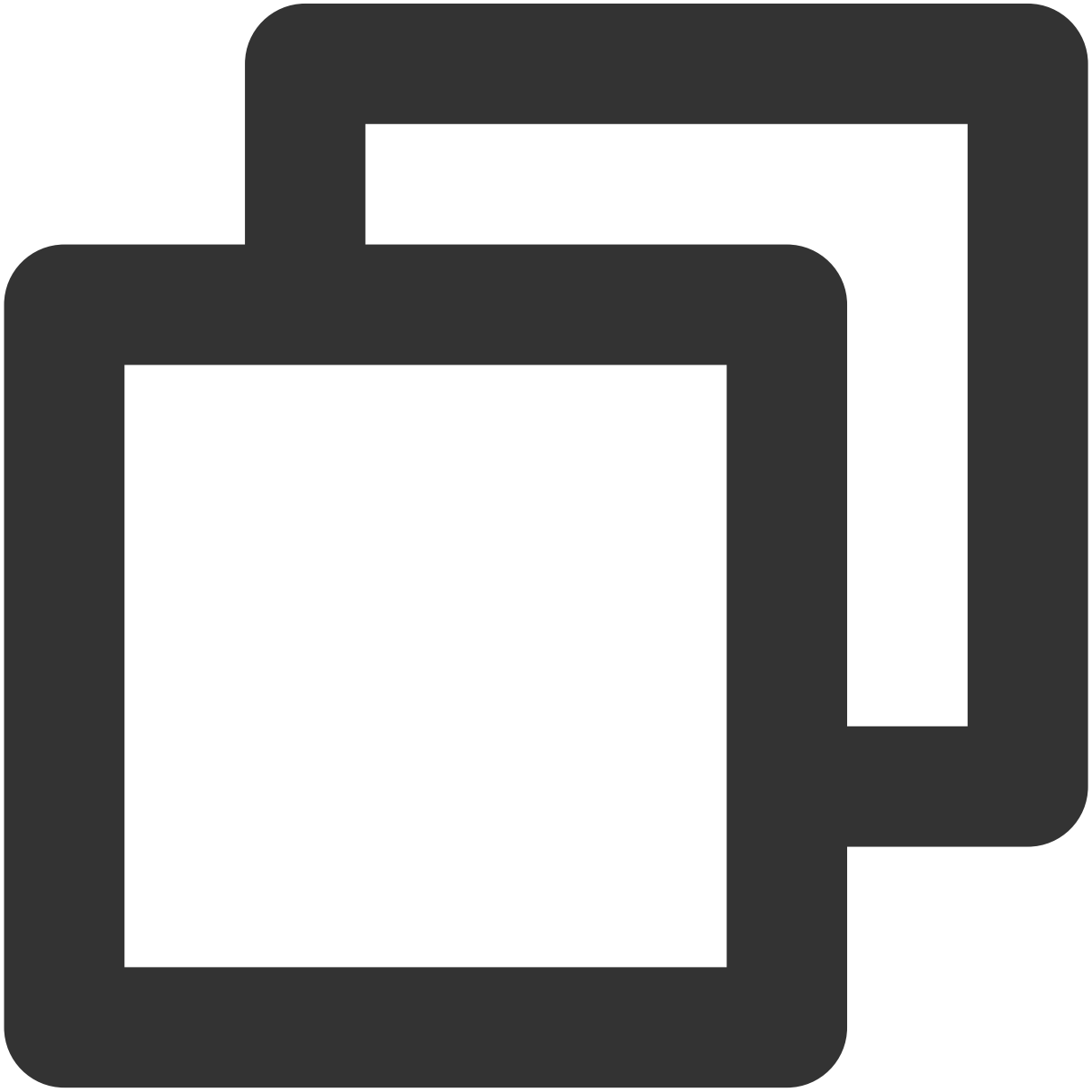
각 서비스 Pod의 평균 QPS 50 도달 시 스케일 아웃이 트리거되는 경우, 최소 및 최대 복제본 수는 각각 1 및 1000이며 구성 예시는 다음과 같습니다.



```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: httpserver
  namespace: httpserver
spec:
  minReplicas: 1
  maxReplicas: 1000
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
```

```
    name: httpserver
  metrics:
  - type: Pods
    pods:
      metric:
        name: httpserver_requests_qps
      target:
        averageValue: 50
        type: AverageValue
```

다음 명령을 실행하여 서비스를 테스트하고 아래와 같이 스케일 아웃이 트리거되는지 관찰합니다.



```
$ kubectl get hpa
NAME         REFERENCE                TARGETS     MINPODS   MAXPODS   REPLICAS   AGE
httpserver   Deployment/httpserver   83933m/50   1         1000      2          18h
$ kubectl get pods
NAME                          READY   STATUS              RESTARTS   AGE
httpserver-6f94475d45-47d5w   1/1     Running             0          3m41s
httpserver-6f94475d45-7rln9   1/1     Running             0          37h
httpserver-6f94475d45-6c5xm   0/1     ContainerCreating   0          1s
httpserver-6f94475d45-wl78d   0/1     ContainerCreating   0          1s
```

스케일 아웃이 정상적으로 트리거되면 HPA가 서비스 사용자 지정 메트릭을 기반으로 오토 스케일링을 구현했음을 의미합니다.

TKE에서 HPA를 사용하여 비즈니스 오토 스케일링 구현

최종 업데이트 날짜: : 2023-04-28 15:30:11

개요

Kubernetes Pod용 HPA(Horizontal Pod Autoscaler)는 CPU 사용량, 메모리 사용량 및 기타 사용자 지정 메트릭을 기반으로 Pod 복제본 수를 자동으로 조정하여 워크로드 서비스의 전체 수준을 사용자 정의 대상 값과 일치시킬 수 있습니다. 이 문서는 TKE의 HPA 기능을 소개하고 이 기능을 사용하여 Pod의 오토 스케일링을 구현하는 방법을 설명합니다.

사용 사례

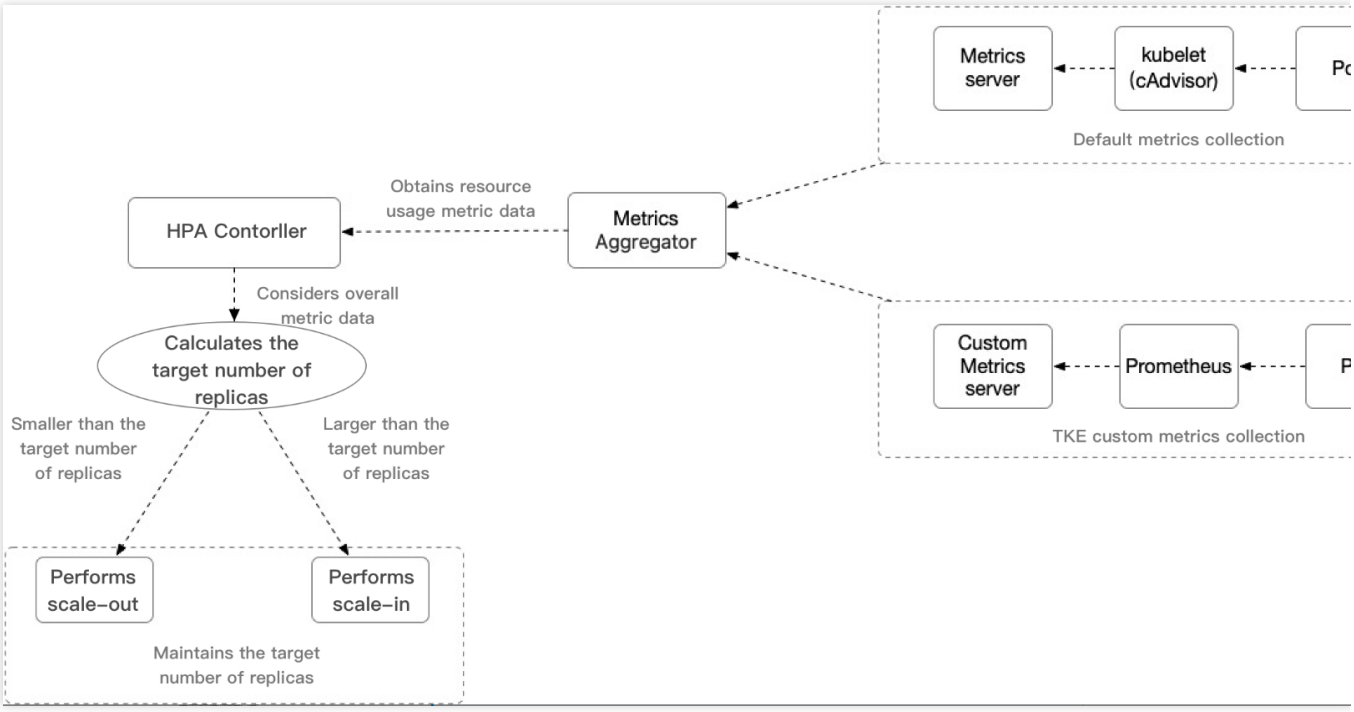
HPA 기능은 TKE에 유연한 어댑티브 기능을 제공하므로, 사용자 정의 범위 내에서 Pod 복제본 수를 빠르게 늘려 서비스 로드의 급격한 증가에 대처하고 서비스 로드가 감소할 때 스케일 인하여 다른 서비스에 대한 컴퓨팅 리소스를 절약할 수 있습니다. 전체 프로세스는 자동이며 수동 개입이 필요하지 않습니다. 전자상거래 서비스, 온라인 교육, 금융 서비스 등 서비스 변동이 크고 서비스 수가 많으며 스케일링이 빈번한 서비스 시나리오에 적합합니다.

원리 개요

HPA 기능은 Kubernetes API 리소스 및 컨트롤러에 의해 구현됩니다. 리소스는 메트릭을 사용하여 컨트롤러의 동작을 결정하는 반면, 컨트롤러는 Pod 리소스 사용량에 따라 서비스 Pod의 복제본 수를 주기적으로 조정합니다. 워크로드 수준을 사용자 정의 대상 값과 일치시킵니다. 다음 이미지는 스케일링 프로세스를 보여줍니다.

주의사항

Pod에 대한 수평적 오토 스케일링은 DaemonSet 리소스와 같이 스케일링할 수 없는 객체에는 적용되지 않습니다.



주요 내용:

HPA Controller: HPA 스케일링 로직을 제어하는 제어 컴포넌트입니다.

Metrics Aggeregator: 일반적으로 컨트롤러는 일련의 집계 API(`metrics.k8s.io` , `custom.metrics.k8s.io` 및 `external.metrics.k8s.io`)에서 메트릭 값을 가져옵니다. `metrics.k8s.io` API는 일반적으로 Metrics 서버에서 제공됩니다. 커뮤니티 에디션은 기본 CPU 및 메모리 메트릭 유형을 제공할 수 있습니다. 커뮤니티 에디션과 비교할 때 TKE에서 사용하는 사용자 지정 Metrics Server 컬렉션은 CPU, 메모리, 디스크, 네트워크 및 GPU 메트릭과 같은 관련 메트릭을 제공하는 광범위한 HPA 메트릭 트리거 유형을 지원합니다. 자세한 내용은 TKE Auto-Scaling 메트릭을 참고하십시오.

설명

컨트롤러는 Heapster에서 메트릭을 얻을 수도 있습니다. 그러나 Kubernetes 1.11부터는 컨트롤러가 더 이상 Heapster에서 메트릭을 가져올 수 없습니다.

대상 복제본 수 계산을 위한 HPA 알고리즘: TKE HPA 스케일링 알고리즘은 작동 원리를 참고하십시오. 알고리즘에 대한 자세한 내용은 알고리즘 세부 정보를 참고하십시오.

전제 조건

Tencent Cloud 계정을 등록합니다.

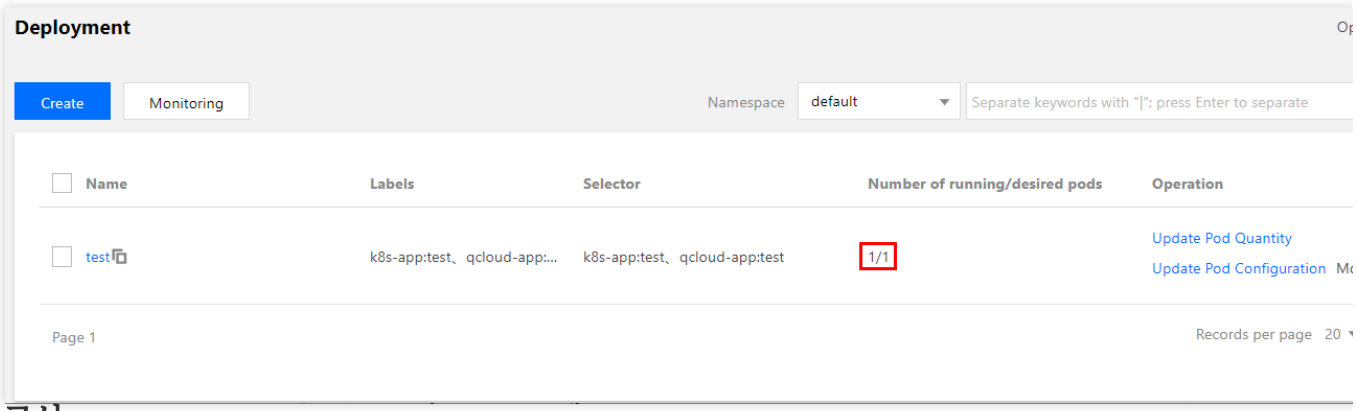
Tencent Cloud TKE 콘솔에 로그인했습니다.

클러스터를 생성합니다. TKE 클러스터 생성 방법에 대한 자세한 내용은 클러스터 생성을 참고하십시오.

작업 단계

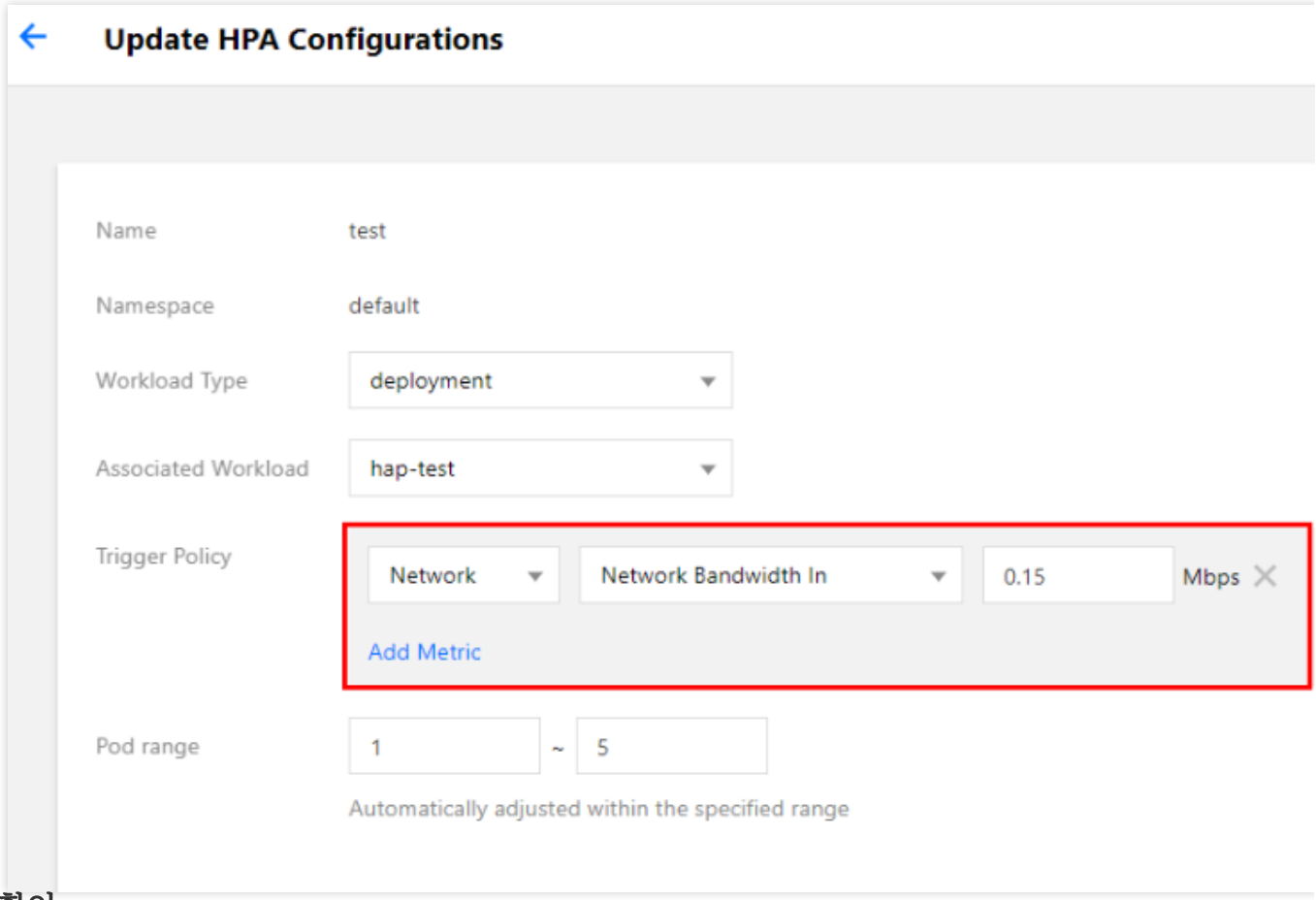
테스트 워크로드 배포

여기서는 Deployment 유형 워크로드를 예로 사용합니다. Web 서비스의 'hpa-test' 워크로드로 설정된 서비스 유형으로 홀수의 복제본을 생성합니다. TKE 콘솔에서 Deployment 유형 워크로드를 생성하는 방법에 대한 자세한 내용은 Deployment 관리를 참고하십시오. 다음 이미지는 이 예시의 생성 결과를 보여줍니다. (이 글의 스크린샷 정보는 콘솔의 실제 인터페이스보다 뒤쳐질 수 있으며 콘솔의 실제 디스플레이가 우선합니다):



HPA 구성

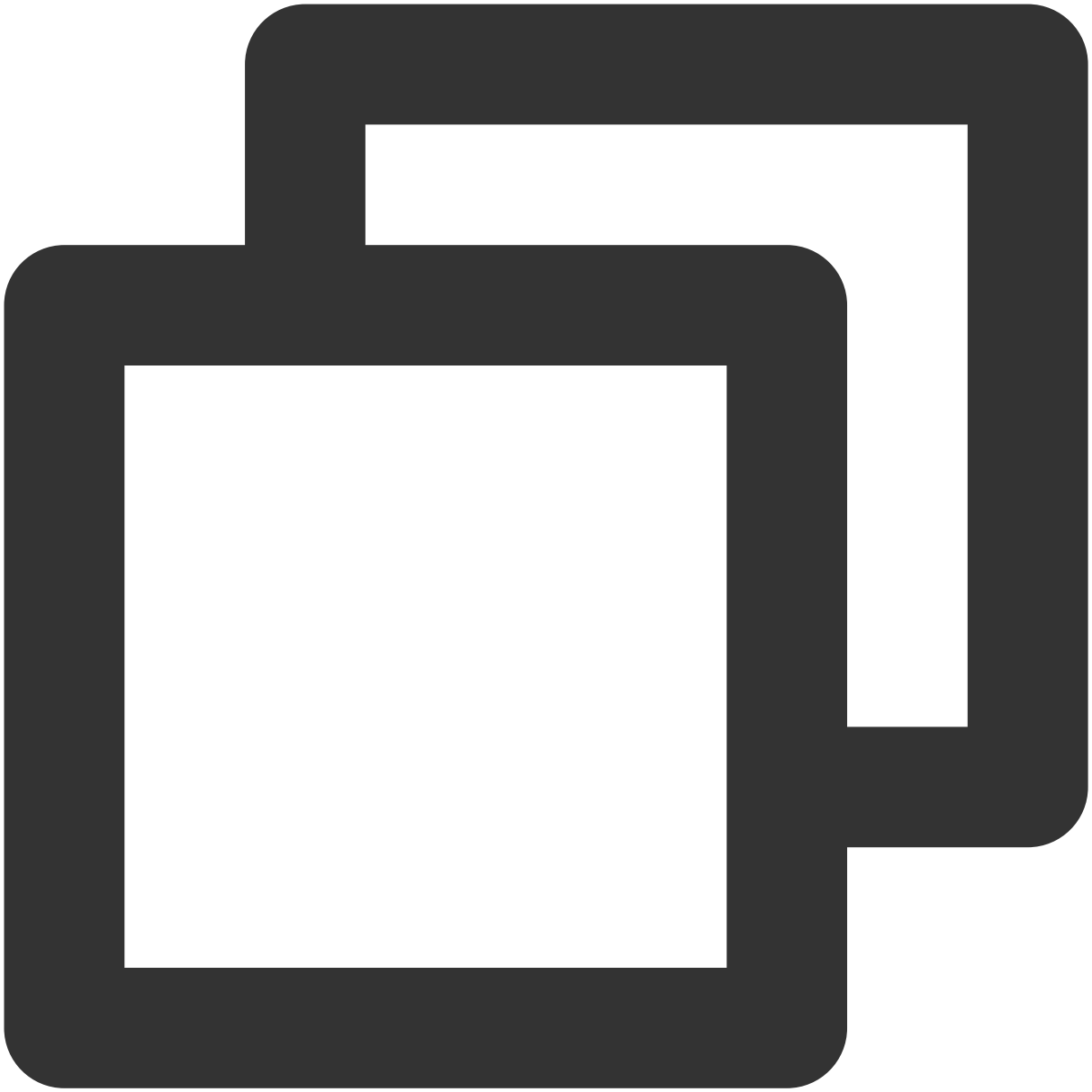
TKE 콘솔에서 테스트 워크로드를 HPA 구성과 바인딩합니다. HPA 구성을 바인딩하는 방법에 대한 자세한 내용은 HPA 지침을 참고하십시오. 본 문서에서는 아래 이미지와 같이 네트워크 이그레스 대역폭이 0.15Mbps(150Kbps)에 도달하면 스케일 아웃이 트리거되는 정책을 구성하는 예를 들어 설명합니다.



기능 확인

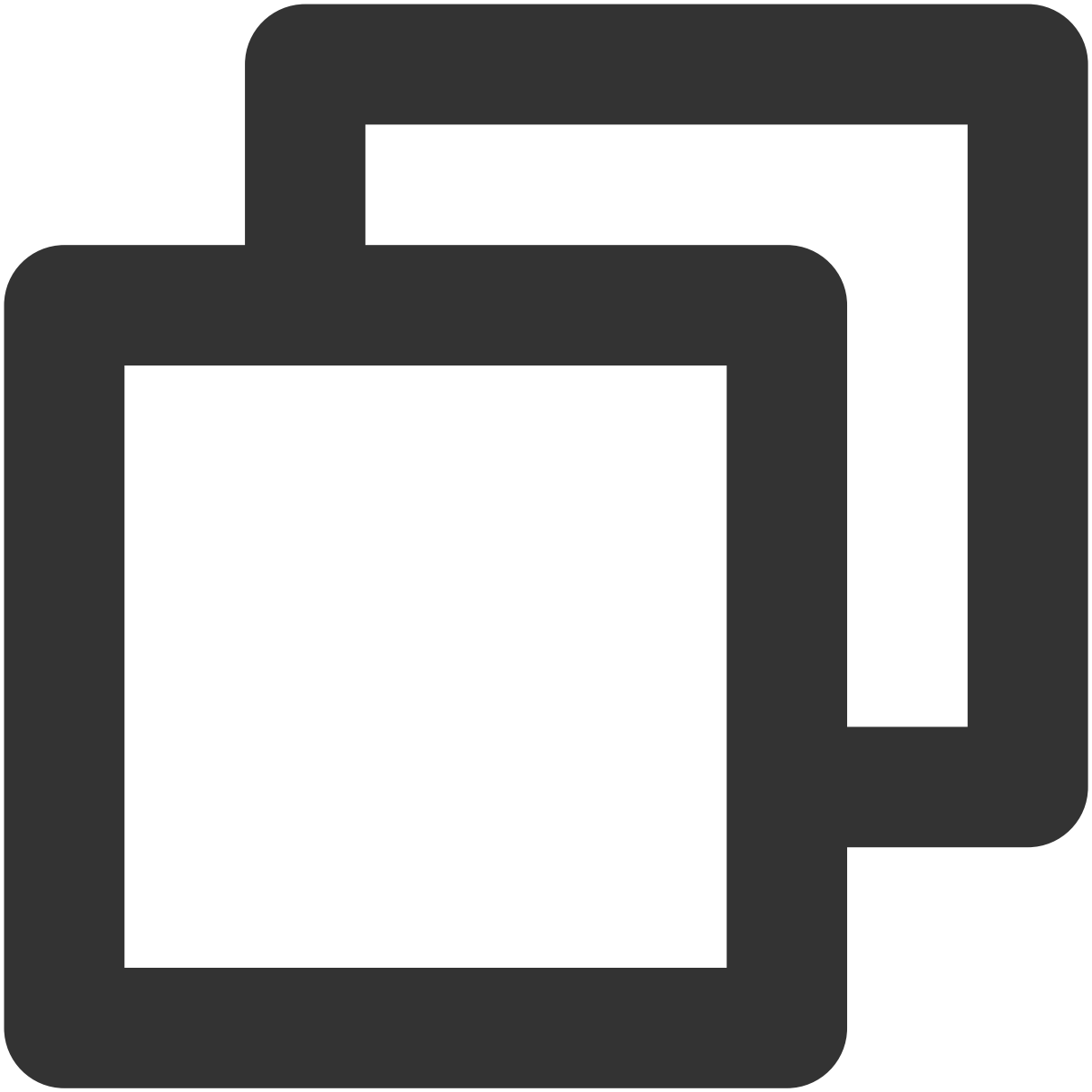
스케일 아웃 프로세스 시뮬레이션

구성된 HPA 기능(시뮬레이션된 클라이언트)을 테스트하기 위해 다음 명령을 실행하여 클러스터에서 임시 Pod를 시작합니다.



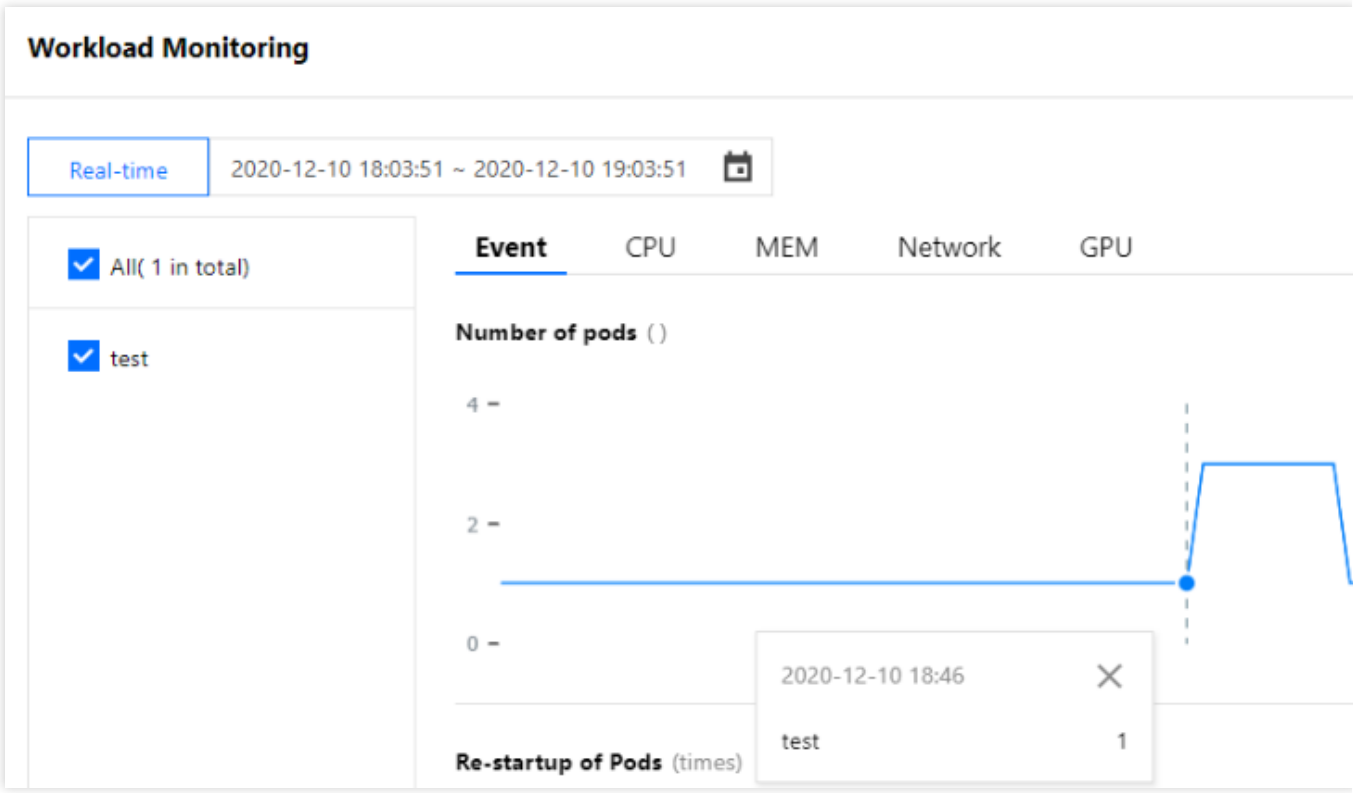
```
kubectl run -it --image alpine hpa-test --restart=Never --rm /bin/sh
```

임시 Pod에서 다음 명령을 실행하여 단기간에 "hpa-test" 서비스에 액세스하는 많은 수의 요청으로 인해 이그레스 트래픽 대역폭이 증가하는 상황을 시뮬레이션합니다.



```
# hpa-test.default.svc.cluster.local 은 클러스터에 있는 서비스의 도메인 이름입니다. 스크립트
while true; do wget -q -O - hpa-test.default.svc.cluster.local; done
```

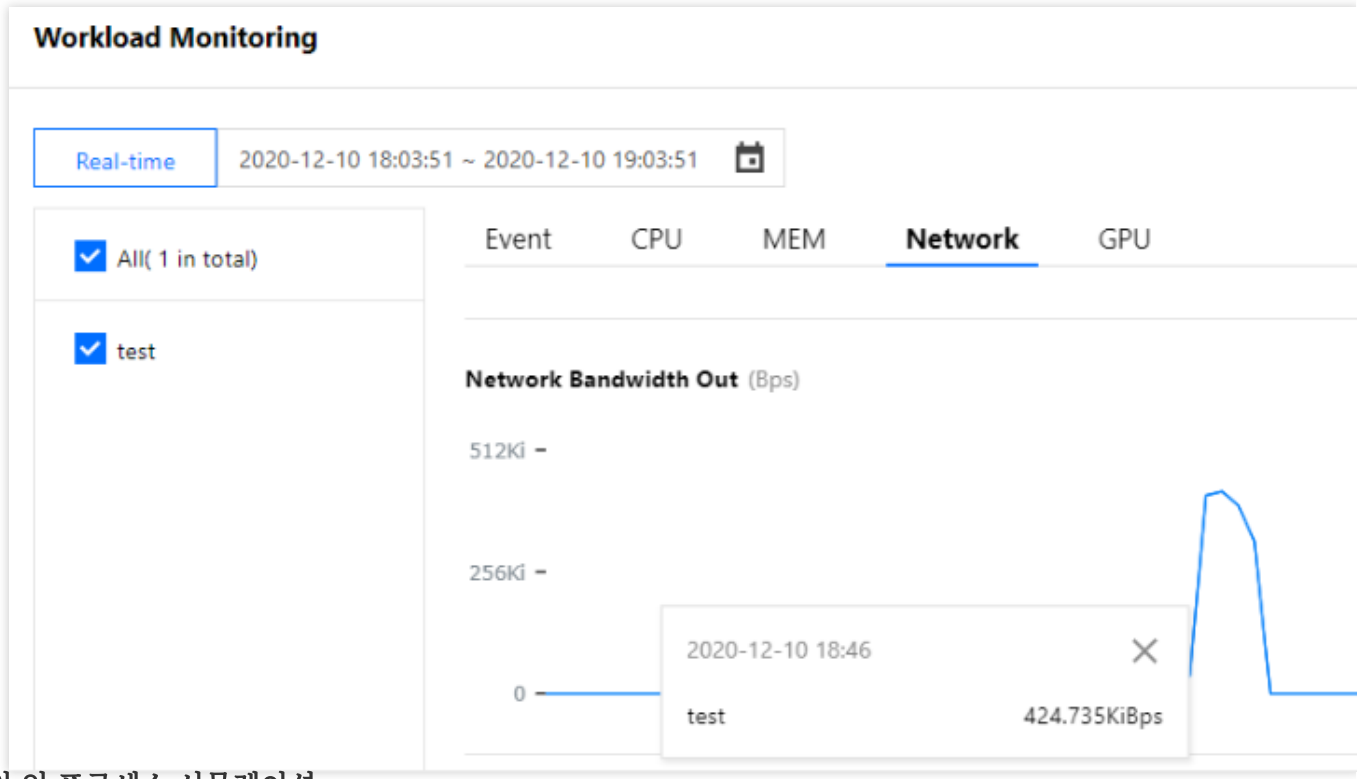
테스트 Pod에서 요청 시뮬레이션 명령을 실행한 후 워크로드의 모니터링되는 Pod 수를 관찰합니다. 워크로드의 복제본 수가 16:21에 2개로 증가하는 것을 볼 수 있습니다. 이는 아래 이미지와 같이 HPA 스케일 아웃 이벤트가 트리거되었음을 나타냅니다. (본문의 스크린샷 정보는 콘솔의 실제 인터페이스보다 뒤쳐질 수 있습니다. 실제 콘솔을 기준으로 하십시오):



그러면 워크로드의 네트워크 이그레스 대역폭 모니터링을 통해 16:21에 네트워크 이그레스 대역폭이 약 196Kbps로 증가하여 HPA에서 설정한 네트워크 이그레스 대역폭의 목표 값을 초과하는 것을 확인할 수 있습니다. 이는 설정된 목표 값을 충족하기 위해 복제본을 추가하기 위해 HPA 스케일링 알고리즘이 트리거되었음을 나타냅니다. 따라서 아래 이미지와 같이 워크로드의 복제본 수가 2로 변경되었습니다. (본문의 스크린샷 정보는 콘솔의 실제 인터페이스보다 뒤쳐질 수 있습니다. 실제 콘솔을 기준으로 하십시오):

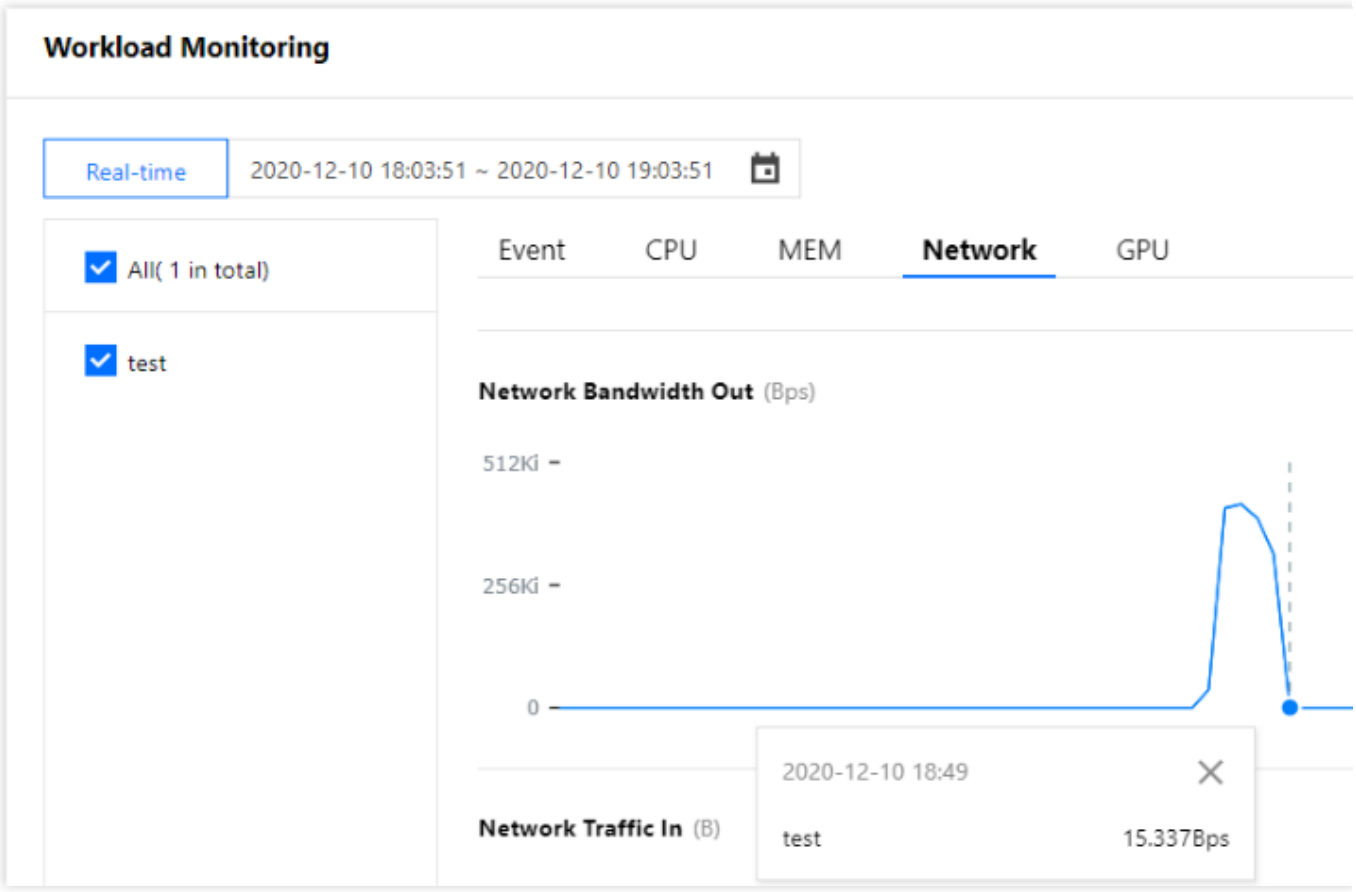
주의사항

HPA 스케일링 알고리즘은 수식 계산에 의존하여 스케일링 로직을 제어하는 것이 아니라 스케일 아웃/인이 필요한지 여부를 결정하기 위해 여러 차원을 고려합니다. 따라서 실제 구현은 예상과 약간 다를 수 있습니다. 자세한 내용은 알고리즘 세부 정보를 참고하십시오.

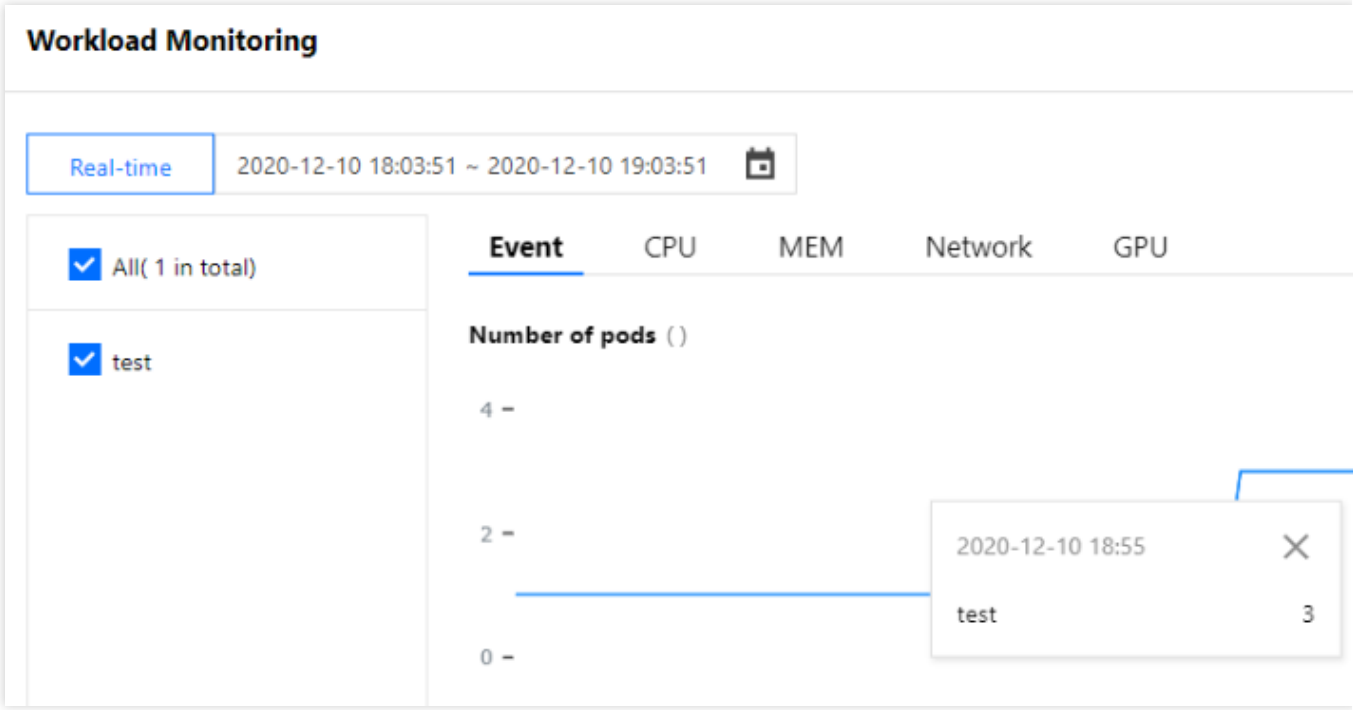


스케일 인 프로세스 시뮬레이션

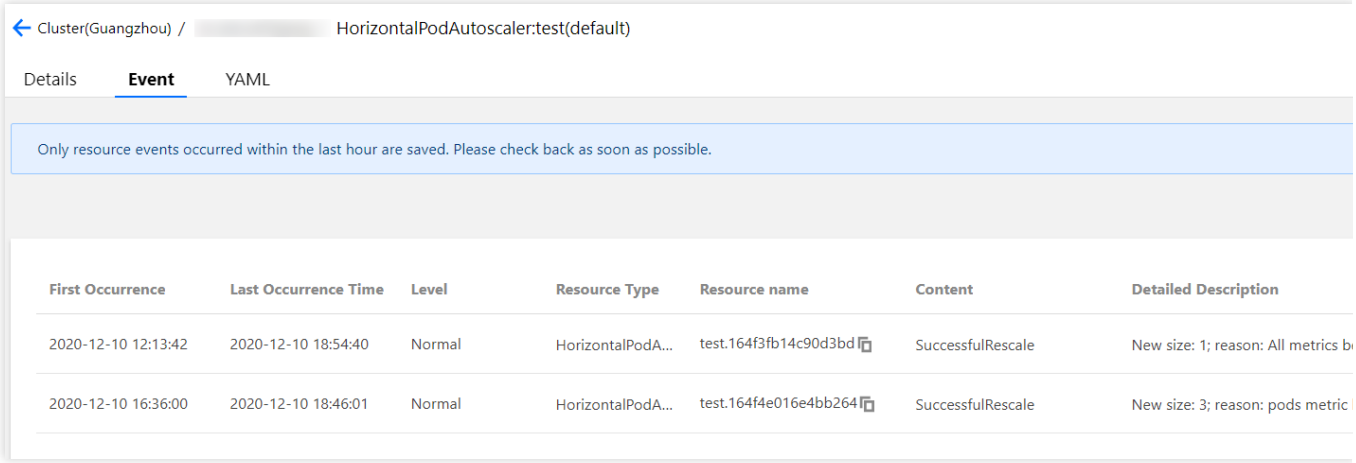
스케일 인 프로세스를 시뮬레이션할 때 요청 시뮬레이션 명령 실행을 약 16:24에 수동으로 중지합니다. 모니터링을 통해 네트워크 이그레스 대역폭이 스케일 아웃 전 수준으로 감소하는 것을 관찰할 수 있습니다. 이 때 HPA 로직에 따라 아래 이미지와 같이 워크로드 스케일 인을 위한 조건이 충족됩니다. (이 글의 스크린샷 정보는 콘솔의 실제 인터페이스보다 뒤쳐질 수 있습니다. 실제 콘솔을 기준으로 하십시오):



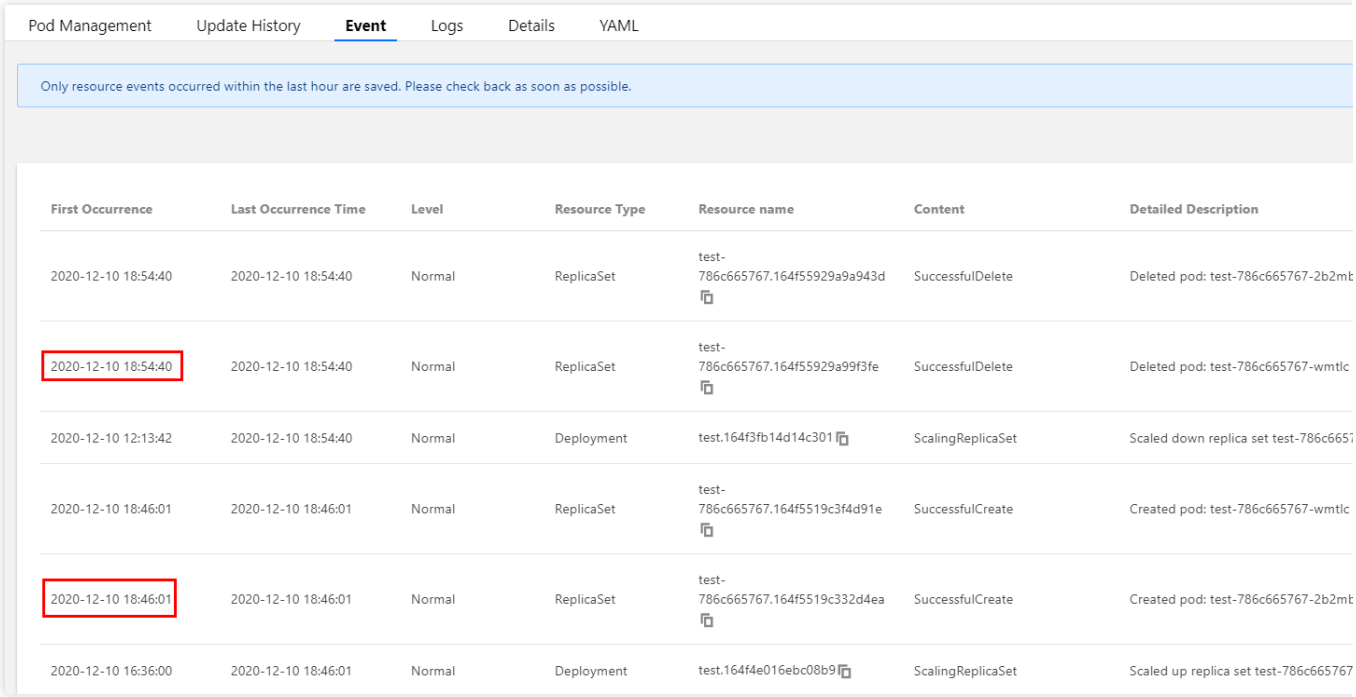
그러나 아래 이미지에 표시된 워크로드 Pod 수 모니터링에 따르면 워크로드는 16:30까지 HPA 스케일 인을 트리거하지 않았습니다. 이는 HPA가 트리거된 후 짧은 시간 내에 메트릭 변동으로 인한 빈번한 조정 작업을 방지하기 위해 기본 5분 허용 시간 알고리즘이 있기 때문입니다. 자세한 내용은 쿨링/딜레이 지원을 참고하십시오. 아래 이미지와 같이 명령이 중지된 지 5분이 지나면 HPA 스케일링 알고리즘에 따라 워크로드 복제본의 수가 초기 설정인 복제본 1개로 다시 줄어듭니다. (이 글의 스크린샷 정보는 콘솔의 실제 인터페이스보다 뒤쳐질 수 있습니다. 실제 콘솔을 기준으로 하십시오):



TKE에서 HPA 스케일링 이벤트가 발생하면 해당 HPA 인스턴스의 이벤트 목록에 해당 이벤트가 표시됩니다. 이벤트 알림 목록의 시간에는 '최초 발생 시간'과 '마지막 발생 시간'이 포함됩니다. '최초 발생 시간'은 동일한 이벤트가 처음 발생한 시간을 나타내고 '마지막 발생 시간'은 동일한 이벤트가 발생한 가장 최근 시간을 나타냅니다. 따라서 아래 그림의 이벤트 목록에서 볼 수 있듯이 '마지막 발생 시간' 필드는 이 예제의 스케일 아웃 이벤트에 대해 16:21:03, 스케일 인 이벤트에 대해 16.29:42을 표시합니다. 여기에 표시되는 시점은 워크로드 모니터링의 시점과 일치합니다.



또한 워크로드 이벤트 목록에는 HPA 발생 시 워크로드별 복제본 추가/삭제 이벤트도 기록됩니다. 아래 이미지와 같이 워크로드 스케일 인/아웃 시점이 HPA 이벤트 목록에 표시되는 시점과 일치합니다. 복제본 수가 증가한 시점은 16:21:03이고, 복제본 수가 감소한 시점은 16:29:42입니다.



요약

이 예시는 TKE의 HPA 기능을 보여주고 워크로드 HPA 조정을 트리거하기 위한 지표로 TKE 사용자 정의 지표 유형 네트워크 이그레스 대역폭을 사용하는 방법을 보여줍니다.
워크로드의 실제 메트릭 값이 HPA에서 설정한 목표 메트릭 값을 초과하면 HPA는 스케일 아웃 알고리즘에 따라 적절한 복제본 수를 계산하고 스케일 아웃을 구현합니다. 이렇게 하면 워크로드의 메트릭 수준이 기대치를 충족하고 워크로드가 건강하고 안정적인 방식으로 실행될 수 있습니다.
워크로드의 실제 메트릭 값이 HPA에서 구성한 대상 메트릭 값보다 훨씬 낮을 경우 HPA는 허용 시간이 만료될 때까지 기다린 후 적절한 복제본 수를 계산하여 스케일 인을 구현하고 유휴 리소스를 릴리스합니다. 이렇게 하면 리소스 활용도가 향상됩니다. 또한 프로세스 전반에 걸쳐 관련 이벤트가 HPA 및 워크로드 이벤트 목록에 기록되므로 전체 워크로드 스케일링 프로세스를 추적할 수 있습니다.

저장
CFS-Turbo 클래스 파일 시스템 정적 마운트
TKE Serverless 클러스터용 CFS-Turbo 정적 마운트

최종 업데이트 날짜: : 2023-04-26 19:04:19

사용 사례

TKE Serverless 클러스터용 CFS(Cloud File Storage) Turbo 스토리지를 마운트할 수 있습니다. 이 애드온은 Tencent Cloud CFS Turbo 파일 시스템을 독점 프로토콜 기반 워크로드에 마운트하는 데 사용됩니다. 현재 정적 구성만 지원됩니다. CFS 스토리지 유형에 대한 자세한 내용은 스토리지 유형 및 성능을 참고하십시오.

전제 조건

v1.14 이상의 TKE Serverless 클러스터를 생성합니다.

사용 순서

파일 시스템 생성

CFS Turbo 파일 시스템을 생성합니다. 자세한 내용은 파일 시스템 생성을 참고하십시오.

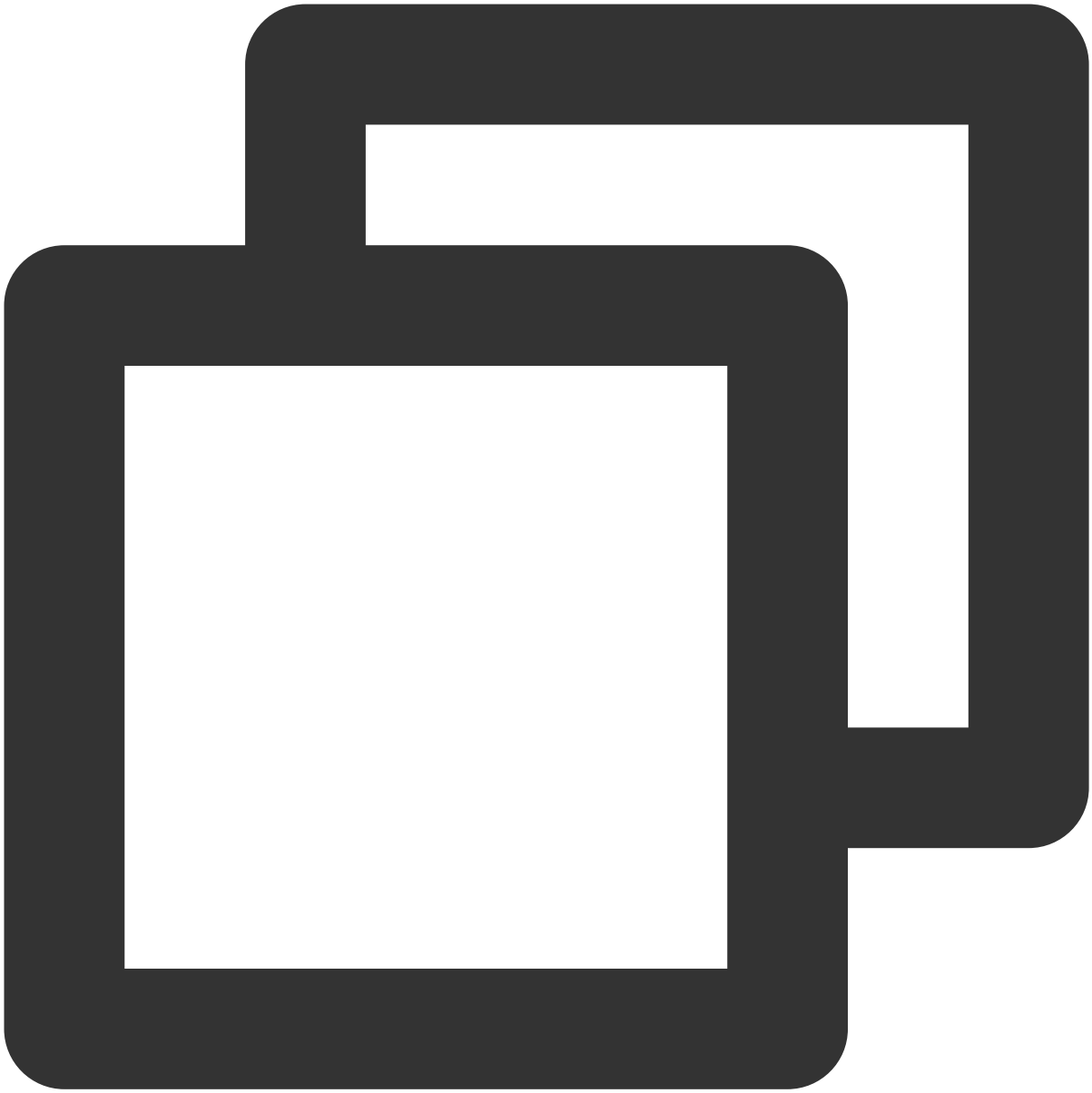
주의사항

파일 시스템이 생성된 후 클러스터 네트워크(vpc-xx)를 파일 시스템의 CCN과 연결해야 합니다. 파일 시스템 마운트 포인트 정보에서 확인할 수 있습니다.

Node Plugin 배포

1단계: csidriver.yaml 파일 생성

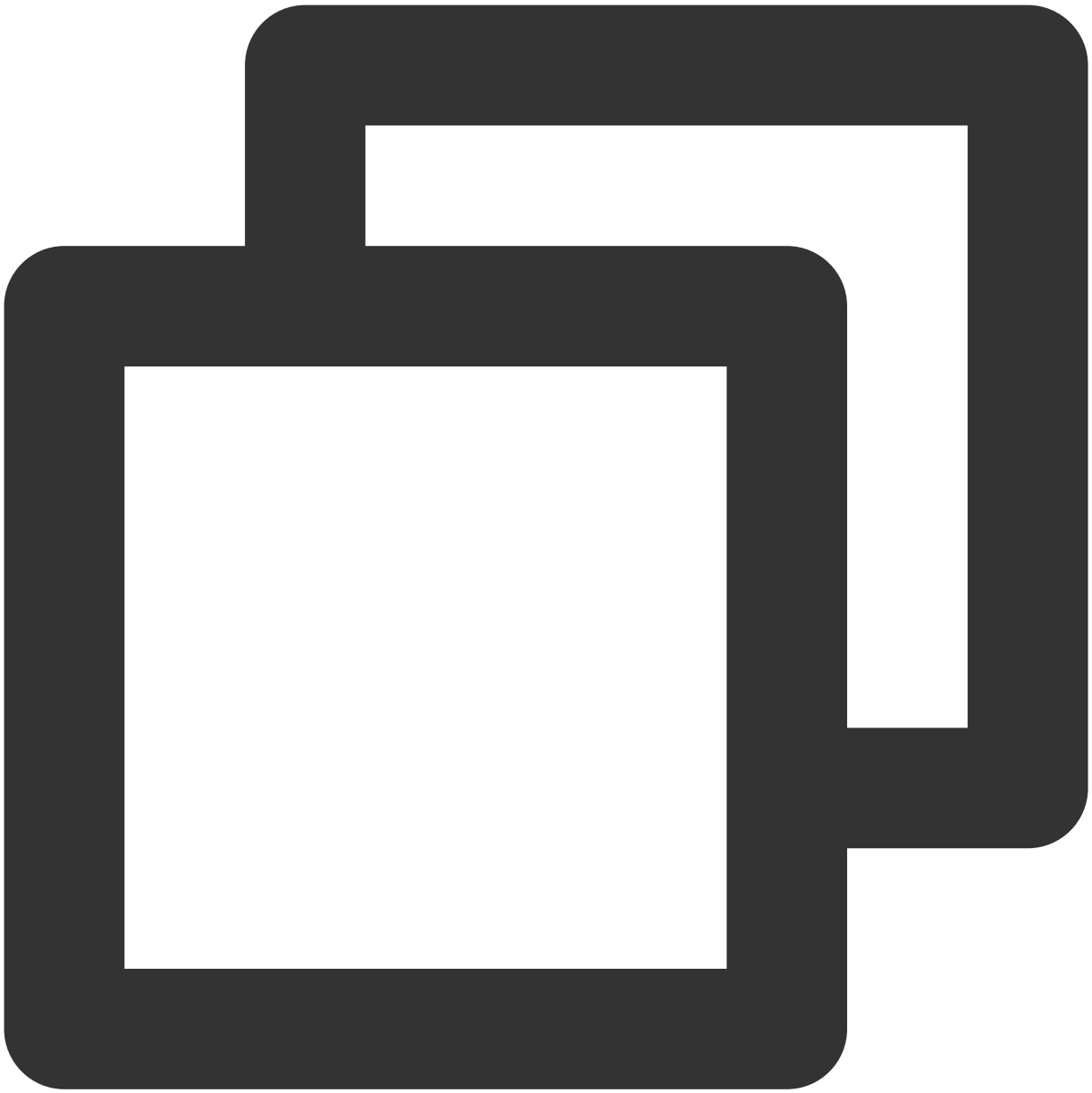
다음은 csidriver.yaml 파일의 예시입니다.



```
apiVersion: storage.k8s.io/v1
kind: CSIDriver
metadata:
  name: com.tencent.cloud.csi.cfsturbo
spec:
  attachRequired: false
  podInfoOnMount: false
```

2단계: csidriver 생성

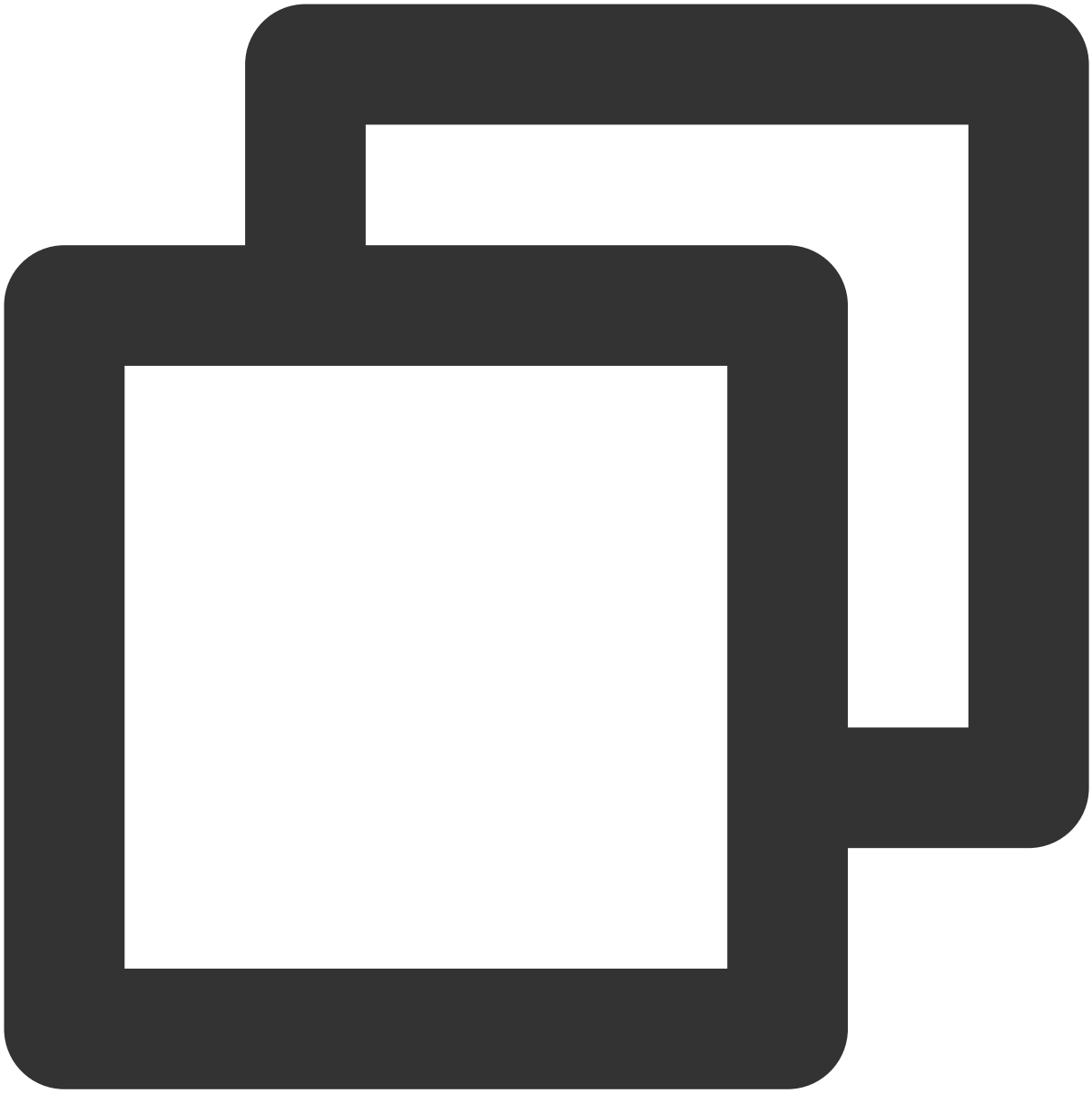
다음 명령을 실행하여 csidriver를 생성합니다.



```
kubectl apply -f csidriver.yaml
```

CFS Turbo 볼륨 생성

1단계: 다음 템플릿을 기반으로 CFS Turbo 유형 PV 생성



```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-cfsturbo
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 10Gi
  csi:
    driver: com.tencent.cloud.csi.cfsturbo
```

```
    volumeHandle: pv-cfsturbo
    volumeAttributes:
      host: *.*.*.*
      fsid: ********
      # cfs turbo subPath
      path: /
  storageClassName: ""
```

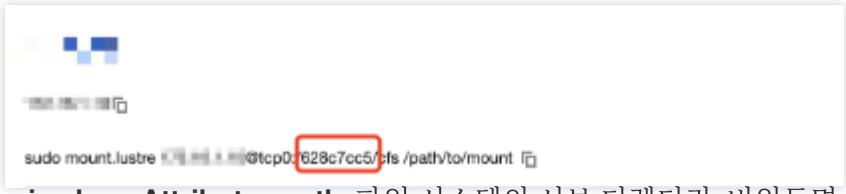
매개변수 설명:

metadata.name: 생성된 PV의 이름입니다.

spec.csi.volumeHandle: PV 이름과 일치해야 합니다.

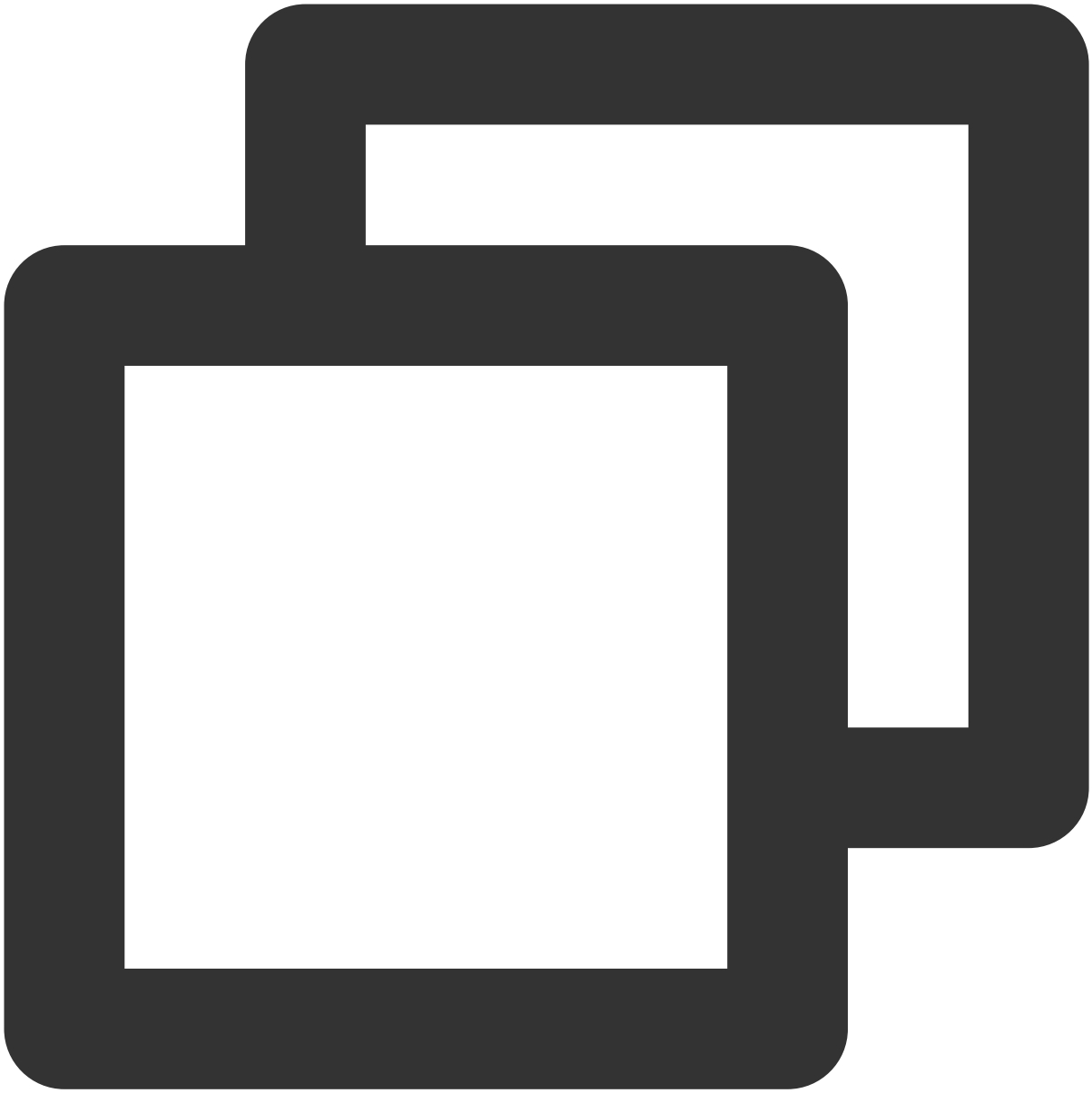
spec.csi.volumeAttributes.host: 파일 시스템의 ip 주소입니다. 파일 시스템 마운트 포인트 정보에서 확인할 수 있습니다.

spec.csi.volumeAttributes.fsid: 파일 시스템의 fsid(파일 시스템 id 아님). 파일 시스템 마운트 포인트 정보에서 확인할 수 있습니다. 다음 이미지와 같이 마운트 명령에서 `tcp0:/` 와 `/cfs` 사이의 문자열입니다.



spec.csi.volumeAttributes.path: 파일 시스템의 서브 디렉터리. 비워두면 `/` 가 입력됩니다(마운트 성능 향상을 위해 `/` 는 `/cfs` 디렉터리 아래에 있음). 마운트할 서브 디렉터리를 지정하려면 파일 시스템의 `/cfs` 에 서브 디렉터리가 있는지 확인해야 합니다. 워크로드는 마운트 후 서브 디렉터리의 상위 디렉터리에 액세스할 수 없습니다. 예를 들어 `path: /test` 의 경우 파일 시스템에 `/cfs/test` 가 있는지 확인해야 합니다.

2단계: 다음 템플릿을 기반으로 PV에 바인딩되는 PVC 생성



```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-cfsturbo
spec:
  storageClassName: ""
  volumeName: pv-cfsturbo
  accessModes:
  - ReadWriteMany
  resources:
    requests:
```

```
      storage: 10Gi
```

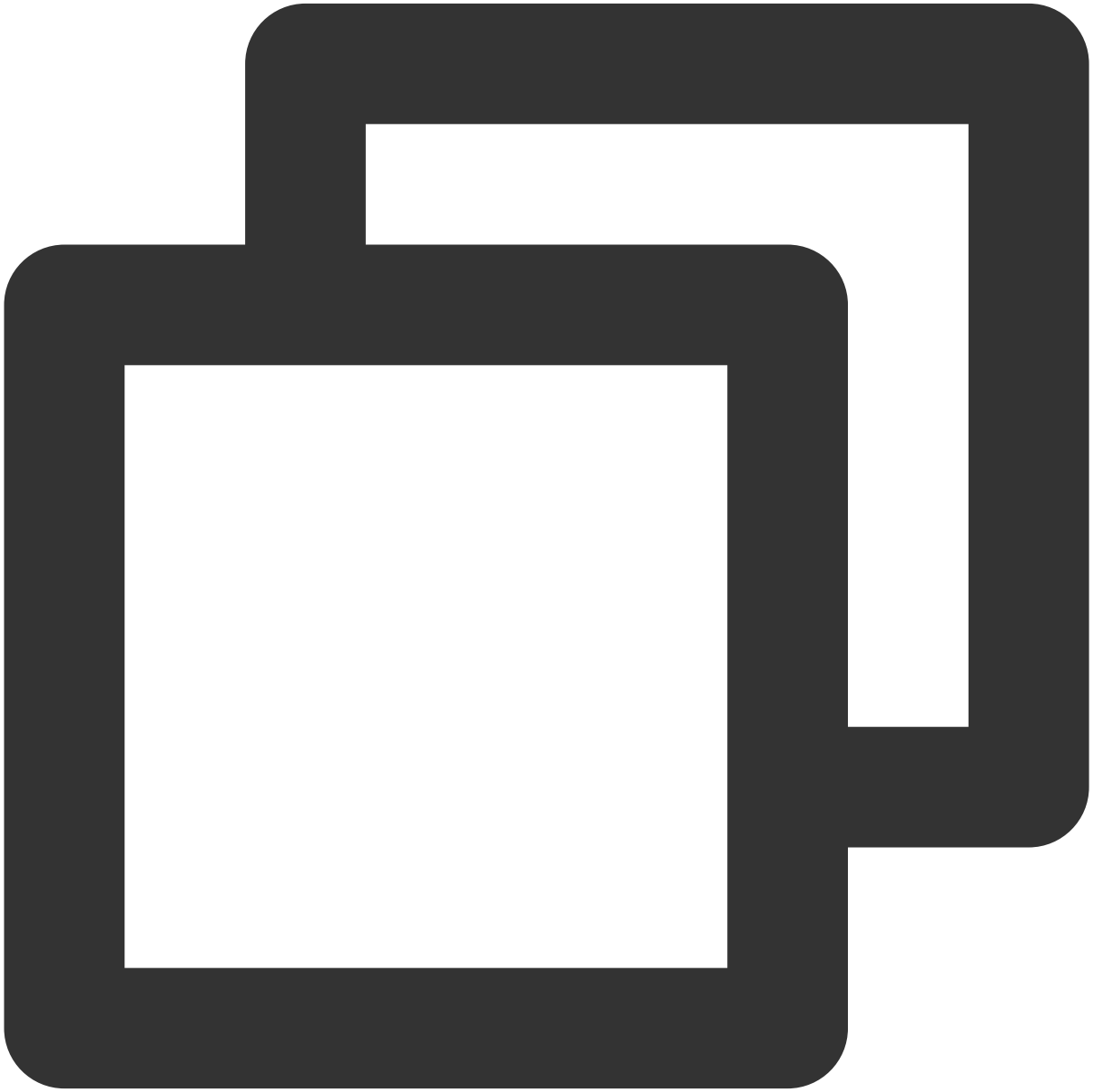
매개변수 설명:

metadata.name: 생성된 PVC의 이름입니다.

spec.volumeName: 1단계에서 생성한 PV의 이름과 일치해야 합니다.

CFS Turbo 볼륨 사용

다음 템플릿을 기반으로 PVC를 마운트할 Pod를 생성합니다.



```
apiVersion: v1
```

```
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: ccr.ccs.tencentyun.com/qcloud/nginx:1.9
    imagePullPolicy: Always
    name: nginx
    ports:
    - containerPort: 80
      protocol: TCP
    volumeMounts:
      - mountPath: /var/www
        name: data
  volumes:
  - name: data
    persistentVolumeClaim:
      claimName: pvc-cfsturbo
```