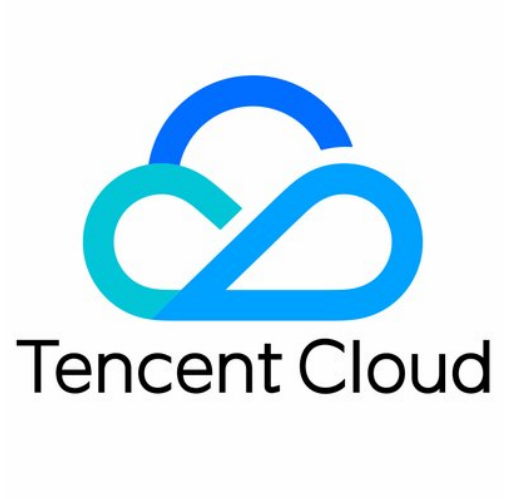
Cloud Object Storage

사례 튜토리얼

제품 문서



목록:

사례 튜토리얼
개요

최종 업데이트 날짜: : 2024-06-24 16:44:04

COS(Cloud Object Storage)는 액세스 제어, 권한 관리, 성능 최적화, 데이터 마이그레이션, 데이터 다이렉트 업로드 및 백업, 데이터 보안 도메인 관리, 빅 데이터, 서버리스 아키텍처 등 다양한 응용 시나리오 및 사례 운영 상세 설명을 제공하여 더욱 빠르고 편리하게 다양한 비즈니스 수요를 충족합니다. 구체적인 사례는 다음과 같습니다.

모범 사례	설명
액세스 제어 및 권한 관리	액세스 제어 및 권한 관리는 Tencent Cloud COS 중 가장 실용적인 기능으로, 이해를 돕기 위해 본 디렉터리 사례를 참고하십시오. ACL 액세스 제어 사례 CAM 사례 서브 계정에 COS 액세스 권한 부여 권한 설정 관련 사례 COS API 권한 부여 정책 사용 가이드 프런트엔드에서 COS로 다이렉트 업로드 시 임시 자격 증명 사용 보안 가이드 임시 키 생성 및 사용 가이드 서브 계정에 태그별 버킷 가져오기 권한 부여 조건 키 설명 및 사용 예시 한 루트 계정의 서브 계정에 다른 루트 계정의 버킷 작업 권한 부여
성능 최적화	COS는 더 높은 요청률을 달성하기 위해 성능 확장을 지원합니다. 자세한 지침은 요청 속도 및 성능 최적화를 참고하십시오. COS는 열악한 네트워크 조건에서 모바일 장치의 업로드 성공률을 향상시키기 위해 멀티파트 업로드의 부분 크기를 동적으로 조정할 수 있습니다. 자세한 내용은 약한 네트워크 환경에서 멀티파트 업로드 재개를 참고하십시오.
AWS S3 SDK로 COS에 액세스	COS는 AWS S3 호환 API를 제공합니다. 아래와 같이 간단한 설정 수정으로 S3 SDK 인터페이스를 사용해 COS 파일에 액세스하는 방법을 소개합니다.
재해 복구 및 백업	재해 복구 및 백업에 대한 모범 사례와 적용 가능한 솔루션은 다음 세 가지 시나리오에 요약되어 있습니다. 버킷 복사 기반의 재해 복구 고가용성 아키텍처 클라우드 데이터 백업 로컬 데이터 백업
도메인 이름 관리	HTTPS 사용자 정의 도메인 이름을 구성하여 COS에 액세스할 수 있습니다. 자세한 내용은 사용자 지정 엔드포인트에 대한 HTTPS 지원을 참고하십시오. COS에서 CORS 규칙을 설정할 수 있습니다. 자세한 내용은 CORS 설정을 참고하십시오.

	COS에서 정적 웹사이트를 호스팅할 수 있습니다. 자세한 내용은 Hosting Static Website를 참고하십시오. COS에서 프런트엔드 단일 페이지 애플리케이션을 구축할 수 있습니다. 자세한 내용은 COS 정적 웹 사이트 기능을 통해 프런트엔드 단일 페이지 애플리케이션 구축을 참고하십시오.
데이터 다이렉트 업로드	다음은 데이터 다이렉트 업로드의 모범 사례입니다. Web의 다이렉트 업로드 사례 미니프로그램 다이렉트 업로드 사례 모바일 애플리케이션 다이렉트 업로드 사례 uni-app 다이렉트 업로드 사례
데이터 보안	사전 이벤트 방지, 중간 이벤트 모니터링 및 사후 이벤트 역추적 측면에서 데이터 보안 솔루션을 설명합니다. 자세한 내용은 COS 데이터 보안 솔루션 소개를 참고하십시오. COS에서 링크 도용 방지를 구성하여 액세스 소스를 제어할 수 있습니다. 자세한 내용은 링크 도용 방지 사례를 참고하십시오.
데이터 검증	MD5 체크섬을 사용하여 COS에 업로드된 데이터의 무결성을 보장할 수 있습니다. 자세한 내용은 MD5 검증을 참고하십시오. CRC-64 체크섬을 사용하여 데이터를 검증할 수 있습니다. 자세한 내용은 CRC64 검사를 참고하십시오.
빅 데이터	Druid의 Deep storage로 COS를 사용할 수 있습니다. 자세한 내용은 COS를 Druid의 Deep storage로 사용을 참고하십시오. Terraform을 사용하여 COS를 관리할 수 있습니다. DataX를 사용하여 COS에서 데이터를 가져오거나 COS에서 내보낼 수 있습니다. 자세한 내용은 DataX로 COS 가져오기 또는 내보내기를 참고하십시오. CDH를 사용하여 COSN을 구성할 수 있습니다. 자세한 내용은 CDH용 COSN 구성을 참고하십시오. COS Ranger를 사용하여 권한을 제어할 수 있습니다. 자세한 내용은 COS Ranger 권한 시스템 솔루션을 참고하십시오. Oceanus와 COS를 연결할 수 있습니다. 자세한 내용은 Oceanus를 COS에 연결을 참고하십시오.
3rd party 애플리케이션에서 COS 사용	COS 일반 설정을 사용하여 S3 호환 3rd party 애플리케이션의 데이터를 COS에 저장할 수 있습니다. 자세한 내용은 S3와 호환되는 타사 애플리케이션에서 COS의 일반 구성 사용를 참고하십시오. 원격 첨부 기능을 사용하여 포럼 첨부 파일을 COS에 저장할 수 있습니다. WordPress 플러그인을 사용하여 COS에 멀티미디어 콘텐츠를 저장할 수 있습니다. 자세한 내용은 COS에 WordPress 원격 첨부 파일 저장하기를 참고하십시오.
API를 통한 파일 일괄 압축	API를 통해 여러 파일을 압축할 수 있습니다.

액세스 제어 및 권한 관리
ACL 액세스 제어 사례

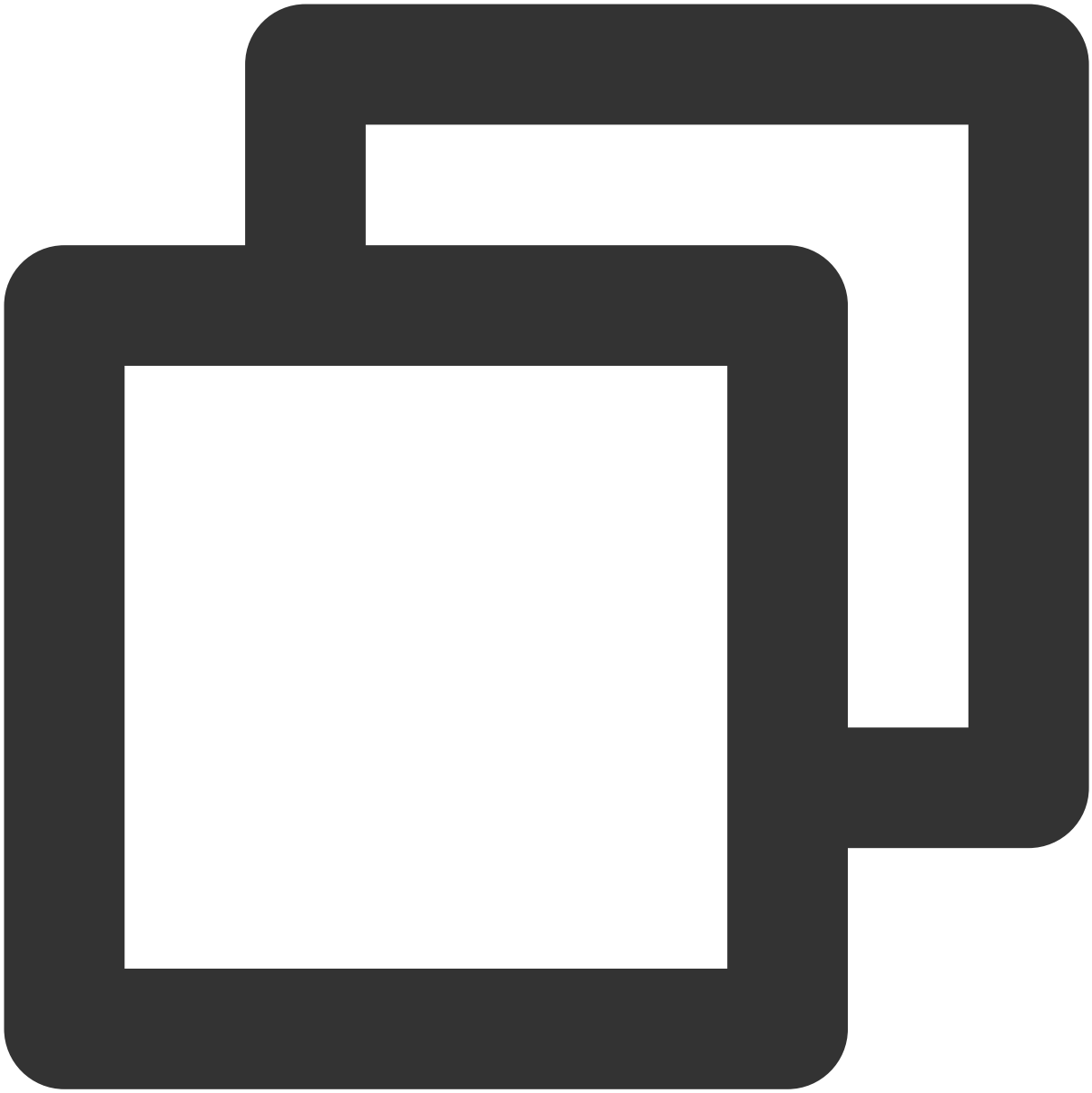
최종 업데이트 날짜: : 2024-06-24 16:44:04

ACL 개요

액세스 제어 리스트(ACL)는 리소스의 액세스 정책 옵션으로, 버킷 및 객체의 액세스를 관리할 수 있으며 다른 루트 계정 및 서브 계정, 사용자 그룹에 기본적인 읽기, 쓰기 권한을 부여할 수 있습니다.
액세스 정책과의 차이점은 ACL은 관리 권한에 다음과 같은 제한이 존재한다는 것입니다.
Tencent Cloud 계정에만 권한 부여 가능
객체 읽기, 객체 쓰기, ACL 읽기, ACL 쓰기, 모든 권한 등 다섯 개 작업 그룹만 지원
효력 발생 조건 부여 지원하지 않음
명시적 거부 효과 지원하지 않음
ACL이 지원하는 제어 단위:
버킷(Bucket)
객체 키 접두사(Prefix)
객체(Object)

ACL의 제어 요소

버킷 또는 객체 생성 시, 해당 리소스가 소속된 루트 계정이 리소스에 대한 모든 권한을 가지며 수정 또는 삭제할 수 없습니다. ACL을 통해 다른 Tencent Cloud 계정에 액세스 권한을 부여할 수 있습니다.
버킷의 ACL 예시는 다음과 같습니다. 여기에서 100000000001은 루트 계정, 100000000011은 루트 계정 아래의 서브 계정, 100000000002는 다른 루트 계정을 나타냅니다. ACL에는 해당 버킷의 소유자인 Owner를 식별할 수 있는 요소가 포함되어 있으며, 해당 버킷의 소유자는 버킷의 모든 권한을 가지고 있습니다. 또한 Grant 요소에서 익명 읽기 권한을 부여하며, `http://cam.qcloud.com/groups/global/AllUsers` 의 READ 권한 형식으로 표시합니다.



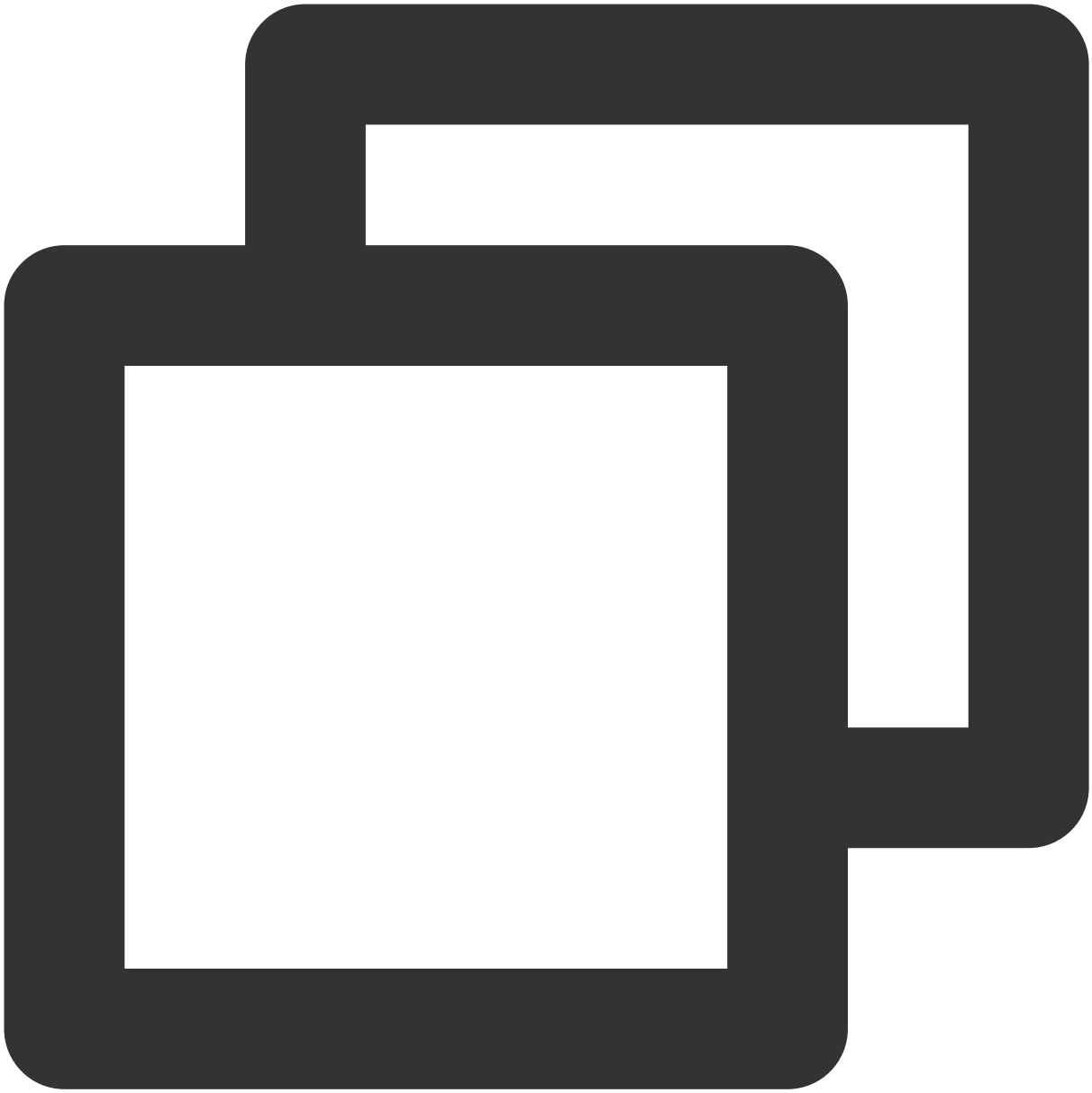
```
<AccessControlPolicy>
  <Owner>
    <ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
    <DisplayName>qcs::cam::uin/100000000001:uin/100000000001</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Root
        <ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
        <DisplayName>qcs::cam::uin/100000000001:uin/100000000001</DisplayName>
      </Grantee>
```

```
      <Permission>FULL_CONTROL</Permission>
    </Grant>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Grou
        <URI>http://cam.qcloud.com/groups/global/AllUsers</URI>
      </Grantee>
      <Permission>READ</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>
```

권한 부여자

루트 계정

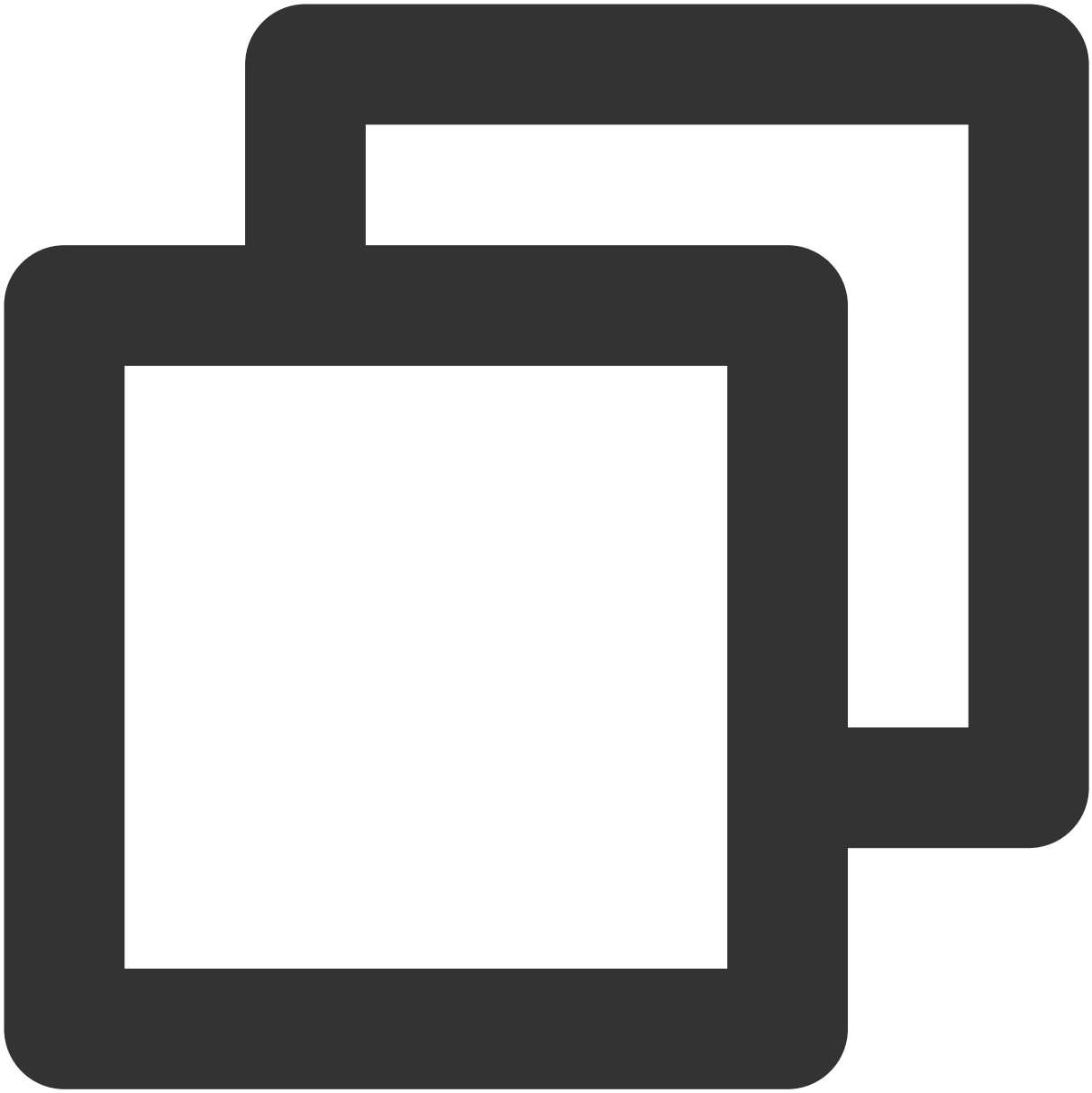
다른 루트 계정에 사용자 액세스 권한을 부여할 수 있으며, CAM의 위탁인(principal) 정의를 이용해 권한을 부여할 수 있습니다. 예시는 다음과 같습니다.



```
qcs::cam::uin/100000000002:uin/100000000002
```

서브 계정

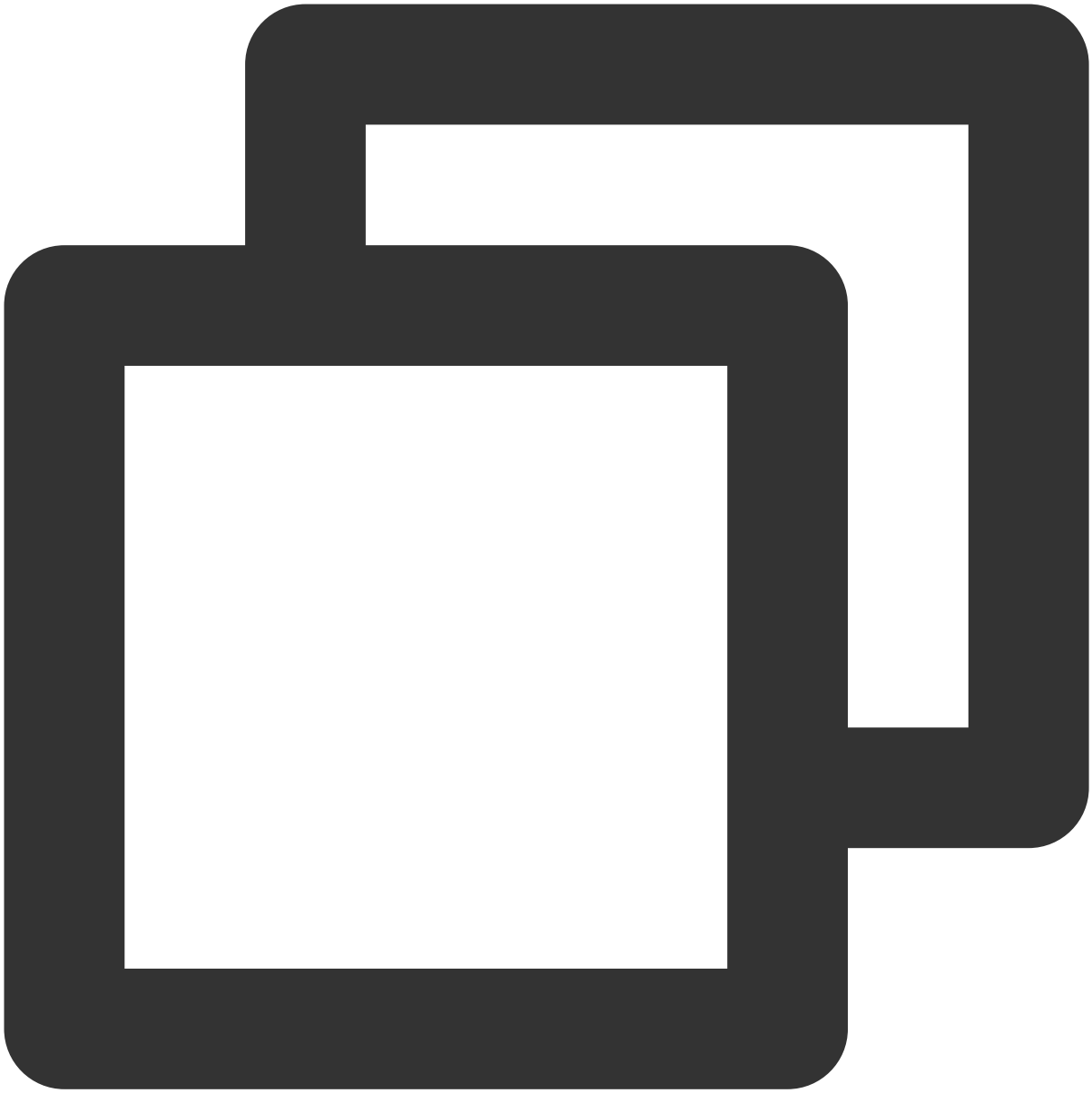
사용자의 루트 계정 아래의 서브 계정(예: 100000000011) 또는 다른 루트 계정 아래의 서브 계정에 권한을 부여할 수 있으며, CAM의 위탁인(principal) 정의를 이용해 권한을 부여할 수 있습니다. 예시는 다음과 같습니다.



```
qcs::cam::uin/100000000001:uin/100000000011
```

익명 사용자

익명 사용자에게 액세스 권한을 부여할 수 있으며, CAM의 위탁인(principal) 정의를 이용해 권한을 부여할 수 있습니다. 예시는 다음과 같습니다.



```
http://cam.qcloud.com/groups/global/AllUsers
```

권한 작업 그룹

ACL이 지원하는 권한 작업 그룹은 다음 표와 같습니다.

작업 그룹	버킷 액세스 권한	접두사 액세스 권한	객체 액세스 권한
READ	버킷의 객체 조회 및 읽기	디렉터리 아래의 객체 조회 및 읽기	객체 읽기

WRITE	버킷의 모든 객체 생성, 덮어쓰기, 삭제	디렉터리 아래의 모든 객체 생성, 덮어쓰기, 삭제	지원하지 않음
READ_ACP	버킷의 ACL 읽기	디렉터리 아래의 ACL 읽기	객체의 ACL 읽기
WRITE_ACP	버킷의 ACL 수정	디렉터리 아래의 ACL 수정	객체의 ACL 수정
FULL_CONTROL	버킷 및 객체에 대한 모든 작업	디렉터리 아래의 객체에 대한 모든 작업	객체에 대해 실행하는 모든 작업

표준 ACL 설명

COS는 일련의 사전 설정 권한을 지원합니다. 이를 표준 ACL이라 부르며, 표준 ACL의 권한 부여에 대한 의미는 다음 표와 같습니다.

주의 :

루트 계정은 항상 FULL_CONTROL 권한을 가지므로 이에 대한 설명은 다음 표에 포함되어 있지 않습니다.

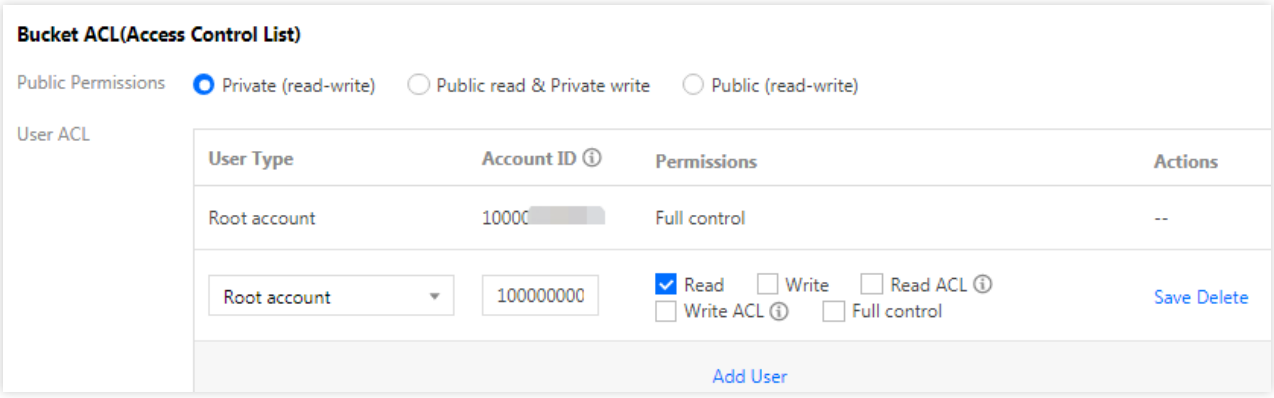
표준 ACL	의미
(비어 있음)	기본 정책. 다른 사용자에게 권한이 없으며, 리소스는 상위 권한 상속
private	다른 사용자에게 권한 없음
public-read	익명의 사용자 그룹에게 READ 권한이 부여됨
public-read-write	익명의 사용자 그룹에게 READ 및 WRITE 권한이 부여됨. 버킷에 해당 권한 부여는 권장하지 않음

사용 방법

콘솔을 이용한 CAL 작업

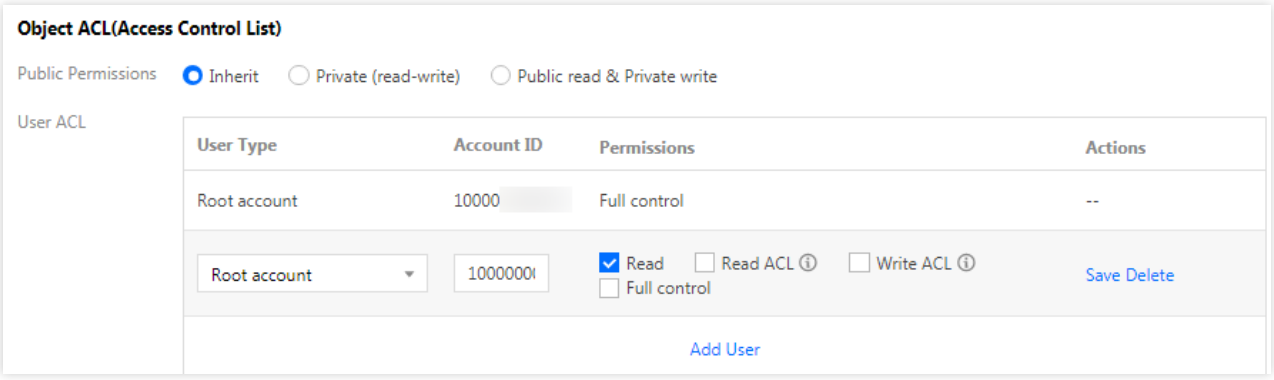
버킷에 ACL 설정

다음은 다른 루트 계정에 특정 **버킷**의 읽기 권한을 부여하는 예시입니다.



객체에 ACL 설정

다음은 다른 루트 계정에 특정 **객체**의 읽기 권한을 부여하는 예시입니다.



주의 :

서브 계정을 사용해 버킷 또는 객체에 액세스할 때 **액세스 권한 없음** 안내가 표시되는 경우, 먼저 루트 계정에서 서브 계정에 버킷에 정상적으로 액세스할 수 있는 권한을 부여해야 합니다. 이에 대한 자세한 작업 방법은 서브 계정으로 버킷 리스트에 액세스를 참조하십시오.

API를 이용한 ACL 작업

버킷 ACL

API	작업명	설명
PUT Bucket acl	버킷 ACL 설정	지정 버킷 액세스 권한 제어 리스트 설정
GET Bucket acl	버킷 ACL 조회	버킷 액세스 제어 리스트 조회

객체 ACL

API	작업명	설명

PUT Object acl	객체 ACL 설정	버킷의 한 객체의 액세스 제어 리스트 설정
GET Object acl	객체 ACL 조회	객체 액세스 제어 리스트 조회

CAM 사례

최종 업데이트 날짜: : 2024-06-24 16:44:04

CAM 개요

Cloud Access Management(CAM)는 Tencent Cloud 리소스에 대한 액세스를 관리하는 데 도움이 되는 Tencent Cloud 인증 및 권한 부여 서비스입니다. 승인된 객체, 리소스 및 작업을 관리하고 권한을 부여할 때 액세스를 제어하는 정책을 설정할 수 있습니다.

기능

루트 계정에서 리소스에 대한 액세스 권한 부여

루트 계정의 자격 증명을 공유하지 않고도 서브 계정 및 기타 루트 계정을 비롯한 다른 사용자에게 루트 계정의 리소스에 대한 액세스 권한을 부여할 수 있습니다.

세분화된 권한 관리

사용자별로 각 리소스에 대해 각각 다른 액세스 권한을 부여할 수 있습니다. 예를 들어 일부 서브 계정에 Cloud Object Storage(COS) 버킷에 대한 읽기 액세스 권한을 부여하고, 다른 서브 계정 또는 루트 계정에는 COS 객체에 대한 쓰기 액세스 권한을 부여할 수 있습니다. 또한 상기 리소스, 액세스 권한, 사용자를 일괄적으로 관리할 수도 있습니다.

최종 일관성

CAM은 현재 정책 복사를 통한 Tencent Cloud 리전 간의 데이터 동기화를 지원합니다. CAM 정책은 적시에 수정할 수 있지만 리전 간 정책 동기화가 적용되는 데 시간이 걸릴 수 있습니다. 또한 CAM은 캐시(현재 1분 동안 유효)를 사용하여 성능을 개선하며 캐시가 만료될 때까지 정책 업데이트가 적용되지 않습니다.

응용 시나리오

기업의 서브 계정 액세스 권한 관리

기업 내의 각 직급별 직원은 해당 기업의 클라우드 리소스에 대한 최소한의 액세스 권한이 필요합니다. 예를 들어 CVM, VPC 인스턴스, CDN 인스턴스, COS 버킷 및 객체 등 방대한 클라우드 리소스를 가지고 있고 개발자, 테스트 직원, 유지보수 직원 등을 포함한 수 많은 직원이 근무하고 있는 기업이 있다고 가정합니다.
개발자는 개발 서버의 프로젝트 관련 클라우드 리소스에 대한 읽기 및 쓰기 권한이 필요하며, 테스트 직원은 테스트 서버의 프로젝트 관련 클라우드 리소스에 대한 읽기 및 쓰기 권한이 필요하고, 유지보수 직원은 서버의 구매 및 일상 유지보수를 책임져야 합니다. 직원의 직무나 프로젝트가 변경된 경우 해당 권한을 종료해야 합니다.

기업 간의 액세스 권한 관리

기업 간에 클라우드 리소스를 공유해야 하는 경우가 있습니다. 예를 들어 클라우드 리소스가 많은 회사는 제품 R&D에 집중하고 클라우드 리소스 운영을 다른 회사에 아웃소싱하려고 하며, 또한 아웃소싱 서비스 계약이 종료되는 즉시 부여된 모든 권한을 취소해야 합니다.

정책 구문

CAM 정책(policy)은 여러 요소로 구성되며 권한 부여에 대한 특정 정보를 설명하는 데 사용됩니다. 핵심 요소는 주체(principal), 작업(action), 리소스(resource), 조건(condition), 효과(effect)가 있으며, 자세한 내용은 액세스 정책 언어 개요를 참고하십시오.

설명：

정책 구문의 설명에는 특별한 순서가 없으며, 행동(action) 요소는 영어 대소문자를 구분합니다.

특정 조건이 필요하지 않은 경우 condition 요소는 옵션 항목입니다.

콘솔에서는 principal 요소는 정의할 수 없으며, 정책 관리 API 또는 정책 구문 관련 매개변수에서를 통해서만 principal을 정의할 수 있습니다.

핵심 요소

핵심 요소	설명	필수 입력 여부
버전 version	정책 구문 버전을 지정합니다. 유효 값: 2.0	예
주체 principal	사용자(개발자, 서브 계정, 익명의 사용자) 및 사용자 그룹이 포함하여 정책에 의해 권한이 부여되는 엔터티를 설명합니다.	정책 관리 API 및 정책 구문 관련 매개변수에서만 해당 요소를 사용할 수 있습니다.
명령 statement	action, resource, condition, effect 등 다른 요소에 의해 정의된 권한 또는 권한 집합에 대한 세부 정보를 설명합니다. 하나의 정책에는 하나의 statement만 있습니다.	예
작업 action	허용 또는 거부할 작업을 설명합니다. API 작업 또는 API 작업 집합일 수 있습니다. 작업(action) 요소는 영어 대소문자를 구분합니다. 예: `name/cos:GetService`	예
리소스 resource	권한이 적용되는 리소스를 설명합니다. 리소스는 6세그먼트 형식으로 설명됩니다. 자세한 리소스 정의는 제품에 따라 다릅니다. 리소스를 지정하는 방법에 대한 자세한 내용은 다음 항목에 대한 설명을 작성할 해당 리소스가 있는 제품 문서를 참고하십시오.	예
조건 condition	정책이 적용되는 조건을 설명합니다. 조건은 연산자, 작업 키 및 작업 값으로 구성됩니다. 조건 값은 시간, IP 주소 등이 될 수 있	아니오

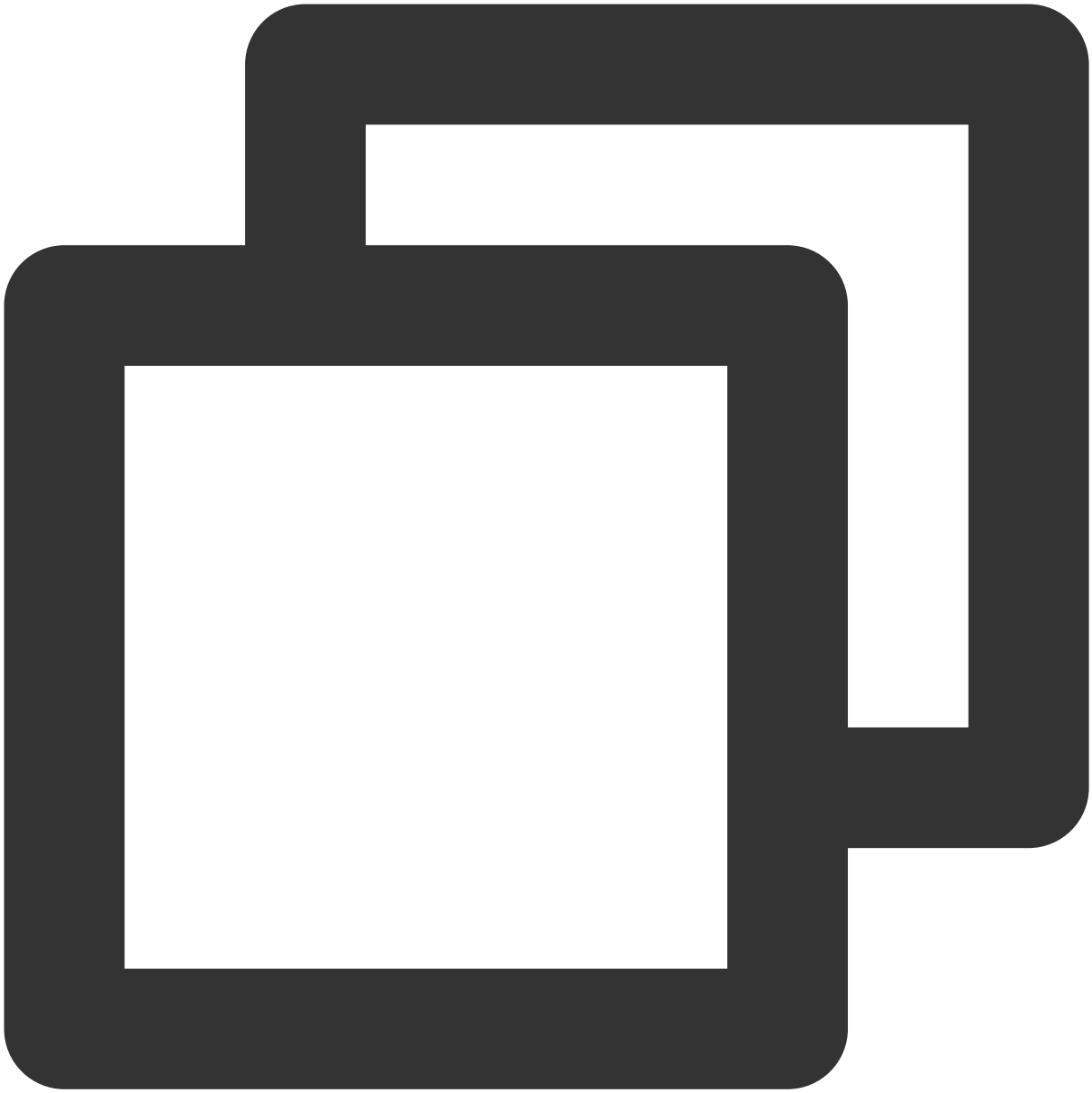
	습니다. 일부 서비스에서는 조건에 추가 값을 지정할 수 있습니다.	
효과 effect	명령문 결과가 allow(허용)인지 deny(명시적 거부)인지를 설명합니다.	예

정책 제한

제한 항목	제한 값
루트 계정의 사용자 그룹 수	300
루트 계정의 서브 계정 수	1000
루트 계정의 역할 수	1000
서브 계정을 추가할 수 있는 사용자 그룹 수	10
공동 작업자가 연결할 수 있는 루트 계정 수	10
사용자 그룹의 서브 계정 수	100
루트 계정에 대해 생성된 사용자 지정 정책 수	1500
CAM 사용자/사용자 그룹/역할에 연결된 정책 수	200
정책 구문에 허용되는 최대 문자 수	4096

정책 예시

다음은 ID가 100000000001(APPID는 1250000000)인 루트 계정 아래의 ID가 100000000011인 서브 계정에 베이징 리전의 버킷 examplebucket-bj와 광저우 리전의 버킷 examplebucket-gz의 객체 exampleobject에 대해, 액세스 IP가 `10.*.*.10/24` 대역인 경우 **객체 업로드** 및 **객체 다운로드** 권한을 부여하는 정책 예시입니다.



```
{
            "version": "2.0",
            "principal":{
                "qcs": ["qcs::cam::uin/100000000001:uin/100000000011"]
            },
            "statement": [{
                "effect": "allow",
                "action": ["name/cos:PutObject", "name/cos:GetObject"],
                "resource": ["qcs::cos:ap-beijing:uid/1250000000:examplebucket-bj-1
                    "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-gz-12500000
                ],
```

```
                "condition":{
                    "ip_equal":{
                        "qcs:ip": "10.*.*.10/24"
                    }
                }
            }]
}
```

서브 계정에 COS 액세스 권한 부여

최종 업데이트 날짜: : 2024-06-24 16:44:03

개요

CAM(Cloud Access Management)을 통해 COS(Cloud Object Storage) 버킷 또는 객체에 대해 서로 다른 작업 권한을 설정할 수 있으므로 서로 다른 회사 또는 부서의 서로 다른 팀 또는 사용자가 서로 더 잘 협업할 수 있습니다.
먼저 루트 계정, 서브 계정(사용자), 사용자 그룹 등 몇 가지 핵심 개념을 이해하고 있어야 합니다. CAM과 관련된 용어 및 설정에 대한 자세한 설명은 CAM Glossary를 참조하십시오.

루트 계정

루트 계정은 개발사라고도 합니다. 사용자가 Tencent Cloud 계정을 신청하면 시스템은 Tencent Cloud 서비스에 로그인할 루트 계정 정보를 생성합니다. 루트 계정은 Tencent Cloud 리소스의 사용 용량 계산 및 과금의 기본 주체입니다.
루트 계정은 기본적으로 해당 계정의 모든 리소스에 대해 청구서 정보 확인, 사용자 비밀번호 변경, 사용자 및 사용자 그룹 생성, 기타 클라우드 서비스 리소스 액세스 등을 포함한 완전한 액세스 권한을 보유하고 있습니다. 기본 설정 시 리소스는 루트 계정만 액세스할 수 있으며 다른 사용자는 루트 계정에서 권한을 부여해야만 액세스할 수 있습니다.

서브 계정(사용자) 및 사용자 그룹

서브 계정은 루트 계정에서 생성하는 엔터티로, 확실한 ID 및 자격 증명을 가지고 있으며 Tencent Cloud 콘솔에 로그인할 수 있는 권한을 가지고 있습니다.
서브 계정은 기본적으로 리소스를 보유하고 있지 않으며, 반드시 소속된 루트 계정에서 권한을 부여해야 합니다.
하나의 루트 계정은 여러 개의 서브 계정(사용자)을 생성할 수 있습니다.
하나의 서브 계정은 여러 개의 루트 계정에 소속될 수 있으며, 각각의 루트 계정과 협업하여 루트 계정별 클라우드 리소스를 관리할 수 있습니다. 단, 하나의 서브 계정이 동시에 여러 루트 계정에 로그인할 수는 없으며, 하나의 루트 계정 소속으로 로그인하여 해당 루트 계정의 클라우드 리소스만 관리할 수 있습니다.
서브 계정은 콘솔을 통해 개발사(루트 계정)를 전환할 수 있으며, 하나의 루트 계정에서 다른 루트 계정으로 전환할 수 있습니다.
서브 계정은 콘솔에 로그인할 때 자동으로 기본 루트 계정으로 전환되며, 기본 루트 계정에서 부여한 액세스 권한을 가집니다.
개발사를 전환하면 서브 계정은 전환된 루트 계정이 부여한 액세스 권한을 가지며, 전환 전의 루트 계정이 부여한 액세스 권한은 즉시 효력이 사라집니다.
사용자 그룹은 동일한 직무의 여러 사용자(서브 계정)가 모인 집합체입니다. 업무 필요에 따라 서로 다른 사용자 그룹을 생성할 수 있으며, 사용자 그룹에 적합한 정책을 연결하여 각 권한을 할당할 수 있습니다.

작업 단계

서브 계정에 COS 액세스 권한을 부여하는 순서는 서브 계정 생성, 서브 계정에 권한 부여, 서브 계정으로 COS 리소스에 액세스, 해당 세 단계로 나뉩니다.

1단계: 서브 계정 생성

CAM 콘솔에서 서브 계정을 생성하고 액세스 권한을 설정할 수 있습니다. 자세한 작업 방법은 다음과 같습니다.

1. CAM 콘솔에 로그인합니다.
2. **사용자 > 사용자 목록 > 사용자 생성**을 선택하여 사용자 생성 페이지로 이동합니다.
3. **사용자 정의 생성**을 선택하고 **리소스 액세스 및 메시지 수신**을 설정한 후 **다음**을 클릭합니다.
4. 필요에 따라 사용자 관련 정보를 입력합니다.

사용자 정보: 서브 계정의 사용자 이름(예: Sub_user) 및 이메일 주소를 설정합니다. 이메일 주소는 서브 계정을 WeChat 계정과 연결하기 위해 Tencent Cloud에서 보낸 이메일을 수신하는 데 필요합니다.

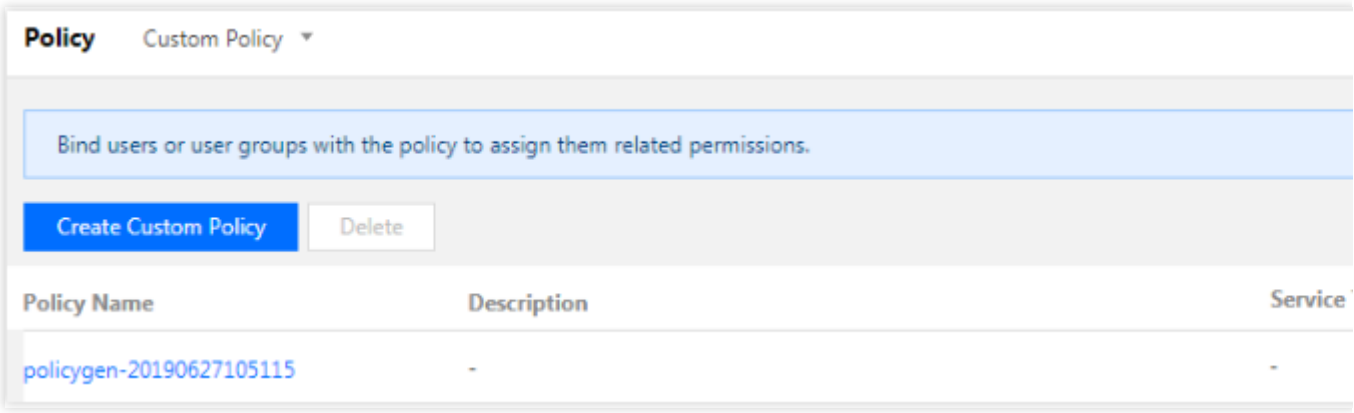
액세스 방식: 프로그래밍 액세스 및 Tencent Cloud 콘솔 액세스를 선택하고, 다른 설정은 필요에 따라 선택합니다.

5. 사용자 정보를 모두 작성한 후, **다음 단계**를 클릭하여 자격을 인증합니다.
6. 서브 계정에 권한을 부여합니다. 제공된 정책 옵션으로 서브 계정에 COS 버킷 목록에 대한 액세스 권한 또는 읽기 전용 권한을 부여하는 것과 같은 간단한 정책을 구성할 수 있습니다. 보다 복잡한 정책을 설정하려면 2단계: 서브 계정에 권한 부여를 진행하십시오.
7. 사용자 태그를 선택적으로 설정하고 **다음 단계**을 클릭합니다.
8. **완료**를 클릭하면 서브 계정이 생성됩니다.

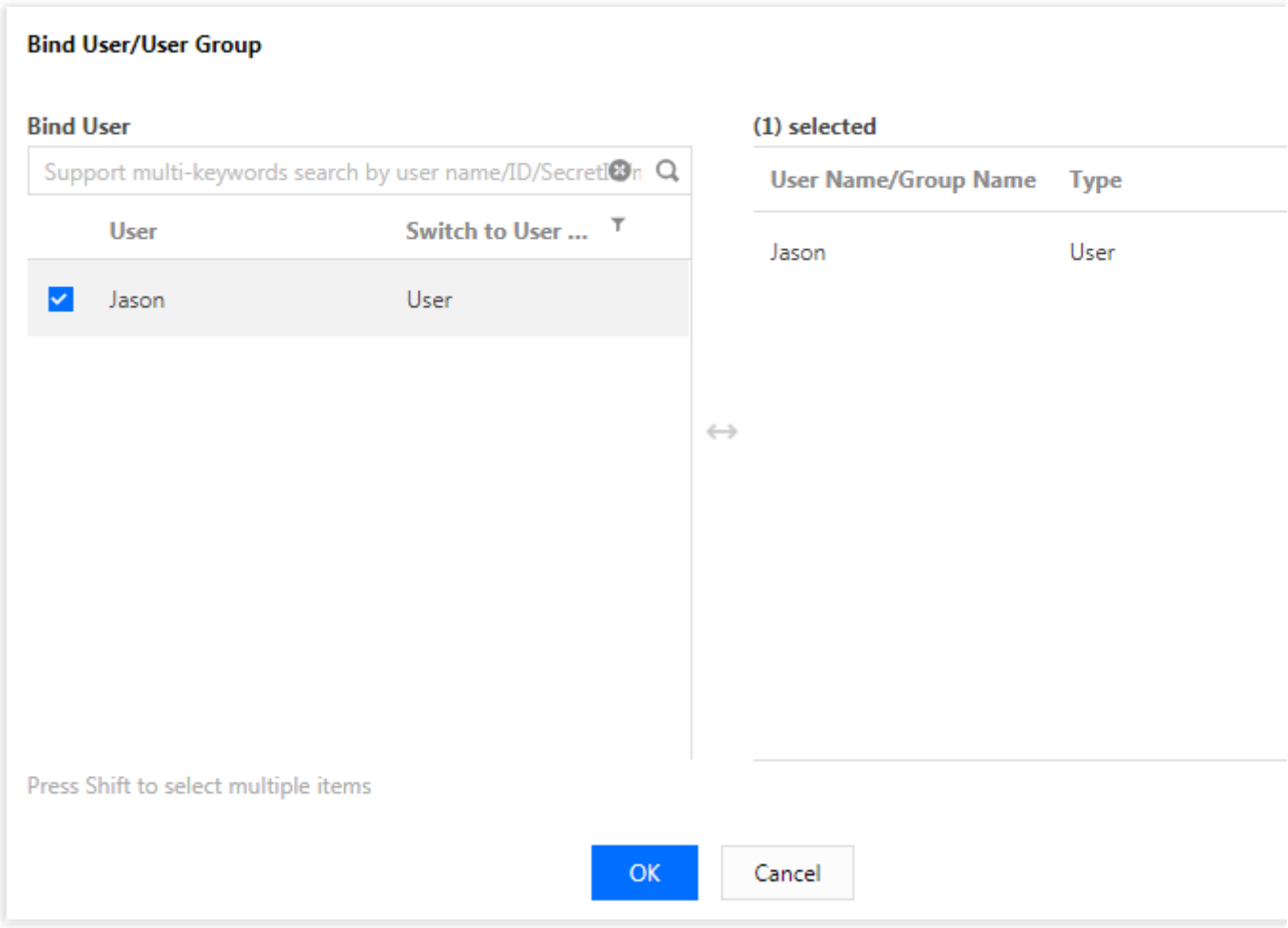
2단계: 서브 계정에 권한 부여

사용자 지정 정책을 생성하거나 기존 정책을 선택하고 서브 계정과 연결합니다.

1. CAM 콘솔에 로그인합니다.
2. **정책 > 사용자 정의 정책 생성 > 정책 구문 생성**을 선택하여 정책 생성 페이지로 이동합니다.
3. 실제 필요에 따라 **빈 템플릿**을 선택하여 권한 부여 정책을 사용자 정의하거나 COS와 연결할 **시스템 템플릿**을 선택하고 **다음**을 클릭합니다.
4. 기억하기 쉬운 정책 이름을 입력합니다. **빈 템플릿**을 선택한 경우 정책 구문을 입력해야 합니다. 자세한 내용은 [정책 예시](#정책 예시)를 참고하십시오. 정책 내용을 복사하여 **정책 내용** 상자에 붙여넣고 입력한 정보가 정확한지 확인한 후 **완료**를 클릭합니다.
5. 생성 완료 후, 방금 생성한 정책을 서브 계정에 연결합니다.



6. 서브 계정을 선택하고 **확인**을 클릭해 권한을 부여하면, 서브 계정을 이용해 한정된 COS 리소스에 액세스할 수 있습니다.



3단계: 서브 계정으로 COS 리소스에 액세스

위 1단계에서 설정한 프로그래밍 액세스 및 Tencent Cloud 콘솔 액세스 방법을 다음과 같이 소개합니다.
(1) 프로그래밍 액세스

서브 계정을 사용하여 프로그램(예: API, SDK, 툴 등)으로 COS 리소스에 액세스할 때에는 루트 계정의 APPID와 서브 계정의 SecretId, SecretKey 정보가 필요합니다. CAM 콘솔에서 서브 계정의 SecretId와 SecretKey를 생성할 수 있습니다.

1. 루트 계정으로 CAM 콘솔에 로그인합니다.
2. **사용자 리스트**를 선택하여 사용자 리스트 페이지로 이동합니다.
3. 서브 계정의 사용자 이름을 클릭하여 서브 계정 정보 상세 페이지로 이동합니다.
4. **API Keys** 탭을 클릭하고 **키 생성**을 클릭하여 서브 계정에 SecretId와 SecretKey를 생성합니다.

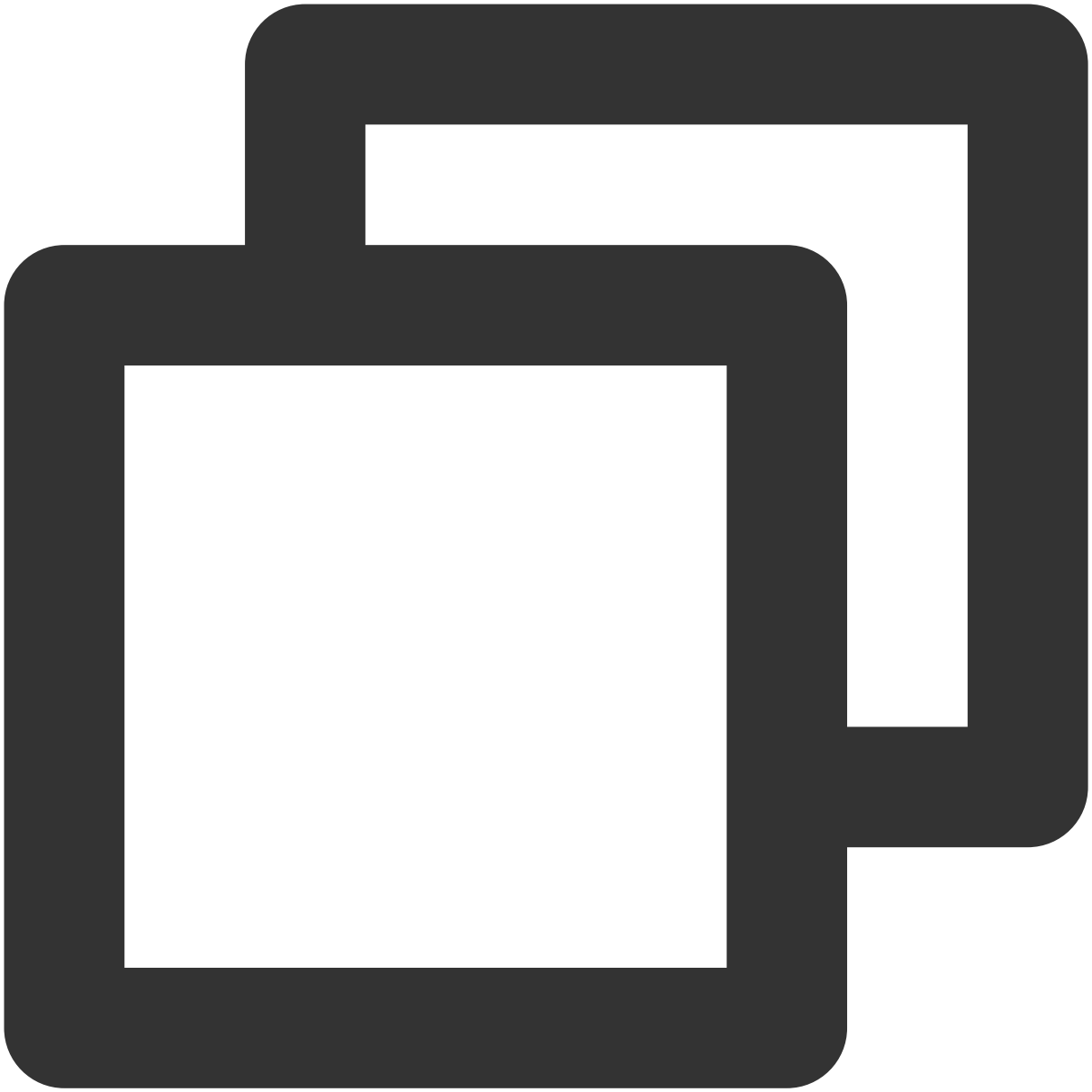
이제 서브 계정의 SecretId와 SecretKey, 루트 계정의 APPID를 사용해 COS 리소스에 액세스할 수 있습니다.

주의 :

서브 계정은 XML API 또는 XML API 기반의 SDK를 통해 COS 리소스에 액세스할 수 있습니다.

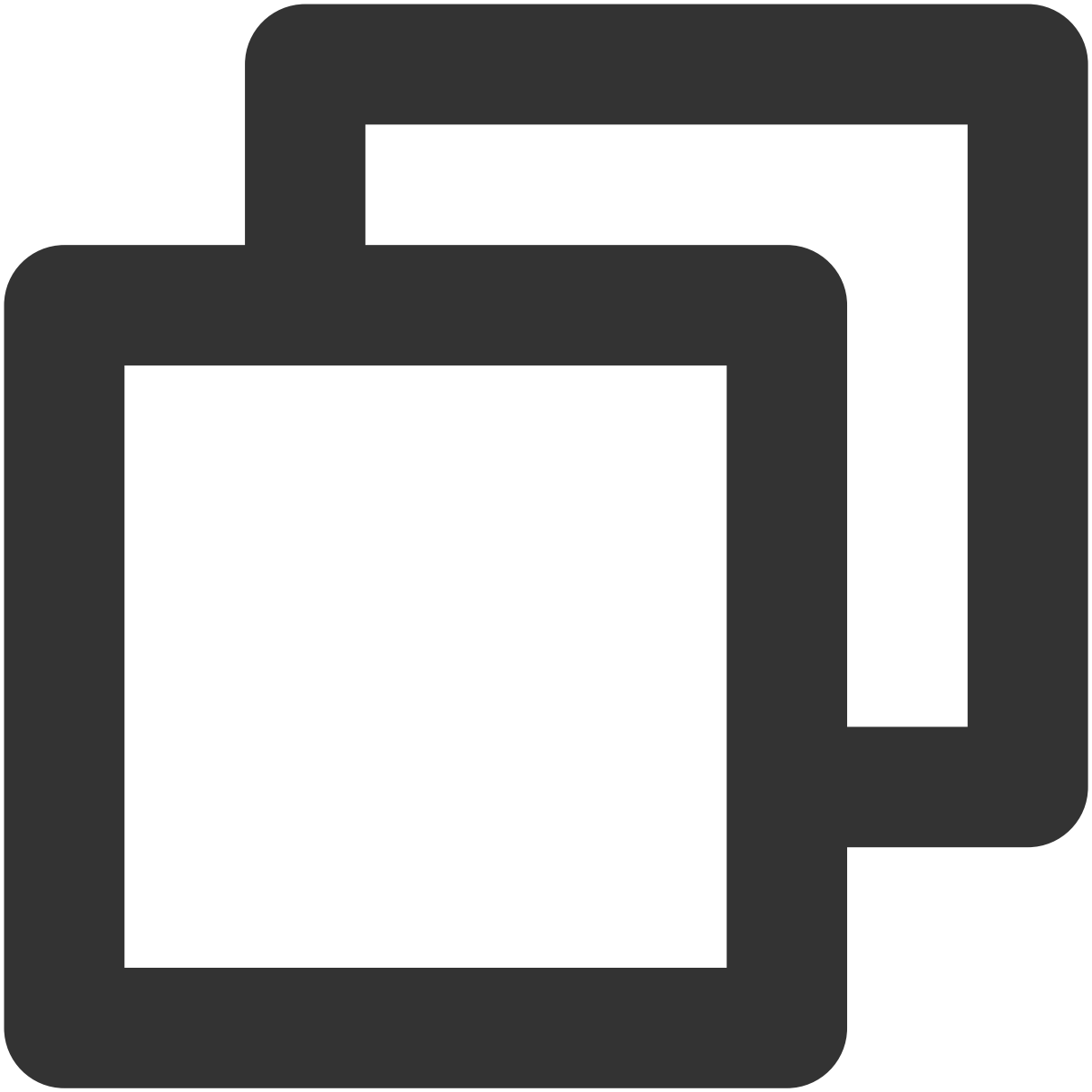
XML 기반의 Java SDK를 통한 액세스 예시

XML 기반의 Java SDK 명령 라인 예시이며, 다음과 같이 매개변수를 입력해야 합니다.



```
// 사용자 인증 정보 초기화
COSCredentials cred = new BasicCOSCredentials("<루트 계정 APPID>", "<서브 계정 SecretI
```

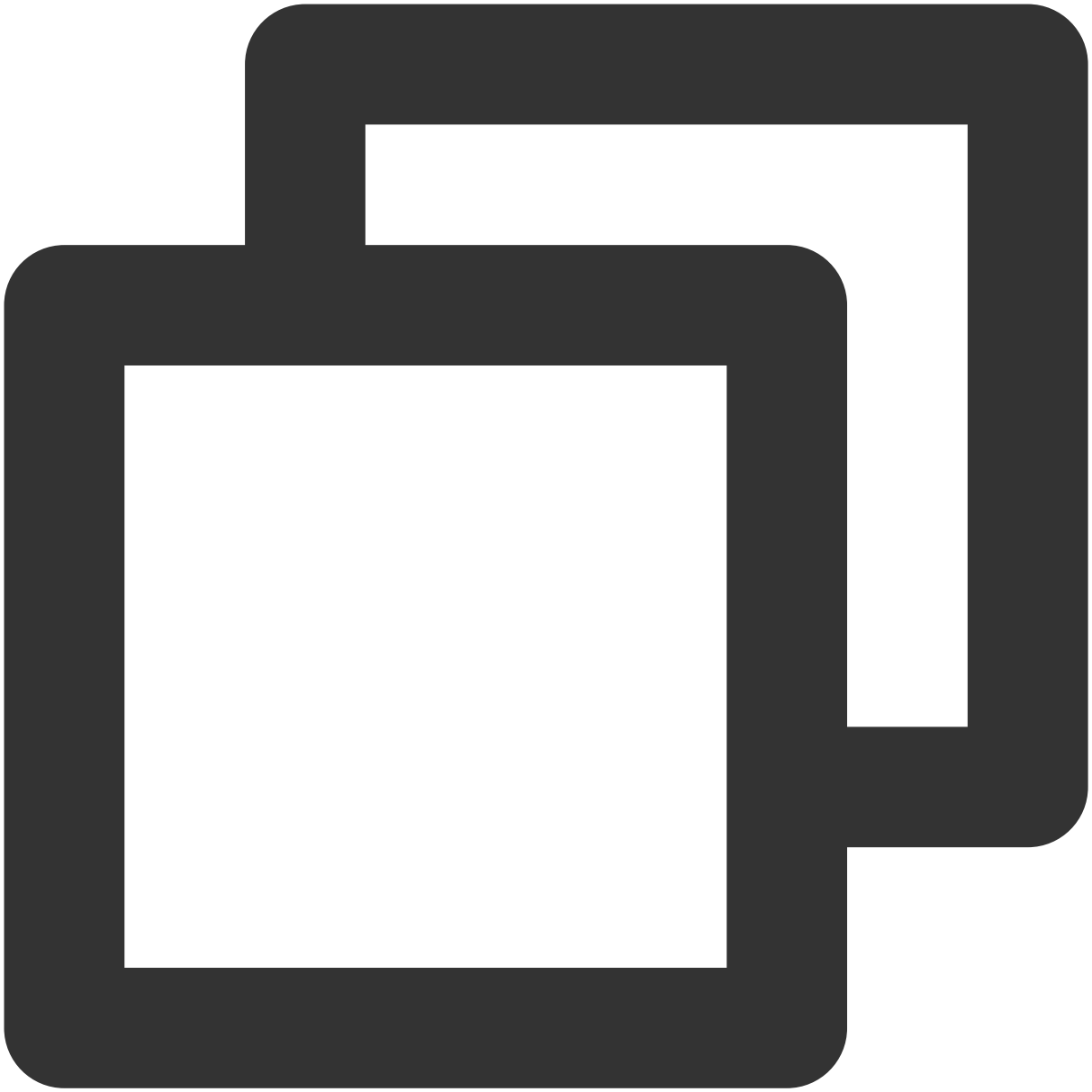
인스턴스는 다음과 같습니다.



```
String secretId = System.getenv("secretId");//서브 계정의 SecretId. 리스크를 줄이기 위해
String secretKey = System.getenv("secretKey");//서브 계정의 SecretKey. 리스크를 줄이기
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);

// 사용자 인증 정보 초기화
COSCredentials cred = new BasicCOSCredentials("<루트 계정 APPID>", secretId, secretKe
```

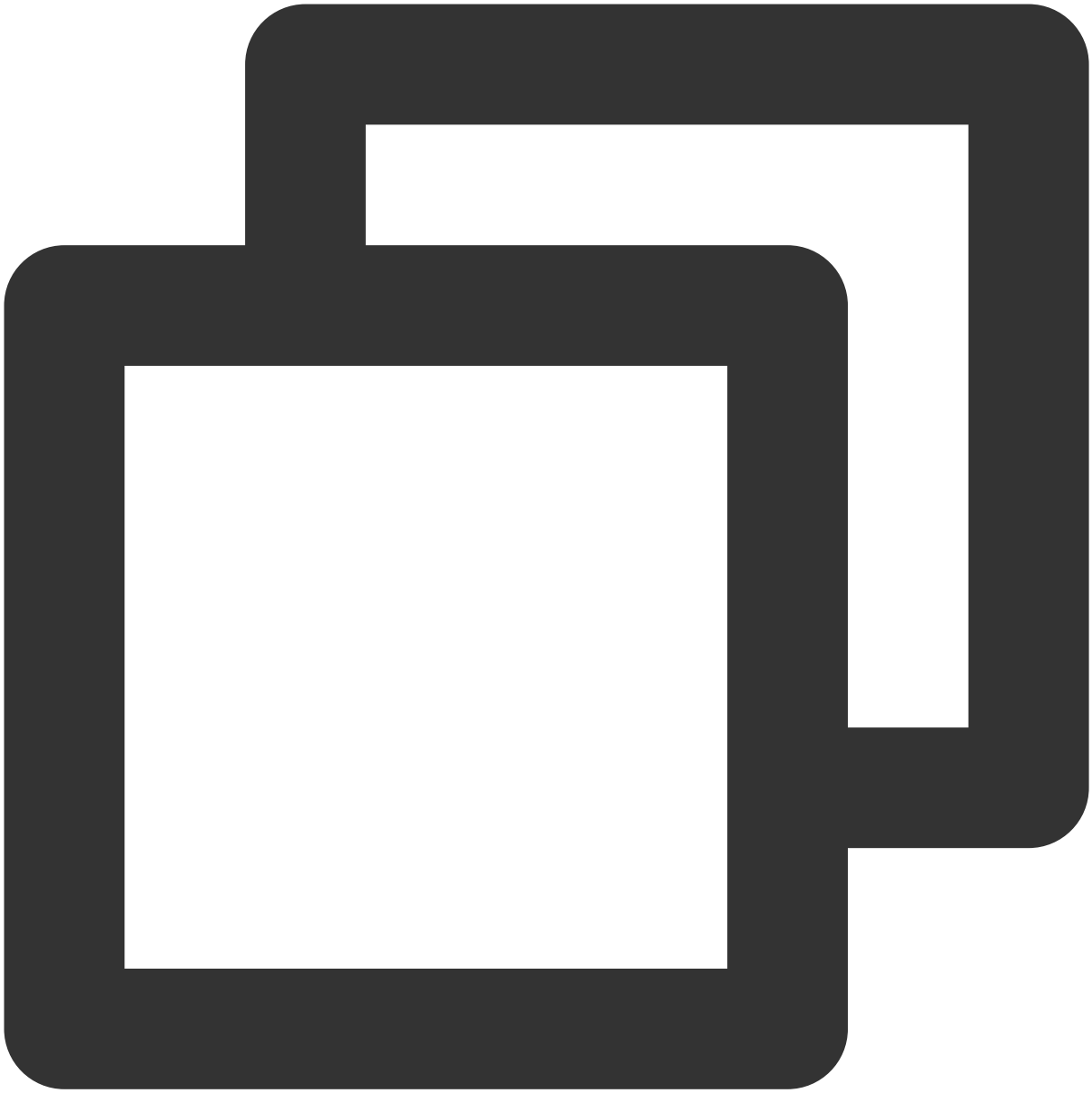
COSCMD TCCLI를 이용한 액세스 예시

COSCMD의 명령어 설정 예시이며, 다음과 같이 매개변수를 입력해야 합니다.



```
coscmd config -u <루트 계정 APPID> -a <서브 계정 SecretId> -s <서브 계정 SecretKey>  -b
```

인스턴스는 다음과 같습니다.



```
coscmd config -u 1250000000 -a AKIDasdfmRxHPa9oLhJp**** -s e8Sdeasdfas2238Vi**** -b
```

(2) Tencent Cloud 콘솔 액세스

서브 계정는 권한 부여 후 서브 계정 로그인 페이지에서 루트 계정 아이디, 서브 계정 이름, 서브 계정 비밀번호를 입력하여 콘솔에 로그인할 수 있습니다. 그런 다음 **제품**에서 **Cloud Object Storage**를 클릭하여 루트 계정에서 스토리지 리소스에 액세스할 수 있습니다.

정책 예시

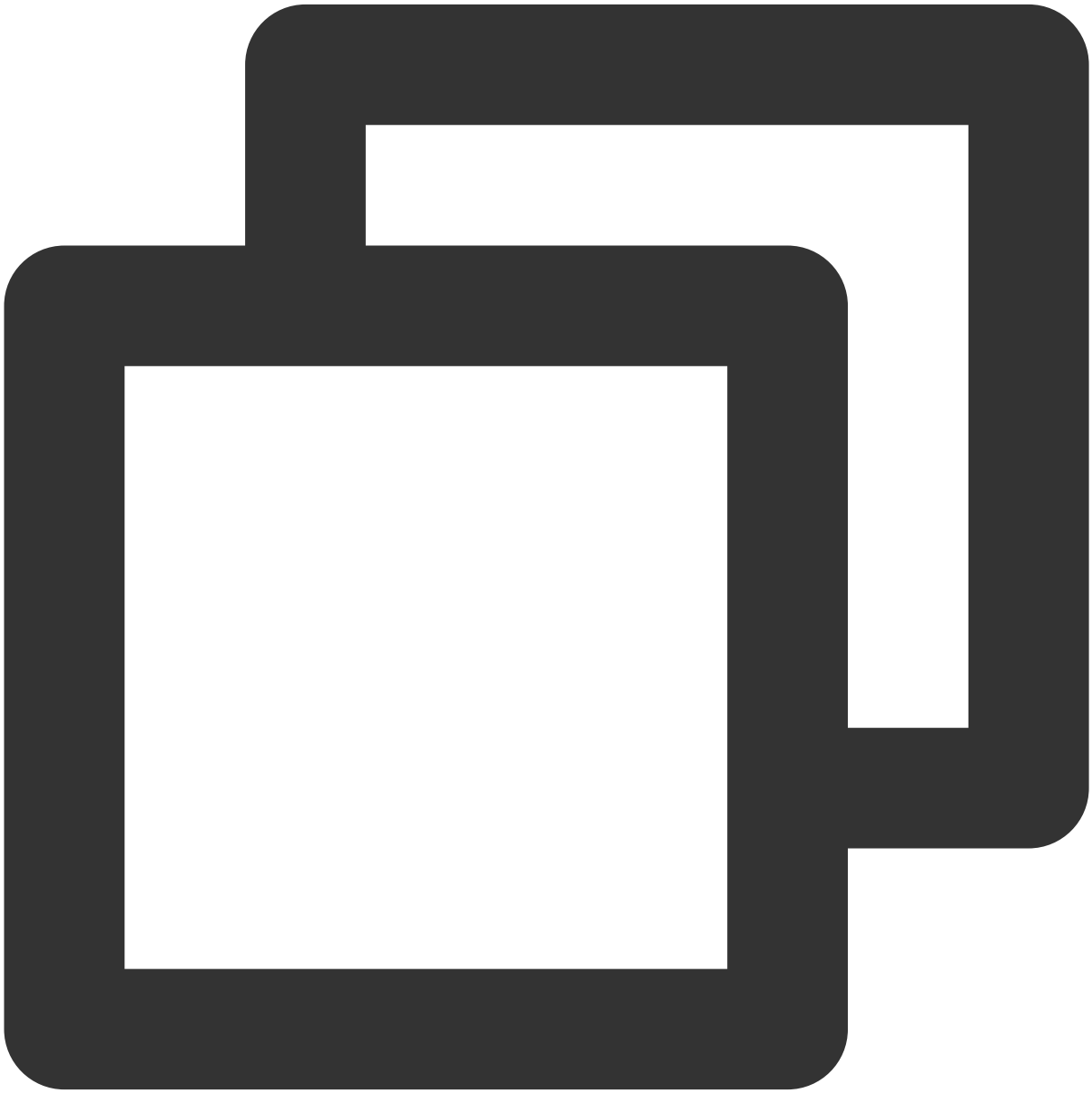
다음과 같은 일반적인 샘플 정책이 여기에서 제공됩니다. 정책을 구성할 때 아래 코드를 참고하여 **정책 내용** 란에 복사 붙여넣기 하시고 필요에 따라 수정하시면 됩니다. 다른 일반적인 COS 시나리오에 대한 자세한 정책 구문은 액세스 정책 언어 개요 또는 CAM 제품 문서의 **비즈니스 사용 사례** 부분을 참고하십시오.

예시1: 서브 계정에 COS 전체 읽기 및 쓰기 권한 설정

주의 :

본 정책의 권한 부여 범위가 넓습니다. 설정 시 유의하십시오.

자세한 정책 설정 구문은 다음과 같습니다.

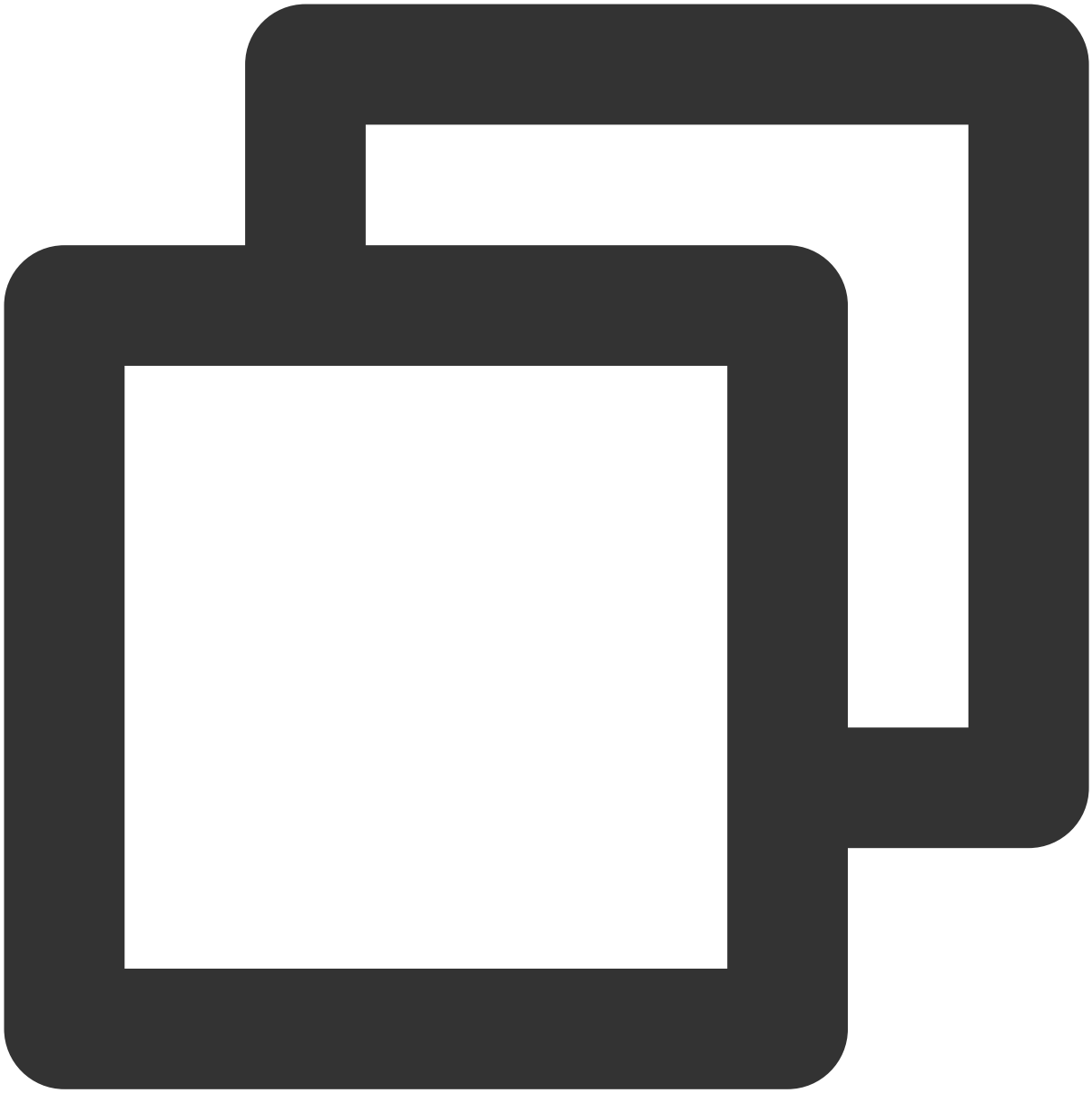


```
{
```

```
    "version": "2.0",
    "statement": [
        {
            "action":[
                "name/cos:*"
            ],
            "resource": "*",
            "effect": "allow"
        },
        {
            "effect": "allow",
            "action": "monitor:*",
            "resource": "*"
        }
    ]
}
```

예시2: 서브 계정에 읽기 전용 권한 설정

서브 계정에 읽기 전용 권한을 부여하는 자세한 정책 설정 구문은 다음과 같습니다.

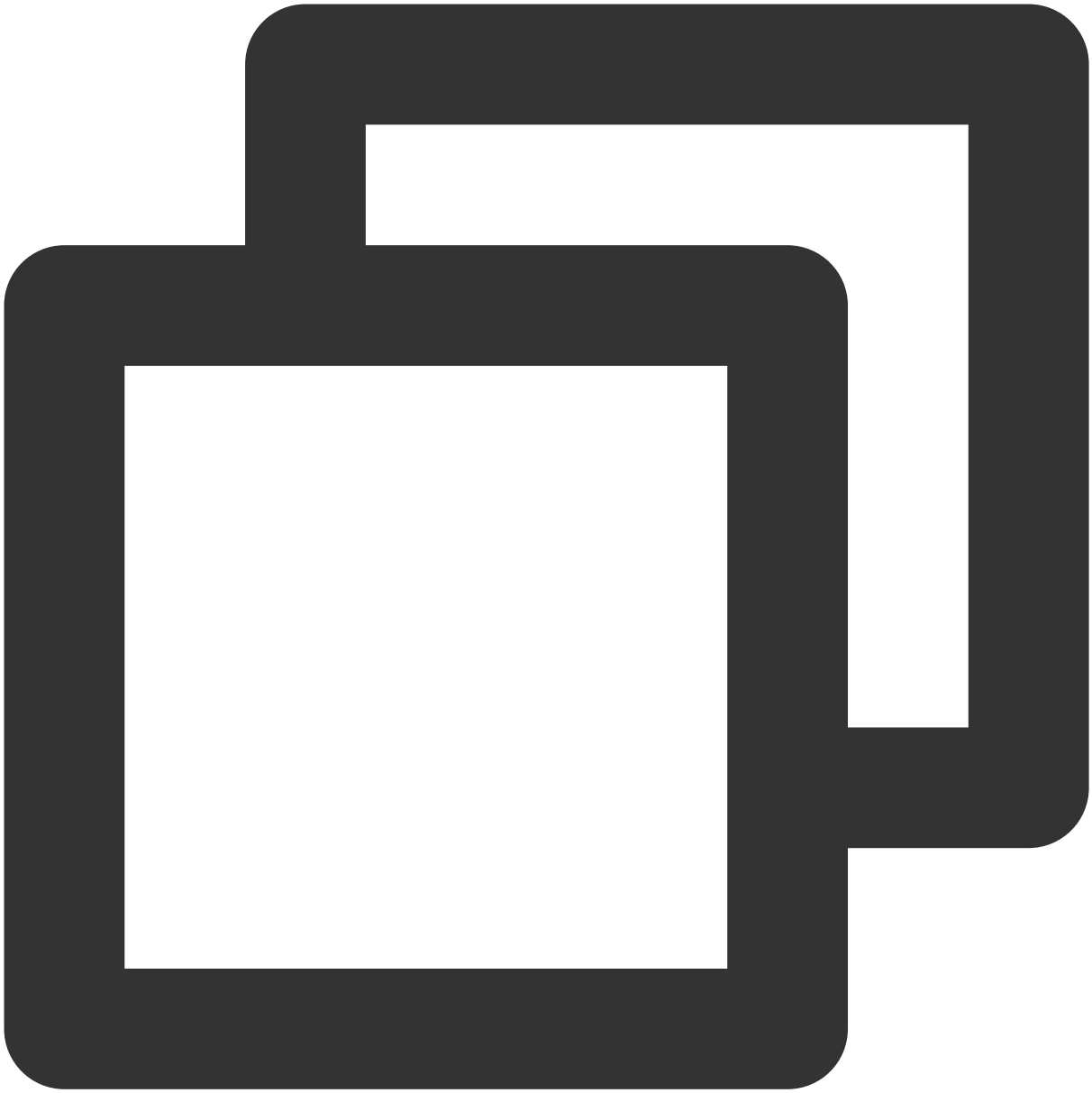


```
{
    "version": "2.0",
    "statement": [
        {
            "action":[
                "name/cos:List*",
                "name/cos:Get*",
                "name/cos:Head*",
                "name/cos:OptionsObject"
            ],
            "resource": "*",
```

```
            "effect": "allow"
        },
        {
            "effect": "allow",
            "action": "monitor:*",
            "resource": "*"
        }
    ]
}
```

예시3: 서브 계정에 쓰기 전용 권한 설정(삭제 제외)

자세한 정책 설정 구문은 다음과 같습니다.

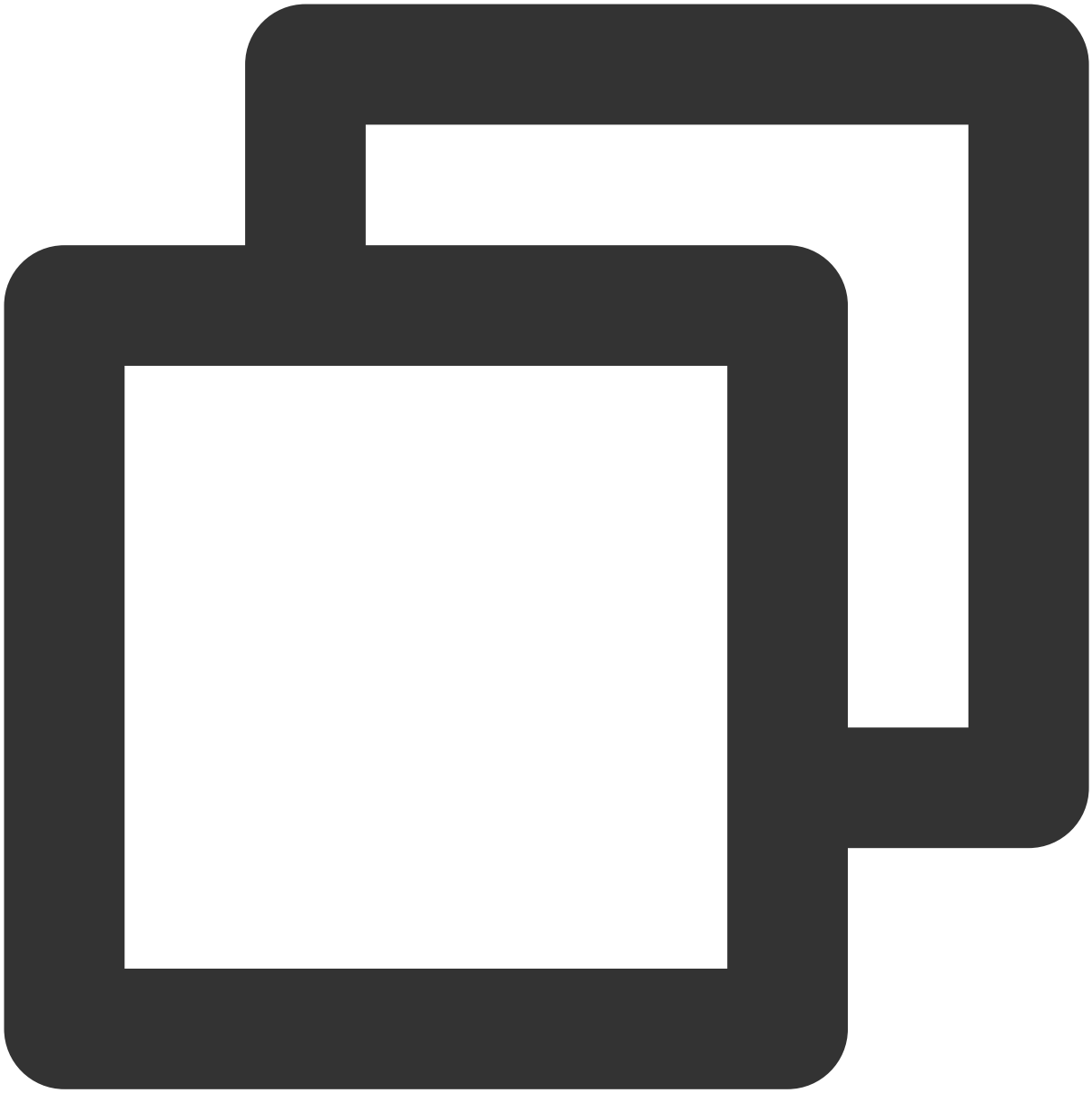


```
{
    "version": "2.0",
    "statement": [
        {
            "effect": "allow",
            "action":[
                "cos:ListParts",
                "cos:PostObject",
                "cos:PutObject*",
                "cos:InitiateMultipartUpload",
                "cos:UploadPart",
```

```
                "cos:UploadPartCopy",
                "cos:CompleteMultipartUpload",
                "cos:AbortMultipartUpload",
                "cos:ListMultipartUploads"
            ],
            "resource": "*"
        }
    ]
}
```

예시4: 특정 IP 범위에 대한 서브 계정 읽기/쓰기 권한 부여

다음 예시는 `192.168.1.0/24` 및 `192.168.2.0/24` 주소 범위에 대해서만 읽기/쓰기 권한을 부여합니다. 더 많은 조건을 입력하려면 조건을 참고하십시오.



```
{
    "version": "2.0",
    "statement": [
        {
            "action":[
    "cos:*"
            ],
            "resource": "*",
            "effect": "allow",
            "condition":{
                "ip_equal":{
```

```
                    "qcs:ip": ["192.168.1.0/24", "192.168.2.0/24"]
                }
            }
        }
    ]
}
```

권한 설정 관련 사례

최종 업데이트 날짜: : 2024-06-24 16:44:03

버킷 정책(Policy)을 이용한 권한 부여 사례

준비 과정

1. 버킷 생성
버킷 정책(Policy)을 통해 특정 버킷에 대해서만 권한을 부여하므로, 먼저 버킷 생성이 필요합니다. 계정 차원에서 권한을 부여해야 하는 경우 본 문서의 CAM을 이용한 권한 부여 사례를 참조하십시오.
2. 권한을 부여 받을 계정의 UIN 준비
본 문서는 타깃 버킷의 루트 계정 UIN이 100000000001이고, 해당 계정의 서브 계정이 100000000011이며, 서브 계정이 권한을 받아야만 타깃 버킷에 액세스할 수 있다고 가정합니다.
설명 :
루트 계정에서 생성한 서브 계정 조회가 필요한 경우 CAM 콘솔에 로그인한 후 사용자 리스트에서 확인할 수 있습니다.
새로운 서브 계정을 생성해야 하는 경우 Creating Sub-user 문서를 확인하십시오.
3. **정책 추가** 대화 상자를 엽니다.
타깃 버킷의 **권한 관리**로 이동하여 **Policy 권한 설정 > 그래픽 설정**을 선택한 후, **정책 추가** 대화 상자를 클릭한 다음 본 문서의 권한 부여 사례를 참고하여 설정을 진행합니다. 정책 추가에 대한 자세한 작업 가이드는 버킷 정책 추가 문서를 참조하십시오.
다음은 몇 가지 권한 부여 사례 예시로, 사용자의 실제 상황에 따라 참고하여 설정합니다.

권한 부여 사례

사례1: 서브 계정에 특정 디렉터리의 모든 권한 부여

설정 정보는 다음과 같습니다.

설정 항목	설정값
효과	허용
사용자	서브 계정을 선택한 후 서브 계정의 UIN을 입력하고, 해당 서브 계정은 반드시 현재 루트 계정의 서브 계정이어야 합니다(예: 100000000011)
리소스	지정된 리소스 경로 선택(예: `folder/sub-folder/*`
작업	모든 작업 선택

사례2: 서브 계정에 특정 디렉터리 내 모든 파일의 읽기 권한 부여

설정 정보는 다음과 같습니다.

설정 항목	설정값
효과	허용
사용자	서브 계정을 선택한 후 서브 계정의 UIN을 입력하고, 해당 서브 계정은 반드시 현재 루트 계정의 서브 계정이어야 합니다(예: 100000000011)
리소스	지정된 리소스 경로 선택(예: `folder/sub-folder/*`
작업	읽기 작업(객체 리스트 나열 포함)

사례3: 서브 계정에 특정 파일의 읽기 및 쓰기 권한 부여

설정 정보는 다음과 같습니다.

설정 항목	설정값
효과	허용
사용자	서브 계정을 선택한 후 서브 계정의 UIN을 입력하고, 해당 서브 계정은 반드시 현재 루트 계정의 서브 계정이어야 합니다(예: 100000000011)
리소스	지정 객체 키 선택(예: `folder/sub-folder/example.jpg`)
작업	모든 작업

사례4: 서브 계정에 특정 디렉터리의 모든 파일에 대한 읽기/쓰기 권한 부여 및 해당 디렉터리의 지정 파일에 대한 읽기/쓰기 금지

해당 사례의 경우 **허용** 정책과 **금지** 정책, 두 가지 정책을 추가해야 합니다.

1. **허용** 정책을 추가합니다. 설정 정보는 다음과 같습니다.

설정 항목	설정값
효과	허용
사용자	서브 계정을 선택한 후 서브 계정의 UIN을 입력하고, 해당 서브 계정은 반드시 현재 루트 계정의 서브 계정이어야 합니다(예: 100000000011)
리소스	지정된 디렉터리의 접두사(예: `folder/sub-folder/*`)
작업	모든 작업

2. **금지** 정책을 추가합니다. 설정 정보는 다음과 같습니다.

설정 항목	설정값

효력	거절
사용자	서브 계정을 선택한 후 서브 계정의 UIN을 입력하고, 해당 서브 계정은 반드시 현재 루트 계정의 서브 계정이어야 합니다(예: 100000000011)
리소스	액세스를 금지할 객체 키(예: `folder/sub-folder/privateobject`)
작업	모든 작업

사례5: 서브 계정에 지정 접두사 파일에 대한 읽기/쓰기 권한 부여

설정 정보는 다음과 같습니다.

설정 항목	설정값
효과	허용
사용자	서브 계정을 선택한 후 서브 계정의 UIN을 입력하고, 해당 서브 계정은 반드시 현재 루트 계정의 서브 계정이어야 합니다(예: 100000000011)
리소스	지정 접두사(예: `folder/sub-folder/prefix`)
작업	모든 작업

CAM을 이용한 권한 부여 사례

계정 차원에서 권한을 부여하는 경우 다음 문서를 참조하여 설정하십시오.

Authorizing Sub-account Full Access to Specific Directory

Authorizing Sub-account Read-only Access to Files in Specific Directory

Authorizing Sub-account Read/Write Access to Specific File

Authorizing Sub-account Read-only Access to COS Resources

Authorizing Sub-account Read/Write Access to Specific File

Authorizing Sub-account Read/Write Access to Files with Specified Prefix

Authorizing Another Account Read/Write Access to Specific Files

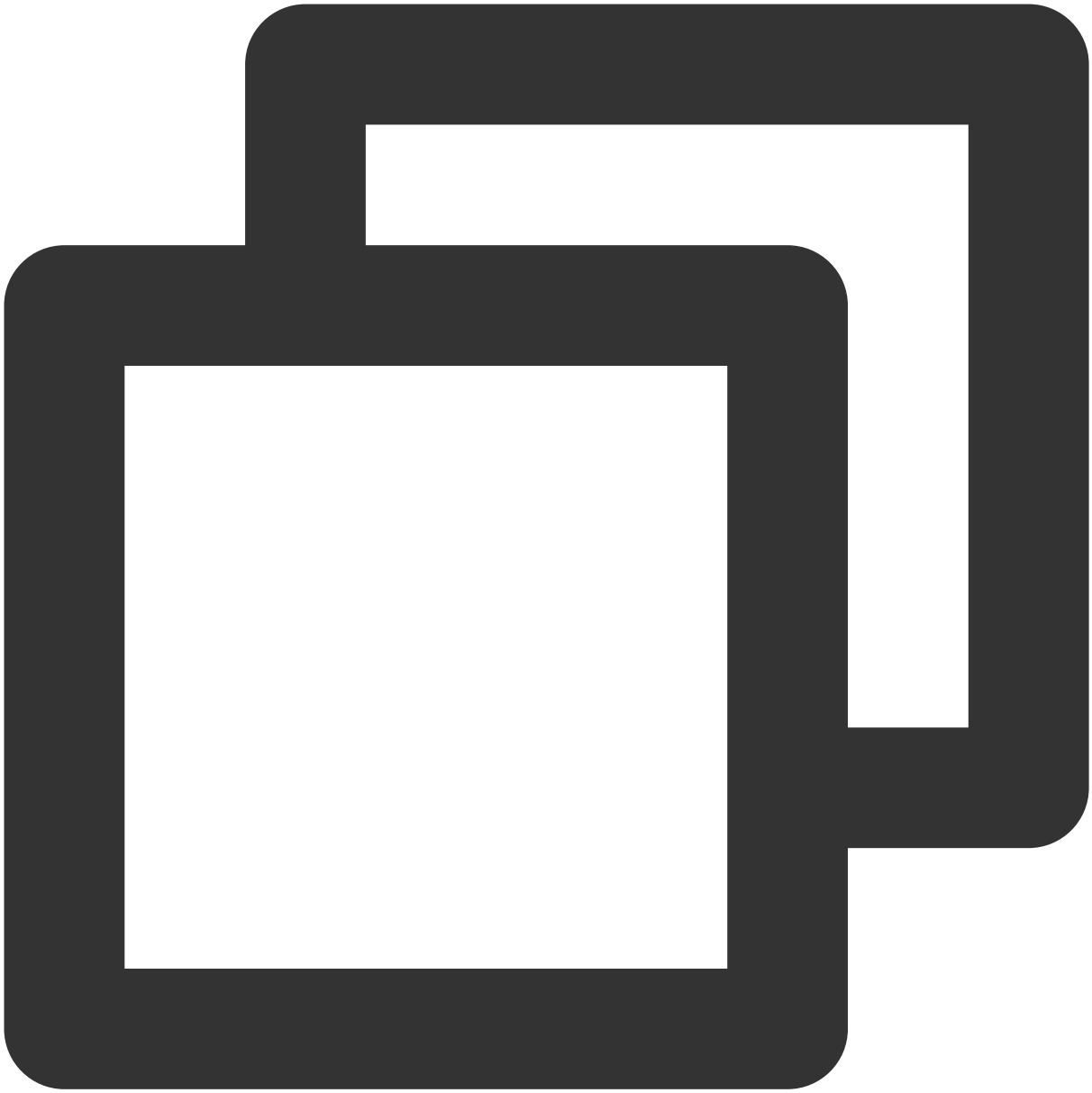
COS API 권한 부여 정책 사용 가이드

최종 업데이트 날짜: : 2024-06-24 16:44:04

주의 :
서브 계정 또는 협업 파트너에게 API 작업 권한 부여 시, 반드시 작업 필요에 따라 최소 권한의 원칙으로 권한을 부여해야 합니다. 서브 계정 또는 협업 파트너에게 직접 모든 리소스 `(resource:*)` 또는 모든 작업 `(action:*)` 권한을 부여하는 경우 권한 범위가 너무 커 데이터 보안 리스크가 발생할 수 있습니다.

개요

Cloud Object Storage(COS)에서 임시 키 서비스를 사용할 때 각 COS API 작업 별로 서로 다른 작업 권한이 필요하며, 동시에 1개 작업 또는 일련의 작업을 지정할 수 있습니다.
COS API 권한 부여 정책(policy)은 일종의 JSON 문자열입니다. 예를 들어, APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 경로 접두사가 doc인 업로드 작업(간편 업로드, 폼 업로드, 멀티파트 업로드 등의 작업 포함) 권한을 부여한 경우, 경로 접두사가 doc2인 다운로드 작업 권한의 정책 내용은 다음과 같습니다.



```
{
    "version": "2.0",
    "statement": [{
            "action": [
                //간편 업로드 작업
                "name/cos:PutObject",
                //폼을 사용한 객체 업로드
                "name/cos:PostObject",
                //멀티파트 업로드: 파트 초기화 작업
                "name/cos:InitiateMultipartUpload",
                //멀티파트 업로드: List에서 진행 중인 멀티파트 업로드
```

```
                "name/cos:ListMultipartUploads",
                //멀티파트 업로드: List에서 업로드 완료한 파트 작업
                "name/cos:ListParts",
                //멀티파트 업로드: 멀티파트 업로드 작업
                "name/cos:UploadPart",
                //멀티파트 업로드: 모든 멀티파트 업로드 작업 완료
                "name/cos:CompleteMultipartUpload",
                //멀티파트 업로드 작업 취소
                "name/cos:AbortMultipartUpload"
            ],
            "effect": "allow",
            "resource": [
                "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
            ]
        },
        {
            "action": [
                //다운로드 작업
                "name/cos:GetObject"
            ],
            "effect": "allow",
            "resource": [
                "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc2/*
            ]
        }
    ]
}
```

권한 부여 정책(policy) 요소 설명

이름	설명
version	정책 구문 버전으로, 기본값은 2.0입니다.
effect	allow(허용), deny(명시적 거부) 두 가지 상태가 있습니다.
resource	모든 리소스, 지정된 경로 접두사가 있는 리소스, 지정된 절대 경로의 리소스 또는 이들의 조합이 될 수 있는 권한 부여된 작업의 특정 데이터입니다. **참고:** 경로가 중국어인 경우 중국어 입력을 그대로 유지하십시오. 예: `examplebucket-1250000000/폴더/파일명.txt` .
action	여기에서는 COS API를 지칭하며, 필요에 따라 1개 또는 일련의 작업 조합이나 모든 작업(`*`)을 지정합니다. 예: `name/cos:GetService` , **영어 대소문자 구분에 유의하십시오.**

condition	규제 조건으로, 입력하지 않아도 됩니다. 자세한 설명은 condition을 참고하십시오.

각 COS API 별 권한 정책 설정 예시는 다음과 같습니다.

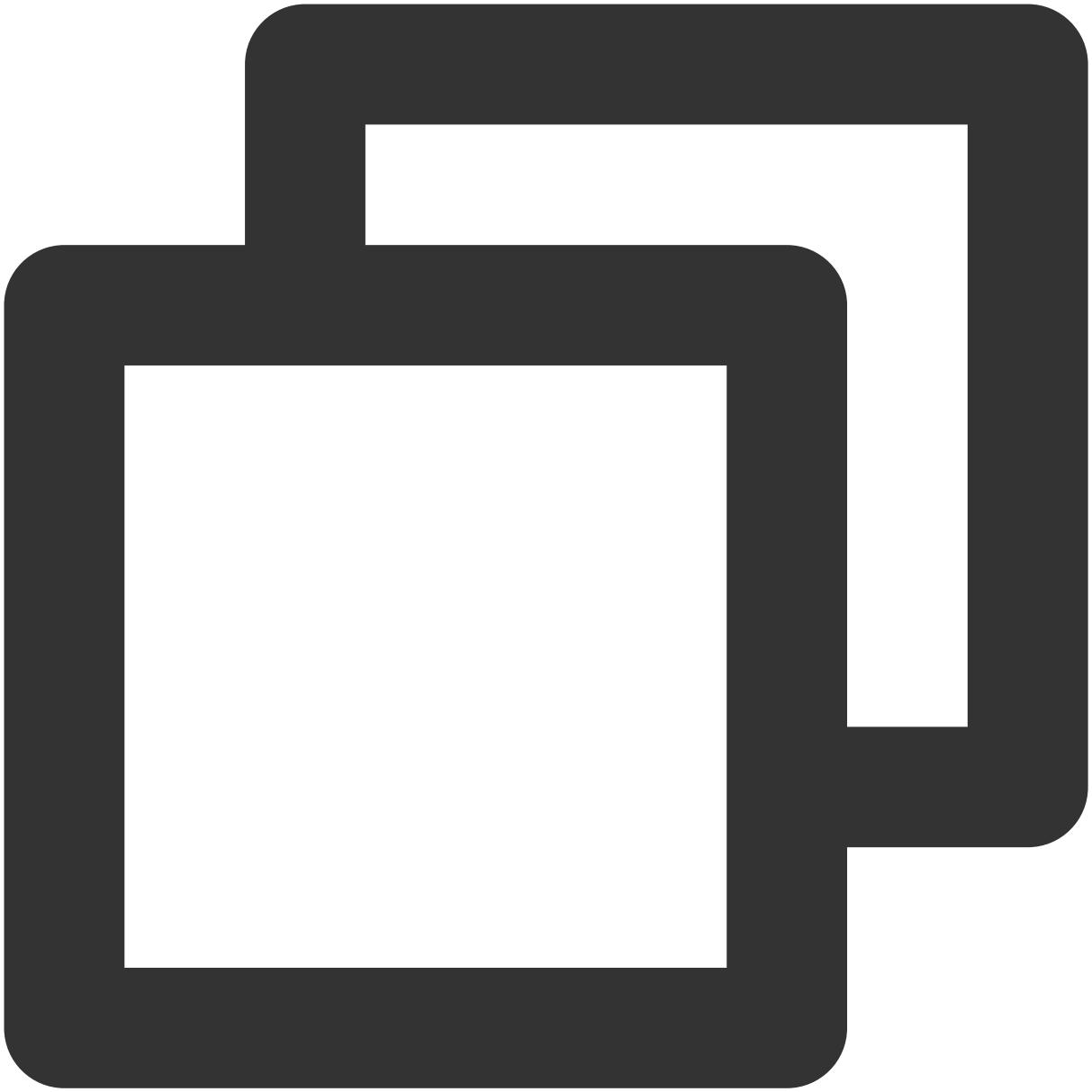
Service API

버킷 리스트 조회

API는 GET Service이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:GetService로 설정하고 resource는 `*` 로 설정합니다.

예시

버킷 리스트 조회 권한을 부여하는 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:GetService"
      ],
      "effect": "allow",
      "resource": [
        "*"
      ]
```

```
      }
    ]
}
```

Bucket API

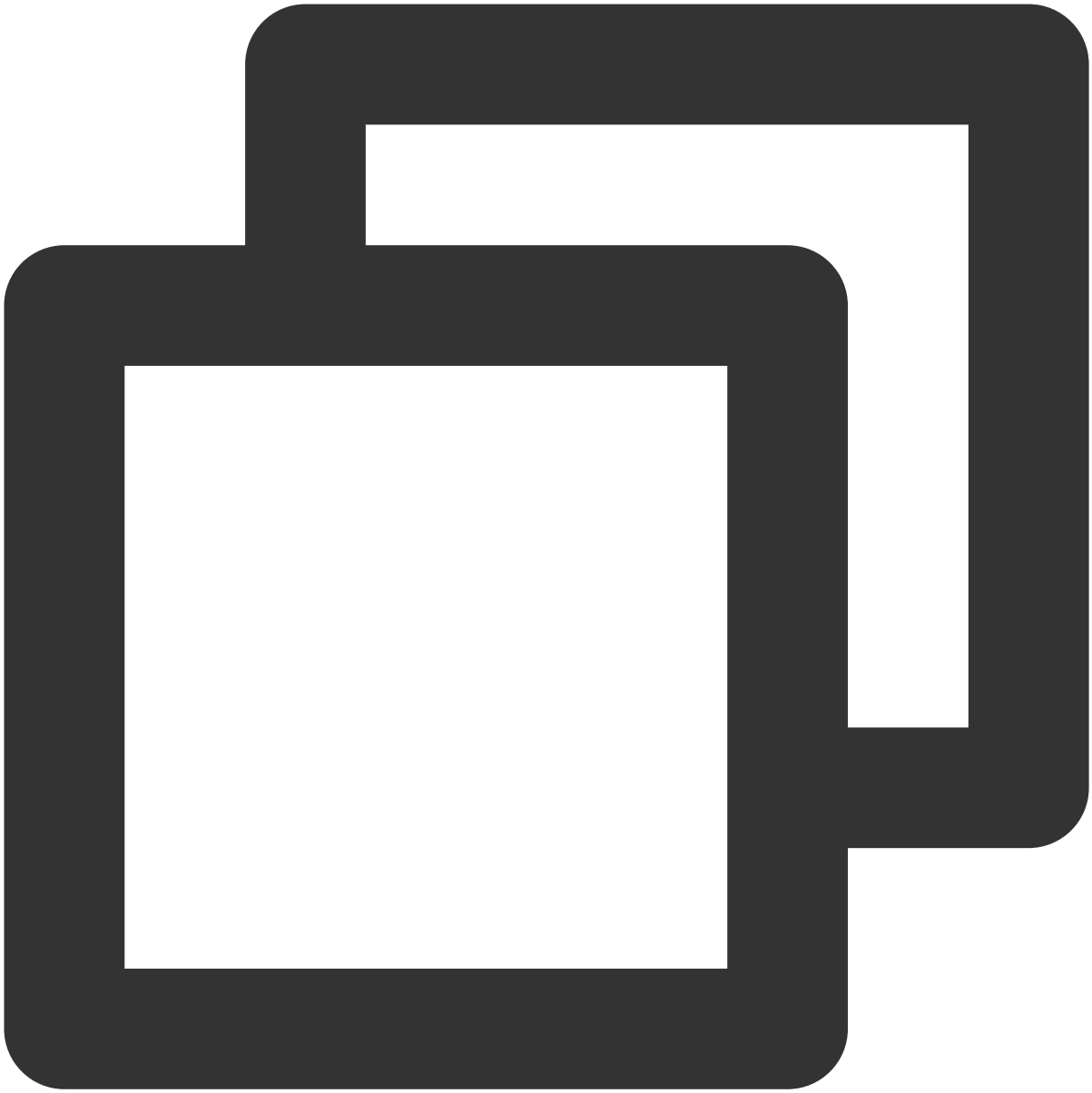
Bucket API 정책의 resource는 다음과 같을 수 있습니다.
모든 리전의 버킷에 대한 작업
정책의 resource를 `*` 로 설정하며, **해당 정책에서 한정하는 리소스 범위의 권한 범위가 너무 커 데이터 보안 리스크가 발생할 수 있으므로 설정에 유의하십시오.**
특정 리전 버킷의 작업만 허용
APPID가 1250000000이고, 리전이 베이징(ap-beijing)인 버킷 examplebucket-1250000000의 작업만 허용하는 경우, 정책의 resource는 `qcs::cos:ap-beijing:uid/1250000000:*` 가 됩니다.
특정 리전의 특정 이름 버킷 작업만 허용
APPID가 1250000000이고, 리전이 ap-beijing이며, 이름이 examplebucket-1250000000인 버킷의 작업만 허용하는 경우, 정책의 resource는 `qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/*` 가 됩니다.
Bucket API 정책의 action은 작업에 따라 값이 달라집니다. 다음은 Bucket API 권한 부여 정책에 관한 예시로, 기타 Bucket API 권한 부여 정책에 참고할 수 있습니다.

버킷 생성

API는 PUT Bucket이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:PutBucket으로 설정합니다.

예시

사용자 APPID 1250000000에 버킷 생성 권한을 부여하고, 리전이 베이징 리전이며 버킷 이름이 examplebucket-1250000000인 버킷을 생성할 경우 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PutBucket"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

```
        }
    ]
}
```

설명：

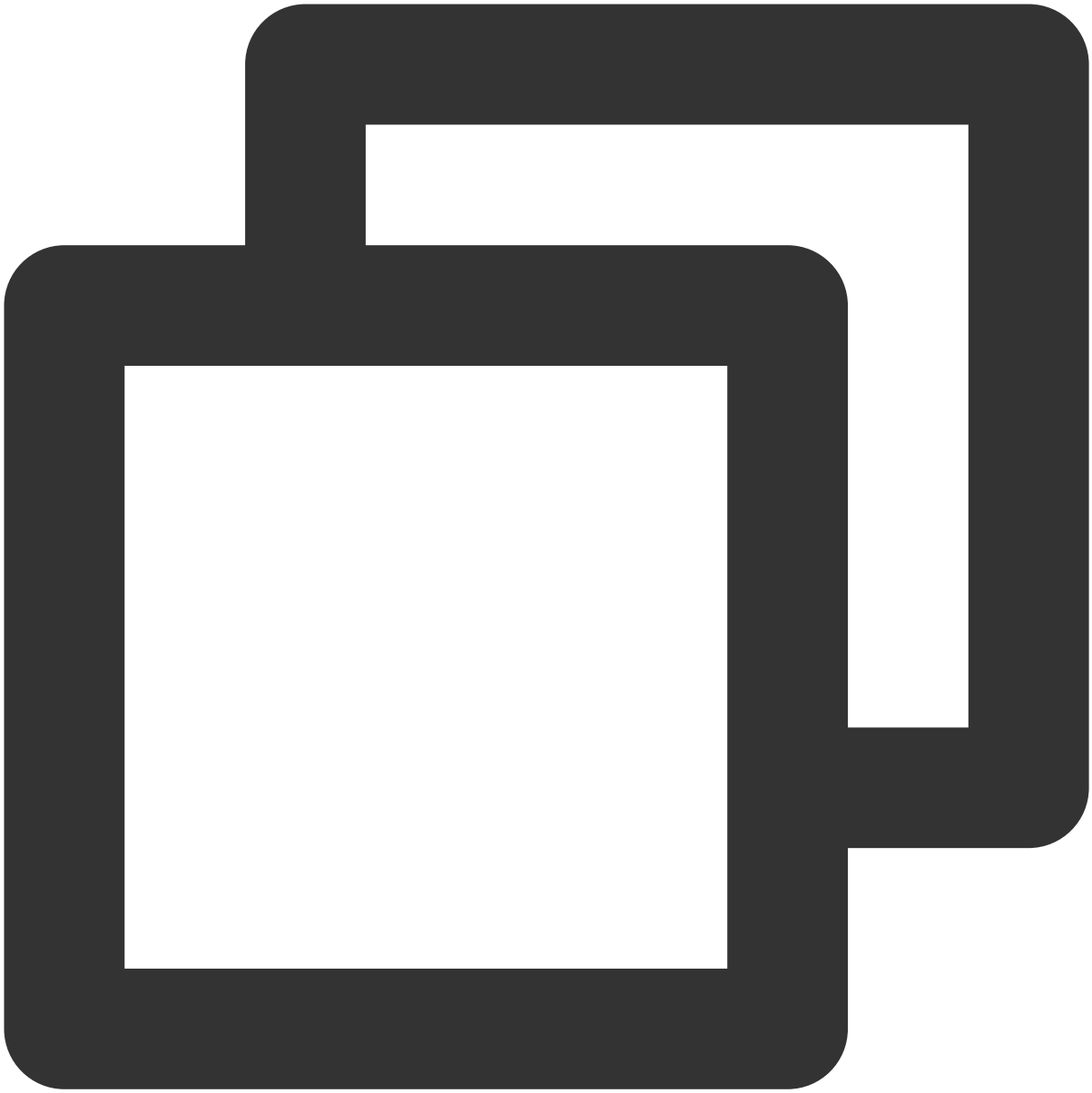
버킷 이름은 이름 생성 규칙에 부합해야 하며, 자세한 내용은 버킷 이름 생성 규칙을 참고하십시오.

버킷 및 해당 권한 인덱스

API는 HEAD Bucket이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:HeadBucket으로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷에 대한 인덱스 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:HeadBucket"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

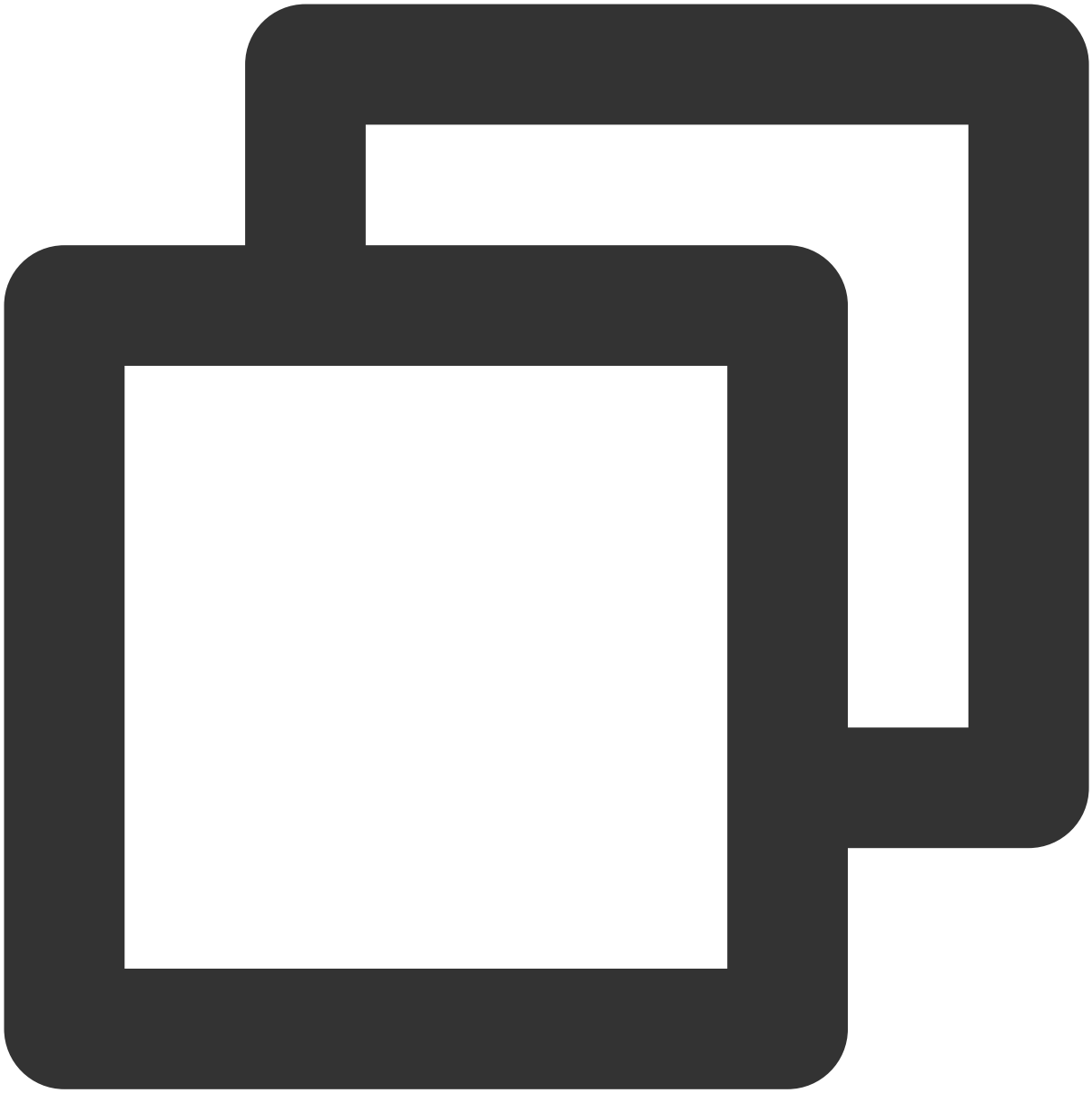
```
      }
    ]
}
```

객체 리스트 쿼리

API는 GET Bucket이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:GetBucket으로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 객체 리스트 조회 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:GetBucket"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

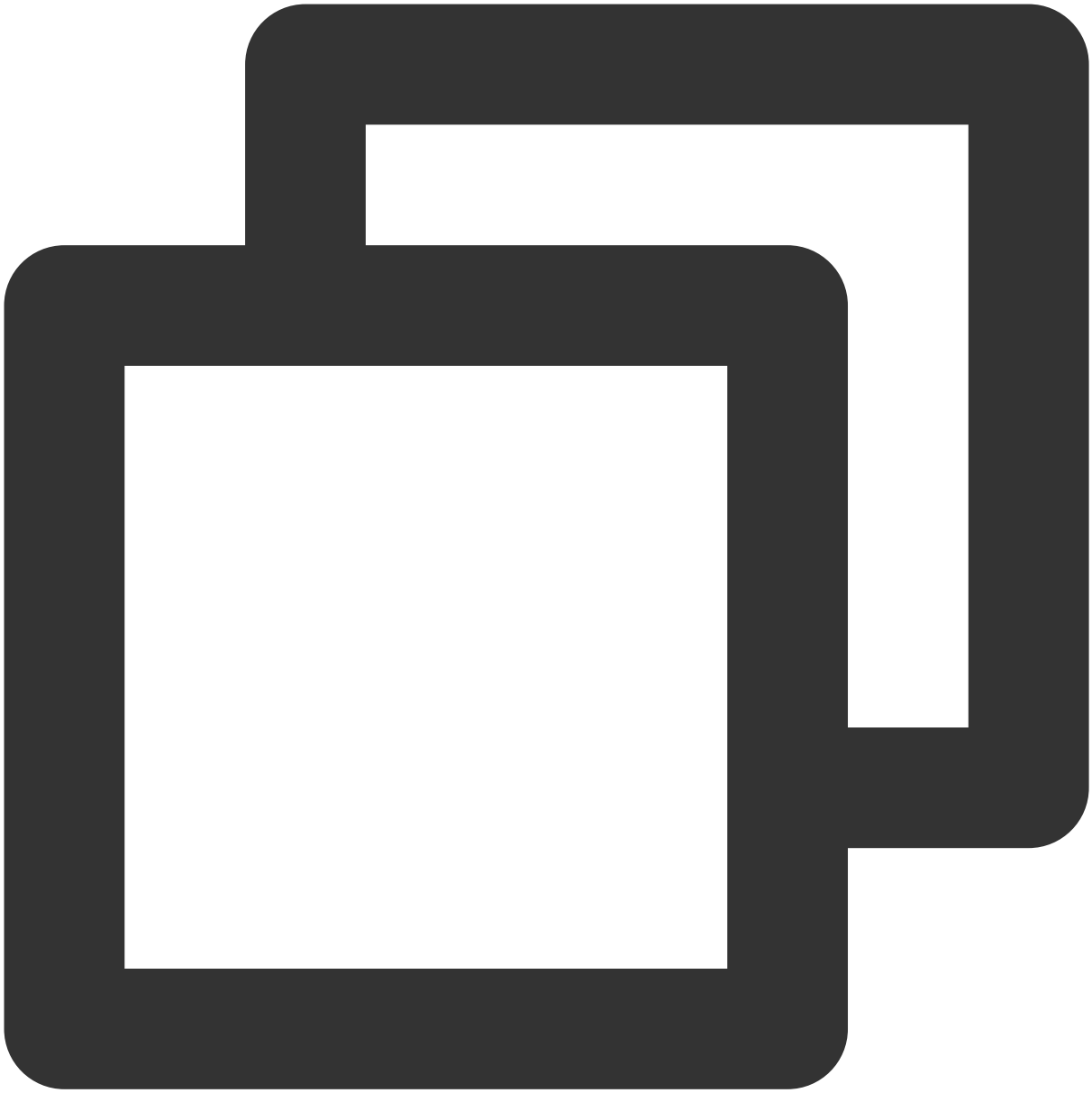
```
        }
    ]
}
```

버킷 삭제

API는 Delete Bucket이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:DeleteBucket으로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷에 대한 삭제 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:DeleteBucket"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

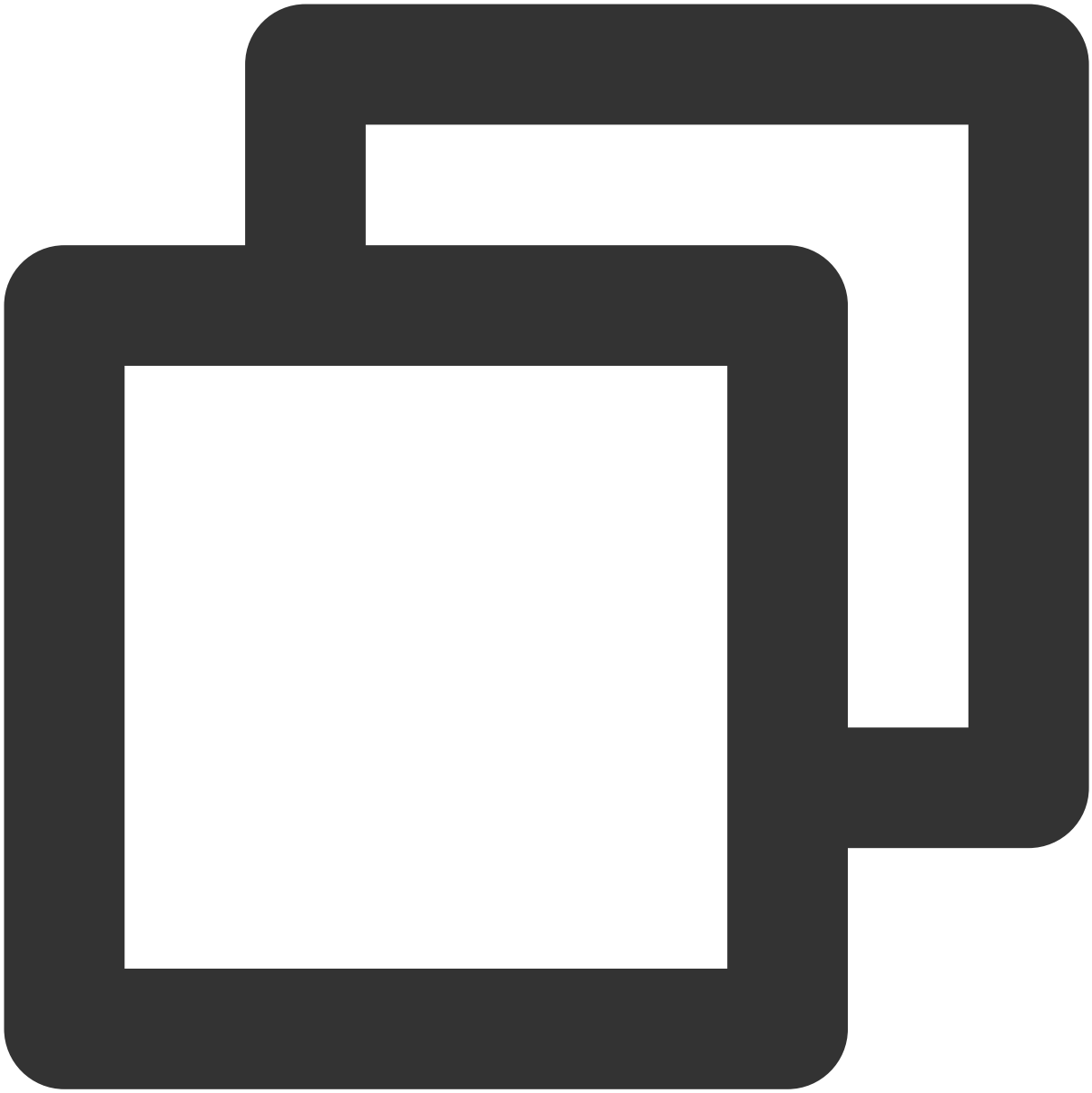
```
        }
    ]
}
```

버킷 ACL 설정

API는 Put Bucket ACL이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:PutBucketACL로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 ACL 설정 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PutBucketACL"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

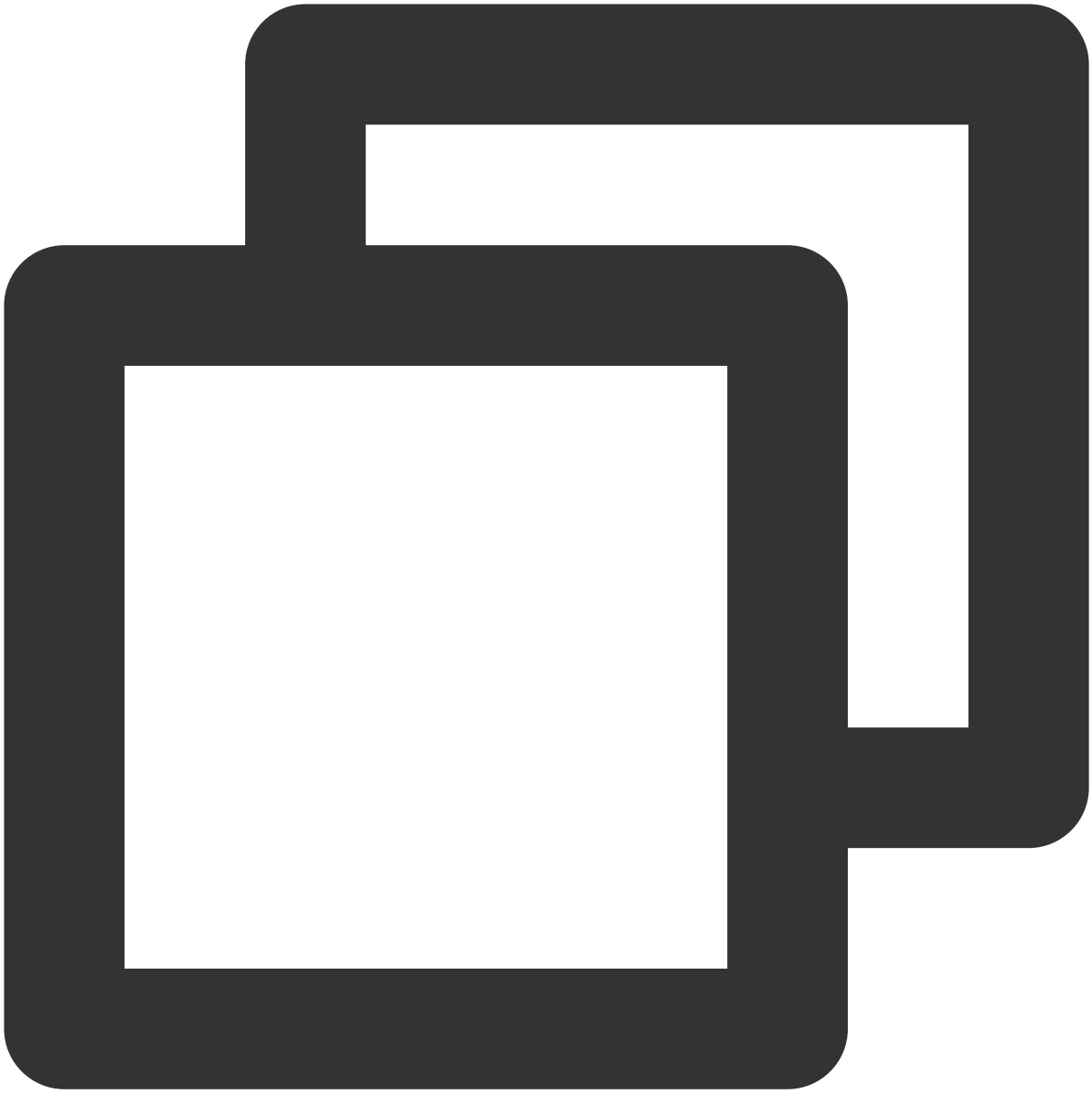
```
        }
    ]
}
```

버킷 ACL 조회

API는 GET Bucket acl이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:GetBucketACL로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 ACL 획득 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:GetBucketACL"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

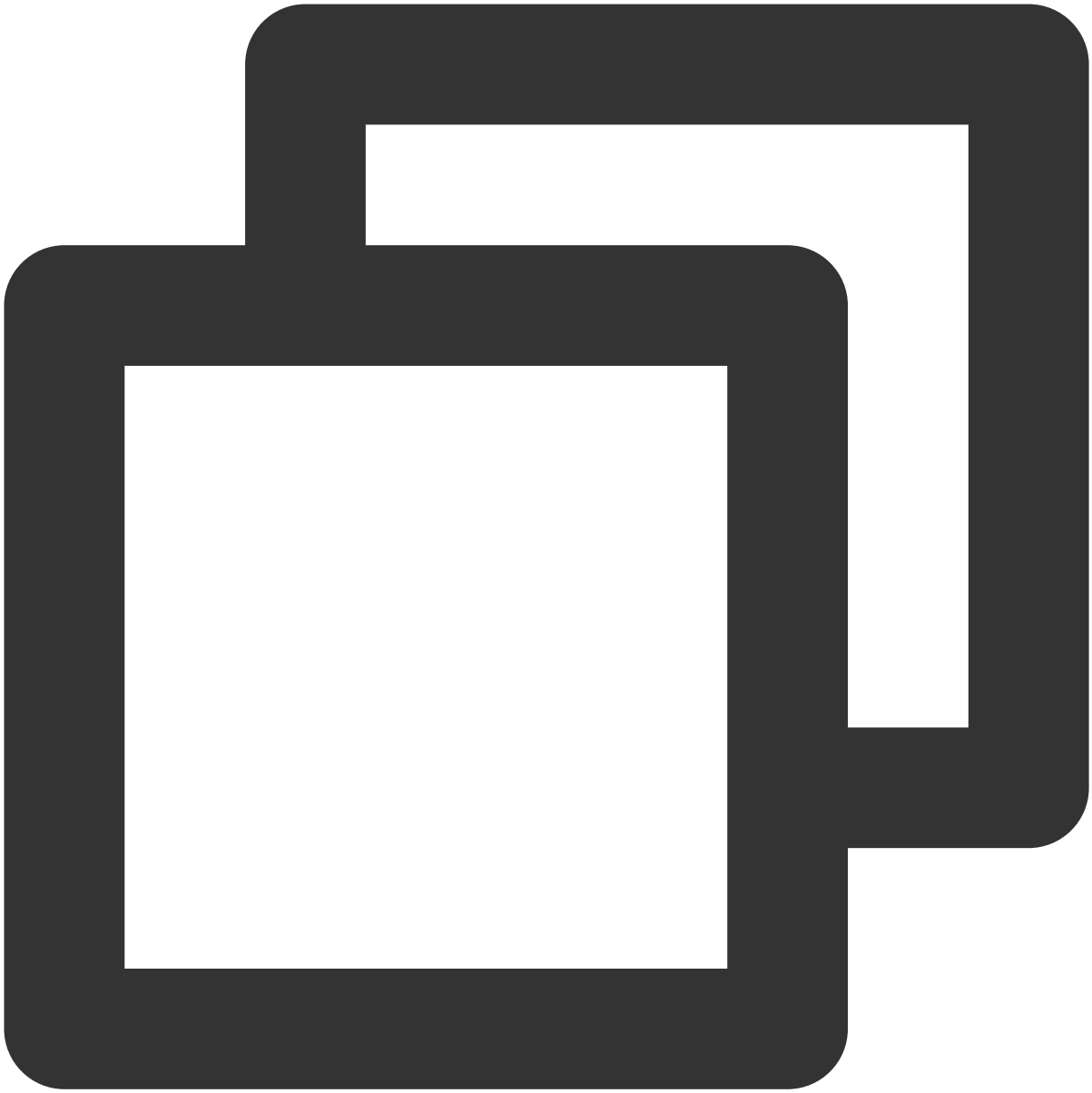
```
        }
    ]
}
```

크로스 도메인 구성 설정

API는 PUT Bucket cors이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:PutBucketCORS로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 크로스 도메인 구성 설정 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PutBucketCORS"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

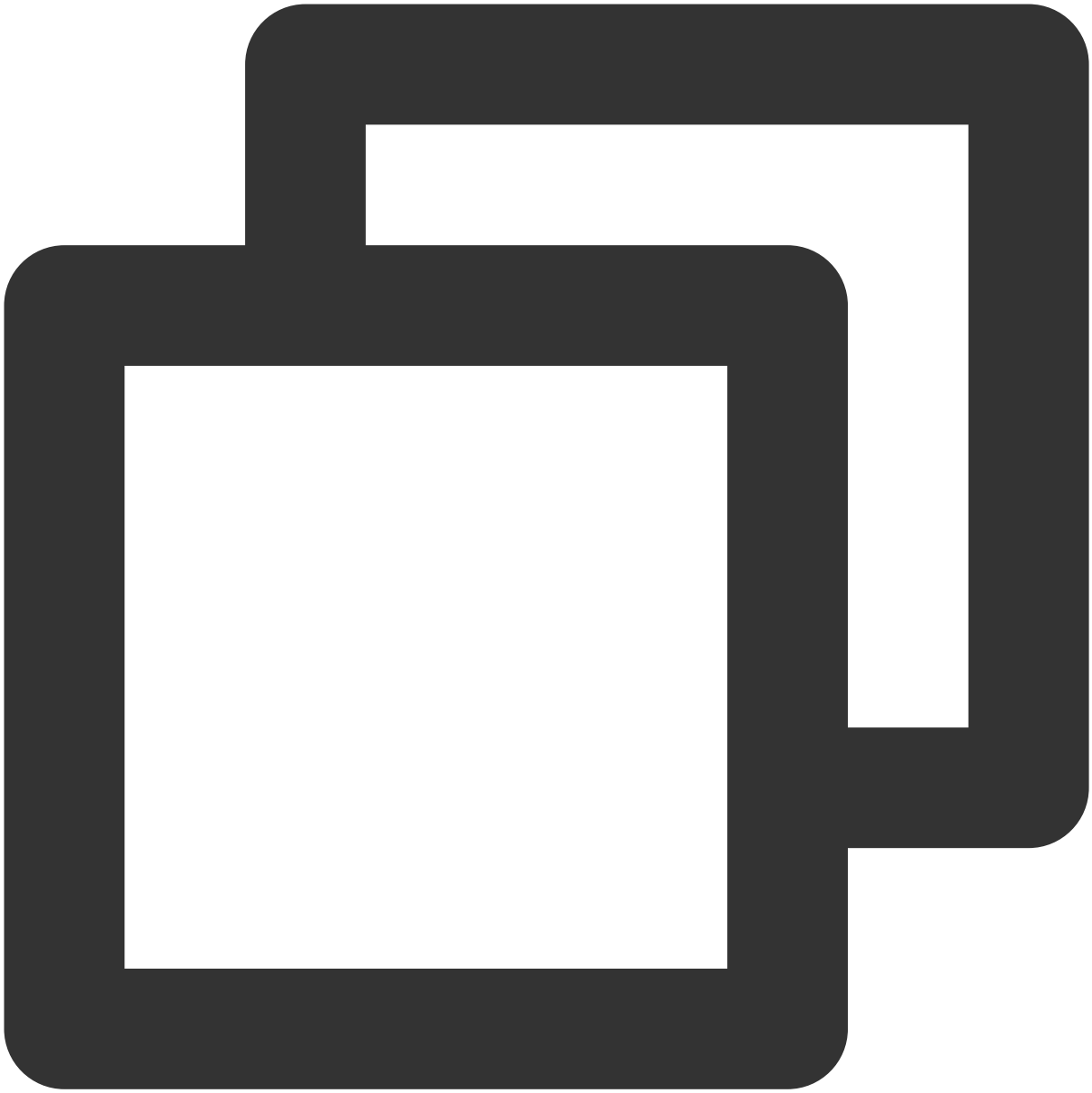
```
        }
    ]
}
```

크로스 도메인 설정 조회

API는 GET Bucket cors이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:GetBucketCORS로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 크로스 도메인 설정 조회 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:GetBucketCORS"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

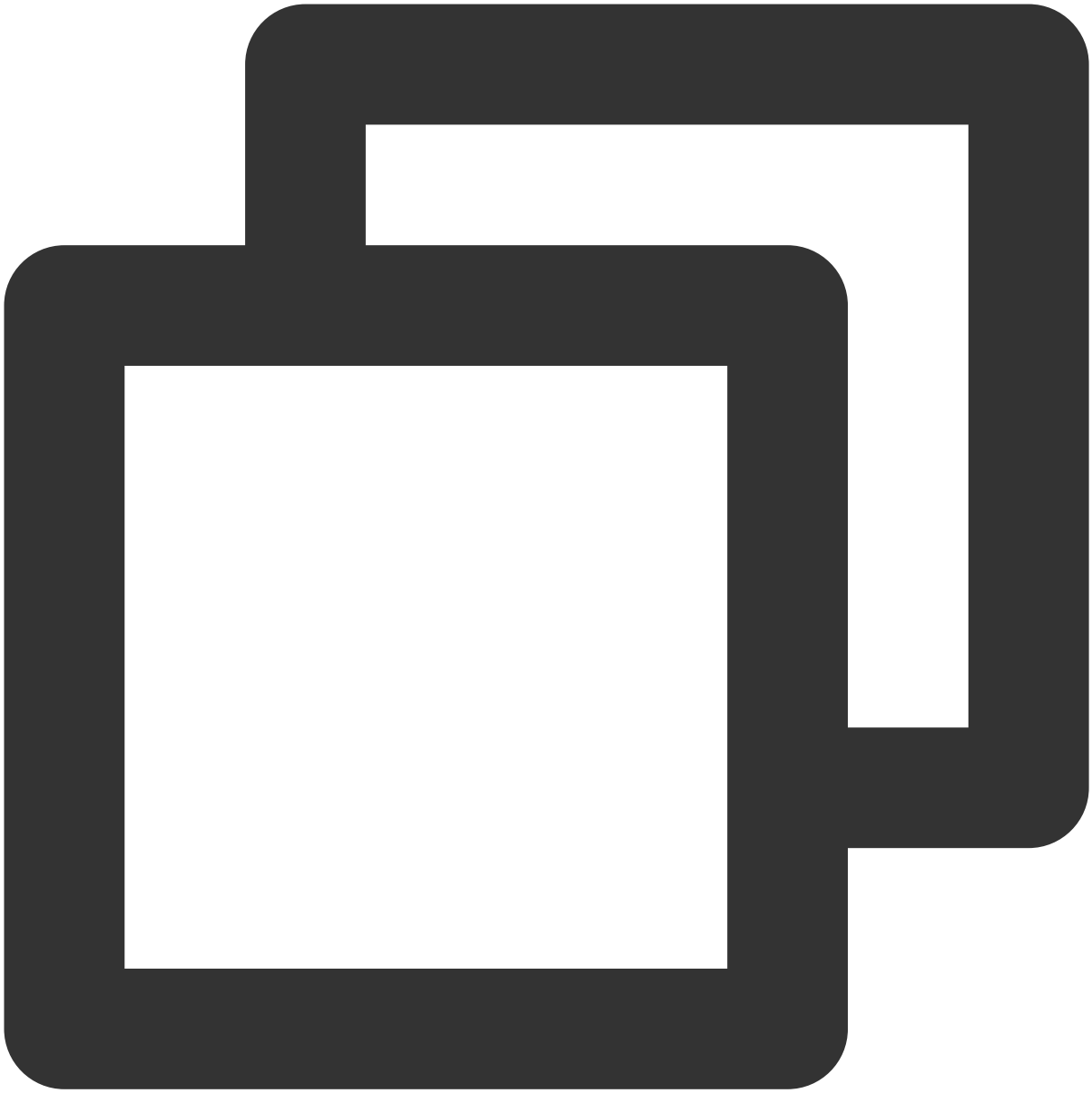
```
      }
    ]
}
```

크로스 도메인 설정 삭제

API는 DELETE Bucket cors이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:DeleteBucketCORS로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 크로스 도메인 설정 삭제 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:DeleteBucketCORS"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

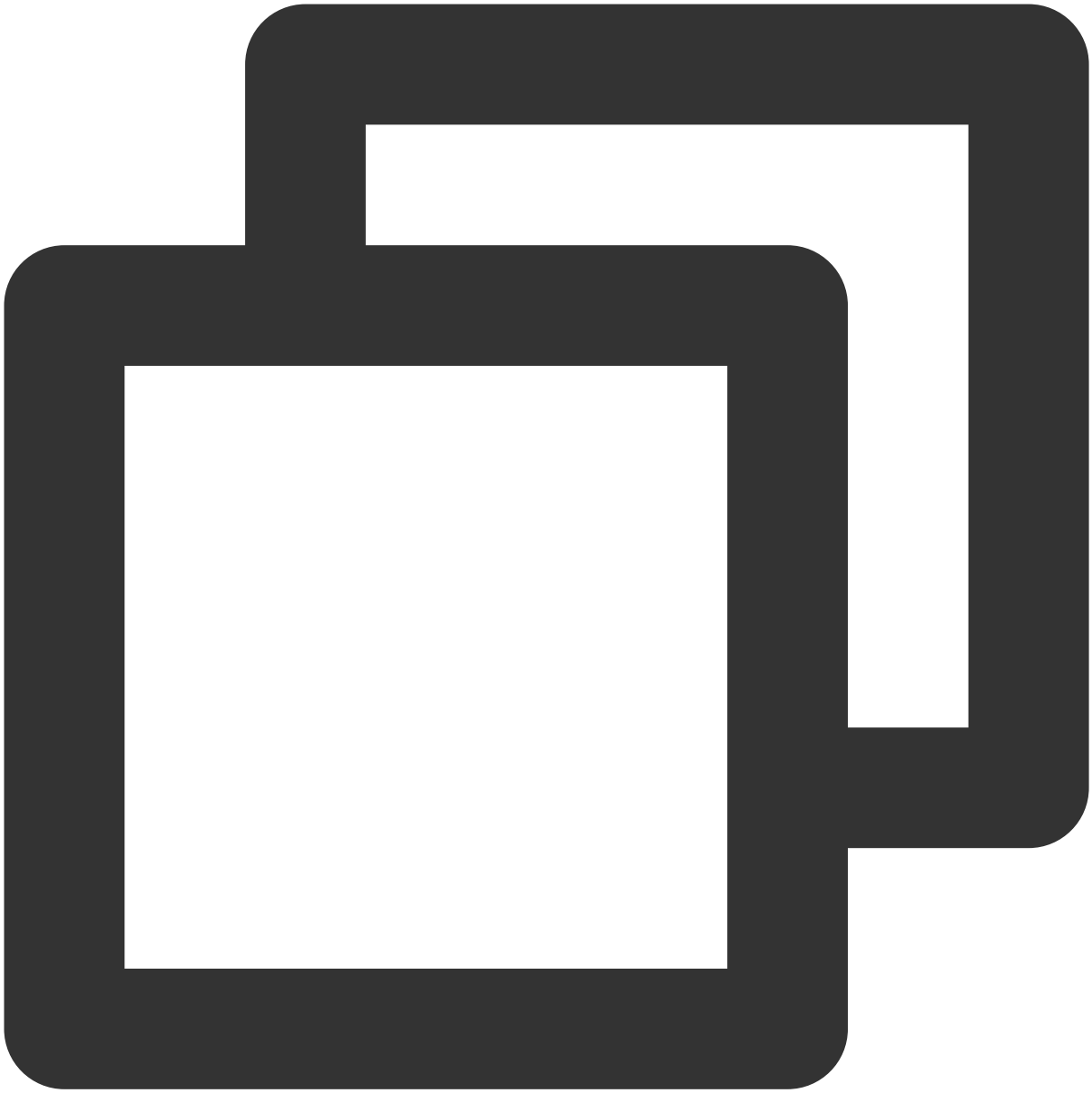
```
        }
    ]
}
```

라이프사이클 설정

API는 PUT Bucket lifecycle이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:PutBucketLifecycle로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 라이프사이클 설정 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PutBucketLifecycle"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

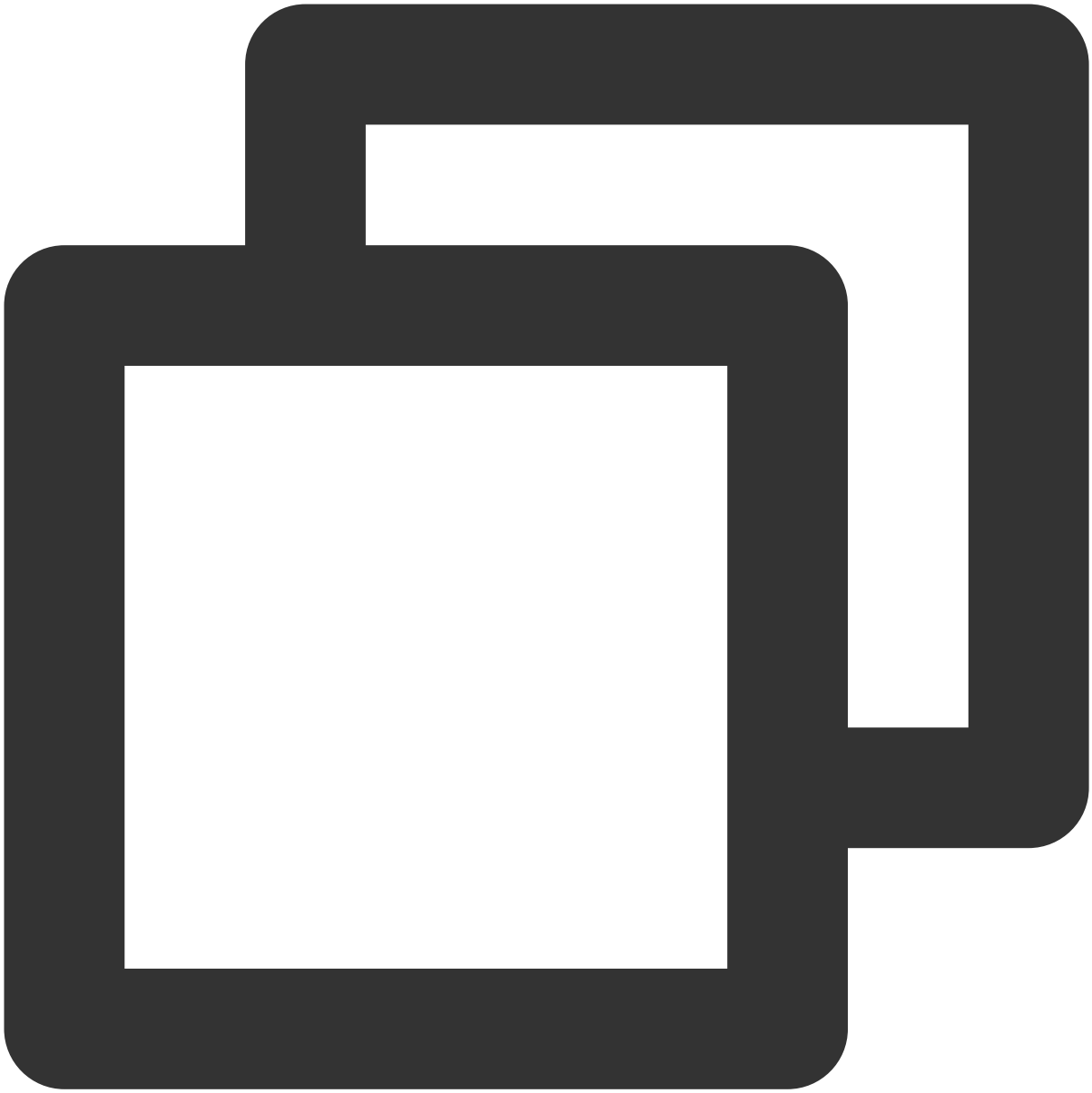
```
        }
    ]
}
```

라이프사이클 조회

API는 GET Bucket lifecycle이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:GetBucketLifecycle로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 라이프사이클 설정 조회 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:GetBucketLifecycle"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

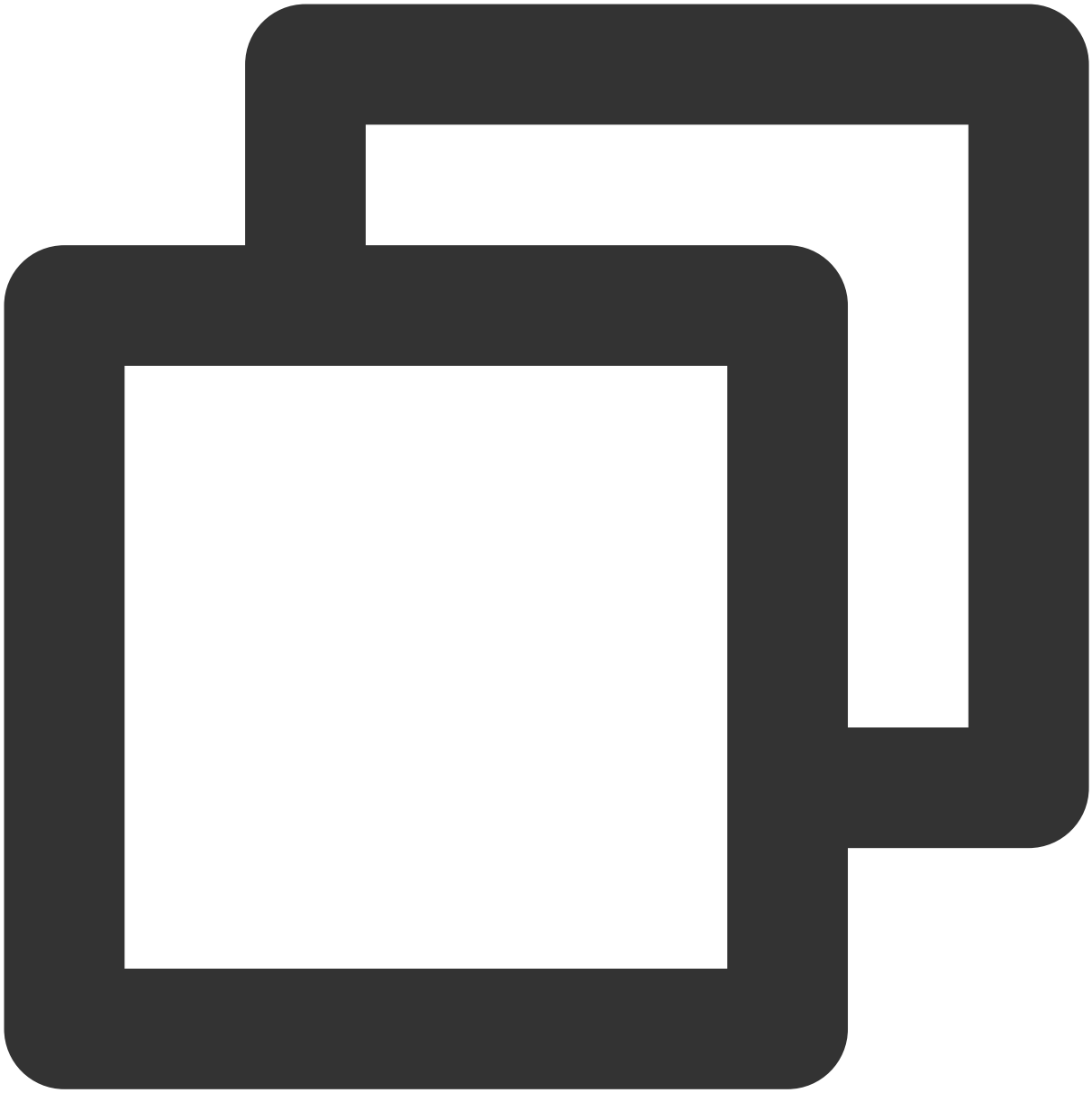
```
        }
    ]
}
```

라이프사이클 삭제

API는 DELETE Bucket lifecycle이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:DeleteBucketLifecycle로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 라이프사이클 설정 삭제 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:DeleteBucketLifecycle"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

```
      }
    ]
  }
```

Object API

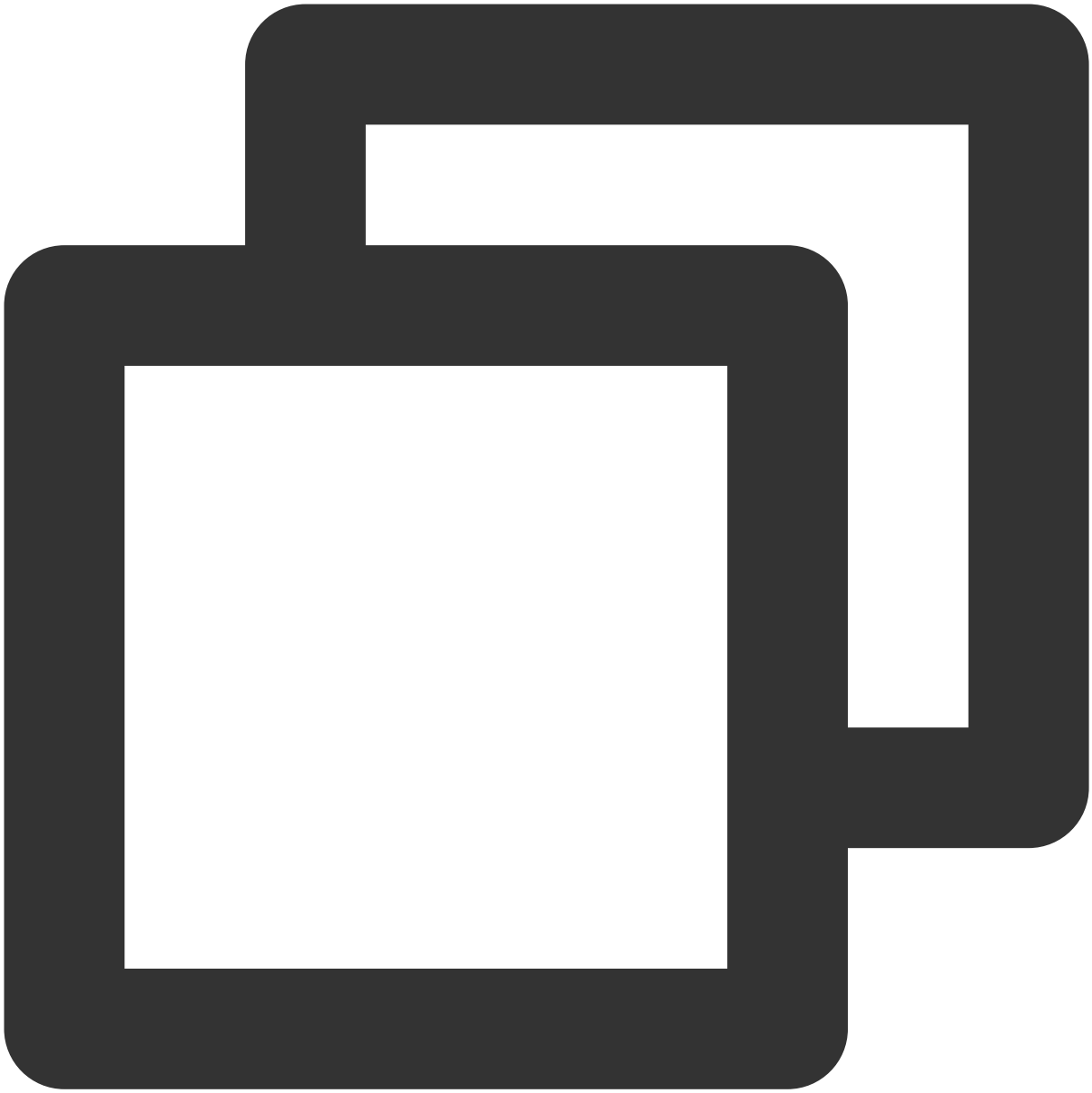
Object API 정책의 resource는 다음과 같을 수 있습니다.
모든 객체에 대한 작업을 허용할 경우 정책의 resource를 `*` 로 설정합니다.
특정 버킷의 모든 객체만 작업할 수 있도록 설정하는 경우, 즉 APPID가 1250000000이고, 리전이 ap-beijing이며, 이름이 examplebucket-1250000000인 버킷의 모든 객체의 작업만 허용하는 경우, 정책의 resource는 `qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/*` 가 됩니다.
특정 버킷의 특정 경로 접두사의 모든 객체 객체만 작업할 수 있도록 설정하는 경우, 즉 APPID가 1250000000이고, 리전이 ap-beijing이며, 이름이 examplebucket-1250000000인 버킷의 경로 접두사가 doc인 모든 객체의 작업만 허용하는 경우, 정책의 resource는 `qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*` 가 됩니다.
특정 절대 경로의 객체만 작업할 수 있도록 설정하는 경우, 즉 APPID가 1250000000이고, 리전이 ap-beijing이며, 이름이 examplebucket-1250000000인 버킷의 절대 경로가 `doc/audio.mp3` 인 객체의 작업만 허용하는 경우, 정책의 resource는 `qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/audio.mp3` 가 됩니다.
Object API 정책의 action은 작업에 따라 값이 달라지며, 모든 Object API 권한 부여 정책은 다음과 같습니다.

객체 간편 업로드

API는 PUT Object이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:PutObject로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 경로 접두사 doc에서의 간편 업로드 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
 "version": "2.0",
 "statement": [
   {
     "action": [
       "name/cos:PutObject"
     ],
     "effect": "allow",
     "resource": [
       "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
     ]
```

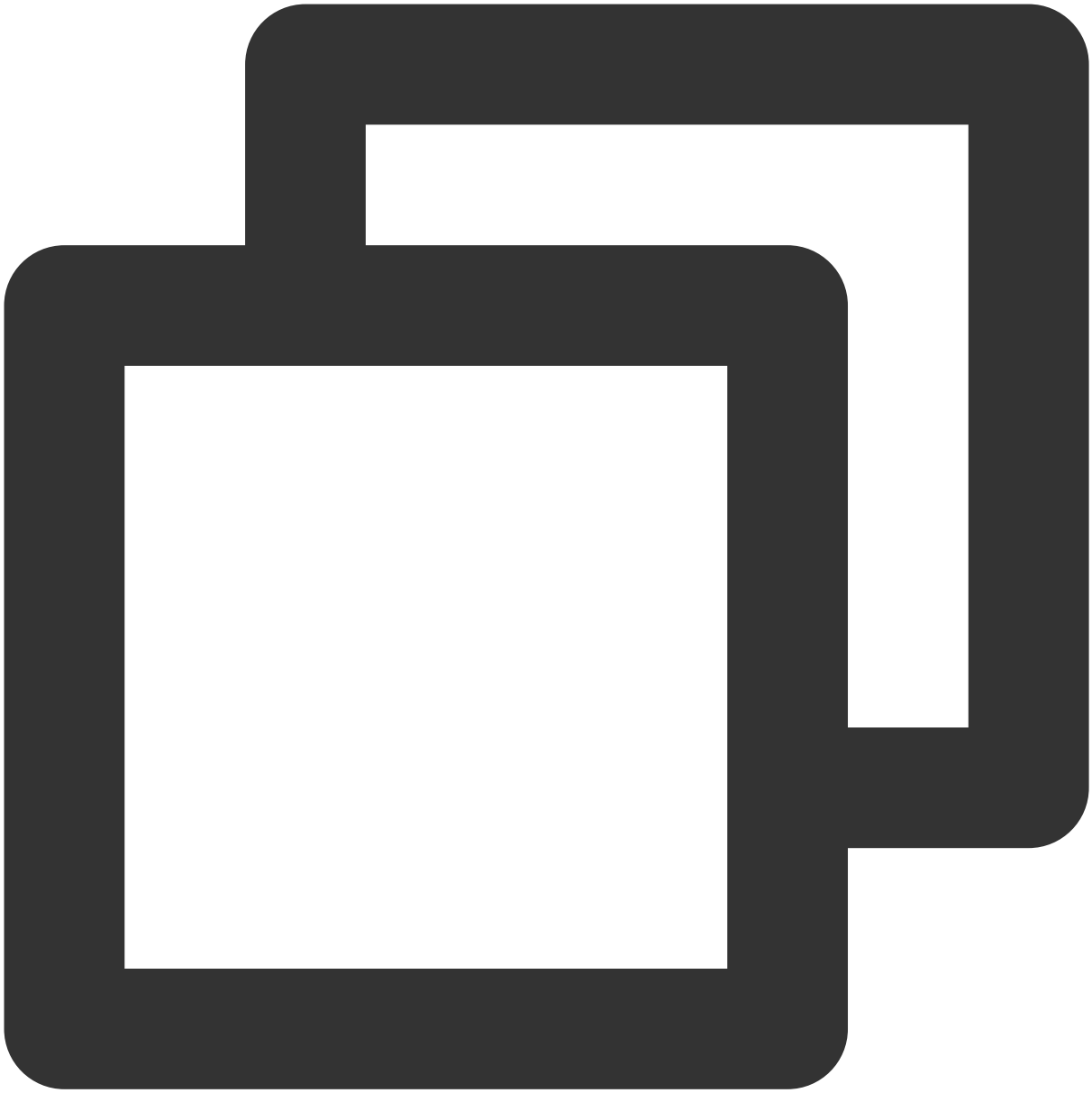
```
      }
    ]
  }
```

멀티파트 업로드

멀티파트 업로드에는 Initiate Multipart Upload, List Multipart Uploads, List Parts, Upload Part, Complete Multipart Upload, Abort Multipart Upload가 포함되며, 해당 작업 권한을 부여할 경우 정책의 action을 `"name/cos:InitiateMultipartUpload","name/cos:ListMultipartUploads","name/cos:ListParts","name/cos:UploadPart","name/cos:CompleteMultipartUpload","name/cos:AbortMultipartUpload"` 의 집합으로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 경로 접두사 doc에서 멀티파트 업로드 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:InitiateMultipartUpload",
        "name/cos:ListMultipartUploads",
        "name/cos:ListParts",
        "name/cos:UploadPart",
        "name/cos:CompleteMultipartUpload",
        "name/cos:AbortMultipartUpload"
```

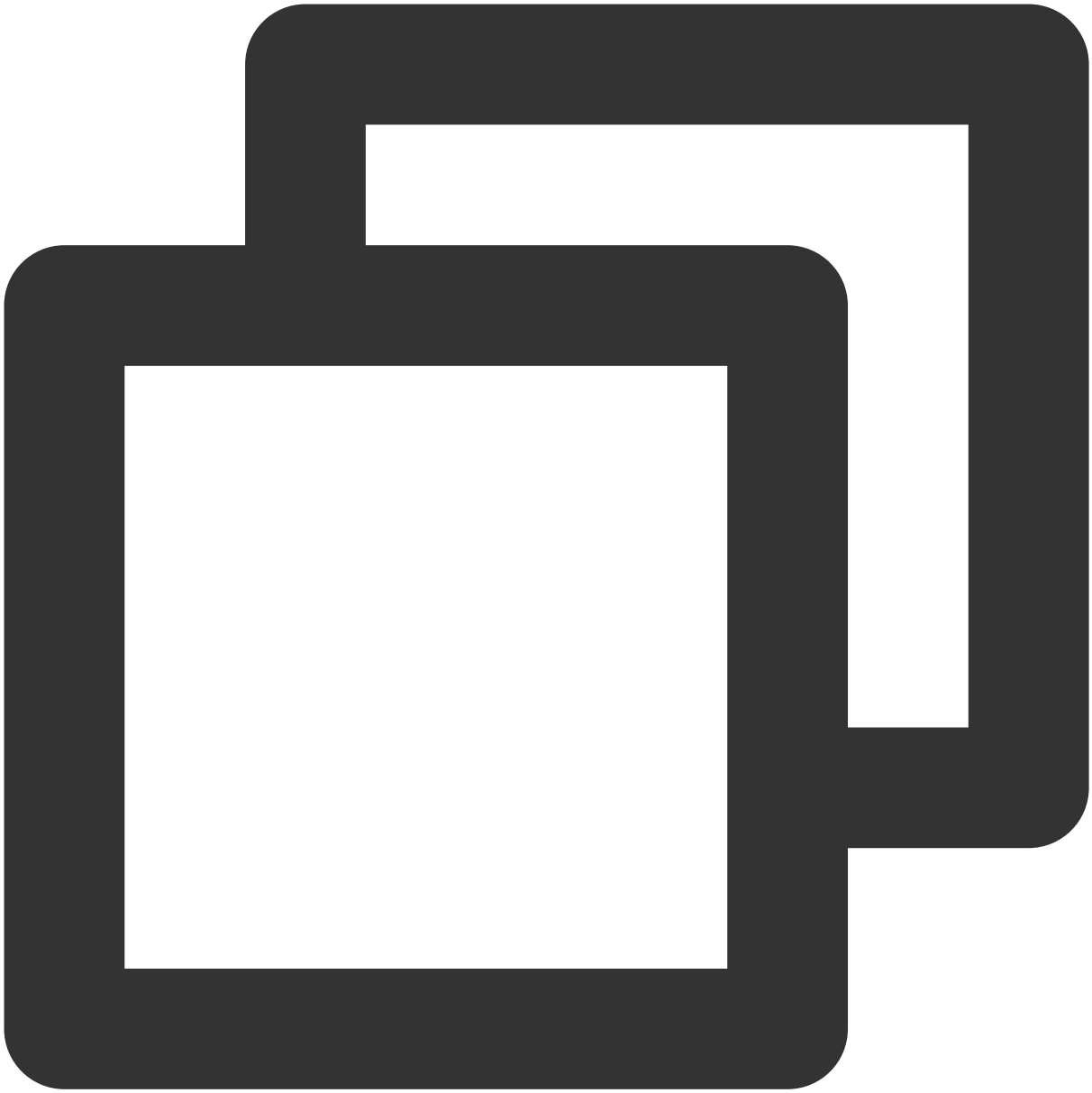
```
        ],
        "effect": "allow",
        "resource": [
          "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
        ]
      }
    ]
}
```

멀티파트 업로드 조회

버킷의 현재 멀티파트 업로드 중인 정보를 조회하며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:ListMultipartUploads로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 현재 멀티파트 업로드 중인 정보에 대한 조회 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:ListMultipartUploads"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

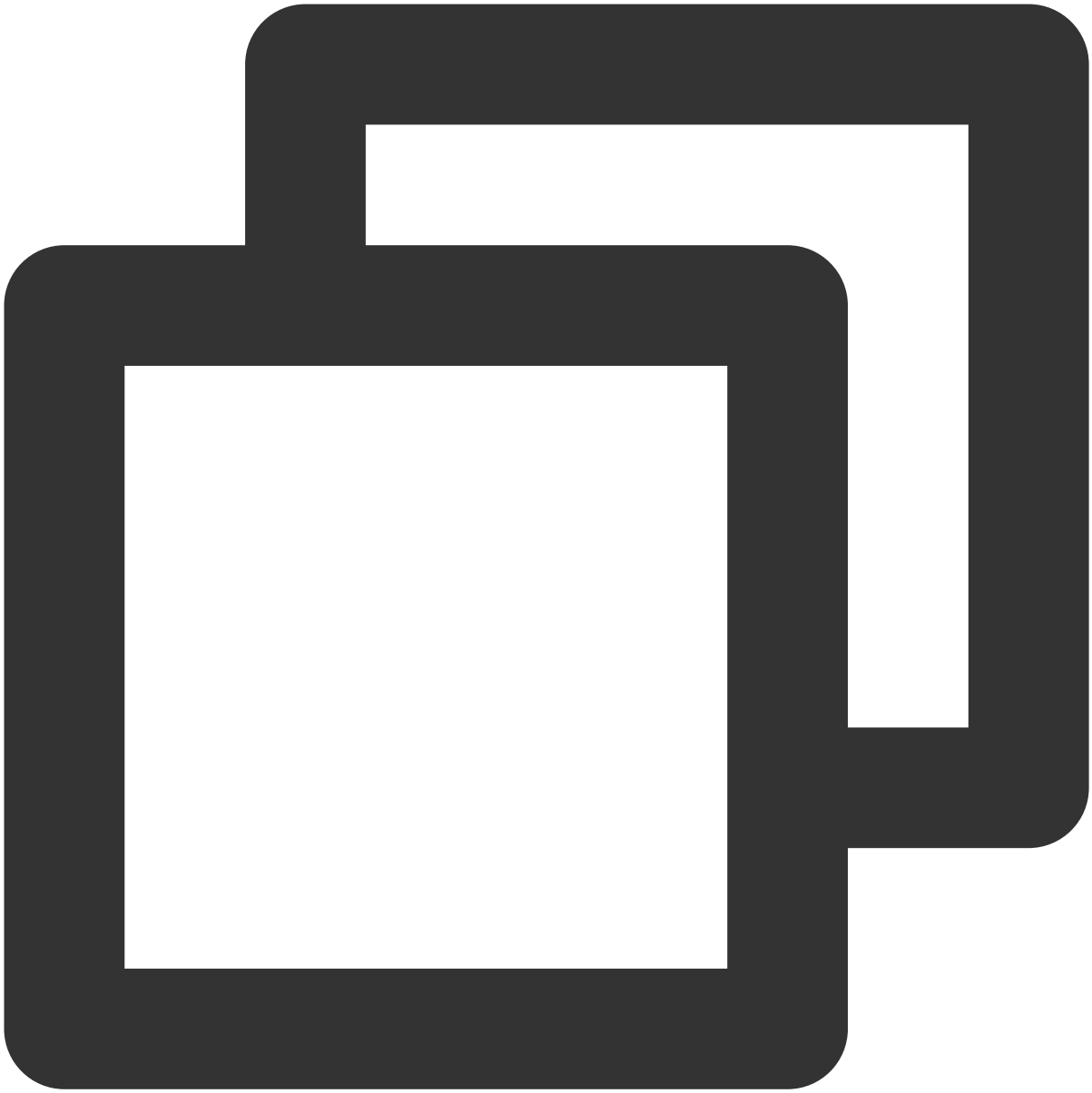
```
        }
    ]
}
```

HTML 양식을 사용한 객체 업로드

API는 POST Object이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:PostObject로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 경로 접두사 doc에서 POST 업로드 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PostObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
```

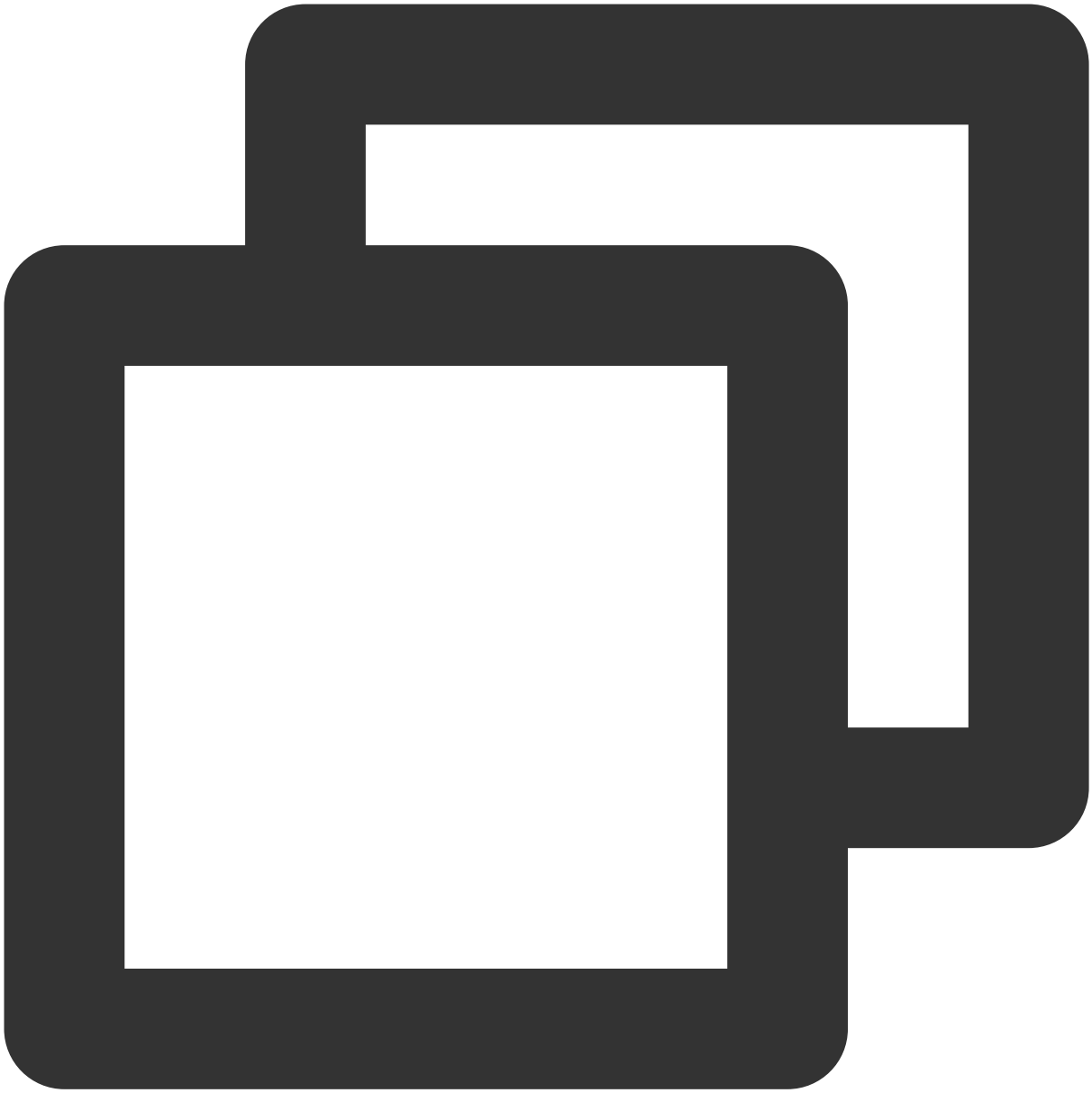
```
        }
    ]
}
```

객체 추가 업로드

API는 Append Object이며, 해당 작업 권한을 부여할 경우 정책의 action은 name/cos:AppendObject입니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 경로 접두사 doc에서 추가 업로드 작업 권한을 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
 "version": "2.0",
 "statement": [
   {
     "action": [
       "name/cos:AppendObject"
     ],
     "effect": "allow",
     "resource": [
       "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
     ]
```

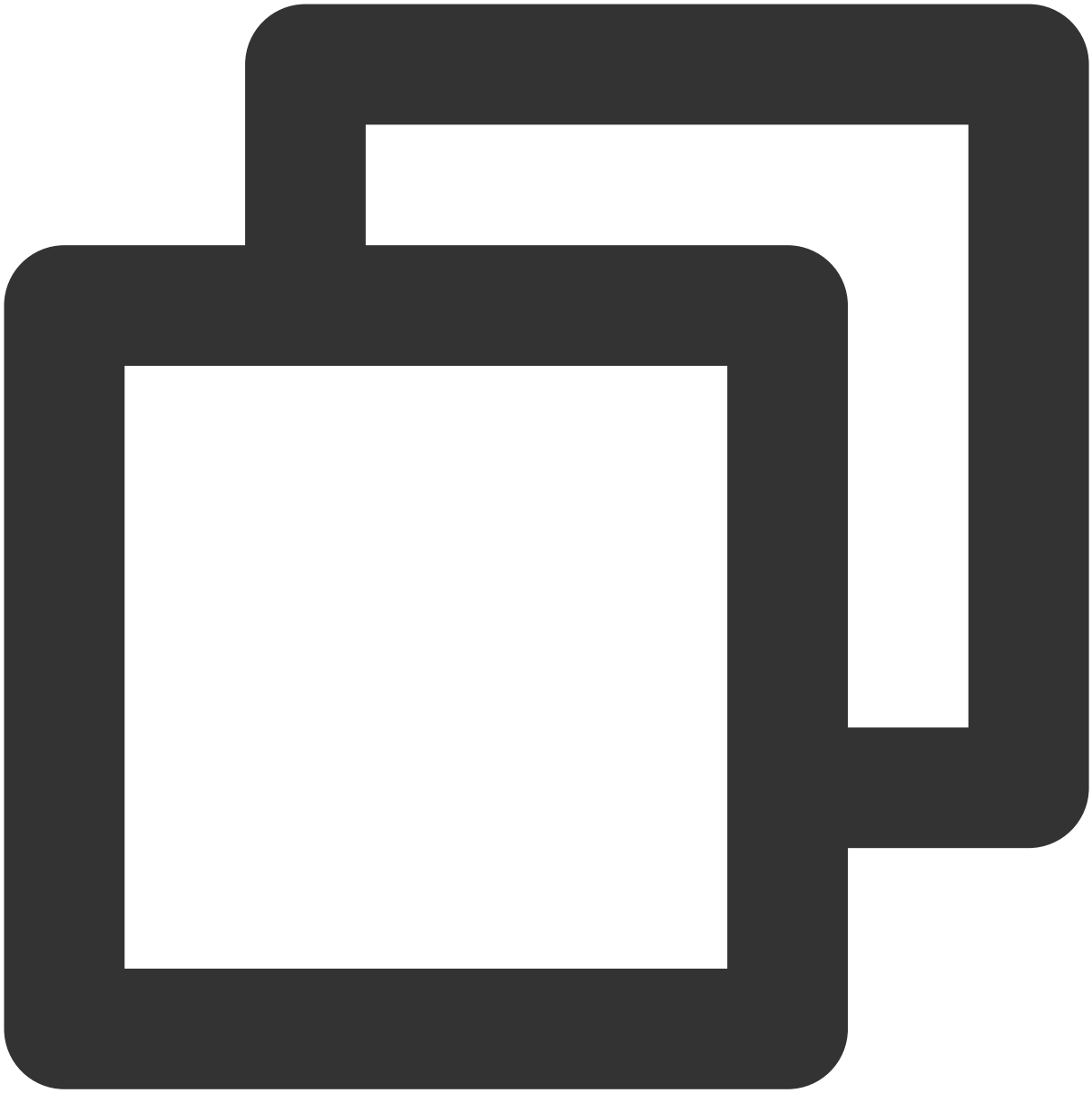
```
        }
    ]
}
```

객체 메타데이터 쿼리

API는 HEAD Object이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:HeadObject로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 경로 접두사 doc의 객체 조회 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:HeadObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
```

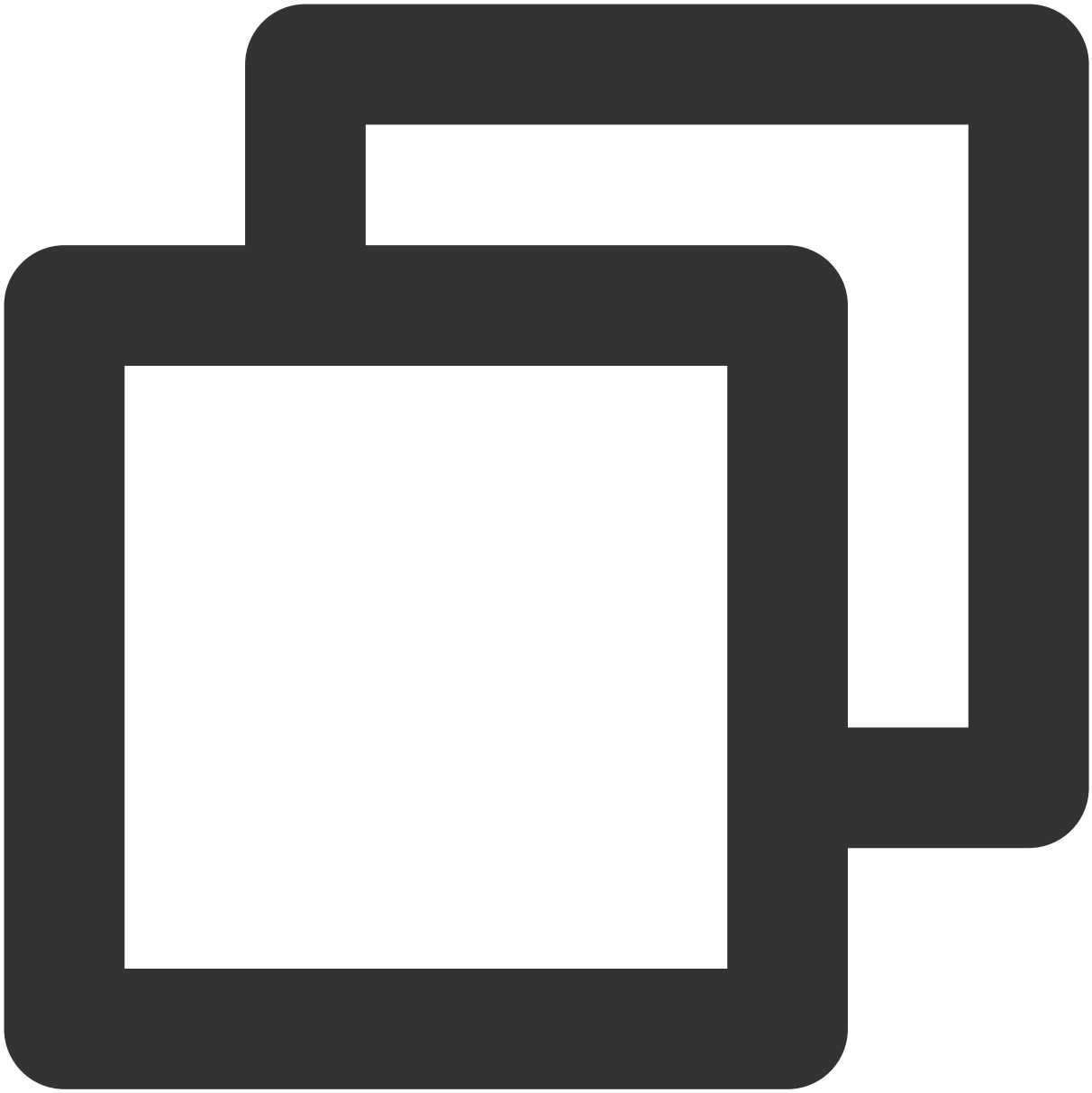
```
        }
    ]
}
```

객체 다운로드

API는 GET Object이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:GetObject로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 경로 접두사 doc의 객체 다운로드 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:GetObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
```

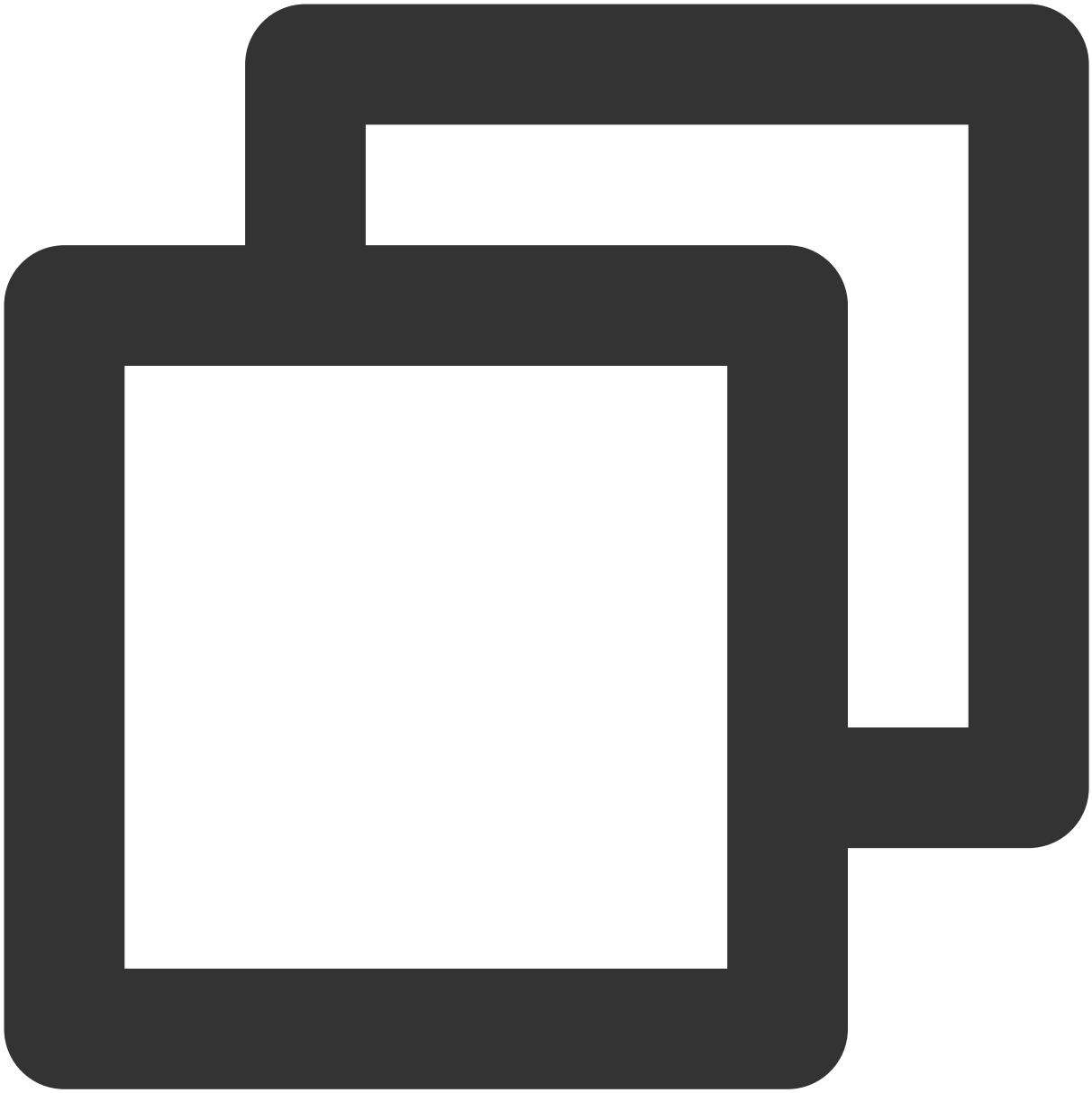
```
        }
    ]
}
```

객체 복사

API는 Put Object Copy이며, 해당 작업 권한을 부여할 경우 정책의 타깃 객체의 action을 name/cos:PutObject로 설정하고, 원본 객체의 action을 name/cos:GetObject로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 경로 접두사 doc와 경로 접두사 doc2 간의 멀티파트 복사 작업 권한을 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PutObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
```

```
      },
      {
        "action": [
          "name/cos:GetObject"
        ],
        "effect": "allow",
        "resource": [
          "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc2/*"
        ]
      }
    ]
  }
```

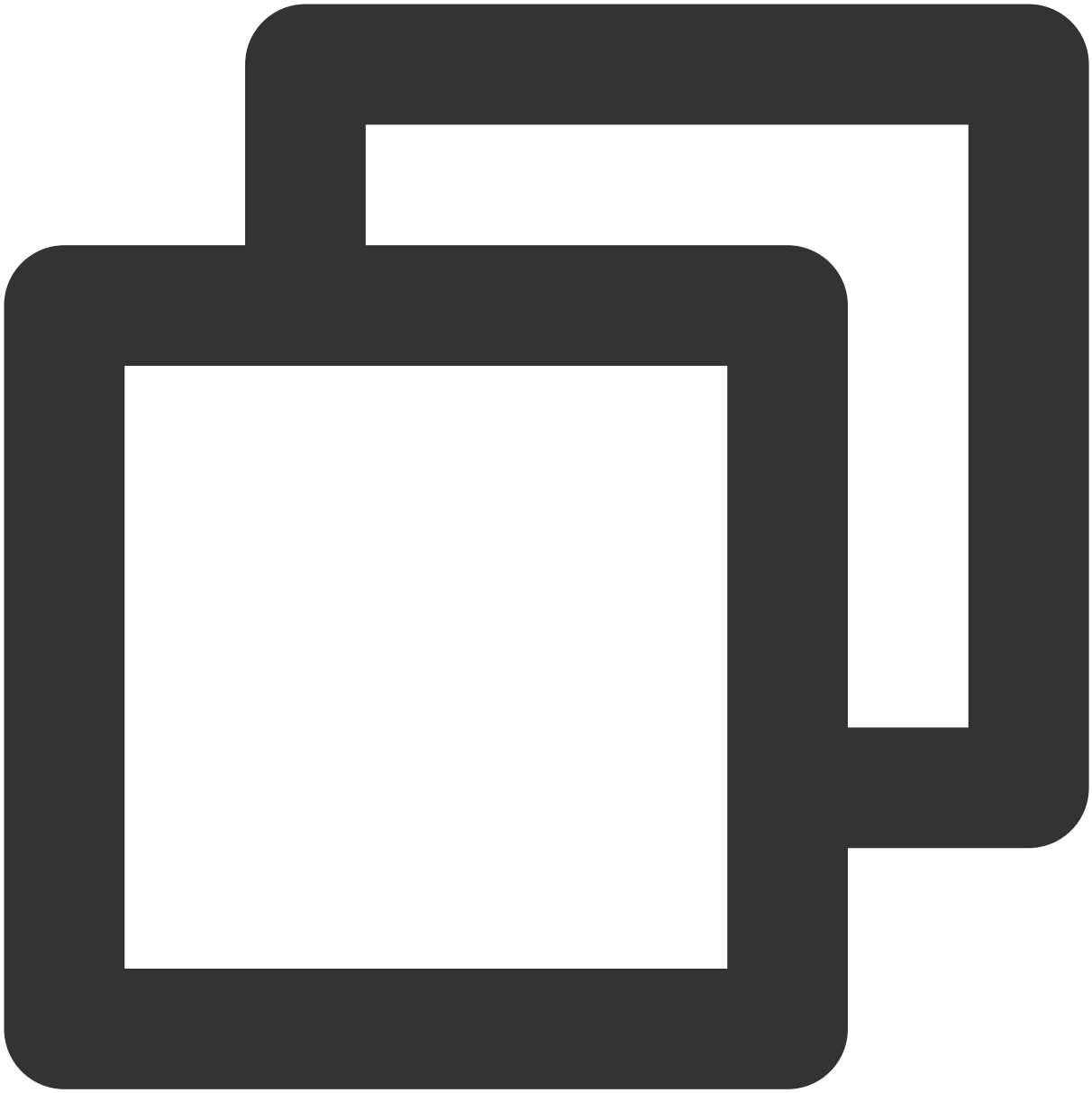
이 중에서, `"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc2/*"` 는 원본 객체입니다.

멀티파트 복사

API는 Upload Part - Copy이며, 해당 작업 권한을 부여할 경우 정책의 타깃 객체의 action은 `"name/cos:InitiateMultipartUpload","name/cos:ListMultipartUploads","name/cos:ListParts","name/cos:PutObject","name/cos:CompleteMultipartUpload","name/cos:AbortMultipartUpload"` 의 집합으로 설정하고, 원본 객체의 action은 name/cos:GetObject로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 경로 접두사 doc와 경로 접두사 doc2 간의 멀티파트 복사 작업 권한을 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:InitiateMultipartUpload",
        "name/cos:ListMultipartUploads",
        "name/cos:ListParts",
        "name/cos:PutObject",
        "name/cos:CompleteMultipartUpload",
        "name/cos:AbortMultipartUpload"
```

```
        ],
        "effect": "allow",
        "resource": [
          "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
        ]
      },
      {
        "action": [
          "name/cos:GetObject"
        ],
        "effect": "allow",
        "resource": [
          "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc2/*"
        ]
      }
    ]
}
```

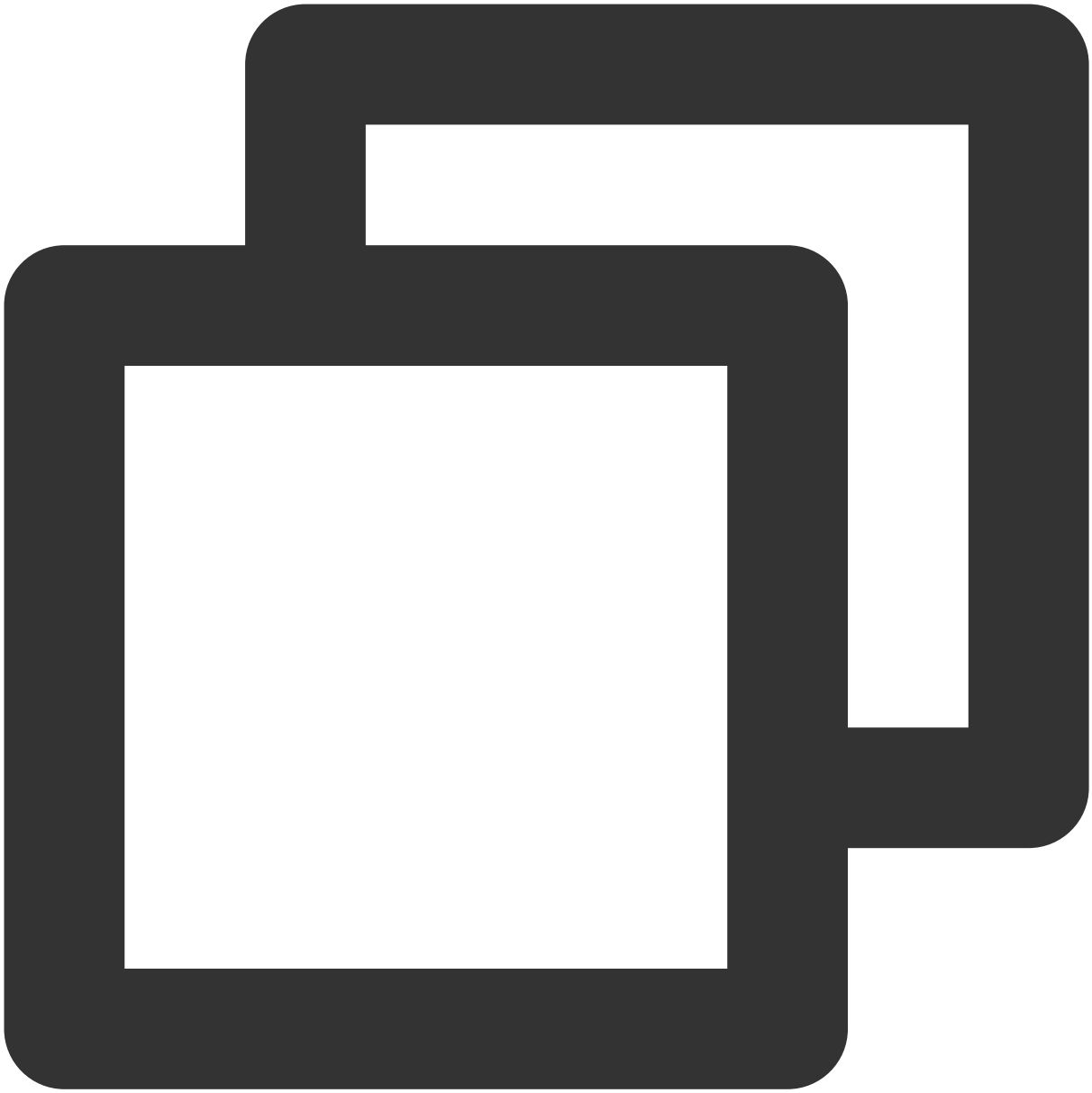
이 중에서, `"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc2/*"` 는 원본 객체입니다.

객체 ACL 설정

API는 Put Object ACL이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:PutObjectACL로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 경로 접두사 doc의 객체 ACL 설정 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PutObjectACL"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
```

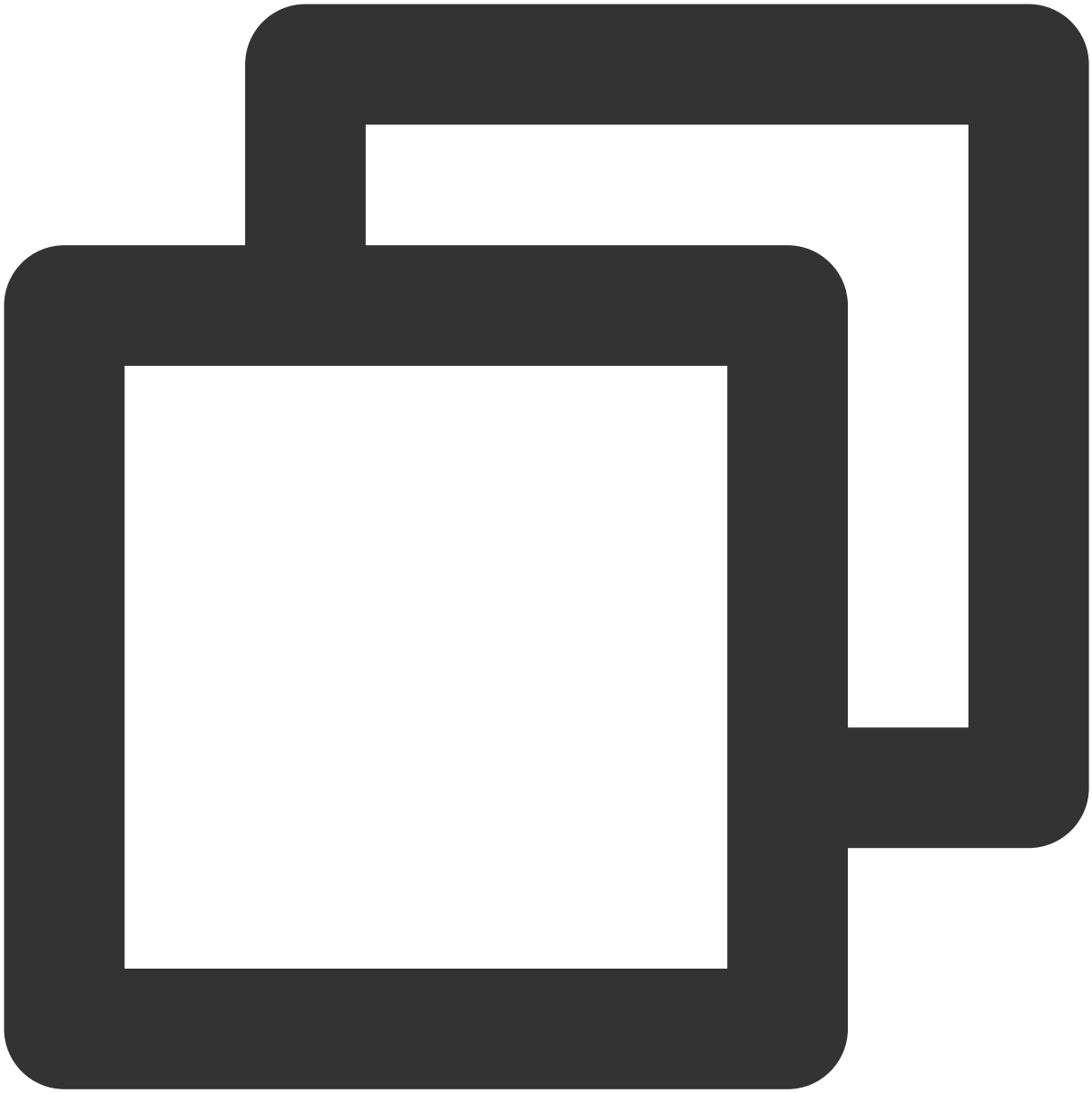
```
        }
    ]
}
```

객체 ACL 조회

API는 Get Object ACL이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:GetObjectACL로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 경로 접두사 doc의 객체 ACL 조회 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:GetObjectACL"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
```

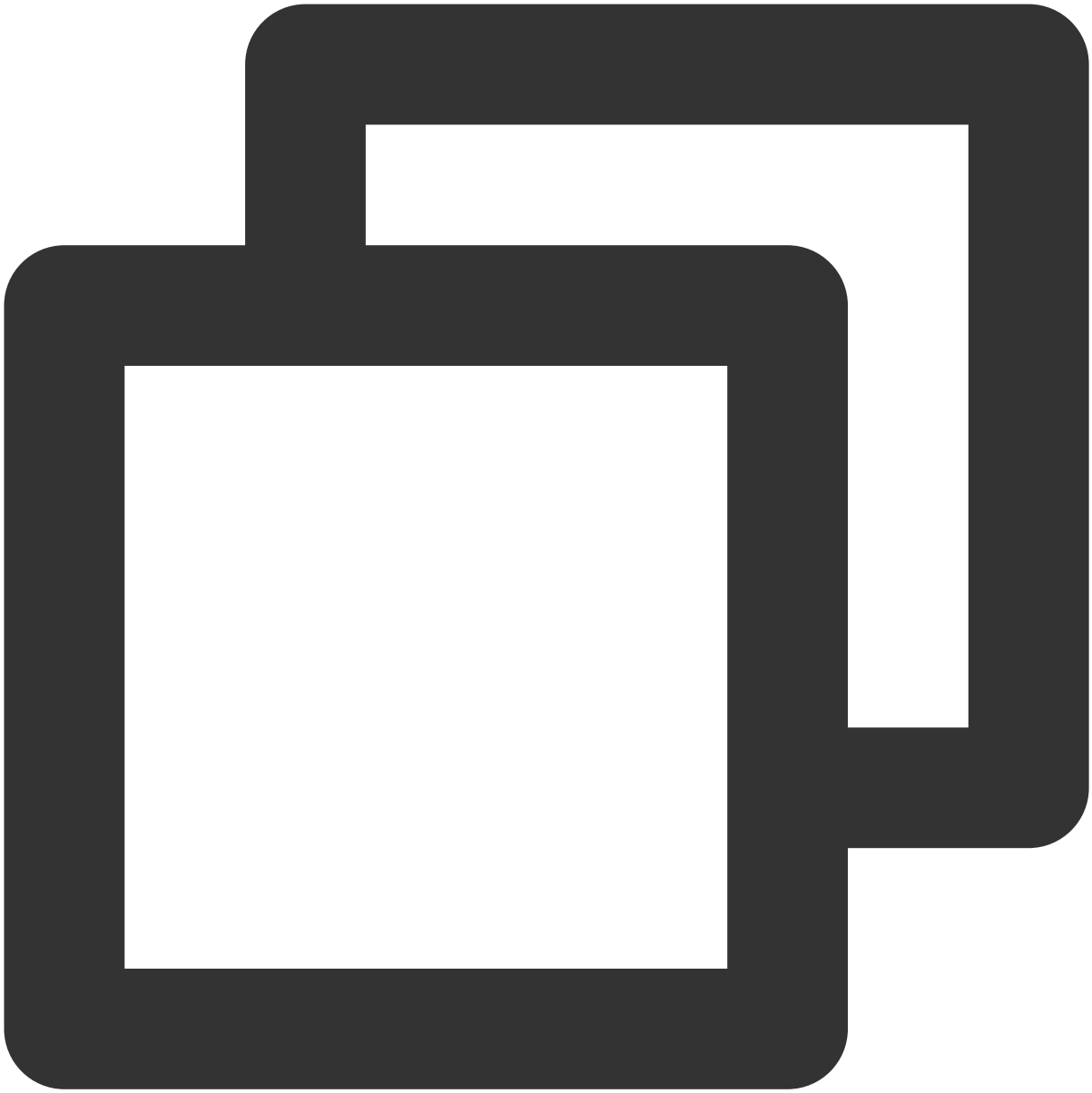
```
        }
    ]
}
```

CORS 구성 확인

API는 OPTIONS Object이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:OptionsObject로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 경로 접두사 doc에서 Options 요청 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:OptionsObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
```

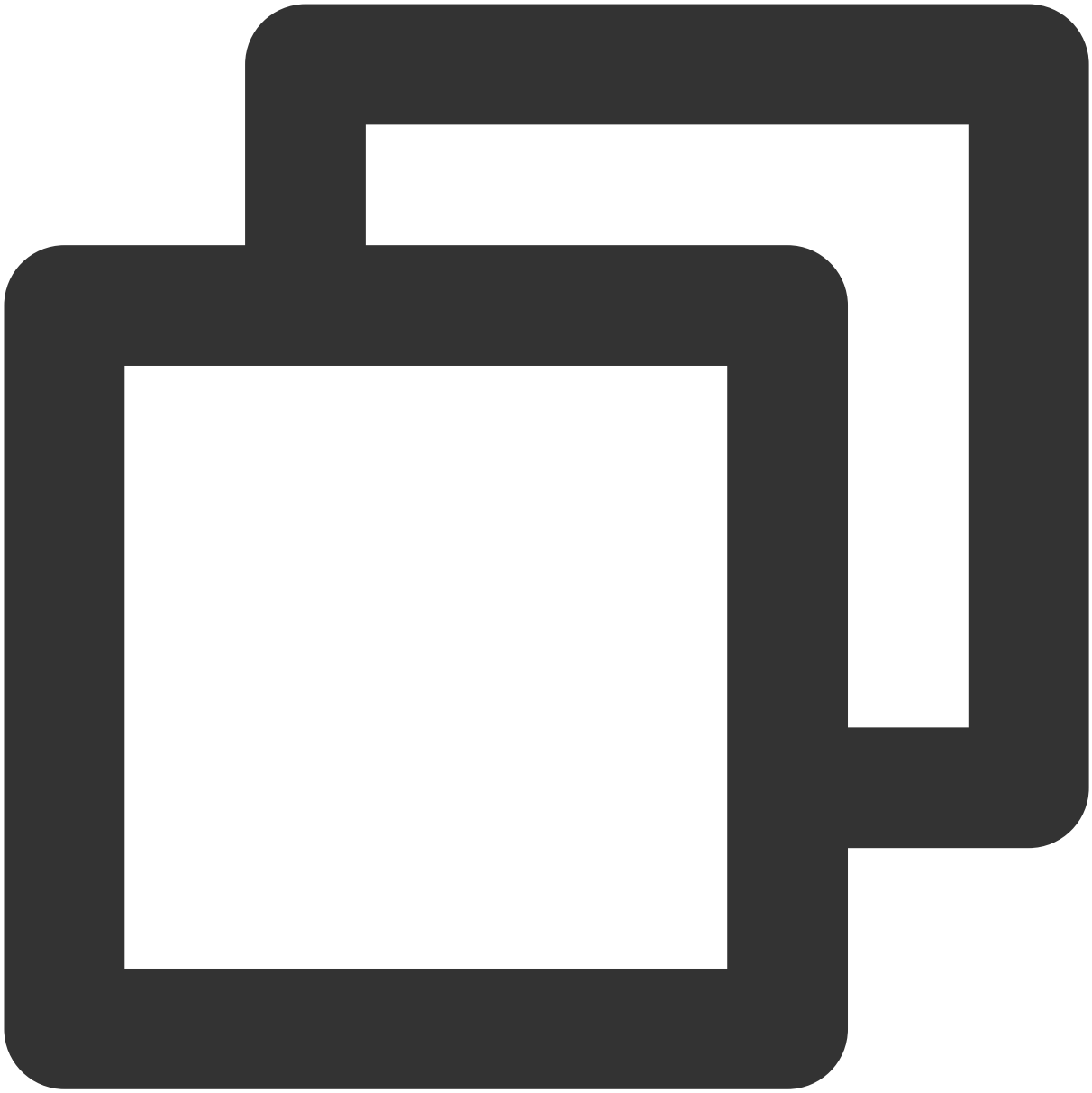
```
        }
    ]
}
```

아카이브된 객체 복구

API는 Post Object Restore이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:PostObjectRestore로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 경로 접두사 doc에서 보관 객체 복구 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PostObjectRestore"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
```

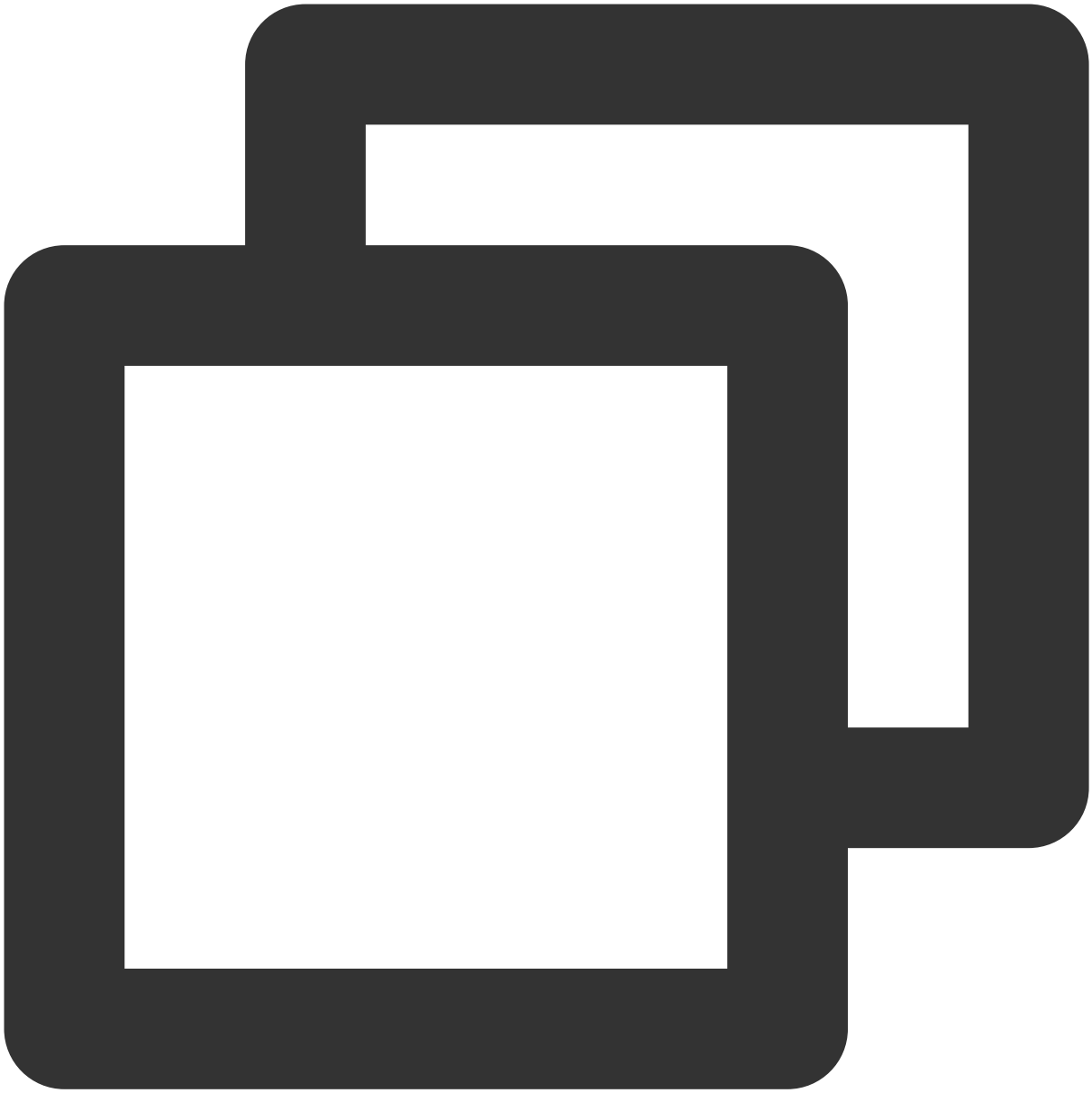
```
        }
    ]
}
```

단일 객체 삭제

API는 DELETE Object이며, 해당 작업 권한을 부여할 경우 정책의 action을 name/cos:DeleteObject로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 audio.mp3 객체에 대한 삭제 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:DeleteObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/audio.mp3"
      ]
```

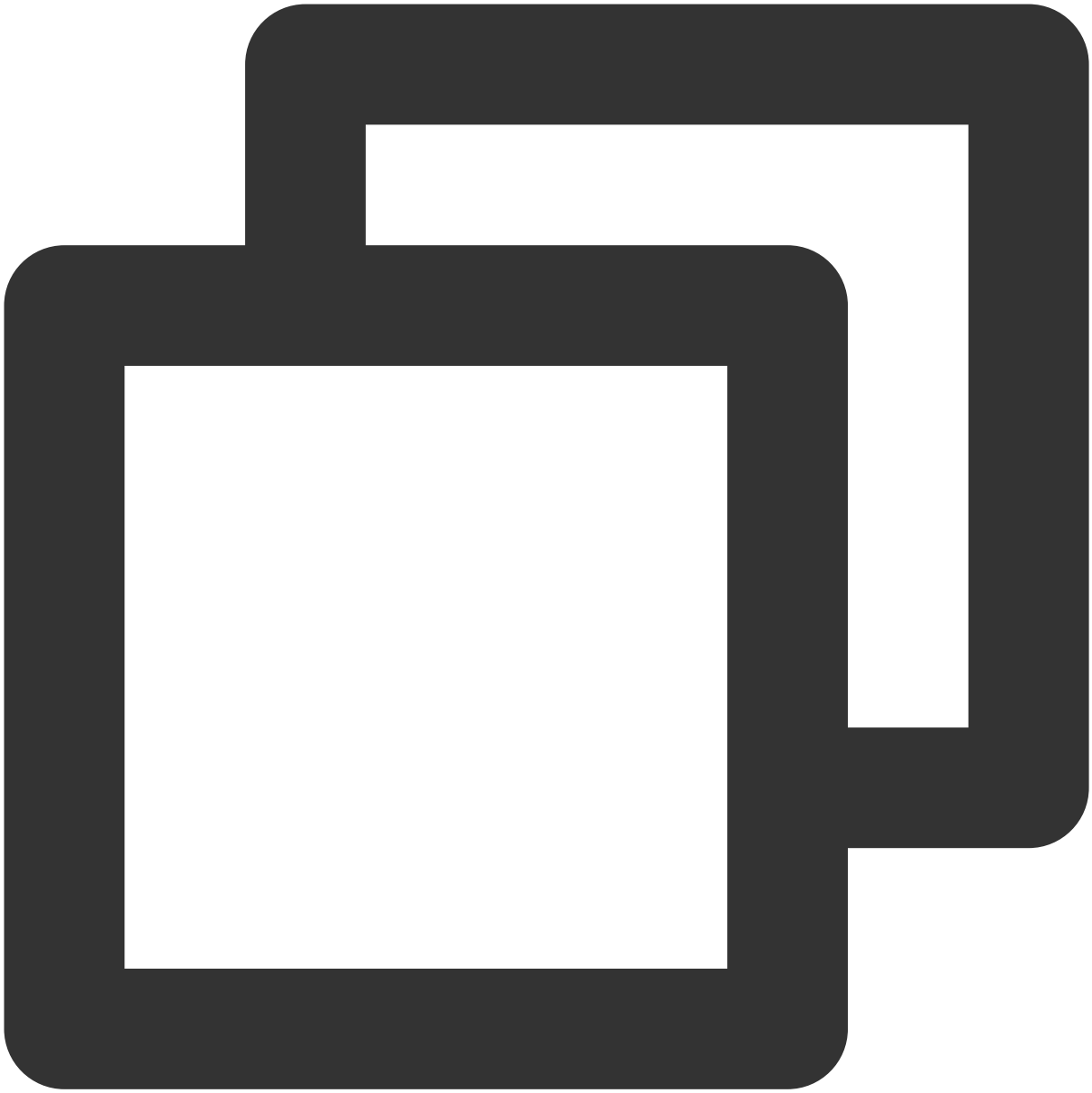
```
      }
    ]
  }
```

다수의 객체 삭제

API는 DELETE Multiple Objects이며, 해당 작업 권한을 부여할 경우 정책의 `action` 을 `name/cos:DeleteObject` 로 설정합니다.

예시

APPID가 1250000000이고 리전이 ap-beijing이며 버킷 이름이 examplebucket-1250000000인 버킷의 audio.mp3와 video.mp4 두 가지 객체에 대한 일괄 삭제 작업 권한만 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



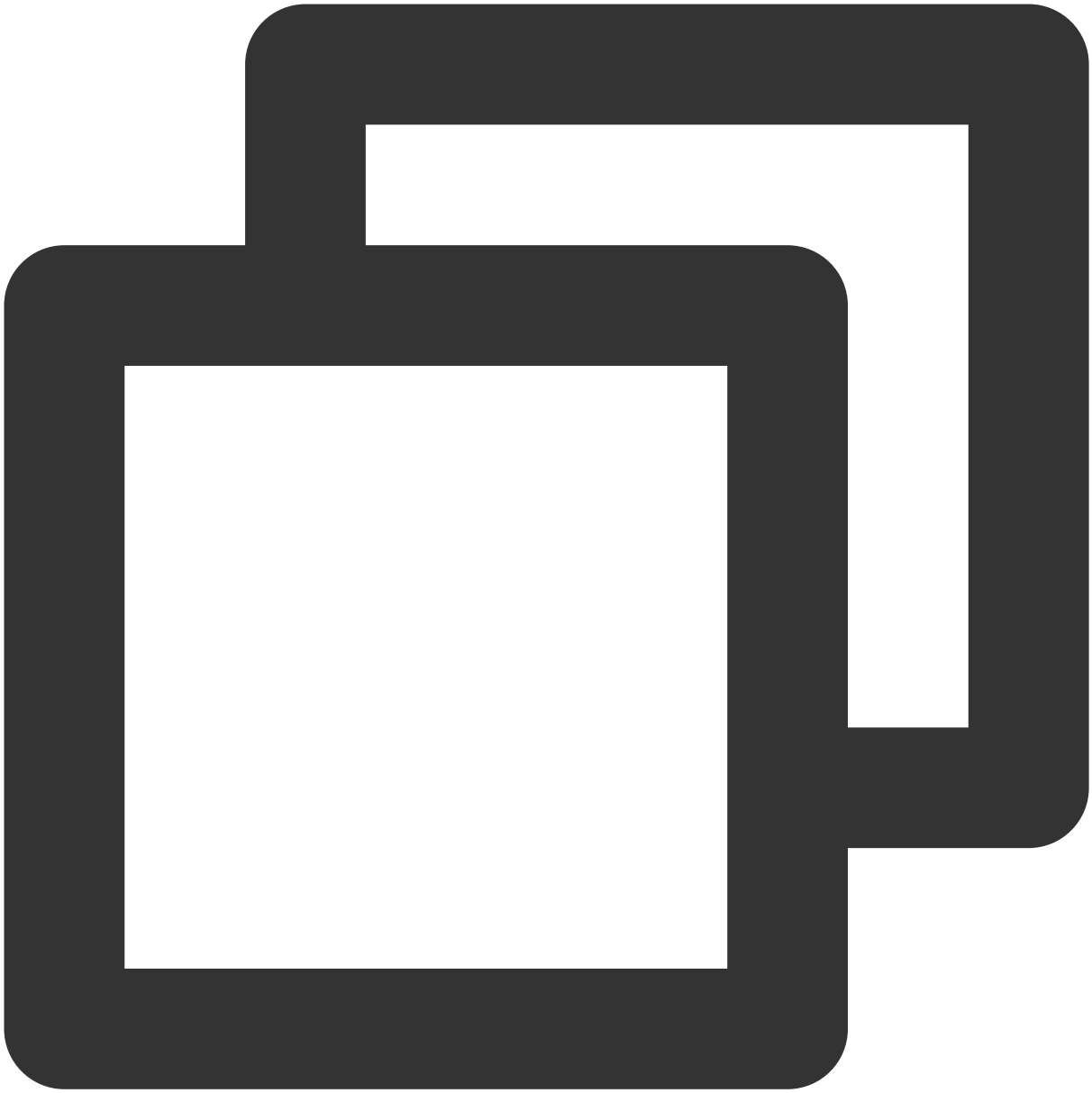
```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:DeleteObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/audio.mp3",
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/video.mp4"
```

```
        ]
      }
    ]
}
```

일반적인 시나리오 권한 부여 정책

모든 리소스에 대한 완전한 읽기 권한 부여

모든 리소스에 대한 완전한 읽기 권한을 부여하는 정책의 자세한 내용은 다음과 같습니다.

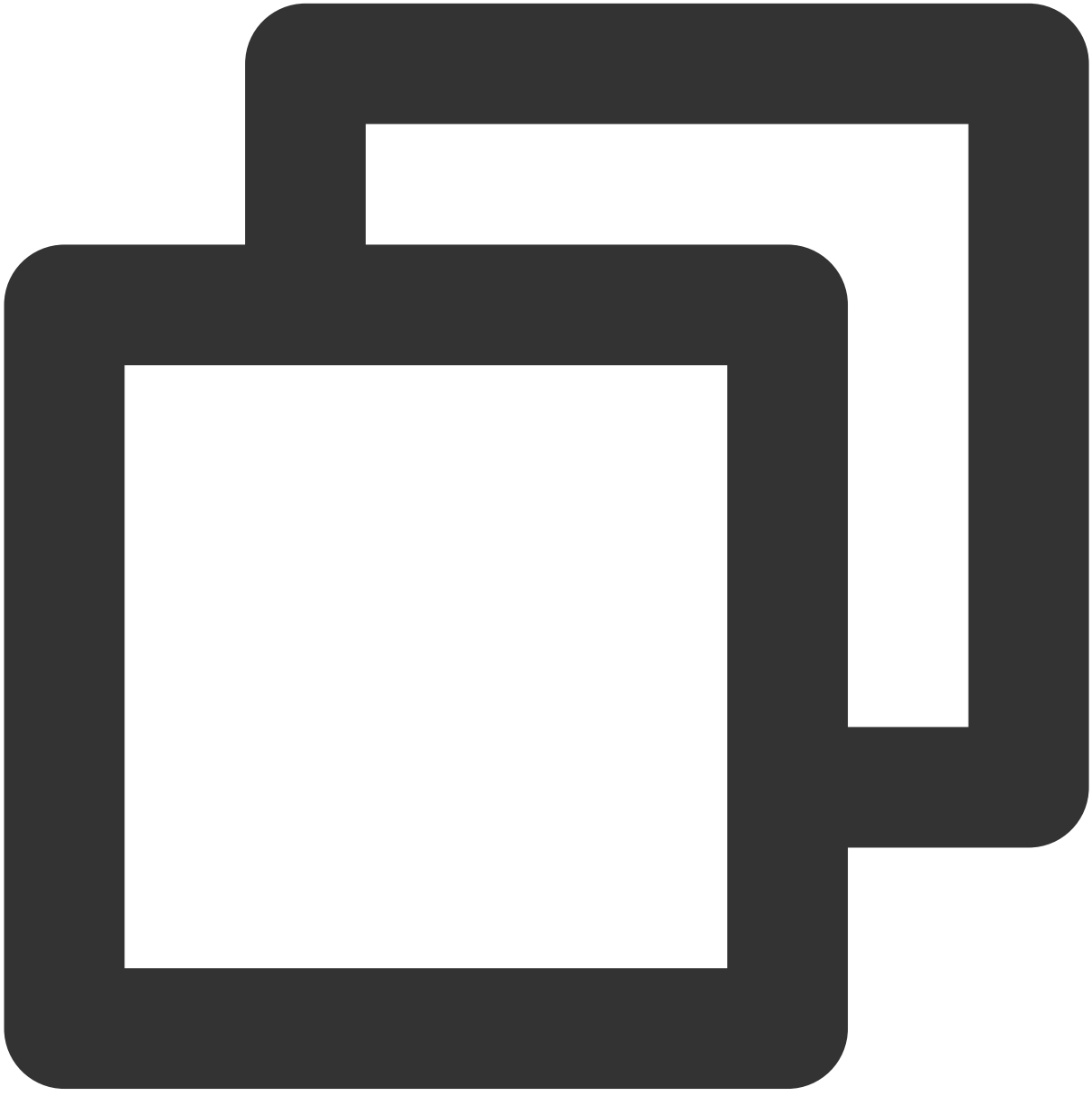


```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "*"
      ],
      "effect": "allow",
      "resource": [
        "*"
      ]
```

```
    }
  ]
}
```

모든 리소스에 대한 읽기 전용 권한 부여

모든 리소스에 대한 읽기 전용 권한을 부여하는 정책의 자세한 내용은 다음과 같습니다.

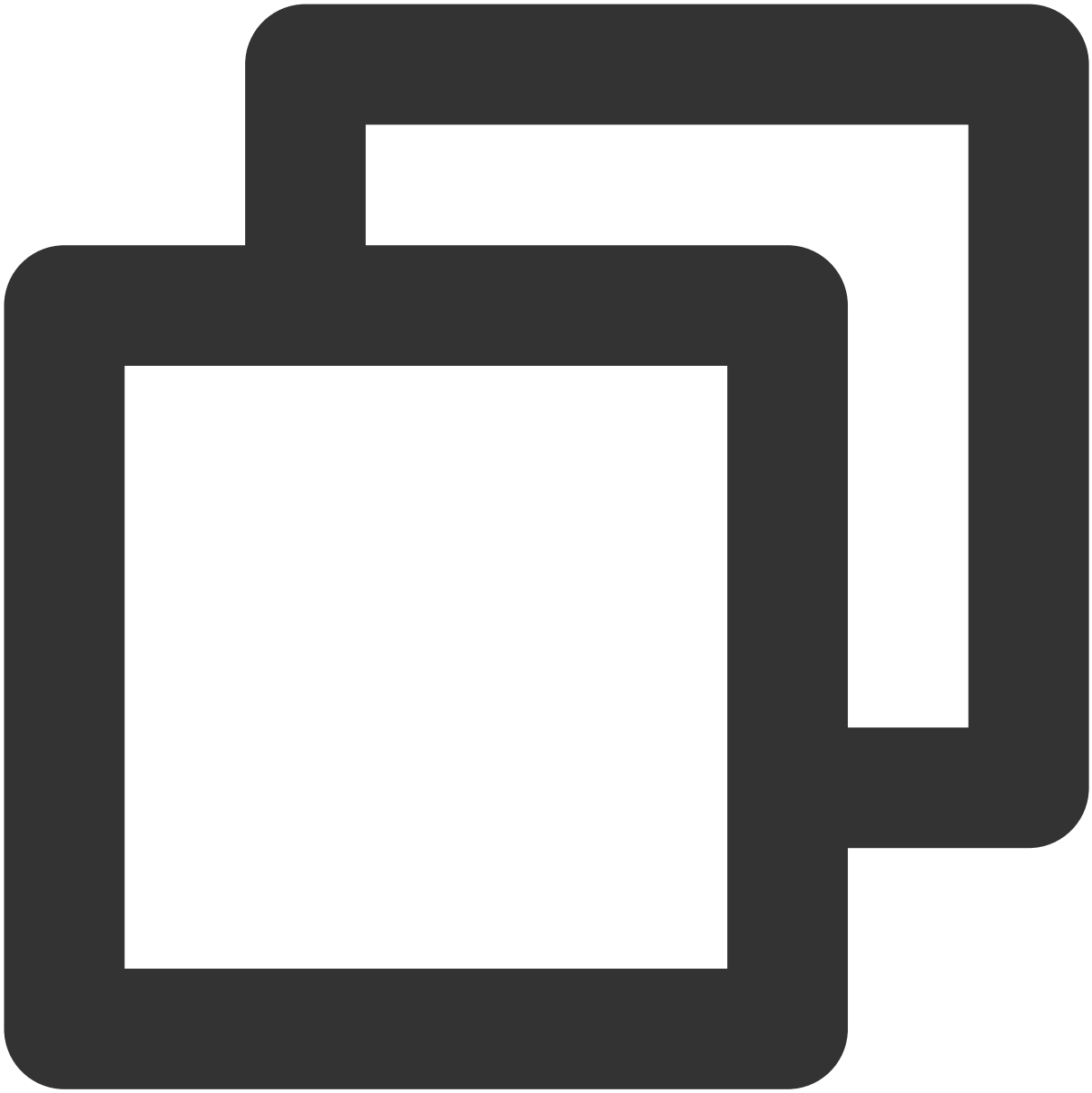


```
{
  "version": "2.0",
  "statement": [
```

```
    {
      "action": [
        "name/cos:HeadObject",
        "name/cos:GetObject",
        "name/cos:GetBucket",
        "name/cos:OptionsObject"
      ],
      "effect": "allow",
      "resource": [
        "*"
      ]
    }
  ]
}
```

특정 경로 접두사에 대한 읽기 권한 부여

사용자에게 examplebucket-1250000000 버킷의 경로 접두사 doc 아래의 파일만 액세스할 수 있고 다른 경로의 파일은 작업할 수 없도록 권한을 부여할 경우 해당 정책의 자세한 내용은 다음과 같습니다.



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "*"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-shanghai:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
```

```
        }
    ]
}
```

프런트엔드에서 COS로 다이렉트 업로드 시 임시 자격 증명 사용 보안 가이드

최종 업데이트 날짜: : 2024-06-24 16:44:04

소개

모바일 애플리케이션 및 Web에서 iOS/Android/JavaScript SDK를 통해 프런트 엔드에서 직접 Cloud Object Storage(COS)에 요청을 전달할 수 있습니다. 이 때 데이터의 업로드 및 다운로드는 사용자의 백엔드 서버를 통하지 않아 백엔드 서버 대역폭 및 부하를 절감할 수 있으며, COS 대역폭과 글로벌 가속 등의 능력을 충분히 이용할 수 있어 사용자의 애플리케이션 경험을 향상시켜 줍니다.
실제 애플리케이션에서 임시 키를 프런트 엔드의 COS 요청 서명으로 사용하여 영구 키 노출 및 권한 없는 액세스 등의 문제를 방지할 수 있습니다. 그러나 임시 키를 사용한다 하더라도 임시 키 생성 시 너무 많은 권한 또는 경로를 지정하는 경우 마찬가지로 권한 없는 액세스 등의 문제가 발생할 수 있습니다. 이는 사용자의 애플리케이션에 일정의 리스크를 야기하며, 본 문서에서는 일부 리스크 사례에 대해 중점적으로 소개합니다. 사용자는 반드시 보안 규범을 준수하여 애플리케이션에서 안전하게 COS를 사용할 수 있도록 보장하십시오.

전제 조건

본 문서에서는 사용자가 임시 키와 관련한 개념을 충분히 이해하여 임시 키를 생성 및 사용할 수 있다는 전제하에 COS에 요청하는 방법을 설명합니다. 임시 키 생성 및 사용 가이드에 대한 자세한 내용은 임시 키 생성 및 사용 가이드 문서를 참고하십시오.
주의 :
임시 키를 사용해 액세스 권한을 부여하는 경우, 반드시 업무 수요에 따라 최소 권한 부여를 원칙으로 권한을 부여해야 합니다. 직접 모든 리소스(`resource:*`) 또는 모든 작업(`action:*`)에 대한 권한을 부여하는 경우 권한 범위가 너무 커 데이터 보안에 대한 리스크가 발생할 수 있습니다.

참고 사례 및 보안 규범

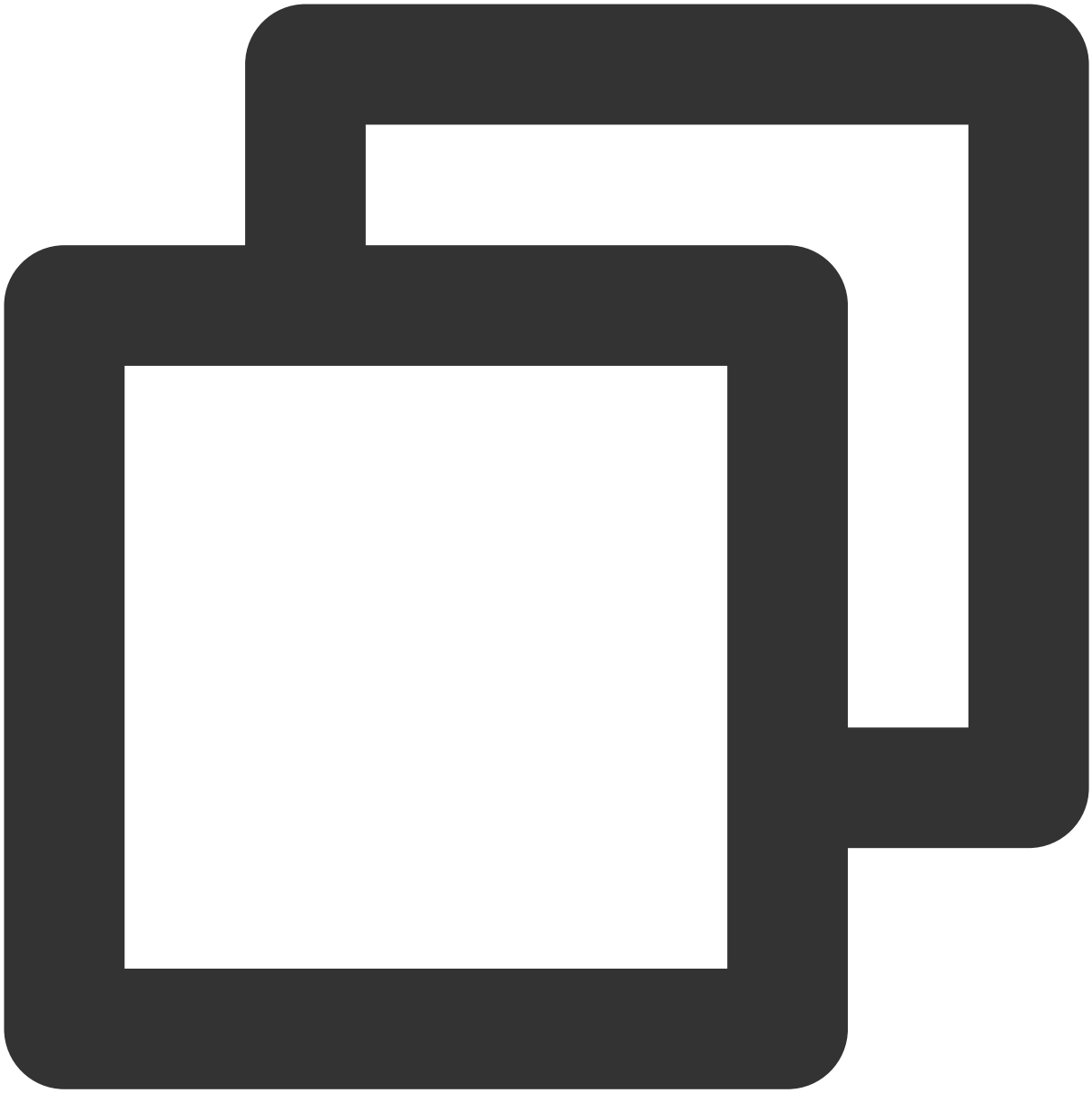
참고 사례1: 리소스(resource)의 범위 제한 초과

애플리케이션 A에서 가입 사용자가 업로드하는 프로필 사진에 COS를 사용하고 있으며, 모든 가입 사용자의 프로필 사진이 고정된 객체 키 `app/avatar/<Username>.jpg` 를 가지고 있고 동시에 서로 다른 크기의 프로필 사진이 포함되어 있습니다. 여기에서 백그라운드에서 사용하기 편리하도록 해당 객체 키를 각각 `app/avatar/<Username>_m.jpg` 와 `app/avatar/<Username>_s.jpg` 로 나누고, 임시 키 생성 시 직접

resource를 `qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/avatar/*` 로 설정했습니다. 이때 악성 사용자가 네트워크 패킷 캡쳐 등의 수단을 사용하여 생성된 임시 키를 획득하면 업로드된 임의의 사용자 프로필 사진을 덮어쓸 수 있으며 권한을 초과한 액세스가 발생해 사용자의 합법적 프로필 사진 데이터가 덮어쓰기 되어 손실될 수 있습니다.

보안 규범

resource 는 임시 키가 액세스를 허용하는 리소스 경로를 대표하며, 이때 해당 경로에서 커버하는 최종 사용자를 고려해야 합니다. 원칙적으로 resource에서 지정하는 리소스는 단일 사용자만 사용할 수 있도록 요구합니다. 해당 사례에서 지정한 `qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/avatar/*` 는 명백하게 모든 사용자를 커버하고 있어 보안 취약점이 존재하게 됩니다.
해당 사례의 경우 사용자의 프로필 사진 경로를 `app/avatar/<Username>/<size>.jpg` 로 수정하고, 이때 resource를 `qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/avatar/<Username>/*` 로 지정한다면 규범 요구를 만족할 수 있습니다. 이외에도 resource 필드는 숫자 조합 형식으로 여러 값을 전송할 수 있습니다. 따라서 명시적으로 여러 resource 값을 지정해 사용자가 액세스 권한을 가진 최종 리소스 경로를 완벽하게 한정할 수 있습니다. 예시는 다음과 같습니다.



```
"resource": [
    "qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/avatar/<Username>.jpg",
    "qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/avatar/<Username>_m.jpg",
    "qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/avatar/<Username>_s.jpg"
]
```

참고 사례2: 작업(action)의 범위 제한 초과

애플리케이션 B는 공유 포토월 기능을 제공하고 있으며, 모든 이미지는 `app/photos/*` 에 저장됩니다. 클라이언트에서는 객체 조회(GET Bucket)와 객체 다운로드(GET Object) 작업이 동시에 필요하며, 백그라운드에서는 사용 편

의를 위해 임시 키 생성 시 직접 action을 `name/cos:*` 로 지정하였습니다. 이때 악성 사용자가 네트워크 패킷 캡쳐 등의 수단을 사용하여 생성된 임시 키를 획득하면 해당 리소스 경로 하의 모든 객체에 대해 모든 객체 작업(예: 업로드 및 삭제 등의 작업)을 실행할 수 있으며, 권한을 초과한 액세스가 발생해 데이터가 손실되고 온라인 비즈니스에 영향을 미칠 수 있습니다.

보안 규범

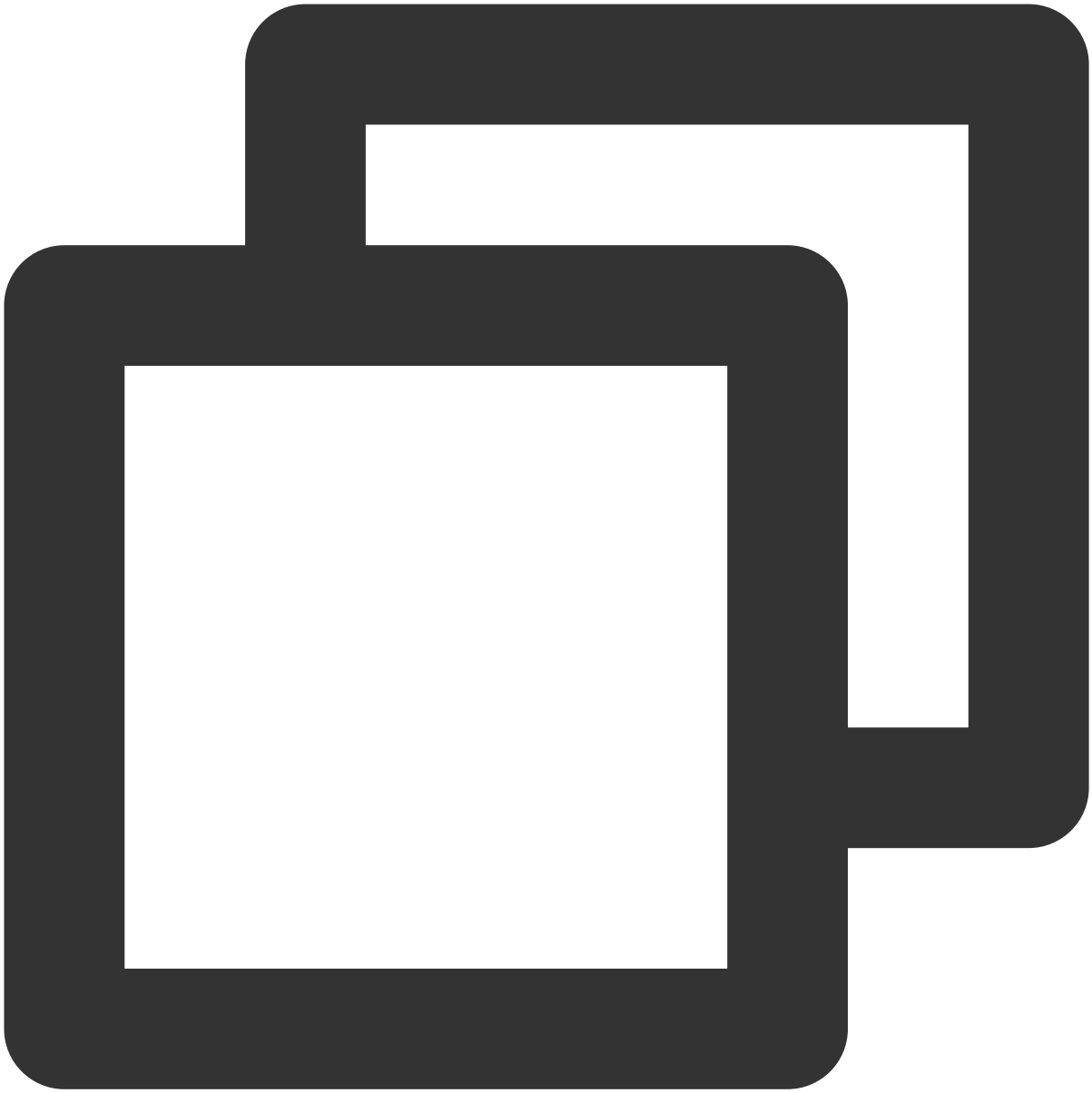
action은 임시 키가 요청을 허용하는 작업을 대표하며, 원칙적으로 `name/cos:*` 등을 사용한 모든 작업의 임시 키를 프런트 엔드에 전달하는 행위를 허용하지 않습니다. 반드시 모든 필요한 작업을 명확하게 나열해야 하며, 각 작업에 필요한 리소스 경로가 서로 다른 경우 병합 처리가 아닌 **작업**과 **리소스** 경로를 단독으로 매칭해야 합니다.
해당 사례에서는 `"action": [ "name/cos:GetBucket", "name/cos:GetObject" ]` 를 사용하여 구체적인 작업을 명확하게 지정해야 합니다. 권한 부여 작업 가이드에 대한 사항은 COS API 권한 부여 정책 사용 가이드를 참고하십시오.

참고 사례3: 리소스와 작업의 범위 제한 초과

애플리케이션 C는 관리 툴을 제공하고 있으며, 사용자에게 모든 사용자들의 파일(`app/files/*`)을 조회하고 다운로드할 수 있도록 허용하고, 업로드 및 삭제는 개인 디렉터리의 파일(`app/files/<Username>/*`)만 작업할 수 있도록 제공합니다. 백그라운드에서의 사용 편의를 위해 임시 키 생성 시 해당 두 권한에 대해 4가지 작업(action)을 함께 혼합하여 사용합니다. 두 권한의 해당 리소스 경로 또한 혼합하여 함께 사용하며, 이때의 임시 키는 리소스 경로에서 지정한 권한보다 더 큰 권한을 가지게 됩니다. 즉, 모든 사용자들의 파일에 대한 조회, 다운로드, 업로드, 삭제 권한을 가지게 됩니다. 악성 사용자가 이에 따라 타인의 파일을 조작하거나 삭제할 수 있으며 권한을 초과한 액세스가 발생하여 사용자의 합법적 데이터가 유출될 리스크가 있습니다.

보안 규범

여러 action과 resource의 조합은 간단하게 이들을 각각 병합하는 방법을 사용해서는 안 되며, 여러 statement 형식으로 조합해야 합니다. 간단하게 각각 병합하는 경우 권한이 커질 수 있습니다.
해당 사례에서는 반드시 아래와 같이 사용해야 합니다.



```
"statement": [
    {
        "effect": "allow",
        "action": [
            "name/cos:GetBucket",
            "name/cos:GetObject"
        ],
        "resource": "qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/files/*"
    },
    {
        "effect": "allow",
```

```
            "action": [
                "name/cos:PutObject",
                "name/cos:DeleteObject"
            ],
            "resource": "qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/files/<Us
        }
    ]
```

참고 사례4: 권한을 초과한 임시 키 획득

애플리케이션 D는 포럼 애플리케이션을 제공하고 있으며, 포럼의 포스팅 첨부파일이 COS에 저장됩니다. 해당 포럼 애플리케이션은 각 사용자 레벨별로 포스팅 수가 일정 임계값에 도달하여 활성화된 사용자만 특정 페이지 내의 포스팅과 첨부파일을 볼 수 있도록 제공하고, 일부 공개 페이지는 모든 사용자가 해당 페이지 내의 포스팅과 첨부 파일을 볼 수 있습니다.
COS 사용 시, 백그라운드의 임시 키 생성 인터페이스는 프런트 엔드가 전송하는 페이지 ID에 따라 해당 페이지의 첨부파일에 대한 다운로드를 허용하는 임시 키를 생성합니다. 그러나 실현 과정에서 백그라운드에서 요청 사용자가 지정 페이지 ID 액세스 권한을 보유하고 있는지 여부를 판단할 수 없어 모든 사람이 해당 인터페이스를 요청하면 개인 페이지의 첨부파일에 대한 액세스 임시 키를 획득할 수 있게 됩니다. 이때 권한을 초과하는 액세스가 발생해 액세스를 제한하는 리소스에 대해 효과적으로 제한할 수 없게 되며 의도치 않은 데이터 유출 문제가 발생할 수 있습니다.

보안 규범

임시 키를 획득하는 인터페이스는 획득하는 임시 키 자체의 권한이 사전 설정된 범위 내여야 하며, 권한이 낮은 사용자가 높은 권한의 임시 키를 획득하지 못하도록 실제 업무 시나리오, 즉 권한이 정확한 사용자에 의해 요청되었는지 여부를 정확하게 판단해야 합니다.
본 사례에서는 백그라운드 인터페이스에서 반드시 현재 요청하는 사용자가 특정 페이지에 액세스할 수 있는 권한이 있는지 여부를 판단하여 액세스 권한이 없는 사용자에게 임시 키 반환을 허용하지 않아야 합니다.

결론

이상의 사례로 예상 외로 임시 키의 권한이 확대되어 발생할 수 있는 보안 리스크를 설명하였습니다. 프런트 엔드에서 직접 COS에 전송하는 경우, 악성 사용자가 비교적 쉽게 임시 키 내용을 획득할 수 있습니다. 따라서 개발자는 해당 시나리오의 경우 백그라운드를 통한 COS 액세스에 비해 권한 제어에 대해 더욱 유의해야 합니다.
본 문서에서의 보안 규범은 최소한의 권한 원칙으로, 실제 사용에서는 action과 resource에 따라 모든 가능한 권한을 나열할 수 있습니다. 예를 들어 action 3개, resource 2개가 있다면 액세스가 허용되는 리소스와 해당 작업을 3x2=6개로 산출할 수 있으며, 이에 따라 각 상황별로 예상에 부합하는지 여부를 평가하여 예상 권한 범위를 초과하는 경우 여러 statement를 나열하는 방식으로 권한을 분할하는 방법을 고려해야 합니다.
이외에도 임시 키 생성에 사용하는 인터페이스는 자체적으로 자격 인증과 인증 처리를 충분히 고려해야 합니다. 임시 키를 획득하는 행위가 안전하다면, 해당 방법으로 획득한 임시 키에 대해 진정한 보안이 실현되는 것입니다. 보안이라는 연결 고리에는 어떠한 소홀함도 있어서는 안 됩니다!

임시 키 생성 및 사용 가이드

최종 업데이트 날짜: : 2024-06-24 16:44:04

주의 :
임시 키를 사용해 액세스 권한을 부여하는 경우, 반드시 업무 수요에 따라 최소 권한 부여를 원칙으로 권한을 부여해야 합니다. 직접 모든 리소스 `(resource:*)` 또는 모든 작업 `(action:*)` 에 대한 권한을 부여하는 경우 권한 범위가 너무 커 데이터 보안에 대한 리스크가 발생할 수 있습니다.
임시 키 신청 시 권한 범위를 지정하면 신청한 임시 키가 권한 범위 내에서만 동작할 수 있습니다. 예를 들어 임시 키를 신청할 때 examplebucket-1-1250000000 버킷에 파일을 업로드할 수 있는 권한 범위를 지정한 다음 적용된 키는 examplebucket-2-1250000000에 파일을 업로드**할 수 없습니다**. 또한 examplebucket-1-1250000000에서 파일을 다운로드**할 수 없습니다**.

임시 키

임시 키(임시 액세스 자격 증명)는 CAM 클라우드 API에서 제공하는 인터페이스를 통해 획득하는 권한이 부여된 키입니다.
COS API는 임시 키를 사용해 서명을 컴퓨팅해 COS API 요청을 전송하는 데 사용합니다.
COS API 요청은 임시 키를 사용해 서명 컴퓨팅 시, 다음 3개의 임시 키 인터페이스 반환 정보 획득에 사용하는 필드가 필요합니다.
TmpSecretId
TmpSecretKey
Token

임시 키 사용의 장점

Web, iOS, Android에서 COS를 사용할 때, 고정 키를 사용해 서명을 컴퓨팅하는 경우 권한을 효과적으로 제어할 수 없을 뿐 아니라, 영구 키를 클라이언트 코드에 삽입하는 경우 커다란 유출 위험이 존재하게 됩니다. 임시 키 방식을 사용하면 권한 제어 문제를 편리하고 효과적으로 해결할 수 있습니다.
예를 들어, 임시 키 신청 과정에서 권한 정책 policy 필드 설정을 통해 작업 및 리소스를 제한하고, 권한을 지정된 범위로 제한할 수 있습니다.
COS API 권한 부여 정책과 관련해서는 다음을 참고하십시오.
COS API 임시 키를 사용한 권한 부여 정책 가이드
자주 볼 수 있는 시나리오에서의 임시 키를 사용한 권한 부여 정책 예시

임시 키 획득

임시 키는 제공되는 COS STS SDK 방식을 통해 획득할 수 있으며, 직접 STS 클라우드 API 요청 방식으로도 획득할 수 있습니다.

주의 :

예시에서는 Java SDK를 사용하였으며, GitHub에서 SDK 코드(버전)을 획득해야 합니다. 해당 SDK 버전을 찾을 수 없다는 안내가 표시되는 경우, GitHub에서 상응하는 버전의 SDK를 획득하였는지 확인하십시오.

COS STS SDK

COS는 STS에 대한 SDK와 샘플을 제공합니다. 현재 Java, Nodejs, PHP, Python, Go 등 여러 가지 언어 샘플을 제공하고 있으며, 자세한 내용은 COS STS SDK를 참고하십시오. 각 SDK별 사용 설명은 Github의 README와 예시를 참고하십시오. 각 언어의 GitHub 주소는 하기 표와 같습니다.

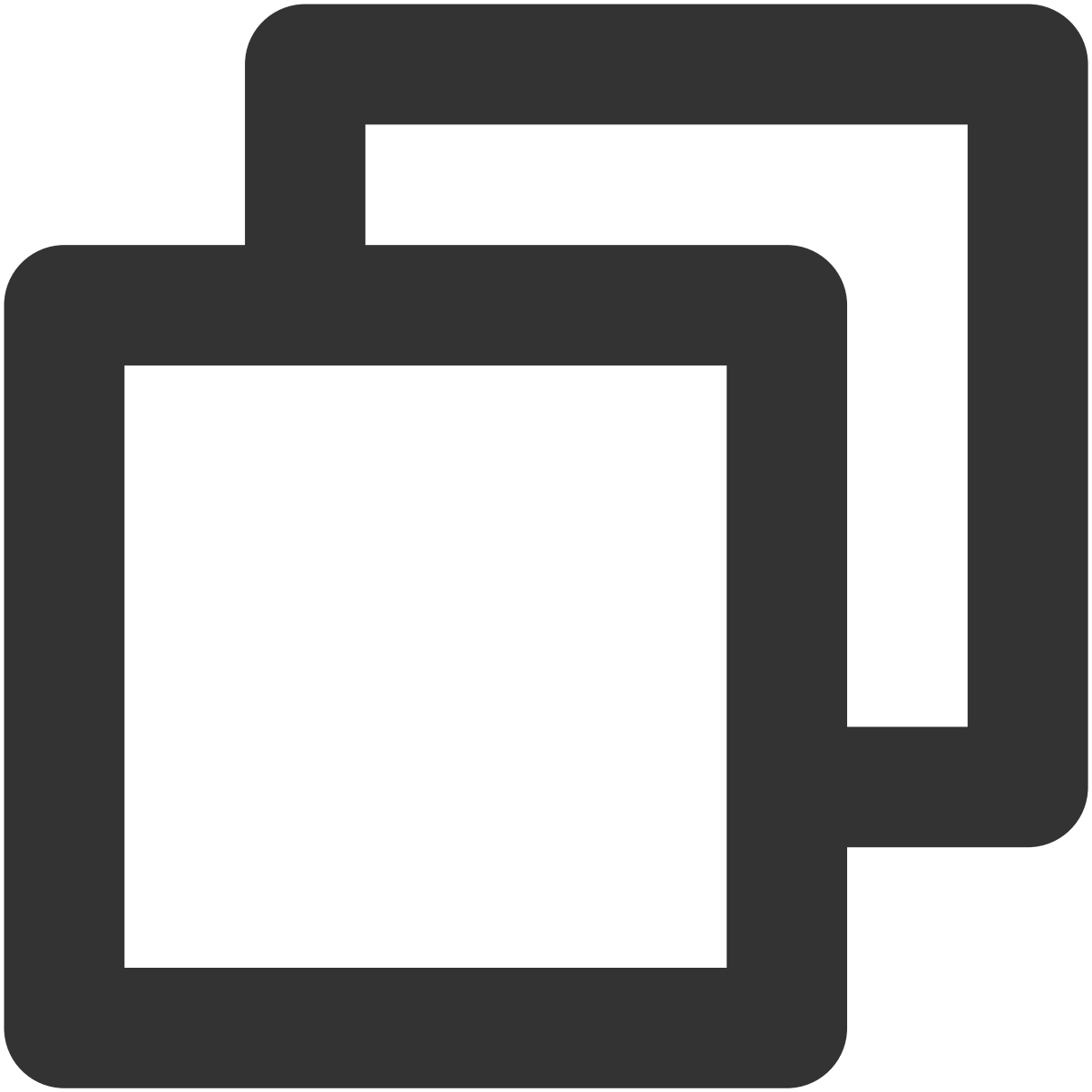
언어 유형	코드 설치 주소	코드 예시 주소
Java	설치 주소	예시 주소
.NET	설치 주소	예시 주소
Go	설치 주소	예시 주소
NodeJS	설치 주소	예시 주소
PHP	설치 주소	예시 주소
Python	설치 주소	예시 주소

주의 :

STS SDK는 STS 인터페이스 자체의 버전별 차이점을 차단하기 위해 반환 매개변수 구조가 STS 인터페이스와 완벽하게 일치하지 않습니다. 세부 사항은 Java SDK 문서를 참고하십시오.

Java SDK를 사용한다고 가정하고, 먼저 Java SDK를 다운로드한 후 실행하여 다음 예시와 같이 임시 키를 획득합니다.

코드 예시



```
// github에서 제공하는 maven 통합 방법에 따라 java sts sdk 가져오기. 3.1.0 이상 버전 사용
public class Demo {
    public static void main(String[] args) {
        TreeMap<String, Object> config = new TreeMap<String, Object>();

        try {
            //여기에서 SecretId와 SecretKey는 임시 키를 신청하기 위한 영구 ID(루트 계정, 서
            String secretId = System.getenv("secretId");//사용자 SecretId. 리스크를 줄
             String secretKey = System.getenv("secretKey");//사용자 SecretKey. 리스크
            // Tencent Cloud api 키 SecretId로 교체
            config.put("secretId", secretId);
```

```
        // Tencent Cloud api 키 SecretKey로 교체
        config.put("secretKey", secretKey);

        // 도메인 설정:
        // Tencent Cloud cvm을 사용하는 경우 내부 도메인 이름 설정 가능
        //config.put("host", "sts.internal.tencentcloudapi.com");

        // 임시 키의 유효 시간은 초 단위입니다. 기본값은 1800 초이며, 현재 루트 계정에서
        config.put("durationSeconds", 1800);

        // 사용자의 bucket으로 변경
        config.put("bucket", "examplebucket-1250000000");
        // bucket 소재 리전으로 변경
        config.put("region", "ap-guangzhou");

        // 여기서 허용된 경로의 접두사로 바꾸면 자신의 웹 사이트의 사용자 로그인 상태에 따
        // 몇 가지 일반적인 접두사 인증 시나리오:
        // 1. 모든 객체에 대한 액세스 허용: "*"
        // 2. 지정된 객체에 대한 액세스 허용: "a/a1.txt", "b/b1.txt"
        // 3. 지정된 접두사의 객체에 대한 액세스 허용: "a*", "a/*", "b/*"
        // '*'를 입력하면 사용자가 모든 리소스에 액세스하는 것을 허용하게 됩니다. 업무에 꼭
        config.put("allowPrefixes", new String[] {
                "exampleobject",
                "exampleobject2"
        });

        // 키에 대한 권한 목록입니다. 이 임시 키에 필요한 권한을 여기에서 지정해야 합니다.
        // 간편 업로드, 폼 업로드 및 멀티파트 업로드 시에는 다음과 같은 권한이 필요합니다.
        String[] allowActions = new String[] {
                 // 간편 업로드
                "name/cos:PutObject",
                // 폼 업로드, 미니프로그램 업로드
                "name/cos:PostObject",
                // 멀티파트 업로드
                "name/cos:InitiateMultipartUpload",
                "name/cos:ListMultipartUploads",
                "name/cos:ListParts",
                "name/cos:UploadPart",
                "name/cos:CompleteMultipartUpload"
        };
        config.put("allowActions", allowActions);
            /**
         * condition 설정(필요한 경우)
         //# 임시 키가 적용되는 조건. COS에서 지원하는 condition 및 condition 유형의 자
         final String raw_policy = "{\\n" +
         "  \\"version\\":\\"2.0\\",\\n" +
         "  \\"statement\\":[\\n" +
```

```
                "    {\\n" +
                "      \\"effect\\":\\"allow\\",\\n" +
                "      \\"action\\":[\\n" +
                "          \\"name/cos:PutObject\\",\\n" +
                "          \\"name/cos:PostObject\\",\\n" +
                "          \\"name/cos:InitiateMultipartUpload\\",\\n" +
                "          \\"name/cos:ListMultipartUploads\\",\\n" +
                "          \\"name/cos:ListParts\\",\\n" +
                "          \\"name/cos:UploadPart\\",\\n" +
                "          \\"name/cos:CompleteMultipartUpload\\"\\n" +
                "        ],\\n" +
                "      \\"resource\\":[\\n" +
                "          \\"qcs::cos:ap-shanghai:uid/1250000000:examplebucket-125000
                "      ],\\n" +
                "      \\"condition\\": {\\n" +
                "        \\"ip_equal\\": {\\n" +
                "            \\"qcs:ip\\": [\\n" +
                "                \\"192.168.1.0/24\\",\\n" +
                "                \\"101.226.100.185\\",\\n" +
                "                \\"101.226.100.186\\"\\n" +
                "            ]\\n" +
                "        }\\n" +
                "      }\\n" +
                "    }\\n" +
                "  ]\\n" +
                "}";

                config.put("policy", raw_policy);
                */


            Response response = CosStsClient.getCredential(config);
            System.out.println(response.credentials.tmpSecretId);
            System.out.println(response.credentials.tmpSecretKey);
            System.out.println(response.credentials.sessionToken);
        } catch (Exception e) {
            e.printStackTrace();
            throw new IllegalArgumentException("no valid secret !");
        }
    }
}
```

FAQ 및 답변

JSONObject 패키지 충돌로 인한 NoSuchMethodError

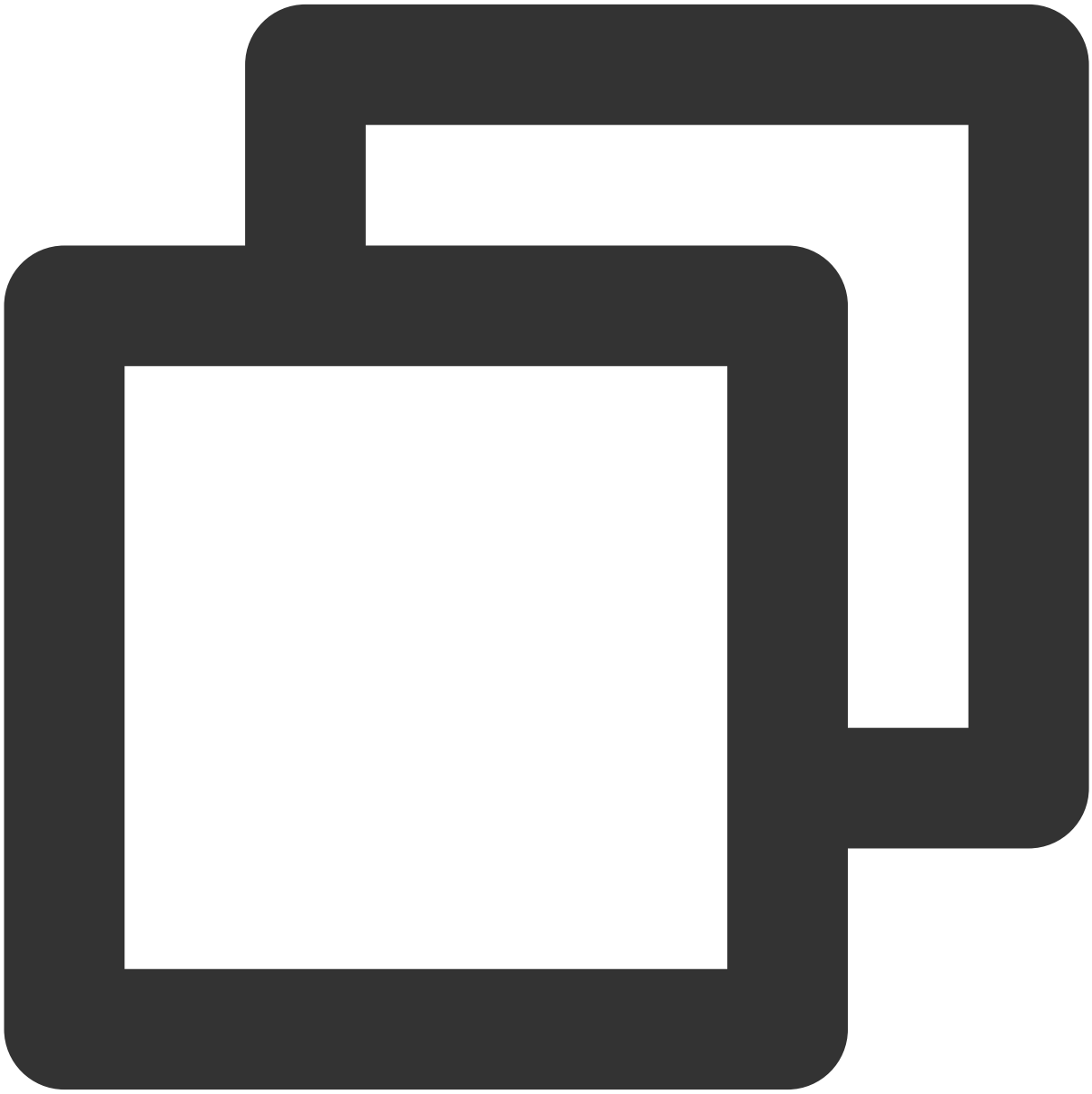
3.1.0 이상 버전을 사용합니다.

임시 키를 사용한 COS 액세스

COS API에서 임시 키를 사용해 COS 서비스에 액세스하는 경우, x-cos-security-token 필드를 통해 임시 sessionToken를 전달하고 임시 SecretId와 SecretKey로 서명을 컴퓨팅합니다.
임시 키를 사용하여 COS에 액세스하는 COS Java SDK 예시는 다음과 같습니다.

설명：

다음 예시 실행 전 Github 프로젝트에서 Java SDK 설치 패키지를 다운로드하십시오.



```
// github에서 제공하는 maven 통합 방법에 따라 cos xml java sdk 가져오기
import com.qcloud.cos.*;
import com.qcloud.cos.auth.*;
```

```
import com.qcloud.cos.exception.*;
import com.qcloud.cos.model.*;
import com.qcloud.cos.region.*;
public class Demo {
    public static void main(String[] args) throws Exception {

        // 사용자 기본 정보
        String tmpSecretId = "COS_SECRETID";    // STS 인터페이스에서 반환하는 임시 Secre
        String tmpSecretKey = "COS_SECRETKEY";  // STS 인터페이스에서 반환하는 임시 Sec
        String sessionToken = "Token";  // STS 인터페이스에서 반환하는 임시 Token으로 변

        // 1 사용자 자격 정보 초기화(secretId, secretKey)
        COSCredentials cred = new BasicCOSCredentials(tmpSecretId, tmpSecretKey);
        // 2. bucket 리전 설정. COS 리전에 대한 자세한 내용은 다음을 참고하십시오. https://
        ClientConfig clientConfig = new ClientConfig(new Region("ap-guangzhou"));
        // 3 cos 클라이언트 생성
        COSClient cosclient = new COSClient(cred, clientConfig);
        // bucket 이름에는 반드시 appid를 포함
        String bucketName = "examplebucket-1250000000";

        String key = "exampleobject";
        // object 업로드, 20M 이상의 파일은 해당 인터페이스를 사용해 업로드하는 것을 권장
        File localFile = new File("src/test/resources/text.txt");
        PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, l

        // x-cos-security-token header 필드 설정
        ObjectMetadata objectMetadata = new ObjectMetadata();
        objectMetadata.setSecurityToken(sessionToken);
        putObjectRequest.setMetadata(objectMetadata);

        try {
            PutObjectResult putObjectResult = cosclient.putObject(putObjectRequest)
            // 성공: putobjectResult에서 파일의 etag 반환
            String etag = putObjectResult.getETag();
        } catch (CosServiceException e) {
            //실패 시 CosServiceException 팝업
            e.printStackTrace();
        } catch (CosClientException e) {
            //실패 시 CosClientException 팝업
            e.printStackTrace();
        }

        // 클라이언트 종료
        cosclient.shutdown();

    }
}
```

서브 계정에 태그별 버킷 가져오기 권한 부여

최종 업데이트 날짜: : 2024-06-24 16:44:04

소개

Cloud Object Storage(COS)를 사용하면 콘솔에서 또는 API를 통해 태그별로 버킷을 필터링할 수 있습니다. 이는 태그별 권한 부여를 기반으로 구현됩니다.

인증 단계

1. 루트 계정 Owner로 CAM 콘솔에 로그인하고 정책 구성 페이지로 이동합니다.
2. 다음과 같이 **정책 생성기** 또는 **정책 구문**을 통해 지정된 태그가 있는 버킷에 대한 서브 계정 SubUser 액세스 권한을 부여합니다.

정책 생성기 사용

정책 구문

1. CAM 콘솔 구성 페이지로 이동합니다.
2. **사용자 지정 정책 생성 > 정책 생성기에서 생성**을 클릭합니다.
3. 권한 구성 페이지로 이동하여 다음과 같이 정보를 구성합니다.

효과: 허용이 선택되어 있으며, 기본값에서 변경하지 않습니다.

서비스: COS를 선택합니다.

작업: **읽기 > GetService(버킷 목록 가져오기)**를 선택합니다.

리소스: **모든 리소스**를 선택합니다.

조건: **기타 조건 추가**를 클릭합니다. 패널에서 다음을 구성합니다.

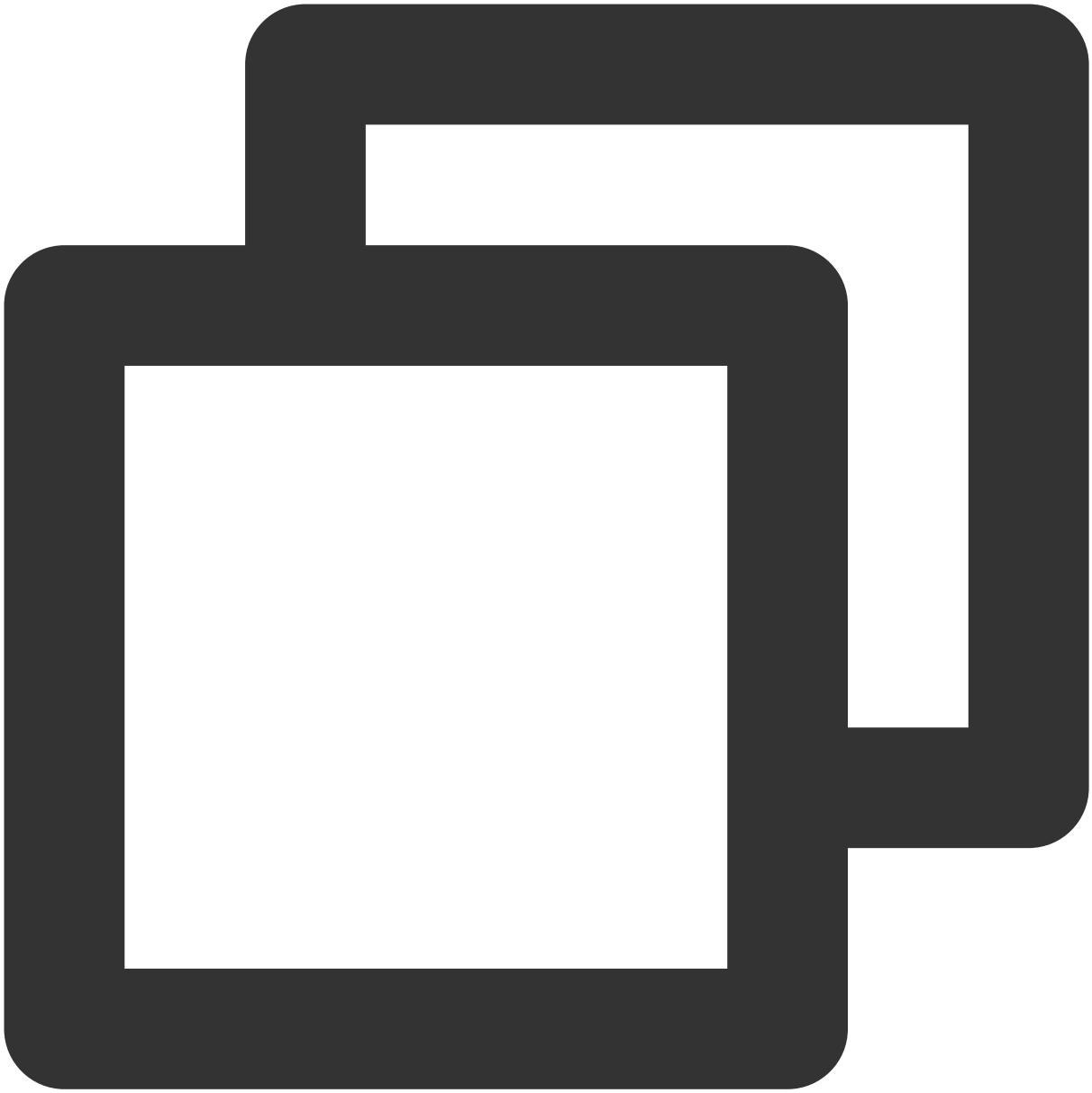
조건 키: `qcs:resource_tag` 를 선택합니다.

오퍼레이터: `string_equal` 을 선택합니다.

조건 값: `key&val` 형식으로 태그를 입력합니다. 여기에서 key와 value을 각각 태그 키와 값으로 바꿉니다.

4. **다음**을 클릭하고 정책 이름을 입력합니다.
5. **완료**를 클릭하면 생성이 완료됩니다.

1. CAM 콘솔 구성 페이지로 이동합니다.
2. **사용자 지정 정책 생성 > 정책 구문으로 생성**을 클릭합니다.
3. 빈 템플릿을 선택하고 **다음**을 클릭합니다.
4. 다음 형식으로 정책을 입력합니다. 여기에서 key와 value을 각각 지정된 태그 키와 값으로 바꿉니다.



```
{
 "version": "2.0",
 "statement": [
     {
         "effect": "allow",
         "action":[
             "name/cos:GetService"
         ],
         "resource": "*",
         "condition":{
             "for_any_value:string_equal":{
```

```
                    "qcs:resource_tag": [
                        "key&value"
                    ]
                }
            }
        }
    ]
}
```

5. **완료**를 클릭하면 생성이 완료됩니다.

3. 정책 페이지의 2단계에서 생성한 정책을 찾고 오른쪽에서 **사용자/사용자 그룹/역할** 연결을 클릭하여 정책을 서브 계정 SubUser와 연결합니다.

4. 팝업 창에서 서브 계정 SubUser를 선택하고 **확인**을 클릭합니다.

콘솔에서 보기

1. 서브 계정 SubUser로 COS 콘솔에 로그인합니다.

2. 버킷 목록 페이지에는 **서브 계정이 액세스할 수 있는 버킷 목록이 자동으로 표시됩니다**.

이제 지정된 태그(key 및 value)가 있는 버킷에 대한 서브 계정 액세스 권한 부여가 완료되었습니다.

API 호출

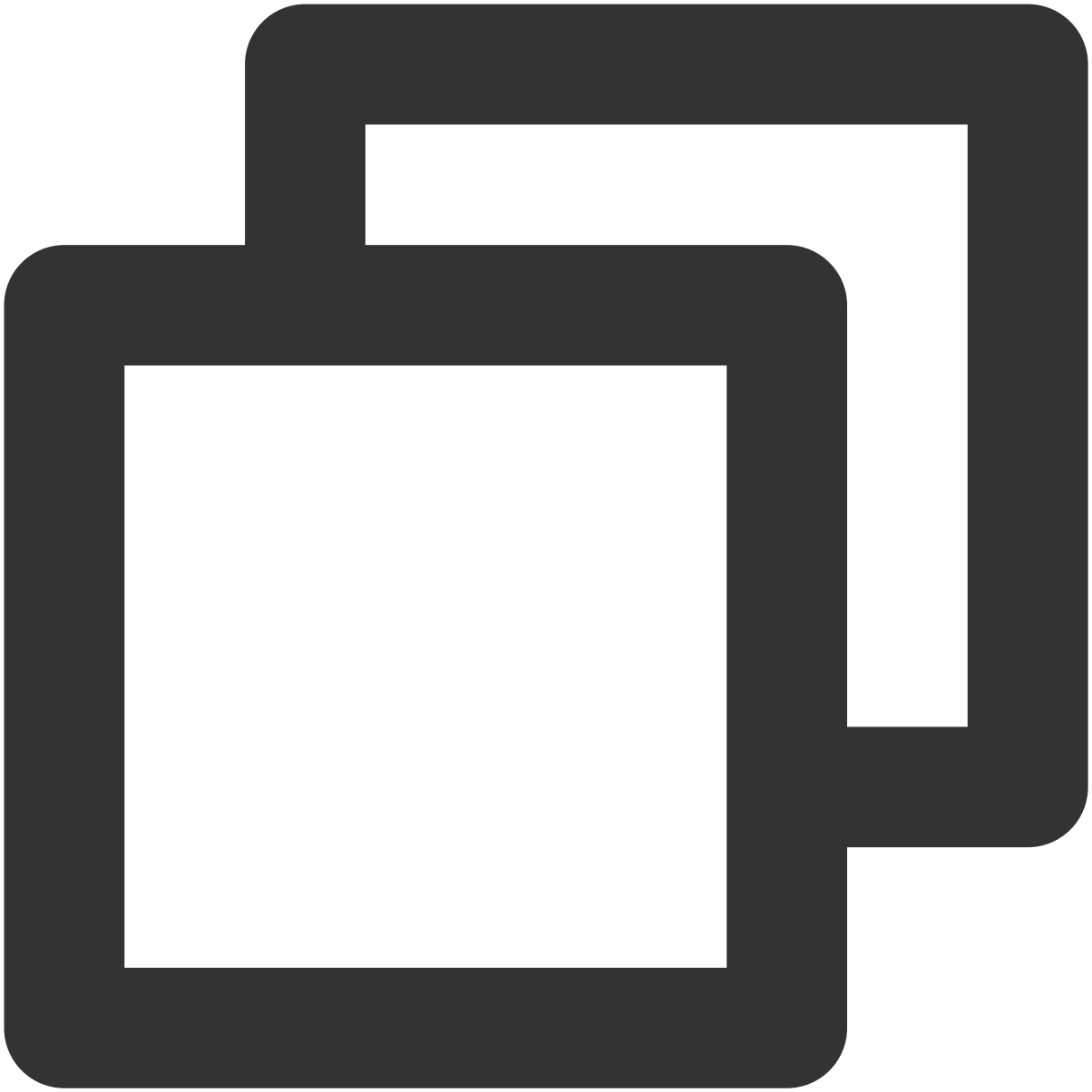
주의 :

콘솔과 달리 GetService API는 서브 계정이 액세스할 수 있는 버킷 목록을 자동으로 표시할 수 없으며 태그 매개변수를 전달해야 합니다.

현재 GetService API를 사용하면 하나의 태그만 전달할 수 있습니다.

1. 서브 계정 SubUser의 키로 요청을 시작합니다.

2. GetService API를 호출하여 (key, value)와 같은 태그 필터링 매개변수를 전달합니다. 아래는 샘플 요청입니다. 자세한 내용은 GET Service(List Buckets)를 참고하십시오.



```
GET /?tagkey=key1&tagvalue=value1 HTTP/1.1
Date: Fri, 24 May 2019 11:59:51 GMT
Authorization: Auth String
```

조건 키 설명 및 사용 예시

최종 업데이트 날짜: : 2024-06-24 16:44:04

액세스 정책을 사용하여 권한을 부여할 때 정책 조건을 지정할 수 있습니다. 예를 들어 정책 조건을 사용하여 업로드할 파일의 스토리지 클래스와 사용자 액세스 소스를 제한할 수 있습니다.
이 문서는 버킷 정책에서 COS(Cloud Object Storage) 조건 키를 사용하는 일반적인 예시를 제공합니다. 조건 문서에서 COS 및 해당 요청이 지원하는 모든 조건 키를 볼 수 있습니다.
설명 :
버킷 정책을 작성 시 조건 키를 사용할 때 최소 권한 원칙을 준수하고, 해당 조건 키를 해당 요청(action)에만 추가하며, 작업 지정(action) 시 “*” 와일드카드를 사용하지 마십시오. 와일드카드를 사용하면 요청이 실패합니다. 조건 키에 대한 자세한 내용은 조건 문서를 참고하십시오.
액세스 관리(CAM) 콘솔을 사용하여 정책을 생성할 때, version, principal, statement, effect, action, resource, condition 구문 요소의 첫 번째 글자는 대문자이거나 모두 소문자여야 합니다.

조건 키의 사용 사례

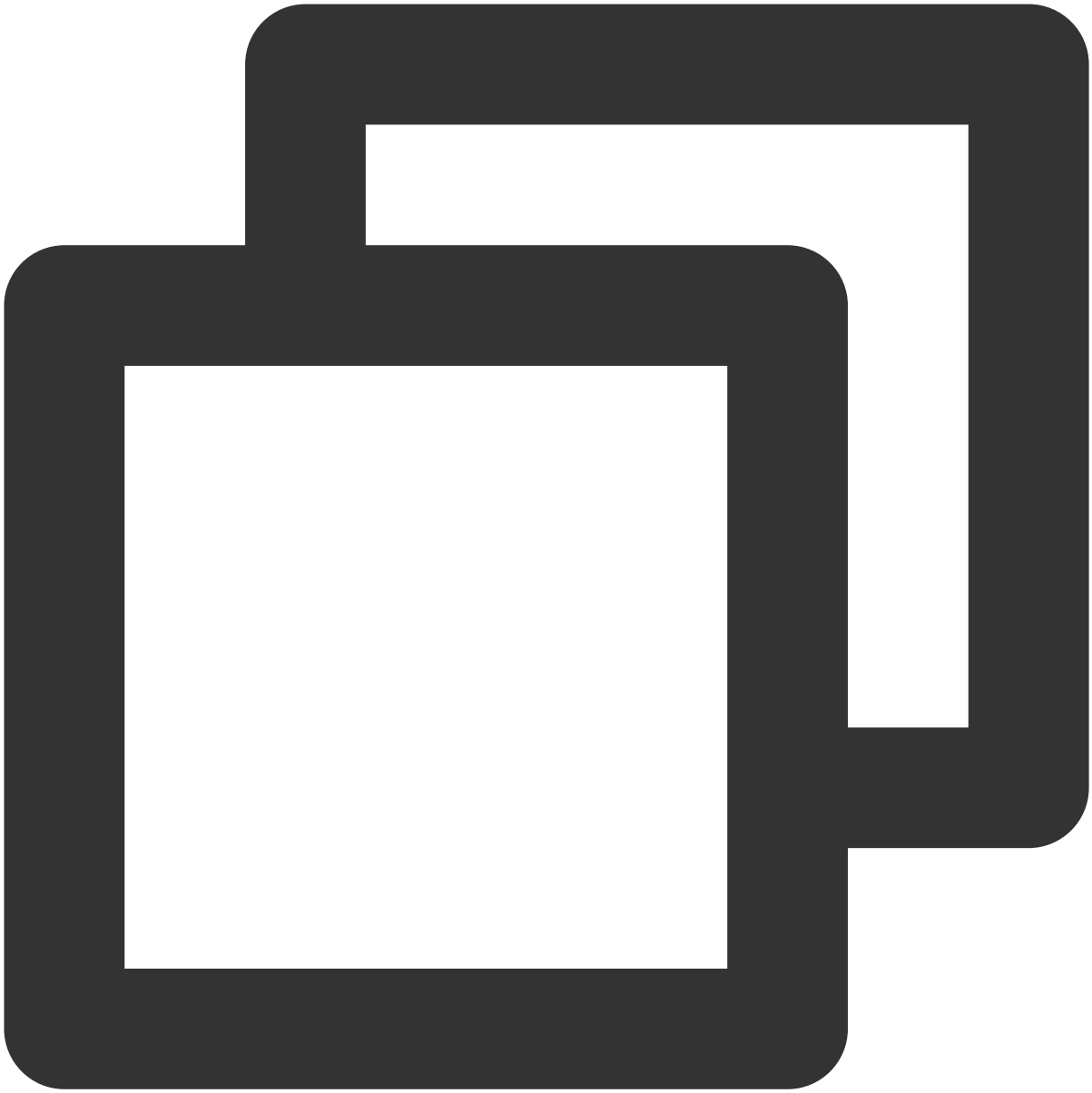
사용자 액세스 IP 제한(qcs:ip)

조건 키 qcs:ip

조건 키 `qcs:ip` 를 사용하여 사용자 액세스 IP를 제한할 수 있습니다. 조건 키는 모든 요청에 적용할 수 있습니다.

예시: 지정 IP의 사용자의 액세스만 허용

다음은 ID가 100000000001(APPID는 1250000000)인 루트 계정에 속하는 ID가 100000000002인 서브 계정이 베이징 리전의 버킷 examplebucket-bj와 광저우 리전의 버킷 examplebucket-gz의 객체 exampleobject에 대해, 액세스 IP가 IP 범위 `192.168.1.0/24` 에 속하거나 액세스가 IP가 `101.226.100.185` 또는 `101.226.100.186` 대역인 경우 객체 업로드 및 객체 다운로드 권한을 부여하는 정책 예시입니다.



```
{
    "version": "2.0",
    "principal":{
        "qcs":[
            "qcs::cam::uin/100000000001:uin/100000000002"
        ]
    },
    "statement": [
        {
            "effect": "allow",
            "action":[
```

```
                "name/cos:PutObject",
                "name/cos:GetObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-gz-1250000000/e
            ],
            "condition":{
                "ip_equal":{
                    "qcs:ip":[
                        "192.168.1.0/24",
                        "101.226.100.185",
                        "101.226.100.186"
                    ]
                }
            }
        }
    ]
}
```

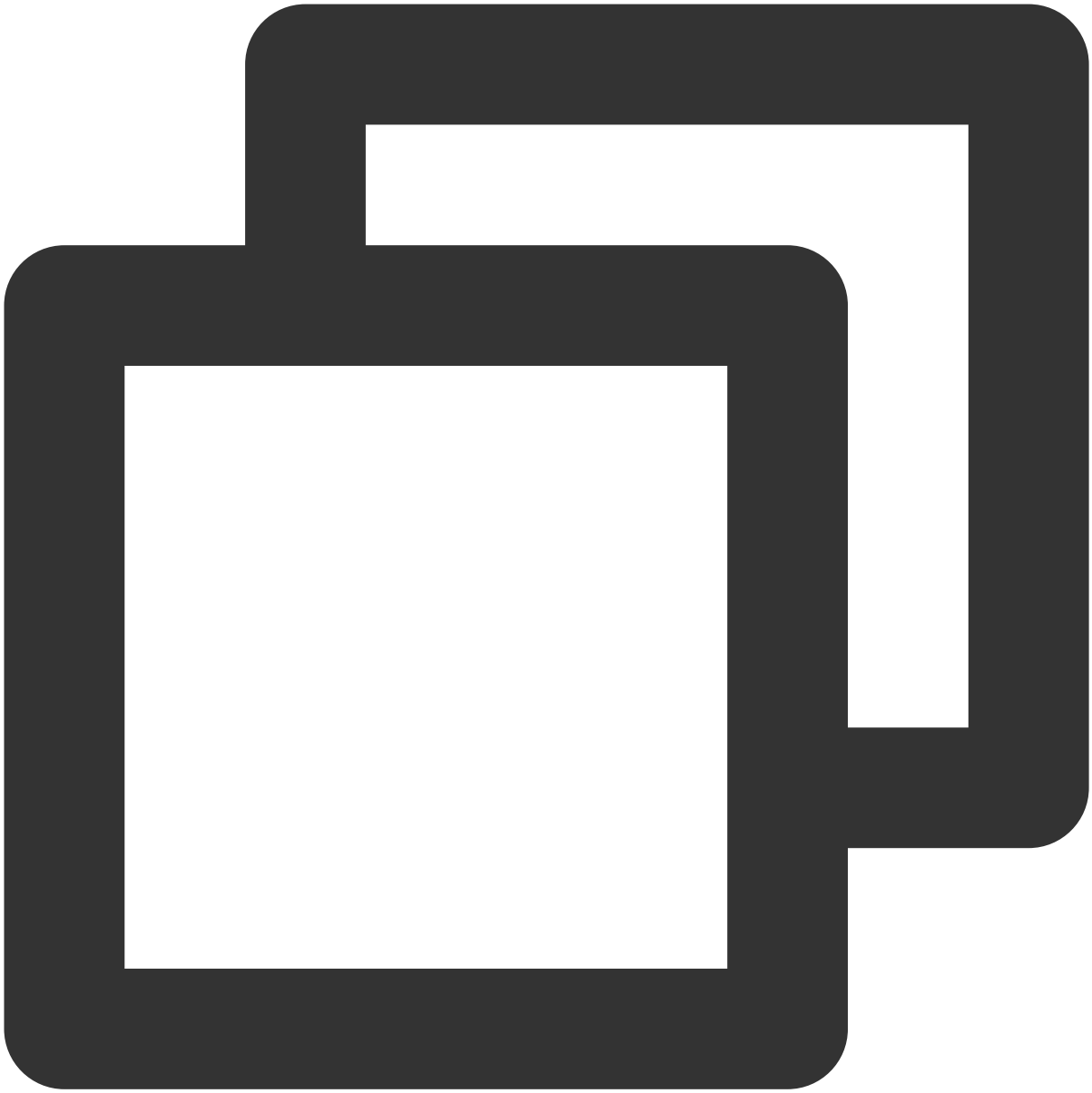
vpcid 제한(vpc:requester_vpc)

조건 키 vpc:requester_vpc

조건 키 `vpc:requester_vpc` 를 사용하여 사용자 액세스 vpcid를 제한할 수 있습니다. vpcid에 대한 자세한 내용은 Tencent Cloud 제품 Virtual Private Cloud를 참고하십시오.

예시: vpcid를 aqp5jrc1로 제한

이 예시의 정책은 vpcid가 aqp5jrc1인 조건에서 ID 100000000001(APPID: 1250000000)의 루트 계정에 속하는 ID 100000000002의 서브 계정이 examplebucket-1250000000 버킷에 액세스하도록 허용합니다.



```
{
  "statement": [
    {
      "action":[
        "name/cos:*"
      ],
      "condition":{
        "string_equal":{
          "vpc:requester_vpc": [
            "vpc-aqp5jrc1"
          ]
```

```
          }
        },
        "effect": "allow",
        "principal":{
          "qcs":[
            "qcs::cam::uin/100000000001:uin/100000000002"
          ]
        },
        "resource":[
          "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/*"
        ]
      }
    ],
    "version": "2.0"
  }
```

객체의 최신 버전 또는 지정 버전(cos:versionid) 액세스만 허용

요청 매개변수 versionid

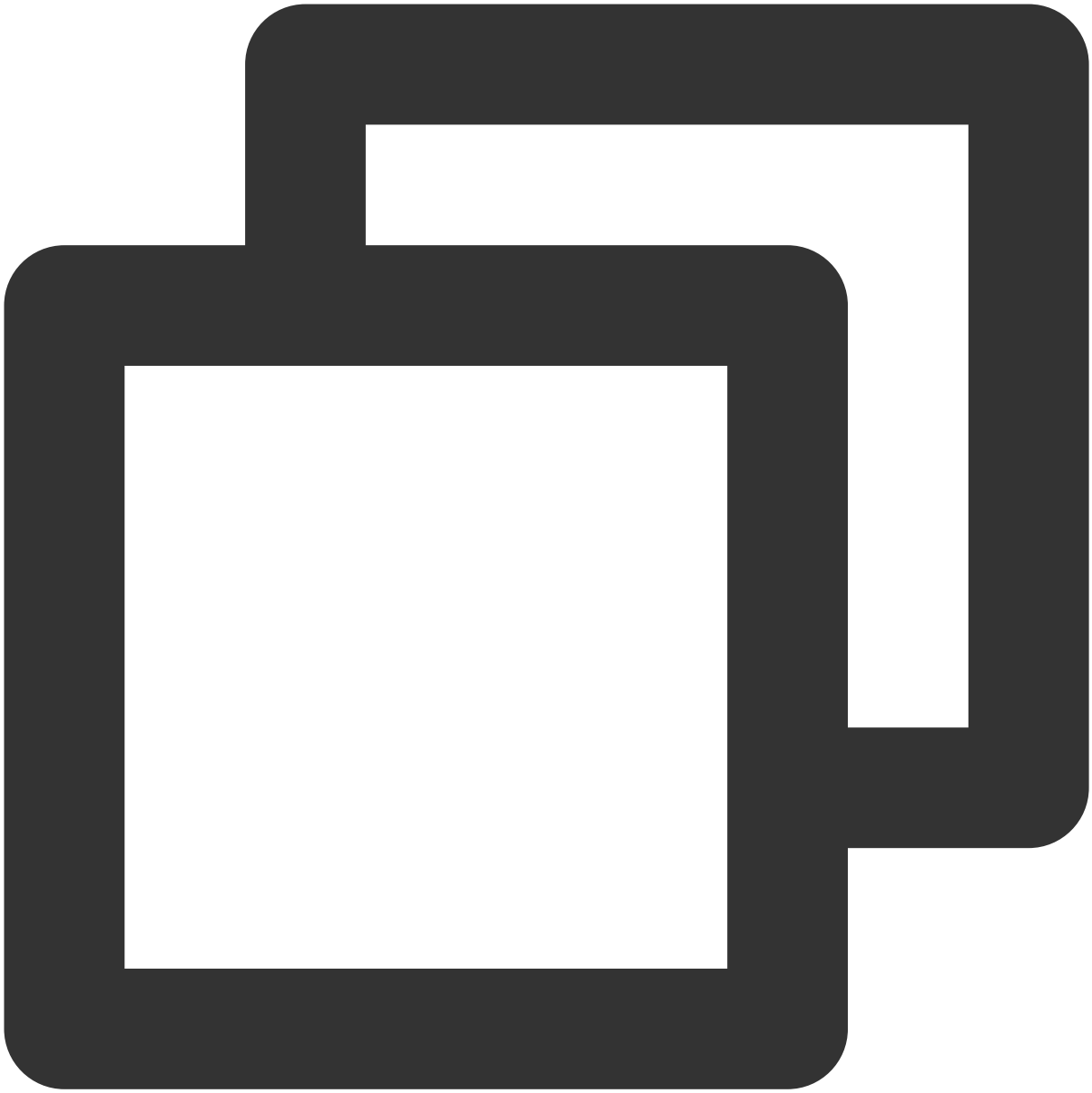
요청 매개변수 `versionid` 는 객체의 버전 번호를 지정합니다. 버전 관리에 대한 자세한 내용은 버전 제어 개요를 참고하십시오. 객체를 다운로드(GetObject)하거나 객체를 삭제할 때(DeleteObject) 요청 매개변수 `versionid` 를 사용하여 작업할 객체 버전을 지정할 수 있습니다. versionid에는 세 가지 다른 경우가 있습니다.
`versionid` 가 없음: 요청은 기본적으로 최신 버전의 객체에 적용됩니다.
`versionid` 가 빈 문자열인 경우: `versionid` 요청 매개변수가 전달되지 않은 경우와 동일합니다.
`versionid` 가 `"null"` 인 경우: 버킷에 대해 버전 관리가 활성화되기 전에 업로드된 객체의 경우 버전 관리가 활성화된 후 버전 번호가 `"null"` 문자열이 됩니다.

조건 키 cos:versionid

조건 키 `cos:versionid` 를 사용하여 요청 매개변수 `versionid` 를 제한할 수 있습니다.

예시1: 사용자가 지정된 버전의 객체를 가져오도록 허용

examplebucket-1250000000 버킷을 소유하는 uin 100000000001의 루트 계정이 다음 버킷 정책을 사용하여 uin이 100000000000인 서브 계정이 지정된 버전의 객체만 가져오도록 허용한다고 가정합니다.
정책에 따르면 uin이 100000000002인 서브 계정으로 전송된 객체 다운로드 요청은 versionid 매개변수를 전달하고 versionid 값이 버전 번호 'MTg0NDUxNTc1NjIzMTQ1MDAwODg'인 경우에만 성공할 수 있습니다.

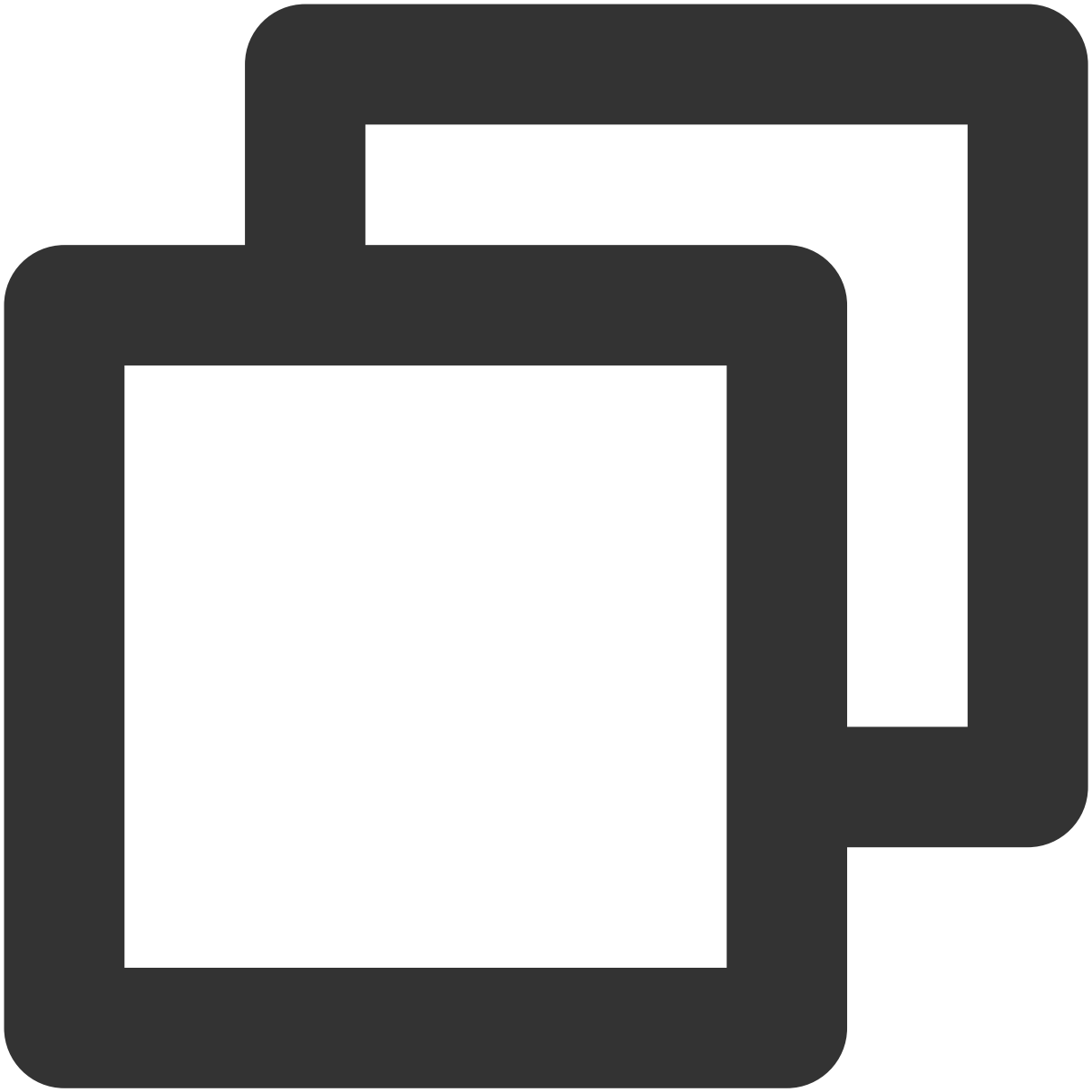


```
{
    "version":"2.0",
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
```

```
                "name/cos:GetObject"
            ],
            "condition":{
                "string_equal":{
                    "cos:versionid":"MTg0NDUxNTc1NjIzMTQ1MDAwODg"
                }
            },
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ]
        }
    ]
}
```

거부 정책 추가

상기 정책을 사용하여 서브 계정에게 권한을 부여하고 서브 계정이 다른 수단을 통해 조건을 부여하지 않고 동일한 권한을 얻는 경우 더 광범위한 권한 부여 정책이 적용됩니다. 예를 들어 서브 계정이 사용자 그룹에 있고 루트 계정이 아무런 조건 없이 사용자 그룹에 GetObject 권한을 부여하면 상기 정책의 버전 번호에 대한 제한이 적용되지 않습니다.
이에 대처하기 위해 위의 정책을 기반으로 명시적 거부 정책(deny)을 추가하여 더 엄격한 권한 제한을 달성할 수 있습니다. 다음 deny 정책은 서브 사용자가 versionid 매개변수를 포함하지 않는 객체 다운로드 요청을 시작하거나 versionid에 의해 지정된 버전 번호가 'MTg0NDUxNTc1NjIzMTQ1MDAwODg'가 아닌 경우 요청이 거부되도록 지정합니다. deny 정책의 우선 순위가 다른 정책보다 높기 때문에 거부 정책을 추가하면 권한 취약성을 최대한 방지할 수 있습니다.

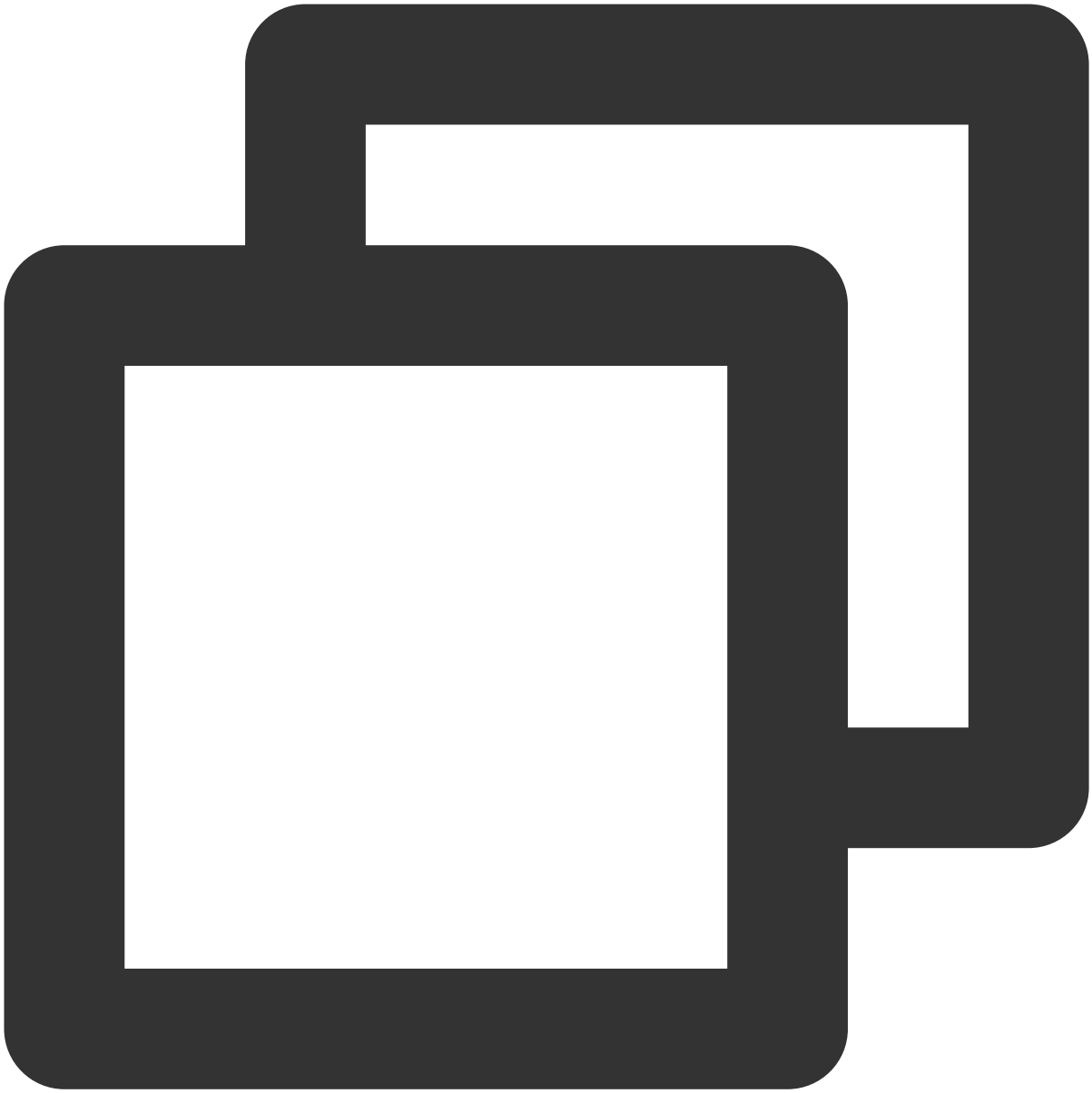


```
{
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
                "name/cos:GetObject"
```

```
            ],
            "condition":{
                "string_equal":{
                    "cos:versionid":"MTg0NDUxNTc1NjIzMTQ1MDAwODg"
                }
            },
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ]
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "name/cos:GetObject"
            ],
            "condition":{
                "string_not_equal_if_exist":{
                    "cos:versionid":"MTg0NDUxNTc1NjIzMTQ1MDAwODg"
                }
            },
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ]
        }
    ],
    "version":"2.0"
}
```

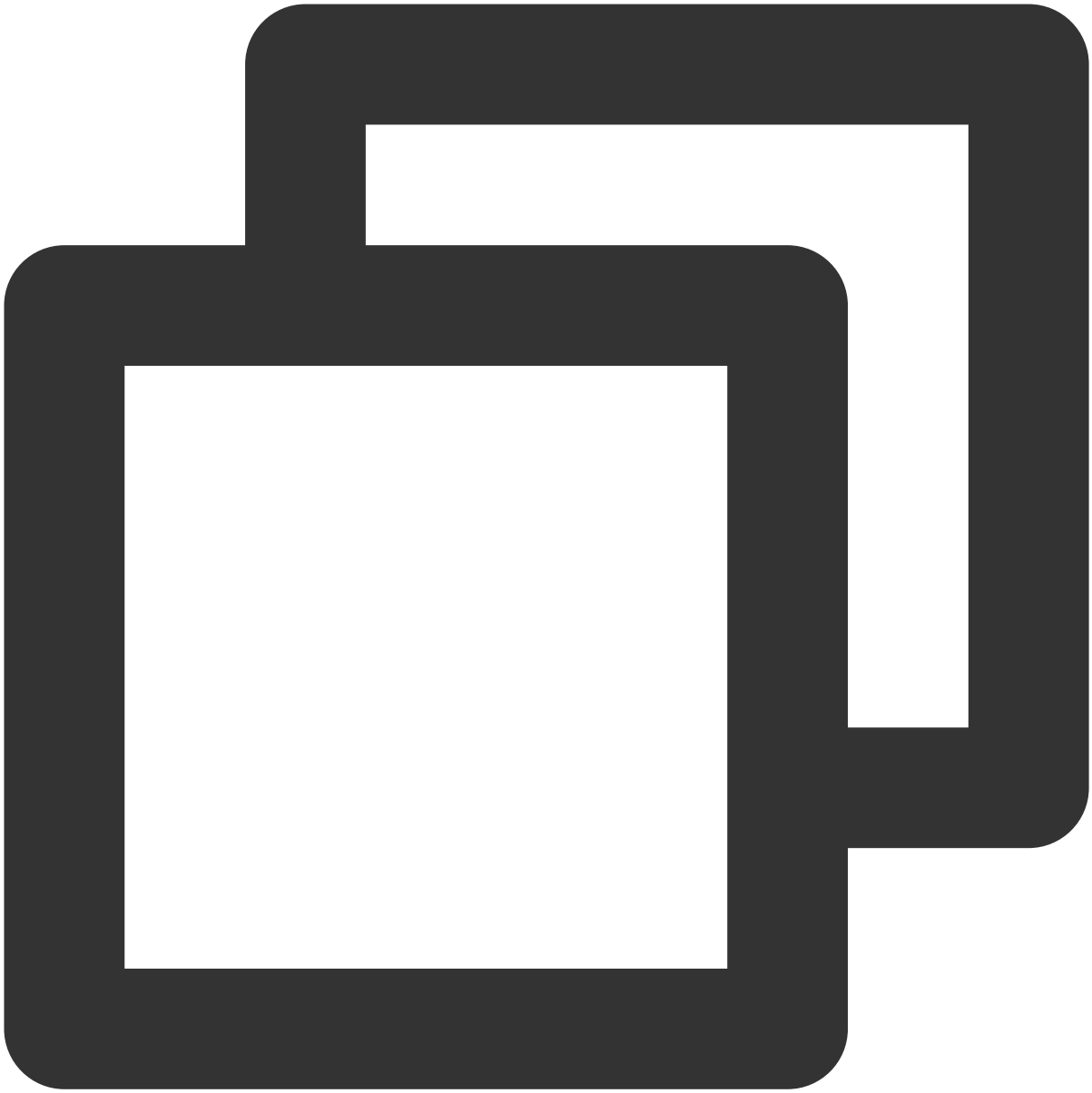
예시2: 사용자가 최신 버전의 객체만 가져올 수 있도록 허용

버킷 examplebucket-1250000000을 소유하는 uin 100000000001의 루트 계정이 다음 버킷 정책을 사용하여 uin이 100000000002인 서브 계정이 최신 버전의 객체만 가져오도록 허용한다고 가정합니다.
정책에 따라 `versionid` 가 전달되지 않거나 해당 값이 빈 문자열인 경우 GetObject 요청은 기본적으로 최신 버전의 객체를 다운로드합니다. 따라서 다음 조건에서 string_equal_if_exsit를 사용할 수 있습니다.
1. versionid가 전달되지 않으면 기본적으로 조건이 충족된 것으로 간주(true)되어 allow 정책이 적용되어 요청이 allow 됩니다.
2. versionid가 빈 문자열(`“”`)인 경우 allow 정책도 적용되며 최신 버전의 객체 다운로드 요청만 승인됩니다.



```
"condition":{
    "string_equal_if_exist": {
        "cos:versionid": ""
    }
}
```

명시적 거부 정책이 추가된 후 전체 버킷 정책은 다음과 같습니다.



```
{
 "statement":[
     {
         "principal":{
             "qcs":[
                 "qcs::cam::uin/100000000001:uin/100000000002"
             ]
         },
         "effect":"allow",
         "action":[
             "name/cos:GetObject"
```

```
            ],
            "condition":{
                "string_equal_if_exist":{
                    "cos:versionid":""
                }
            },
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ]
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "name/cos:GetObject"
            ],
            "condition":{
                "string_not_equal":{
                    "cos:versionid":""
                }
            },
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ]
        }
    ],
    "version":"2.0"
}
```

예시3: 버전 관리 활성화 전 업로드된 객체를 사용자가 삭제하는 것을 허용하지 않음

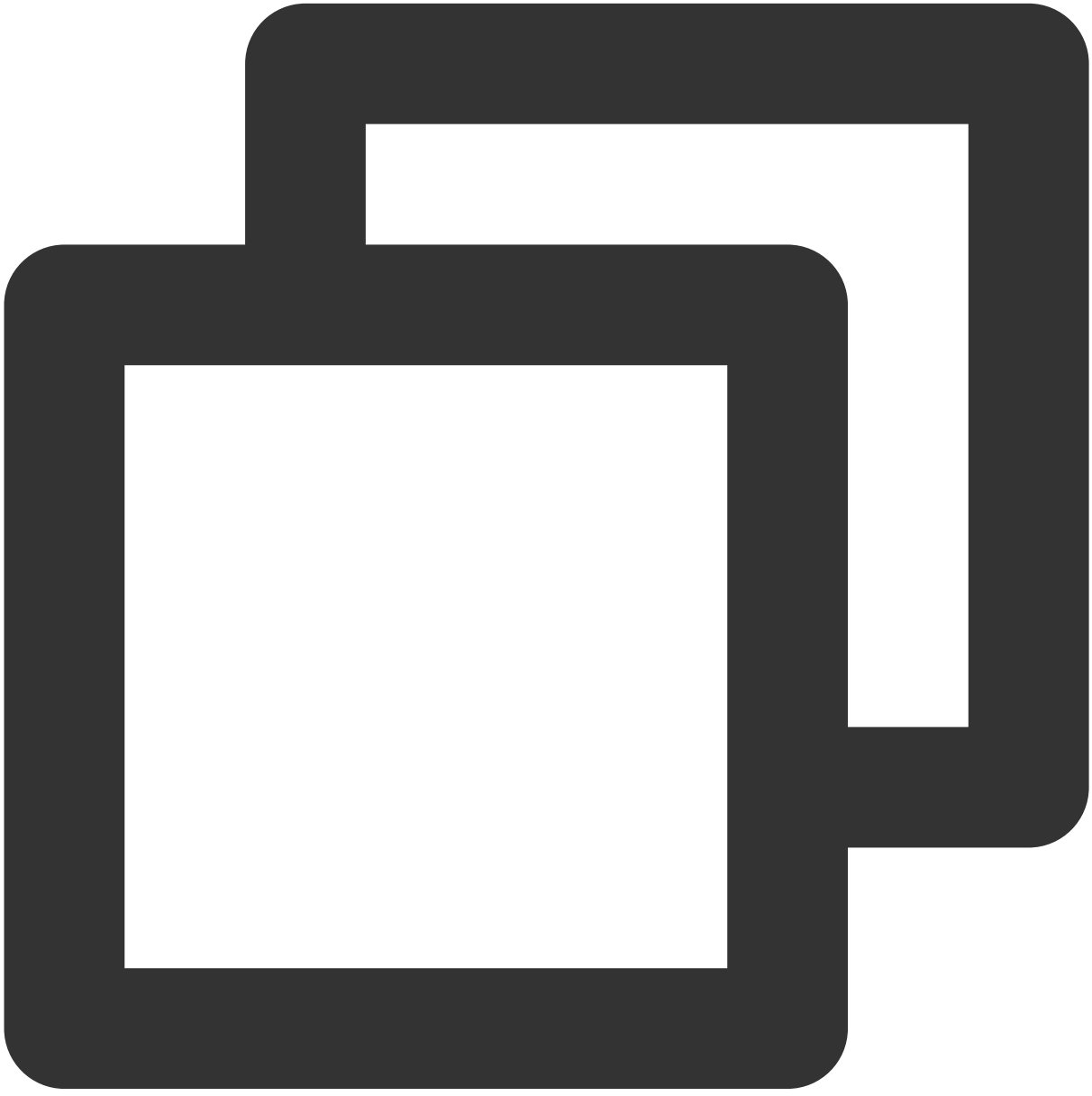
버전 관리가 활성화되기 전에 버킷에 업로드된 일부 객체가 있을 수 있으며 이러한 객체의 버전 번호는 버전 관리가 활성화된 후에 "null"이 됩니다. 경우에 따라 이러한 객체에 대한 추가 보호를 활성화해야 할 수 있습니다. 예를 들어 사용자가 이러한 객체를 영구적으로 삭제하지 못하도록 하는 것, 즉 버전 번호가 있는 객체의 삭제를 거부하는 것입니다.

아래 예시에는 두 가지 버킷 정책이 있습니다.

1. 서브 계정에 DeleteObject 요청을 사용하여 버킷의 객체를 삭제할 수 있는 권한을 부여합니다.

2. DeleteObject 요청에 대한 조건을 제한합니다. DeleteObject 요청이 값이 "null"인 요청 매개변수 versionid를 전달하면 요청이 거부됩니다.

따라서 버전 관리가 활성화되기 전에 객체 A가 버킷 examplebucket-1250000000에 업로드된 경우 버전 관리가 활성화된 후 객체 A의 버전 번호는 "null" 문자열이 됩니다.
버킷 정책이 추가되면 객체 A가 보호됩니다. 서브 사용자가 객체 A를 삭제하기 위해 시작한 DeleteObject 요청에 버전 번호가 없으면 버전 관리가 활성화되어 있기 때문에 객체 A가 영구적으로 삭제되지 않습니다. 대신 객체 A에 대한 삭제 표시가 추가됩니다. 요청에 객체 A의 "null" 버전 번호가 포함되어 있으면 요청이 거부되고 객체 A는 영구적으로 삭제되지 않습니다.



```
{
    "statement":[
        {
```

```
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
                "name/cos:DeleteObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ]
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "name/cos:DeleteObject"
            ],
            "condition":{
                "string_equal":{
                    "cos:versionid":"null"
                }
            },
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ]
        }
    ],
    "version":"2.0"
}
```

업로드 파일 크기 제한(cos:content-length)

요청 헤더 Content-Length

RFC 2616에 정의된 바이트 단위의 HTTP 요청 콘텐츠 길이는 PUT 및 POST 요청에서 자주 사용됩니다. 자세한 내용은 Common Request Headers를 참고하십시오.

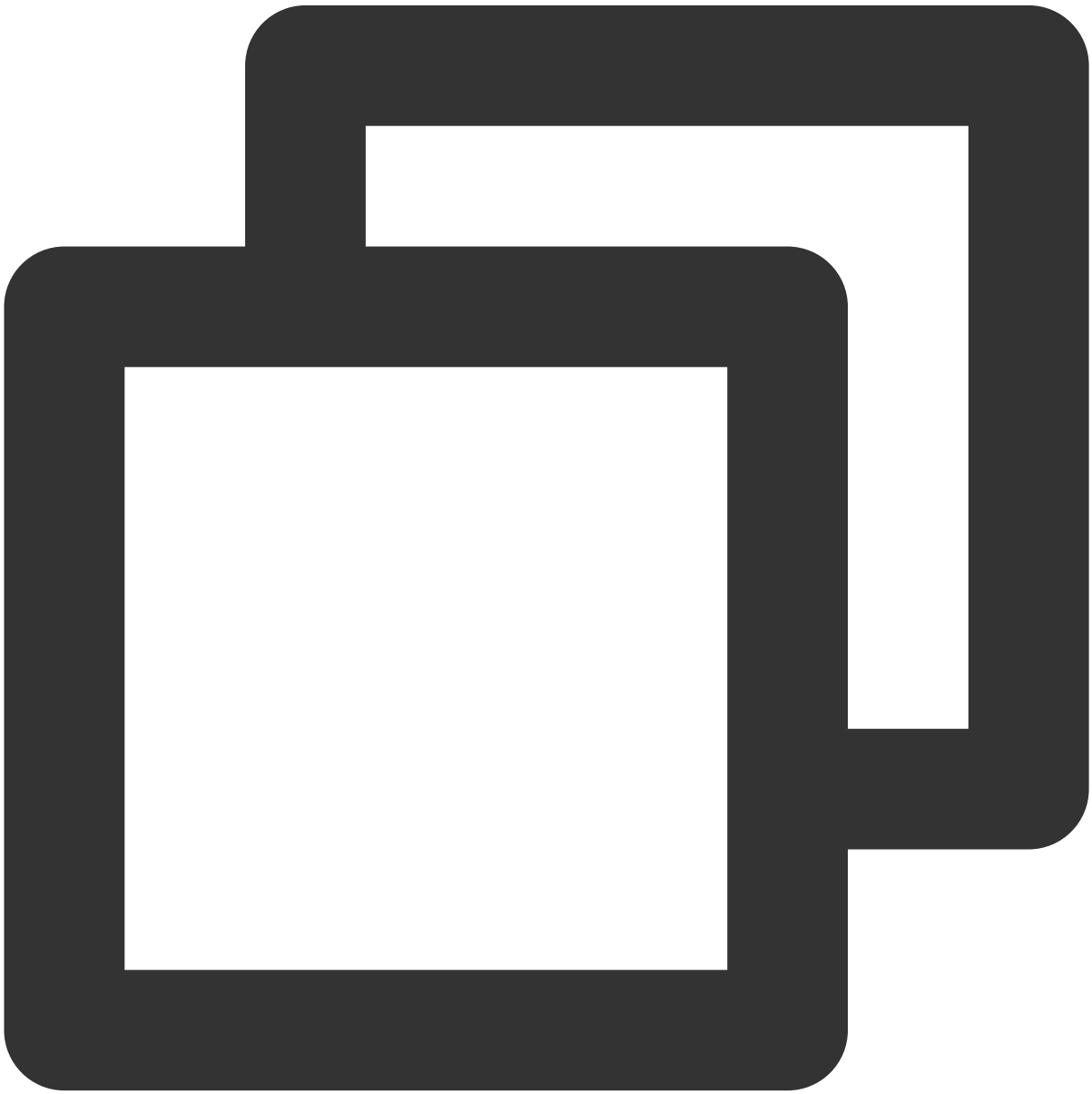
조건 키 cos:content-length

객체를 업로드할 때 조건 키 `cos:content-length` 를 사용하여 요청 헤더 `Content-Length` 를 제한하고 업로드할 객체의 파일 크기를 제한할 수 있습니다. 이러한 방식으로 저장 공간을 유연하게 관리하고 너무 크거나 작은

파일을 업로드하여 저장 공간과 네트워크 대역폭을 낭비하지 않도록 할 수 있습니다.
아래 두 예시에서는 루트 계정(uin: 100000000001)이 버킷 examplebucket-1250000000을 소유하고 있다고 가정하며, `cos:content-length` 조건 키를 사용하여 서브 계정(uin: 100000000002)에서 업로드한 요청의 Content-Length 헤더 값을 제한할 수 있습니다.

예시1: 요청 헤더 Content-Length의 최대값 제한

PutObject 및 PostObject 업로드 요청이 10 바이트 이하의 값을 가진 Content-Length 헤더를 전달해야 한다고 제한합니다.

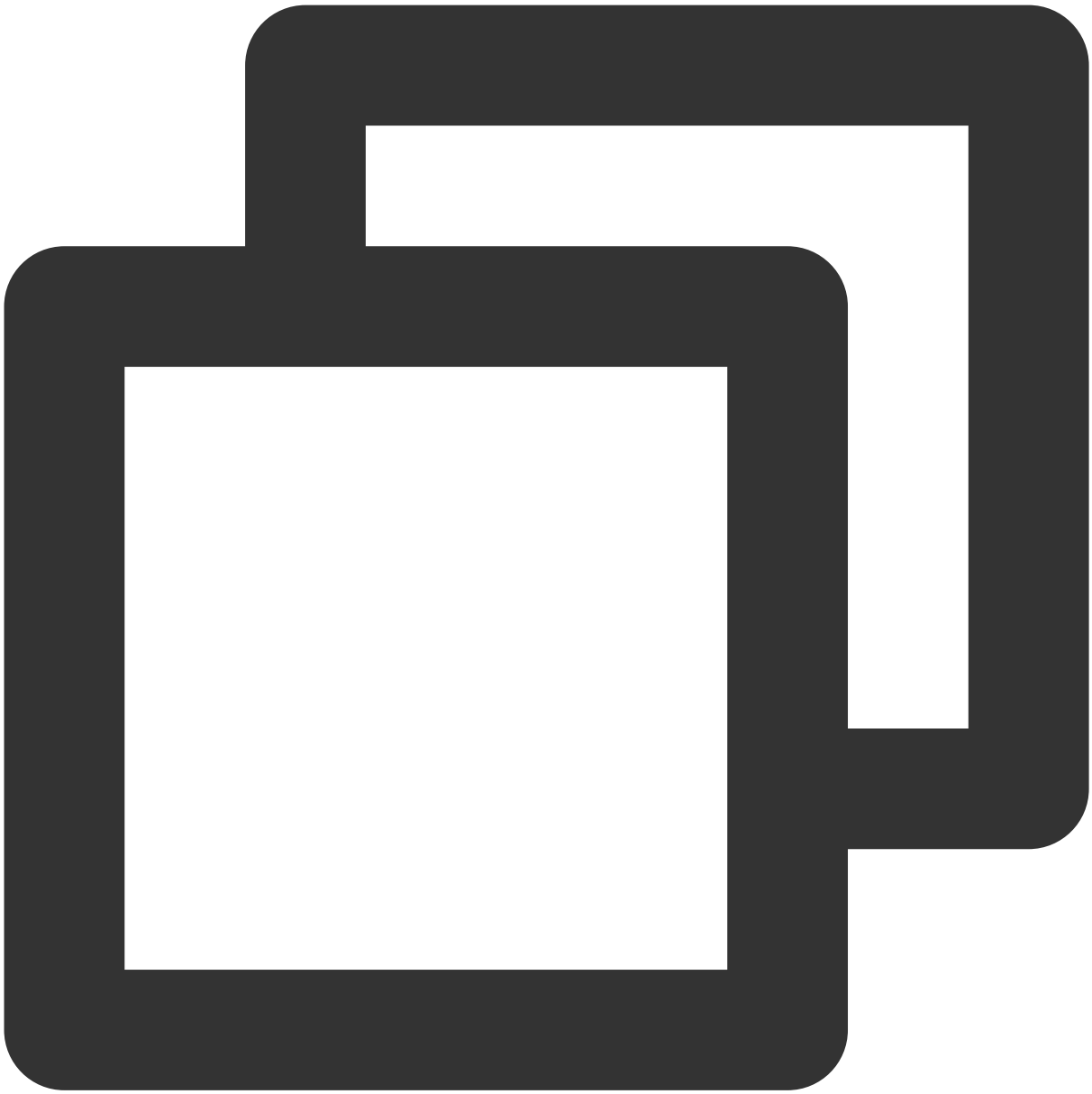


```
{
```

```
    "version":"2.0",
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
                "name/cos:PutObject",
                "name/cos:PostObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "numeric_less_than_equal":{
                    "cos:content-length":10
                }
            }
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "name/cos:PutObject",
                "name/cos:PostObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "numeric_greater_than_if_exist":{
                    "cos:content-length":10
                }
            }
        }
    ]
}
```

예시2: 요청 헤더 Content-Length의 최소값 제한

PutObject 및 PostObject 업로드 요청이 2 바이트 이상의 값을 가진 Content-Length 헤더를 전달해야 한다고 제한합니다.



```
{
    "version":"2.0",
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
```

```
            },
            "effect":"allow",
            "action":[
                "name/cos:PutObject",
                "name/cos:PostObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "numeric_greater_than_equal":{
                    "cos:content-length":2
                }
            }
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "name/cos:PutObject",
                "name/cos:PostObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "numeric_less_than_if_exist":{
                    "cos:content-length":2
                }
            }
        }
    ]
}
```

업로드할 파일 형식 제한(cos:content-type)

요청 헤더 Content-Type

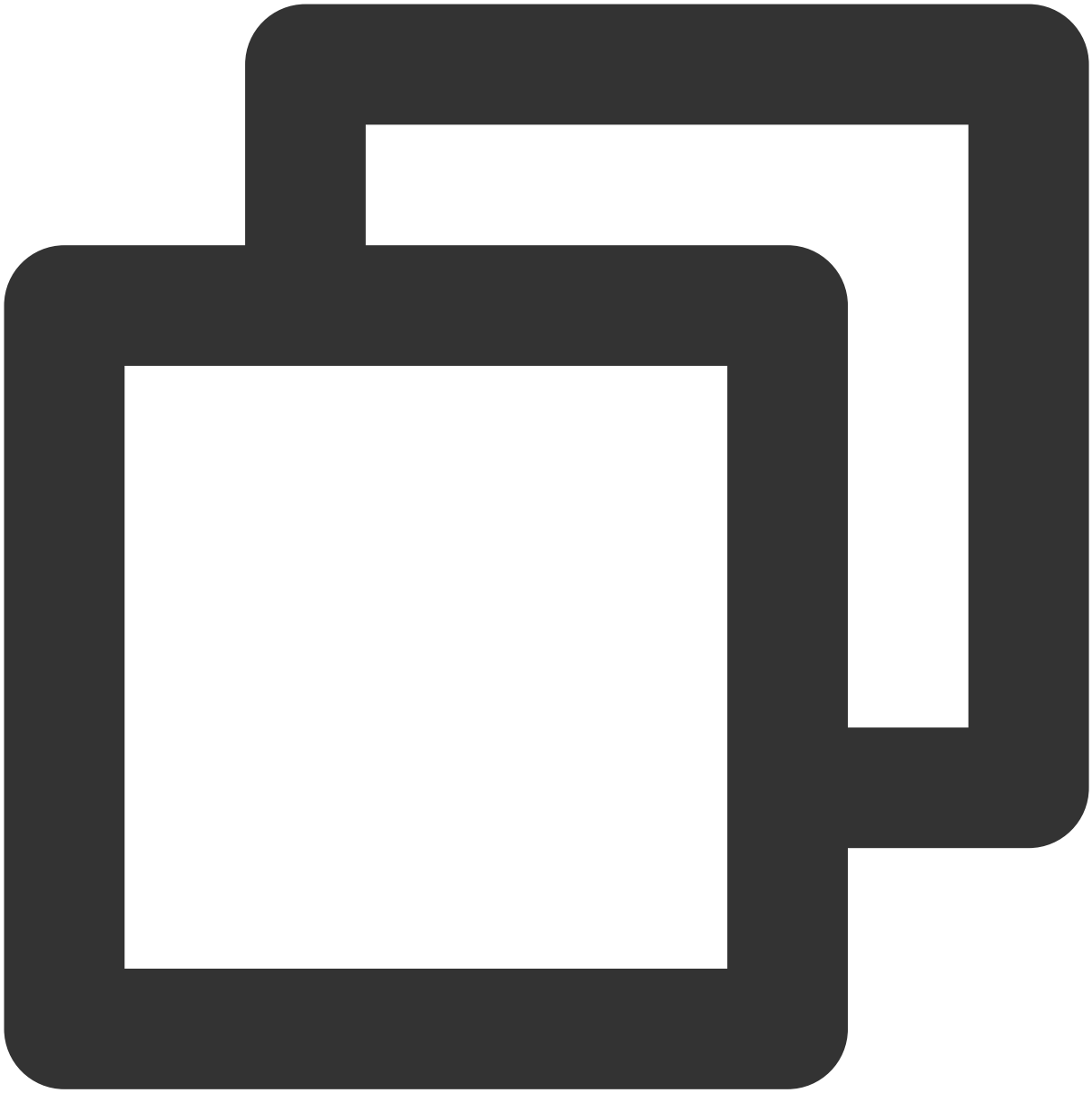
Content-Type은 `application/xml` 및 `image/jpeg` 와 같이 RFC 2616(MIME)에 정의된 HTTP 요청 콘텐츠 유형이어야 합니다. 자세한 내용은 Common Request Headers를 참고하십시오.

조건 키 cos:content-type

조건 키 `cos:content-type` 을 사용하여 요청 헤더 `Content-Type` 을 제한할 수 있습니다.

예시1: PutObject 요청의 Content-Type을 "image/jpeg"로 제한

버킷 examplebucket-1250000000을 소유하는 uin이 100000000001인 루트 계정이 `cos:content-type` 조건 키를 사용하여 uin이 100000000002인 서브 계정이 시작한 업로드 요청에서 Content-Type 헤더의 콘텐츠를 제한한다고 가정합니다.
이 예시의 버킷 정책은 객체 업로드 요청(PutObject)이 Content-Type 헤더와 'image/jpeg' 값을 포함해야 하는 것을 제한하는 것입니다.
string_equal은 요청이 지정된 값과 정확히 동일한 값을 가진 Content-Type 헤더를 전달해야 함을 요구합니다. 실제 요청에서는 **요청의 Content-Type 헤더를 명시적으로 지정해야 합니다**. 그렇지 않고 요청에 Content-Type 헤더가 없으면 요청이 실패합니다. 또한 특정 도구를 사용하여 요청을 시작하고 Content-Type을 명시적으로 지정하지 않으면 도구에서 예기치 않은 Content-Type 헤더를 요청에 자동으로 추가할 수 있으며 요청도 실패할 수 있습니다.



```
{
    "version":"2.0",
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
```

```
                "name/cos:PutObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "string_equal":{
                    "cos:content-type":"image/jpeg"
                }
            }
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "name/cos:PutObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "string_not_equal_if_exist":{
                    "cos:content-type":"image/jpeg"
                }
            }
        }
    ]
}
```

다운로드 요청에서 반환되는 파일 형식 제한(cos:response-content-type)

요청 매개변수 response-content-type

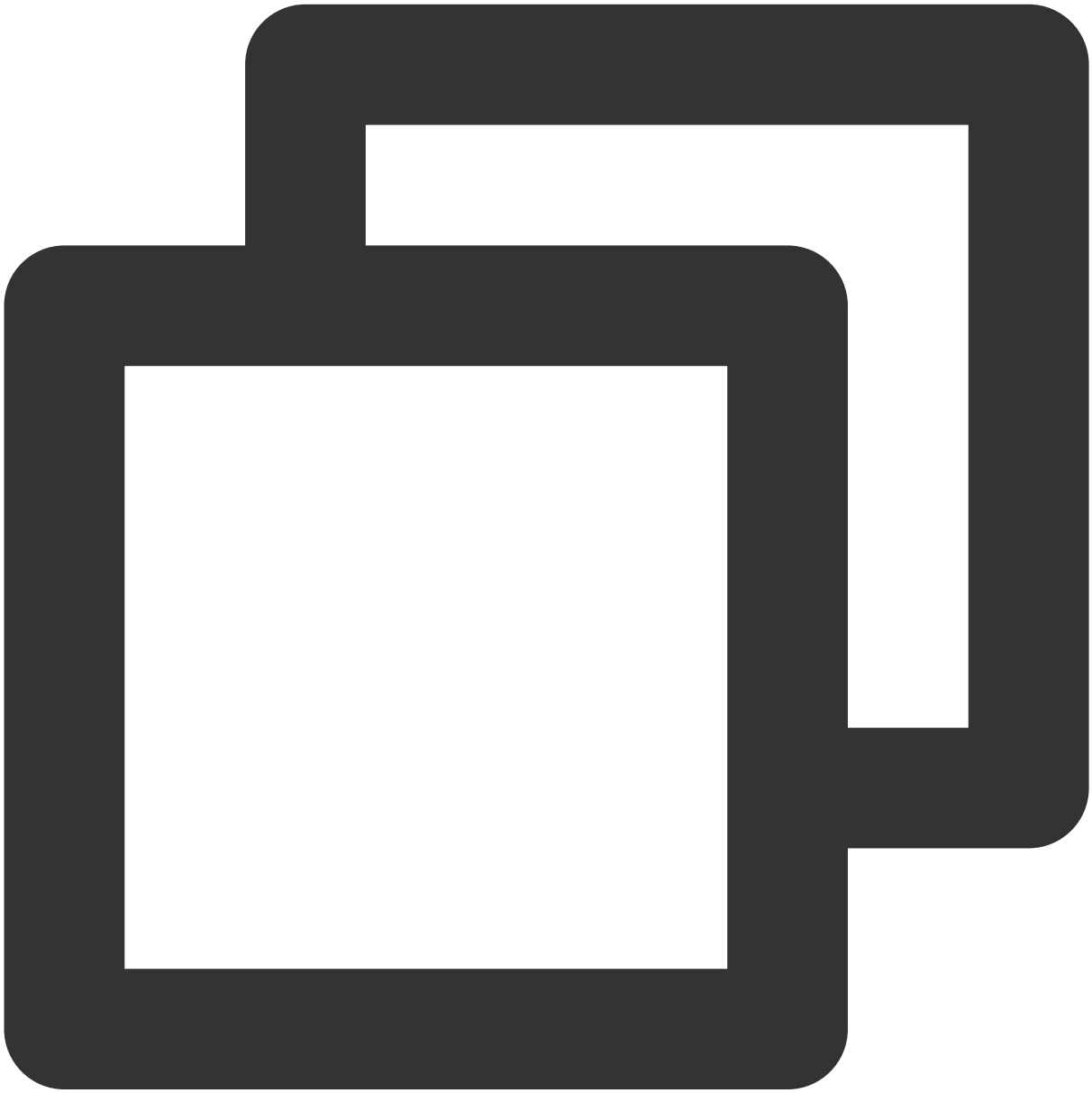
GetObject API를 사용하면 요청 매개변수인 `response-content-type` 을 추가하여 응답의 Content-Type 헤더 값을 지정할 수 있습니다.

조건 키 cos:response-content-type

`cos:response-content-type` 조건 키를 사용하여 요청이 요청 매개변수인 `response-content-type` 을 전달해야 하는지 여부를 지정할 수 있습니다.

예시1: GetObject 요청 매개변수 response-content-type을 'image/jpeg'로 제한

버킷 examplebucket-1250000000을 소유하고 uin이 100000000001인 루트 계정이 다음 버킷 정책을 사용하여 uin이 100000000002인 서브 사용자가 시작한 GetObject 요청이 “image/jpeg” 값과 함께 response-content-type 요청 매개변수를 전달해야 한다고 가정합니다. `response-content-type` 매개변수는 요청 매개변수이며 요청이 시작될 때 urlencode가 필요합니다(urlencode된 값: `response-content-type=image%2Fjpeg`). 따라서 Policy를 설정할 때 "image/jpeg"도 인코딩(urlencode)해야 하며 "image%2Fjpeg"를 입력해야 합니다.



```
{
    "version":"2.0",
    "statement":[
        {
```

```
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
                "name/cos:GetObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "string_equal":{
                    "cos:response-content-type":"image%2Fjpeg"
                }
            }
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "name/cos:GetObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "string_not_equal_if_exist":{
                    "cos:response-content-type":"image%2Fjpeg"
                }
            }
        }
    ]
}
```

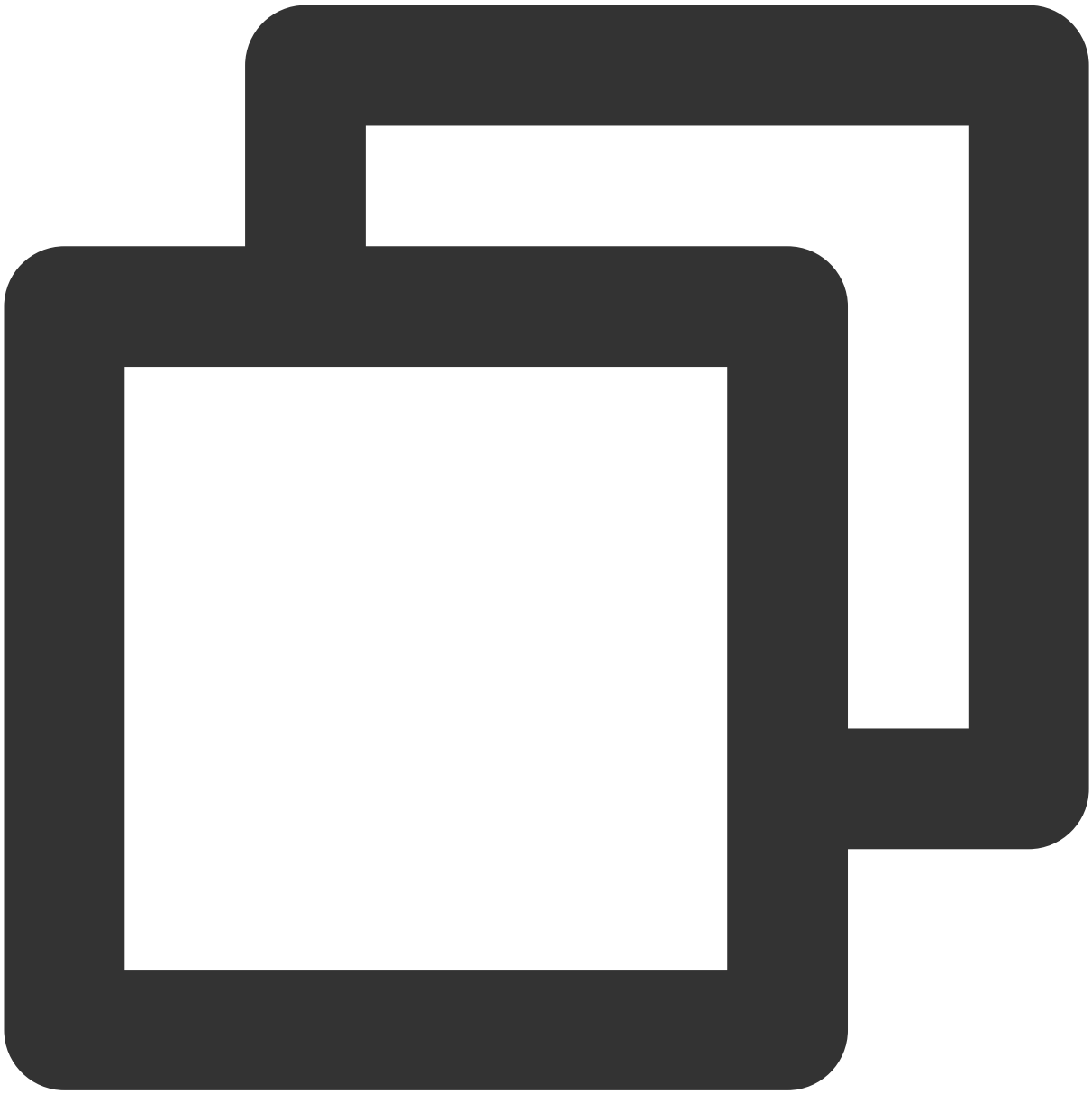
HTTPS 요청만 허용(cos:secure-transport)

조건 키 cos:secure-transport

조건 키 `cos:secure-transport` 를 사용하여 HTTPS 프로토콜 사용 요청 제한

예시1: HTTPS를 사용하도록 다운로드 요청 제한

버킷 examplebucket-1250000000을 소유하고 uin이 100000000001인 루트 계정이 다음 버킷 정책을 사용하여 uin이 100000000002인 서브 계정에서 보낸 HTTPS 기반 GetObject 요청만 허용한다고 가정합니다.

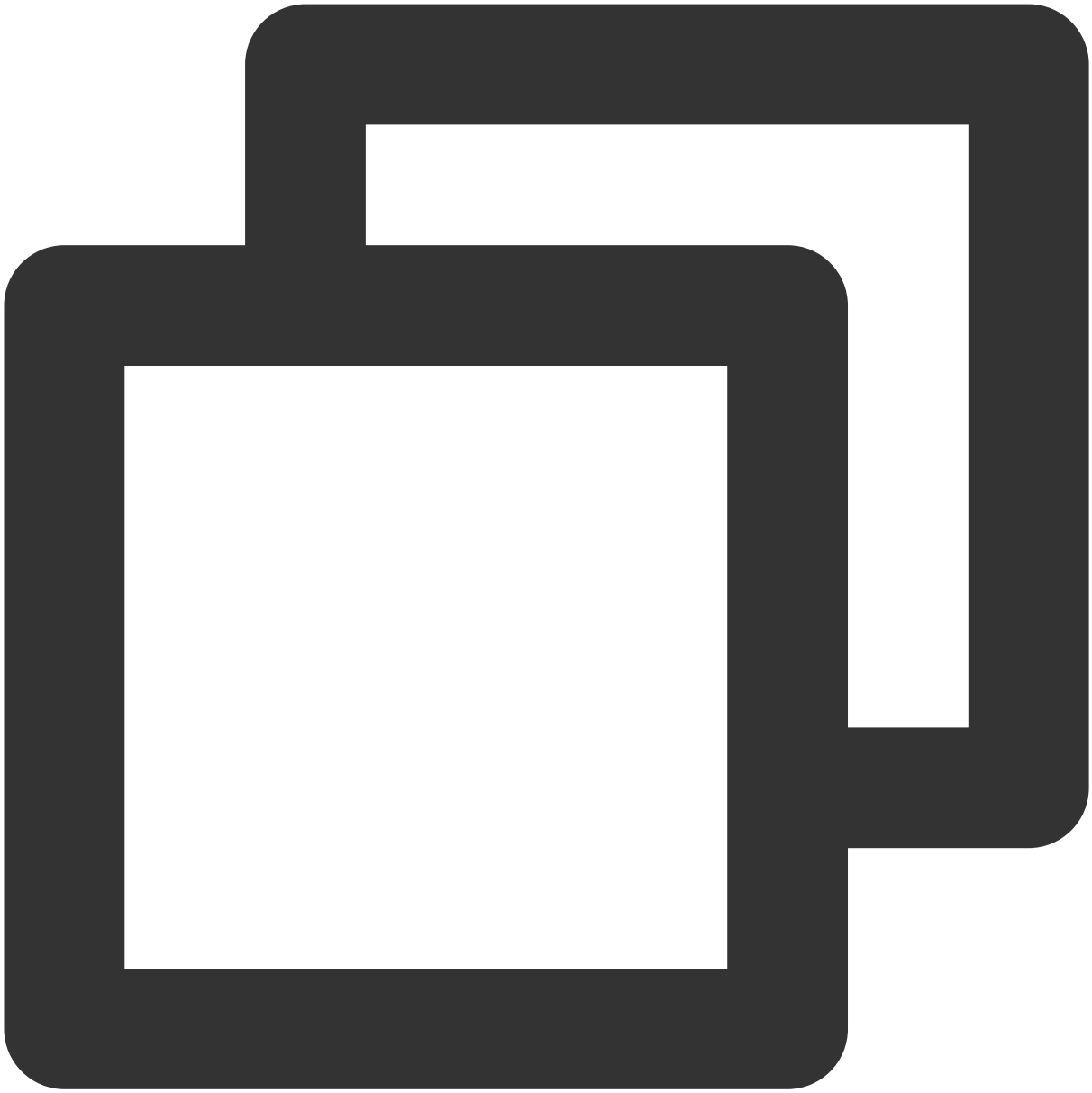


```
{
    "version":"2.0",
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
```

```
            },
            "effect":"allow",
            "action":[
                "name/cos:GetObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "bool_equal":{
                    "cos:secure-transport":"true"
                }
            }
        }
    ]
}
```

예시2: 비 HTTPS 요청 거부

버킷 examplebucket-1250000000을 소유하고 uin이 100000000001인 루트 계정이 다음 버킷 정책을 사용하여 uin이 1000000000002인 서브 계정에서 보낸 비 HTTPS 요청을 거부한다고 가정합니다.



```
{
    "version":"2.0",
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
```

```
                "*"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "bool_equal":{
                    "cos:secure-transport":"false"
                }
            }
        }
    ]
}
```

지정된 스토리지 클래스 설정 허용(cos:x-cos-storage-class)

요청 헤더 x-cos-storage-class

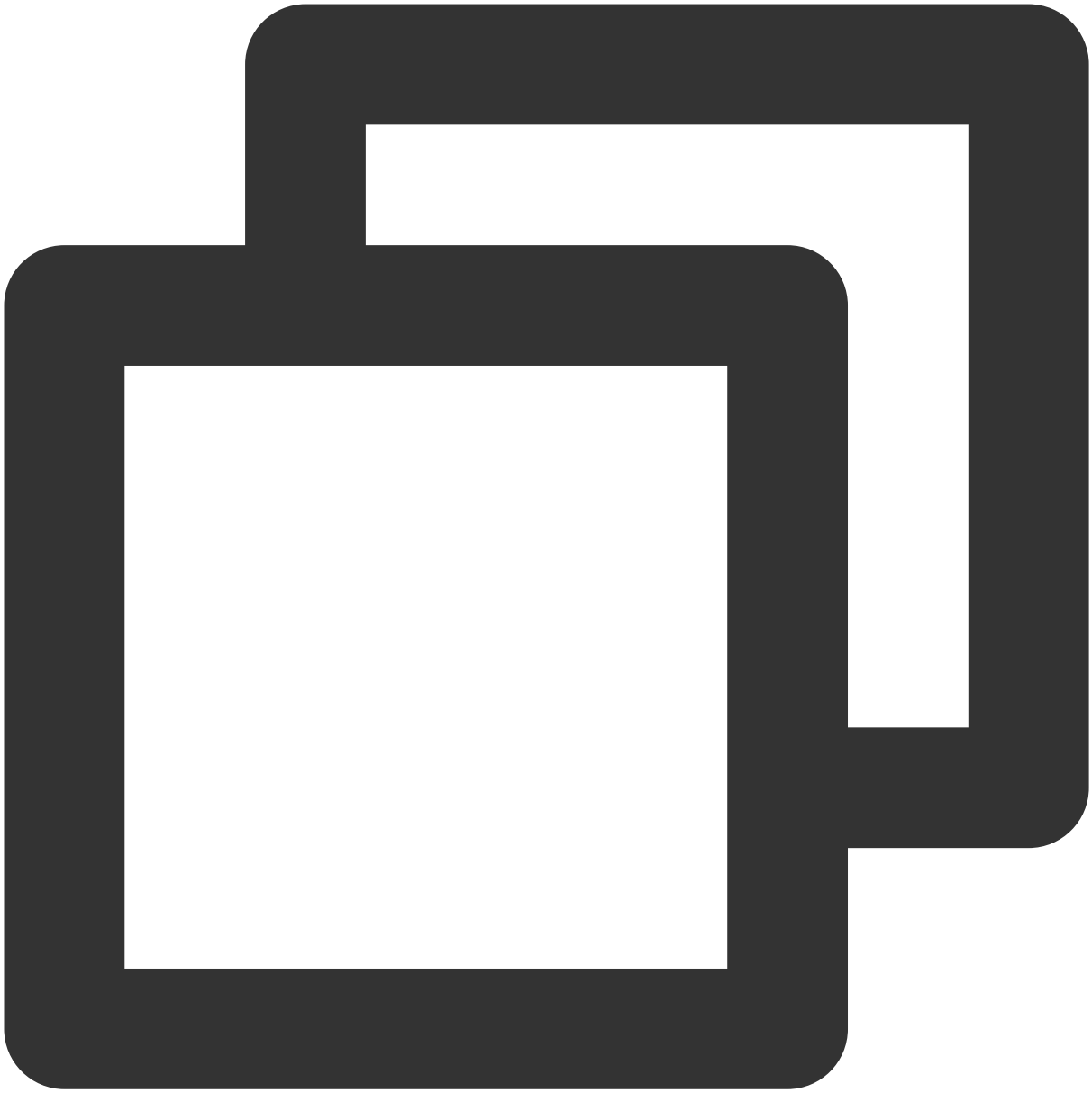
요청 헤더 `x-cos-storage-class` 를 사용하여 객체를 업로드할 때 객체의 스토리지 클래스를 지정하거나 수정할 수 있습니다.

조건 키 cos:x-cos-storage-class

조건 키 `cos:x-cos-storage-class` 를 사용하여 요청 헤더 `x-cos-storage-class` 를 제한하여 스토리지 클래스 수정 요청을 제한할 수 있습니다.
COS의 스토리지 클래스 필드에는 STANDARD, MAZ_STANDARD, STANDARD_IA, MAZ_STANDARD_IA, INTELLIGENT_TIERING, MAZ_INTELLIGENT_TIERING, ARCHIVE 및 DEEP_ARCHIVE가 있습니다.

예시1: PutObject 요청 시 스토리지 클래스를 반드시 STANDARD로 설정

버킷 examplebucket-1250000000을 소유하는 uin이 100000000001인 루트 계정이 다음 버킷 정책을 사용하여 값이 STANDARD인 x-cos-storage-class 헤더를 전달하기 위해 uin이 100000000002인 서브 계정에서 보낸 PutObject 요청을 제한한다고 가정합니다.



```
{
    "version":"2.0",
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
```

```
                "name/cos:PutObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "string_equal":{
                    "cos:x-cos-storage-class":"STANDARD"
                }
            }
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "name/cos:PutObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "string_not_equal_if_exist":{
                    "cos:x-cos-storage-class":"STANDARD"
                }
            }
        }
    ]
}
```

지정된 버킷/객체 ACL 설정 허용(cos:x-cos-acl)

요청 헤더 x-cos-acl

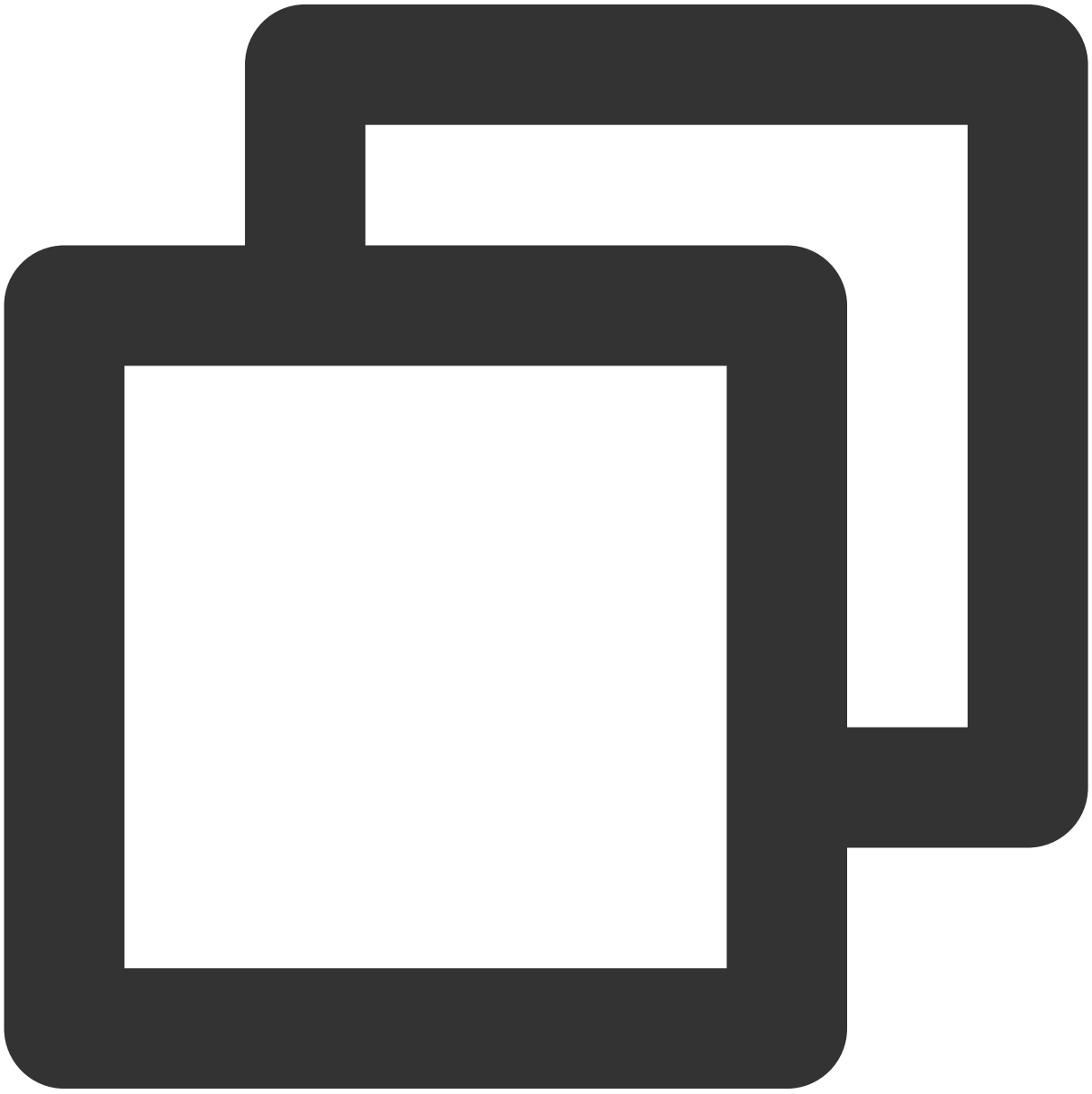
객체를 업로드하거나 버킷을 생성할 때 요청 헤더 `x-cos-acl` 을 사용하여 ACL을 지정하거나 객체 또는 버킷 ACL을 수정할 수 있습니다. ACL에 대한 자세한 내용은 ACL을 참고하십시오.
버킷용 사전 설정 ACL: `private` , `public-read` , `public-read-write` , `authenticated-read` .
객체에 대한 사전 설정 ACL: `default` , `private` , `public-read` , `authenticated-read` , `bucket-owner-read` , `bucket-owner-full-control` .

조건 키 cos:x-cos-acl

조건 키 `cos:x-cos-acl` 을 사용하여 요청 헤더 `x-cos-acl` 을 제한하고 객체/버킷 ACL 수정 요청을 제한할 수 있습니다.

예시1: PutObject 요청에서 객체 ACL을 비공개로 설정 필요

examplebucket-1250000000 버킷을 소유하는 uin 100000000001의 루트 계정이 다음 버킷 정책을 사용하여 uin이 100000000002인 서브 계정을 제한하여 프라이빗 객체만 업로드한다고 가정합니다. 정책은 모든 PutObject 요청이 `private` 값을 가진 x-cos-acl 헤더를 전달하도록 요구합니다.



```
{
    "version":"2.0",
```

```
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
                "name/cos:PutObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "string_equal":{
                    "cos:x-cos-acl":"private"
                }
            }
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "name/cos:PutObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "string_not_equal_if_exist":{
                    "cos:x-cos-acl":"private"
                }
            }
        }
    ]
}
```

지정된 디렉터리에서만 객체 나열 허용(cos:prefix)

조건 키 cos:prefix

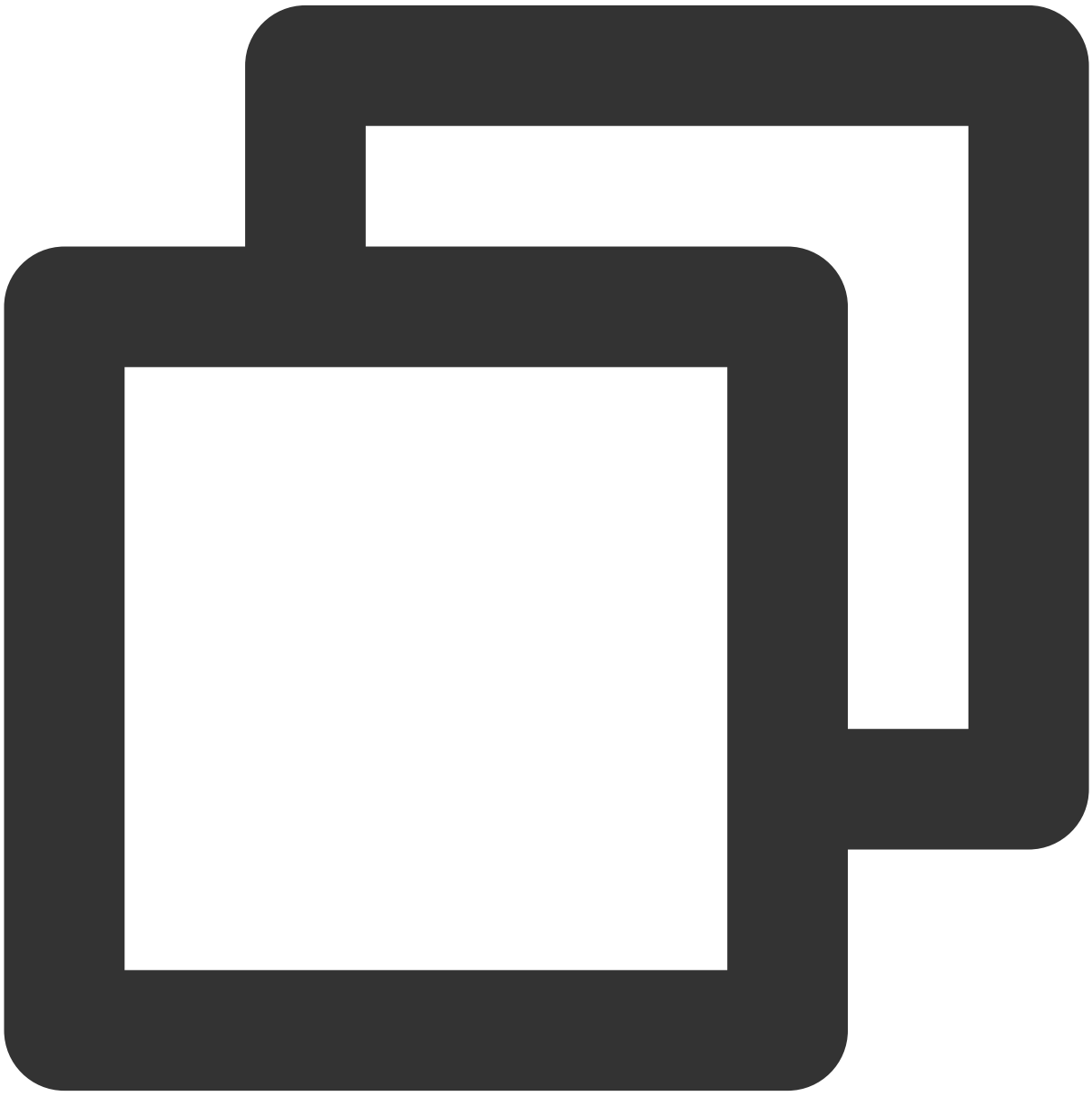
조건 키 `cos:prefix` 를 사용하여 요청 매개변수 prefix를 제한할 수 있습니다.

주의 :

prefix 값에 중국어 및 `/` 와 같은 특수 문자가 포함된 경우 버킷 정책에 쓰기 전에 값을 인코딩(urlencode)해야 합니다.

예시1: 버킷의 지정된 디렉터리에 있는 객체만 나열 허용

버킷 examplebucket-1250000000을 소유하는 uin이 100000000001인 루트 계정이 다음 버킷 정책을 사용하여 버킷의 folder1 디렉터리에 있는 객체만 나열하도록 uin이 100000000002인 서브 계정을 제한한다고 가정합니다. 정책에서는 모든 GetBucket 요청이 값 “folder1”과 함께 prefix 매개변수를 전달해야 합니다.



```
{
```

```
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
                "name/cos:GetBucket"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "string_equal":{
                    "cos:prefix":"folder1"
                }
            }
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "name/cos:GetBucket"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "string_equal_if_exist":{
                    "cos:prefix":"folder1"
                }
            }
        }
    ],
    "version":"2.0"
}
```

지정된 버전의 TLS 프로토콜만 사용 허용(cos:tls-version)

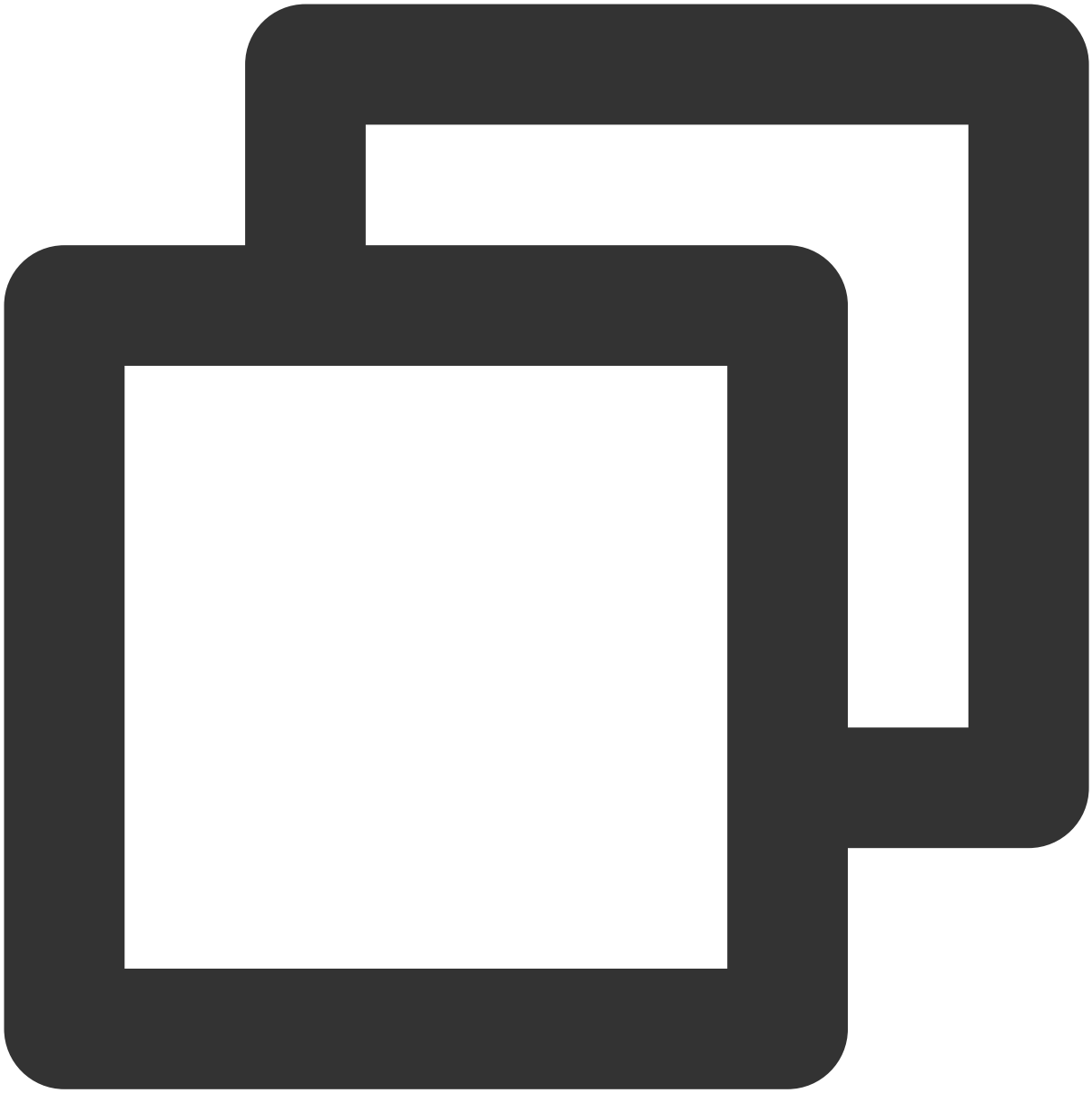
조건 키 cos:tls-version

조건 키 `cos:tls-version` 을 사용하여 HTTPS 요청의 TLS 버전을 제한할 수 있습니다. 해당 값은 Numric 유형이며 1.0, 1.1 또는 1.2와 같은 부동 소수점을 지원합니다.

예시1: TLS v1.2를 사용하는 HTTP 요청만 승인

요청 시나리오	예상 결과
TLS v1.0을 사용한 HTTPS 요청	403, 실패
TLS v1.2를 사용한 HTTPS 요청	200, 성공

정책 예시는 다음과 같습니다.



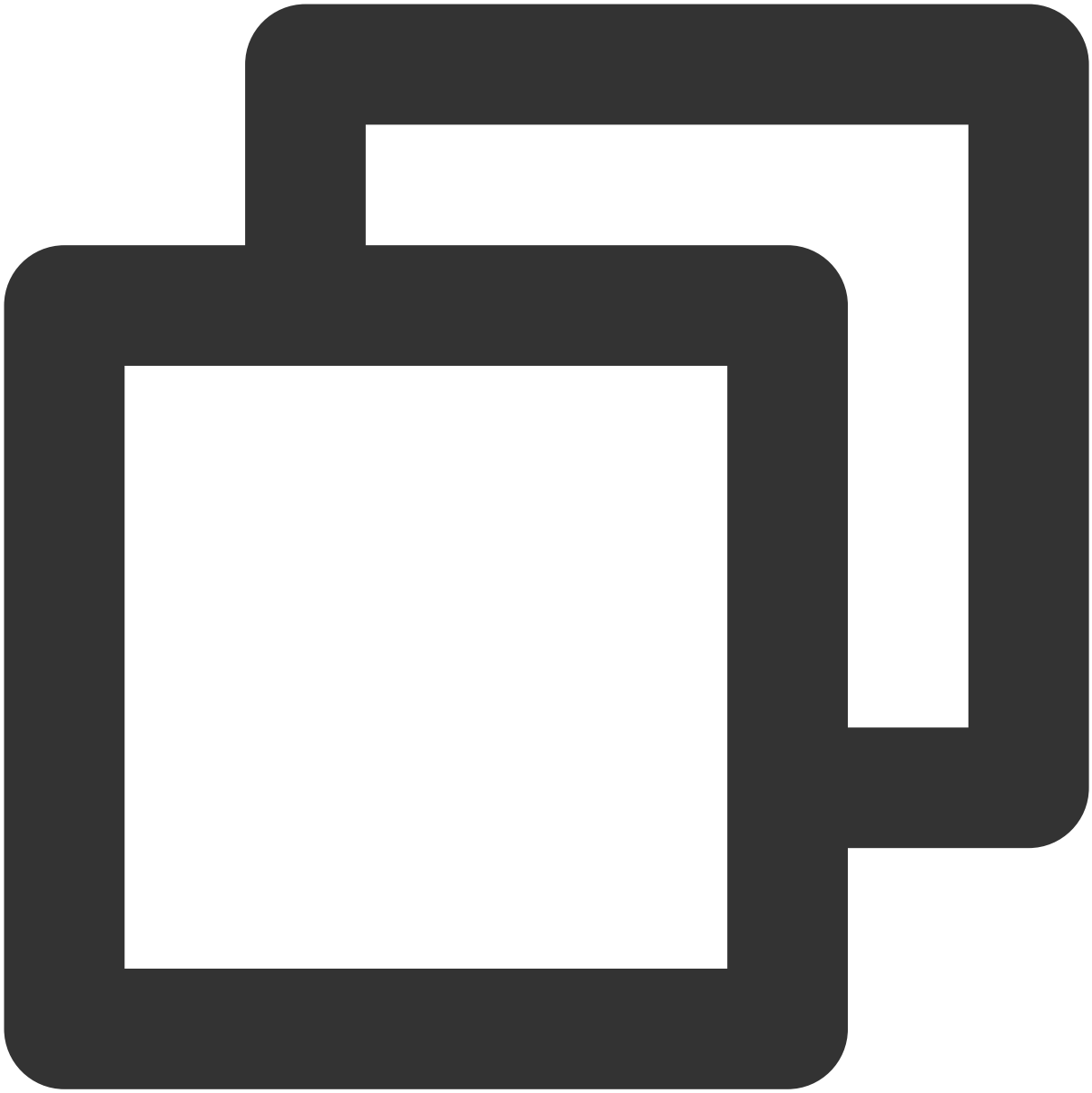
```
{
    "version":"2.0",
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
```

```
                "*"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "numeric_equal":{
                    "cos:tls-version":1.2
                }
            }
        }
    ]
}
```

예시2: v1.2 이전의 TLS를 사용하는 HTTP 요청 거부

요청 시나리오	예상 결과
TLS v1.0을 사용한 HTTPS 요청	403, 실패
TLS v1.2를 사용한 HTTPS 요청	200, 성공

정책 예시는 다음과 같습니다.



```
{
    "version":"2.0",
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
```

```
                "*"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "numeric_greater_than_equal":{
                    "cos:tls-version":1.2
                }
            }
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "*"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "numeric_less_than_if_exist":{
                    "cos:tls-version":1.2
                }
            }
        }
    ]
}
```

버킷 생성 시 지정된 버킷 태그 강제 설정(qcs:request_tag)

설명 :

request_tag 조건 키는 PutBucket 및 PutBucketTagging 작업에만 적용되며 GetService, PutObject 또는 PutObjectTagging에는 적용되지 않습니다.

조건 키 qcs:request_tag

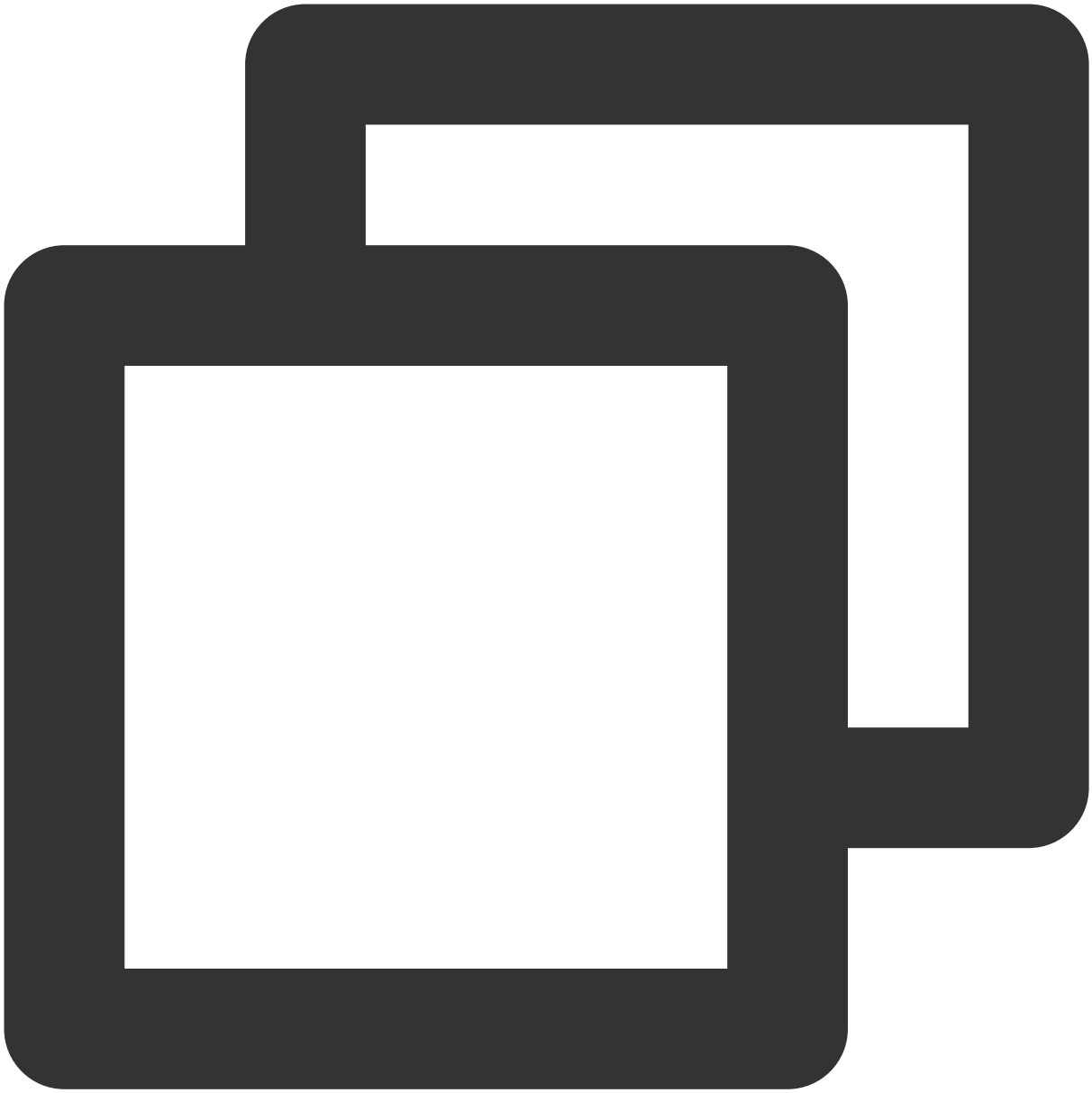
조건 키 `qcs:request_tag` 를 사용하여 PutBucket 또는 PutBucketTagging 요청을 시작할 때 사용자가 지정된 버킷 태그를 포함하도록 제한할 수 있습니다.

예시: 사용자가 버킷을 생성할 때 지정된 버킷 태그를 포함하도록 제한

많은 사용자가 버킷 태그를 사용하여 버킷을 관리할 수 있습니다. 다음 정책 예시는 사용자가 버킷 생성 시 지정된 버킷 태그 `<a,b>` 및 `<c,d>` 를 설정한 후에만 권한을 얻을 수 있음을 나타냅니다.
여러 버킷 태그를 설정할 수 있습니다. 각기 다른 버킷 태그 키/값 및 태그 수량은 각각 다른 조합으로 사용됩니다. 요청 양식 세트 A에서 사용자가 여러 매개변수 값을 가지고 있고 조건 양식 세트 B에 지정된 여러 매개변수 값을 가지고 있다고 가정합니다. 이 조건 키를 통해 사용자는 for_any_value 및 for_all_value 한정어의 다양한 조합을 사용하여 다른 의미를 나타낼 수 있습니다.
`for_any_value:string_equal` 은 A와 B가 교차하는 경우 요청이 적용됨을 나타냅니다.
`for_all_value:string_equal` 은 A가 B의 subset인 경우 요청이 적용됨을 나타냅니다.
`for_any_value:string_equal` 을 사용하는 경우 해당 정책 및 요청은 다음과 같습니다.

요청 시나리오	예상 결과
PutBucket, 요청 헤더 `x-cos-tagging: a=b&c=d`	200, 성공
PutBucket, 요청 헤더 `x-cos-tagging: a=b`	200, 성공
PutBucket, 요청 헤더 `x-cos-tagging: a=b&c=d&e=f`	200, 성공

정책 예시는 다음과 같습니다.



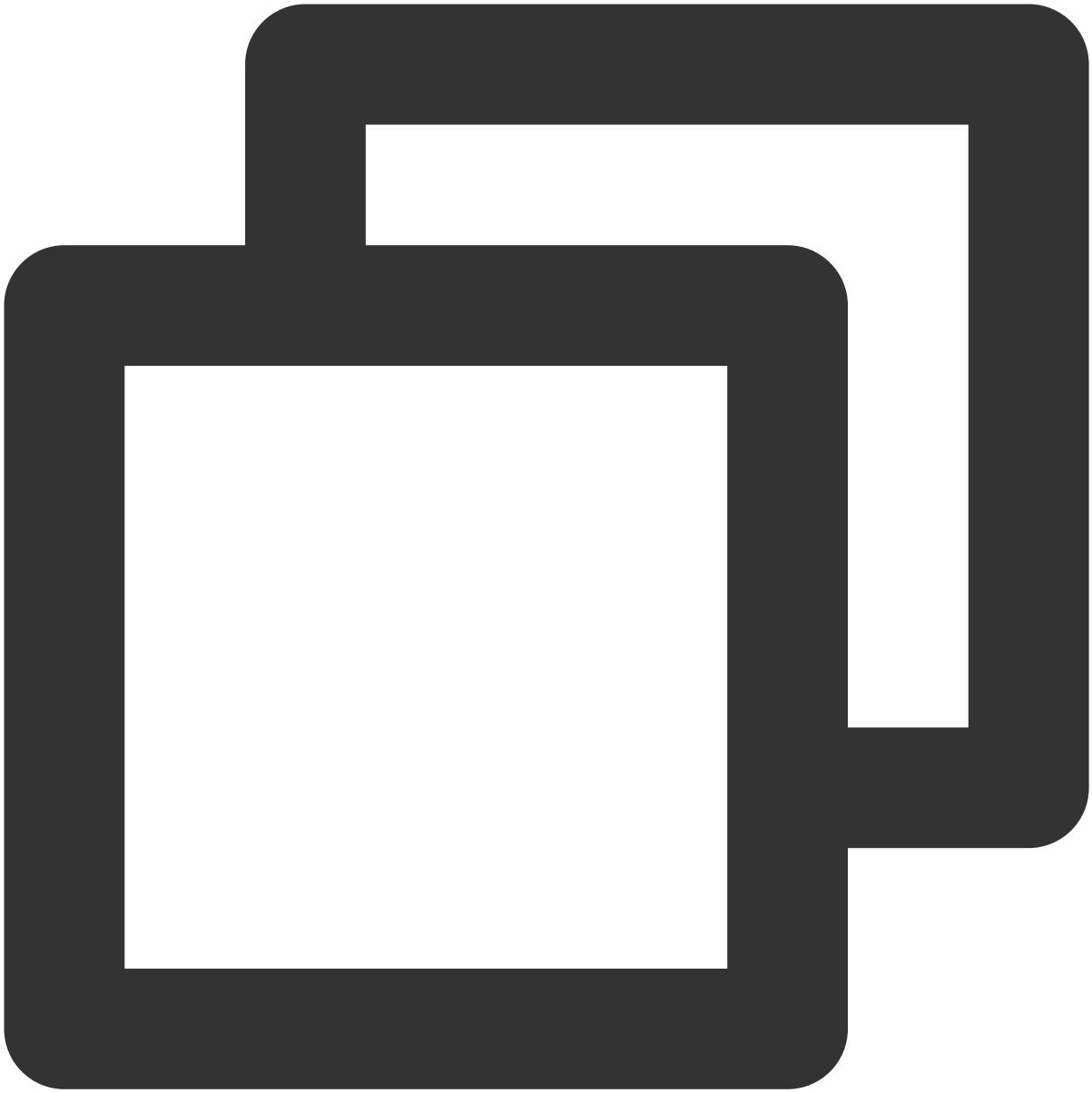
```
{
    "version": "2.0",
    "statement": [
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect": "allow",
            "action":[
```

```
                "name/cos:PutBucket"
            ],
            "resource": "*",
            "condition":{
                "for_any_value:string_equal":{
                    "qcs:request_tag": [
                        "a&b",
                        "c&d"
                    ]
                }
            }
        }
    ]
}
```

`for_all_value:string_equal` 을 사용하는 경우 해당 정책 및 요청은 다음과 같습니다.

요청 시나리오	예상 결과
PutBucket, 요청 헤더 `x-cos-tagging: a=b&c=d`	200, 성공
PutBucket, 요청 헤더 `x-cos-tagging: a=b`	200, 성공
PutBucket, 요청 헤더 `x-cos-tagging: a=b&c=d&e=f`	403, 실패

정책 예시는 다음과 같습니다.



```
{
    "version": "2.0",
    "statement": [
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect": "allow",
            "action":[
```

```
                "name/cos:PutBucket"
            ],
            "resource": "*",
            "condition":{
                "for_all_value:string_equal": {
                    "qcs:request_tag": [
                        "a&b",
                        "c&d"
                    ]
                }
            }
        }
    ]
}
```

다른 루트 계정에 있는 서브 계정에 버킷 권한 부여

최종 업데이트 날짜: : 2024-06-24 16:44:04

작업 시나리오

루트 계정 A(APPID: 1250000000)에는 `examplebucket1-1250000000` 및 `examplebucket2-1250000000` 이라는 두 개의 버킷이 있으며 루트 계정 B의 서브 계정 B0은 비즈니스 요구 사항을 충족하기 위해 조작하려고 합니다. 이 문서에서는 그렇게 할 수 있는 권한을 부여하는 방법을 설명합니다.

작업 단계

루트 계정 B에 A 계정이 소유한 버킷에 대한 권한 부여

1. 루트 계정 A로 COS 콘솔에 로그인합니다.
2. **버킷 리스트**를 클릭하고 인증할 버킷을 찾은 다음 이름을 클릭하여 버킷 세부 정보 페이지로 들어갑니다.
3. 왼쪽 사이드바에서 **권한 관리**를 클릭하여 버킷의 권한 관리 페이지로 이동합니다.
4. **Policy 권한 설정** 구성 항목을 찾아 **정책 추가**를 클릭하고 사용자 지정 정책을 선택한 후 **다음**을 클릭합니다.
5. 아래와 같이 양식을 작성합니다.

정책 ID: 선택 사항입니다.
효과: 허용
사용자: 사용자 추가를 클릭하고 사용자 유형을 루트 계정으로 선택한 후 계정 ID에 루트 계정 B의 UIN을 입력합니다. 예시: 100000000002.
리소스: 필요에 따라 선택합니다. 기본값은 버킷 전체입니다.
리소스 경로: 리소스 지정 시에만 작성하며, 필요에 따라 작성합니다.
작업: 작업 추가를 클릭하고 모든 작업을 선택합니다. 일부 작업만 권한을 부여할 경우 실제 필요한 작업 한 개 또는 여러 개를 선택할 수 있습니다.
조건: 필요에 따라 작성하며, 필요 없는 경우 비워 둘 수 있습니다.

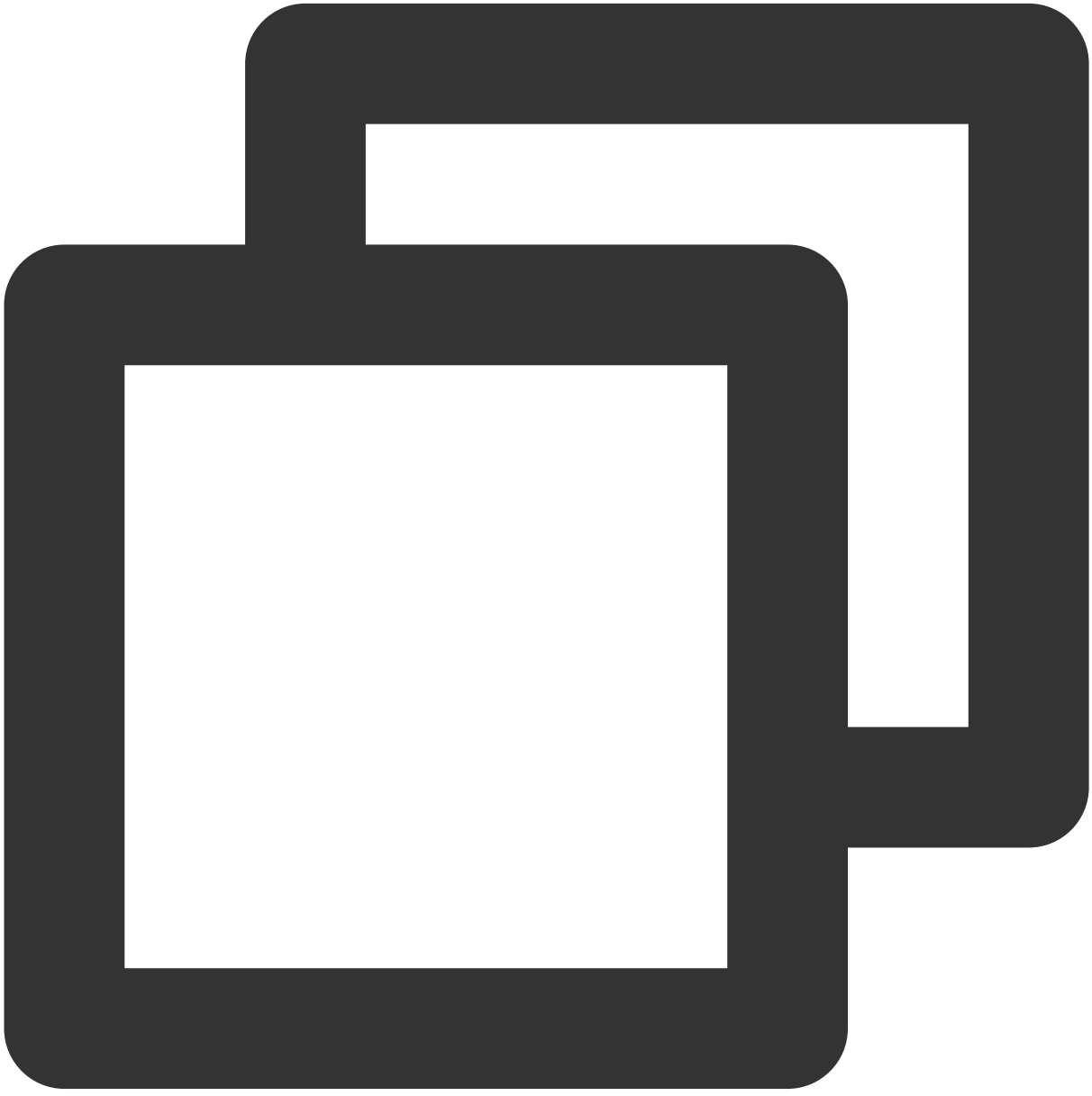
6. **완료**를 클릭하여 루트 계정 B가 버킷에 대해 지정한 권한을 부여합니다.
7. 기타 버킷에 대한 권한 부여가 필요한 경우 이상의 순서를 다시 반복합니다.

서브 계정 B0에 A 계정이 소유한 버킷에 대한 권한 부여

1. 루트 계정 B를 사용하여 CAM 콘솔에 로그인한 후 정책 페이지로 이동합니다.
2. **사용자 정의 정책 생성 > 정책 구문으로 생성**을 선택하고 빈 템플릿을 선택한 후 **다음**을 클릭합니다.

설명 :

루트 계정 B에서 서브 계정 B0에 권한을 부여합니다. 사용자 정의 정책을 통해서만 권한을 부여할 수 있으며, 사전 설정 정책을 통한 권한 부여는 지원하지 않습니다.
3. 다음과 같이 작성합니다.
정책 이름: 중복되지 않고 의미가 있는 정책 이름을 사용자 정의합니다. 예: cos-child-account
비고: 선택 가능하며, 직접 작성합니다.
정책 내용:



```
{
 "version": "2.0",
 "statement": [
```

```
    {
        "action": "cos:*",
        "effect": "allow",
              "resource": [
                  "qcs::cos::uid/1250000000:examplebucket1-1250000000/*",
                  "qcs::cos::uid/1250000000:examplebucket2-1250000000/*"
              ]
    }
  ]
}
```

위의 정책은 루트 계정 B에 작업 권한이 있는 A 명의로 된 버킷을 모두 서브 계정 B0에 인가하는 것을 의미합니다. 여기서, `uid/1250000000` 중의 1250000000는 루트 계정 A의 APPID이고, `examplebucket1-1250000000` 및 `examplebucket2-1250000000` 은 루트 계정 B에 작업 권한이 있는 A 명의로 된 버킷입니다.

4. **완료**를 클릭하면 정책 생성이 완료됩니다.
5. **정책** 리스트에서 생성된 정책을 찾아 오른쪽 **사용자/사용자 그룹/역할 연결**을 클릭합니다.
6. 팝업창에서 서브 계정 B0을 선택하고 **확인**을 클릭합니다.
7. 권한 부여 작업이 완료됩니다. 이제 서브 계정 B0의 키를 사용하여 A 소유의 버킷에 대한 작업을 테스트해 볼 수 있습니다.

성능 최적화
요청률 및 성능 최적화

최종 업데이트 날짜: : 2024-06-24 16:44:04

주의 :

현재 COS는 하위 인덱스 분산 메커니즘을 통해 높은 QPS를 구현했으며 더 높은 성능의 QPS가 필요한 경우 고객센터로 문의할 수 있습니다. 파일을 정리하는 일상적인 프로세스에서 여전히 이 문서의 요구 사항을 따르고 지나치게 중앙 집중화된 인덱스 저장소를 피하는 것이 좋습니다.

소개

본문은 Tencent Cloud Cloud Object Storage(COS)에서 요청 속도 성능 최적화를 위한 모범 사례를 설명합니다. Tencent Cloud COS는 초당 30000 PUT 요청 또는 초당 30000 GET 요청의 일반적인 워크로드 용량을 지원합니다. 워크로드가 위의 임계값을 초과하는 경우 이 가이드에 따라 요청 속도 성능을 확장하고 최적화할 수 있습니다.

설명 :

요청 로드는 동시 연결 수가 아닌 초당 시작된 요청 수를 나타냅니다. 즉, 문제 없이 수천 개의 기존 연결을 유지하면서 초당 수백 개의 새로운 연결 요청을 보낼 수 있습니다.

Tencent Cloud COS는 더 높은 요청률을 제공하기 위해 성능 확장을 지원합니다. GET 요청 로드가 높은 경우 Tencent Cloud CDN 제품과 함께 COS를 사용하는 것이 좋습니다. 자세한 내용은 도메인 관리를 참고하십시오. 버킷의 전체 요청 속도가 초당 30000 PUT/LIST/DELETE 요청을 초과할 것으로 예상되는 경우 고객센터로 문의하여 워크로드에 대비하고 요청 제한을 피할 수 있습니다.

설명 :

가끔 초당 30000에 도달하고 버스트 중에 초당 30000을 초과하지 않는 혼합 요청 로드가 있는 경우 이 가이드를 무시할 수 있습니다.

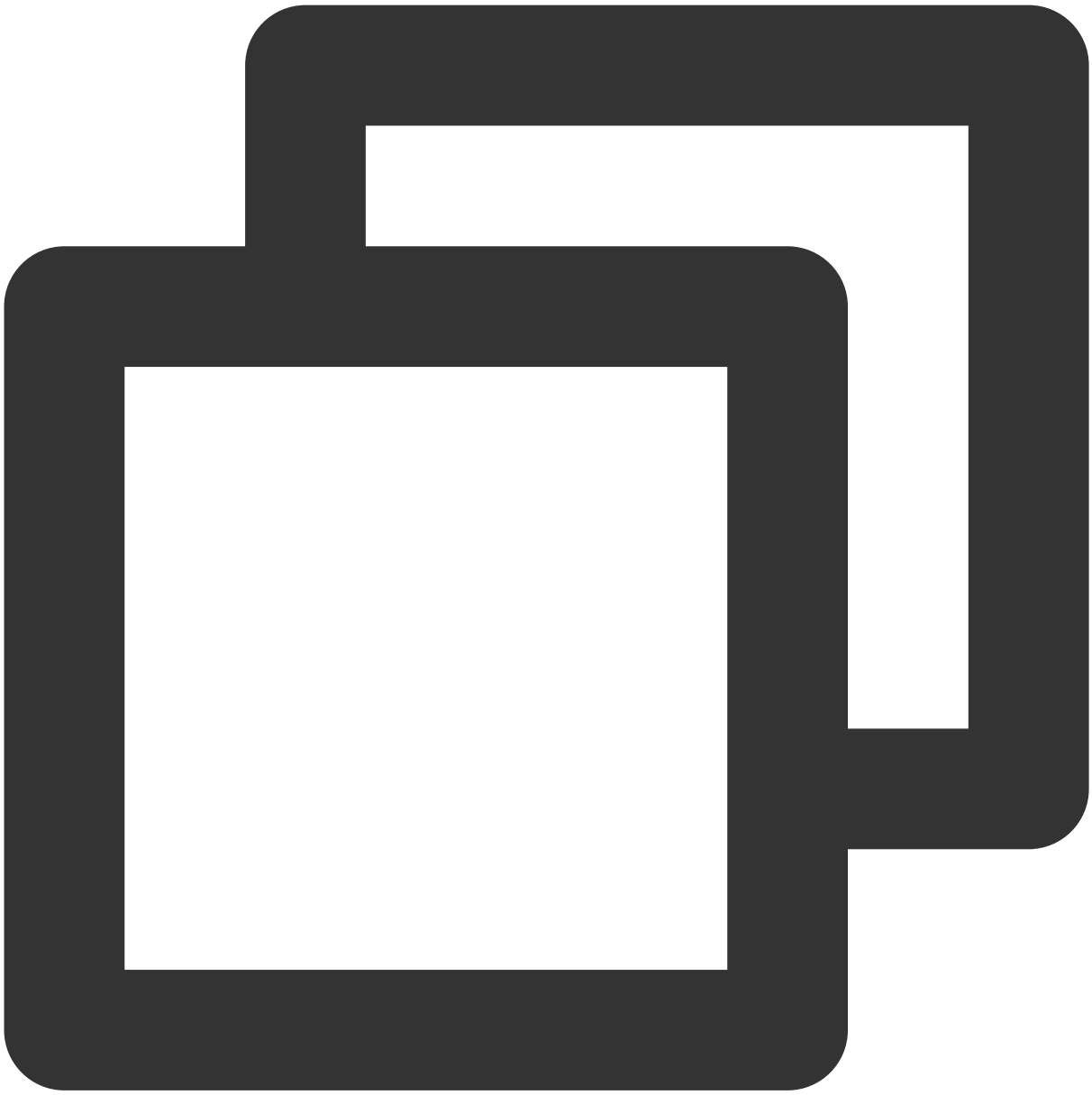
실행 순서

혼합 요청 로드

대량의 객체를 업로드하는 경우, 선택한 객체 키의 성능에 문제가 생길 수 있습니다. 다음은 Tencent Cloud COS가 Object 키 값을 저장하는 방법을 간략히 소개합니다.

Tencent Cloud는 COS의 모든 서비스 리전에 버킷(Bucket)과 객체(Object)의 키 값을 인덱스로 유지하고 있습니다. 객체 키는 UTF-8 이진법의 순서로 저장됩니다. 이처럼 많은 키 값으로 인해 타임스탬프나 알파벳 순서 등을 사용하면 키 값이 위치한 파티션의 읽기/쓰기 성능 용량이 소모될 수 있습니다. 버킷 경로 `examplebucket-`

`1250000000.cos.ap-beijing.myqcloud.com` 을 예시로 인덱스 성능 용량을 소모할 수 있는 몇 가지 경우는 다음과 같습니다.



```
20170701/log120000.tar.gz
20170701/log120500.tar.gz
20170701/log121000.tar.gz
20170701/log121500.tar.gz
...
image001/indexpage1.jpg
image002/indexpage2.jpg
image003/indexpage3.jpg
```

```
...
```

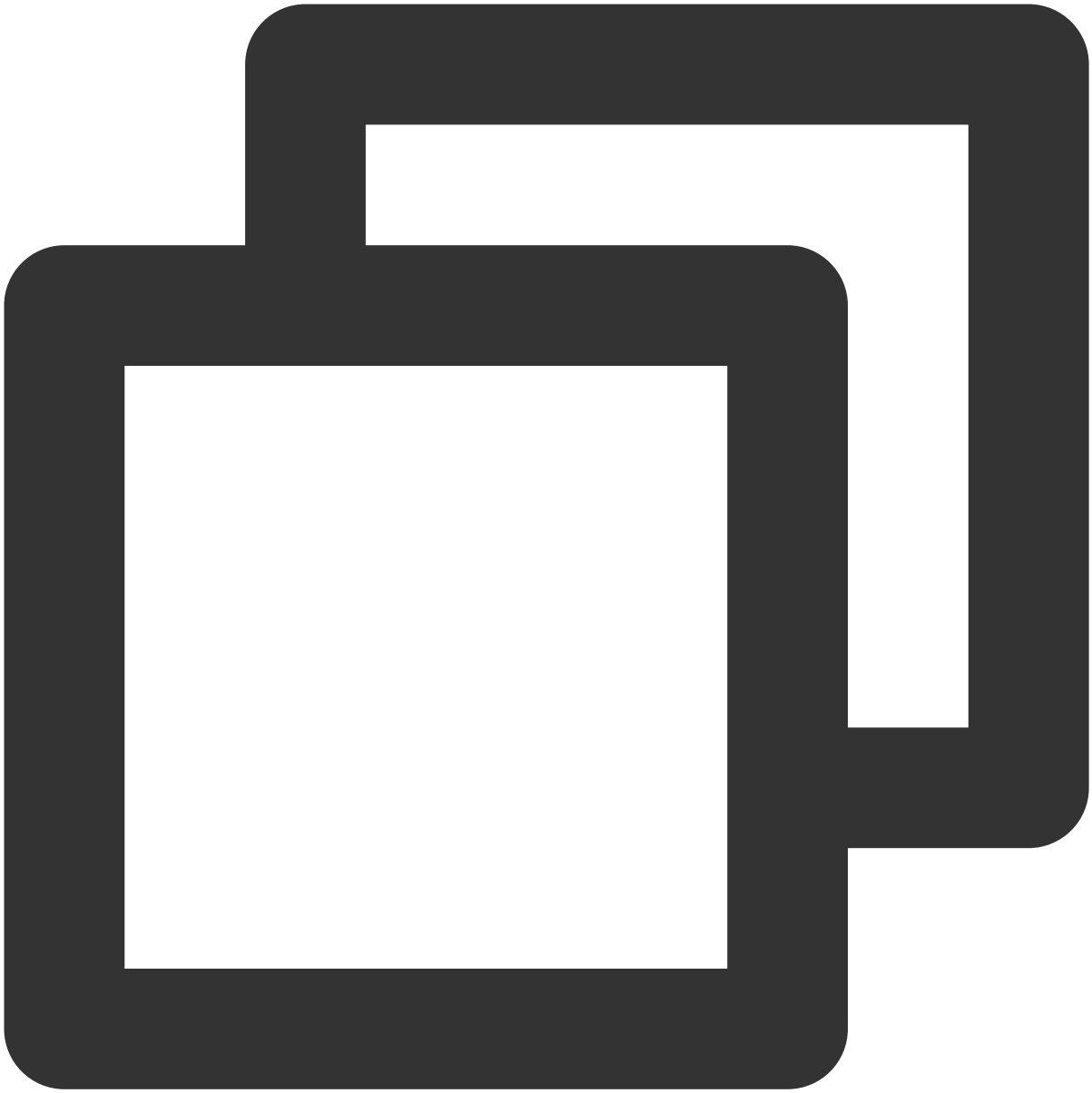
일반적인 워크로드가 초당 30000개 요청을 초과하는 경우 위의 경우와 같이 순차 키 값을 사용하지 않아야 합니다. 일련 번호, 날짜 또는 시간 값과 같은 문자를 객체 키로 사용해야 하는 경우 키 이름에 임의의 접두사를 추가하여 여러 인덱스 파티션의 키 값을 관리하고 중앙 집중식 로드 성능을 개선할 수 있습니다. 다음은 키 값에 임의의 요소를 추가하는 몇 가지 방법입니다.

주의 :

단일 버킷의 액세스 성능을 향상시키기 위해 다음 방법을 모두 사용할 수 있습니다. 업무의 일반적인 부하가 초당 30000 요청을 초과하는 경우에도 다음 방법을 수행하는 동안에도 미리 업무 부하에 대비하기 위해 고객센터로 문의하십시오.

16진법 해시 접두사 추가

객체 키에 랜덤성을 높이는 가장 직접적인 방법은 키 이름의 맨 앞에 해시 문자열 접두사를 추가하는 것입니다. 예를 들어 객체를 업로드할 때 경로 키 값의 SHA1 또는 MD5 해시를 계산하고 키 이름에 접두사로 몇 개의 문자를 추가합니다. 일반적으로 길이가 2 - 4자인 해시 접두사로 충분합니다.



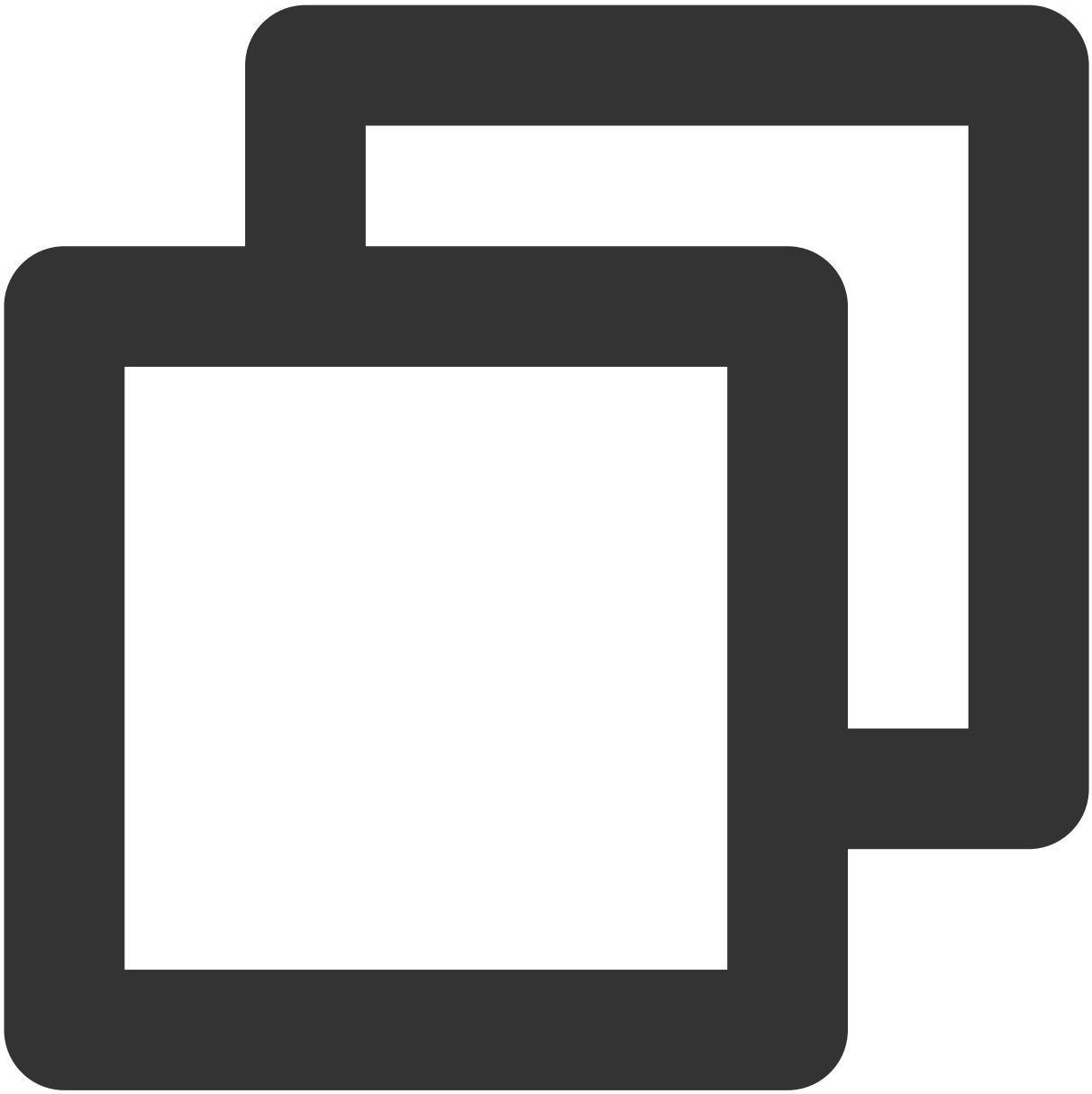
```
faf1-20170701/log120000.tar.gz
e073-20170701/log120500.tar.gz
333c-20170701/log121000.tar.gz
2c32-20170701/log121500.tar.gz
```

주의 :

Tencent Cloud COS의 키 값은 UTF-8 바이너리 순서로 인덱싱되므로 원래의 완전한 20170701 접두사 구조를 얻으려면 65536 GET Bucket 작업을 시작해야 할 수 있습니다.

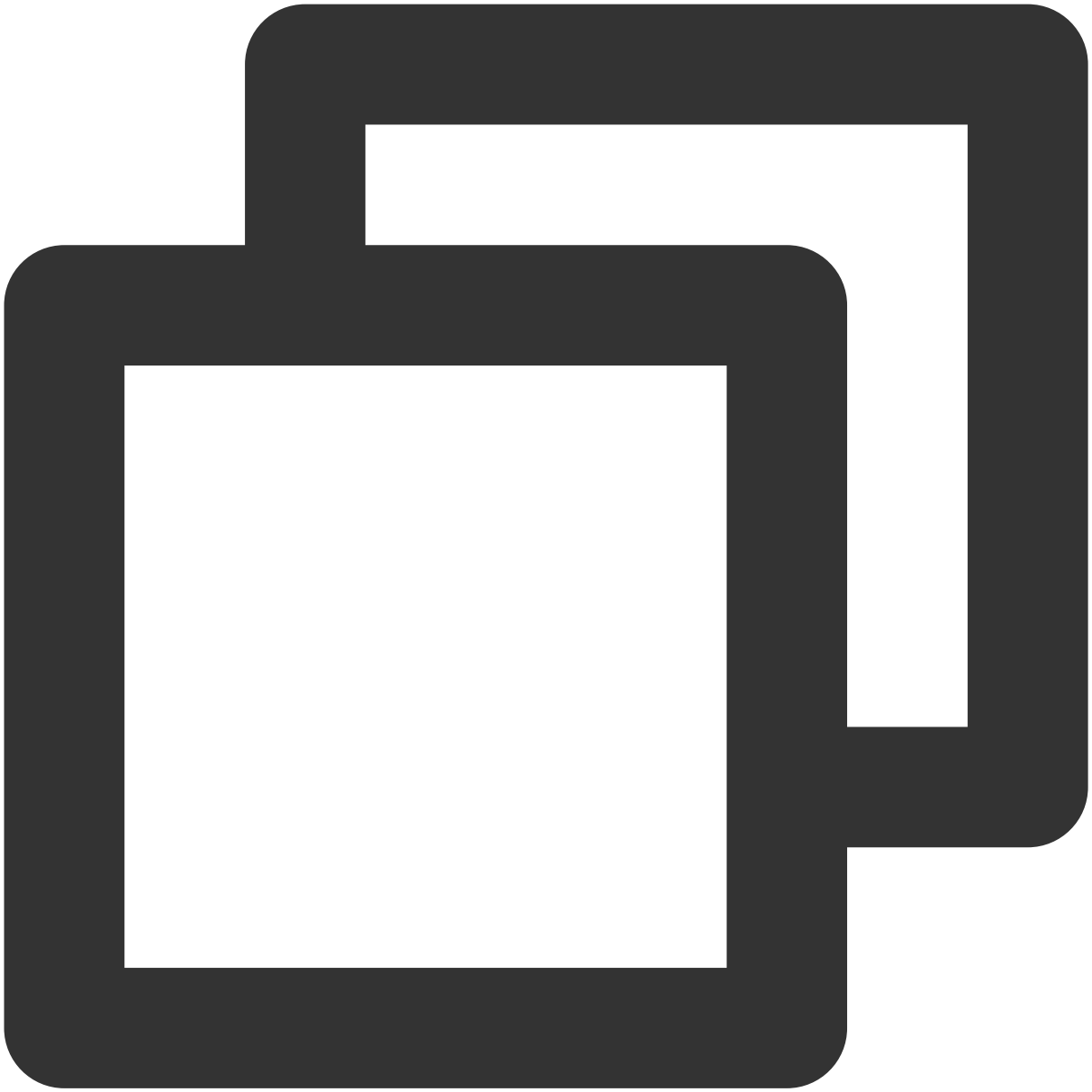
열거된 값 접두사 추가

여전히 객체 키를 쉽게 검색할 수 있는지 확인하려면 파일 유형에 따라 접두사를 열거하여 객체를 그룹화할 수 있습니다. 동일한 열거 값을 가진 접두사는 접두사가 위치한 인덱스 파티션의 기능을 공유합니다.



```
logs/20170701/log120000.tar.gz
logs/20170701/log120500.tar.gz
logs/20170701/log121000.tar.gz
...
images/image001/indexpage1.jpg
images/image002/indexpage2.jpg
images/image003/indexpage3.jpg
...
```

열거된 접두사에 대한 액세스 로드가 초당 30000개 이상의 요청으로 유지되는 경우 이전 방법을 참고하여 열거된 값 뒤에 해시 접두사를 추가하여 다중 인덱스 파티션을 구현하여 읽기/쓰기 성능을 향상시킬 수 있습니다.

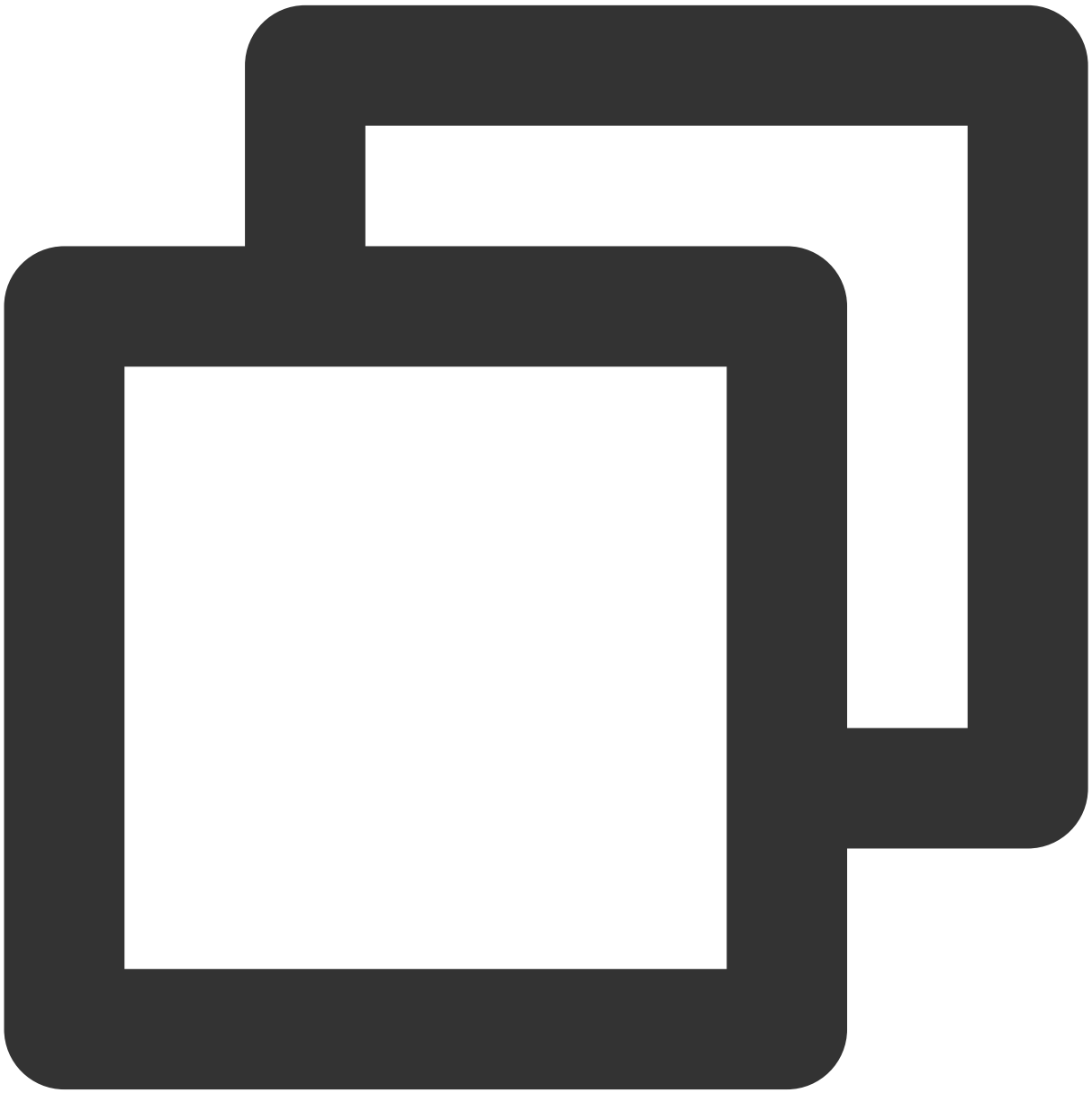


```
logs/faf1-20170701/log120000.tar.gzlogs/
e073-20170701/log120500.tar.gz
logs/333c-20170701/log121000.tar.gz
...
images/0165-image001/indexpage1.jpg
images/a349-image002/indexpage2.jpg
images/ac00-image003/indexpage3.jpg
```

```
...
```

키 이름 문자열 반전

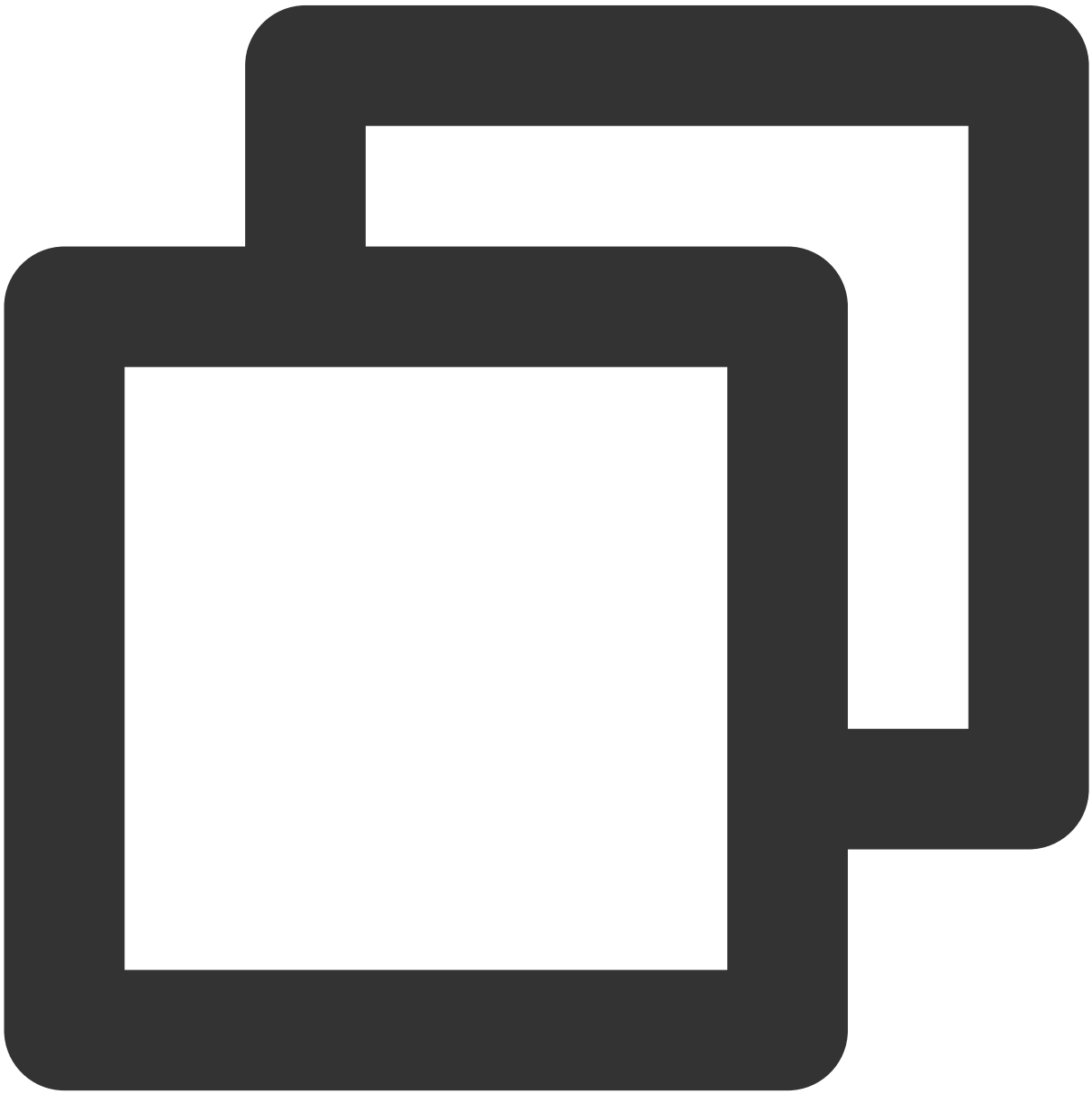
증분 ID 또는 날짜를 사용해야 하거나 단일 업로드에서 접두사가 연속적인 많은 객체를 업로드하려면 다음 방법을 참고하십시오.



```
20170701/log0701A.tar.gz
20170701/log0701B.tar.gz
20170702/log0702A.tar.gz
```

```
20170702/log0702B.tar.gz
...
id16777216/album/hongkong/img20170701121314.jpg
id16777216/music/artist/tony/anythinggoes.mp3
id16777217/video/record20170701121314.mov
id16777218/live/show/date/20170701121314.mp4
...
```

상기 키 값에 대한 네이밍 방식은 2017과 ID가 붙은 키 값이 위치한 인덱스 파티션의 성능을 쉽게 소모합니다. 이 경우 임의의 특정 요소를 허용하기 위해 키 접두사의 일부를 반전시킵니다.



```
10707102/log0701A.tar.gz
10707102/log0701B.tar.gz
20707102/log0702A.tar.gz
20707102/log0702B.tar.gz
...
61277761di/album/hongkong/img20170701121314.jpg
61277761di/music/artist/tony/anythinggoes.mp3
71277761di/video/record20170701121314.mov
81277761di/live/show/date/20170701121314.mp4
...
```

높은 GET 요청 로드

워크로드가 주로 GET 요청(예: 다운로드 요청)과 관련된 경우 위 지침을 준수하는 것 외에도 Tencent Cloud CDN 서비스와 함께 COS를 사용하는 것이 좋습니다.
Tencent Cloud CDN은 콘텐츠를 사용자에게 배포할 때 대기 시간을 최소화하고 속도를 향상시키는 데 사용할 수 있는 중국과 전 세계에 엣지 가속 노드를 보유하고 있습니다. 핫 파일은 프리패치 기능을 사용하여 캐시할 수 있으므로 COS 원본 서버로 반환되는 GET 요청 수를 줄일 수 있습니다. 자세한 내용은 도메인 관리를 참고하십시오.

COS 스트레스 테스트 가이드

최종 업데이트 날짜: : 2024-06-24 16:44:04

COSBench 소개

COSBench는 Intel의 오픈 소스로서 COS의 부하 테스트 툴로 사용됩니다. Tencent Cloud Object Storage, COS)는 S3 프로토콜과 호환되는 COS 시스템으로서 COSBench를 사용해 읽기/쓰기 성능에 대한 부하 테스트를 실시할 수 있습니다.

시스템 환경

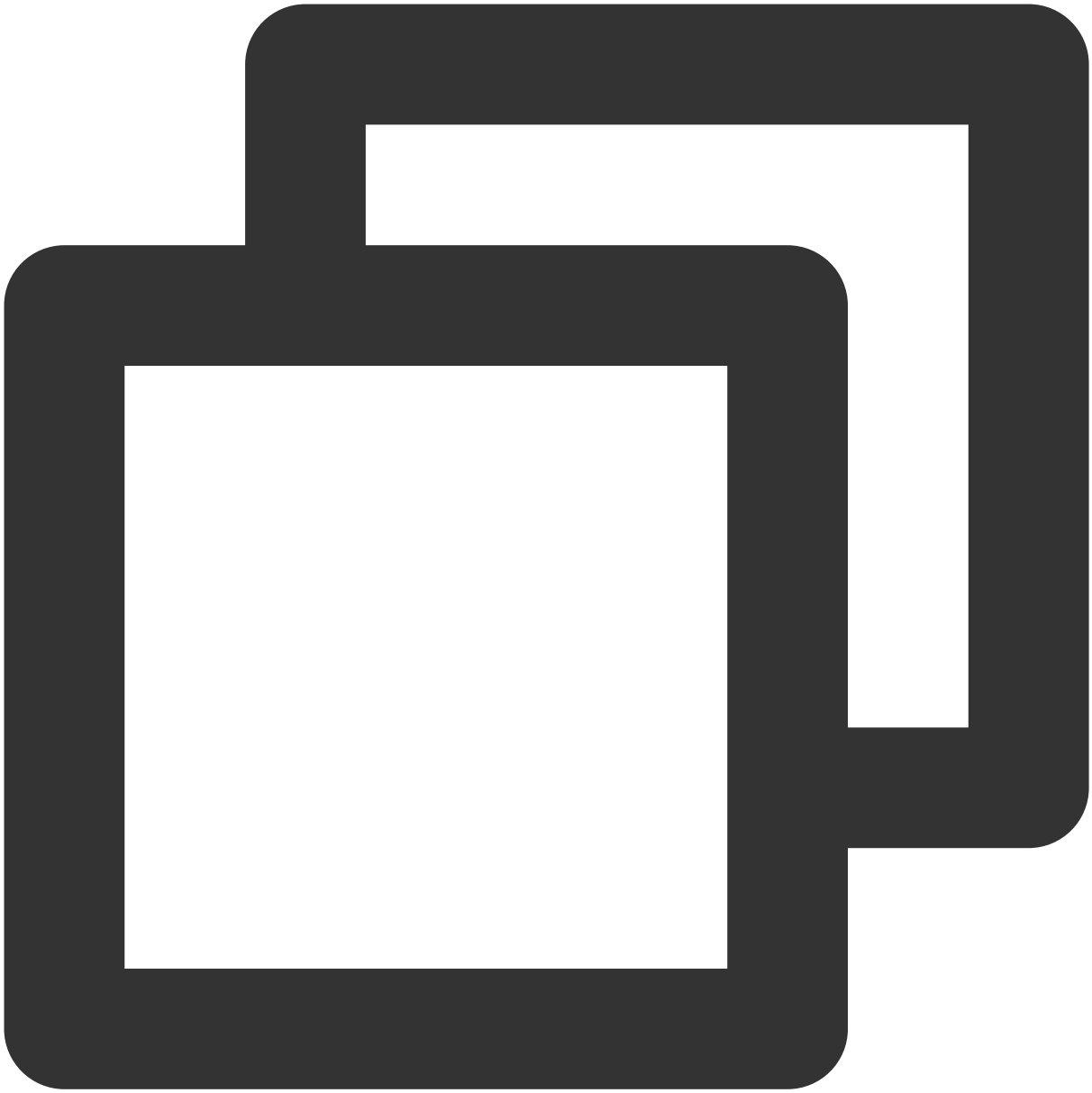
CentOS 7.0 이상에서 COSBench를 실행하는 것이 좋습니다. ubuntu에서 실행하면 예상치 못한 문제가 발생할 수 있습니다.

성능 영향 요인

CPU 코어: 많은 수의 실행 중인 worker와 결합된 적은 수의 CPU 코어는 컨텍스트 전환에 높은 오버헤드를 유발할 가능성이 큽니다. 따라서 테스트에는 32개 또는 64개의 코어가 권장됩니다.
NIC: 서버로부터의 아웃바운드 트래픽이 제한됩니다. 대용량 파일에 대한 트래픽을 테스트하려면 10GB 이상의 NIC가 권장됩니다.
네트워크 링크: 공중망 링크는 품질이 다릅니다. 공중망을 통한 다운로드를 위한 공중망 다운스트림 트래픽에 대해 요금이 부과됩니다. 따라서 동일한 리전 내에서 액세스하기 위해서는 사설망을 권장합니다.
테스트 기간: 신뢰할 수 있는 값을 얻으려면 더 긴 테스트 기간을 권장합니다.
테스트 환경: 프로그램에서 실행 중인 JDK 버전도 핵심 성능 요소입니다. 예를 들어, 이전 JDK 버전을 사용하여 HTTPS에서 테스트할 때 암호화 알고리즘에 대한 GCM BUG와 난수 생성기의 잠금 문제가 발생할 수 있습니다.

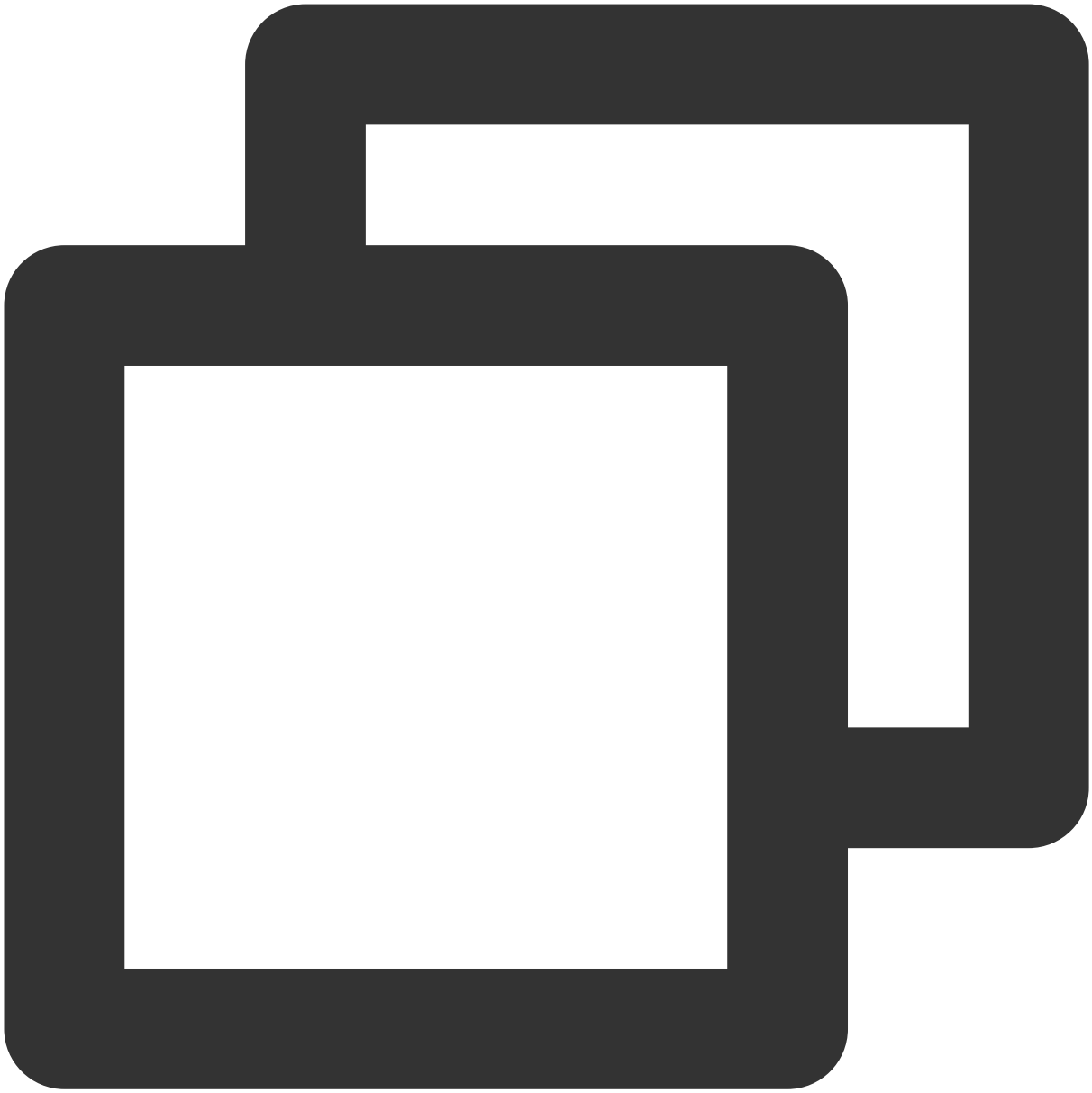
COSBench 실행 단계

1. GitHub에서 COSBench 0.4.2.c4.zip을 다운로드하고 서버에서 압축을 해제합니다.
2. COSBench의 종속 라이브러리를 설치한 뒤 다음 명령어를 실행합니다.
centos의 경우 다음 명령을 실행하여 종속성을 설치합니다:



```
sudo yum install nmap-ncat java curl java-1.8.0-openjdk-devel -y
```

ubuntu의 경우 다음 명령을 실행하여 종속성을 설치합니다.



```
sudo apt install nmap openjdk-8-jdk
```

3. s3-config-sample.xml 파일을 편집하여 작업 설정 정보를 추가합니다. 작업 설정 시 다음 다섯 단계를 주로 포함합니다.

3.1 init 단계: 버킷을 생성합니다.

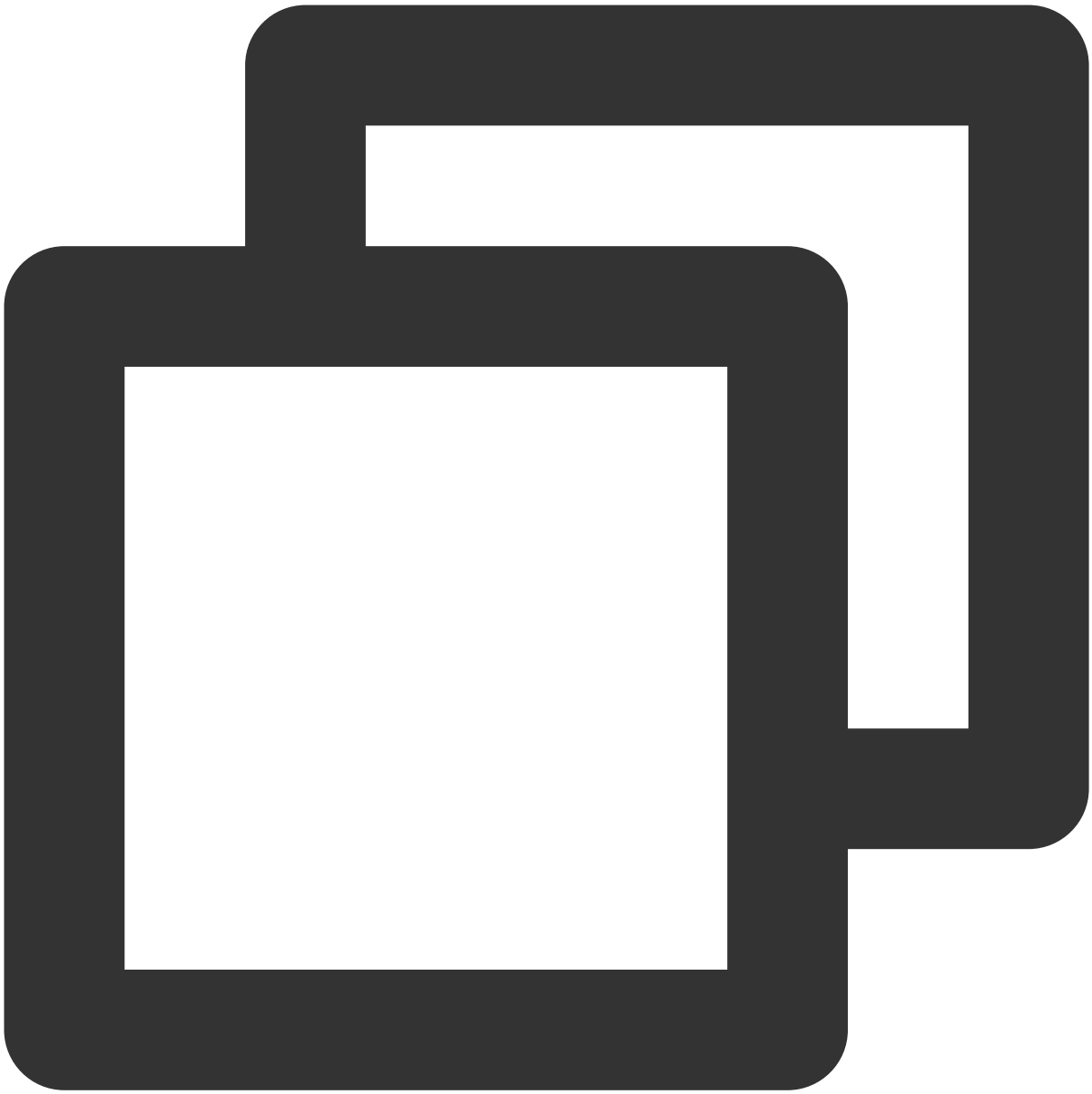
3.2 prepare 단계: worker 스레드, main단계에서 읽어오기 위해 PUT 업로드 시 객체 크기를 지정합니다.

3.3 main 단계: worker 스레드, 읽기/쓰기 객체를 혼합하고, 실행 시간을 지정합니다.

3.4 cleanup 단계: 생성된 객체를 삭제합니다.

3.5 dispose 단계: 버킷을 삭제합니다.

설정 예시는 다음과 같습니다.



```
<?xml version="1.0" encoding="UTF-8" ?>
<workload name="s3-50M-sample" description="sample benchmark for s3">

  <storage type="s3" config="accesskey=AKIDHZRLB9Ibhdp7Y7gyQq6BOk1997xxxxxx;secretk

  <workflow>

    <workstage name="init">
      <work type="init" workers="10" config="cprefix=examplebucket;csuffix=-1250000
```

```
    </workstage>

    <workstage name="prepare">
      <work type="prepare" workers="100" config="cprefix=examplebucket;csuffix=-125
    </workstage>

    <workstage name="main">
      <work name="main" workers="100" runtime="300">
        <operation type="read" ratio="50" config="cprefix=examplebucket;csuffix=-12
        <operation type="write" ratio="50" config="cprefix=examplebucket;csuffix=-1
      </work>
    </workstage>

    <workstage name="cleanup">
      <work type="cleanup" workers="10" config="cprefix=examplebucket;csuffix=-1250
    </workstage>

    <workstage name="dispose">
      <work type="dispose" workers="10" config="cprefix=examplebucket;csuffix=-1250
    </workstage>

  </workflow>

</workload>
```

매개변수 설명

매개변수	설명
accesskey, secretkey	키 정보. 리스크를 줄이기 위해 서브 계정 키를 사용하고 최소 권한의 원칙을 따르는 것이 좋습니다. 서브 계정 키를 얻는 방법에 대한 자세한 내용은 Access Key를 참고하십시오.
cprefix	examplebucket과 같은 버킷 이름 접두사
containers	버킷 이름의 값 범위. 버킷 이름은 examplebucket1 또는 examplebucket2와 같은 cprefix와 containers로 구성됩니다
csuffix	APPID. APPID에는 -1250000000과 같이 - 접두사가 붙습니다
runtime	스트레스 테스트 기간
ratio	읽기 대 쓰기 비율
workers	스트레스 테스트 스레드 수

4. cosbench-start.sh 파일을 편집하여 Java에서 다음 매개변수로 행 추가를 실행하고, s3의 md5 인증 기능을 비활성화합니다.```plaintext-Dcom.amazonaws.services.s3.disableGetObjectMD5Validation=true```![]

(https://qcloudimg.tencent-cloud.cn/raw/ac010bb86f091d709a0776b4e20a5858.png)5. cosbench 서비스를 시작합니다. - centos의 경우 다음 명령을 실행합니다.```plaintextsudo bash start-all.sh``` - ubuntu의 경우 다음 명령을 실행합니다.```plaintextsudo bash start-driver.sh &sudo bash start-controller.sh &```6. 다음 명령어를 실행하여 작업을 제출합니다.```plaintextsudo bash cli.sh submit conf/s3-config-sample.xml```URL `http://ip:19088/controller/index.html`(ip는 사용자의 부하 테스트 기기 IP로 대체)에서 실행 상태를 조회합니다.이때 다음 이미지와 같이 다섯 단계로 실행되는 것을 확인할 수 있습니다.7. 다음은 소속 리전이 베이징이며, 32코어, 사설망 대역폭 17Gbps인 CVM에 대한 업로드 및 다운로드 성능 테스트 예시로, 두 단계를 포함하고 있습니다. 1. prepare 단계: worker 스레드가 100개이며, 50MB 객체를 1000개 업로드합니다. 2. main 단계: 100개의 worker 스레드가 300초 동안 읽기/쓰기 객체를 혼합합니다.
위와 같은 두 단계의 성능 테스트 결과는 다음과 같습니다.

Basic Info

ID: w27 **Name**: s3-50M-sample **Current State**: finished

Submitted At: 2019-3-20 10:39:53 **Started At**: 2019-3-20 10:39:53 **Stopped At**: 2019-3-20 10:50:29

more info

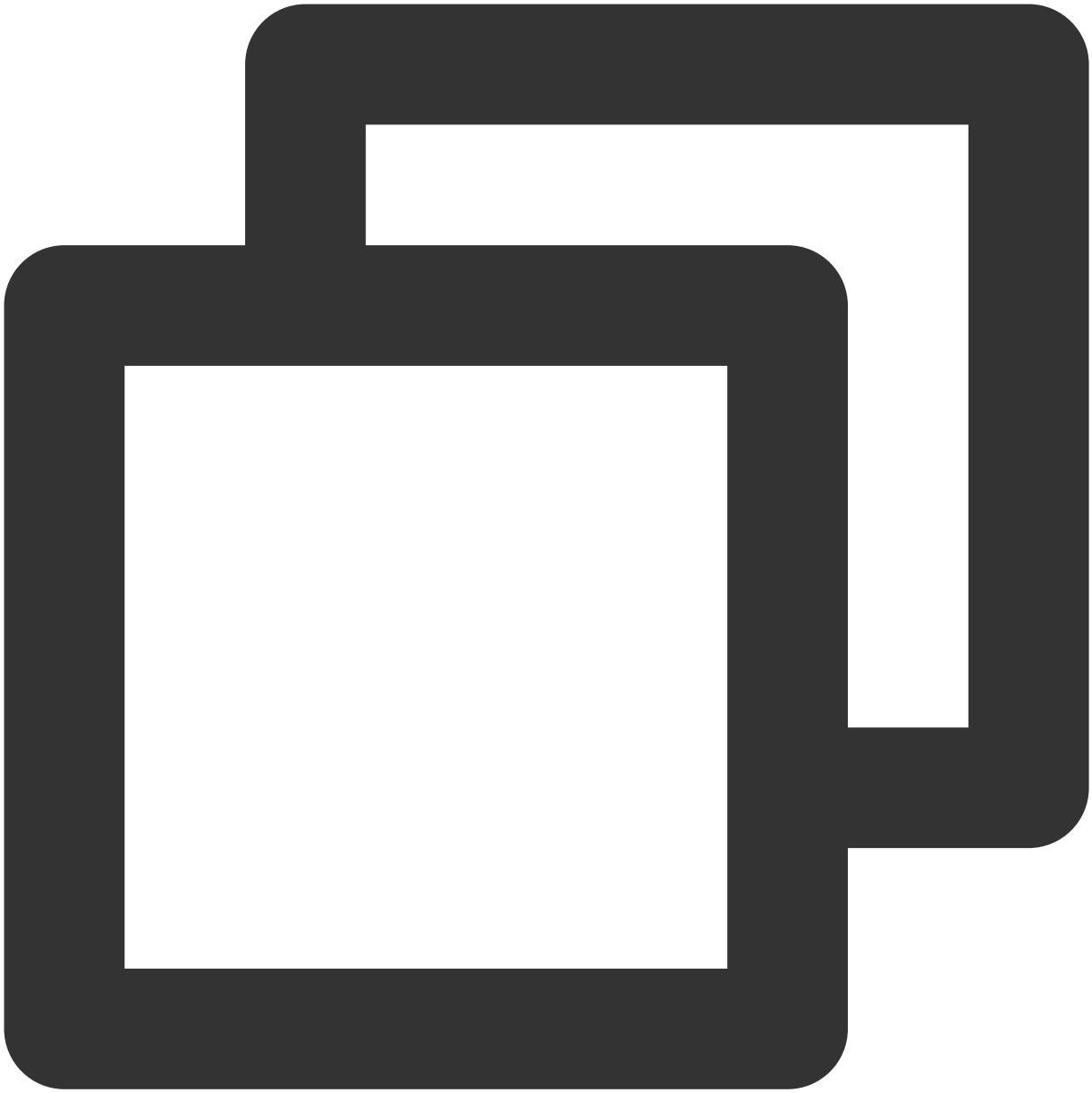
Final Result

General Report

Op-Type	Op-Count	Byte-Count	Avg-ResTime	Avg-ProcTime	Throughput	Bandwidth	Succ-Ratio
op1: init -write	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A
op1: prepare - write	10 kops	500 GB	2460.65 ms	181.83 ms	41.27 op/s	2.06 GB/S	100%
op1: read	10.22 kops	510.75 GB	2012.74 ms	118.33 ms	34.15 op/s	1.71 GB/S	100%
op2: write	10.28 kops	514.2 GB	908.98 ms	317.71 ms	34.38 op/s	1.72 GB/S	100%
op1: cleanup - delete	20 kops	0 B	14.19 ms	14.19 ms	704.74 op/s	0 B/S	100%
op1: dispose - delete	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A

show peformance details

8. 다음 명령어를 실행하여 테스트 서비스를 중지합니다.



```
sudo bash stop-all.sh
```

데이터 마이그레이션
로컬 데이터 COS로 마이그레이션

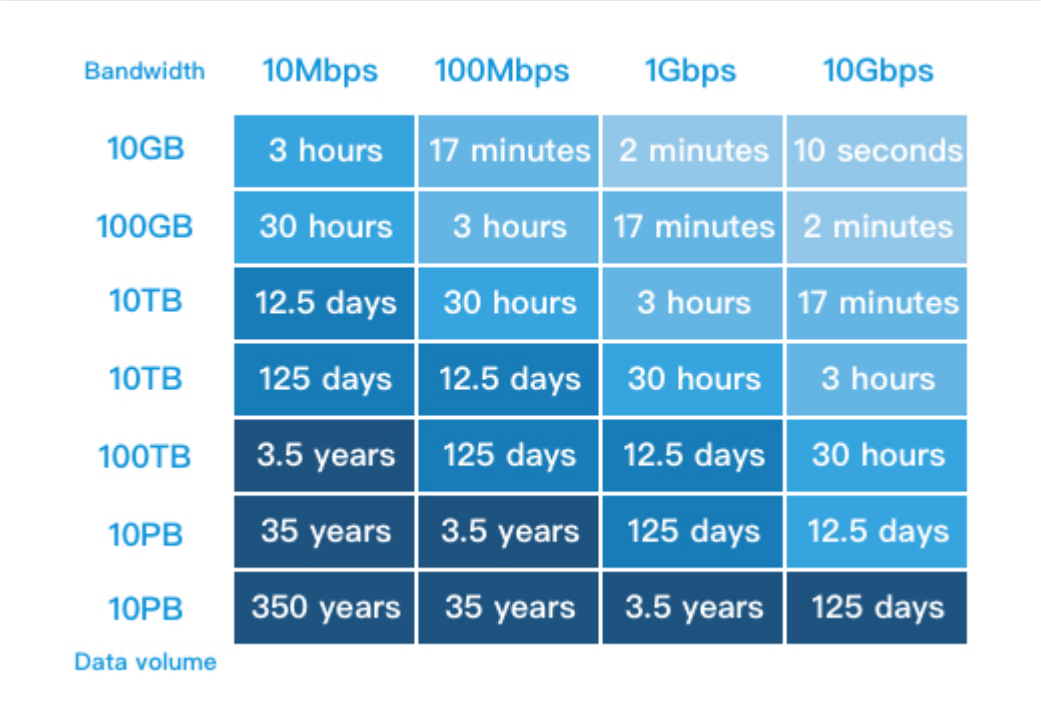
최종 업데이트 날짜: : 2024-06-24 16:44:04

실행 시나리오

로컬 IDC를 보유하고 있는 사용자에 대해 COS는 각 마이그레이션 유형별로 다음과 같은 마이그레이션 방법을 지원합니다. 사용자는 로컬 IDC에 있는 대량의 데이터를 빠르게 COS로 마이그레이션할 수 있습니다.

마이그레이션 방법	설명
COS Migration(온라인 마이그레이션)	COS Migration은 COS 데이터 마이그레이션 기능이 결합된 통합형 툴입니다. 사용자는 간단한 설정 작업을 통해 데이터를 COS에 신속하게 마이그레이션할 수 있습니다.
Cloud Data Migration(CDM)(오프라인 마이그레이션)	CDM은 Tencent Cloud에서 제공하는 오프라인 마이그레이션 전용 디바이스를 이용하여 사용자가 로컬 데이터를 클라우드에 마이그레이션할 수 있는 방법입니다. 로컬 데이터센터에서 네트워크를 통해 클라우드로 마이그레이션할 때의 긴 시간, 높은 비용, 보안성 문제를 해결할 수 있습니다.

사용자는 데이터 마이그레이션 용량, IDC 게이트 대역폭, IDC 유휴 서버 리소스, 수용 가능한 마이그레이션 완료 시간 등의 요소를 고려하여 마이그레이션 방법을 선택할 수 있습니다. 아래 이미지는 온라인 마이그레이션 이용 시 소요되는 예상 시간으로, 마이그레이션 주기가 10일 이상이거나 마이그레이션 데이터 용량이 50TB가 넘는 경우 CDM을 이용한 오프라인 마이그레이션을 권장합니다. 이외에는 온라인 마이그레이션을 선택하시기 바랍니다.



주의 :

1MB 이하의 저용량 파일의 수량이 비교적 많고, 디스크 IO 성능이 부족한 등의 경우에도 데이터 마이그레이션 진도에 영향을 미칠 수 있습니다.

마이그레이션 실행

COS Migration

마이그레이션 작업 순서는 다음과 같습니다.

1. Java 환경을 설치합니다.
2. COS Migration 툴을 설치합니다.
3. 구성 파일을 수정합니다.
4. 툴을 실행합니다.

자세한 방법은 COS Migration 툴 문서를 참조하십시오.

작업 팁

COS Migration을 설정하여 마이그레이션 속도를 최대로 향상시키는 방법을 소개합니다.

1. 자체 네트워크 환경에 따라 파일 크기를 구분하는 임계값과 마이그레이션 동시 접속 수를 조정하여 대용량 파일은 멀티파트 방식으로 전송하고 저용량 파일은 동시 접속 방식으로 전송하는 최적의 마이그레이션 방식을 실현합니다. 툴의 실행 시간을 조정하고 대역폭 제한을 설정해 비즈니스 운영에 데이터 마이그레이션의 대역폭 점유로 인한 영향이 미치지 않도록 합니다. 해당 사항은 구성 파일 config.ini의 `[common]` 에서 조정할 수 있으며, 다음 매개변수를 수정해 조정합니다.

매개변수 이름	매개변수 설명
smallFileThreshold	저용량 파일의 임계값 매개변수입니다. 임계값 이상인 경우 멀티파트 업로드를 실행하며, 기본값은 5MB입니다.
bigFileExecutorNum	대용량 파일의 동시 접속 수이며, 기본값은 8입니다.외부 네트워크를 통해 COS에 접속하고 대역폭이 비교적 작은 경우 해당 동시 접속 수를 작게 설정하십시오.
smallFileExecutorNum	저용량 파일의 동시 접속 수이며, 기본값은 64입니다. 외부 네트워크를 통해 COS에 접속하고 대역폭이 비교적 작은 경우 해당 동시 접속 수를 작게 설정하십시오.
executeTimeWindow	해당 매개변수로 마이그레이션 툴이 매일 실행되는 시간대를 정의합니다. 다른 시간에는 휴면 상태가 되며, 휴면 상태에서는 마이그레이션을 일시 중지하고 다음 번 시간창이 자동으로 이어서 실행할 때까지 마이그레이션 진도를 유지합니다.

2. 분산형 병렬 전송 방식을 사용해 마이그레이션 속도를 더욱 향상시킬 수 있습니다. 여러 서버에 COS Migration을 설치하고 각 서버별로 서로 다른 원본 데이터를 마이그레이션하는 방법을 고려할 수 있습니다.

CDM

마이그레이션 작업 순서는 다음과 같습니다.

1. CDM 콘솔에서 신청을 제출합니다.
2. 신청 심사가 완료되면 디바이스 발송을 기다립니다.
3. 디바이스 수취 후, 마이그레이션 디바이스 설명서에 따라 데이터를 디바이스로 복사합니다.
4. 데이터 복사 완료 후, 콘솔에서 반환 신청을 제출해 Tencent Cloud가 데이터를 COS로 마이그레이션을 완료할 때까지 기다립니다.

자세한 정보는 CDM 제품 문서를 참조하십시오.

작업 팁

오프라인 마이그레이션을 통해 데이터를 효율적이고 안전하게 Tencent Cloud COS에 마이그레이션하는 방법을 소개합니다.

1. IDC에서 10Gbps의 네트워크 환경을 설정한 경우 로컬 데이터 환경에서 전송 병목 현상이 발생하는 것을 방지하기 위해 고성능 마운트 포인트 기기를 사용해 최대 속도로 복사할 수 있습니다.
2. CDM을 통한 데이터 전송 중 가장 빠른 방법은 병렬 전송 방식입니다. 사용자는 디바이스의 CPU와 메모리 사용률을 모니터링하여 현제 마이그레이션 속도가 예상보다 느린 경우 다음과 같이 병렬 전송 방식을 선택할 수 있습니다.

여러 디바이스에서 서로 다른 네트워크 인터페이스를 사용하여 동일한 CDM 디바이스로 연결
여러 디바이스에서 서로 다른 네트워크 인터페이스를 사용하여 여러 CDM 디바이스로 연결

타사 클라우드 스토리지 데이터를 COS로 마이그레이션

최종 업데이트 날짜: : 2024-06-24 16:44:04

실행 배경

Cloud Object Storage(COS)는 타사 클라우드 스토리지 플랫폼의 데이터를 COS로 빠르게 마이그레이션할 수 있도록 지원합니다.

마이그레이션 방법	인터랙티브 방식	파일 크기를 구분하는 임계값	마이그레이션 동시 접속 수	HTTPS 보안 전송
MSP	시각화 페이지 작업	기본 설정 사용	전역 통합	활성화

이 플랫폼은 기본 마이그레이션 요구 사항을 충족할 수 있는 데이터 마이그레이션 진행률 조회, 파일 일관성 확인, 실패 후 다시 업로드, 체크포인트 업로드 재개 및 기타 기능을 지원합니다.

마이그레이션 실행

마이그레이션 서비스 플랫폼(MSP)

Migration Service Platform(MSP)은 여러 마이그레이션 툴을 통합하여 시각화 인터페이스를 제공하는 플랫폼으로, 사용자가 손쉽게 대규모 데이터 마이그레이션 작업을 모니터링하고 관리할 수 있습니다. MSP의 "파일 마이그레이션 툴"은 사용자가 데이터를 다른 공유 클라우드 또는 데이터 원본 서버에서 COS로 마이그레이션하는 데 도움을 줍니다.

마이그레이션 작업 순서는 다음과 같습니다.

1. MSP 콘솔에 로그인합니다.
2. 왼쪽 사이드바에서 **COS 마이그레이션**을 클릭하여 COS 마이그레이션 페이지로 이동합니다.
3. **작업 생성**을 클릭해 마이그레이션 작업을 생성하고 작업 정보를 설정합니다.
4. 작업을 실행합니다.

자세한 작업 방법은 다음 마이그레이션 튜토리얼을 참조하십시오.

Alibaba Cloud OSS 마이그레이션
Huawei Cloud OBS 마이그레이션
Qiniu Cloud KODO 마이그레이션
UCLOUD UFile 마이그레이션

KS Cloud KS3 마이그레이션
Baidu Cloud BOS 마이그레이션
AWS S3 Migration Tutorial

작업 팁

데이터 마이그레이션 중 데이터 소스 읽기 속도는 서로 다른 네트워크 환경에 따라 차이가 있을 수 있으며, 사용자는 실제 상황에 따라 "파일 마이그레이션 작업 생성" 시 비교적 높은 QPS 동시 접속률을 선택하여 마이그레이션 속도를 향상시킬 수 있습니다.

URL이 소스 주소인 데이터를 COS로 마이그레이션

최종 업데이트 날짜: : 2024-06-24 16:44:04

실행 배경

사용자가 URL 리스트를 데이터 원본 주소로 사용해 데이터 마이그레이션을 진행하고 싶은 경우, COS에서는 다음과 같은 마이그레이션 방법을 지원합니다.

마이그레이션 방법	설명
마이그레이션 서비스 플랫폼(MSP)	MSP는 여러 마이그레이션 툴을 통합하여 시각화 인터페이스를 제공하는 플랫폼으로, 사용자가 손쉽게 대규모 데이터 마이그레이션 작업을 모니터링하고 관리할 수 있습니다. MSP의 "파일 마이그레이션 툴"은 사용자가 데이터를 다른 공유 클라우드 및 데이터 원본 서버에서 COS로 마이그레이션하는 데 도움을 줍니다.
COS Origin-pull	COS Origin-pull은 데이터 원본 서버에서 읽기 및 쓰기 액세스 요청이 있는 데이터를 자동으로 Tencent Cloud COS에 마이그레이션합니다. 해당 마이그레이션 방법은 데이터를 빠르게 핫/콜드 티어링을 진행하는데 도움이 될 뿐만 아니라 서비스 시스템의 핫 데이터에 대한 읽기 및 쓰기 액세스 속도를 향상시켜 줍니다.
COS Migration	COS Migration은 COS 데이터 마이그레이션 기능이 결합된 통합형 툴입니다. 사용자는 간단한 설정 작업을 통해 데이터를 COS에 신속하게 마이그레이션할 수 있습니다.

데이터 원본 서버에서 클라우드에 마이그레이션하는 과정에서 사용자가 데이터 원본 서버의 핫 데이터만 모두 클라우드에 마이그레이션하고 콜드 데이터는 원본 서버에 보관하고 싶은 경우, COS Origin-pull 마이그레이션 방법을 사용할 수 있습니다. COS Origin-pull은 읽기 및 쓰기 액세스 요청이 있는 핫 데이터만 COS에 마이그레이션하며, 저빈도 액세스의 비즈니스 데이터는 자동으로 핫/콜드 티어링합니다.

주의 :

현재 COS는 인증 정보가 있는 URL 데이터에 대한 마이그레이션을 지원하지 않습니다.

마이그레이션 실행

마이그레이션 서비스 플랫폼(MSP)

마이그레이션 작업 순서는 다음과 같습니다.
1. MSP에 로그인합니다.
1. 왼쪽 메뉴바에서 [마이그레이션 툴]을 클릭하고 "파일 마이그레이션 툴"을 찾아 [시작하기]를 클릭합니다.
2. 마이그레이션 작업을 생성하고 작업 정보를 설정합니다.
3. 작업을 실행합니다.
자세한 작업 방법은 URL 리스트 마이그레이션을 참조하십시오.

작업 팁

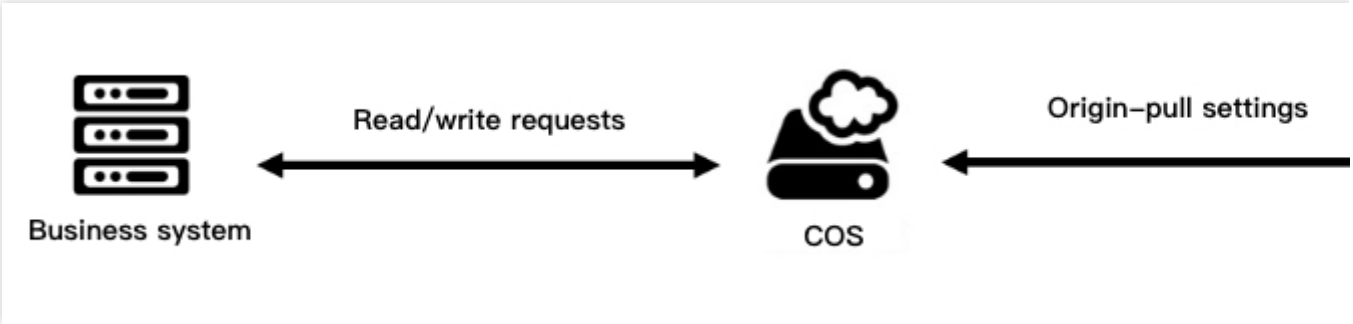
데이터 마이그레이션 중 데이터 소스 읽기 속도는 서로 다른 네트워크 환경에 따라 차이가 있을 수 있으며, 사용자는 실제 상황에 따라 "파일 마이그레이션 작업 생성" 시 비교적 높은 QPS 동시 접속률을 선택하여 마이그레이션 속도를 향상시킬 수 있습니다.

COS Origin-pull

마이그레이션 작업 순서는 다음과 같습니다.
1. COS 콘솔로 이동하여 마이그레이션할 데이터의 타깃 버킷에 Origin-pull 설정을 활성화합니다.
2. 버킷의 Origin-pull 주소를 설정한 후 저장합니다.
3. 서비스 시스템의 읽기 및 쓰기 요청을 Tencent Cloud COS로 전송합니다.
자세한 작업 방법은 Origin-pull 설정 문서를 참조하십시오.

작업 팁

다음은 원본 데이터에 대한 핫/콜드 티어링을 완료하여, 핫 데이터를 Tencent Cloud COS에 무결성 마이그레이션해 핫 데이터의 읽기 요청 속도를 향상시킬 수 있는 방법입니다.
1. 서비스 시스템의 읽기 및 쓰기 요청을 COS로 전환합니다. COS 콘솔의 Origin-pull 기능 설정 페이지에서 Origin-pull 주소를 데이터 원본 서버로 설정합니다. 이때의 시스템 구조는 다음 이미지와 같습니다.



2. 일정 시간이 지나면 콜드 데이터는 여전히 원본 서버에 남아 있지만 핫 데이터는 Tencent Cloud COS에 마이그레이션되어 있으며, 마이그레이션 과정 중 서비스 시스템은 영향을 받지 않습니다.

COS Migration

마이그레이션 작업 순서는 다음과 같습니다.
1. Java 환경을 설치합니다.

2. COS Migration 툴을 설치합니다.
3. 구성 파일을 수정합니다.
4. 툴을 실행합니다.
자세한 방법은 COS Migration 툴 문서를 참조하십시오.

작업 팁

COS Migration을 설정하여 마이그레이션 속도를 최대로 향상시키는 방법을 소개합니다.
1. 자체 네트워크 환경에 따라 파일 크기를 구분하는 임계값과 마이그레이션 동시 접속 수를 조정하여 대용량 파일은 멀티파트 방식으로 전송하고 저용량 파일은 동시 접속 방식으로 전송하는 최적의 마이그레이션 방식을 실현합니다. 툴의 실행 시간을 조정하고 대역폭 제한을 설정해 비즈니스 운영에 데이터 마이그레이션의 대역폭 점유로 인한 영향이 미치지 않도록 합니다. 해당 사항은 구성 파일 config.ini의 `[common]` 에서 조정할 수 있으며, 다음 매개변수를 수정해 조정합니다.

매개변수 이름	매개변수 설명
smallFileThreshold	저용량 파일의 임계값 매개변수입니다. 임계값 이상인 경우 멀티파트 업로드를 실행하며, 기본값은 5MB입니다.
bigFileExecutorNum	대용량 파일의 동시 접속 수이며, 기본값은 8입니다. 외부 네트워크를 통해 COS에 접속하고 대역폭이 비교적 작은 경우 해당 동시 접속 수를 작게 설정하십시오.
smallFileExecutorNum	저용량 파일의 동시 접속 수이며, 기본값은 64입니다. 외부 네트워크를 통해 COS에 접속하고 대역폭이 비교적 작은 경우 해당 동시 접속 수를 작게 설정하십시오.
executeTimeWindow	해당 매개변수로 마이그레이션 툴이 매일 실행되는 시간대를 정의합니다. 다른 시간에는 휴면 상태가 되며, 휴면 상태에서는 마이그레이션을 일시 중지하고 다음 번 시간 창이 자동으로 이어서 실행할 때까지 마이그레이션 진도를 유지합니다.

2. 분산형 병렬 전송 방식을 사용해 마이그레이션 속도를 더욱 향상시킬 수 있습니다. 여러 서버에 COS Migration을 설치하고 각 서버별로 서로 다른 원본 데이터를 마이그레이션하는 방법을 고려할 수 있습니다.

COS 간 데이터 마이그레이션

최종 업데이트 날짜: : 2024-06-24 16:44:04

실행 배경

Tencent Cloud COS 사용자가 버킷에 있는 데이터를 Tencent Cloud COS로 마이그레이션하는 경우, 다음과 같은 방식을 권장합니다.
온라인 마이그레이션: 마이그레이션 서비스 플랫폼(MSP)
온라인 마이그레이션: 버킷 복사

마이그레이션 실행

마이그레이션 서비스 플랫폼(MSP)

MSP는 여러 마이그레이션 툴을 통합하여 시각화 인터페이스를 제공하는 플랫폼으로, 대량의 데이터 마이그레이션 작업을 손쉽게 모니터링하고 관리할 수 있습니다. 또한 MSP의 '파일 마이그레이션 툴'을 사용하여 데이터를 다른 공유 클라우드 및 데이터 원본 서버에서 COS로 마이그레이션할 수 있습니다.
마이그레이션 작업 순서는 다음과 같습니다.
1. MSP에 로그인합니다.
2. 왼쪽 메뉴에서 [COS 마이그레이션]을 클릭하여 COS 마이그레이션 페이지로 이동합니다.
3. [작업 생성]을 클릭해 마이그레이션 작업을 생성하고 작업 정보를 설정합니다.
4. 작업을 실행합니다.
자세한 작업 방법은 Tencent Cloud COS 간 마이그레이션을 참조하십시오.
데이터 마이그레이션 중 데이터 원본 읽기 속도는 네트워크 환경에 따라 차이가 있을 수 있습니다. 사용자는 실제 상황에 따라 '파일 마이그레이션 작업 생성' 시 비교적 높은 QPS 동시 접속률을 선택하여 마이그레이션 속도를 향상시킬 수 있습니다.

버킷 복사

버킷 복사는 버킷에 대한 Tencent Cloud COS의 설정 항목으로, 버킷 복사 규칙 설정을 통해 서로 다른 리전의 버킷 간에 자동 및 비동기화 방식으로 **증분 객체**를 복사할 수 있습니다. 버킷 복사를 활성화하면 COS에서 원본 버킷의 객체 콘텐츠(예: 객체 메타데이터 및 버전 ID 등)를 타깃 버킷으로 정확하게 복사하며, 복사된 객체 사본은 원본과 완전히 동일한 속성 정보를 갖게 됩니다. 이외에도 원본 버킷에서의 객체 업로드, 객체 삭제 등 객체에 대한 작업 또한 타깃 버킷에 복사합니다. 버킷 복사에 대한 자세한 설명 및 작업 방법은 버킷 복사를 참조하십시오.

Hadoop 파일 시스템과 COS 간 데이터 마이그레이션

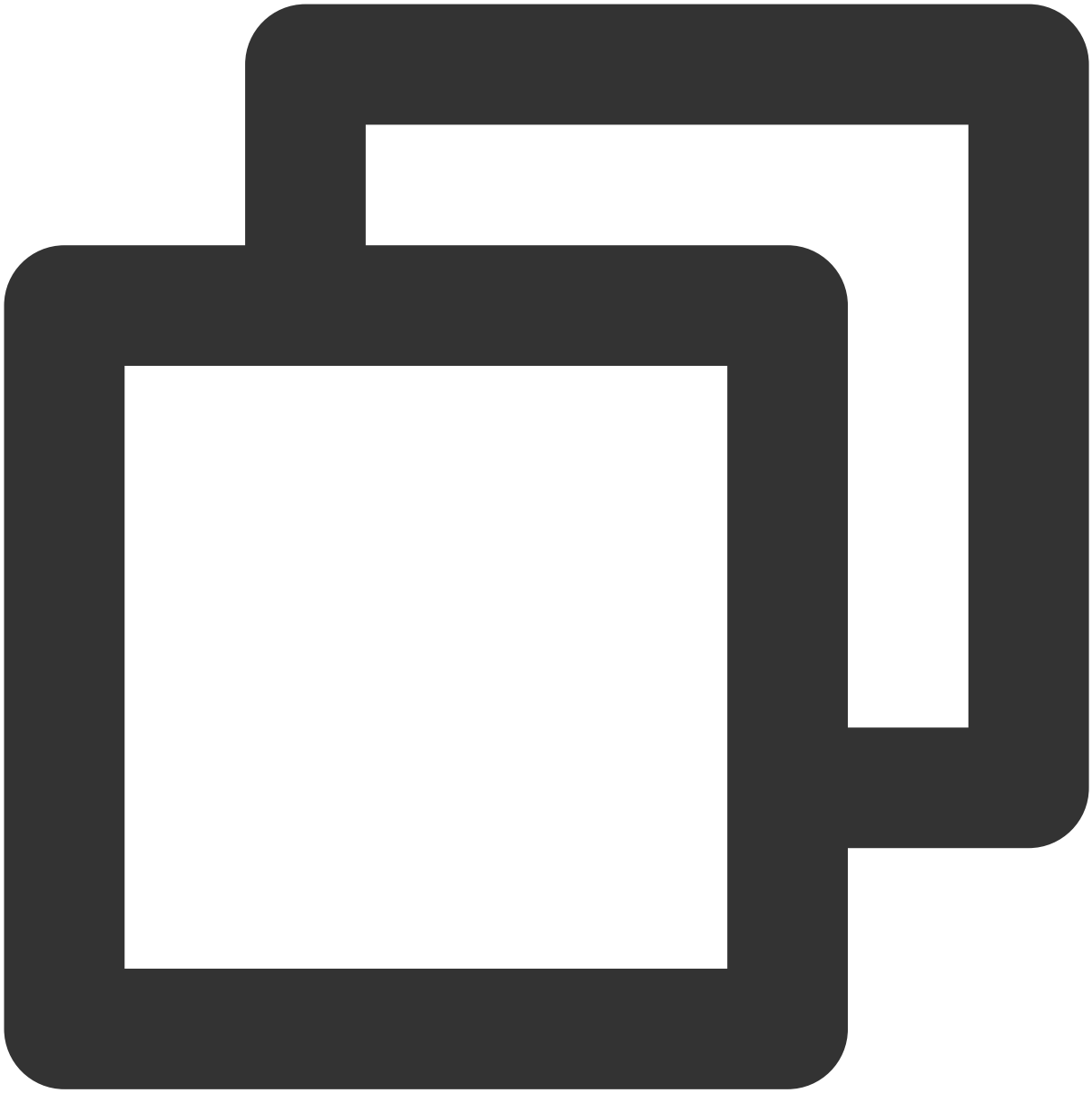
최종 업데이트 날짜: : 2024-06-24 16:44:04

소개

Hadoop Distcp(Distributed copy)는 주로 Hadoop 파일 시스템 내부 또는 해당 시스템 간의 대규모 데이터 복사에 사용되는 툴이며, Map/Reduce를 기반으로 파일 배포, 오류 처리, 최종 보고서 생성을 구현합니다. Map/Reduce는 동시 처리 능력이 있어 Map 작업을 할 때마다 원본 경로의 일부 파일이 복사되는데, 이를 통해 클러스터 리소스를 충분히 이용함으로써 클러스터 또는 Hadoop 파일 시스템 간의 대규모 데이터 마이그레이션을 신속하게 완료할 수 있습니다. Hadoop-COS는 Hadoop 파일 시스템의 semantics를 구현하기 때문에 Hadoop Distcp 툴로 COS와 기타 Hadoop 파일 시스템 간에 양방향 데이터 마이그레이션을 편리하게 진행할 수 있습니다. 본 문서에서는 HDFS를 예시로, Hadoop 파일 시스템과 COS 사이에서 Hadoop Distcp 툴을 사용한 데이터 마이그레이션 방법을 소개합니다.

전제 조건

1. Hadoop 클러스터에 Hadoop-COS 플러그 인이 설치되어 있어야 하며, COS 액세스 키 등이 정확하게 설정되어 있어야 합니다. 다음 Hadoop 명령어를 사용하여 COS가 정상적으로 액세스되는지 확인할 수 있습니다.



```
hadoop fs -ls cosn://examplebucket-1250000000/
```

COS Bucket의 파일 리스트가 정확하게 조회되는 경우 Hadoop-COS가 정상적으로 설치 및 설정되었다는 의미이며, 다음 실행 순서를 진행할 수 있습니다.

2. COS의 액세스 계정은 COS 버킷의 타깃 경로에 대한 읽기/쓰기 권한을 보유해야 합니다.

주의 :

필요에 따라 서브 계정에 COS 버킷 내 리소스에 대한 읽기/쓰기 권한을 부여할 수 있으며, 최소 권한 원칙 및 서브 계정 라이선스 가이드에 따른 권한 부여를 권장합니다. 다음은 자주 볼 수 있는 사전 설정 정책입니다.

DataFullControl: 데이터 전체 읽기/쓰기 권한은 읽기, 쓰기, 파일 리스트 나열 및 삭제 작업을 포함하므로 신중한 부여를 권장합니다.
QcloudCOSDataReadOnly: 데이터 읽기 전용 권한입니다.
QcloudCOSDataWriteOnly: 데이터 쓰기 전용 권한입니다.
사용자 정의 모니터링 기능을 사용해야 하는 경우 클라우드 모니터링 지표를 보고하고 읽을 수 있는 인터페이스 작업 권한이 필요합니다. QcloudMonitorFullAccess 권한 부여에 신중하시기 바랍니다.

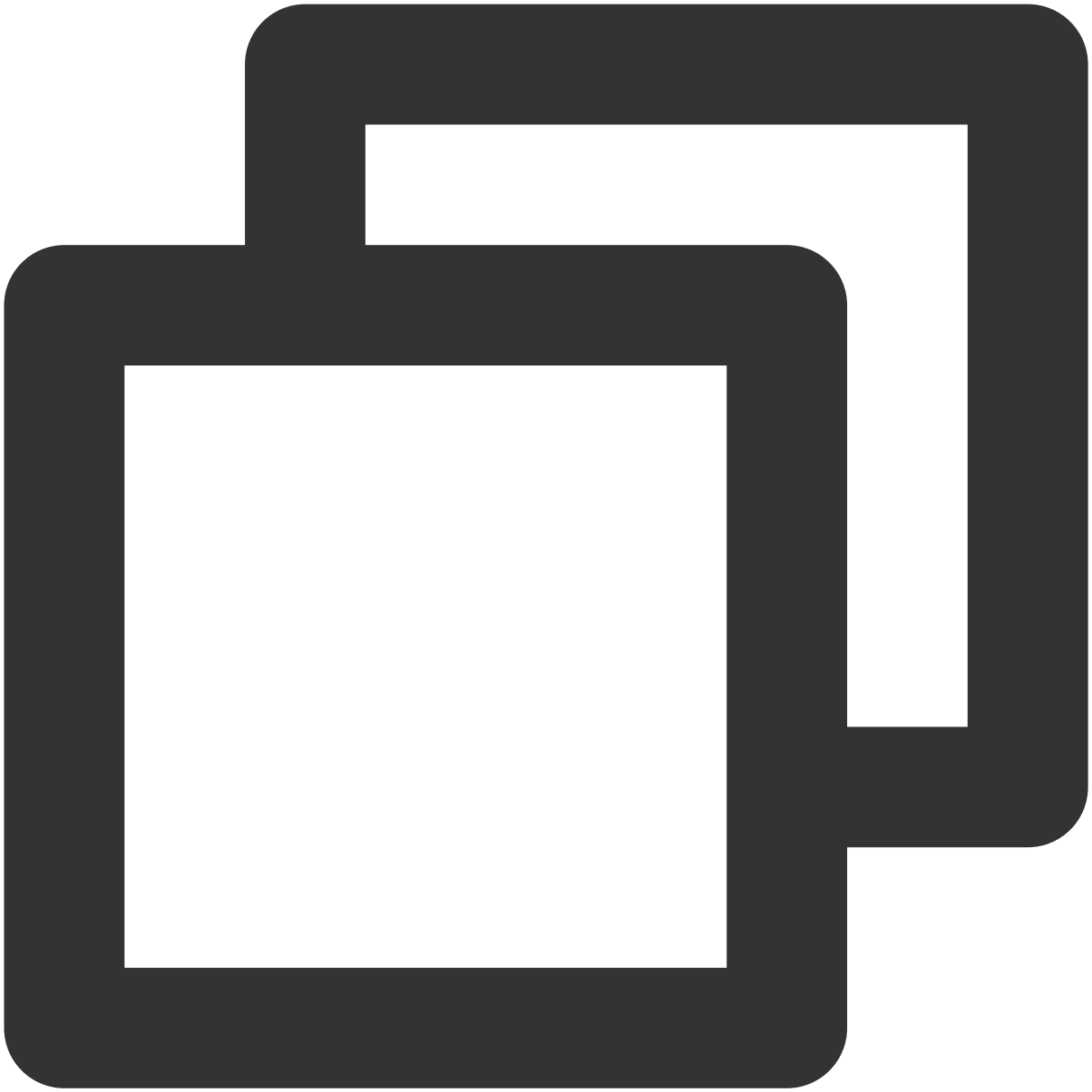
실행 순서

COS 버킷에 HDFS 데이터 복사하기

Hadoop Distcp로 로컬 HDFS 클러스터 `/test` 디렉터리에 있는 파일을 COS의 hdfs-test-1250000000 버킷으로 마이그레이션 합니다.

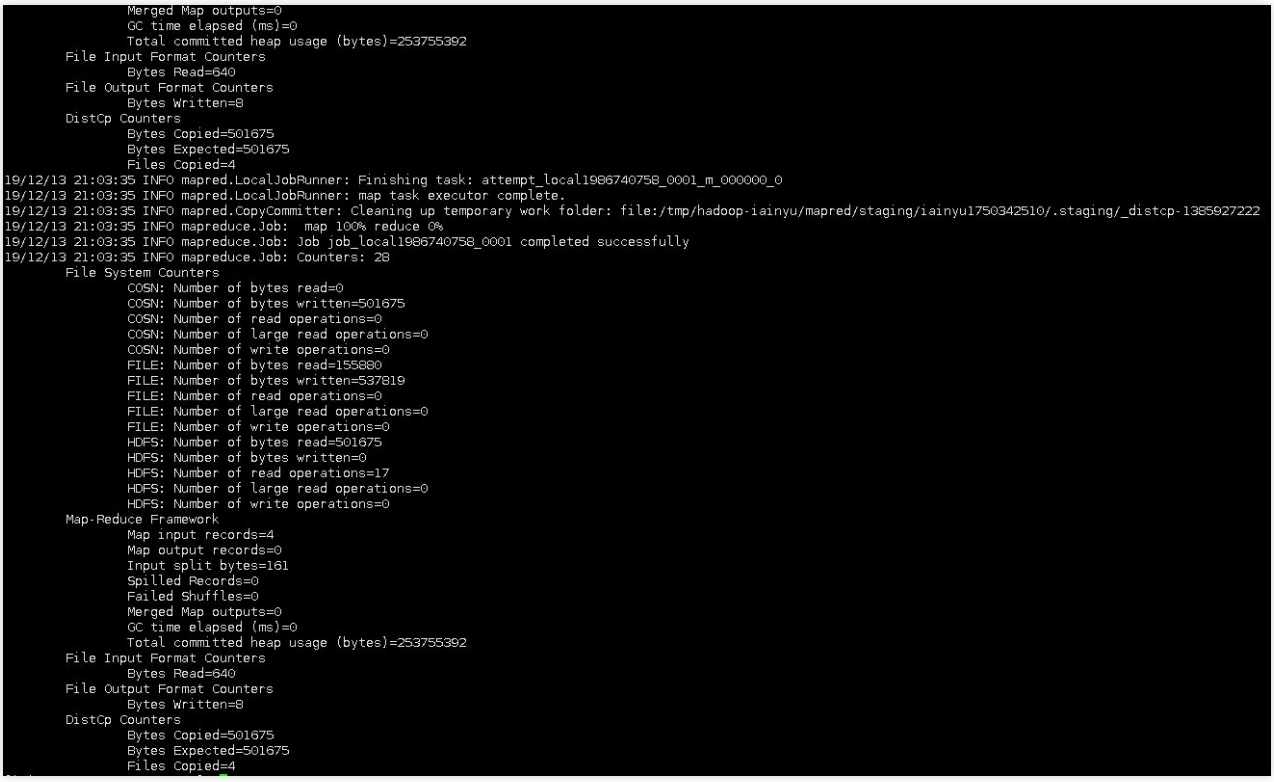
```
[iainyu@VM_38_97_centos ~]$ hadoop fs -ls -R /
drwxr-xr-x   - iainyu supergroup          0 2019-12-13 20:48 /test
-rw-r--r--   1 iainyu supergroup     167225 2019-12-13 20:47 /test/test-1
-rw-r--r--   1 iainyu supergroup     167225 2019-12-13 20:47 /test/test-2
-rw-r--r--   1 iainyu supergroup     167225 2019-12-13 20:47 /test/test-3
```

1. 다음 명령어를 실행하여 마이그레이션을 실행합니다.

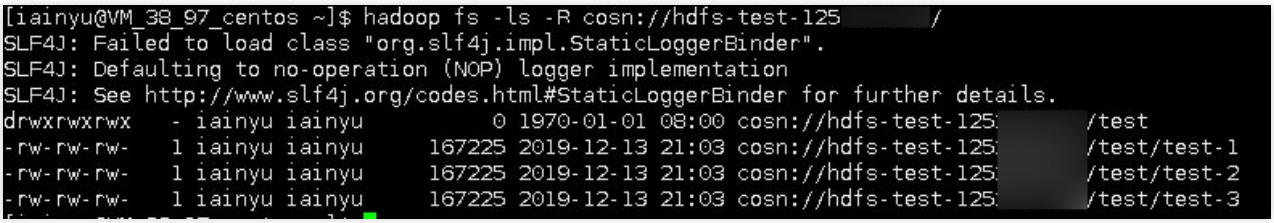


```
hadoop distcp hdfs://10.0.0.3:9000/test cosn://hdfs-test-1250000000/
```

Hadoop Distcp는 MapReduce 작업을 실행해 파일을 복사하며, 완료 후 다음 이미지와 같이 간단한 리포트 정보를 출력합니다.

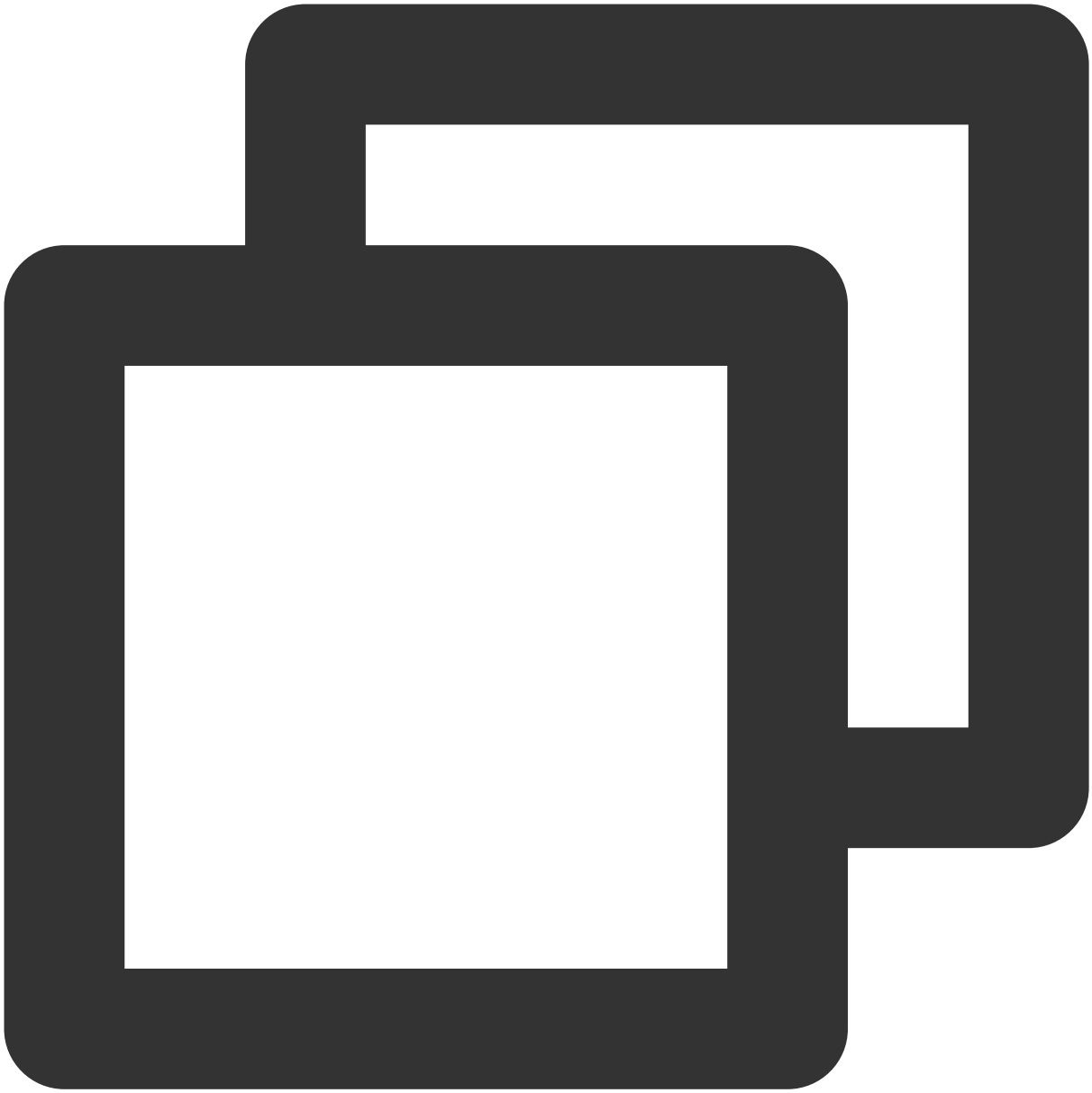


2. `hadoop fs -ls -R cosn://hdfs-test-1250000000/` 명령어를 실행하여 방금 버킷 hdfs-test-1250000000에 마이그레이션한 디렉터리와 파일을 나열할 수 있습니다.



COS의 버킷에 있는 파일을 로컬 HDFS 클러스터로 복사하기

Hadoop Distcp는 서로 다른 클러스터와 파일 시스템 간의 데이터 복사를 지원하는 툴입니다. COS 버킷에 있는 객체 경로를 원본 경로, HDFS 파일 경로를 타깃 경로로 간주하여 COS에 있는 데이터 파일을 로컬 HDFS로 복사할 수 있습니다.



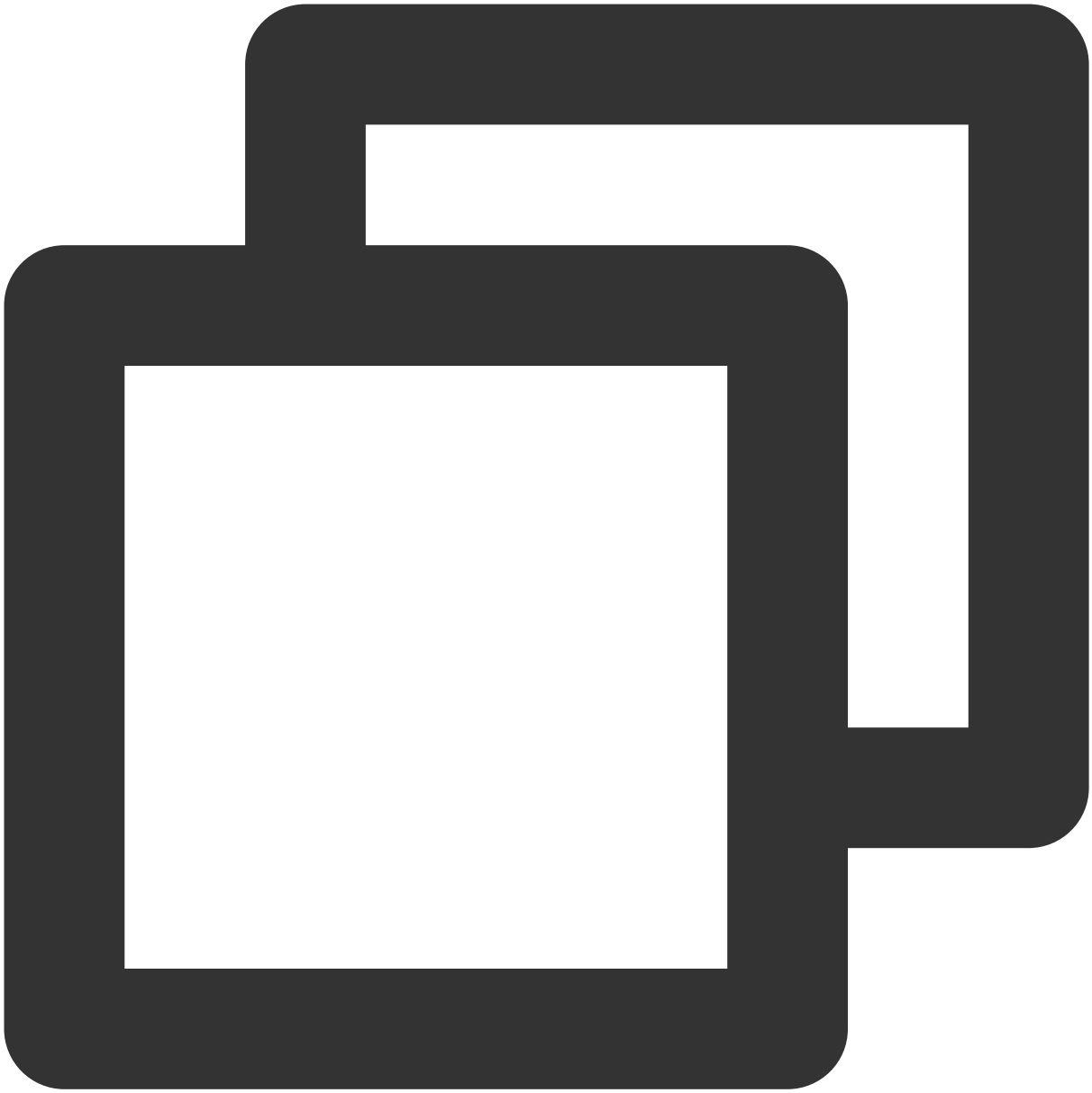
```
hadoop distcp cosn://hdfs-test-1250000000/test hdfs://10.0.0.3:9000/
```

Distcp 명령 라인 설정 매개변수를 지정하여 HDFS와 COS 간에 데이터 마이그레이션 진행

설명：

해당 명령 라인 설정은 양방향 작업을 지원하여 HDFS 데이터를 COS에 마이그레이션하거나 COS 데이터를 HDFS에 마이그레이션할 수 있습니다.

사용자는 다음 명령어를 직접 설정할 수 있습니다.



```
hadoop distcp -Dfs.cosn.impl=org.apache.hadoop.fs.CosFileSystem -Dfs.cosn.bucket.re
```

매개변수 설명은 다음과 같습니다.

Dfs.cosn.impl: 항상 org.apache.hadoop.fs.CosFileSystem으로 설정합니다.

Dfs.cosn.bucket.region: 버킷의 소재 리전을 입력합니다. COS 콘솔 버킷 리스트에서 확인할 수 있습니다.

Dfs.cosn.userinfo.secretId: 버킷 소유자 계정의 SecretId를 입력합니다. Tencent Cloud API 키에서 획득할 수 있습니다.

Dfs.cosn.userinfo.secretKey: 버킷 소유자 계정의 secretKey를 입력합니다. Tencent Cloud API 키에서 획득할 수 있습니다.

libjars: Hadoop-COS jar 패키지 위치를 지정합니다. Hadoop-COS jar 패키지는 Github 웨어하우스의 dep 디렉터리에서 다운로드할 수 있습니다.

설명：

기타 매개변수는 Hadoop 툴 문서를 참조하십시오.

Hadoop distcp의 확장 매개변수

Hadoop distcp 툴은 다양한 실행 매개변수를 지원합니다. 예를 들어 `-m` 으로 동시 복사에 쓰이는 Map의 최대 작업 수를 지정할 수 있으며, `-bandwidth` 로 map마다 사용하는 최대 대역폭 등을 제한할 수 있습니다. 자세한 내용은 Apache Hadoop distcp 툴의 공식 홈페이지 문서 DistCp Guide를 참조하십시오.

AWS S3 SDK를 사용하여 COS에 액세스하기

최종 업데이트 날짜: : 2024-06-24 16:44:04

소개

Cloud Object Storage(COS)는 AWS S3 호환 API를 제공합니다. 데이터를 S3에서 COS로 마이그레이션한 후에 간단한 설정 수정만으로 클라이언트 애플리케이션을 손쉽게 COS 서비스와 호환 사용할 수 있습니다. 본 문서는 다양한 개발 플랫폼의 S3 SDK 적용 절차를 소개합니다. 이 추가 적용 절차가 끝나면 S3 SDK 인터페이스를 사용해 COS 파일에 액세스할 수 있습니다.

준비 작업

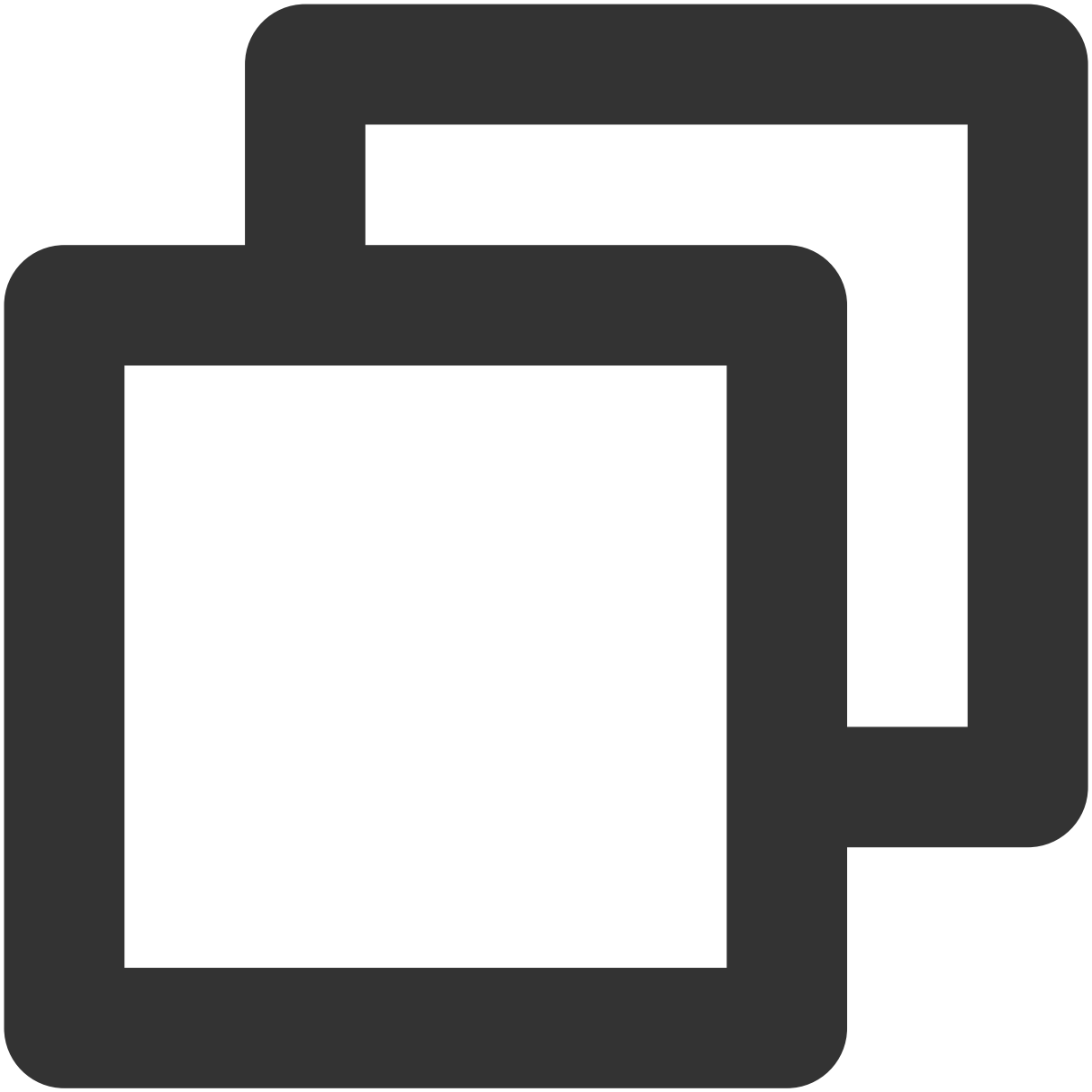
1. Signing Up하고 CAM 콘솔에서 Tencent Cloud 키 SecretID와 SecretKey를 얻습니다.
2. 이제 S3 SDK를 통합해 정상 실행할 수 있는 클라이언트 애플리케이션이 준비되었습니다.

Android

다음은 AWS Android SDK 2.14.2 버전을 예시로 COS 서비스에 액세스하기 위한 방법을 소개합니다. 디바이스가 COS에 액세스하는 경우 영구 키를 클라이언트 코드에 두면 노출 위험이 커지므로 STS 서비스에 연결해 임시 키를 사용하십시오. 자세한 내용은 임시 키 생성 및 사용 가이드를 참조하십시오.

초기화

인스턴스를 초기화할 때 임시 키 제공자와 Endpoint 설정이 필요합니다. 버킷이 위치한 리전이 `ap-guangzhou` 인 경우를 예로 들어봅니다.



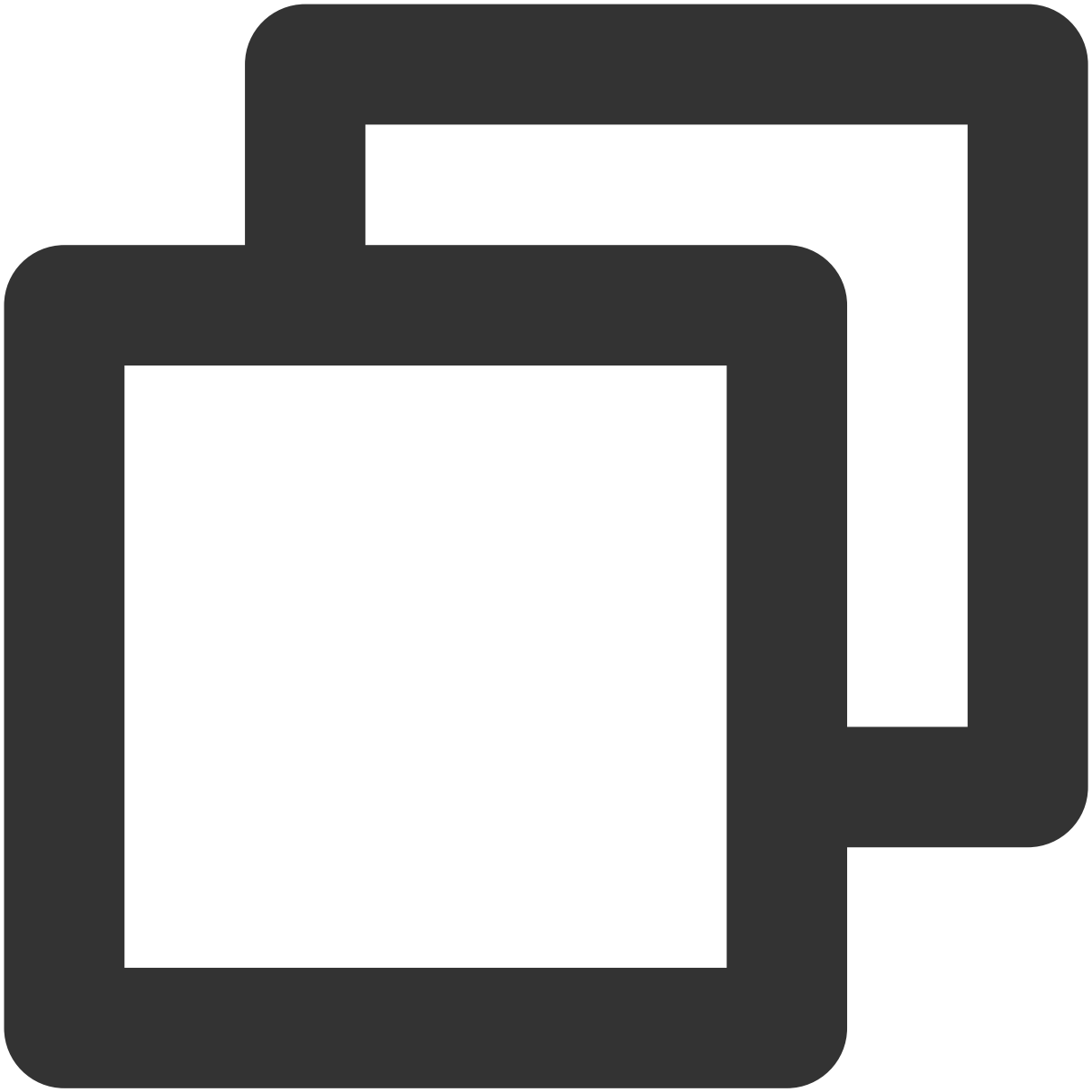
```
AmazonS3Client s3 = new AmazonS3Client(new AWSCredentialsProvider() {
    @Override
    public AWSCredentials getCredentials() {
         // 이 백그라운드에서 STS로 임시 키 정보를 요청합니다.
        return new BasicSessionCredentials(
                "<TempSecretID>", "<TempSecretKey>", "<STSSessionToken>"
        );
    }

    @Override
    public void refresh() {
```

```
            //
        }
});

s3.setEndpoint("cos.ap-guangzhou.myqcloud.com");
```

iOS

AWS iOS SDK 2.10.2 버전을 예시로 COS 서비스에 액세스하기 위한 방법을 소개합니다. 디바이스가 COS에 액세스하는 경우 영구 키를 클라이언트 코드에 두면 노출 위험이 커지므로 STS 서비스에 연결해 임시 키를 사용하십시오. 자세한 내용은 임시 키 생성 및 사용 가이드를 참조하십시오.

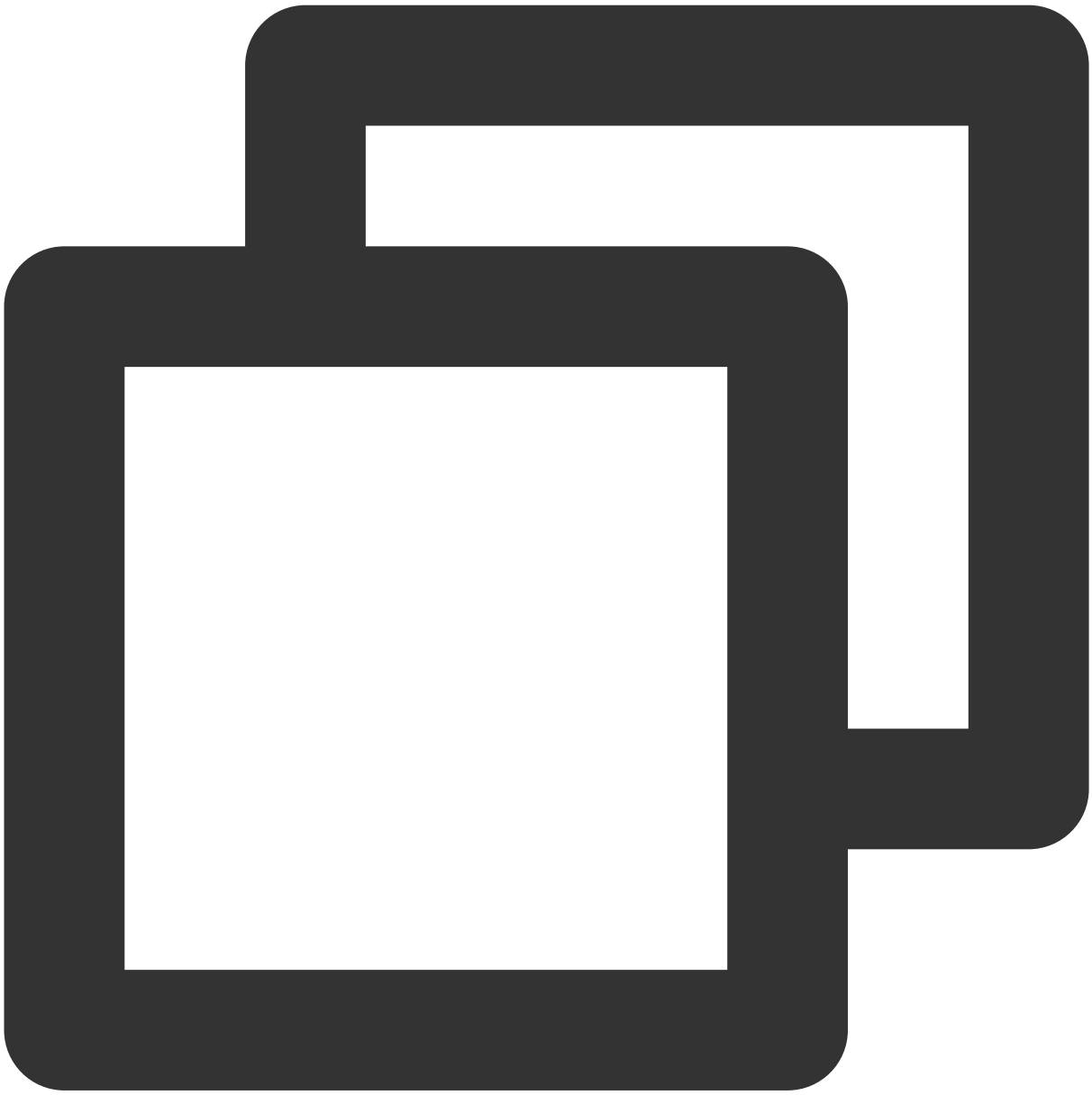
1. AWSCredentialsProvider 프로토콜 실행



```
-(AWSTask<AWSCredentials *> *)credentials{
    // 이 백그라운드에서 STS로 임시 키 정보를 요청합니다.
    AWSCredentials *credential = [[AWSCredentials alloc]initWithAccessKey:@"<TempSe

    return [AWSTask taskWithResult:credential];

}

- (void)invalidateCachedTemporaryCredentials{

}
```

2. 임시 키 제공자와 Endpoint 제공

버킷이 위치한 리전이 `ap-guangzhou` 인 경우



```
NSURL* bucketURL = [NSURL URLWithString:@"http://cos.ap-guangzhou.myqcloud.com"];

AWSEndpoint* endpoint = [[AWSEndpoint alloc] initWithRegion:AWSRegionUnknown servic
AWSServiceConfiguration* configuration = [[AWSServiceConfiguration alloc]
    initWithRegion:AWSRegionUSEast2 endpoint:endpoint
    credentialsProvider:[MyCredentialProvider new]]; // MyCredentialProvider가 AWSC
```

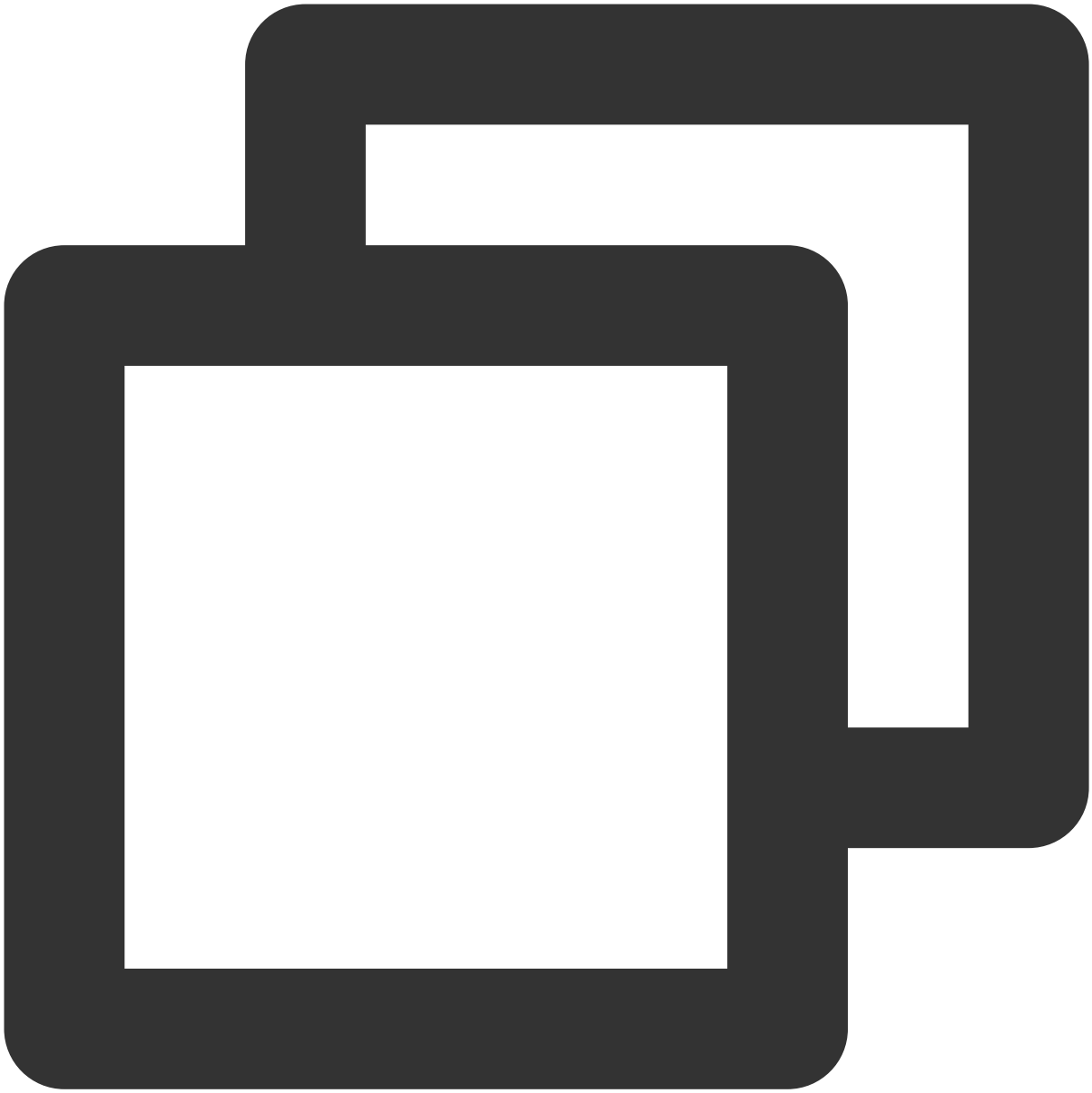
```
[[AWSServiceManager defaultServiceManager] setDefaultServiceConfiguration:configura
```

Node.js

AWS JS SDK 2.509.0 버전을 예시로 COS 서비스에 액세스하기 위한 방법을 소개합니다.

초기화

인스턴스를 초기화할 때 Tencent Cloud 키와 Endpoint를 설정합니다. 버킷이 위치한 리전이 `ap-guangzhou` 라고 가정했을 경우 코드는 아래와 같이 표시됩니다.



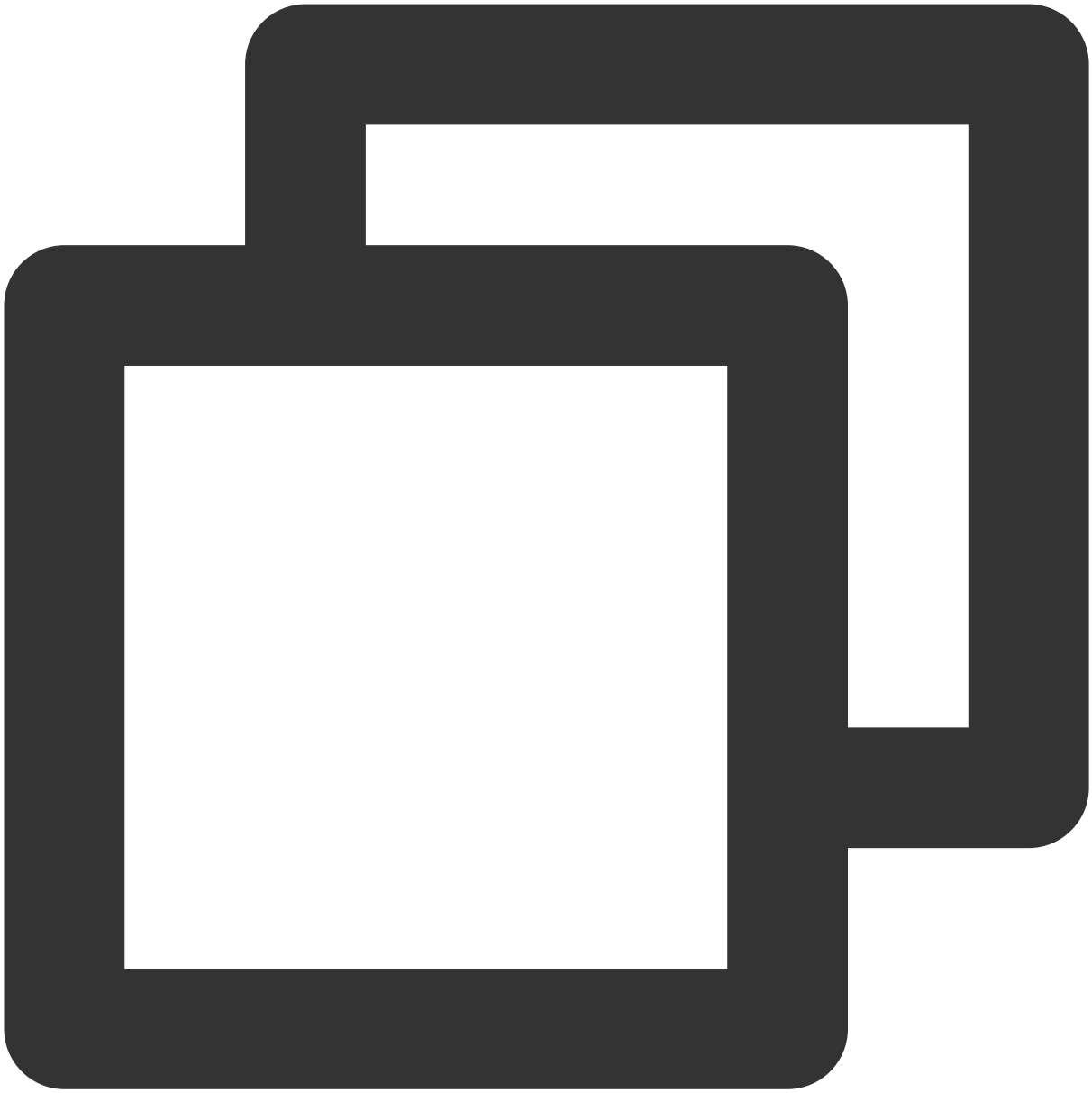
```
var AWS = require('aws-sdk');

AWS.config.update({
    accessKeyId: "COS_SECRETID",
    secretAccessKey: "COS_SECRETKEY",
    region: "ap-guangzhou",
    endpoint: 'https://cos.ap-guangzhou.myqcloud.com',
});

s3 = new AWS.S3({apiVersion: '2006-03-01'});
```

Java

다음은 AWS Java SDK 1.11.609 버전을 예시로 COS 서비스에 액세스하기 위한 방법을 소개합니다.

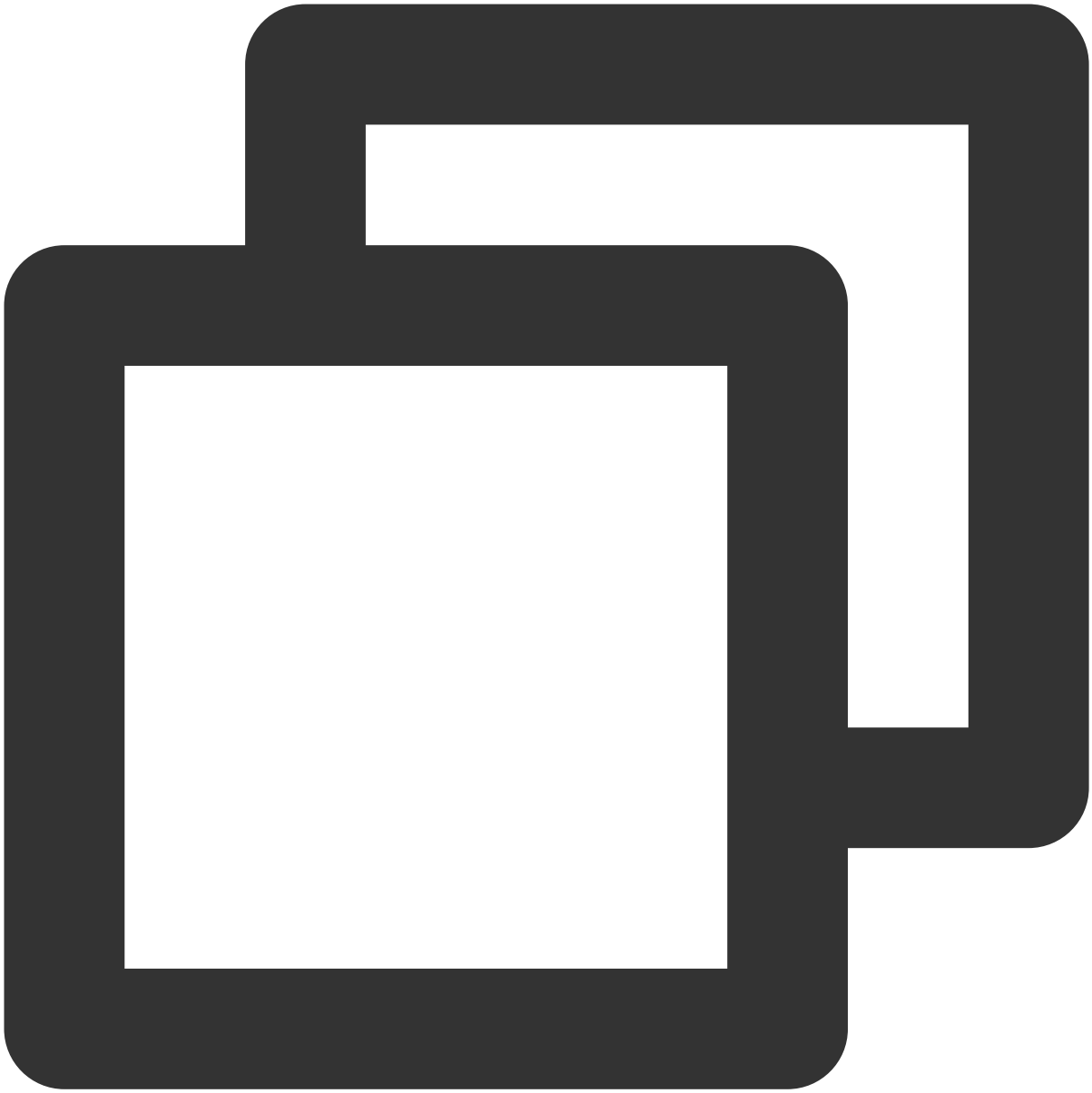
1. AWS 구성 및 인증서 파일 수정하기

설명：
다음은 Linux를 예시로 AWS 구성 및 인증서 파일을 수정합니다.
AWS SDK 기본 구성 파일은 사용자 디렉터리에서 구성 및 인증서 파일을 참고하십시오.
설정파일(파일 위치는 `~/.aws/config`)에서 다음 설정을 추가합니다.- 구성 파일(파일 위치는 `~/.aws/config`)에서 다음 설정 정보를 추가합니다.



```
[default]
s3 =
addressing_style = virtual
```

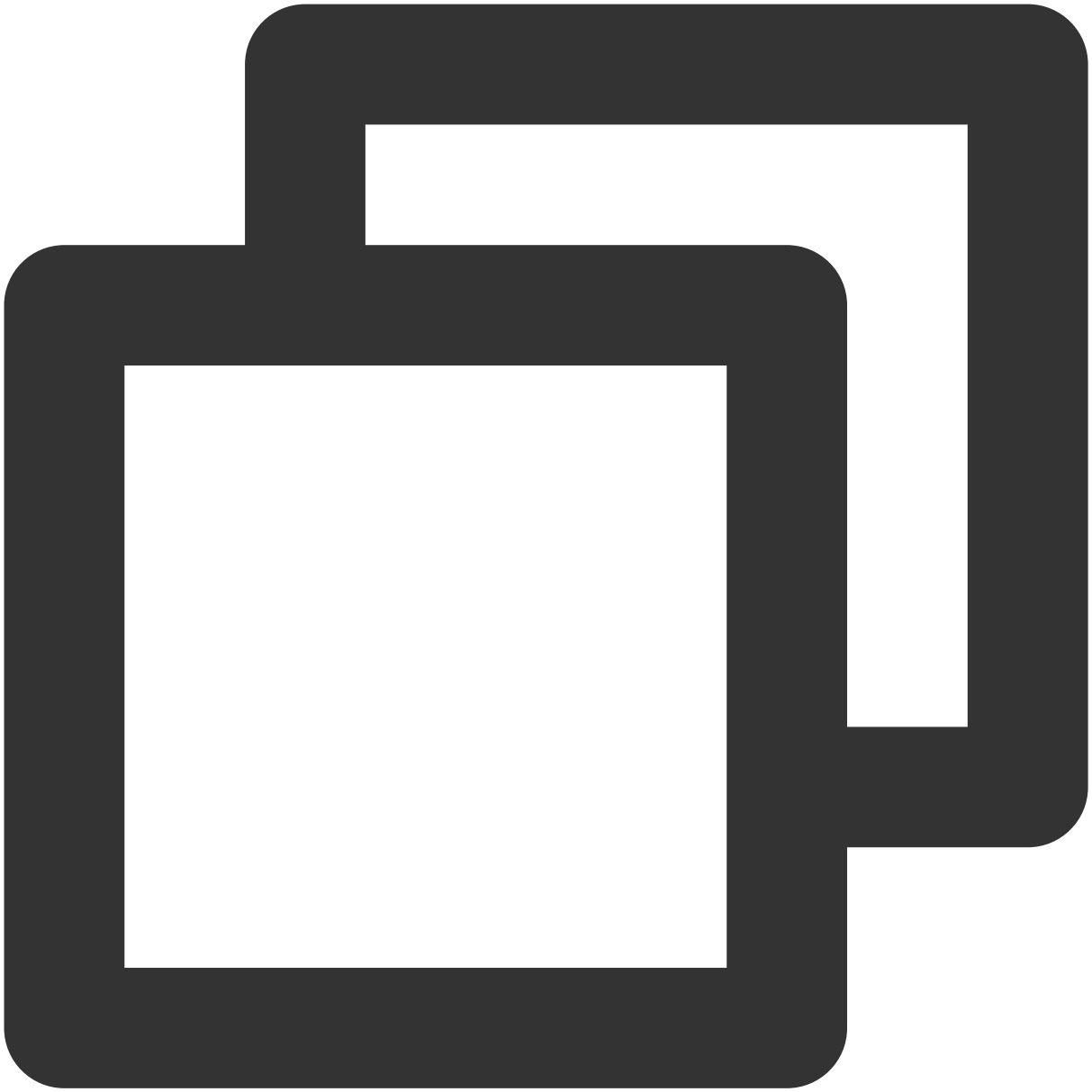
인증서 파일(파일 위치는 `~/.aws/credentials`)에서 Tencent Cloud 키를 설정합니다.



```
[default]
aws_access_key_id = [COS_SECRETID]
aws_secret_access_key = [COS_SECRETKEY]
```

2. 코드에서 Endpoint 설정하기

버킷이 위치한 리전이 `ap-guangzhou` 라고 가정했을 때 코드는 다음과 같습니다.



```
AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
    .withEndpointConfiguration(new AwsClientBuilder.EndpointConfiguration(
            "http://cos.ap-guangzhou.myqcloud.com",
            "ap-guangzhou"))
    .build();
```

Python

다음은 AWS Python SDK 1.9.205 버전을 예시로 COS 서비스에 액세스하기 위한 방법을 소개합니다.

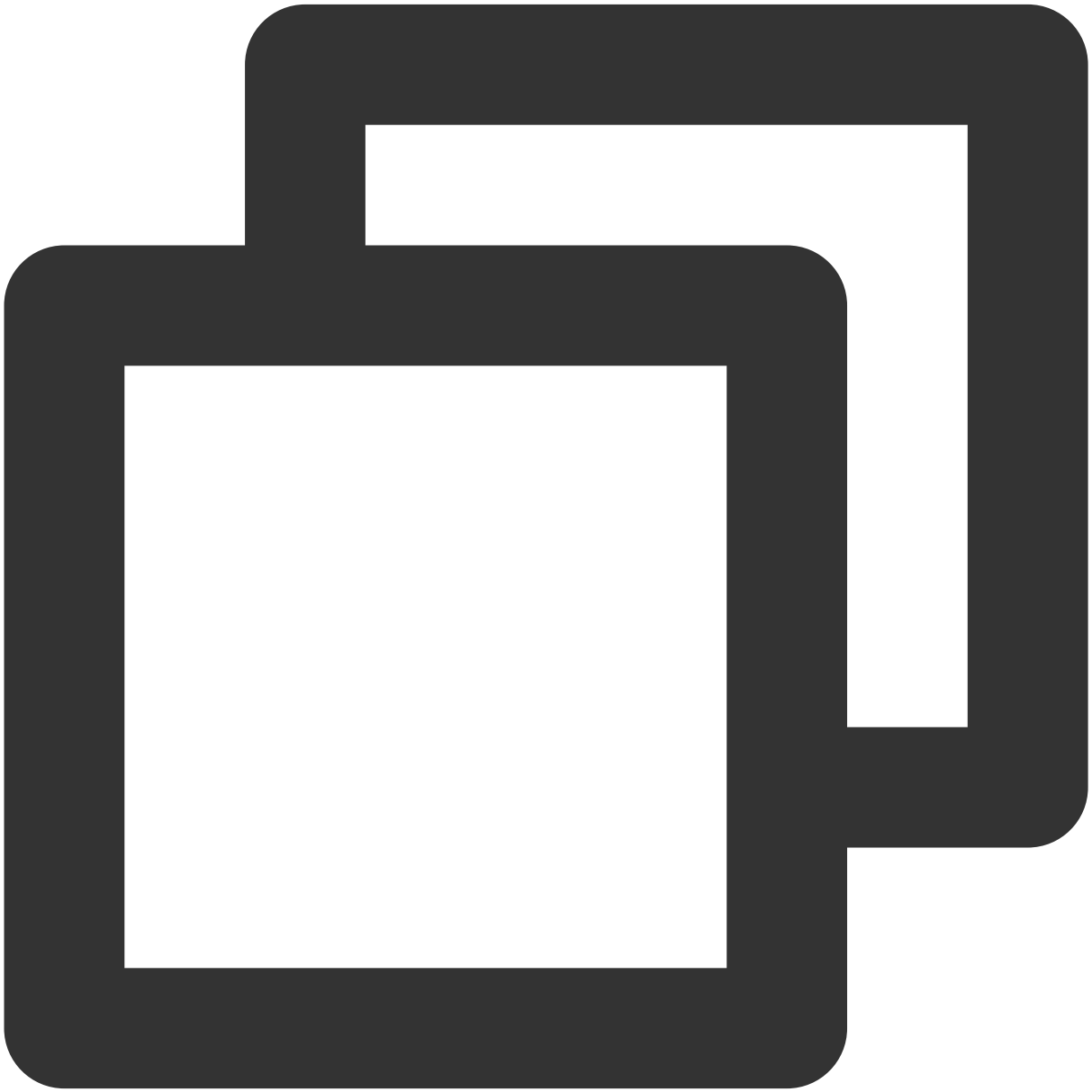
1. AWS 구성 및 인증서 파일 수정하기

설명：

다음은 Linux를 예시로 AWS 구성 및 인증서 파일을 수정합니다.

AWS SDK 기본 구성 파일은 사용자 디렉터리에서 구성 및 인증서 파일을 참고하십시오.

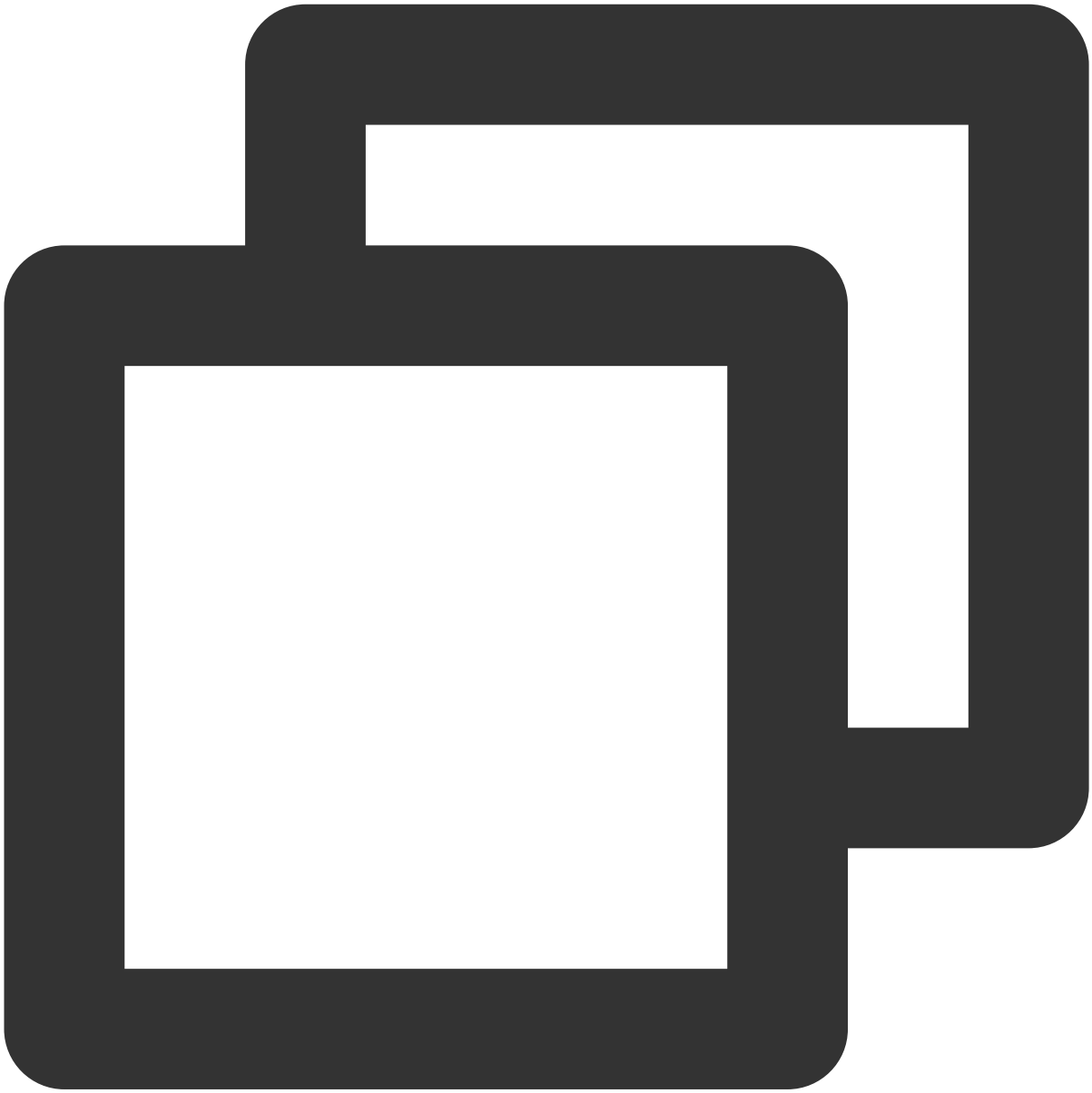
구성 파일(파일 위치는 `~/.aws/config`)에서 다음 설정을 추가합니다.



```
[default]
s3 =
```

```
    signature_version = s3
    addressing_style = virtual
```

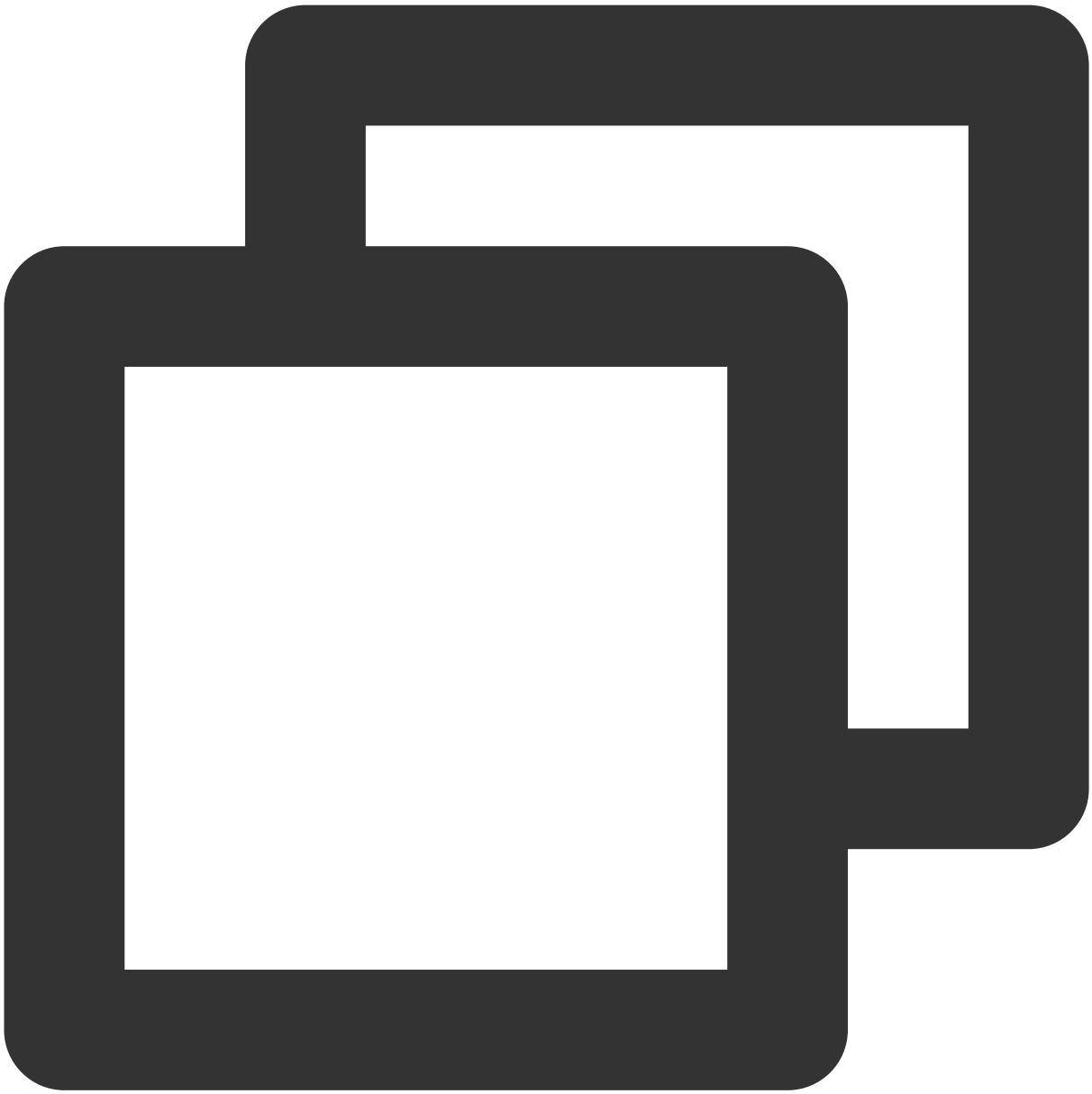
인증서 파일(파일 위치는 `~/.aws/credentials`)에서 Tencent Cloud 키를 설정합니다.



```
[default]
aws_access_key_id = [COS_SECRETID]
aws_secret_access_key = [COS_SECRETKEY]
```

2. 코드에서 Endpoint 설정하기

버킷이 위치한 리전이 `ap-guangzhou` 인 경우



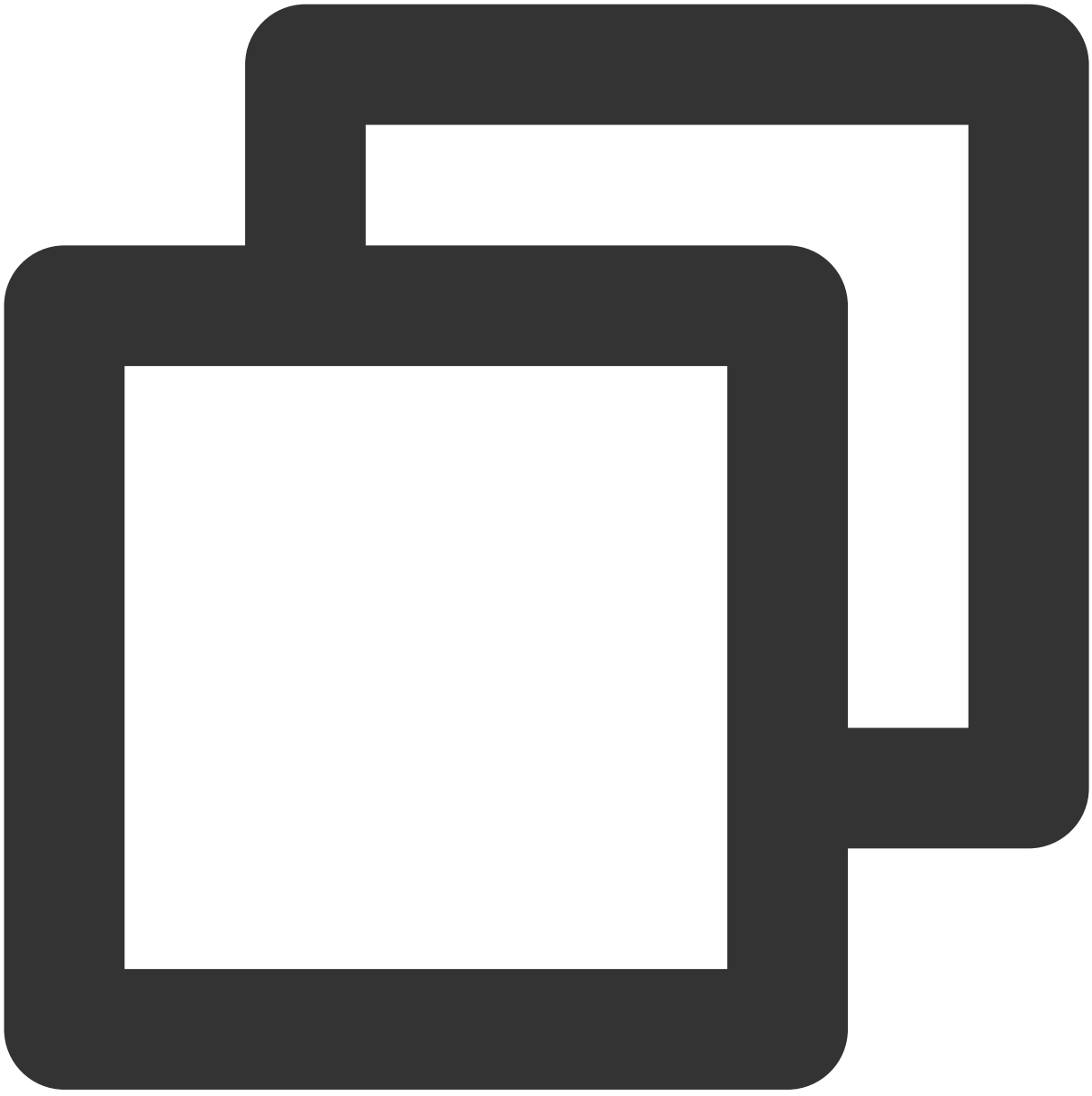
```
client = boto3.client('s3', endpoint_url='https://cos.ap-guangzhou.myqcloud.com')
```

PHP

다음은 AWS PHP SDK 3.109.3 버전을 예시로 COS 서비스에 액세스하기 위한 방법을 소개합니다.

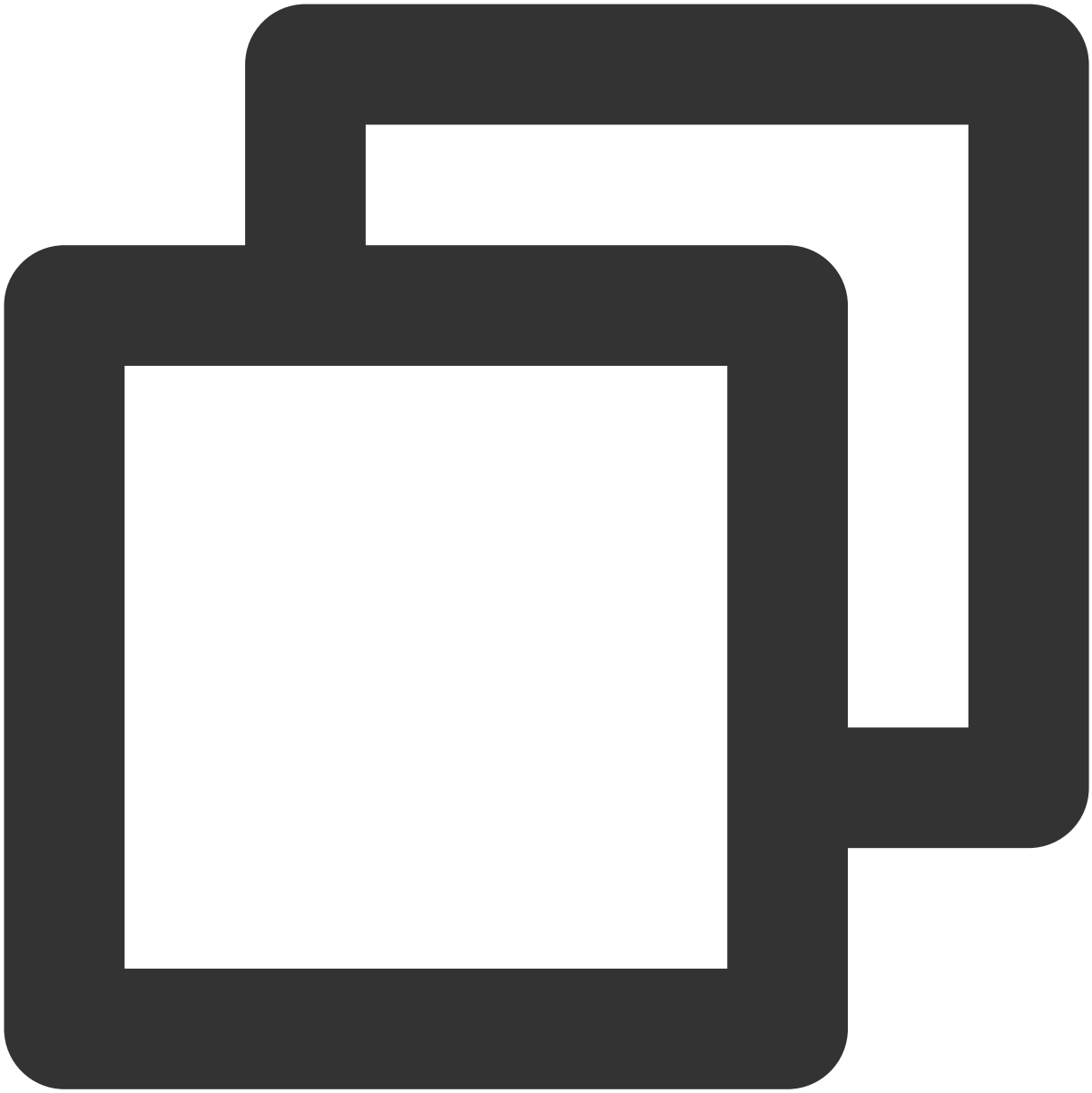
1. AWS 구성 및 인증서 파일 수정하기

설명：
다음은 Linux를 예시로 AWS 구성 및 인증서 파일을 수정합니다.
AWS SDK 기본 구성 파일은 사용자 디렉터리에서 구성 및 인증서 파일을 참고하십시오.
구성 파일(파일 위치는 `~/.aws/config`)에서 다음 설정을 추가합니다.



```
[default]
s3 =
addressing_style = virtual
```

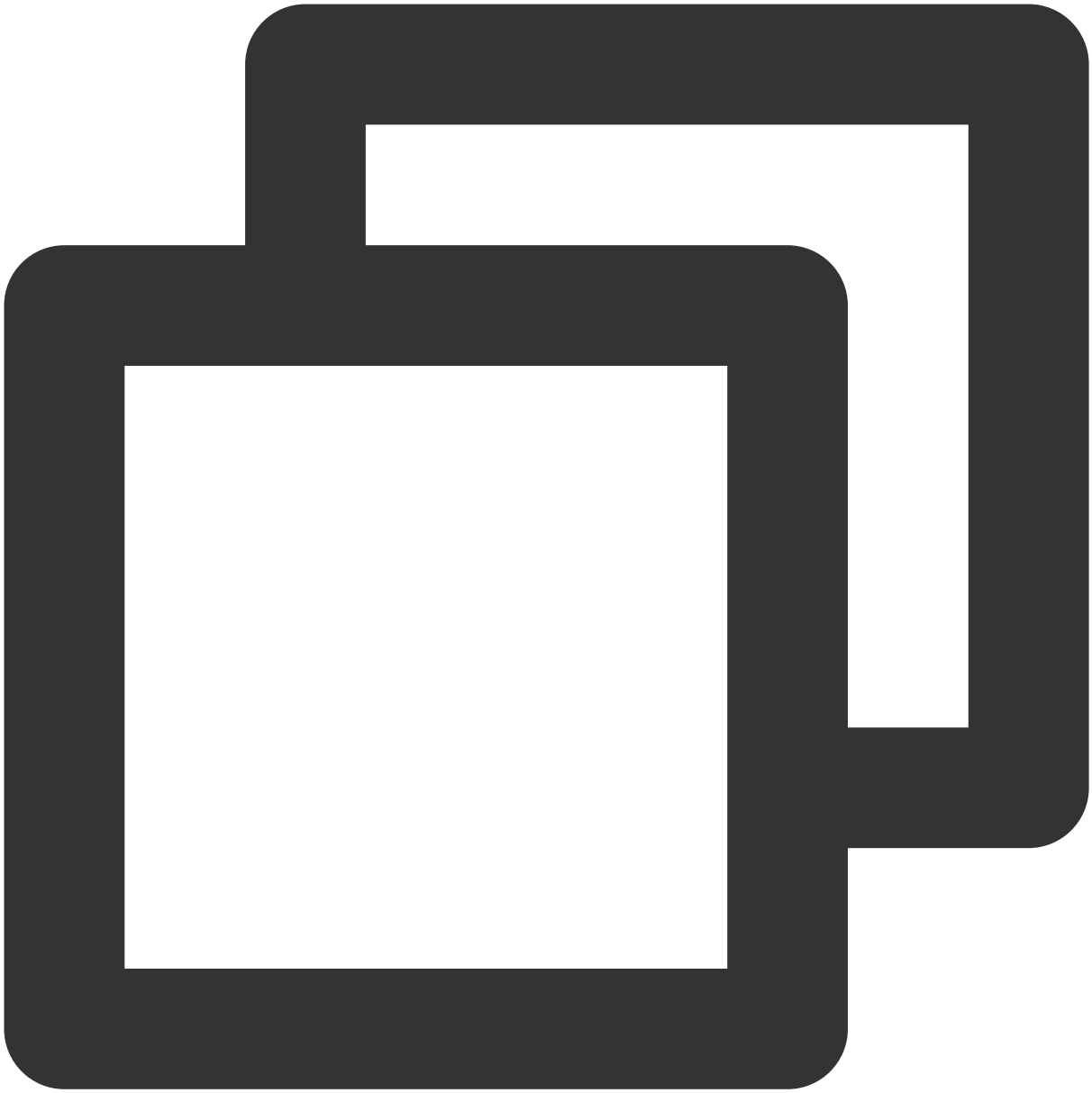
인증서 파일(파일 위치는 `~/.aws/credentials`)에서 Tencent Cloud 키를 설정합니다.



```
[default]
aws_access_key_id = [COS_SECRETID]
aws_secret_access_key = [COS_SECRETKEY]
```

2. 코드에서 Endpoint 설정하기

버킷이 위치한 리전이 `ap-guangzhou` 인 경우



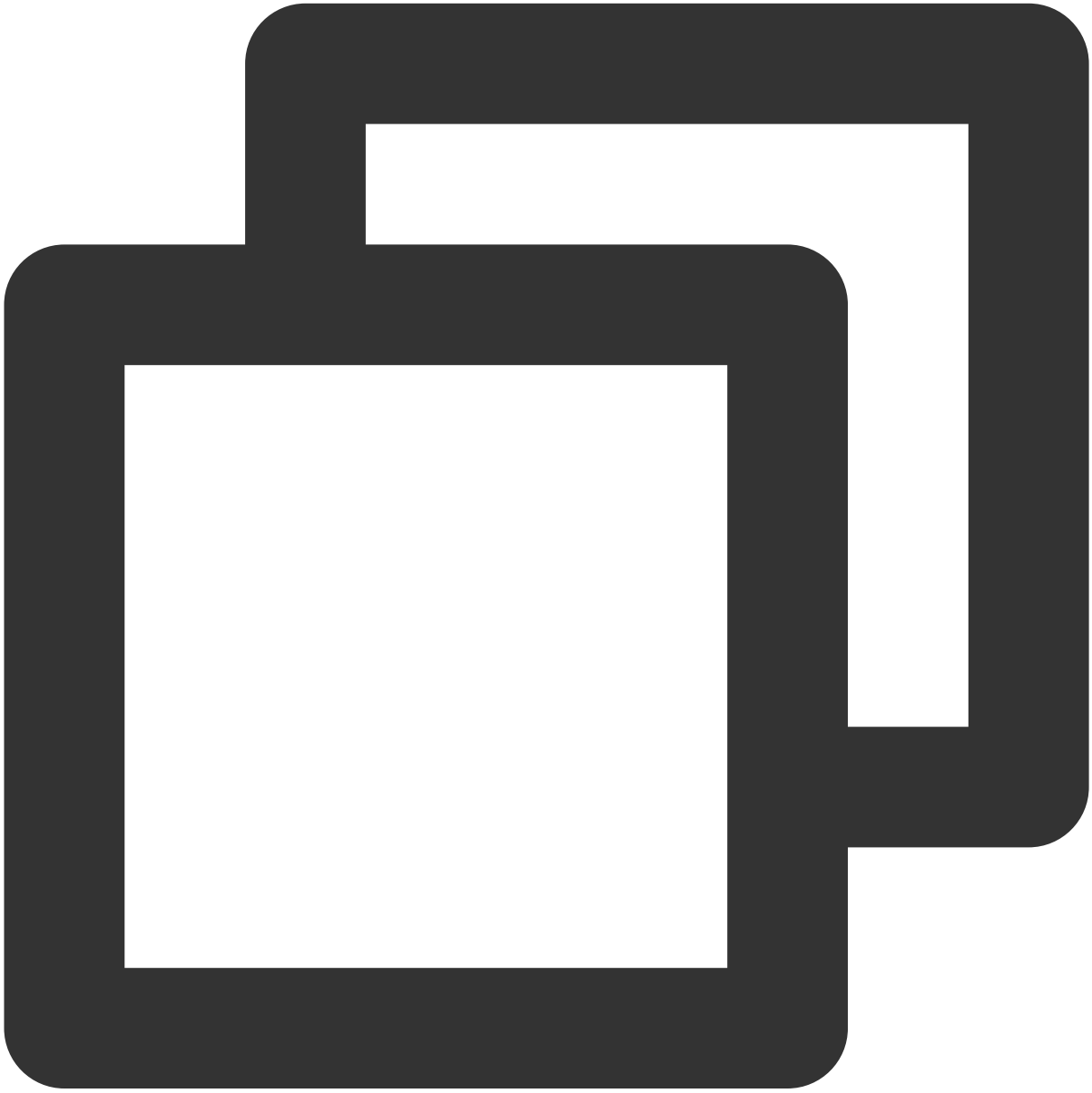
```
$S3Client = new S3Client([
  "Region": "ap-guangzhou",
  'version'         => '2006-03-01',
  'endpoint'        => 'https://cos.ap-guangzhou.myqcloud.com'
]);
```

.NET

다음은 AWS .NET SDK 3.3.104.12 버전을 예시로 COS 서비스에 액세스하기 위한 방법을 소개합니다.

초기화

인스턴스를 초기화할 때 Tencent Cloud 키와 Endpoint를 설정합니다. 버킷이 속한 리전이 `ap-guangzhou` 라고 가정했을 경우



```
string sAccessKeyId = "COS_SECRETID";
string sAccessKeySecret = "COS_SECRETKEY";
string region = "ap-guangzhou";

var config = new AmazonS3Config() { ServiceURL = "https://cos." + region + ".myqclo
```

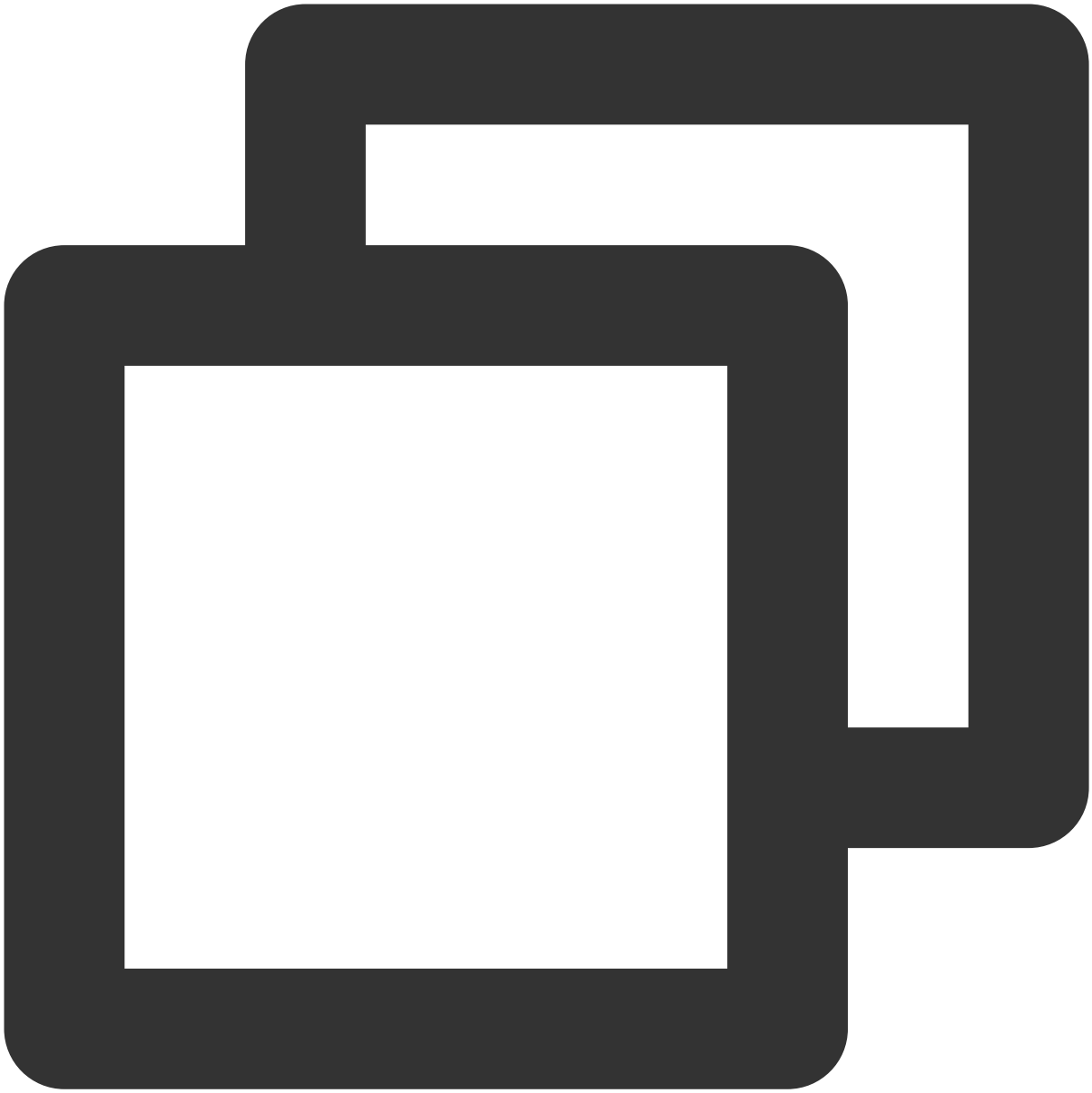
```
var client = new AmazonS3Client(sAccessKeyId, sAccessKeySecret, config);
```

Go

다음은 AWS Go SDK 1.21.9 버전을 예시로 COS 서비스에 액세스하기 위한 방법을 소개합니다.

1. 키에 따라 session 생성하기

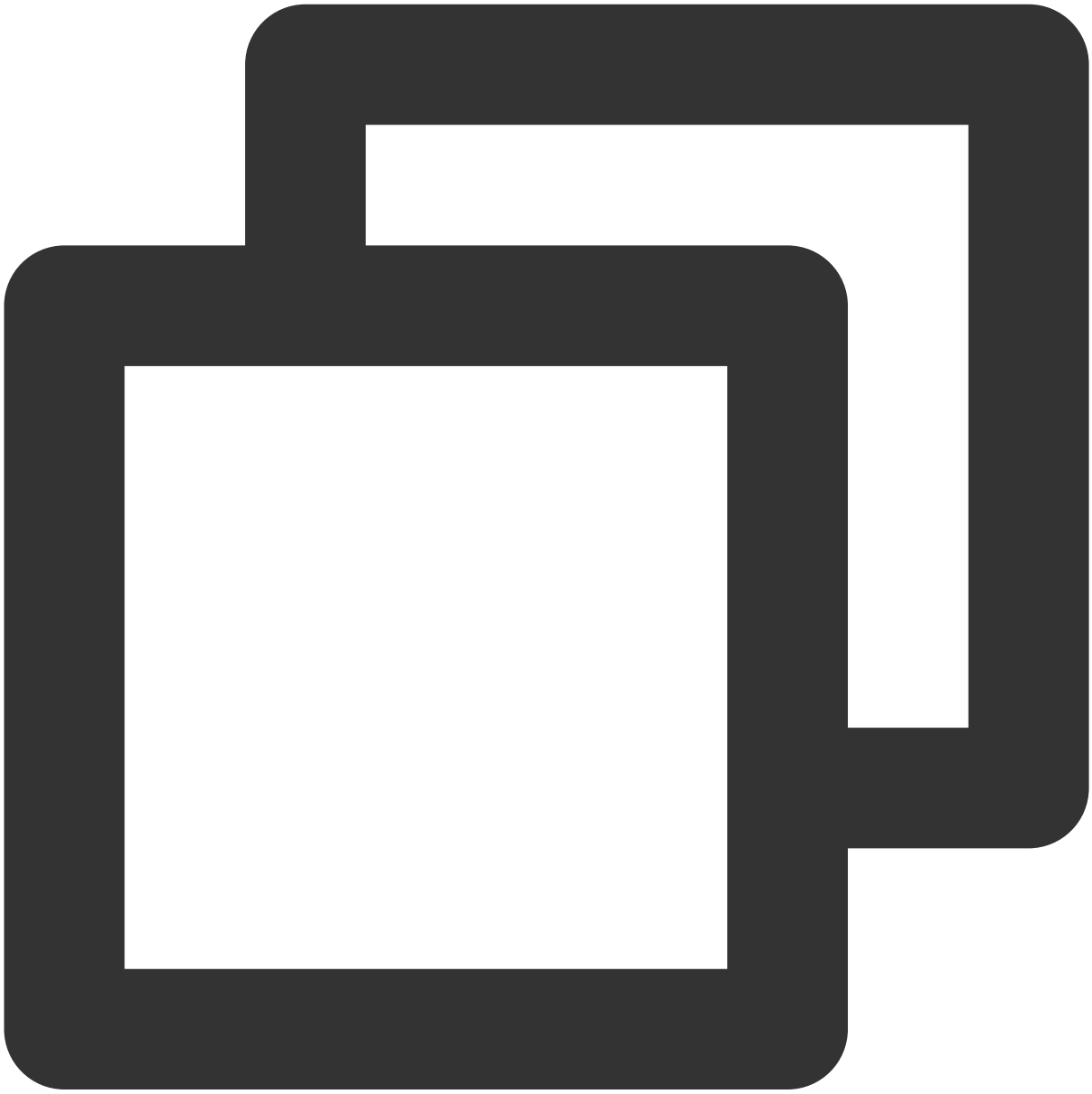
버킷이 위치한 리전이 `ap-guangzhou` 인 경우



```
func newSession() (*session.Session, error) {
    creds := credentials.NewStaticCredentials("COS_SECRETID", "COS_SECRETKEY", "")
    "Region": "ap-guangzhou",
    endpoint := "http://cos.ap-guangzhou.myqcloud.com"
    config := &aws.Config{
        Region:           aws.String(region),
        Endpoint:         &endpoint,
        S3ForcePathStyle: aws.Bool(true),
        Credentials:      creds,
        // DisableSSL:       &disableSSL,
    }
```

```
    return session.NewSession(config)
}
```

2. session에 따라 server 생성 요청 발송하기



```
sess, _ := newSession()
service := s3.New(sess)

// 파일 업로드의 경우
fp, _ := os.Open("yourLocalFilePath")
defer fp.Close()
```

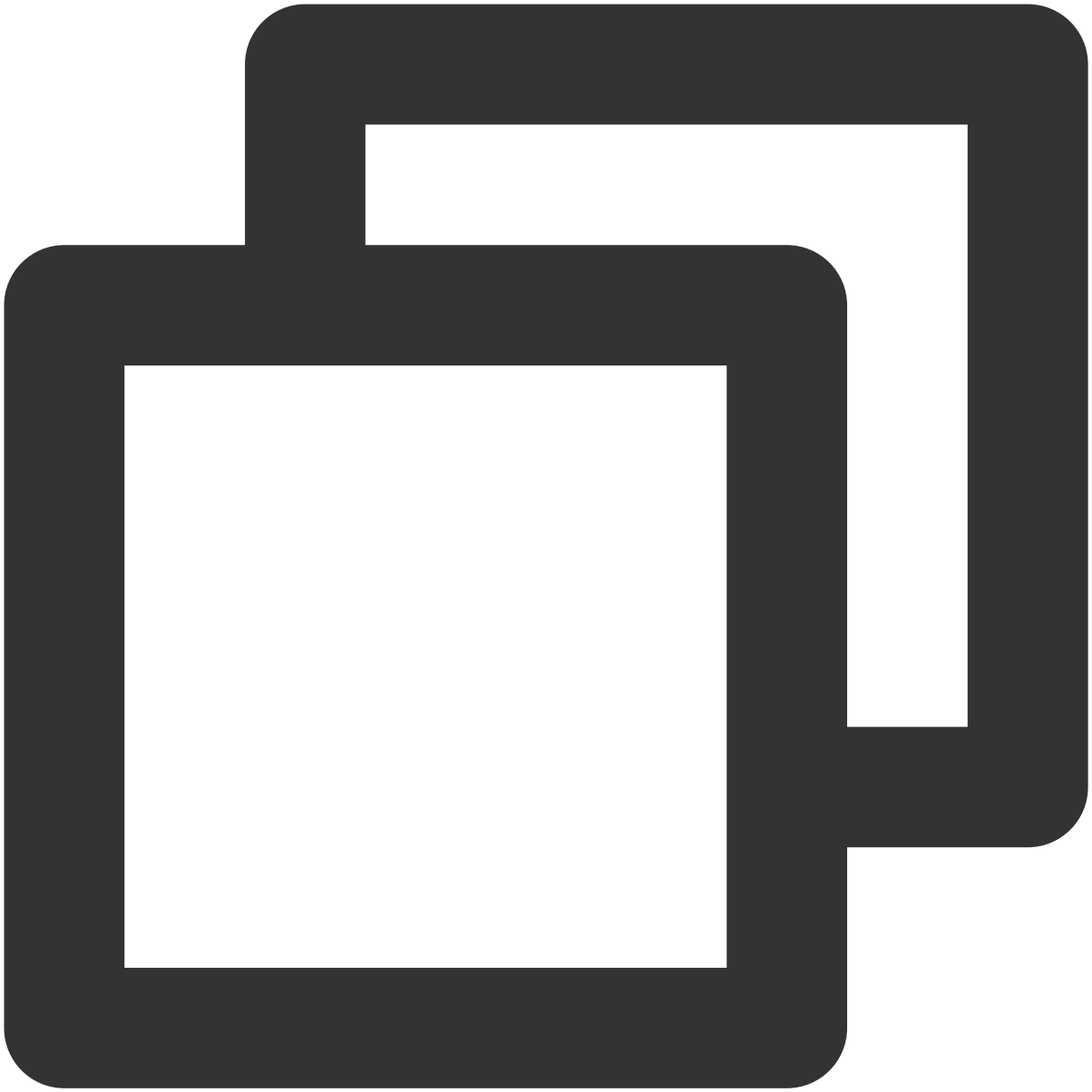
```
ctx, cancel := context.WithTimeout(context.Background(), time.Duration(30)*time.Sec
defer cancel()

service.PutObjectWithContext(ctx, &s3.PutObjectInput{
    Bucket: aws.String("examplebucket-1250000000"),
    Key:    aws.String("exampleobject"),
    Body:   fp,
})
```

C++

다음은 AWS C++ SDK 1.7.68 버전을 예시로 COS 서비스에 액세스하기 쉽게 적용하는 방법을 소개합니다.

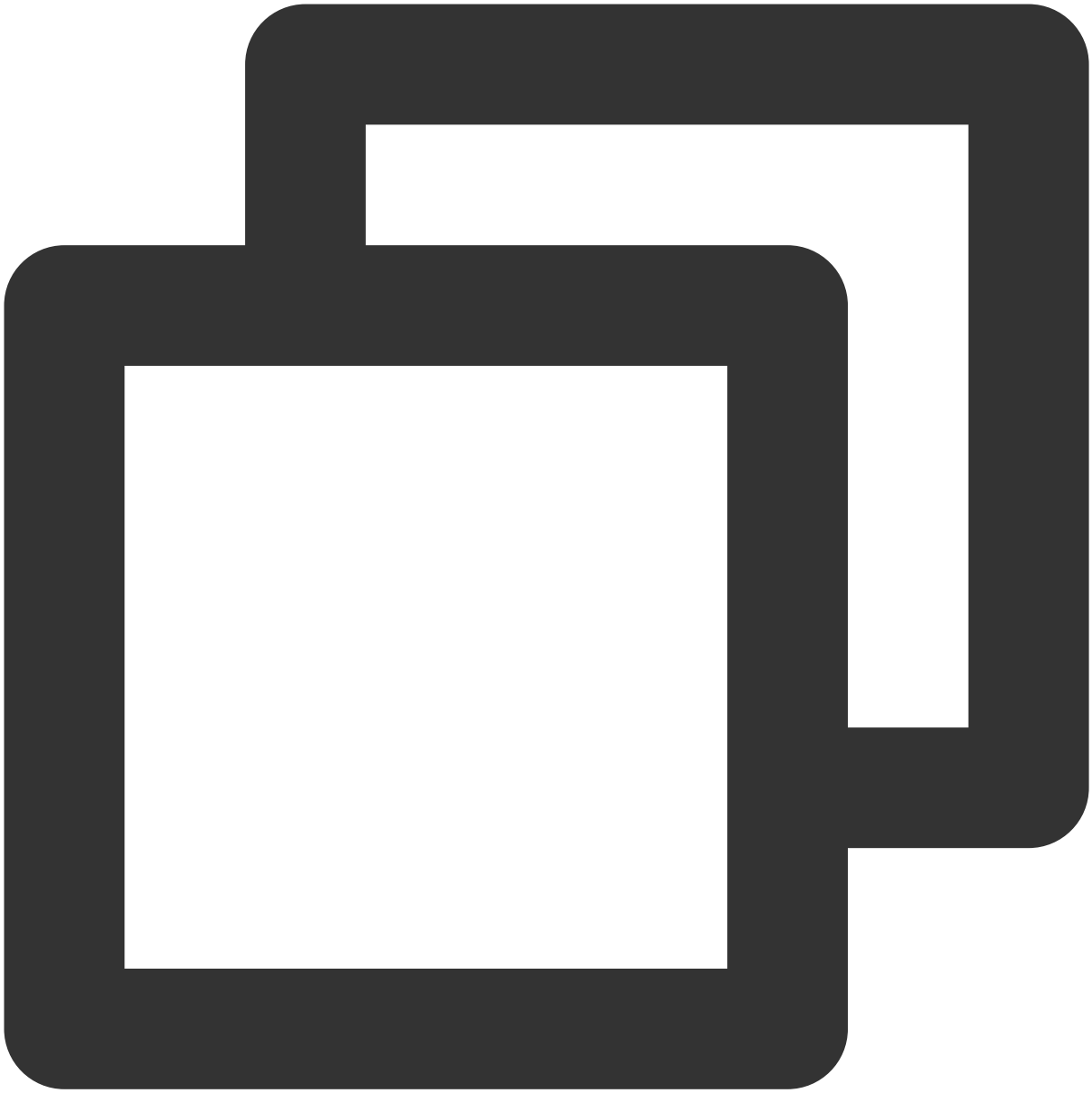
1. AWS 구성 및 인증서 파일 수정하기

설명：
다음은 Linux를 예시로 AWS 구성 및 인증서 파일을 수정합니다.
AWS SDK 기본 구성 파일은 사용자 디렉터리에서 구성 및 인증서 파일을 참고하십시오.
구성 파일(파일 위치는 `~/.aws/config`)에서 다음 설정을 추가합니다.



```
[default]
s3 =
addressing_style = virtual
```

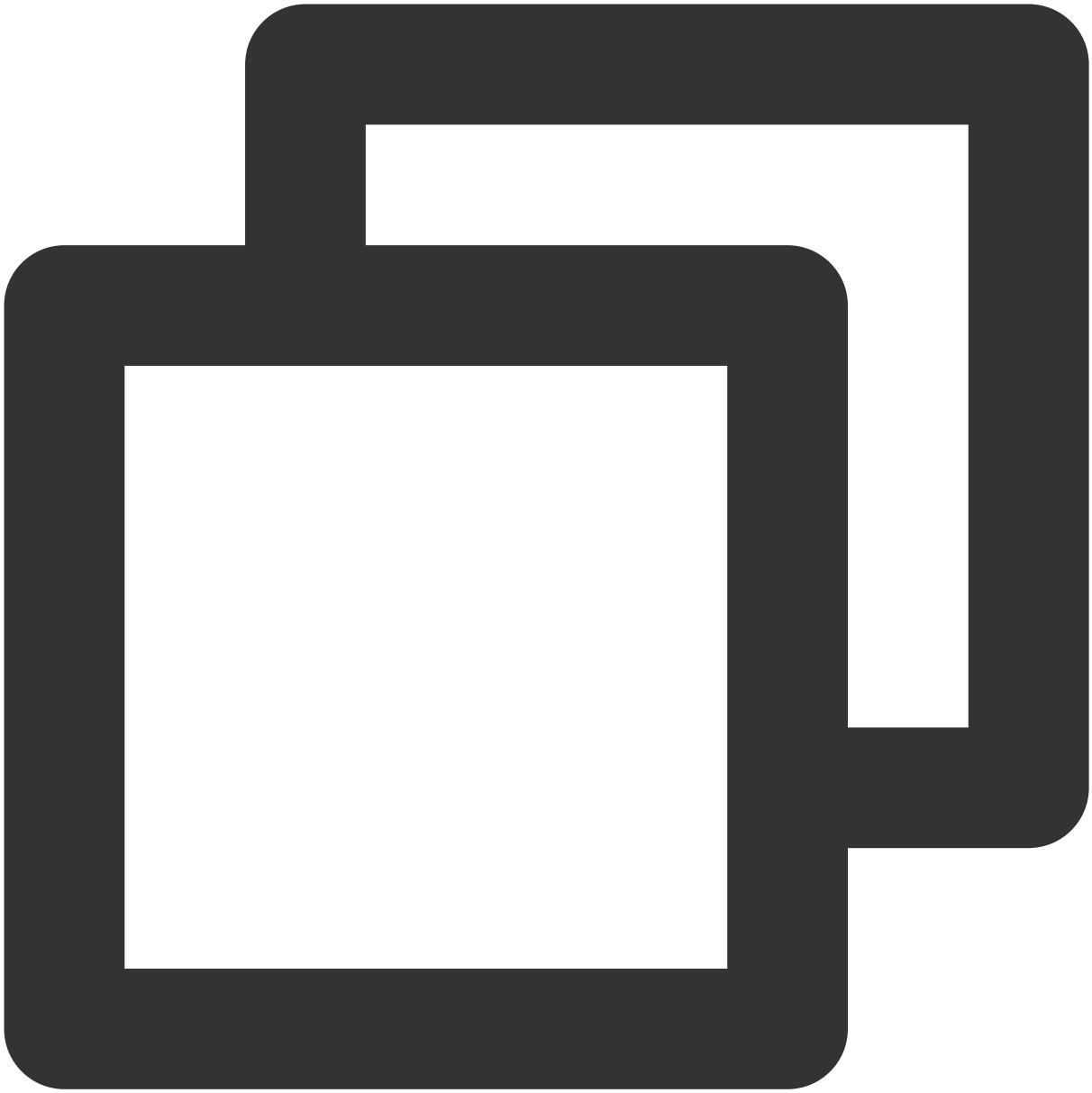
인증서 파일(파일 위치는 `~/.aws/credentials`)에서 Tencent Cloud 키를 설정합니다.



```
[default]
aws_access_key_id = [COS_SECRETID]
aws_secret_access_key = [COS_SECRETKEY]
```

2. 코드에서 Endpoint 설정하기

버킷이 위치한 리전이 `ap-guangzhou` 라고 가정했을 때 코드는 다음과 같습니다.



```
Aws::Client::ClientConfiguration awsCC;
awsCC.scheme = Aws::Http::Scheme::HTTP;
awsCC.region = "ap-guangzhou";
awsCC.endpointOverride = "cos.ap-guangzhou.myqcloud.com";
Aws::S3::S3Client s3_client(awsCC);
```

데이터 재해 복구 백업
버킷 복사 기반의 재해 복구 고가용성 아키텍처

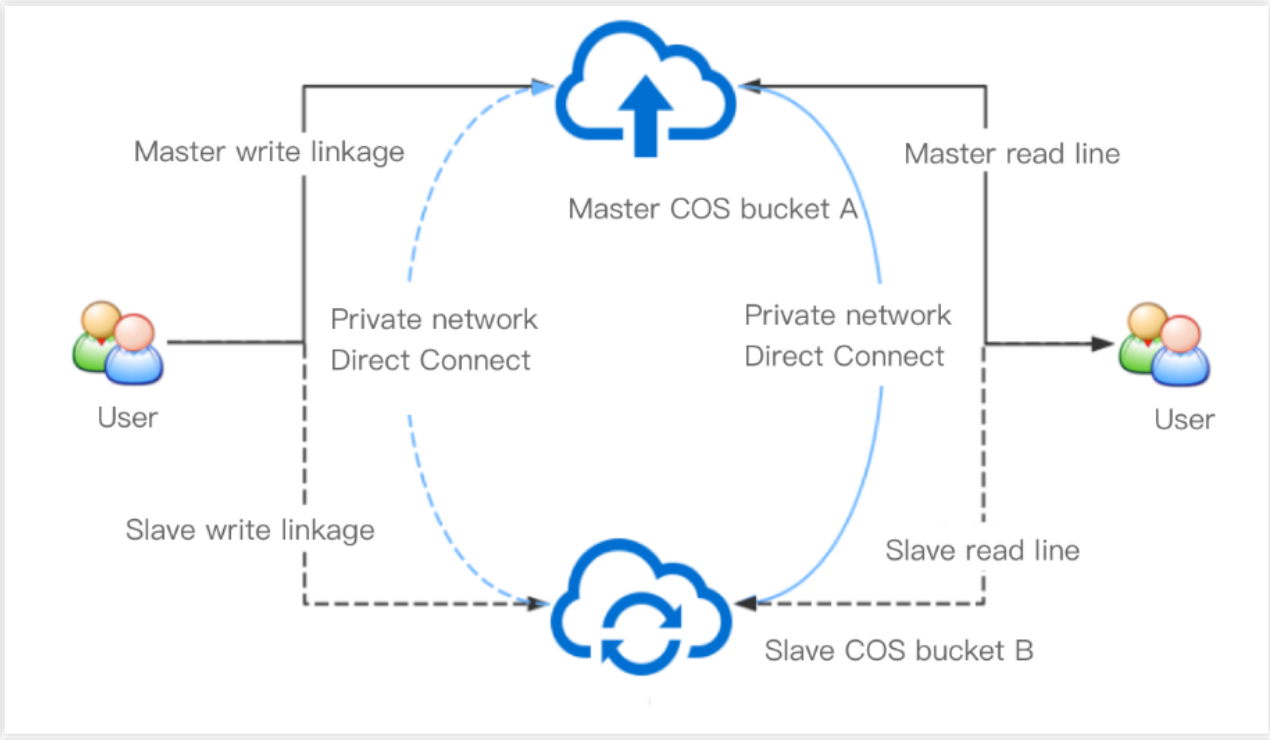
최종 업데이트 날짜: : 2024-06-24 16:44:04

소개

Tencent Cloud COS는 99.95%의 가용성과 99.999999999%의 신뢰성을 자부합니다. 하지만 자연재해, 광케이블 장애와 같은 불가항력적 요인들로 인해 클라우드 데이터의 가용성과 신뢰성이 100%에 도달할 수는 없습니다. 반면 금융업과 같은 일부 업종은 그 특수성으로 인해 상당히 높은 수준의 가용성과 신뢰성이 요구됩니다.
Tencent Cloud COS는 기업 비즈니스의 연속성과 안정성을 도모하여 기업의 고가용성 및 고신뢰성에 대한 요구사항을 충족하기 위해 버킷 복사 기능을 기반으로 한 고가용성의 데이터 재해 복구 솔루션을 제공합니다. 기업은 클라우드 이용 시 비즈니스 요구사항에 맞춰 클라우드 내 데이터를 재해 복구하고 백업함으로써 비즈니스를 지속적이며 안정적으로 운영할 수 있습니다.
본 문서에서는 버킷 복사를 기반으로 한 클라우드 비즈니스의 액티브/스탠바이 전환이라는 재해 복구 솔루션과, 버킷 복사를 기반으로 한 고가용성 솔루션의 두 가지 내용을 주로 소개합니다. 버킷 복사, Origin-pull, SCF, CDN 등 다양한 제품과 기능을 통해 비즈니스 가용성을 향상시킬 수 있습니다.

버킷 복사 기반의 재해 복구 백업 솔루션

재해 복구를 위해서는 이중화(Redundance), 원거리(Remote), 데이터 전체 백업(Replication)라는 세 가지 요소가 충족돼야 합니다.
이중화: 데이터 이중화를 뜻하며, 데이터를 사용 가능한 다른 시스템으로 백업해야 합니다.
원거리: 백업 데이터를 원거리의 다른 리전에 보관해야 합니다. 재해는 보통 인근 지역으로 여파가 미치기 때문에, 충분한 거리를 확보해야만 이중화 데이터의 가용성을 보장할 수 있습니다.
데이터 전체 백업: 백업 데이터의 유실률을 '제로(zero)'로 확보해야 합니다.
COS의 버킷 복사 기능은 증분 데이터의 리전 간 동기화를 지원합니다. 사용자가 업로드한 데이터는 파일 크기와 리전 거리에 따라 다른 리전의 버킷으로 복제되기까지 수 초에서 수 분의 시간이 소요될 수 있습니다. 버킷 복사를 기반으로 할 경우, 데이터를 타 리전에 이중으로 백업함으로써 비즈니스를 위한 재해 복구 조치를 취할 수 있습니다. 버킷 복사에 관한 내용은 버킷 복사 개요를 참조하십시오. 버킷 복사 기능을 활성화하려면 우선 버전 제어 기능을 활성화해야 합니다. 버전 제어와 관련한 내용은 버전 제어 개요를 참조하십시오.
버킷 복사 기반의 재해 복구 백업 아키텍처의 순서도는 다음과 같습니다.



이 아키텍처에서 사용자의 버킷 A와 버킷 B는 액티브/스탠바이 관계입니다. 예를 들어 기업 사용자의 데이터가 버킷 A에 보관되어 있는 경우, 다른 리전의 버킷 B는 스탠바이용 버킷입니다. 이 기업이 비즈니스의 연속성과 안정성을 보장하기 위해 버킷 A와 B에 각각 버킷 복사 규칙을 설정 및 적용했다면, 버킷 A의 증분 데이터는 버킷 B에 자동 복제되며 버킷 B의 증분 데이터도 버킷 A로 자동 복제됩니다.

주의 :

버킷 A에서 버킷 B로 복제된 증분 데이터는 비록 버킷 B의 증분 데이터일지라도 버킷 A로 재차 복제되지 않습니다.

특별한 문제가 없는 경우, 기업의 주요 읽기/쓰기 요청 링크는 버킷 A를 지향하며, 모든 증분 데이터는 백업 데이터로서 버킷 B로 자동 증분 되어 동기화 복제됩니다. 사용자는 프로그램을 업로드하거나 다운로드할 때 네트워크 품질 점검 모듈을 추가할 수 있으며, 액티브 버킷 A가 다운됐음을 감지하면 읽기/쓰기 요청 링크를 신속히 스탠바이 버킷 B로 전환합니다.

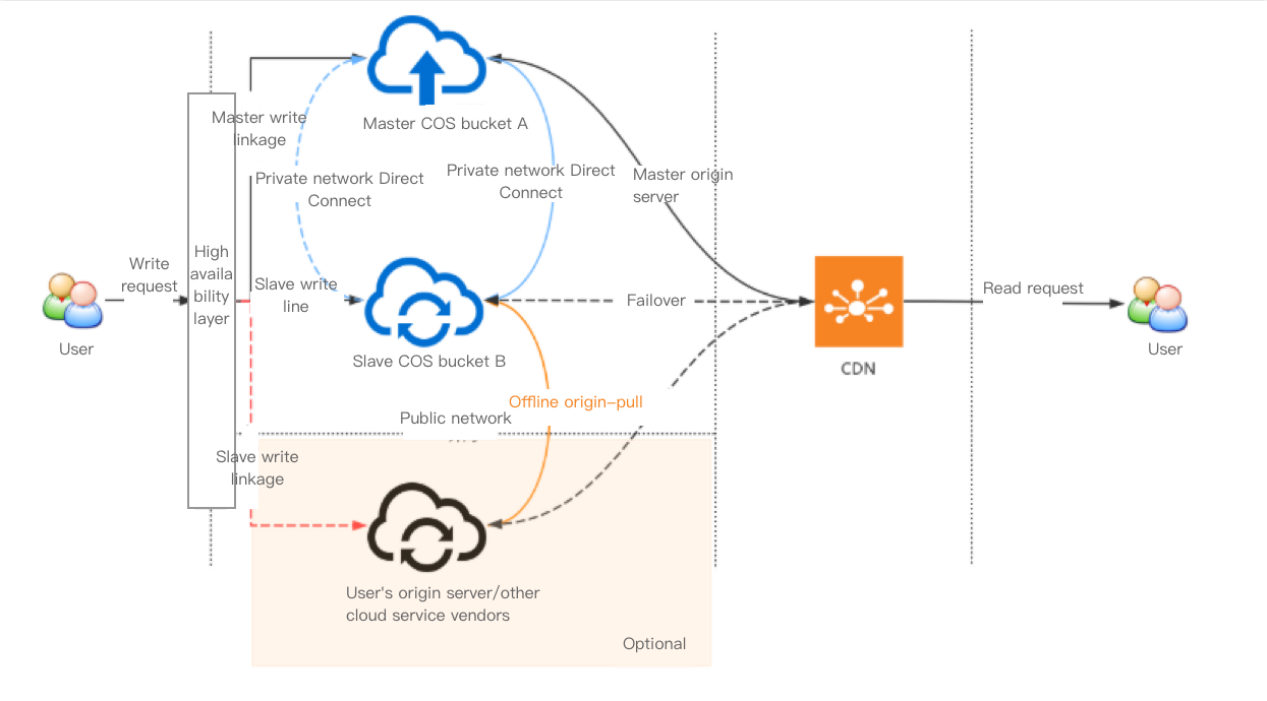
주의 :

Tencent Cloud의 SCF를 기반으로 네트워크 품질을 점검할 수 있습니다. SCF 코드를 수정하여 자동화 테스팅 주소를 액티브/스탠바이 버킷의 도메인으로 변경할 수 있으며, 비즈니스 요구사항에 맞춰 알람 코드 스니펫을 다른 비즈니스에 필요한 상태로 변경할 수 있습니다.

버킷 복사 기반의 고가용성 솔루션

앞서 버킷 복사 기반의 재해 복구 백업 솔루션을 소개했습니다. 이 솔루션은 클라우드의 제품과 기능을 활용한 데이터 백업 및 재해 복구를 위한 전환 작업을 지원합니다. 하지만 실제 비즈니스는 복잡하고 다양하게 운영되기 때문에 상술한 재해 복구 백업 솔루션이 비즈니스적으로 충분한 가용성을 보장하지 못할 수 있습니다. 따라서 다음과 같은

버킷 복사 기반의 고가용성 솔루션을 소개합니다. 버킷 복사, Origin-pull, SCF, CDN 등 다양한 제품과 기능을 활용해 비즈니스 가용성을 높일 수 있습니다.
버킷 복사 기반의 비즈니스 고가용성 아키텍처의 순서도는 다음과 같습니다.



본 아키텍처는 다음과 같은 몇 개의 레이어로 나뉩니다.
고가용성 레이어: 통합 네트워크 점검 및 비즈니스 스케쥴링 시 링크의 상호접속률 등 지표에 따라 링크를 전환하며, 사용자가 SCF를 토대로 실현(설명 참조)할 수 있습니다. 또한 비즈니스 요구사항에 맞춰 클라이언트에서 직접 실현하는 것도 가능합니다.
스토리지 레이어: 보통 COS의 다른 리전의 버킷으로 구성됩니다. 사용자는 Origin-pull 정책 설정을 통해 **외부 원본 서버나 타 클라우드 벤더에 있는** 버킷을 가져와 데이터 일관성을 보다 강화할 수 있습니다.
CDN 레이어: Tencent Cloud CDN의 대용량 엣지 노드는 근거리 액세스 기능을 지원합니다. 사용자가 직접 원본 서버 데이터에 액세스하는 것을 방지함으로써 원본 서버 데이터의 보안성을 보장합니다.
본 아키텍처는 다음과 같은 방법으로 비즈니스의 고가용성을 보장합니다.
1. 특별한 문제가 없는 경우, 기업의 주요 쓰기 요청 링크는 버킷 A를 지향하며, 모든 증분 데이터는 백업 데이터로서 버킷 B로 자동 동기화 복제됩니다.
2. 액티브 버킷 A의 링크에 장애(예: 자동화 테스팅 품질 저하 또는 업로드 실패 감지)가 발생하면 클라이언트에서 쓰기 요청 링크를 액티브 버킷 B로 전환합니다. 이때 모든 증분 데이터는 버킷 A로 자동으로 동기화되어 복제됩니다.
3. 사용자는 외부 원본 서버나 타 클라우드 벤더 측에 이중화 데이터를 백업하는 것과 함께 버킷 B에 Origin-pull 정책을 설정할 수 있습니다. 만약 액티브 버킷 A와 B의 링크가 동시에 연결되지 않는 최악의 상황을 가정할 경우, 버킷 B에 대한 데이터 업로드가 실패하면 버킷 B가 원본 서버로부터 데이터를 가져오게 됩니다.
주의 :

데이터 전체를 이중화하여 백업할 경우 많은 비용이 부과됩니다. 핫 데이터(예: 몇 시간 내 업로드한 파일)에 한해 이중화 백업을 함으로써 데이터 스토리지 비용을 절약할 수 있습니다.
원본 서버를 고가용성 아키텍처의 일부로 선택할 경우, 원본 서버의 대역폭과 대역폭 제한에 따른 결과를 고려하여 아키텍처를 설계하십시오.
4. 버킷에 직접 액세스하여 데이터를 읽어올 수 있으며, 버킷에 CDN 가속 도메인 바인딩을 하여 Tencent Cloud CDN의 엣지 노드에서 근거리 액세스를 할 수 있습니다. 비즈니스 데이터가 콘텐츠 전송 시나리오와 관련이 있거나 사용자가 버킷에 직접 액세스하는 것을 원치 않을 경우, Tencent Cloud CDN을 함께 사용하는 것을 권장합니다.
설명：
버킷에서 직접 데이터를 읽어오려면 클라이언트에서 follow HTTP 302를 지원해야 합니다.
Tencent Cloud CDN은 약 천 개의 엣지 노드를 제공합니다. 사용자와 가까운 거리에 액세스 노드를 제공하여 읽어오는 속도를 높이고 있습니다. CDN에 여러 개의 원본 서버를 액티브/스탠바이 관계로 바인딩함으로써 고가용성을 보장할 수 있습니다. 원본 서버 설정을 참고하여 설정하십시오.
원본 서버의 보안을 강화하려면 원본 서버의 개인 읽기/쓰기 및 CDN Origin-pull 인증을 활성화하십시오. 사용자는 CDN 엣지 노드에 캐싱된 데이터에 익명으로 액세스할 수 있을 뿐만 아니라 원본 서버의 데이터 보안도 확보할 수 있습니다.

참조 문서

재해 복구를 위한 고가용성 아키텍처를 사용할 때 다음 문서를 참조할 수 있습니다.
버전 제어 개요
버킷 복사 개요
Origin-pull 설정
CDN 가속 설정
원본 서버 설정
도메인 액세스

클라우드 데이터 백업

최종 업데이트 날짜: : 2024-06-24 16:44:04

소개

본 문서에서는 클라우드에 보관된 데이터를 백업하는 방법을 개략적으로 소개합니다. Tencent Cloud COS는 사용자에게 COS에 보관된 데이터를 편리하게 백업할 수 있는 다음의 세 가지 방법을 제공합니다.
리전 간 복제 기반의 재해 복구 백업 솔루션
데이터 일괄 복제 솔루션(COS의 일괄 프로세스 기능)
데이터 마이그레이션 툴 MSP를 기반으로 한 마이그레이션 백업 솔루션

리전 간 복제 기반의 재해 복구 백업 솔루션

리전 간 복제 기능은 버킷의 설정 항목 중 하나로, 리전 간 복제 규칙을 설정하여 상이한 스토리지 리전에 있는 버킷에서 **증분 객체**를 비동기화하여 자동으로 복제합니다. 리전 간 복제를 활성화하면 COS에서 원본 버킷의 객체 콘텐츠(예: 객체 메타데이터 및 버전 ID 등)를 타깃 버킷으로 정확하게 복제하며, 복제한 객체의 복제본은 완전히 동일한 속성 정보를 지닙니다.
자세한 내용은 리전 간 복제 기반의 재해 복구 고가용성 아키텍처를 참조하십시오.

적용 시나리오

원격 재해 복구: COS는 객체 데이터에 99.999999999%의 가용성을 제공하지만 전쟁, 자연재해와 같은 불가항력적 요인으로 인해 데이터 유실이 발생할 수 있습니다. 데이터 유실로 인한 손해를 방지하려면 다른 리전에 데이터 복제본을 마련해 두는 것이 좋습니다. 리전 간 복제를 통해 데이터의 원격 재해 복구를 실행할 수 있으며, 특정 데이터센터가 불가항력적 요인으로 파손되어도 다른 리전의 데이터센터를 통해 사본 데이터를 제공받을 수 있습니다.
컴플라이언스 요건: COS는 기본적으로 물리적 디스크 안 데이터의 여러 복제본과 이레이저 코딩 등을 제공(기본값)하는 방법으로 데이터 가용성을 보장합니다. 일부 업종의 경우, 컴플라이언스 요건에 따라 타 버킷 리전에 데이터 복제본을 보관해야 합니다. 이런 경우, 리전 간 복제를 활성화함으로써 리전 간 데이터 복제를 실행하고, 컴플라이언스 요건을 충족할 수 있습니다.
액세스 딜레이 감소: 사용자의 계정이 타 지역에서 객체에 액세스하는 경우, 리전 간 복제를 통해 사용자의 위치와 가장 가깝고 사용 가능한 버킷 리전에 객체의 복제본을 마련할 수 있습니다. 이를 통해 액세스 딜레이를 최소화하고, 더 나은 사용자 환경을 제공할 수 있습니다.
작업적 요인: 두 개의 다른 리전에 모두 컴퓨팅 클러스터를 보유하고 있으며, 이 컴퓨팅 클러스터에서 동일한 데이터를 처리해야 할 경우, 리전 간 복제를 이용해 두 리전의 객체 복제본을 관리할 수 있습니다.
데이터 마이그레이션 및 백업: 비즈니스의 향후 상황에 따라 비즈니스 데이터를 가용 리전 간 복제를 통해 데이터 마이그레이션과 데이터 백업을 구현할 수 있습니다.

사용 방법

리전 간 복제 설정에 관한 콘솔 문서를 참조하십시오.

데이터 일괄 복제 솔루션

COS의 일괄 프로세스 기능으로 객체 레벨에 대한 대량 복제 작업을 일괄적으로 처리할 수 있습니다.

적용 시나리오

일부 비지니스 상의 이유로 데이터의 가용성과 신뢰성 확보를 위해 기존 버킷에 보관된 모든 데이터를 타 버킷에 백업해야 할 수도 있습니다.

사용 방법

일괄 프로세스에 관한 콘솔 문서를 참조하십시오.

사용 설명

객체를 일괄 복제하려면 리스트 기능을 함께 적용해야 합니다. 리스트에 관한 내용은 리스트 기능 개요를 참조하십시오. 객체를 지정하여 원본 버킷에서 타깃 버킷으로 일괄 복제할 수 있으며, 타깃 버킷이 속한 리전은 같거나 다를 수 있습니다. 객체를 일괄 복제할 때 복제본의 메타데이터 정보나 스토리지 유형과 같은 정보에 영향을 미치는 PUT Object - copy 관련 매개변수를 사용자 정의할 수 있습니다. PUT Object-copy에 관한 자세한 내용은 PUT Object - copy를 참조하십시오.

주의：

처리할 모든 객체는 동일한 버킷에 보관되어 있어야 합니다.
일괄 프로세스 1회 작업 시 한 개의 타깃 버킷을 설정할 수 있습니다.
원본 버킷에서 객체를 읽어오는 권한과 타깃 버킷으로 객체를 입력하는 권한이 있어야 합니다.
처리할 객체의 크기는 5GB를 초과할 수 없습니다.
ETags 검사 및 사용자 지정 키에 의한 서버 암호화를 지원하지 않습니다.
타깃 버킷에 버전 제어 기능이 활성화되지 않았으며, 동일한 이름의 객체 파일이 존재하는 경우, COS는 객체 파일 덮어쓰기 작업을 실행합니다.

데이터 마이그레이션 백업 솔루션

Tencent Cloud의 MSP는 보다 편리하고 신속한 리소스 마이그레이션 솔루션을 제공합니다. 사용자는 마이그레이션 작업을 실행한 뒤 마이그레이션 서비스 콘솔에서 마이그레이션 진도, 마이그레이션 보고서와 같은 정보를 확인할 수 있습니다. 자세한 이용 방법은 Tencent Cloud COS 간 마이그레이션을 참조하십시오.

로컬 데이터 백업

최종 업데이트 날짜: : 2024-06-24 16:44:04

소개

본 문서에서는 로컬 데이터를 클라우드에 업로드하여 백업하는 방법을 개략적으로 소개합니다. Tencent Cloud COS는 로컬 데이터를 COS 버킷으로 편리하게 백업할 수 있는 다음 세 가지 백업 방법을 지원합니다.

COSBrowser 파일 동기화 백업

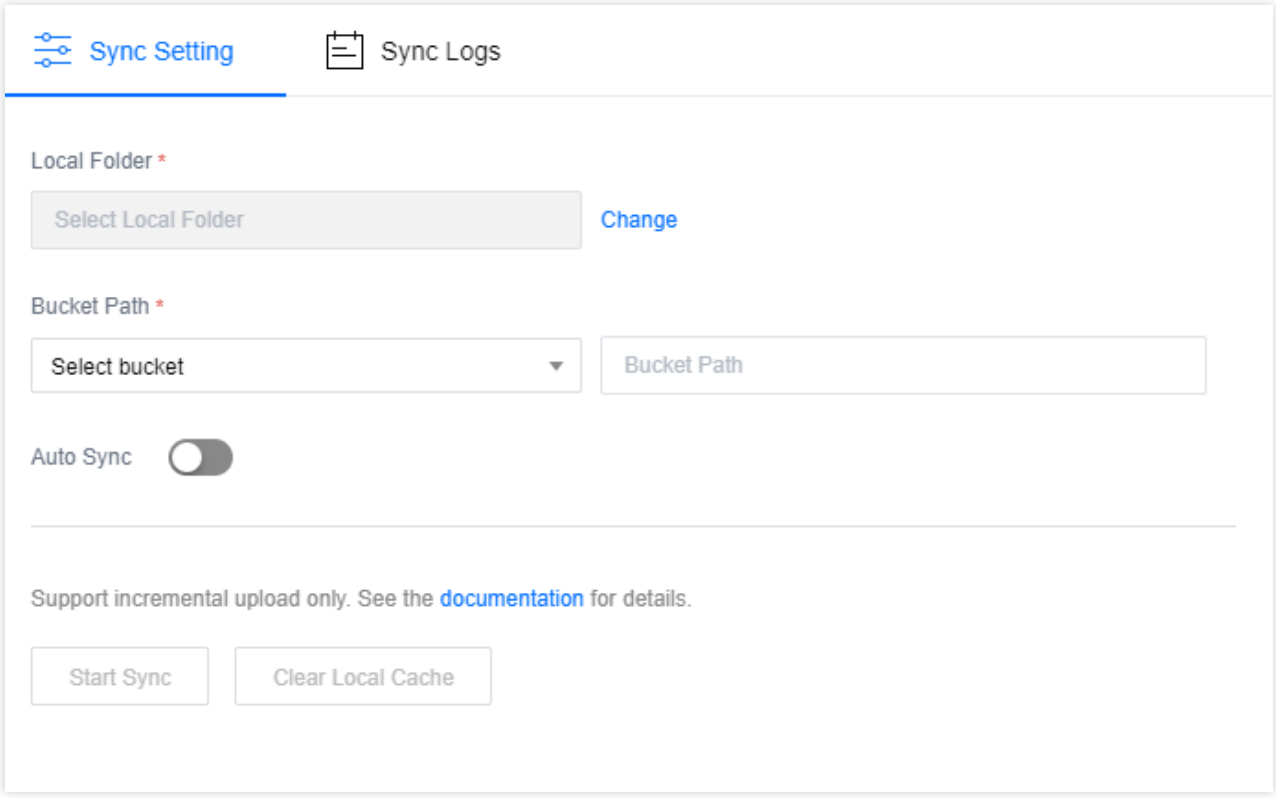
COS Migration 온라인 마이그레이션 백업

CDM 오프라인 마이그레이션 백업

파일 동기화 백업

COSBrowser는 Tencent Cloud COS가 출시한 클라우드 파일 관리용 시각화 인터페이스 툴로, COS 리소스 조회, 전송, 관리를 간편하고 인터랙티브하게 실현할 수 있습니다. 현재 COSBrowser는 데스크톱 버전 및 모바일 버전의 두 버전을 제공하며, 자세한 내용은 COSBrowser 소개를 참조하십시오.

COSBrowser의 데스크톱 버전은 파일 동기화 기능을 통합하여 로컬 폴더와 버킷을 연결함으로써 로컬 파일을 클라우드로 동기화하여 실시간 업로드할 수 있습니다.



사용 방법

파일 동기화 사용 설명을 참조하십시오.

온라인 마이그레이션 방안

COS Migration은 COS의 데이터 마이그레이션 기능을 결합한 통합형 툴입니다. 간단한 설정만으로도 데이터를 COS로 빠르게 마이그레이션할 수 있습니다.

적용 시나리오

로컬 IDC를 보유한 사용자는 COS Migration을 통해 로컬 IDC의 대용량 데이터를 빠르게 COS로 마이그레이션할 수 있습니다.

사용 방법

COS 마이그레이션을 참조하십시오.

오프라인 마이그레이션 방안

CDM은 Tencent Cloud가 제공하는 오프라인 마이그레이션 전용 디바이스로 사용자가 로컬 데이터를 클라우드로 마이그레이션하도록 돕습니다. 네트워크 전송을 통해 로컬 데이터센터에서 클라우드로 마이그레이션할 때보다 시간, 비용, 보안성 면에서 우수한 성능을 구현합니다.
데이터 마이그레이션 규모, IDC 출력 대역폭, IDC 유휴 서버 리소스, 마이그레이션 실행 시 제한 시간 등 요소를 감안하여 마이그레이션 방법을 선택할 수 있습니다. 아래 이미지는 온라인 마이그레이션 실행 시 예상 소요 시간입니다.
마이그레이션 주기가 10일이 넘거나 마이그레이션 데이터양이 50TB가 넘으면 CDM을 선택하여 오프라인으로 마이그레이션을 실행할 것을 권장합니다.

Bandwidth / Data Volume	10Mbps	100Mbps	1Gbps	10Gbps
10GB	3 hours	17 minutes	2 minutes	10 seconds
100GB	30 hous	3 hours	17 minutes	2 minutes
1TB	12.5 days	30 hous	3 hours	17 minutes
10TB	125 days	12.5 days	30 hous	3 hours
100TB	3.5 years	125 days	12.5 days	30 hous
1PB	35 years	3.5 years	125 days	12.5 days
10PB	350 years	35 years	3.5 years	125 days

사용 방법

CDM을 참조하십시오.

도메인 관리 사례
버킷 사용자 정의 도메인 이름 전환

최종 업데이트 날짜: : 2024-07-05 18:49:22

배경

전체 서비스의 보안과 안정성을 위해 2024년 1월 1일 이후에 생성된 버킷은 COS 기본 도메인 이름을 사용하여 객체에 액세스할 경우 모든 유형 파일의 미리보기를 지원하지 않고 apk/ipa 유형 파일의 다운로드를 지원하지 않습니다. 자세한 내용은 COS 버킷 도메인 이름 사용 안전 관리 통지를 참고하시기 바랍니다.

2024년 1월 1일 이후에 생성된 버킷의 경우, 브라우저를 통한 파일 미리보기 또는 버킷 내의 apk/ipa 유형 객체 다운로르를 원하시면 사용자 정의 도메인 이름으로 객체에 액세스하시기 바랍니다. 2024년 1월 1일 이전에 생성된 버킷의 경우, 버킷 기본 도메인 이름의 미리보기, 다운로드 동작은 영향을 받지 않지만 더 나은 서비스 안정성을 위해 사용자 정의 도메인 이름을 우선적으로 사용하시기 바랍니다.

이 문서에서는 버킷에 대해 사용자 정의 도메인 이름을 구성하고, 버킷 기본 도메인 이름 액세스에서 사용자 정의 도메인 이름 액세스로 전환하는 방법에 대해 소개합니다.

첫 번째 단계: 도메인 이름 등록과 파일링

먼저, 사용자는 하나의 파일링된 사용자 정의 도메인 이름을 준비해야 합니다.
도메인 이름 등록: 사용자 정의 도메인 이름이 없으면 도메인 이름 등록으로 이동하여 도메인 이름을 구매할 수 있습니다.
도메인 이름 파일링: 당신의 사용자 정의 도메인 이름이 중국 본토 리전의 버킷에 구성하기 위한 것이면 반드시 파일링해야 합니다.

두 번째 단계: 버킷에 대해 사용자 정의 도메인 이름 구성

1. 사용자 정의 도메인 이름을 준비한 다음 COS 콘솔로그인하고, 버킷 리스트에서 구성할 버킷을 선택합니다.
2. 버킷 상세정보 페이지로 이동하여 **도메인 이름과 전송 관리 > 사용자 정의 원본 서버 도메인 이름**을 선택합니다.
3. **도메인 이름 추가**를 클릭하여 도메인 이름 정보를 구성합니다.
도메인 이름: 준비된 사용자 정의 도메인 이름을 입력합니다.
원본 서버 유형: 다음과 같은 몇 가지로 구분됩니다.

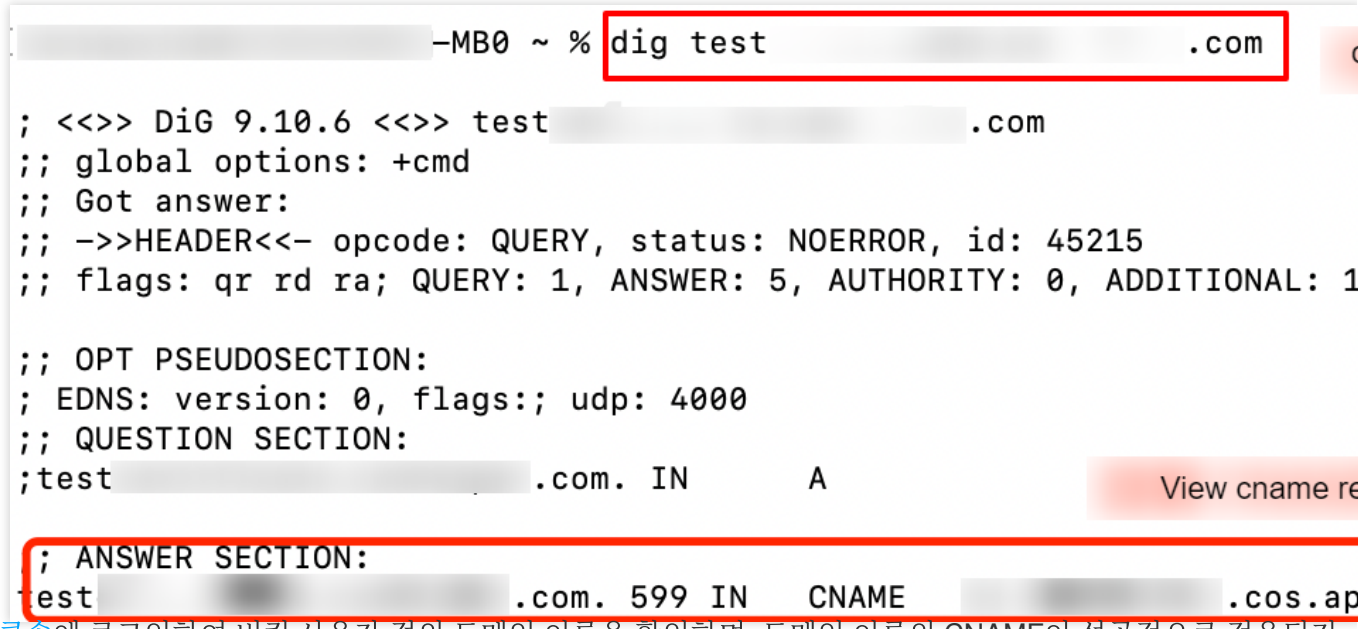
기본 원본 서버: 사용자 정의 도메인 이름을 기본 원본 서버로 사용하려면 기본 원본 서버를 선택하십시오.
정적 웹사이트 원본 서버: 사용자 정의 도메인 이름을 정적 웹사이트로 사용하려면 먼저 버킷에 대해 정적 웹사이트 기능을 활성화한 다음 정적 웹사이트 원본 서버를 선택하십시오.
글로벌 가속 원본 서버: 사용자 정의 도메인 이름을 글로벌 가속으로 사용하려면 먼저 버킷에 대해 글로벌 가속 기능을 활성화한 다음 글로벌 가속 원본 서버를 선택하십시오.
4. HTTPS 인증서를 구성합니다. HTTPS 프로토콜을 사용하여 액세스해야 하는 경우 사용자 정의 도메인 이름에 대해 인증서를 구성해야 합니다.
자체 인증서를 사용해야 하는 경우 인증서 내용 및 비밀키 내용을 지정 입력 상자에 붙혀넣어야 합니다.
Tencent Cloud에서 신청한 인증서를 사용하는 경우 직접 팝업창에서 현재 계정에 있는 Tencent Cloud 인증서를 선택할 수 있습니다.
5. 사용자 정의 도메인 이름 구성 완료 후, 후속 도메인 이름 해석을 구성하기 위해 CNAME난의 정보(예: bucket-1250000000.cos.ap-beijing.myqcloud.com)를 기록합니다.

세 번째 단계: 도메인 이름 해석 구성

Tencent Cloud 도메인 이름

도메인 이름 DNS 제공업체가 Tencent Cloud인 경우 DNS 콘솔로 이동하여 CNAME 해석 기록을 구성할 수 있습니다.

1. DNS 콘솔로 이동하여 대응하는 도메인 이름을 찾고 **해석** 버튼을 클릭합니다.
2. **빠른 해석 추가**를 클릭하여 해당 도메인 이름에 하나의 해석 기록을 추가합니다.
3. 팝업창에서 **웹사이트 해석**을 선택하고, 웹사이트 주소는 **도메인 이름 매핑(CNAME)**을 선택합니다. 두 번째 단계에서 기록독 CNAME 정보, 예를 들어 bucket-1250000000.cos.ap-beijing.myqcloud.com을 입력합니다.
4. 해석 기록은 일정한 시간이 지나야 적용됩니다. dig 명령을 사용하거나 COS 콘솔을 통해 해석이 성공적으로 적용되었는지 여부를 확인할 수 있습니다. 검증 방법은 다음과 같습니다.
명령행 창에 명령: `dig mydomain.com` 을 입력하여 CNAME 기록이 정확하게 적용되었는지 여부를 확인합니다. (사용 시 `mydomain.com` 을 사용자 정의 도메인 이름으로 교체하세요)



COS 콘솔에 로그인하여 버킷 사용자 정의 도메인 이름을 확인하며, 도메인 이름의 CNAME이 성공적으로 적용되지 않았으면 해당 알림이 뜨게 됩니다.

타업체 도메인 이름

도메인 이름 DNS 제공업체가 Tencent Cloud 아닌 경우 해당 DNS 서비스로 이동하여 CNAME 해석 기록을 구성해야 합니다.

네 번째 단계: 사용자 정의 도메인 이름 액세스

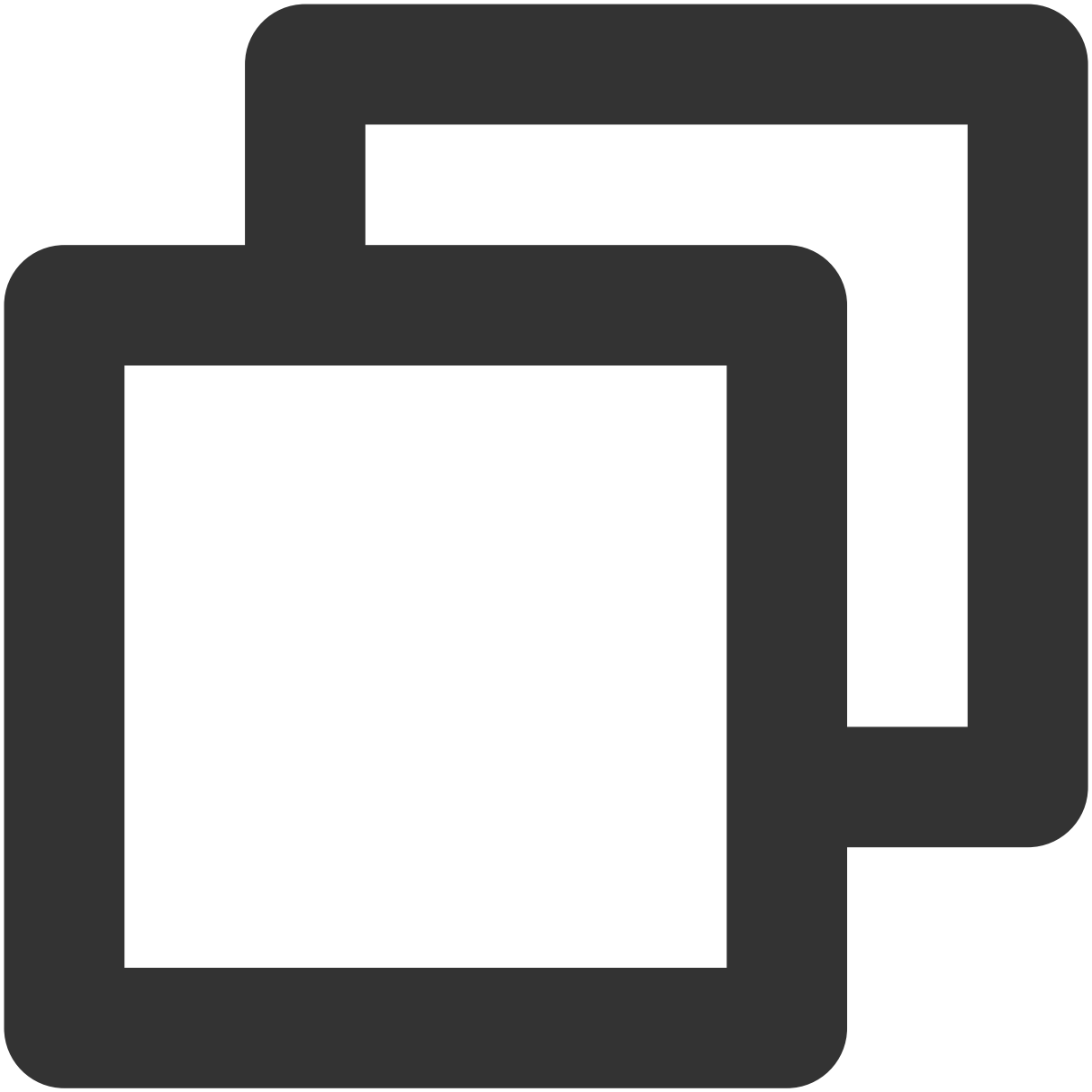
상기 단계를 거쳐 사용자 정의 도메인 이름의 구성을 완료하였습니다. 아래에서는 사용자 정의 도메인 이름을 사용하여 COS에 액세스하는 방법에 대해 설명합니다.

객체 액세스 링크 확인

1. COS 콘솔에 로그인하여 사용자 정의 도메인 이름이 구성된 버킷을 찾고 클릭하여 **파일 목록**으로 들어갑니다. 하나의 객체를 선택하고 객체 상세정보로 들어갑니다. 조작 안내는 객체 정보 조회를 참고하시기 바랍니다.
2. 지정된 도메인 이름을 **사용자 정의 원본 서버 도메인 이름**으로 전환합니다. 아래의 객체 주소, 임시 링크는 사용자 정의 도메인 이름의 링크로 전환됩니다. 공개 읽기 객체에 액세스할 때 객체 주소(서명 없음)를 사용하고 비공개 읽기 객체에 액세스할 때 임시 링크(서명 있음)를 사용할 수 있습니다.

API 액세스를 위한 사용자 정의 도메인 이름 전환

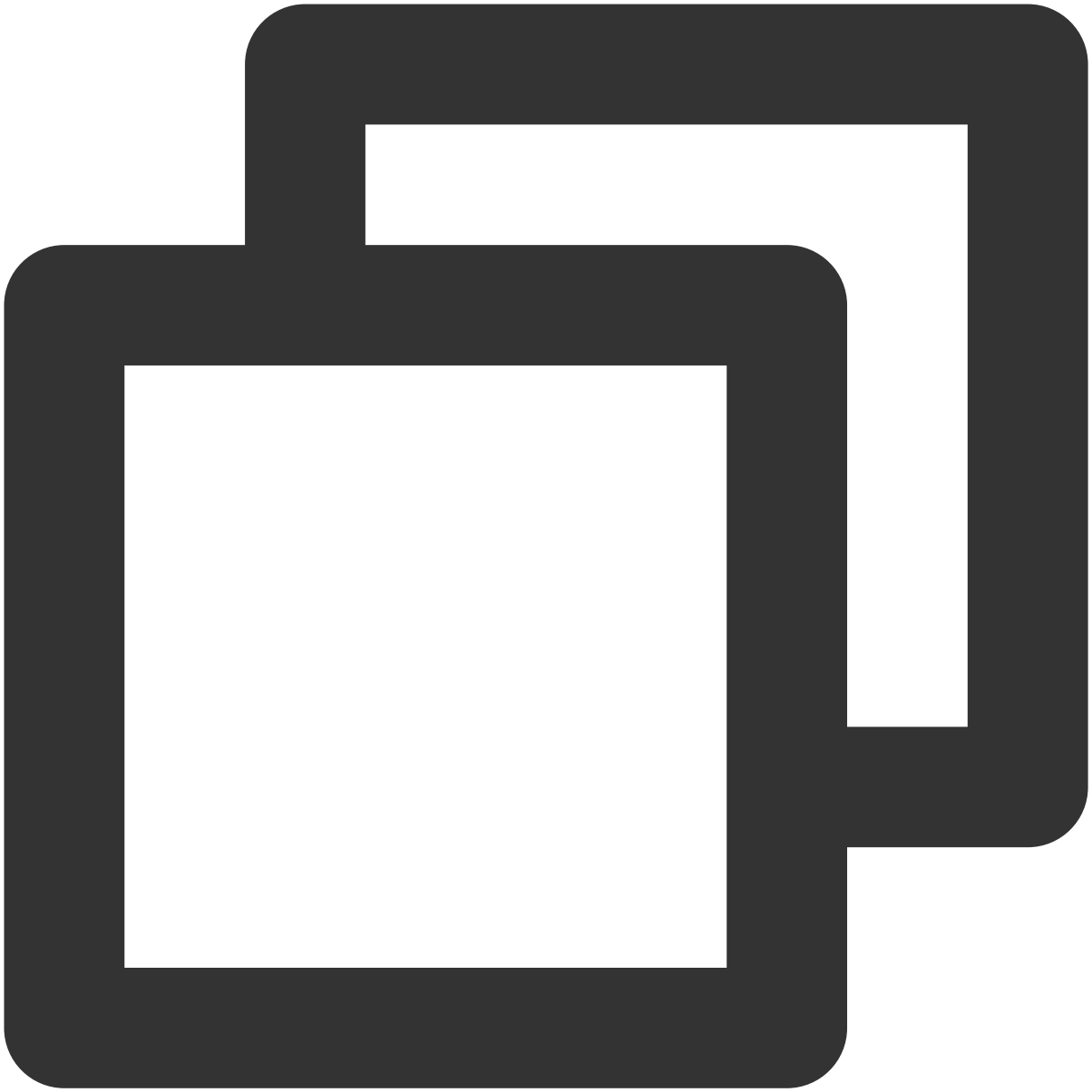
직접 API를 사용하여 COS에 액세스하는 경우 액세스할 때 요청 Host를 사용자 정의 도메인 이름으로 전환하면 됩니다.



```
GET /\\<0bjectKey> HTTP/1.1
Host: <BucketName-APPID>.cos.<Region>.myqcloud.com          #  사용자 정의 도메인 이름.
Date: GMT Date
Authorization: Auth String
```

SDK 액세스를 위한 사용자 정의 도메인 이름 전환

SDK를 사용하는 경우 Client를 초기화할 때 domain 매개변수를 사용자 정의 도메인 이름으로 설정하면 됩니다. Python SDK로 예를 들면, 코드 예는 다음과 같습니다.



```
domain = 'user-define.example.com' # 사용자 정의 도메인 이름
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=t
client = CosS3Client(config)
```

각 언어의 COS SDK에서 사용자 정의 도메인 이름으로 전환하는 코드 예는 다음 문서를 참고하시기 바랍니다.

Go SDK

JAVA SDK

Python SDK

PHP SDK

Android SDK
iOS SDK
JS SDK
Node.js SDK
.NET SDK
C++ SDK
C 언어 SDK
미니 프로그램 SDK

사용자 지정 엔드포인트에 대한 HTTPS 지원

최종 업데이트 날짜: : 2024-06-24 16:56:13

소개

자체 도메인(사용자 정의 도메인, 예: `test.cos.com`)을 통해 버킷(Bucket)의 객체(Object)에 액세스할 수 있습니다. 구체적인 작업 가이드는 다음과 같습니다.
CDN 가속 활성화 시 사용자 정의 도메인에 HTTPS 액세스 지원 설정
CDN 가속 비활성화 시 사용자 정의 도메인에 HTTPS 액세스 지원 설정

작업 단계

CDN 가속 활성화

1단계: 사용자 정의 도메인 바인딩

버킷을 자체 도메인에 바인딩하고 CDN 가속을 활성화합니다. 자세한 작업 가이드는 사용자 정의 CDN 가속 도메인 활성화 문서를 참고하십시오.

2단계: HTTPS 액세스 설정

CDN 콘솔에서 HTTPS를 설정합니다. 자세한 작업 가이드는 HTTPS 가속 설정 가이드를 참고하십시오.

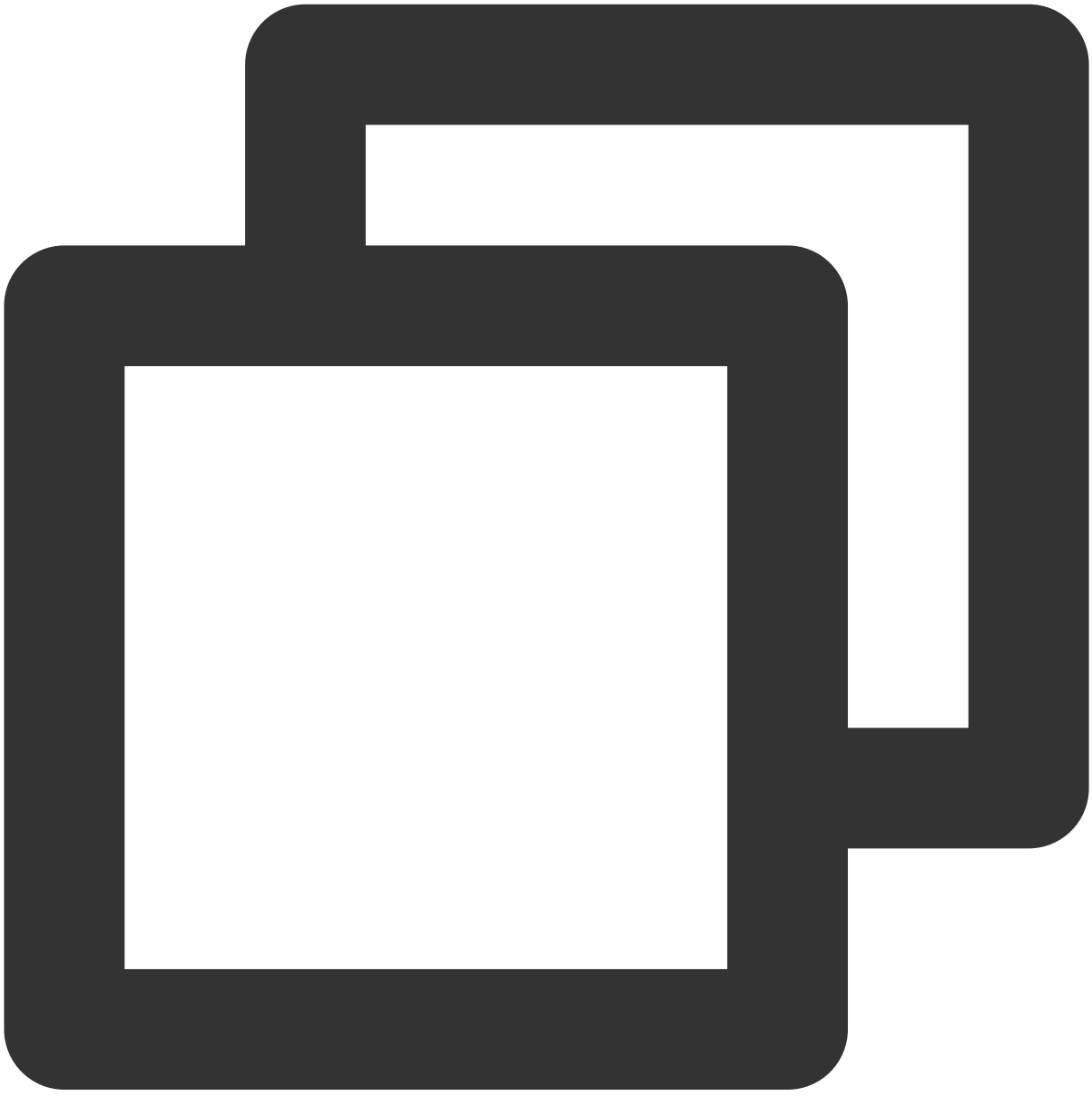
CDN 가속 비활성화

본 챕터에서는 Cloud Object Storage(COS)에서 리버스 프록시를 통해 사용자 정의 도메인(CDN 가속 비활성화)을 설정하고 HTTPS 액세스를 지원하는 작업을 예시를 통해 소개합니다. 본 예시에서는 CDN 가속이 비활성화된 상태에서 직접 사용자 정의 도메인 `https://test.cos.com` 을 통해 소속 리전이 광저우이고 이름이 testhttps-1250000000인 버킷에 액세스하게 됩니다. 구체적인 작업 순서는 다음과 같습니다.

1단계: 사용자 정의 도메인 바인딩

COS 중국 내 공유 클라우드 리전과 싱가포르 리전은 이미 사용자 정의 원본 서버 도메인 호스팅을 위한 HTTPS 인증서를 지원하고 있으며, 추가된 사용자 정의 원본 서버 도메인은 콘솔을 통해 인증서를 바인딩 할 수 있습니다. 자세한 내용은 방법1을 참고하십시오. 도메인에 HTTPS 인증서가 없는 경우 Tencent Cloud 인증서 신청을 클릭할 수 있습니다.
기타 해외 리전의 경우 현재 HTTPS 인증서 호스팅이 지원되지 않으며, HTTPS 인증서를 사용해야 하는 경우 방법2를 참고하십시오.
방법1: COS 콘솔을 통해 사용자 정의 원본 서버 도메인 바인딩
버킷 testhttps-1250000000를 `https://test.cos.com` 도메인에 바인딩하고, CDN 가속을 비활성화합니다. 자

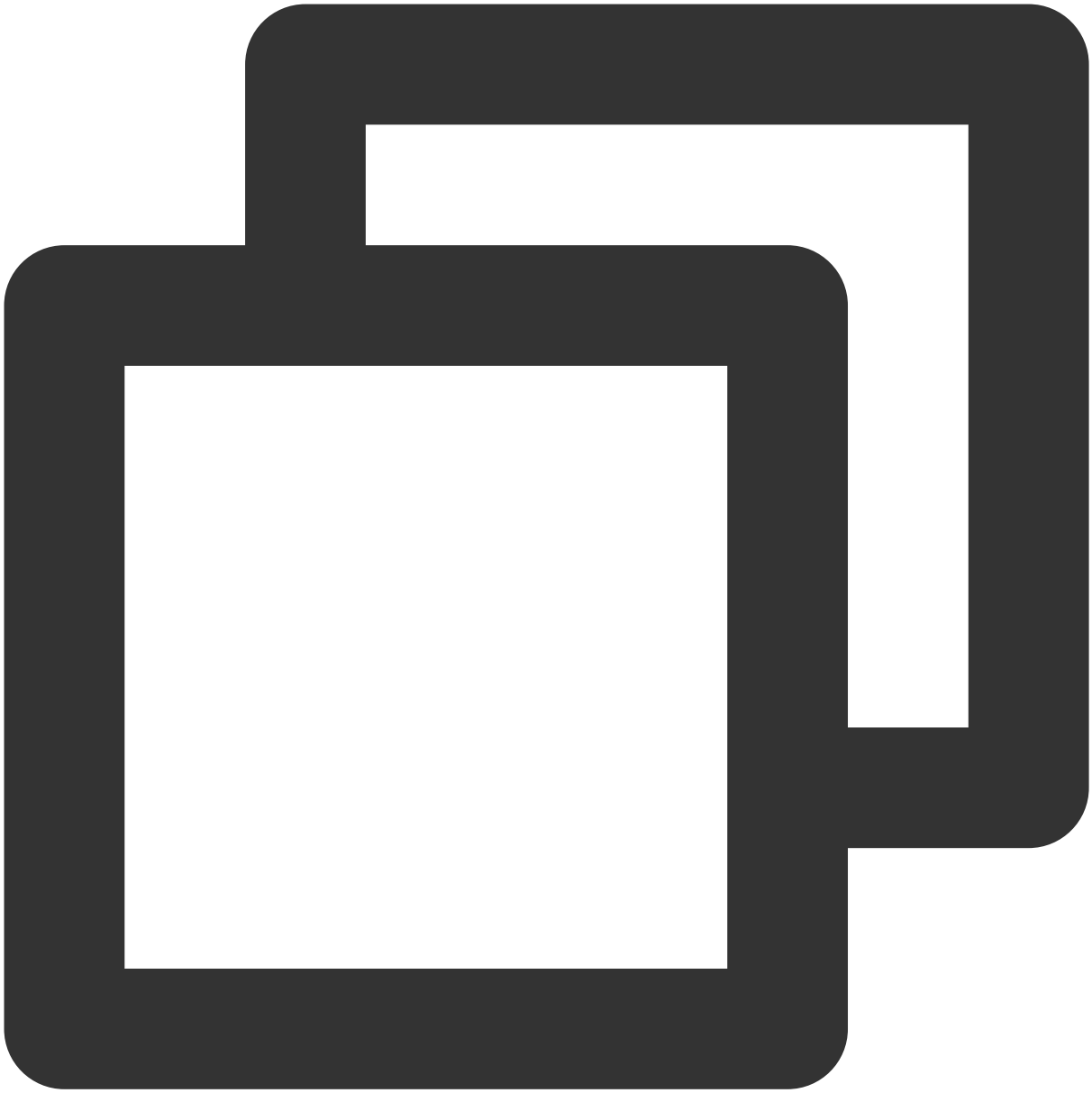
세한 작업 가이드는 사용자 정의 원본 서버 도메인 활성화 문서를 참고하십시오.
방법2: 도메인에 리버스 프록시 설정
서버 상의 `https://test.cos.com` 도메인에 리버스 프록시를 설정합니다. 구체적인 설정 방법은 다음을 참고하십시오(Nginx 설정은 참고용으로만 제공).



```
server {
  listen        443;
  server_name  test.cos.com ;

  ssl on;
  ssl_certificate /usr/local/nginx/conf/server.crt;
```

```
    ssl_certificate_key /usr/local/nginx/conf/server.key;

    error_log logs/test.cos.com.error_log;
    access_log logs/test.cos.com.access_log;
    location / {
        root /data/www/;
        proxy_pass  http://testhttps-1250000000.cos.ap-guangzhou.myqcloud.com; //버킷(
    }
}
```

`server.crt;` , `server.key` 는 귀하의 자체(사용자 정의) 도메인 HTTPS 인증서입니다. 귀하의 도메인에 HTTPS 인증서가 없는 경우 Tencent Cloud SSL 인증서 페이지에서 신청하십시오.
현재 인증서가 없는 경우 다음과 같은 설정 정보를 삭제할 수 있습니다. 액세스 시 알람이 뜰 수 있으나 계속을 클릭하면 액세스할 수 있습니다.



```
ssl on;
ssl_certificate /usr/local/nginx/conf/server.crt;
ssl_certificate_key /usr/local/nginx/conf/server.key;
```

2단계: 서버로 도메인 리졸브

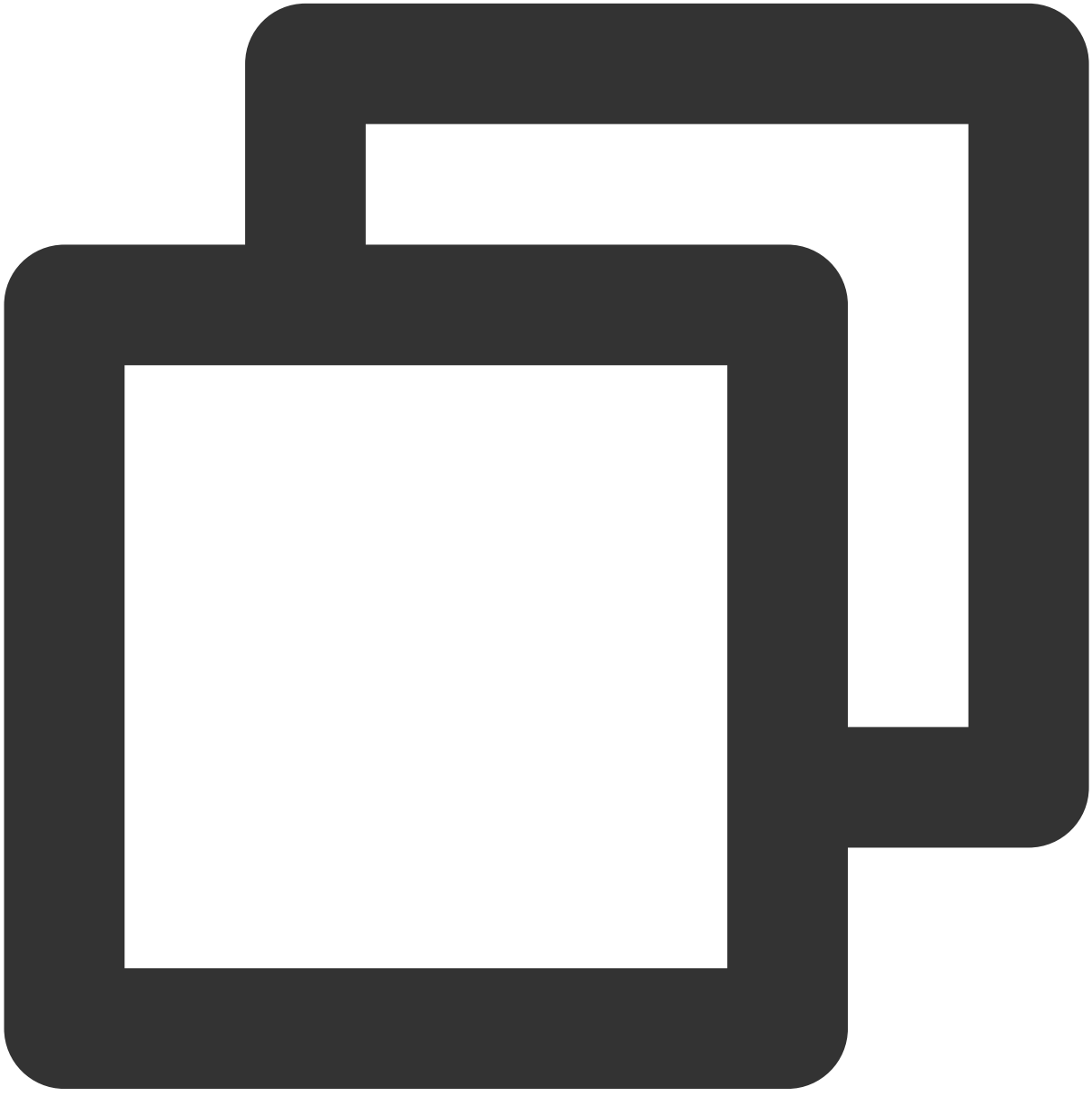
도메인 DNS 리졸브 서비스 제공 업체에서 사용자의 도메인을 리졸브합니다.

3단계: 고급 설정

브라우저로 직접 웹 페이지 열기

사용자 정의 도메인의 HTTPS 액세스 지원 설정을 완료하면 사용자의 도메인을 통해 버킷(Bucket)에 있는 객체(Object)를 다운로드할 수 있습니다. 비즈니스 수요에 따라 브라우저에서 직접 웹 페이지 및 이미지 등을 액세스해야 하는 경우 정적 웹 사이트 기능을 통해 실행할 수 있습니다. 자세한 작업 가이드는 정적 웹 사이트 설정을 참고하십시오.

설정 완료 후, Nginx 설정에 정보를 한 행 추가한 다음 Nginx를 재시작해 브라우저 캐시를 새로 고칩니다.



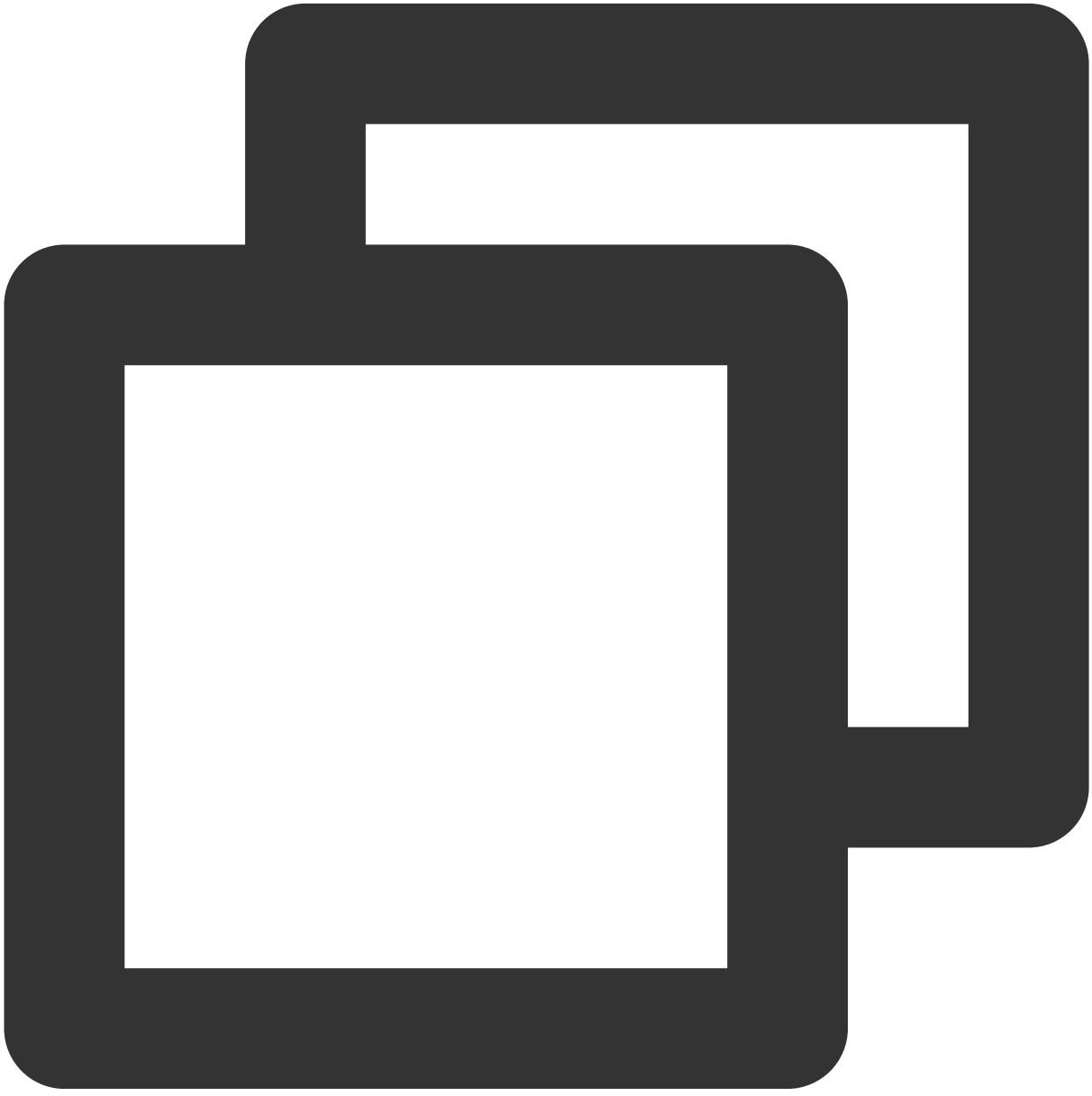
```
proxy_set_header Host $http_host;
```

refer 링크 도용 방지 설정

버킷(Bucket)이 공용인 경우 링크 도용 리스크가 있습니다. 링크 도용 방지 설정을 통해 Referer 얼로우리스트를 활성화하여 악성 링크 도용을 방지할 수 있습니다. 구체적인 작업 순서는 다음과 같습니다.

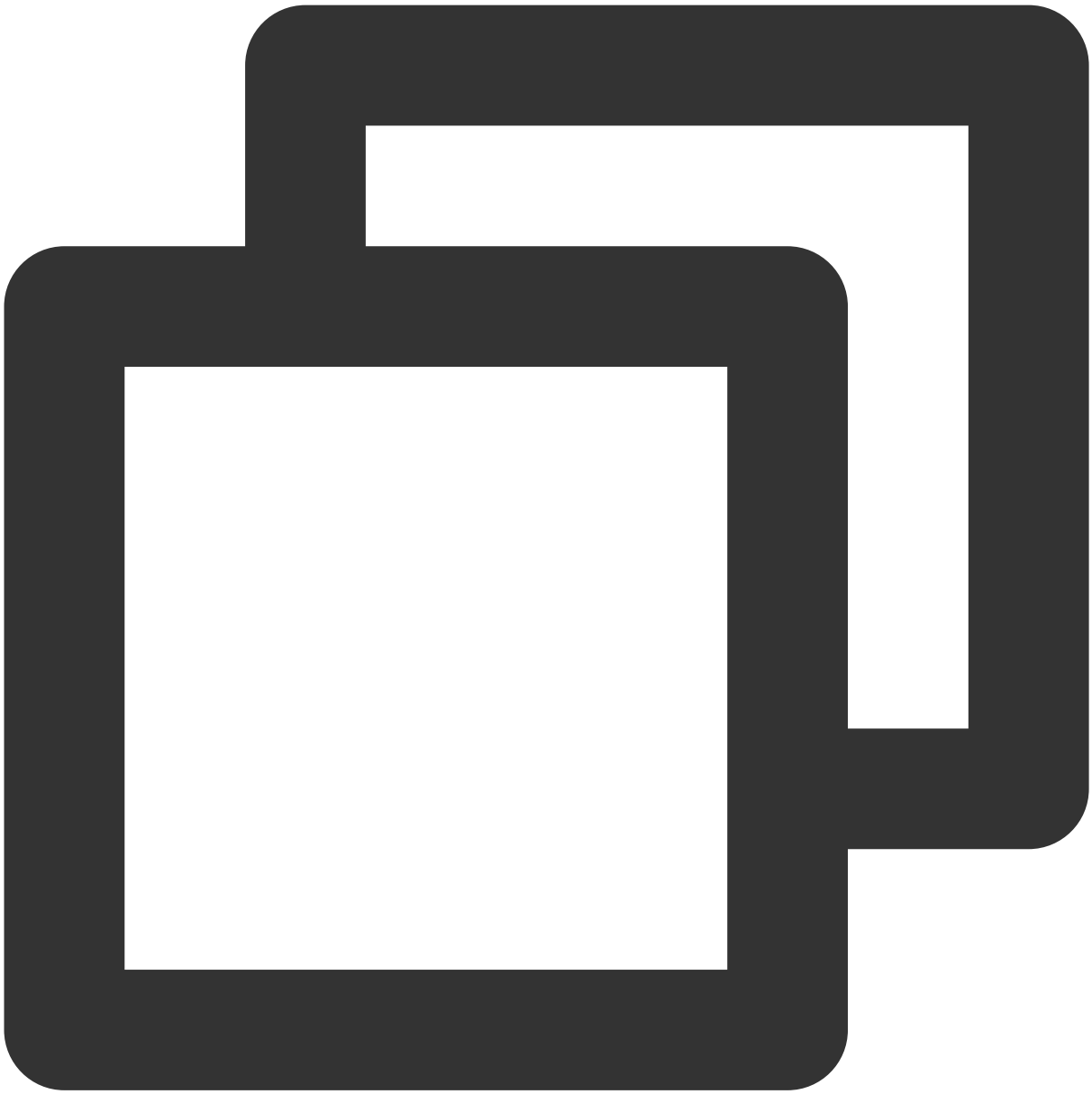
1.1 COS 콘솔에 로그인한 후, 링크 도용 방지 설정 기능을 활성화하여 얼로우리스트를 선택합니다. 자세한 작업 가이드는 링크 도용 방지 설정을 참고하십시오.

1.2 Nginx 구성 파일에 정보를 한 행 추가한 다음 Nginx를 재시작해 브라우저 캐시를 새로 고칩니다.



```
proxy_set_header   Referer www.test.com;
```

1.3 설정 완료 후, 바로 파일을 열면 `errorcode: -46616` 오류 알림이 뜹니다. 오류 알림: refer 얼로우리스트가 미스되었으나 프록시를 통해 사용자 정의 도메인에 액세스하면 정상적으로 웹 페이지를 열 수 있습니다.



```
{
errorcode: -46616,
errormsg: "not hit white refer, retcode:-46616"
}
```

CORS 설정

최종 업데이트 날짜: : 2024-06-24 16:44:04

동일 출처 정책

동일 출처 정책은 어떤 출처에서 불러온 파일이나 스크립트가 다른 출처에서 가져온 리소스와 상호작용하는 것을 제한하며, 악성 위험이 있는 파일을 격리하는 핵심 보안 메커니즘에 사용됩니다. 동일한 프로토콜, 동일한 도메인(또는 IP), 동일한 포트를 동일한 출처로 보며, 한 출처 내의 스크립트는 해당 출처 내의 권한만 가집니다. 즉, 해당 출처의 스크립트는 해당 출처 내의 리소스만 읽고 쓸 수 있고 다른 출처의 리소스는 액세스할 수 없습니다. 이러한 보안 제한을 동일 출처 정책이라고 합니다.

동일 출처의 정의

두 페이지의 프로토콜, 도메인, 포트(포트를 지정한 경우)가 동일한 경우 동일 출처로 봅니다. 다음 표는 `http://www.example.com/dir/page.html` 과의 동일 출처를 점검하는 예시입니다.

URL	결과	이유
`http://www.example.com/dir2/other.html`	성공	프로토콜, 도메인, 포트 동일
`http://www.example.com/dir/inner/another.html`	성공	프로토콜, 도메인, 포트 동일
`https://www.example.com/secure.html`	실패	프로토콜 불일치(HTTPS)
`http://www.example.com:81/dir/etc.html`	실패	포트 불일치(81)
`http://news.example.com/dir/other.html`	실패	도메인 불일치

크로스 도메인 액세스

Cross-Origin Resource Sharing(CORS) 메커니즘은 크로스 도메인 액세스라고도 하며, Web 애플리케이션 서버에서의 크로스 도메인 액세스 제어를 허용해 크로스 도메인 데이터 전송 시 보안을 보장합니다. CORS는 브라우저 및 서버에서 모두 지원해야 하며, 현재 모든 브라우저에서 해당 기능을 지원하고, IE 브라우저는 IE10 이상의 버전만 지원합니다.
모든 CORS 통신 프로세스는 브라우저에서 자동으로 완료하며 사용자가 개입할 필요가 없습니다. 개발자 입장에서 CORS 통신과 동일 출처의 AJAX 통신은 차이가 없으며 코드도 완전히 동일합니다. 브라우저에서 AJAX의 크로스 도메인 요청을 발견하면 자동으로 일부 헤더 정보가 추가되며, 때때로 다시 한번 부가 요청이 있게 되지만 사용자는 이를 감지할 수 없습니다.
따라서 CORS 통신 구현의 핵심은 서버입니다. 서버가 CORS 인터페이스를 구현하면 바로 크로스 도메인으로 통신됩니다.

CORS의 주요 사용 시나리오

사용자는 브라우저를 사용하는 상황에서 CORS를 사용할 수 있으며, 서버가 아닌 브라우저가 액세스 권한을 제어하기 때문에 다른 클라이언트를 사용해도 크로스 도메인 문제를 걱정하지 않아도 됩니다.
CORS의 주요 애플리케이션은 사용자의 애플리케이션 서버를 통한 중개가 필요 없이 브라우저에서 AJAX를 사용해 직접 COS 데이터에 액세스하거나 데이터의 업로드 및 다운로드를 실현합니다. 동시에 COS와 AJAX 기술을 사용한 웹 사이트는 CORS를 사용해 COS와 직접 통신하는 것을 권장합니다.

COS의 CORS에 대한 지원

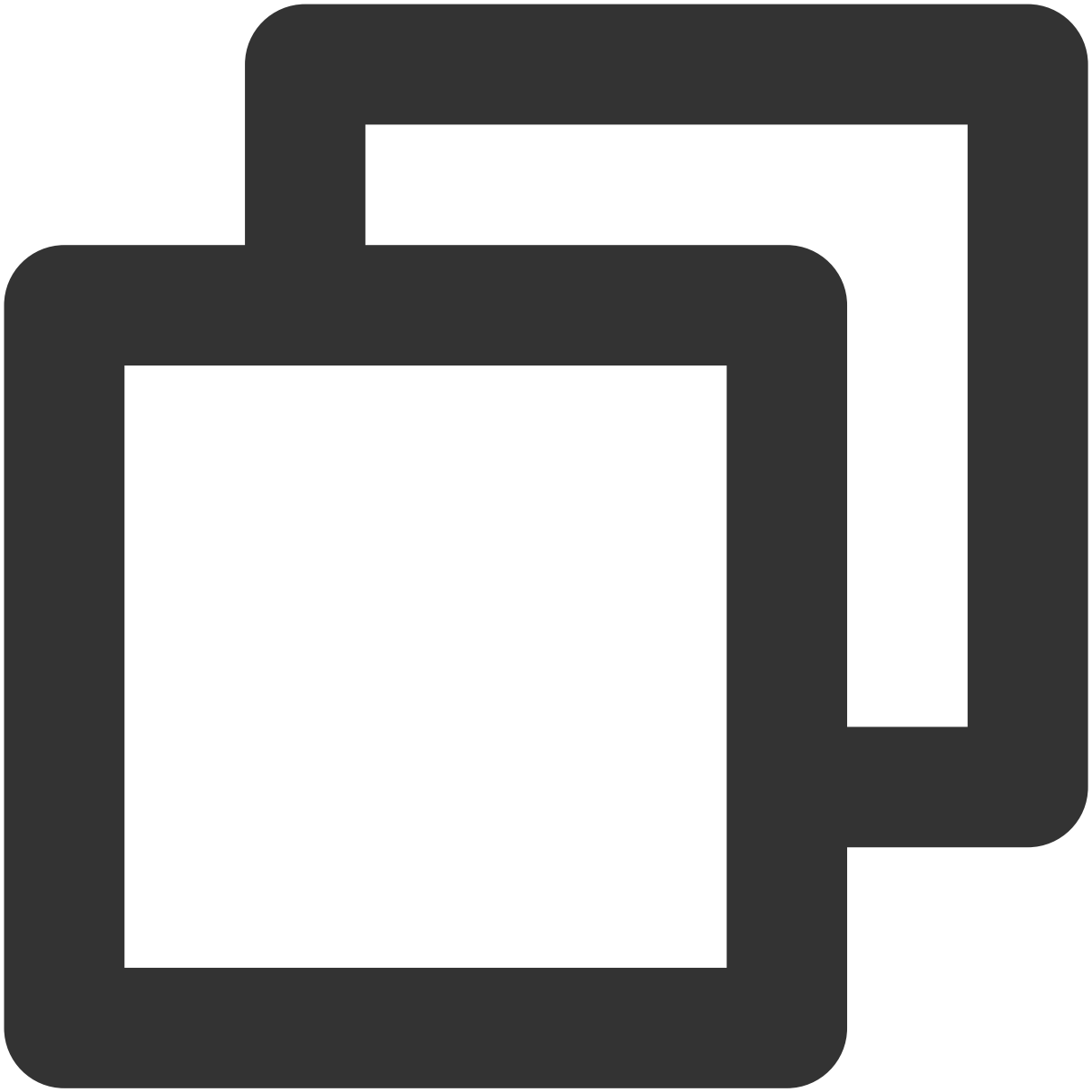
COS에서 CORS 규칙 설정을 설정하여 필요에 따라 해당 크로스 도메인 요청을 허용하거나 거부할 수 있습니다.
CORS 규칙 설정은 버킷 등급에 귀속됩니다.
CORS 요청의 통과 여부는 COS의 인증 등과 완전히 독립적입니다. 즉 COS의 CORS 규칙은 CORS 관련 Header 추가 여부를 결정하는 하나의 규칙일 뿐입니다. 해당 요청을 차단할지 여부는 완전히 브라우저에 의해 결정됩니다.
현재 COS의 모든 Object 관련 인터페이스에서 CORS 관련 인증을 제공하며, 이외에도 현재 Multipart와 관련한 인터페이스에서도 CORS 인증을 완벽하게 지원합니다.
설명：
동일한 브라우저 상의 각 `www.a.com` 와 `www.b.com` 두 페이지에서 동시에 동일한 크로스 도메인 리소스를 요청했을 때, `www.a.com` 의 요청이 먼저 서버에 도달한 경우 서버는 리소스에 Access-Control-Allow-Origin 헤더를 추가하여 `www.a.com` 사용자에게 반환합니다. 이때 `www.b.com` 에서 다시 요청하는 경우 서버는 Cache의 이전 요청에 대한 응답을 사용자에게 반환하여 헤더 내용과 CORS 요청이 매칭되지 않아 `www.b.com` 의 요청이 실패하게 됩니다.

CORS 설정 예시

다음은 AJAX를 사용해 COS에서 데이터를 획득할 수 있도록 설정하는 간단한 예시입니다. 예시에서 사용한 버킷(Bucket)의 권한은 공유(Public)로 설정되어 있으며, 액세스 권한이 개인 버킷(Bucket)인 경우 요청에 서명만 추가하면 되고 다른 설정은 모두 동일합니다.
다음 예시에서 사용한 버킷 이름은 corstest이며, 버킷 액세스 권한은 공개 읽기, 개인 쓰기입니다.

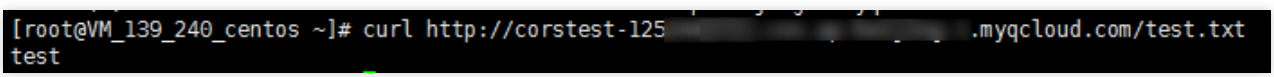
준비 과정

1. **파일에 정상적으로 액세스할 수 있는지 확인**
test.txt의 텍스트 파일을 corstest에 업로드합니다. test.txt의 액세스 주소는 `http://corstest-125xxxxxxx.cos.ap-beijing.myqcloud.com/test.txt` 가 됩니다.
curl을 사용해 직접 해당 텍스트 파일에 액세스합니다. 다음 주소를 사용자의 파일 주소로 대체합니다.



```
curl http://corstest-125xxxxxxx.cos.ap-beijing.myqcloud.com/test.txt
```

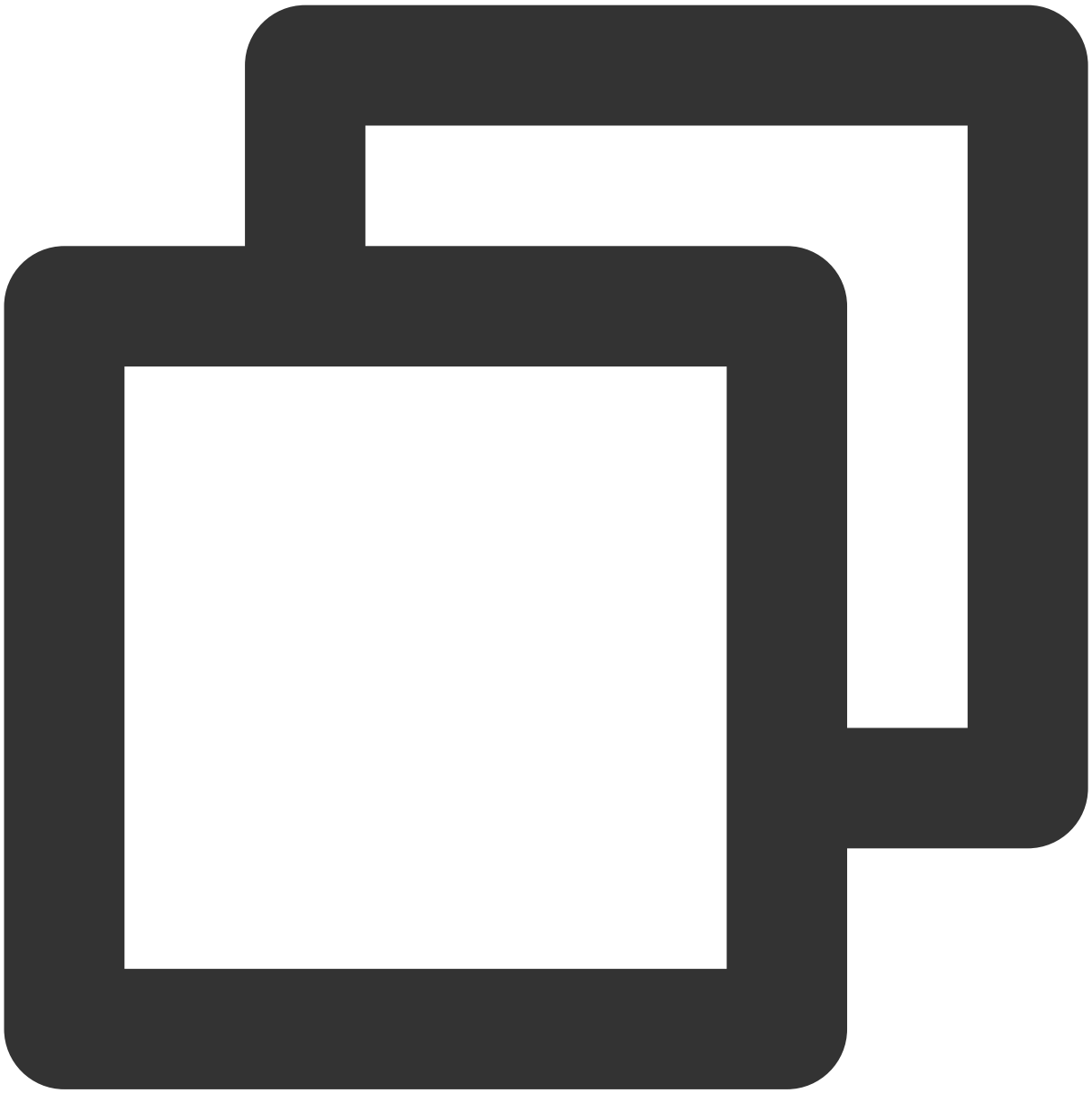
test.txt 파일 반환 내용이 test이면, 해당 파일에 정상적으로 액세스할 수 있다는 의미입니다.



2. **AJAX 기술을 사용해 파일 액세스**

이제 AJAX 기술을 사용해 직접 해당 test.txt 파일에 액세스를 시도해 봅니다.

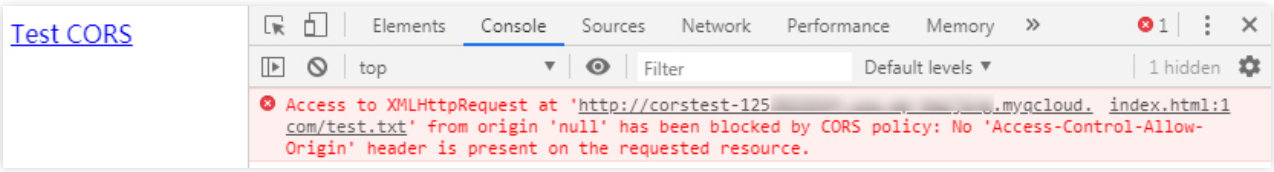
1. 간단한 HTML 파일을 생성하여 다음 코드를 복사한 후, 로컬에 HTML 파일로 저장하여 브라우저로 파일을 엽니다. 사용자 정의 헤더를 설정하지 않았기 때문에 해당 요청은 사전 인증이 필요 없습니다.



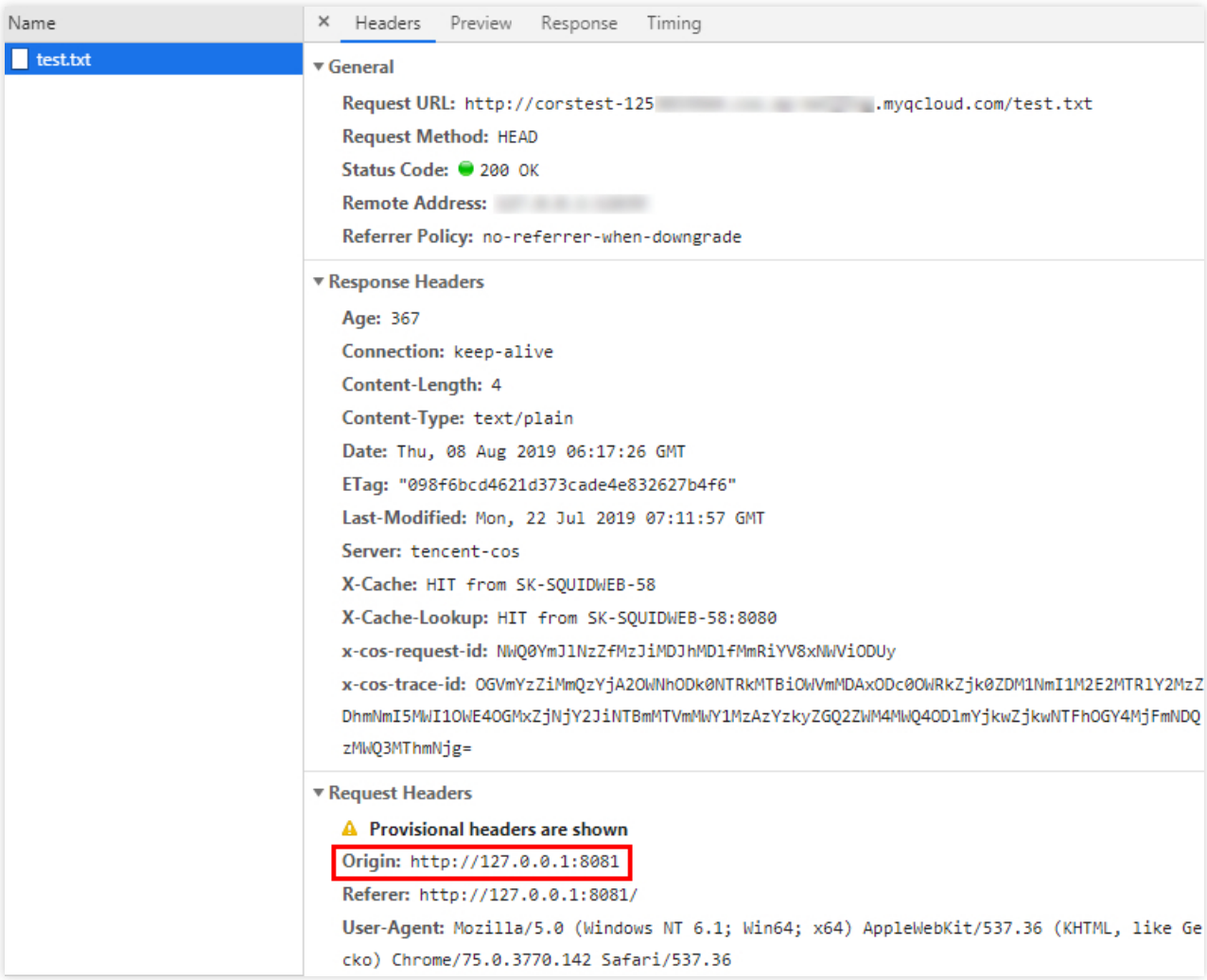
```
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
</head>
<body>
<a href="javascript:test()">Test CORS</a>
<script>
```

```
function test() {
    var url = 'http://corstest-125xxxxxxx.cos.ap-beijing.myqcloud.com/test.txt';
    var xhr = new XMLHttpRequest();
    xhr.open('HEAD', url);
    xhr.onload = function () {
        var headers = xhr.getAllResponseHeaders().replace(/\\r\\n/g, '\\n');
        alert('request success, CORS allow.\\n' +
            'url: ' + url + '\\n' +
            'status: ' + xhr.status + '\\n' +
            'headers:\\n' + headers);
    };
    xhr.onerror = function () {
        alert('request error, maybe CORS error.');
    };
    xhr.send();
}
</script>
</body>
</html>
```

2. 브라우저에서 해당 HTML 파일을 열고 **Test CORS**를 클릭하여 요청을 발송하면 다음 오류가 나타납니다. 오류 안내: 액세스 권한이 없습니다. 오류 발생 원인은 Access-Control-Allow-Origin Header를 찾을 수 없기 때문으로, 이는 서버에서 CORS를 설정하지 않기 때문입니다.



3. 액세스에 실패한 후, 다시 Header 인터페이스로 이동해 원인을 확인하면 브라우저에서 Origin이 있는 Request를 발송했다는 것을 확인할 수 있으며, 이에 따라 해당 요청은 크로스 도메인인 것을 알 수 있습니다.

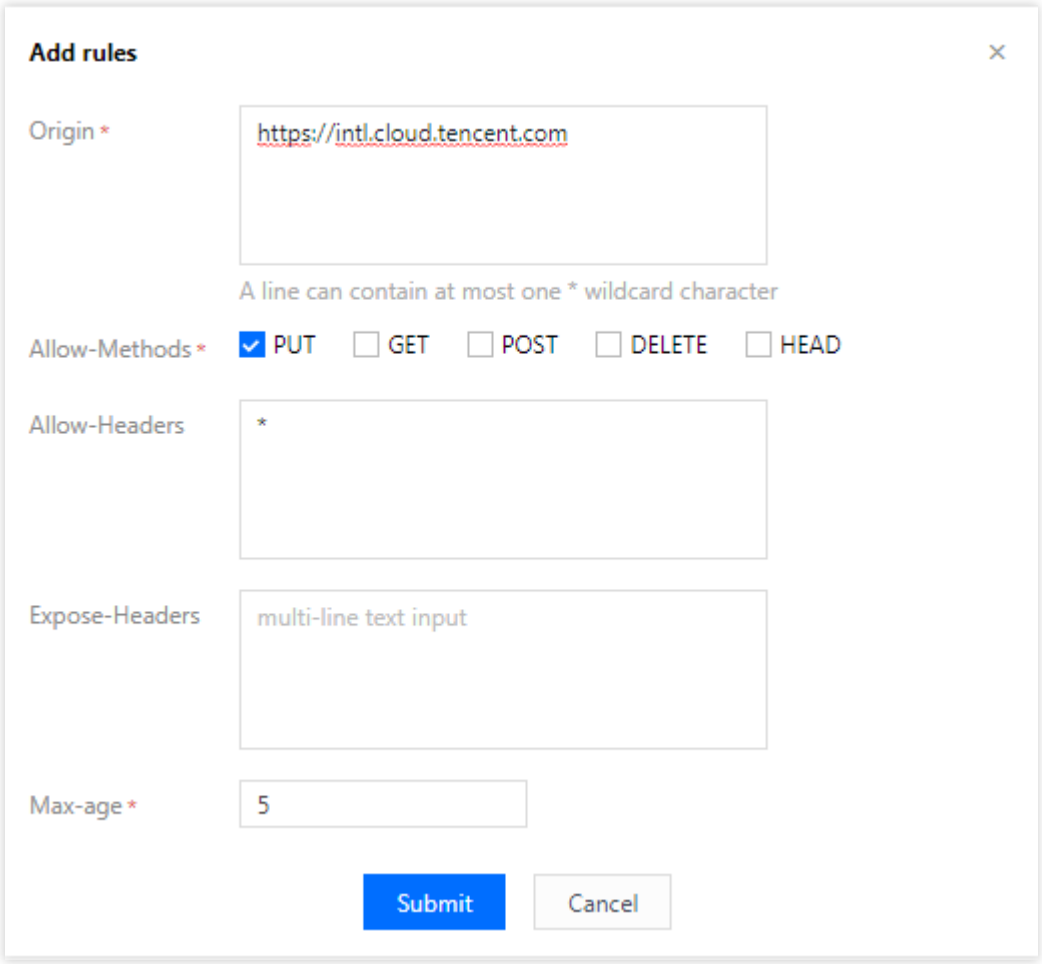


설명：
웹 페이지를 서버에 구축할 때 주소가 `http://127.0.0.1:8081` 이면 Origin은 `http://127.0.0.1:8081` 이 됩니다.

CORS 설정

액세스 실패 원인을 확인한 후, 버킷에 관련된 CORS 설정을 하여 해당 문제를 해결할 수 있습니다. COS 콘솔에서 CORS를 설정할 수 있으며, 본 예시에서는 콘솔을 사용해 CORS를 설정하는 방법을 소개합니다. 사용자의 CORS 설정이 특별하게 복잡하지 않다면 콘솔을 사용해 CORS를 설정하는 것을 권장합니다.

1. COS 콘솔에 로그인한 후 **버킷 리스트**를 클릭하여 해당 버킷으로 이동하고, **보안 관리** 탭을 클릭한 뒤 페이지를 내리면 "CORS 설정"을 찾을 수 있습니다.
2. **규칙 추가**를 클릭해 첫 번째 규칙을 추가하고, 다음과 같이 가장 완화된 설정을 사용합니다.

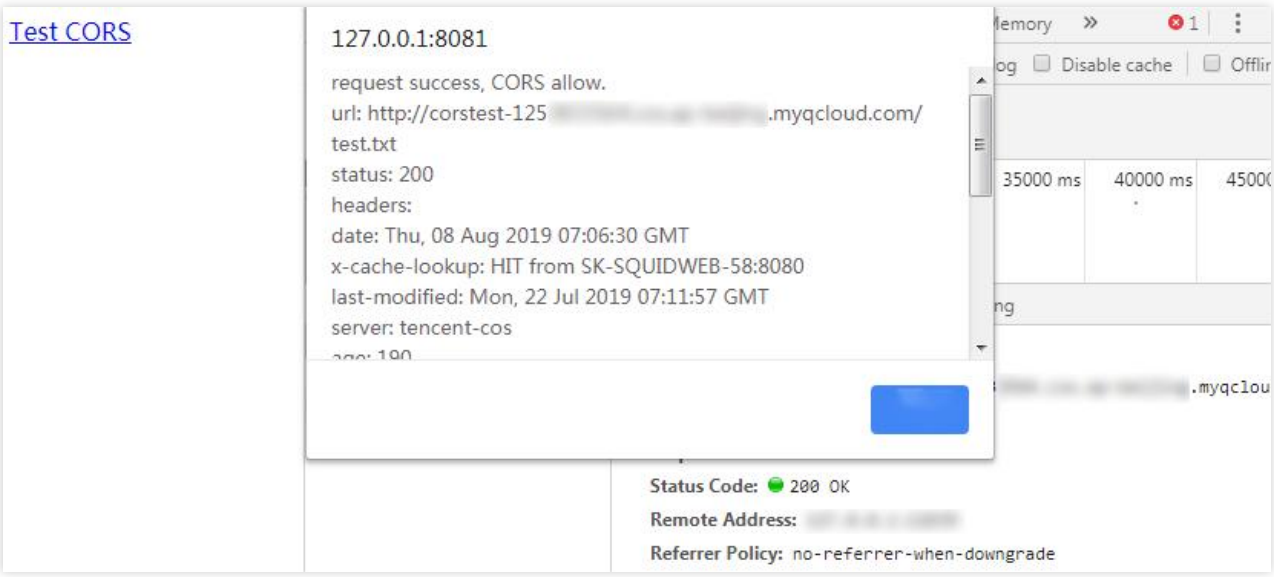


주의 :

CORS 설정은 규칙 하나 하나가 모여 구성되며, 첫 번째 규칙부터 차례로 매칭하고 가장 처음에 매칭된 규칙을 기준으로 합니다.

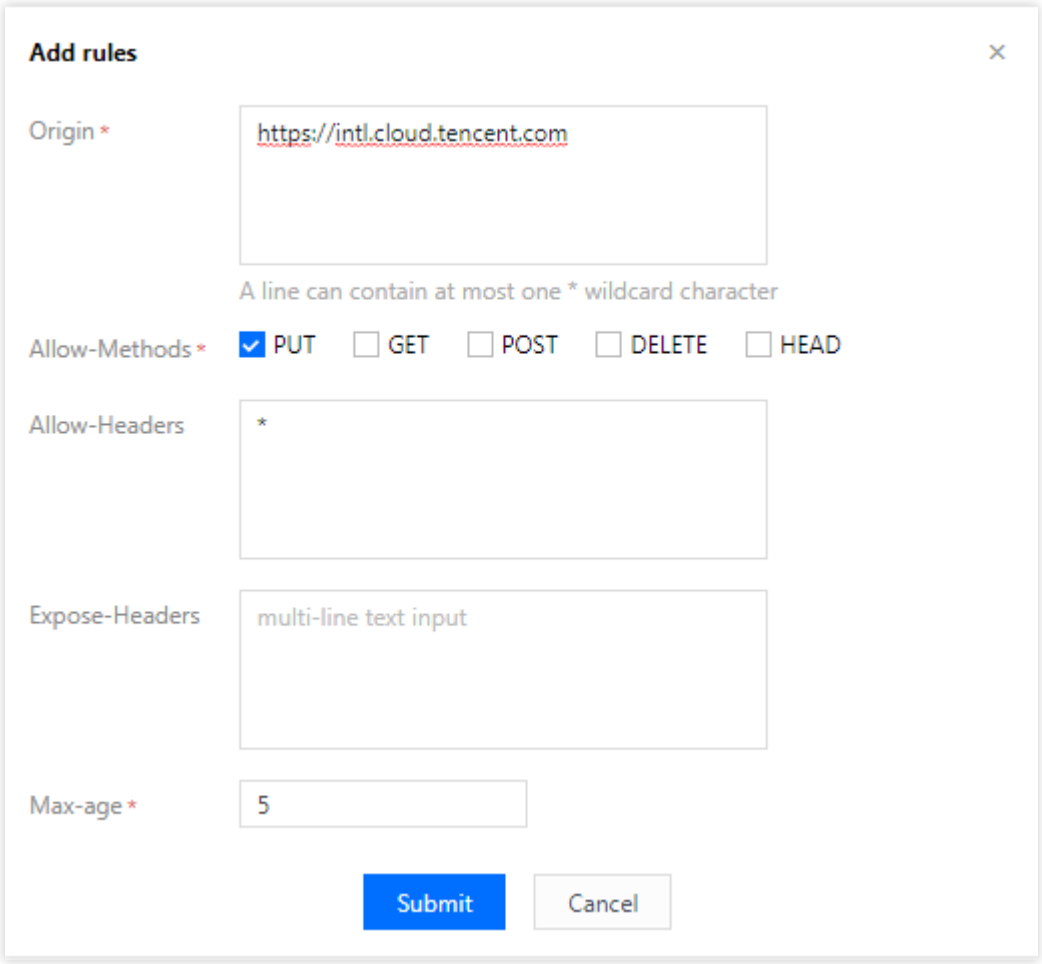
인증 결과

설정 완료 후, 다시 text.txt 텍스트 파일에 액세스를 시도합니다. 다음과 같은 결과가 나타나면 액세스 요청이 정상적으로 처리된 것입니다.



장애 해결 및 권장사항

크로스 도메인으로 인한 액세스 문제를 해결해야 하는 경우 CORS 설정을 다음과 같이 완화된 설정으로 설정합니다. 해당 설정은 모든 크로스 도메인 요청을 허용하며, 해당 설정 후에도 여전히 문제가 발생하는 경우 CORS가 아닌 다른 부분에서 오류가 발생한 것입니다.
가장 완화된 설정 이외에도, 더 자세한 제어 메커니즘을 설정하여 맞춤형 제어를 실현할 수 있습니다. 예를 들어, 본 예시에 다음과 같은 최소 설정을 사용해 매칭에 성공할 수 있습니다.



따라서 대부분의 시나리오에서는 실제 필요에 따라 최소 설정을 사용해 보안성을 보장하십시오.

CORS 설정 항목 설명

CORS 설정에는 다음과 같은 설정 항목이 있습니다.

출처 Origin

크로스 도메인 요청을 허용하는 출처입니다.
동시에 여러 개의 출처를 지정할 수 있으며, 한 행에 하나만 입력합니다.
`*` 설정을 지원하며, 모든 도메인을 허용한다는 의미로 권장하지 않습니다.
`http://www.abc.com` 과 같은 단일의 구체적인 도메인을 지원합니다.
`http://*.abc.com` 형식의 두 번째 레벨 와일드카드 도메인 이름이 지원되지만 각 줄에는 `*` 가 하나만 포함될 수 있습니다.
프로토콜명 HTTP 또는 HTTPS를 반드시 입력해야 하며, 포트가 기본값이 80이 아닌 경우 포트가 포함되어 있어야 합니다.

작업 Methods

허용하는 크로스 도메인 요청 방법(1개 이상)을 열거합니다.
예시: GET, PUT, POST, DELETE, HEAD

Allow-Header

허용하는 크로스 도메인 요청 Header입니다.
동시에 여러 개의 출처를 지정할 수 있으며, 한 행에 하나만 입력합니다.
Header는 쉽게 누락될 수 있으며, 특별히 필요한 경우가 아니라면 `*` 로 설정하는 것을 장합니다. 이는 모두 허용한다는 의미입니다.
대소문자를 구분하지 않습니다.
Access-Control-Request-Headers에 지정된 각 Header는 Allowed-Header의 값과 일치해야 합니다.

Expose-Header

브라우저에 노출되는 Header 리스트입니다. 즉, 사용자가 응용 프로그램에서 액세스하는 응답 헤더(예: Javascript의 XMLHttpRequest 객체)입니다.
구체적인 설정은 애플리케이션의 수요에 따라 확정되며, 기본적으로 Etag 입력을 권장합니다.
와일드카드 부호는 사용할 수 없으며, 대소문자를 구분하지 않고 각 행에 하나만 입력할 수 있습니다.

시간 초과 Max-Age

브라우저의 특정 리소스의 선취 요청(OPTIONS 요청)에 대한 반환 결과 캐시 시간으로, 단위는 초입니다. 특별한 요구가 없는 경우 약간 크게 설정할 수 있으며(예: 60초), 해당 항목은 선택 설정 항목입니다.

COS 정적 웹 사이트 기능을 사용하여 프런트엔드 단일 페이지 애플리케이션 구축

최종 업데이트 날짜: : 2024-06-24 16:44:04

단일 페이지 애플리케이션이란 무엇입니까?

단일 페이지 애플리케이션(single-page application, SPA)은 서버에서 전체 새 페이지를 다시 로딩하는 기존 방법 대신 현재 페이지를 동적으로 다시 작성하여 사용자와 상호 작용하는 웹 애플리케이션 또는 웹 사이트의 모델입니다. 이 접근 방식은 페이지 사이를 전환하여 사용자 경험이 중단되는 것을 방지하고 애플리케이션을 데스크톱 애플리케이션처럼 만들어 줍니다. SPA에서 필요한 모든 코드(HTML, JavaScript 및 CSS)는 단일 페이지 로딩으로 검색되거나 일반적으로 사용자 작업에 대한 응답으로 필요에 따라 적절한 리소스가 동적으로 로딩되고 페이지에 추가됩니다.
현재 프런트 엔드 개발 분야에서 일반적인 SPA 개발 프레임워크에는 React, Vue 및 Angular가 포함됩니다.
본문은 두 가지 인기 있는 프레임워크를 사용하여 **Tencent Cloud의 Cloud Object Storage(COS)**에서 제공하는 **정적 웹 사이트** 기능을 사용하여 온라인에서 사용 가능한 SPA를 빠르게 구축하는 방법을 설명하고 몇 가지 일반적인 문제에 대한 솔루션을 제공합니다.

준비 작업

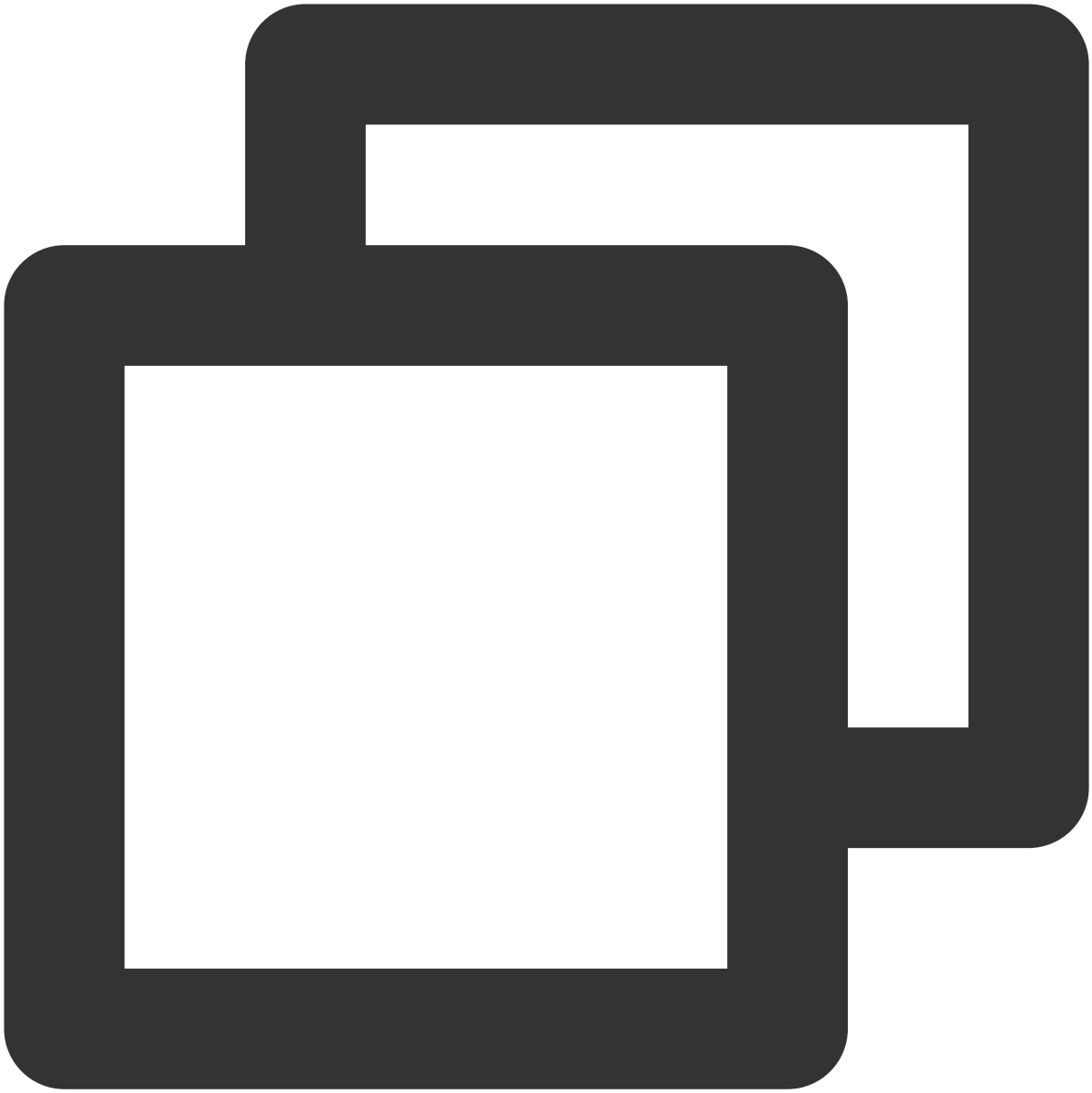
1. Node.js 환경을 설치합니다.
2. Tencent Cloud 계정 가입 및 실명 인증을 완료하여 Tencent Cloud COS 콘솔에 정상적으로 로그인할 수 있도록 합니다.
3. 버킷을 생성합니다(테스트를 용이하게 하려면 버킷 권한을 **공개 읽기/비공개 쓰기**로 설정).

프런트 엔드 코드 작성

주의 :
이미 코드를 직접 구현했다면 버킷 정적 웹 사이트 설정 활성화 단계로 건너뛰십시오.

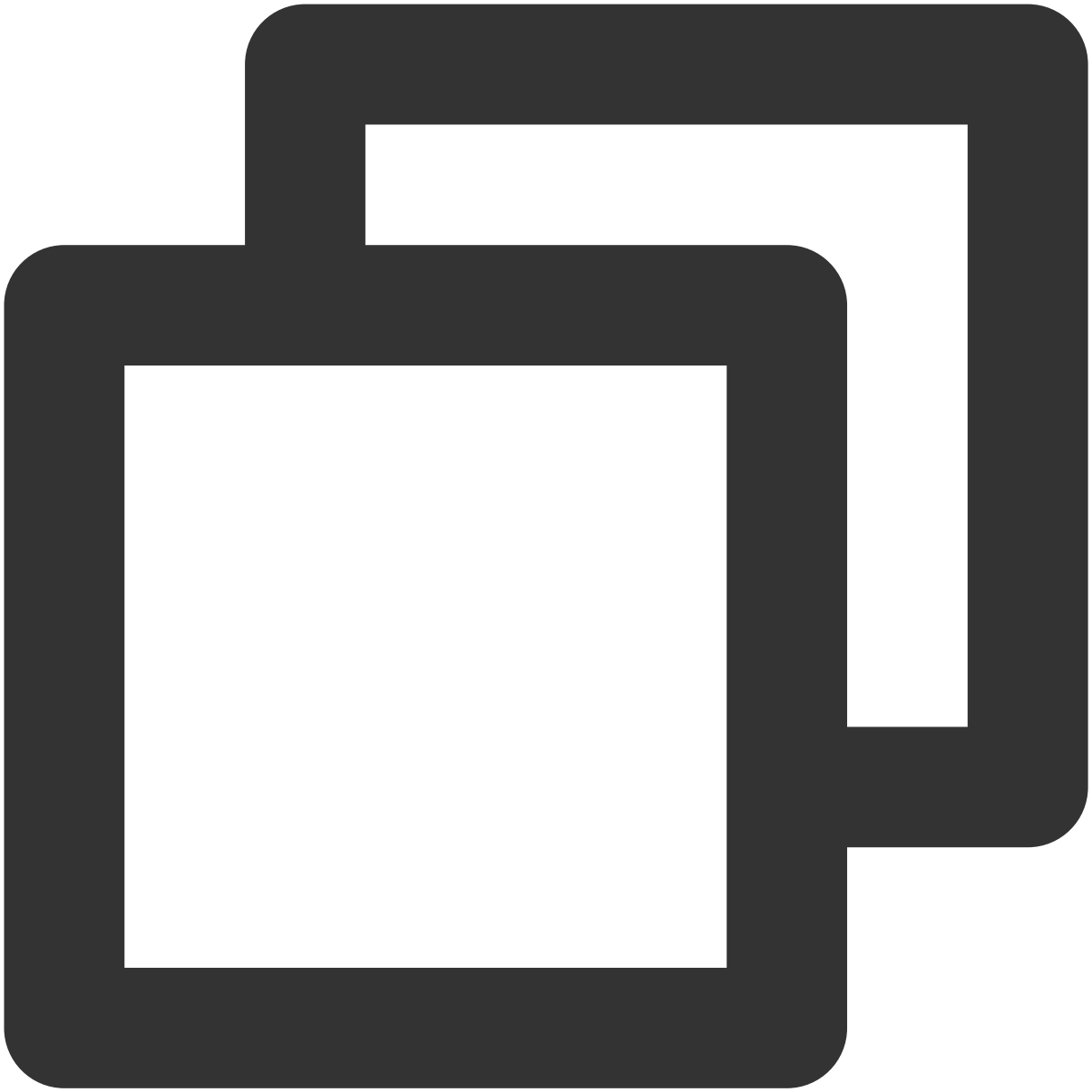
Vue를 사용하여 단일 페이지 애플리케이션 빠르게 구축

1. 다음 명령어를 실행하여 vue-cli를 설치합니다.



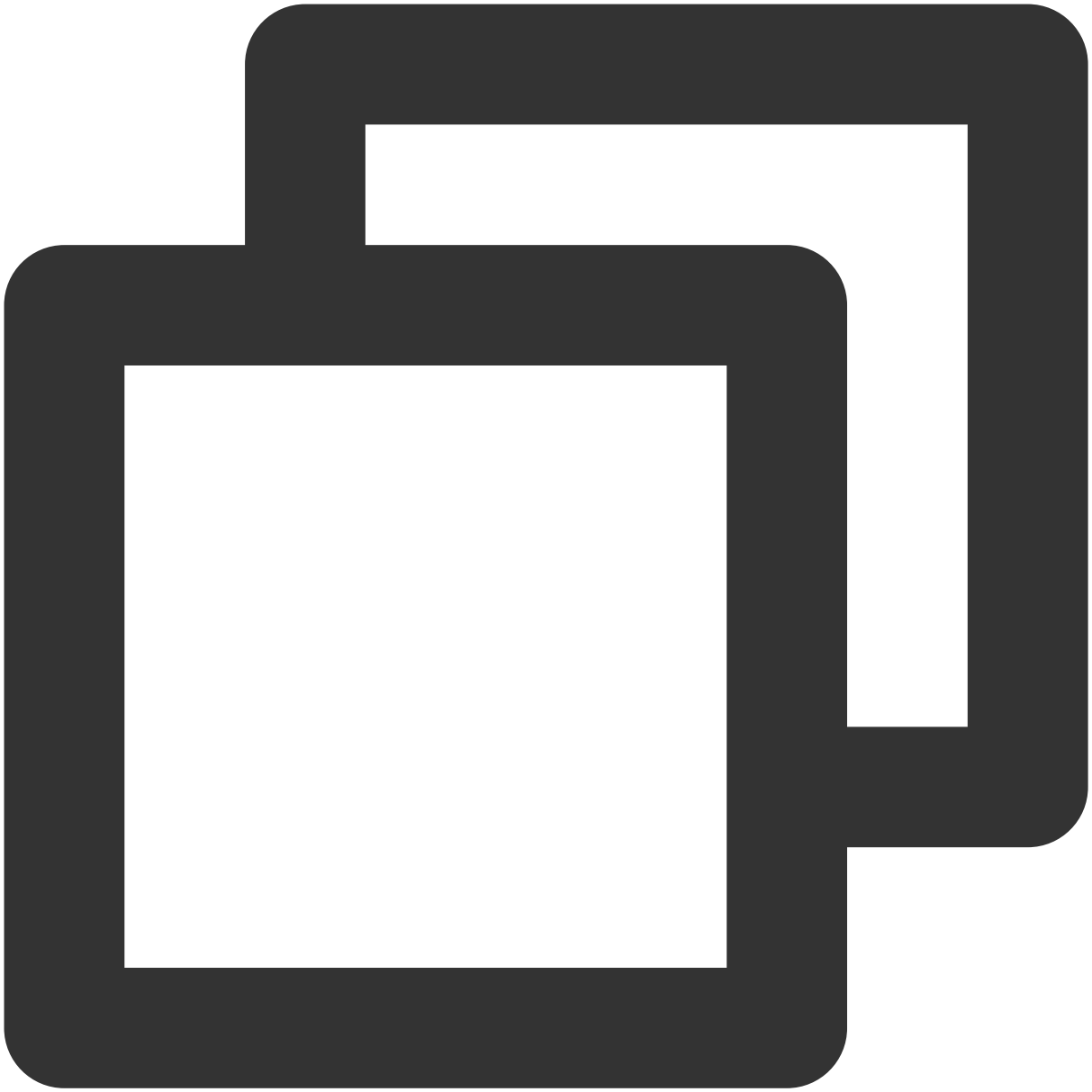
```
npm install -g @vue/cli
```

2. vue-cli에서 다음 명령을 실행하여 vue 프로젝트를 빠르게 생성합니다. 자세한 내용은 공식 문서를 참고하십시오.



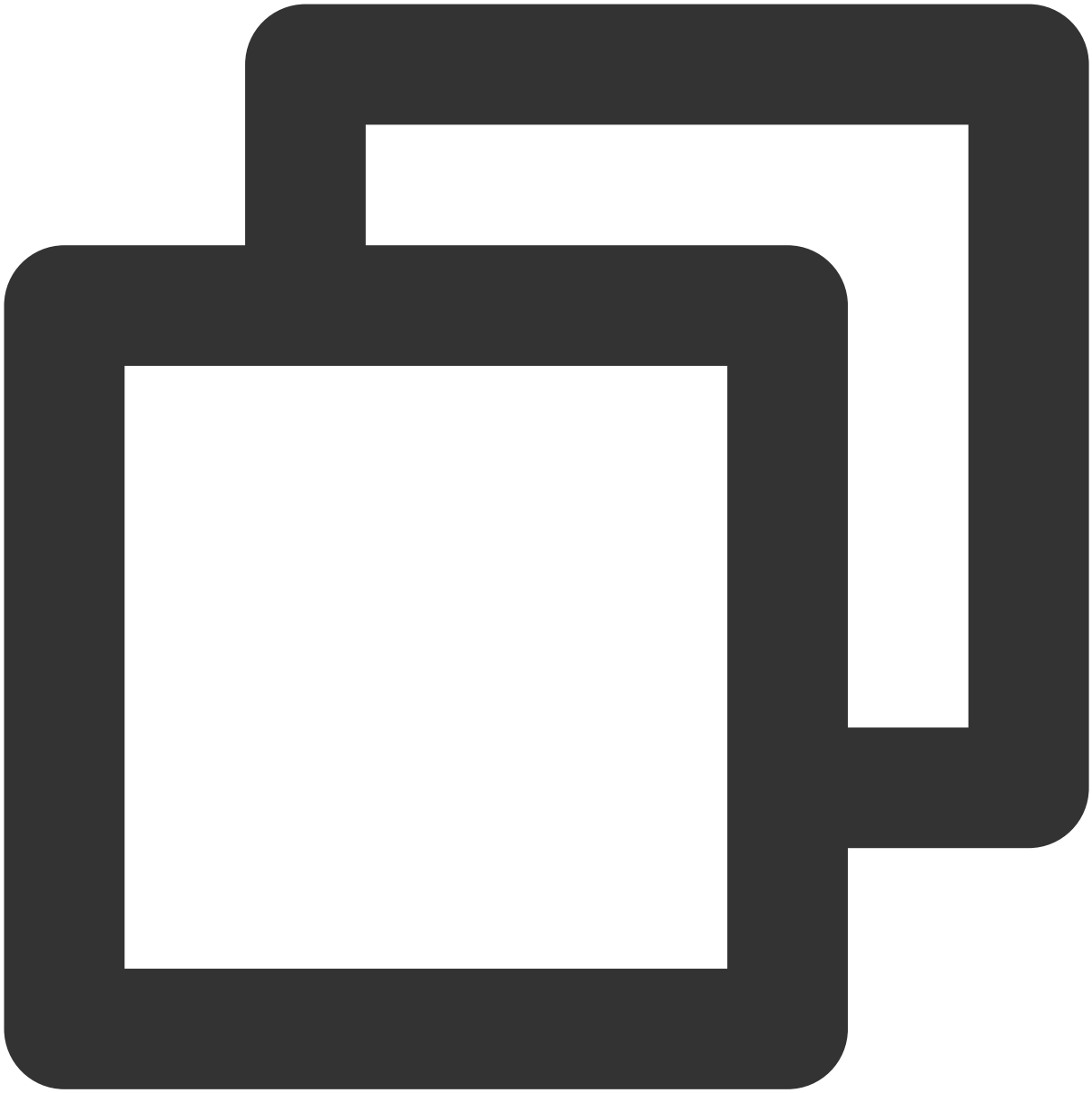
```
vue create vue-spa
```

3. 다음 명령어를 실행하여 프로젝트 루트 디렉터리에 vue-router를 설치합니다.



```
npm install vue-router -S  (Vue2.x)
```

또는



```
npm install vue-router@4 -S  (Vue3.x)
```

4. 프로젝트에서 main.js 및 App.vue 파일을 수정합니다.

main.js는 아래 이미지와 같습니다:

```
import { createApp } from 'vue'
import { createRouter, createWebHistory } from 'vue-router'
import App from './App.vue'
import Home from './components/Home.vue'
import Foo from './components/Foo.vue'
import Bar from './components/Bar.vue'
import Default from './components/404.vue'

const routes = [
    { path: '/', component: Home },
    { path: '/foo', component: Foo },
    { path: '/bar', component: Bar },
    { path: '/:w+', component: Default }
]

const router = createRouter({
    history: createWebHistory(),
    routes
})

const app = createApp(App)
app.use(router)
app.mount('#app')
```

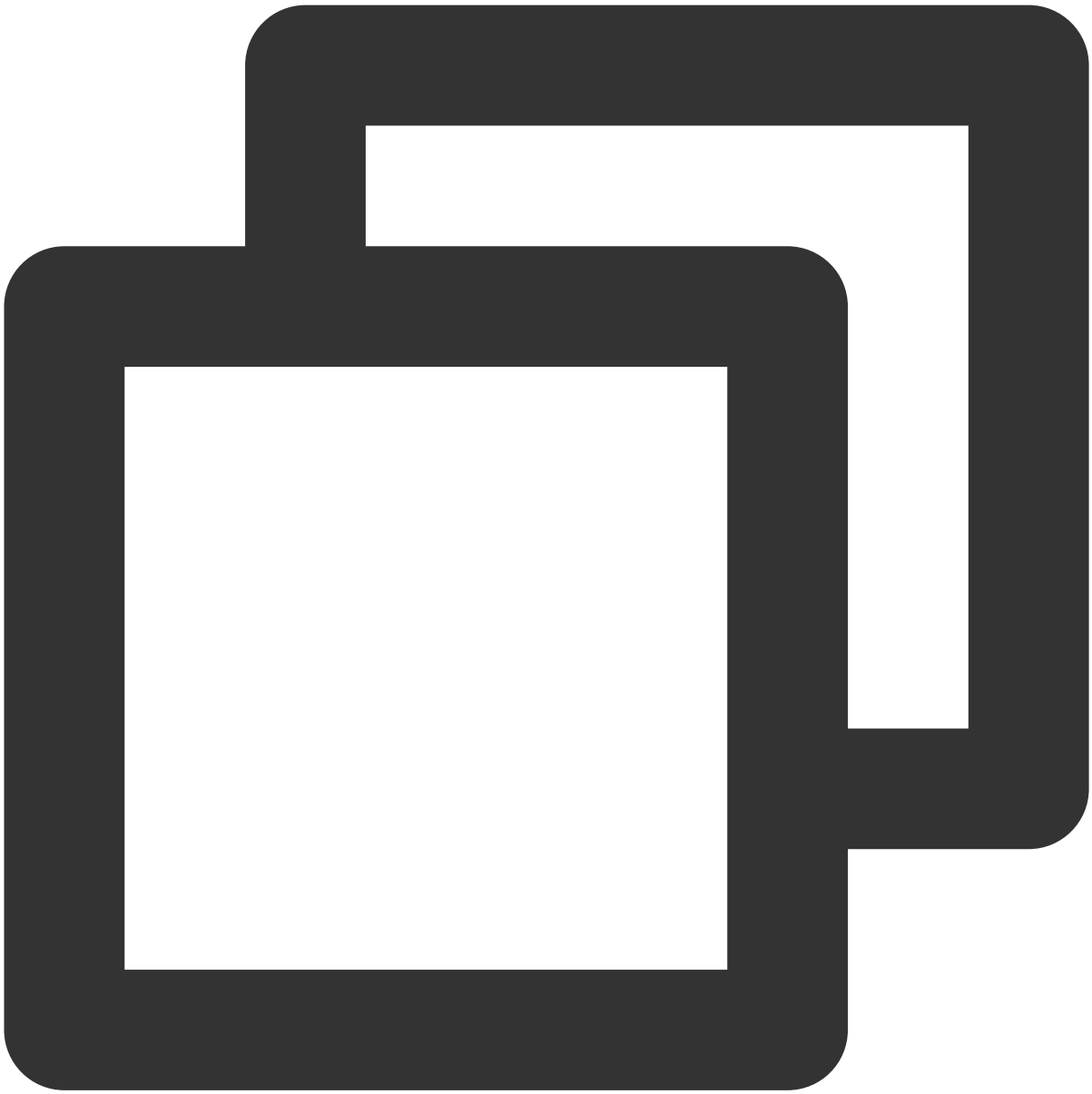
주로 아래 이미지와 같이 App.vue 컴포넌트의 template을 수정합니다.

```
<template>
  <div>
    <ul>
      <li>
        <router-link to="/">Home</router-link>
      </li>
      <li>
        <router-link to="/foo">Foo</router-link>
      </li>
      <li>
        <router-link to="/bar">Bar</router-link>
      </li>
    </ul>
    <img alt="Vue logo" src="./assets/logo.png">
    <router-view></router-view>
  </div>
</template>
```

설명：

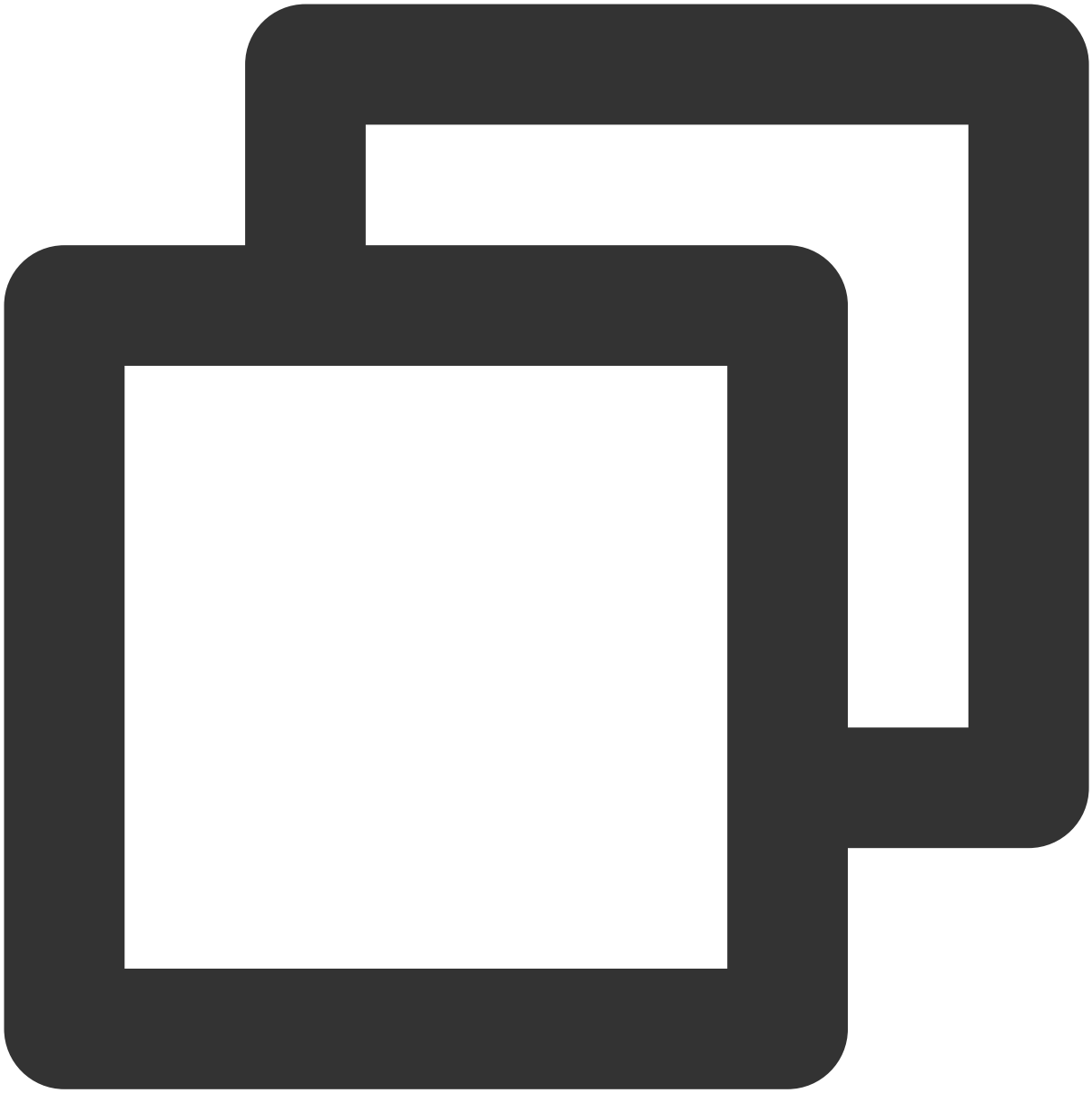
본문에서는 일부 키 코드만 소개합니다. 전체 코드는 여기를 클릭하여 다운로드할 수 있습니다.

5. 코드 수정 후 다음 명령어를 실행하여 로컬 미리보기를 실행합니다.



```
npm run serve
```

6. 디버깅 및 미리보기에 오류 없음을 확인한 후, 다음 명령어를 실행하여 프로덕션 환경 코드를 패키징 합니다.

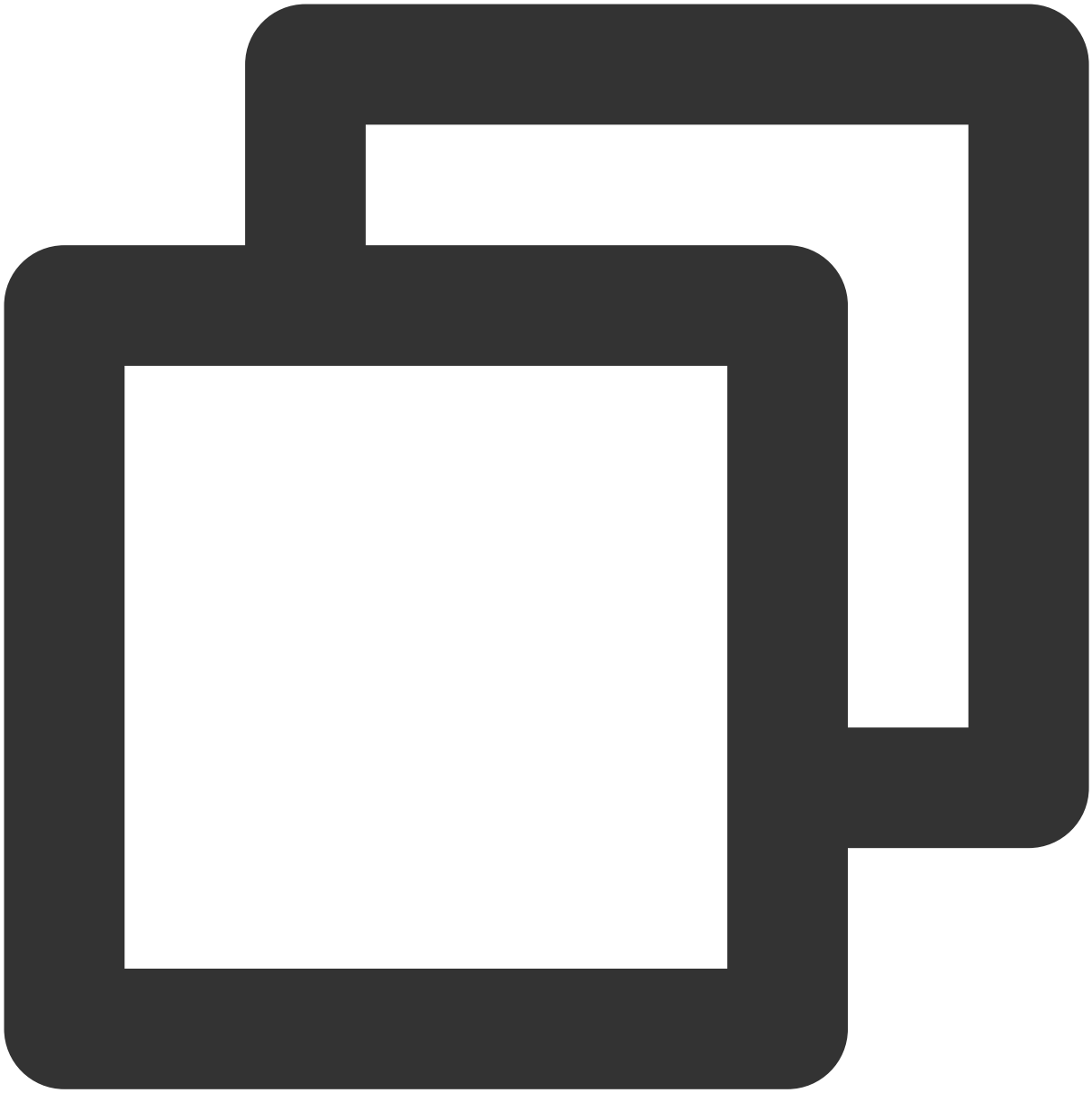


```
npm run build
```

이 때 프로젝트 루트 디렉터리에 dist 디렉터리가 생성되며 Vue 프로그램 코드가 준비됩니다.

React를 사용하여 단일 페이지 애플리케이션을 빠르게 구축

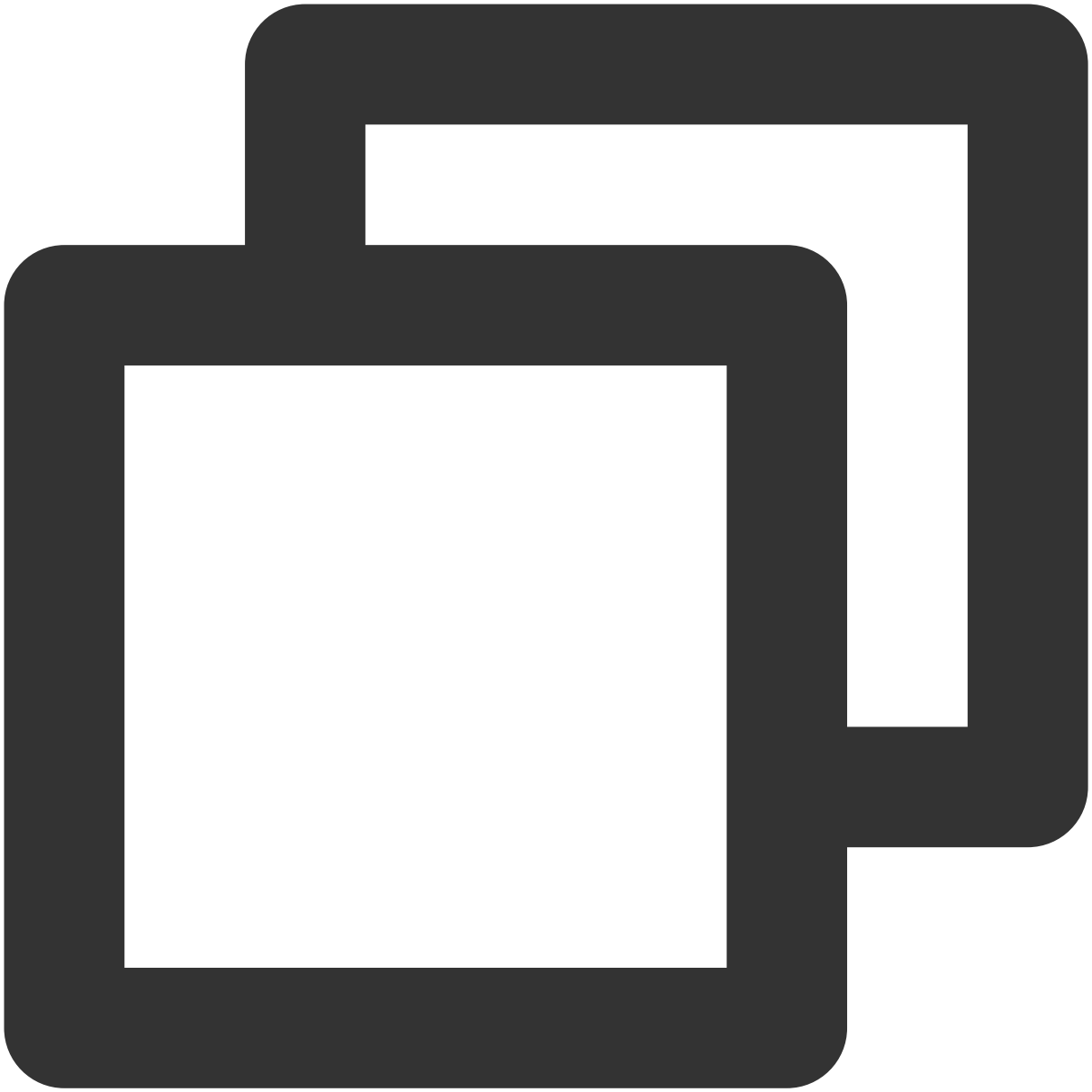
1. 다음 명령어를 실행하여 create-react-app을 설치합니다.



```
npm install -g create-react-app
```

2. create-react-app을 사용하여 react 프로젝트를 빠르게 생성합니다. 공식 홈페이지 문서를 참고하십시오.

3. 다음 명령어를 실행하여 프로젝트 루트 디렉터리에 react-router-dom을 설치합니다.



```
npm install react-router-dom -S
```

4. 프로젝트에서 App.js 파일을 수정합니다.

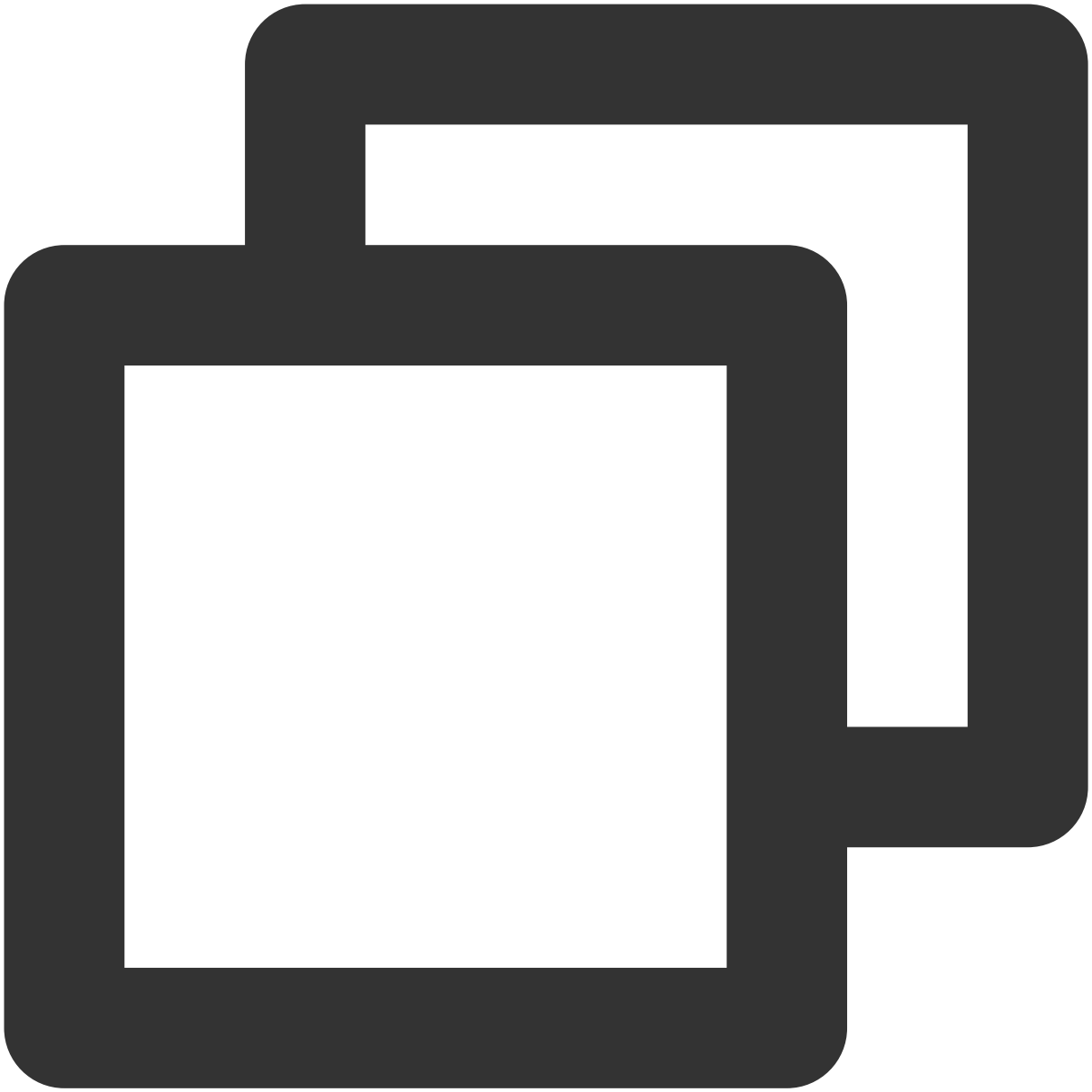
```
import React from 'react';
import { BrowserRouter as Router, Switch, Route, Link } from 'react-router-dom'
import './App.css';

function Home() {
  return <h2>Home</h2>;
}

function About() {
  return <h2>About</h2>;
}

function NoMatch() {
  return <h2>404 Page</h2>
}

function App() {
  return (
    <Router>
      <div className="App">
        <nav>
          <ul>
            <li>
              <Link to="/">Home</Link>
            </li>
            <li>
              <Link to="/about">About</Link>
            </li>
          </ul>
        </nav>
        <Switch>
          <Route exact path="/">
            <Home />
          </Route>
          <Route path="/about">
            <About />
          </Route>
          <Route path="*">
            <NoMatch />
          </Route>
        </Switch>
      </div>
    </Router>
  );
}

export default App;
```

</br>

설명：

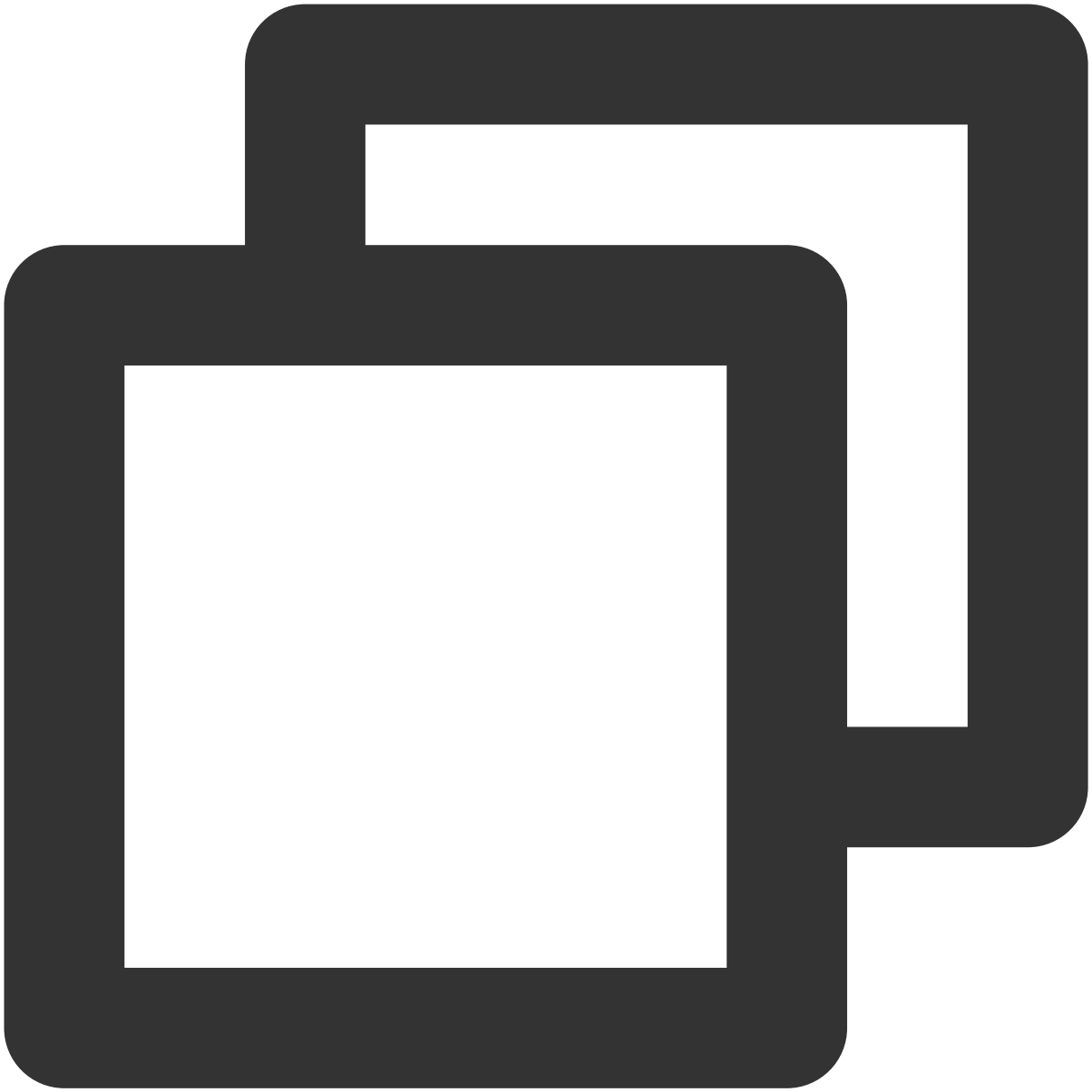
본문에서는 일부 키 코드만 소개합니다. 전체 코드는 여기를 클릭하여 다운로드할 수 있습니다.

5. 코드 수정 후 다음 명령어를 실행하여 로컬 미리보기를 실행합니다.



```
npm run start
```

6. 디버깅 및 미리보기에 오류 없음을 확인한 후, 다음 명령어를 실행하여 프로덕션 환경 코드를 패키징 합니다.



```
npm run build
```

이 때 프로젝트 루트 디렉터리에 build 디렉터리가 생성되며 React 프로그램 코드가 준비됩니다.

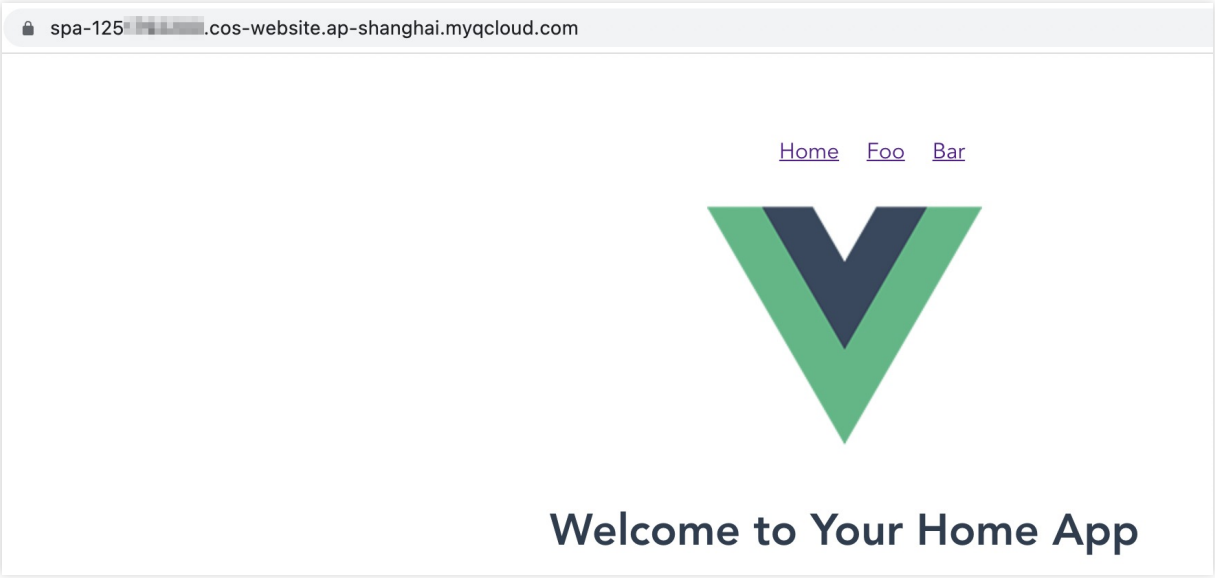
버킷 정적 웹 사이트 설정 활성화

1. 생성된 버킷의 세부 정보 페이지로 이동하여 **기본 구성** > **정적 웹 사이트**를 선택합니다.

2. 정적 웹 사이트 관리 페이지에서 아래 이미지를 참고하여 설정합니다. 자세한 작업은 정적 웹 사이트 설정을 참고하십시오.

COS에 배포

1. 이전에 설정되어 있던 정적 웹 사이트의 버킷을 찾아서 파일 리스트 페이지로 들어갑니다.
2. 컴파일 디렉터리(Vue 기본값: dist 디렉터리, react 기본값: build 디렉터리)의 모든 파일을 버킷의 루트 디렉터리에 업로드합니다. 자세한 작업은 객체 업로드를 참고하십시오.
3. 버킷의 정적 웹 사이트 도메인(아래 이미지의 액세스 노드)에 액세스합니다.
배포된 애플리케이션의 메인 페이지를 볼 수 있습니다. 다음은 Vue 애플리케이션 예시입니다.



4. 라우팅(Home, Foo, Bar)을 전환하고 페이지를 새로고침하여 예상에 부합하는지 확인합니다(즉, 라우팅에서 새로고침할 때 404 오류가 발생하지 않음).

FAQ

정적 웹사이트의 기본 도메인 이름을 사용하고 싶지 않으면 어떻게 하나요? 내 도메인 이름을 사용할 수 있나요?

상기 기본 정적 웹사이트 엔드포인트 외에도 COS를 사용하면 사용자 정의 CDN 가속 도메인 이름과 사용자 정의 엔드포인트를 설정할 수 있습니다. 구성 세부 정보는 도메인 관리 개요를 참고하십시오. 구성에 성공하면 원하는 도메인 이름을 사용하여 애플리케이션에 액세스할 수 있습니다.
CDN 가속 도메인 이름 구성을 선택한 경우 노드 캐시 유효성 구성을 참고하여 업데이트된 데이터를 얻으십시오.

애플리케이션 배포 후 라우팅 전환 시 렌더링이 성공적이지만 페이지 새로고침 시 404 오류 발생 원인이 무엇인가요?

가능한 원인은 정적 웹 사이트를 설정할 때 설정이 누락되었거나 **오류 문서**가 잘못 설정되었기 때문일 수 있습니다. 본문 시작 부분의 표준 설정 스크린샷에서 오류 문서와 인덱스 문서 모두 `index.html` 로 설정되어 있는 것을 확인할 수 있습니다.
단일 페이지 애플리케이션의 특성으로 인해 후속 라우팅을 위한 일련의 내부 로직을 트리거 하려면 어떤 상황에서도 애플리케이션 게이트(일반적으로 'index.html')에 성공적으로 액세스할 수 있는지 확인해야 합니다.

라우팅 전환 후 페이지가 정상적으로 표시되지만 HTTP 상태 코드는 여전히 404입니다. 어떻게 해야 정상적으로 200이 반환됩니까?

원인은 정적 웹 사이트를 설정할 때 설정 **오류 문서 응답 코드**가 부족하기 때문입니다. 본문 시작 부분의 표준 설정 스크린샷을 참고하여 200으로 설정하면 해결됩니다.

오류 문서 설정 후 오류 액세스 경로에 여전히 404 기능을 표시하려면 어떻게 처리해야 하나요?

프런트 엔드 코드에서 404 로직을 구현하는 것을 추천합니다. 라우팅 설정 하단에 하위 수준에 일치하는 규칙을 설정하고 이전의 모든 규칙이 일치하지 않을 때 404 컴포넌트가 렌더링됩니다. 컴포넌트 콘텐츠는 필요에 따라 설계 및 구현할 수 있습니다. 자세한 내용은 본문에서 제공하는 코드 demo의 라우팅 설정의 마지막 설정을 참고하십시오.

페이지 액세스 시 403 Access Denied 오류가 발생하는 이유는 무엇입니까?

버킷의 권한이 **개인 읽기/쓰기**로 설정되어 있을 수 있습니다. **공개 읽기/개인 쓰기**로 수정하여 해결할 수 있습니다.
또한 CDN 가속 도메인을 사용하여 **개인 읽기/쓰기** 버킷에 액세스하는 경우 **Origin-pull 인증** 설정도 활성화해야 CDN에 COS 리소스 액세스 권한이 부여됩니다.

이미지 처리 사례
COS 오디오/비디오 플레이어 개요

최종 업데이트 날짜: : 2024-06-24 16:44:04

작업 시나리오

CI의 기본 이미지 처리 기능으로 이미지에 이미지 워터마크 또는 텍스트 워터마크를 추가할 수 있습니다. 그러나 실제 비즈니스 시나리오에서는 이미지에 고정 Logo 워터마크와 동적으로 변경되는 텍스트 워터마크(사용자 이름)가 모두 포함될 수 있습니다. 이러한 시나리오를 위해 다음과 같은 통합 방법이 제공됩니다. 실제 비즈니스 시나리오에 따라 적절한 것을 선택할 수 있습니다.

메소드 비교

메소드	장점	단점
1	통합 및 구현이 쉽습니다.	워터마크 크기는 이미지 크기에 따라 동적으로 변경될 수 없으며, 채우기할 수 없습니다.
2	이미지 크기가 자주 변경될 때 이미지에 더 잘 맞을 수 있습니다.	통합이 어렵고 처리 요금이 두 번 청구됩니다.

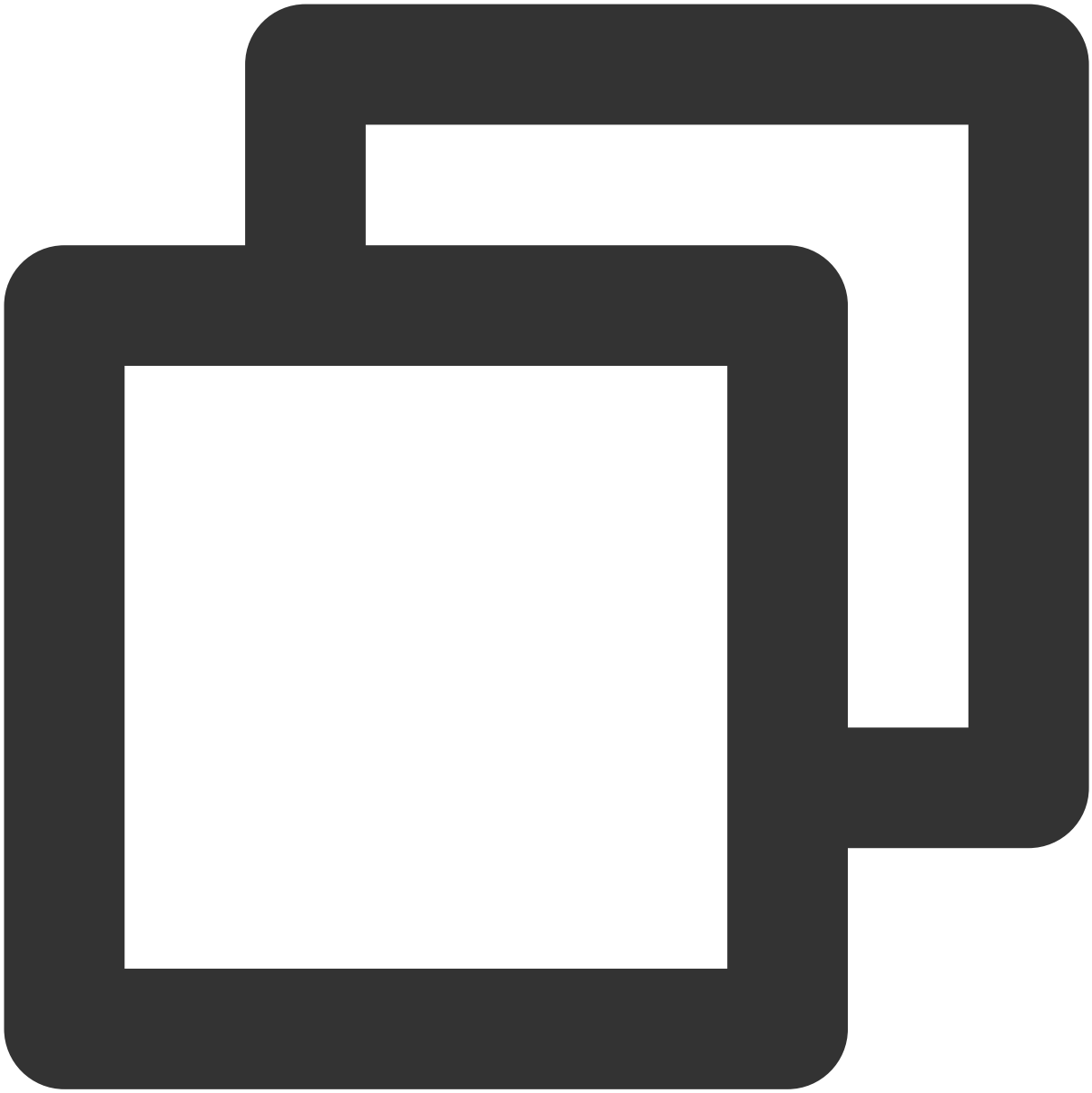
사용 방법

방법1: 파이프라인 연산자를 사용하여 URL 하나만으로 두 가지 유형의 워터마크 추가

CI(Cloud Infinite)의 기본 이미지 처리 기능을 통해 파이프라인 연산자 "|"를 사용할 수 있습니다. 처리 및 트래픽 요금의 일회성 요금으로, 단 한 번의 요청으로 이미지를 여러 번 처리합니다. 이 방법은 반복 요청으로 인한 대기 시간과 추가 요금을 크게 줄입니다.

작업 순서

1. 이미지 워터마크에 설명된 대로 이미지 워터마크 매개변수를 정의합니다.
API 매개변수에 익숙하지 않은 경우 콘솔에서 스타일을 추가하여 기본 처리에 설명된 대로 매개변수를 생성할 수 있습니다.
다음은 처리 매개변수입니다.



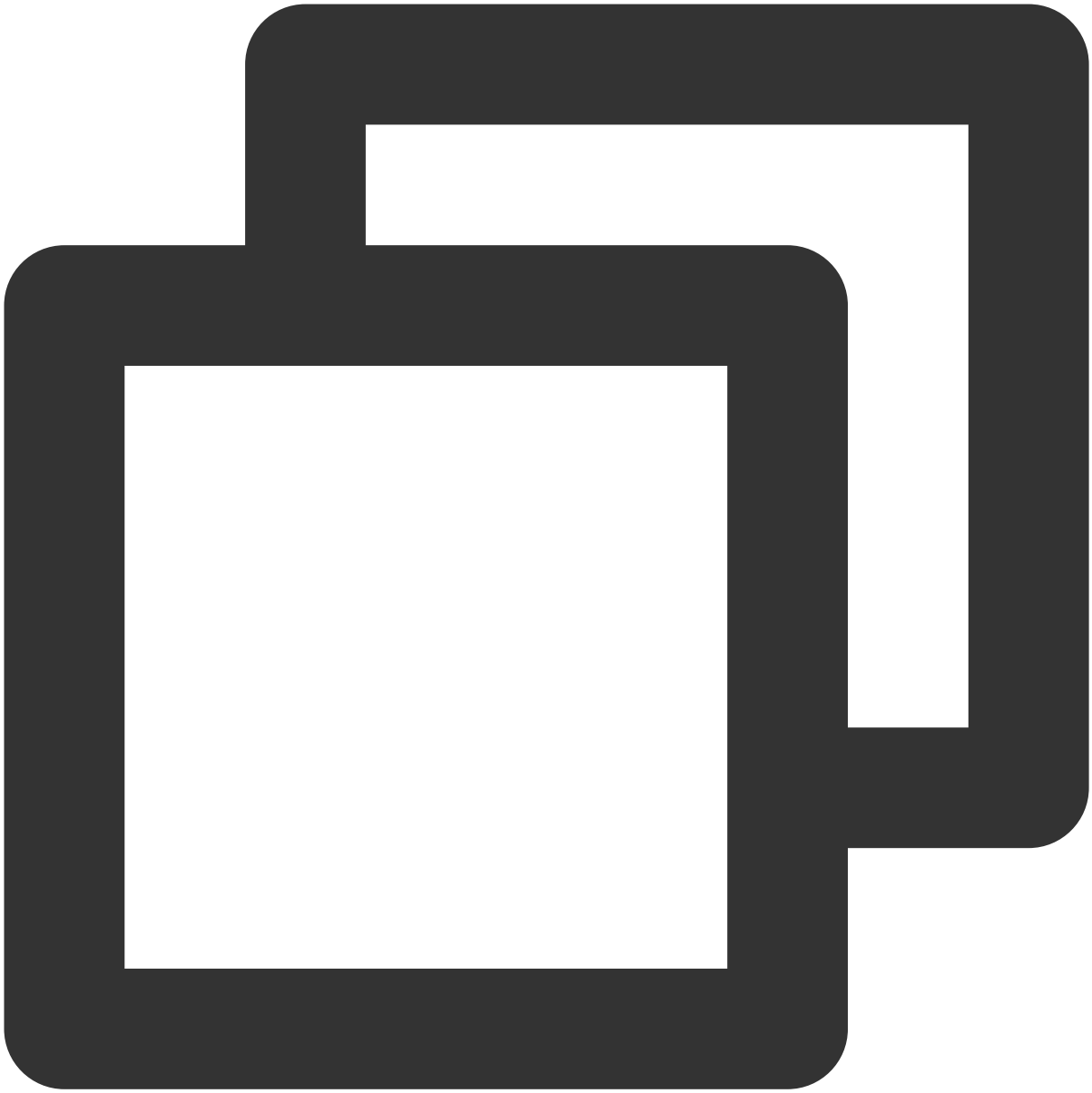
```
watermark/1/image/aHR0cDovL2V4YW1wbGVzLTEyNTgxMjU2MzguY29zLmFwLWd1YW5nemhvdS5teXFjb
```

설명：

여기서 `aHR0cDovL2V4YW1wbGVzLTEyNTgxMjU2MzguY29zLmFwLWd1YW5nemhvdS5teXFjbG91ZC5jb20vbG9nby5wbmc` 는 이미지 워터마크(COS bucket에 저장된 이미지)의 URL 보안 base64 인코딩 URL입니다.

2. 텍스트 워터마크에 설명된 대로 텍스트 워터마크 매개변수를 정의합니다.

API 매개변수에 익숙하지 않은 경우 콘솔에서 스타일을 추가하여 기본 처리에 설명된 대로 매개변수를 생성할 수 있습니다.

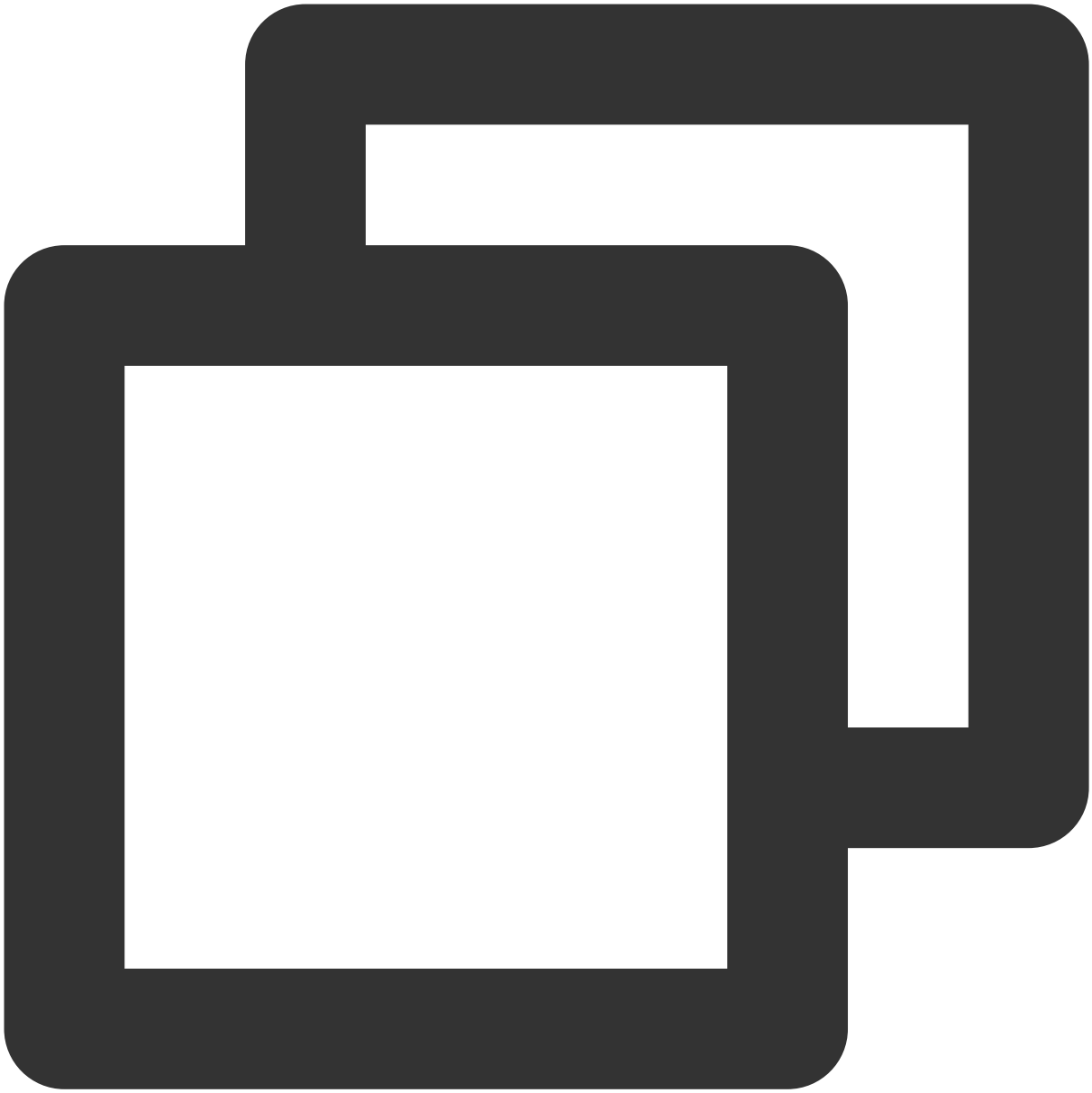


```
watermark/2/text/VUlOOiAxMjM0NTY3OA/font/SGVsdmV0aWNhLmRmb250/fontsize/36/fill/IzAw
```

설명：

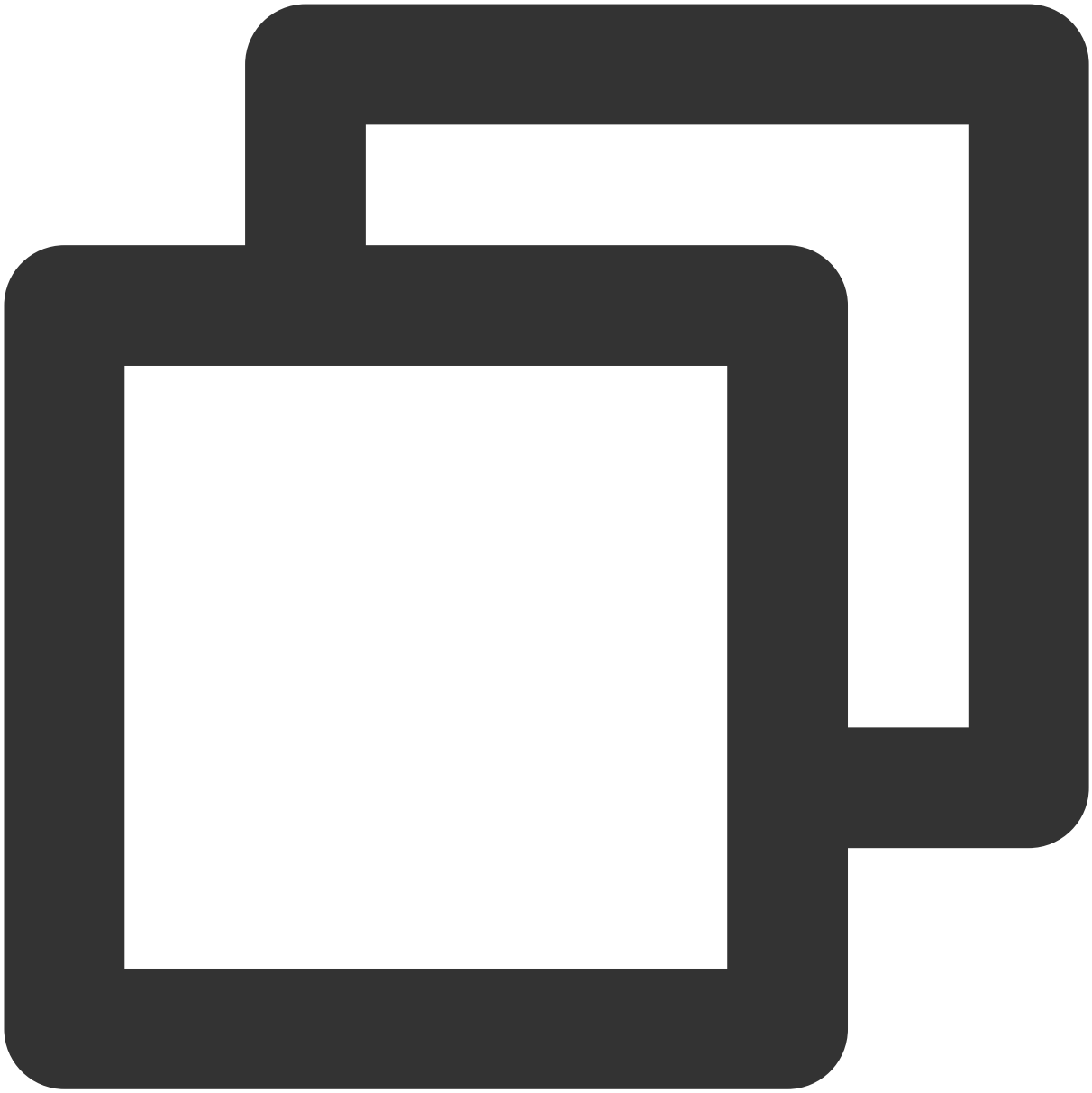
여기서 `VUlOOiAxMjM0NTY3OA` 는 `UIN: 12345678` 의 URL 보안 base64 인코딩 텍스트입니다.

3. 파이프라인 연산자를 사용하여 이미지 및 텍스트 워터마크 매개변수를 연결합니다.



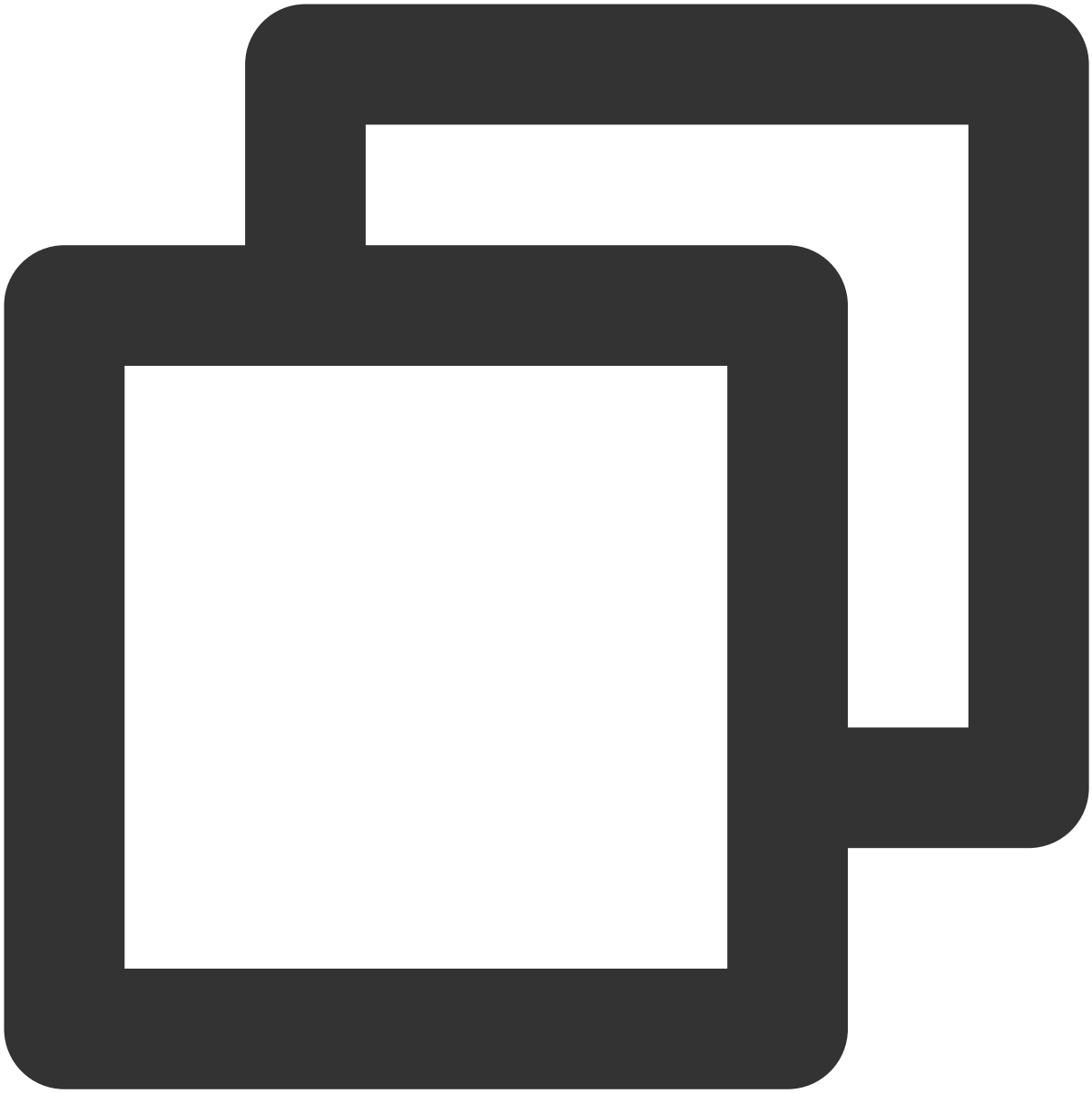
```
watermark/2/text/VUlOOiAxMjM0NTY3OA/font/SGVsdmV0aWNhLmRmb250/fontsize/36/fill/IzAw
```

4. 이미지 다운로드 URL 끝에 연결된 매개변수를 추가합니다.



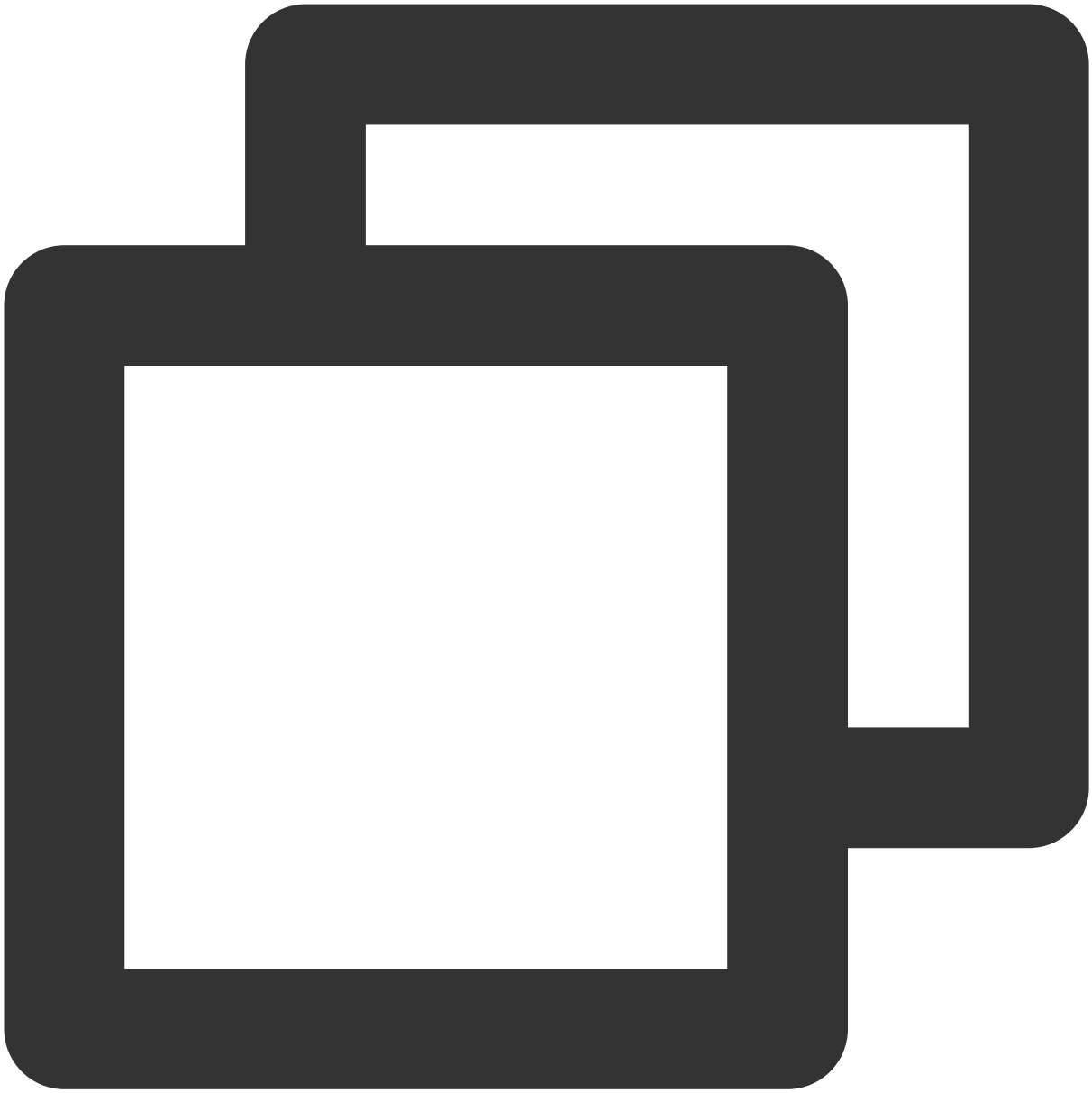
```
https://examples-1258125638.cos.ap-guangzhou.myqcloud.com/preview.png?watermark/2/t
```

혼합 워터마크 이미지를 가져올 수 있습니다.
URL을 단축하려면 기본 처리의 지침에 따라 콘솔에서 `watermark1` 스타일로 이미지 워터마크(변경되지 않음)의 일부를 추가할 수 있습니다.
이런 식으로 URL은 다음과 같이 단축될 수 있습니다.



```
https://examples-1258125638.cos.ap-guangzhou.myqcloud.com/preview.png/watermark1?wa
```

추후에 텍스트 콘텐츠를 변경해야 하는 경우 URL의 `VUlOOiAxMjM0NTY3OA` 만 업데이트된 base64 코드로 바꾸면 됩니다. 예를 들어 `UIN: 88888888` 은 `VUlOOiA4ODg4ODg4OA` 로 인코딩되므로 텍스트를 바꾸려면 URL을 다음 콘텐츠로 변경하기만 하면 됩니다.



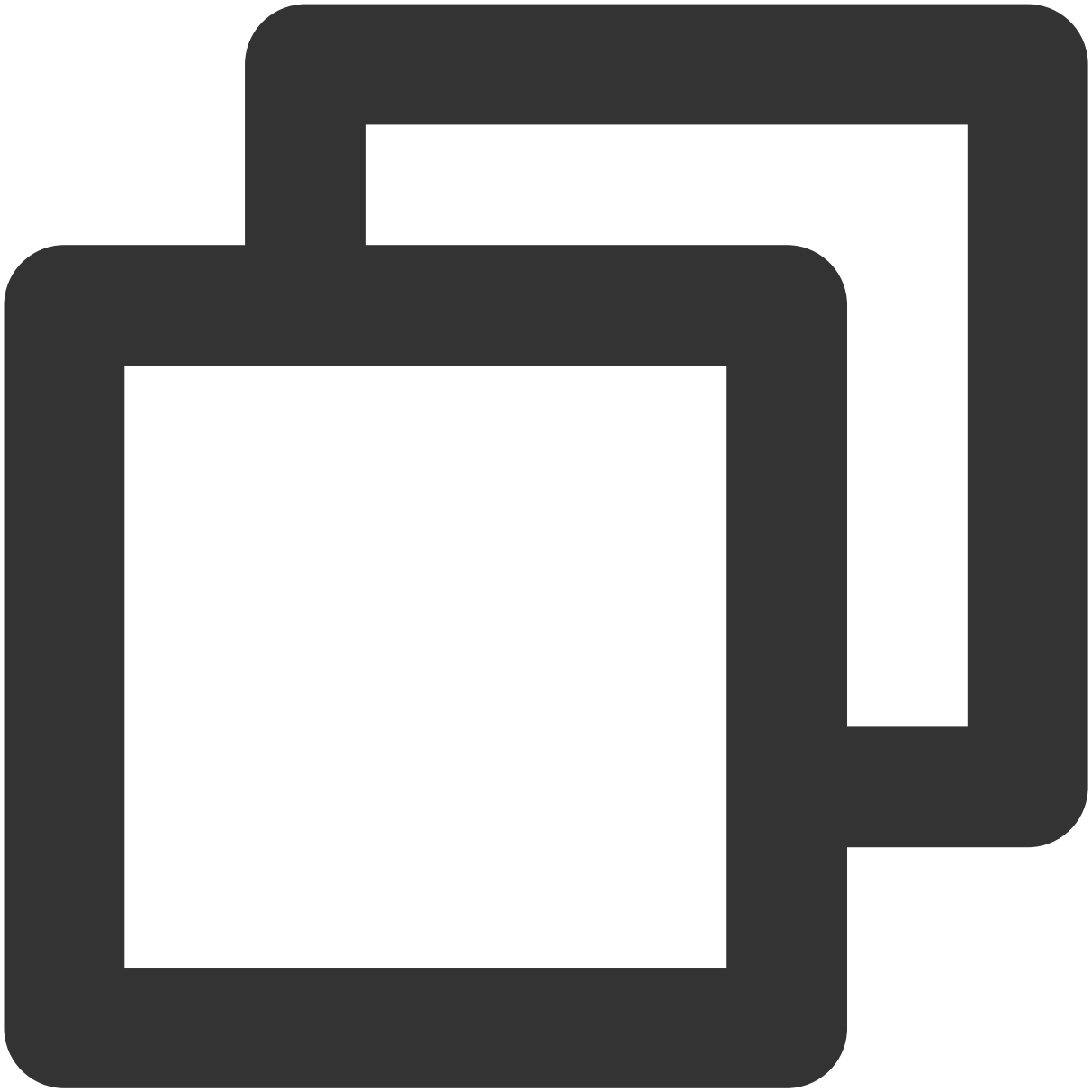
```
https://examples-1258125638.cos.ap-guangzhou.myqcloud.com/preview.png/watermark1?wa
```

방법2: 텍스트 및 이미지 워터마크를 투명 이미지에 출력하고 이를 최종 이미지 워터마크로 사용

1. 400px * 400px의 투명 PNG 이미지를 파일 관리의 지시에 따라 버킷에 업로드합니다.

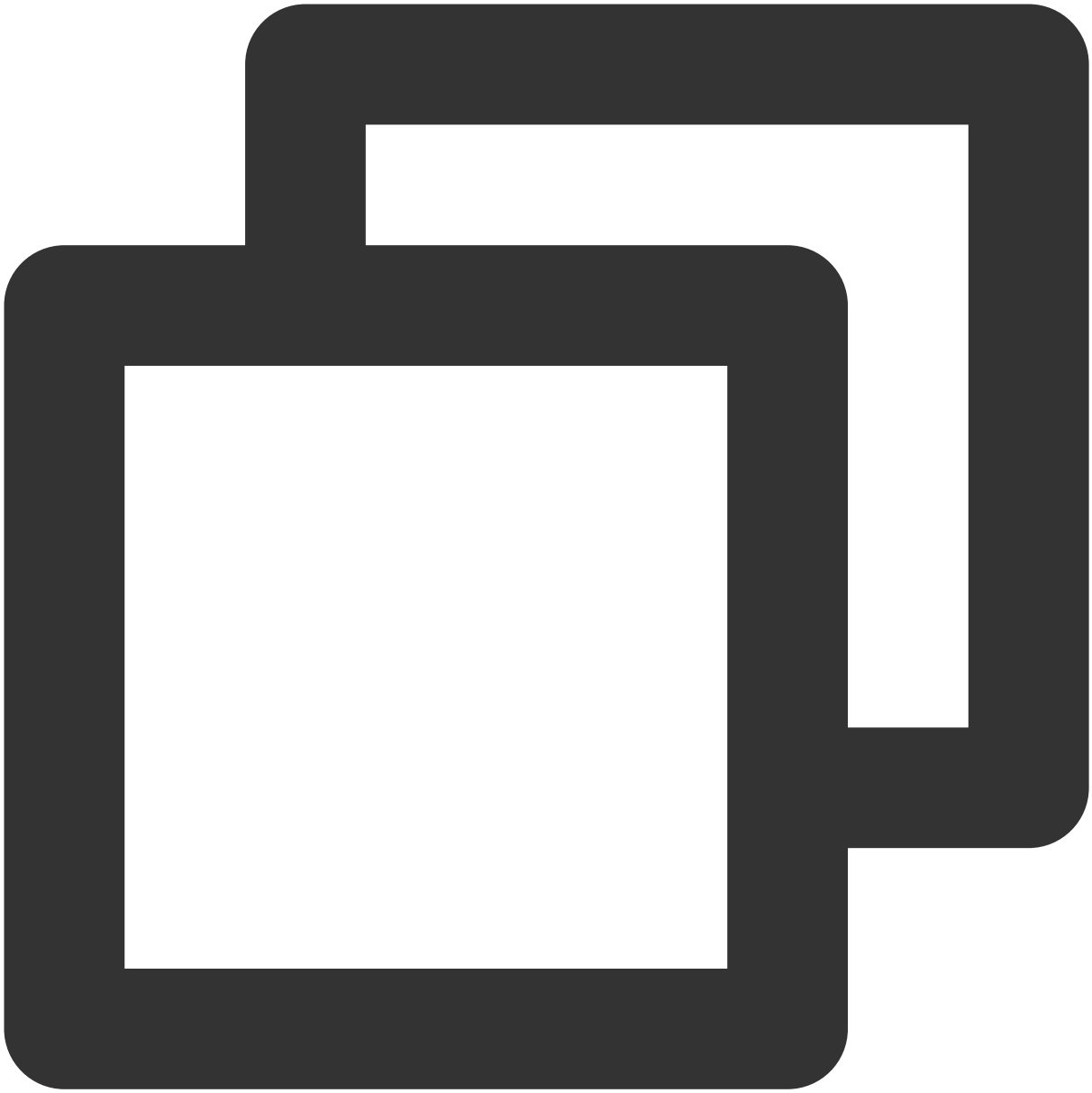
예시: `https://examples-1258125638.cos.ap-guangzhou.myqcloud.com/transparent.png`

2. **방법1의 1 - 3단계**에 설명된 대로 이미지 및 텍스트 워터마크 매개변수를 생성하고 연결합니다.
3. 투명한 PNG 이미지의 다운로드 URL 끝에 연결된 매개변수를 추가합니다.



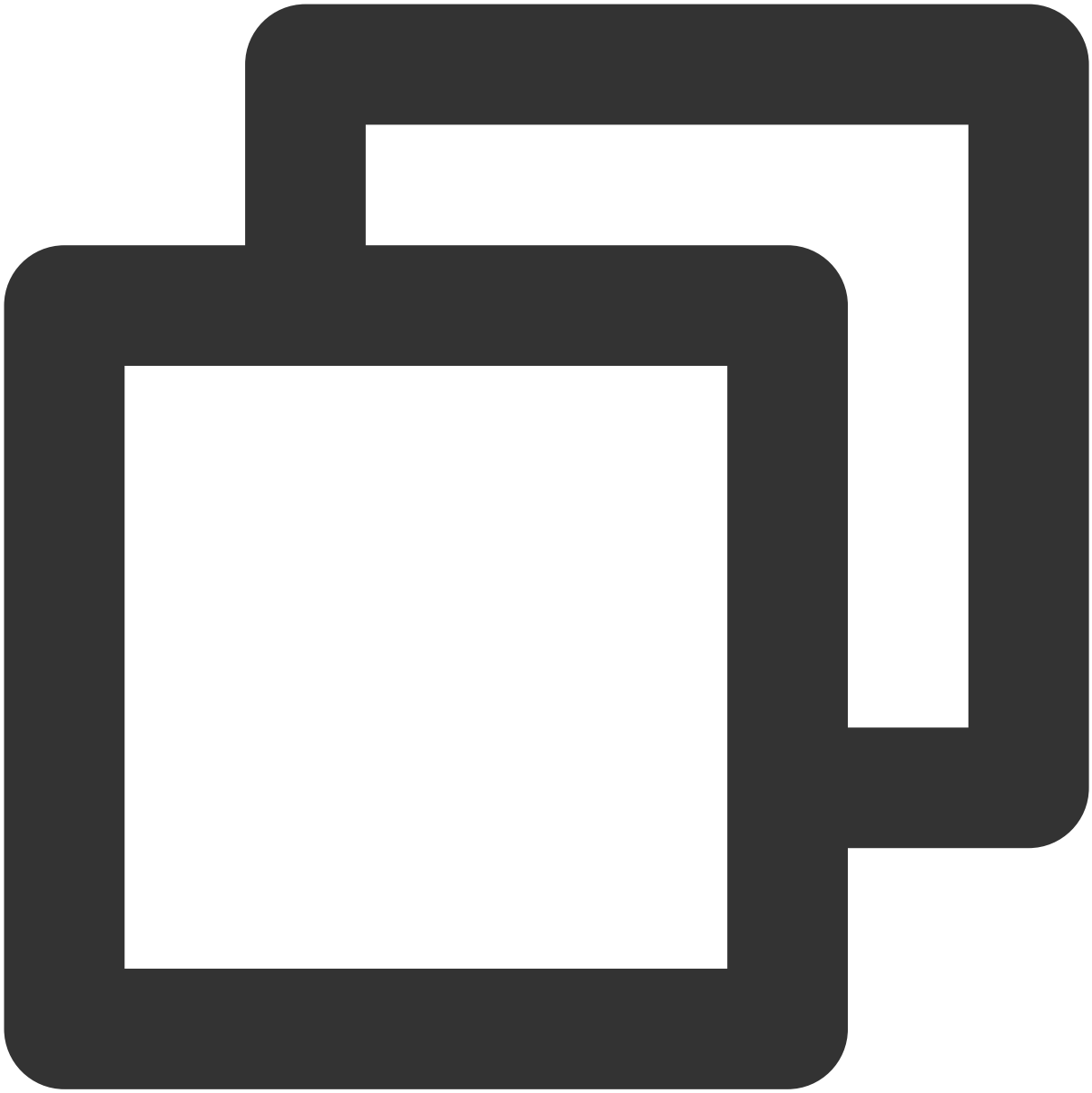
```
https://examples-1258125638.cos.ap-guangzhou.myqcloud.com/transparent.png?watermark
```

4. 투명 이미지를 이미지 워터마크로 사용하고 원본 이미지에 워터마크를 추가합니다.



```
https://examples-1258125638.cos.ap-guangzhou.myqcloud.com/preview.png?watermark/1/i
```

또한 `scatype` 매개변수를 사용하여 이미지 크기에 비례하여 이미지 워터마크의 크기를 조정하고 `batch` 매개변수를 사용하여 이미지 워터마크를 바둑판식으로 배열할 수 있습니다.



```
https://examples-1258125638.cos.ap-guangzhou.myqcloud.com/preview.png?watermark/1/i
```

COS 오디오/비디오 플레이어 사례
하이브리드 워터마크

최종 업데이트 날짜: : 2024-06-24 16:44:04

본문은 클라우드에서 COS 오디오/비디오 파일을 처리하고 클라이언트에서 재생하는 방법을 설명합니다. 이 문서의 예시는 오디오/비디오 처리를 위해 지원되는 프로토콜 및 기능을 다루며 재생 성능을 향상시키기 위해 CI에서 Media Processing Service의 풍부한 오디오/비디오 처리 기능을 기반으로 제품 기능을 사용하는 방법에 대해 더 많은 아이디어를 제공합니다.

지원되는 프로토콜

오디오/비디오 프로토콜	URL 형식	PC 브라우저	모바일 브라우저
MP3	https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp3	지원	지원
MP4	https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4	지원	지원
HLS(M3U8)	https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8	지원	지원
FLV	https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.flv	지원	지원
DASH	https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mpd	지원	지원

주의 :
HLS, FLV, DASH 비디오는 일부 브라우저 환경에서 재생하려면 Media Source Extensions 에 종속해야 합니다.

지원되는 기능

기능	TCPlayer	DPlayer	Videojs Player
MP4 비디오 재생	상세 보기	상세 보기	상세 보기

HLS 비디오 재생	상세 보기	상세 보기	상세 보기
FLV 비디오 재생	상세 보기	상세 보기	상세 보기
DASH 비디오 재생	상세 보기	상세 보기	상세 보기
PM3U8(개인 M3U8) 비디오 재생	상세 보기	상세 보기	상세 보기
썸네일 구성	상세 보기	상세 보기	상세 보기
표준 HLS 암호화 구성	상세 보기	상세 보기	상세 보기
해상도 전환	상세 보기	상세 보기	-
동적 워터마크 구성	상세 보기	-	-
왼쪽 상단 LOGO 구성	-	상세 보기	-
진행률 표시줄 미리보기 이미지 구성	상세 보기	-	-
자막 구성	상세 보기	-	-
다국어 구성	상세 보기	-	-
롤 이미지 광고 구성	상세 보기	-	-

설명：
플레이어는 일반 브라우저와 호환되며 최적의 재생 구성표를 사용할 플랫폼을 자동으로 식별할 수 있습니다. 예를 들어 Chrome과 같은 최신 브라우저에서 비디오 재생을 위해 HTML5 기술을 사용하고 모바일 브라우저에서 HTML5 기술 또는 브라우저 커널 기능을 직접 사용하는 것이 좋습니다.

사용 가이드

TCPlayer 사용하여 COS에서 비디오 재생
DPlayer 사용하여 COS에서 비디오 재생
VideojsPlayer 사용하여 COS 비디오 재생

TCPlayer 사용하여 COS에서 비디오 재생

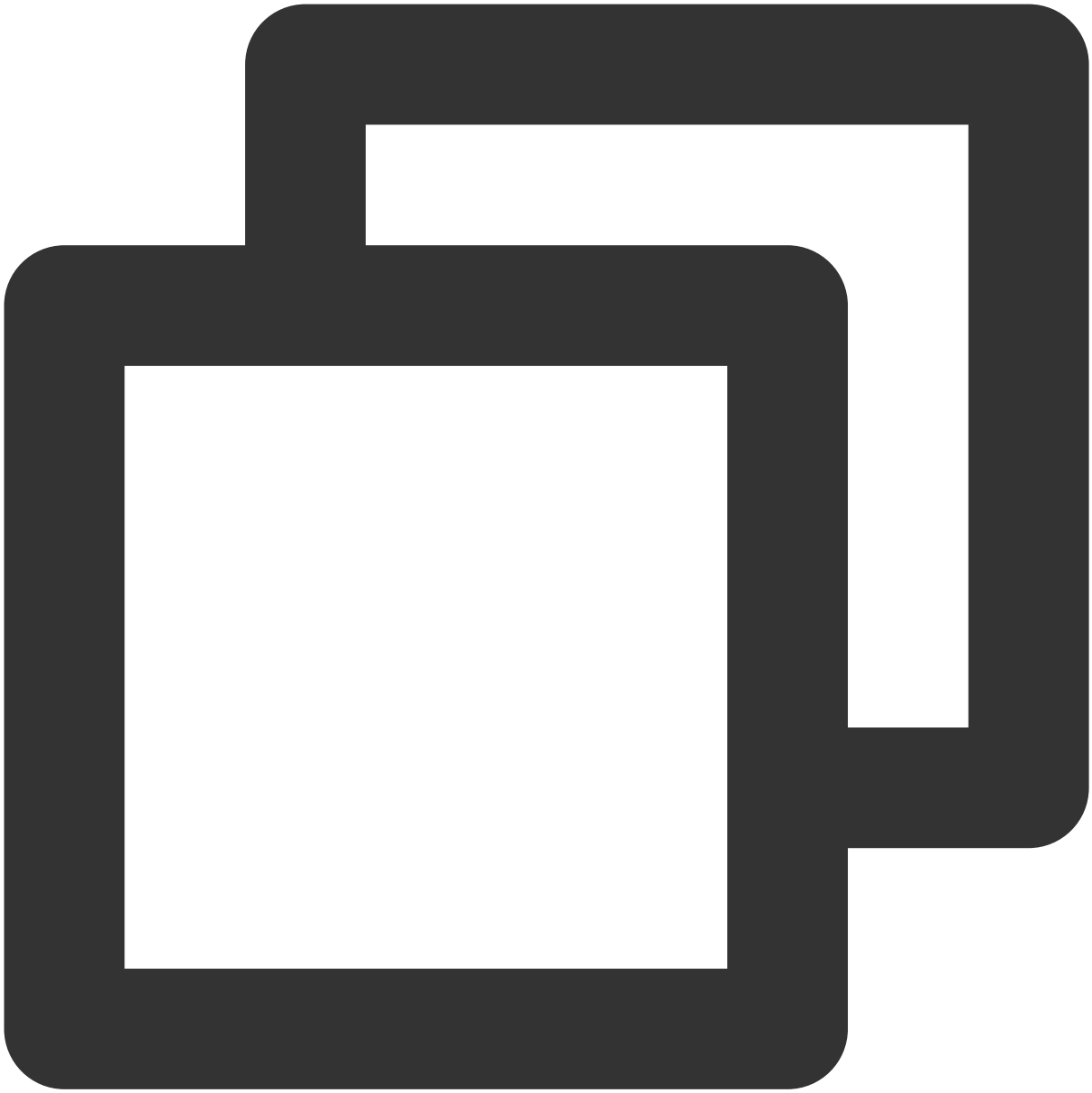
최종 업데이트 날짜: : 2024-06-24 16:44:04

소개

본문은 TCToolkit SDK에 통합된 TCPlayer와 CI(Cloud Infinite)의 풍부한 오디오/비디오 기능을 사용하여 웹 브라우저에서 COS에 저장된 비디오 파일을 재생하는 방법을 설명합니다.

통합 가이드

1단계: 플레이어 양식 및 스크립트 파일을 페이지로 가져오기



```
<!--플레이어 양식 파일-->
<link href="https://web.sdk.qcloud.com/player/tcplayer/release/v4.2.1/tcplayer.min.
<!--플레이어 스크립트 파일-->
<script src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.0/tcplayer.v4.
```

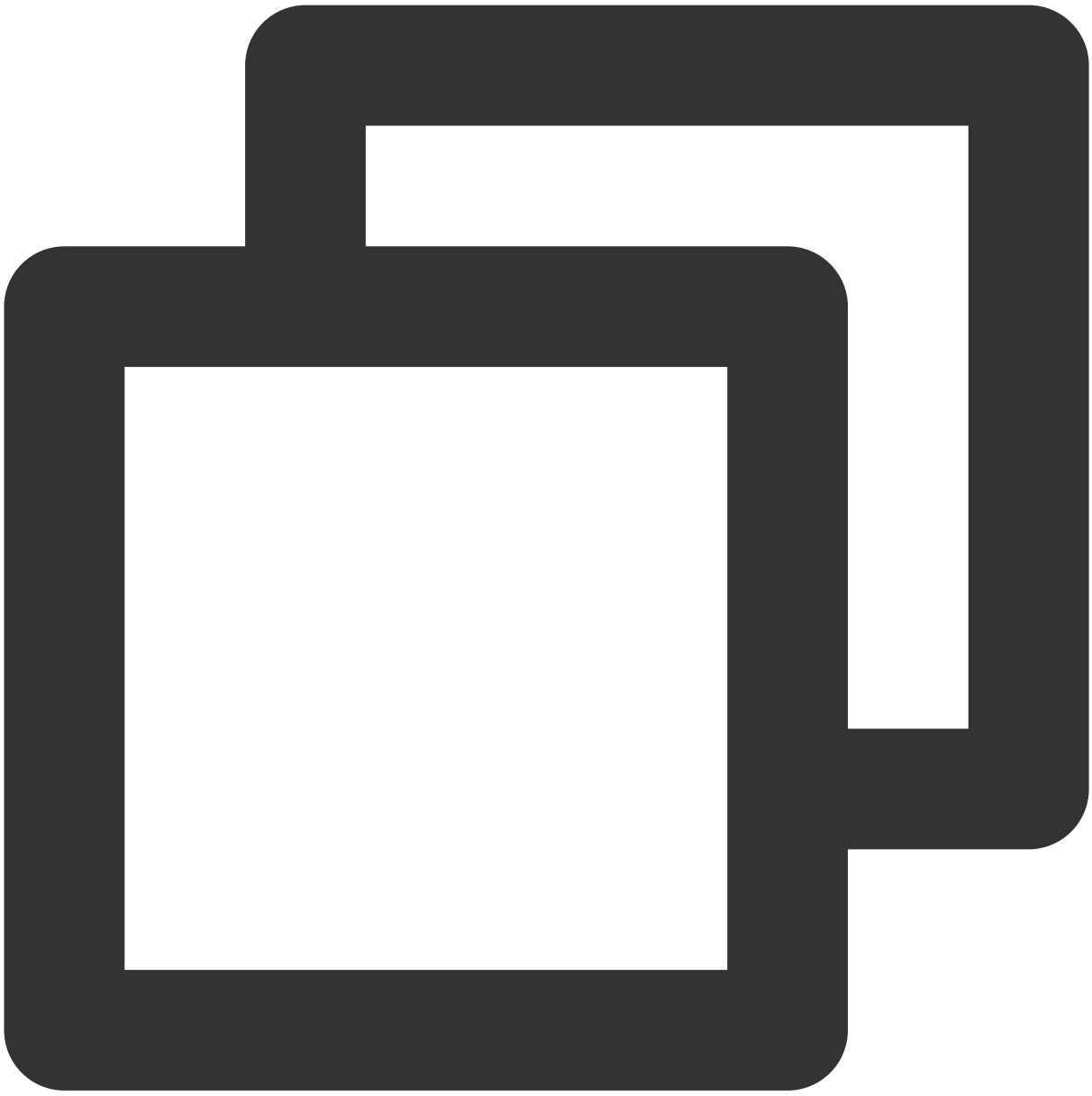
설명：

플레이어 SDK를 사용할 때 위의 정적 리소스를 직접 배포하는 것이 좋습니다. 플레이어 리소스를 다운로드하려면 여기 클릭하십시오.

압축 해제 후 생성된 폴더를 배포합니다. 폴더의 디렉터리를 조정하지 마십시오. 그렇지 않으면 리소스 가져오기 예외가 발생할 수 있습니다.

2단계: 플레이어 컨테이너 노드 설정

플레이어를 표시할 페이지에 플레이어 컨테이너를 추가합니다. 예를 들어 index.html에 다음 코드를 추가합니다(컨테이너 ID와 너비 및 높이를 사용자 지정할 수 있음).



```
<video id="player-container-id" width="414" height="270" preload="auto" playsinline
</video>
```

설명：

플레이어 컨테이너는 `<video>` 태그여야 합니다.

예시의 `player-container-id` 는 사용자 정의할 수 있는 플레이어 컨테이너의 ID입니다.

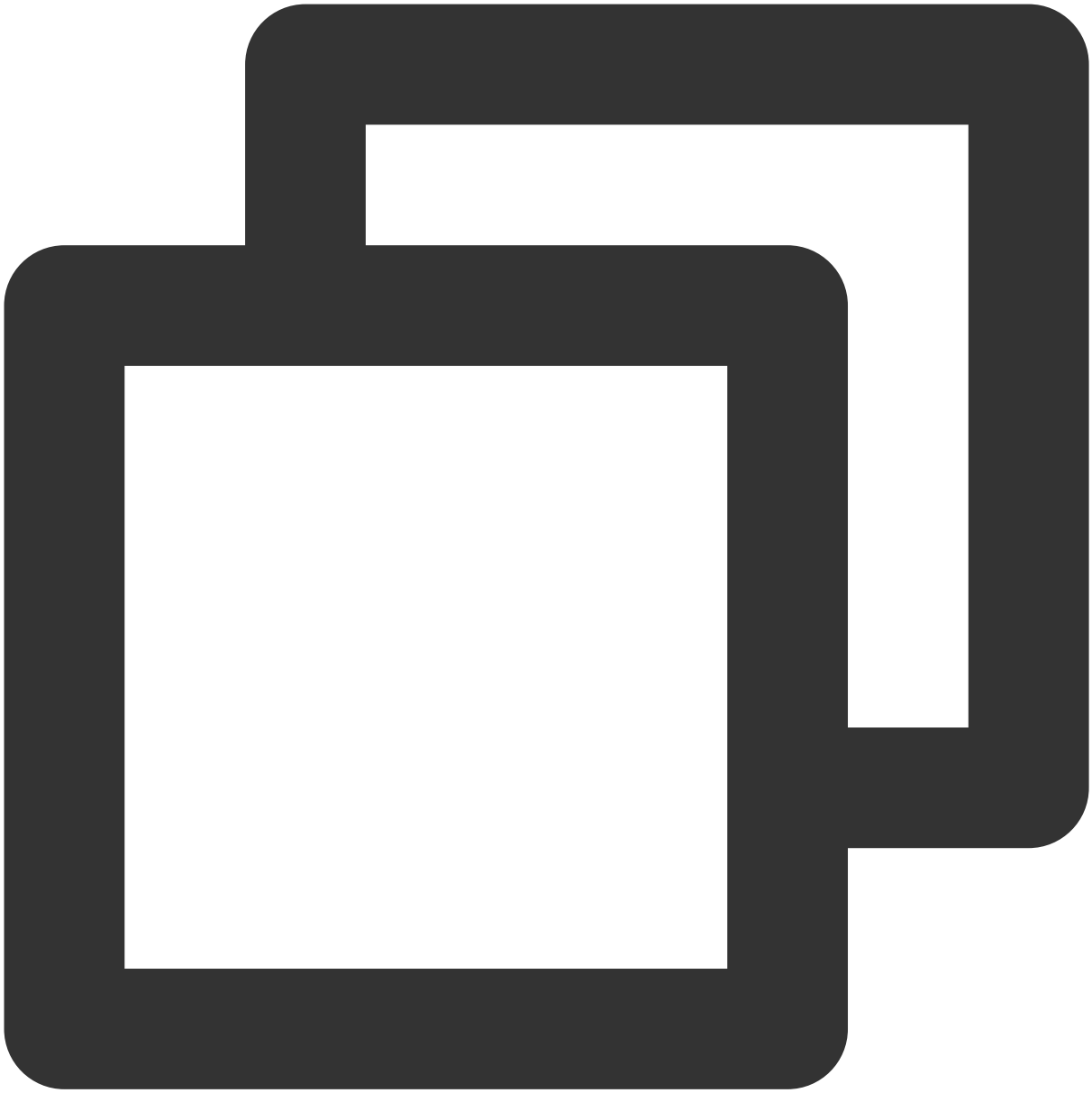
CSS를 통해 플레이어 컨테이너 영역의 크기를 설정하는 것이 좋습니다. CSS는 속성보다 유연하고 전체 화면에 맞추기 및 컨테이너 적응과 같은 효과를 얻을 수 있습니다.
예시의 `preload` 속성은 페이지가 로딩된 후 비디오 로딩 여부를 지정하며 일반적으로 재생을 더 빨리 시작하기 위해 `auto` 로 설정됩니다. 다른 옵션에는 `meta` (페이지가 로딩된 후에만 메타데이터를 로딩함) 및 `none` (페이지가 로딩된 후 비디오를 로딩하지 않음)이 있습니다. 시스템 제한으로 인해 동영상은 모바일 장치에 자동으로 로딩되지 않습니다.
`playsinline` 및 `webkit-playsinline` 속성은 표준 모바일 브라우저가 비디오 재생을 가로채지 않을 때 인라인 재생을 구현하는 데 사용됩니다. 여기에서는 참고용일 뿐이며 필요에 따라 사용할 수 있습니다.
`x5-playsinline` 속성을 설정하면 TBS 커널에서 X5 UI 플레이어가 사용됩니다.

3단계: 비디오 파일 객체 주소 가져오기

1. 버킷 생성.
2. 동영상 파일 객체 업로드.
3. `https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.<비디오 형식>` 형식의 비디오 파일 객체 주소를 가져옵니다.

설명：
Cross-Origin 액세스가 포함된 경우 CORS 설정에서 설명한 대로 버킷에 대한 CORS를 설정해야 합니다.
버킷 권한이 비공개 읽기/쓰기인 경우 객체 주소에 서명이 있어야 합니다. 자세한 내용은 Request Signature를 참고하십시오.

4단계: 플레이어 초기화 및 COS 동영상 파일 객체 URL 전달



```
var player = TCPlayer("player-container-id", {}); // player-container-id는 플레이어 컨
player.src("https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4"); // COS
```

기능 가이드

다양한 형식의 비디오 파일 재생

1. COS 버킷에 있는 비디오 파일의 객체 주소를 가져옵니다.

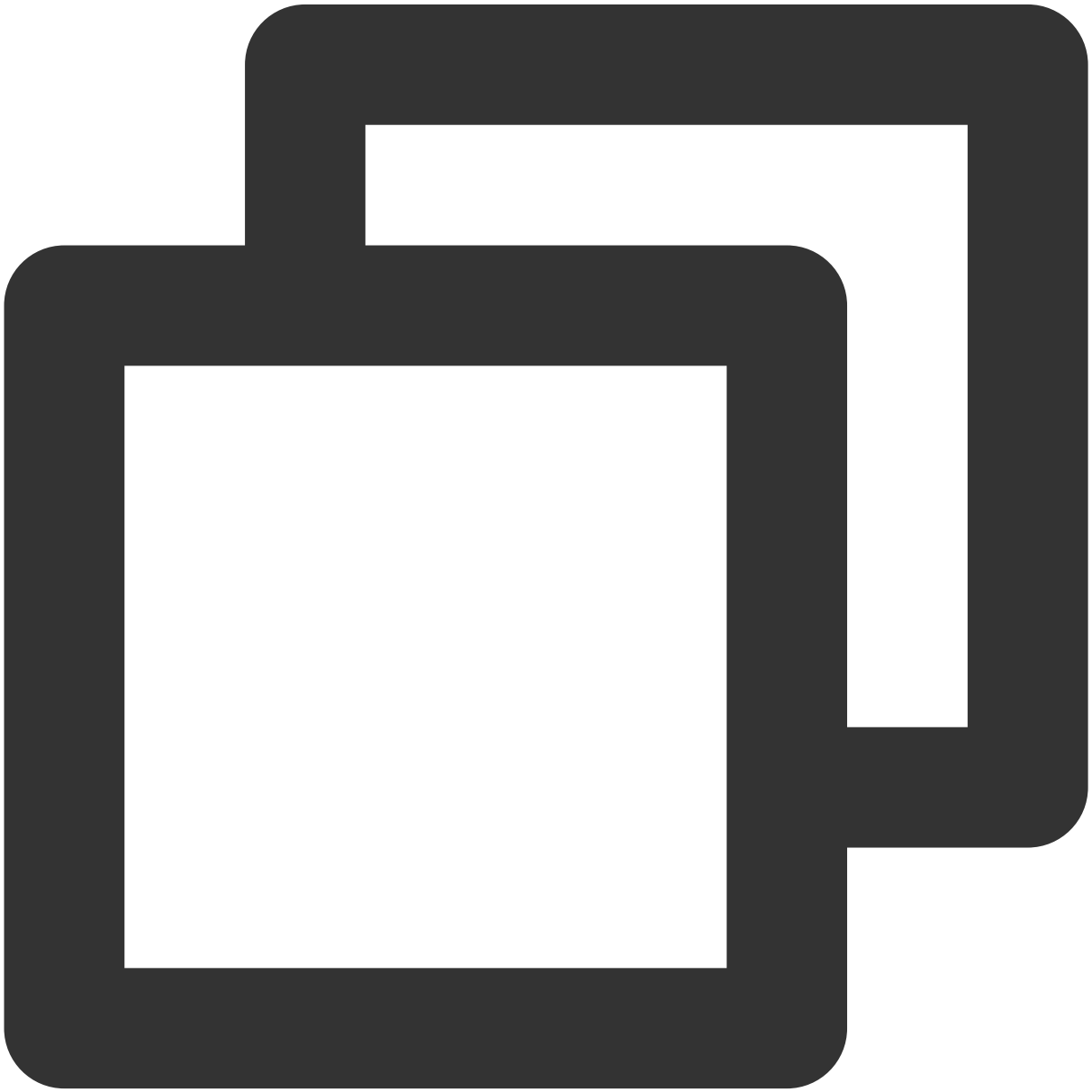
설명：

트랜스코딩되지 않은 비디오는 재생 중에 호환성 문제가 발생할 수 있으므로 재생을 위해 비디오를 트랜스코딩하는 것이 좋습니다. CI의 Audio/Video Transcoding 기능을 사용하여 다양한 형식의 비디오 파일을 얻을 수 있습니다.

2. 다른 비디오 형식의 경우 다른 브라우저와의 호환성을 보장하기 위해 해당 종속성을 가져와야 합니다.

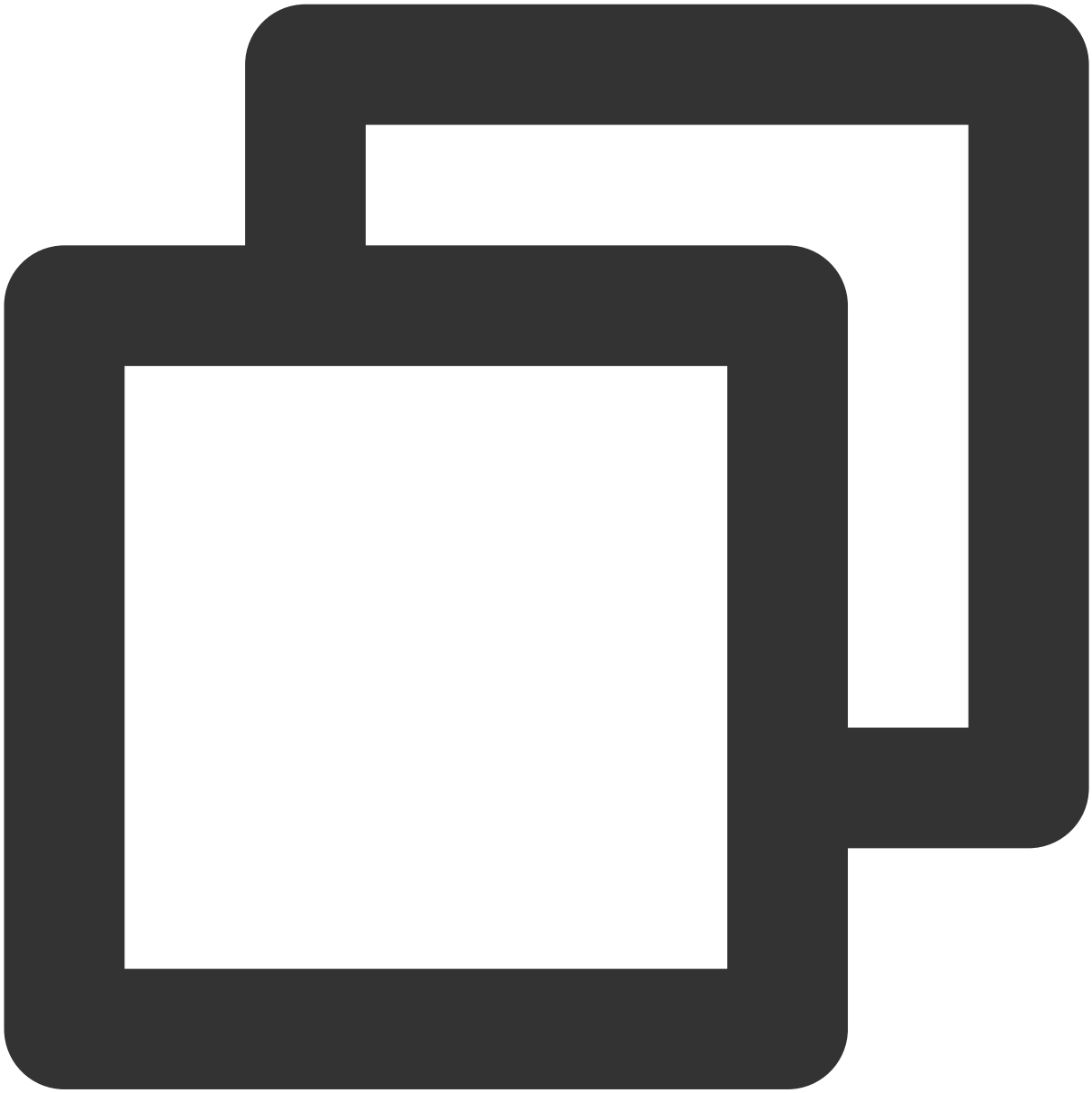
MP4: 다른 종속성을 가져올 필요가 없습니다.

HLS: Chrome 및 Firefox와 같은 최신 브라우저에서 HTML5를 통해 HLS 동영상을 재생하려면 tcplayer.min.js를 가져오기 전에 hls.min.js를 가져와야 합니다.



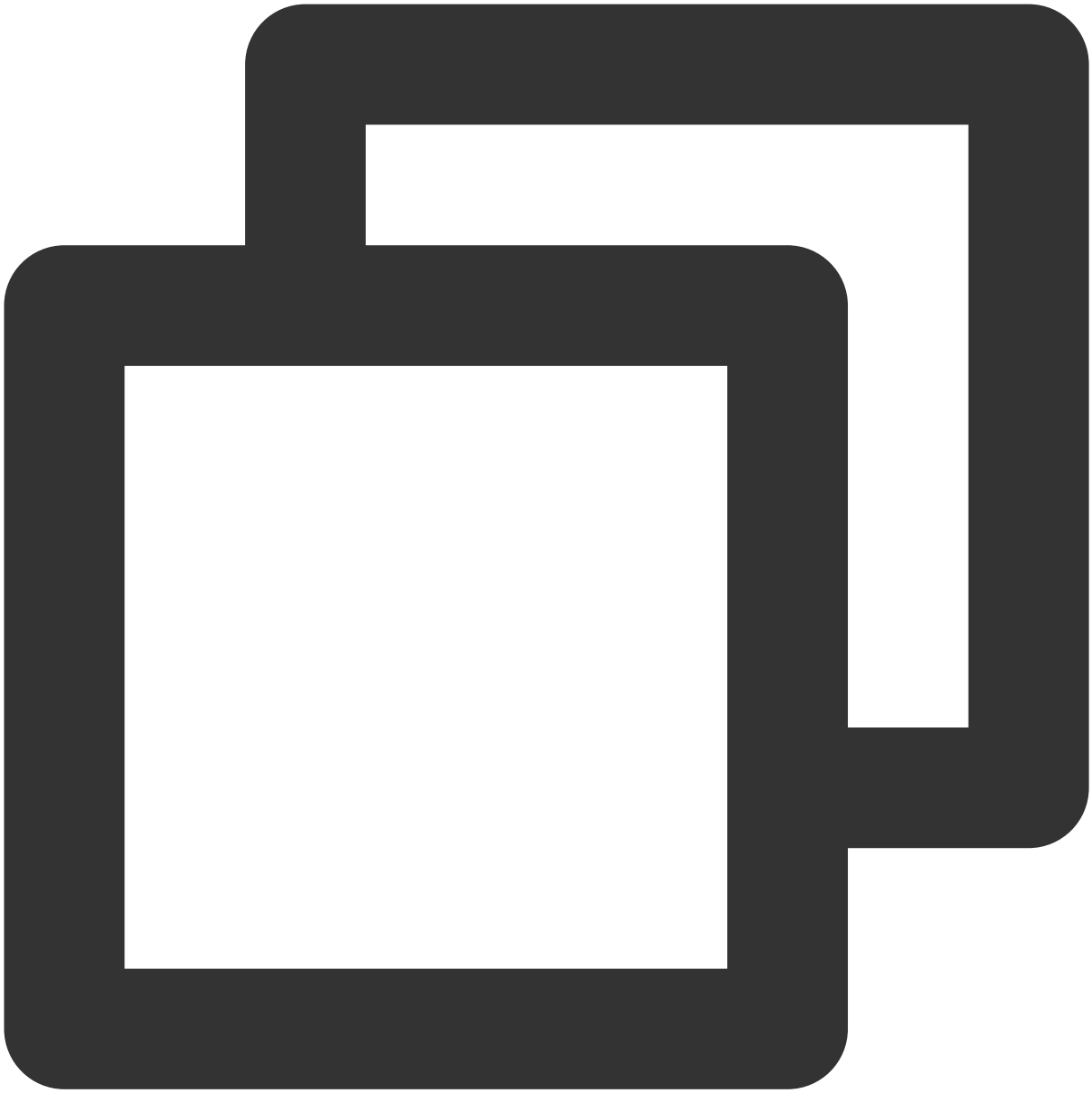
```
<script src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.2.1/libs/hls.m
```

FLV: Chrome 및 Firefox와 같은 최신 브라우저에서 H5를 통해 FLV 비디오를 재생하려면 tcplayer.min.js를 가져오기 전에 flv.min.js를 가져와야 합니다.



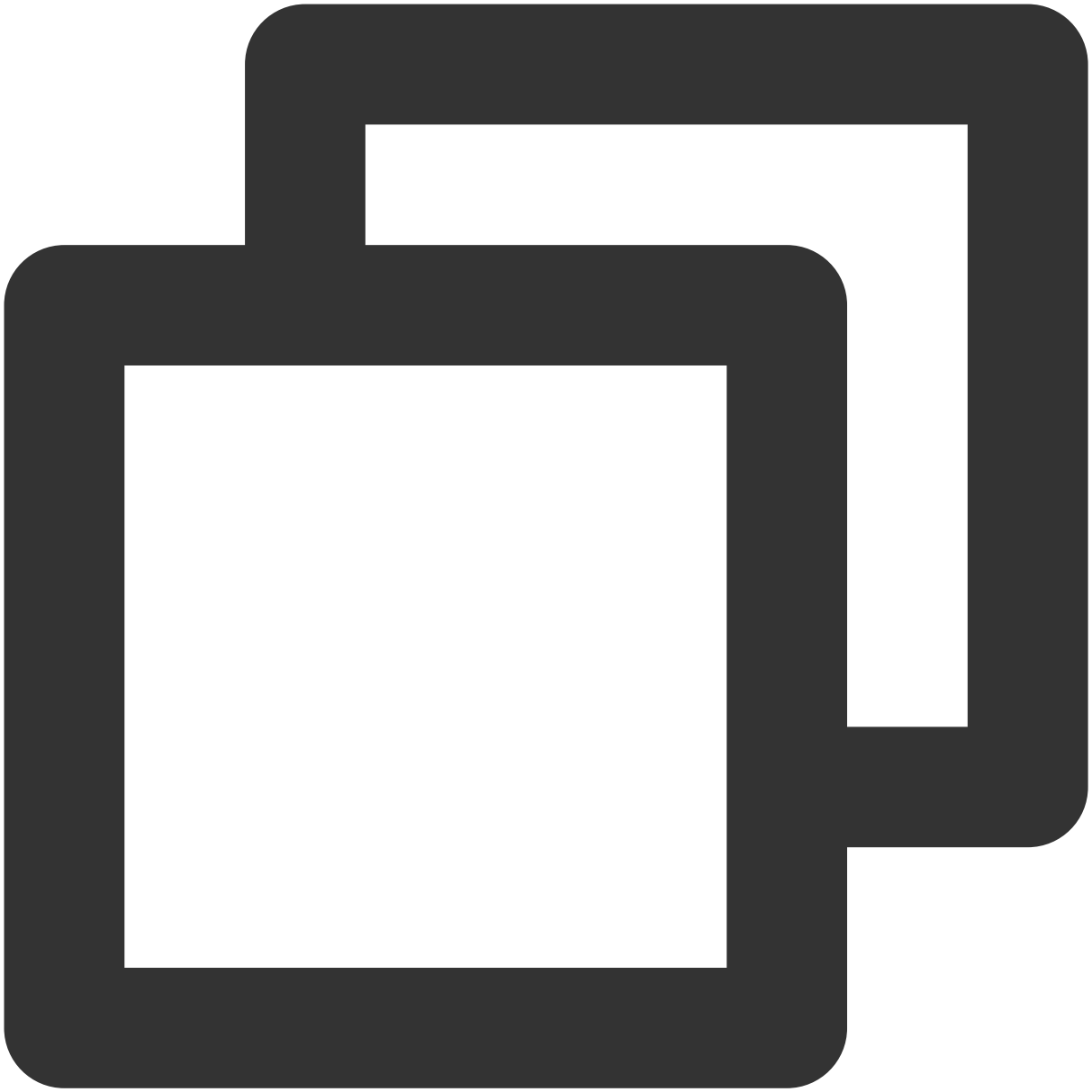
```
<script src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.2/libs/flv.m
```

DASH: dash.all.min.js 파일을 가져와야 합니다.



```
<script src="https://cos-video-1258344699.cos.ap-guangzhou.myqcloud.com/lib/dash.
```

3. 플레이어를 초기화하고 객체 주소를 전달합니다.



```
var player = TCPlayer("player-container-id", {}); // player-container-id는 플레이어 컨
player.src("https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4"); // COS
```

코드 샘플 가져오기:

MP4 재생을 위한 샘플 코드

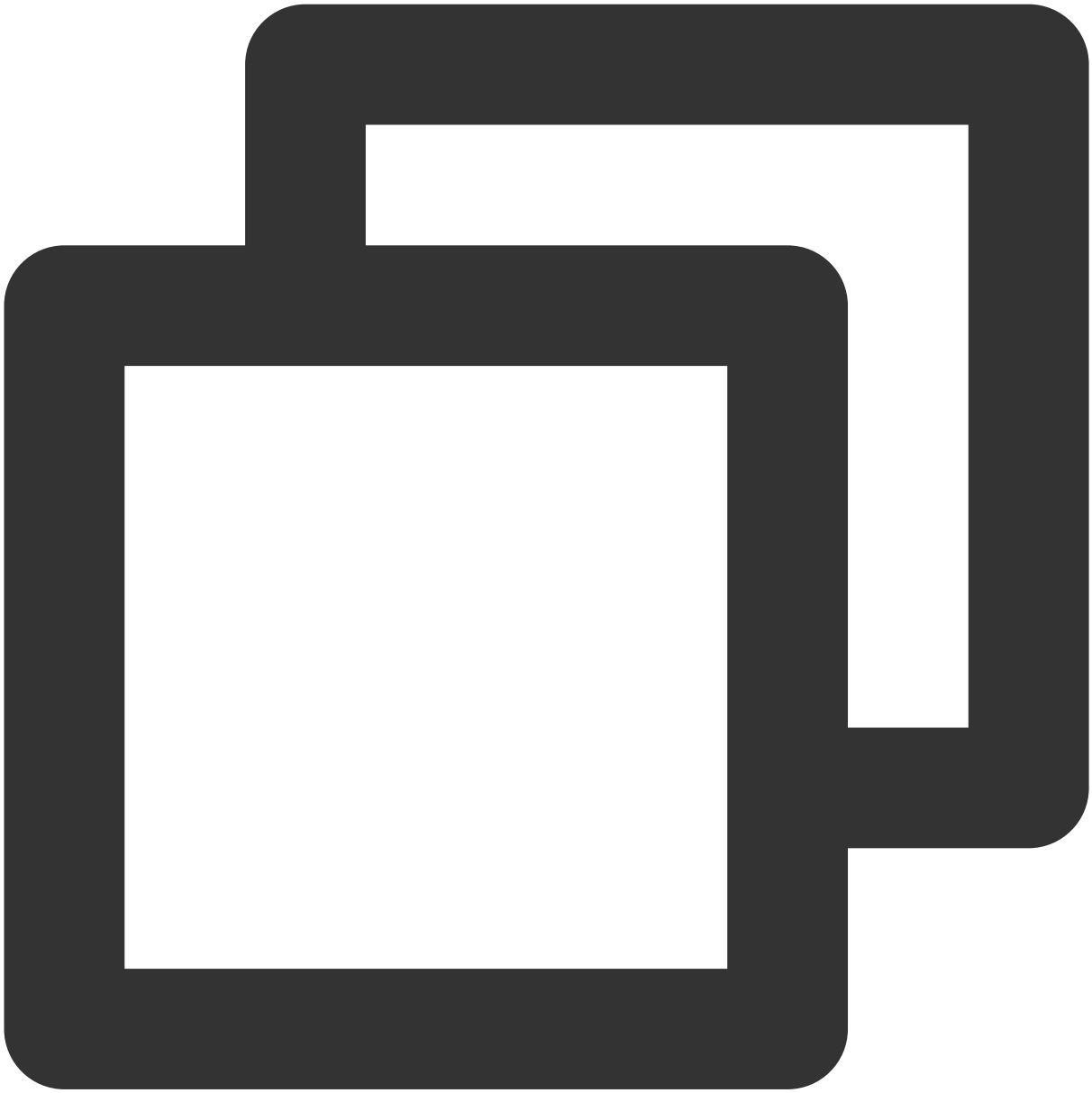
FLV 재생을 위한 샘플 코드

HLS 재생을 위한 샘플 코드

DASH 재생 샘플 코드

PM3U8 비디오 재생

PM3U8은 개인 M3U8 비디오 파일을 나타냅니다. COS는 개인 M3U8 TS 리소스를 가져오기 위한 다운로드 권한 API를 제공합니다. 자세한 내용은 Private M3U8 API를 참고하십시오.



```
var player = TCPlayer("player-container-id", {
    poster: "https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8?ci-proce
});
```

코드 샘플 가져오기:

PM3U8 재생 샘플 코드

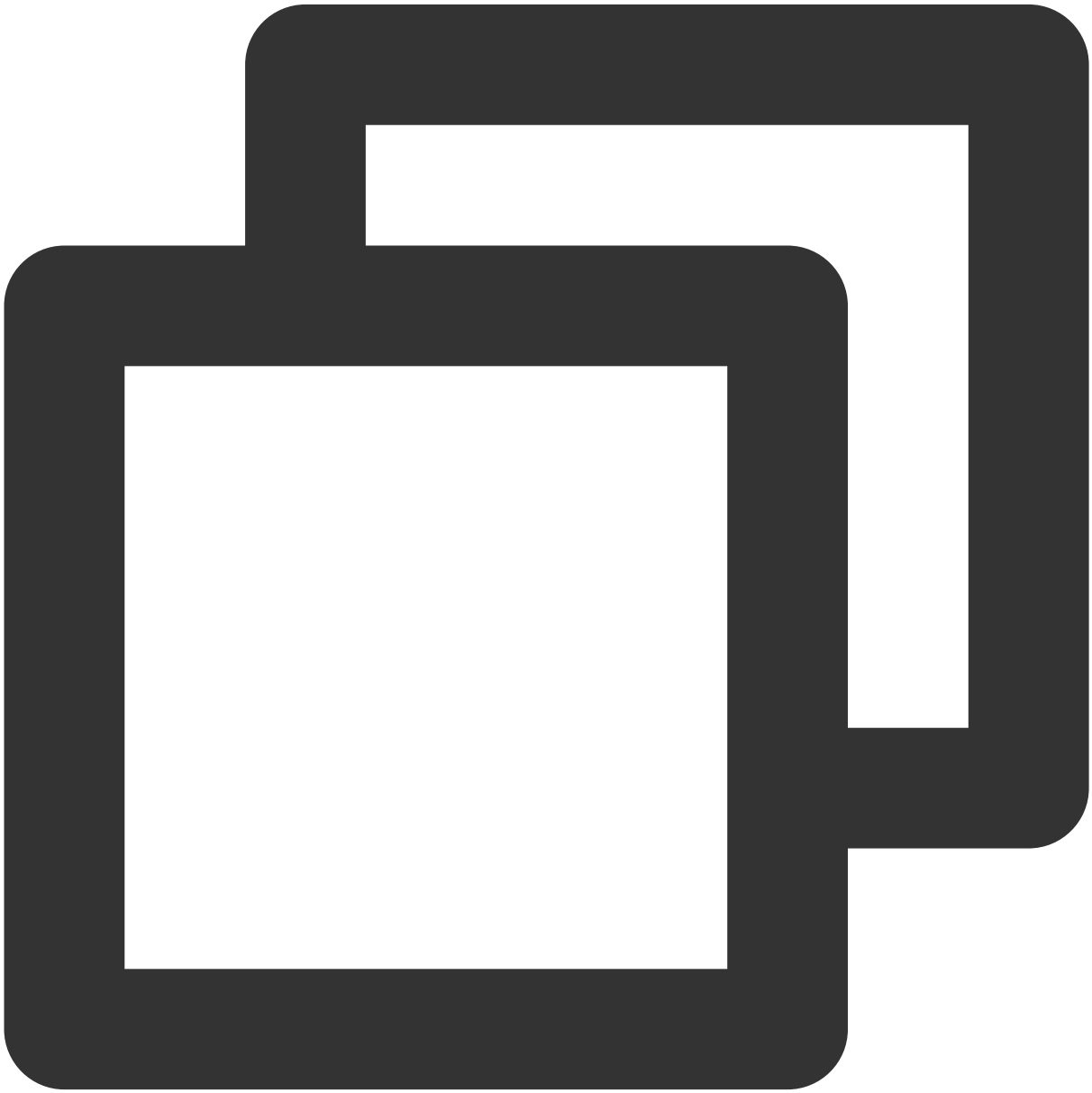
썸네일 설정

1. COS 버킷에 있는 썸네일의 객체 주소를 가져옵니다.

주의 :

CI의 Intelligent Thumbnail 기능은 최적의 프레임을 추출하여 썸네일을 생성하여 비디오 콘텐츠를 더 매력적으로 만들 수 있습니다.

2. 썸네일을 설정합니다.



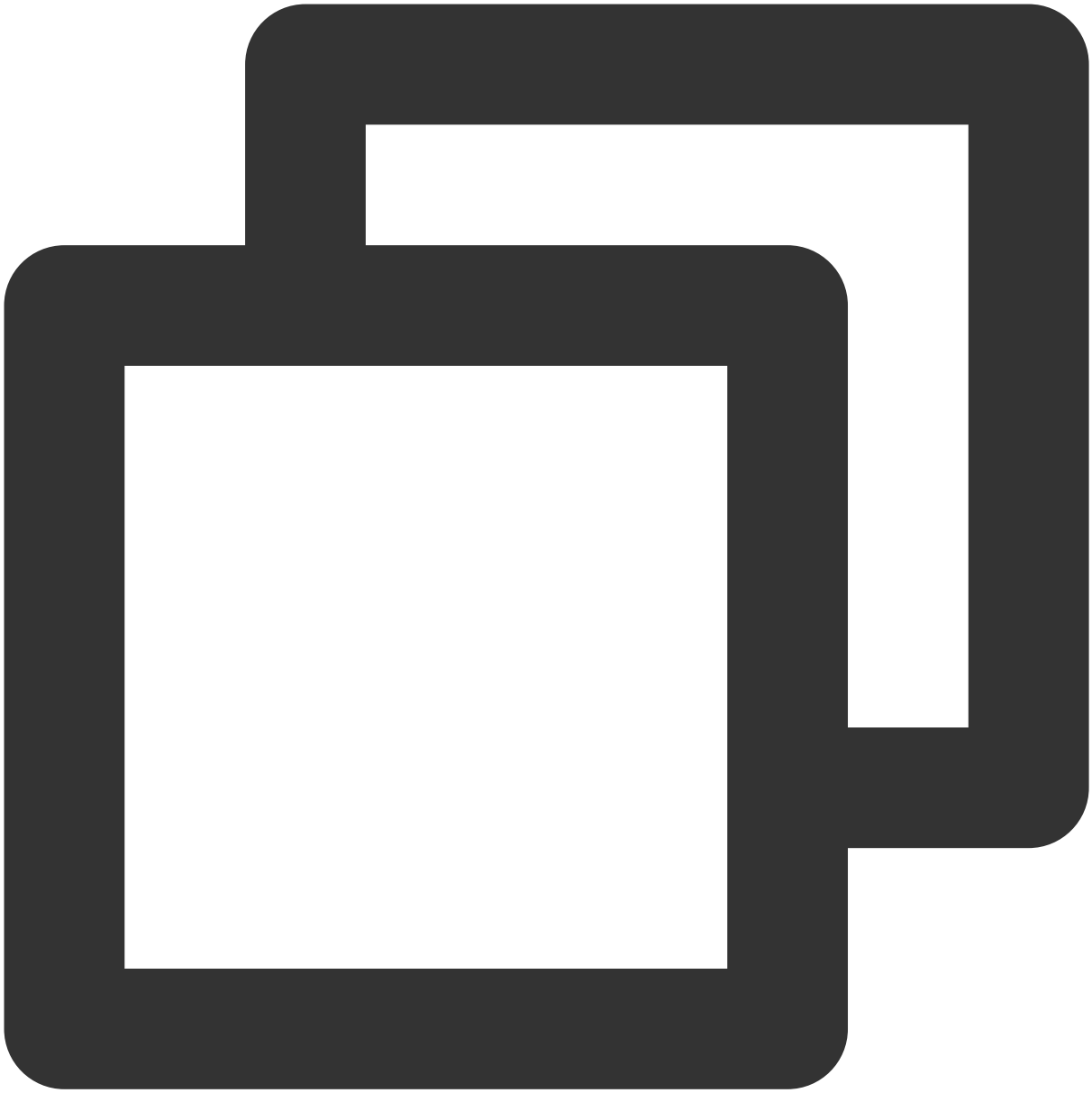
```
var player = TCPlayer("player-container-id", {
```

```
  poster: "https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.png"
 });
```

코드 샘플 가져오기:
썸네일 구성 샘플 코드

HLS 암호화 비디오 재생

CI는 동영상 콘텐츠의 보안을 확보하고 동영상의 무단 다운로드 및 배포를 방지하기 위해 개인이 읽을 수 있는 파일보다 안전한 HLS 동영상 콘텐츠를 암호화하는 기능을 제공합니다. 암호화된 비디오는 재생 액세스 권한이 없는 사용자에게 배포할 수 없습니다. 다운로드하더라도 여전히 암호화되어 악의적으로 재배포할 수 없습니다. 이렇게 하면 비디오 저작권이 침해되는 것을 방지할 수 있습니다.
작업 순서는 다음과 같습니다.
1. HLS 암호화 비디오 재생의 지침에 따라 암호화된 비디오를 생성합니다.
2. 플레이어를 초기화하고 비디오 객체 주소를 전달합니다.



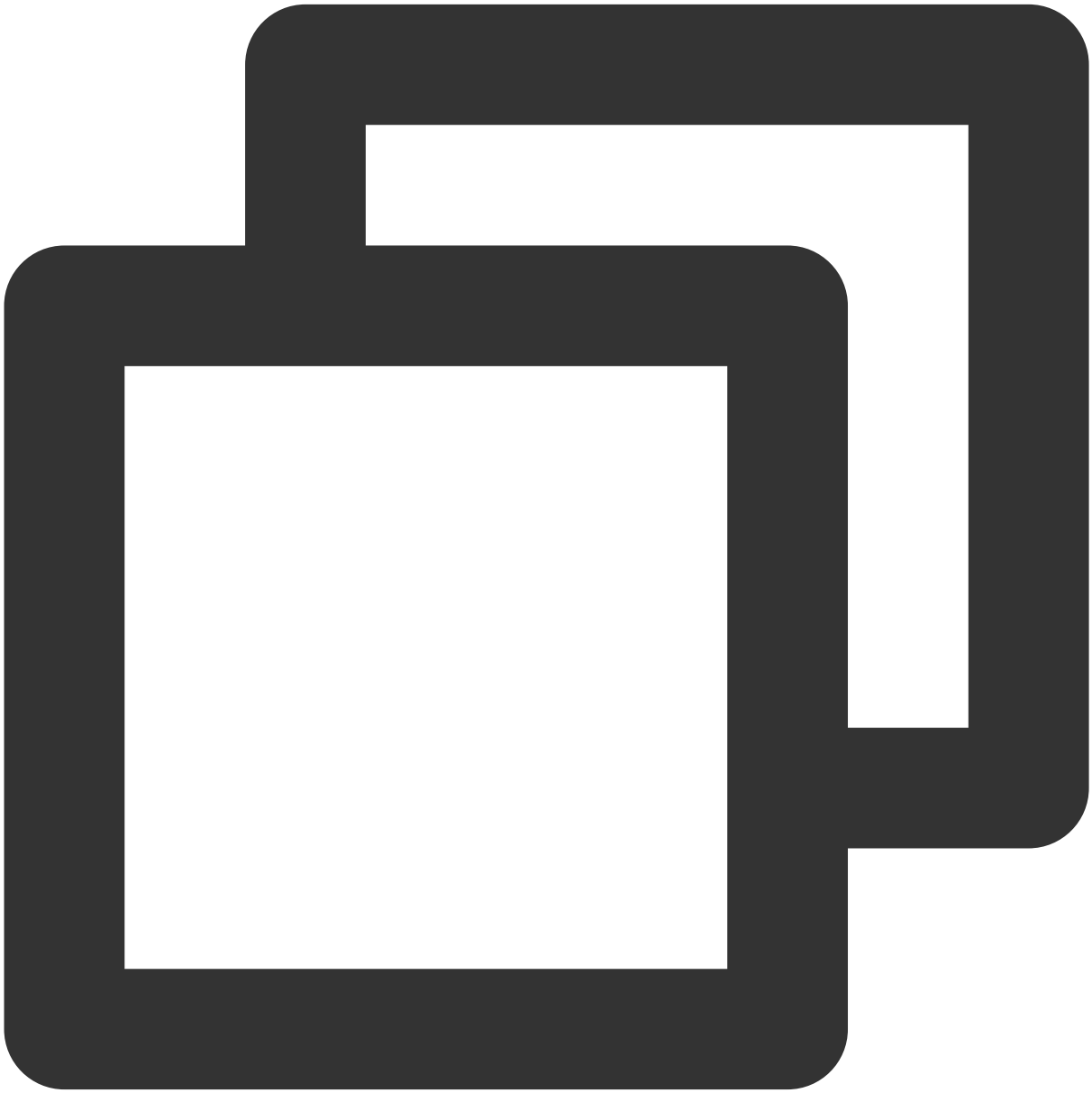
```
var player = TCPlayer("player-container-id", {}); // player-container-id는 플레이어 컨
player.src("https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8"); // hls
```

코드 샘플 가져오기:

HLS 암호화 재생 샘플 코드

해상도 전환

CI의 Adaptive Bitrate Streaming 기능은 비디오를 트랜스코딩하고 출력을 위해 어댑티브 비트스트림으로 리먹스할 수 있으므로 다양한 네트워크 조건에서 비디오 콘텐츠를 빠르게 배포할 수 있습니다. 플레이어는 현재 대역폭을 기반으로 비디오를 재생하는 데 가장 적절한 비트레이트를 동적으로 선택할 수 있습니다.
작업 순서는 다음과 같습니다.
1. CI의 어댑티브 비트레이트 스트리밍 기능을 사용하여 다중 비트레이트 어댑티브 HLS 또는 DASH 대상 파일을 생성합니다.
2. 플레이어를 초기화하고 비디오 객체 주소를 전달합니다.

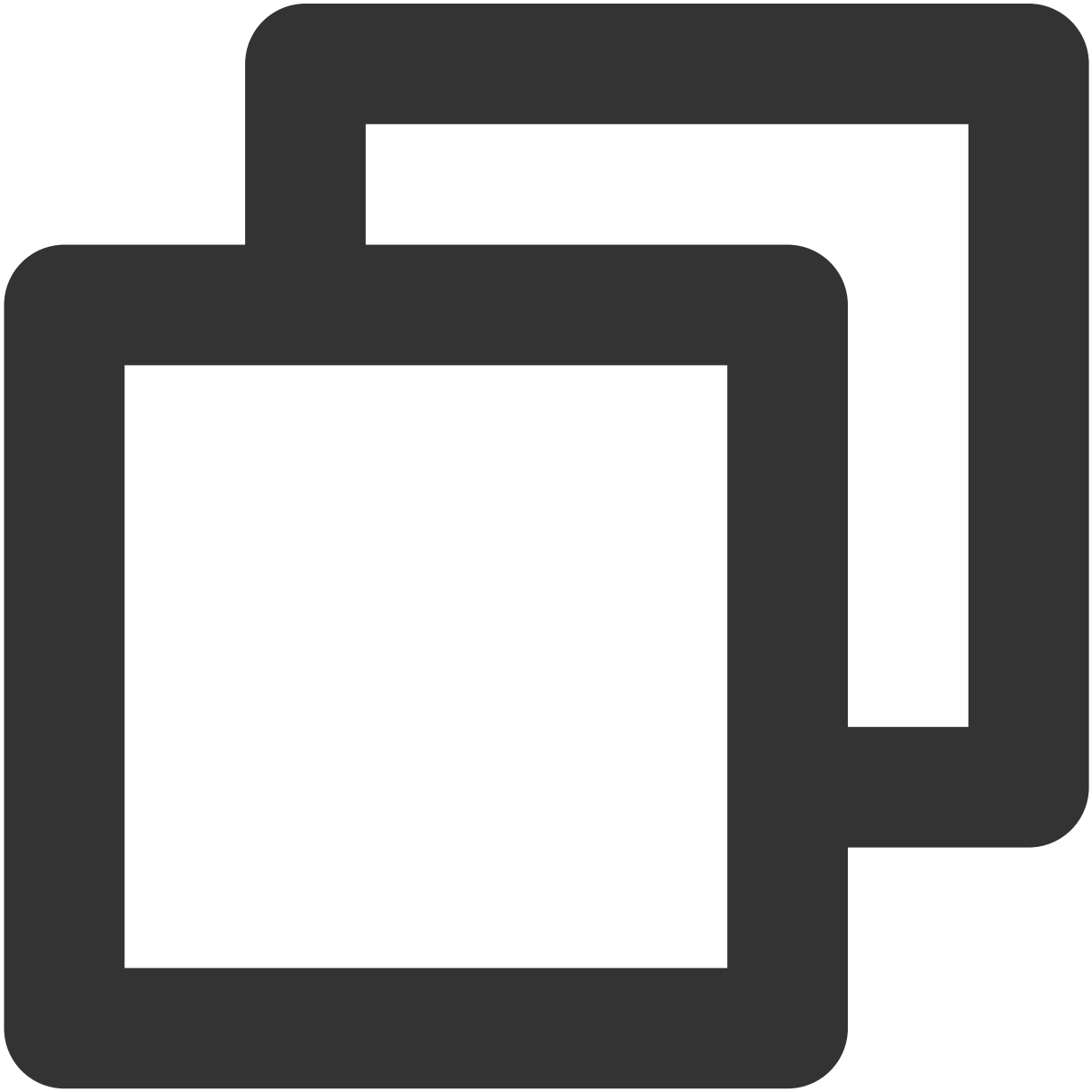


```
var player = TCPlayer("player-container-id", {}); // player-container-id는 플레이어 컨
player.src("https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8"); // 다중
```

코드 샘플 가져오기:
해상도 전환을 위한 샘플 코드

동적 워터마크 설정

플레이어는 비디오에 위치와 속도가 변화하는 동적 워터마크 추가를 지원합니다. 동적 워터마크 기능을 사용할 때 플레이어 객체의 참고가 전역 환경에 노출되어서는 안 됩니다. 그렇지 않으면 동적 워터마크를 쉽게 제거할 수 있습니다. CI를 사용하면 클라우드의 비디오에 동적 워터마크를 추가할 수도 있습니다. 자세한 내용은 워터마크 템플릿 API를 참고하십시오.



```
var player = TCPlayer("player-container-id", {
    plugins:{
      DynamicWatermark: {
        speed: 0.2, // 속도
        content: "Tencent Cloud CI", // 텍스트
        opacity: 0.7 // 불투명도
      }
    }
  });
```

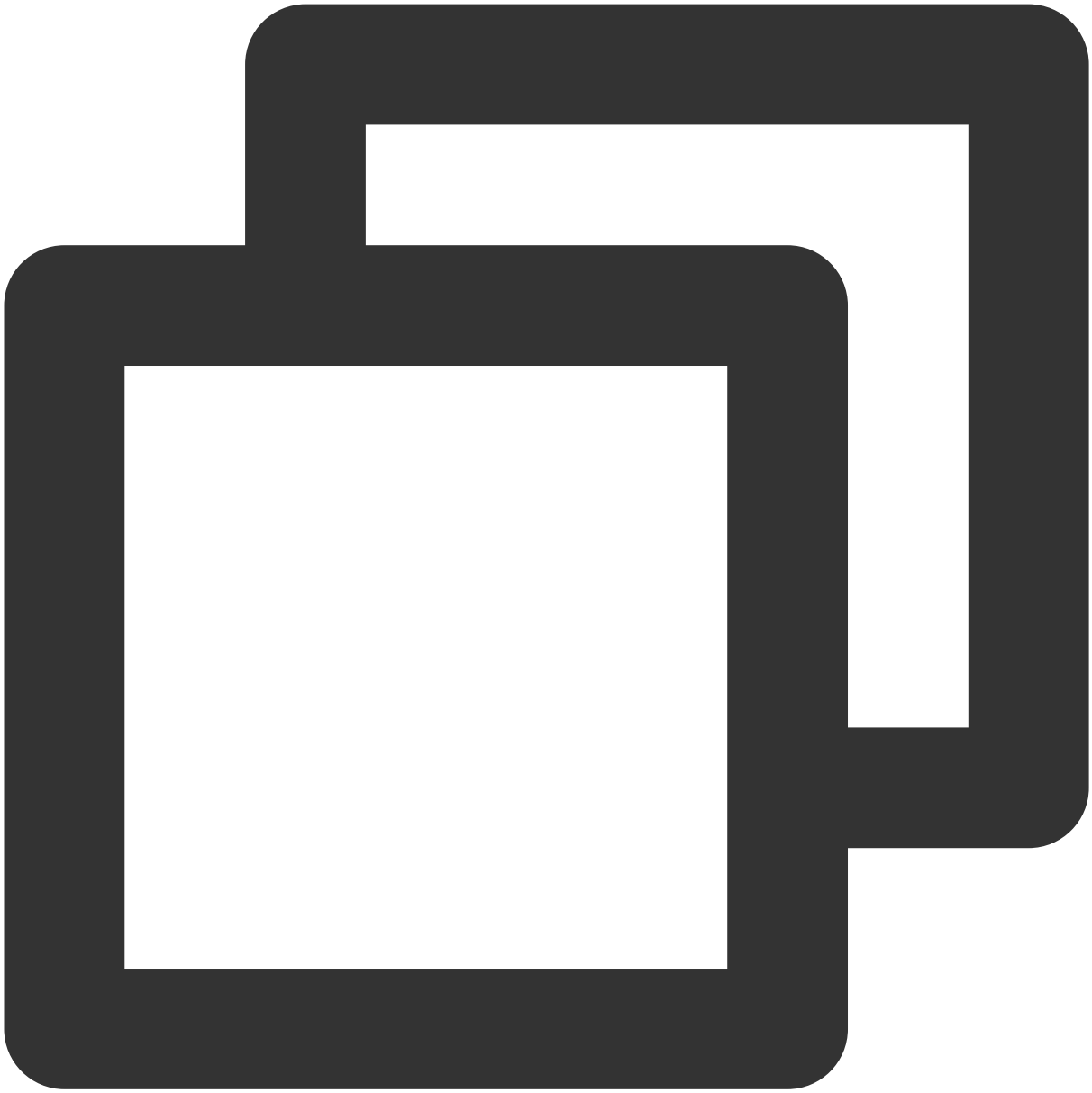
코드 샘플 가져오기:

동적 워터마크 설정을 위한 샘플 코드

롤 이미지 광고 설정

작업 순서는 다음과 같습니다.

1. 광고 썸네일과 링크를 준비합니다.
2. 플레이어를 초기화하고, 광고 썸네일과 링크를 설정하고, 광고 노드를 설정합니다.



```
var PosterImage = TCPlayer.getComponent('PosterImage');
PosterImage.prototype.handleClick = function () {
 window.open('https://cloud.tencent.com/product/ci'); // 광고 링크 설정
};

var player = TCPlayer('player-container-id', {
 poster: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx..png', // 광고 미
});
player.src('https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4');

var adTextNode = document.createElement('span');
```

```
adTextNode.className = 'ad-text-node';
adTextNode.innerHTML = '광고';

var adCloseIconNode = document.createElement('i');
adCloseIconNode.className = 'ad-close-icon-node';
adCloseIconNode.onclick = function (e) {
  e.stopPropagation();
  player.posterImage.hide();
};

player.posterImage.el_.appendChild(adTextNode);
player.posterImage.el_.appendChild(adCloseIconNode);
```

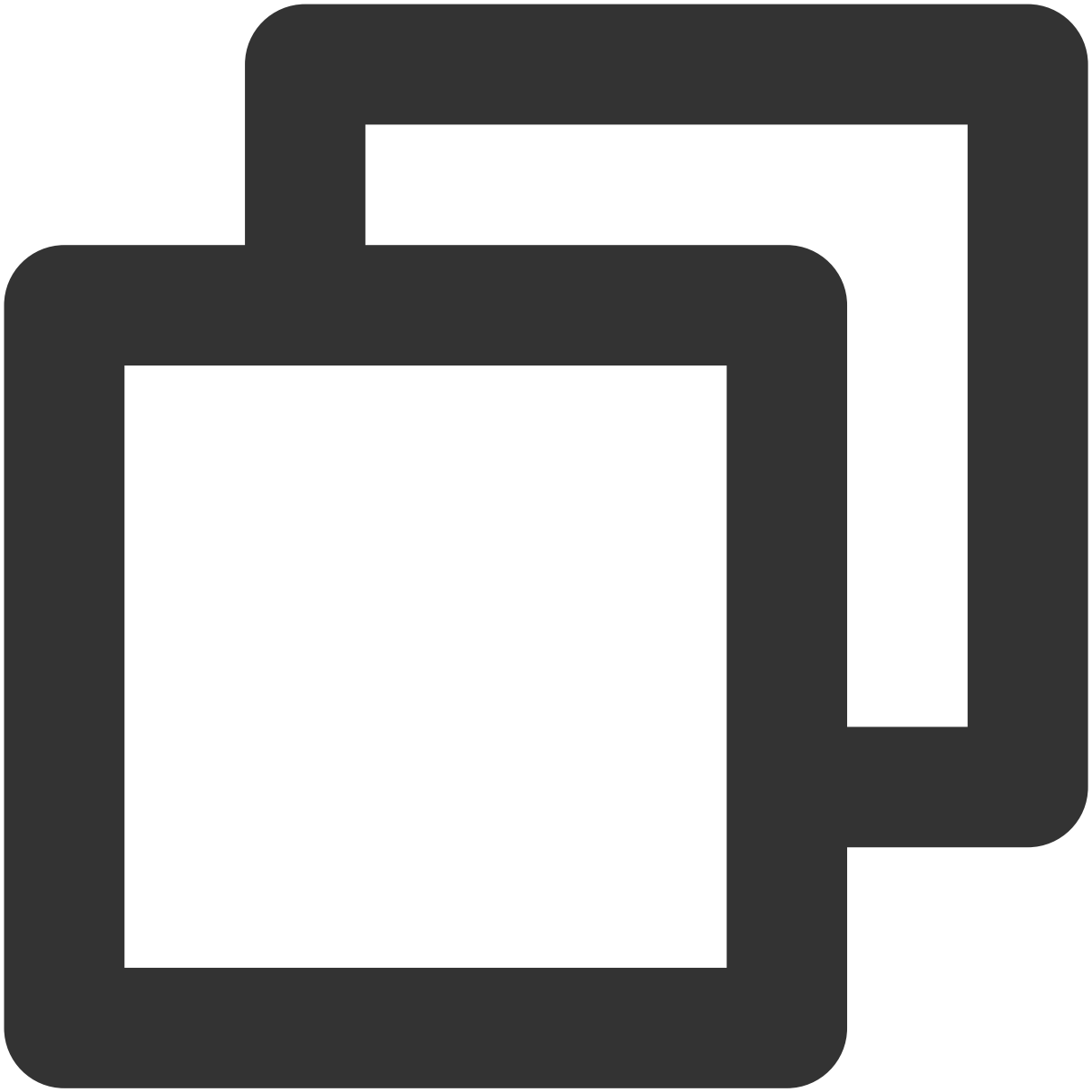
코드 샘플 가져오기:

롤 이미지 광고 구성을 위한 샘플 코드

비디오 진행 썸네일 설정(이미지 스프라이트)

작업 순서는 다음과 같습니다.

1. CI의 Video Frame Capturing 기능을 사용하여 이미지 스프라이트를 생성합니다.
2. 1단계에서 생성된 이미지 스프라이트 및 VTT(이미지 스프라이트 위치 설명 파일)의 객체 주소를 가져옵니다.
3. 플레이어를 초기화하고 비디오 및 VTT 파일 주소를 설정합니다.



```
var player = TCPlayer('player-container-id', {
  plugins: {
 VttThumbnail: {
   vttUrl: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.vtt' // VTT 파
 },
  },
});
player.src('https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4');
```

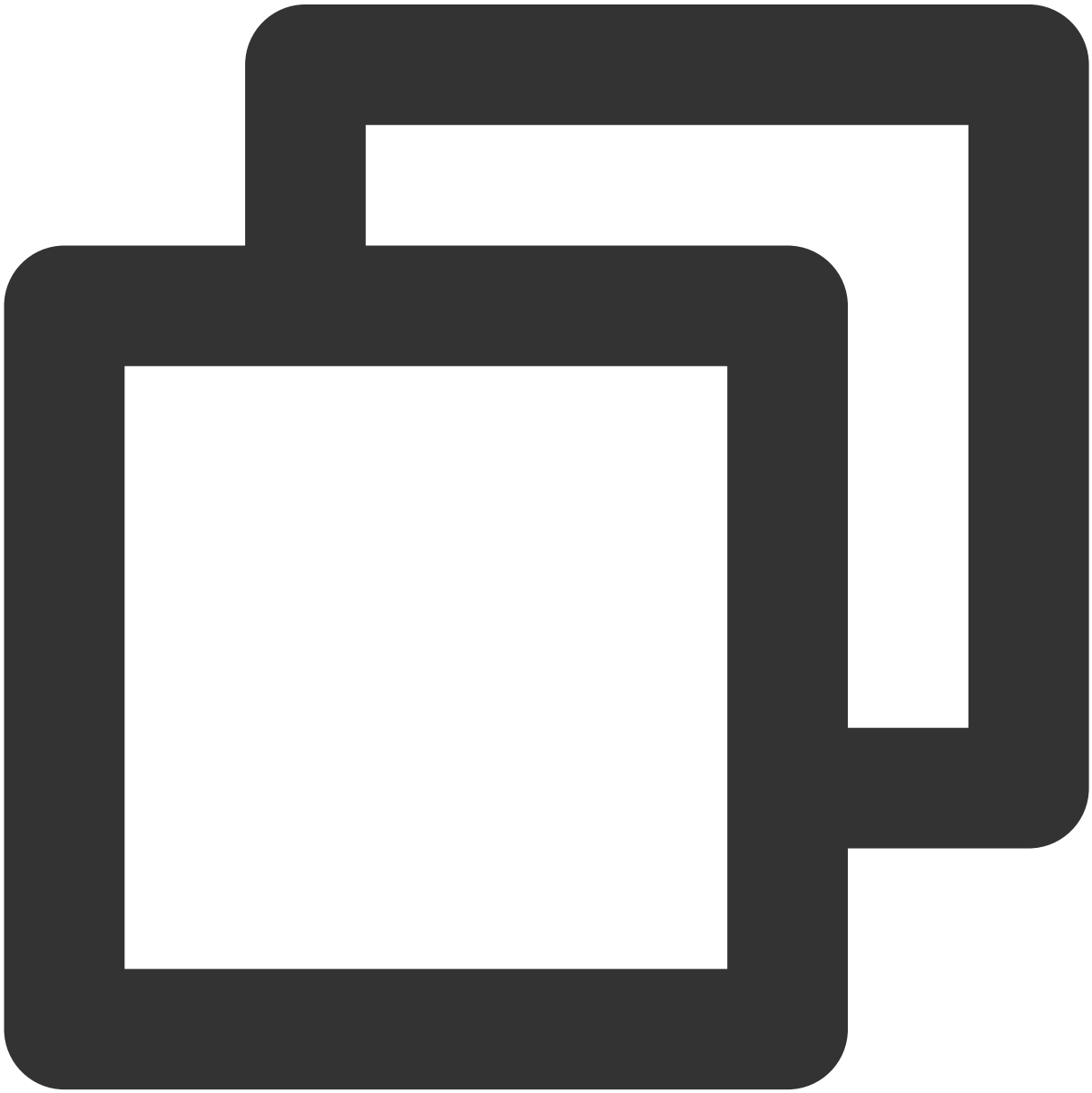
코드 샘플 가져오기:

이미지 스프라이트 구성을 위한 샘플 코드

동영상 자막 설정

작업 순서는 다음과 같습니다.

1. CI의 음성 인식 기능으로 자막 파일을 생성합니다.
2. 1단계에서 생성된 SRT 파일의 객체 주소를 가져옵니다.
3. 플레이어를 초기화하고 비디오 및 SRT 파일 주소를 설정합니다.



```
var player = TCPlayer('player-container-id', {});
```

```
player.src('https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4');
player.on('ready', function () {
  // 자막 파일 추가
  var subTrack = player.addRemoteTextTrack({
 src: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.srt', // 자막 파일
 kind: 'subtitles',
 srclang: 'zh-cn',
 label: '중국어',
 default: 'true',
  }, true);
});
```

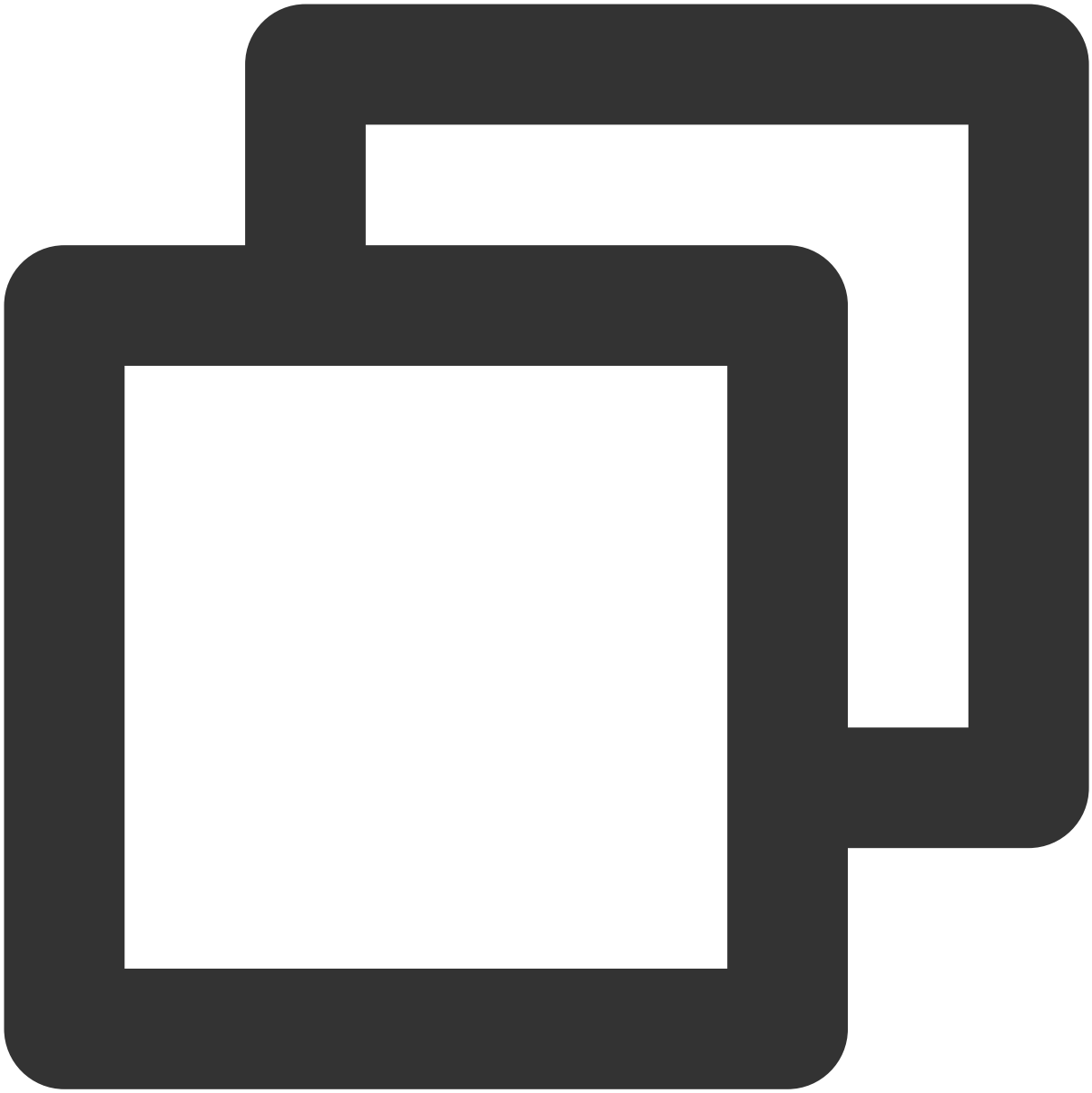
코드 샘플 가져오기:

동영상 자막 구성을 위한 샘플 코드

다국어 동영상 자막 설정

작업 순서는 다음과 같습니다.

1. CI의 음성 인식 기능으로 자막 파일을 생성하고 다국어로 번역합니다.
2. 1단계에서 생성된 다국어 SRT 파일의 객체 주소를 가져옵니다.
3. 플레이어를 초기화하고 비디오 및 다국어 SRT 파일 주소를 설정합니다.



```
var player = TCPlayer('player-container-id', {});
player.src('https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4');
player.on('ready', function () {
  // 중국어 자막 설정
  var subTrack = player.addRemoteTextTrack({
 src: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/zh.srt', // 자막 파일
 kind: 'subtitles',
 srclang: 'zh-cn',
 label: '중국어',
 default: 'true',
  }, true);
```

```
    // 영어 자막 설정
    var subTrack = player.addRemoteTextTrack({
   src: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/en.srt', // 자막 파일
   kind: 'subtitles',
   srclang: 'en',
   label: '영어',
   default: 'false',
    }, true);
  });
```

코드 샘플 가져오기:

다국어 자막 구성을 위한 샘플 코드

DPlayer를 사용하여 COS에서 비디오 재생

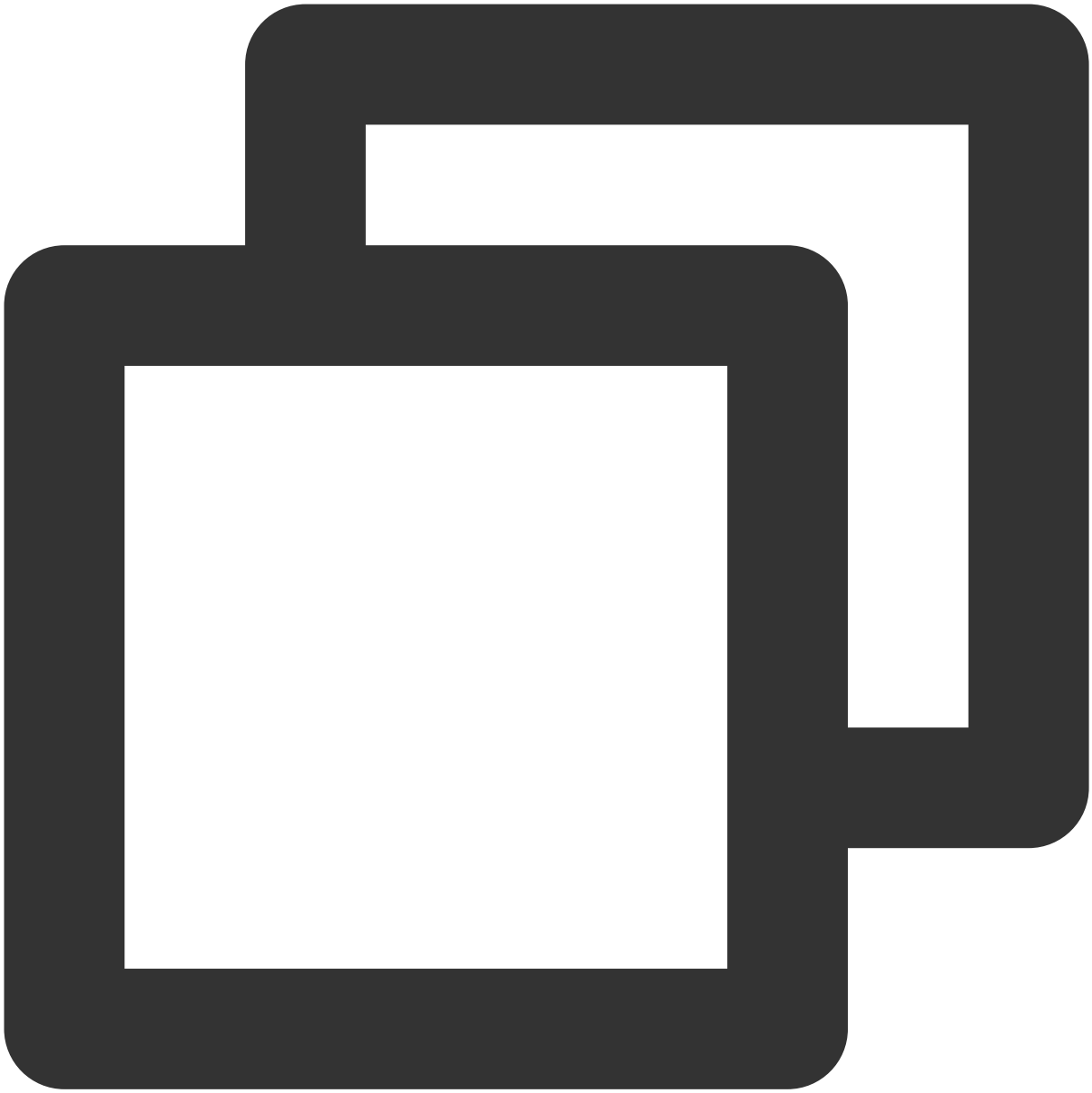
최종 업데이트 날짜: : 2024-06-24 16:44:04

소개

본문은 DPlayer를 Cloud Infinite(CI)의 풍부한 오디오/비디오 기능과 함께 사용하여 Web 브라우저에서 COS에 저장된 비디오 파일을 재생하는 방법을 설명합니다.

통합 가이드

1단계: 플레이어 스크립트 파일 및 필요한 종속성 파일을 페이지로 가져오기



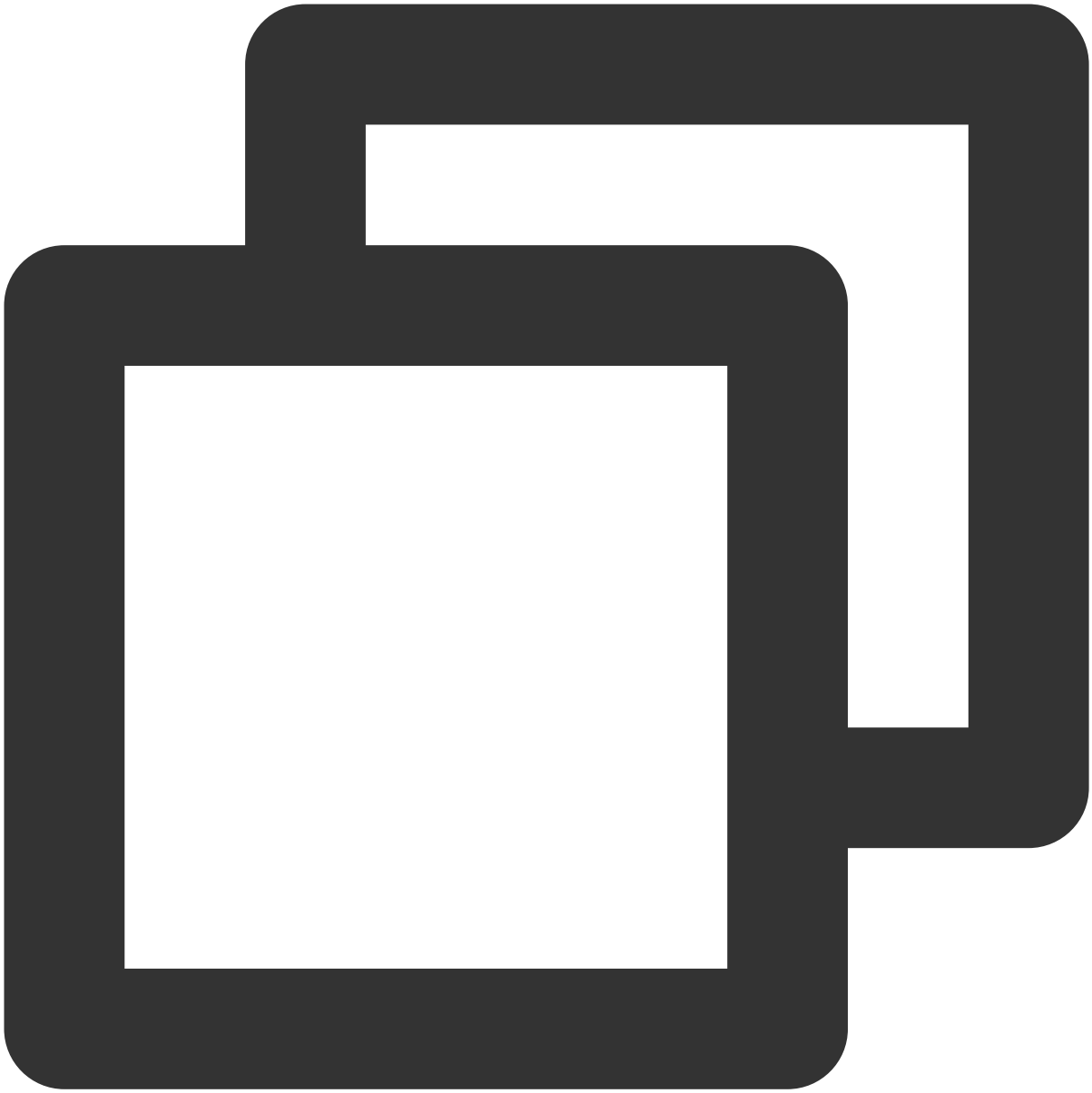
```
<!-- 플레이어 스크립트 파일 -->
<script src="https://cdn.jsdelivr.net/npm/dplayer@1.26.0/dist/DPlayer.min.js"></scr
```

설명：

플레이어 사용 시 상기 정적 리소스를 직접 배포하는 것을 권장합니다.

2단계: 플레이어 컨테이너 노드 설정

플레이어를 표시할 페이지에 플레이어 컨테이너를 추가합니다. 예를 들어 index.html에 다음 코드를 추가합니다(컨테이너 ID와 너비 및 높이를 사용자 지정할 수 있음).



```
<div id="dplayer" style="width: 100%; height: 100%"></div>
```

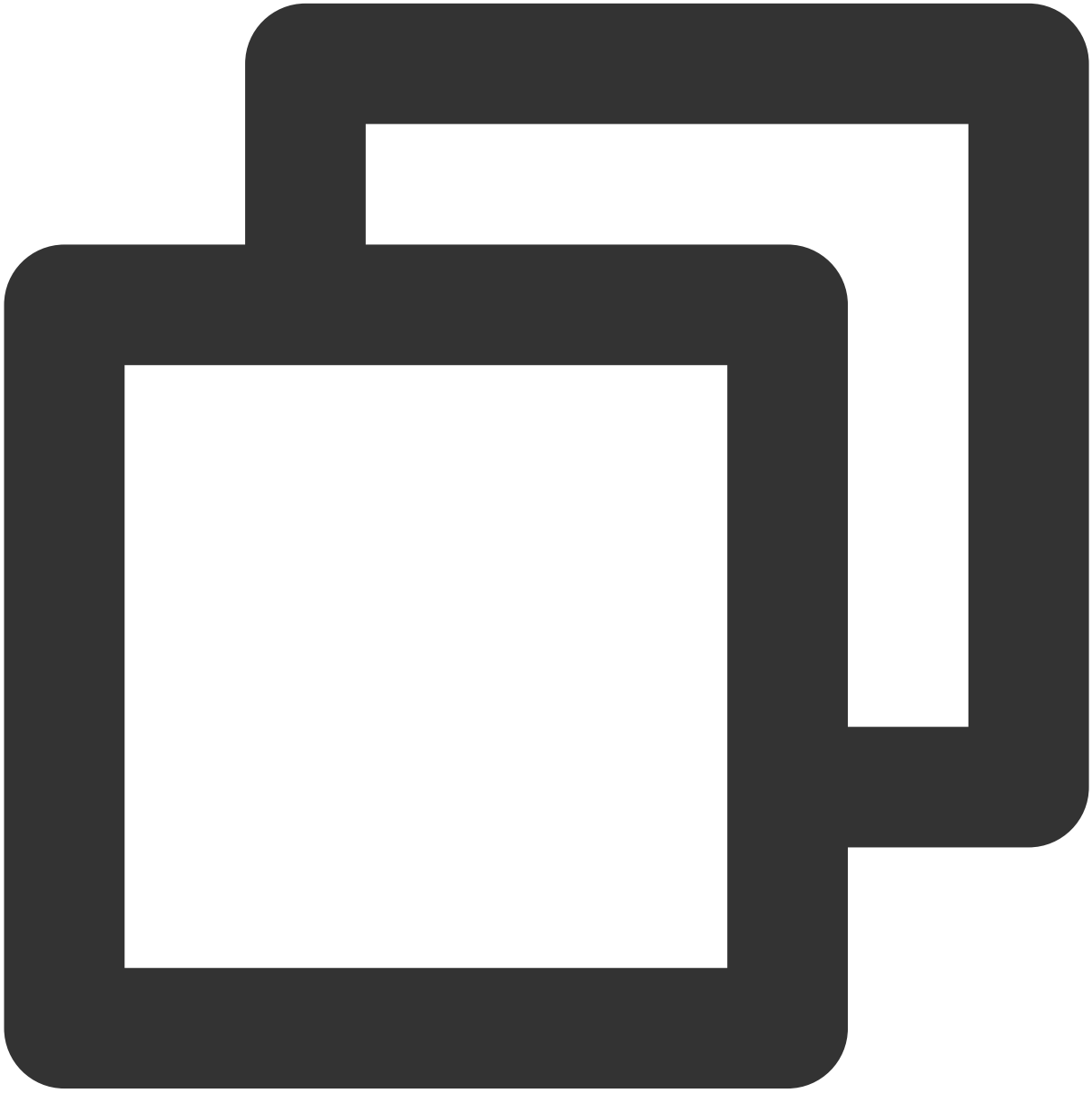
3단계: 비디오 파일 객체 주소 가져오기

1. 버킷을 생성합니다.
2. 비디오 파일 객체를 업로드합니다.
3. `https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.<비디오 형식>` 형식의 비디오 파일 객체 주소를 가져옵니다.

설명：

Cross-Origin 액세스가 포함된 경우 CORS 설정을 참고하여 버킷에 대한 CORS를 설정해야 합니다.
버킷 권한이 비공개 읽기/쓰기인 경우 객체 주소에 서명이 있어야 합니다. 자세한 내용은 Request Signature를 참고하십시오.

4단계: 플레이어 초기화 및 COS 동영상 파일 객체 URL 전달



```
const dp = new DPlayer({
  container: document.getElementById('dplayer'),
  video: {
      url: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4', // COS 버
  },
```

```
});
```

기능 가이드

다양한 형식의 비디오 파일 재생

1. COS 버킷에 있는 비디오 파일의 객체 주소를 가져옵니다.

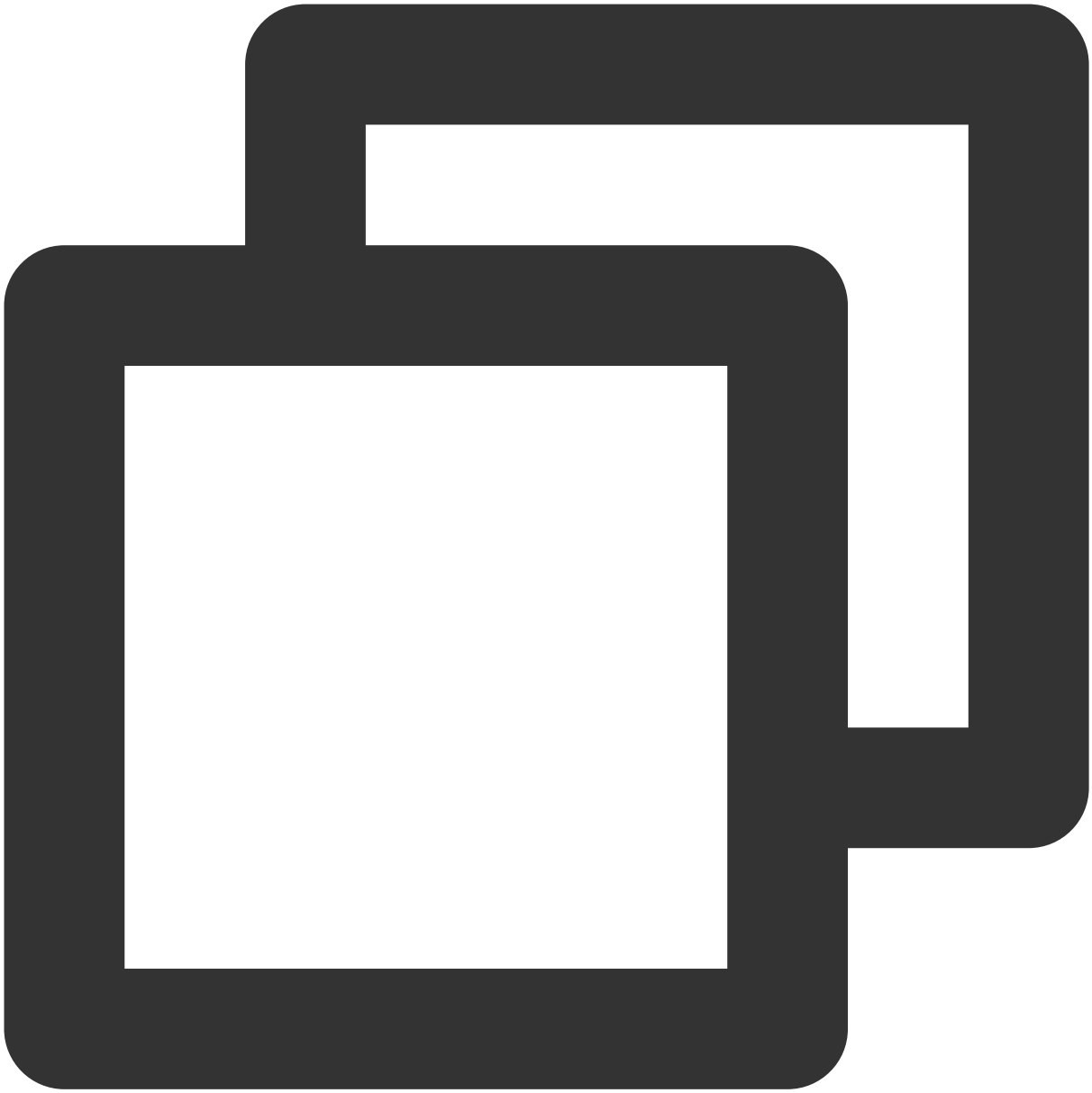
설명：

트랜스코딩되지 않은 비디오는 재생 중에 호환성 문제가 발생할 수 있으므로 재생을 위해 비디오를 트랜스코딩하는 것이 좋습니다. CI의 Audio/Video Transcoding 기능을 사용하여 다양한 형식의 비디오 파일을 얻을 수 있습니다.

2. 다른 비디오 형식의 경우 다른 브라우저와의 호환성을 보장하기 위해 해당 종속성을 가져와야 합니다.

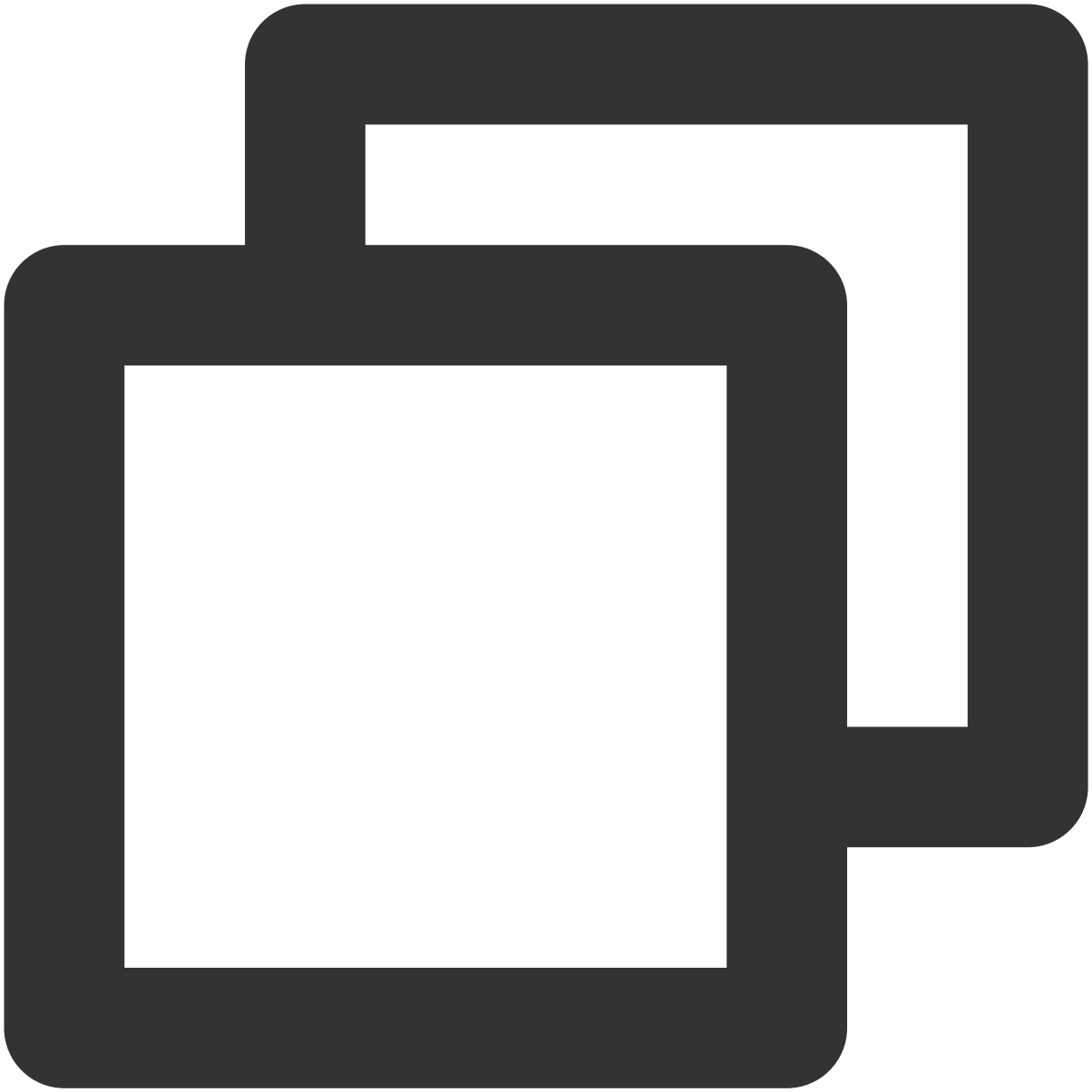
MP4: 다른 종속성을 가져올 필요가 없습니다.

HLS: Chrome 및 Firefox와 같은 최신 브라우저에서 HTML5를 통해 HLS 동영상을 재생하려면 tcplayer.min.js를 가져오기 전에 hls.min.js를 가져와야 합니다.



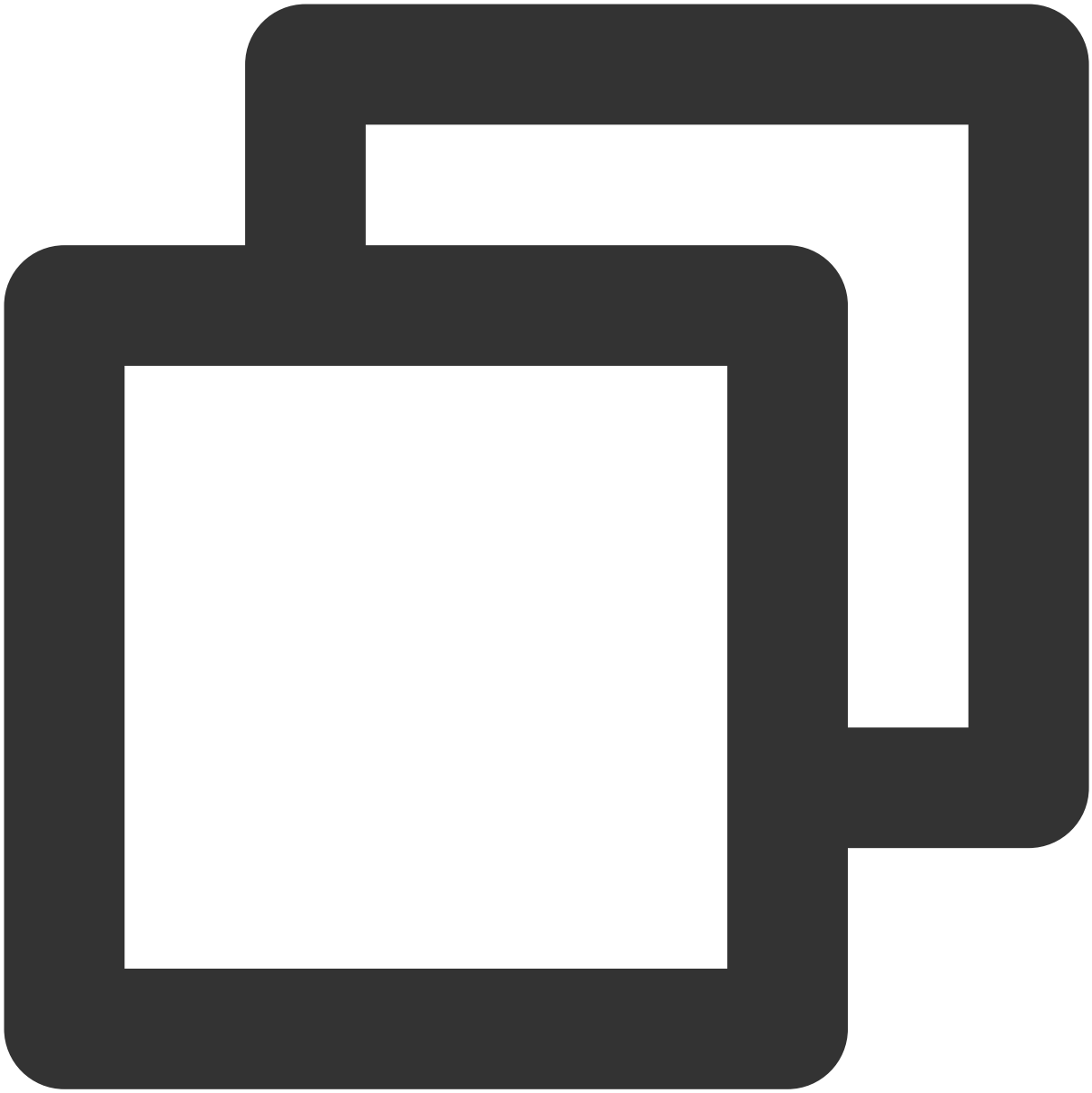
```
<script src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.2.1/libs/hls.m
```

FLV: Chrome 및 Firefox와 같은 최신 브라우저에서 H5를 통해 FLV 비디오를 재생하려면 tcplayer.min.js를 가져오기 전에 flv.min.js를 가져와야 합니다.



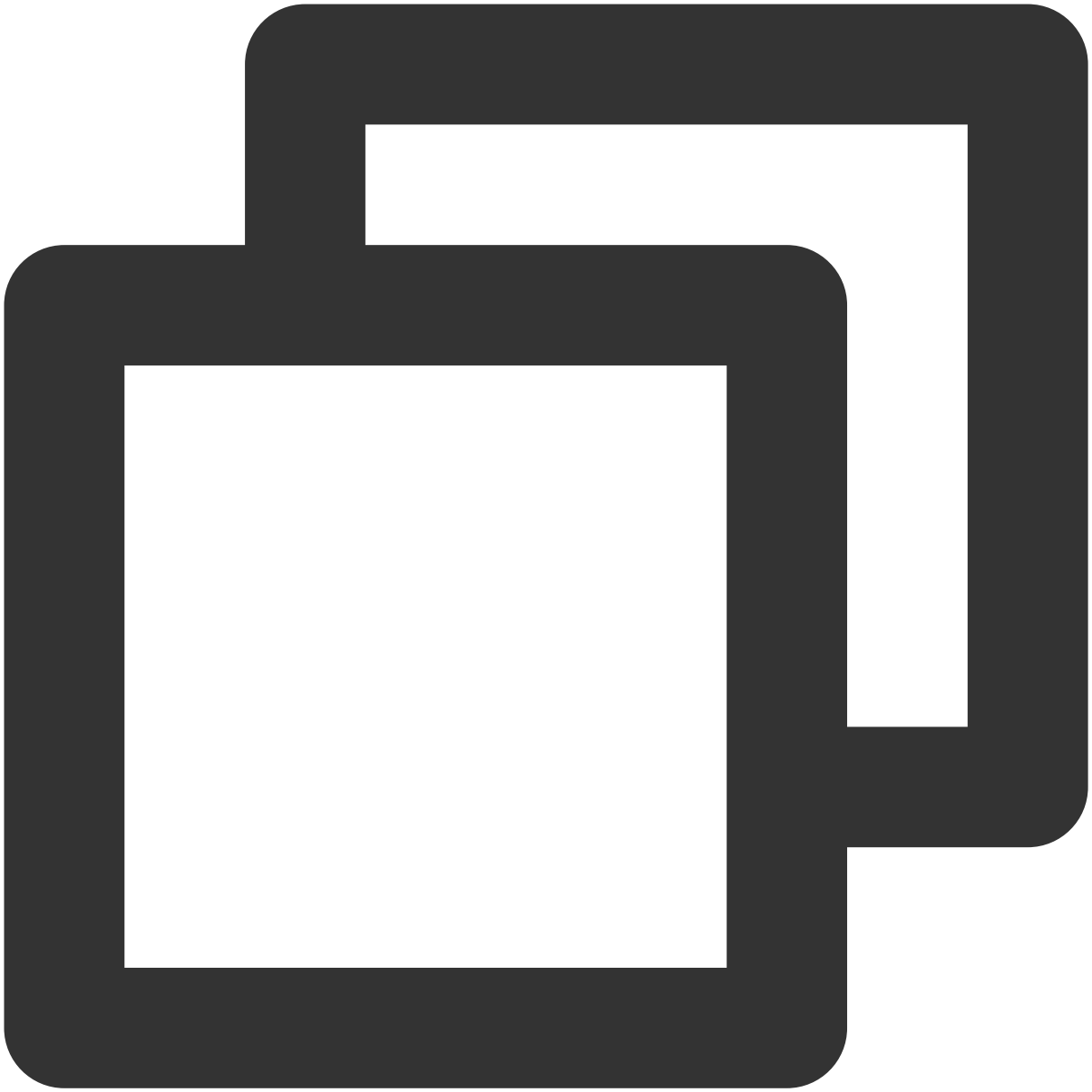
```
<script src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.2/libs/flv.m
```

DASH: dash.all.min.js 파일을 가져와야 합니다.



```
<script src="https://cos-video-1258344699.cos.ap-guangzhou.myqcloud.com/lib/dash.
```

3. 플레이어를 초기화하고 객체 주소를 전달합니다.



```
const dp = new DPlayer({
  container: document.getElementById('dplayer'),
  video: {
   url: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4', // COS 비디오
  },
});
```

코드 샘플 가져오기:

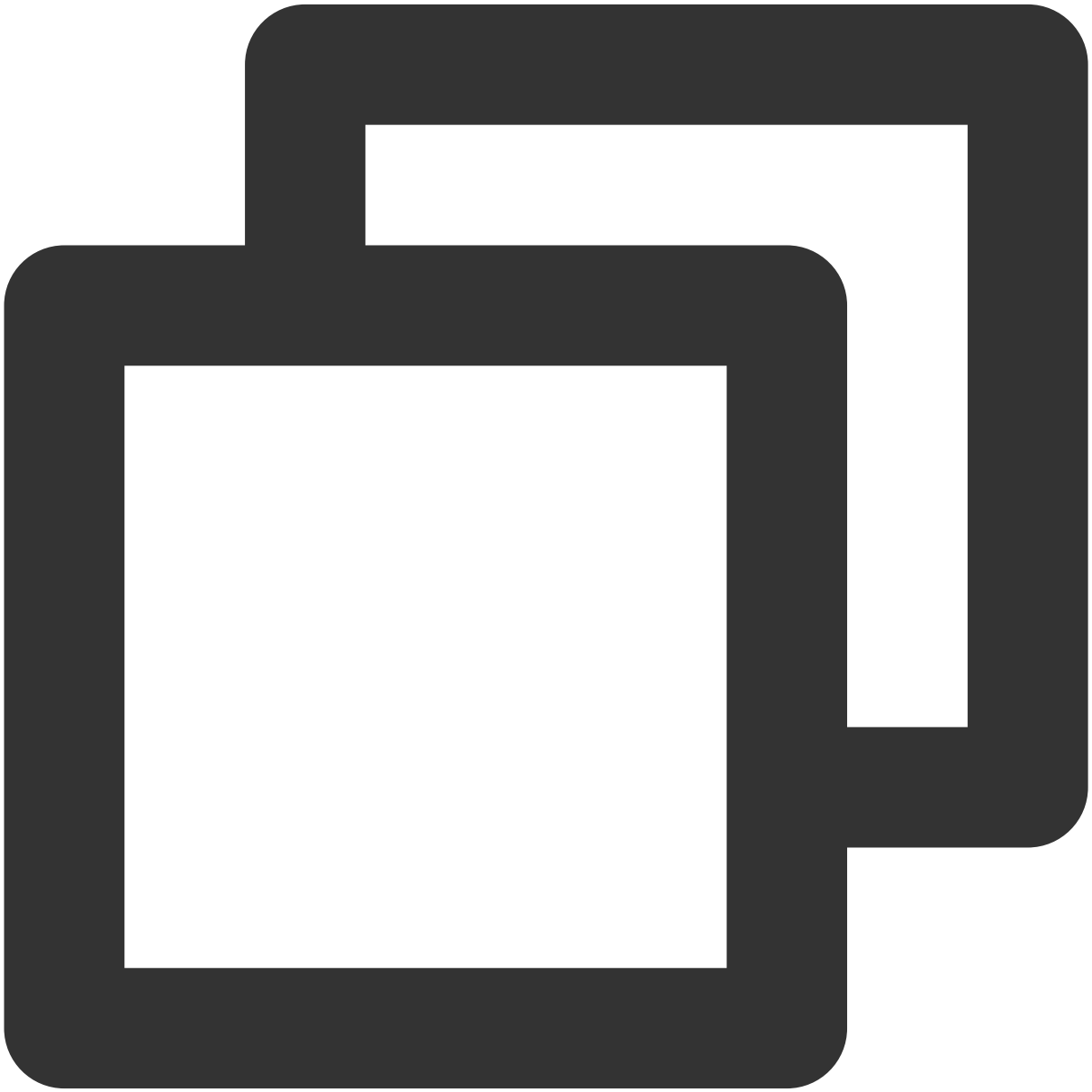
MP4 재생 샘플 코드

FLV 재생 샘플 코드

HLS 재생 샘플 코드
DASH 재생 샘플 코드

PM3U8 비디오 재생

PM3U8은 비공개 M3U8 비디오 파일을 나타냅니다. COS는 비공개 M3U8 TS 리소스를 얻기 위한 다운로드 인증 API를 제공합니다. 자세한 내용은 Private M3U8 API를 참고하십시오.



```
const dp = new DPlayer({
  container: document.getElementById('dplayer'),
```

```
    // pm3u8에 대한 자세한 내용: https://www.tencentcloud.com/document/product/436/4722
    video: {
      url: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8?ci-process
    }
  });
```

코드 샘플 가져오기:

PM3U8 재생 샘플 코드

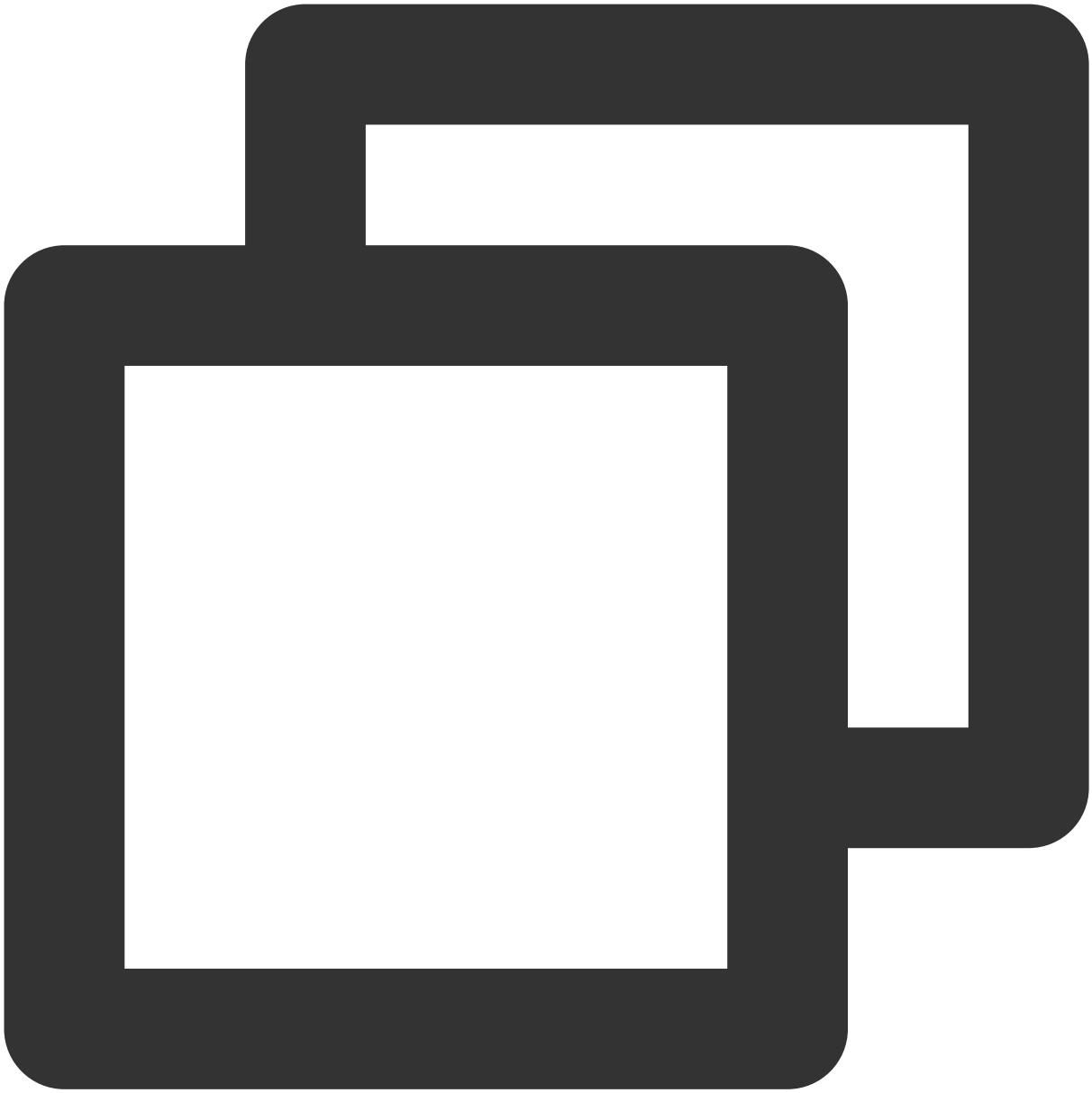
썸네일 설정

1. COS 버킷에 있는 썸네일의 객체 주소를 가져옵니다.

주의 :

CI의 Intelligent Thumbnail 기능은 최적의 프레임을 추출하여 썸네일을 생성하여 비디오 콘텐츠를 더 매력적으로 만들 수 있습니다.

2. 플레이어를 초기화하고 썸네일 이미지를 설정합니다.



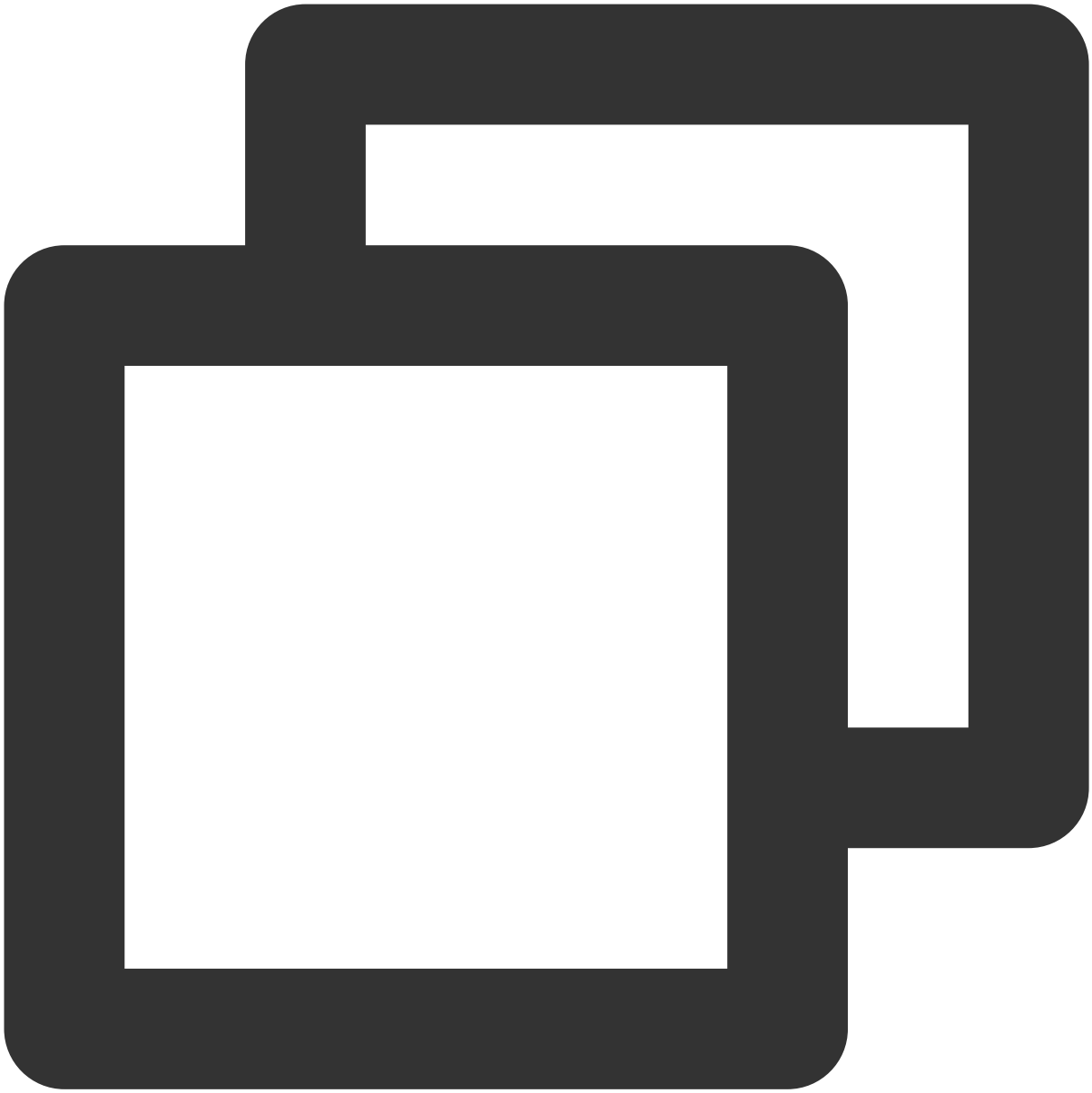
```
const dp = new DPlayer({
  container: document.getElementById('dplayer'),
  video: {
 url: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4',
 pic: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.png',
  },
});
```

코드 샘플 가져오기:

썸네일 구성 샘플 코드

HLS 암호화 비디오 재생

CI는 동영상 콘텐츠의 보안을 확보하고 동영상의 무단 다운로드 및 배포를 방지하기 위해 개인이 읽을 수 있는 파일보다 안전한 HLS 동영상 콘텐츠를 암호화하는 기능을 제공합니다. 암호화된 비디오는 재생 액세스 권한이 없는 사용자에게 배포할 수 없습니다. 다운로드하더라도 여전히 암호화되어 악의적으로 재배포할 수 없습니다. 이렇게 하면 비디오 저작권이 침해되는 것을 방지할 수 있습니다.
작업 순서는 다음과 같습니다.
1. HLS 암호화 비디오 재생 및 COS 오디오/비디오 실습 | 비디오 암호화를 참고하여 암호화된 비디오를 생성합니다.
2. 플레이어를 초기화하고 비디오 객체 주소를 전달합니다.



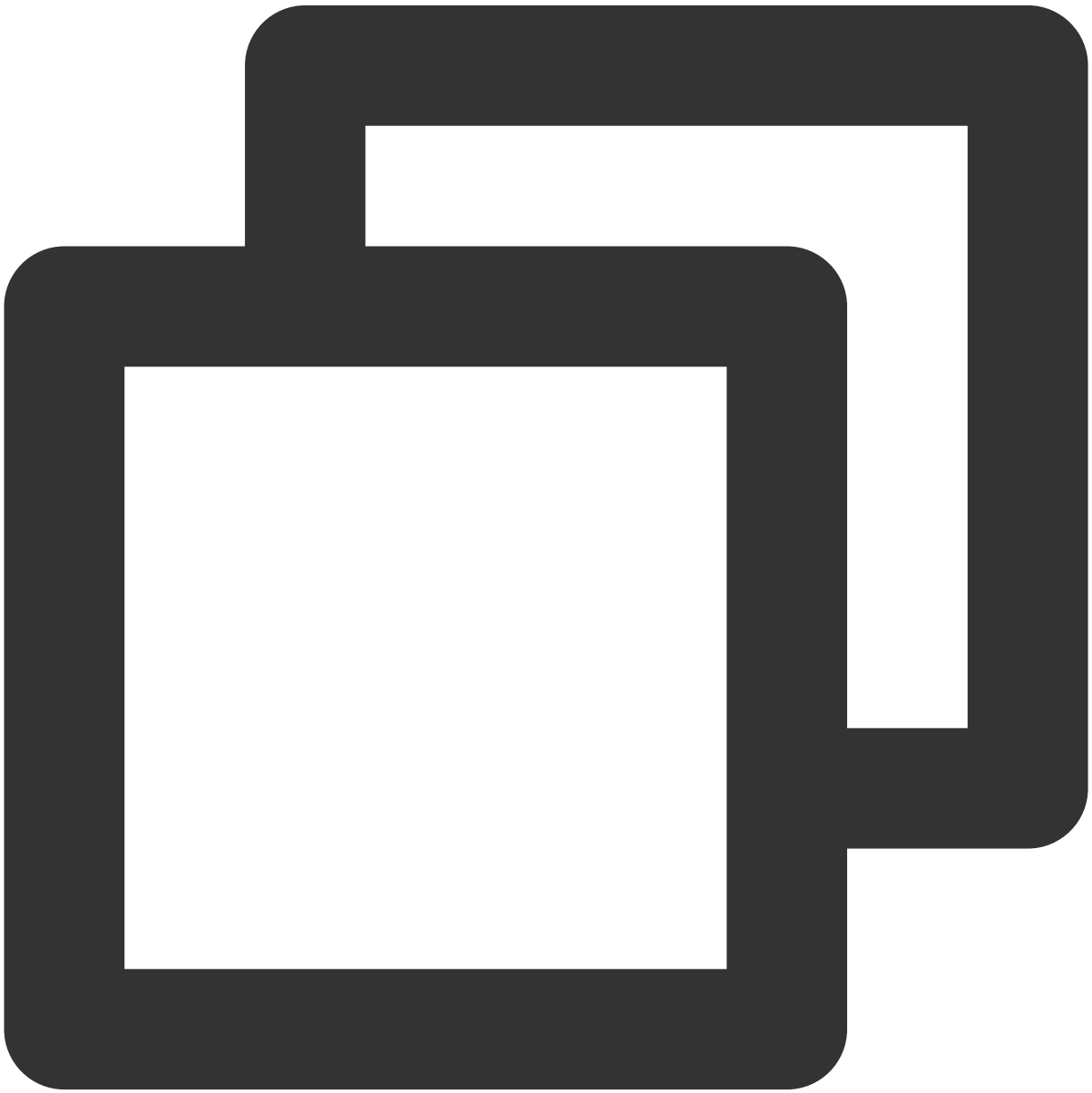
```
 const dp = new DPlayer({
container: document.getElementById('dplayer'),
video: {
  url: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8' // 암호화된 비
}
 });
```

코드 샘플 가져오기:

HLS 암호화 재생 샘플 코드

다중 해상도 전환

CI의 Adaptive Bitrate Streaming 기능은 비디오를 트랜스코딩하고 이를 어댑티 비트스트림으로 리먹싱하여 출력할 수 있으므로 다양한 네트워크 조건에서 비디오 콘텐츠를 빠르게 배포할 수 있습니다. 플레이어는 현재 대역폭에 따라 비디오를 재생하는 데 가장 적합한 비트레이트를 동적으로 선택할 수 있습니다. 자세한 내용은 COS 오디오/비디오 실습 | 데이터 처리 워크플로로 다중 해상도 비디오 재생을 참고하십시오.
작업 순서는 다음과 같습니다.
1. CI의 Adaptive Bitrate Streaming 기능으로 멀티 비트레이트 어댑티브 HLS 또는 DASH 대상 파일을 생성합니다.
2. 플레이어를 초기화하고 비디오 객체 주소를 전달합니다.



```
const dp = new DPlayer({
  container: document.getElementById('dplayer'),
  video: {
   url: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8', //  멀티 비
  },
});
```

코드 샘플 가져오기:

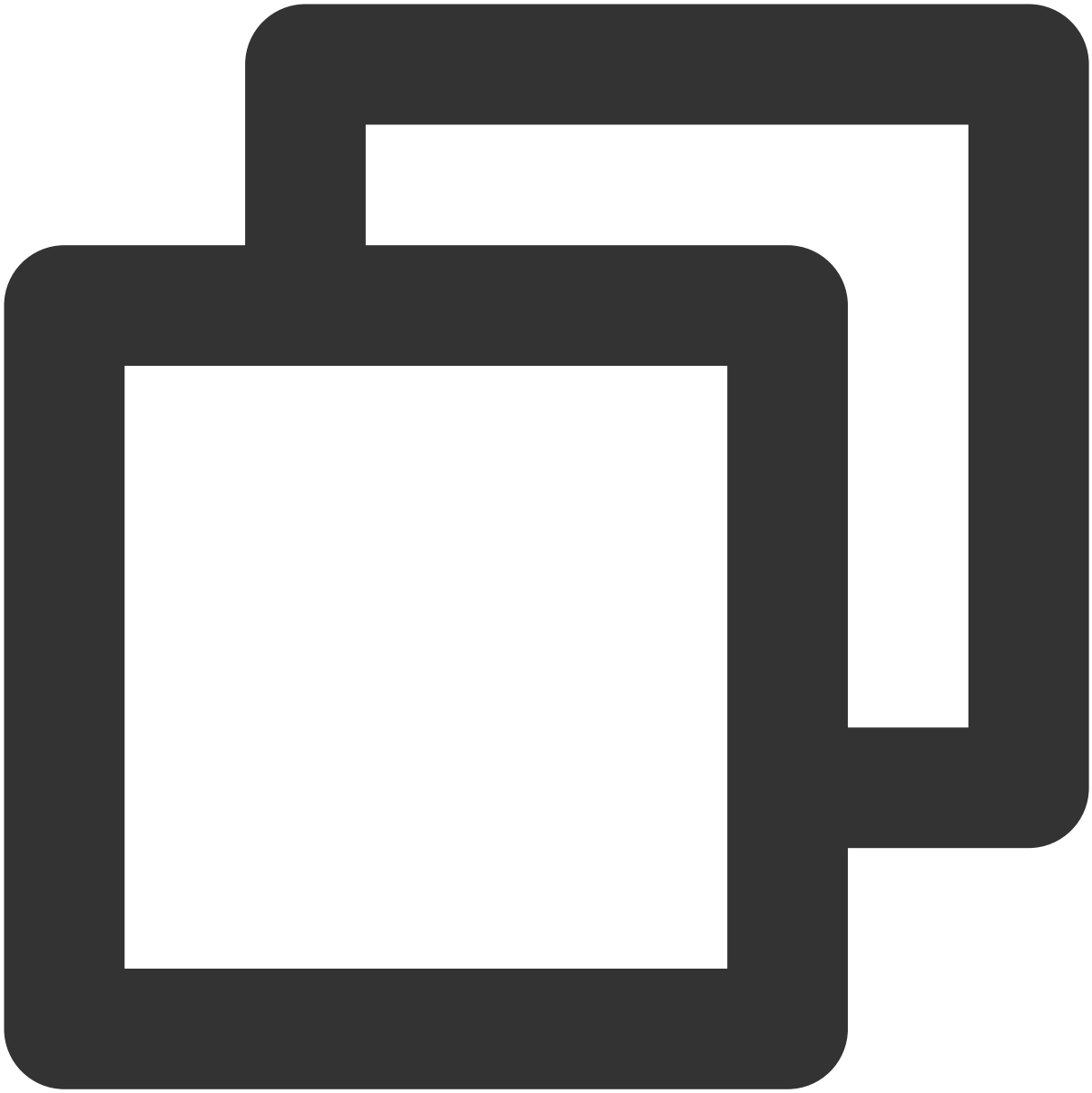
해상도 전환 샘플 코드

왼쪽 상단 LOGO 설정

플레이어를 사용하면 왼쪽 상단 모서리에 LOGO를 설정할 수 있습니다.

작업 순서는 다음과 같습니다.

1. COS 버킷에서 LOGO의 객체 주소를 가져옵니다.
2. 플레이어를 초기화하고 LOGO를 설정합니다.



```
const dp = new DPlayer({
 container: document.getElementById('dplayer'),
  video: {
   url: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4',
  },
 logo: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.svg'
});
```

코드 샘플 가져오기:

좌측 상단 LOGO 설정 샘플 코드

VideojsPlayer를 사용하여 COS에서 비디오 재생

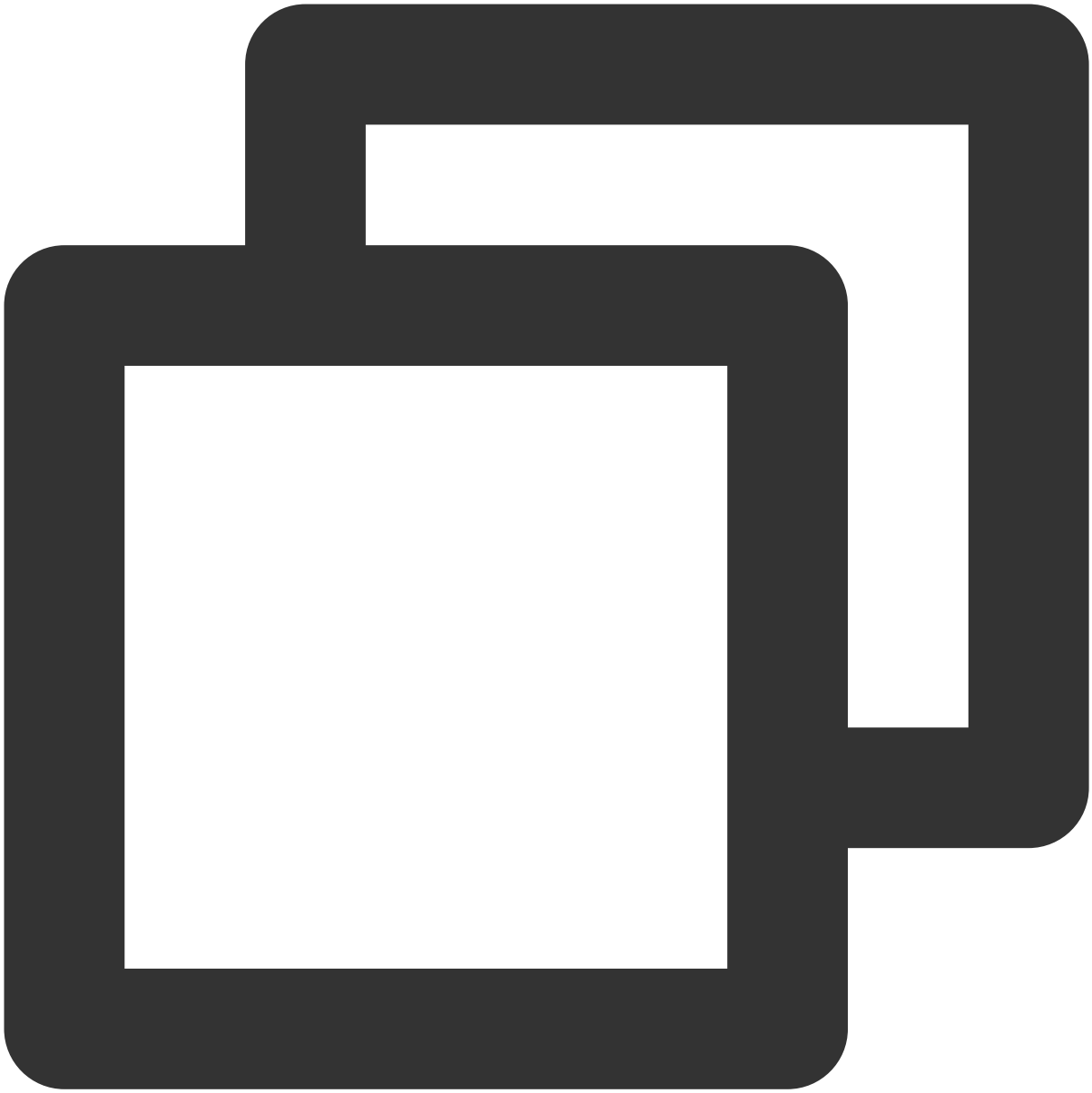
최종 업데이트 날짜: : 2024-06-24 16:44:04

소개

본문은 VideojsPlayer를 Cloud Infinite (CI)의 풍부한 오디오/비디오 기능과 함께 사용하여 Web 브라우저에서 COS에 저장된 비디오 파일을 재생하는 방법을 설명합니다.

통합 가이드

1단계: 플레이어 양식 및 스크립트 파일을 페이지로 가져오기



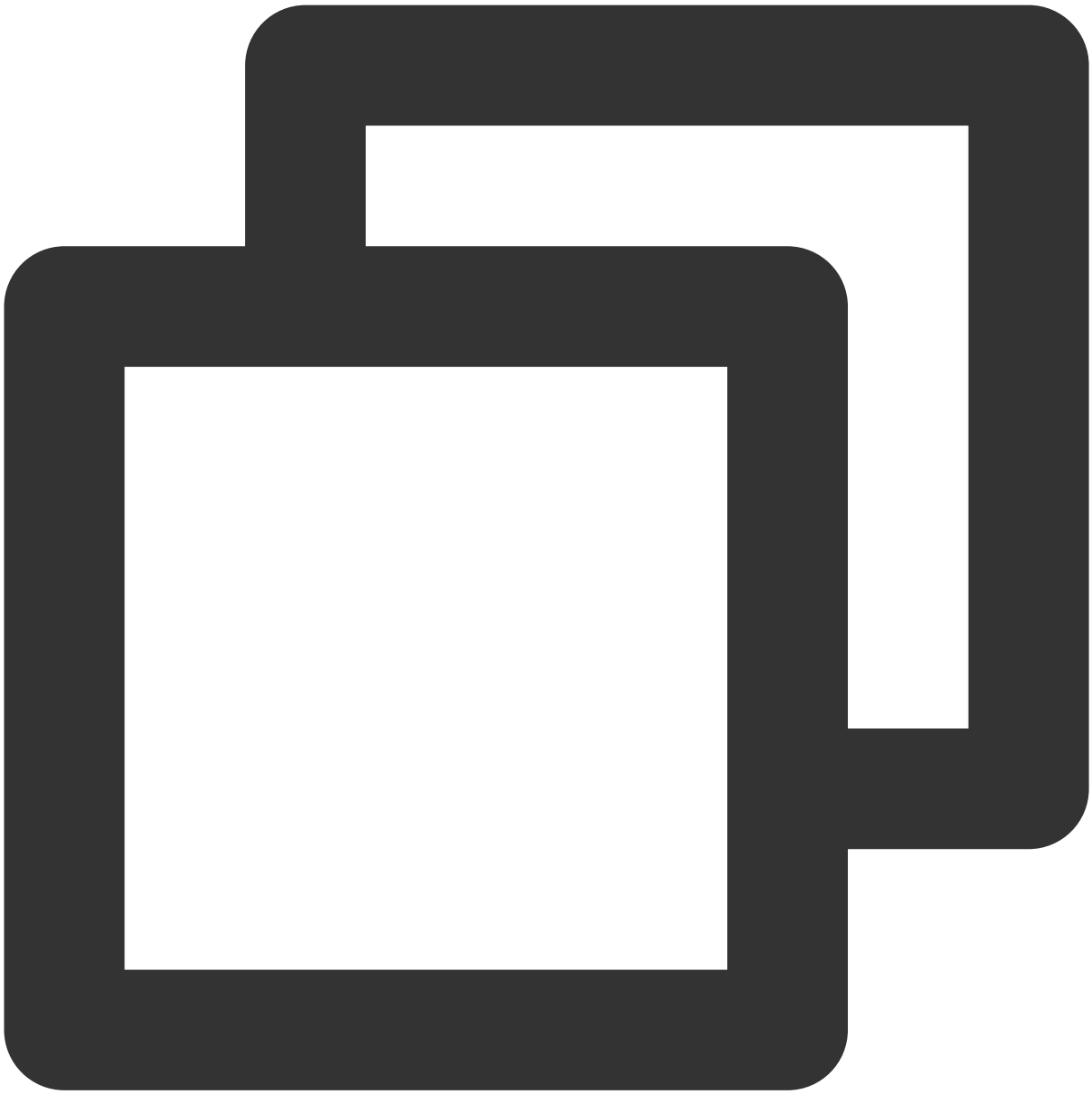
```
<!-- 플레이어 양식 파일 -->
<link href="https://vjs.zencdn.net/7.19.2/video-js.css" rel="stylesheet" />
<!-- 플레이어 스크립트 파일 -->
<script src="https://vjs.zencdn.net/7.19.2/video.min.js"></script>
```

설명 :

플레이어 사용 시 위의 정적 리소스를 직접 배포하는 것을 권장합니다.

2단계: 플레이어 컨테이너 노드 설정

플레이어를 표시할 페이지에 플레이어 컨테이너를 추가합니다. 예를 들어 index.html에 다음 코드를 추가합니다(컨테이너 ID와 너비 및 높이를 사용자 지정할 수 있음).



```
<video
  id="my-video"
  class="video-js"
  controls
  preload="auto"
  width="100%"
  height="100%"
  data-setup="{}"
```

```
></video>
```

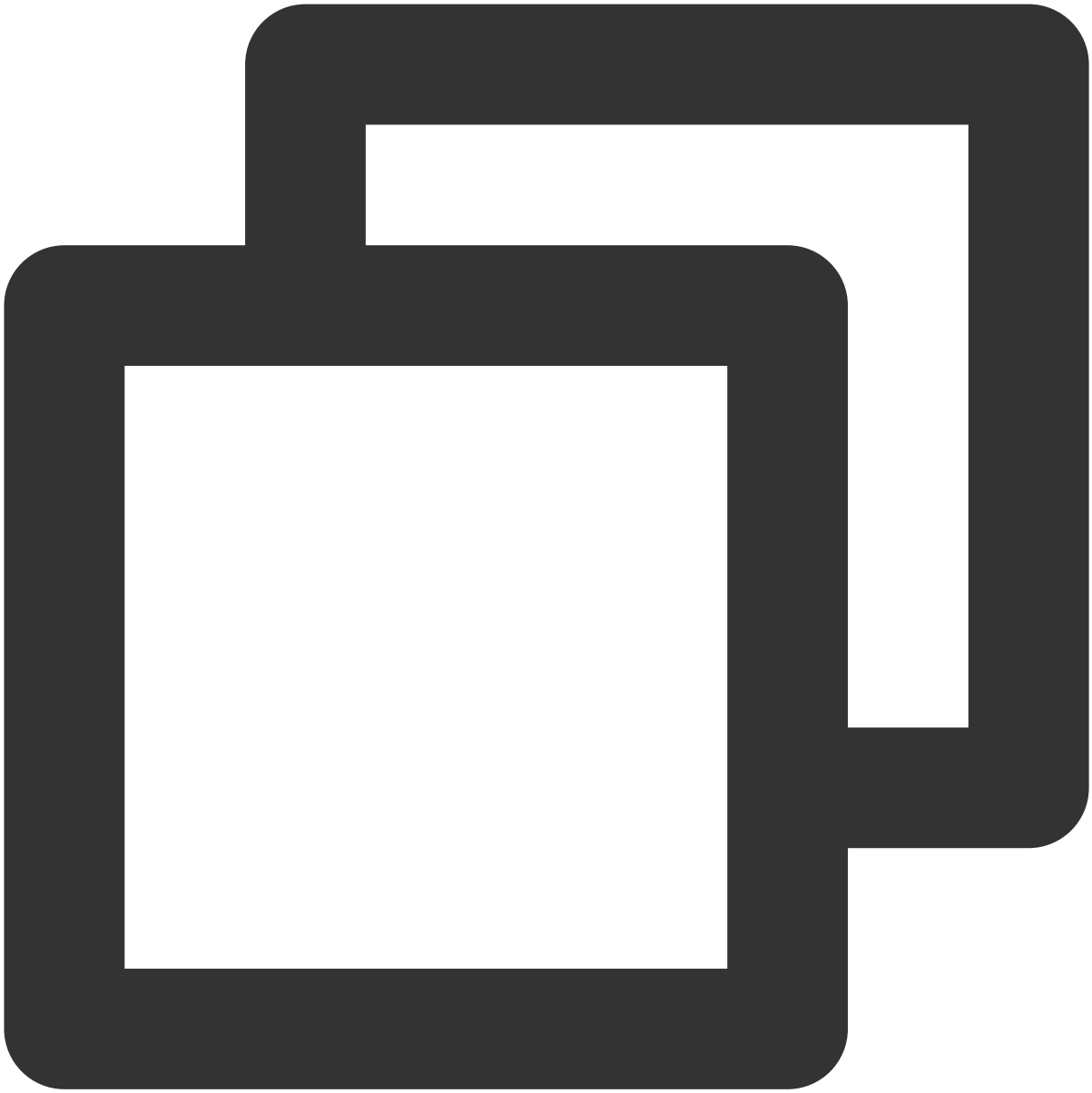
3단계: 비디오 파일 객체 주소 가져오기

1. 버킷을 생성합니다.
2. 비디오 파일 객체를 업로드합니다.
3. `https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.<비디오 형식>` 형식의 비디오 파일 객체 주소를 가져옵니다.

설명：

Cross-Origin 액세스가 포함된 경우 CORS 설정을 참고하여 버킷에 대한 CORS를 설정해야 합니다.
버킷 권한이 비공개 읽기/쓰기인 경우 객체 주소에 서명이 있어야 합니다. 자세한 내용은 Request Signature를 참고하십시오.

4단계: 플레이어 컨테이너에 비디오 주소를 설정하고 COS 비디오 파일 객체 URL을 전달합니다.



```
<video
  id="my-video"
  class="video-js"
  controls
  preload="auto"
  width="100%"
  height="100%"
  data-setup="{}"
>
  <source
    src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4"
```

```
    type="video/mp4"
  />
</video>
```

기능 가이드

다양한 형식의 비디오 파일 재생

1. COS 버킷에 있는 비디오 파일의 객체 주소를 가져옵니다.

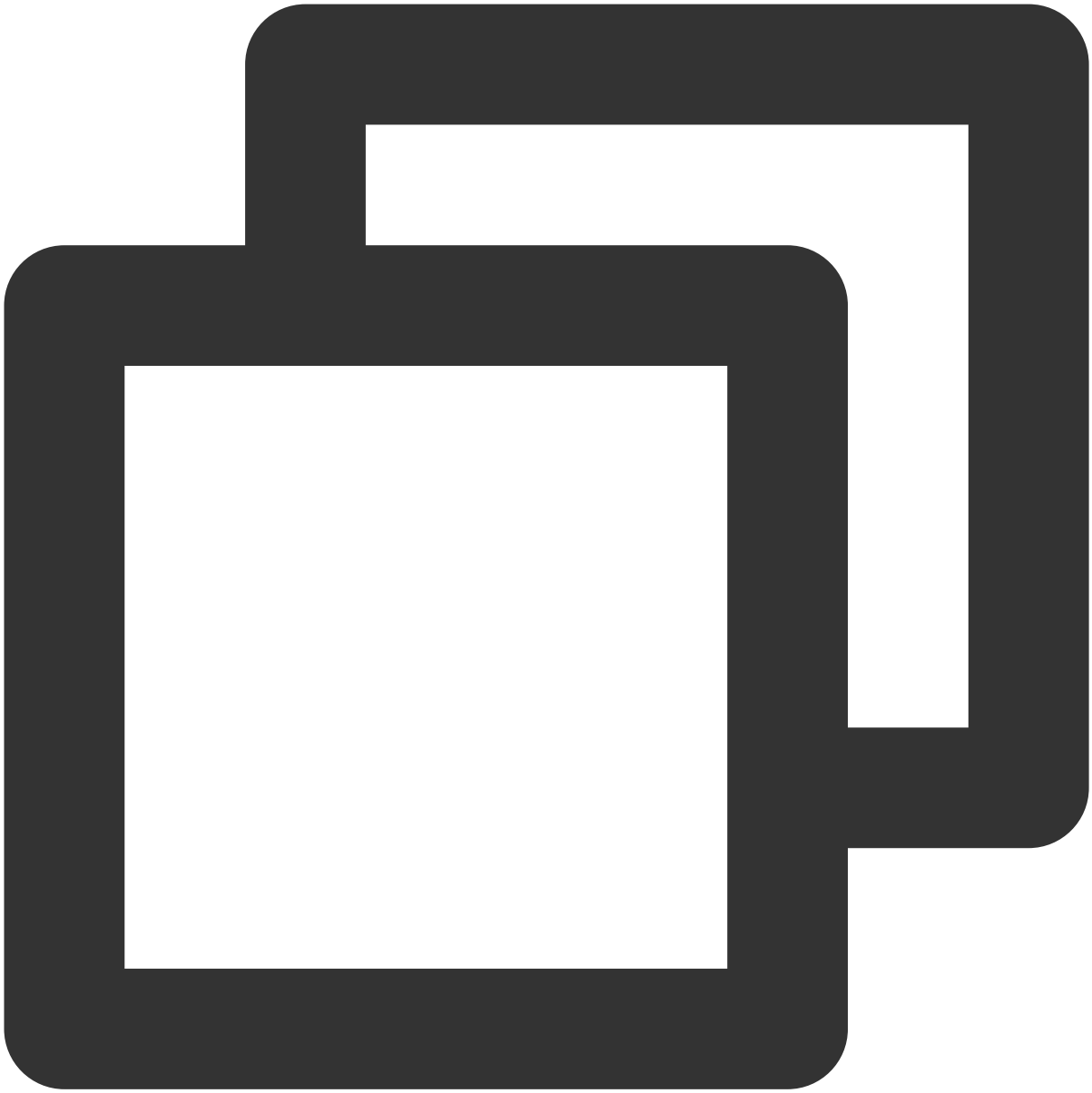
설명：

트랜스코딩되지 않은 비디오는 재생 중에 호환성 문제가 발생할 수 있으므로 재생을 위해 비디오를 트랜스코딩하는 것이 좋습니다. CI의 Audio/Video Transcoding 기능을 사용하여 다양한 형식의 비디오 파일을 얻을 수 있습니다.

2. 다른 비디오 형식의 경우 다른 브라우저와의 호환성을 보장하기 위해 해당 종속성을 가져와야 합니다.

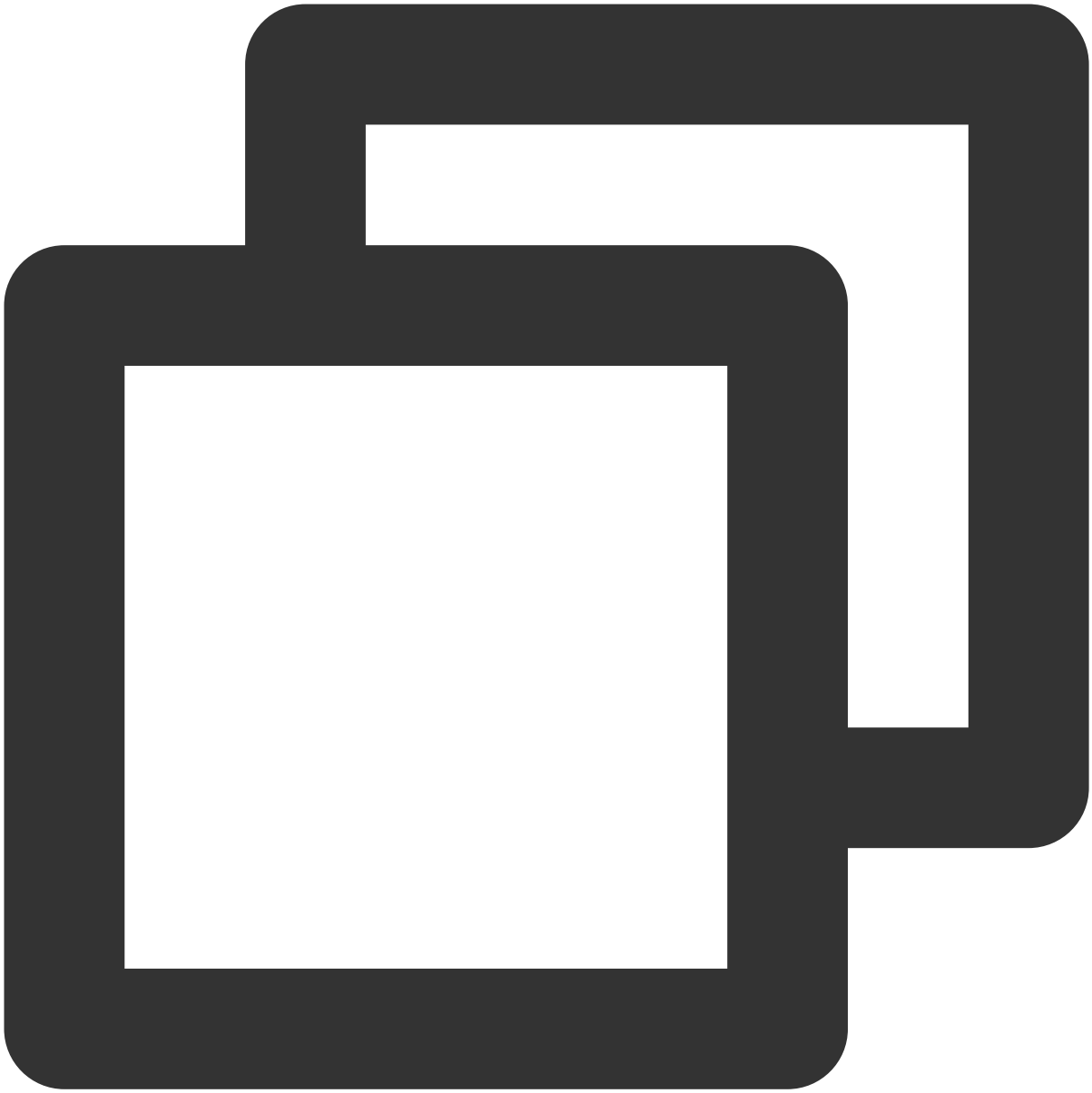
MP4: 다른 종속성을 가져올 필요가 없습니다.

HLS: Chrome 및 Firefox와 같은 최신 브라우저에서 HTML5를 통해 HLS 동영상을 재생하려면 tcplayer.min.js를 가져오기 전에 hls.min.js를 가져와야 합니다.



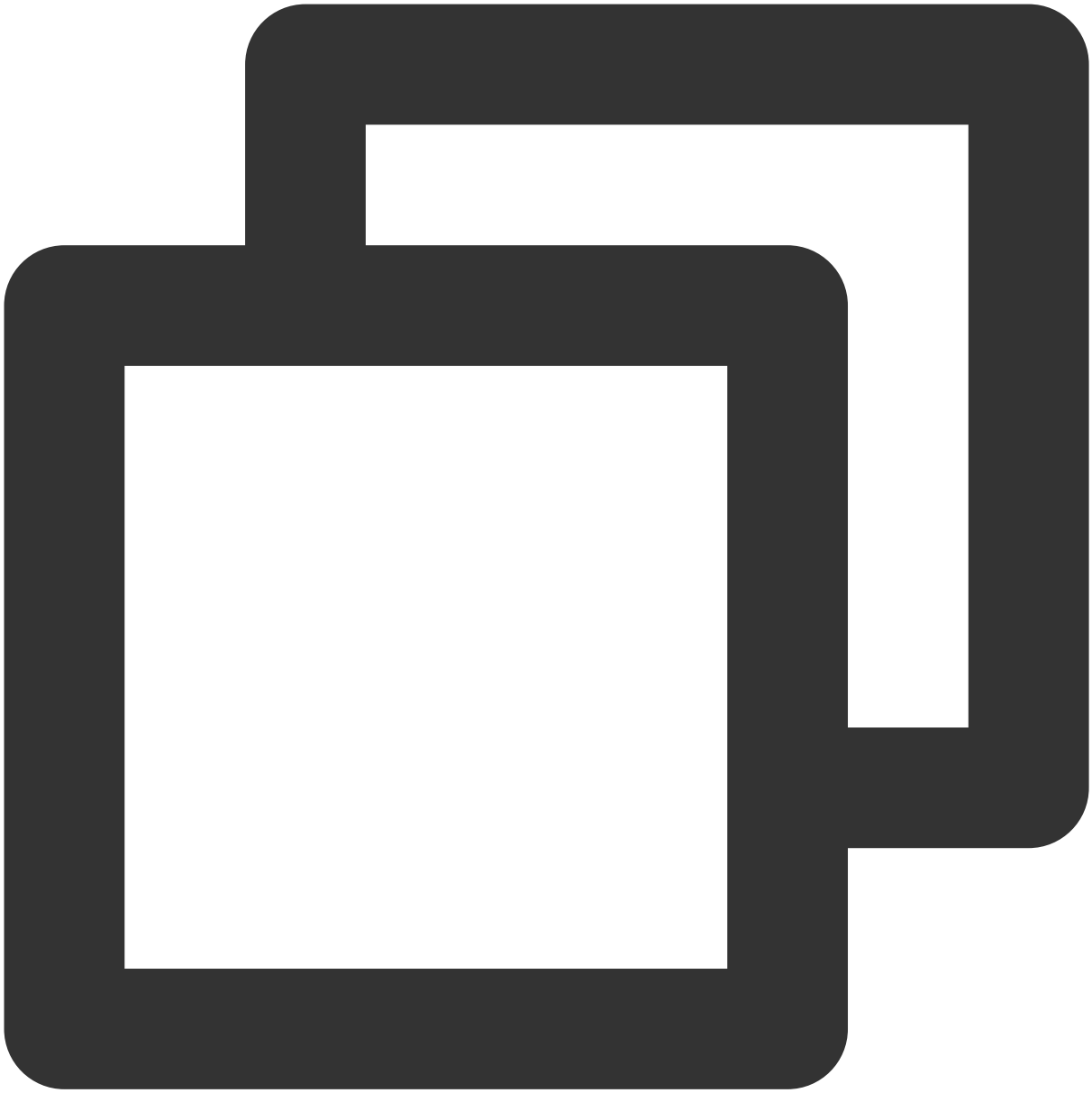
```
<script src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.2.1/libs/hls.m
```

FLV: Chrome 및 Firefox와 같은 최신 브라우저에서 H5를 통해 FLV 비디오를 재생하려면 tcplayer.min.js를 가져오기 전에 flv.min.js를 가져와야 합니다.



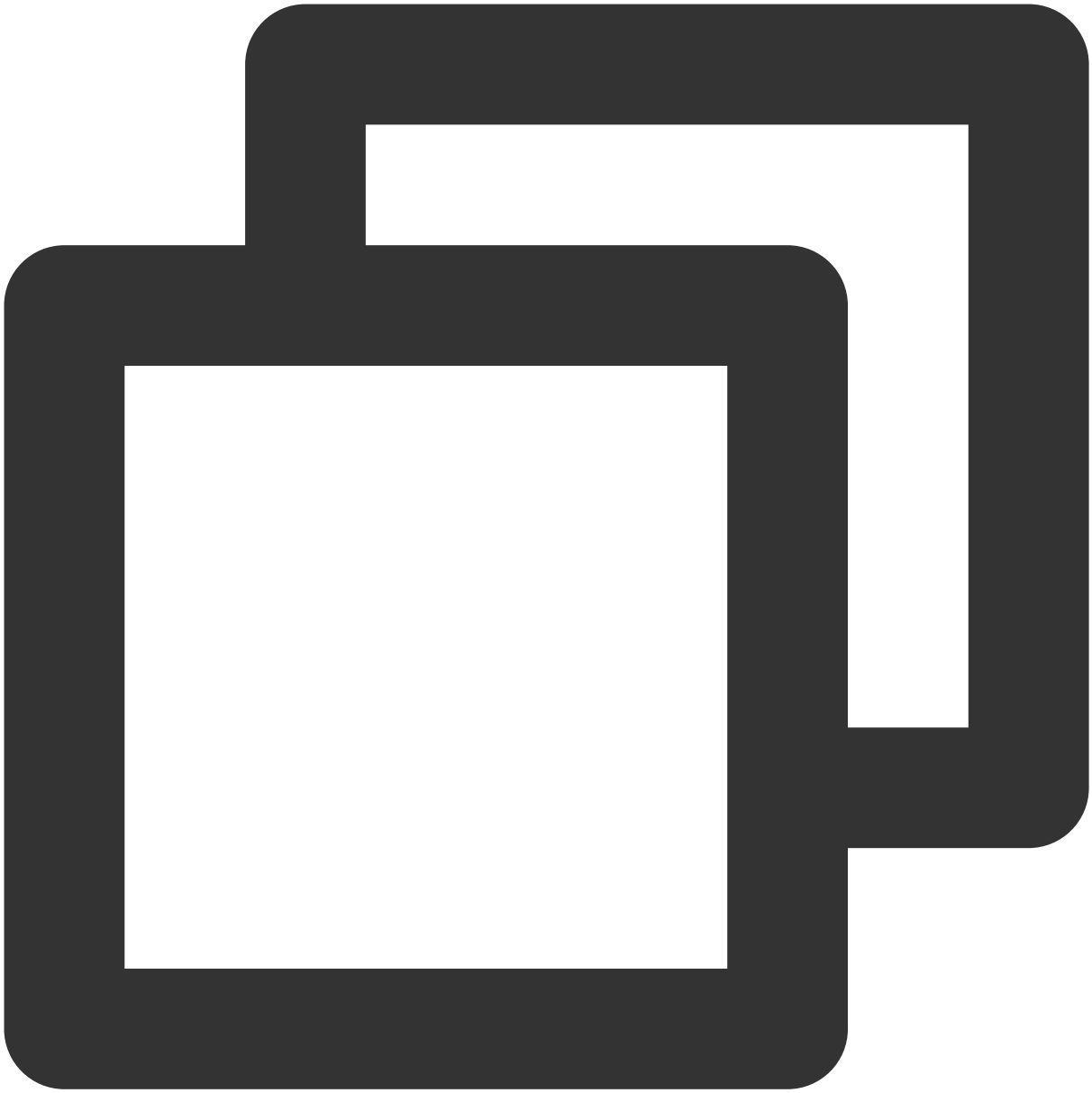
```
<script src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.2/libs/flv.m
```

DASH: dash.all.min.js 파일을 가져와야 합니다.



```
<script src="https://cos-video-1258344699.cos.ap-guangzhou.myqcloud.com/lib/dash.
```

3. 플레이어를 초기화하고 객체 주소를 전달합니다.



```
<!-- MP4 -->
<source
  src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4"
  type="video/mp4"
/>

<!-- HLS -->
<source
  src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8"
  type="application/x-mpegURL"
/>
```

```
<!-- FLV -->
<source
  src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.flv"
  type="video/x-flv"
/>

<!-- DASH -->
<source
  src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mpd"
  type="application/dash+xml"
/>
```

코드 샘플 가져오기:

MP4 재생 샘플 코드

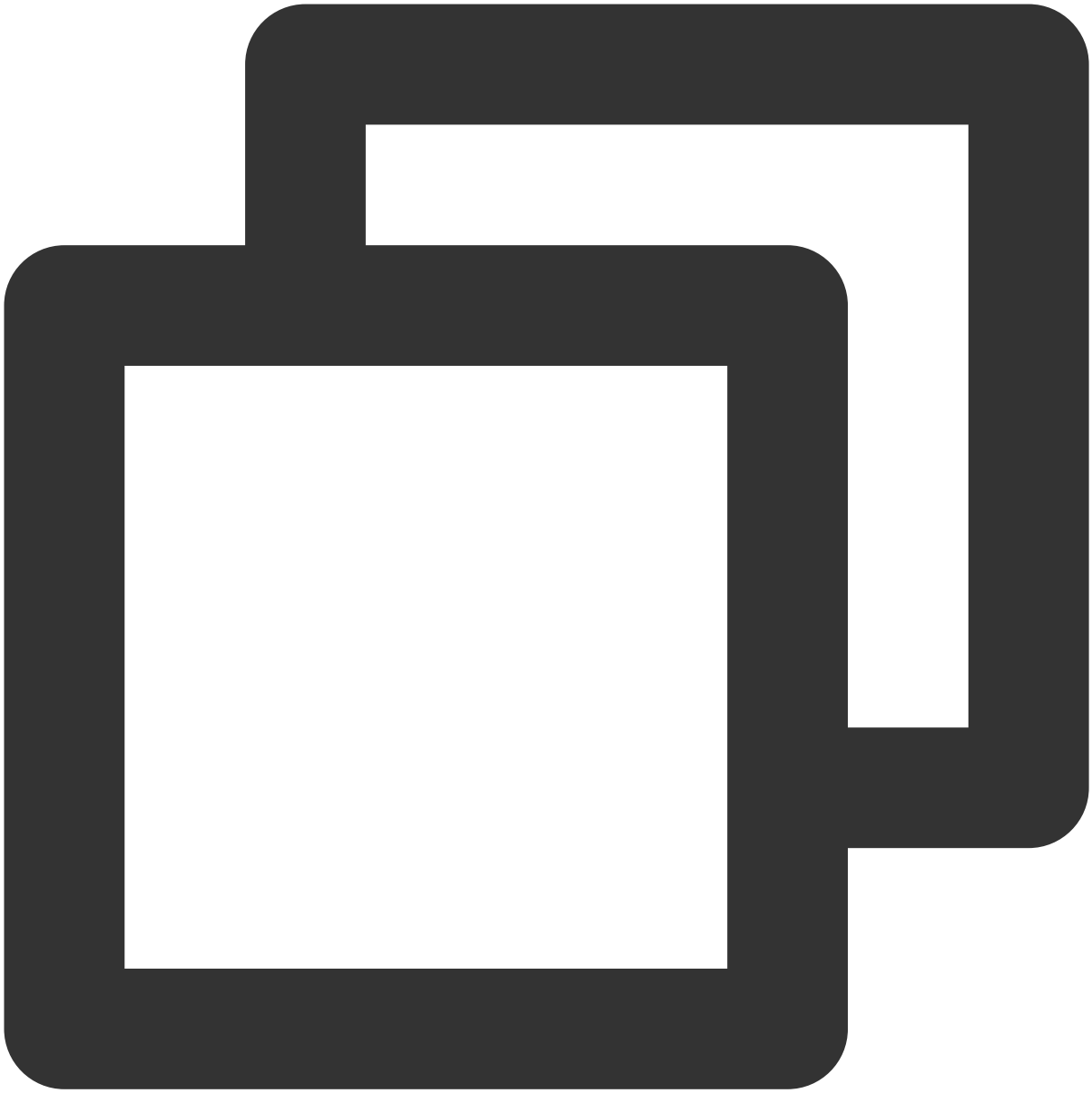
FLV 재생 샘플 코드

HLS 재생 샘플 코드

DASH 재생 샘플 코드

PM3U8 비디오 재생

PM3U8은 개인 M3U8 비디오 파일을 나타냅니다. COS는 개인 M3U8 TS 리소스를 가져오기 위한 다운로드 권한 API를 제공합니다. 자세한 내용은 Private M3U8 API를 참고하십시오.



```
<source
  src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8?ci-process=pm3
  type="application/x-mpegURL"
/>
```

코드 샘플 가져오기:

PM3U8 재생 샘플 코드

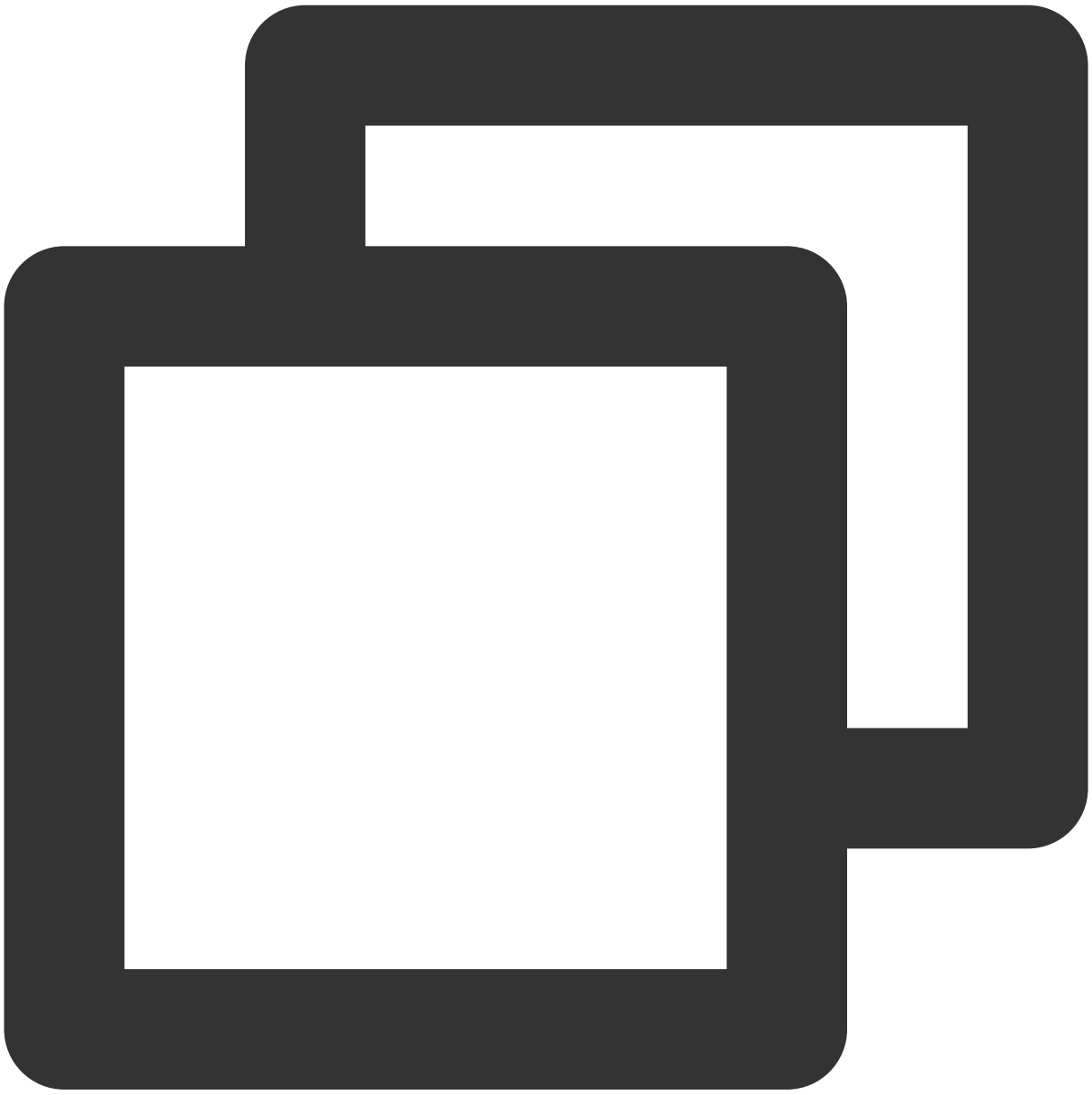
썸네일 설정

1. COS 버킷에 있는 썸네일의 객체 주소를 가져옵니다.

주의 :

CI의 Intelligent Thumbnail 기능은 최적의 프레임을 추출하여 썸네일을 생성하여 비디오 콘텐츠를 더 매력적으로 만들 수 있습니다.

2. 플레이어를 초기화하고 썸네일 이미지를 설정합니다.



```
<video
  id="my-video"
  class="video-js"
  controls
```

```
    preload="auto"
    width="100%"
    height="100%"
    data-setup="{}"
    poster="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/poster.png"
  >
    <source
      src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4"
      type="video/mp4"
    />
  </video>
```

코드 샘플 가져오기:

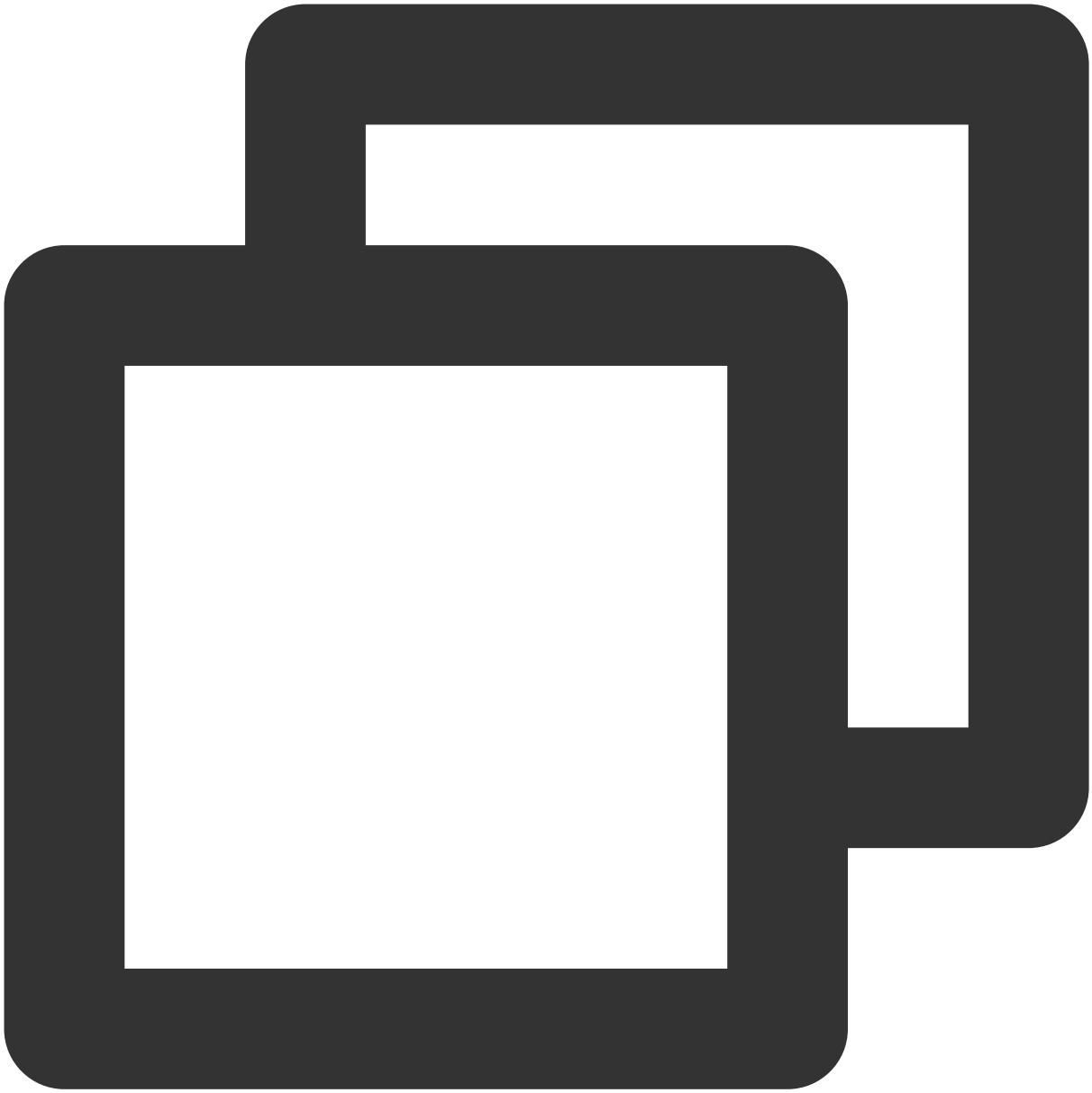
썸네일 구성 샘플 코드

HLS 암호화 비디오 재생

CI는 동영상 콘텐츠의 보안을 확보하고 동영상의 무단 다운로드 및 배포를 방지하기 위해 개인이 읽을 수 있는 파일보다 안전한 HLS 동영상 콘텐츠를 암호화하는 기능을 제공합니다. 암호화된 비디오는 재생 액세스 권한이 없는 사용자에게 배포할 수 없습니다. 다운로드하더라도 여전히 암호화되어 악의적으로 재배포할 수 없습니다. 이렇게 하면 비디오 저작권이 침해되는 것을 방지할 수 있습니다.

작업 순서는 다음과 같습니다.

1. HLS 암호화 비디오 재생 및 COS 오디오/비디오 실습 | 비디오 암호화를 참고하여 암호화된 비디오를 생성합니다.
2. 플레이어를 초기화하고 비디오 객체 주소를 전달합니다.



```
<source
  src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8"
  type="application/x-mpegURL"
/>
```

코드 샘플 가져오기:

HLS 암호화 재생 샘플 코드

데이터 다이렉트 업로드
Web의 다이렉트 업로드 사례

최종 업데이트 날짜: : 2024-06-24 16:44:04

소개

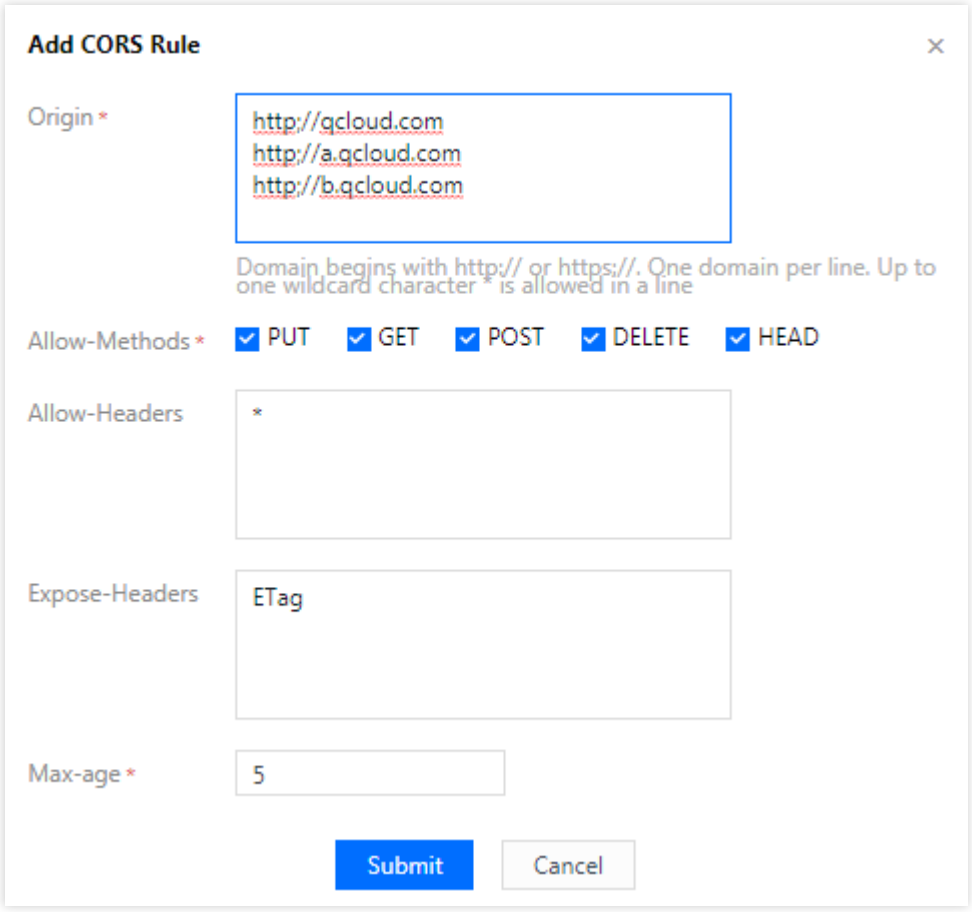
본 문서는 SDK에 종속되지 않고 간단한 코드를 사용하여 웹 페이지(Web)에서 파일을 COS의 버킷에 다이렉트 업로드하는 방법을 소개합니다.

주의 :

본 문서의 내용은 XML 버전의 API를 기반으로 합니다.

전제 조건

1. COS 콘솔에 로그인한 후 버킷을 생성하고 Bucket(버킷 이름)과 Region(리전 이름)을 획득합니다. 자세한 내용은 버킷 생성 문서를 참조하십시오.
2. 버킷 상세 페이지로 이동하여 **보안 관리** 탭을 클릭합니다. 페이지를 아래로 내려 **크로스 도메인 액세스 CORS 설정** 항목을 찾아 **규칙 추가**를 클릭한 후, 다음 이미지와 같이 설정합니다. 자세한 내용은 크로스 도메인 액세스 설정 문서를 참조하십시오.



3. CAM 콘솔에 로그인한 뒤 프로젝트의 SecretId와 SecretKey를 획득합니다.

실행 순서

주의 :

정식 배포에 앞서 서버에 본인 웹 사이트의 권한 인증 기능을 한 레이어 추가하십시오.

임시 키 획득 및 서명 계산

보안을 위해 서명에 임시 키를 사용합니다. 서버에 임시 키 서비스 구축에 대한 자세한 내용은 PHP 예시 및 Nodejs 예시를 참조하십시오.

다른 언어를 사용하거나 직접 실행하는 경우 다음 절차를 참고할 수 있습니다.

1. 서버에서 임시 키를 획득합니다. 서버는 먼저 고정 키의 SecretId와 SecretKey를 사용해 STS 서비스에서 tmpSecretId, tmpSecretKey, sessionToken과 같은 임시 키를 획득합니다. 자세한 방법은 임시 키 생성 및 사용 가이드 또는 cos-sts-sdk 문서를 참조하십시오.
2. 프런트 엔드에서 tmpSecretId, tmpSecretKey, method, pathname을 통해 서명을 계산합니다. cos-auth.js 구문을 참고 및 사용해 서명을 계산하며, 비즈니스에 필요한 경우 백그라운드에서 서명을 계산할 수도 있습니다.
3. PutObject 인터페이스를 사용해 파일을 업로드하는 경우, 요청 전송 시 계산된 서명과 sessionToken을 각각 header의 authorization과 x-cos-security-token 필드에 넣습니다.

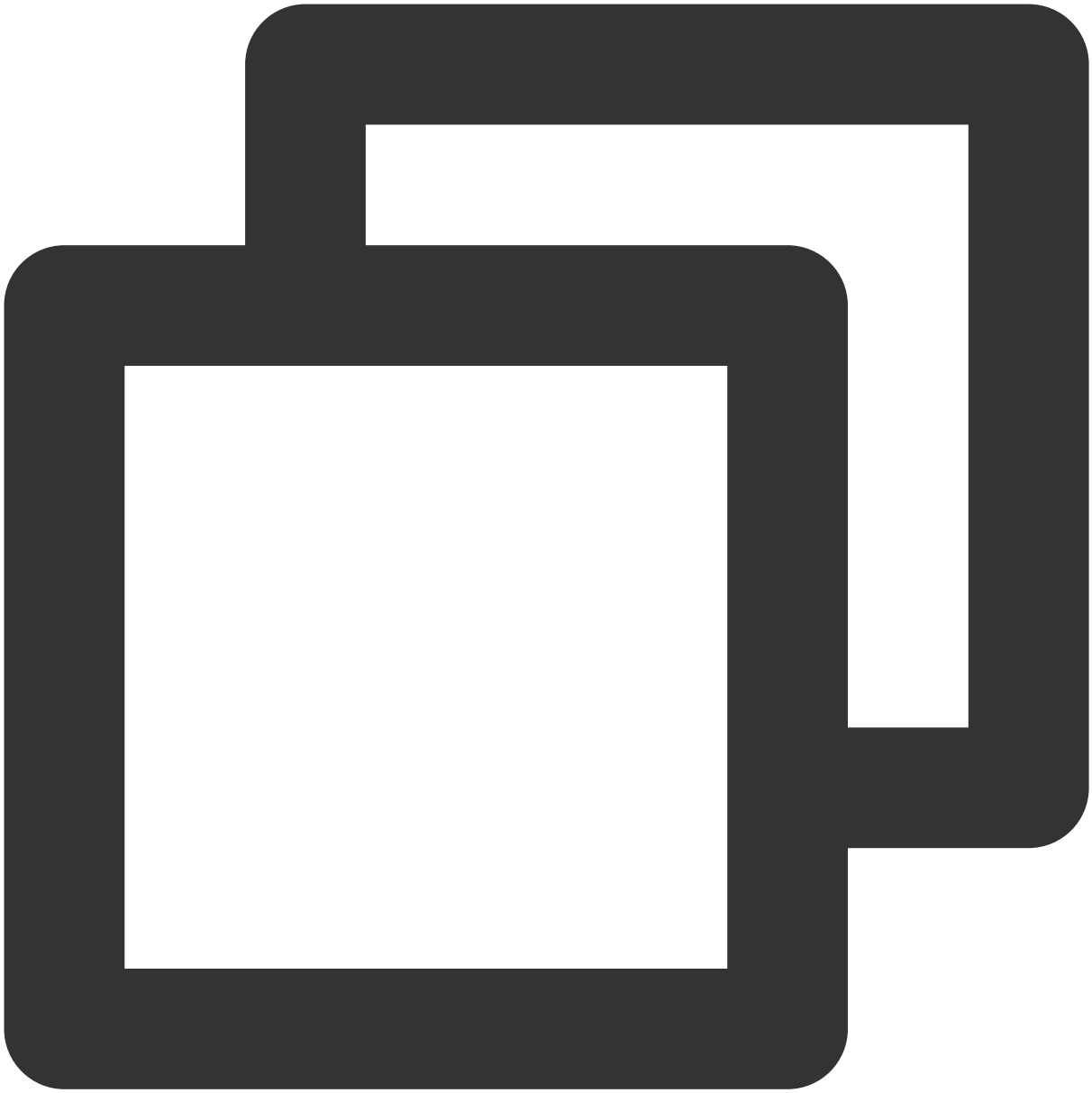
PostObject 인터페이스를 사용해 파일을 업로드하는 경우, 요청 전송 시 계산된 서명과 sessionToken을 각각 테이블의 Signature와 x-cos-security-token 필드에 넣습니다.

프런트 엔드 업로드

방법 A: AJAX를 사용하여 업로드

AJAX 업로드 시에는 브라우저에서 기본적으로 HTML5 특성을 지원해야 합니다. 해당 방법은 PUT Object 문서를 사용하며, 작업 가이드는 다음과 같습니다.

1. 전제 조건 순서에 따라 버킷에 관련 정보를 설정합니다.
2. `test.html` 파일을 생성하고, 아래 코드에서 Bucket과 Region 정보를 수정하여 `test.html` 파일에 복사합니다.
3. 백그라운드의 서명 서비스를 배포하고, `test.html` 의 서명 서비스 주소를 수정합니다.
4. `test.html` 을 Web 서버에 올리고, 브라우저로 페이지에 액세스하여 파일 업로드 기능을 테스트합니다.



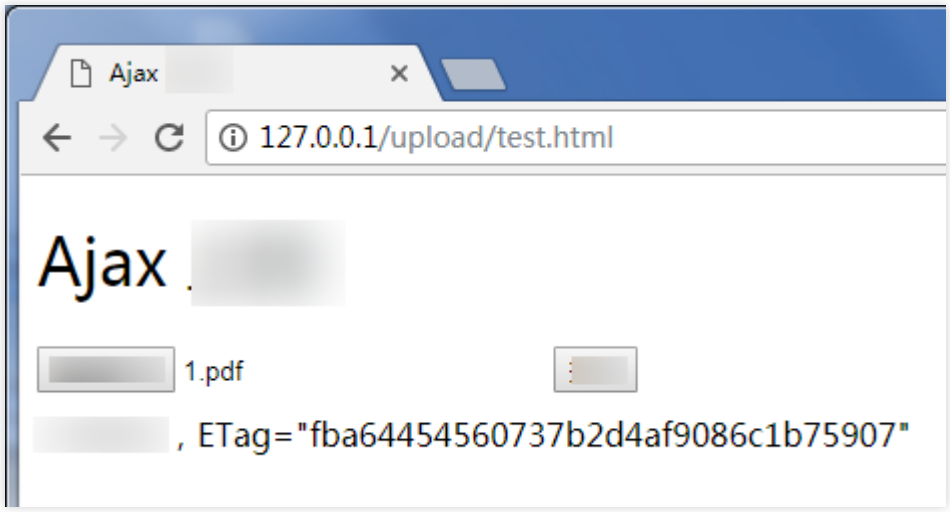
```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Ajax Put 업로드</title>
    <style>
        h1, h2 {
            font-weight: normal;
        }

        #msg {
```

```
            margin-top: 10px;
        }
    </style>
</head>
<body>

<h1>Ajax Put 업로드</h1>

<input id="fileSelector" type="file">
<input id="submitBtn" type="submit">

<div id="msg"></div>

<script src="https://unpkg.com/cos-js-sdk-v5/demo/common/cos-auth.min.js"></script>
<script>
    (function () {
        // 요청 시 사용한 매개변수
        var Bucket = 'examplebucket-1250000000';
        var Region = 'ap-guangzhou';
        var protocol = location.protocol === 'https:' ? 'https:' : 'http:';
        var prefix = protocol + '//' + Bucket + '.cos.' + Region + '.myqcloud.com/'

        // 더 많은 문자열 인코딩의 url encode 포맷
        var camSafeUrlEncode = function (str) {
            return encodeURIComponent(str)
                .replace(/!/g, '%21')
                .replace(/'/g, '%27')
                .replace(/\\(/g, '%28')
                .replace(/\\)/g, '%29')
                .replace(/\\*/g, '%2A');
        };

        // 서명 계산
        var getAuthorization = function (options, callback) {
            // var url = 'http://127.0.0.1:3000/sts-auth' +
            var url = '../server/sts.php';
            var xhr = new XMLHttpRequest();
            xhr.open('GET', url, true);
            xhr.onload = function (e) {
                var credentials;
                try {
                    credentials = (new Function('return ' + xhr.responseText))().cr
                } catch (e) {}
                if (credentials) {
                    callback(null, {
                        SecurityToken: credentials.sessionToken,
                        Authorization: CosAuth({
```

```
                        SecretId: credentials.tmpSecretId,
                        SecretKey: credentials.tmpSecretKey,
                        Method: options.Method,
                        Pathname: options.Pathname,
                    })
                });
            } else {
                console.error(xhr.responseText);
                callback('서명 획득 오류');
            }
        };
        xhr.onerror = function (e) {
            callback('서명 획득 오류');
        };
        xhr.send();
    };

    // 파일 업로드
    var uploadFile = function (file, callback) {
        var Key = 'dir/' + file.name; // 여기에서 업로드 디렉터리 및 파일 이름 지정
        getAuthorization({Method: 'PUT', Pathname: '/' + Key}, function (err, i

            if (err) {
                alert(err);
                return;
            }

            var auth = info.Authorization;
            var SecurityToken = info.SecurityToken;
            var url = prefix + camSafeUrlEncode(Key).replace(/%2F/g, '/');
            var xhr = new XMLHttpRequest();
            xhr.open('PUT', url, true);
            xhr.setRequestHeader('Authorization', auth);
            SecurityToken && xhr.setRequestHeader('x-cos-security-token', Secur
            xhr.upload.onprogress = function (e) {
                console.log('업로드 진행률' + (Math.round(e.loaded / e.total * 10
            };
            xhr.onload = function () {
                if (/^2\\d\\d$/.test('' + xhr.status)) {
                    var ETag = xhr.getResponseHeader('etag');
                    callback(null, {url: url, ETag: ETag});
                } else {
                    callback(Key + ' 파일 업로드에 실패했습니다. 상태 코드: ' + xhr.s
                }
            };
            xhr.onerror = function () {
                callback(Key + ' 파일 업로드에 실패했습니다. CORS 크로스 도메인 규칙을
```

```
                };
                xhr.send(file);
            });
        };

        // 테이블 제출 수신
        document.getElementById('submitBtn').onclick = function (e) {
            var file = document.getElementById('fileSelector').files[0];
            if (!file) {
                document.getElementById('msg').innerText = '업로드할 파일을 선택하지 않
                return;
            }
            file && uploadFile(file, function (err, data) {
                console.log(err || data);
                document.getElementById('msg').innerText = err ? err : ('업로드 성공,
            });
        };
    })();
</script>

</body>
</html>
```

실행 결과는 다음 이미지와 같습니다.

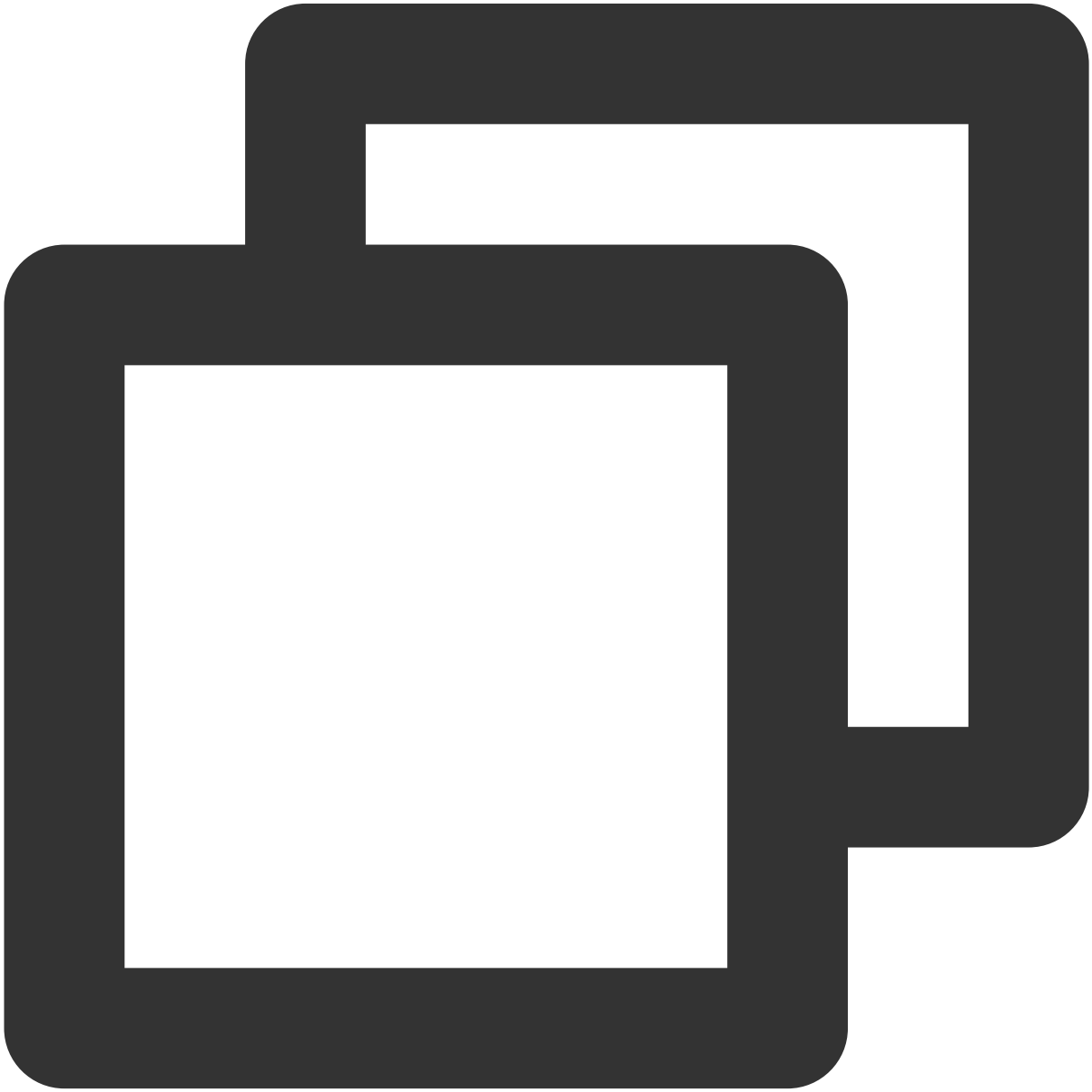


방법 B: 폼(Form)을 사용하여 업로드

폼(Form) 업로드는 낮은 버전의 브라우저에서의 업로드(예를 들어 IE8)를 지원하며, 해당 방법은 POST Object 인터페이스를 사용해야 합니다. 작업 가이드는 다음과 같습니다.

1. 전제 조건 순서에 따라 버킷을 준비합니다.

2. `test.html` 파일을 생성하고, 아래 코드에서 Bucket과 Region 정보를 수정하여 `test.html` 파일에 복사합니다.
3. 백그라운드의 서명 서비스를 배포하고, `test.html` 의 서명 서비스 주소를 수정합니다.
4. `test.html` 과 동일한 디렉터리에 비어 있는 `empty.html` 을 생성하여 업로드 성공 시 리디렉션하는 데 사용합니다.
5. `test.html` 과 `empty.html` 을 Web 서버에 올리고, 브라우저로 페이지에 액세스하여 파일 업로드 기능을 테스트합니다.



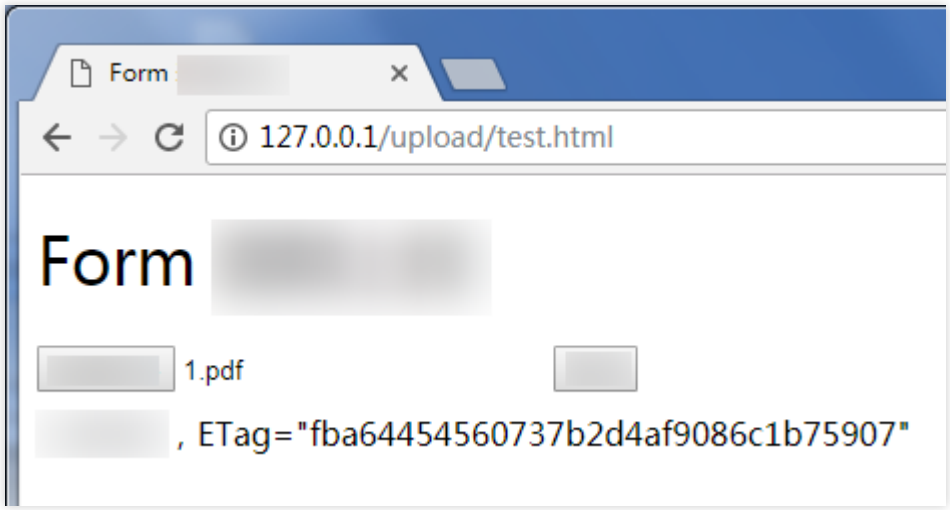
```
<!doctype html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <title>Form 테이블 간편 업로드</title>
    <style>h1, h2 {font-weight: normal;}#msg {margin-top:10px;}</style>
</head>
<body>

<h1>테이블 간편 업로드(IE8 호환)</h1>
<div>최저 버전은 IE6까지 호환해 업로드하며, onprogress는 미지원</div>

<form id="form" target="submitTarget" action="" method="post" enctype="multipart/fo
    <input id="name" name="name" type="hidden" value="">
    <input name="success_action_status" type="hidden" value="200">
    <input id="success_action_redirect" name="success_action_redirect" type="hidden
    <input id="key" name="key" type="hidden" value="">
    <input id="Signature" name="Signature" type="hidden" value="">
    <input name="Content-Type" type="hidden" value="">
    <input id="x-cos-security-token" name="x-cos-security-token" type="hidden" valu

    <!-- 파일 콘텐츠가 너무 길어 서명 판단 및 인증에 영향이 미치지 않도록 file 필드는 테이블의
    <input id="fileSelector" name="file" type="file">
    <input id="submitBtn" type="button" value="제출">
</form>
<iframe id="submitTarget" name="submitTarget" style="display:none;" frameborder="0"

<div id="msg"></div>

<script src="https://unpkg.com/cos-js-sdk-v5/demo/common/cos-auth.min.js"></script>
<script>
    (function () {

        // 요청 시 사용한 매개변수
        var Bucket = 'examplebucket-1250000000';
        var Region = 'ap-guangzhou';
        var protocol = location.protocol === 'https:' ? 'https:' : 'http:';
        var prefix = protocol + '//' + Bucket + '.cos.' + Region + '.myqcloud.com/'
        var form = document.getElementById('form');
        form.action = prefix;

        // 더 많은 문자열 인코딩의 url encode 포맷
        var camSafeUrlEncode = function (str) {
            return encodeURIComponent(str)
                .replace(/!/g, '%21')
                .replace(/'/g, '%27')
                .replace(/\(/g, '%28')
                .replace(/\)/g, '%29')
                .replace(/\*/g, '%2A');
```

```
};

// 서명 계산
var getAuthorization = function (options, callback) {
    // var url = 'http://127.0.0.1:3000/sts' +
    var url = '../server/sts.php';
    var xhr = new XMLHttpRequest();
    xhr.open('GET', url, true);
    xhr.onreadystatechange = function (e) {
        if (xhr.readyState === 4) {
            if (/^2\\d\\d$/.test('' + xhr.status)) {
                var credentials;
                try {
                    credentials = (new Function('return ' + xhr.responseTex
                } catch (e) {}
                if (credentials) {
                    callback(null, {
                        SecurityToken: credentials.sessionToken,
                        Authorization: CosAuth({
                            SecretId: credentials.tmpSecretId,
                            SecretKey: credentials.tmpSecretKey,
                            Method: options.Method,
                            Pathname: options.Pathname,
                        })
                    });
                } else {
                    console.error(xhr.responseText);
                    callback('서명 획득 오류');
                }
            } else {
                callback('서명 획득 오류');
            }
        }
    };
    xhr.send();
};

// 업로드 완료 수신
var Key;
var submitTarget = document.getElementById('submitTarget');
var showMessage = function (err, data) {
    console.log(err || data);
    document.getElementById('msg').innerText = err ? err : ('업로드 성공, ETa
};
submitTarget.onload = function () {
    var search;
    try {
```

```
                search = submitTarget.contentWindow.location.search.substr(1);
            } catch (e) {
                showMessage(Key + ' 파일 업로드에 실패했습니다.');
            }
            if (search) {
                var items = search.split('&');
                var i, arr, data = {};
                for (i = 0; i < items.length; i++) {
                    arr = items[i].split('=');
                    data[arr[0]] = decodeURIComponent(arr[1] || '');
                }
                showMessage(null, {url: prefix + camSafeUrlEncode(Key).replace(/%2F
            } else {
            }
        };

        // 업로드 시작
        document.getElementById('submitBtn').onclick = function (e) {
            var filePath = document.getElementById('fileSelector').value;
            if (!filePath) {
                document.getElementById('msg').innerText = '업로드할 파일을 선택하지 않
                return;
            }
            Key = 'dir/' + filePath.match(/[\\\\\\/]?([^\\\\\\/]+)$/)[1]; // 여기에서
            getAuthorization({Method: 'POST', Pathname: '/'}, function (err, AuthDa
                // 현재 디렉터리에 빈 empty.html를 넣어 인터페이스에서 업로드 완료 시 리디렉
                document.getElementById('success_action_redirect').value = location
                document.getElementById('key').value = Key;
                document.getElementById('Signature').value = AuthData.Authorization
                document.getElementById('x-cos-security-token').value = AuthData.Se
                form.submit();
            });
        };
    })();
</script>
</body>
</html>
```

실행 결과는 다음 이미지와 같습니다.



관련 문서

더 다양한 인터페이스 호출이 필요한 경우 다음 JavaScript SDK 문서를 참조하십시오.

JavaScript SDK

미니프로그램 다이렉트 업로드 사례

최종 업데이트 날짜: : 2024-06-24 16:44:04

소개

본 문서에서는 SDK를 사용하지 않고 미니프로그램을 통해 직접 COS(Cloud Object Storage) 버킷에 파일을 업로드하는 간단한 코드를 사용하는 방법을 설명합니다.

주의 :

본 문서의 내용은 XML 버전의 API를 기반으로 합니다.

전제 조건

1. COS 콘솔에 로그인하고, 버킷을 생성하여 BucketName(버킷 이름)과 Region(리전 이름)을 구성합니다. 자세한 내용은 버킷 생성 문서를 참고하십시오.
2. CAM 콘솔에 로그인하고 API 키 관리 페이지에서 프로젝트의 SecretId 및 SecretKey를 가져옵니다.

실행 순서

1. WeChat 미니프로그램 얼로우리스트 구성

미니프로그램은 COS가 WeChat의 공개 플랫폼에 로그인하고 '개발'->'개발 설정'에서 도메인 이름 얼로우리스트를 구성해야 한다고 요청합니다. SDK는 wx.uploadFile 및 wx.request의 두 가지 API를 사용합니다.
cos.postObject 메소드의 경우 wx.uploadFile을 사용하여 요청이 전송됩니다.
다른 방법의 경우 wx.request를 사용하여 요청을 보냅니다.
해당 얼로우리스트에서 COS 도메인 이름을 구성해야 합니다. 허용되는 도메인 이름에는 두 가지 형식이 있습니다.
버킷이 하나만 사용되는 경우 Bucket 도메인 이름을 얼로우리스트 도메인 이름으로 구성할 수 있습니다(예시: `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com`).
여러 버킷을 사용하는 경우 pathname에 bucket을 배치하여 접미사 COS 요청을 사용하도록 선택할 수 있습니다. 이 경우 `cos.ap-guangzhou.myqcloud.com` 과 같이 리전 도메인 이름을 얼로우리스트로 구성해야 합니다.

2. 임시 키 획득 및 서명 계산

보안을 위해 서명에 임시 키를 사용합니다. 서버에 임시 키 서비스 구축에 대한 자세한 내용은 PHP 예시 및 Nodejs 예시를 참고하십시오.
다른 언어를 사용하거나 직접 구현하려면 다음 단계를 따르십시오.
(1) 서버에서 임시 키를 획득합니다. 서버는 먼저 고정 키의 SecretId와 SecretKey를 사용해 STS 서비스에서

tmpSecretId, tmpSecretKey, sessionToken과 같은 임시 키를 획득합니다. 자세한 방법은 임시 키 생성 및 사용 가이드 또는 cos-sts-sdk 문서를 참고하십시오.

주의 :

요청이 put인지 post인지에 따라 "name/cos:PutObject" 또는 "name/cos:PostObject"를 허용하도록 STS의 policy action을 추가해야 합니다.

(2) 프런트 엔드에서 tmpSecretId, tmpSecretKey, method, pathname을 통해 서명을 계산합니다. cos-auth.js 구문을 참고 및 사용해 서명을 계산하며, 비즈니스에 필요한 경우 백그라운드에서 서명을 계산할 수도 있습니다.

(3) 계산된 서명과 sessionToken을 넣고,

post 요청의 formData의 Signature 및 x-cos-security-token 필드에 업로드 요청을 COS API로 보냅니다.

put 요청의 headers의 Signature 및 x-cos-security-token 필드에 업로드 요청을 COS API로 보냅니다.

주의 :

정식 배포에 앞서 서버에 본인 웹 사이트의 권한 인증 기능을 추가하십시오.

3. 접미사 요청

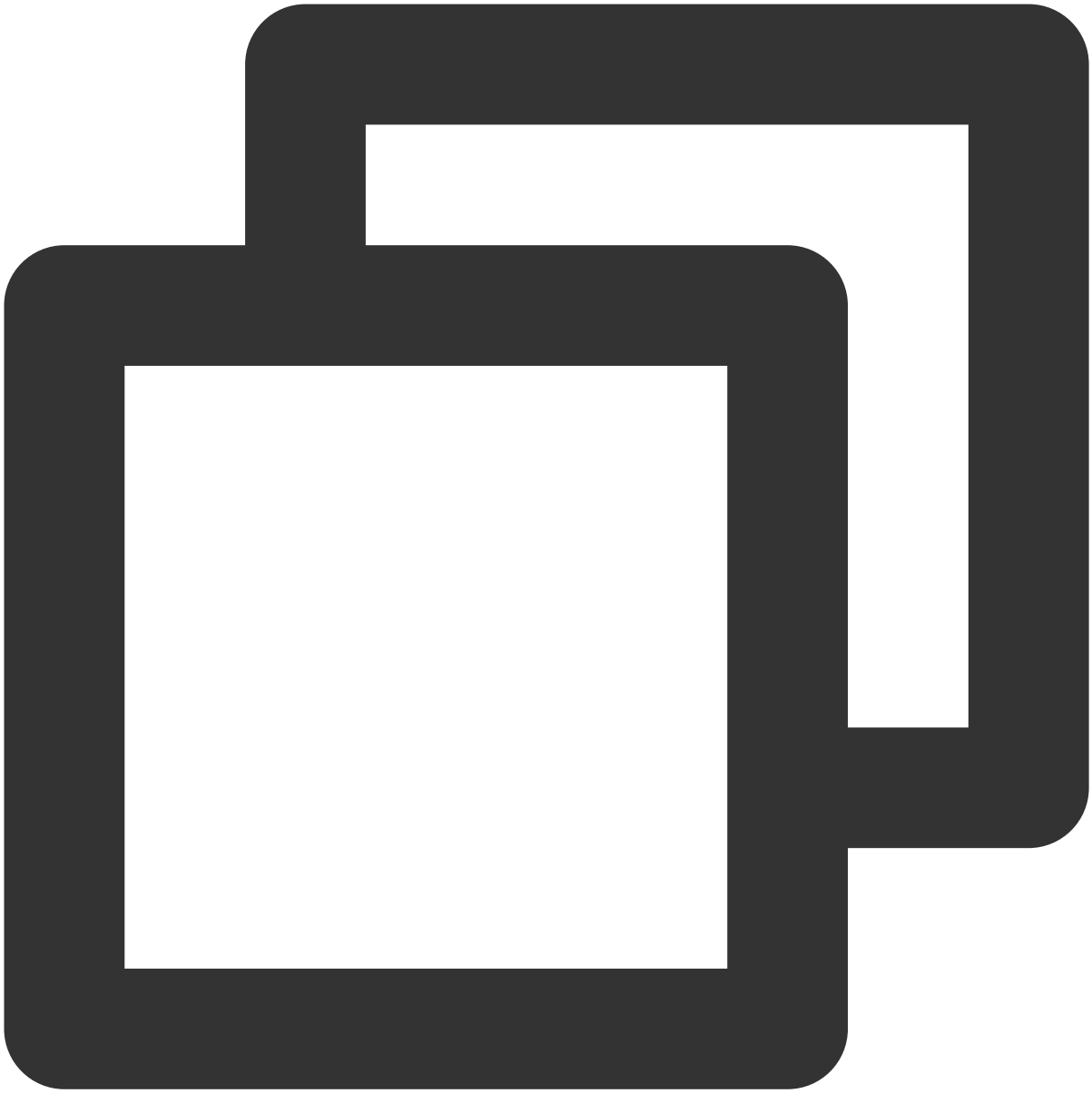
COS API의 일반적인 요청 형식은 `POST http://examplebucket-1250000000.cos.ap-beijing.myqcloud.com/` 와 비슷합니다. 요청한 도메인 이름은 버킷 도메인 이름입니다. 이와 같이 미니프로그램에서 하나 이상의 버킷을 사용하는 경우 버킷 도메인 이름을 얼로우리스트 도메인 이름으로 설정해야 합니다. 솔루션은 다음과 같습니다.

COS는 접미사 요청 형식 `POST http://cos.ap-beijing.myqcloud.com/examplebucket-1250000000/` 를 제공합니다. 요청된 도메인 이름은 리전 도메인 이름이며 요청한 경로에 버킷 이름이 위치합니다. 미니프로그램에서 동일한 리전에서 여러 버킷을 사용하는 경우 하나의 도메인 이름 `cos.ap-beijing.myqcloud.com` 만 얼로우리스트 도메인 이름으로 구성하면 됩니다.

접미사 요청 형식은 서명할 때 사용되는 경로 앞에 버킷 이름을 붙여야 합니다(예시: `/examplebucket-1250000000/`).

4. 다이렉트 업로드 예시 코드

다음 코드는 PUT Object API(사용 권장) 및 POST Object API를 모두 예시한 것이며 작업 가이드는 다음과 같습니다.



```
var CosAuth = require('./cos-auth'); // 여기에서 다운로드 주소가 https://unpkg.com/cos-

var Bucket = 'examplebucket-1250000000';
var Region = 'ap-shanghai';
var ForcePathStyle = false; // 서명 계산 및 도메인 이름 얼로우리스트 구성과 관련된 접미사 시

var uploadFile = function () {

    // 요청 시 사용한 매개변수
    var prefix = 'https://' + Bucket + '.cos.' + Region + '.myqcloud.com/';
    if (ForcePathStyle) {
```

```
        // 접미사 요청의 도메인 이름은 버킷 도메인 이름 대신 리전 도메인 이름을 사용하여 서명되
        prefix = 'https://cos.' + Region + '.myqcloud.com/' + Bucket + '/';
    }

    // 더 많은 문자열 코드의 url encode 형식
    var camSafeUrlEncode = function (str) {
        return encodeURIComponent(str)
            .replace(/!/g, '%21')
            .replace(/'/g, '%27')
            .replace(/\\(/g, '%28')
            .replace(/\\)/g, '%29')
            .replace(/\\*/g, '%2A');
    };

    // 임시 키 획득
    var stsCache;
    var getCredentials = function (callback) {
        if (stsCache && Date.now() / 1000 + 30 < stsCache.expiredTime) {
            callback(data.credentials);
            return;
        }
        wx.request({
            method: 'GET',
            url: 'https://example.com/sts.php', // 서버측 서명, 자세한 내용은 위의 임시
            dataType: 'json',
            success: function (result) {
                var data = result.data;
                var credentials = data.credentials;
                if (credentials) {
                    stsCache = data
                } else {
                    wx.showModal({title: '임시 키 가져오기 실패', content: JSON.string
                }
                callback(stsCache && stsCache.credentials);
            },
            error: function (err) {
                wx.showModal({title: '임시 키 가져오기 실패', content: JSON.stringify(
            }
        });
    };

    // 서명 계산
    var getAuthorization = function (options, callback) {
        getCredentials(function (credentials) {
            callback({
                XCosSecurityToken: credentials.sessionToken,
                Authorization: CosAuth({
```

```
                    SecretId: credentials.tmpSecretId,
                    SecretKey: credentials.tmpSecretKey,
                    Method: options.Method,
                    Pathname: options.Pathname,
                })
            });
        });
    };

    // post 파일 업로드
    var postFile = function (filePath) {
        var Key = filePath.substr(filePath.lastIndexOf('/') + 1); // 여기에 업로드할 
        var signPathname = '/'; // PostObject API의 경우 Key가 전송을 위해 Body에 배치도
        if (ForcePathStyle) {
              // 접미사 요청의 서명에 사용된 경로에는 버킷의 이름이 포함되어야 하며, 자세한 내
            signPathname = '/' + Bucket + '/';
        }
        getAuthorization({Method: 'POST', Pathname: signPathname}, function (AuthDa
            var requestTask = wx.uploadFile({
                url: prefix,
                name: 'file',
                filePath: filePath,
                formData: {
                    'key': Key,
                    'success_action_status': 200,
                    'Signature': AuthData.Authorization,
                    'x-cos-security-token': AuthData.XCosSecurityToken,
                    'Content-Type': '',
                },
                success: function (res) {
                    var url = prefix + camSafeUrlEncode(Key).replace(/%2F/g, '/');
                    console.log(res.statusCode);
                    console.log(url);
                    if (/^2\\d\\d$/.test('' + res.statusCode)) {
                        wx.showModal({title: '업로드 성공', content: url, showCancel:
                    } else {
                        wx.showModal({title: '업로드 실패', content: JSON.stringify(r
                    }
                },
                fail: function (res) {
                    wx.showModal({title: '업로드 실패', content: JSON.stringify(res),
                }
            });
            requestTask.onProgressUpdate(function (res) {
                console.log('진행률:', res);
            });
        });
```

```
};

// put 파일 업로드
var putFile = function (filePath) {
    var Key = filePath.substr(filePath.lastIndexOf('/') + 1); // 여기에 업로드할 
    var signPathname = '/' + Key; // PutObject API Key는 url 전송에 위치하므로 요청
    if (ForcePathStyle) {
          // 접미사 요청의 서명에 사용된 경로에는 버킷의 이름이 포함되어야 하며, 자세한 내
        signPathname = '/' + Bucket + '/' + Key;
    }
    getAuthorization({Method: 'PUT', Pathname: signPathname}, function (AuthDat
      // put 요청은 임시 파일 경로에서 파일 콘텐츠를 읽어야 함
      var wxfs = wx.getFileSystemManager();
      wxfs.readFile({
        filePath: filePath,
        success: function (fileRes) {
          var requestTask = wx.request({
            url: prefix + signPathname.substr(signPathname.lastIndexOf('/') + 1
            method: 'PUT',
            header: {
              'Authorization': AuthData.Authorization,
              'x-cos-security-token': AuthData.XCosSecurityToken,
            },
            data: fileRes.data,
            success: function success(res) {
              var url = prefix + camSafeUrlEncode(Key).replace(/%2F/g, '/');
              if (res.statusCode === 200) {
                  wx.showModal({title: '업로드 성공', content: url, showCancel: f
              } else {
                  wx.showModal({title: '업로드 실패', content: JSON.stringify(res
              }
              console.log(res.statusCode);
              console.log(url);
            },
            fail: function fail(res) {
              wx.showModal({title: '업로드 실패', content: JSON.stringify(res), s
            },
          });
          requestTask.onProgressUpdate(function (res) {
              console.log('진행 중:', res);
          });
        },
      });
    });
};

// 파일 선택
```

```
    wx.chooseImage({
        count: 1, // 기본값: 9
        sizeType: ['original'], // 원본 이미지 또는 압축 이미지를 지정할 수 있습니다. 여기
        sourceType: ['album', 'camera'], // 출처를 갤러리 또는 카메라로 지정할 수 있습니다
        success: function (res) {
            putFile(res.tempFiles[0].path); // put 요청 업로드, 사용 권장
            // postFile(res.tempFiles[0].path); // post 요청 업로드
        }
    })
};
```

관련 문서

미니프로그램 SDK를 사용해야 하는 경우 미니프로그램 SDK 시작하기 문서를 참고하십시오.

모바일 애플리케이션 다이렉트 업로드 사례

최종 업데이트 날짜: : 2024-06-24 16:44:04

소개

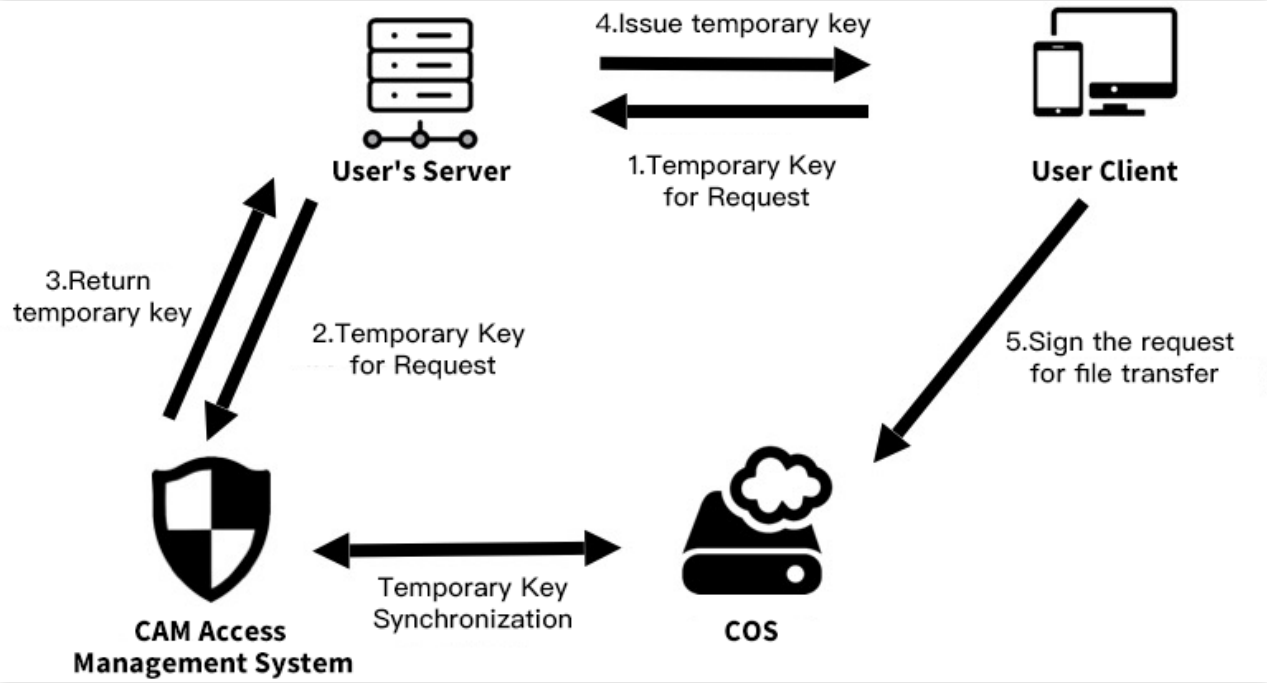
본 문서에서는 Tencent Cloud COS를 기반으로 모바일 애플리케이션에서의 파일 다이렉트 업로드 기능을 신속히 구현하는 방법을 소개합니다. 서버에서 복잡한 조작을 할 필요 없이 액세스 키를 생성 및 관리하는 것만으로도 파일 데이터를 Tencent Cloud COS에 저장할 수 있습니다.

아키텍처 설명

클라이언트 애플리케이션의 경우, 영구 키를 클라이언트 코드에 입력하면 키 정보가 쉽게 노출될 수 있으며, 사용자 액세스 권한 제어에도 불편이 따릅니다. 영구 키의 유출 방지를 위해 요청 시 임시 키를 사용하고, App에 스토리지 리소스에 대한 임시 액세스 권한을 부여할 것을 권장합니다. 키의 유효기간은 직접 지정할 수 있으며, 기간이 만료되면 키는 자동 무효화됩니다.
COS의 모바일 SDK(Android/IOS)는 임시 키로 권한 부여 요청을 할 수 있습니다. 백그라운드에서 임시 키 서비스를 구축하면 단말 COS로 권한 부여를 위한 요청이 매끄럽게 전달됩니다.

아키텍처

아키텍처 예시는 다음 이미지를 참조하십시오.



다음을 참고하십시오.
사용자 클라이언트: 사용자 휴대폰의 App입니다.
COS: Tencent Cloud COS, App에서 업로드한 데이터를 보관합니다.
CAM: Tencent Cloud CAM, COS 임시 키 생성에 쓰입니다.
사용자 서버: 사용자의 백그라운드 서버로, 임시 키를 발급하여 애플리케이션 App으로 반환합니다.

전제 조건

1. 버킷을 생성합니다.
COS 콘솔에서 버킷을 생성합니다. 요구사항에 따라 버킷 권한을 개인 읽기/쓰기 또는 공개 읽기/개인 쓰기로 설정할 수 있습니다. 자세한 단계에 관한 내용은 버킷 생성과 액세스 권한 설정을 참조하십시오.
2. 영구 키를 받습니다.
임시 키는 영구 키를 통해 생성해야 합니다. API Keys로 이동하여 SecretId, SecretKey를 획득한 뒤 계정 정보로 이동하여 APPID를 부여받습니다.

실행 순서

임시 키 서비스 구축

보안 강화를 위해 임시 키로 서명하려면 서버에 임시 키 서비스를 구축하고, API 인터페이스를 클라이언트로 전달해야 합니다. 서비스 구축 단계에 관한 자세한 내용은 임시 키 생성 및 사용 가이드를 참조하십시오.

주의 :

정식 배포에 앞서 서버에 본인 웹 사이트의 권한 인증 기능을 추가하십시오.

적합한 권한 선택

최소 권한의 원칙과 본인의 요구사항에 따라 Policy를 적용하여 임시 키의 권한 범위를 제어할 것을 권장합니다. 서버에서 전달한 완전한 읽기/쓰기 권한을 가진 키가 해킹에 노출되면 다른 사용자의 데이터가 유출될 위험이 있습니다. 설정에 관한 자세한 내용은 임시 키 생성 및 사용 가이드를 참조하십시오.

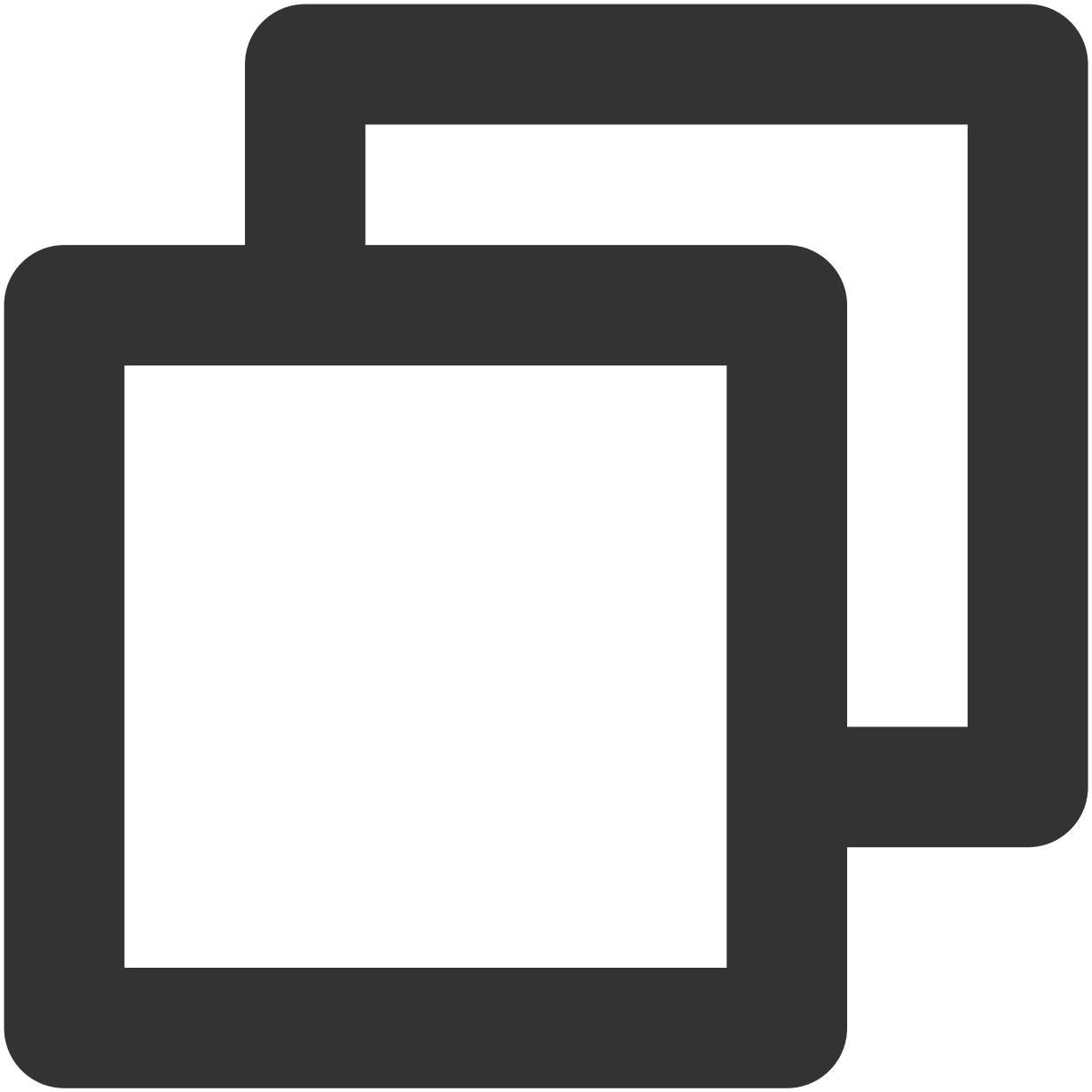
SDK로 라이선스 서비스 액세스

Android

임시 키 서비스를 구축한 뒤 SDK로 라이선스 서비스에 액세스해야 합니다. SDK는 요청한 동시 접속 수를 제어하고, 유효한 키를 로컬에 캐싱합니다. 키가 무효화되면 재요청을 하기 때문에 키를 따로 관리하지 않아도 됩니다.

표준 응답 본문에 의한 권한 부여

STS SDK에서 획득한 JSON 데이터를 임시 키 서비스의 응답 본문으로 할 경우(cossign은 해당 방법 적용), 다음 코드로 COS SDK의 권한 부여 클래스를 생성할 수 있습니다.



```
import android.content.Context;
import com.tencent.cos.xml.*;
import com.tencent.qcloud.core.auth.*;
import com.tencent.qcloud.core.common.*;
import com.tencent.qcloud.core.http.*;
import java.net.*;


Context context = ...;
CosXmlServiceConfig cosXmlServiceConfig = ...;
```

```
/**
 * 라이선스 서비스의 url 주소 획득
 */
URL url = null; // 백그라운드의 라이선스 서비스 url 주소
try {
    url = new URL("your_auth_server_url");
} catch (MalformedURLException e) {
    e.printStackTrace();
}

/**
 * {@link QCloudCredentialProvider} 객체 초기화로 SDK에 임시 키 부여
 */
QCloudCredentialProvider credentialProvider = new SessionCredentialProvider(new Htt
                .url(url)
                .method("GET")
                .build());

CosXmlService cosXmlService = new CosXmlService(context, cosXmlServiceConfig, crede
```

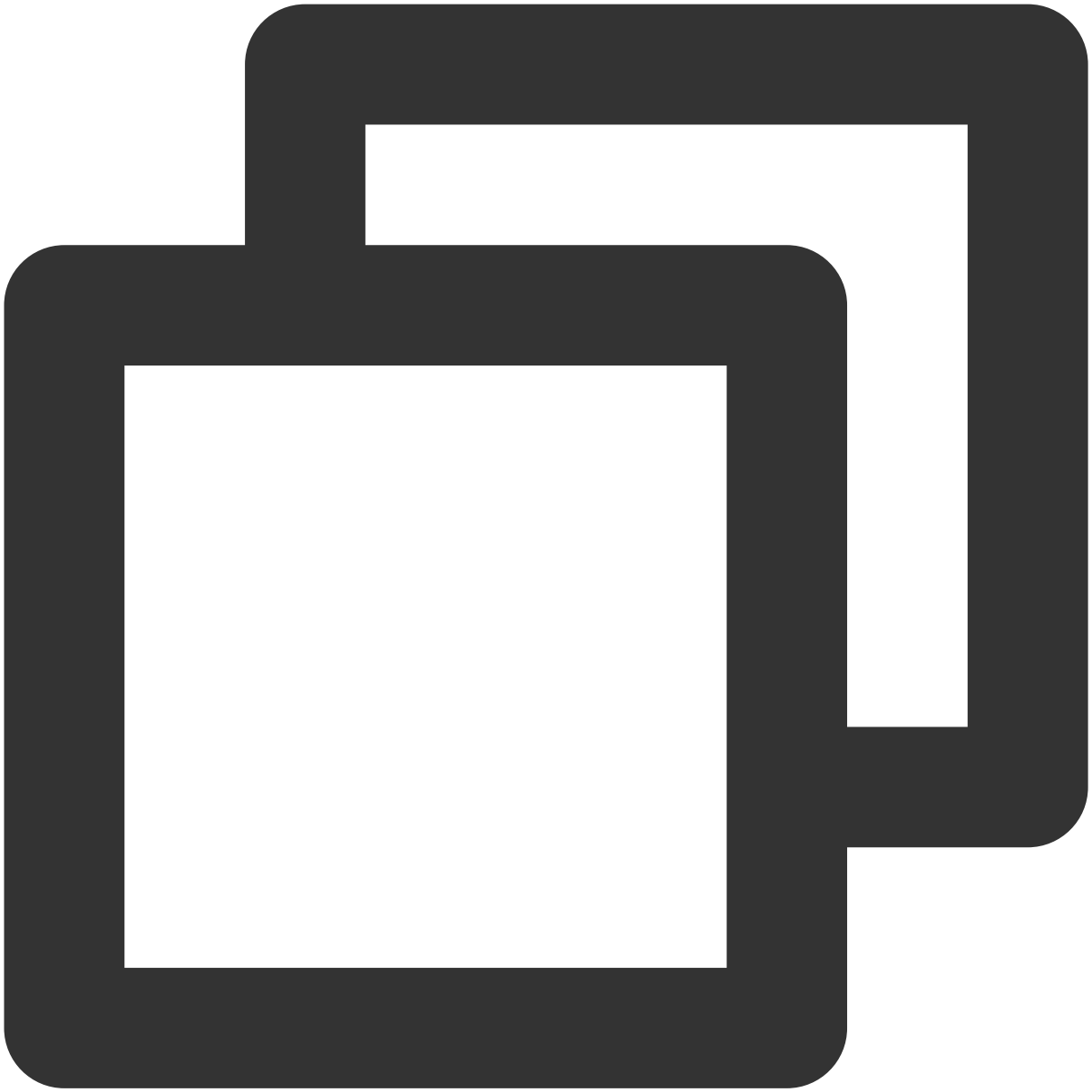
설명：

이 방법을 적용할 경우, 서명 시작 시간은 휴대폰 로컬 시간이 기준입니다. 휴대폰 로컬 시간이 10분 이상 차이가 난다면 서명 오류가 발생할 수 있습니다. 이런 경우 다음의 사용자 정의 응답 본문으로 권한을 부여하는 것을 권장합니다.

사용자 정의 응답 본문에 의한 권한 부여

사용자 정의한 임시 키 서비스의 HTTP 응답 본문이 단말에서 서버로 반환되는 시간을 서명 시작 시간으로 설정함으로써 사용자 휴대폰의 로컬 시간 오차로 서명 오류가 발생하는 것을 방지하거나 다른 프로토콜로 단말과 서버 간 통신하는 등 보다 효율적인 작업을 원한다면, BasicLifecycleCredentialProvider 클래스를 상속하여 fetchNewCredentials()를 출력합니다.

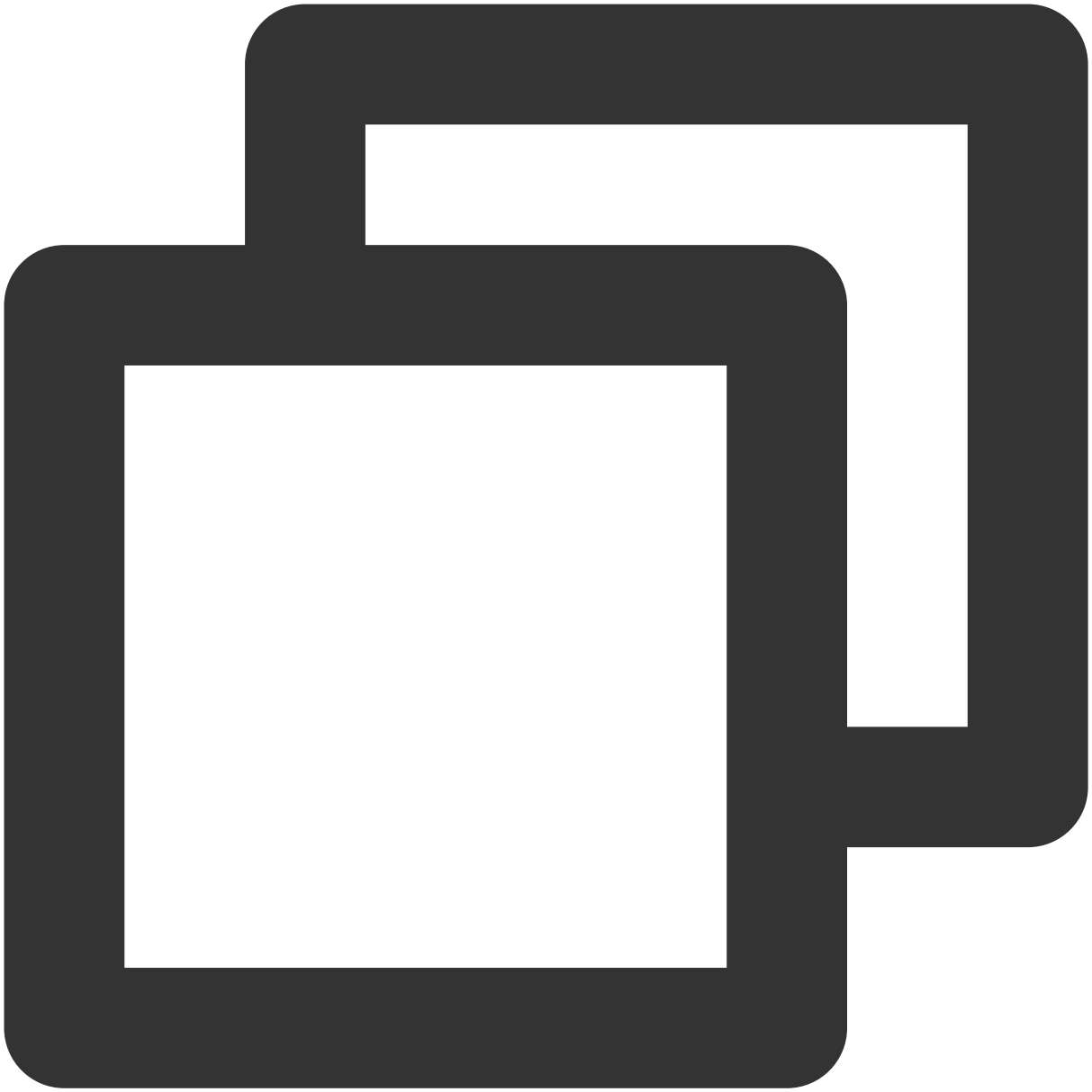
먼저 MyCredentialProvider 클래스를 정의하십시오.



```
import android.content.Context;
import com.tencent.cos.xml.*;
import com.tencent.qcloud.core.auth.*;
import com.tencent.qcloud.core.common.*;
import com.tencent.qcloud.core.http.*;
import java.net.*;


public class MyCredentialProvider extends BasicLifecycleCredentialProvider {

    @Override
```

```
    protected QCloudLifecycleCredentials fetchNewCredentials() throws QCloudClientE

        // 먼저 임시 키 서버로부터 서명 정보가 담긴 응답 획득
        ....

        // 응답을 분석하여 키 정보 획득
        String tmpSecretId = ...;
        String tmpSecretKey = ...;
        String sessionToken = ...;
        long expiredTime = ...;

        // 서버 반환 시간을 서명 시작 시간으로 설정
        long beginTime = ...;

        // todo something you want

        // 최종적으로 임시 키 정보 객체 반환
        return new SessionQCloudCredentials(tmpSecretId, tmpSecretKey, sessionToken
    }
}
```

본인이 정의한 MyCredentialProvider 인스턴스로 권한 부여를 요청합니다.



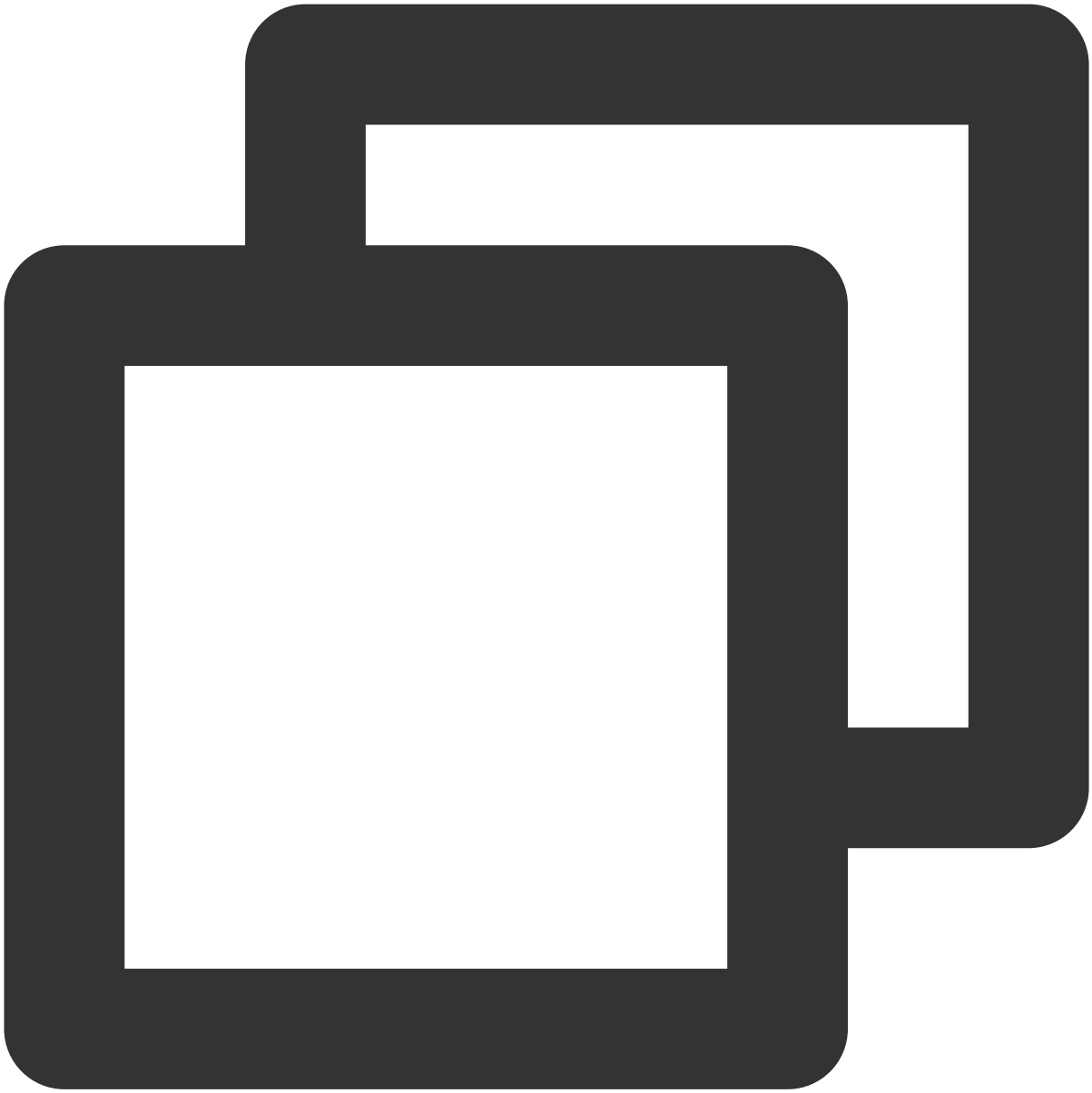
```
import android.content.Context;
import com.tencent.cos.xml.*;
import com.tencent.qcloud.core.auth.*;
import com.tencent.qcloud.core.common.*;
import com.tencent.qcloud.core.http.*;
import java.net.*;

Context context = ...;
CosXmlServiceConfig cosXmlServiceConfig = ...;

/**
```

```
 * {@link QCloudCredentialProvider} 객체 초기화로 SDK에 임시 키 부여
 */
QCloudCredentialProvider credentialProvider = new MyCredentialProvider();

CosXmlService cosXmlService = new CosXmlService(context, cosXmlServiceConfig, crede
```

Android에서 COS로 파일을 업로드 및 다운로드하는 방법은 Android SDK의 시작하기를 참조하십시오.

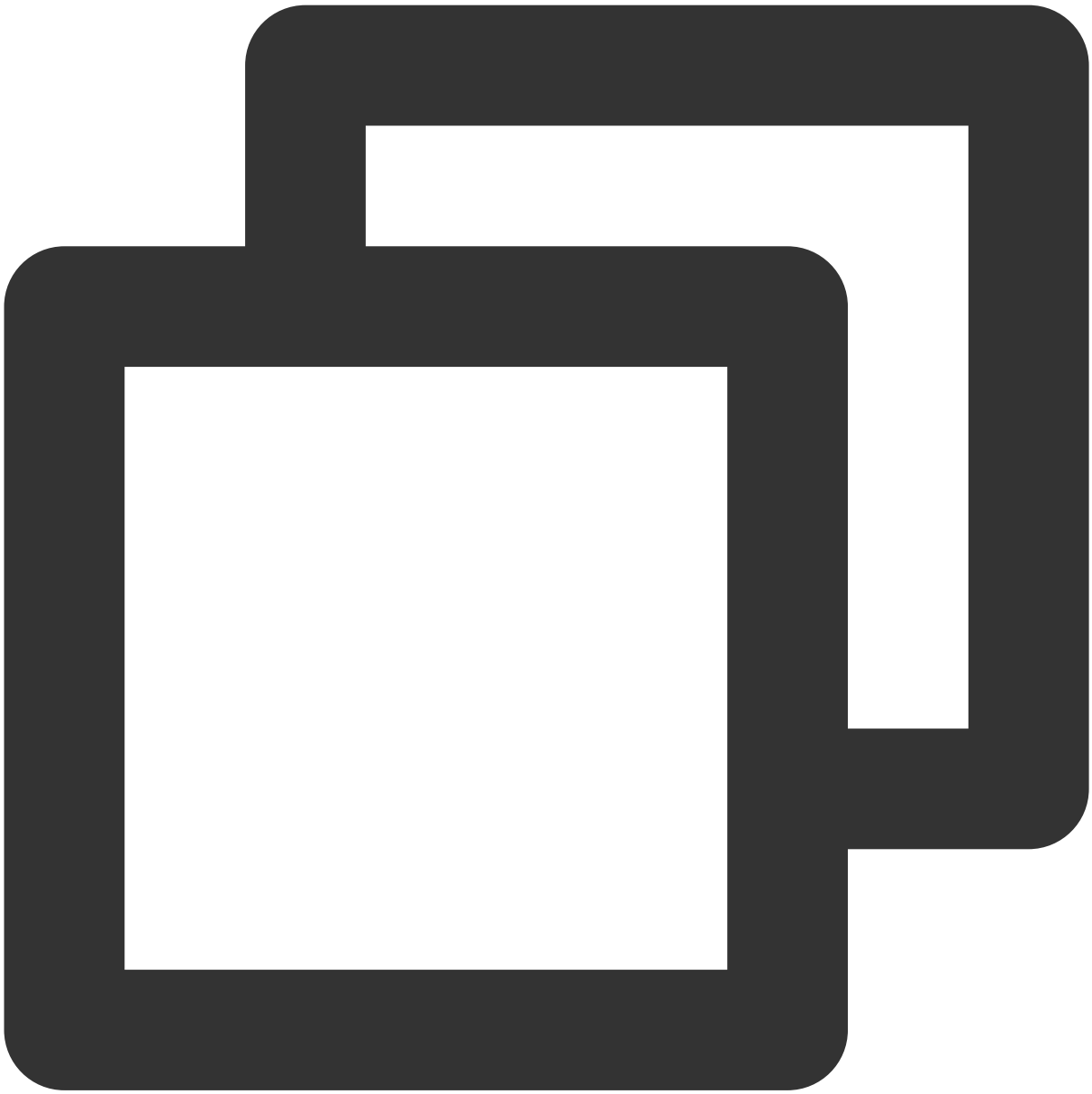
iOS

임시 서명을 간편하게 획득하고 관리할 수 있는 QCloudCredentailFenceQueue를 제공합니다.
QCloudCredentailFenceQueue로 서명을 획득할 경우, 모든 서명이 완료된 뒤 서명 요청이 실행되는 동기화 클래스 CyclicBarrier를 지원하여 스스로 비동기화를 관리하는 과정이 생략됩니다.
QCloudCredentailFenceQueue를 적용할 경우, 먼저 인스턴스 하나를 생성해야 합니다.



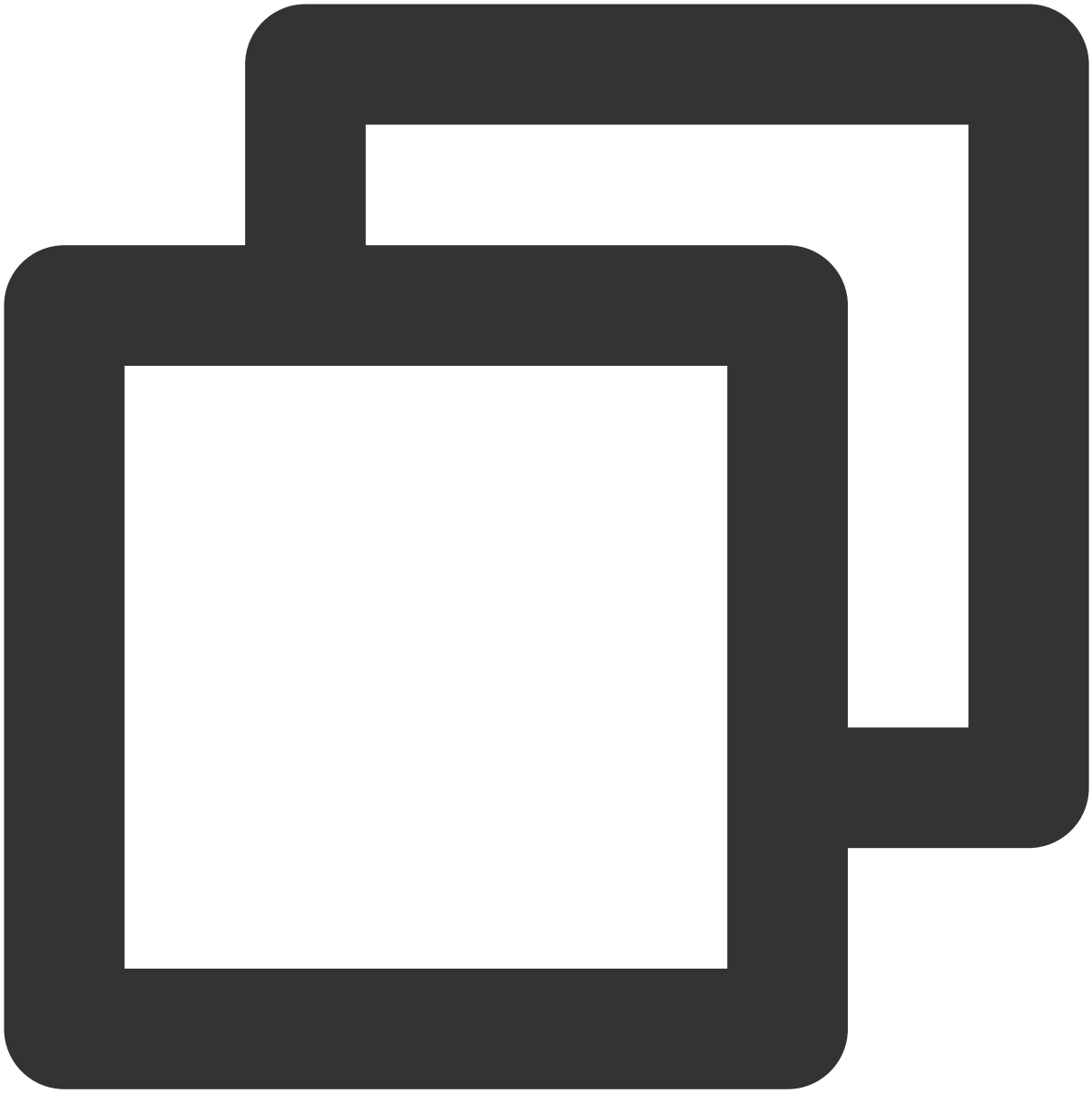
```
 //AppDelegate.m
//AppDelegate는 QCloudCredentailFenceQueueDelegate 프로토콜 따름
//
- (BOOL)application:(UIApplication * )application didFinishLaunchingWithOptions:(NS
    // init step
    self.credentialFenceQueue = [QCloudCredentailFenceQueue new];
    self.credentialFenceQueue.delegate = self;
    return YES;
}
```

QCloudCredentailFenceQueue의 클래스를 호출하려면 QCloudCredentailFenceQueueDelegate를 따르고, 프로토콜에 정의된 방법을 실행합니다.



```
- (void) fenceQueue:(QCloudCredentailFenceQueue * )queue requestCreatorWithContinue
```

QCloudCredentailFenceQueue로 서명을 획득할 경우, 프로토콜에서 정의한 방법으로 서명 매개변수를 확보하여 유효한 서명을 생성한 뒤에야 SDK 내의 모든 서명 요청이 실행됩니다. 다음 예시를 참조하십시오.



```
- (void)fenceQueue:(QCloudCredentailFenceQueue *)queue requestCreatorWithContinue:(
    QCloudHTTPRequest* request = [QCloudHTTPRequest new];
    request.requestData.serverURL = @“your sign service url”;//요청한 URL

    [request setConfigureBlock:^(QCloudRequestSerializer *requestSerializer, QCloud
        requestSerializer.serializerBlocks = @[QCloudURLFuseWithURLEncodeParamters]
        responseSerializer.serializerBlocks = @[QCloudAcceptRespnseCodeBlock([NSSet
                                              QCloudResponseJSONSerilizerBlock];/
    }];

    [request setFinishBlock:^(id response, NSError *error) {
```

```
        if (error) {
            error = [NSError errorWithDomain:@"com.tac.test" code:-1111 userInfo:@{
            continueBlock(nil, error);
        } else {
            QCloudCredential* crendential = [[QCloudCredential alloc] init];
            crendential.secretID = response[@"data"][@"credentials"][@"tmpSecretId"
            crendential.secretKey = response[@"data"][@"credentials"][@"tmpSecretKe
            credential.startDate =[NSDate dateWithTimeIntervalSince1970:@"반환된 서버
            crendential.experationDate = [NSDate dateWithTimeIntervalSinceNow:[resp
            crendential.token = response[@"data"][@"credentials"][@"sessionToken"];
            QCloudAuthentationV5Creator* creator = [[QCloudAuthentationV5Creator al
            continueBlock(creator, nil);

        }
    }];
    [[QCloudHTTPSessionManager shareClient] performRequest:request];
}
```

iOS에서 COS로 파일을 업로드 및 다운로드하는 방법은 iOS SDK의 시작하기를 참조하십시오.

체험

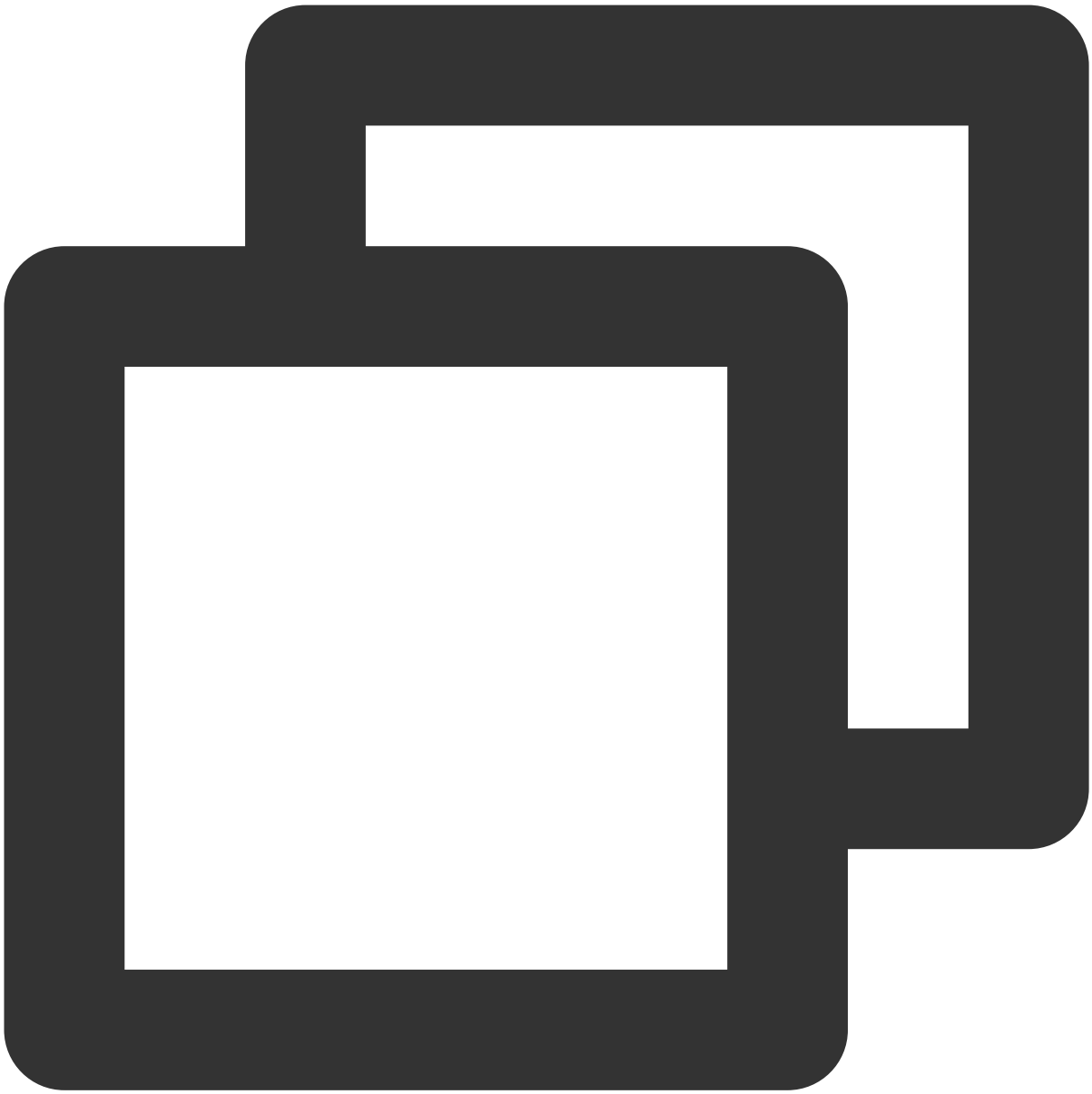
Android

여기를 클릭하거나 Android 휴대폰 브라우저 App으로 아래의 QR코드를 스캔하면 체험용 demo를 다운로드할 수 있습니다.

iOS

iOS의 전체 코드에 관한 내용은 COS iOS Demo를 참조하십시오.

QCloudCOSXMLDemo/QCloudCOSXMLDemo/key.json 파일에 APPID, secretID, secretKey 매개변수 값을 입력한 뒤 다음 명령어를 실행합니다.



```
pod install
```

설명：

APPID, secretID, secretKey는 API Keys 페이지에서 받을 수 있습니다.

명령어를 실행한 뒤 QCloudCOSXMLDemo.xcworkspace를 열면 Demo 체험으로 이동할 수 있습니다.

uni-app 다이렉트 업로드 사례

최종 업데이트 날짜: : 2024-06-24 16:44:04

소개

본 문서에서는 SDK를 사용하지 않고 uni-app을 통해 직접 COS(Cloud Object Storage) 버킷에 파일을 업로드하는 간단한 코드를 사용하는 방법을 설명합니다.
설명 :
본 문서는 XML API PostObject를 기반으로 합니다.

솔루션

실행 프로세스

1. 프런트엔드에서 파일을 선택하면 프런트엔드가 파일 확장자를 서버로 보냅니다.
2. 서버는 파일 확장자에 따라 시간과 함께 임의의 COS 파일 경로를 생성하고 해당 PostObject policy 서명을 계산하고 URL 및 서명 정보를 프런트엔드에 반환합니다.
3. 프런트엔드는 PostObject API를 호출하여 파일을 COS에 업로드합니다.

솔루션 장점

권한 보안: PostObject policy 서명은 보안 권한 범위를 효과적으로 제한하며 지정된 파일 경로에 업로드하는 데만 사용할 수 있습니다.
경로 보안: 서버가 임의의 COS 파일 경로를 결정하므로 기존 파일을 덮어쓰는 문제와 보안 위험을 효과적으로 방지할 수 있습니다.
다중 터미널 호환성: uni-app에서 제공하는 파일 선택 및 업로드 API를 사용하여 동일한 코드를 다중 터미널(Web/미니프로그램/App)과 호환되도록 할 수 있습니다.

전제 조건

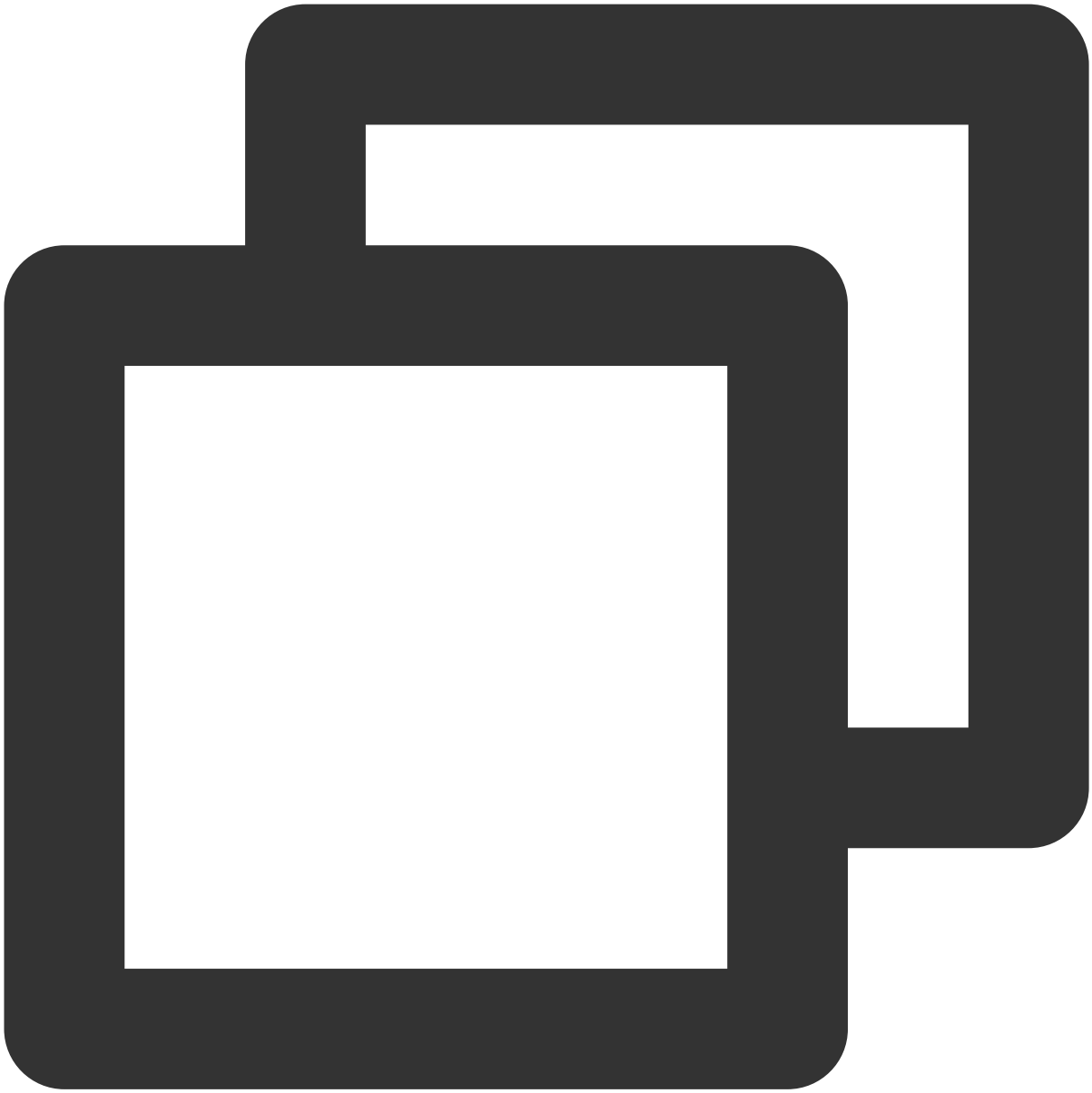
1. COS 콘솔에 로그인하고 버킷을 생성하여 Bucket(버킷 이름)과 Region(리전 이름)을 획득합니다. 자세한 내용은 버킷 생성 문서를 참조하십시오.
2. CAM 콘솔에 로그인한 뒤 프로젝트의 SecretId와 SecretKey를 획득합니다.
3. 새로 생성된 버킷의 세부 정보 페이지로 이동하여 **보안 관리 > CORS(Cross-Origin Resource Sharing)**를 선택하고 **규칙 추가**를 클릭합니다. 아래 이미지는 구성 예시입니다. 자세한 내용은 크로스 도메인 액세스 설정을 참고하십시오.

실행 순서

주의 :
정식 배포 전에 서버 측에서 웹사이트 자체에 대한 권한 확인 단계를 추가하는 것이 좋습니다.

프런트 엔드 업로드

1. 서버 API를 구현하여 임의의 파일 경로를 생성하고 서명을 계산하여 프런트엔드로 반환합니다. 구현 세부 정보는 post-policy 예시를 참고하십시오.
2. HBuilderX의 기본 템플릿을 사용하여 uni-app 앱을 만듭니다.
생성된 앱은 Vue 프로젝트입니다.
3. 다음 코드를 복사하여 pages/index/index.vue 파일의 내용을 바꾸고 post-policy API 호출 링크를 수정하여 서버 주소(1단계에서 구현된 서버 API)를 가리키도록 합니다.



```
<template>
<view class="content">
   <button type="default" @click="selectUpload">파일 업로드 선택</button>
   <image v-if="fileUrl" class="image" :src="fileUrl"></image>
</view>
</template>

<script>
export default {
   data() {
      return {
```

```
            title: 'Hello',
            fileUrl: ''
        };
    },
    onLoad() {

    },
    methods: {
        selectUpload() {

            var vm = this;

            // 더 많은 문자열 코드의 url encode 형식
            var camSafeUrlEncode = function (str) {
                return encodeURIComponent(str)
                        .replace(/!/g, '%21')
                        .replace(/'/g, '%27')
                        .replace(/\\(/g, '%28')
                        .replace(/\\)/g, '%29')
                        .replace(/\\*/g, '%2A');
            };

            // 업로드 경로 및 자격 증명 가져오기
            var getUploadInfo = function (extName, callback) {
                // 백엔드가 임의의 COS 객체 경로를 생성하고 업로드 도메인과 PostObject API에 필
                // 서버 예시 참고: https://github.com/tencentyun/cos-demo/tree/main/serve
                uni.request({
                    url: 'http://127.0.0.1:3000/post-policy?ext=' + extName,
                    success: (res) => {
                        callback && callback(null, res.data.data);
                    },
                    error(err) {
                        callback && callback(err);
                    },
                });
            };

            // 업로드 요청을 시작하고 업로드는 PostObject API 및 policy 서명 보호 사용
            // API 문서: https://www.tencentcloud.com/document/product/436/14690#.E7.AD
            var uploadFile = function (opt, callback) {
                var formData = {
                    key: opt.cosKey,
                    policy: opt.policy, // Base64 policy 문자열
                    success_action_status: 200,
                    'q-sign-algorithm': opt.qSignAlgorithm,
                    'q-ak': opt.qAk,
                    'q-key-time': opt.qKeyTime,
```

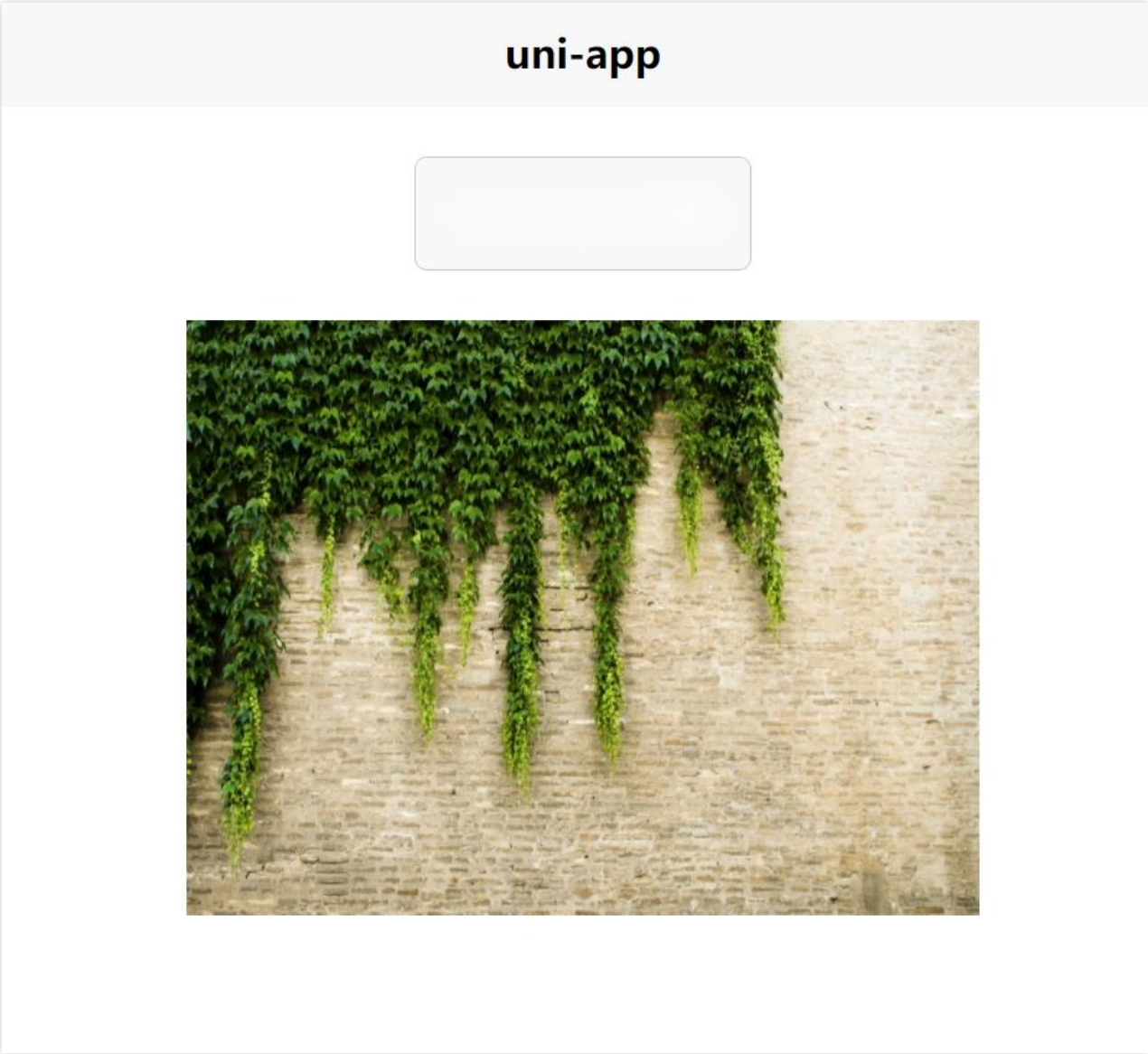
```
                'q-signature': opt.qSignature,
            };
            // 서버가 계산을 위해 임시 키를 사용하는 경우 x-cos-security-token 전달 필요
            if (opt.securityToken) formData['x-cos-security-token'] = opt.securityT
            uni.uploadFile({
                url: 'https://' + opt.cosHost, // 실제 API 주소가 아닌 예시
                filePath: opt.filePath,
                name: 'file',
                formData: formData,
                success: (res) => {
                    if (![200, 204].includes(res.statusCode)) return callback && call
                    var fileUrl = 'https://' + opt.cosHost + '/' + camSafeUrlEncode(o
                    callback && callback(null, fileUrl);
                },
                error(err) {
                    callback && callback(err);
                },
            });
        };

        // 파일 선택
        uni.chooseImage({
            success: (chooseImageRes) => {
                var file = chooseImageRes.tempFiles[0];
                if (!file) return;
                // 업로드할 로컬 파일의 경로 가져오기
                var filePath = chooseImageRes.tempFilePaths[0];
                // 업로드할 파일의 확장자를 가져오면 백엔드가 임의의 COS 경로 생성
                var fileName = file.name;
                var lastIndex = fileName.lastIndexOf('.');
                var extName = lastIndex > -1 ? fileName.slice(lastIndex + 1) : '';
                // 사전 업로드를 위한 도메인 이름, 경로 및 자격 증명 가져오기
                getUploadInfo(extName, function (err, info) {
                    // 파일 업로드
                    info.filePath = filePath;
                    uploadFile(info, function (err, fileUrl) {
                        vm.fileUrl = fileUrl;
                    });
                });
            }
        });
    },
  }
}
</script>

<style>
```

```
.content {
   padding: 20px 0;
   display: flex;
   flex-direction: column;
   align-items: center;
   justify-content: center;
}

.image {
   margin-top: 20px;
   margin-left: auto;
   margin-right: auto;
}
</style>
```

4. HBuilderX에서 **실행 > 브라우저로 실행 > Chrome**을 선택합니다. 그 다음 브라우저에서 업로드할 파일을 선택할 수 있습니다.

실행 결과는 다음 이미지와 같습니다.

직접 업로드 효과:



Demo 코드 주소

Demo 다운로드 주소: upload-demo

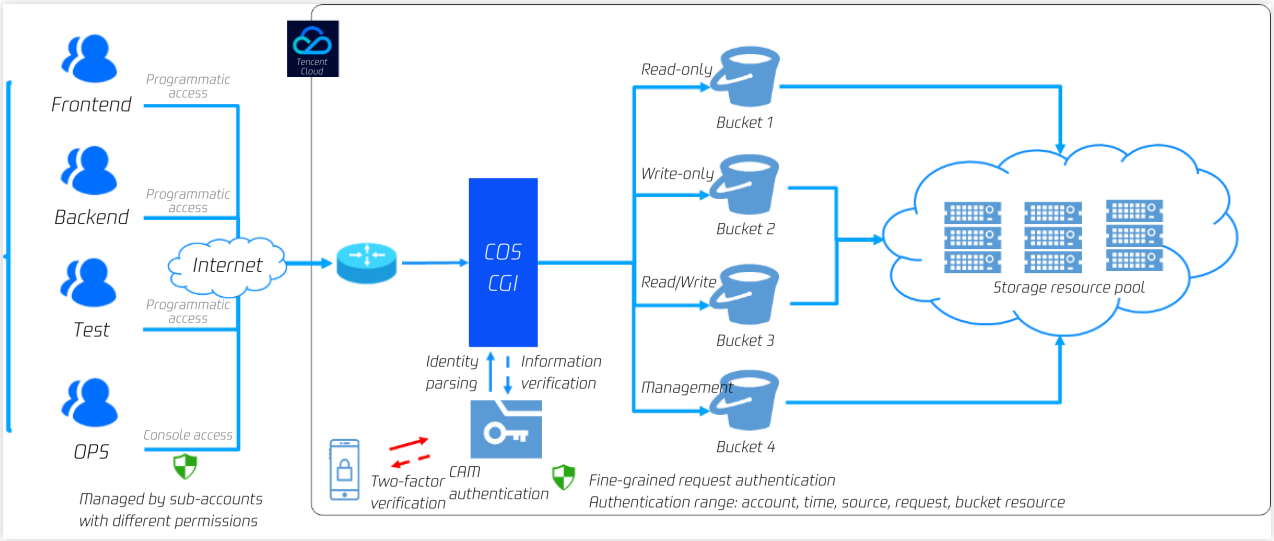
데이터 보안
COS 데이터 보안 솔루션 소개

최종 업데이트 날짜: : 2024-06-24 16:53:18

사전 보안 수단

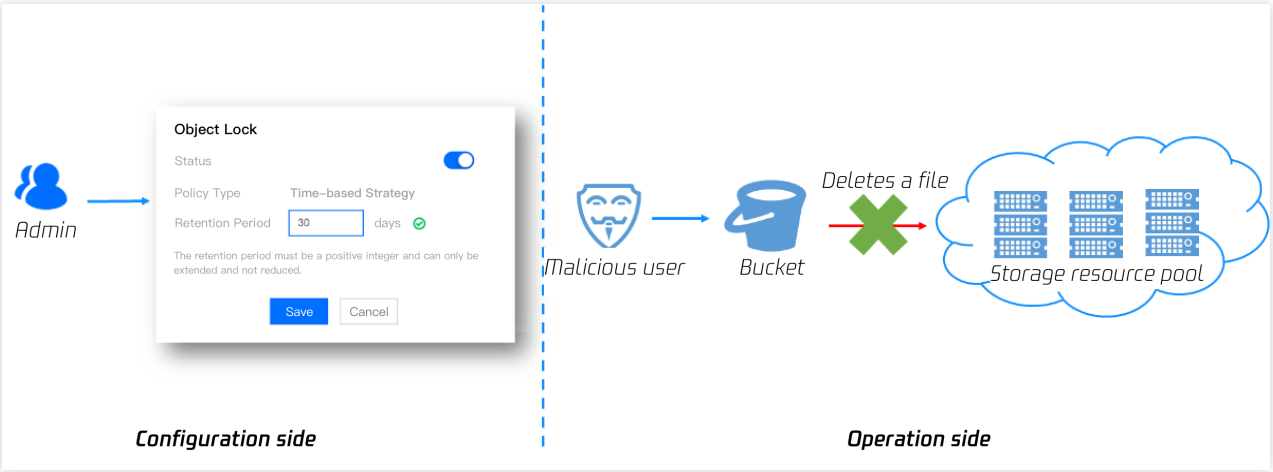
1. 권한 격리

클라우드 기업에게 있어 계정 보안과 리소스에 대한 합리적인 권한 부여는 입체적인 보호 체계를 구축하기 위한 첫 번째 관문입니다. 클라우드 상의 리소스 관리로 권한 부여를 할 때 아래 위험 요소는 반드시 피해야 합니다.
Tencent Cloud 루트 계정으로 일상 업무를 하는 경우
직원들에게 서브 계정을 부여했지만 지나치게 큰 권한을 부여한 경우
높은 권한이 부여된 서브 계정과 고위험 작업 수행 시 액세스 권한 조건이 규제되지 않은 경우
사용자 권한과 로그인 정보에 정기적인 감사가 없는 경우
권한 관리 제도와 절차가 허술한 경우
Tencent Cloud CAM은 계정 등급이나 권한 등급 등 여러 가지 조치를 통해 권한을 명확히 하고 보안을 제어합니다. 계정 등급에서 루트 계정은 서브 계정, 협업 파트너 등 모든 합법적인 CAM 사용자에게 프로그램 액세스와 콘솔 액세스 등 다양한 액세스 권한을 부여할 수 있습니다. 권한 등급에서는 서비스 등급, 인터페이스 등급, 리소스 등급 등 다양한 등급의 권한 부여를 통해 CAM 사용자가 **어떤 조건에서 어떤 방식을 통해 어떤 리소스에 대해 어떤 작업**을 하게 할지 권한 부여가 가능합니다.
루트 계정에 서브 계정을 생성하고 서브 계정에 루트 계정 리소스 관리 권한을 부여하면 루트 계정에 관련한 자격 증명은 필요하지 않습니다. 그 외에도 다양한 유저에게 다양한 리소스에 대한 액세스 권한을 부여할 수 있습니다. 서브 계정이 어떤 COS 버킷의 읽기 권한을 가지고 있다면 다른 서브 계정이나 루트 계정은 어떤 COS 객체의 쓰기 권한 등을 가질 수 있습니다. 이 곳의 리소스, 액세스 권한, 사용자를 모두 일괄 관리할 수 있어 정교한 권한 관리가 가능합니다.
고위험 작업(예: 데이터 삭제) 권한은 제한할 수 있으며 사용자가 콘솔에서 작업을 하도록 허용하거나 MFA 인증 활성화를 통해 2차 인증을 진행합니다. MFA 인증을 활성화하면 사용자가 유사한 고위험 조작을 할 경우 SMS를 통해 검증 코드를 트리거하여 인증을 진행합니다.



2. 객체 잠금

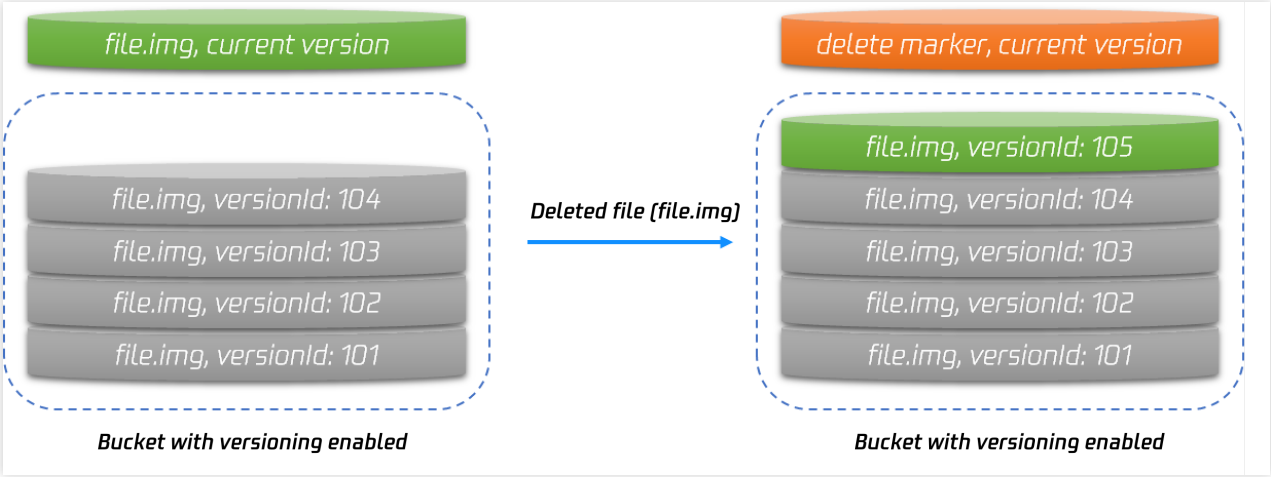
금융 거래 데이터, 의료 영상 데이터 등 중요한 내용의 중요 데이터인 경우 객체 잠금 기능으로 파일 업로드 후 삭제 및 수정을 방지할 수 있습니다. 객체 잠금 기능을 설정한 후 설정한 유효 기간 동안 버킷 내의 모든 데이터는 읽기 전용 상태가 되며 덮어쓰기 및 삭제가 불가합니다. 이 작업의 적용 대상은 루트 계정을 포함한 CAM 사용자 및 익명 사용자입니다.



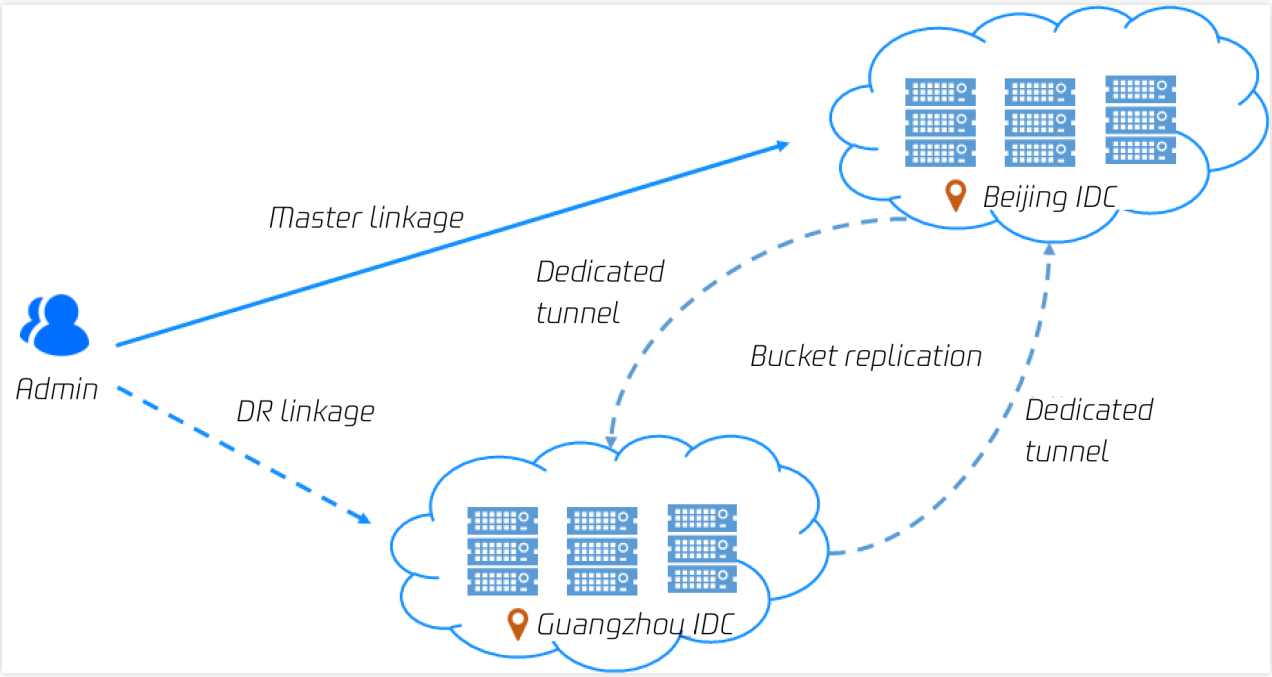
3. 데이터 재해 복구

Tencent cloud COS는 데이터 암호화, 버전 제어, 버킷 복제 및 라이프사이클 등을 포함한 데이터 관리를 지원합니다. 민감한 내용이 들어 있는 파일은 암호화 기능으로 데이터 읽기/쓰기 보안을 보장하며 버전 제어와 버킷 복제를 통해 원격 재해 복구를 실행하여 데이터의 영구성을 보장하고, 데이터를 실수로 삭제하거나 악의적으로 삭제할 때 백업 지점에서 데이터를 복구할 수 있습니다. 또한 라이프사이클을 통해 데이터를 전환하고 삭제하여 데이터 저장 비용을 줄일 수 있습니다.

버전 제어는 파일 덮어쓰기 및 삭제 방지 기능입니다. 버전 제어 설정을 활성화한 후 동일한 이름을 가진 파일을 업로드하면 동일 이름을 가진 새로운 버전 파일이 생성됩니다. 파일을 삭제하면 삭제 마커가 생성됩니다. 데이터를 실수로 삭제했거나 덮어쓰기 한 경우 지정된 버전 번호를 통해 이전 버전의 데이터에 액세스하고 데이터를 롤백하여 문제를 해결할 수 있습니다.

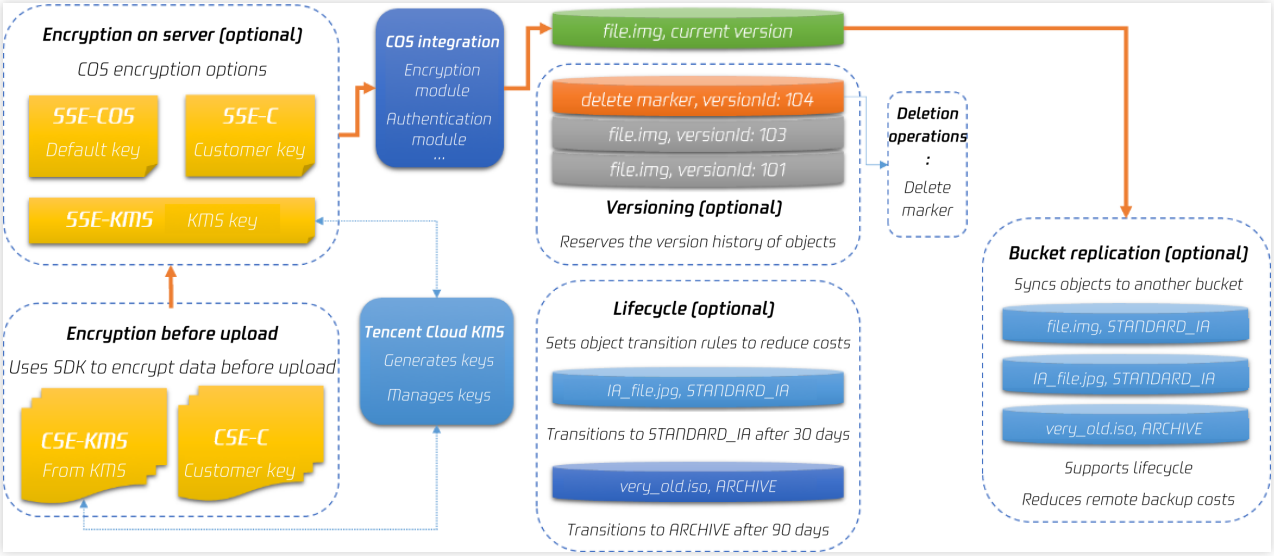


그 외에도 COS는 버킷 복제 기능을 제공합니다. 사용자는 전용선을 통해 모든 증분 파일을 다른 리전의 데이터 센터로 복제하여 원격으로 재해 복구가 가능합니다. 루트 버킷에 있는 데이터가 삭제된 경우 백업 버킷에서 일괄 복사를 통해 데이터를 복구할 수 있습니다.

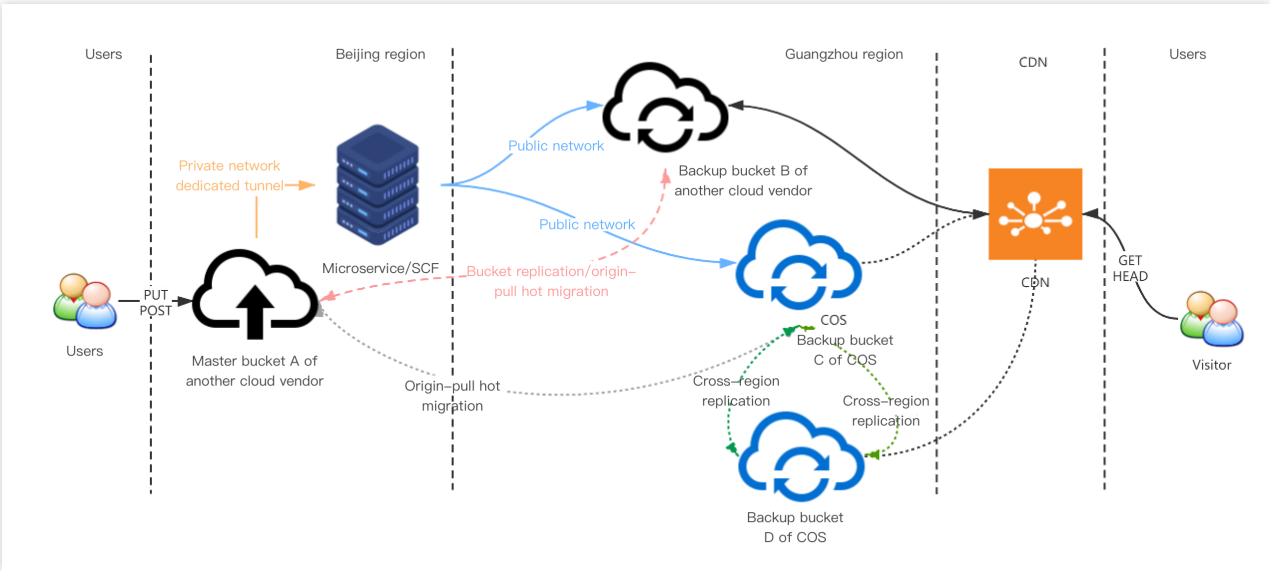


버전 제어와 버킷 복제가 파일 수를 증가시킬 수 있는 점을 고려하여, 사용자는 라이프사이클 기능을 통해 일부 백업 데이터를 표준IA 또는 CAS 등 더욱 저렴한 스토리지 유형으로 전환할 수 있으며 이로써 저비용 콜드 스탠바이를 실

현할 수 있습니다. 데이터 암호화, 버전 제어, 버킷 복제 및 라이프사이클 기능을 종합하여 Tencent Cloud COS는 다음 그림과 같은 완벽한 콜드 스탠바이 솔루션을 제공합니다.

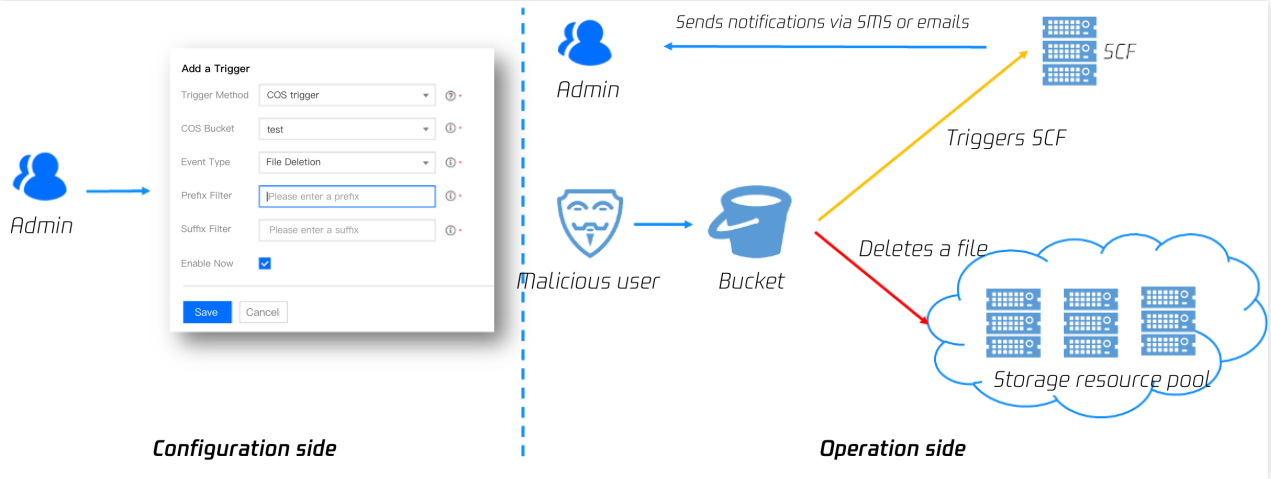


다른 클라우드 벤더의 스토리지 서비스를 주로 이용하거나 데이터 영구성에 대한 요구가 높은 편이라면 SCF에 기반한 COS 재해 복구 솔루션을 고려할 수 있습니다. 먼저 데이터가 다른 클라우드 벤더(AWS 또는 OSS)에 저장되어 있다면 SCF를 통해 데이터 동기화를 트리거하거나 버킷 복제로 원격 재해 복구하여 데이터의 영구성을 보장합니다. 또한 SCF를 통해 데이터 마이그레이션을 트리거하면 중요 데이터가 Tencent Cloud COS 서비스에 백업되며, Tencent Cloud 버킷 복제 기능으로 원격 재해 복구를 실행할 수 있습니다. 마지막으로 Tencent Cloud 권한 관리 통제를 통해 COS의 데이터 액세스 권한을 관리하여 극단적인 상황에서 데이터를 Tencent Cloud COS에서 복구할 수 있습니다.



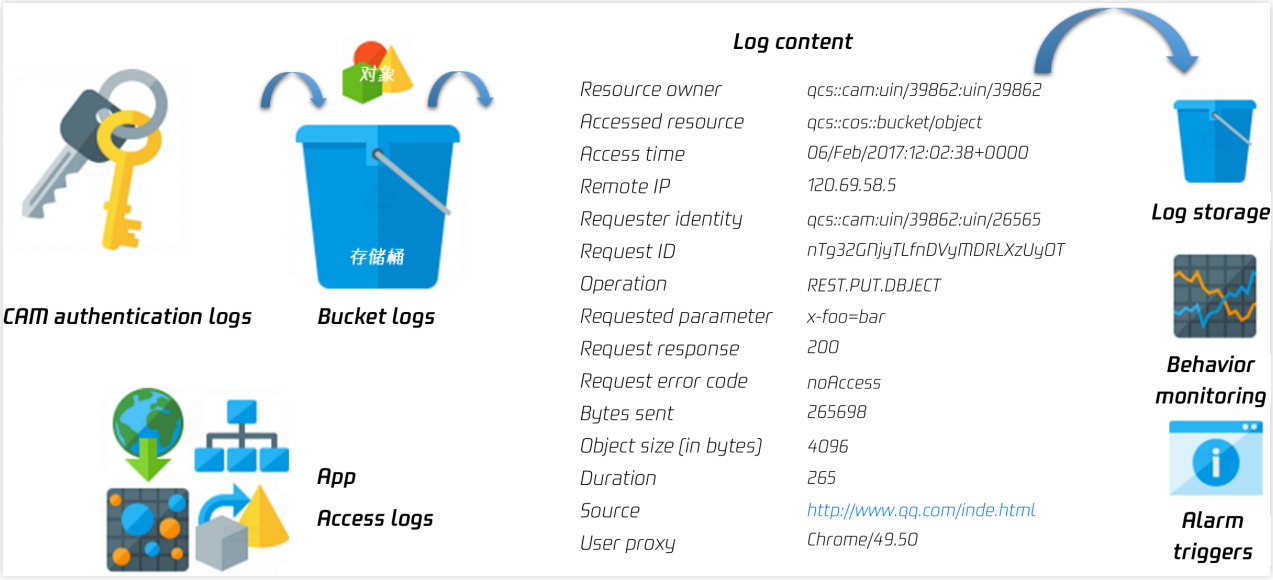
작업 중 모니터링 수단

Tencent Cloud COS는 SCF를 기반으로 알림 기능을 제공합니다. SCF를 통해 DeleteObject와 같은 고위험 작업에 SCF를 설정합니다. 고위험 조작이 일어났을 때 이메일이나 핸드폰에 알림을 보내 곧바로 고위험 행위를 알리고 행위를 중지시킬 수 있습니다.



사후 트래킹 수단

Tencent Cloud COS는 여러 가지 진입 장벽이 낮은 로그 모니터링과 감사 기능을 제공합니다. 버킷 사용자 액세스 로그에는 파일 삭제(DeleteObject), 파일 덮어쓰기(PutObjectCopy), 파일 권한 수정(PutObjectACL) 등의 작업이 포함되며, 버킷 액세스 로그를 통해 트래킹을 진행하고 삭제 작업 등 고위험 행위를 트래킹하고 검증할 수 있습니다. 버킷 설정 관리 행위에는 버킷 삭제(DeleteBucket), 버킷 액세스 제어 리스트 수정(PutBucketACL), 버킷 정책 수정(PutBucketPolicy) 등의 작업이 포함되며 CA 로그 감사를 통해 트래킹을 진행하고 권한 설정 수정 등으로 트래킹 인증을 진행할 수 있습니다.



도용방지 안내

최종 업데이트 날짜: : 2024-07-17 16:27:00

머리말

최근 몇 년간 점점 더 많은 사용자가 웹사이트나 이미지 호스팅 서비스를 구축할 때 이미지 및 비디오와 같은 리소스를 Cloud Object Storage(COS)에 업로드하여 서버 스토리지 공간의 압력을 줄이는 동시에 액세스 안정성을 향상시켰지만, 이에 따른 트래픽 도용, 이미지 도용 링크 문제도 많은 개발자를 괴롭히고 있으며, 스토리지 공간이 악의적으로 액세스되면 고액의 트래픽 요금이 발생하고 불필요한 분쟁이 발생하게 됩니다. 이러한 문제는 실제로 다양한 방법으로 보호할 수 있으며, 여기서는 개발자가 버킷을 합리적으로 구성하고 보안 메커니즘을 구축하여 유사한 문제로 인한 거액 자금 손실의 위험을 줄일 수 있도록 돕기 위해 몇 가지 일반적인 보호 수단을 주로 소개합니다.

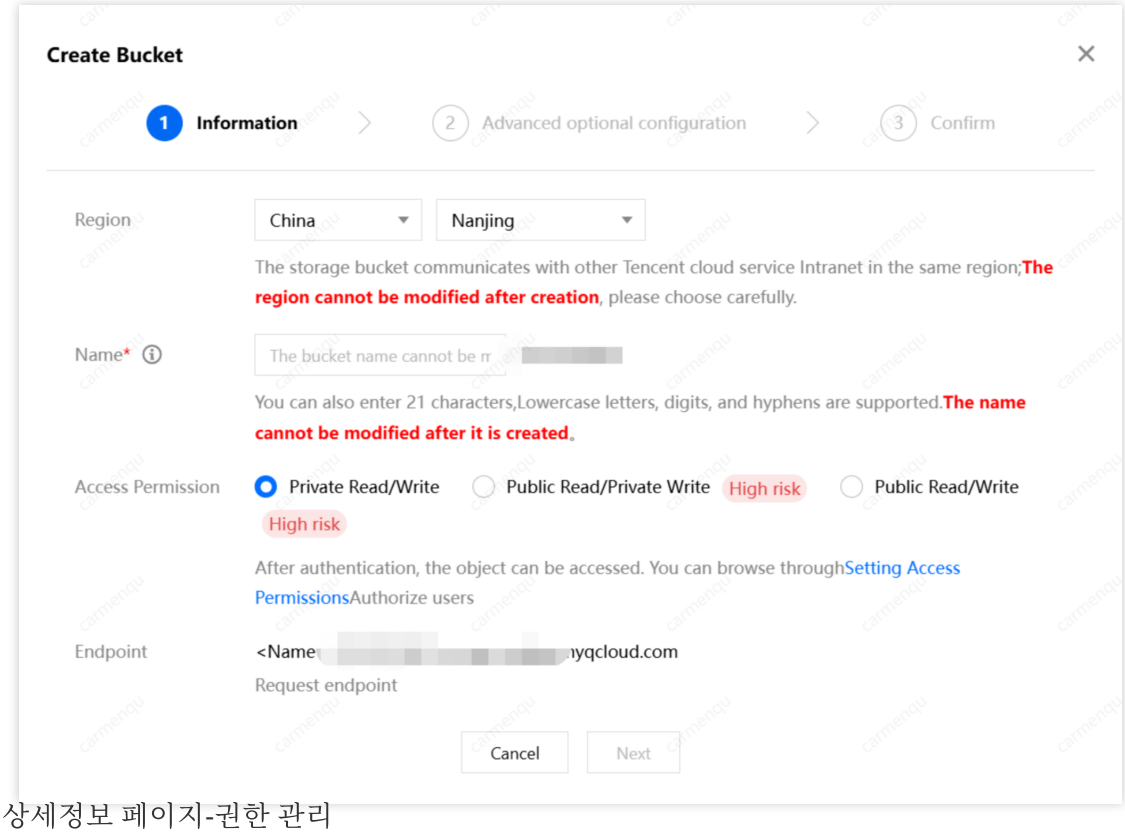
보호 솔루션

도용에 대해서는 여러 가지 보호 방식이 있으며, 여기서는 구성 난이도 및 조건에 따라 구분하며, 개발자는 실제 상황 및 수요에 따라 선택할 수 있습니다.

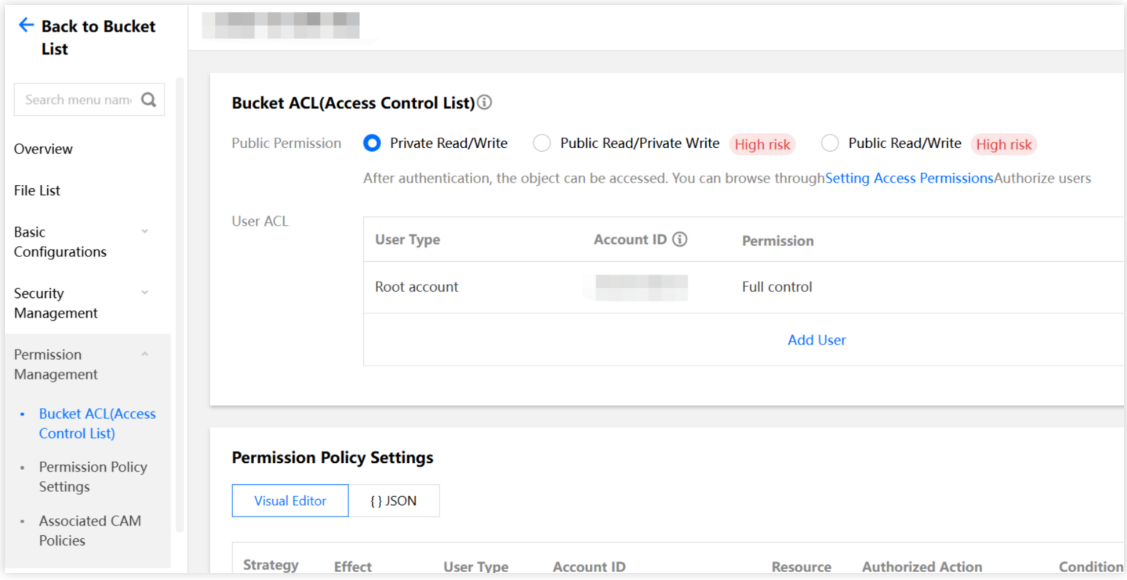
기본 보호 솔루션

버킷 액세스 권한 변경

액세스 권한은 버킷의 가장 핵심적이고 민감한 구성 중 하나이며, 불완전 통계에 따르면, 절대다수의 도용 원인은 모두 사용자가 **공개 읽기** 권한을 설정하였기 때문이며, 공개 읽기 권한은 서명 없이 직접 익명으로 액세스할 수 있으므로 불법 생산자 및 공격자에게 기회를 제공합니다. 버킷을 개인 읽기/쓰기로 설정하면 도용 위험을 크게 줄일 수 있습니다. 비밀 키가 없으면 서명을 계산할 수 없으므로 액세스가 거부됩니다.
업무에 특별한 요구사항이 없으면 가능한 한 **개인 읽기/쓰기** 권한을 구성하는 것을 권장합니다. COS 콘솔은 두 곳에서 버킷 권한을 설정할 수 있습니다.
버킷 팝업창 생성

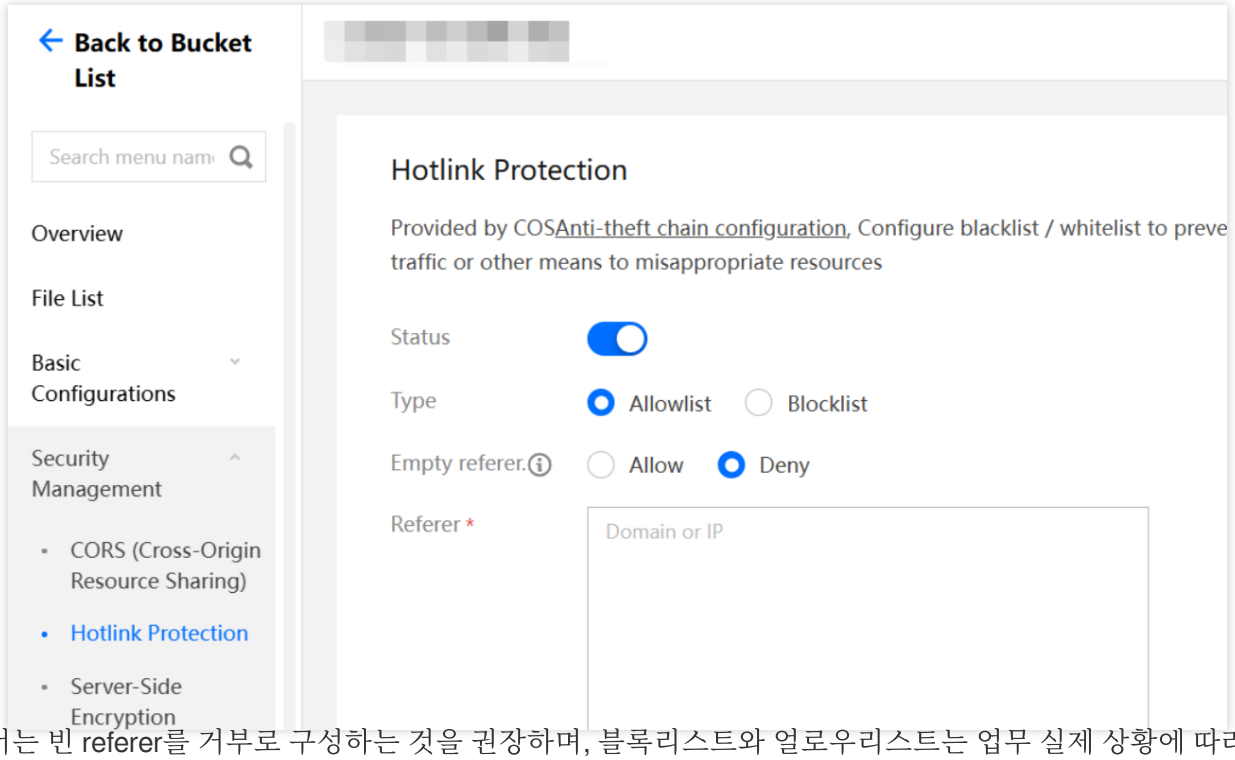


버킷 상세정보 페이지-권한 관리



버킷 핫링크 보호 활성화

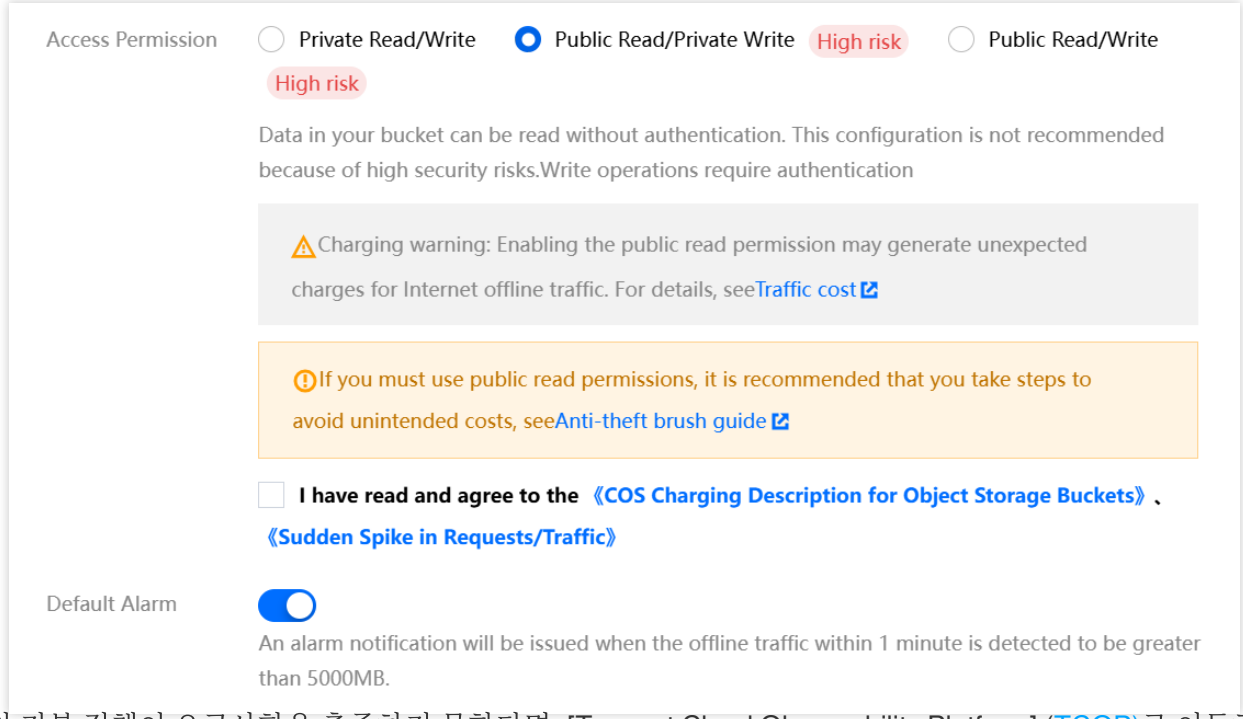
핫링크 보호도 가장 자주 사용되는 보호 수단 중 하나이며, 그 원리는 HTTP의 Referer 헤더를 통해 판단 및 검증하는 것으로, 사용자는 얼로우리스트 또는 블록리스트를 통해 특정 액세스 출처를 허용하거나 금지할 수 있습니다.
COS 콘솔-버킷 상세정보 페이지-안전 관리에서 핫링크 보호 설정을 찾을 수 있습니다.



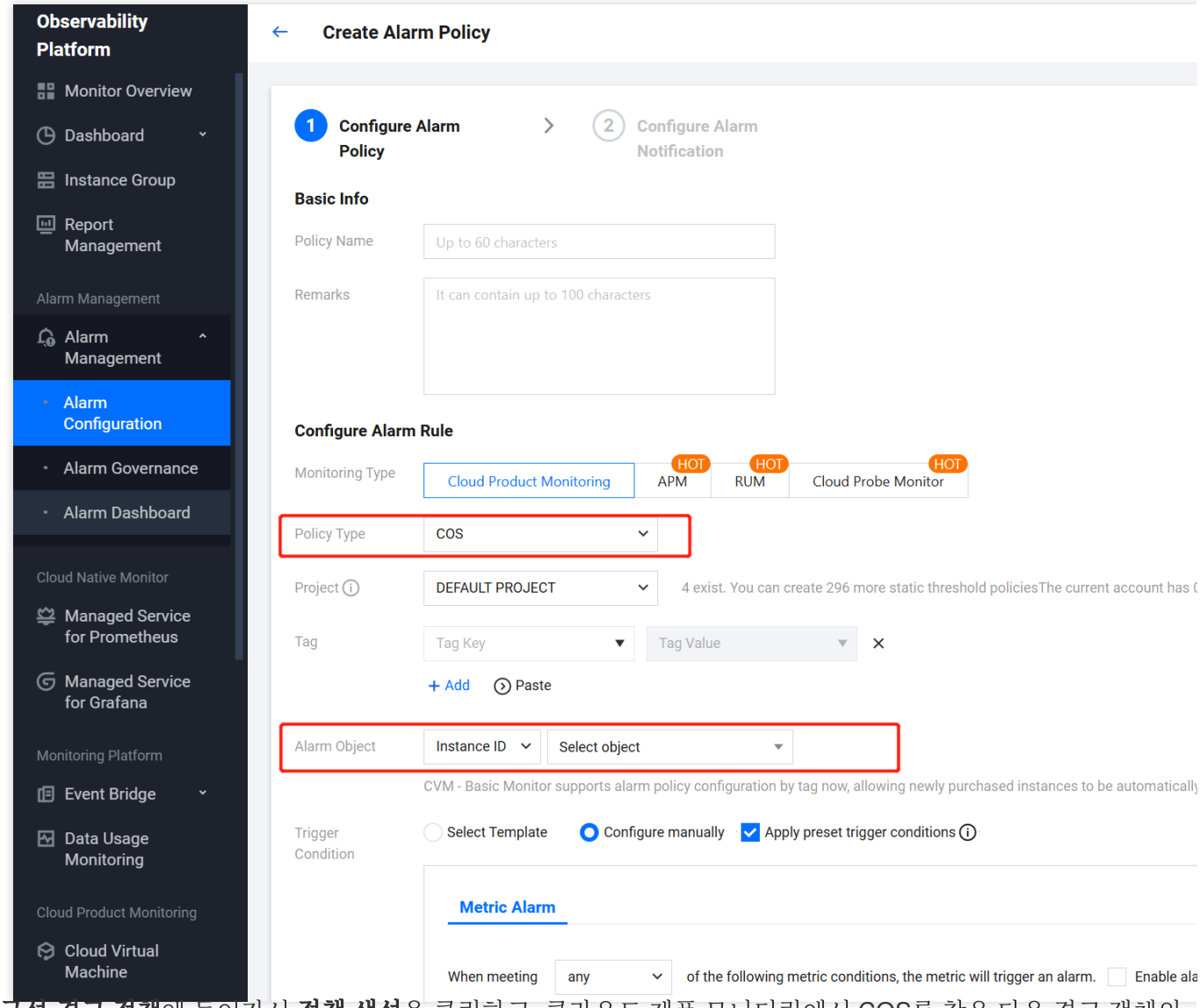
여기서는 빈 referer를 거부로 구성하는 것을 권장하며, 블록리스트와 얼로우리스트는 업무 실제 상황에 따라 선택할 수 있습니다. 특정 도메인에서 고정적으로 액세스하는 경우 얼로우리스트로 설정할 수 있고, 악의적인 액세스가 발견되었고 액세스 도메인 또는 IP를 명확히 알고 있다면 수동으로 블록리스트를 설정하여 차단할 수 있습니다. 더 많은 핫링크 보호의 사용 방법을 알고 싶다면 핫링크 보호 실천을 참고하시기 바랍니다.

Cloud Monitor 경고 설정

로그 관리를 구성하면 도용의 출처를 찾고 분석하는 데 도움이 되며, 조회를 위한 풍부한 필드가 제공되지만 개발자가 적극적으로 로그에 주의를 기울여야 한다는 단점이 있습니다. 도용 문제를 적시에 발견하기를 원하시면 Cloud Monitor 경고를 구성할 수 있습니다. 현재 COS 콘솔에서 버킷을 생성할 때 동시에 경고를 구성할 수 있습니다.



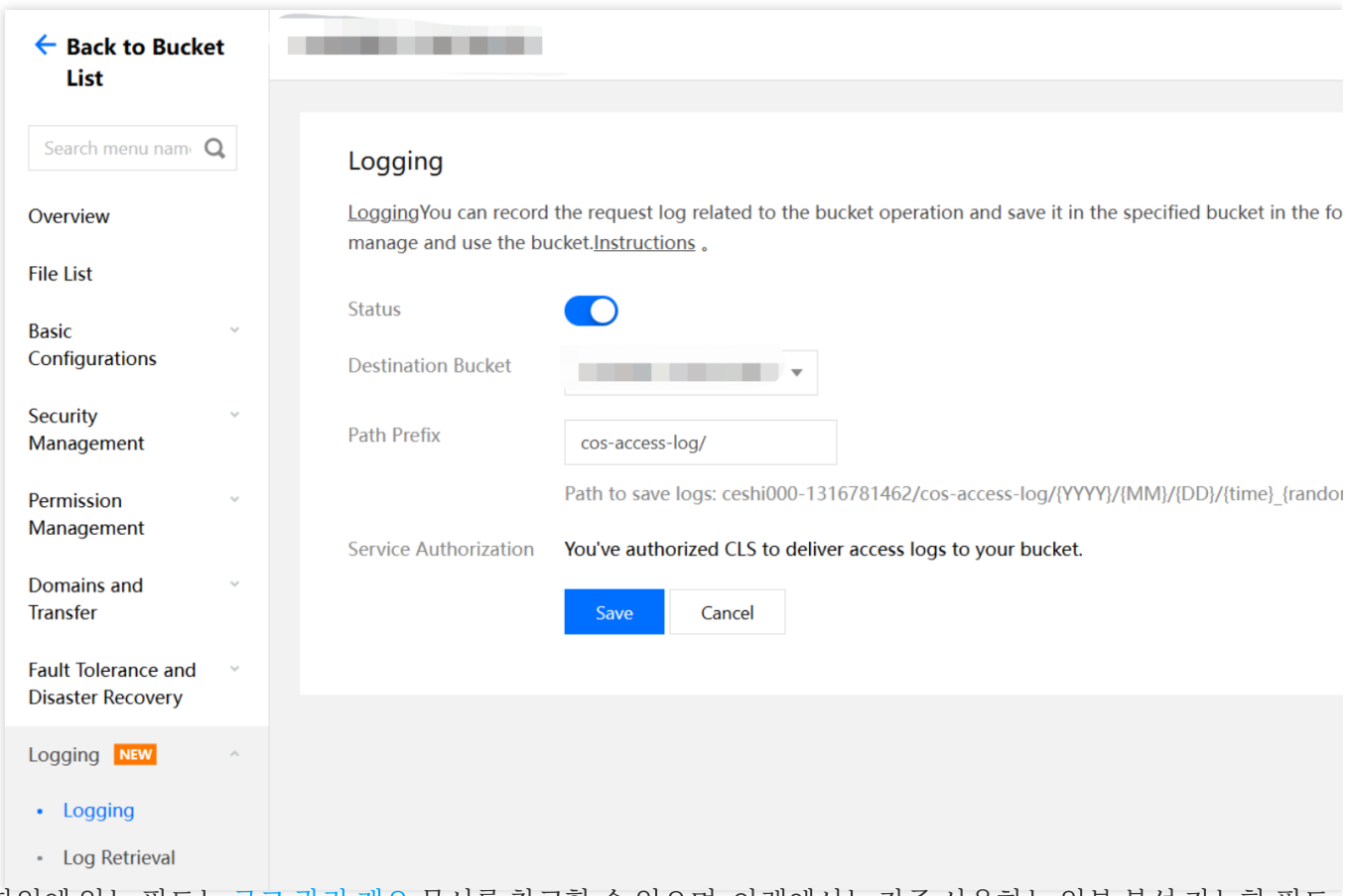
경고의 기본 정책이 요구사항을 충족하지 못한다면, [Tencent Cloud Observability Platform] (TCOP)로 이동하셔서 사용자 정의 경고 정책을 수동으로 생성할 수 있습니다.



경고 구성-경고 정책에 들어가서 **정책 생성**을 클릭하고, 클라우드 제품 모니터링에서 COS를 찾은 다음 경고 객체의 인스턴스 ID에서 구성해야 할 버킷을 찾으면 트리거 조건을 자체적으로 지정할 수 있습니다. COS 버킷의 데이터 모니터링 페이지에 표시된 모든 지표는 여기에서 구성이 지원됩니다.

로그 관리 활성화

위에서 핫링크 보호 블록리스트를 설명할 때 언급한 바와 같이 '악의적인 액세스 도메인 또는 IP가 발견되었다'면, 여기에서 버킷의 로그 관리를 활성화해야 하며, 액세스 로그에는 각 요청에 대한 여러 가지 필드가 기록되어 액세스 출처를 빠르게 찾을 수 있도록 돕습니다.
COS 콘솔-버킷 상세정보 페이지-로그 관리에서 로그 스토리지를 찾을 수 있으며, 활성화한 후 버킷의 지정된 경로 프리픽스에서 액세스 로그를 찾을 수 있습니다.

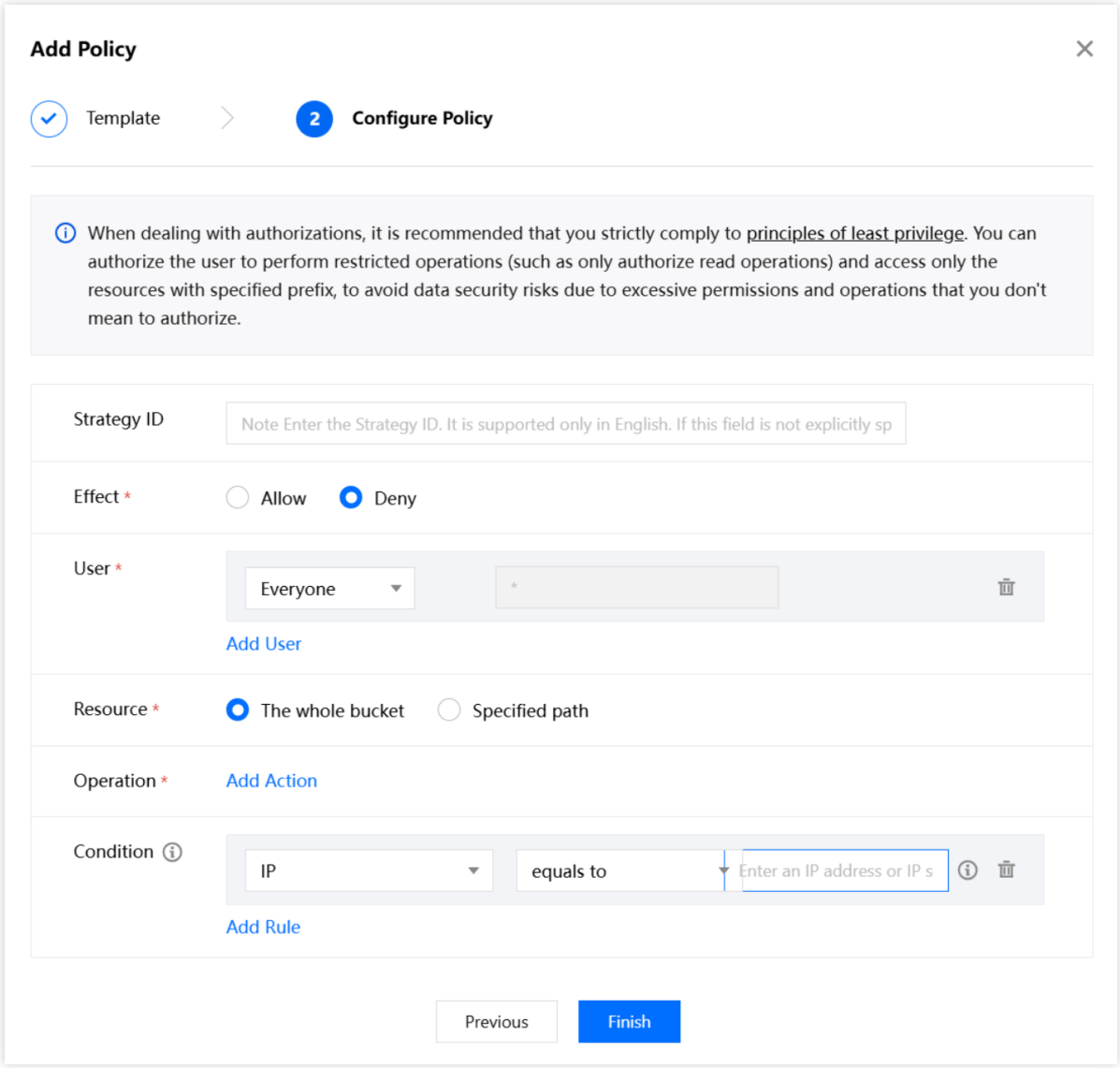


로그 파일에 있는 필드는 로그 관리 개요 문서를 참고할 수 있으며, 아래에서는 자주 사용하는 일부 분석 가능한 필드를 소개하도록 합니다.

userSecretKeyId: 요청이 어느 비밀 키 KeyId를 통해 액세스되었는지를 확정할 수 있으며, 업무 자체에서 개시된 요청이 아닌 것으로 확정되면 비밀 키가 누출되었을 수 있으므로, Tencent Cloud CAM 콘솔로 이동하여 불안전한 비밀 키를 금지할 수 있습니다.

referer: 즉 핫링크 보호 설정에서 판단을 위한 조건이며, 모르는 referer가 발견되면 다른 웹사이트에 의한 핫링크일 수 있으므로, 핫링크 보호-블록리스트를 구성하여 해당 referer 액세스를 제한할 수 있으며, 1.2 버킷 핫링크 보호 활성화를 참고하시기 바랍니다.

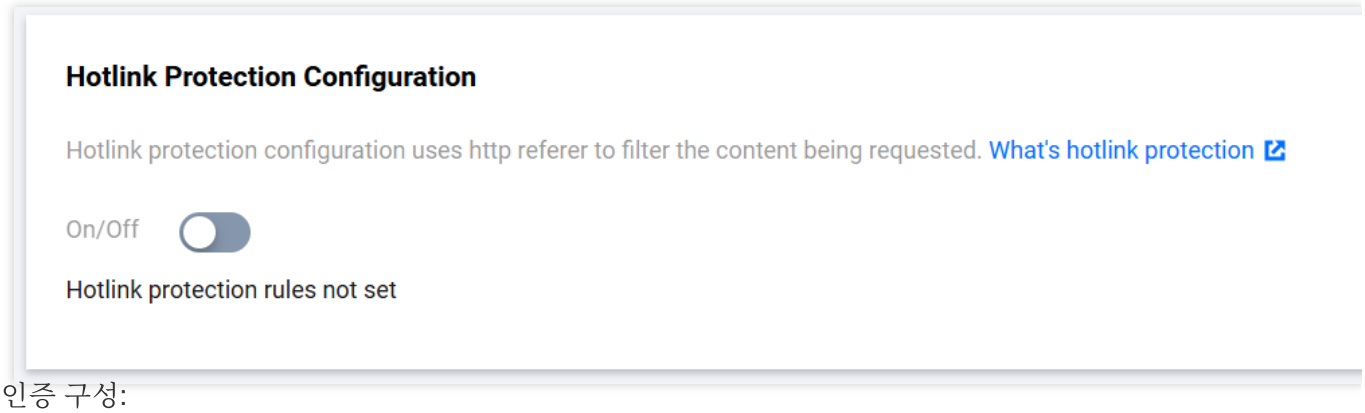
remoteIp: 액세스 출처 IP를 확정할 수 있으며, 신뢰할 수 없는 IP가 발견되면 **버킷 상세정보 페이지-권한 관리-Policy 권한 설정**으로 이동한 후 **정책 추가**를 클릭하여 Policy를 구성하여 해당 IP의 버킷 액세스를 금지할 수 있습니다. 예시는 다음과 같습니다.



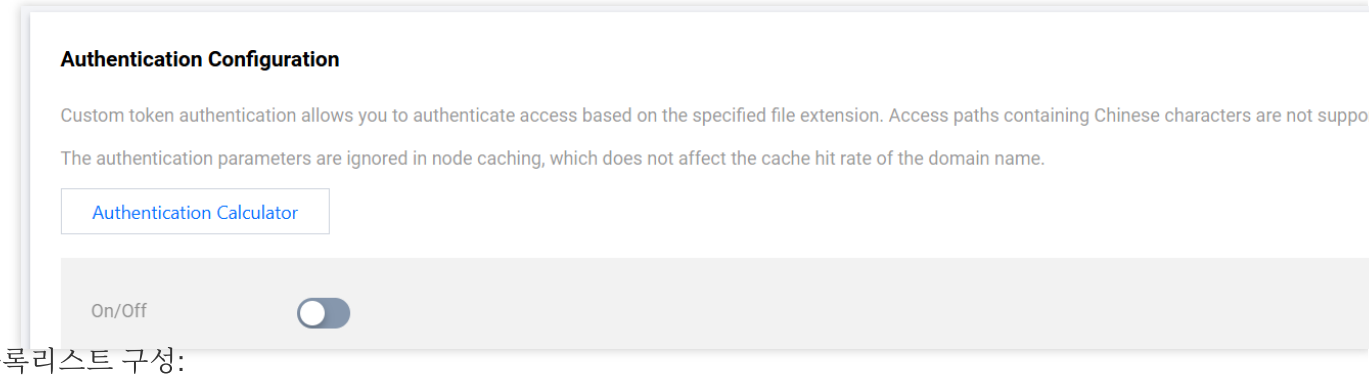
진화된 보호 솔루션

사용자 정의 CDN 가속 도메인 사용

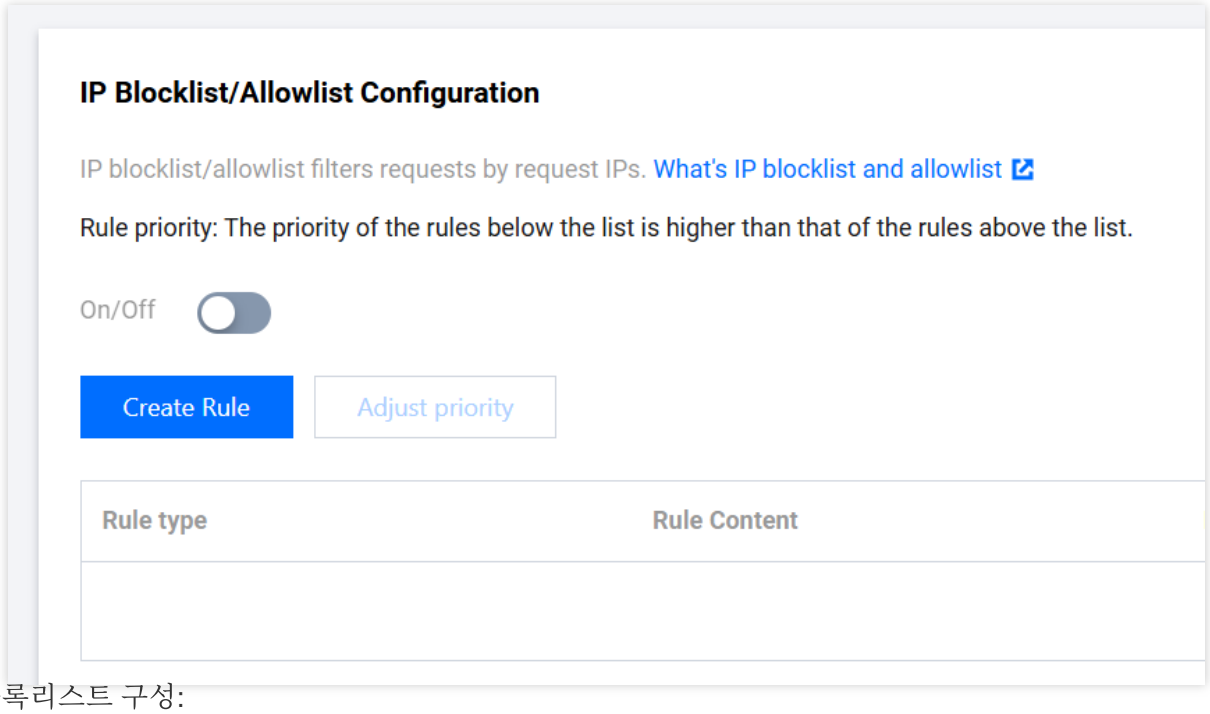
Tencent Cloud CDN도 보호를 위한 많은 구성을 제공하여 도용을 효과적으로 방지할 수 있습니다.
업무에 사용자 정의 CDN 도메인이 사용된 경우, Tencent Cloud CDN 콘솔**-도메인 상세정보-액세스 제어 페이지**로 이동하여 구성할 수 있으며, 자주 사용되는 일부 구성은 다음과 같습니다.
핫링크 보호 구성:



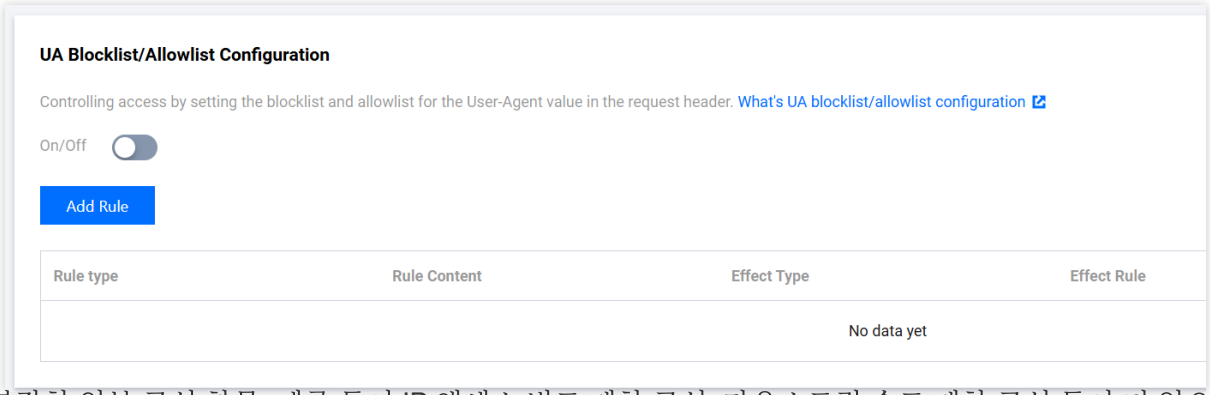
CDN 인증 구성:



IP 블록리스트 구성:



UA 블록리스트 구성:



더욱 복잡한 일부 구성 항목, 예를 들어 IP 액세스 빈도 제한 구성, 다운스트림 속도 제한 구성 등이 더 있으며, CDN 콘솔 관련 문서를 참고하여 사용할 수 있습니다.

SCF+Cloud Monitor+COS API를 통한 자동 블로킹 구현

기본 보호에서 Cloud Monitor 경고를 구성하여 트래픽 이상 상황을 적시에 발견하는 것에 관해 언급했지만, 정보 누락 또는 사용자가 적시에 처리하기 불편한 상황이 존재할 수 있기 때문에 여기서는 자동화 스크립트 코드를 통해 API를 호출함으로써 하나의 간단한 자동화 논리를 구현할 수 있습니다.
Cloud Monitor에서 공중망 다운스트림 트래픽 지표(InternetTraffic) 이상을 획득 -> 자동으로 COS PutBucketAcl를 호출하여 버킷 권한을 개인 읽기/쓰기로 변경. 주로 아래와 같은 2개 API 인터페이스가 사용되며, 여기서는 디버깅 참고용으로 온라인 호출 예시를 제공합니다.
Cloud Monitor의 GetMonitorData - 호출 예시
COS의 PutBucketAcl - 호출 예시

관련 건의

위에서는 일부 도용 문제의 일반적인 보호 수단에 대해 소개하였으며, 이외에도 우리는 일상적으로 COS를 사용할 때 일부 세부 사항에 주의해야 하며, 여기에서는 도용 위험을 어느 정도로 줄일 수 있는 일부 추가적인 사용 건의를 제공합니다.
대외 오픈 소스 코드에서 명문의 API 액세스 비밀 키를 사용하는 것을 피하며, 비밀 키 누출로 인한 안전 위험을 피하기 위해 **최소 권한 원칙**을 참고하여 사용하시는 것을 권장합니다.
크로스 도메인 규칙을 구성할 때 모든 출처(즉 Origin: *) 액세스를 허용하는 것을 피하고, 가능한 한 명확한 출처를 설정합니다.
많은 양의 파일을 업로드할 때, 규칙이 너무 간단한 시퀀스 프리픽스(예: 숫자 번호, 타임스탬프 등)의 사용을 피해야 하며, 그렇지 않으면 공격자가 버킷의 파일을 더 쉽게 순회할 수 있습니다.

링크 도용 방지 사례

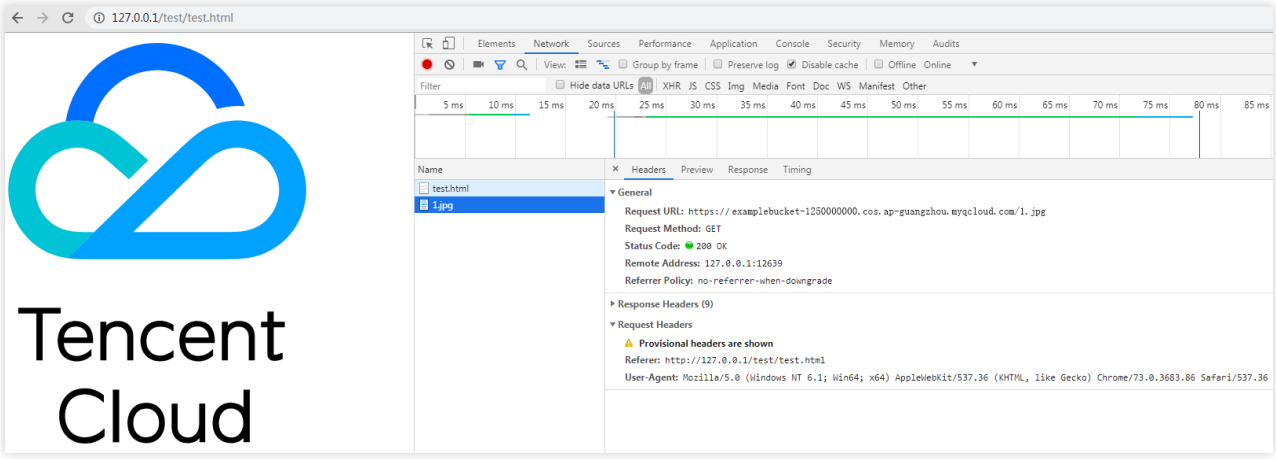
최종 업데이트 날짜: : 2024-06-24 16:53:18

소개

Tencent Cloud COS는 링크 도용 방지 설정을 지원합니다. 사용자는 버킷에 링크 도용 방지 기능을 통해 액세스 출처에 대한 블랙리스트/화이트리스트 설정으로 리소스 도용을 방지할 수 있습니다. 본 문서에서는 버킷에 링크 도용 방지 기능을 설정하여 리소스가 도용되지 않도록 하는 방법을 자세히 소개합니다.

링크 도용 방지 판단 원리

링크 도용 방지는 Header의 Referer 주소를 요청하여 판단합니다.
Referer는 Header의 일부입니다. 브라우저가 Web 서버에 요청을 발송할 때 Referer를 통해 해당 요청이 어느 페이지에서 링크된 것인지 서버에 알리면, 서버에서 특정 출처의 웹 사이트에 대한 리소스 액세스를 금지하거나 허용할 수 있습니다.
브라우저에서 직접 파일 링크 `https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/1.jpg` 를 열 경우, 요청 Header에 Referer가 존재하지 않습니다.
예를 들어, 아래 이미지와 같이 `https://127.0.0.1/test/test.html` 에 이미지 `1.jpg` 를 삽입하고 `https://127.0.0.1/test/test.html` 에 액세스할 경우 액세스 출처를 표시하는 Referer가 존재합니다.

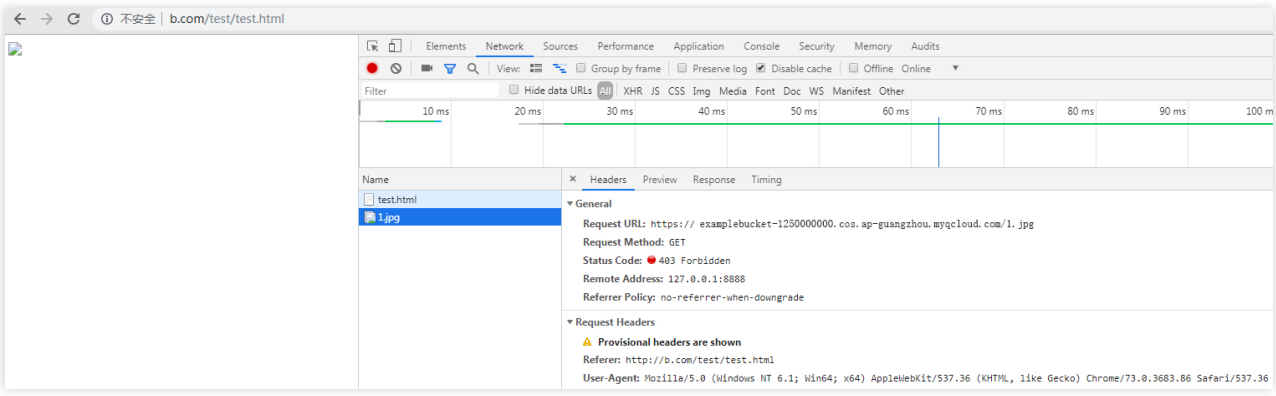


링크 도용 사례 분석

사용자 A가 COS에 이미지 리소스 `1.jpg` 를 업로드하여 이미지 액세스 링크 `https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/1.jpg` 를 획득한 경우,
사용자 A가 해당 이미지를 자신의 웹 페이지 `https://example.com/index.html` 에 삽입하면 이미지에 정상적으로 액세스할 수 있습니다.
사용자 B가 사용자 A의 웹 페이지에서 해당 이미지를 보고 해당 이미지를 자신의 웹 페이지 `https://b.com/test/test.html` 에 삽입하면, 해당 이미지는 사용자 B의 웹 페이지에서도 정상적으로 표시됩니다.
위 사례는 사용자 A의 이미지 리소스 `1.jpg` 의 링크가 사용자 B에 의해 도용된 경우입니다. 이때 사용자 A가 알지 못하는 상황에서 COS 상의 리소스가 사용자 B의 웹 페이지에서 계속 사용되는 경우, 사용자 A는 별도의 트래픽 요금을 부담하는 손실이 발생합니다.

해결 방법

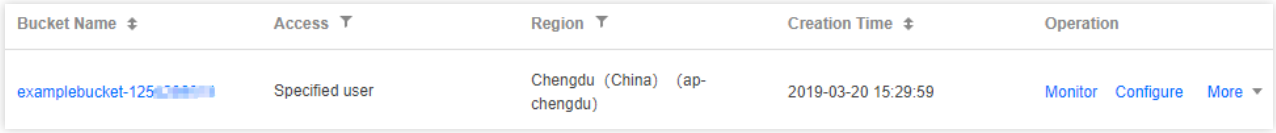
위 링크 도용 사례 분석의 경우 사용자 A는 링크 도용 방지 설정을 통해 사용자 B의 이미지 링크 도용을 방지할 수 있습니다. 자세한 방법은 다음과 같습니다.
1. 사용자 A가 버킷 examplebucket-1250000000에 링크 도용 방지 규칙을 설정합니다. 사용자 B의 링크 도용을 방지하는 방법은 두 가지가 있습니다.
방법1: 블랙리스트 모드를 설정합니다. 도메인 설정에 `*.b.com` 을 입력하고 저장하면 적용됩니다.
방법2: 화이트리스트 모드를 설정합니다. 도메인 설정에 `*.example.com` 을 입력하고 저장하면 적용됩니다.
2. 링크 도용 방지 설정 활성화 후,
`https://example.com/index.html` 에 액세스하면 이미지가 정상적으로 표시됩니다.
`https://b.com/test/test.html` 에 액세스하면 이미지가 표시되지 않으며, 다음과 같이 표시됩니다.



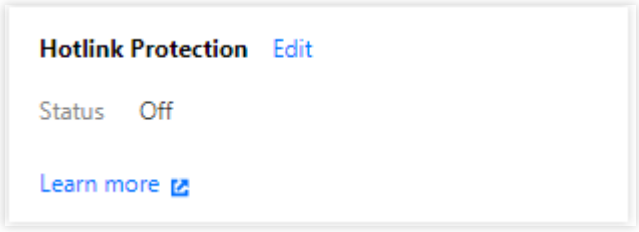
자세한 방법

1. COS 콘솔에 로그인한 후, 왼쪽 메뉴에서 [버킷 리스트]를 클릭해 버킷 리스트 페이지로 이동합니다.

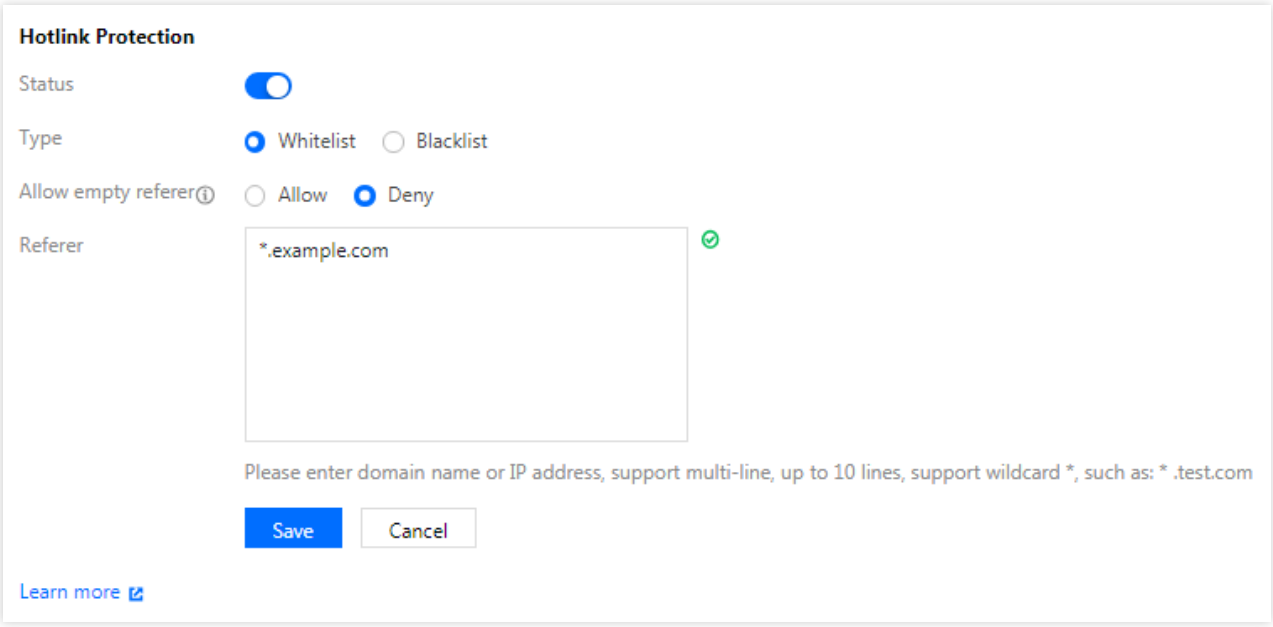
2. 링크 도용 방지를 설정할 버킷을 선택하여 버킷으로 이동합니다.



3. 왼쪽 메뉴에서 [보안 관리]>[링크 도용 방지 설정]을 클릭해 버킷 링크 도용 방지 설정 페이지로 이동합니다.
4. '링크 도용 방지 설정'에서 [편집]을 클릭해 편집 페이지로 이동합니다.



5. 링크 도용 방지를 활성화하고 리스트 유형과 도메인을 설정합니다. 본 문서에서는 방법2를 예시로 설명합니다.
유형: 블랙리스트/화이트리스트 중 선택할 수 있습니다.
블랙리스트: 리스트에 있는 도메인이 버킷의 기본 액세스 주소에 액세스하는 것을 제한합니다. **리스트에 있는** 도메인이 버킷의 기본 액세스 주소에 액세스하는 경우 403을 반환합니다.
화이트리스트: 리스트에 없는 도메인이 버킷의 기본 액세스 주소에 액세스하는 것을 제한합니다. **리스트에 없는** 도메인이 버킷의 기본 액세스 주소에 액세스하는 경우 403을 반환합니다.
Referer: 도메인 설정은 최대 10개의 도메인 및 접두사 매칭이 가능하고 도메인, IP, 와일드카드 부호 `*` 등 형식의 주소를 지원합니다. 주소 1개당 한 행을 차지하며, 주소가 여러 개인 경우 줄 바꿈을 합니다. 규칙 설정에 대한 설명과 예시는 다음과 같습니다.
포트가 있는 도메인 및 IP를 지원합니다(예: `example.com:8080` , `10.10.10.10:8080` 등의 주소).
`example.com` 을 설정하면 `example.com/123` 과 같이 `example.com` 을 접두사로 하는 주소가 히트됩니다.
`example.com` 을 설정하면 `https://example.com` 및 `http://example.com` 을 접두사로 하는 주소가 히트됩니다.
`example.com` 을 설정하면 포트가 있는 도메인 `example.com:8080` 이 히트됩니다.
`example.com:8080` 을 설정하면 도메인 `example.com` 이 히트되지 않습니다.
`*.example.com` 을 설정하면 2단계, 3단계 도메인 `example.com` , `b.example.com` , `a.b.example.com` 을 제한할 수 있습니다.
주의 :
링크 도용 방지를 **활성화**하는 경우 반드시 상응하는 도메인을 입력해야 합니다.
6. 설정 완료 후 [저장]을 클릭하면 활성화됩니다.



FAQ

링크 도용 방지에 대해 궁금한 사항은 COS FAQ의 데이터 보안 문서에서 확인할 수 있습니다.

데이터 검증
MD5 검사

최종 업데이트 날짜: : 2024-06-24 16:53:18

소개

COS는 클라이언트와 서버 간 데이터를 전송할 때 발생할 수 있는 오류에 대해 MD5 검사법을 적용하여 업로드하는 데이터의 무결성을 보장합니다. COS 서버가 수락한 데이터의 MD5 해시값이 사용자가 설정한 MD5 해시값과 일치해야만 데이터를 성공적으로 업로드할 수 있습니다.
COS의 모든 객체는 대응하는 ETag가 하나씩 있습니다. ETag는 객체 생성 시 객체 콘텐츠의 정보를 표시합니다. 하지만 ETag가 객체 콘텐츠의 MD5 해시값과 반드시 동일하지는 않습니다. 따라서 ETag로 다운로드한 객체와 원본 객체의 일치성 여부를 검사할 수 없습니다. 사용자는 사용자 정의한 객체 메타데이터(x-cos-meta-*)로 다운로드한 객체와 원본 객체의 일치성 검사를 실행할 수 있습니다.

데이터 검사 방법

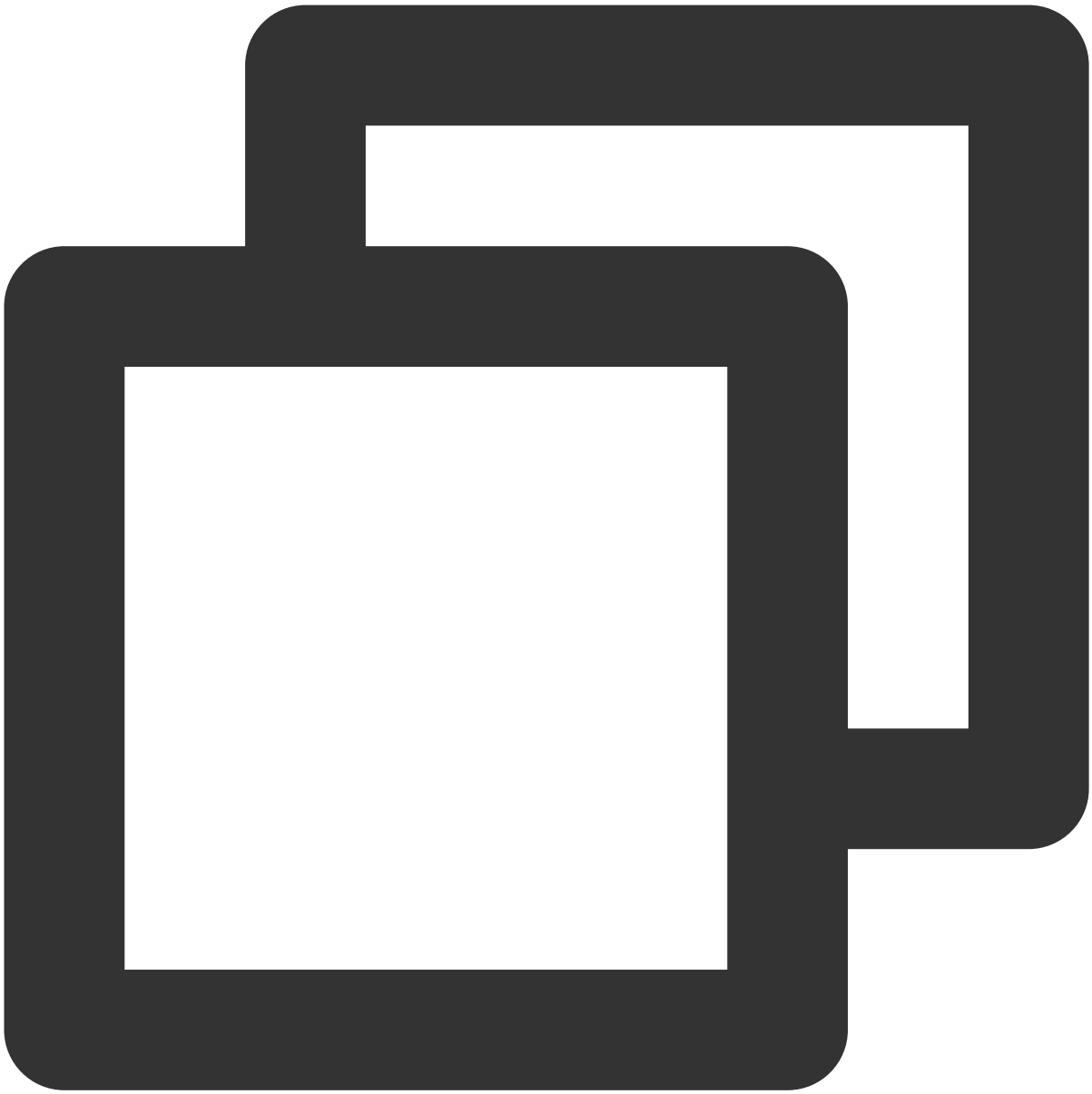
업로드된 객체 검사
COS에 업로드된 객체와 로컬 객체의 일치성 여부를 검사하려면, 업로드 시 HTTP로 요청한 Content-MD5 필드를 Base64로 인코딩된 객체 콘텐츠의 MD5 해시값으로 설정합니다. 이때 COS 서버에서 사용자가 업로드한 객체를 검사하고, COS 서버가 수락한 객체의 MD5 해시값이 사용자가 설정한 Content-MD5와 일치해야만 객체를 성공적으로 업로드할 수 있습니다.
다운로드한 객체 검사
객체 업로드 시 검사 알고리즘으로 객체의 해시값을 계산하여 다운로드한 객체와 원본 객체의 일치 여부를 검사할 수 있습니다. 사용자 정의한 메타데이터로 객체 해시값을 설정한 뒤, 다운로드한 객체의 해시값을 다시 계산하여 사용자 정의한 메타데이터와 비교 인증합니다. 이 경우, 사용자는 검사 알고리즘을 직접 선택할 수 있으며, 같은 객체를 업로드 및 다운로드할 때 사용하는 검사 알고리즘은 동일해야 합니다.

API 예시

간편 업로드 요청

다음은 사용자의 객체 업로드 요청 예시입니다. 객체를 업로드할 때 Content-MD5를 Base64로 인코딩된 객체 콘텐츠의 MD5 해시값으로 설정합니다. 이 때 COS 서버에서 수락한 객체의 MD5 해시값과 사용자가 설정한 Content-MD5가 동일해야 객체를 성공적으로 업로드할 수 있으며, 사용자 정의한 메타데이터 x-cos-meta-md5를 객체의 해시값으로 설정합니다.

설명：
다음 예시는 MD5 검사 알고리즘으로 얻은 객체 해시값이며, 다른 검사 알고리즘을 적용해도 됩니다.

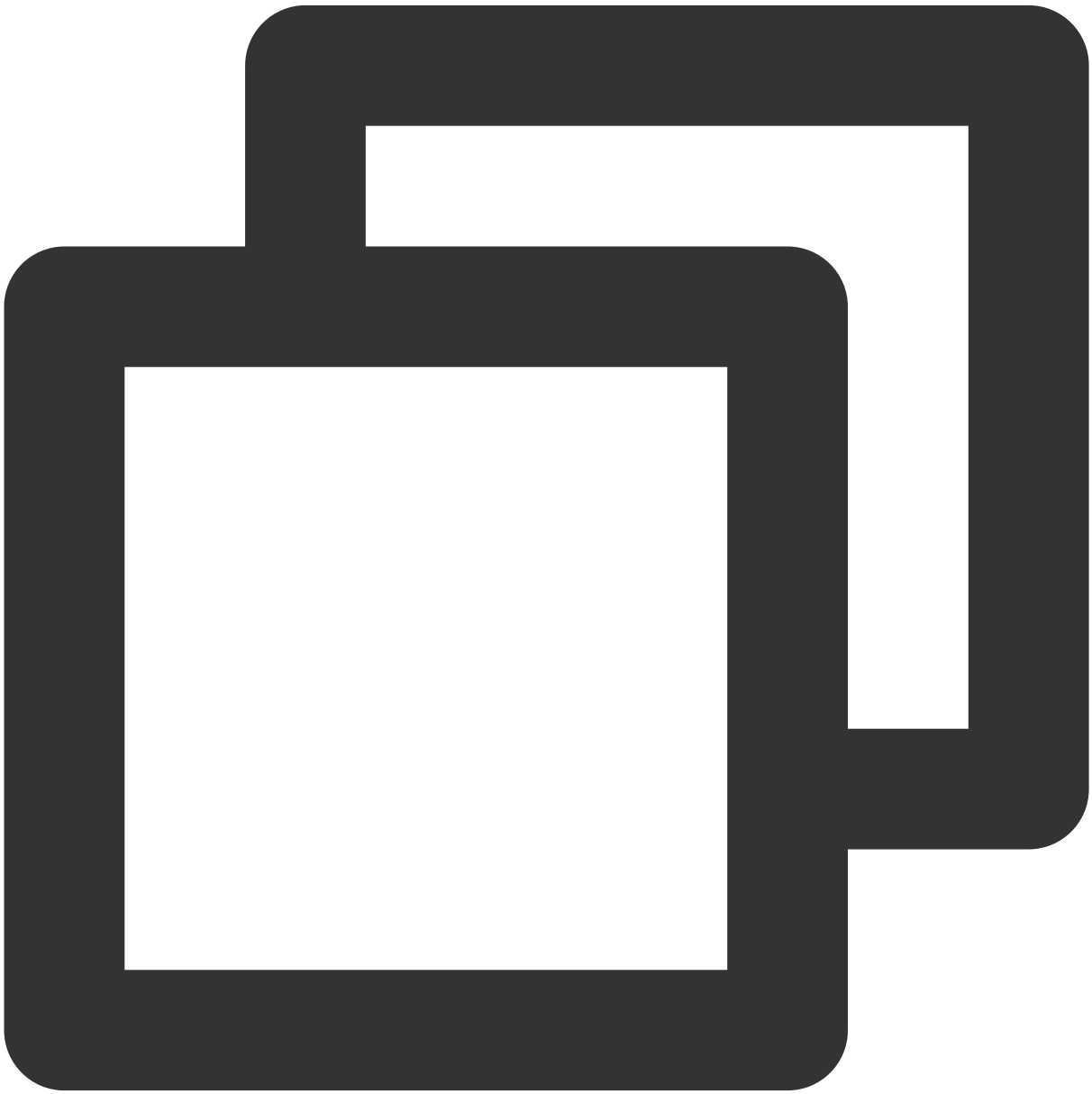


```
PUT /exampleobject HTTP/1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Date: Fri, 21 Jun 2019 09:24:28 GMT
Content-Type: image/jpeg
Content-Length: 13
Content-MD5: ti4QvKtVqIJAvZxDbP/c+Q==
Authorization: q-sign-algorithm=sha1&q-ak=AKID8A0fBVtYFrNm02oY1g1JQQF0c3JO****&q-si
x-cos-meta-md5: b62e10bcab55a88240bd9c436cffdcf9
```

```
Connection: close

[Object Content]
```

멀티파트 업로드 요청

다음은 멀티파트 업로드 초기화에 관한 요청 예시입니다. 객체를 멀티파트 업로드할 때 멀티파트 업로드 초기화를 통해 객체의 사용자 정의 메타데이터를 설정할 수 있습니다. 여기서는 사용자 정의한 메타데이터 x-cos-meta-md5를 객체 해시값으로 설정합니다.



```
POST /exampleobject?uploads HTTP/1.1
```

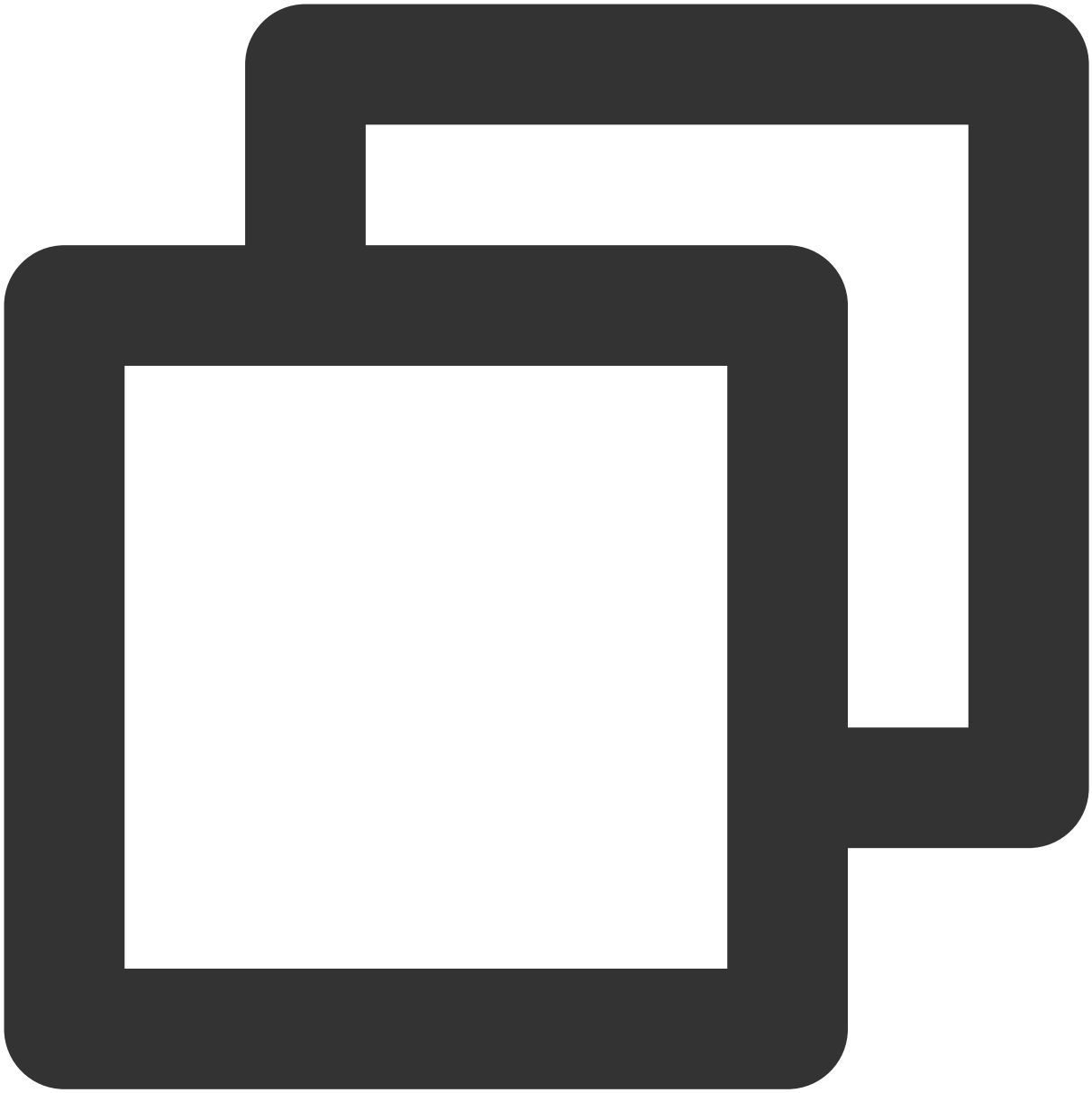
```
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Date: Fri, 21 Jun 2019 09:45:12 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKID8A0fBVtYFrNm02oY1g1JQQF0c3JO****&q-si
x-cos-meta-md5: b62e10bcab55a88240bd9c436cffdcf9
```

주의 :

멀티파트 업로드 파일에 대해 COS는 모든 파트에 대한 MD5 값을 검사하지만 병합한 전체 파일의 MD5 값은 계산하지 않습니다.

객체 다운로드 응답

다음은 사용자가 객체 다운로드 요청으로 얻은 응답 예시입니다. 사용자는 응답으로 객체의 사용자 정의 메타데이터 x-cos-meta-md5를 얻고, 객체 해시값을 재차 계산한 뒤 비교를 통해 다운로드한 객체와 원본 객체의 일치성 여부를 인증합니다.



```
HTTP/1.1 200 OK
Content-Type: application/octet-stream
Content-Length: 13
Connection: close
Accept-Ranges: bytes
Cache-Control: max-age=86400
Content-Disposition: attachment; filename=example.jpg
Date: Thu, 04 Jul 2019 11:33:00 GMT
ETag: "b62e10bcab55a88240bd9c436cffdcf9"
Last-Modified: Thu, 04 Jul 2019 11:32:55 GMT
Server: tencent-cos
```

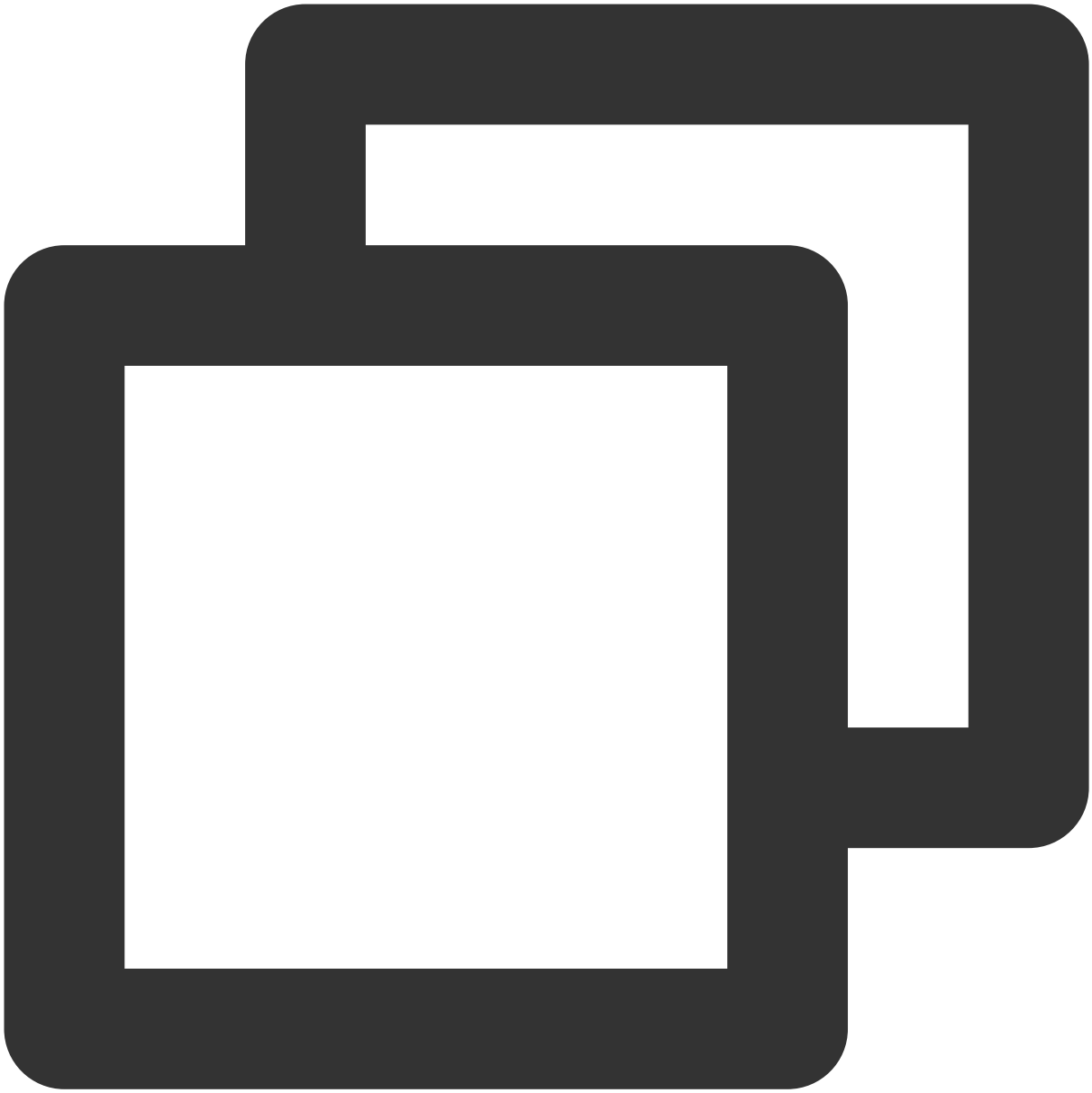
```
x-cos-request-id: NWQxZGUzZWNfNjI4NWQ2NF9lMWYyXzk1NjFj****
x-cos-meta-md5: b62e10bcab55a88240bd9c436cffdcf9

[Object Content]
```

SDK 예시

다음은 Python SDK를 사례로 한 객체 검사법으로, 전체 코드에 대한 예시는 다음과 같습니다.
설명：
코드는 Python 2.7을 기반으로 하였으며, Python SDK의 자세한 사용 방법은 Python SDK의 객체 작업 문서를 참조하십시오.

1. 초기화 설정

SecretId, SecretKey, Region을 포함한 사용자 속성을 설정하고, 클라이언트 객체를 생성합니다.



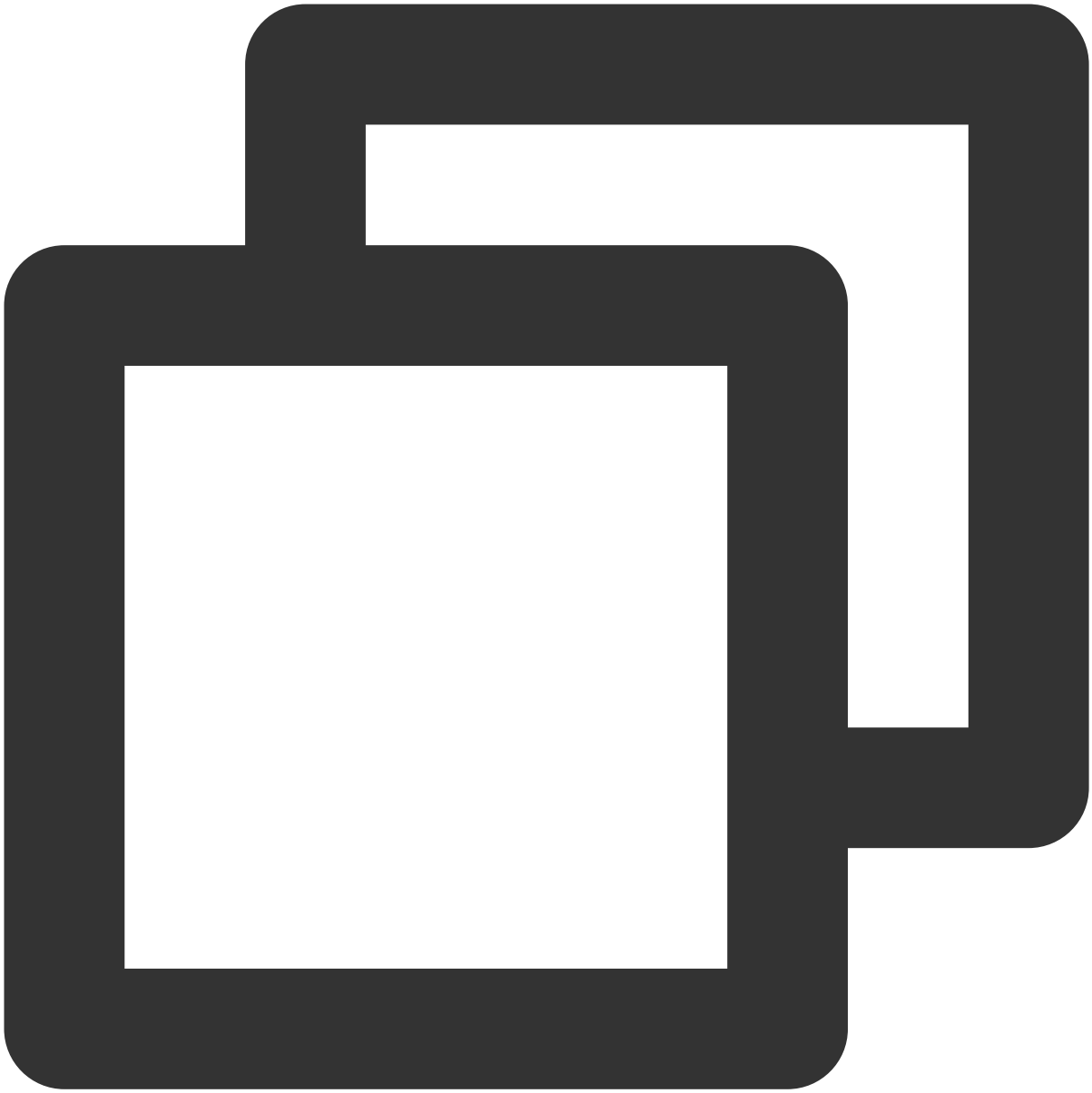
```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
from qcloud_cos import CosServiceError
from qcloud_cos import CosClientError
import sys
import os
import logging
import hashlib

logging.basicConfig(level=logging.INFO, stream=sys.stdout)
```

```
# SecretId, SecretKey, Region을 포함한 사용자 속성을 설정합니다.
# APPID는 설정에서 삭제되었으니 매개변수 Bucket에 APPID를 입력하십시오. Bucket은 BucketName
secret_id = os.environ['COS_SECRET_ID']     # 사용자 SecretId. 리스크를 줄이기 위해 서브
secret_key = os.environ['COS_SECRET_KEY']   # 사용자 SecretKey. 리스크를 줄이기 위해 서
region = 'ap-beijing'      # 사용자 Region으로 대체, 예시는 베이징 리전
token = None               # 임시 키 Token. 임시 키 생성 및 사용 방법에 대한 자세한 내용은
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=t
client = CosS3Client(config)
```

2. 간편한 객체 업로드 검사

(1) 객체의 해시값 계산

MD5 검사 알고리즘으로 객체 해시값을 얻으며, 다른 검사 알고리즘을 적용할 수도 있습니다.

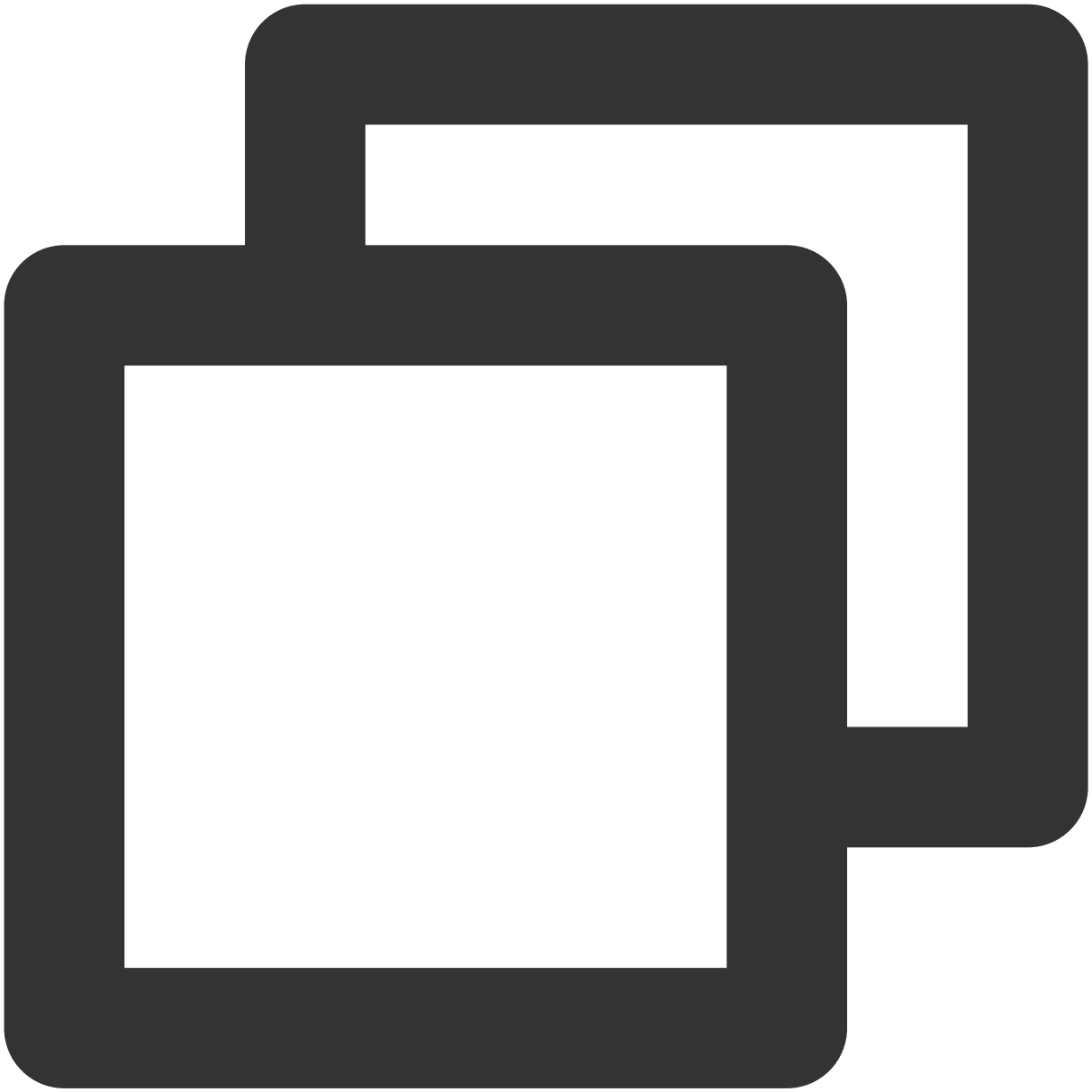


```
object_body = 'hello cos'
#객체의 md5 해시값 획득
md5 = hashlib.md5()
md5.update(object_body)
md5_str = md5.hexdigest()
```

(2) 간편한 객체 업로드

코드 중 EnableMD5=True는 객체 업로드 시 MD5 검사 활성화를 의미합니다. Python SDK에서 Content-MD5를 계산하면 업로드 시간이 다소 지연됩니다. COS 서버가 수락한 객체의 MD5 해시값과 Content-MD5가 일치해야 객체가 성

공적으로 업로드됩니다.
x-cos-meta-md5는 사용자 정의한 매개변수(사용자 정의 매개변수의 이름 포맷은 x-cos-meta-*)이며, 본 예시에서는 객체의 MD5 해시값을 나타냅니다.

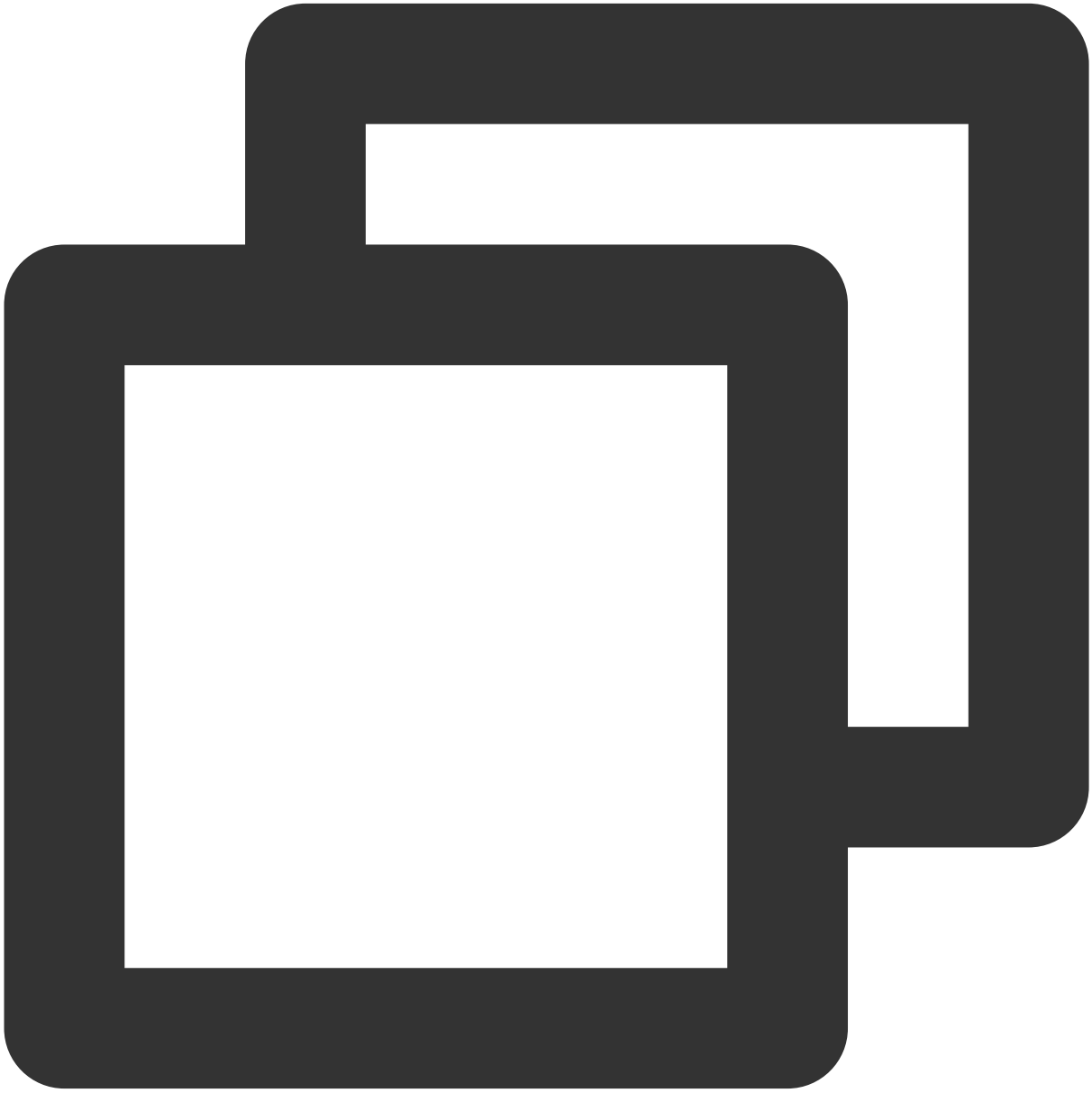


```
#간편한 객체 업로드 및 MD5 검사 활성화
response = client.put_object(
    Bucket='examplebucket-1250000000',      #사용자 Bucket 이름으로 대체. examplebucke
    Body='hello cos',                   #업로드한 객체 콘텐츠
    Key='example-object-1',             #업로드한 객체의 Key 값으로 대체
    EnableMD5=True,                     #업로드한 MD5 검사 활성화
    Metadata={                          #사용자 정의 매개변수 설정. 객체의 MD5 해시값을 매개
```

```
        'x-cos-meta-md5' : md5_str
    }
)
print 'ETag: ' + response['ETag']      # Object의 Etag 값
```

(3) 객체 다운로드

객체를 다운로드하고, 사용자 정의 매개변수를 획득합니다.

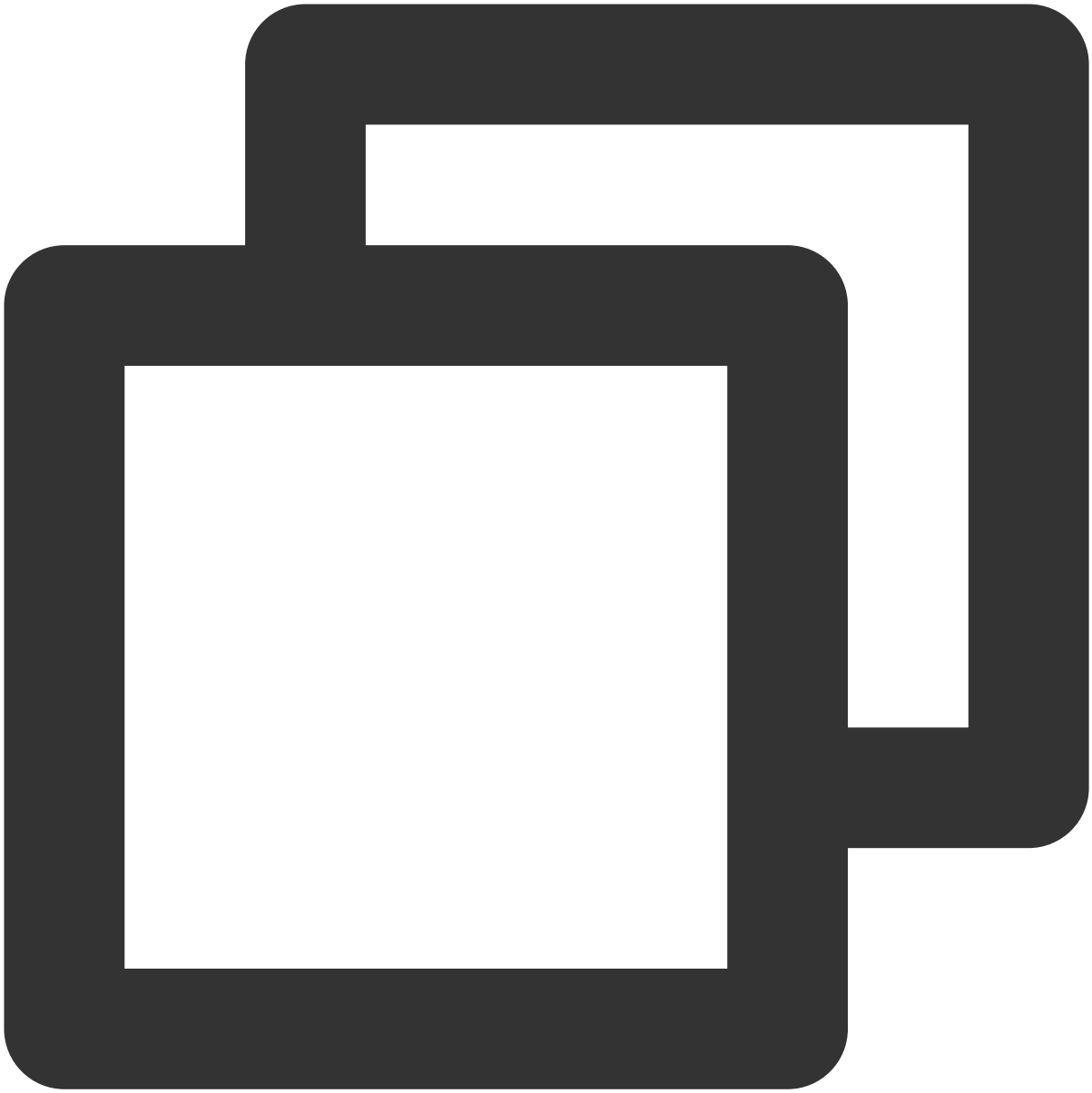


```
#객체 다운로드
response = client.get_object(
```

```
    Bucket='examplebucket-1250000000',       #사용자 Bucket 이름으로 대체. examplebucke
    Key='example-object-1'                   #다운로드한 객체의 Key 값
)
fp = response['Body'].get_raw_stream()
download_object = fp.read()                  #객체 콘텐츠 획득
print "get object body: " + download_object
print 'ETag: ' + response['ETag']
print 'x-cos-meta-md5: ' + response['x-cos-meta-md5']   #사용자 정의 매개변수 x-cos-me
```

(4) 객체 검사

객체를 성공적으로 다운로드한 뒤, 객체 해시값을 재차 계산하여(검사 알고리즘은 객체 업로드 때와 동일해야 함) 사용자 정의 매개변수 x-cos-meta-md5와 비교함으로써 다운로드한 객체와 업로드된 객체의 콘텐츠가 일치하는지 인증합니다.

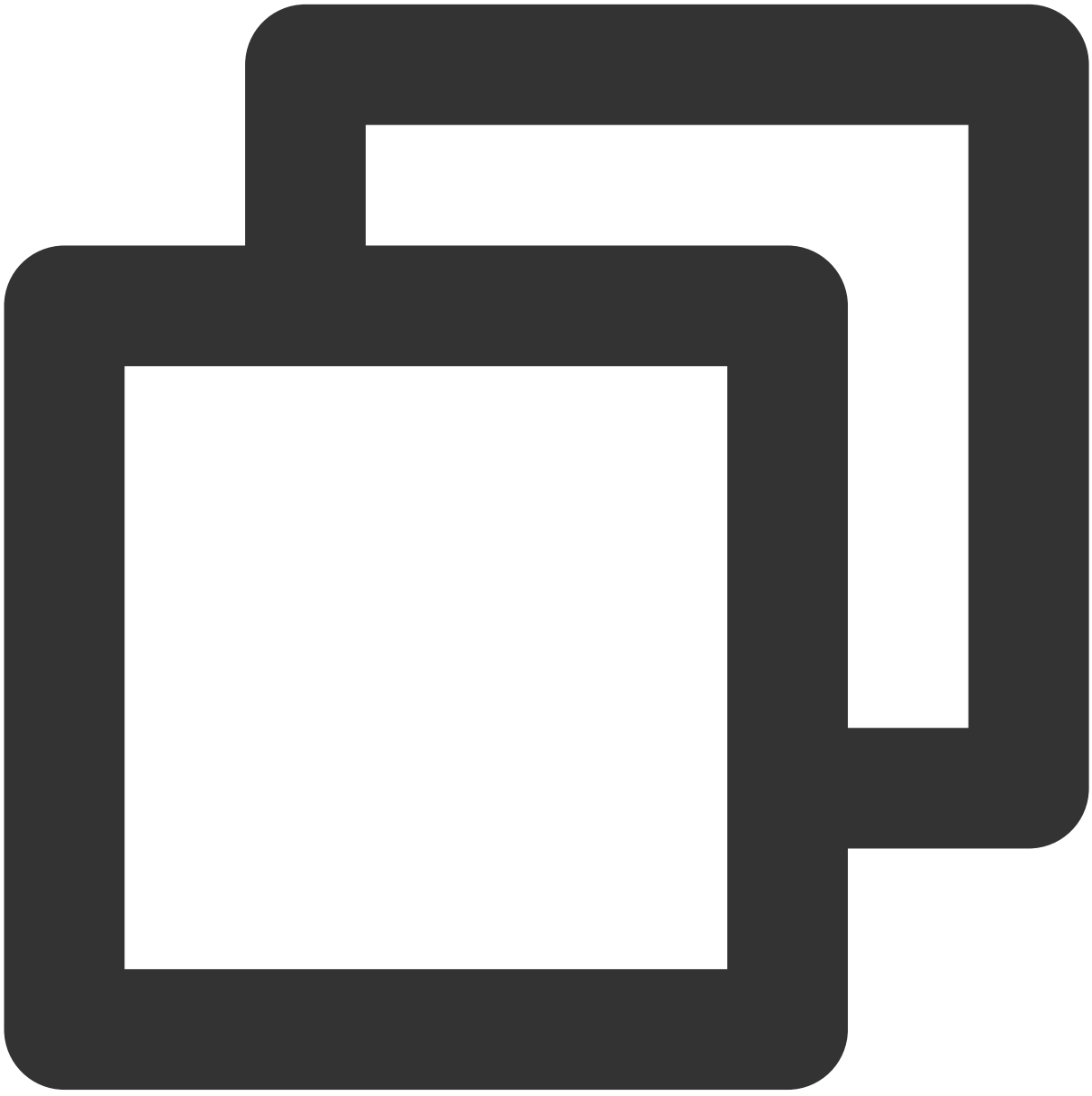


```
#다운로드한 객체의 MD5 해시값 계산
md5 = hashlib.md5()
md5.update(download_object)
md5_str = md5.hexdigest()
print 'download object md5: ' + md5_str

#다운로드한 객체의 MD5 해시값과 업로드된 객체의 MD5 해시값을 비교하여 객체 일치 여부 인증
if md5_str == response['x-cos-meta-md5']:
    print 'MD5 check OK'
else:
    print 'MD5 check FAIL'
```

3. 객체 멀티파트 업로드 검사

(1) 객체의 해시값 계산

객체의 멀티파트를 시뮬레이션하고, 전체 객체의 해시값을 계산합니다. 다음은 MD5 검사 알고리즘으로 얻은 객체의 해시값이며, 다른 검사 알고리즘을 적용할 수도 있습니다.

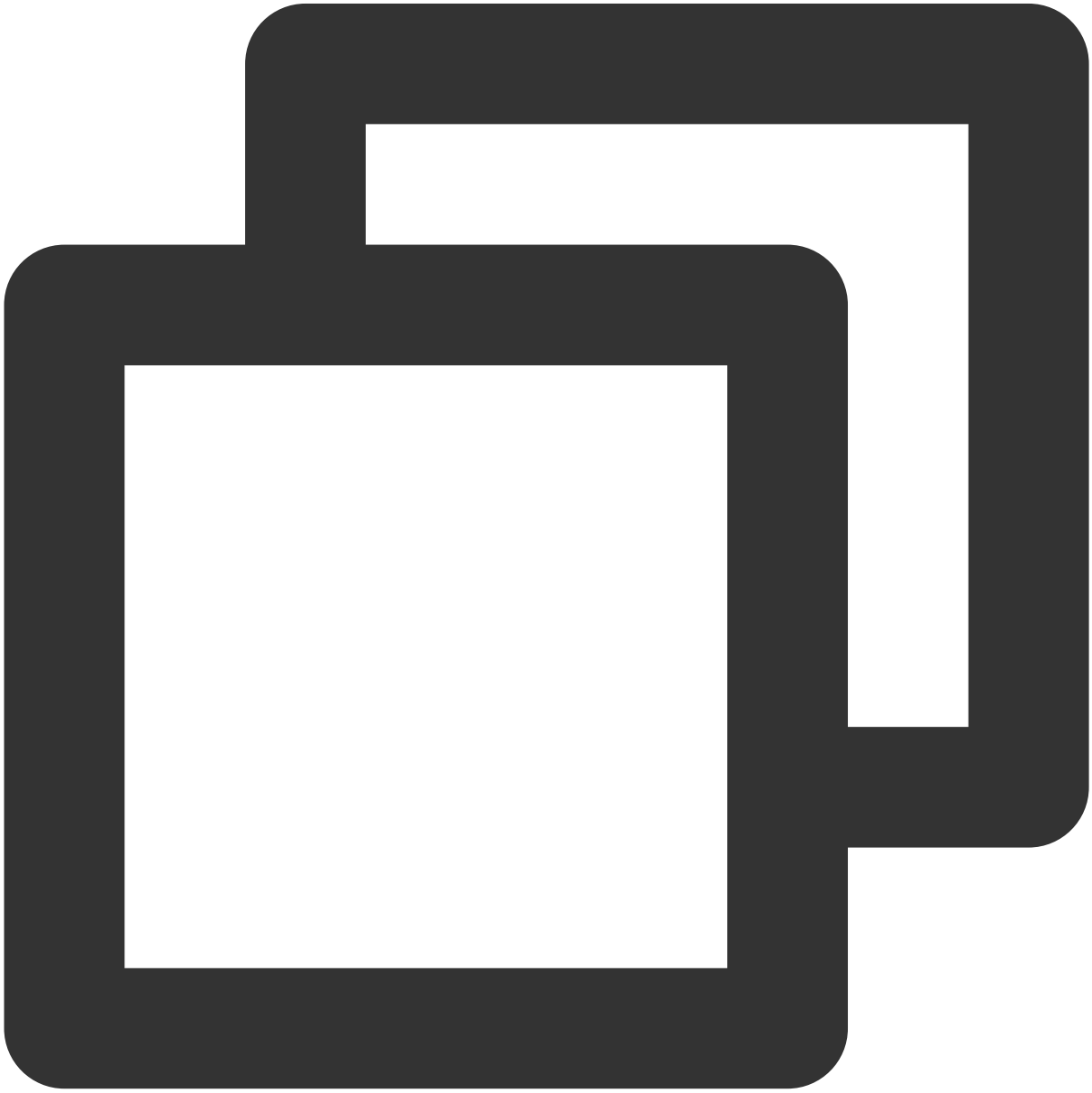


```
OBJECT_PART_SIZE = 1024 * 1024       #각 멀티파트 크기 시뮬레이션
OBJECT_TOTAL_SIZE = OBJECT_PART_SIZE * 1 + 123       #총 객체 크기
object_body = '1' * OBJECT_TOTAL_SIZE        #객체 콘텐츠
```

```
#전체 객체 콘텐츠의 MD5 해시값 계산
md5 = hashlib.md5()
md5.update(object_body)
md5_str = md5.hexdigest()
```

(2) 멀티파트 업로드 초기화

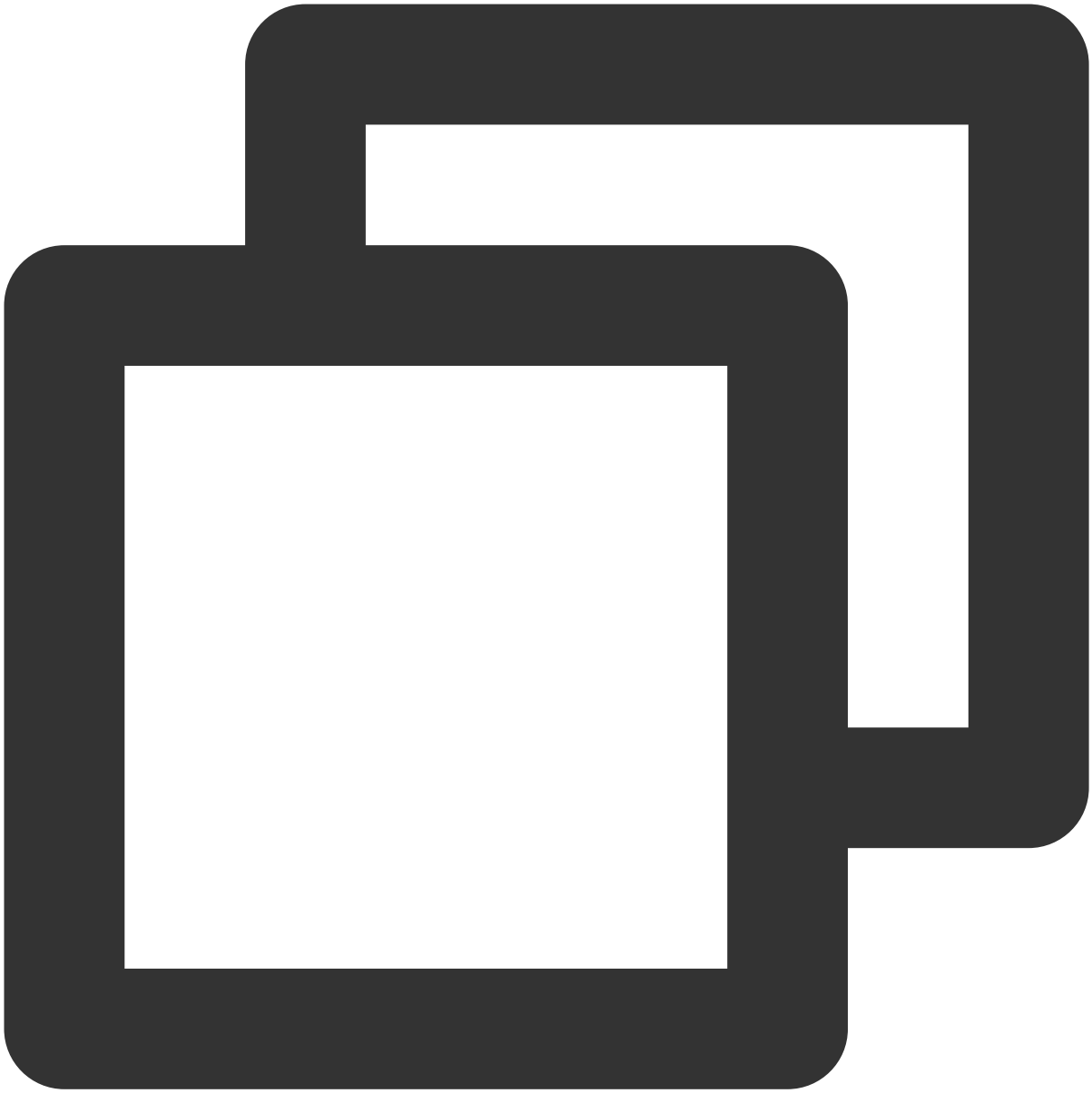
멀티파트 업로드 초기화 시 사용자 정의 매개변수 x-cos-meta-md5를 설정하고, 전체 객체의 MD5 해시값을 매개변수 콘텐츠로 정합니다.



```
#멀티파트 업로드 초기화
response = client.create_multipart_upload(
    Bucket='examplebucket-1250000000',  #사용자 Bucket 이름으로 대체. examplebucket은 
    Key='exampleobject-2',              #업로드한 객체의 Key 값으로 대체
    StorageClass='STANDARD',            #객체의 스토리지 레벨
    Metadata={
        'x-cos-meta-md5' : md5_str      #사용자 정의 매개변수를 MD5 해시값으로 설정
    }
)
#멀티파트 업로드한 UploadId 획득
upload_id = response['UploadId']
```

(3) 객체 멀티파트 업로드

객체 멀티파트 업로드는 객체를 여러 파트로 분할하여 업로드하는 것을 의미합니다. 최대 10000개까지 분할할 수 있으며, 각 파트의 크기는 1MB~5GB, 마지막 파트의 크기는 1MB 미만일 수 있습니다. 멀티파트로 업로드할 때 각 파트의 PartNumber(번호)를 설정해야 합니다. EnableMD5=True는 멀티파트 검사 활성화를 뜻하며, 활성화 시 업로드 시간이 다소 지연됩니다. 이때 Python SDK에서 각 파트의 Content-MD5를 계산하며, COS 서버가 수락한 객체의 MD5 해시값이 Content-MD5와 일치해야 멀티파트를 성공적으로 업로드할 수 있습니다. 업로드에 성공하면 각 파트의 ETag가 반환됩니다.

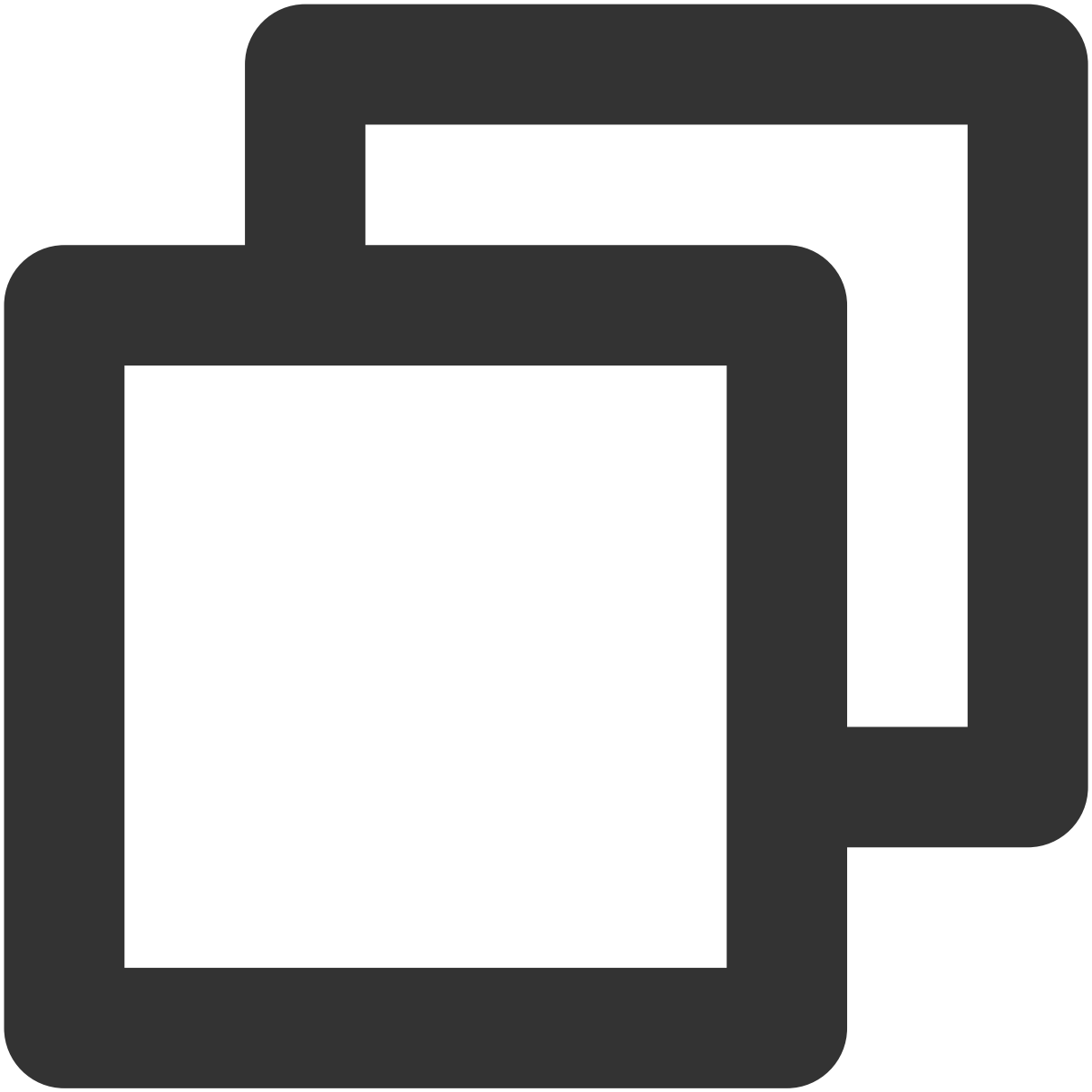


```
#객체를 멀티파트 업로드할 때, 각 파트의 크기는 OBJECT_PART_SIZE이며, 마지막 파트의 크기는 OBJ
part_list = list()
position = 0
left_size = OBJECT_TOTAL_SIZE
part_number = 0
while left_size > 0:
    part_number += 1
    if left_size >= OBJECT_PART_SIZE:
        body = object_body[position:position+OBJECT_PART_SIZE]
    else:
        body = object_body[position:]
```

```
    position += OBJECT_PART_SIZE
    left_size -= OBJECT_PART_SIZE

    #멀티파트 업로드
    response = client.upload_part(
        Bucket='examplebucket-1250000000',  #사용자 Bucket 이름으로 대체. examplebucke
        Key='exampleobject-2',              #객체의 Key 값
        Body=body,
        PartNumber=part_number,
        UploadId=upload_id,
        EnableMD5=True       #멀티파트 검사 활성화. COS 서버가 각 파트에 MD5 검사 실시.
    )
    etag = response['ETag']     #ETag는 각 파트의 MD5 값 표시
    part_list.append({'ETag' : etag, 'PartNumber' : part_number})
    print etag + ', ' + str(part_number)
```

(4) 멀티파트 업로드 완료

모든 멀티파트 업로드를 마치면 이를 완료하는 작업을 수행해야 합니다. 각 파트의 ETag와 PartNumber가 모두 대응되어야 하며, COS 서버는 이를 각 파트의 정확성 검사에 활용합니다. 멀티파트 업로드가 끝난 후 반환되는 ETag는 전체 객체 콘텐츠의 MD5 해시값이 아닌 병합된 객체의 고유 태그값입니다. 객체를 다운로드할 때는 사용자 정의 매개변수를 통해 검사하십시오.

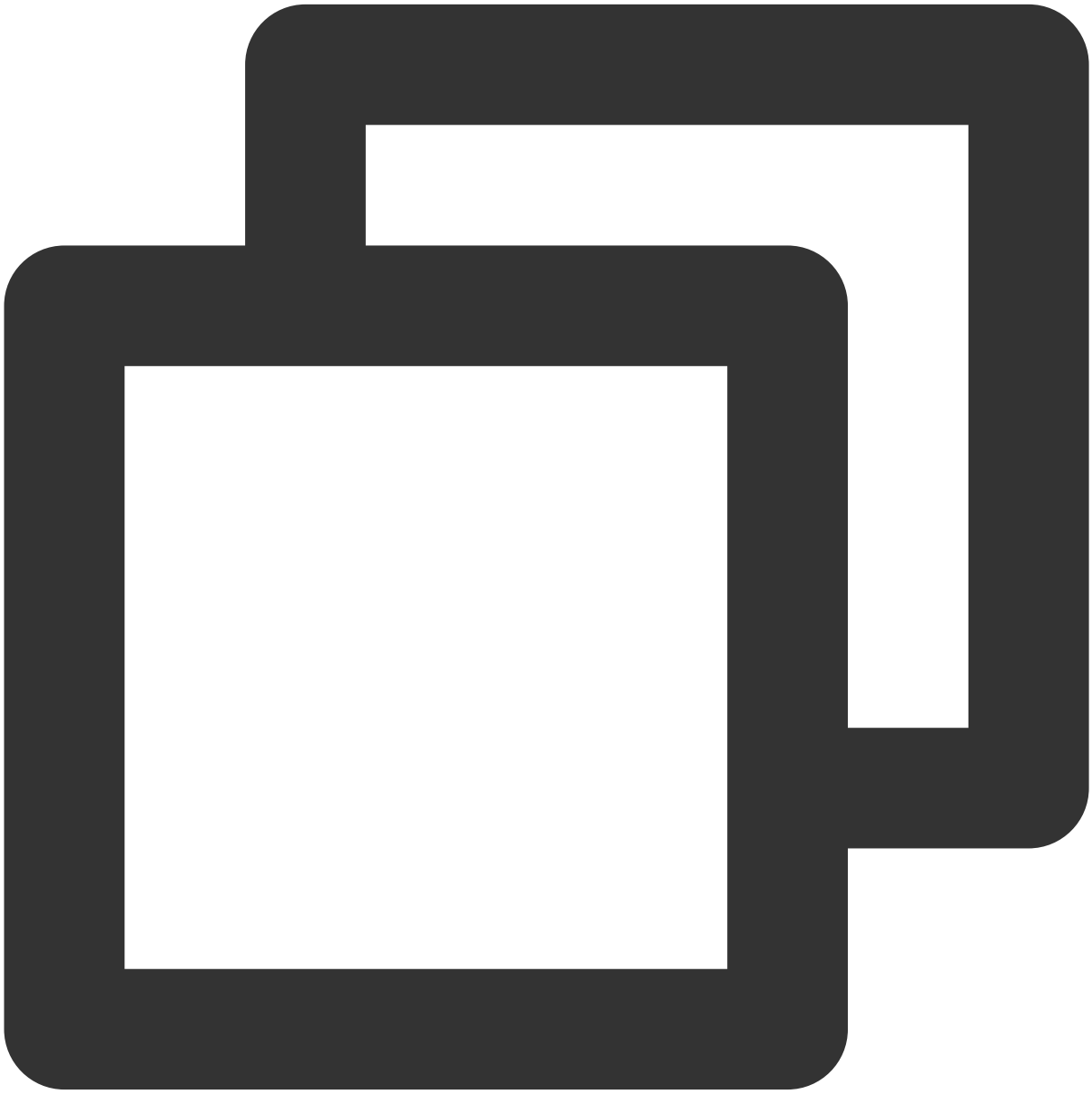


```
#멀티파트 업로드 완료
response = client.complete_multipart_upload(
    Bucket='examplebucket-1250000000',  #사용자 Bucket 이름으로 대체. examplebucket은
    Key='exampleobject-2',              #객체의 Key 값
    UploadId=upload_id,
    MultipartUpload={       #모든 파트의 ETag와 PartNumber가 모두 대응해야 함
        'Part' : part_list
    },
)

#ETag는 병합한 객체의 고유성을 검사하기 위한 고유 태그값으로, 객체 콘텐츠의 MD5 해시값이 아닙니
```

```
print "ETag: " + response['ETag']
print "Location: " + response['Location']    #URL 주소
print "Key: " + response['Key']
```

(5) 객체 다운로드

객체를 다운로드하고, 사용자 정의 매개변수를 획득합니다.

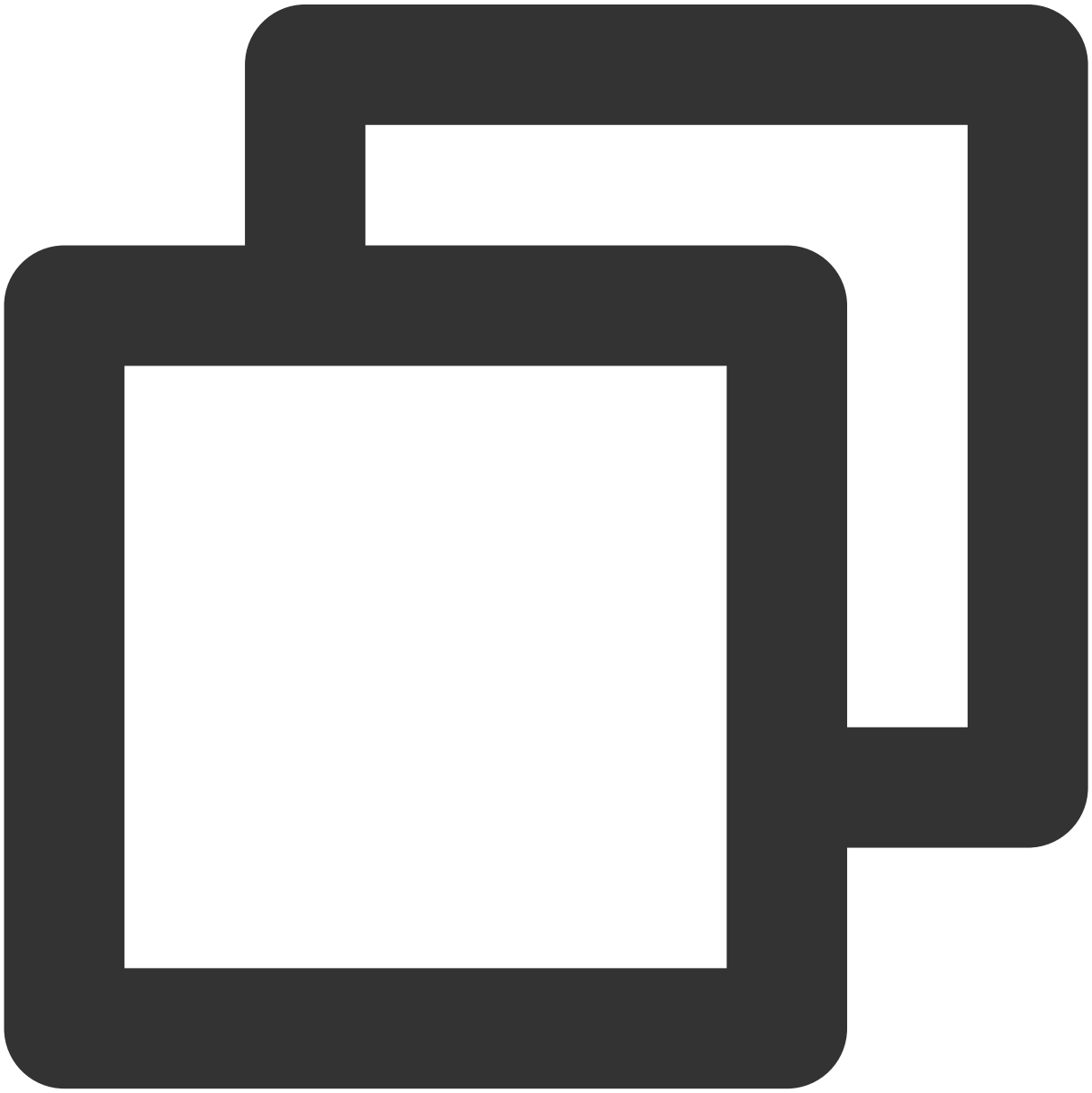


```
# 객체 다운로드
response = client.get_object(
    Bucket='examplebucket-1250000000',  #사용자 Bucket 이름으로 대체. examplebucket은 
```

```
    Key='exampleobject-2'                #객체의 Key 값
)
print 'ETag: ' + response['ETag']                              #객체의 ETag는 객체 콘텐츠의 M
print 'x-cos-meta-md5: ' + response['x-cos-meta-md5']          #사용자 정의 매개변수 x-cos-m
```

(6) 객체 검사

객체를 성공적으로 다운로드한 뒤, 객체의 MD5 해시값을 재차 계산하여 사용자 정의 매개변수 x-cos-meta-md5와 비교함으로써 다운로드한 객체와 업로드된 객체의 콘텐츠가 일치하는지 인증합니다.



```
#다운로드한 객체의 MD5 해시값 계산
```

```
fp = response['Body'].get_raw_stream()
DEFAULT_CHUNK_SIZE = 1024*1024
md5 = hashlib.md5()
chunk = fp.read(DEFAULT_CHUNK_SIZE)
while chunk:
    md5.update(chunk)
    chunk = fp.read(DEFAULT_CHUNK_SIZE)
md5_str = md5.hexdigest()
print 'download object md5: ' + md5_str

#다운로드한 객체의 MD5 해시값과 업로드된 객체의 MD5 해시값을 비교하여 객체 일치 여부 인증
if md5_str == response['x-cos-meta-md5']:
    print 'MD5 check OK'
else:
    print 'MD5 check FAIL'
```

CRC64 검사

최종 업데이트 날짜: : 2024-06-24 16:53:18

소개

클라이언트와 서버 간에 데이터가 전송되면서 데이터에 오류가 발생하기도 합니다. COS는 MD5 검증으로 데이터 무결성을 검증하며, CRC64 검사 코드를 이용해 데이터 검사를 합니다.
COS는 새로 업로드되는 객체에 CRC64 계산을 실시하고, 그 결과를 객체의 속성으로 보관합니다. 이후 반환되는 응답 헤더에 x-cos-hash-crc64ecma가 포함되는데, 이 헤더는 업로드한 객체의 CRC64 값을 나타내며 ECMA-182 표준에 따라 계산하여 값을 얻습니다. CRC64의 특성상 런칭 이전 시점부터 COS의 객체에 존재하기 때문에 COS에서 객체의 CRC64 값을 계산하지 않습니다. 따라서 해당 유형의 객체를 획득할 때 CRC64 값이 반환되지 않습니다.

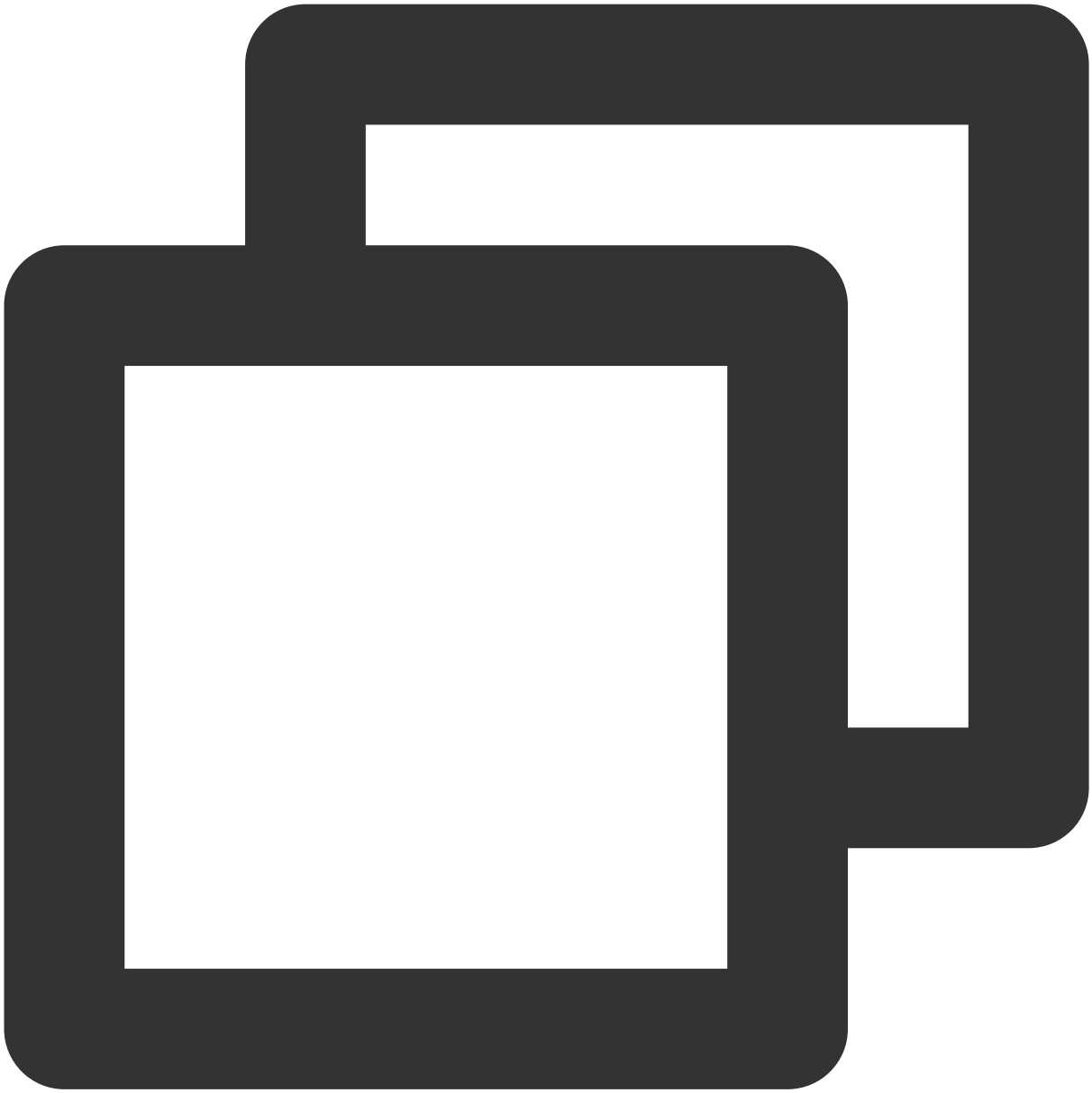
작업 설명

현재 CRC64를 지원하는 API는 다음과 같습니다.
간편 업로드 인터페이스
PUT Object와 POST Object: 반환된 응답 헤더에서 파일의 CRC64 검사 값을 획득할 수 있습니다.
멀티파트 업로드 인터페이스
Upload Part: COS에서 반환된 CRC64 값과 로컬에서 계산된 값을 비교 검증합니다.
Complete Multipart Upload: 멀티파트 각각에 CRC64 속성이 있으면 객체의 CRC64 값이 반환됩니다. 하지만 일부 멀티파트에 CRC64 값이 없는 경우, 반환되지 않습니다.
Upload Part - Copy를 실행하면 이와 대응하는 CRC64 값이 반환됩니다.
PUT Object - Copy를 실행할 때 원본 객체에 CRC64 값이 있으면 CRC64가 반환되고, 그렇지 않으면 반환되지 않습니다.
HEAD Object와 GET Object를 실행할 때 객체에 CRC64가 있으면 반환됩니다. 사용자는 COS에서 반환된 CRC64 값과 로컬에서 계산된 CRC64 값을 비교 검증할 수 있습니다.

API 예시

멀티파트 업로드 응답

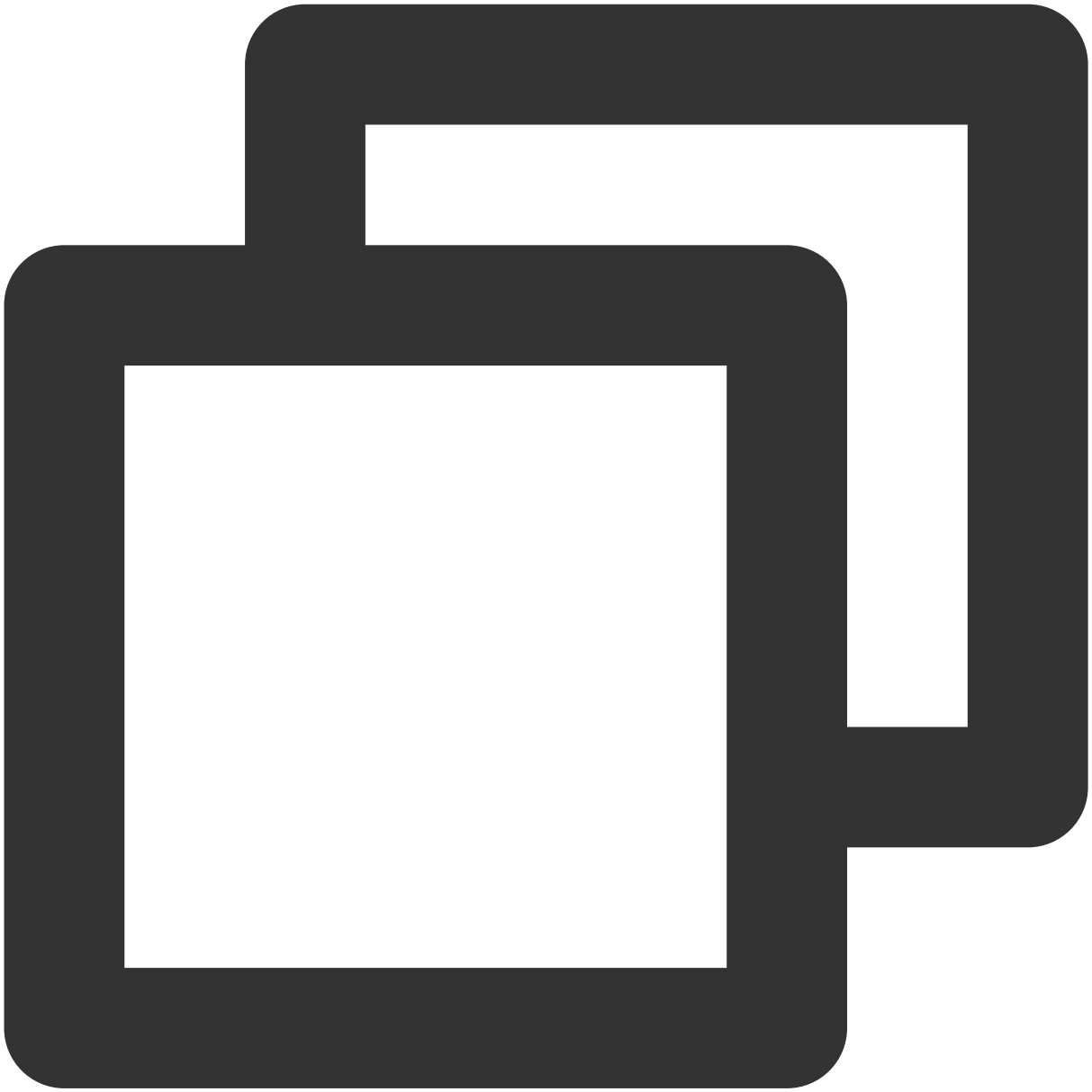
다음은 사용자가 Upload Part 요청을 한 뒤 얻은 응답 예시입니다. x-cos-hash-crc64ecma 헤더는 멀티파트의 CRC64 값을 나타냅니다. 해당 값과 로컬에서 계산한 CRC64 값을 비교하여 멀티파트의 무결성을 검사할 수 있습니다.



```
HTTP/1.1 200 OK
content-length: 0
connection: close
date: Thu, 05 Dec 2019 01:58:03 GMT
etag: "358e8c8b1bfa35ee3bd44cb3d2cc416b"
server: tencent-cos
x-cos-hash-crc64ecma: 15060521397700495958
x-cos-request-id: NWRlODY0MmJfMjBiNDU4NjRfNjkyZl80ZjZi****
```

멀티파트 업로드 완료 응답

다음은 사용자가 Complete Multipart Upload 요청을 한 뒤 얻은 응답 예시입니다. x-cos-hash-crc64ecma 헤더는 전체 객체의 CRC64 값을 나타냅니다. 해당 값과 로컬에서 계산한 CRC64 값을 비교하여 객체 무결성 검사를 할 수 있습니다.



```
HTTP/1.1 200 OK
content-type: application/xml
transfer-encoding: chunked
connection: close
date: Thu, 05 Dec 2019 02:01:17 GMT
server: tencent-cos
x-cos-hash-crc64ecma: 15060521397700495958
```

```
x-cos-request-id: NWRlODY0ZWRfMjNiMjU4NjRfOGQ4Ml81MDEw****

[Object Content]
```

SDK 예시

Python SDK

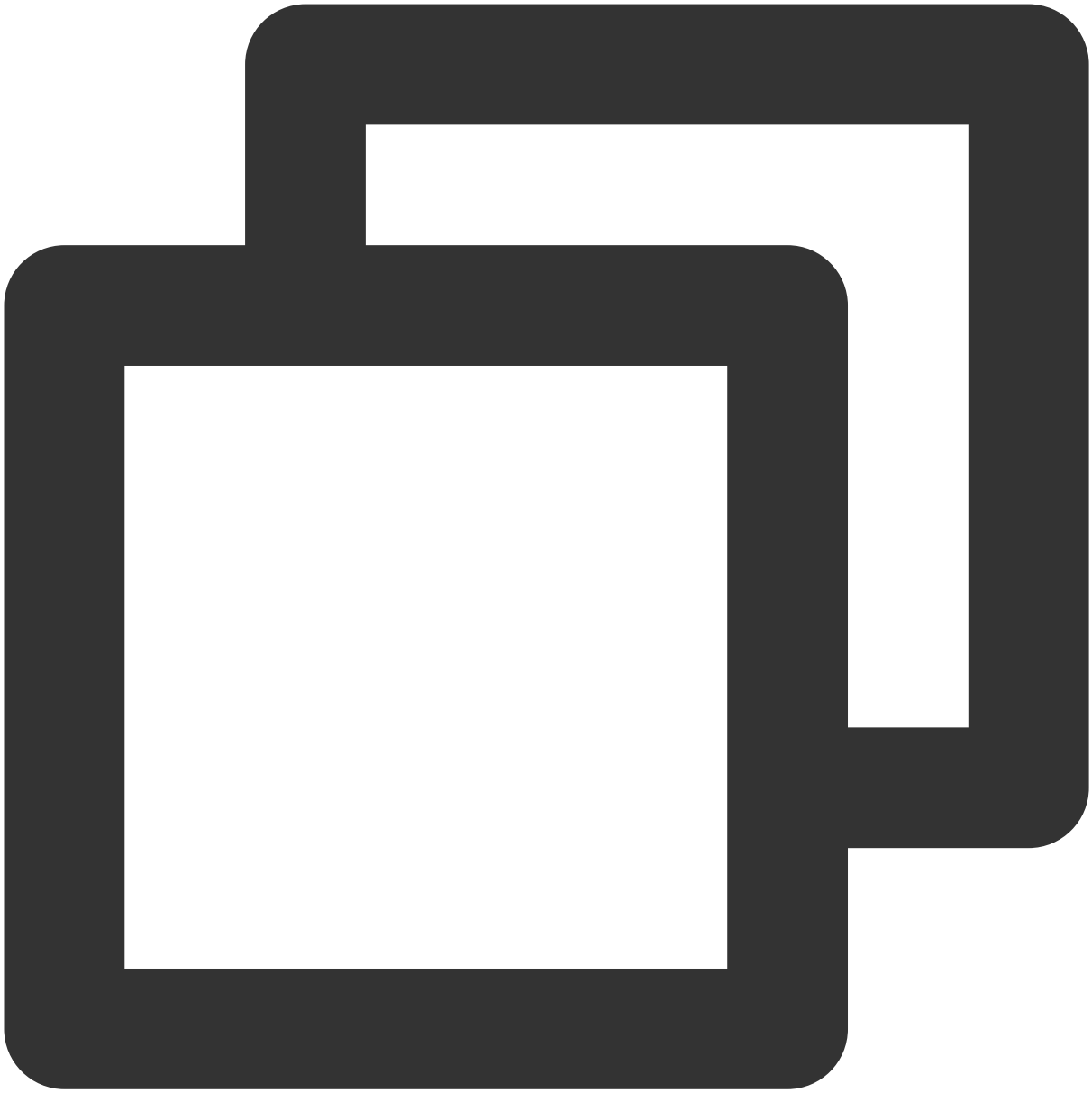
다음은 Python SDK를 사례로 한 객체 검사법으로, 전체 코드에 대한 예시는 다음과 같습니다.

설명：

코드는 Python 2.7을 기반으로 하였으며, Python SDK의 자세한 사용 방법은 Python SDK의 객체 작업 문서를 참고하십시오.

1. 초기화 설정

SecretId, SecretKey, Region을 포함한 사용자 속성을 설정하고, 클라이언트 객체를 생성합니다.

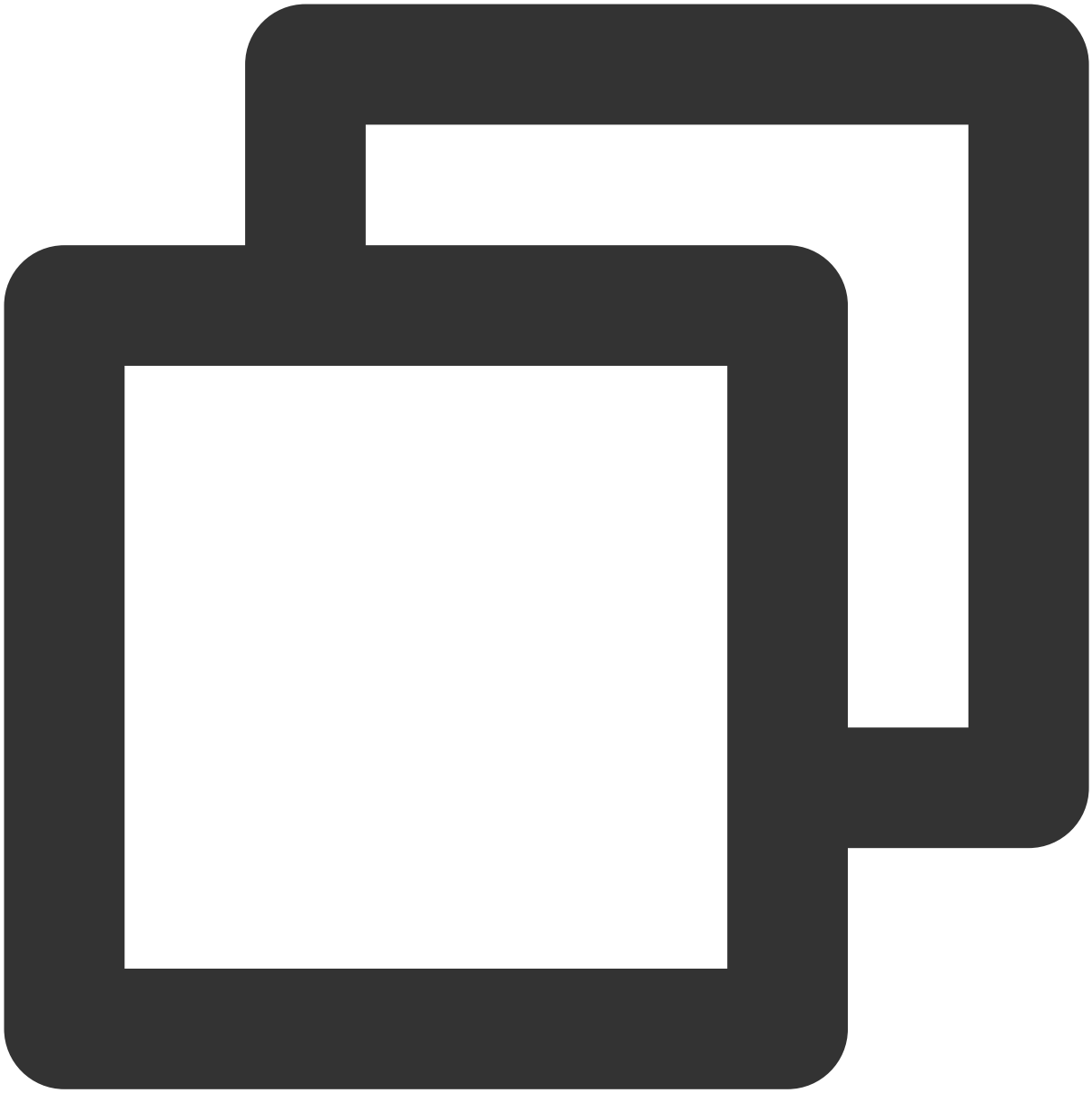


```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
from qcloud_cos import CosServiceError
from qcloud_cos import CosClientError
import sys
import logging
import hashlib
import crcmod

logging.basicConfig(level=logging.INFO, stream=sys.stdout)
```

```
# SecretId, SecretKey, Region을 포함한 사용자 속성을 설정합니다.
# APPID는 설정에서 삭제되었으니 매개변수 Bucket에 APPID를 입력하십시오. Bucket은 BucketName
secret_id = COS_SECRETID           # 사용자 SecretId 정보로 대체
secret_key = COS_SECRETKEY         # 사용자 SecretKey 정보로 대체
region = 'ap-beijing'      # 사용자 Region으로 대체, 예시는 베이징 리전
token = None               # 임시 키를 사용할 경우 Token 입력, 기본값이 null이면 입력하지
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=t
client = CosS3Client(config)
```

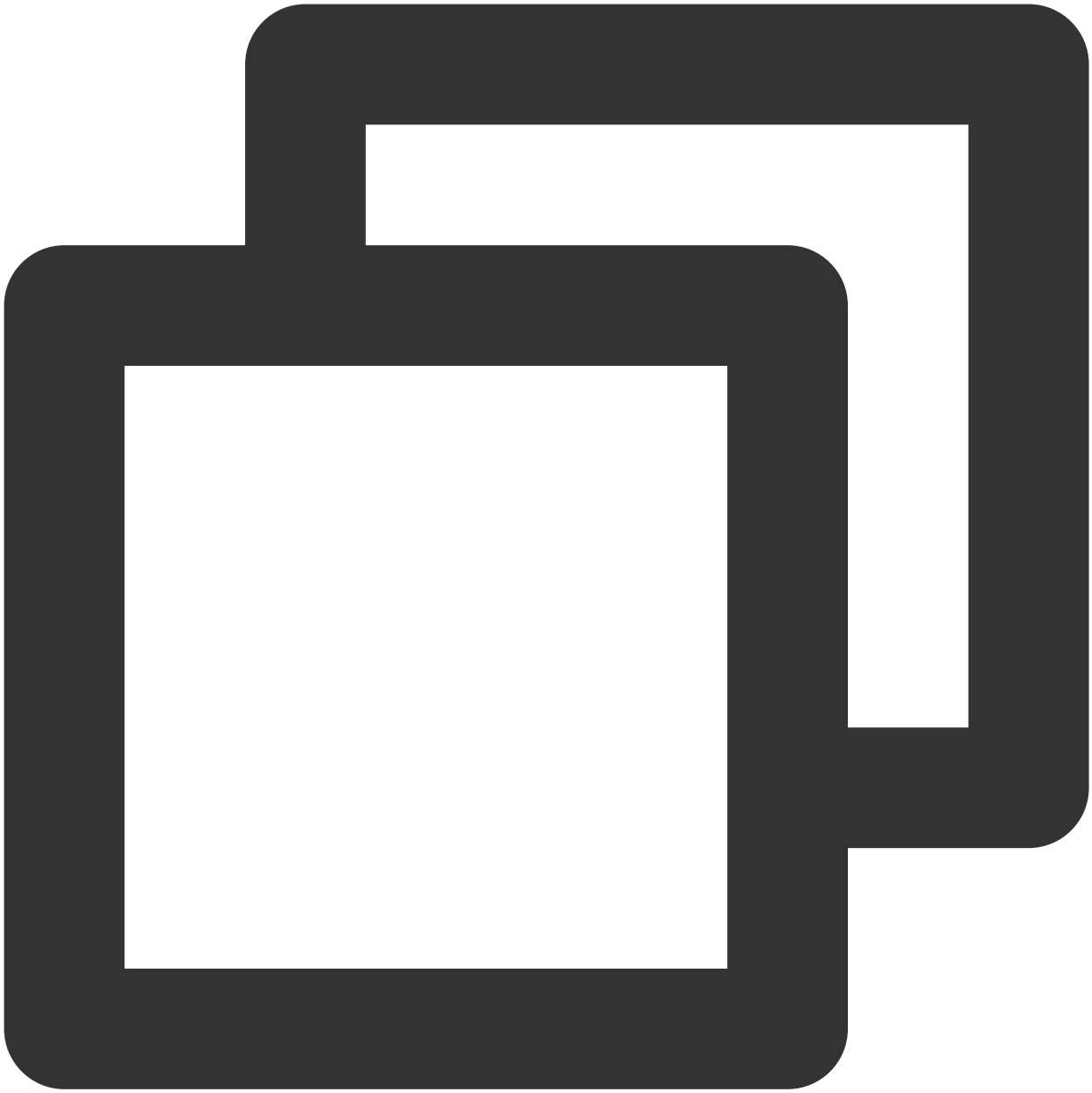
2. 객체의 검사 값 계산

객체의 멀티파트를 시뮬레이션하고, 전체 객체의 CRC64 검사 값을 계산합니다.



```
OBJECT_PART_SIZE = 1024 * 1024       #각 멀티파트 크기 시뮬레이션
OBJECT_TOTAL_SIZE = OBJECT_PART_SIZE * 1 + 123       #총 객체 크기
object_body = '1' * OBJECT_TOTAL_SIZE        #객체 콘텐츠

#전체 객체의 crc64 검사 값 계산
c64 = crcmod.mkCrcFun(0x142F0E1EBA9EA3693L, initCrc=0L, xorOut=0xffffffffffffffffL,
local_crc64 =str(c64(object_body))
```

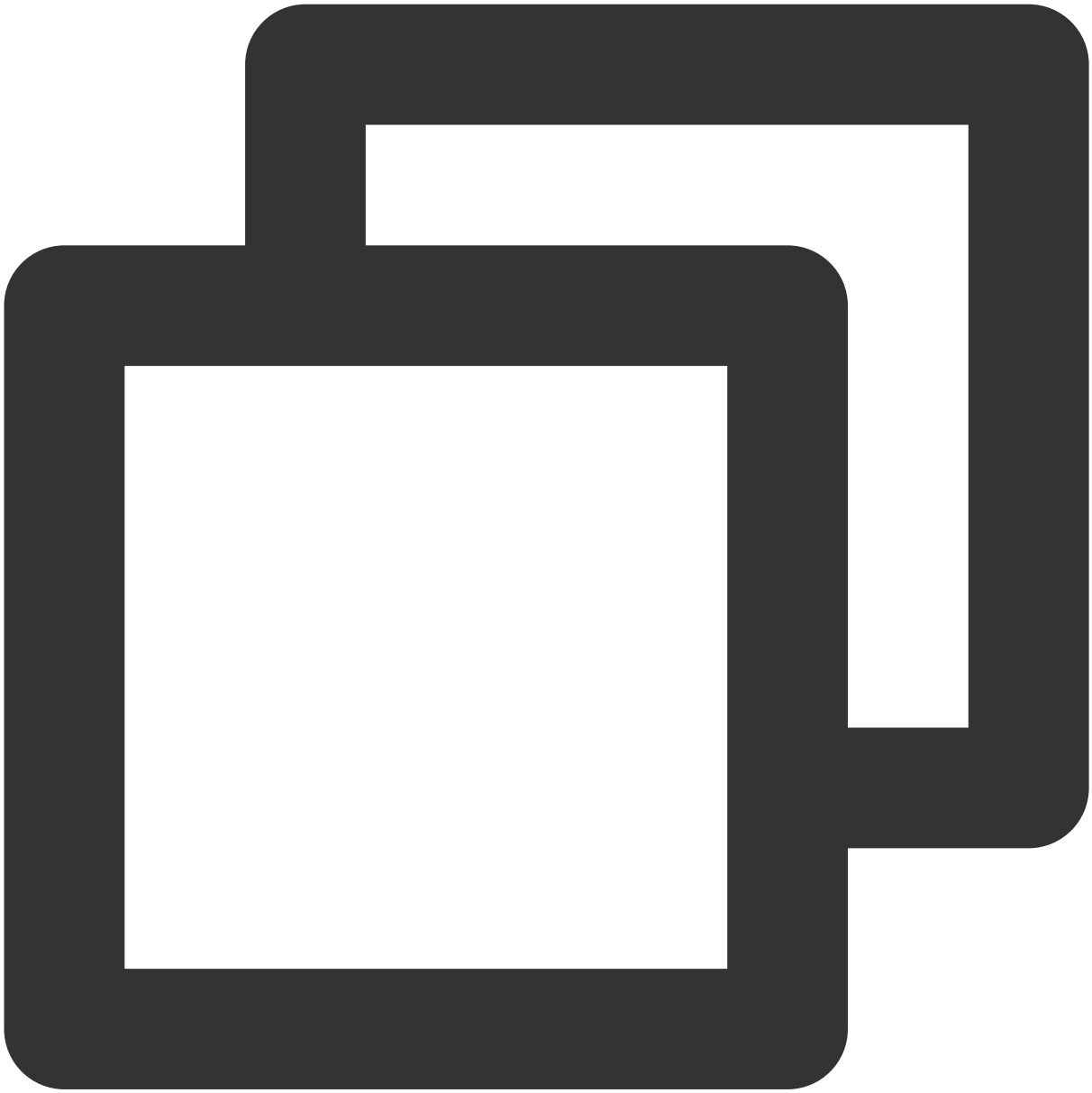
3. 멀티파트 업로드 초기화



```
#멀티파트 업로드 초기화
response = client.create_multipart_upload(
    Bucket='examplebucket-1250000000',  #사용자 Bucket 이름으로 대체. examplebucket은 
    Key='exampleobject',                #업로드한 객체의 Key 값으로 대체
    StorageClass='STANDARD',            #객체의 스토리지 유형
)
#멀티파트 업로드한 UploadId 획득
upload_id = response['UploadId']
```

4. 객체 멀티파트 업로드

객체 멀티파트 업로드는 객체를 여러 파트로 분할하여 업로드하는 것을 뜻합니다. 최대 10000개 파트까지 분할할 수 있습니다. 각 파트의 크기는 1MB-5GB이며, 마지막 파트의 크기는 1MB 미만일 수 있습니다. 멀티파트를 업로드할 때 각 파트의 PartNumber(번호)를 설정하고, 각 파트의 CRC64 값을 계산해야 합니다. 멀티파트 업로드를 성공한 후 반환되는 CRC64 값과 로컬에서 계산한 값을 검사할 수 있습니다.

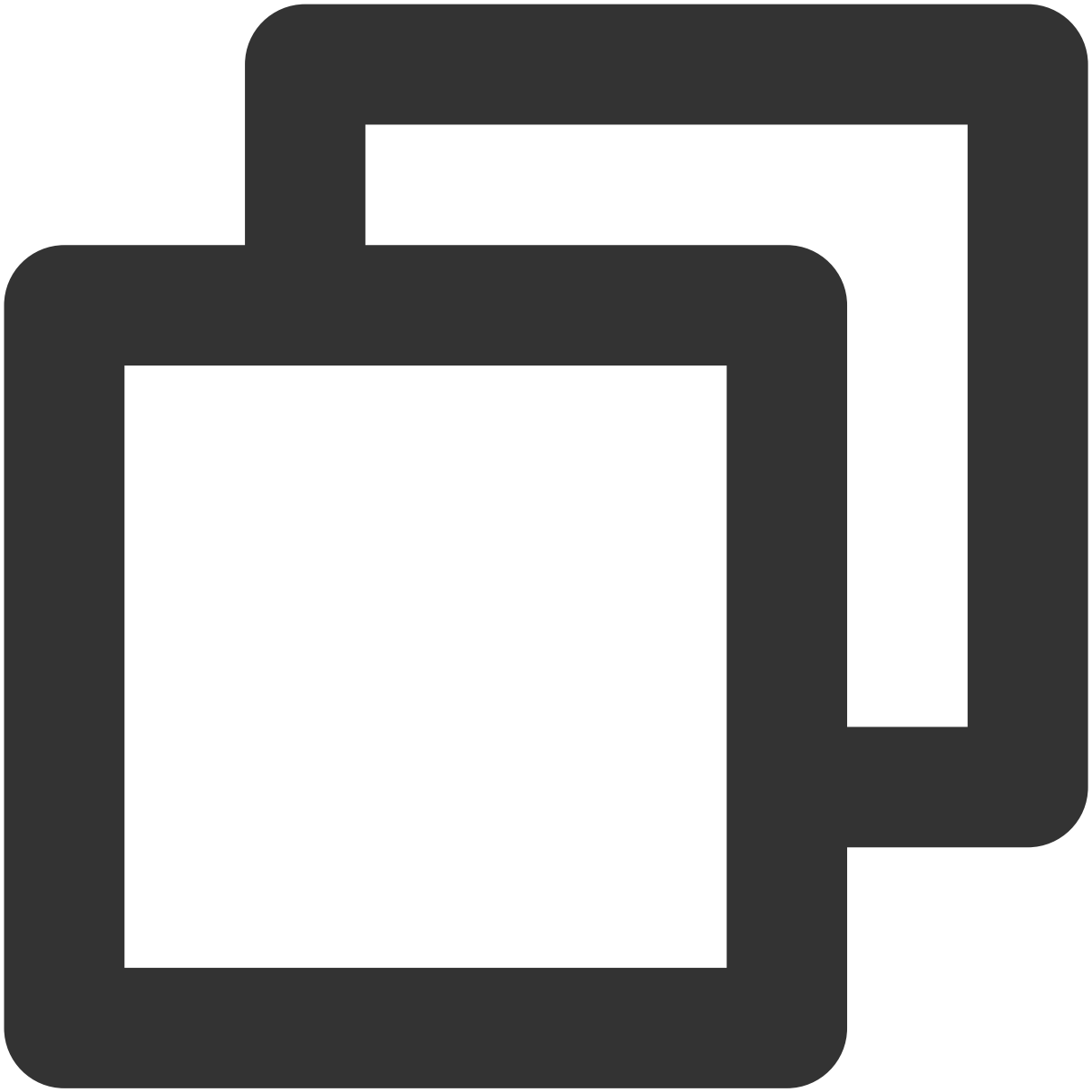


```
#객체를 멀티파트 업로드할 때, 각 파트의 크기는 OBJECT_PART_SIZE이며, 마지막 파트의 크기는 OBJ
part_list = list()
position = 0
left_size = OBJECT_TOTAL_SIZE
part_number = 0
while left_size > 0:
```

```
    part_number += 1
    if left_size >= OBJECT_PART_SIZE:
        body = object_body[position:position+OBJECT_PART_SIZE]
    else:
        body = object_body[position:]
    position += OBJECT_PART_SIZE
    left_size -= OBJECT_PART_SIZE
    local_part_crc64 = str(c64(body))    #로컬에서 CRC64 계산

    response = client.upload_part(
        Bucket='examplebucket-1250000000',
        Key='exampleobject',
        Body=body,
        PartNumber=part_number,
        UploadId=upload_id,
    )
    part_crc_64 = response['x-cos-hash-crc64ecma']# 서버에서 반환된 CRC64
    if local_part_crc64 != part_crc_64:          # 데이터 검사
        print 'crc64 check FAIL'
        exit(-1)
    etag = response['ETag']
    part_list.append({'ETag' : etag, 'PartNumber' : part_number})
```

5. 멀티파트 업로드 완료

모든 멀티파트를 업로드한 뒤 멀티파트 업로드 작업을 완료해야 합니다. COS에서 반환된 CRC64와 로컬 객체의 CRC64를 비교 검증할 수 있습니다.



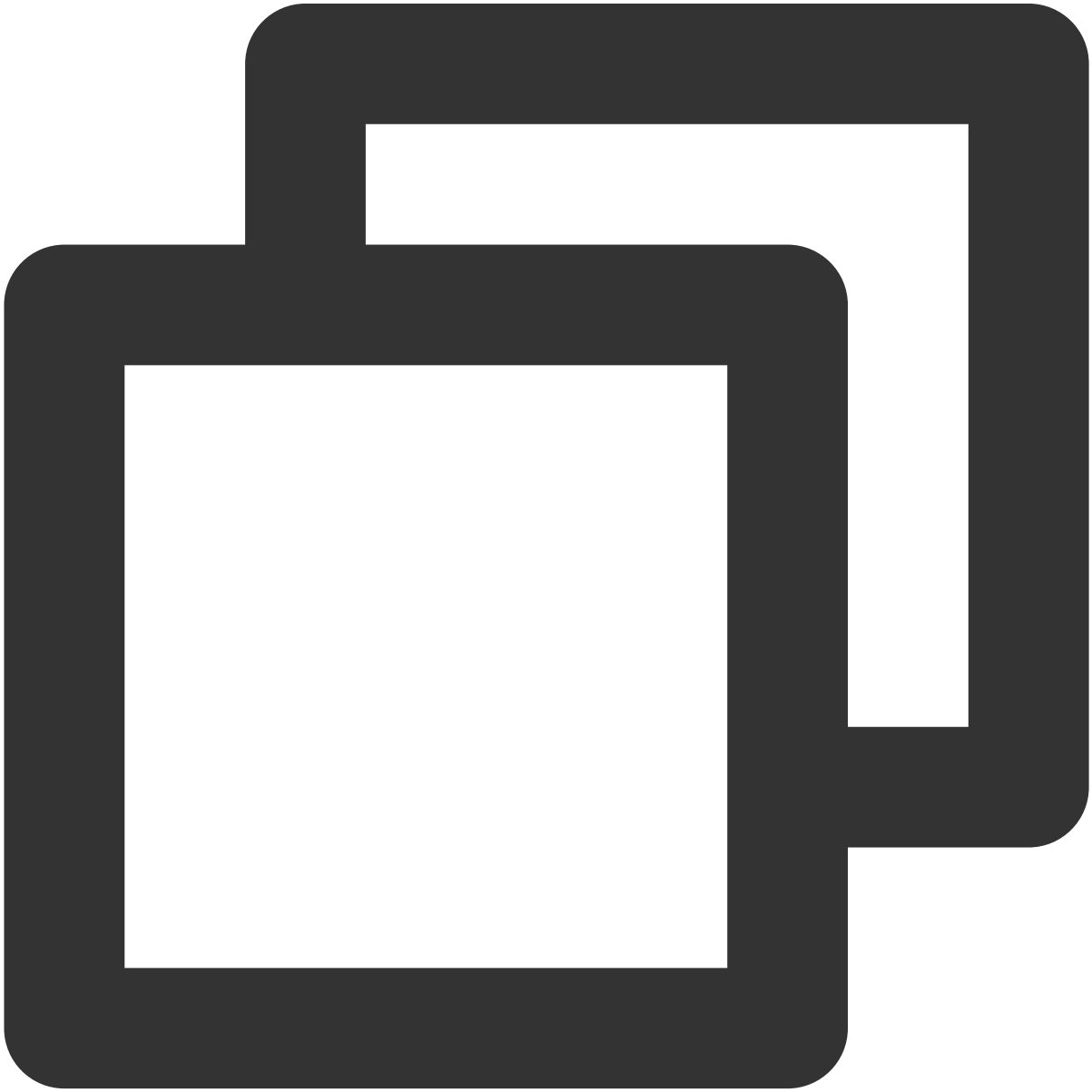
```
#멀티파트 업로드 완료
response = client.complete_multipart_upload(
    Bucket='examplebucket-1250000000',  #사용자 Bucket 이름으로 대체. examplebucket은 
    Key='exampleobject',                #객체의 Key 값
    UploadId=upload_id,
    MultipartUpload={       #모든 파트의 ETag와 PartNumber가 모두 대응되어야 함
        'Part' : part_list
    },
)
crc64ecma = response['x-cos-hash-crc64ecma']
if crc64ecma != local_crc64:                    # 데이터 검사
```

```
    print 'check crc64 Failed'
    exit(-1)
```

Java SDK

객체 업로드는 Java SDK의 고급 인터페이스 사용을 권장합니다. Java SDK 객체 작업을 참고하십시오.

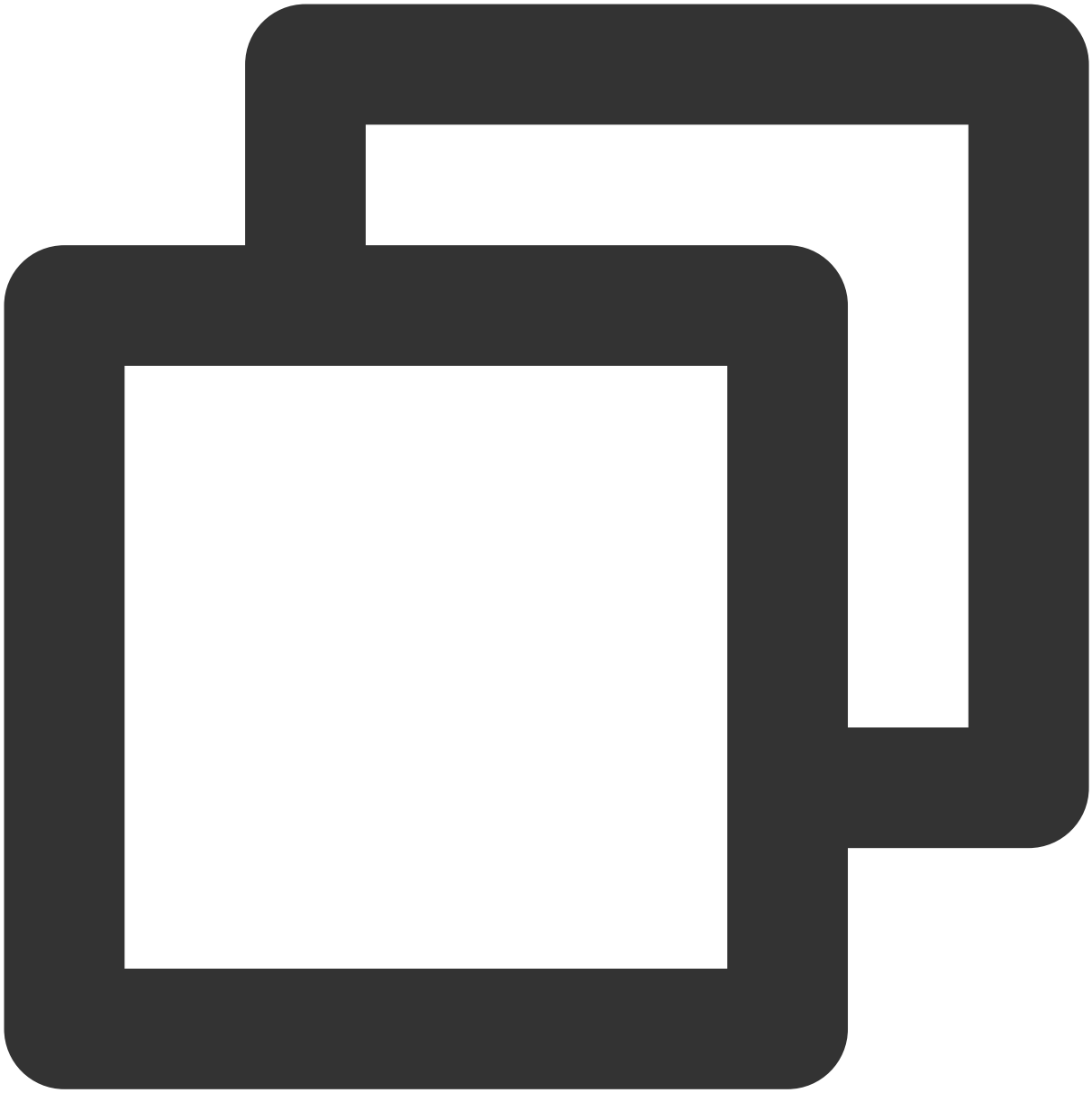
로컬에서 파일의 crc64를 계산하는 방법



```
String calculateCrc64(File localFile) throws IOException {
    CRC64 crc64 = new CRC64();
```

```
    try (FileInputStream stream = new FileInputStream(localFile)) {
        byte[] b = new byte[1024 * 1024];
        while (true) {
            final int read = stream.read(b);
            if (read <= 0) {
                break;
            }
            crc64.update(b, read);
        }
    }

    return Long.toUnsignedString(crc64.getValue());
}
```

COS에서 파일의 crc64 값을 가져와서 로컬 파일과 검사하는 방법



```
// COSClient 생성 참고: [시작하기](https://www.tencentcloud.com/document/product/436/1
ObjectMetadata cosMeta = COSClient().getObjectMetadata(bucketName, cosFilePath);
String cosCrc64 = cosMeta.getCrc64Ecma();
String localCrc64 = calculateCrc64(localFile);

if (cosCrc64.equals(localCrc64)) {
    System.out.println("ok");
} else {
    System.out.println("fail");
}
```

빅 데이터 사례
COS를 Druid의 Deep storage로 사용

최종 업데이트 날짜: : 2024-06-24 16:53:18

환경 종속

HADOOP-COS와 Hadoop-COS-Java-SDK(HADOOP-COS의 dep 디렉터리 안에 포함)
Druid 버전: Druid-0.12.1.

다운로드 및 설치

HADOOP-COS 다운로드

운영사 Github에서 HADOOP-COS를 다운로드하십시오.

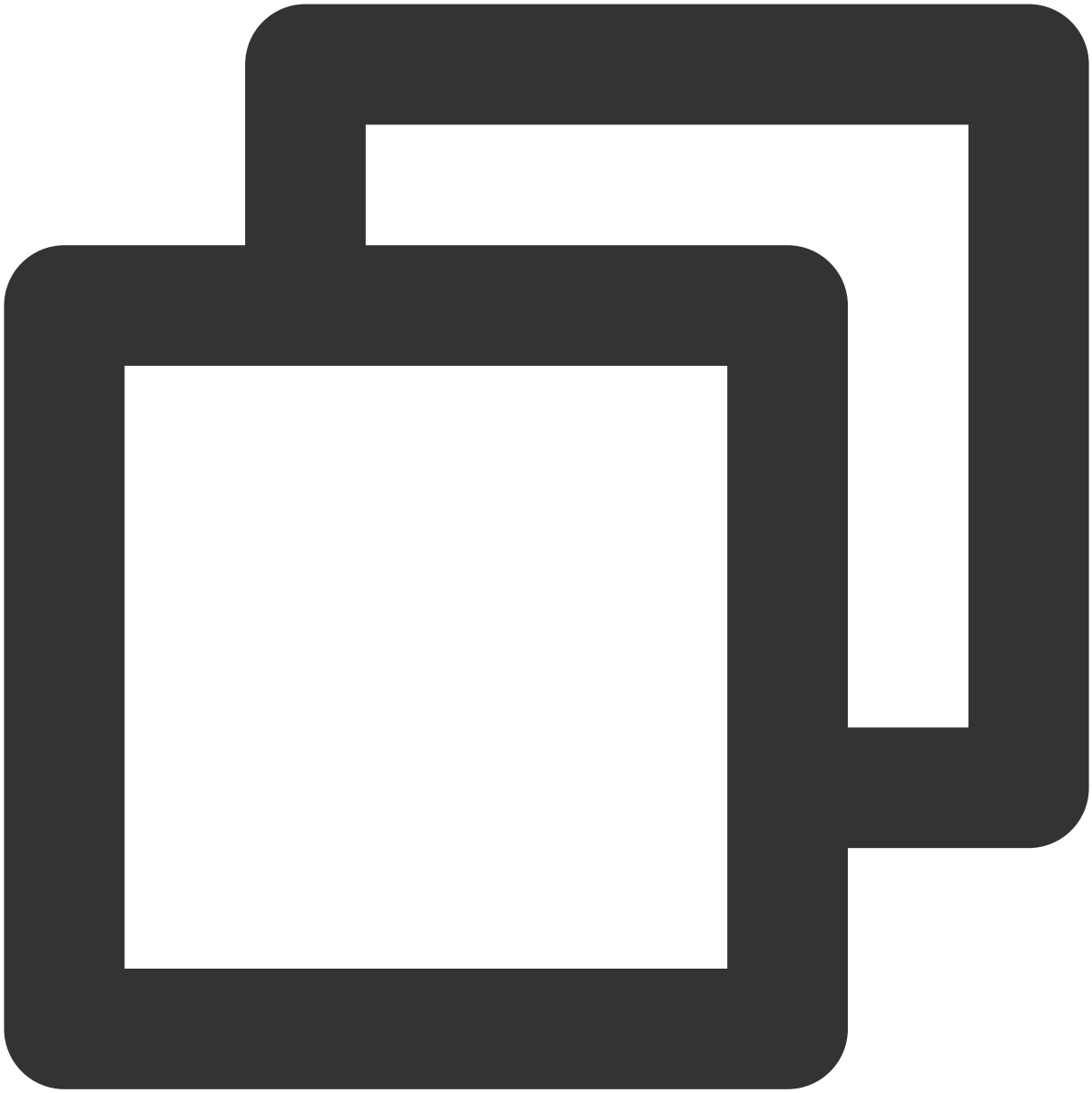
HADOOP-COS 설치

Druid가 COS를 Deep Storage로 사용하려면 Druid-hdfs-extension이 필요합니다.
HADOOP-COS를 다운로드한 후 dep 디렉터리의 `hadoop-cos-2.x.x.jar` 버전을 Druid의 설치 경로인 `extensions/druid-hdfs-storage` 와 `hadoop-dependencies/hadoop-client/2.x.x` 에 복사합니다. Druid가 HDFS plugin을 사용하여 COS에 액세스하기 때문에 선택한 버전이 Druid의 HDFS plugin 버전과 일치해야 합니다.

사용 방법

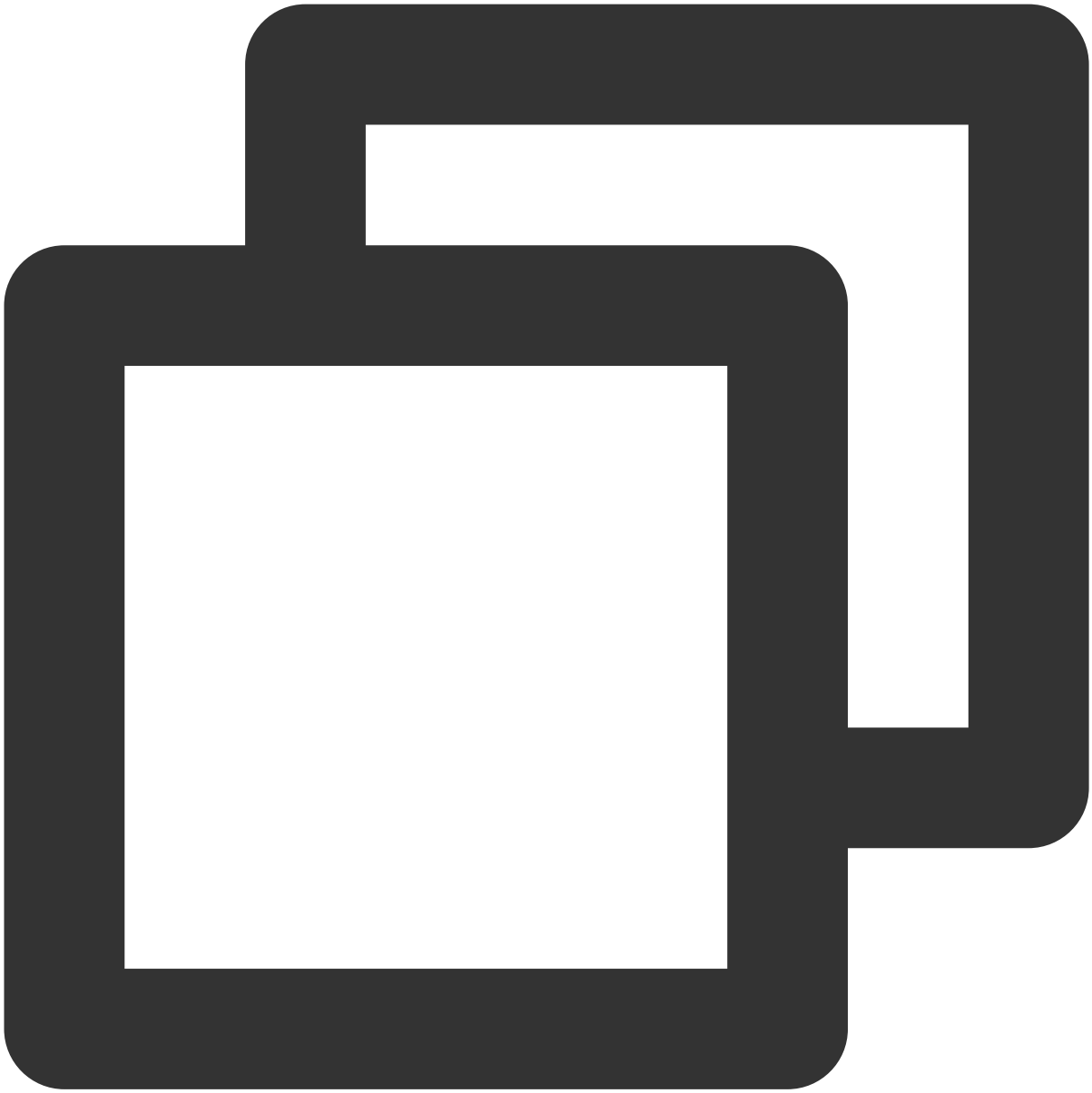
수정 설정

1. Druid 설치 경로인 `conf/druid/_common/common.runtime.properties` 파일을 수정합니다. hdfs의 extension을 `druid.extensions.loadList` 에 추가합니다. hdfs를 Druid의 deep storage로 지정하고 경로는 cosn 경로로 작성합니다.



```
properties
druid.extensions.loadList=["druid-hdfs-storage"]
druid.storage.type=hdfs
druid.storage.storageDirectory=cosn://bucket-appid/<druid-path>
```

2. `conf/druid/_common/` 디렉터리에 hdfs의 구성 파일 hdfs-site.xml을 새로 만들고 COS의 키 정보 등을 작성합니다.



```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at
 http://www.apache.org/licenses/LICENSE-2.0
  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
```

```
   limitations under the License. See accompanying LICENSE file.
  -->
  <!-- Put site-specific property overrides in this file. -->
 <configuration>
  <property>
      <name>fs.cosn.userinfo.secretId</name>
      <value>xxxxxxxxxxxxxxxxxxxxxxx</value>
  </property>
  <property>
      <name>fs.cosn.userinfo.secretKey</name>
      <value>xxxxxxxxxxxxxxx</value>
  </property>
  <property>
      <name>fs.cosn.impl</name>
      <value>org.apache.hadoop.fs.CosFileSystem</value>
  </property>
  <property>
      <name>fs.cosn.userinfo.region</name>
      <value>ap-xxxx</value>
  </property>
  <property>
      <name>fs.cosn.tmp.dir</name>
      <value>/tmp/hadoop_cos</value>
  </property>
 </configuration>
```

위에서 설정한 지원 항목은 HADOOP-COS 공식 홈페이지의 문서 설명과 동일하며 자세한 내용은 Hadoop 툴 문서를 참조하십시오.

사용하기

Druid 프로세스에 따라 실행하면 Druid 데이터가 COS에 로딩됩니다.

DataX로 COS 가져오기 또는 내보내기

최종 업데이트 날짜: : 2024-06-24 16:53:18

환경 종속

HADOOP-COS 및 해당 버전의 cos_api-bundle.
DataX 버전: DataX-3.0.

다운로드 및 설치

HADOOP-COS 다운로드

공식 Github에서 HADOOP-COS 및 해당 버전의 cos_api-bundle을 다운로드하십시오.

DataX 소프트웨어 패키지 다운로드

운영사 Github에서 DataX를 다운로드하십시오.

HADOOP-COS 설치

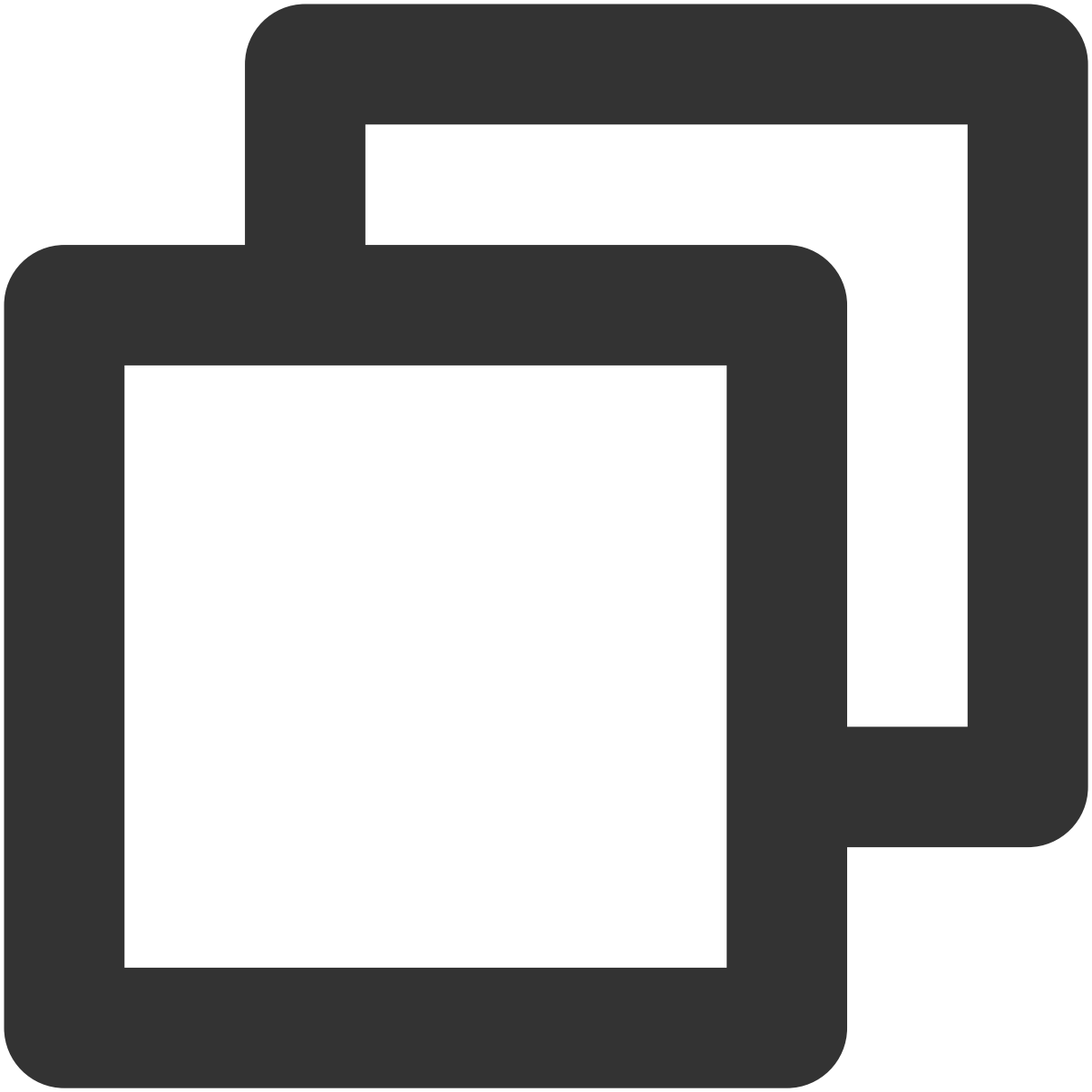
HADOOP-COS 다운로드 후 `hadoop-cos-2.x.x-${version}.jar` 및 `cos_api-bundle-${version}.jar` 를 Datax 압축 해제 경로 `plugin/reader/hdfsreader/libs/` 및 `plugin/writer/hdfswriter/libs/` 에 복사하십시오.

사용 방법

DataX 설정

datax.py 스크립트 수정

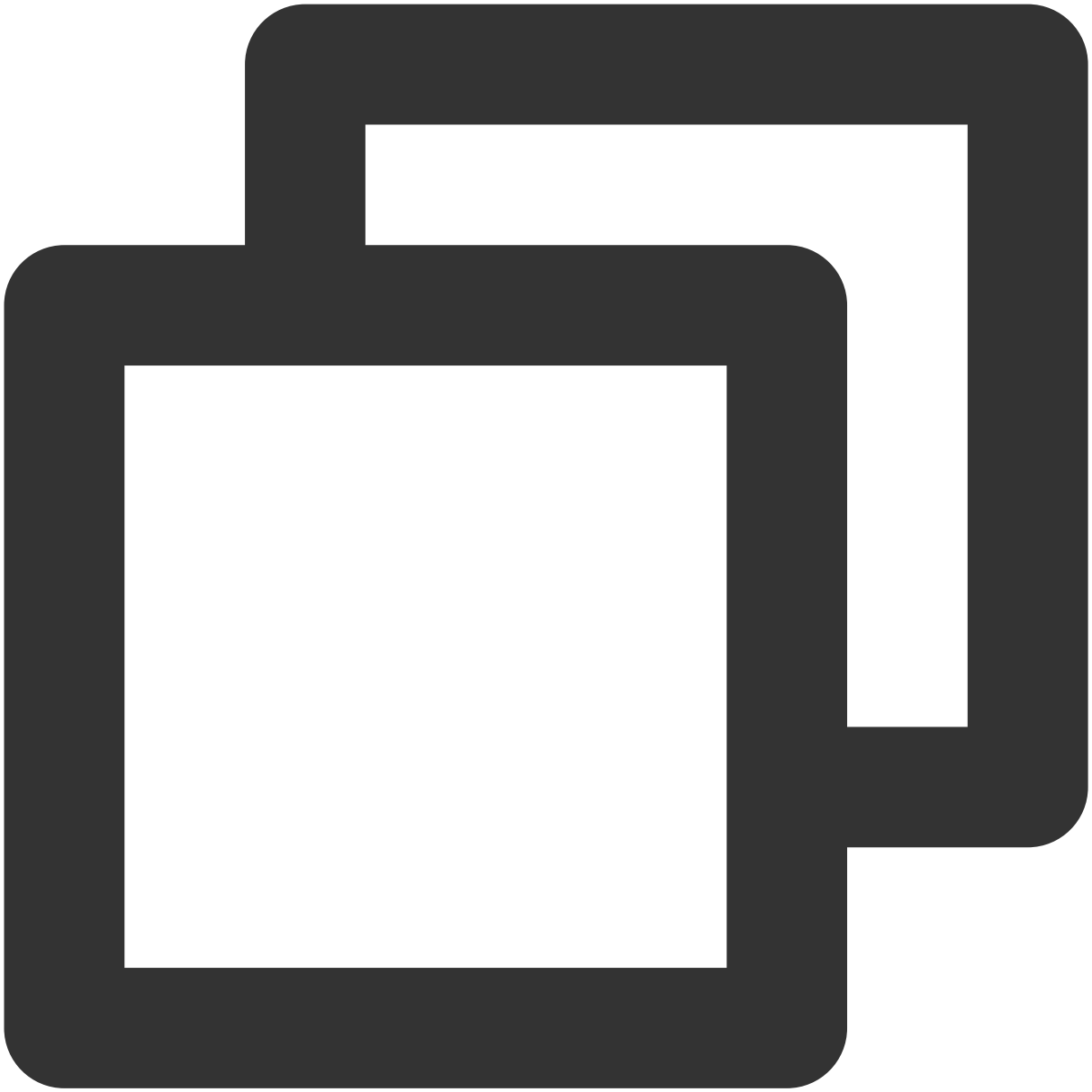
DataX의 압축 해제 디렉터리에 속한 bin/datax.py 스크립트를 열어 스크립트의 CLASS_PATH 변수를 아래와 같이 수정하십시오.



```
CLASS_PATH = ("%s/lib/*:%s/plugin/reader/hdfsreader/libs/*:%s/plugin/writer/hdfswri
```

JSON 설정 파일에 hdfsreader 및 hdfswriter 설정

JSON 파일 예시는 아래와 같습니다.



```
{
    "job": {
        "setting": {
            "speed": {
                "byte": 10485760
            },
            "errorLimit": {
                "record": 0,
                "percentage": 0.02
            }
        },
```

```
        "content": [{
            "reader": {
                "name": "hdfsreader",
                "parameter": {
                    "path": "testfile",
                    "defaultFS": "cosn://examplebucket-1250000000/",
                    "column": ["*"],
                    "fileType": "text",
                    "encoding": "UTF-8",
                    "hadoopConfig": {
                        "fs.cosn.impl": "org.apache.hadoop.fs.CosFileSystem",
                        "fs.cosn.userinfo.region": "ap-beijing",
                        "fs.cosn.tmp.dir": "/tmp/hadoop_cos",
                        "fs.cosn.userinfo.secretId": "COS_SECRETID",
                        "fs.cosn.userinfo.secretKey": "COS_SECRETKEY"
                    },
                    "fieldDelimiter": ","
                }
            },
            "writer": {
                "name": "hdfswriter",
                "parameter": {
                    "path": "/user/hadoop/",
                    "fileName": "testfile1",
                    "defaultFS": "cosn://examplebucket-1250000000/",
                    "column": [{
                            "name": "col",
                            "type": "string"
                        },
                        {
                            "name": "col1",
                            "type": "string"
                        },
                        {
                            "name": "col2",
                            "type": "string"
                        }
                    ],
                    "fileType": "text",
                    "encoding": "UTF-8",
                    "hadoopConfig": {
                        "fs.cosn.impl": "org.apache.hadoop.fs.CosFileSystem",
                        "fs.cosn.userinfo.region": "ap-beijing",
                        "fs.cosn.tmp.dir": "/tmp/hadoop_cos",
                        "fs.cosn.userinfo.secretId": "COS_SECRETID",
                        "fs.cosn.userinfo.secretKey": "COS_SECRETKEY"
                    },
```

```
                    "fieldDelimiter": ":",
                    "writeMode": "append"
                }
            }
        }]
    }
}
```

설정은 다음 내용을 참조하십시오.

hadoopConfig 설정은 cosn에서 필요한 설정입니다.

defaultFS는 cosn 경로로 작성합니다. 예를 들어 `cosn://examplebucket-1250000000/` 와 같이 쓸 수 있습니다.

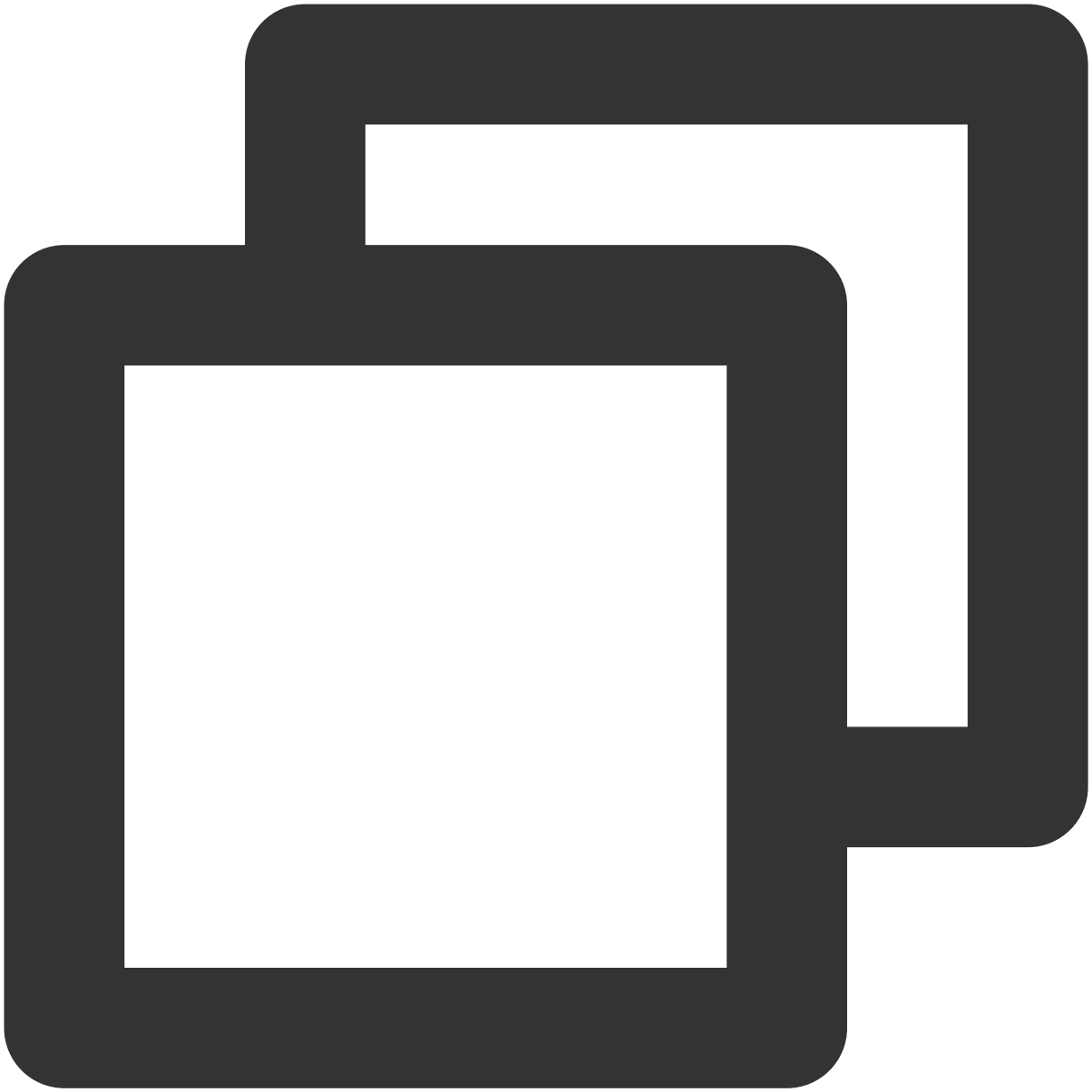
fs.cosn.userinfo.region을 버킷이 속한 리전(`ap-beijing` 등)으로 변경합니다. 자세한 내용은 리전과 액세스 도메인을 참조하십시오.

COS_SECRETID와 COS_SECRETKEY를 COS 키로 변경합니다.

다른 항목은 hdfs 설정 항목과 동일하게 설정합니다.

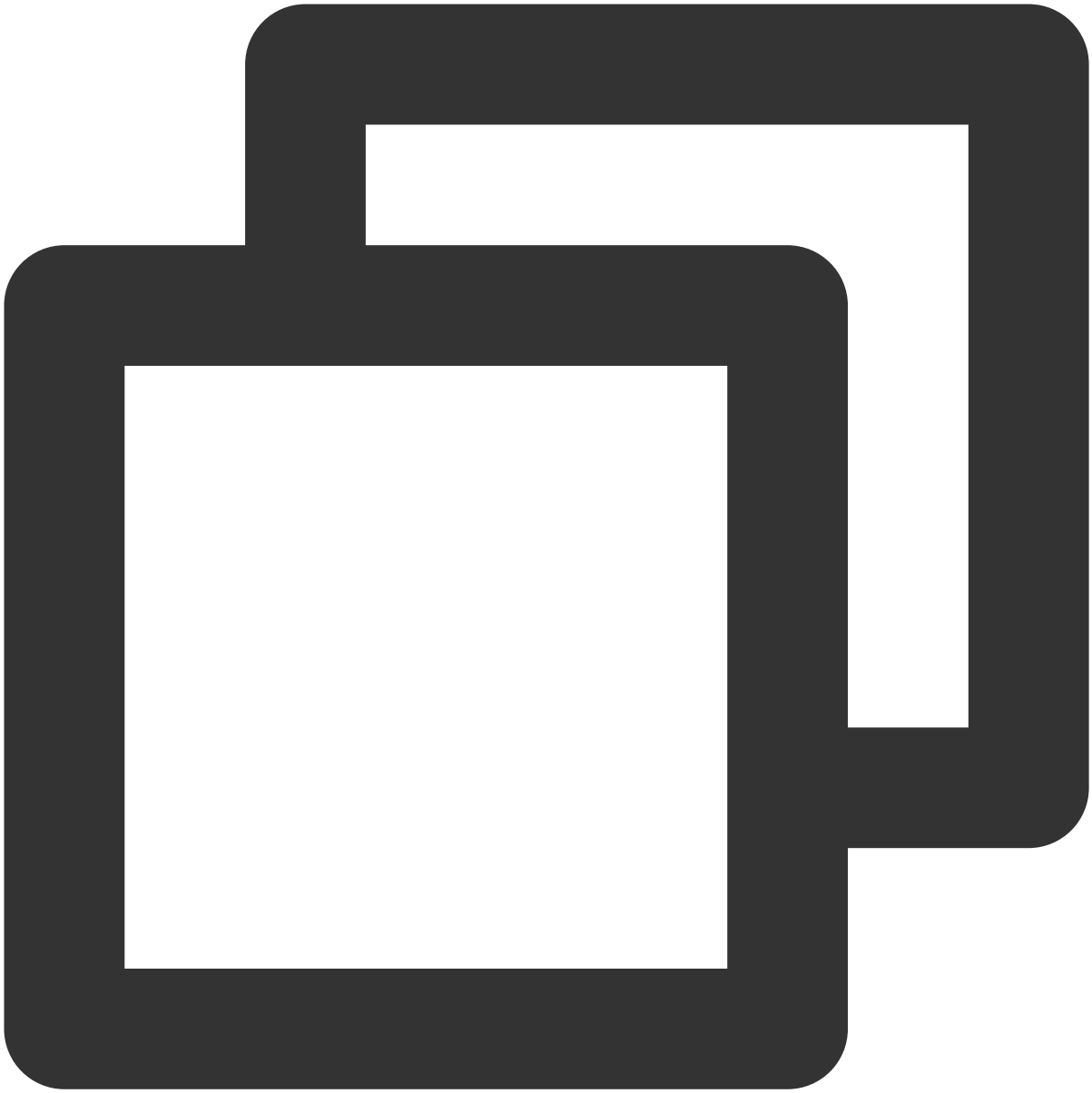
데이터 마이그레이션 실행

구성 파일 이름을 hdfs_job.json으로 설정하고 job 디렉터리에 저장해 아래와 같이 명령 라인을 실행합니다.



```
bin/datax.py job/hdfs_job.json
```

스크린에서 보이는 정상적인 결과 출력은 아래와 같습니다.



```
2020-03-09 16:49:59.543 [job-0] INFO  JobContainer -
         [total cpu info] =>
                averageCpu                     | maxDeltaCpu                    | m
                -1.00%                         | -1.00%                         | -


         [total gc info] =>
                 NAME                 | totalGCCount       | maxDeltaGCCount    | m
                 PS MarkSweep         | 1                  | 1                  | 1
                 PS Scavenge          | 1                  | 1                  | 1
```

```
2020-03-09 16:49:59.543 [job-0] INFO  JobContainer - PerfTrace not enable!
2020-03-09 16:49:59.543 [job-0] INFO  StandAloneJobContainerCommunicator - Total 2
2020-03-09 16:49:59.544 [job-0] INFO  JobContainer -
작업 활성화 시간                    : 2020-03-09 16:49:48
작업 종료 시간                     : 2020-03-09 16:49:59
작업 총 소요 시간                    :                11s
작업 평균 트래픽                    :               3B/s
기록 입력 속도                     :             0rec/s
읽은 기록 총 횟수                    :                  2
읽기/쓰기 실패 총 횟수                    :                  0
```

CDH용 COSN 구성

최종 업데이트 날짜: : 2024-06-24 16:53:18

소개

CDH(Cloudera's Distribution, including Apache Hadoop)는 업계에서 각광받는 Hadoop의 릴리스 버전입니다. 본 문서에서는 CDH 환경에서의 COSN 스토리지 서비스 사용 방법을 안내합니다. 이를 통해 빅 데이터 연산 및 스토리지 분산을 구현함으로써 효율적인 저비용 빅 데이터 솔루션을 제공합니다.

설명：

본 문서에서 COSN은 COSN을 기반으로 구축된 파일 시스템인 Hadoop-COS를 의미합니다.

COSN 빅 데이터 모듈 지원 현황은 다음과 같습니다.

모듈 이름	COSN 빅 데이터 모듈 지원 현황	서비스 모듈 재시작 필요 여부
Yarn	지원	NodeManager 재시작
Hive	지원	HiveServer 및 HiveMetastore 재시작
Spark	지원	NodeManager 재시작
Sqoop	지원	NodeManager 재시작
Presto	지원	HiveServer, HiveMetastore, Presto 재시작
Flink	지원	필요 없음
Impala	지원	필요 없음
EMR	지원	필요 없음
자체구축 모듈	향후 지원	필요 없음
HBase	권장하지 않음	필요 없음

버전 종속

본 문서에 종속된 모듈 버전은 다음과 같습니다.

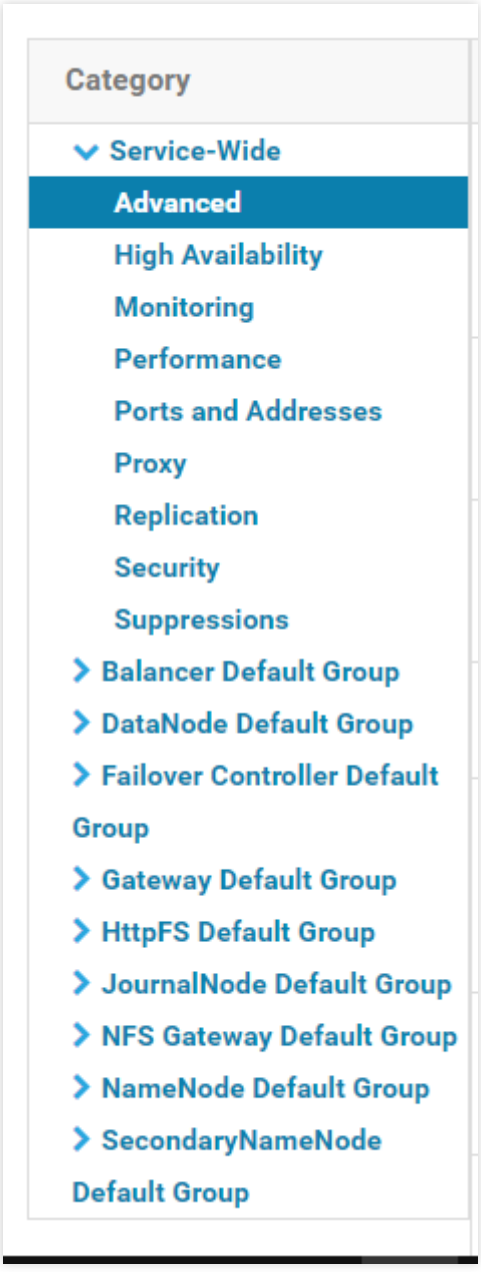
CDH 5.16.1

Hadoop 2.6.0

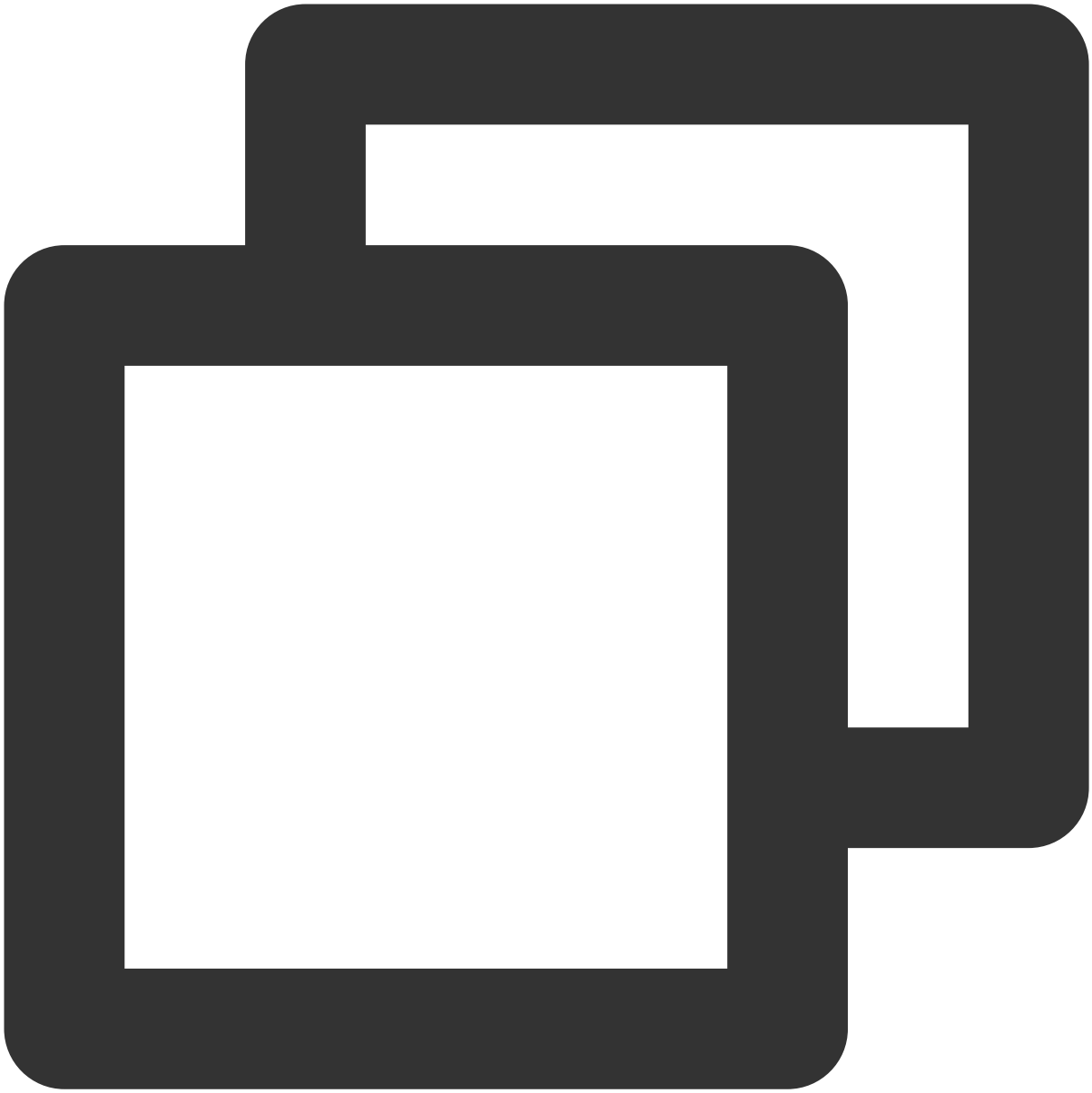
사용 방법

스토리지 환경 설정

1. CDH 관리 페이지에 로그인합니다.
2. 시스템 메인 페이지에서 **설정>서비스 범위>고급**을 선택하여 코드 스니펫 고급 설정 페이지로 이동하면 다음 그림과 같이 표시됩니다.



3. Cluster-wide Advanced Configuration Snippet(Safety Valve) for core-site.xml 코드 창에 COSN 설정을 입력합니다.



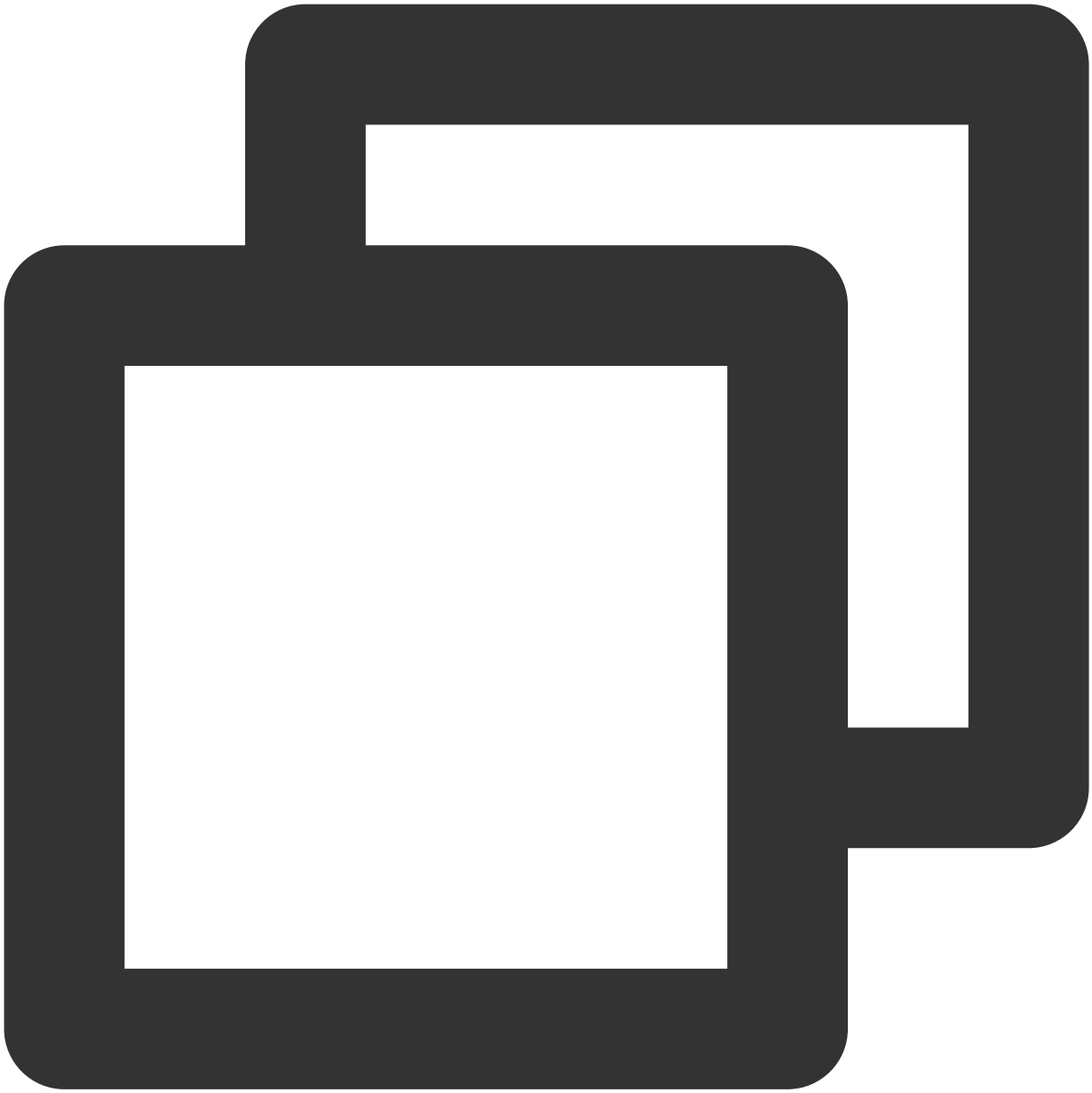
```
<property>
<name>fs.cosn.userinfo.secretId</name>
<value>AK***</value>
</property>
<property>
<name>fs.cosn.userinfo.secretKey</name>
<value></value>
</property>
<property>
<name>fs.cosn.impl</name>
<value>org.apache.hadoop.fs.CosFileSystem</value>
```

```
</property>
<property>
<name>fs.AbstractFileSystem.cosn.impl</name>
<value>org.apache.hadoop.fs.CosN</value>
</property>
<property>
<name>fs.cosn.bucket.region</name>
<value>ap-shanghai</value>
</property>
```

다음은 COSN 필수 설정 항목(core-site.xml에 추가 필요)입니다. COSN의 기타 설정 항목은 Hadoop 툴 문서를 참고하십시오.

COSN 매개변수	값	설명
fs.cosn.userinfo.secretId	AKxxxx	Tencent Cloud 계정의 API 키 정보
fs.cosn.userinfo.secretKey	Wpxxxx	Tencent Cloud 계정의 API 키 정보
fs.cosn.bucket.region	ap-shanghai	COS 버킷이 있는 리전
fs.cosn.impl	org.apache.hadoop.fs.CosFileSystem	FileSystem용 cosn 구현 클래스. org.apache.hadoop.fs.CosFileSysten으로 고정
fs.AbstractFileSystem.cosn.impl	org.apache.hadoop.fs.CosN	AbstractFileSystem에 대한 cosn 구현 클래스. org.apache.hadoop.fs.CosN으로 고정

4. HDFS 서비스 작업을 진행하기 위해 클라이언트 배포 설정을 클릭하면 core-site.xml 설정이 클러스터 기기에 업데이트됩니다.
5. COSN 최신 SDK 패키지를 CDH HDFS 서비스의 jar 패키지 경로에 설치합니다. 다음 예시와 같이 실제 값에 따라 변경해 주십시오.



```
cp hadoop-cos-2.7.3-shaded.jar /opt/cloudera/parcels/CDH-5.16.1-1.cdh5.16.1.p0.3/li
```

주의 :

클러스터의 기기마다 동일한 위치에 SDK 패키지를 설치해야 합니다.

데이터 마이그레이션

Hadoop Distcp 툴을 사용해 CDH HDFS 데이터를 COSN에 마이그레이션합니다. 자세한 내용은 Hadoop 파일 시스템과 COS 간 데이터 마이그레이션을 참고하십시오.

빅 데이터 세트에서 COSN 사용하기

1. MapReduce

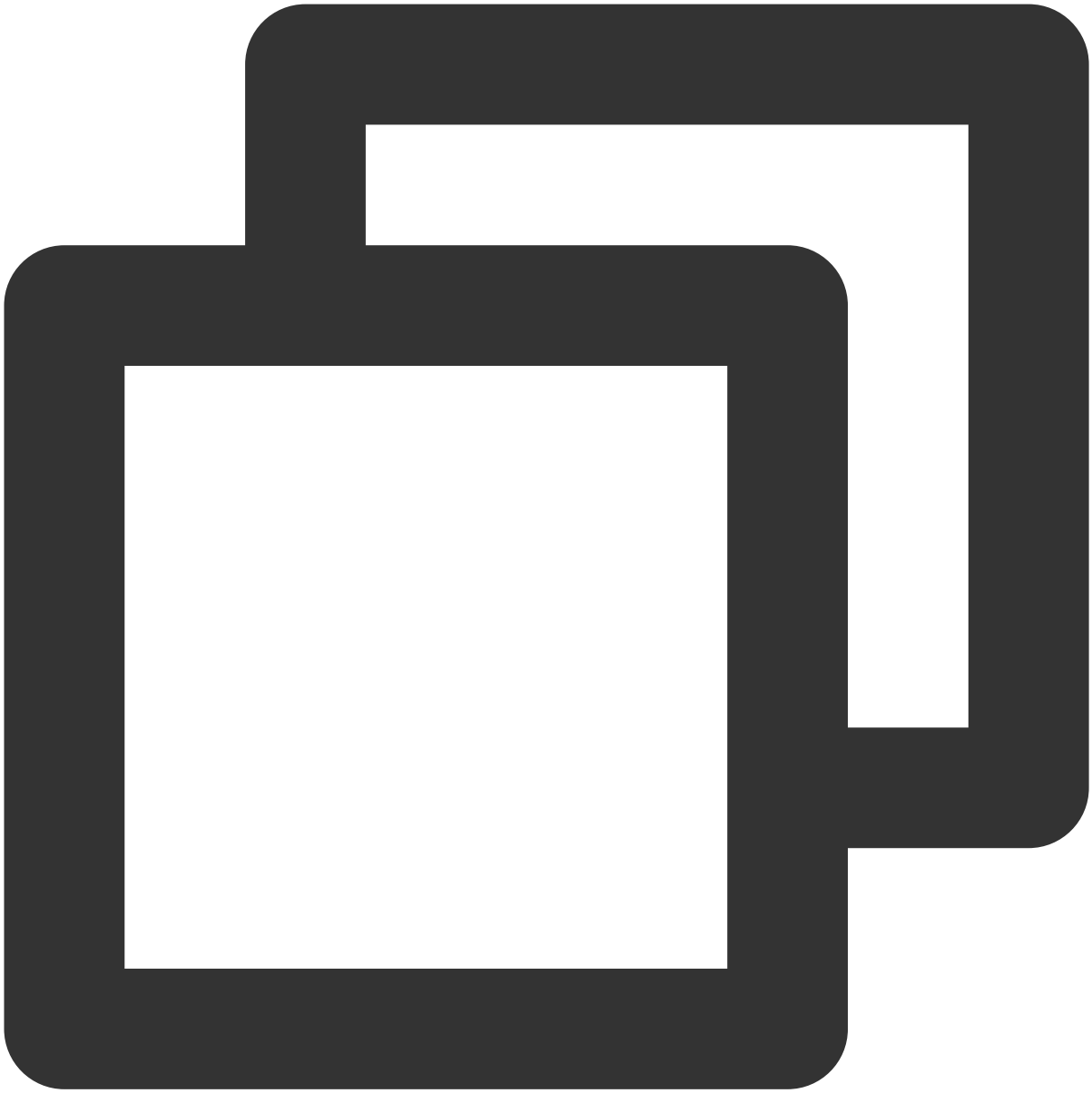
작업 단계

(1) 데이터 마이그레이션 섹션에 따라 HDFS 관련 설정을 완료하고, COSN의 SDK jar 패키지를 HDFS의 해당 디렉터리에 넣습니다.

(2) CDH 시스템 메인 페이지에서 YARN을 찾아 NodeManager 서비스를 재시작합니다. TeraGen 명령어는 재시작할 필요가 없으나 TeraSort는 비즈니스 내부 로직으로 인해 NodeManger를 재시작해야 하므로, NodeManager 서비스를 일괄 재시작할 것을 권장합니다.

예시

Hadoop 표준 테스트의 TeraGen 및 TeraSort입니다.



```
hadoop jar ./hadoop-mapreduce-examples-2.7.3.jar teragen  -Dmapred.job.maps=500  -D

hadoop jar ./hadoop-mapreduce-examples-2.7.3.jar terasort -Dmapred.max.split.size=1
```

설명：

`cosn://schema` 뒷부분을 사용자 빅 데이터 비즈니스의 버킷 경로로 변경하십시오.

2. Hive

2.1 MR 엔진

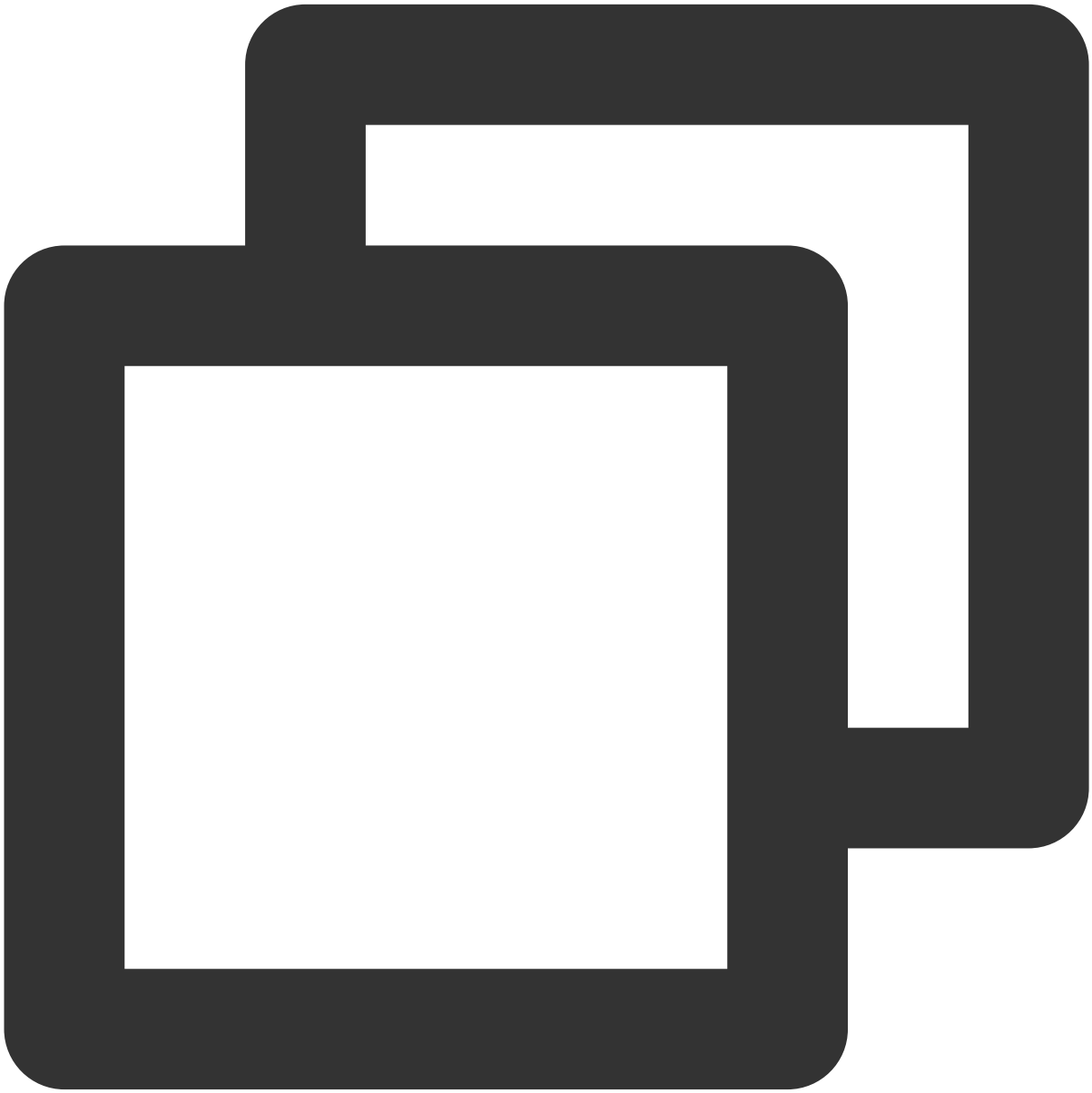
작업 단계

(1) 데이터 마이그레이션 섹션에 따라 HDFS 관련 설정을 완료하고, COSN의 SDK jar 패키지를 HDFS의 해당 디렉터리에 넣습니다.

(2) CDH 메인 페이지에서 HIVE 서비스를 찾아 Hiveserver2와 HiverMetastore 역할을 재시작합니다.

예시

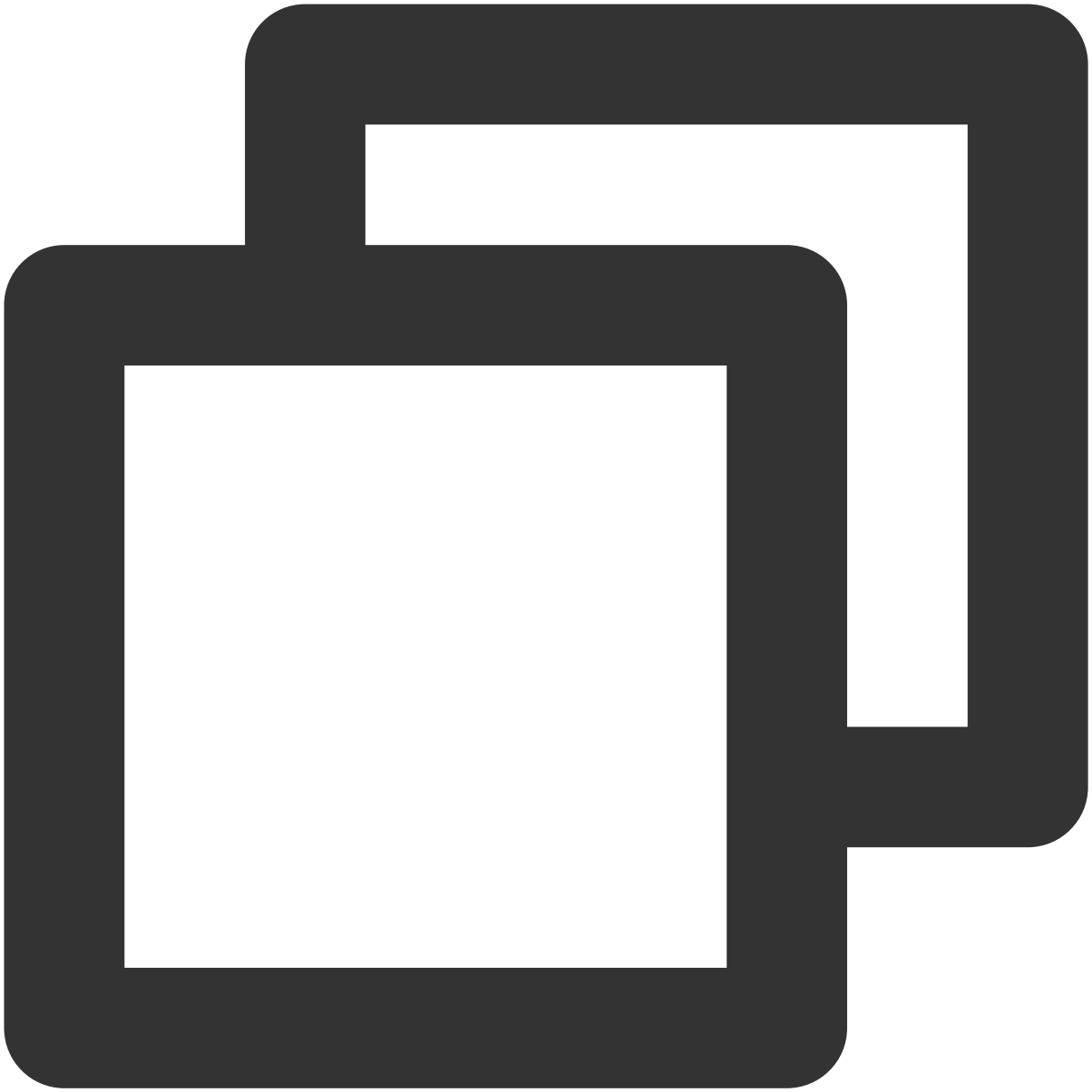
사용자의 실제 작업 쿼리입니다. Hive 명령 라인을 실행하여 Location을 생성하고, 이를 CHDFS의 파티션 테이블로 사용하는 경우입니다.



```
CREATE TABLE `report.report_o2o_pid_credit_detail_grant_daily`(
  `cal_dt` string,
```

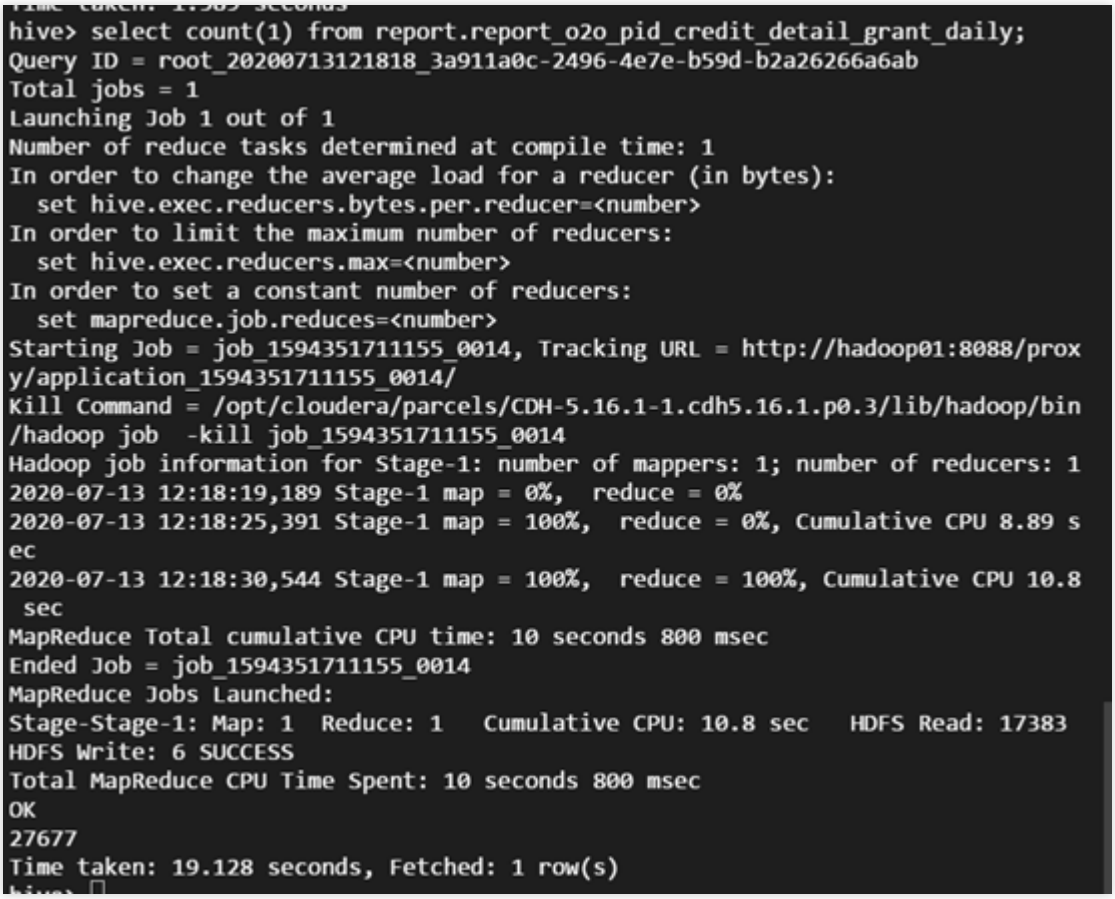
```
  `change_time` string,
  `merchant_id` bigint,
  `store_id` bigint,
  `store_name` string,
  `wid` string,
  `member_id` bigint,
  `meber_card` string,
  `nickname` string,
  `name` string,
  `gender` string,
  `birthday` string,
  `city` string,
  `mobile` string,
  `credit_grant` bigint,
  `change_reason` string,
  `available_point` bigint,
  `date_time` string,
  `channel_type` bigint,
  `point_flow_id` bigint)
PARTITIONED BY (
  `topicdate` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.orc.OrcSerde'
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcInputFormat'
    OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat'
LOCATION
  'cosn://examplebucket-1250000000/user/hive/warehouse/report.db/report_o2o_pid_cre
TBLPROPERTIES (
  'last_modified_by'='work',
  'last_modified_time'='1589310646',
  'transient_lastDdlTime'='1589310646')
```

sql 쿼리 실행:



```
select count(1) from report.report_o2o_pid_credit_detail_grant_daily;
```

모니터링 결과는 다음과 같습니다.



2.2 Tez 엔진

Tez 엔진은 COSN의 jar 패키지를 Tez의 압축 패키지 내로 가져와야 합니다. 다음은 apache-tez.0.8.5를 예시로 한 설명입니다.

작업 단계

(1) CDH 클러스터에 설치한 tez 패키지를 찾은 후 압축을 해제합니다. /usr/local/service/tez/tez-0.8.5.tar.gz를 예로 들 수 있습니다.

(2) COSN의 jar 패키지를 압축 해제한 디렉터리에 넣고 다시 하나의 압축 패키지로 만듭니다.

(3) 새로 생성한 압축 패키지를 tez.lib.uris에서 지정한 경로(경로가 이미 있는 경우 변경 가능)에 업로드합니다.

(4) CDH 메인 페이지에서 HIVE를 찾아 hiveserver와 hivemetastore를 재시작합니다.

3. Spark

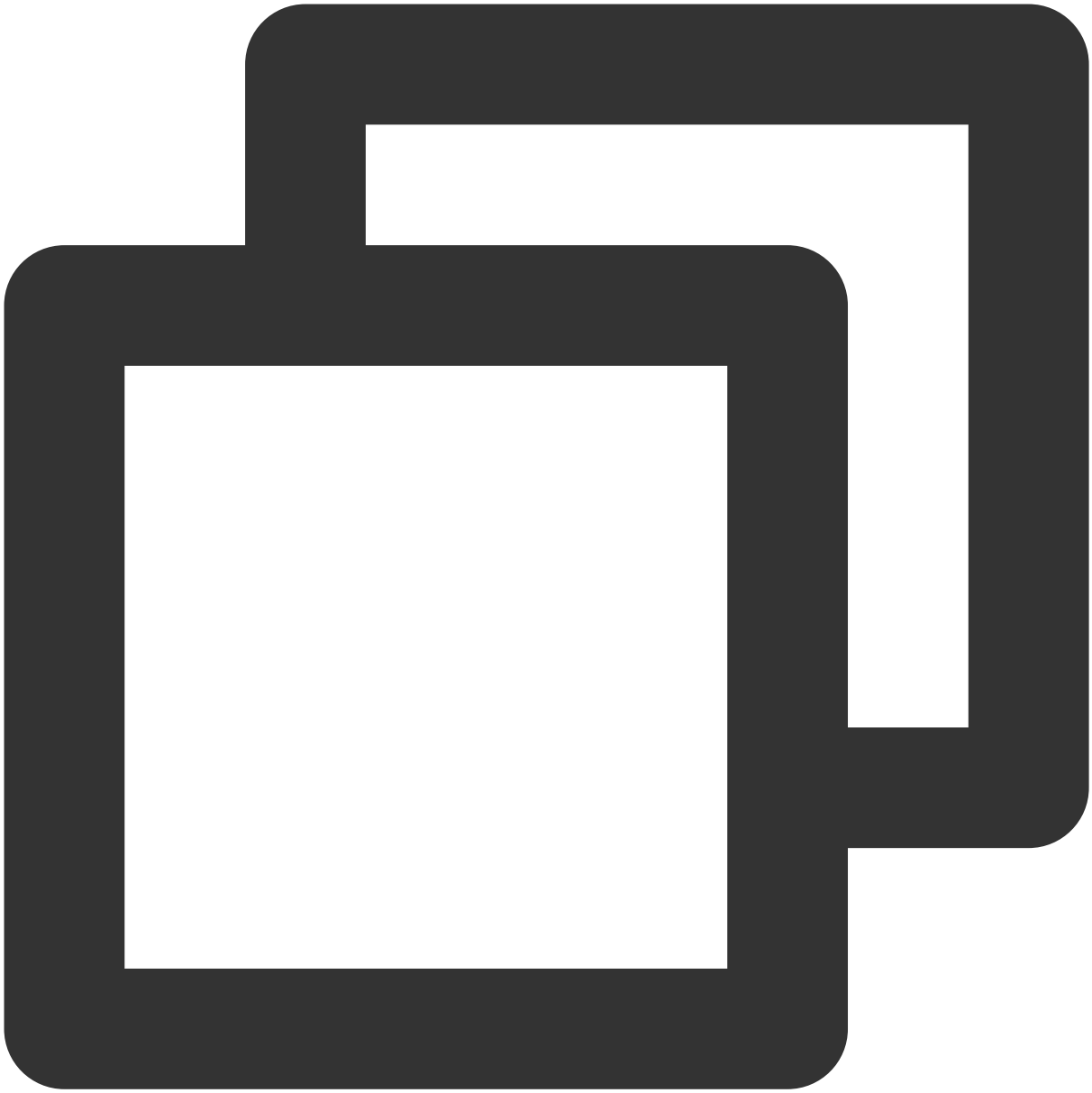
작업 단계

(1) 데이터 마이그레이션 섹션에 따라 HDFS 관련 설정을 완료하고, COSN의 SDK jar 패키지를 HDFS의 해당 디렉터리에 넣습니다.

(2) NodeManager 서비스를 재시작합니다.

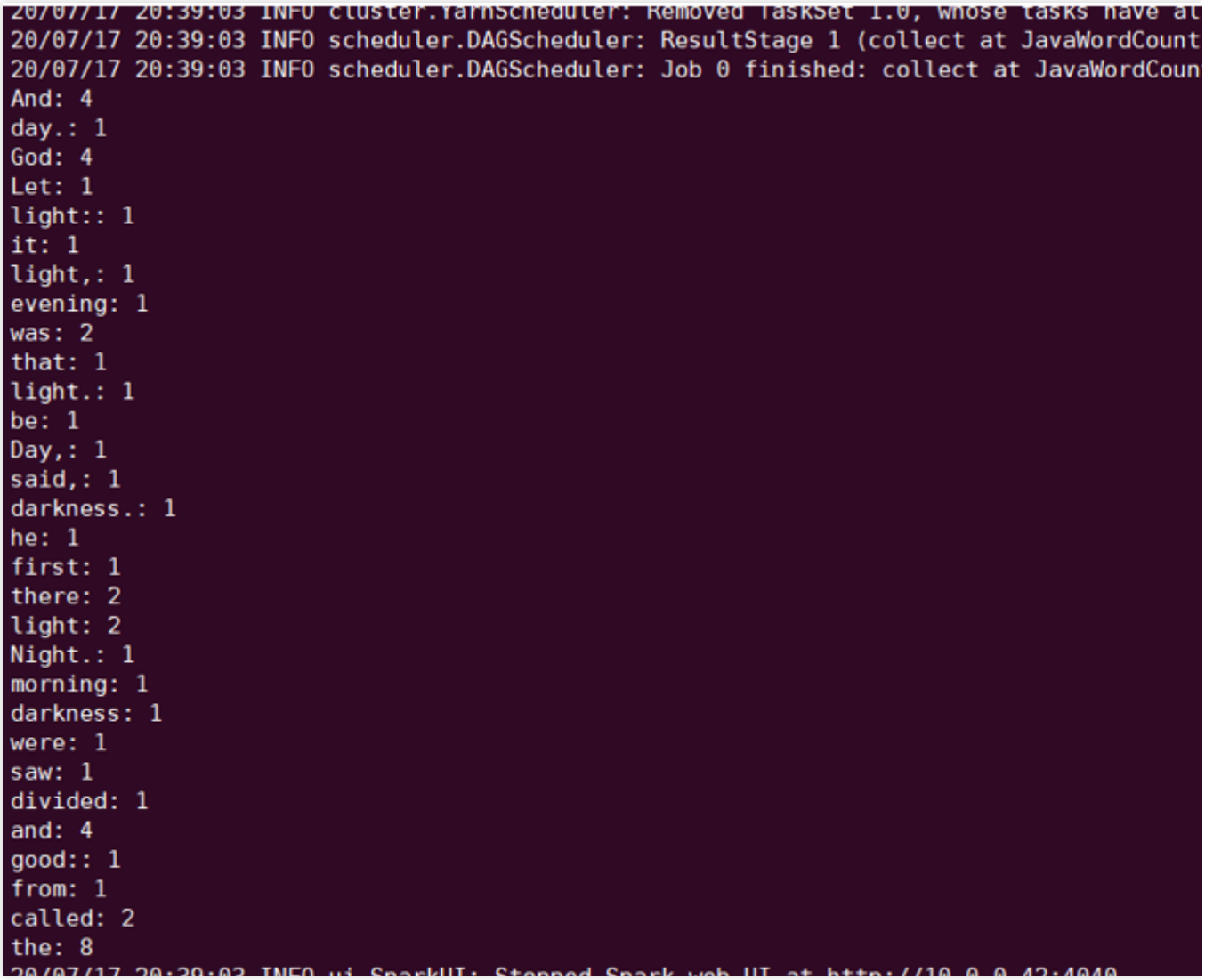
예시

COSN에서 Spark example word count 테스트를 진행합니다.



```
spark-submit  --class org.apache.spark.examples.JavaWordCount --executor-memory 4g
```

실행 결과는 다음과 같습니다.



4. Sqoop

작업 단계

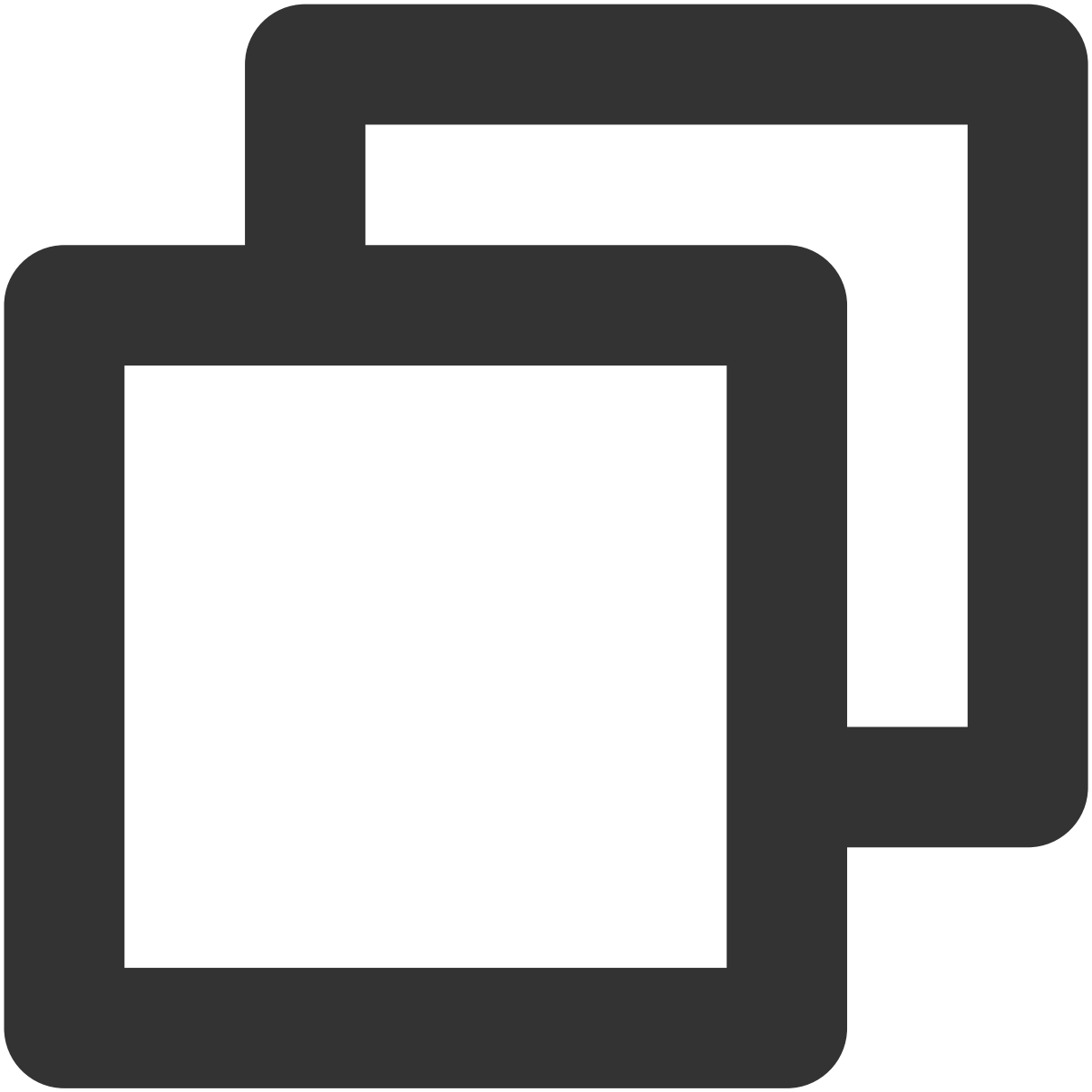
(1) 데이터 마이그레이션 섹션에 따라 HDFS 관련 설정을 완료하고, COSN의 SDK jar 패키지를 HDFS의 해당 디렉터리에 넣습니다.

(2) COSN의 SDK jar 패키지를 sqoop 디렉터리에 넣어야 합니다(예시: /opt/cloudera/parcels/CDH-5.16.1-1.cdh5.16.1.p0.3/lib/sqoop/).

(3) NodeManager 서비스를 재시작합니다.

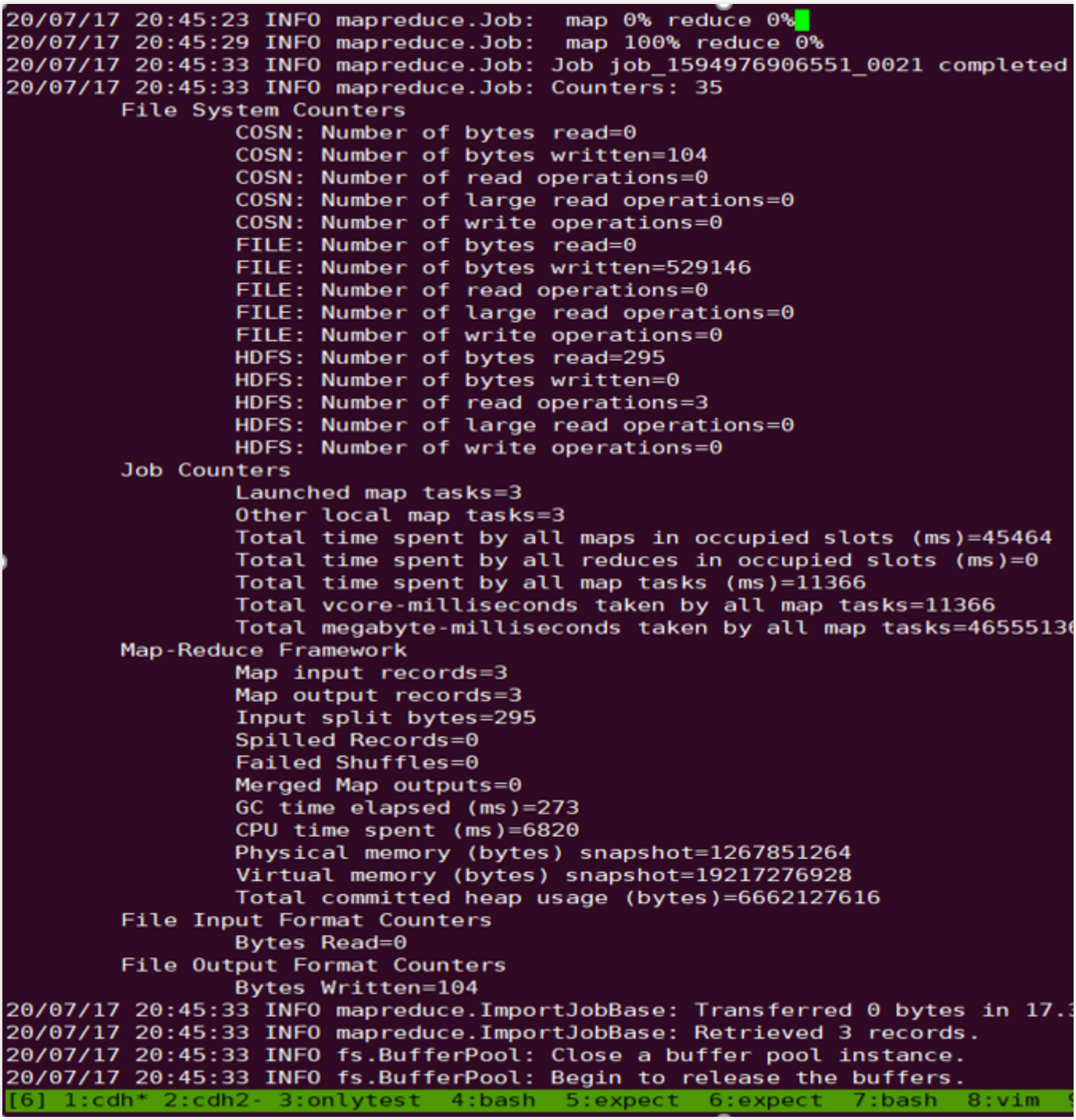
예시

MYSQL 테이블을 COSN에 내보냅니다. Import/Export of Relational Database and HDFS 문서를 참고하여 테스트를 진행하십시오.



```
sqoop import --connect "jdbc:mysql://IP:PORT/mysql" --table sqoop_test --username r
```

실행 결과는 다음과 같습니다.



5. Presto

작업 단계

(1) 데이터 마이그레이션 섹션에 따라 HDFS 관련 설정을 완료하고, COSN의 SDK jar 패키지를 HDFS의 해당 디렉터리에 넣습니다.

(2) COSN의 SDK jar 패키지를 presto 디렉터리에 넣어야 합니다(예시: /usr/local/services/cos_presto/plugin/hive-hadoop2).

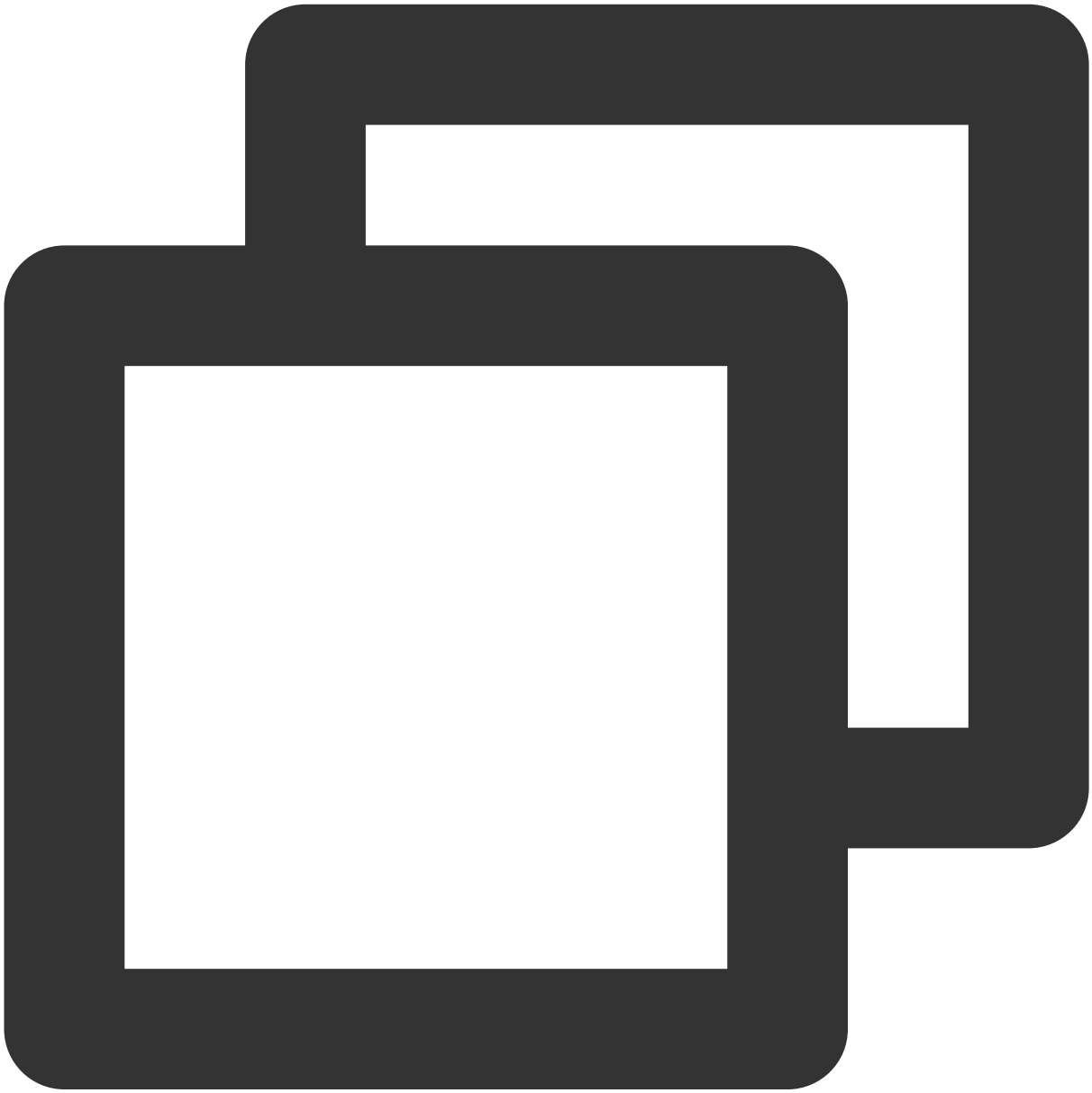
(3) presto는 hadoop common의 gson-2...jar를 로딩할 수 없으므로 gson-2...jar 또한 presto 디렉터리에 넣어야 합니다

(예시: /usr/local/services/cos_presto/plugin/hive-hadoop2, CHDFS만 gson에 종속됨).

(4) HiveServer, HiveMetaStore, Presto 서비스를 재시작합니다.

예시

HIVE에서 Location이 COSN인 테이블 쿼리를 생성합니다:



```
select * from cosn_test_table where bucket is not null limit 1;
```

설명：

cosn_test_table은 location이 cosn scheme인 테이블입니다.

쿼리 결과는 다음과 같습니다.

```
[root@TENCENT64 /usr/local/services/cos_presto_logging-1.0/plugin]# presto
080 --catalog hive --schema inventory_search --debug;
presto:inventory_search> select * from oppo_list_gz_table where bucket is
   appid    |       bucket       |                          path
------------+--------------------+-----------------------------------------
 125        | migrate80105841qq1 | 127454151/20180125/3/5a9df97740b54ab2ba
(1 row)
(END)
```

COS Ranger 권한 시스템 솔루션

최종 업데이트 날짜: : 2024-06-24 16:53:18

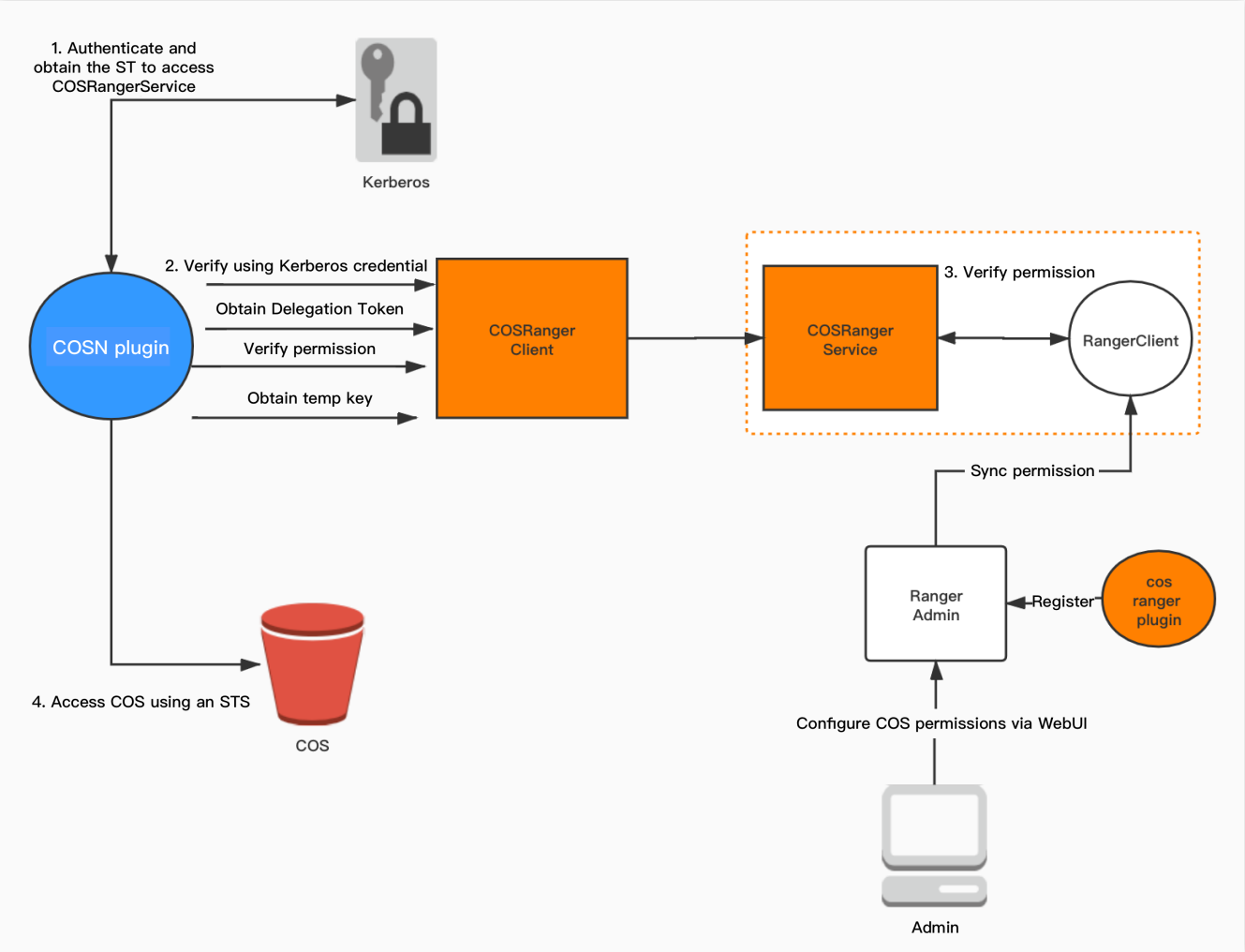
배경

Hadoop Ranger 권한 시스템은 빅 데이터 시나리오에서의 권한 솔루션입니다. 사용자는 Hadoop을 사용해 데이터를 Cloud Object Storage(COS)에 보관합니다. COS는 Tencent Cloud Access Management(CAM) 권한 시스템을 사용하며, 사용자 정보, 권한 정책 등의 부분에서 로컬 Hadoop Ranger 시스템과 다릅니다. Tencent Cloud는 클라이언트의 원활한 사용을 위해 COS의 Ranger 액세스 솔루션을 제공합니다.

강점

세밀하게 분할된 권한 제어, Hadoop 권한 로직 호환으로 사용자가 빅 데이터 컴포넌트와 클라우드 호스팅 스토리지의 권한을 통합 관리합니다.
플러그 인 측에서 core-site 키를 설정할 필요가 없으며, 키는 COS Ranger Service에 통합 설치되어 있어 플레인 텍스트가 노출되지 않습니다.

솔루션 아키텍처



Hadoop 권한 시스템에서 인증은 Kerberos에서, 권한 인증은 Ranger에서 담당합니다. 여기에 다음과 같은 컴포넌트를 제공해 COS의 Ranger 권한 솔루션을 지원합니다.

1. COS Ranger Plugin: Ranger 서버의 서비스 정의 플러그 인을 제공합니다. Ranger 측에서 필요한 매개변수(COS의 bucket 및 region)의 권한 유형 및 정의를 포함하여 COS 서비스 설명을 제공합니다. 해당 플러그 인이 배포되면, 사용자는 Ranger의 제어판에서 권한 정책을 설정할 수 있습니다.
2. COS Ranger Service: 이 서비스는 Ranger의 클라이언트를 통합하여 주기적으로 Ranger 서버의 권한 정책을 동기화하고, 클라이언트의 인증 요청이 수신되면 로컬에서 권한 인증을 진행합니다. 또한, Hadoop의 DelegationToken과 관련된 생성/리스 갱신 API를 제공합니다. 모든 API는 Hadoop IPC를 통해 정의됩니다.
3. COS Ranger Client: COSN 플러그 인을 동적으로 로딩하여 권한 인증 요청을 COS Ranger Service에 전달합니다.

설치 환경

Hadoop 환경.

ZooKeeper, Ranger, Kerberos 서비스(인증 요구 사항이 있는 경우).

설명：

위 서비스는 기술적으로 안정된 오픈 소스 컴포넌트이므로 클라이언트가 직접 설치할 수 있습니다.

설치 컴포넌트

CHDFS Ranger Plugin, COS Ranger Service, COS Ranger Client 및 COSN의 순서로 컴포넌트를 설치합니다.
COS Ranger Plugin 배포
COS Ranger Service 배포
COS Ranger Client 배포
COSN 배포
COS-Ranger-Plugin이 Ranger Admin 콘솔의 서비스 종류를 확장함에 따라, 사용자는 Ranger 콘솔에서 COS와 관련된 작업 권한을 설정할 수 있습니다.

코드 주소

Github의 ranger-plugin 목록으로 이동 후 획득할 수 있습니다.

버전

V1.0 이상 버전.

배포 순서

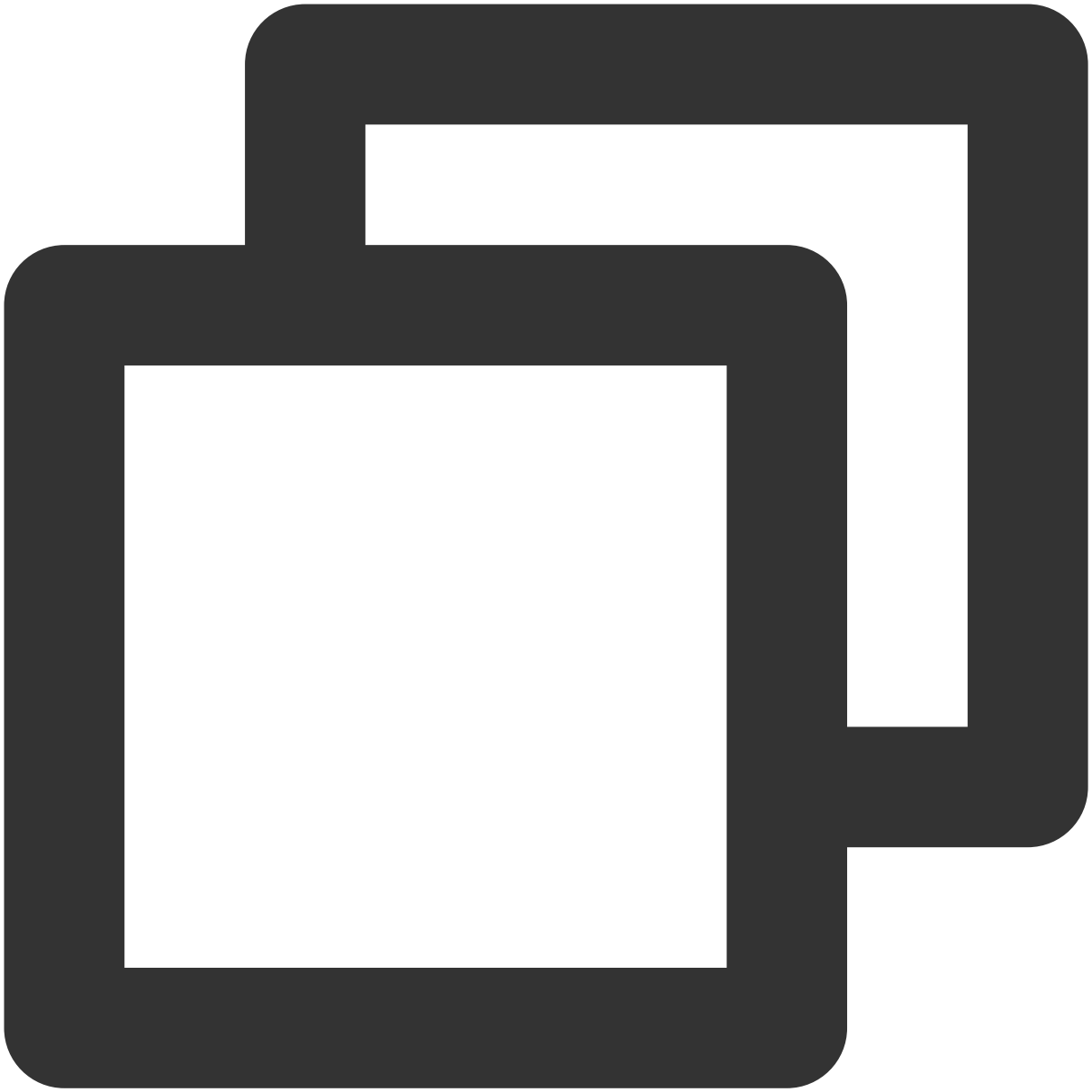
1. Ranger의 서비스 정의 디렉터리 하위에 새로운 COS 디렉터리를 만듭니다(디렉터리 권한은 최소한 x와 r 권한이 필요합니다).
a. Tencent Cloud의 EMR 환경, 경로는 ranger/ews/webapp/WEB-INF/classes/ranger-plugins입니다.
b. 자체구축 hadoop 환경은 ranger 디렉터리 아래에서 hdfs 등 이미 ranger 서비스에 액세스 된 컴포넌트를 찾아 디렉터리 위치를 찾을 수 있습니다.

```
[hadoop@10 ranger-plugins]$ ls -l
total 68
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 atlas
drwxr-xr-x 2 hadoop hadoop 4096 Dec 15 10:57 chdfs
drwxr-xr-x 2 hadoop hadoop 4096 Dec 15 10:57 cos
drwxr-xr-x 2 hadoop hadoop 4096 Feb 25  2020 hbase
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 hdfs
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 hive
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 kafka
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 kms
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 knox
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 kylin
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 nifi
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 nifi-registry
drwxr-xr-x 2 hadoop hadoop 4096 Aug  5 20:48 presto
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 solr
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 sqoop
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 storm
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 yarn
```

2. cos-chdfs-ranger-plugin-xxx.jar를 COS 디렉터리에 저장합니다. 최소한 jar 패키지에 대한 r 권한이 있어야 합니다. 동시에 cos-ranger.json 파일을 넣어주어야 하며, Github에서 얻을 수 있습니다.

3. Ranger 서비스를 재시작합니다.

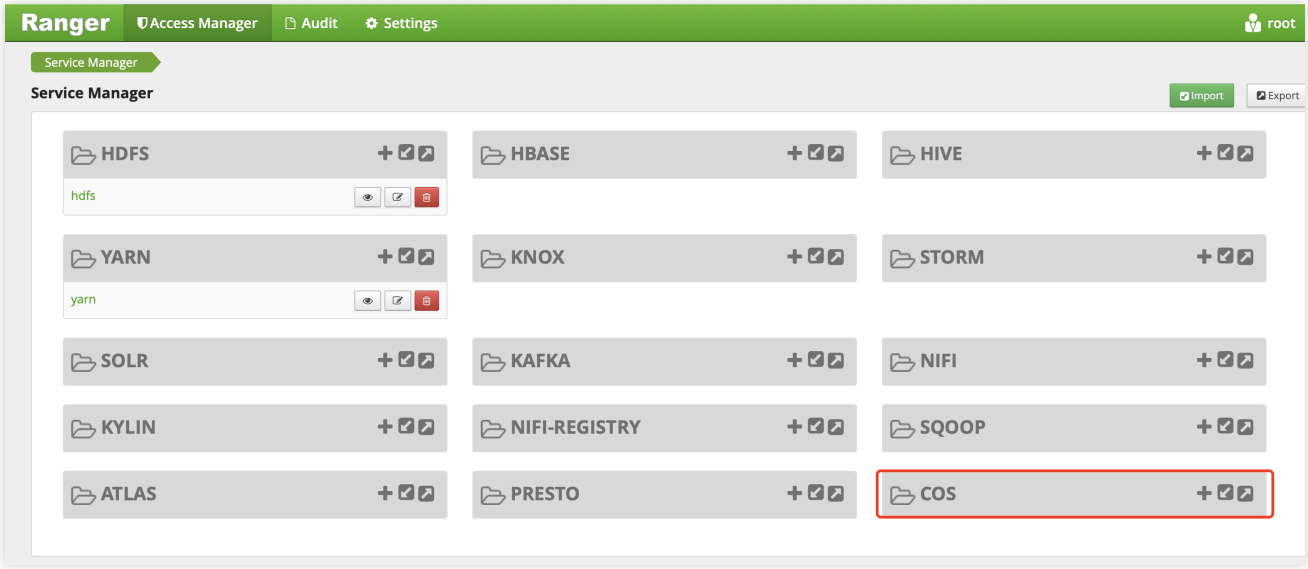
4. Ranger에 COS Service를 등록합니다. 다음과 같은 명령어를 참고할 수 있습니다



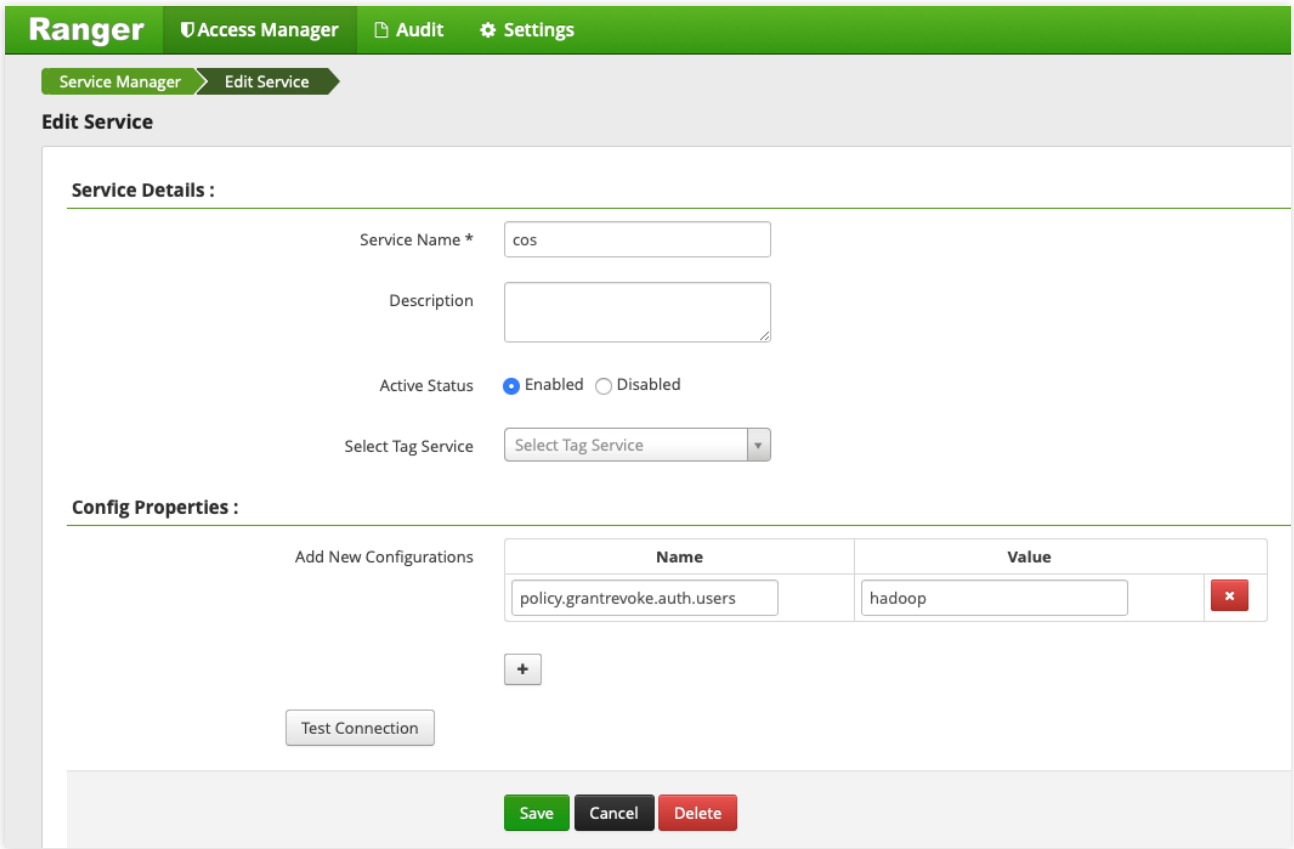
```
##서비스를 생성하려면 Ranger 관리자 계정 암호 및 Ranger 서비스 주소를 입력해야 합니다.
##Tencent Cloud EMR 클러스터의 경우, 관리자는 root이고, 암호는 emr 클러스터를 구축할 때 설정
adminUser=root
##EMR 클러스터 구축 시 설정한 비밀번호는 ranger 서비스 web 페이지 로그인 비밀번호이기도 합니다
adminPasswd=xxxxxx
##ranger 서비스에 여러 master 노드가 있는 경우 하나의 master를 선택할 수 있습니다.
rangerServerAddr=10.0.0.1:6080
##명령 라인에서 -d는 2단계에서 json 파일을 지정합니다.
curl -v -u${adminUser}:${adminPasswd} -X POST -H "Accept:application/json" -H "Cont
##방금 정의된 서비스를 삭제하려면 관련 서비스를 생성할 때 반환했던 서비스 ID를 입력합니다.
serviceId=102
```

```
curl -v -u${adminUser}:${adminPasswd} -X DELETE -H "Accept:application/json" -H "Co
```

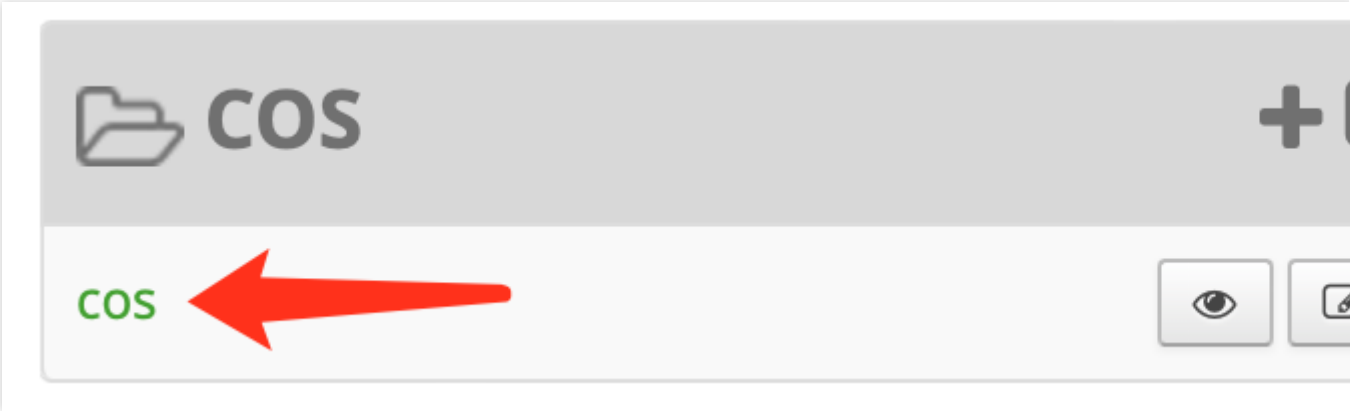
5. 서비스가 생성되면 Ranger 콘솔에서 COS 서비스를 다음과 같이 볼 수 있습니다.



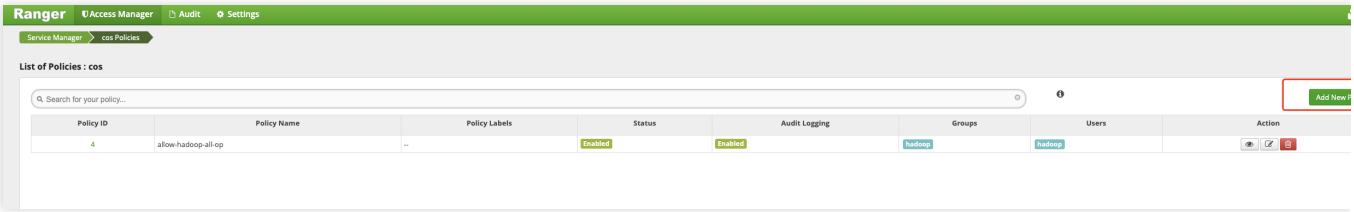
6. COS 서비스에서 + 를 클릭해 새로운 서비스의 인스턴스를 정의합니다. 예를 들어 `cos` 혹은 `cos_test` 서비스의 설정은 다음과 같습니다.



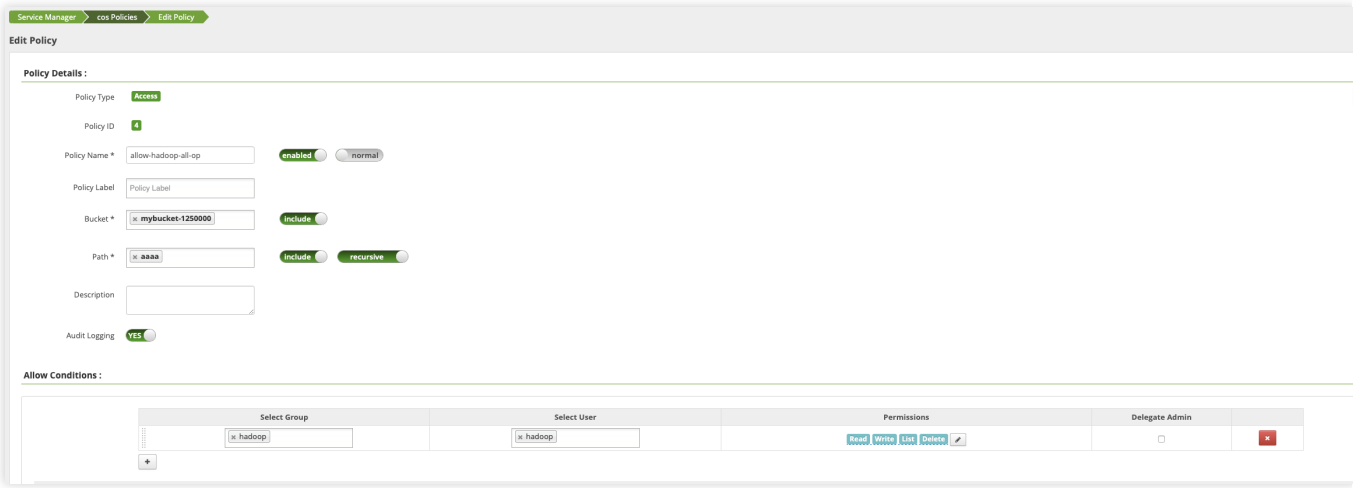
여기서 policy.grantrevoke.auth.users는 COS Ranger Service를 시작하는 데 사용되는 사용자 이름(권한 가져오기 정책을 허용하는 사용자)으로 설정해야 합니다. 일반적으로 hadoop 설정이 권장되며, 이후 작업에서 COS Ranger Service는 이 사용자 이름으로 실행할 수 있습니다.
7. 생성된 COS 서비스 인스턴스를 클릭합니다.



다음 그림과 같이 policy를 추가합니다.



8. 리디렉션 페이지에서 다음 매개변수를 설정합니다.
bucket: examplebucket-1250000000과 같은 버킷 이름은 COS 콘솔에 로그인하여 조회할 수 있습니다.
path: COS객체 경로입니다. COS의 객체 경로가 /로 시작되지 않도록 주의하십시오.
include: 설정된 권한이 path 자체에 적용되었는지, 혹은 path 이외의 다른 경로에 적용되었는지 나타냅니다.
recursive: 권한이 path 외에 path 경로의 하위 구성원(귀속되는 구성원)에도 적용된다는 것을 나타냅니다. 보통 path 설정을 디렉터리로 사용하는 경우입니다.
user/group: 사용자 이름과 사용자 그룹입니다. 둘 중 하나만 충족하면 되는 관계로, 사용자 이름 혹은 사용자 그룹 중 하나를 만족하면 상응하는 작업 권한을 보유할 수 있습니다.
Permissions:
Read: 읽기 작업입니다. 객체 다운로드, 객체 메타데이터 조회 등 COS의 GET, HEAD와 같은 작업에 대응됩니다
Write: 쓰기 작업입니다. 객체 스토리지 안에 있는 PUT와 같은 수정 작업, 예를 들어 객체 업로드와 같은 작업에 대응합니다.
Delete: 삭제 작업입니다. COS의 Object 삭제와 대응됩니다. Hadoop의 Rename 작업의 경우, 기존 경로에 대한 삭제 작업과 신규 경로에 대한 입력 작업 권한이 필요합니다.
List: 탐색 권한입니다. COS의 List Object에 대응됩니다.



COS Ranger Service는 전체 권한 시스템의 핵심으로, ranger 클라이언트와 통합되어 ranger client의 인증 요청, token 생성 및 리스 갱신 요청, 임시 키 생성 요청을 수신합니다. 민감한 정보(Tencent Cloud 키 정보)가 저장되는 곳이기도 합니다. 보통 점프 서버에 배포되며 클러스터 관리자만 작업을 수행하고 구성을 쿼리할 수 있습니다.
COS Ranger Service는 단일 프라이머리, 다중 세컨더리 HA 배포를 지원하며, DelegationToken은 HDFS에 저장할 수 있습니다. ZK 잠금 장치로 Leader의 신분을 결정합니다. Leader의 신분을 획득한 프라이머리 노드는 COS Ranger Client가 주소 라우팅을 수행할 수 있도록 ZK에 주소를 기록합니다.

코드 주소

Github의 cos-ranger-server 디렉터리로 이동하여 얻을 수 있습니다.

버전

V5.1.2 이상 버전

배포 순서

1. COS Ranger Service 서비스 코드를 클러스터의 기기 몇 대에 복사하고, 생성 환경은 최소 기기 두 대(메인 기기와 예비용 기기)를 권장합니다. 중요한 정보를 다루기 때문에 점프 서버 혹은 권한이 엄격히 통제되는 기기를 권장합니다.
2. cos-ranger.xml 파일에서 구성을 수정합니다. 다음은 필요한 수정 사항입니다. 구성 항목에 대한 자세한 내용은 파일의 설명을 참고하십시오(구성 파일은 Github의 cos-ranger-service/conf 디렉터리에서 얻을 수 있습니다).

qcloud.object.storage.rpc.address

qcloud.object.storage.status.port

qcloud.object.storage.enable.cos.ranger

qcloud.object.storage.zk.address (zk 주소, cos ranger service 실행 후 zk에 등록)

qcloud.object.storage.cos.secret.id

qcloud.object.storage.cos.secret.key

3. ranger-cos-security.xml 파일에서 구성을 수정합니다. 다음은 필요한 수정 사항입니다. 구성 항목에 대한 자세한 내용은 파일의 설명을 참고하십시오(구성 파일은 Github의 cos-ranger-service/conf 디렉터리에서 얻을 수 있습니다).

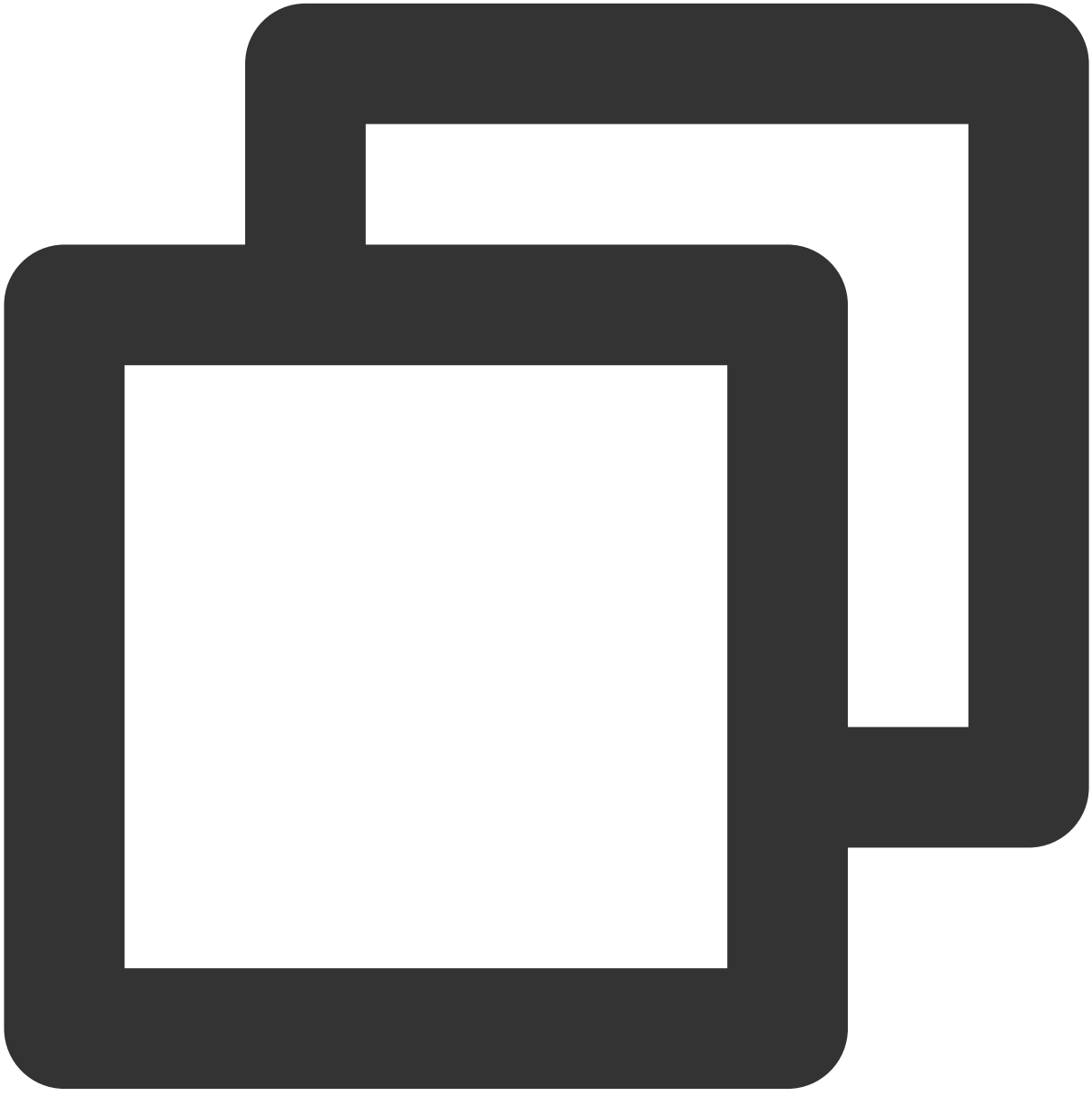
ranger.plugin.cos.policy.cache.dir

ranger.plugin.cos.policy.rest.url

ranger.plugin.cos.service.name

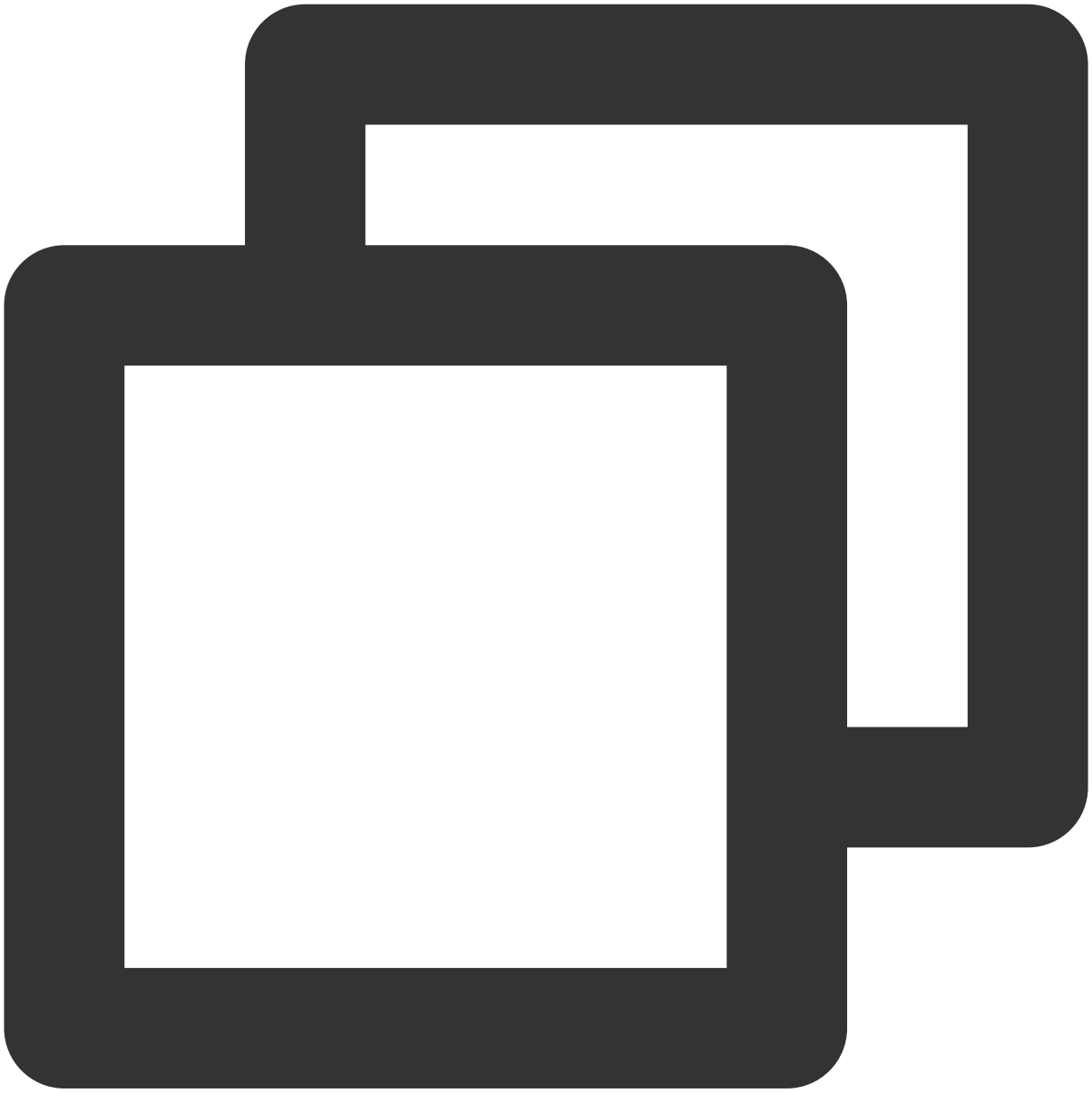
4. start_rpc_server.sh에서 hadoop_conf_path, java.library.path의 설정을 수정합니다. 이 두 가지 설정은 각각 hadoop 설정 파일이 들어 있는 디렉터리(core-site.xml, hdfs-site.xml 등) 및 hadoop native lib 경로를 가리킵니다.

5. 다음 명령어를 실행하여 서비스를 시작합니다.



```
chmod +x start_rpc_server.sh
nohup ./start_rpc_server.sh &> nohup.txt &
```

6. 실행에 실패하면 log의 error 로그에 잘못된 정보가 있는지 조회합니다.
7. COS Ranger Service는 HTTP 포트 상태(포트 이름은 qcloud.object.storage.status.port, 기본값은 9998)표시를 지원합니다. 사용자는 다음의 명령을 실행하여 leader의 포함 여부 및 인증 통계와 같은 상태 정보를 얻을 수 있습니다.



```
# 10.xx.xx.xxx를 ranger service와 함께 배포된 장치의 IP 주소로 바꿉니다.
# 명령의 9998을 구성 파일의 qcloud.object.storage.status.port 값으로 바꿉니다.
curl -v http://10.xx.xx.xxx:9998/status
```

하나의 cos ranger service 노드만 배포된 경우 위의 인터페이스 응답에서 현재 노드가 leader로 표시됩니다.

여러 cos ranger service 노드가 배치된 경우 위의 인터페이스 응답에서 다른 노드가 leader가 되는 것을 볼 수 있으며 모든 노드를 다시 시작한 후 가장 먼저 다시 시작된 노드가 leader로 표시됩니다.
COS Ranger Client는 hadoop cosn 플러그인에 의해 동적으로 로딩됩니다. 임시 키 획득, token 획득, 인증 작업과 같은 모든 COS Ranger Service 액세스 요청을 캡슐화하는 프록시입니다.

소스 코드 다운로드

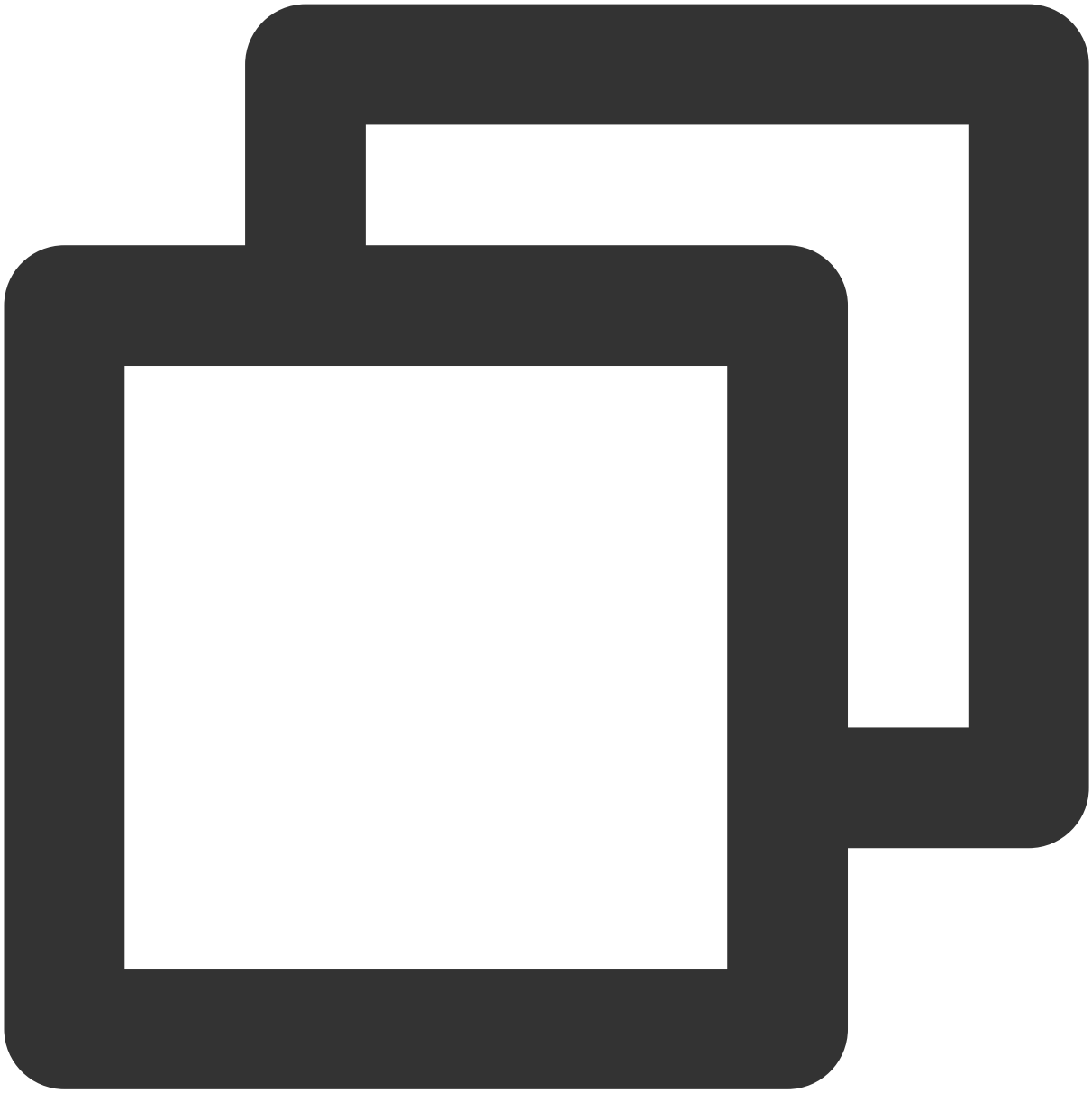
Github의 cos-ranger-client 및 cosn-ranger-interface 디렉터리로 이동하여 소스 코드를 얻을 수 있습니다.

버전

cos-ranger-client는 V5.0 이상 버전이 필요합니다.cosn-ranger-interface는 v1.0.4 이상 버전이 필요합니다.

배포 방식

1. cos-ranger-client jar 패키지와 cosn-ranger-interface jar 패키지를 COSN과 동일한 디렉터리(일반적으로 /usr/local/service/hadoop/share/hadoop/common/lib/ 디렉터리)에 복사하십시오. 복사를 선택하고 hadoop 자체의 주요 버전과 일치하는 jar 패키지를 선택하고 마지막으로 jar 패키지에 읽기 권한이 있는지 확인합니다.
2. core-site.xml에 다음과 같은 설정 항목을 추가하십시오.



```
<configuration>
            <!--******필수 설정*********-->
            <!-- 이전 단계에서 배포된 cos ranger server 주소 -->
            <property>
                <name>qcloud.object.storage.ranger.service.address</name>
                <value>10.0.0.8:9999,10.0.0.10:9999</value>
            </property>

            <!--***선택적 구성****-->
            <!-- cos ranger service 측에서 사용되는 kerberos 자격 증명을 설정하고, cos ra
```

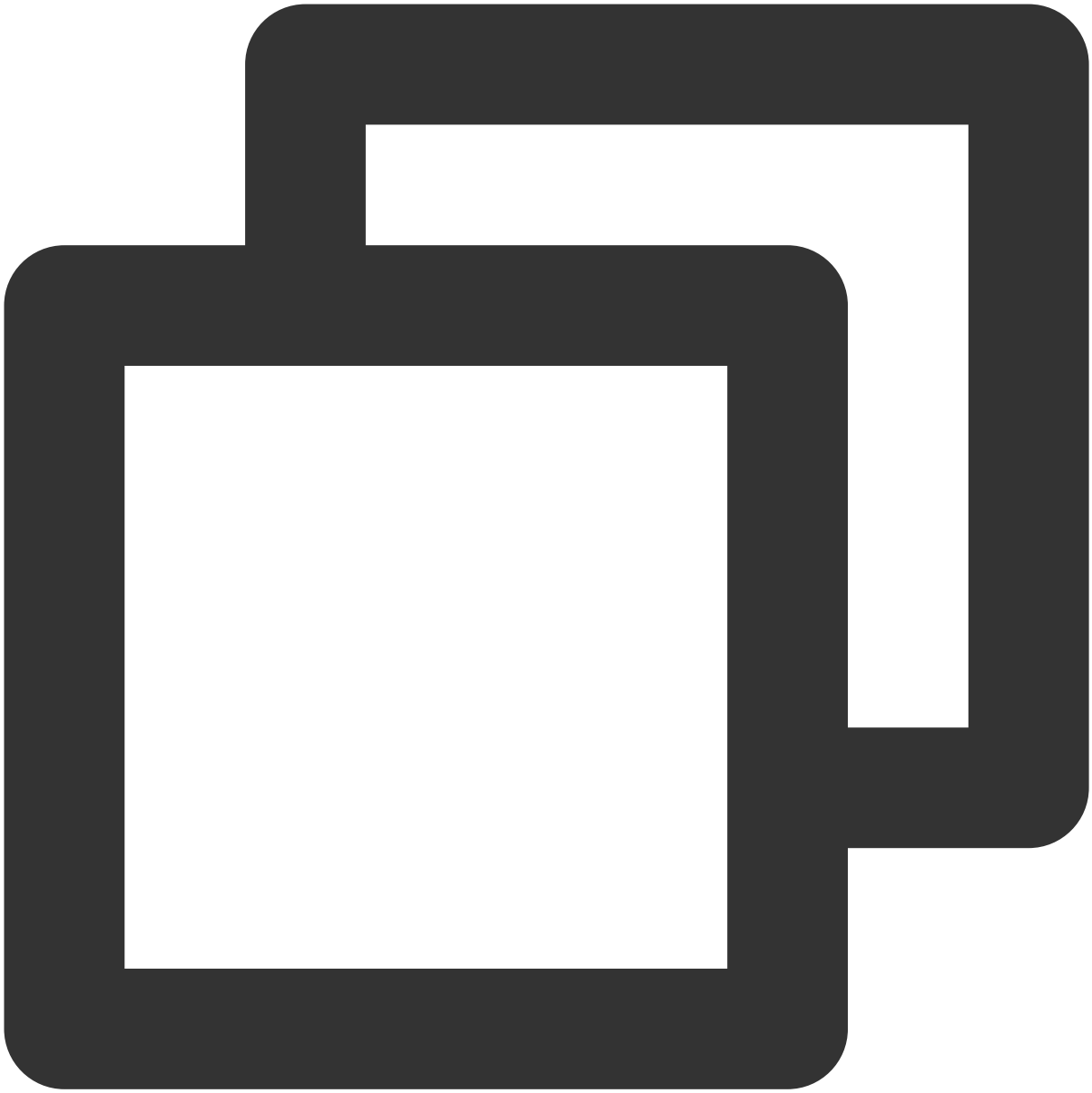
```
            <property>
                      <name>qcloud.object.storage.kerberos.principal</name>
                      <value>hadoop/_HOST@EMR-XXXX</value>
            </property>

          <!--****선택적 구성****-->
          <!-- zk에 기록된 ranger server의 IP 주소 경로, 여기서는 기본값을 사용합니다. 값은
           <property>
                     <name>qcloud.object.storage.zk.leader.ip.path</name>
                     <value>/ranger_qcloud_object_storage_leader_ip</value>
           </property>
          <!-- zk에 기록된 cos ranger service follower의 IP 주소 경로입니다. 여기서 기본값
           <property>
                     <name>qcloud.object.storage.zk.follower.ip.path</name>
                     <value>/ranger_qcloud_object_storage_follower_ip</value>
           </property>
</configuration>
```

버전

V8.0.1 이상 버전

배포 방식

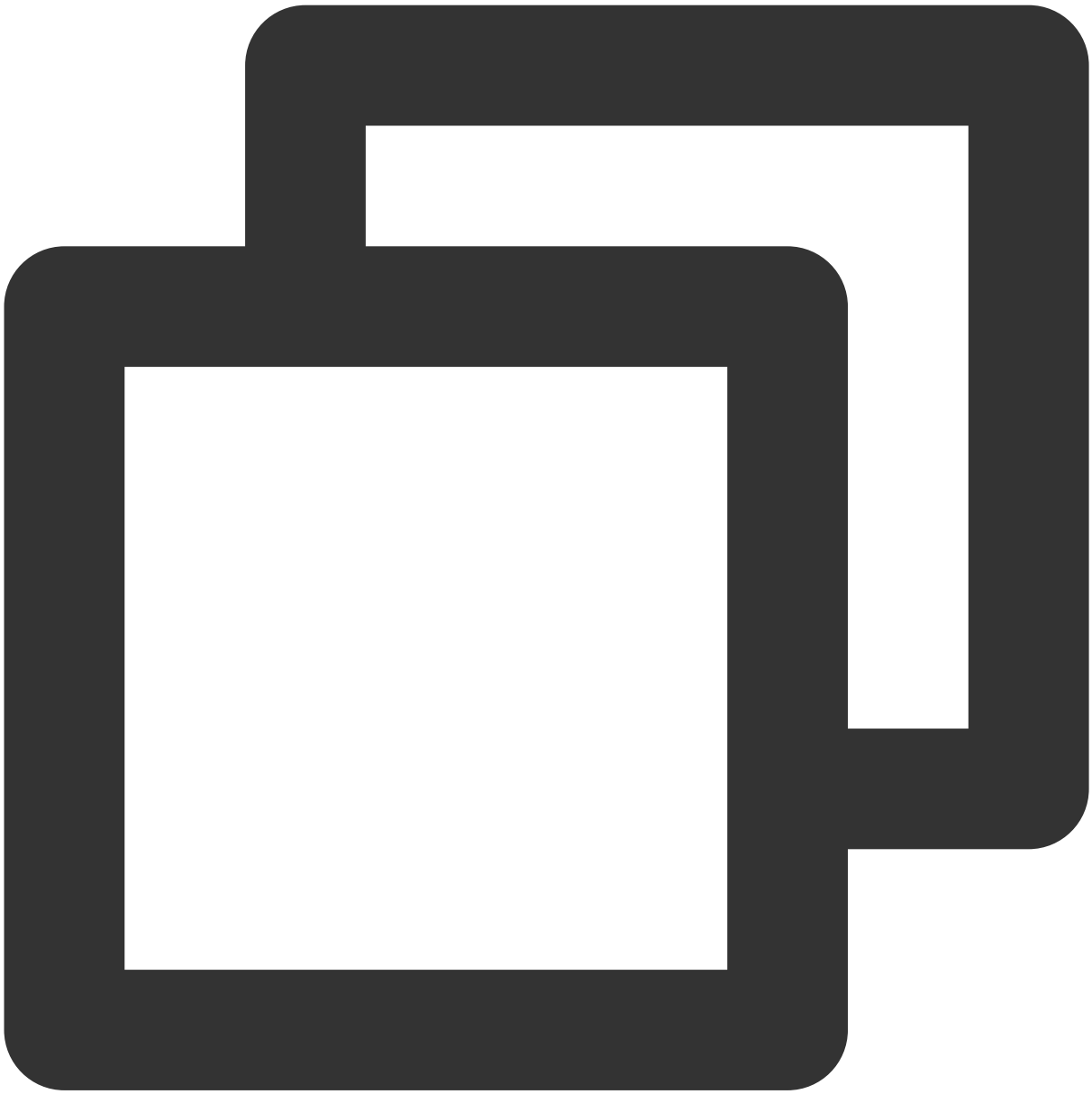
COSN 플러그 인 배포 방법은 Hadoop 툴 문서를 참고하십시오. 단, 다음 몇 가지를 주의하십시오.
1. ranger를 사용한 후, fs.cosn.userinfo.secretId와 fs.cosn.userinfo.secretKey 키 정보는 설정이 필요하지 않습니다. COSN 플러그 인은 향후 COSRangerService를 통해 임시 키를 부여받게 됩니다.
2. fs.cosn.credentials.provider는 org.apache.hadoop.fs.auth.RangerCredentialsProvider 설정이 되어야만 Ranger를 통해 다음과 같은 인증 진행이 가능합니다.



```
<property>
         <name>fs.cosn.credentials.provider</name>
         <value>org.apache.hadoop.fs.auth.RangerCredentialsProvider</value>
</property>
```

인증

1. hadoop cmd를 사용해 COSN 관련 작업을 액세스합니다. 현재 사용자가 하고 있는 작업이 루트 계정의 권한 설정 예측에 부합하는지 확인합니다.



```
#bucket, 경로 및 기타 정보를 루트 계정의 정보로 교체합니다.
hadoop fs -ls cosn://examplebucket-1250000000/doc
hadoop fs -put ./xxx.txt cosn://examplebucket-1250000000/doc/
hadoop fs -get cosn://examplebucket-1250000000/doc/exampleobject.txt
hadoop fs -rm cosn://examplebucket-1250000000/doc/exampleobject.txt
```

2. MR Job을 사용하여 인증합니다. 인증 전 Yarn, Hive 등 관련 서비스를 재시작해야 합니다.

FAQ

kerberos 설치가 꼭 필요한가요?

Kerberos는 인증 요구 사항을 충족합니다. 클러스터와 사용자를 신뢰할 수 있고 인증 목적이 권한이 없는 사용자로 인한 오작동을 방지하는 것뿐이라면 Kerberos 설치를 건너뛰고 인증을 위해 ranger만 사용할 수 있습니다. 사실 Kerberos는 일부 성능을 저하시킬 수도 있습니다. 따라서 보안과 성능에 대한 요구 사항의 균형을 맞출 수 있습니다. 인증이 필요한 경우 Kerberos를 활성화한 다음 COS Ranger Service 및 COS Ranger Client를 구성할 수 있습니다.

Ranger를 실행했는데 Policy가 설정되어 있지 않거나 Policy가 매칭되지 않았을 경우, 어떻게 작업하나요?

Policy가 매칭되지 않았을 경우 기본적으로 해당 작업이 거부됩니다.

COS Ranger Service 측의 키 설정은 서브 계정도 가능한가요?

서브 계정도 가능합니다. 그러나 bucket을 실행할 수 있는 권한이 있어야만, COSN 플러그 인에 들어가 관련 작업을 할 수 있는 임시 키 생성이 가능합니다. 일반적으로 여기서 설정하는 키는 해당 bucket의 모든 권한을 갖기를 권장합니다.

임시 키는 어떻게 업데이트 되나요? 그리고 COS를 방문할 때마다 COS Ranger Service에서 임시 키를 획득해야 하나요?

임시 키는 COSN 플러그 인 cache에 있으며, 주기적으로 비동기화 업데이트됩니다.

ranger 페이지에서 Policy를 변경했는데 적용되지 않으면 어떻게 해야 하나요?

ranger-cos-security.xml 파일에서 ranger.plugin.cos.policy.pollIntervalMs 값(밀리초)을 줄이고 COS Ranger Service를 다시 시작합니다. Policy 관련 테스트가 끝난 후에는 다시 원래 값으로 변경하는 것이 좋습니다(시간 간격이 너무 짧으면 라운드 로빈 빈도가 높아 CPU 사용률이 높아짐).

Oceanus를 COS에 연결

최종 업데이트 날짜: : 2024-06-24 16:53:18

Oceanus 개요

Oceanus는 빅 데이터 생태계의 강력한 실시간 분석 도구입니다. 이를 통해 웹사이트 클릭스트림 분석, 이커머스 정밀 타켓팅 추천, IoT 등 다양한 애플리케이션을 단 몇 분 만에 쉽게 구축할 수 있습니다. Oceanus는 Apache Flink를 기반으로 개발되었으며 완전 관리형 클라우드 서비스를 제공하므로 인프라의 Ops에 대해 걱정할 필요가 없습니다. 또한 클라우드의 데이터 소스에 연결하여 완전한 지원 서비스를 받을 수 있습니다.
Oceanus는 SQL 분석 문을 작성하고, 사용자 정의 JAR 패키지를 업로드 및 실행하고, 작업을 관리할 수 있는 편리한 콘솔과 함께 제공됩니다. Flink 기술을 기반으로 PB 수준의 데이터 세트에서 1초 미만의 처리 대기 시간을 달성할 수 있습니다.
이 문서에서는 Oceanus를 COS에 연결하는 방법에 대해 설명합니다. 현재 Oceanus는 전용 클러스터 모드에서 사용할 수 있으며, 여기에서 다양한 작업을 실행하고 자신의 클러스터에서 관련 리소스를 관리할 수 있습니다.

준비 작업

Oceanus 클러스터 생성

Oceanus 콘솔에 로그인하여 Oceanus 클러스터를 생성합니다.

COS 버킷 생성

1. COS 콘솔에 로그인합니다.
2. 왼쪽 사이드바에서 **버킷 리스트**를 클릭합니다.
3. **버킷 생성**을 클릭하여 버킷 생성의 안내에 따라 버킷을 생성합니다.

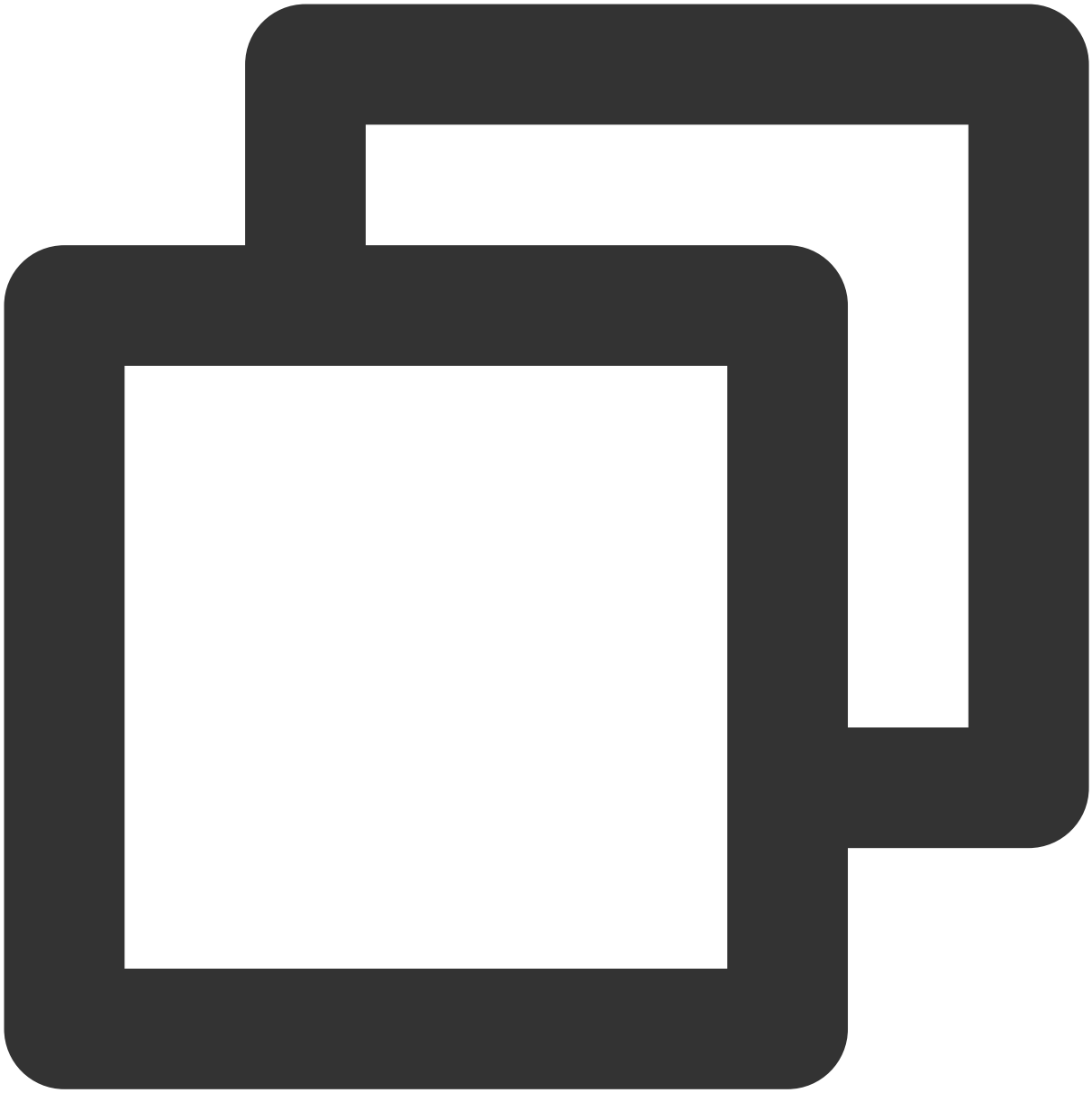
설명 :

COS에 데이터를 쓸 때 Oceanus 작업은 COS와 동일한 리전에서 실행되어야 합니다.

실행 순서

Oceanus 콘솔로 이동하여 SQL 작업을 생성하고 COS와 동일한 리전의 클러스터를 선택합니다.

1. Source 생성



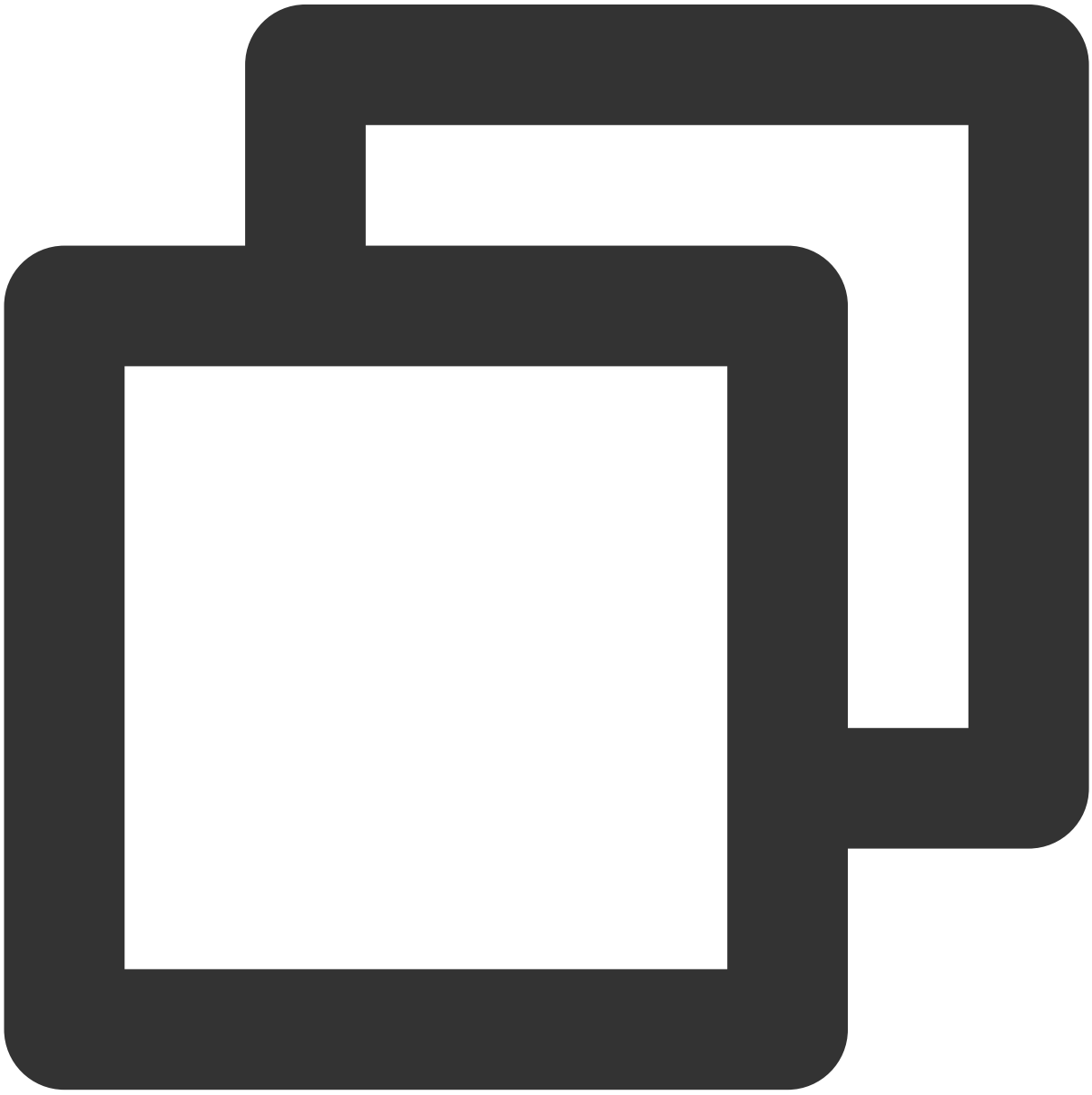
```
CREATE TABLE `random_source` (
  f_sequence INT,
  f_random INT,
  f_random_str VARCHAR
  ) WITH (
  'connector' = 'datagen',
  'rows-per-second'='10',                    -- 초당 생성된 데이터 행 수
  'fields.f_sequence.kind'='random',         -- 랜덤 숫자
  'fields.f_sequence.min'='1',               -- 최소 순차 번호
  'fields.f_sequence.max'='10',              -- 최대 순차 번호
  'fields.f_random.kind'='random',           -- 랜덤 숫자
```

```
    'fields.f_random.min'='1',                  -- 최소 랜덤 숫자
    'fields.f_random.max'='100',                -- 최대 랜덤 숫자
    'fields.f_random_str.length'='10'           -- 랜덤 문자열 길이
  );
```

설명：

여기서는 내장 connector 'datagen'이 선택되었습니다. 실제 비즈니스 요구 사항에 따라 데이터 소스를 선택하십시오.

2. Sink 생성

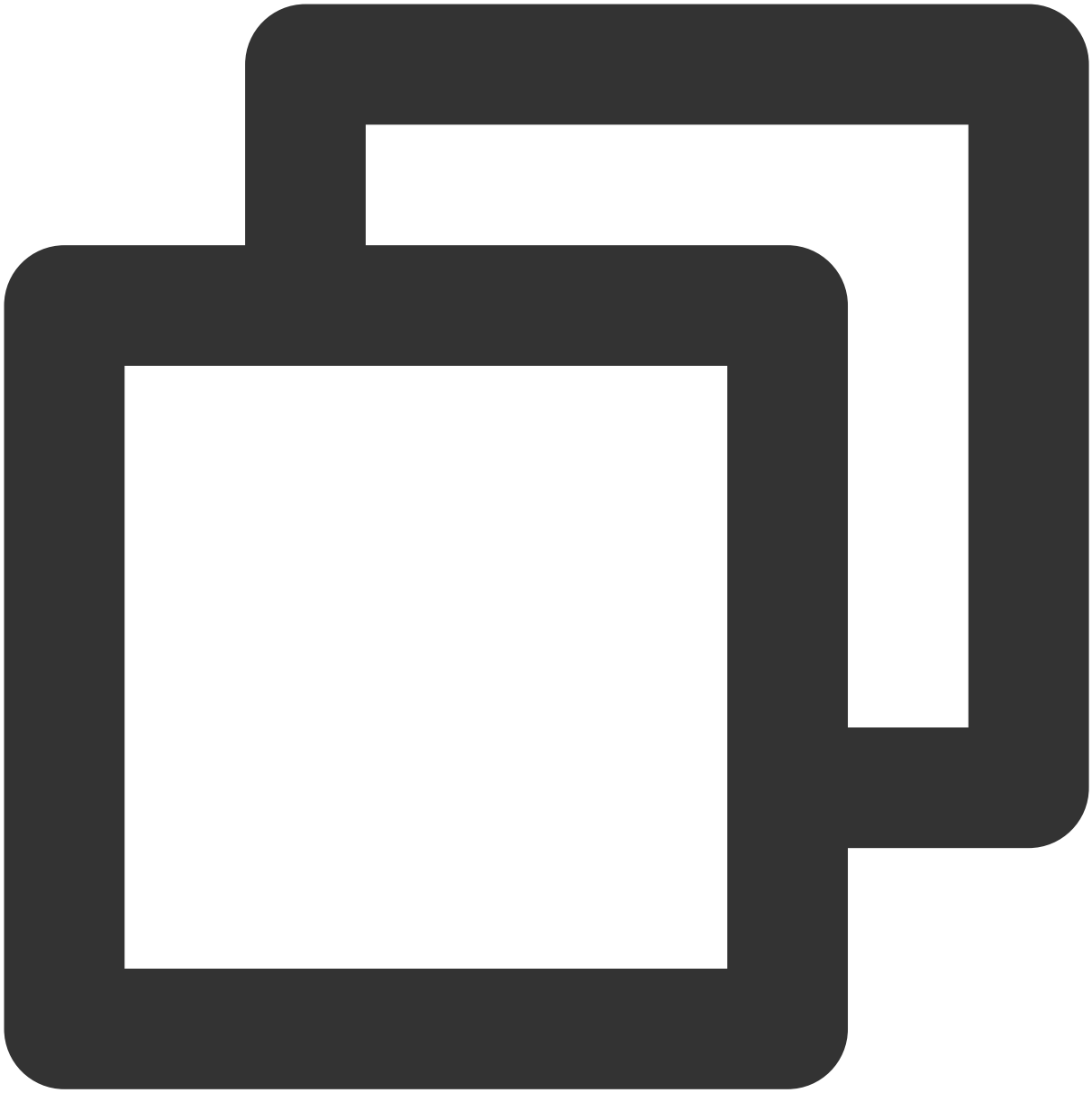


```
-- <bucket name> 및 <folder name>을 실제 버킷 및 폴더 이름으로 바꿉니다.
CREATE TABLE `cos_sink` (
  f_sequence INT,
  f_random INT,
  f_random_str VARCHAR
) PARTITIONED BY (f_sequence) WITH (
    'connector' = 'filesystem',
    'path'='cosn://<bucket name>/<folder name>/',                  --- 데이터가 기록될
    'format' = 'json',                                            --- 기록된 데이터의 형식
    'sink.rolling-policy.file-size' = '128MB',                    --- 최대 파일 크기
    'sink.rolling-policy.rollover-interval' = '30 min',           --- 최대 파일 쓰기 시간
    'sink.partition-commit.delay' = '1 s',                        --- 파티션 커밋 지연
    'sink.partition-commit.policy.kind' = 'success-file'          --- 파티션 커밋 방법
);
```

설명 :

Sink의 WITH 매개변수에 대한 자세한 내용은 Filesystem (HDFS/COS)을 참고하십시오.

3. 비즈니스 로직 구성



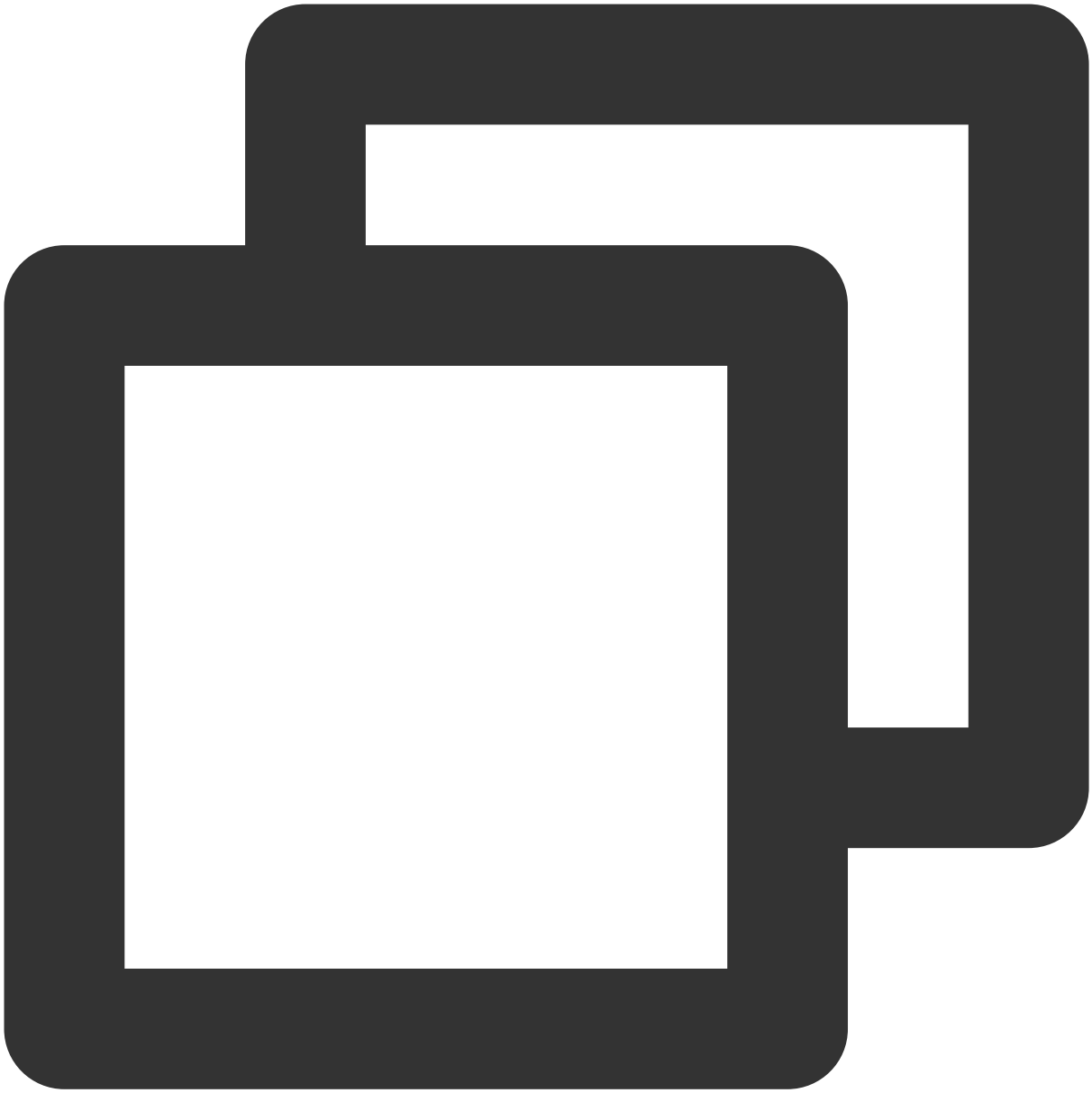
```
INSERT INTO `cos_sink`
SELECT * FROM `random_source`;
```

주의 :

이는 예시용이며 실제 비즈니스 목적이 없습니다.

4. 작업 매개변수 설정

내장 Connector로 `flink-connector-cos` 를 선택하고 **고급 매개변수**에서 COS URL을 다음과 같이 구성합니다.



```
fs.AbstractFileSystem.cosn.impl: org.apache.hadoop.fs.CosN
fs.cosn.impl: org.apache.hadoop.fs.CosFileSystem
fs.cosn.credentials.provider: org.apache.flink.fs.cos.OceanusCOSCredentialsProvider
fs.cosn.bucket.region: <COS 리전>
fs.cosn.userinfo.appid: <COS 사용자 appid>
```

작업은 다음과 같이 구성됩니다.

`<COS region>` 을 ap-guangzhou와 같은 실제 COS 리전으로 바꿉니다.

`<COS user appid>` 를 실제 APPID로 바꿉니다. 이는 계정 센터에서 볼 수 있습니다.

설명 :

작업 매개변수 설정에 대한 자세한 내용은 Filesystem (HDFS/COS)을 참고하십시오.

5. 작업 시작

저장 > 구문 확인 > 초안 릴리스를 클릭하고 SQL 작업이 시작될 때까지 기다렸다가 해당 COS 디렉터리로 이동하여 작성된 데이터를 확인합니다.

3rd party 애플리케이션에서 COS 사용
S3와 호환되는 타사 애플리케이션에서 COS의 일반 구성 사용

최종 업데이트 날짜: : 2024-06-24 16:53:18

Amazon Simple Storage Service(Amazon S3, 이하 S3)는 AWS에서 출시한 최초의 클라우드 서비스 중 하나입니다. 수년간의 개발 끝에 S3 프로토콜은 객체 스토리지 분야에서 사실상의 표준이 되었습니다. Tencent COS(Cloud Object Storage)는 S3 호환 구현 방식을 제공하므로 대부분의 S3 호환 애플리케이션에서 COS 서비스를 직접 사용할 수 있습니다. 본 문서에서는 COS를 사용하도록 이러한 애플리케이션을 구성하는 방법을 설명합니다.

준비 작업

애플리케이션의 COS 서비스 사용 가능 여부 확인

설명에 `S3 Compatible` 이 가능한 애플리케이션은 대부분의 경우 COS를 사용할 수 있습니다. 일부 기능이 제대로 작동하지 않는 경우 고객센터에 도움을 요청하십시오. 본 문서의 가이드를 따랐음을 표시하고 애플리케이션 이름 및 스크린샷과 같은 정보를 제공하십시오.
애플리케이션 설명에 `Amazon S3` 가 지원된다는 내용만 있으면 애플리케이션이 S3 서비스를 사용할 수 있지만 COS를 사용할 수 있는지 여부는 아래에 설명된 대로 관련 구성에서 추가로 평가해야 합니다.

COS 서비스 준비

1단계: Tencent Cloud 계정 가입

(이미 Tencent Cloud 계정이 있는 경우 이 단계를 생략할 수 있습니다.)

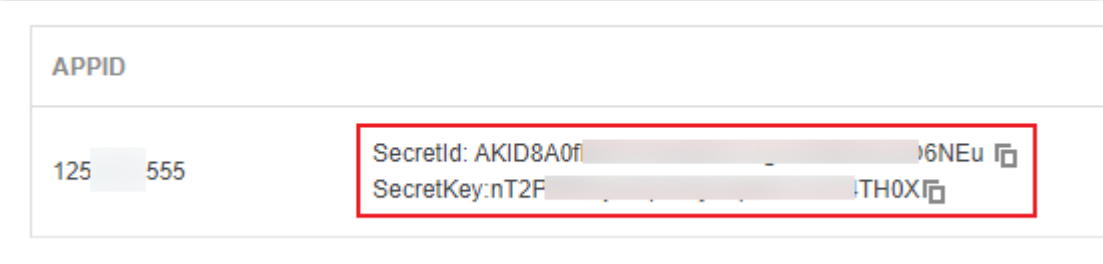
2단계: 실명 인증

(이미 완료한 경우 이 단계를 생략할 수 있습니다.)
자세한 인증 과정은 실명 인증 가이드 를 참고하십시오.

3단계: COS 서비스 활성화

4단계: APPID 및 액세스 키 준비

CAM 콘솔의 API Keys 페이지에서 **APPID**, **SecretId**, **SecretKey**를 획득 및 기록합니다.



5단계: 버킷 생성

버킷 생성의 지침에 따라 COS 버킷을 생성합니다.
일부 애플리케이션에는 버킷 생성 프로세스가 내장되어 있습니다. 이러한 애플리케이션이 버킷을 생성하도록 하려면 이 단계를 생략할 수 있습니다.

애플리케이션에서 COS 서비스 설정

기본 설정

대부분의 애플리케이션에는 스토리지 서비스를 사용하기 위한 유사한 구성 항목이 있습니다. 이러한 구성 항목의 일반 이름과 설명은 다음과 같습니다.

설명：

설정 과정에서 궁금한 점이 있으시면 고객센터로 문의하시기 바랍니다. 본 문서의 가이드를 따랐음을 표시하고 애플리케이션 이름 및 스크린샷과 같은 정보를 제공하십시오.

설정 항목에서 흔히 볼 수 있는 이름	관련 설명
공급자/서비스 공급자/스토리지 서비스 공급자/Service Provider/Storage Provider/Provider 등	애플리케이션이 사용해야 할 스토리지를 선택할 때 다음과 같은 상황이 있을 수 있습니다. 선택 항목에 S3 Compatible Storage/S3 Compatible과 같은 텍스트가 있는 경우 해당 옵션이 먼저 사용됩니다. 선택 항목에 amazon web services/AWS/Amazon S3와 같은 텍스트만 있는 경우 이를 사용하되 구성하는 동안 추가 지침에 주의하십시오. 비슷한 선택 항목이 없지만 애플리케이션 설명에 애플리케이션이 S3 서비스 또는 S3 호환 서비스를 지원한다고 언급하는 경우 아래 구성을 계속할 수 있지만 추가 지침에도 주의해야 합니다. 기타 다른 경우에는 애플리케이션이 COS를 사용하지 못할 수 있습니다.
서버/서버 주소/서비스 URL/Endpoint/Custom Endpoint/Server URL 등	S3 호환 서비스의 서비스 주소 입력란입니다. COS 서비스 사용 시 여기에 COS 서비스 주소를 입력합니다(포맷:

	cos.<Region>.myqcloud.com 또는 https://cos.<Region>.myqcloud.com). https:// 입력 여부는 애플리케이션마다 조금씩 다르므로 직접 시도해 보시기 바랍니다. 이 중 <Region> 는 COS의 사용 가능 리전을 의미합니다. 귀하는 애플리케이션에서 서비스 주소가 지정한 리전에서만 버킷을 생성 또는 선택할 수 있습니다. 귀하의 버킷이 광저우 리전에 있는 경우 서비스 주소는 cos.ap-guangzhou.myqcloud.com으로 설정되어 있어야 합니다. 기타 리전으로 설정한 경우 애플리케이션에서 광저우 리전의 버킷을 찾을 수 없습니다. 애플리케이션의 서비스 공급자 중 Amazon S3만 선택할 수 있고 서버 포트 설정이 가능한 경우 서버 포트를 위의 cos.<Region>.myqcloud.com 또는 https://cos.<Region>.myqcloud.com으로 수정합니다. 서버 포트 설정이 불가능하거나 서버 포트 설정 항목이 없는 경우 귀하의 애플리케이션은 COS 서비스를 사용할 수 없습니다.
Access Key/Access Key ID 등	4단계에서 기록된 SecretId를 입력하십시오.
Secret Key/Secret/Secret Access Key 등	4단계에서 기록된 SecretKey를 입력하십시오.
리전/Region 등	기본값, 자동, Auto 또는 Automatic을 선택합니다.
버킷/Bucket 등	기존의 버킷 이름을 선택 또는 입력합니다. 포맷은 <BucketName-APPID>입니다(예: examplebucket-1250000000). 이 중 BucketName은 5단계에서 버킷 생성 시 입력한 버킷 이름이며, APPID는 4단계에서 기록한 APPID입니다.위의 설명과 마찬가지로 이곳의 버킷은 서버 주소가 지정한 리전에 한정되어 있으며, 기타 리전의 버킷은 나열되지 않거나 정상적인 사용이 불가능합니다. 새로운 버킷을 생성해야 하는 경우 새로 생성된 버킷 이름도 위의 <BucketName-APPID> 포맷에 부합해야 합니다. 그렇지 않으면 정상적으로 버킷을 생성할 수 없습니다.

기타 항목 및 고급 설정 설명

일부 애플리케이션은 위의 기본 설정 외에도 기타 항목 및 고급 설정이 더 있습니다. 애플리케이션에서 COS 서비스를 원활하게 사용할 수 있도록 아래 일부 COS의 기능 설명을 추가합니다.

서버 포트와 프로토콜

COS 서비스가 지원하는 HTTP 프로토콜과 HTTPS 프로토콜은 모두 프로토콜의 기본 포트인 80과 443 포트를 사용합니다. 보안상의 이유로 HTTPS 프로토콜을 통한 COS 서비스 사용을 권장합니다.

Path-Style과 Virtual Hosted-Style

COS는 두 종류의 스타일을 모두 지원합니다.

AWS V2 서명과 AWS V4 서명

COS는 두 종류의 서명 포맷을 모두 지원합니다.

결론

COS는 S3와의 완전한 호환성을 보장하지 않습니다. 애플리케이션에서 COS 사용 시 질문이 있는 경우 고객센터에 도움을 요청하십시오. 본 문서의 가이드를 따랐음을 표시하고 애플리케이션 이름 및 스크린샷과 같은 정보를 제공하십시오.

COS에 WordPress 원격 첨부 파일 저장하기

최종 업데이트 날짜: : 2024-06-24 16:53:18

소개

WordPress는 PHP 언어를 사용해 개발된 블로그 플랫폼입니다. 사용자는 PHP와 MySQL 데이터베이스를 지원하는 서버에 개인 웹 사이트를 구축할 수 있고, WordPress를 콘텐츠 관리 시스템(CMS)으로 사용할 수도 있습니다.
WordPress의 강력한 기능과 확장성은 플러그 인이 많아 기능 확충이 쉽기 때문입니다. 완전한 하나의 웹 사이트가 기본적으로 갖추어야 할 모든 기능은 3rd party 플러그 인을 통해 실행할 수 있습니다.
본 문서에서는 플러그 인을 사용한 원격 파일 첨부 기능을 통해 WordPress의 미디어 라이브러리 첨부 파일을 Tencent Cloud의 Cloud Object Storage(COS)에 저장하는 방법을 소개합니다.
COS는 고확장성, 저비용, 신뢰성, 보안과 같은 특징이 있습니다. 미디어 라이브러리 첨부 파일을 COS에 저장할 경우 장점은 다음과 같습니다.
첨부 파일의 신뢰성이 높아집니다.
서버에 첨부 파일을 위한 별도의 스토리지 용량을 마련하지 않아도 됩니다.
이미지 첨부 파일을 조회할 때 바로 COS 서버에 연결하면 서버의 다운스트림 대역폭/트래픽을 점유하지 않으며, 액세스 속도가 더 빨라집니다.
Tencent Cloud Content Delivery Network(CDN)와 함께 사용하면 사용자의 이미지 첨부 파일 확인 속도를 더욱 향상시키고 웹사이트 액세스 속도를 최적화할 수 있습니다.

전제 조건

1. 버킷을 생성 완료해야 하며, 그렇지 않은 경우 버킷 생성의 지침에 따라 버킷을 생성합니다.
2. Cloud Virtual Machine에 설명된 대로 CVM(Cloud Virtual Machine) 인스턴스와 같은 서버를 생성했습니다.

실행 순서

Wordpress 웹사이트 배포

CVM에서 WordPress 웹사이트를 빠르게 구축하려면 이미지를 통해 또는 수동으로 배포할 수 있습니다. 비즈니스 웹사이트의 확장성에 대한 요구 사항이 높은 경우 다음 문서의 지침에 따라 수동으로 배포할 수 있습니다.
WordPress 웹사이트 구축(Linux)
WordPress 웹사이트 구축(Windows)
다음과 같이 쉽게 이미지를 통해 WordPress 웹 사이트를 배포할 수 있습니다.
1. 이미지를 통해 WordPress 웹사이트를 배포합니다.

1.1 CVM 콘솔에 로그인 한 후 인스턴스 관리 페이지에서 **생성**을 클릭합니다.
1.2 프롬프트에 따라 모델을 선택합니다. **인스턴스 구성 > 이미지**에서 **이미지 마켓플레이스**를 클릭하고 **이미지 마켓플레이스에서 선택**을 선택합니다.
1.3 '이미지 마켓플레이스' 팝업 창에서 **기본 소프트웨어**를 선택하고 검색을 위해 **wordpress**를 입력합니다.
1.4 필요에 따라 이미지를 선택합니다. 여기서는 **WordPress 블로그 프로그램 _v5.5.3(CentOS | LAMP)**을 선택합니다. 그 다음 **무료 평가판**을 클릭합니다.
1.5 구매 후 CVM 콘솔에 로그인하여 새로 생성된 인스턴스를 보안 그룹과 연결하고 인바운드 규칙에서 포트 80을 엽니다.
2. CVM 인스턴스 관리 페이지에서 인스턴스의 **공중망 IP**를 복사하고 로컬 브라우저에서 `http://공중망 IP/wp-admin` 에 액세스하여 WordPress 웹 사이트 설치를 시작합니다.
2.1 WordPress 언어를 선택하고 **Continue**를 클릭합니다.
2.2 WordPress 웹 사이트 제목과 관리자 사용자 이름, 비밀번호 및 이메일 주소를 입력합니다.
2.3 **WordPress 설치**를 클릭합니다.
2.4 **로그인**을 클릭합니다.
3. WordPress를 v6.0.2로 업그레이드합니다.
왼쪽 사이드바에서 대시보드 > '업데이트'를 클릭하여 WordPress를 v6.0.2로 업데이트합니다.

COS 버킷 생성

1. **공개 읽기 및 개인 쓰기** 버킷을 생성합니다. 버킷 리전은 WordPress 블로그 플랫폼을 실행하는 CVM과 동일한 리전을 권장합니다. 자세한 생성 방법은 버킷 생성 문서를 참고하십시오.
2. **버킷 리스트**에서 방금 생성한 버킷을 찾아 이름을 클릭한 후 버킷 페이지로 이동합니다.
3. 왼쪽 메뉴에서 **개요**를 클릭하여 액세스 도메인을 조회하고 기록합니다.

플러그인 설치 및 구성

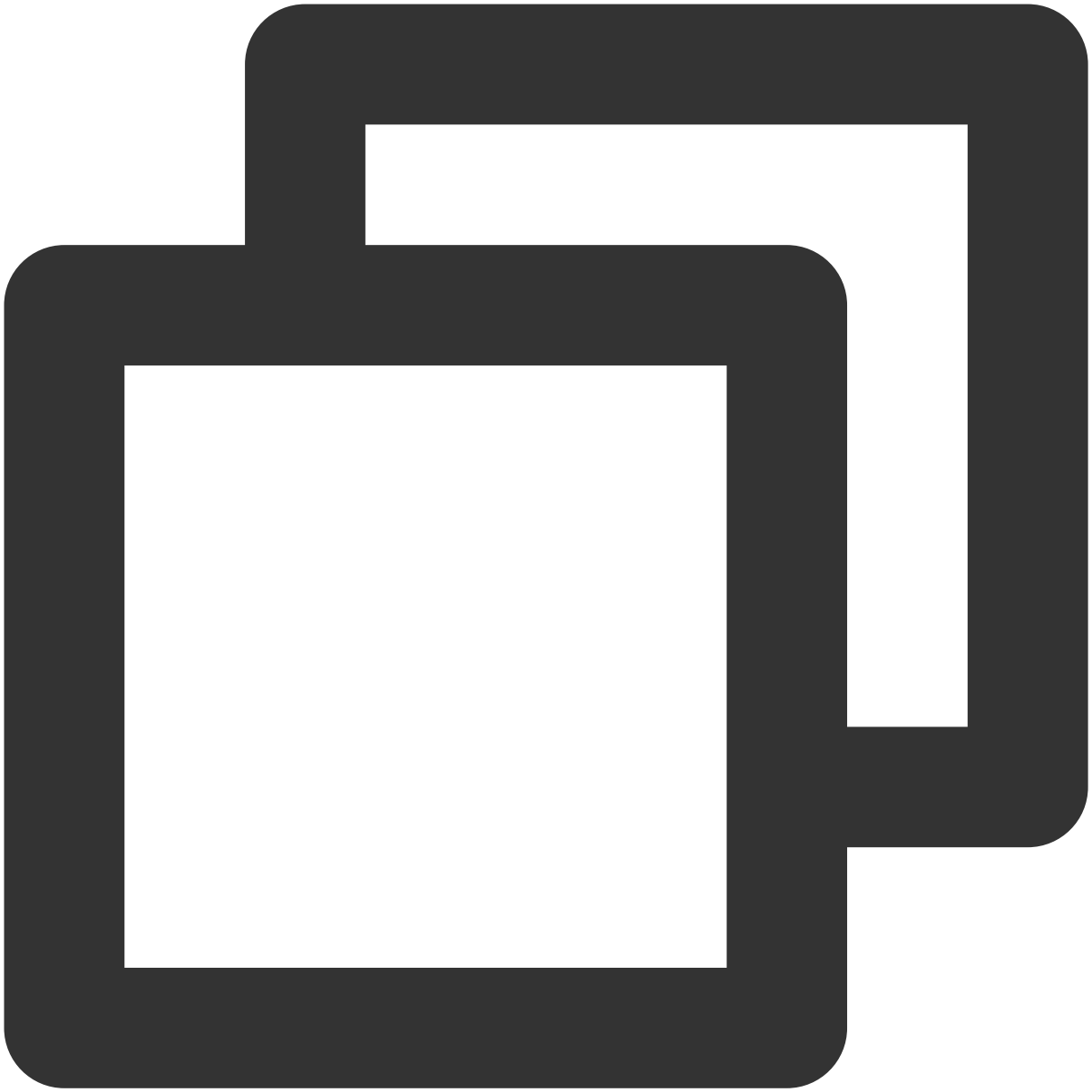
플러그인 라이브러리 또는 소스 코드를 통해 플러그인을 설치할 수 있습니다.

플러그인 라이브러리에 설치(권장)

WordPress 백엔드에서 **플러그인**을 클릭하고 **tencentcloud-cos** 플러그인을 직접 검색한 다음 **지금 설치**를 클릭합니다.

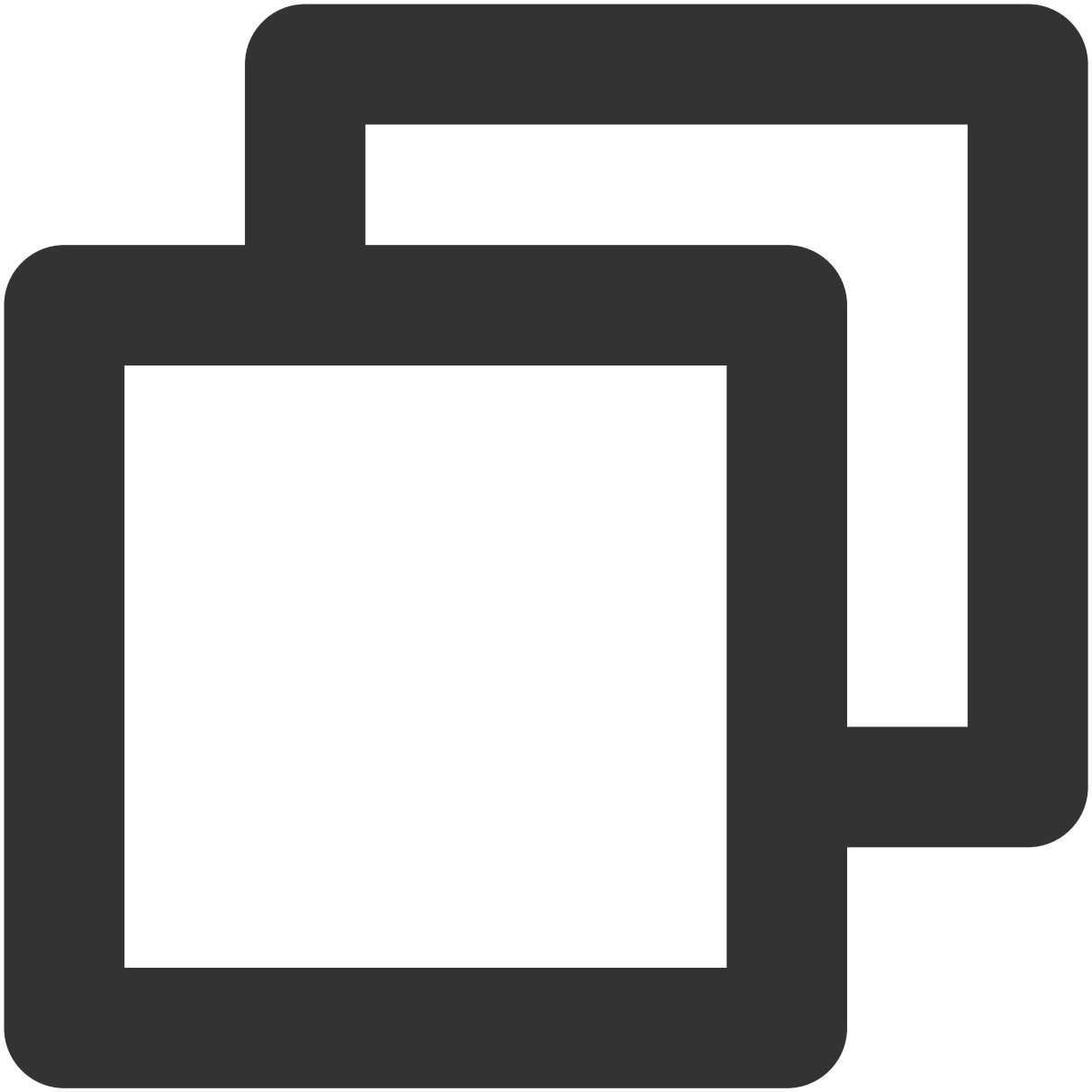
소스 코드를 통한 설치

플러그인 소스 코드를 다운로드하고 WordPress 플러그인 디렉터리 `wp-content/plugins` 에 업로드하고 백엔드에서 실행합니다.
다음 단계에서는 Ubuntu를 예로 들어 플러그인을 설치하는 방법을 설명합니다.
1. wp-content의 상위 디렉터리로 이동:



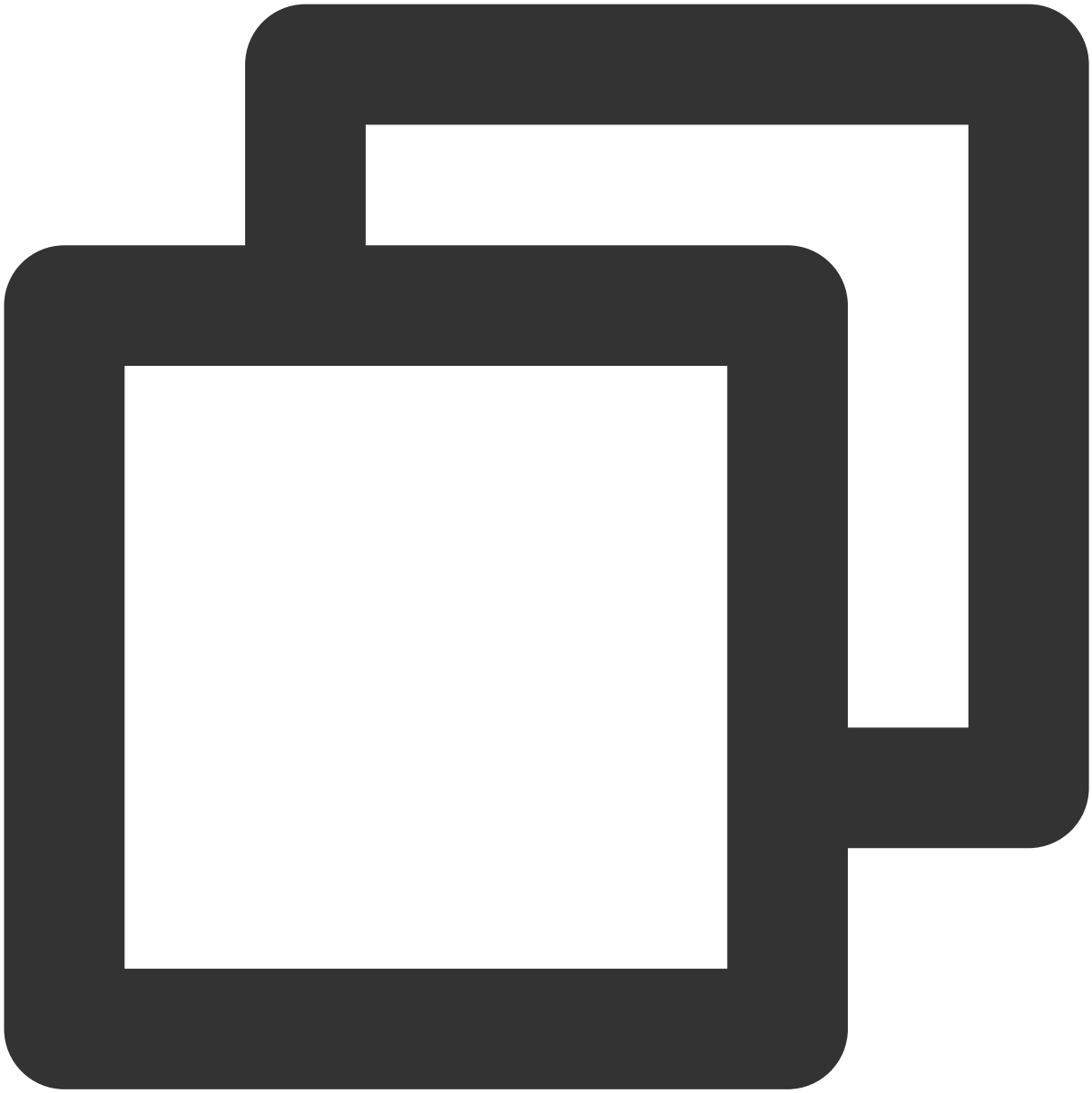
```
cd /var/www/html
```

2. 권한 추가:



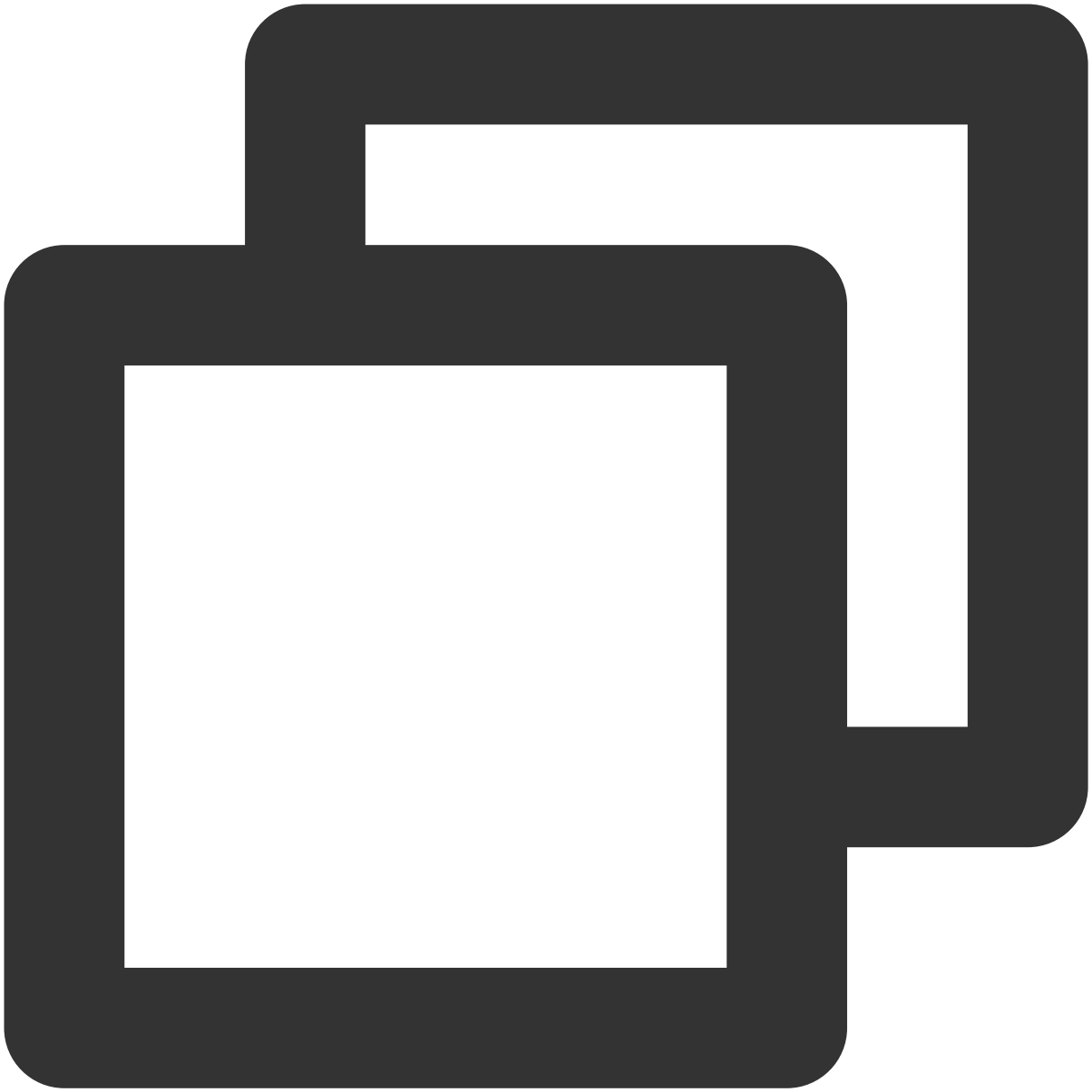
```
chmod -R 777 wp-content
```

3. 플러그인 디렉터리 생성:



```
cd wp-content/plugins/
mkdir tencent-cloud-cos
cd tencent-cloud-cos
```

4. 플러그인 디렉터리에 플러그인 다운로드:



```
wget https://cos5.cloud.tencent.com/cosbrowser/code/tencent-cloud-cos.zip
unzip tencent-cloud-cos.zip
rm tencent-cloud-cos.zip -f
```

5. 왼쪽 사이드바에서 '플러그인'을 클릭하면 플러그인을 볼 수 있습니다. 활성화를 클릭하여 활성화합니다.

플러그인 구성

tencent-cloud-cos 플러그인에서 COS 버킷 정보를 구성합니다.

1. '설정'을 클릭하여 tencent-cloud-cos 플러그인을 구성합니다.

2. 다음과 같이 COS 정보를 구성합니다.

구성 항목	구성 값
SecretId, SecretKey	키 정보에 액세스하려면 Tencent Cloud API 키로 이동하여 생성 및 가져오기
소속 리전	버킷 생성 시 선택한 리전
스페이스 이름	버킷 생성 시 사용자 지정된 버킷 이름, 예시: examplebucket-1250000000
액세스 도메인	COS의 기본 버킷 도메인을 말하며, 사용자가 버킷을 생성하면 시스템에서 버킷 이름과 리전에 따라 자동으로 생성합니다. 다른 리전의 버킷은 각각 다른 기본 도메인이 있습니다. COS 콘솔로 이동하여 버킷의 개요 > 도메인 정보에서 확인하실 수 있습니다.
자동 이름 변경	파일을 COS에 업로드한 후 동일한 이름의 기존 파일과 충돌을 피하기 위해 자동으로 지정된 형식에 따라 이름 변경
로컬에 저장하지 않음	이 옵션을 활성화 후에는 원본 파일이 로컬에 보관되지 않습니다
원격 파일 보관	이 옵션을 활성화한 후 파일이 삭제되면 로컬 파일 사본만 삭제되고 원격 COS 버킷의 파일 사본은 복구하려는 경우에 대비하여 계속 보관됩니다
썸네일 금지	이 옵션을 활성화하면 썸네일 파일이 업로드되지 않습니다
Cloud Infinite(CI)	CI 서비스가 활성화되면 이미지에 편집, 압축, 형식 변환, 워터마크 추가 등 이미지에 대한 다양한 작업을 수행할 수 있으며, 자세한 내용은 Cloud Infinite를 참고하십시오
파일 조정	파일 조정이 활성화되면 이미지, 비디오, 오디오, 텍스트, 파일 및 웹 페이지의 멀티미디어 콘텐츠를 지능적으로 조정합니다. 음란물, 저속, 불법, 혐오, 공격적인 정보와 같은 비준수 콘텐츠를 효과적으로 식별하여 운영상의 위험을 방지할 수 있습니다. 자세한 내용은 민감한 콘텐츠 조정 개요를 참고하십시오.
파일 미리보기	파일 미리보기가 활성화되면 파일을 이미지, PDF 파일 또는 HTML5 페이지로 변환하여 웹 페이지에 파일 콘텐츠를 표시합니다. 자세한 내용은 파일 미리보기 개요를 참고하십시오.
디버깅	이 기능은 오류, 예외 및 경고를 기록합니다

3. 구성이 완료되면 **구성 저장**을 클릭합니다.

COS에 대한 WordPress 첨부 파일 스토리지 확인

Wordpress에서 이미지로 게시물을 작성하고 이미지가 COS에 저장되어 있는지 확인할 수 있습니다.

1. WordPress 대시보드에서 왼쪽 사이드바의 '게시물'을 클릭하여 이미지가 포함된 게시물을 생성합니다.

2. 'Hello world!' 편집 기본적으로 WordPress에서 생성된 게시물입니다.
3. 오른쪽에서 '+'를 클릭합니다.
4. 이미지를 업로드합니다.
5. 그 다음 업로드된 이미지의 URL이 `https://<BucketName-APPID>.cos.<Region>.myqcloud.com/<ObjectKey>` 형식의 `https://wd-125000000.cos.ap-nanjing.myqcloud.com/2022/10/입하-1200x675.jpeg` 와 같은 COS URL인지 확인하고 그렇다면 이미지가 COS 버킷에 업로드된 것입니다.
6. COS 콘솔에 로그인하면 버킷에서 새로 업로드된 이미지를 볼 수 있습니다.

설명：

상기 테스트에 성공하면 COSCMD 또는 COS Migration을 사용하여 이전 WordPress 리소스를 COS 버킷에 동기화합니다. **이 단계를 수행해야 합니다. 그렇지 않으면 COS에서 이러한 리소스를 볼 수 없습니다.** 그 다음 아래의 Origin-pull 설정에 따라 Origin-pull을 선택적으로 활성화할 수 있습니다.

확장

1. CDN 가속을 사용하여 액세스합니다.

버킷에 CDN 가속 설정이 필요한 경우 CDN 가속 구성 문서를 참고하십시오. 플러그 인 설정에서 URL 접두사의 기본값을 CDN 가속 도메인 또는 사용자 정의 가속 도메인으로 수정하면 됩니다.

2. 데이터베이스의 리소스 주소를 변경합니다.

새로 생성한 사이트가 아니라면 데이터베이스에는 반드시 기존 리소스 링크 주소가 존재하며, 이 리소스 주소를 변경해야 합니다. 플러그 인에서 변경 기능을 제공하며, 최초 변경 전 반드시 백업해 두시기 바랍니다.

기존 도메인에 원본 리소스 도메인을 입력합니다. 예: `https://example.com/`

신규 도메인에 현재의 리소스 도메인을 입력합니다. 예: `https://img.example.com/`

3. 크로스 도메인 액세스를 설정합니다.

본 문서의 상응하는 리소스 링크 인용 시 콘솔은 크로스 도메인 오류 `No 'Access-Control-Allow-Origin' header is present on the requested resource` 를 표시합니다. 이는 header를 추가하지 않았기 때문입니다. 크로스 도메인 액세스 CORS 설정에서 HTTP Header 설정을 추가해야 하며, 다음 두 가지 방법으로 구성할 수 있습니다.

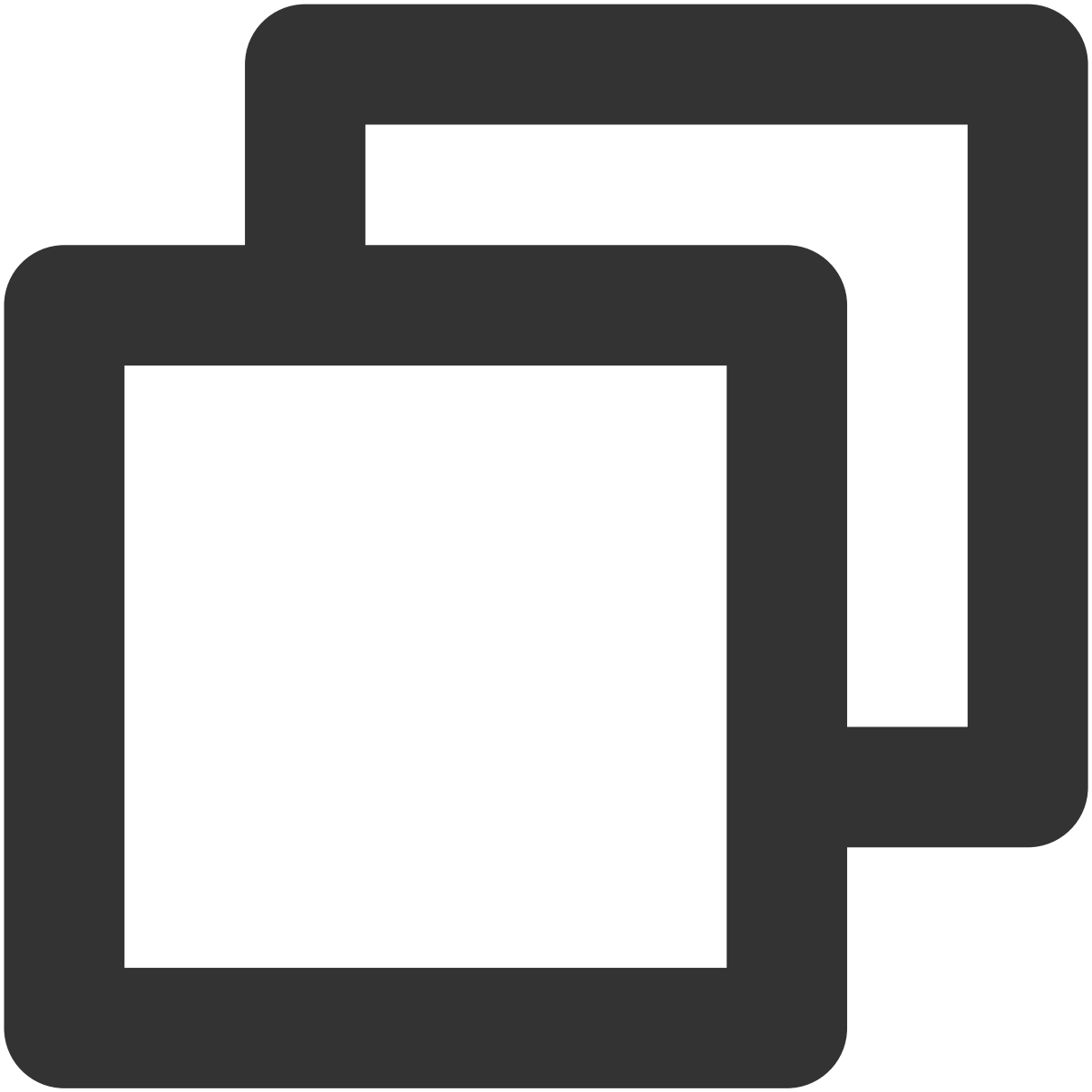
COS 콘솔에서 구성

설명：

크로스 도메인 설정 작업 순서는 CORS 설정 문서를 참고하십시오.

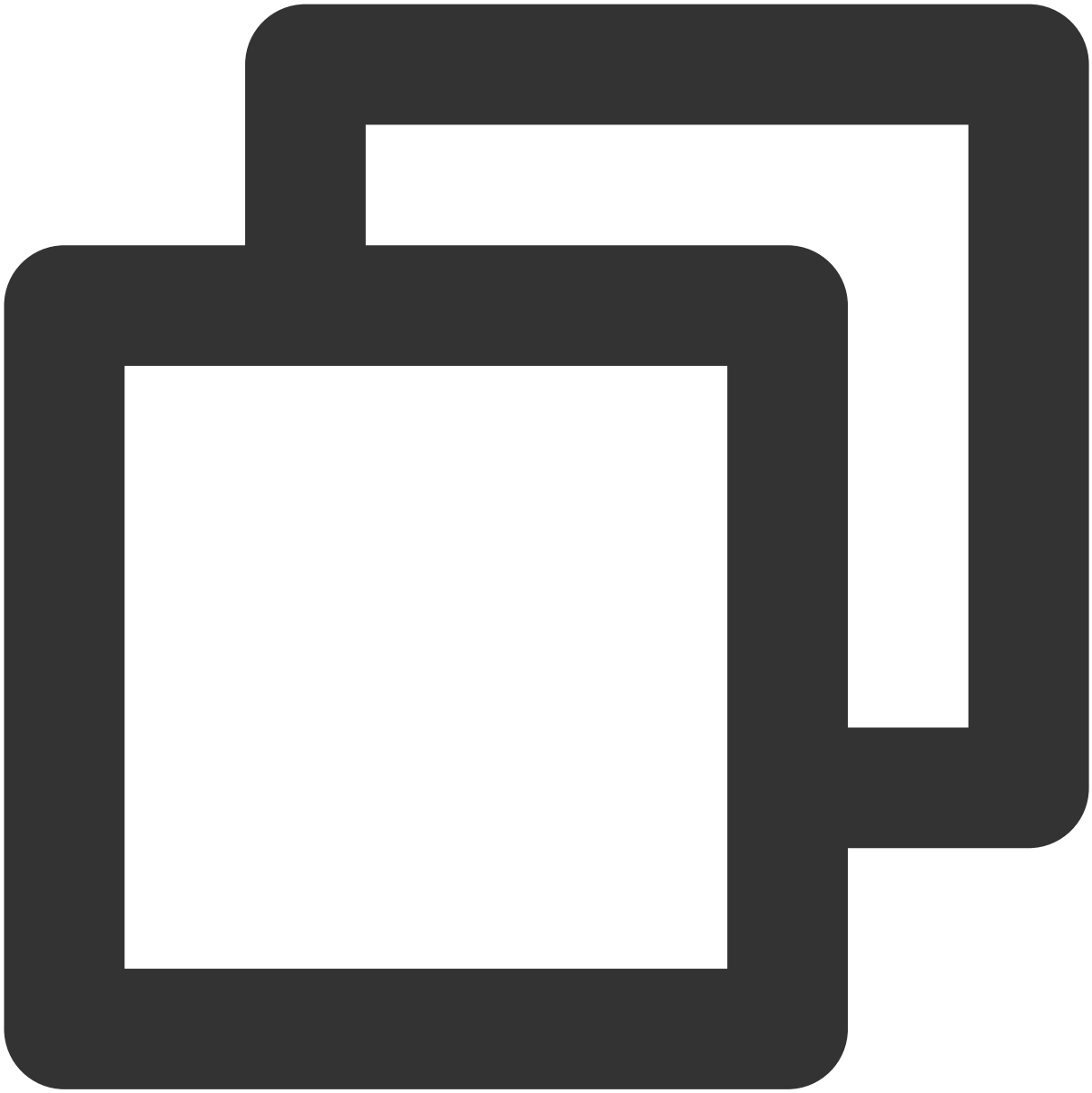
CDN 콘솔에서 구성

모든 도메인을 허용할 경우, 다음과 같이 구성합니다.



```
Access-Control-Allow-Origin: *
```

사용자 개인 도메인 액세스만 허용할 경우, 다음과 같이 구성합니다.



```
Access-Control-Allow-Origin: https://example.com
```

4. Origin-pull을 설정합니다.

COS 콘솔을 통해 WordPress 미디어 라이브러리에 리소스를 업로드하지 않는 경우 COS Origin-pull을 사용하는 것이 좋습니다. 자세한 내용은 Origin-pull 설정을 참고하십시오.

Origin-pull이 활성화되면 클라이언트가 처음으로 COS의 원본 객체에 액세스할 때 COS가 객체에 도달할 수 없는 경우 302 HTTP 상태 코드를 반환하고 자동으로 요청을 Origin-pull 주소로 리디렉션합니다. 그 다음 원본에서 액세스할 객체를 제공합니다. 한편 COS는 원본에서 이 객체를 복사하여 해당 COS 디렉터리에 저장하므로 COS는 후속 요청에서 객체를 클라이언트에 직접 반환할 수 있습니다.

COS에 Ghost 첨부 파일 저장하기

최종 업데이트 날짜: : 2024-06-24 16:53:18

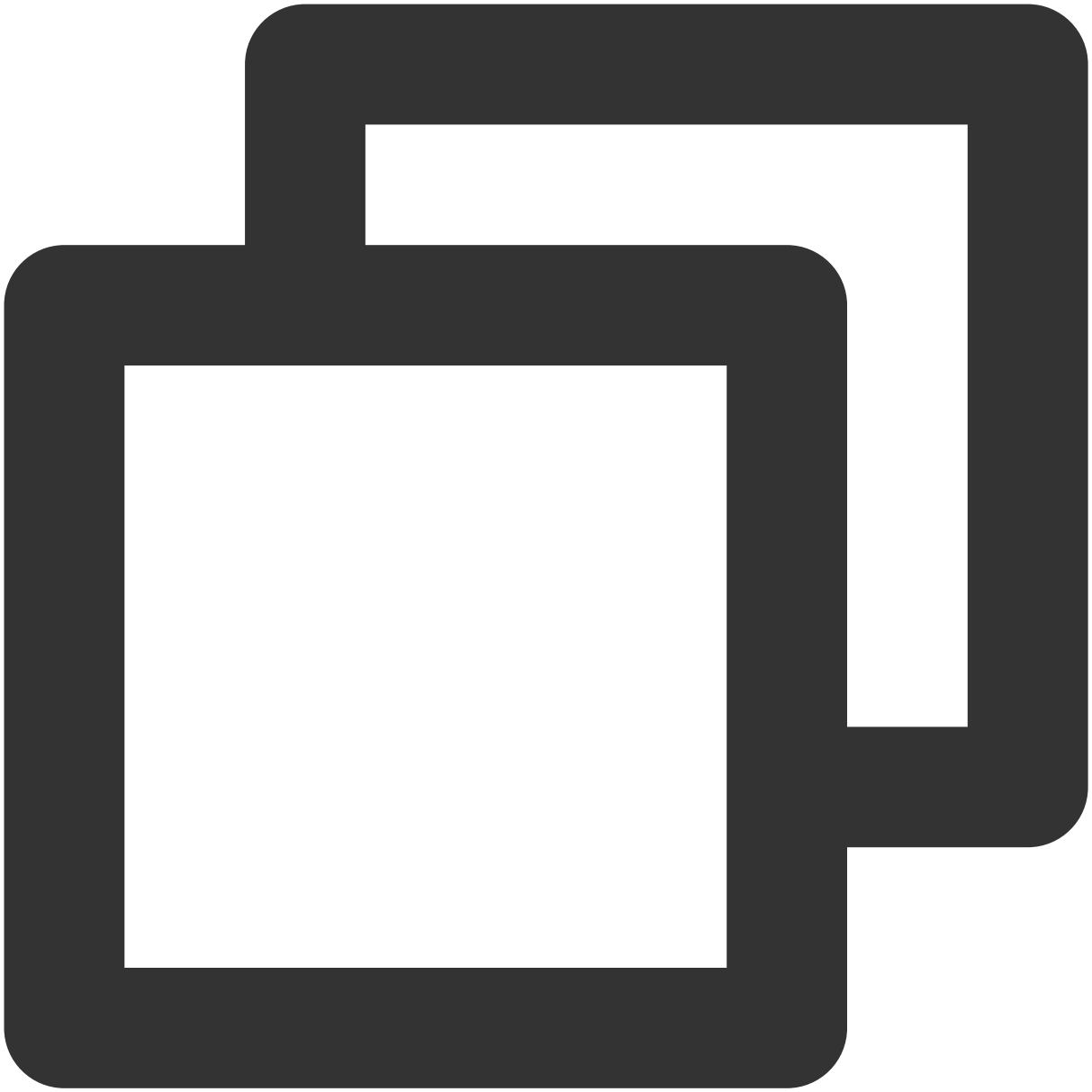
소개

Ghost는 블로그 웹 사이트를 빠르게 구축하기 위한 Node.js 기반 프레임워크입니다. 공식 cli 툴을 사용하여 개인 웹사이트를 빠르게 생성하고 CVM 또는 Docker에 배포할 수 있습니다.
블로그 사이트로서 첨부파일 업로드는 필수 기능입니다. Ghost는 기본적으로 첨부 파일을 로컬에 저장합니다. 이 문서는 플러그인을 통해 COS(Cloud Object Storage)에 첨부 파일을 저장하는 방법을 설명합니다. COS에 포럼 첨부 파일을 저장하면 다음과 같은 이점이 있습니다.
첨부 파일의 신뢰성이 높아집니다.
포럼 첨부 파일을 위해 서버에 추가 스토리지 용량을 준비할 필요가 없습니다.
다운스트림 대역폭을 차지하거나 자신의 서버에서 트래픽을 증가시키는 대신 COS 서버를 통해 이미지 첨부 파일에 더 빠르게 액세스합니다.
Content Delivery Network(CDN)을 통해 이미지 첨부 파일에 대한 포럼 사용자 액세스를 가속화할 수 있습니다.

준비 작업

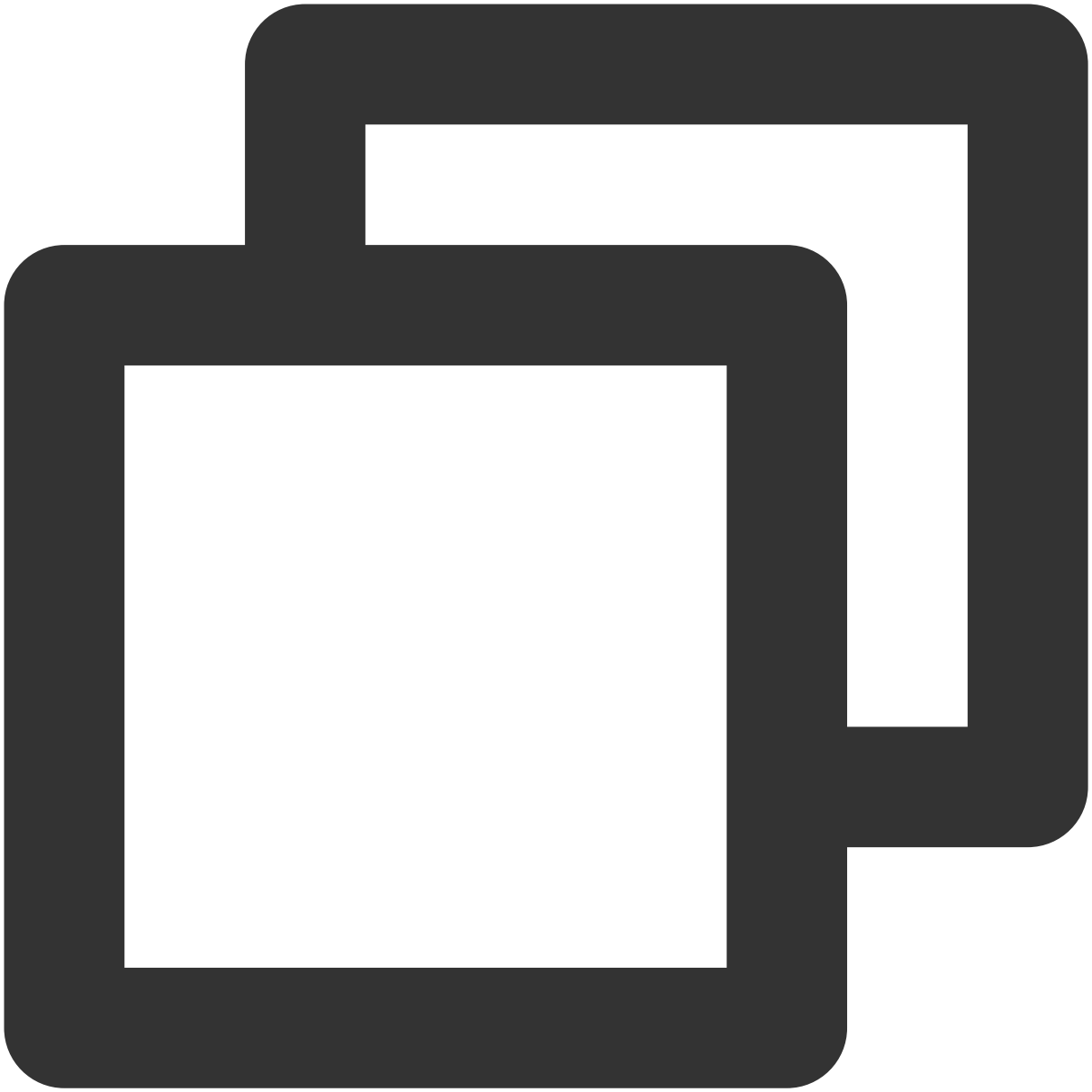
Ghost 웹사이트 구축

1. Node.js 환경을 설치합니다.
2. ghost-cli를 설치합니다.



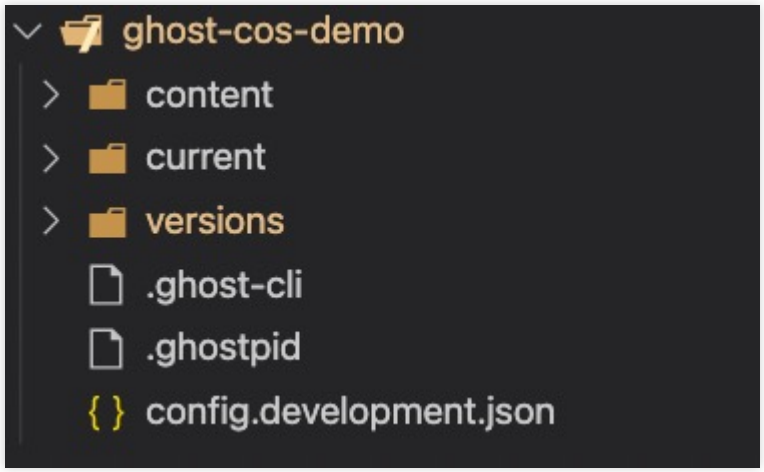
```
npm install ghost-cli@latest -g
```

3. 프로젝트를 생성하고 프로젝트의 루트 디렉터리에서 다음 명령을 실행합니다.

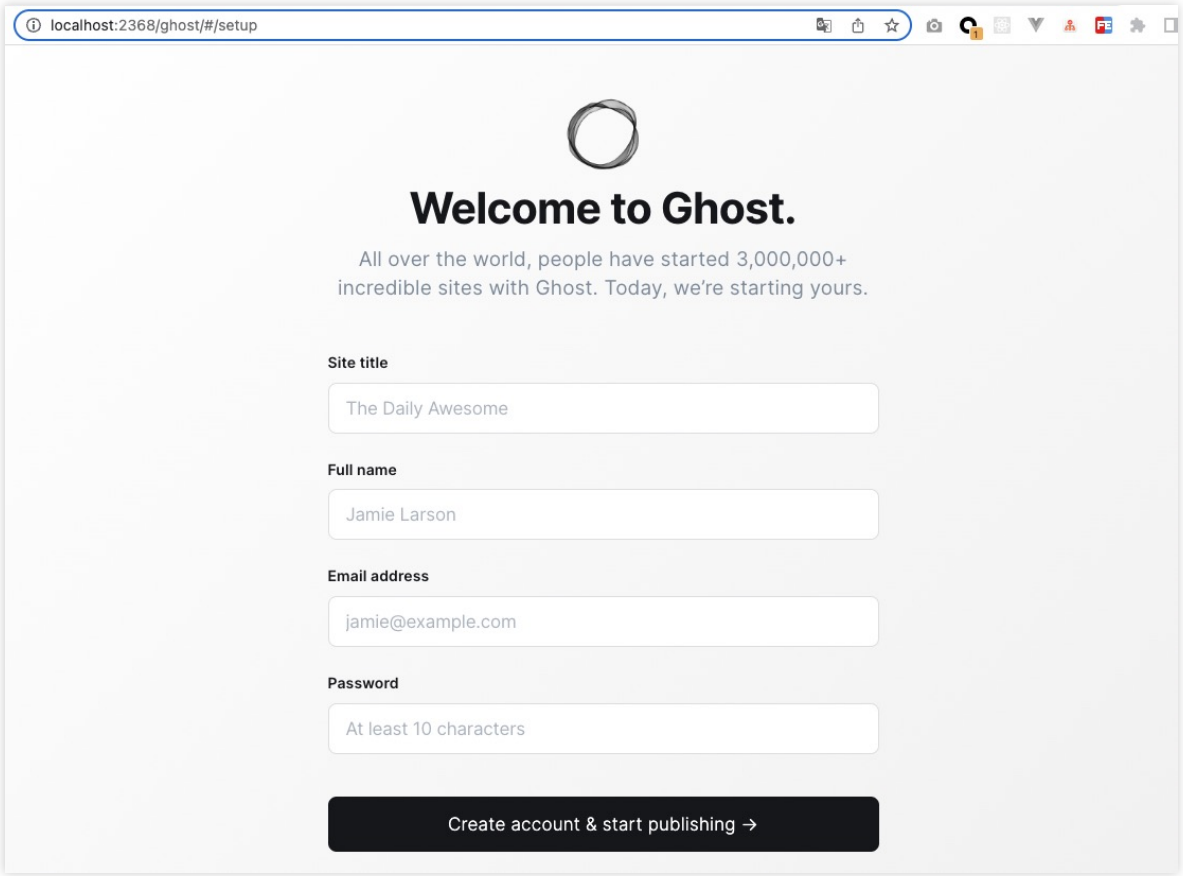


```
ghost install local
```

성공적으로 생성된 후 프로젝트 구조는 다음과 같습니다.



4. 브라우저를 열고 localhost:2368에 액세스합니다. 가입 페이지에서 계정에 가입하고 관리 백엔드로 이동합니다.



COS 버킷 생성

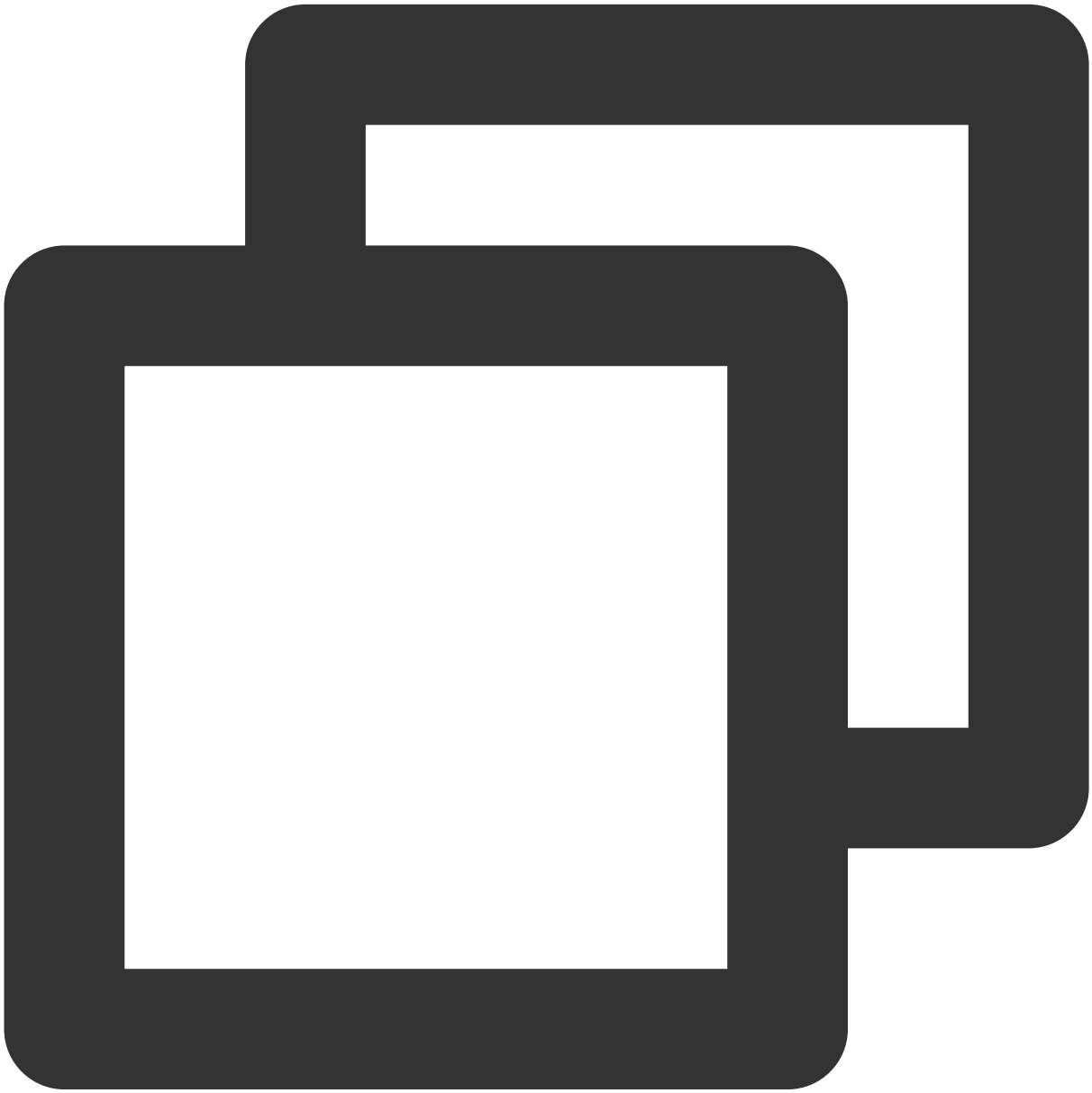
1. COS 콘솔에서 버킷 생성에 설명된 대로 **공개 읽기/비공개 쓰기** 액세스 권한의 버킷을 생성합니다. 자세한 내용은 버킷 생성을 참고하십시오.
2. **보안 관리 > CORS(Cross-Origin Resource Sharing)**를 클릭하고 CORS 설정에 설명된 대로 CORS 구성을 추가합니다. 다음 구성을 사용하여 디버깅을 용이하게 할 수 있습니다.

Ghost를 COS 버킷과 연결

주의 :

리스크를 줄이기 위해 서브 계정 키를 사용하고 최소 권한의 원칙 설명을 따르는 것이 좋습니다. 서브 계정 키를 가져오는 방법에 대한 자세한 내용은 액세스 키를 참고하십시오.

1. Ghost 프로젝트의 루트 디렉터리에 있는 config.development.json 구성 파일에 다음 구성을 추가합니다.



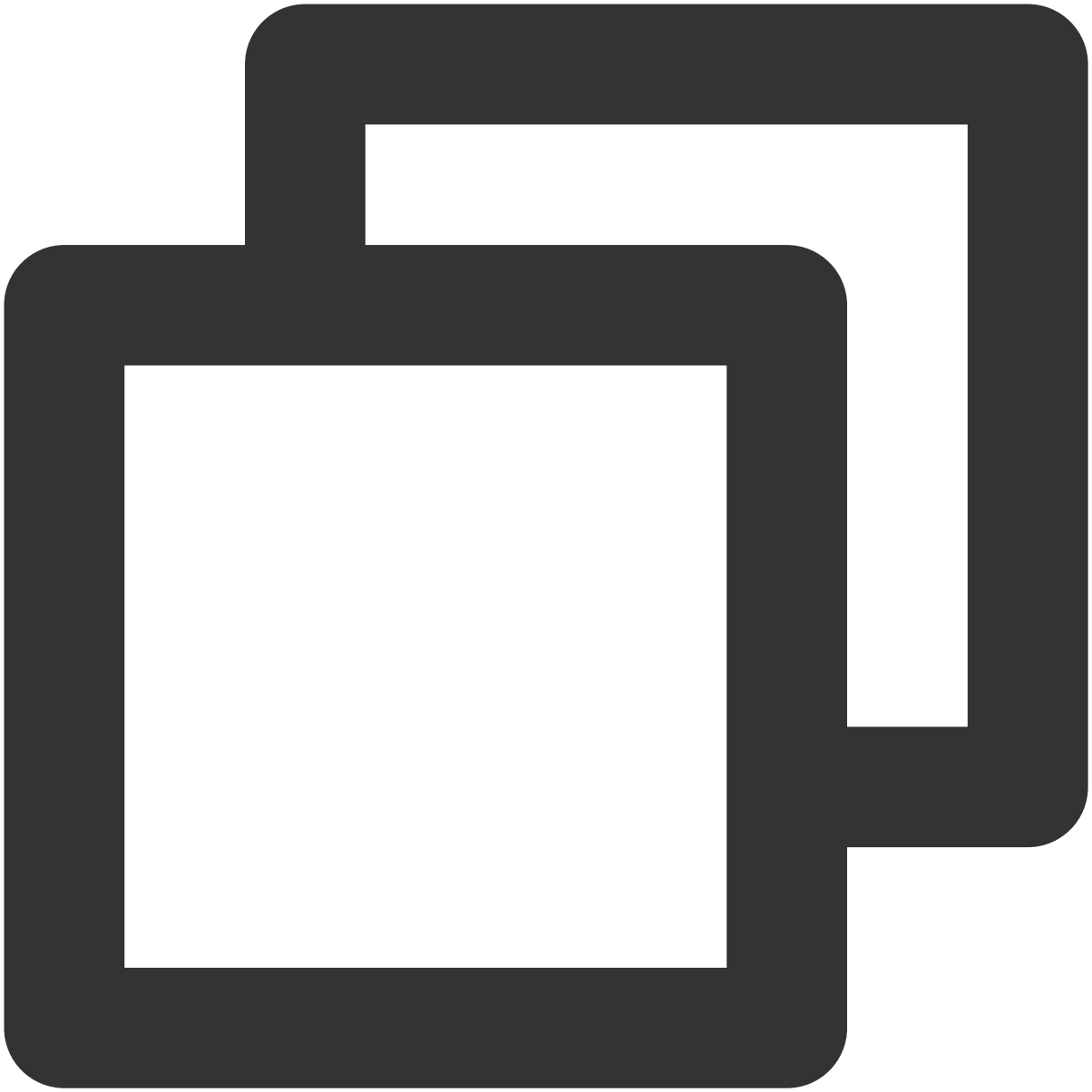
```
 "storage": {
"active": "ghost-cos-store",
"ghost-cos-store": {
```

```
    "BasePath": "ghost/", // 디렉터리 이름으로 변경할 수 있습니다. 비워두면 기본적으로 루트 디
    "SecretId": "AKID*************",
    "SecretKey": "***************",
    "Bucket": "xxx-125********",
    "Region": "**-*******"
  }
   }
```

매개변수 설명은 다음과 같습니다.

설정 항목	설정 항목
BasePath	파일이 저장되는 COS 경로입니다. 필요에 따라 수정할 수 있습니다. 비워두면 기본적으로 루트 디렉터리가 사용됩니다.
SecretId	API Key 관리페이지에서 생성 및 획득할 수 있는 액세스 키 정보입니다.
SecretKey	API Key 관리페이지에서 생성 및 획득할 수 있는 액세스 키 정보입니다.
Bucket	examplebucket-1250000000과 같이 버킷 생성 중에 사용자 정의된 이름입니다.
Region	버킷 생성 중에 선택한 리전입니다.

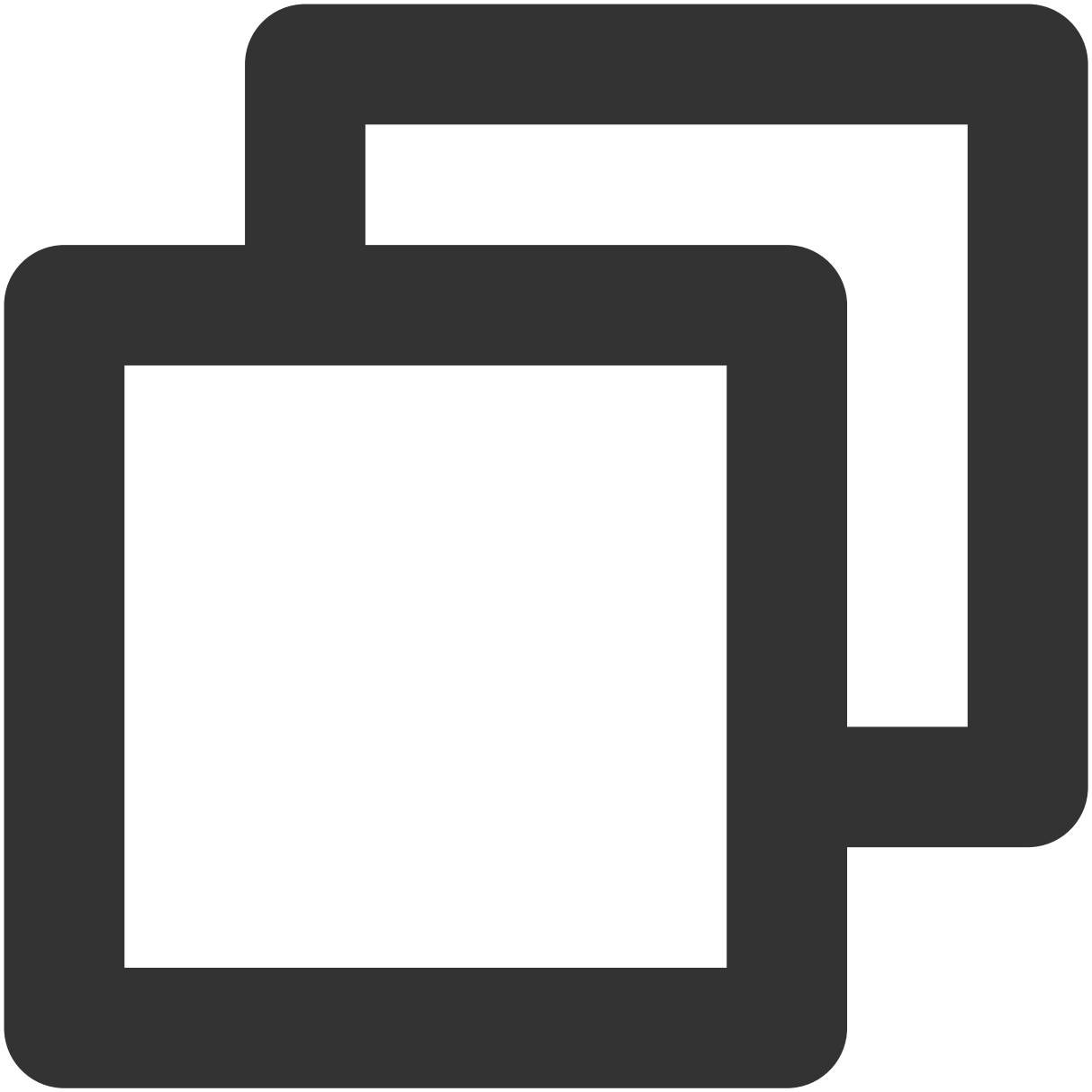
2. 사용자 지정 스토리지 디렉터리를 생성하고 프로젝트의 루트 디렉터리에서 다음 명령을 실행합니다.



```
mkdir -p content/adapters/storage
```

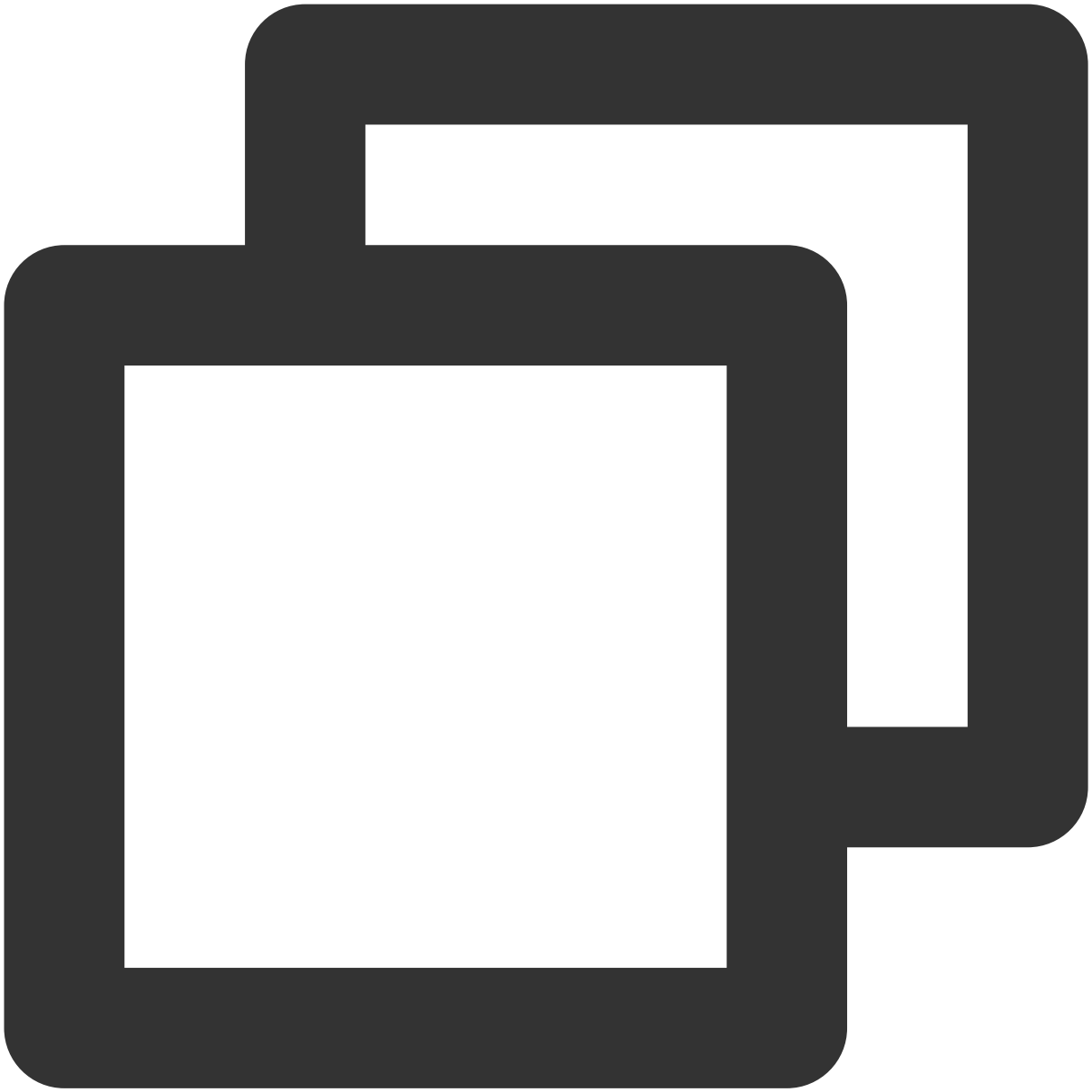
3. Tencent Cloud에서 제공하는 ghost-cos-store 플러그인을 설치합니다.

3.1 npm을 통해 설치합니다.



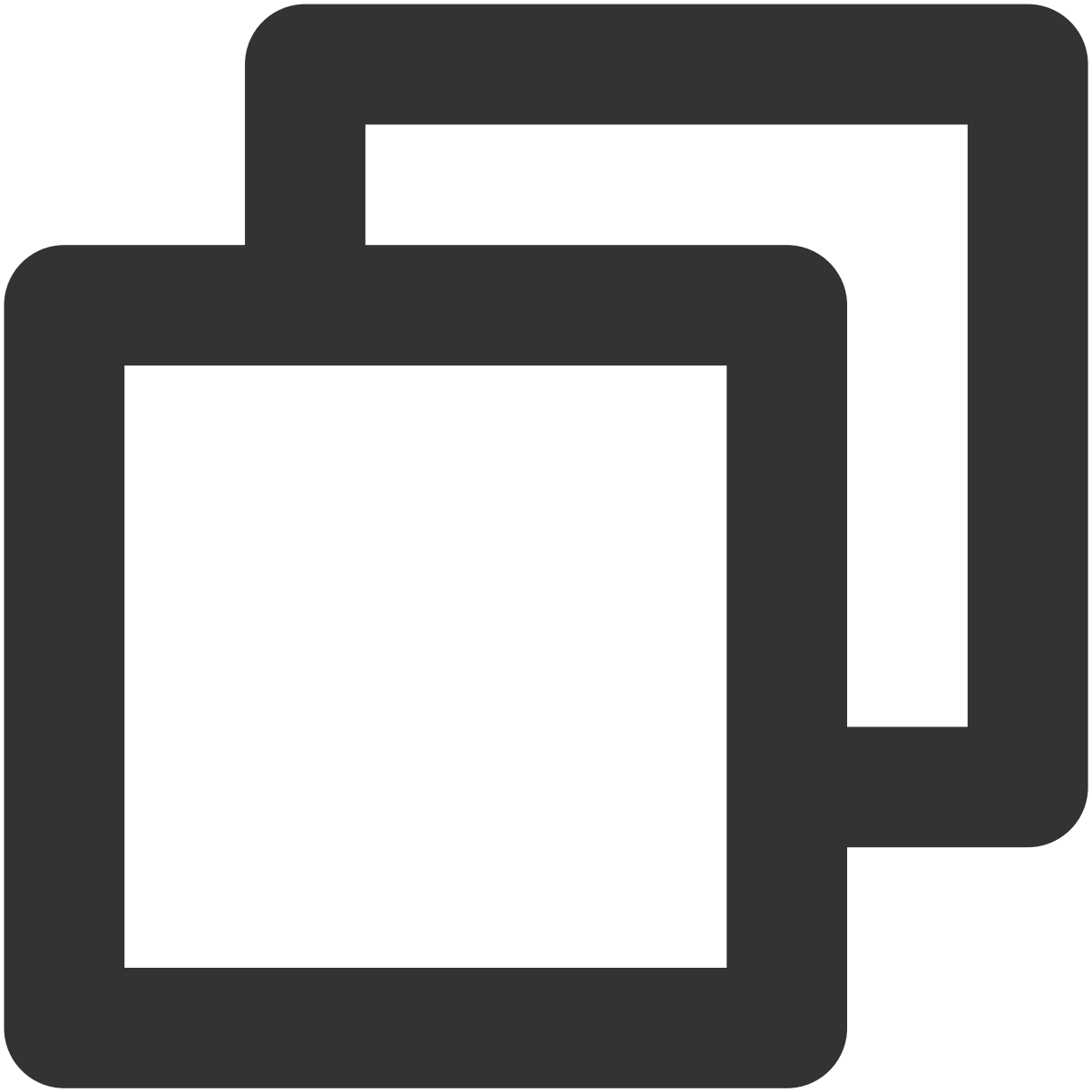
```
npm install ghost-cos-store
```

3.2 storage 디렉터리에 다음 콘텐츠가 포함된 ghost-cos-store.js 파일을 생성합니다.



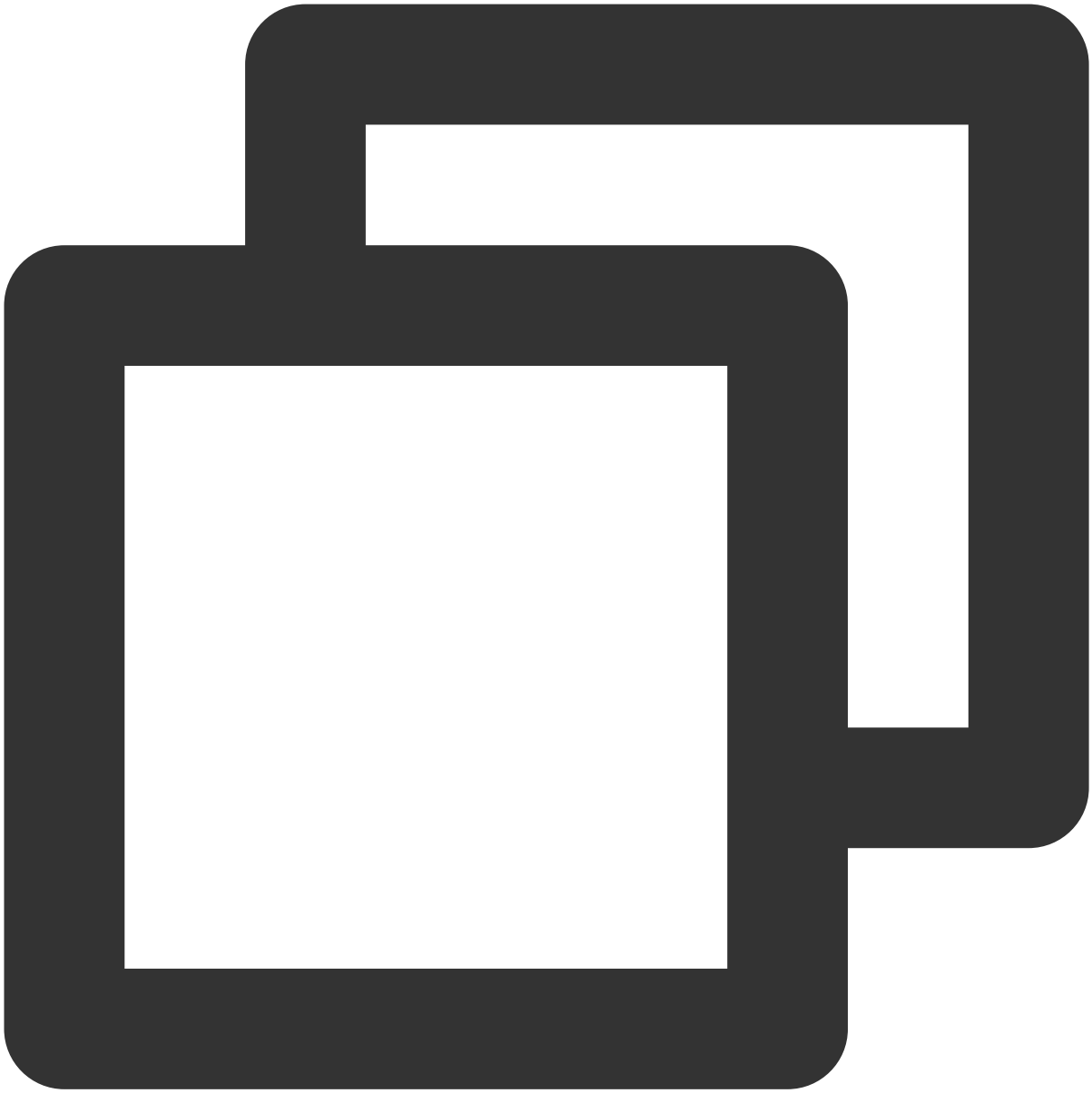
```
//  content/adapters/storage/ghost-cos-store.js
module.exports = require('ghost-cos-store');
```

3.3 git clone을 통해 실행합니다.



```
cd content/adapters/storage
git clone https://github.com/tencentyun/ghost-cos-store.git
cd ghost-cos-store
npm i
```

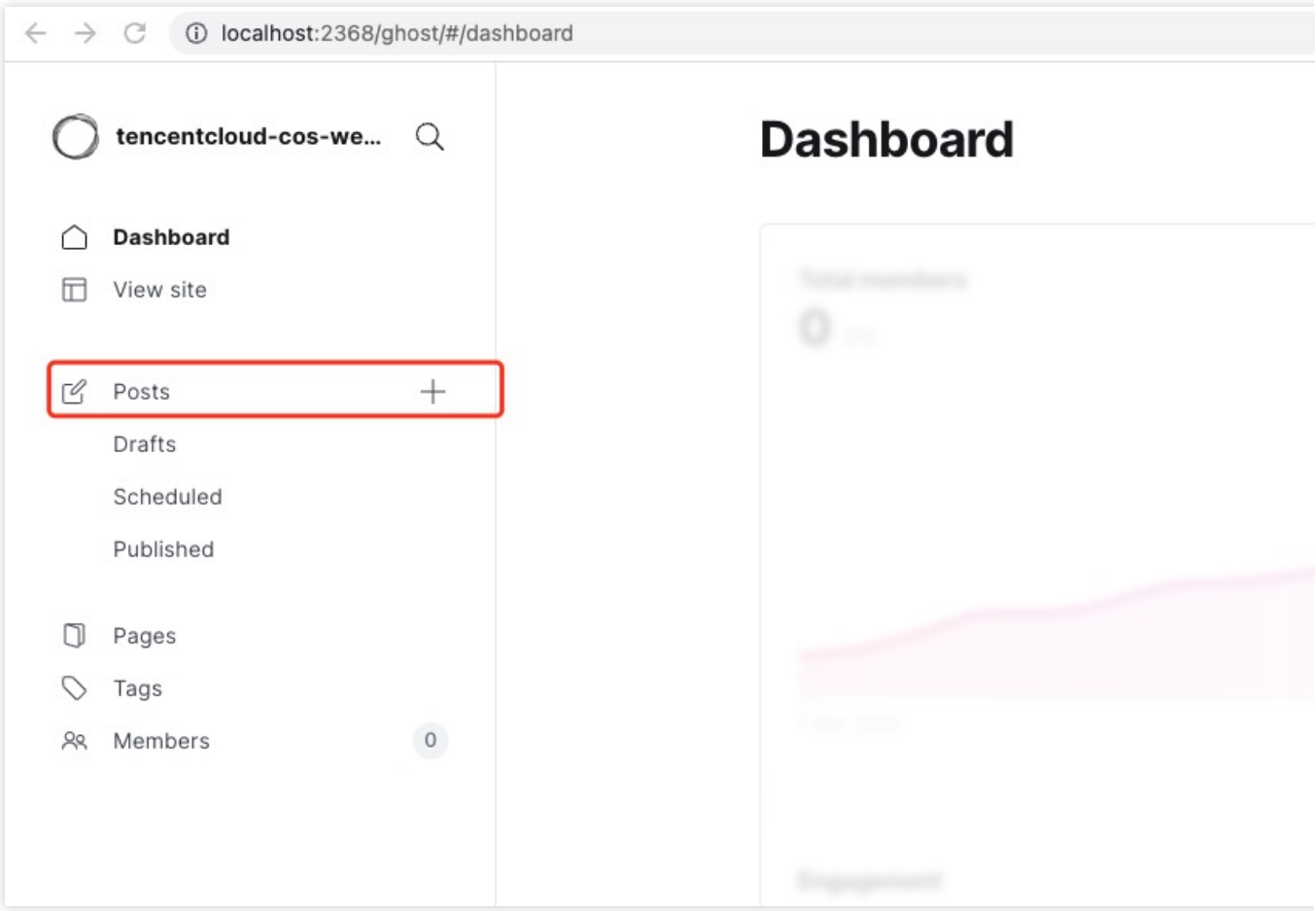
3.4 설치 후 Ghost를 다시 시작합니다.



```
ghost restart
```

게시물 게시 및 업로드 테스트

1. Ghost 콘솔에서 +를 클릭하여 게시물을 게시합니다.



2. +를 클릭하여 이미지를 업로드합니다. 브라우저에서 캡처한 패킷에서 upload 요청이 성공하고 이미지의 COS URL이 반환되는 것을 확인할 수 있습니다.

PC 파일 COS에 백업

최종 업데이트 날짜: : 2024-06-24 16:53:18

서문

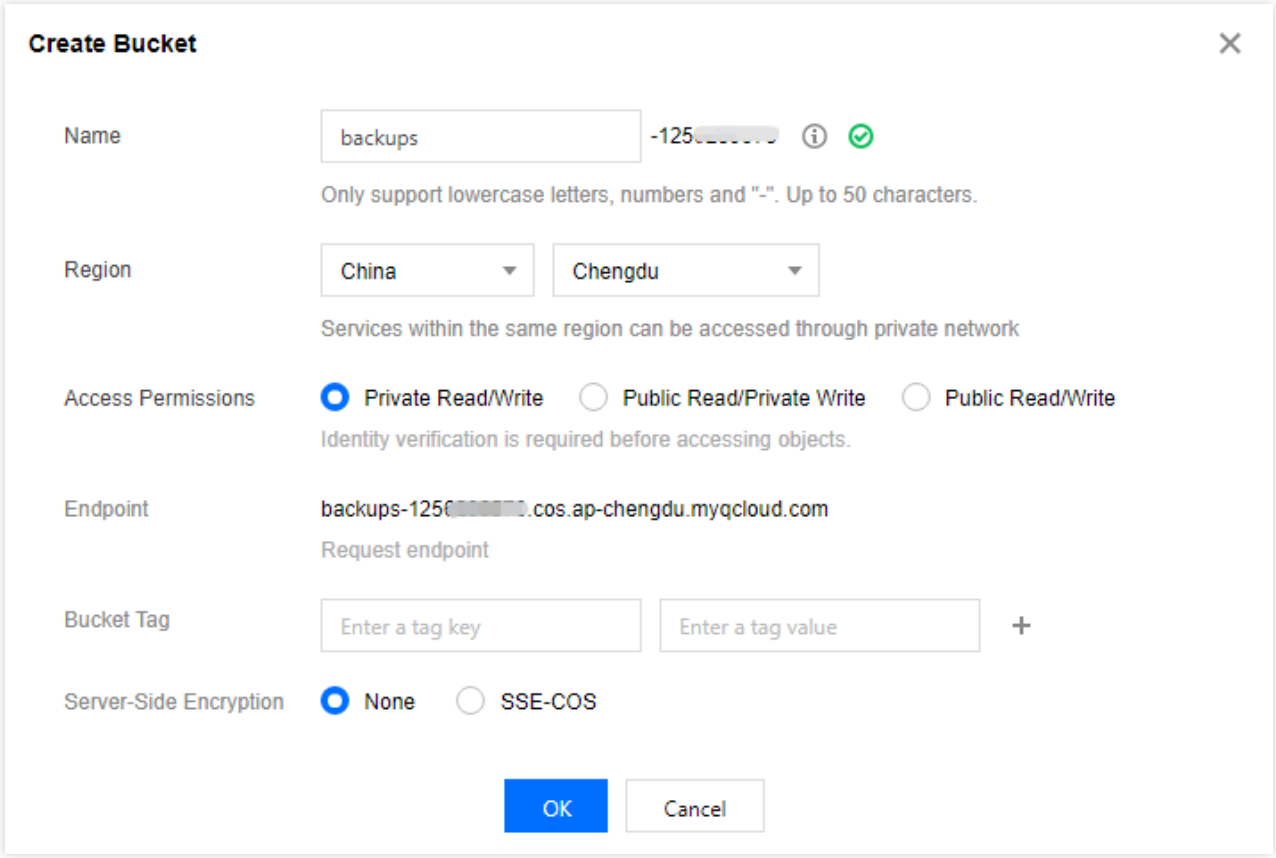
데이터의 가치가 무한하다는 데에는 누구도 이견이 없을 것입니다. 디지털 사진, 전자 문서, 업무 자료, 게임 아카이브 등 잃어버렸을 때 곤란하지 않을 데이터는 없습니다. 디스크 장애로 인한 파일 손실뿐 아니라 조작 오류, 컴퓨터 다운 또는 소프트웨어 크래쉬로 인한 파일 손실, '버전 롤백'이 필요할 때 저장된 이전 버전이 없다면 매우 당황스럽습니다. 그러니 백업의 중요성은 두말할 필요도 없지요.
백업이라고 하면 외장 하드 또는 LAN에 구축한 NAS 스토리지에 파일을 업로드하면 그만이라고 생각하는 사람이 많습니다. 정말 실제로도 이렇게 간단할까요?
백업은 사실 하나의 시스템 프로세스입니다. 파일을 백업 매체에 복사하는 것뿐만 아니라 백업 내용의 정확성도 검증해야 합니다. 복사와 검증은 파일 손실 시의 피해를 최소화할 수 있도록 정기적으로 실시해야 합니다. 또한 백업 매체도 점검이 필요하며, 손상된 디스크는 즉시 교체해야 합니다.
그렇다면 파일의 안전을 보장할 수 있는 간단한 방법은 없을까요? 답은 '있습니다'.
클라우드 서비스가 발전함에 따라 신뢰할 만한 기업급 클라우드 스토리지 서비스가 생겨났고, Tencent Cloud COS 객체 스토리지가 바로 이런 서비스입니다. 속도를 높이고 비용을 절감하고자 하는 정부의 노력에 부응하여 더욱 빠르고 저렴한 광대역이 등장함으로써 클라우드 파일 백업은 이미 상용화되었습니다. 이제는 컴퓨터의 파일과 클라우드 스토리지를 연결하여 파일을 정기적으로 클라우드에 자동 백업하고 백업 파일의 정확성을 검증하는 소프트웨어가 필요합니다.

소프트웨어 소개

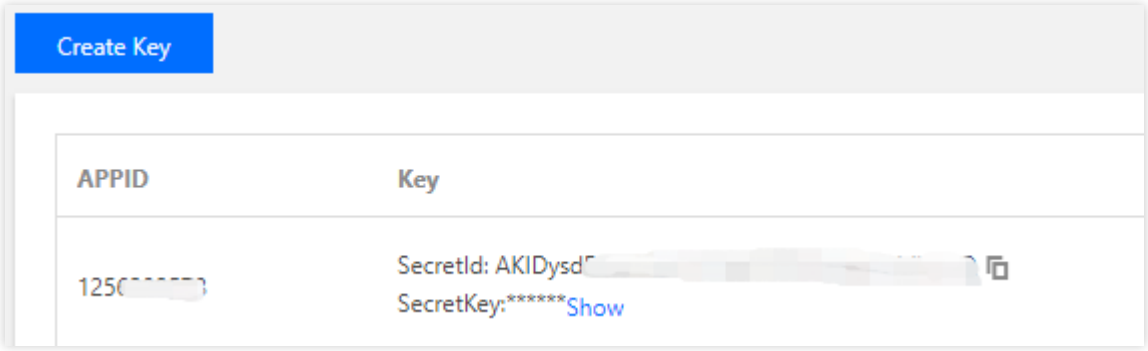
Arq® Backup은 Windows와 macOS 시스템을 지원하는 비즈니스 백업 소프트웨어입니다. 시스템 백그라운드에서 실행되며 설정에 따라 일정 시간마다 지정 디렉터리를 자동 백업하고, 매시간 파일을 백업해 두기 때문에 파일의 이전 버전을 쉽게 찾을 수 있습니다. 또한 시간마다 실행되는 백업은 변경 사항이 있는 파일만 백업하고 경로가 다른 중복 파일은 한 번만 백업함으로써 백업 용량을 최대한 작게, 백업 속도는 최대한 빠르게 합니다. 백업 파일을 네트워크로 전송하기 전, 소프트웨어는 네트워크 전송 과정 또는 클라우드 스토리지에서 도용되는 일이 없도록 사용자가 입력한 비밀번호를 바탕으로 백업 파일을 암호화하여 중요 데이터의 보안성을 보장합니다.
Arq® Backup 비즈니스 라이선스는 사용자 1명당 49.99USD입니다. 사용자는 구매 후 한 대의 컴퓨터에서 사용할 수 있습니다. 소프트웨어는 30일 무료 사용이 가능하므로 직접 사용해본 후 구매할 수도 있습니다.
설명：
Arq® Backup 소프트웨어는 현재 중국어 간체 버전이 없습니다. 소프트웨어의 다운로드, 구매 및 관련 설명은 해당 소프트웨어의 공식 홈페이지에서 확인할 수 있습니다.

Tencent Cloud COS 준비

설명：
현재 COS를 사용 중인 경우 1-2단계는 생략합니다.
1. Tencent Cloud 계정 생성 및 실명 인증을 완료합니다. 실명 인증을 하지 않은 사용자는 중국 내 리소스를 구매할 수 없습니다.
2. COS 콘솔에 로그인하여 지시에 따라 COS를 활성화합니다.
3. COS 콘솔에서 왼쪽 메뉴의 **버킷 리스트** 클릭 후 **버킷 생성**을 클릭하여 버킷 생성을 시작합니다.
이름: 버킷 이름입니다(예: “backups”).
소속 리전: 귀하의 소재지와 가까운 곳을 선택할 수 있으나 금융 클라우드 리전은 선택하지 마십시오. 현재 서남 리전에 가격 할인 혜택이 있으므로 '청두' 또는 '충칭'을 선택하면 가격 할인 혜택을 받을 수 있습니다.



기타 설정 항목은 기본값을 유지합니다. **도메인 요청** 주소를 복사 및 저장한 후 **확인**을 클릭하여 생성을 완료합니다.
설명：
자세한 버킷 생성 순서는 버킷 생성을 참조하십시오.
4. API Keys 콘솔에 로그인하여 키 정보 SecretId와 SecretKey를 생성 및 기록합니다.

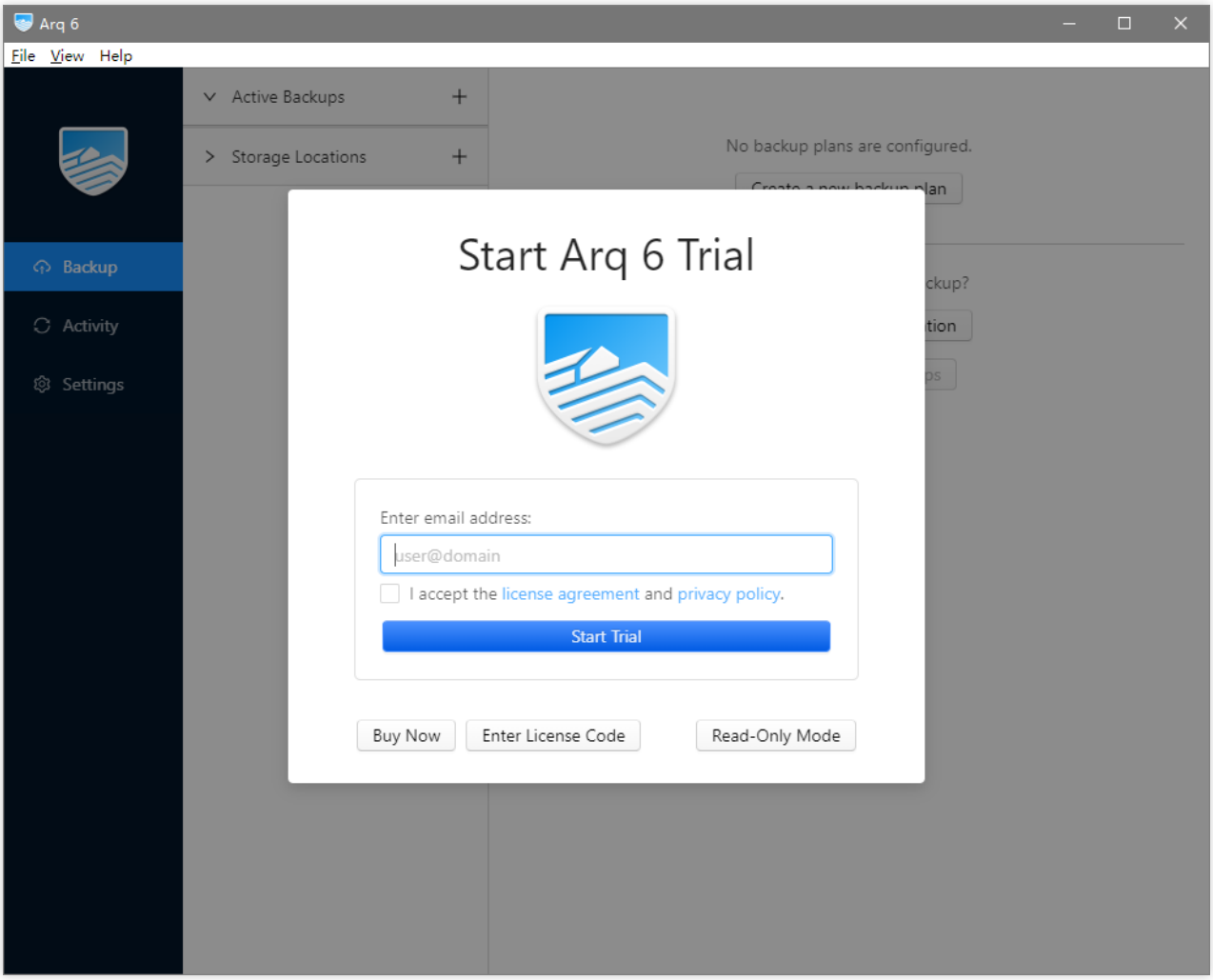


Arq® Backup 설치 및 설정

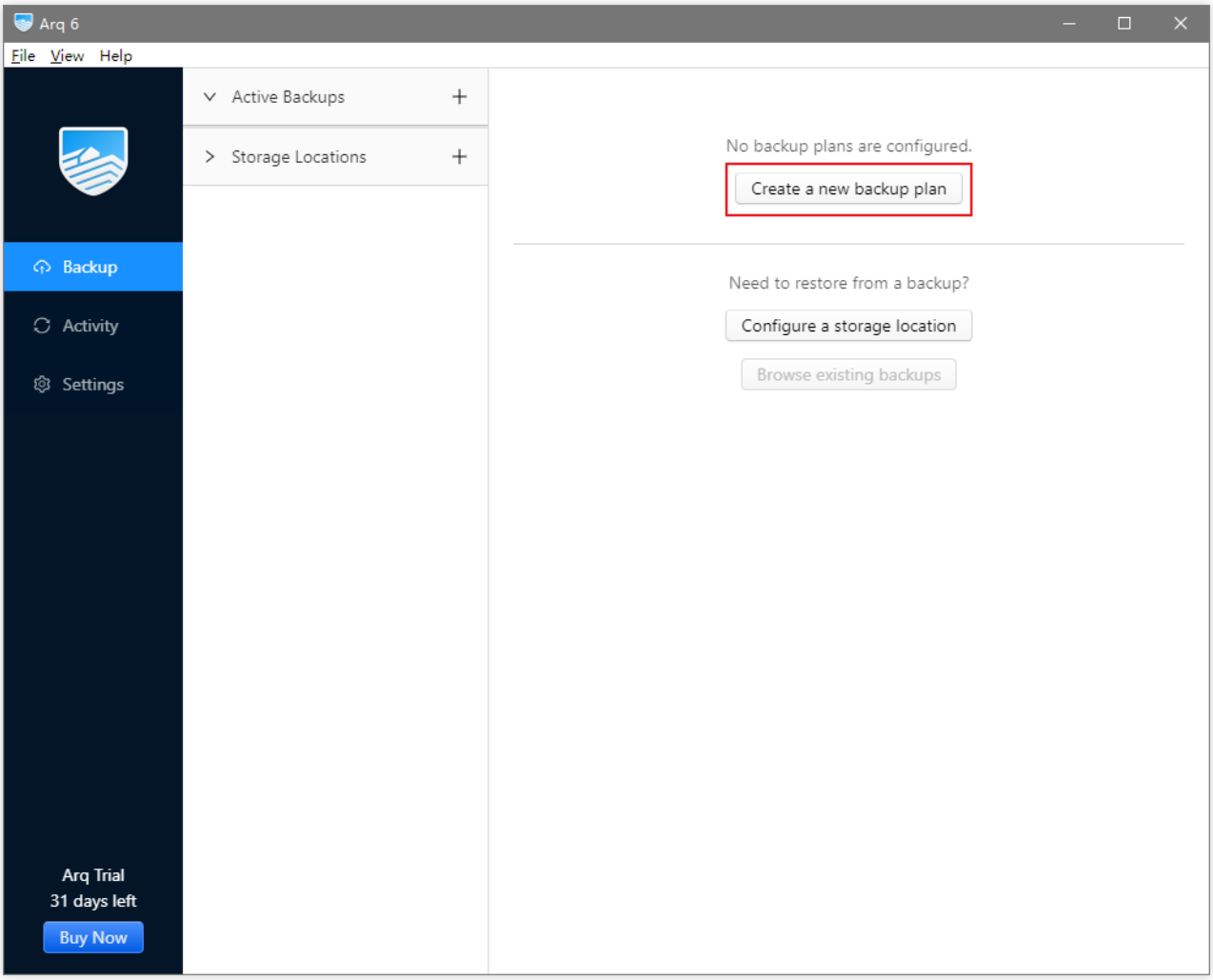
설명：

본 문서는 Windows의 Arq Backup 6.2.11 버전을 예시로 사용하였습니다.

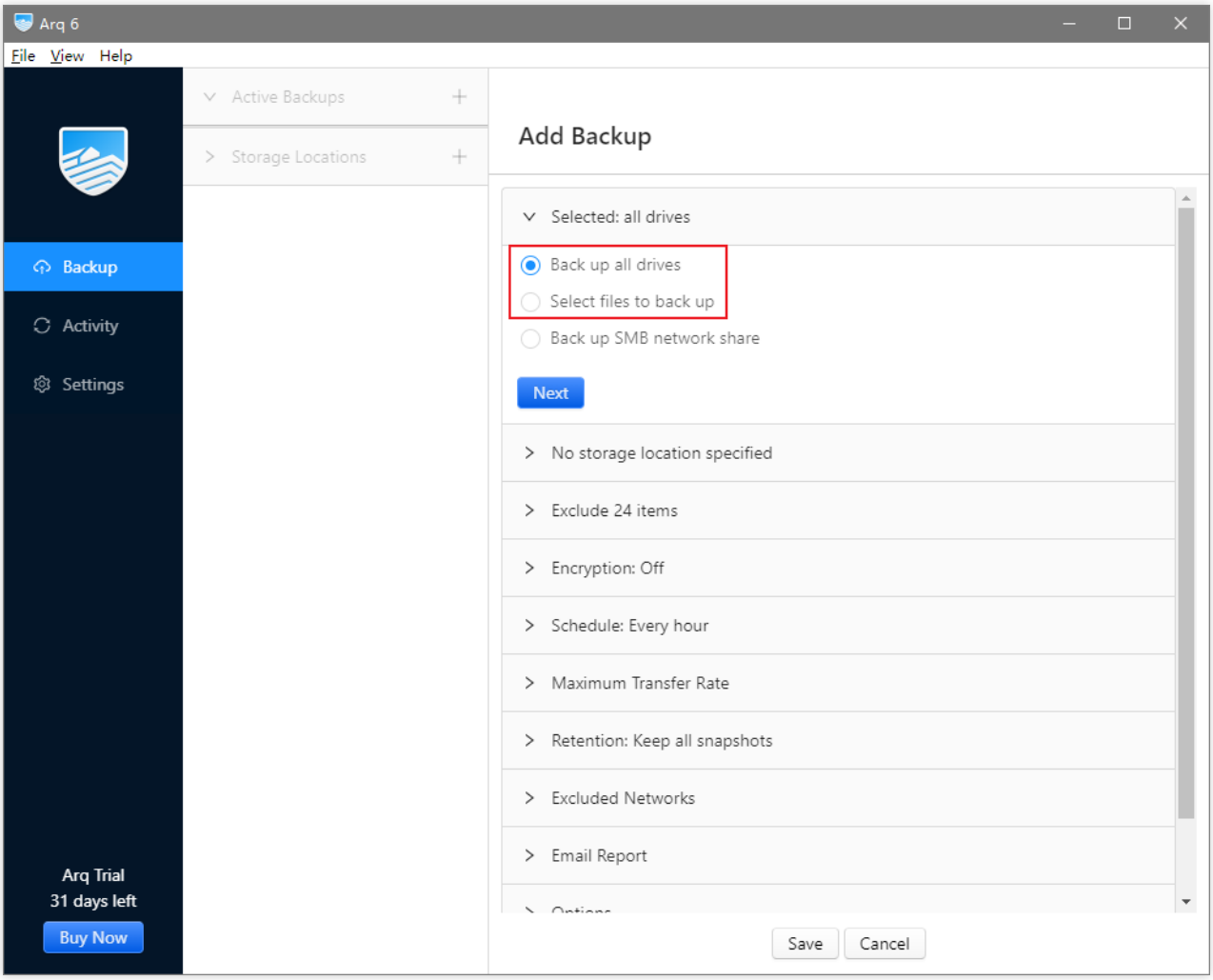
1. Arq® Backup 공식 홈페이지에서 소프트웨어를 다운로드합니다.
2. 지시에 따라 소프트웨어 설치를 완료하면 소프트웨어가 자동으로 실행됩니다. 최초 실행 시 로그인 알림이 뜹니다. 이때 이메일 주소를 입력하고 [Start Trial]을 클릭합니다.



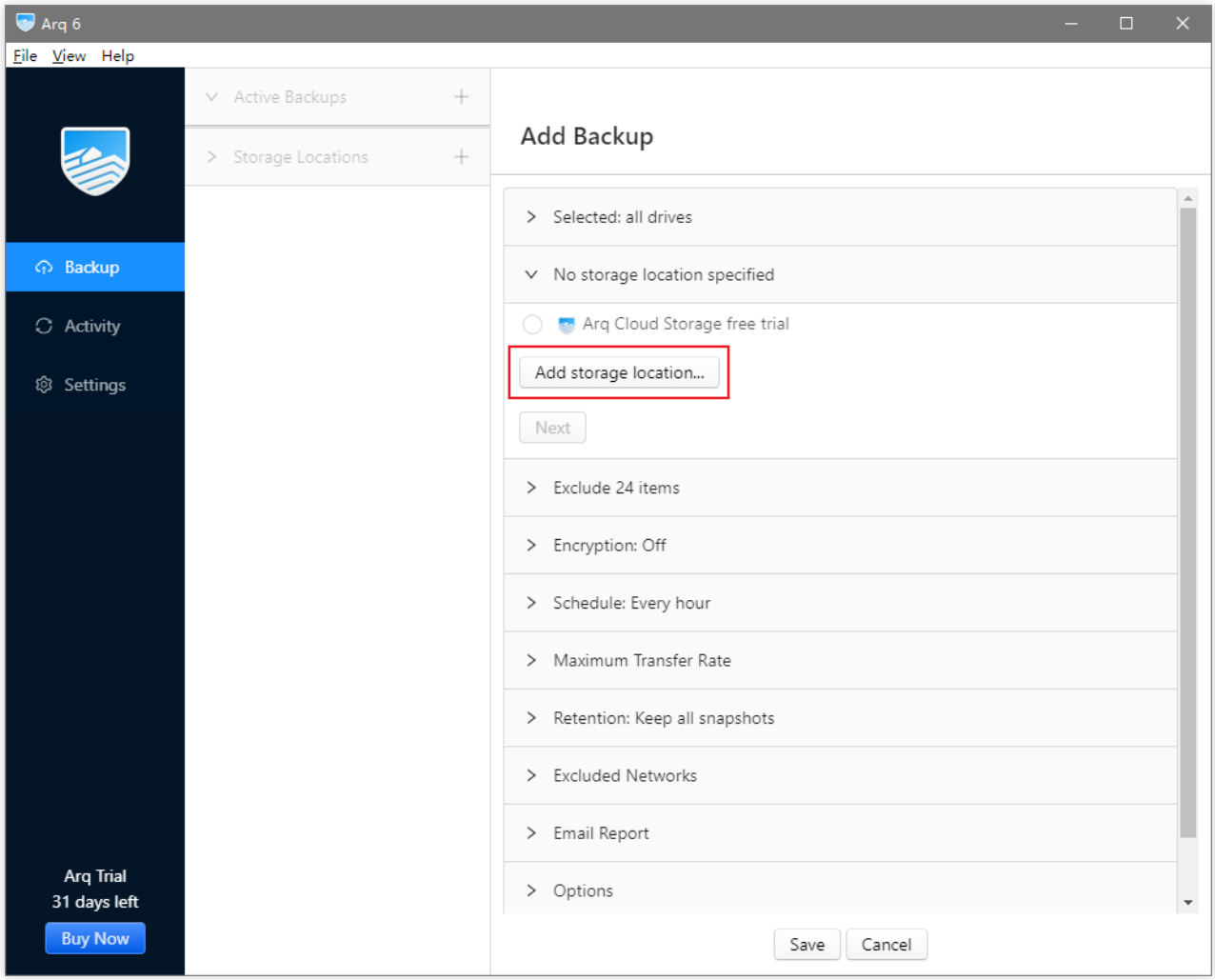
3.**Backup** 인터페이스에서**Create a new backup plan** 을 클릭하고 백업 플랜을 추가합니다.



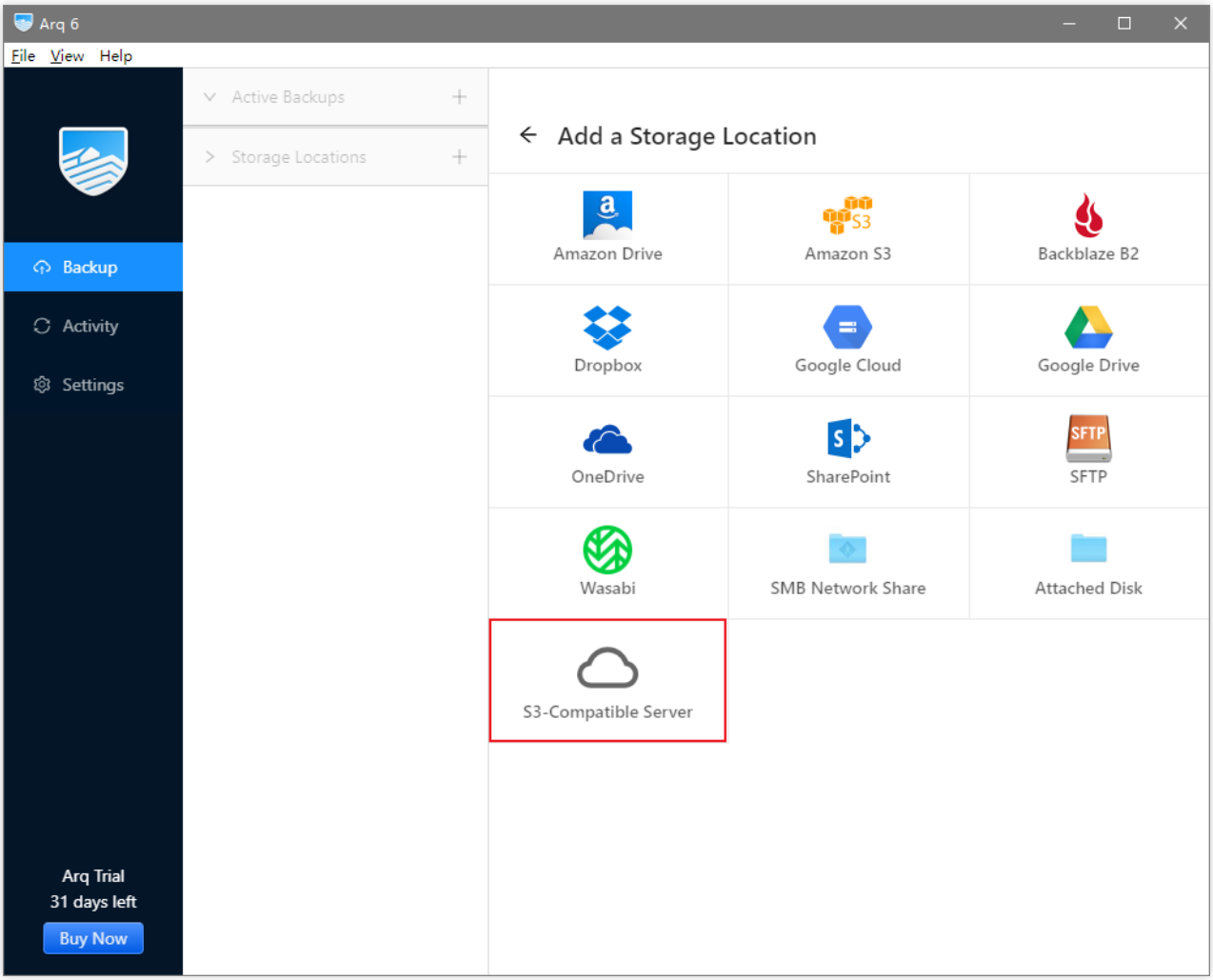
3. 리디렉션 인터페이스에서 백업할 디렉터리를 선택합니다. 전체 디스크 또는 지정 디렉터리를 선택할 수 있습니다.



5.**Add storage location** 을 클릭하고 아래 이미지와 같이 백업 스토리지 위치를 추가합니다.



6.**S3-Compatible Server** 를 클릭합니다.

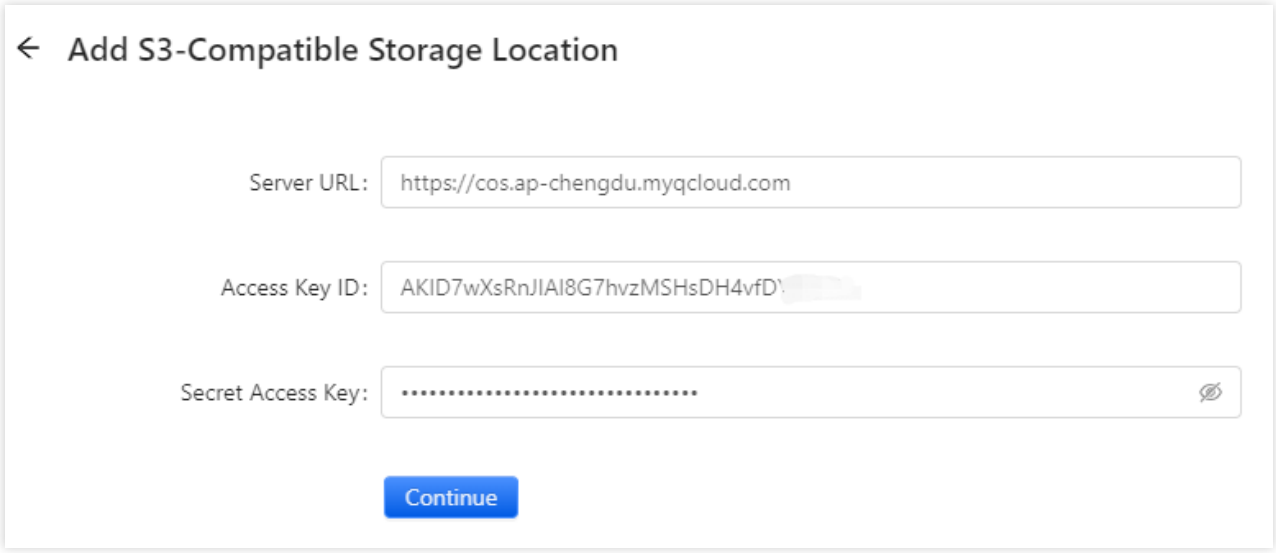


4. 리디렉션 인터페이스에서 다음 설명에 따라 설정합니다. 설정 완료 후**Continue** 를 클릭합니다.

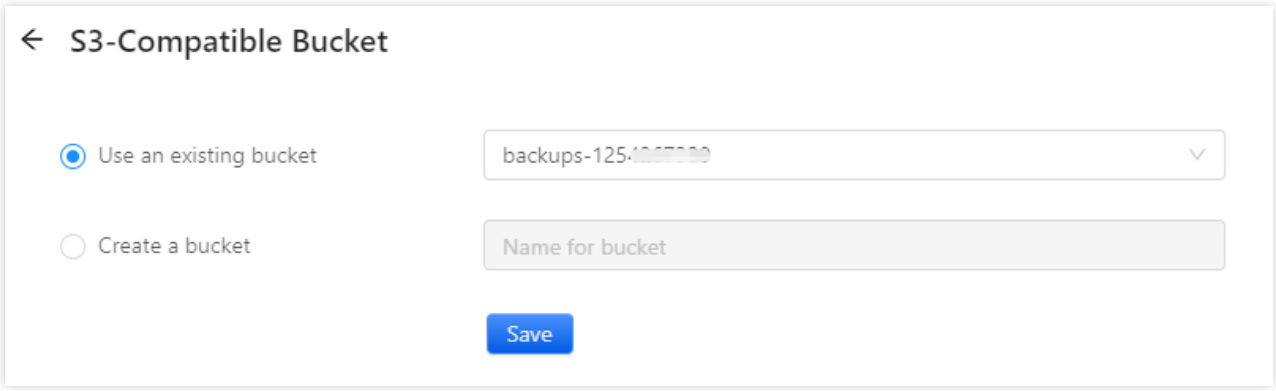
Server URL: 위에 기록된 도메인 요청 중 `cos` 로 시작하면서 앞에 `https://` 가 있는 주소를 입력합니다(예: `https://cos.ap-chengdu.myqcloud.com`). 버킷 이름은 포함되어 있지 않다는 것에 주의하십시오.

Access Key ID: 위에 기록된 키 정보 중 SecretId입니다.

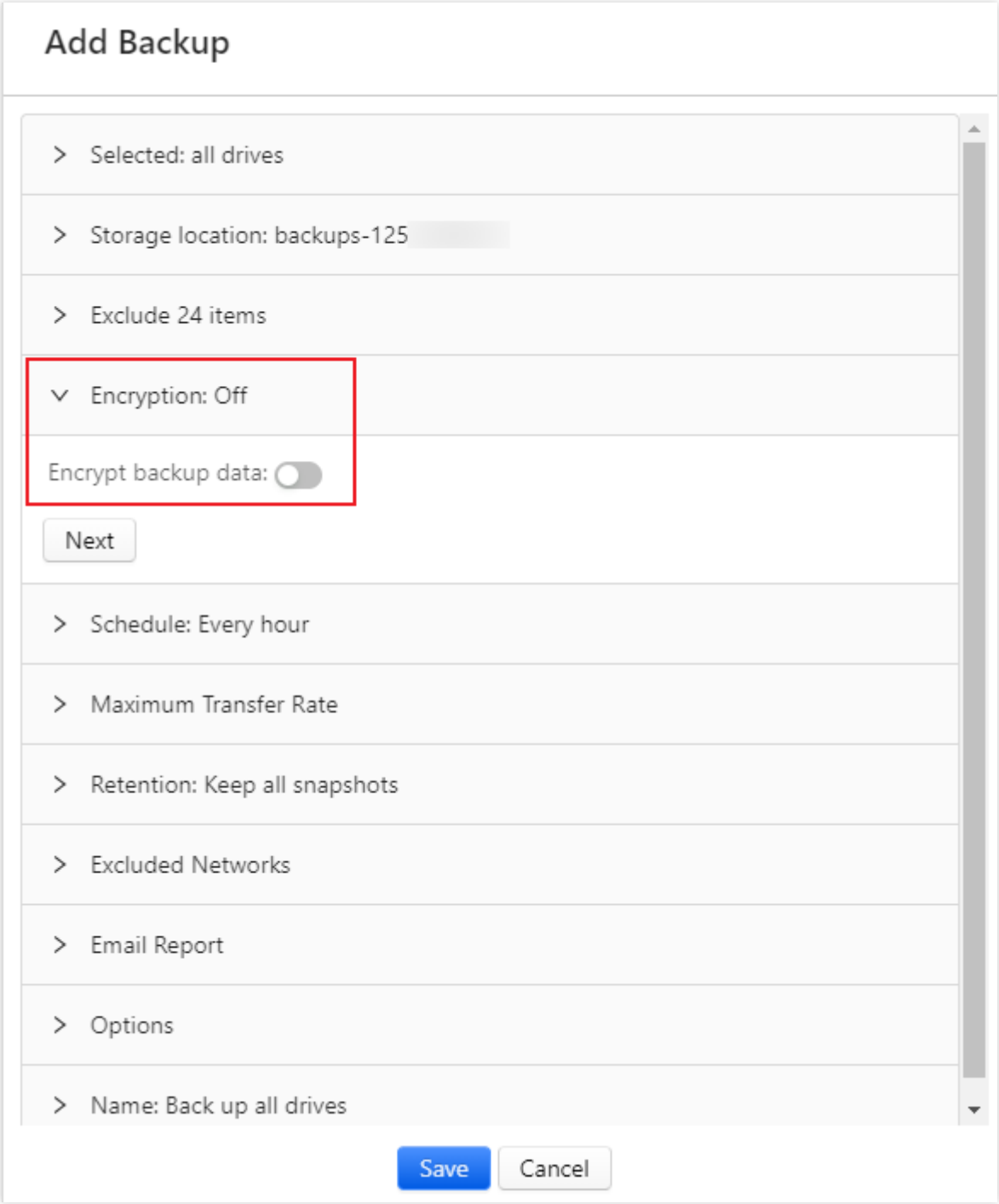
Secret Access Key: 위에 기록된 키 정보 중 SecretKey입니다.



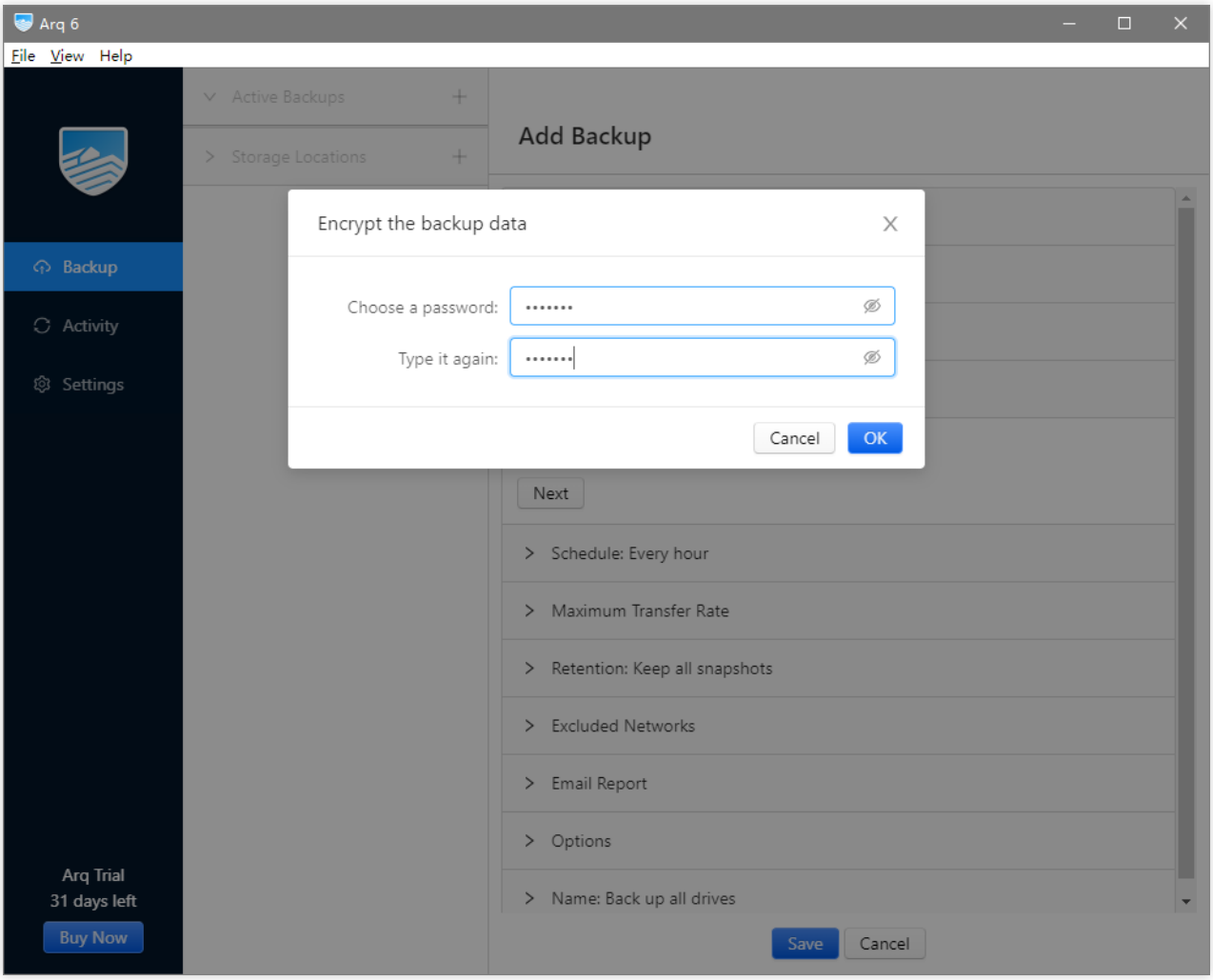
5. 다음 인터페이스에서**Use an existing bucket** 을 선택한 후 위에서 생성한 버킷(예: backups-1250000000)을 선택하고**Save** 를 클릭합니다.



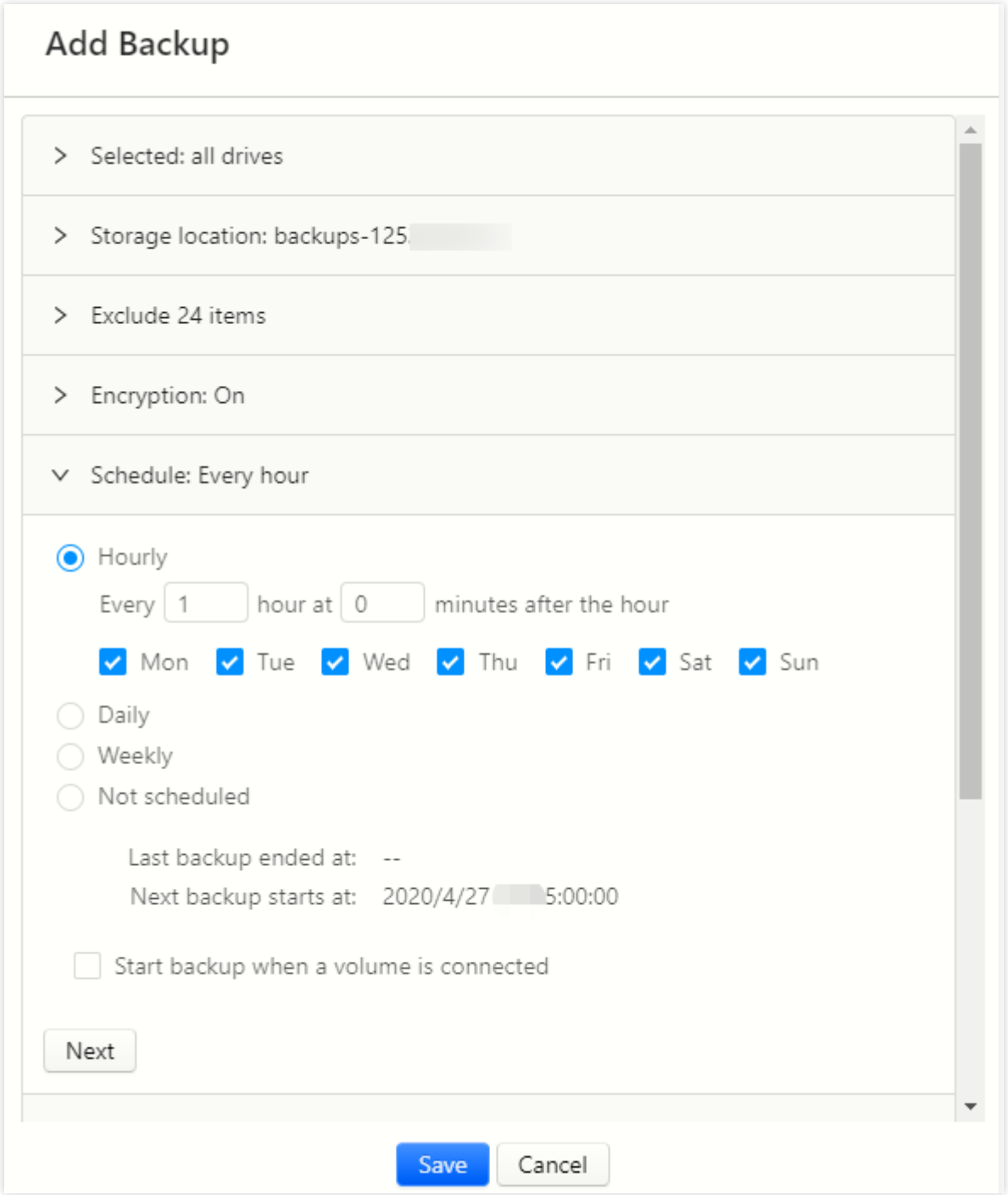
6. (옵션) 백업 데이터 암호화 여부를 선택합니다. **활성화** 버튼을 선택합니다.



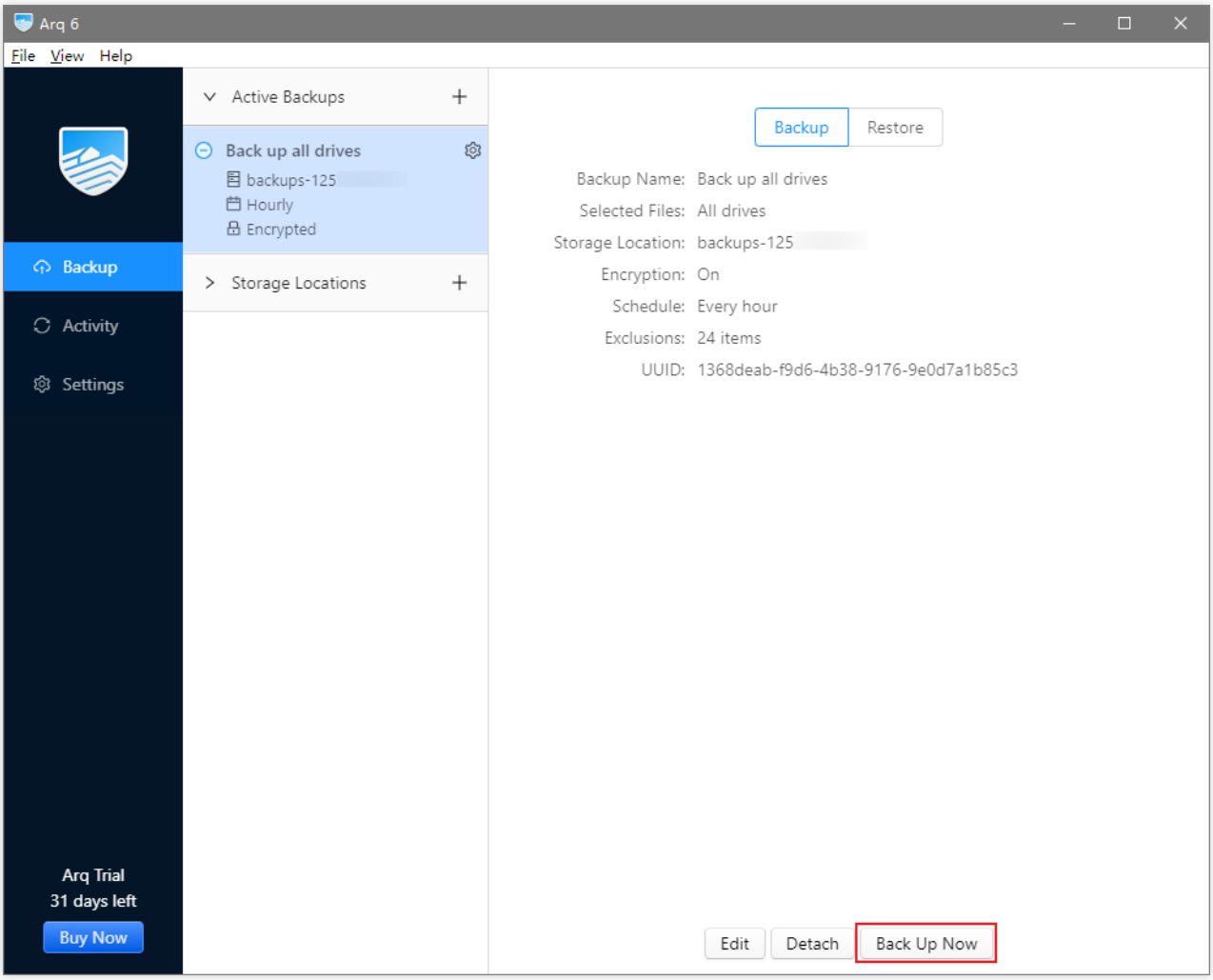
7. 팝업 창에서 암호화에 사용되는 비밀번호를 설정합니다. 백업 파일 암호화에 사용되는 비밀번호를 두 번 입력하고 **OK** 를 클릭합니다. **주의: 백업 비밀번호를 기억하지 않으면 백업에서 파일을 복구할 수 없습니다!**



8. (옵션) 백업 주기를 설정합니다.

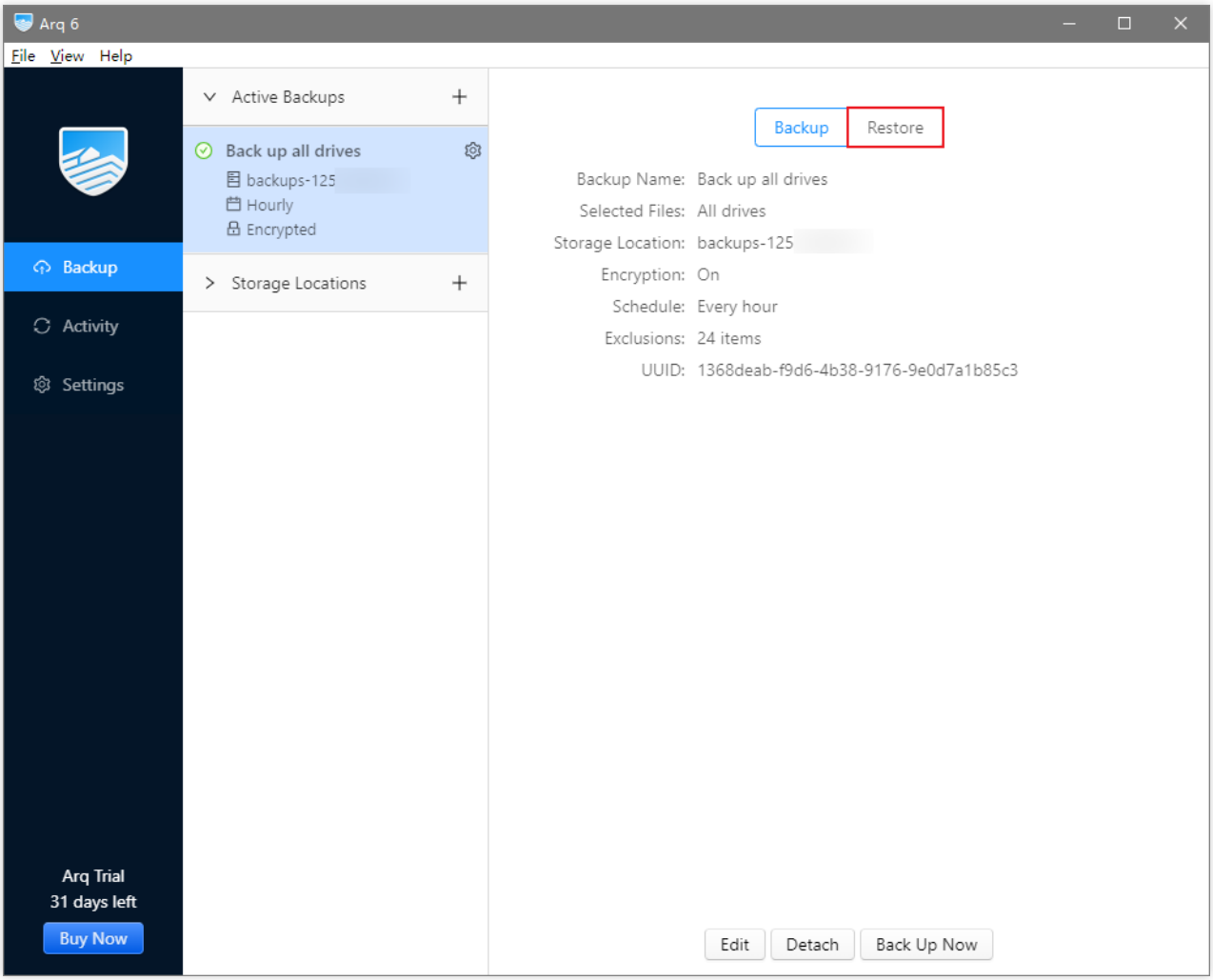


12.**Save** 를 클릭하여 설정을 저장한 후**Back Up Now** 를 클릭하여 백업을 시작합니다.

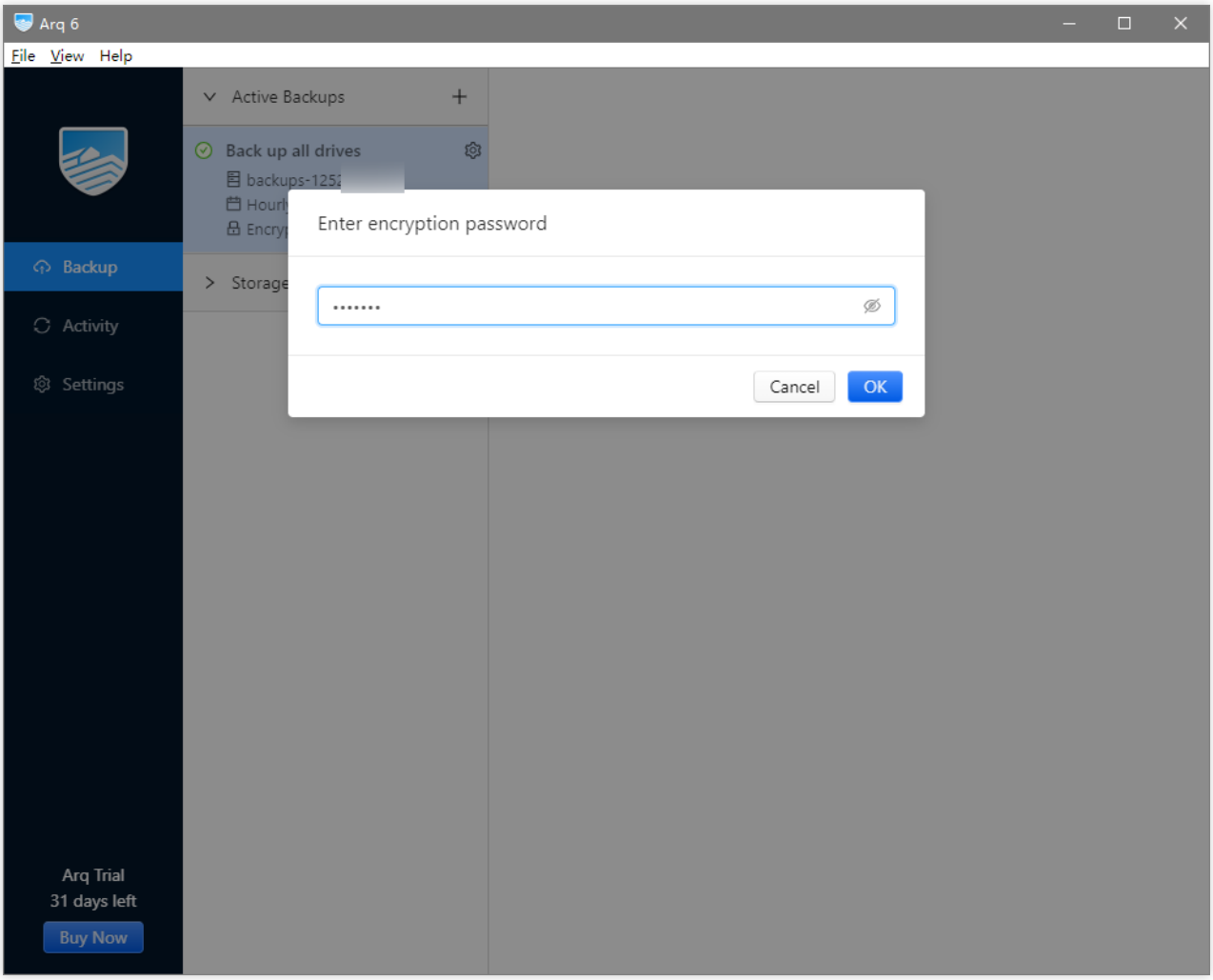


백업에서 파일 복구

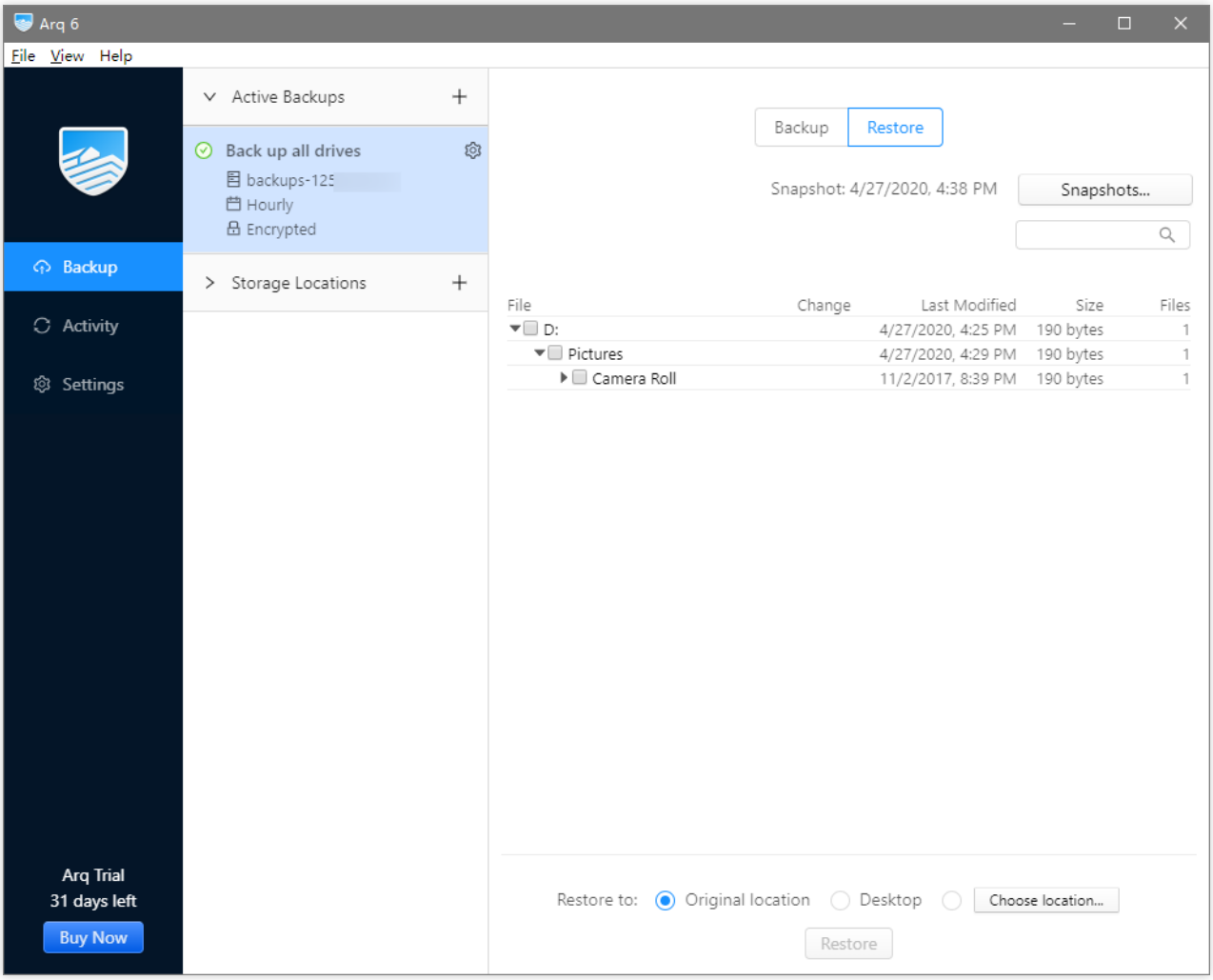
1. 메인 인터페이스 왼쪽 **Backup** 리스트에서**Restore** 를 클릭합니다.



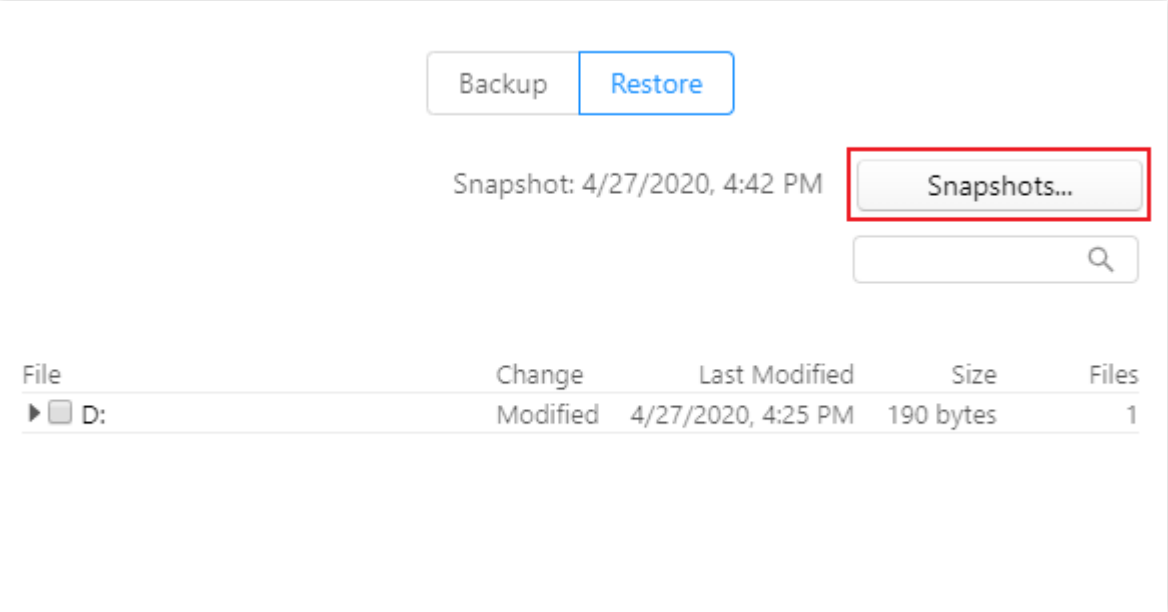
2. 위의 9단계에서 백업 데이터 암호화를 설정한 경우 비밀번호를 입력해야 합니다.



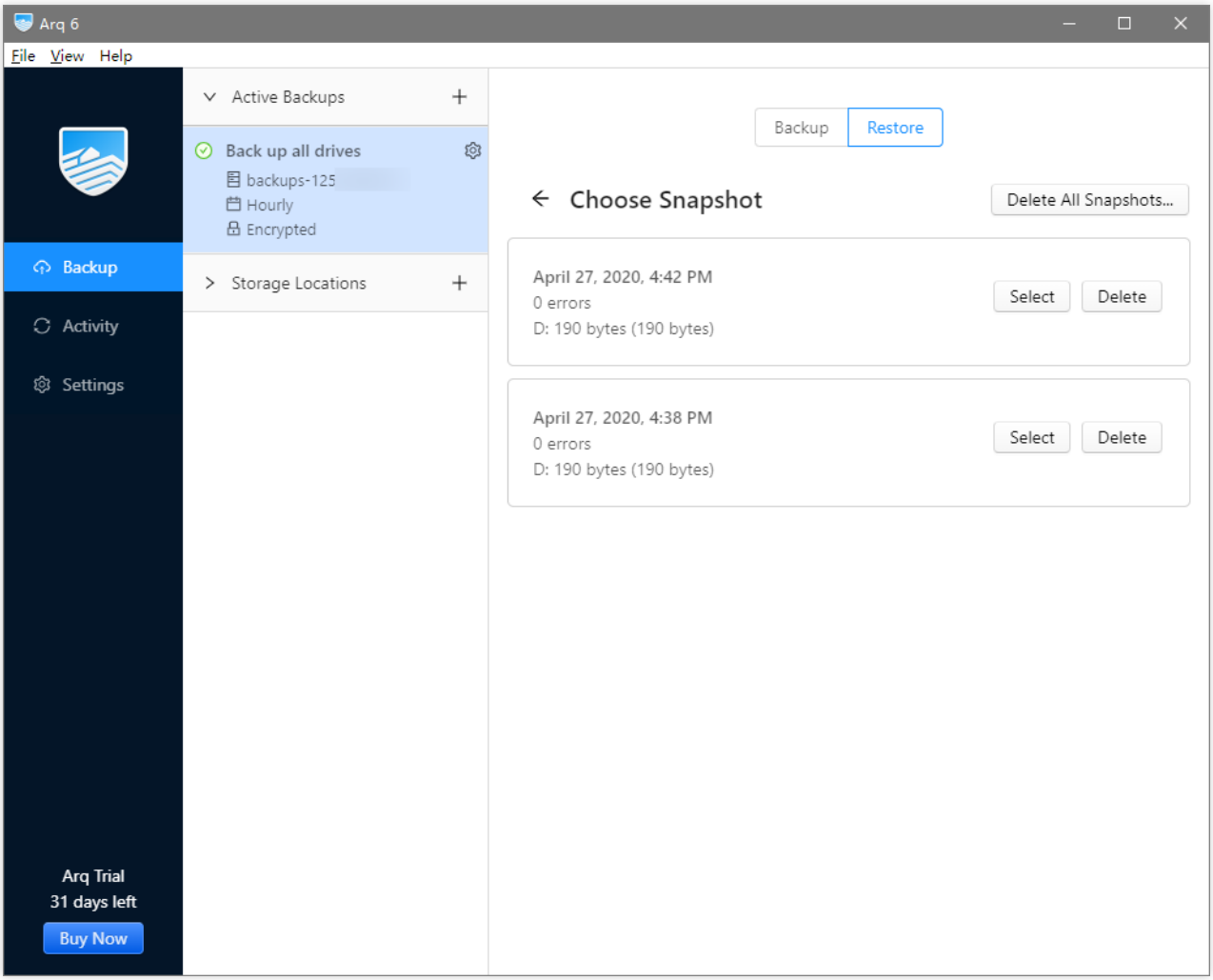
3. 복구할 디렉터리 또는 파일 및 복구한 디렉터리 또는 파일의 저장 위치를 선택한 후 **Restore** 를 클릭하여 복구를 시작합니다.



4. 복구 작업은 기본적으로 최신 백업에서 복구됩니다. 필요에 따라 스냅샷에서 이전 버전 백업을 찾아 이전 버전 백업에서 복구를 진행할 수도 있습니다.**Snapshots** 를 클릭하여 이전 스냅샷을 조회합니다.



5. 이전 스냅샷을 선택합니다.



6. 복구할 디렉터리 또는 파일 및 복구한 디렉터리 또는 파일의 저장 위치를 선택한 후 **Restore** 를 클릭하여 복구를 시작합니다.
7. 복구가 완료되었다는 인터페이스 안내가 뜨면 조금 전 지정한 디렉터리에서 복구된 파일을 확인할 수 있습니다.

NextCloud + COS로 개인 클라우드 생성

최종 업데이트 날짜: : 2024-06-24 16:53:18

서문

NextCloud는 오픈 소스 클라이언트 및 서버 소프트웨어 애플리케이션으로 개인 온라인 파일 스토리지 서비스를 직접 만들 수 있도록 도와줍니다.
NextCloud 서버는 PHP로 작성되었으며 서버의 로컬 디스크를 기본 스토리지로 사용합니다. COS(Cloud Object Storage)를 기본 스토리지로 사용하도록 NextCloud 구성을 수정하여 더 낮은 스토리지 비용으로 더 높은 안정성, 더 강력한 재해 복구 기능 및 무제한 스토리지 공간을 누릴 수 있습니다.
본 문서는 NextCloud 서버가 의존하는 환경과 로컬 스토리지와 COS의 차이점을 비교 분석하고 개인 온라인 파일 스토리지 서비스 구축 방법을 설명합니다.
주의 :
기존의 NextCloud 서버 인스턴스를 로컬 스토리지에서 COS로 전환하면 기존 파일이 보이지 않을 수 있습니다. 기존 인스턴스의 저장 방법을 변경하려면 새로운 NextCloud 서버를 구축하고 COS를 스토리지로 구성한 다음 이전 인스턴스에서 새 인스턴스로 데이터를 마이그레이션하는 것이 좋습니다.

NextCloud 서버 환경 소개

NextCloud의 서버는 PHP로 작성되며, 데이터베이스는 SQLite, MySQL, MariaDB 또는 PostgreSQL을 사용할 수 있습니다. 이중 SQLite는 성능의 제약으로 인해 실제 애플리케이션에는 일반적으로 권장되지 않습니다. PHP, MySQL과 관련된 서버 소프트웨어는 Windows 버전을 모두 갖추고 있지만, Windows에서 실행되는 NextCloud 서버에 텍스트 인코딩 등의 문제가 있을 수 있다는 NextCloud 커뮤니티의 피드백에 따라 공식 홈페이지는 Windows에 설치된 NextCloud 서버를 지원하지 않겠다고 발표했습니다.

서버 구성

Cloud Virtual Machine(CVM)은 현재 각각 여러 서브 패밀리가 있는 여러 인스턴스 패밀리를 제공합니다. 다른 인스턴스 패밀리는 큰 메모리 또는 높은 IO와 같은 다른 강점을 가지고 있습니다. NextCloud는 개인, 가정, 중소기업 사용자를 대상으로 하므로 하드웨어 리소스에 대한 요구 사항이 낮고 리소스가 균형 잡힌 표준형 모델을 선택할 수 있습니다. CVM 인스턴스 서브 패밀리의 경우 일반적으로 최신 제품이 비용 대비 성능이 높으므로 최신 서브 패밀리를 선택하면 됩니다.
인스턴스 패밀리 및 서브 패밀리를 결정한 후 특정 vCPU 및 메모리 사양을 선택해야 합니다(사설망 대역폭 및 PPS는 사양에 따라 다름). PHP용 OPcache를 사용하여 성능을 향상시킬 수 있으며, NextCloud 서버는 APCu 메모리 캐시를 사용하여 성능을 더욱 향상시킬 수 있으므로 대용량 메모리를 선택하는 것이 좋습니다.

구매 후 CVM 인스턴스 구성을 조정할 수 있으므로 1코어 vCPU와 4GB MEM과 같은 낮은 사양의 인스턴스를 먼저 구매하고 온라인 파일 스토리지 서비스를 구축하여 운영 환경에 런칭한 후 사용자 및 파일 수와 CVM 모니터링 데이터를 기반으로 성능 향상을 위한 사양 업그레이드 여부를 결정합니다. 가정 또는 중소기업과 같은 다중 사용자 시나리오에서 서비스를 사용해야 하는 경우 충분한 성능을 제공하기 위해 2코어 8GB MEM 또는 4코어 16GB MEM 인스턴스를 구매하는 것이 좋습니다.

서버 운영 체제

주요 Linux 릴리스 버전은 NextCloud 서버를 잘 유지할 수 있습니다. 이들의 구성은 소프트웨어 패키지 설치에 사용되는 명령(패키지 관리 툴)을 제외하고는 기본적으로 동일합니다.

설명：

본문은 CentOS 7.7의 CVM 인스턴스를 예로 들어 설명합니다.

데이터베이스

앞서 언급한 바와 같이 실제 업무에서는 MySQL을 PHP와 함께 사용하는 것이 일반적입니다. MariaDB는 MySQL의 '포크(fork)' 버전으로, MySQL과의 호환성이 높기 때문에 NextCloud 서버는 MySQL 5.7+ 또는 MariaDB 10.2+에서 잘 실행될 수 있습니다.

Tencent Cloud는 TencentDB for MySQL과 TencentDB for MariaDB를 제공합니다. 두 서비스 모두 원본-복제 고가용성 아키텍처를 채택하고 CVM에서 자체 구축한 데이터베이스에 비해 안정성이 더 높습니다. 또한 자동 백업과 같은 사용하기 쉬운 Ops 기능을 지원합니다. 따라서 실제 비즈니스에서 TencentDB를 사용하는 것이 좋습니다.

설명：

본문은 TencentDB for MySQL 5.7 인스턴스를 예로 들어 설명합니다.

Web 서버 및 PHP Runtime

일부 NextCloud 서버 구성은 `.htaccess` 파일에 지정되어 있으므로 Apache 서버 소프트웨어 애플리케이션을 사용할 때 NextCloud의 내장 구성 항목을 직접 사용할 수 있습니다. Nginx는 최근 급속도로 발전하고 있는 Web 서버 소프트웨어 애플리케이션으로 Apache에 비해 설치 및 구성이 쉽고 리소스 사용량이 적으며 로드 용량이 큰 것이 특징입니다. NextCloud 서버의 `.htaccess` 구성을 Nginx 구성으로 변환하여 NextCloud 서버를 더 잘 유지할 수 있습니다. 본문은 Nginx 서버 소프트웨어 애플리케이션을 사용하며 참고용으로 완전한 Nginx 구성 예시를 제공합니다.

PHP Runtime은 PHP 7로 업그레이드되었으며 주요 유지 관리 버전에는 모두 NextCloud 서버를 지원하는 7.2, 7.3 및 7.4가 포함됩니다. 여기에서는 최신 버전 7.4를 사용합니다. 또한 NextCloud는 특정 PHP 모듈에 의존합니다. 특정 모듈에 대한 요구 사항은 아래에 자세히 설명되어 있습니다.

Tencent Cloud 네트워크 환경

현재 Tencent Cloud는 클래식 네트워크 및 VPC 환경을 제공합니다. 클래식 네트워크는 모든 Tencent Cloud 사용자가 공유하는 공중망 리소스 풀로, 모든 CVM 인스턴스의 사설망 IP가 Tencent Cloud에 의해 할당되며 IP 범위 또는 IP 주소를 사용자 지정할 수 없습니다. VPC는 Tencent Cloud에서 논리적으로 격리된 네트워크 공간입니다. VPC에서 IP 범위, IP 주소 및 라우팅 정책을 사용자 지정할 수 있습니다. 현재 클래식 네트워크 리소스가 부족하고 확장할 수 없기

때문에 클래식 네트워크는 더 이상 새 계정과 일부 새 AZ에서 지원되지 않습니다. 따라서 이 문서에서는 VPC를 예로 들어 설명합니다.

설명：

VPC에 대한 자세한 내용은 개요를 참고하십시오.

CBS와 COS의 비교

CBS(Cloud Block Storage) 디스크는 CVM 인스턴스의 로컬 디스크로 운영 체제에 마운트됩니다. NextCloud는 기본적으로 파일 시스템을 사용하여 온라인 파일 스토리지 데이터를 저장하므로 NextCloud 데이터를 운영 체제의 CBS 디스크에 직접 저장할 수 있습니다. CBS와 비교하여 COS는 다음과 같은 이점을 제공합니다.

사용 사례

CBS

CBS는 일종의 블록 스토리지로 CVM 운영 체제에 디스크로 직접 마운트할 수 있습니다. 일반적으로 CBS 디스크는 운영 체제에서만 사용되며 하나의 CVM 인스턴스에만 마운트할 수 있습니다. 그러나 읽기/쓰기 성능이 높고 높은 IO와 낮은 대기 시간이 필요한 시나리오에 적합하며 하나의 CVM 인스턴스에서만 독점적으로 사용됩니다.

COS

COS는 HTTP 프로토콜을 통해 읽기/쓰기 API를 개방하므로 프로그래밍을 통해 COS에 저장된 객체(파일)에 액세스해야 합니다. 파일 경로와 같은 객체 Key를 인덱스로 사용하며 저장 용량이 무제한입니다. 데이터가 네트워크를 통해 전송되기 때문에 COS는 속도가 느리고 대기 시간이 더 깁니다. 그러나 객체에 대한 작업이 수행되므로 애플리케이션에서 객체를 조작한 후 다른 애플리케이션에서 즉시 동일한 객체를 조작할 수 있습니다. 결론적으로 COS는 고성능이 필요하지 않지만 비용 효율적인 대용량 스토리지 또는 공유 액세스가 필요한 시나리오에 적합합니다. 다음과 같은 이유로 온라인 파일 스토리지 애플리케이션을 COS와 함께 사용하는 것이 더 적합합니다. 온라인 파일 스토리지 애플리케이션은 네트워크를 통해 데이터를 전송하며 짧은 대기 시간이 필요하지 않습니다. 온라인 파일 저장 클라이언트에서 서버로 그리고 COS로 연결되는 속도와 대기 시간은 주로 클라이언트 네트워크에 따라 달라집니다. COS는 자체 속도를 제한하지 않습니다.

유지 관리

CBS

CBS에는 고정 용량이 있으며 콘솔 또는 Tencent Cloud API를 통해 확장할 수 있습니다. 확장 후에는 운영 체제에 파티션을 추가해야 하며 파티션 예외가 발생할 수 있으며 일정한 유지 관리 비용이 발생합니다.

COS

COS는 주문형으로 사용되며 총 용량이나 객체(파일)의 수를 제한하지 않으므로 유지 관리가 전혀 필요하지 않습니다.

데이터 보안

CBS와 COS 모두 데이터의 신뢰성을 보장하기 위해 다중 복사와 같은 방법을 사용합니다.

NextCloud 서버 실행 환경 구축

NextCloud 서버가 의존하는 Tencent Cloud 제품 준비

CVM

CVM을 시작하려면 사용자 정의 구성을 참고하십시오.

TencentDB for MySQL

시작하려면 MySQL 인스턴스 생성을 참고하십시오.

COS

1. COS 콘솔에 로그인(처음 사용하는 경우 COS를 먼저 활성화해야 함)하고 **버킷 리스트** 페이지로 이동하여 **버킷 생성**을 클릭하고 다음과 같이 구성합니다.

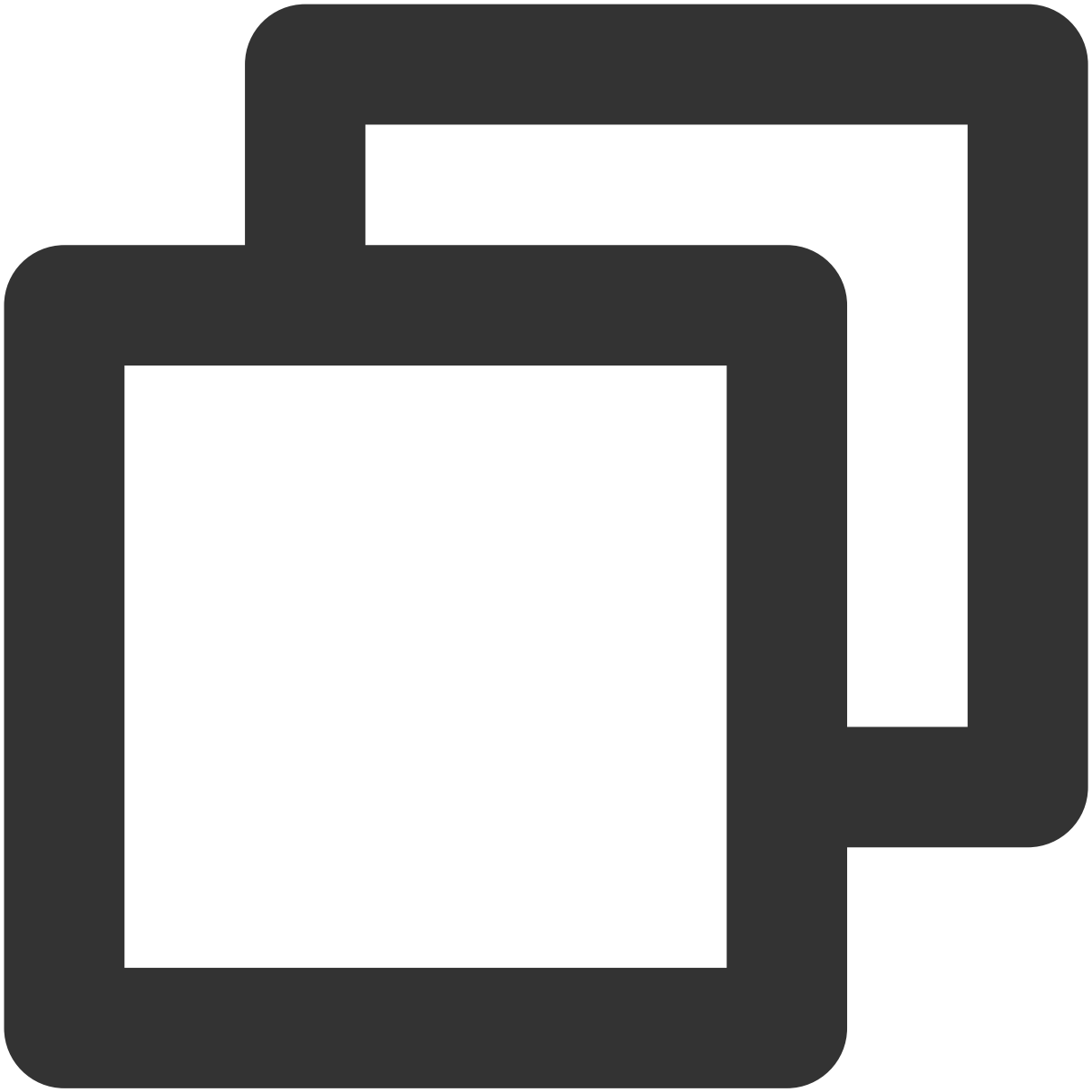
구성 항목	값
이름	nextcloud와 같은 사용자 정의 버킷 이름을 입력하십시오. 한 번 확인된 이름은 변경할 수 없습니다.
리전	CVM 인스턴스의 리전을 선택합니다
기타	기본 설정을 유지합니다

2. 상기 구성 항목을 설정한 후 **확인**을 클릭합니다.

Web 서버와 PHP Runtime 설치 및 구성

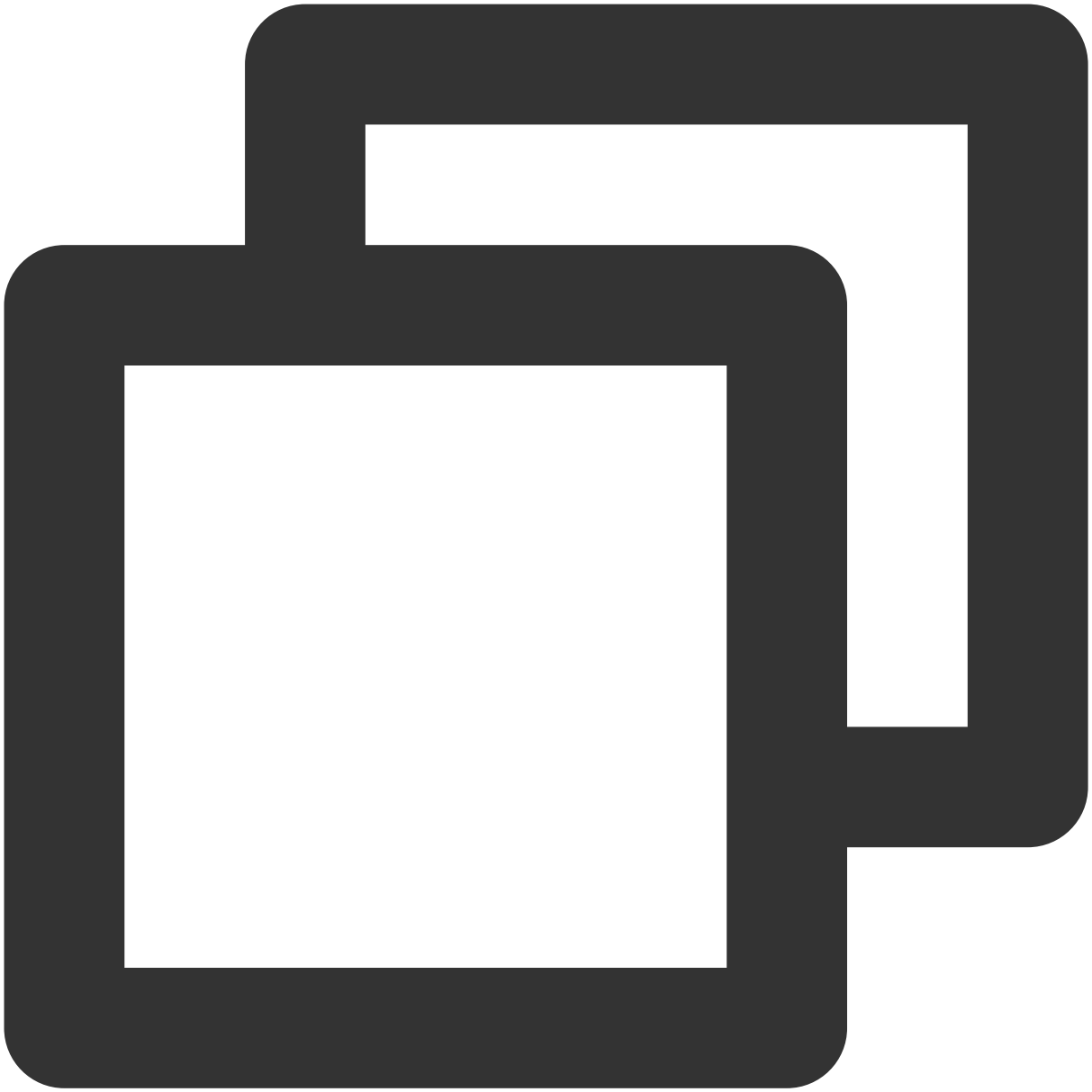
Nginx 설치

1. SSH 툴을 사용하여 새로 구매한 서버에 로그인합니다.
2. 다음 명령을 실행하여 Nginx를 설치합니다.



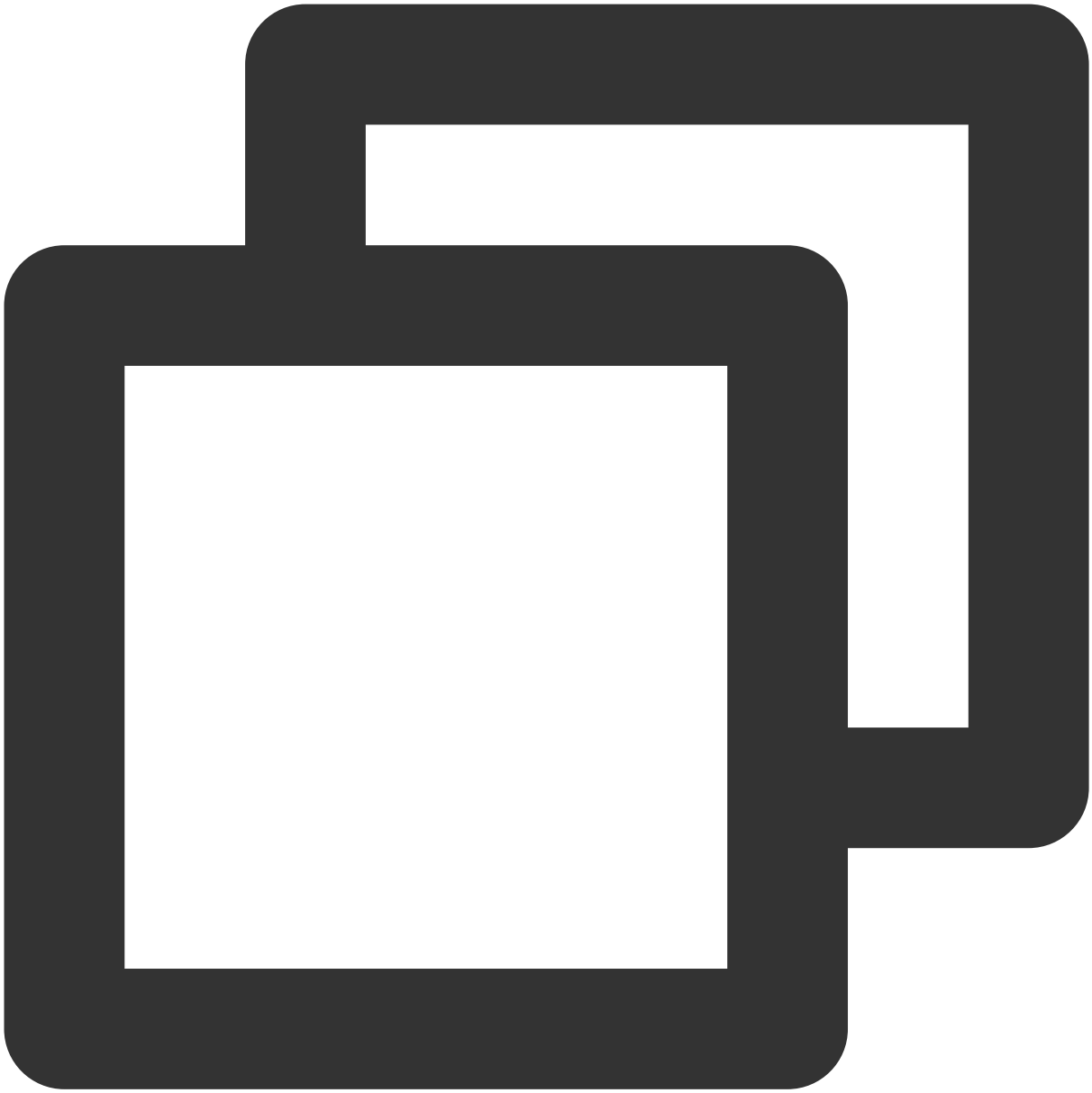
```
yum install nginx
```

다음과 같은 정보가 뜨면 `Y` 와 Enter 키를 눌러 설치를 확인합니다(이하 동일).



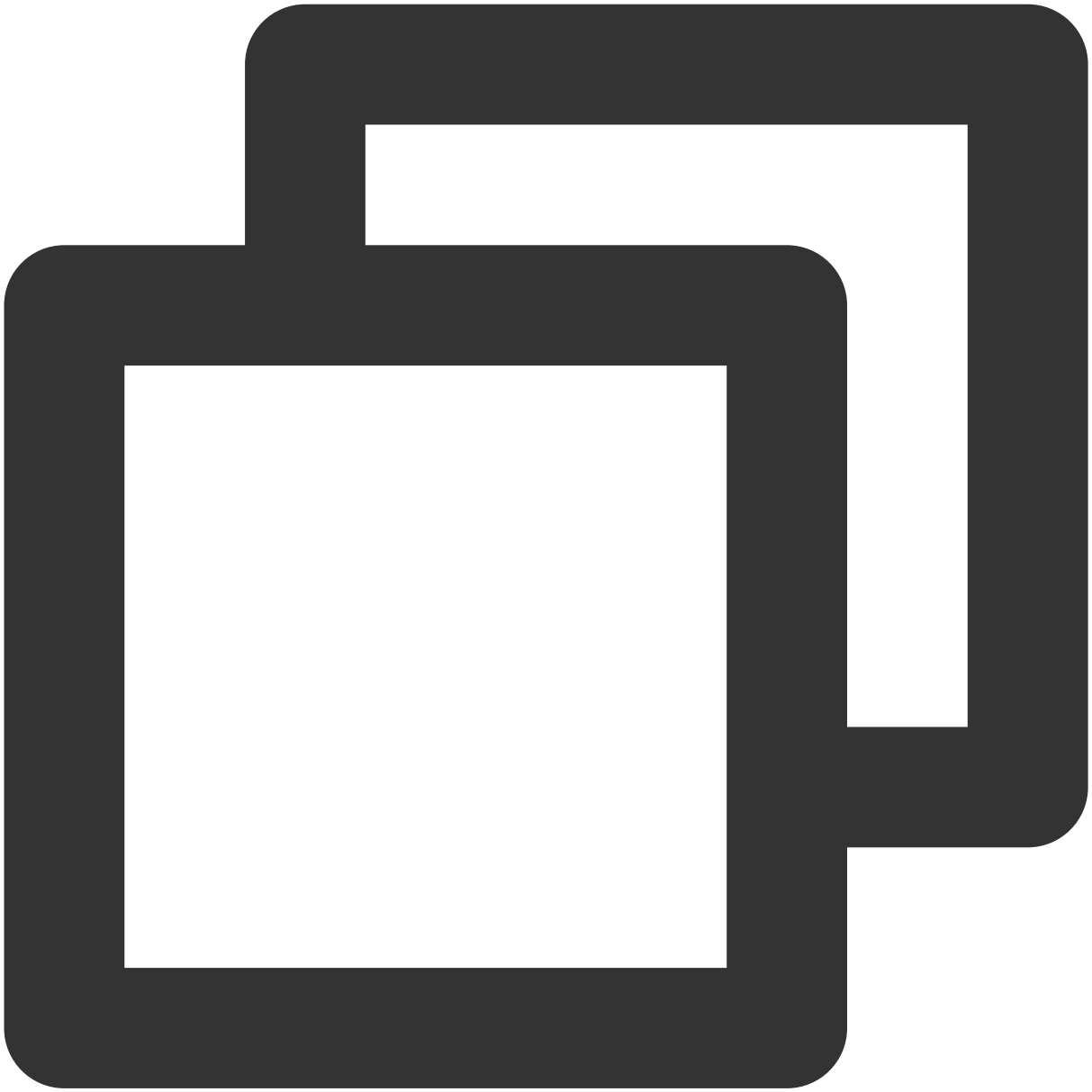
```
Is this ok [y/d/N]:
```

3. 다음과 같은 정보가 뜨면 설치가 완료된 것입니다.



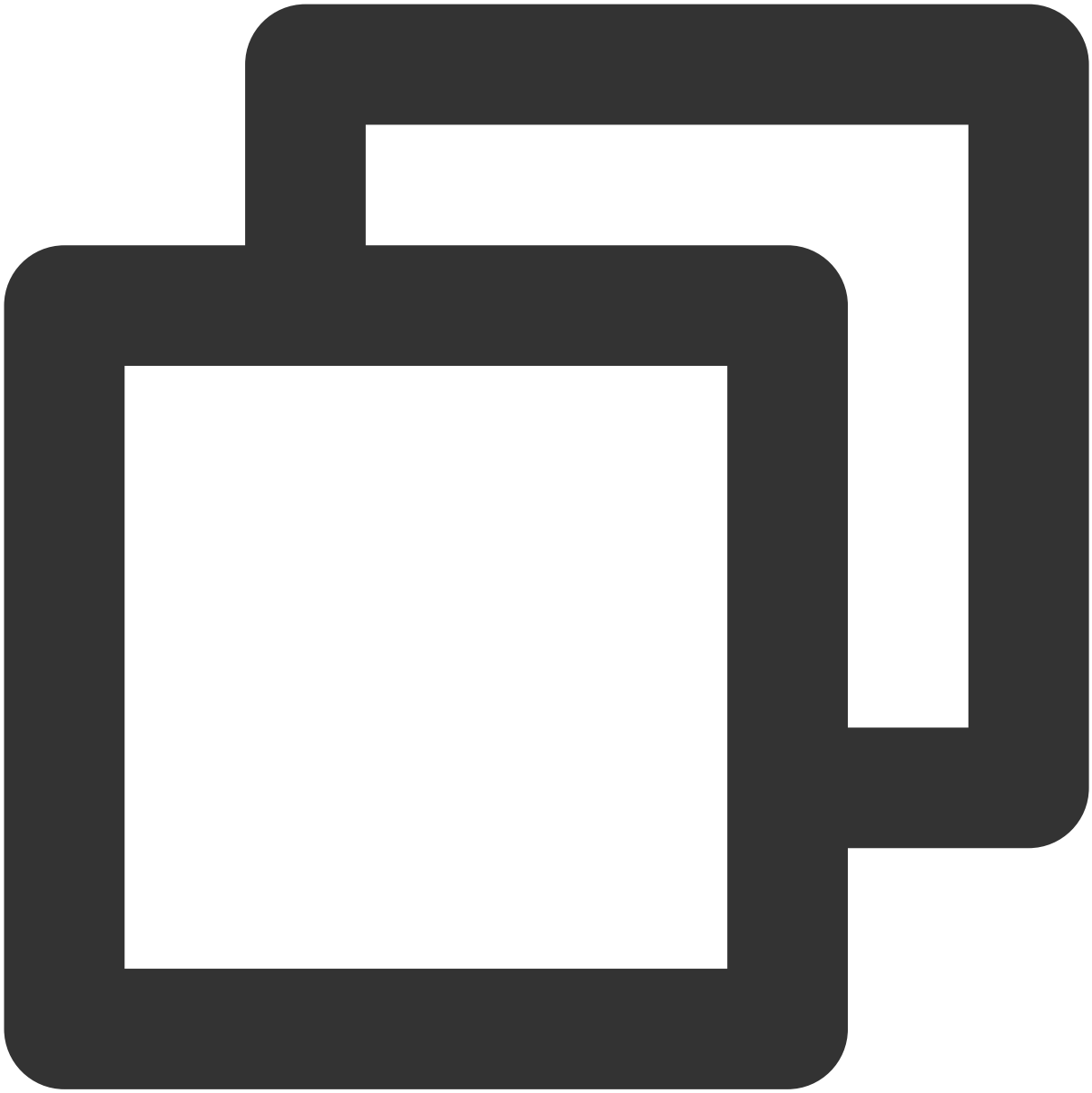
```
Complete!
[root@VM-0-10-centos ~]#
```

4. 다음 명령어를 실행하여 Nginx 버전이 정상적으로 조회되는지 검증합니다.



```
nginx -v
```

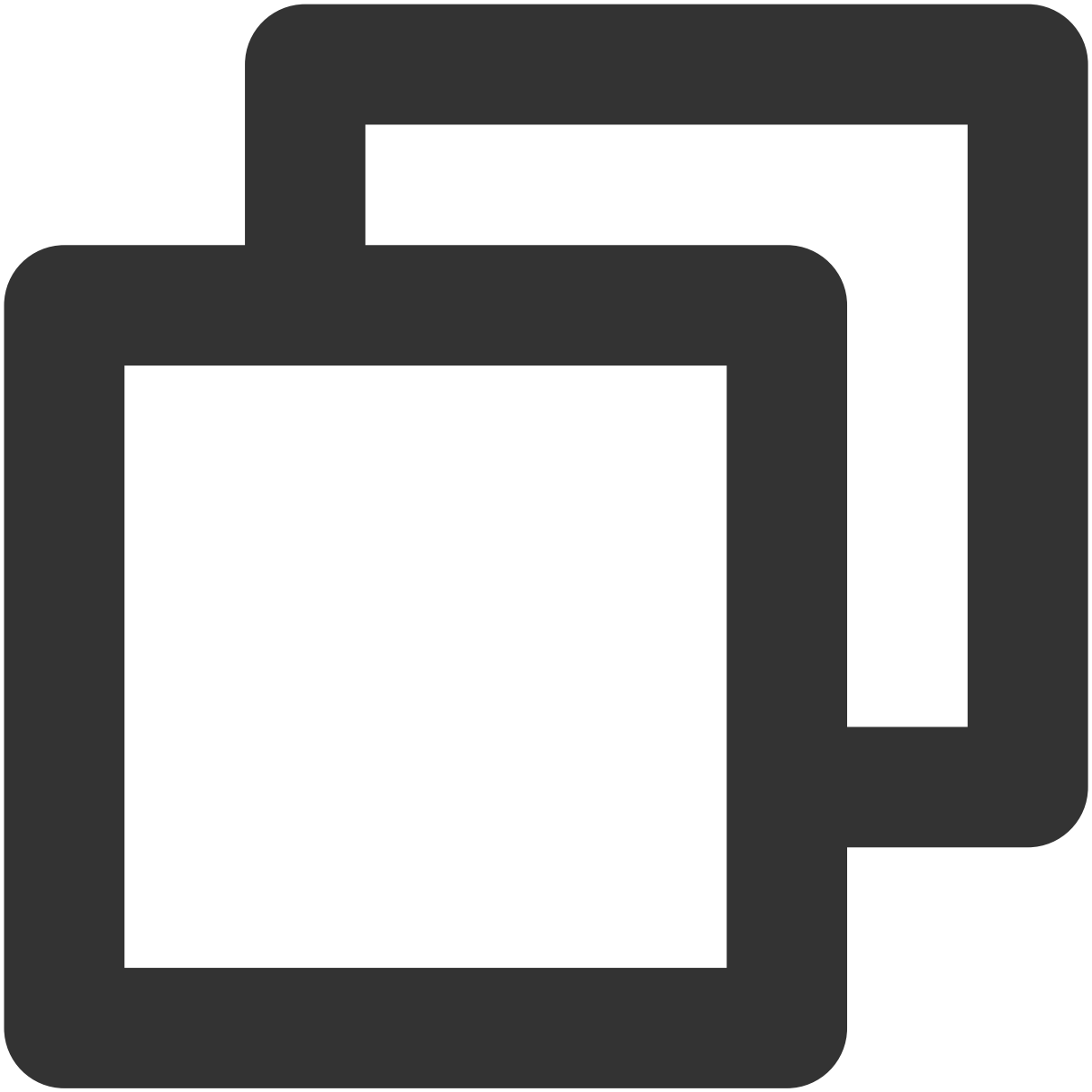
다음과 같은 정보가 뜨면 설치 검증이 끝난 것입니다.



```
nginx version: nginx/1.16.1
```

PHP 설치

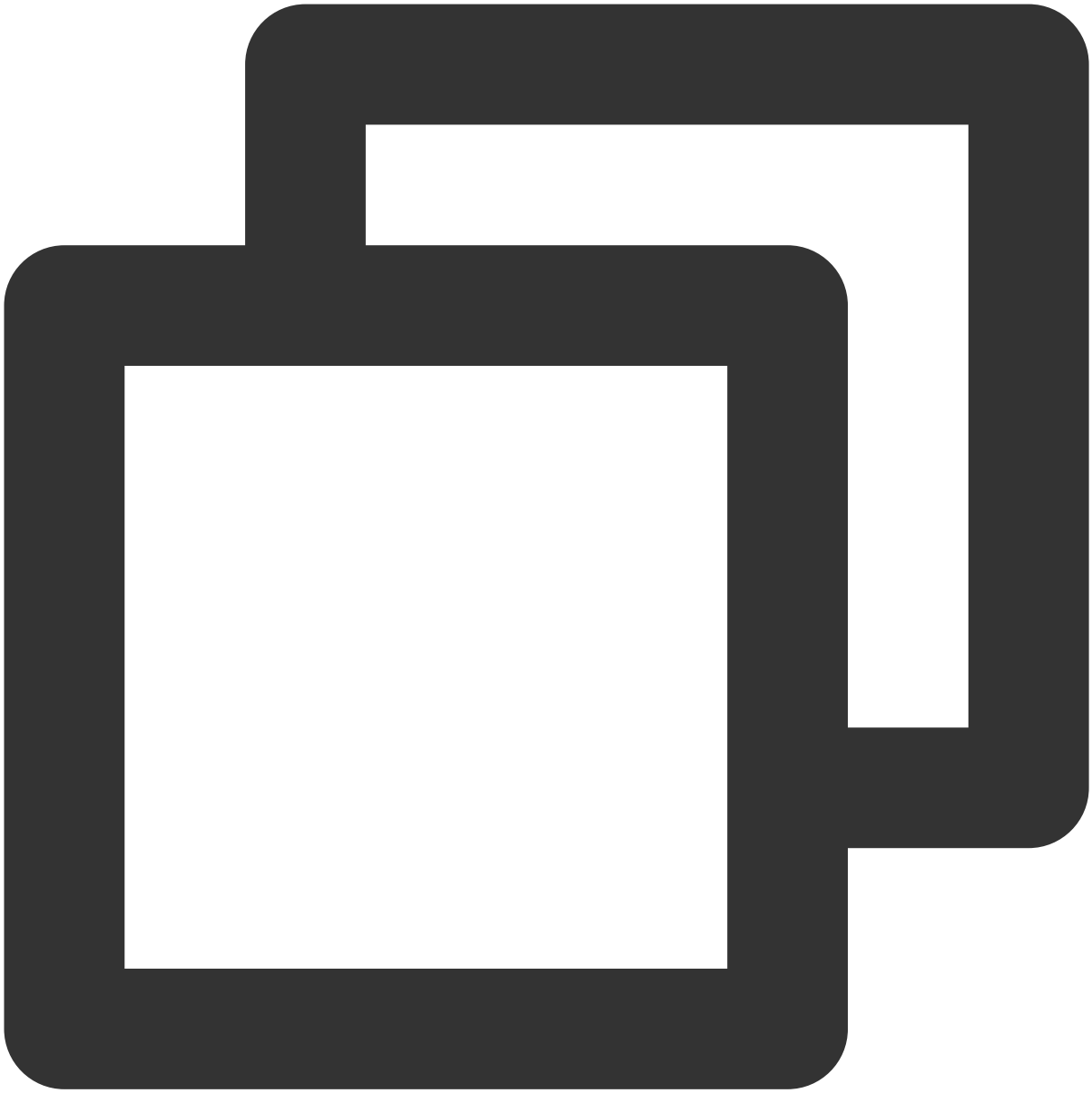
1. SSH 툴을 사용하여 구매한 서버에 로그인합니다.
2. 다음 명령어를 실행하여 PHP 7.4를 설치합니다.



```
yum install epel-release yum-utils
```

3. 다음 명령을 순서대로 실행합니다.

명령어 1:

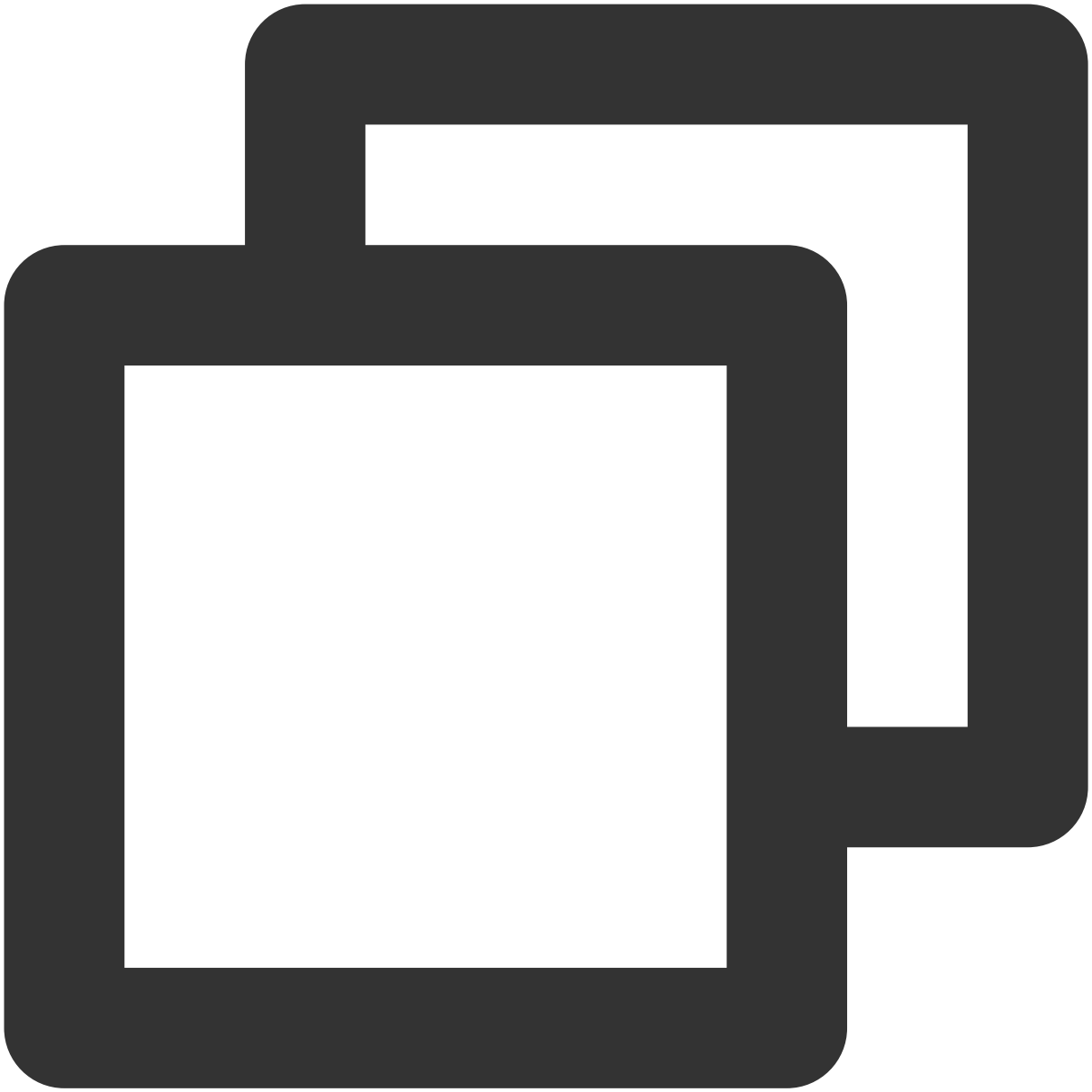


```
yum install http://rpms.remirepo.net/enterprise/remi-release-7.rpm
```

설명：

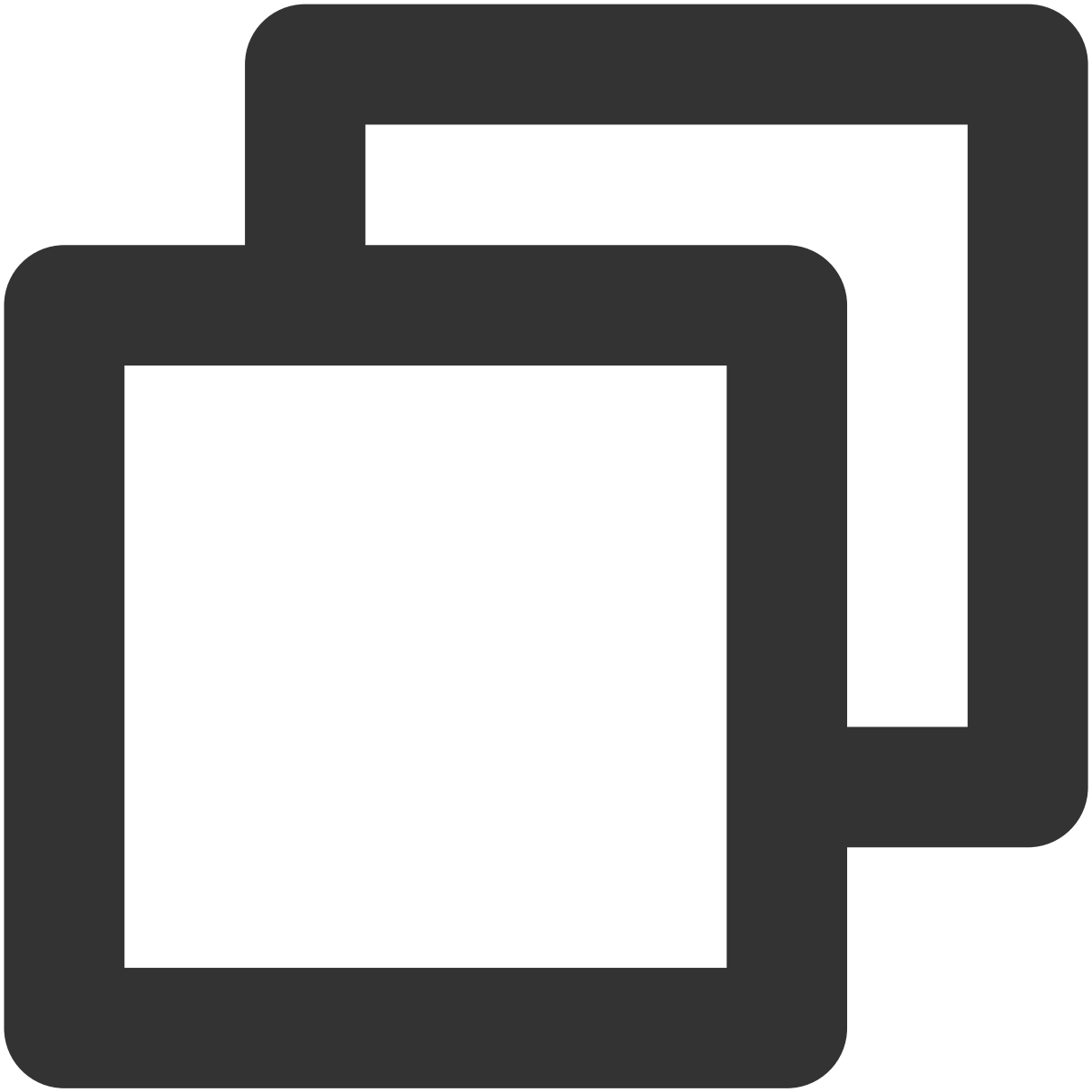
명령 실행이 너무 느리거나 오랫동안 중단된 경우 `Ctrl+C` 를 눌러 실행을 취소하고 명령을 다시 실행할 수 있습니다(이하 동일).

명령어 2:



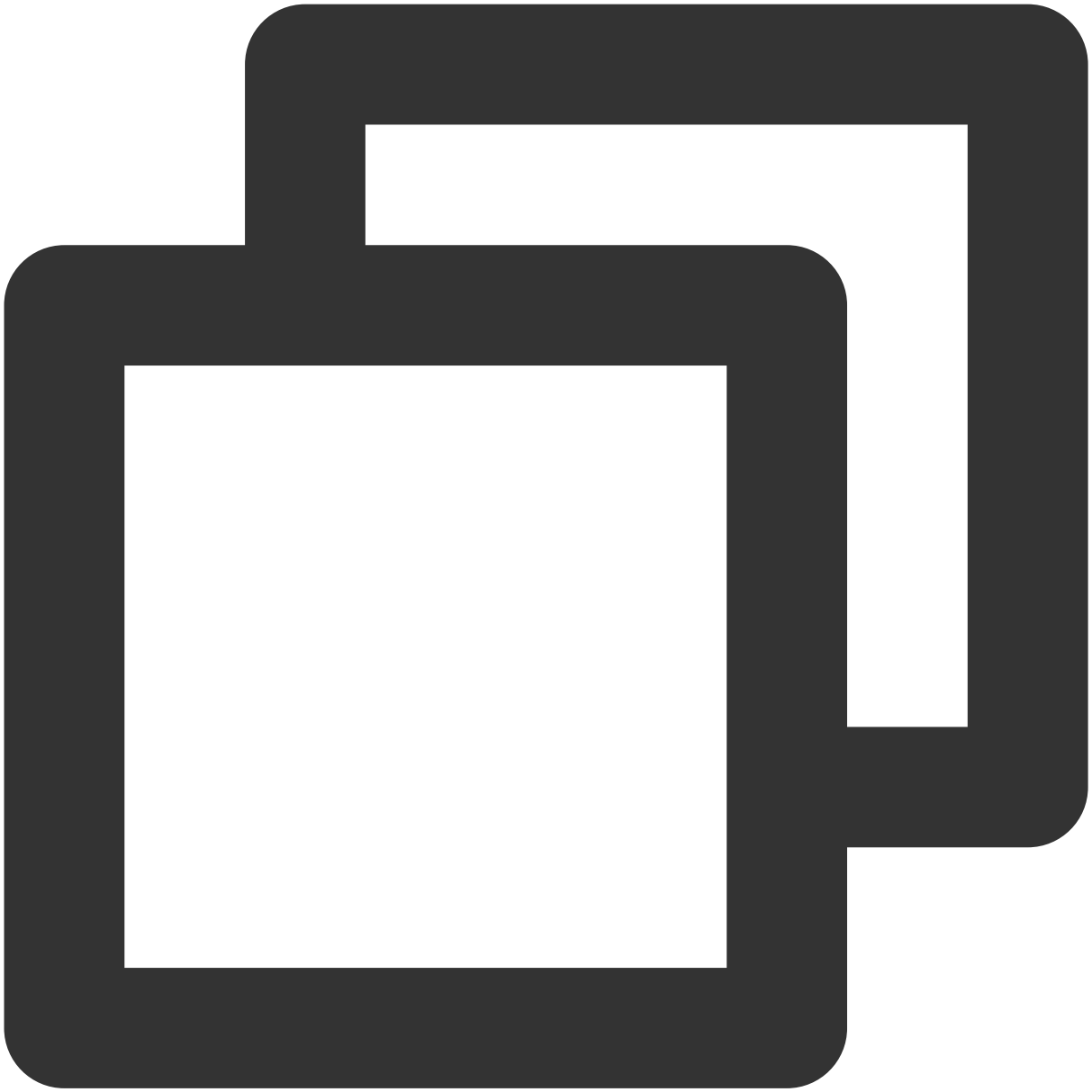
```
yum-config-manager --enable remi-php74
```

명령어 3:



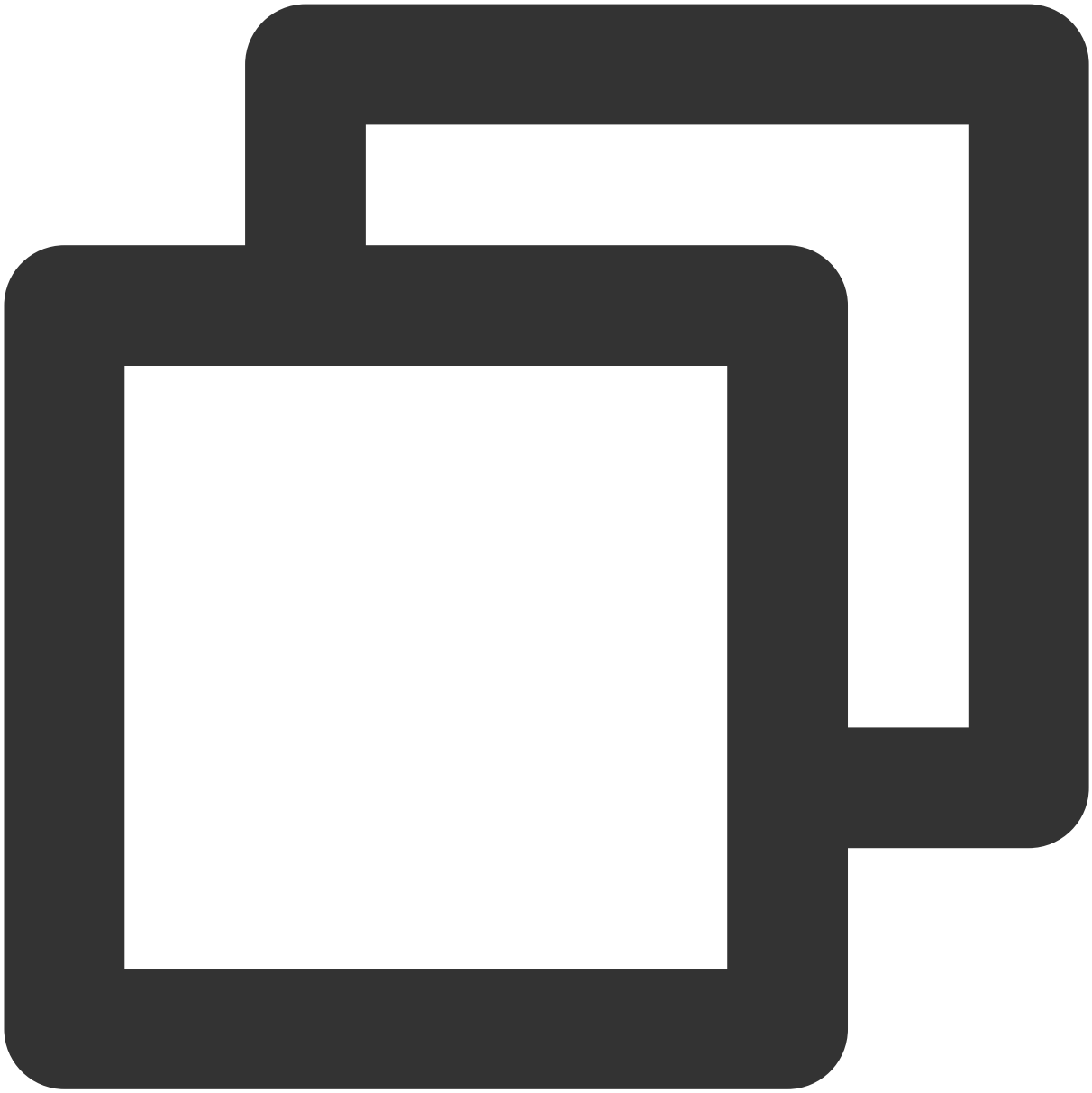
```
yum install php php-fpm
```

4. 설치 완료 후 다음 명령어를 실행하여 PHP 버전이 정상적으로 조회되는지 검증합니다.



```
php -v
```

다음과 같은 정보가 뜨면 설치 검증이 끝난 것입니다.

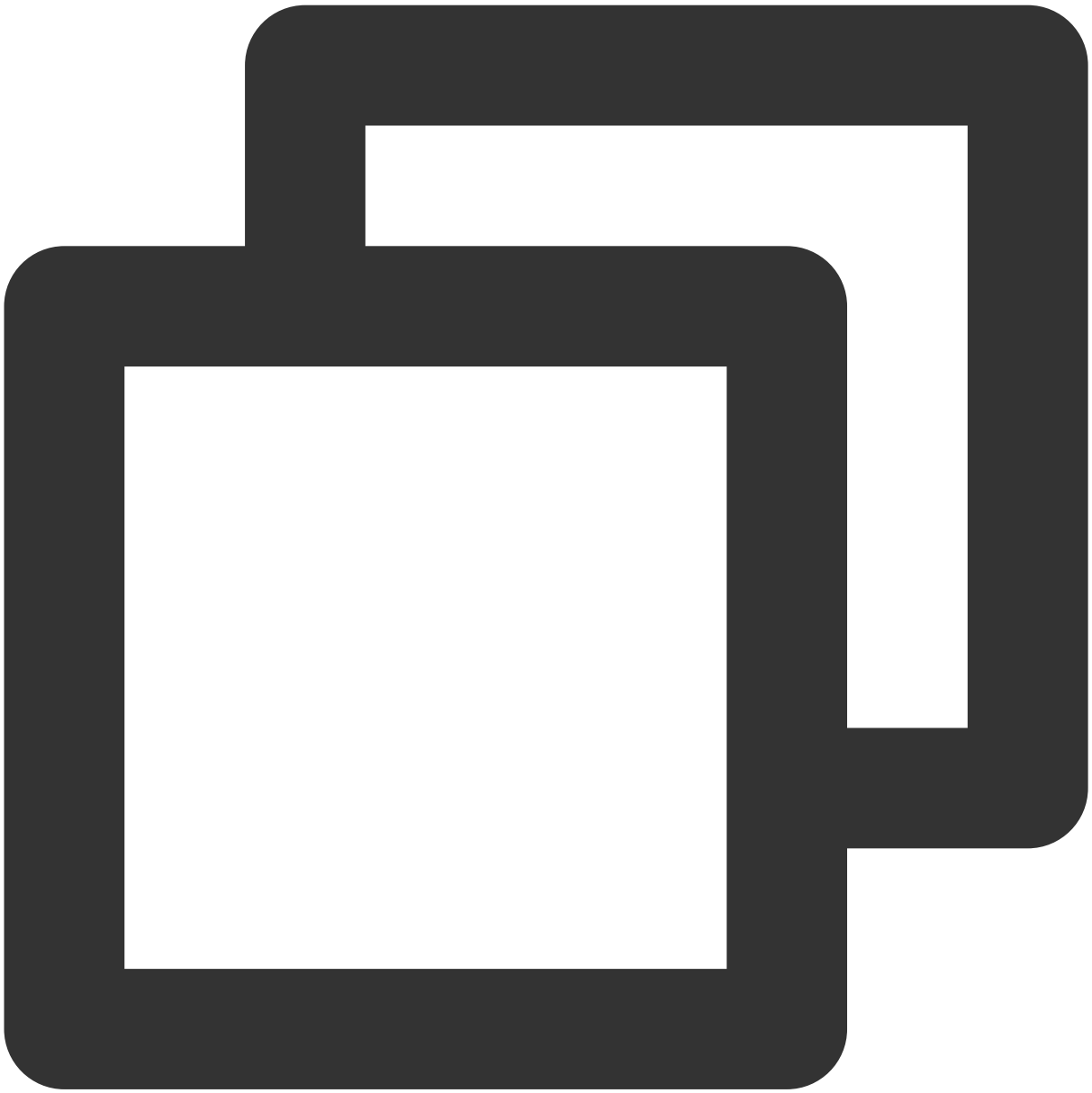


```
PHP 7.4.8 (cli) (built: Jul 9 2020 08:57:23) ( NTS )
Copyright (c) The PHP Group
Zend Engine v3.4.0, Copyright (c) Zend Technologies
```

PHP 모듈 설치

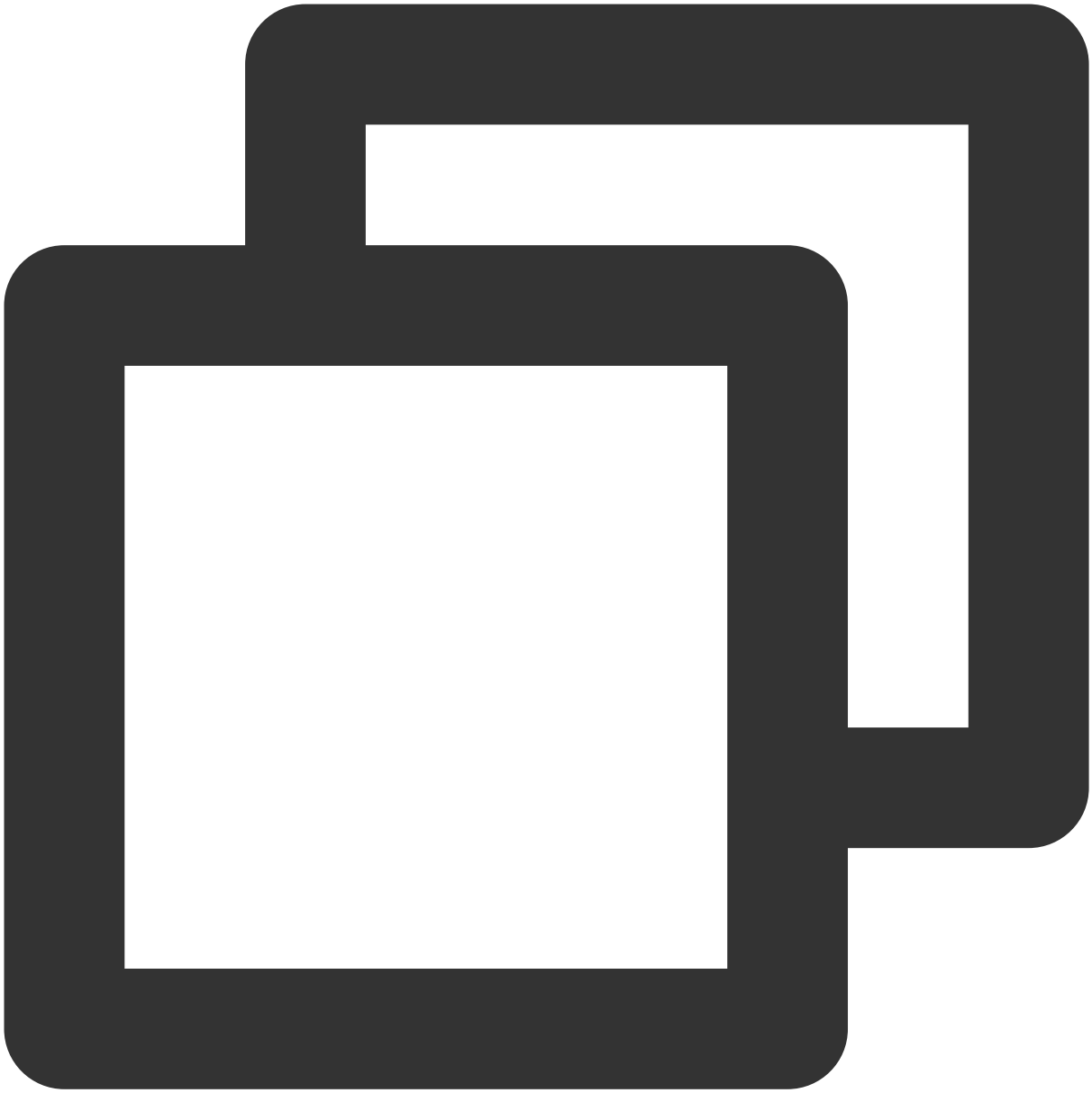
NextCloud는 기본적인 PHP 외에도 다른 PHP 모듈에 종속되어 일부 기능을 실행합니다. NextCloud의 종속에 관련된 자세한 모듈 정보는 NextCloud 공식 홈페이지 문서를 참조하십시오.
본 튜토리얼에는 NextCloud 필수 PHP 모듈이 설치됩니다. NextCloud의 다른 옵션 기능을 사용할 계획이라면 종속된 기타 PHP 모듈에 주의하여 설치하십시오.

1. SSH 툴을 사용하여 구매한 서버에 로그인합니다.
2. 다음 명령어를 실행하여 PHP 모듈을 설치합니다.



```
yum install php-xml php-gd php-mbstring php-mysqlnd php-intl php-zip
```

3. 설치 완료 후 다음 명령어를 실행하여 설치된 PHP 모듈을 조회합니다.



```
php -m
```

4. 기타 모듈 설치가 필요한 경우 `yum install <php-module-name>` 을 다시 실행하면 됩니다.

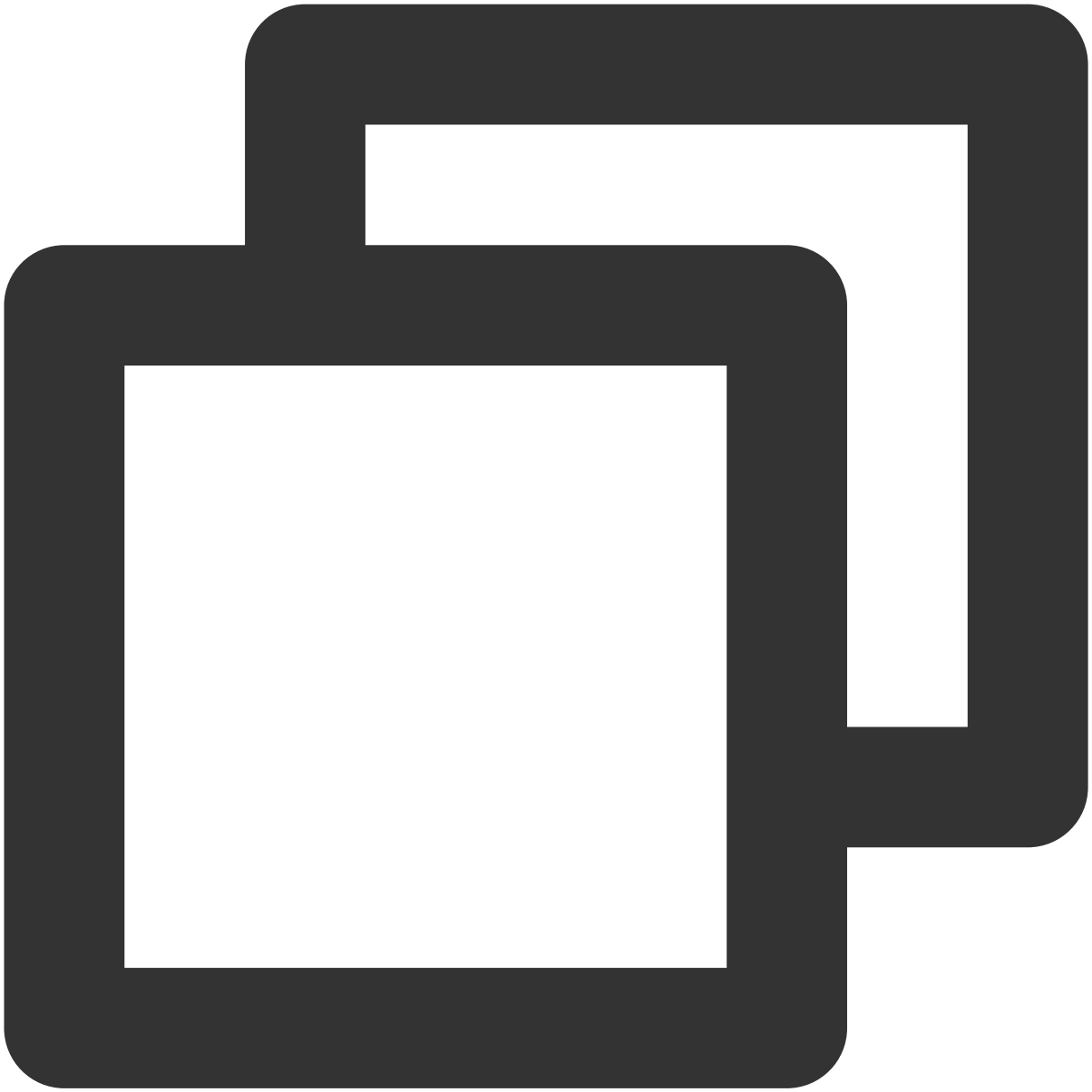
NextCloud 서버 코드 업로드 및 압축 해제

1. NextCloud 공식 홈페이지에서 NextCloud 서버 설치 패키지 최신 버전을 다운로드한 다음 서버 `/var/www/` 디렉터리에 업로드합니다. 다음과 같은 방법으로 업로드할 수 있습니다.
1. `wget` 명령어를 사용하여 서버에서 직접 설치 패키지를 다운로드합니다. 예: `/var/www/` 디렉터리로 이동 후 명령어 `wget https://download.nextcloud.com/server/releases/nextcloud-19.0.1.zip` 실행

2. 로컬 컴퓨터에 다운로드한 후 SFTP 또는 SCP 등 소프트웨어를 통해 `/var/www/` 디렉터리에 설치 패키지를 업로드합니다.
3. 로컬 컴퓨터에 다운로드한 후 lrzsz를 사용하여 업로드합니다. 방법은 다음과 같습니다.
3.1 SSH 툴을 사용하여 구매한 서버에 로그인합니다.
3.2 `yum install lrzsz` 를 실행하여 lrzsz를 설치합니다.
3.3 `cd /var/www/` 를 실행하여 타깃 디렉터리로 이동합니다.
3.4 `rz -bye` 를 실행하고 SSH 툴에서 로컬로 다운로드할 NextCloud 서버 설치 패키지를 선택합니다(해당 조작은 SSH 툴에 따라 상이함).
4. SSH 툴을 사용하여 구매한 서버에 로그인합니다.
5. `unzip nextcloud-<version>.zip` 을 실행하여 설치 패키지를 압축 해제합니다(예: `unzip nextcloud-19.0.1.zip`).

PHP 설정

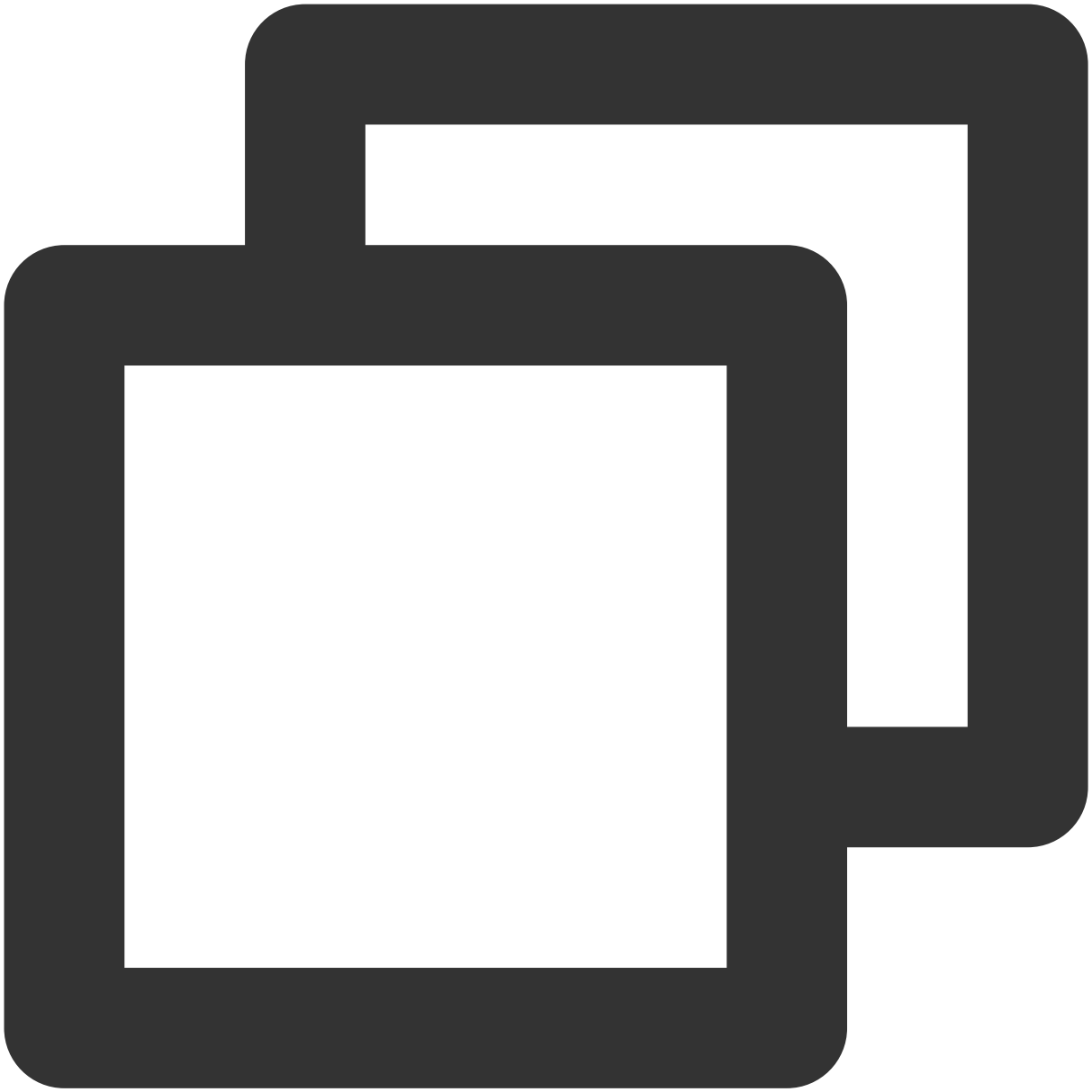
1. SSH 툴을 사용하여 구매한 서버에 로그인합니다.
2. `vim /etc/php-fpm.d/www.conf` 를 실행하여 PHP-FPM 구성 파일을 열고 순서대로 설정 항목을 수정합니다. vim의 구체적인 사용법은 관련 자료를 참조하십시오. 다른 방식을 사용해 해당 구성 파일을 수정할 수도 있습니다.
1. `user = apache` 를 `user = nginx` 로 수정합니다.
2. `group = apache` 를 `group = nginx` 로 수정합니다.
3. 수정 완료 후 `:wq` 를 입력하여 파일을 저장하고 종료합니다. vim의 구체적인 작업 가이드는 관련 문서를 참조하십시오.
4. 다음 명령어를 실행하여 PHP가 Nginx 사용에 최적화되도록 디렉터리 소유자를 수정합니다.



```
chown -R nginx:nginx /var/lib/php
```

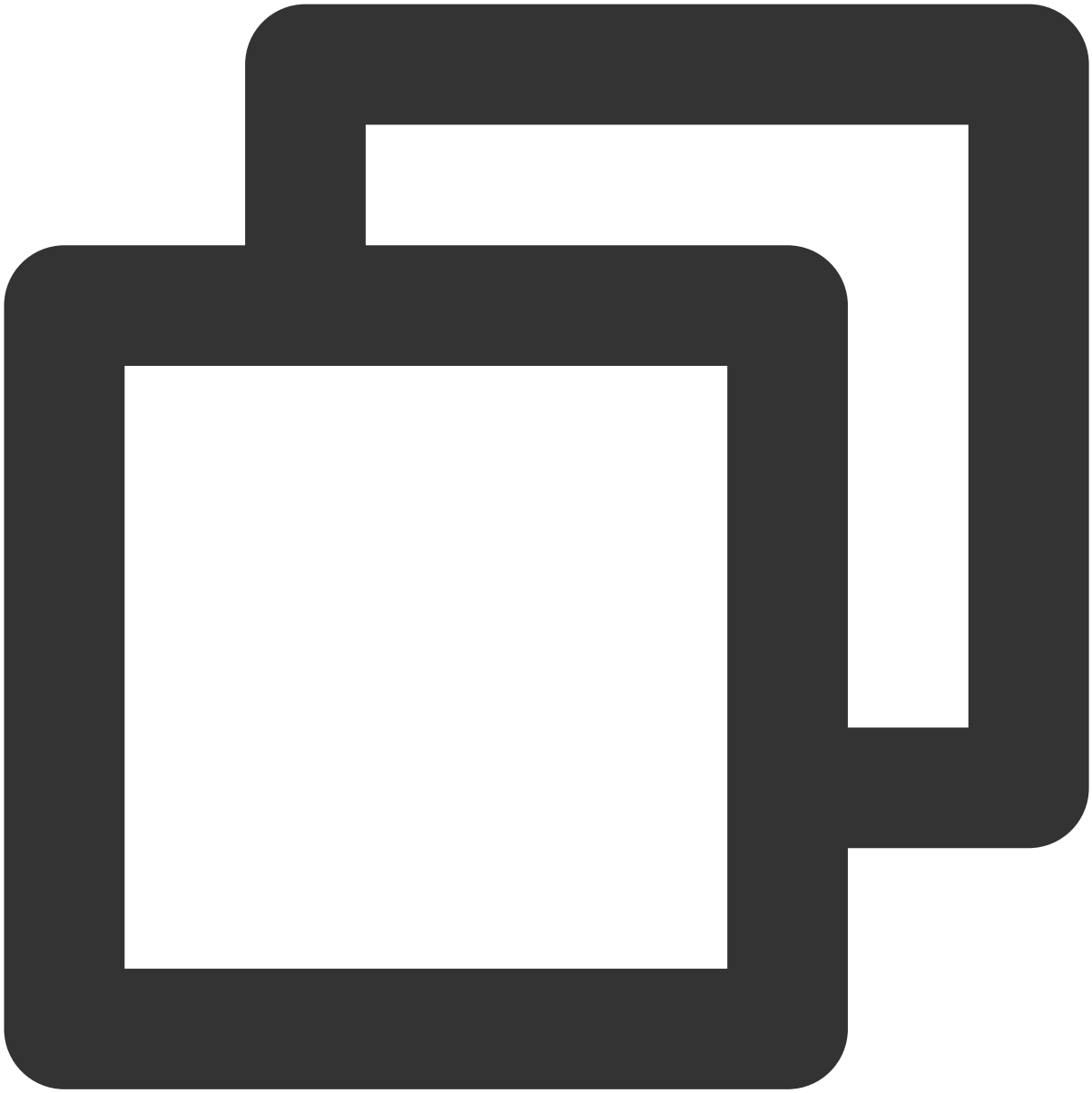
5. 계속해서 다음 명령어를 실행하고 PHP-FPM 서비스를 실행합니다.

명령어 1:



```
systemctl enable php-fpm   # 명령어 1
```

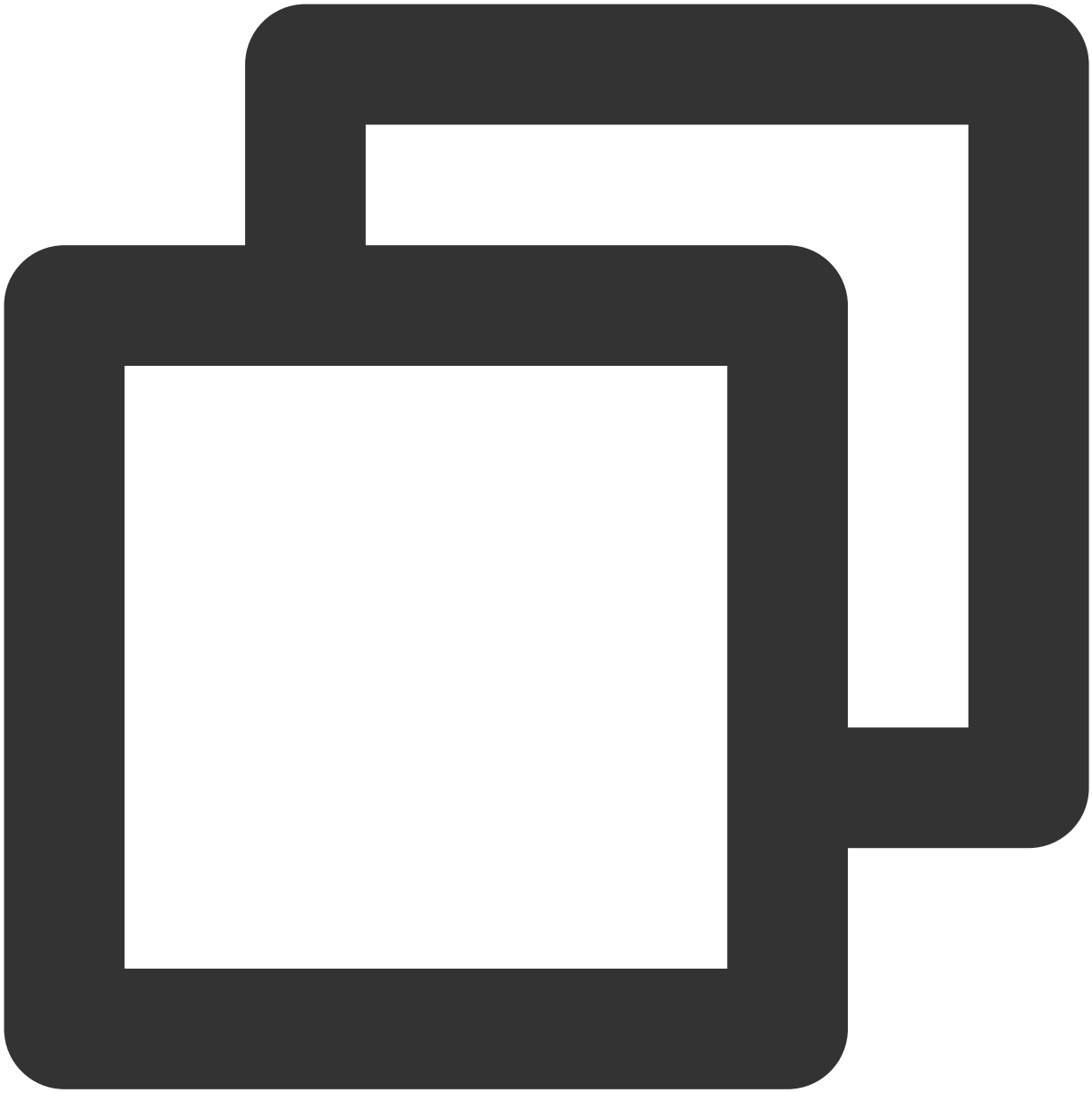
명령어 2:



```
systemctl start php-fpm   # 명령어 2
```

Nginx 설정

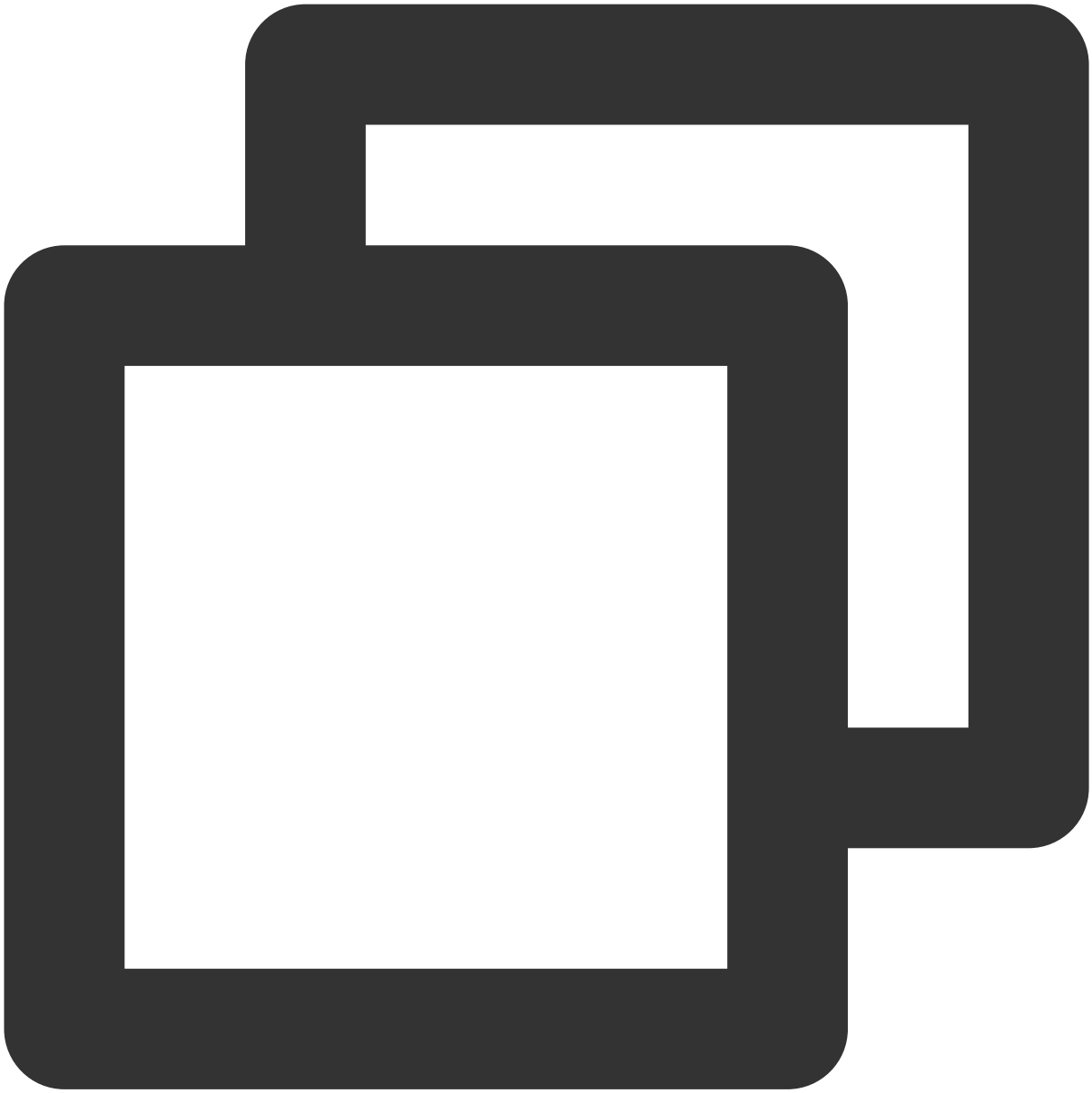
1. SSH 툴을 사용하여 구매한 서버에 로그인합니다.
2. 다음 명령어를 실행하여 웹 사이트 디렉터리 소유자를 수정합니다.



```
chown -R nginx:nginx /var/www
```

3. 기존의 Nginx 구성 파일 `/etc/nginx/nginx.conf` 를 백업합니다. 다음 조작이 가능합니다.

1. `cp /etc/nginx/nginx.conf ~/nginx.conf.bak` 를 실행하여 기존 구성 파일을 홈(HOME) 디렉터리에 백업합니다.
2. SFTP 또는 SCP 등 소프트웨어를 사용하여 기존 구성 파일을 로컬 컴퓨터에 다운로드합니다.
3. `/etc/nginx/nginx.conf` 를 다음 내용으로 수정 또는 대체합니다.



```
# For more information on configuration, see:
#   * Official English Documentation: http://nginx.org/en/docs/
#   * Official Russian Documentation: http://nginx.org/ru/docs/

user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;

# Load dynamic modules. See /usr/share/doc/nginx/README.dynamic.
include /usr/share/nginx/modules/*.conf;
```

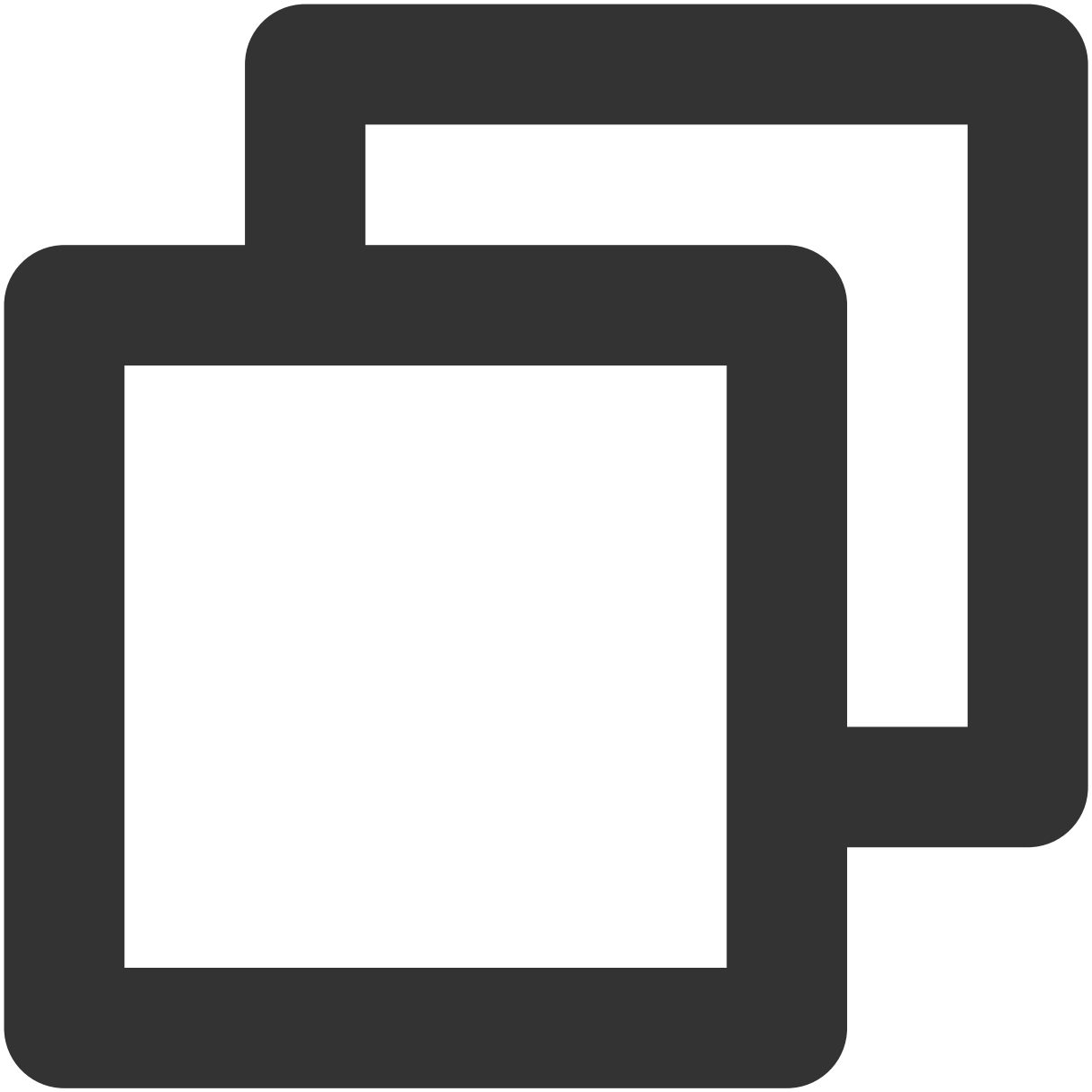
```
events {
    worker_connections 1024;
}

http {
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile            on;
    tcp_nopush          on;
    tcp_nodelay         on;
    keepalive_timeout   65;
    types_hash_max_size 2048;

    include             /etc/nginx/mime.types;
    default_type        application/octet-stream;

    # Load modular configuration files from the /etc/nginx/conf.d directory.
    # See http://nginx.org/en/docs/ngx_core_module.html#include
    # for more information.
    include /etc/nginx/conf.d/*.conf;

    server {
        listen       80 default_server;
        listen       [::]:80 default_server;
        server_name  _;
        root         /var/www/nextcloud;

        add_header Referrer-Policy "no-referrer" always;
        add_header X-Content-Type-Options "nosniff" always;
        add_header X-Download-Options "noopen" always;
        add_header X-Frame-Options "SAMEORIGIN" always;
        add_header X-Permitted-Cross-Domain-Policies "none" always;
        add_header X-Robots-Tag "none" always;
        add_header X-XSS-Protection "1; mode=block" always;

        client_max_body_size 512M;
        fastcgi_buffers 64 4K;

        gzip on;
        gzip_vary on;
        gzip_comp_level 4;
        gzip_min_length 256;
```

```
        gzip_proxied expired no-cache no-store private no_last_modified no_etag aut
        gzip_types application/atom+xml application/javascript application/json app

        # Load configuration files for the default server block.
        include /etc/nginx/default.d/*.conf;

        location / {
            try_files $uri $uri/ =404;
            index index.php;
        }

        location ~ ^\\/(?:build|tests|config|lib|3rdparty|templates|data)\\/ {
            deny all;
        }
        location ~ ^\\/(?:\\.|autotest|occ|issue|indie|db_|console) {
            deny all;
        }

        location ~ ^\\/(?:index|remote|public|cron|core\\/ajax\\/update|status|ocs\
            fastcgi_split_path_info ^(.+?\\.php)(\\/.*|)$;
            set $path_info $fastcgi_path_info;
            try_files $fastcgi_script_name =404;
            include        fastcgi_params;
            fastcgi_param  SCRIPT_FILENAME  $document_root$fastcgi_script_name;
            fastcgi_param PATH_INFO $path_info;
            fastcgi_param modHeadersAvailable true;
            fastcgi_pass   127.0.0.1:9000;
            fastcgi_intercept_errors on;
            fastcgi_request_buffering off;
        }

        location ~ ^\\/(?:updater|oc[ms]-provider)(?:$|\\/) {
            try_files $uri/ =404;
            index index.php;
        }

        location ~ \\.(css|js|svg|gif)$ {
            add_header Cache-Control "max-age=15778463";
        }

        location ~ \\.woff2?$ {
            add_header Cache-Control "max-age=604800";
        }
    }

# Settings for a TLS enabled server.
#
```

```
#    server {
#        listen       443 ssl http2 default_server;
#        listen       [::]:443 ssl http2 default_server;
#        server_name  _;
#        root         /usr/share/nginx/html;
#
#        ssl_certificate "/etc/pki/nginx/server.crt";
#        ssl_certificate_key "/etc/pki/nginx/private/server.key";
#        ssl_session_cache shared:SSL:1m;
#        ssl_session_timeout  10m;
#        ssl_ciphers HIGH:!aNULL:!MD5;
#        ssl_prefer_server_ciphers on;
#
#        # Load configuration files for the default server block.
#        include /etc/nginx/default.d/*.conf;
#
#        location / {
#        }
#
#        error_page 404 /404.html;
#            location = /40x.html {
#        }
#
#        error_page 500 502 503 504 /50x.html;
#            location = /50x.html {
#        }
#    }

}
```

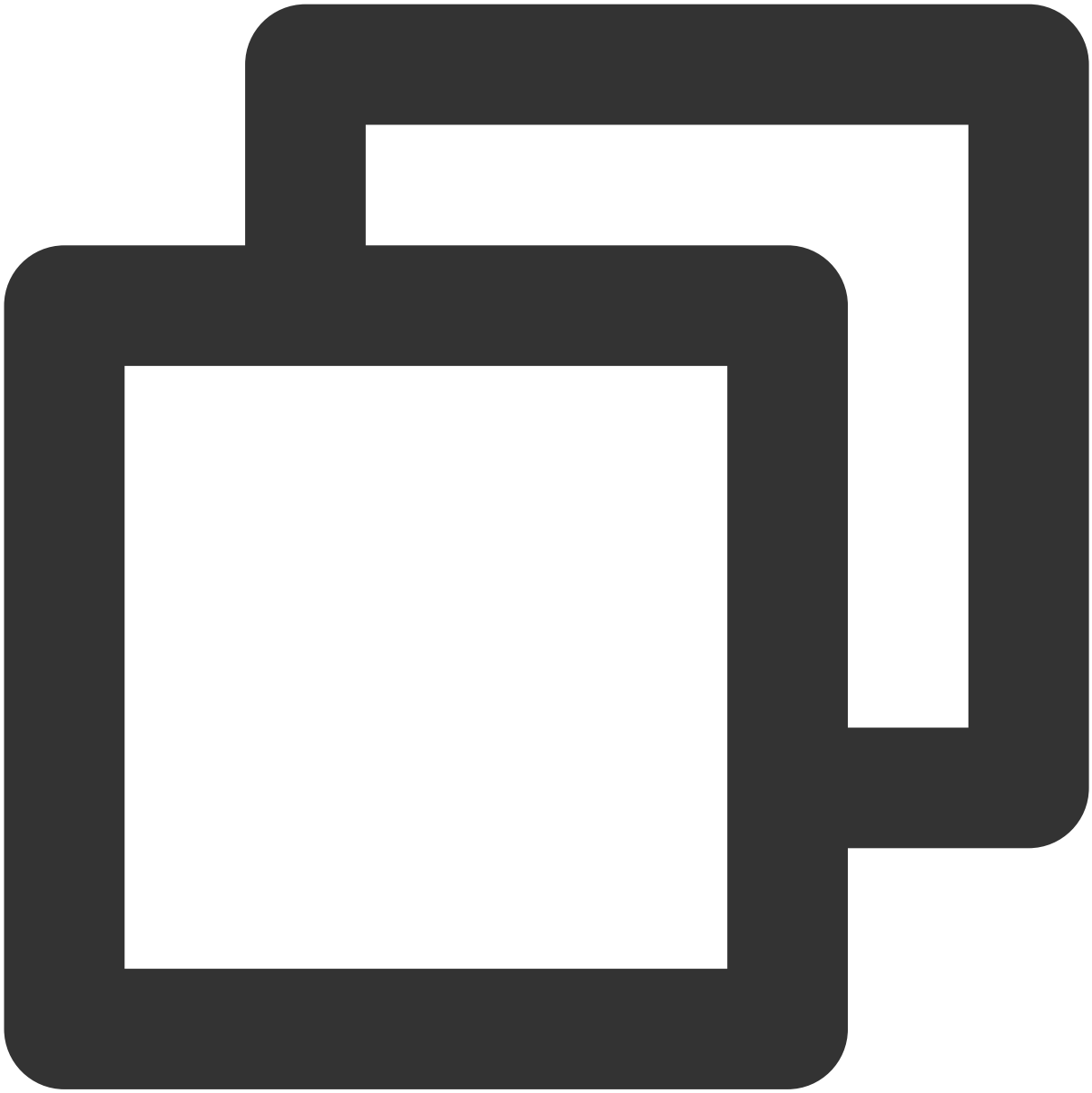
4. 계속해서 다음 명령어를 실행하고 Nginx 서비스를 실행합니다.

명령어 1:



```
systemctl enable nginx
```

명령어 2:

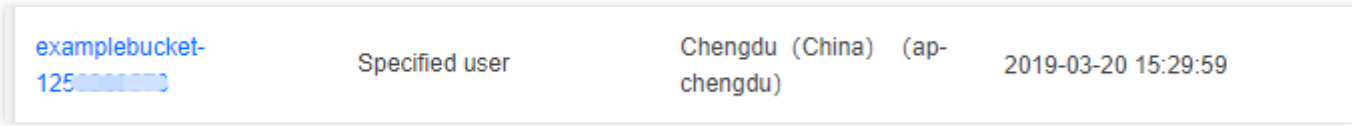


```
systemctl start nginx
```

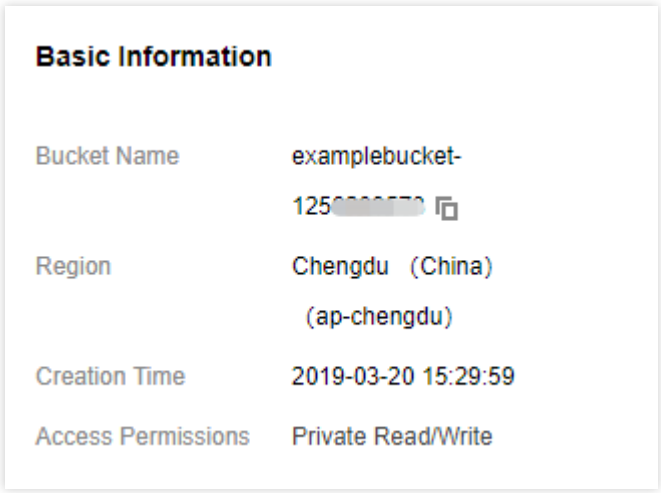
COS를 사용하도록 NextCloud 서버 구성

COS 정보 가져오기

1. COS 콘솔에 로그인합니다.
2. 이전에 생성한 버킷을 찾아 버킷 이름을 클릭합니다.



3. 왼쪽 사이드바에서 **개요**를 선택하고 **기본 정보**의 **버킷 이름** 및 **리전**에 정보를 기록합니다.

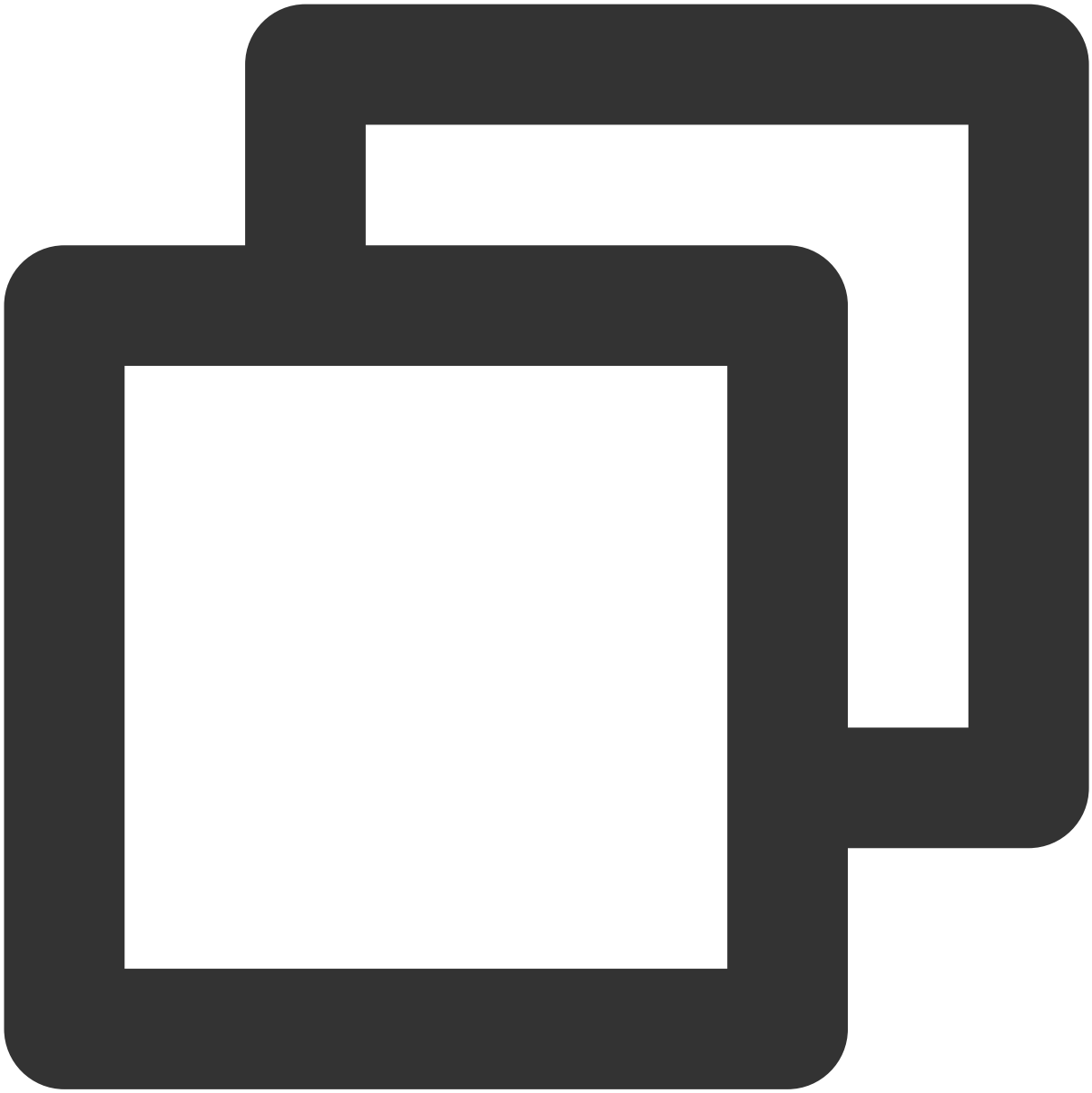


API 키 획득

리스크를 줄이기 위해 서브 계정 키를 사용하고 최소 권한의 원칙 설명을 따르는 것이 좋습니다. 서브 계정 키를 얻는 방법에 대한 자세한 내용은 Access Key를 참고하십시오.

NextCloud 서버 구성 파일 수정

1. 텍스트 편집 툴을 사용하여 `config.php` 를 생성하고 다음 내용을 입력한 후 주석에 따라 관련 값을 수정합니다.

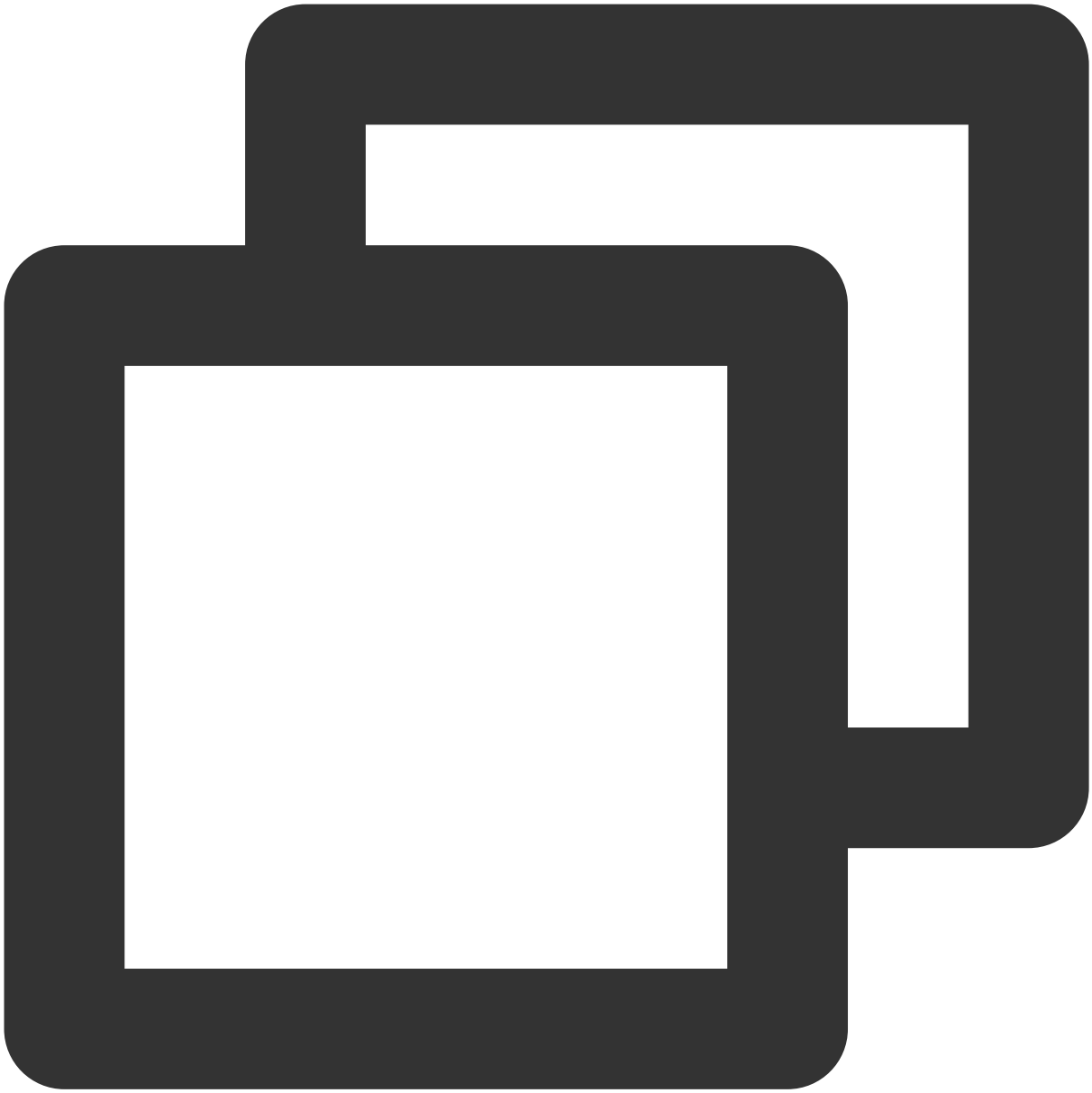


```
<?php
$CONFIG = array(
  'objectstore' => array(
    'class' => '\\\\OC\\\\Files\\\\ObjectStore\\\\S3',
    'arguments' => array(
      'bucket' => 'nextcloud-1250000000', // 버킷 이름(공간의 이름)
      'autocreate' => false,
      'key'  => 'AKIDxxxxxxxx', // 사용자의 SecretId로 변경
      'secret' => 'xxxxxxxxxxxx', // 사용자의 SecretKey로 변경
      'hostname' => 'cos.<Region>.myqcloud.com', // <Region>을 소속 리전으로 수정(예: a
      'use_ssl' => true,
```

```
    ),
  ),
);
```

다음 이미지 참고:

2. 해당 파일을 저장하고 `/var/www/nextcloud/config/` 디렉터리에 업로드합니다(파일 이름은 `config.php` 로 유지). SFTP 또는 SCP 소프트웨어를 통해 파일을 업로드할 수 있고, `rz -bye` 명령어를 통해 업로드할 수도 있습니다.
2. 다음 명령어를 실행하여 구성 파일의 소유자를 수정합니다.



```
chown nginx:nginx /var/www/nextcloud/config/config.php
```

도메인 이름 구성

NextCloud 서버에 IP 주소가 아닌 도메인 이름을 사용하려면, 도메인 이름을 등록하고 CVM IP 주소로 리졸브하고, 도메인 등록 기관의 안내 문서에 따라 ICP 비안을 받을 수 있습니다.

NextCloud 서버는 설치 시 사용한 도메인 이름이나 IP 주소를 기록하므로 설치 전에 도메인 이름을 등록, 리졸브 및 ICP 비안을 받아 도메인 이름을 사용하여 NextCloud 서버의 보안 페이지로 이동하는 것이 좋습니다.
NextCloud 서버 설치 완료 후 도메인 또는 IP 주소를 변경해야 할 경우 직접 `/var/www/nextcloud/config/config.php` 구성 파일의 `trusted_domains` 를 수정할 수 있습니다. 세부 사항은 NextCloud 공식 홈페이지 문서를 참조하십시오.

NextCloud 서버 설치

1. 브라우저를 사용하여 NextCloud 서버에 액세스한 후 관리자 및 사용자 이름과 비밀번호를 생성하고 기억합니다.
2. **스토리지와 데이터베이스**를 확장하고 다음과 같이 구성합니다.

구성 항목	값
데이터 폴더	/var/www/nextcloud/data(기본값 유지)
데이터베이스 구성	MySQL/MariaDB
데이터베이스 사용자	root
데이터베이스 비밀번호	TencentDB for MySQL 초기화 중에 입력한 root 사용자의 비밀번호
데이터베이스 이름	nextcloud(또는 사용하지 않는 다른 이름)
데이터베이스 호스트(기본적으로 localhost가 표시됨)	TencentDB for MySQL 인스턴스의 사설망 주소

3. **설치 완료**를 클릭하고 NextCloud 서버 설치가 완료될 때까지 기다립니다.4. 설치 과정에서 504 Gateway Timeout과 같은 오류 정보가 뜨는 경우 새로고침 후 다시 시도합니다.5. 설치 완료 후 관리자 계정을 사용해 NextCloud 서버에 로그인하면 웹 페이지 버전의 NextCloud를 사용할 수 있습니다.

NextCloud 서버 환경 최적화

백그라운드 작업

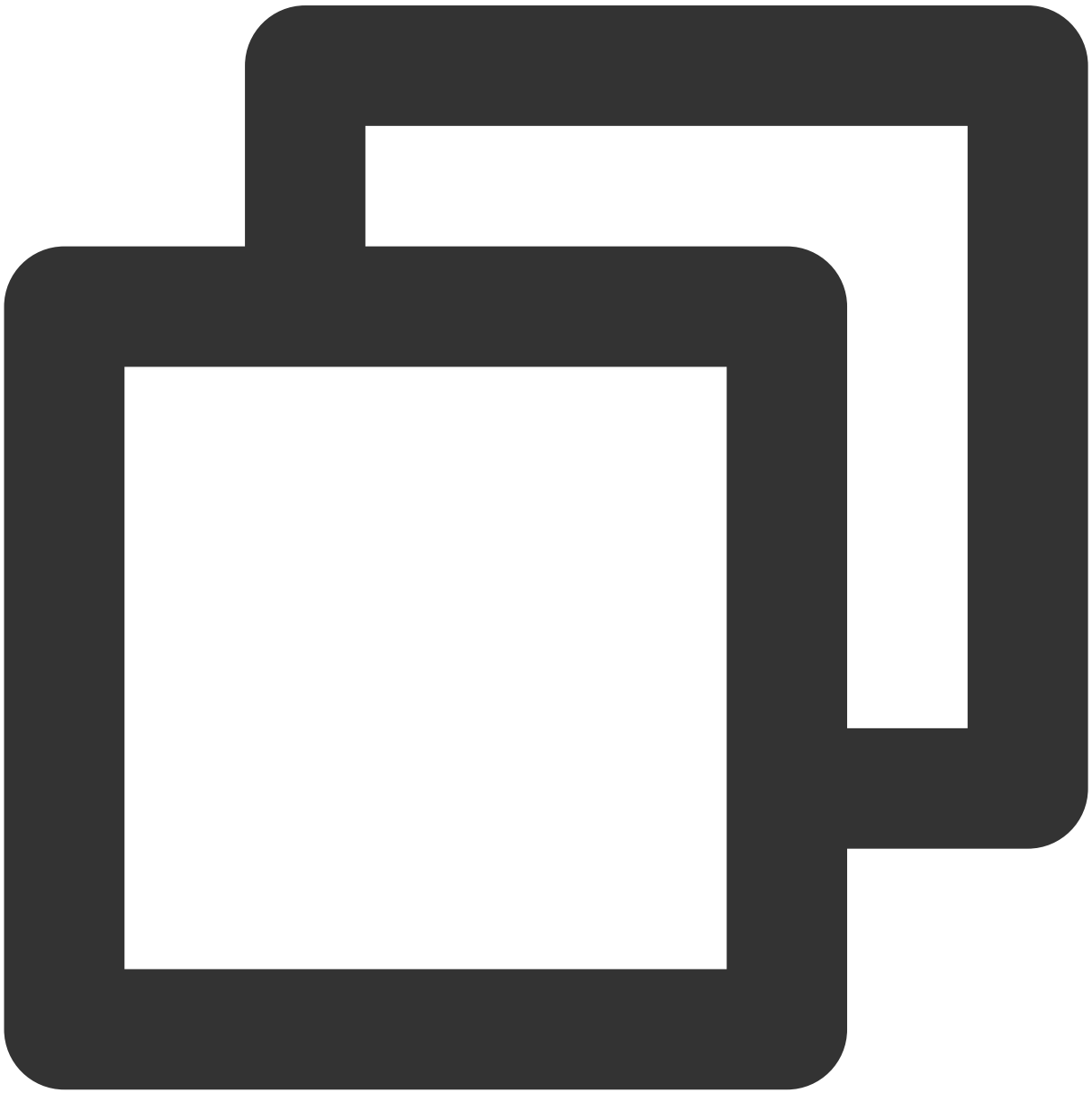
NextCloud 서버는 사용자와 상호작용이 필요하지 않을 때 데이터베이스 정리와 같은 백그라운드 작업을 실행하는 경우가 있습니다. PHP 실행 특성상 PHP 기반 프로그램은 내부적으로 하나의 독립된 작업 진행 또는 스레드를 유지할 수 없으므로 백그라운드 작업과 같은 시나리오는 외부에서 호출하여 대응하는 PHP 프로그램으로 실행해야 합니다. NextCloud 서버는 세 가지 백그라운드 작업 호출 방식을 제공합니다. 기본값은 웹 페이지에 로그인한 사용자를 통해 브라우저에서 자동으로 AJAX 요청을 보낸 후 서버의 백그라운드 작업 실행을 호출하는 방식으로, 사용자의 로그인

상태에 크게 의존합니다. 사용자가 로그인하지 않으면 이러한 백그라운드 작업은 실행될 수 없으므로 신뢰성이 가장 낮습니다.

AJAX 기반 백그라운드 작업의 신뢰성이 떨어지는 문제를 피하고자 Linux의 cron을 사용한 백그라운드 작업 설정을 권장합니다. Linux의 cron은 작업 호출 시간을 정확하게 제어합니다. 설정 가능한 Interval은 분, 시간, 한 달 중 특정 일, 월, 요일을 포함합니다. 예를 들어 5분마다(분 수는 5의 배수) 또는 매시 10분 등으로 설정할 수 있습니다. 일부 운영 체제는 초 단위, 연 단위도 지원하여 매우 효율적입니다. cron에 대한 설명과 설정은 관련 자료를 참조하십시오.

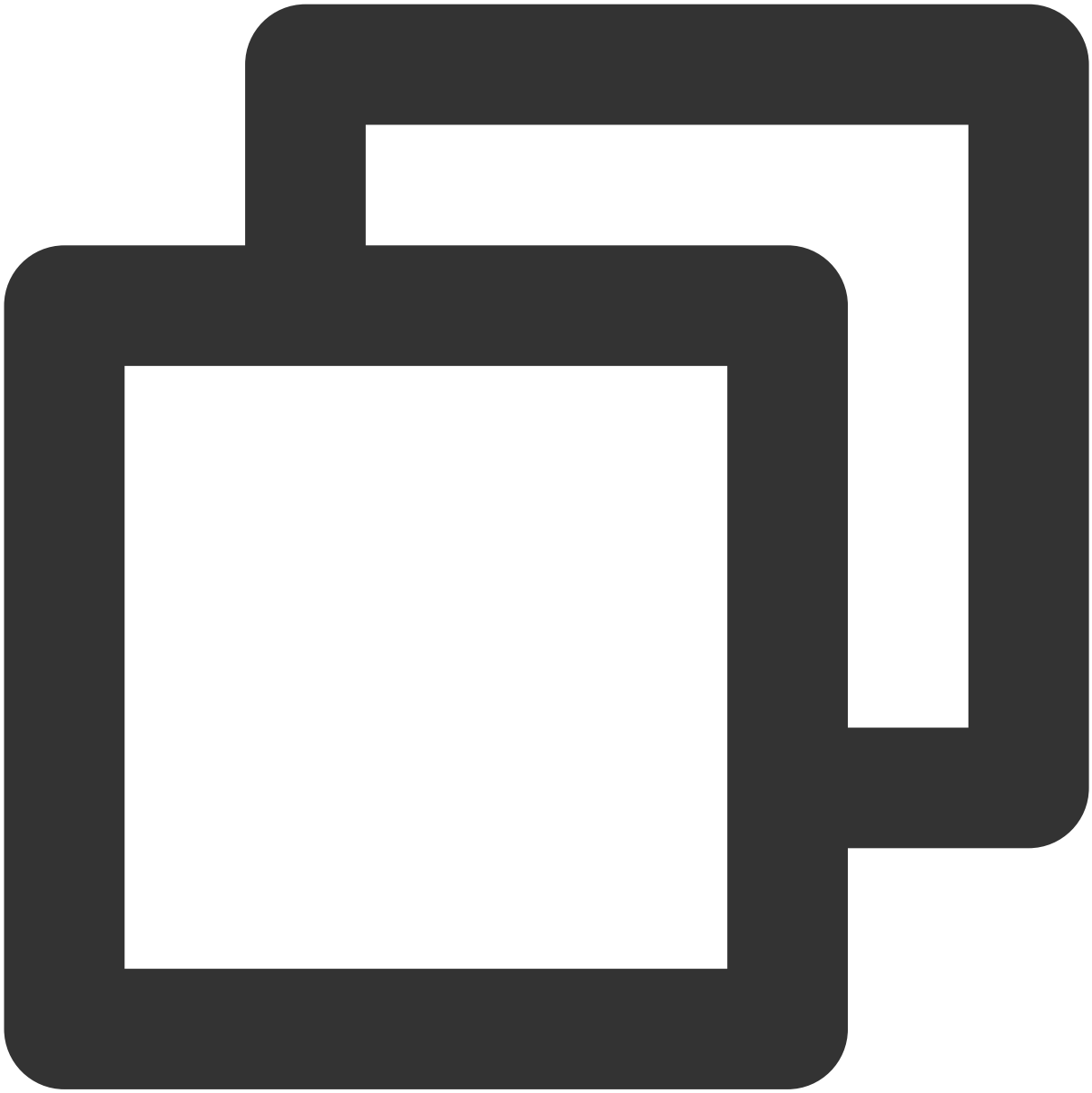
NextCloud 서버의 백그라운드 작업을 위해 cron을 설정하는 방법을 소개합니다.

1. SSH 툴을 사용하여 구매한 서버에 로그인합니다.
2. 다음 명령어를 실행하여 PHP 모듈을 설치합니다.



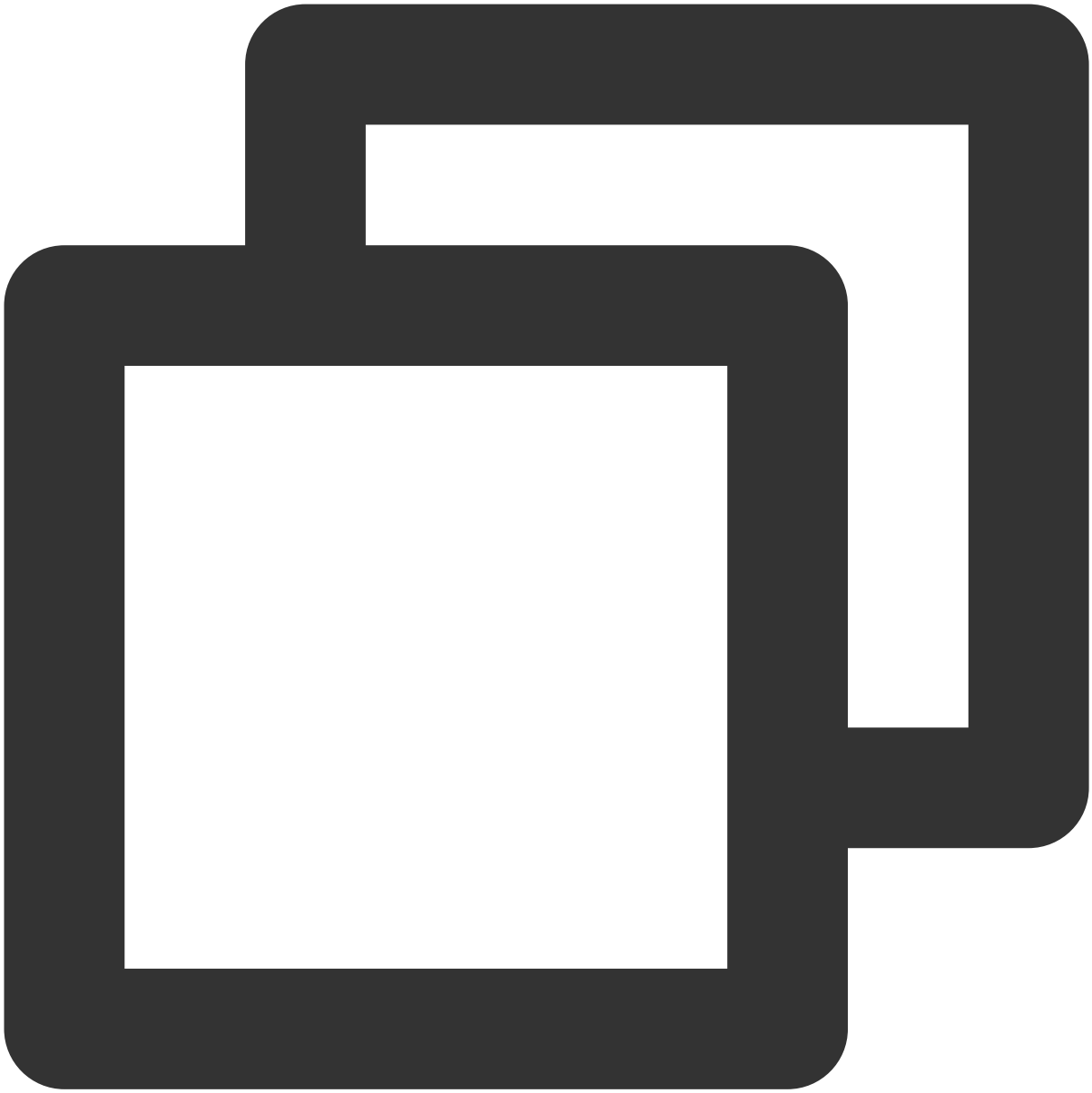
```
yum install php-posix
```

3. 다음 명령어를 실행하여 nginx 계정에 사용할 cron 설정을 열거나 생성합니다.



```
crontab -u nginx -e
```

4. 다음 편집 인터페이스는 vi/vim입니다. `i` 키를 눌러 편집 모드로 이동한 후 한 행을 삽입합니다. 내용은 다음과 같습니다.



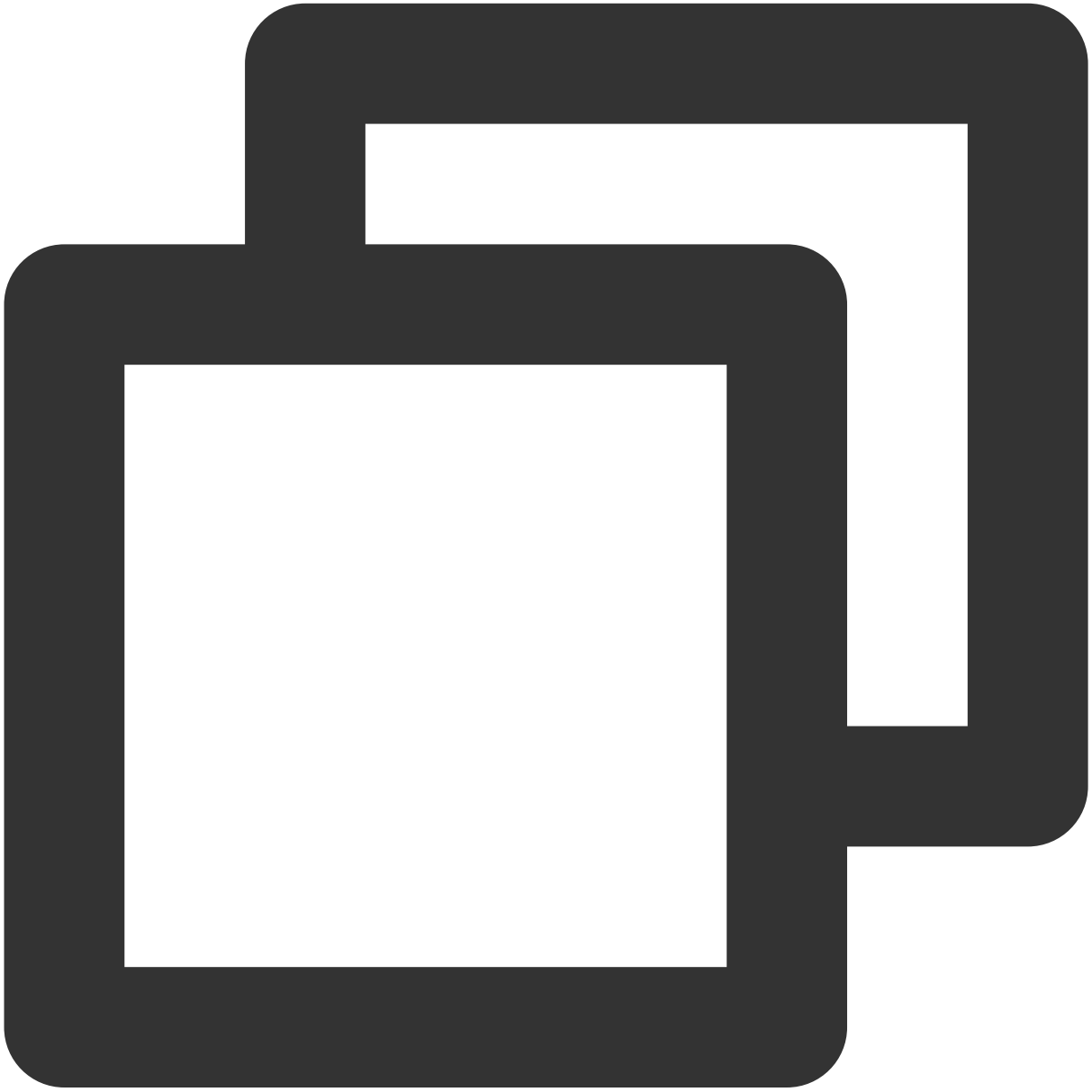
```
*/5 * * * * php -f /var/www/nextcloud/cron.php
```

`ESC` 를 눌러 편집 모드를 종료하고 `:wq` 를 입력하여 저장 후 종료합니다. vi/vim의 구체적인 작업 가이드는 관련 문서를 참조하십시오.
상기 구성에서는 공식적으로 권장되는 실행 빈도를 5분에 한 번씩 설정합니다. 백그라운드 작업이 5분 안에 실행된 후에는 브라우저를 열어 NextCloud 서버에 로그인할 수 있습니다. 그 후, 오른쪽 상단 모서리에서 사용자 이름 첫 글자를 클릭하여 **설정** 페이지로 이동한 후, 좌측 사이드바에서 **기본 설정**을 선택하면 기본적으로 Cron이 **백그라운드 작업**으로 선택되어 있는 것을 확인할 수 있습니다.

메모리 캐시

PHP는 OPcache를 사용하여 성능을 향상할 수 있습니다. NextCloud 서버도 APCu를 사용한 메모리 캐시 성능 향상을 지원합니다. 아래 관련 작업 과정을 소개합니다.

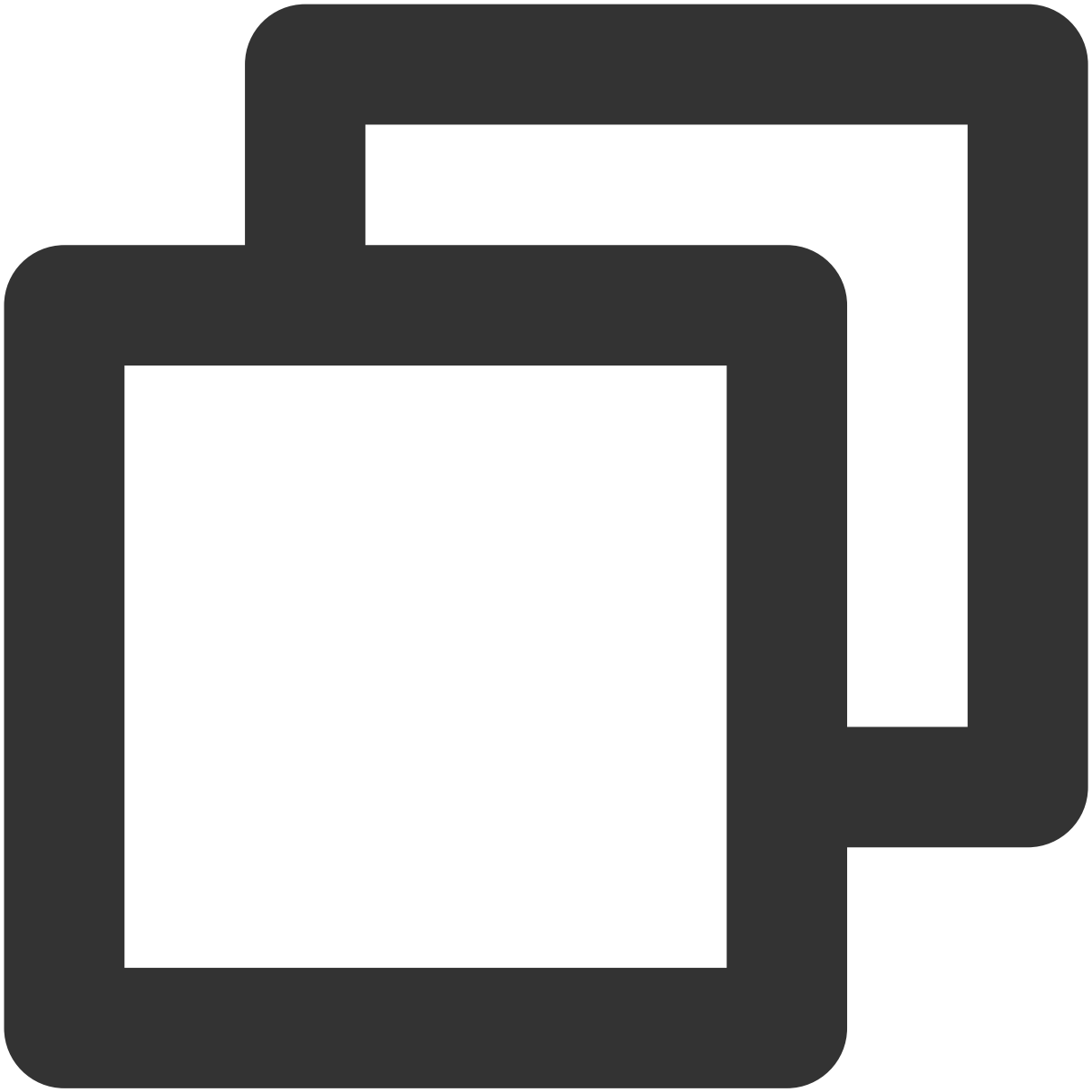
1. SSH 툴을 사용하여 구매한 서버에 로그인합니다.
2. 다음 명령어를 실행하여 PHP 모듈을 설치합니다.



```
yum install php-pecl-apcu
```

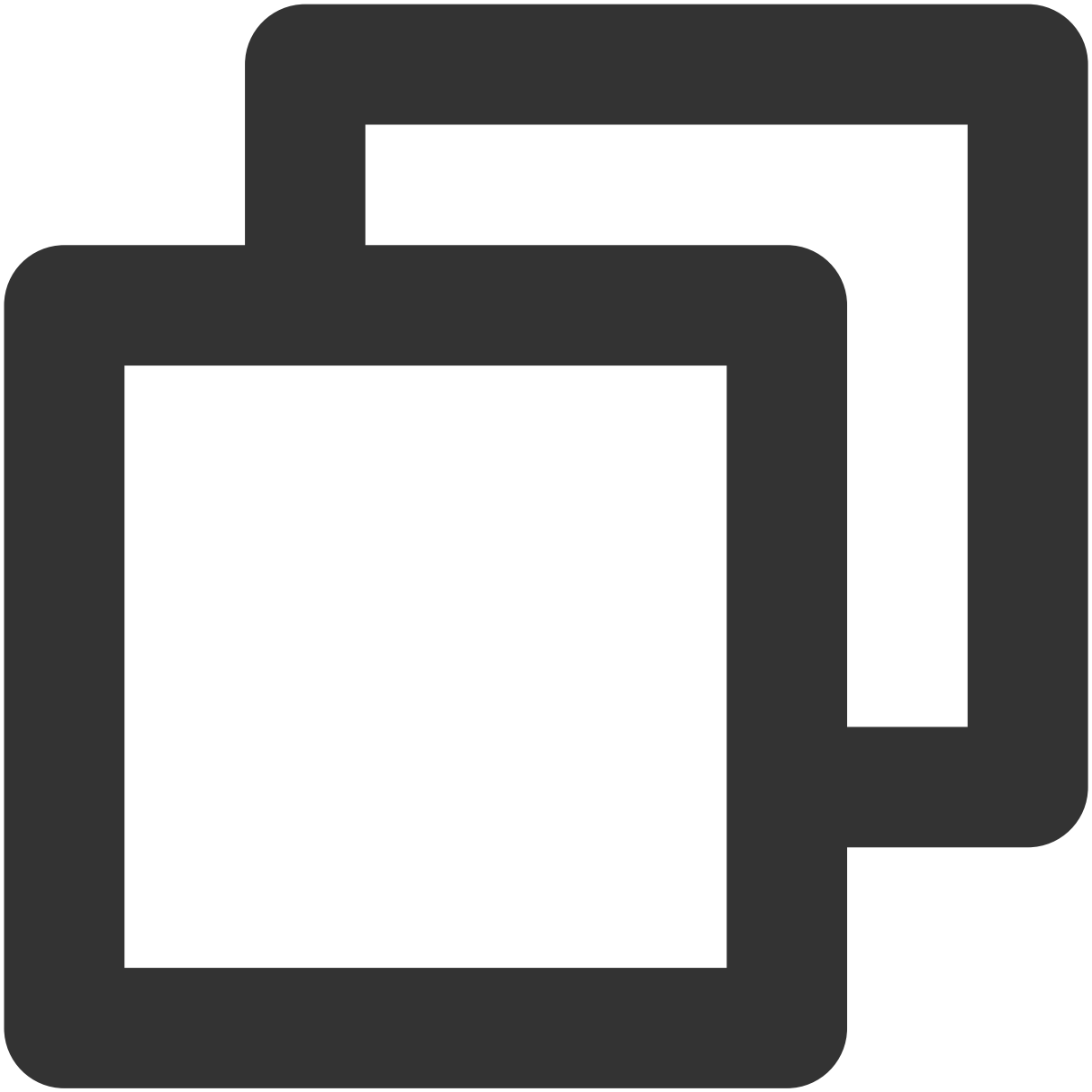
3. 계속해서 다음 명령어를 실행하여 Nginx와 PHP-FPM을 재시작합니다.

명령어 1:



```
systemctl restart nginx
```

명령어 2:



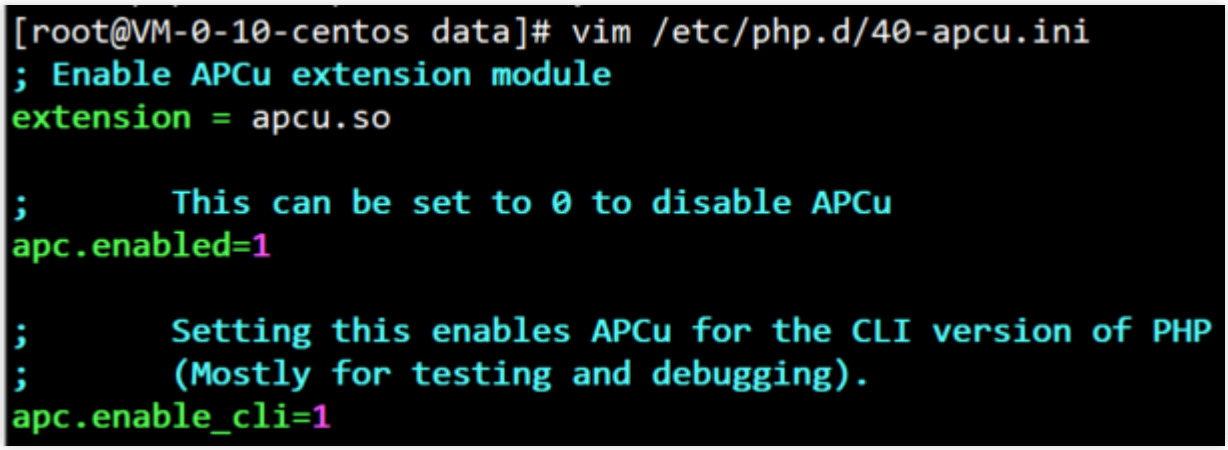
```
systemctl restart php-fpm
```

4. `vim /var/www/nextcloud/config/config.php` 를 실행하여 NextCloud 서버의 구성 파일을 열고 `$CONFIG = array (` 에 한 행 `'memcache.local' => '\\OC\\Memcache\\APCu',` 을 추가합니다. 파일을 저장하고 종료합니다.

```
[root@VM-0-10-centos config]# vim /var/www/nextcloud/config/config.
<?php
$CONFIG = array (
  'objectstore' =>
  array (
    'class' => '\\OC\\Files\\ObjectStore\\S3',
    'arguments' =>
    array (
      'bucket' => 'nextcloud-125     555',
      'autocreate' => false,
      'key' => 'AKID                                  6NEu',
      'secret' => 'nT2P                               TH0X',
      'hostname' => 'cos.ap-shanghai.myqcloud.com',
      'use_ssl' => true,
    ),
  ),
  'instanceid' => '          ',
  'passwordsalt' => '                                ',
  'secret' => '                                                                ',
  'trusted_domains' =>
  array (
    0 => '                ',
  ),
  'datadirectory' => '/var/www/nextcloud/data',
  'dbtype' => 'mysql',
  'version' => '19.0.1.1',
  'overwrite.cli.url' => 'http://                ',
  'dbname' => 'nextcloud',
  'dbhost' => '172.17.0.13:3306',
  'dbport' => '',
  'dbtableprefix' => 'oc_',
  'mysql.utf8mb4' => true,
  'dbuser' => 'oc_          ',
  'dbpassword' => '                                ',
  'installed' => true,
  'memcache.local' => '\OC\Memcache\APCu',
);
```

5. cron 설정으로 백그라운드 작업을 최적화한 후 PHP의 apc 설정 변경이 필요합니다. `vim /etc/php.d/40-apcu.ini` 를 실행하여 PHP APCu의 구성 파일을 열고 `;apc.enable_cli=0` 을 `apc.enable_cli=1` 로 수정(주의: 맨 앞의 세미콜론을 모두 삭제)한 다음 저장하고 종료합니다. `/etc/php.d/40-apcu.ini` 경로가 존재

하지 않을 경우 직접 `/etc/php.d/` 디렉터리에서 apc 또는 apcu 문구가 들어간 `.ini` 구성 파일을 찾아 편집합니다.



클라이언트 액세스 설정

NextCloud 공식 홈페이지가 제공하는 데스크톱 동기화 클라이언트와 모바일 클라이언트는 NextCloud 공식 홈페이지 또는 애플리케이션 스토어에서 다운로드할 수 있습니다. NextCloud 설정 시 NextCloud의 서버 주소(도메인 또는 IP)를 입력해야 합니다. 그런 다음 사용자 이름과 비밀번호를 입력하고 로그인하면 바로 클라이언트를 사용할 수 있습니다.

COS를 로컬 드라이브로 Windows 서버에 마운트하기

최종 업데이트 날짜: : 2024-06-24 16:53:18

작업 시나리오

현재 Windows시스템에서의 Tencent Cloud COS 작업은 주로 API, COSBrowser, COSCMD 툴로 구현됩니다. Windows 서버를 사용하는 사용자는 COSBrowser를 클라우드 저장소로만 사용할 수 있으므로 프로그램을 실행하거나 작업을 수행하는 데 적합하지 않습니다. 본 문서는 비용 효율적인 COS를 로컬 드라이브로 Windows 서버에 마운트하는 방법을 소개합니다.

설명：

본 문서에 제공된 예시는 Windows 7 또는 Windows Server 2012 / 2016 / 2019 / 2022에만 적용됩니다.

작업 단계

다운로드 및 설치

본 문서에 제공된 예시에는 세 가지 유형의 소프트웨어가 포함됩니다. 시스템과 호환되는 소프트웨어 버전을 설치할 수 있습니다.

Github로 이동하여 Winfsp를 다운로드합니다.

예시로 winfsp-1.12.22301을 다운로드 합니다. 그 다음 기본 옵션으로 설치할 수 있습니다.

설명：

Windows Server 2012 R2의 경우 Winfsp 1.12.22242는 호환되지 않지만 Winfsp 1.11.22176은 호환됩니다.

1. Git 또는 Github로 이동하여 Git을 다운로드합니다.

Git-2.38.1-64-bit는 예시로 다운로드 됩니다. 그런 다음 기본 옵션으로 설치할 수 있습니다.

2. Rclone 또는 Github에 접속하여 Rclone을 다운로드합니다.

예시로 rclone-v1.60.1-windows-amd64가 다운로드됩니다. 영어로 이름이 지정된 디렉터리에만 압축을 풀면 됩니다(한자가 포함된 경로로 압축을 풀면 오류가 발생할 수 있음). 이 예시에서 패키지는 E:\\AutoRclone으로 압축 해제됩니다.

설명：

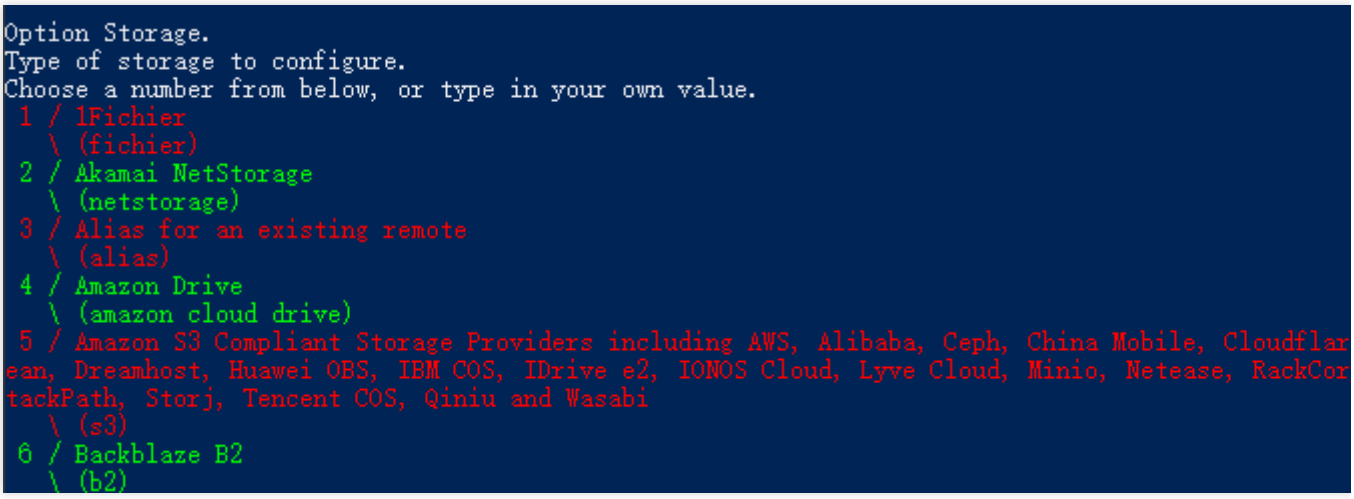
GitHub를 열 수 없거나 다운로드 속도가 느린 경우 다른 다운로드 방법을 찾을 수 있습니다.

Rclone 구성

주의：

다음 구성 프로세스는 rclone-v1.60.1-windows-amd64를 예로 들어 설명합니다. 구성 프로세스는 버전에 따라 다를 수 있습니다.

1. 폴더를 열고 왼쪽 탐색 창에서 **이 PC**를 찾아 우클릭한 다음 **속성 > 고급 시스템 설정 > 환경 변수 > 시스템 변수 > Path**를 선택합니다. **생성**을 클릭합니다.
2. 팝업 창에 Rclone이 압축 해제된 경로(E:\\AutoRclone)를 입력하고 **확인**을 클릭합니다.
3. Windows Powershell을 열고 `rclone --version` 명령을 실행하여 **Enter**를 눌러 Rclone이 성공적으로 설치되었는지 확인합니다.
4. Rclone 설치 확인 후 Windows Powershell에서 `rclone config` 명령을 입력하고 **Enter**를 눌러 명령을 실행합니다.
5. Windows Powershell에서 **n**을 입력하고 **Enter**를 눌러 New remote를 생성합니다.
6. Windows Powershell에서 myCOS와 같은 해당 디스크의 이름을 입력하고 **Enter**를 누릅니다.
7. 표시되는 옵션에서 'Tencent COS'가 포함된 옵션을 선택(즉, **5** 입력)한 다음 **Enter**를 누릅니다.



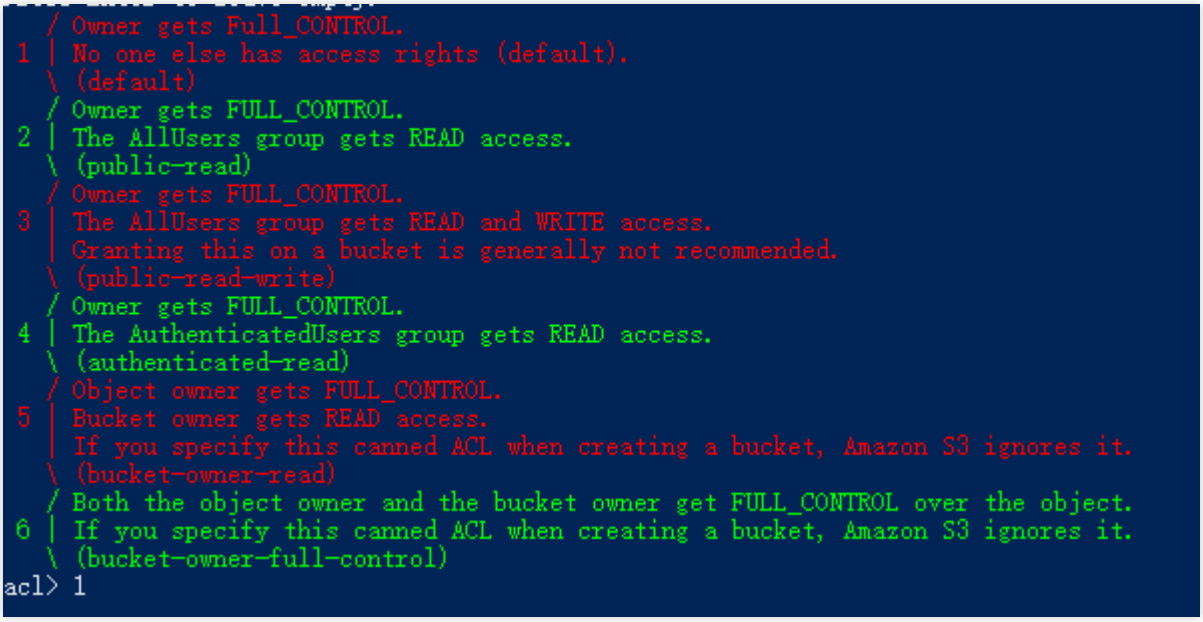
8. 표시되는 옵션에서 'TencentCOS'가 포함된 옵션(즉, **21** 입력)을 선택한 다음 **Enter**를 누릅니다.

9. `env_auth>` 가 표시되면 Enter를 누릅니다.
10. `access_key_id>` 가 표시되면 COS의 SecretId를 입력하고 **Enter**를 누릅니다.

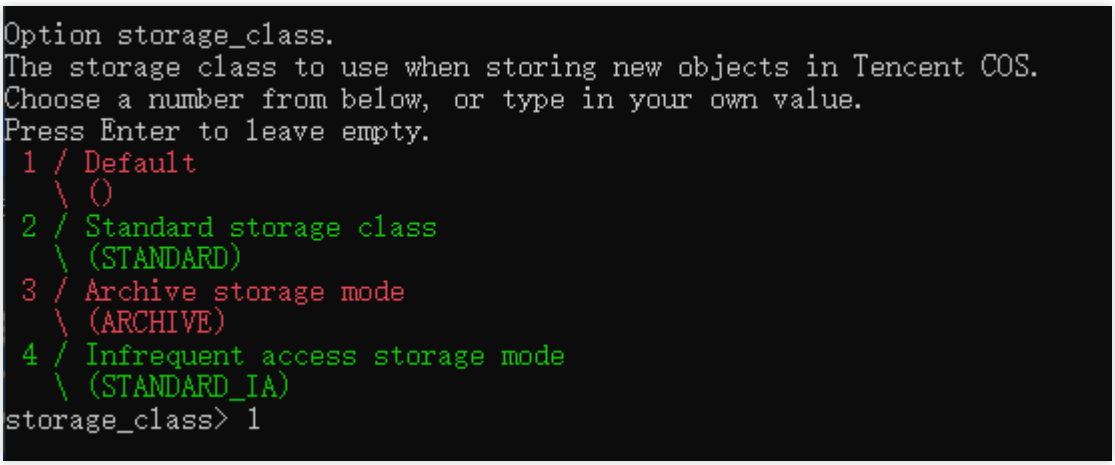
설명：

여기서는 서브 계정 권한 사용을 권장합니다. API 키 관리로 이동하여 SecretId와 SecretKey를 확인할 수 있습니다.

11. `secret_access_key>` 가 표시되면 COS의 SecretKey를 입력하고 **Enter**를 누릅니다.
12. 표시되는 Tencent Cloud 리전의 게이트웨이 주소에 따라 버킷 리전을 선택합니다.
여기에서는 광저우 리전을 예로 사용합니다. 따라서 **4**(`cos.ap-guangzhou.myqcloud.com`)를 입력한 후 **Enter**를 누릅니다.
13. 필요에 따라 객체 권한(예시: default, public-read 등)을 선택합니다. 이 권한은 나중에 업로드되는 객체에만 적용됩니다. 여기서는 default가 예로 사용됩니다. 따라서 **1**을 입력한 다음 **Enter**를 누릅니다.



14. COS에 업로드된 객체의 스토리지 클래스를 선택합니다. 여기서는 Default가 예로 사용됩니다. 따라서 **1**을 입력한 다음 **Enter**를 누릅니다.



Default: 기본값
Standard storage class: 스탠다드 스토리지(STANDARD)
Archive storage mode: 아카이브 스토리지(ARCHIVE)
Infrequent access storage mode: 스탠다드IA 스토리지(STANDARD_IA)

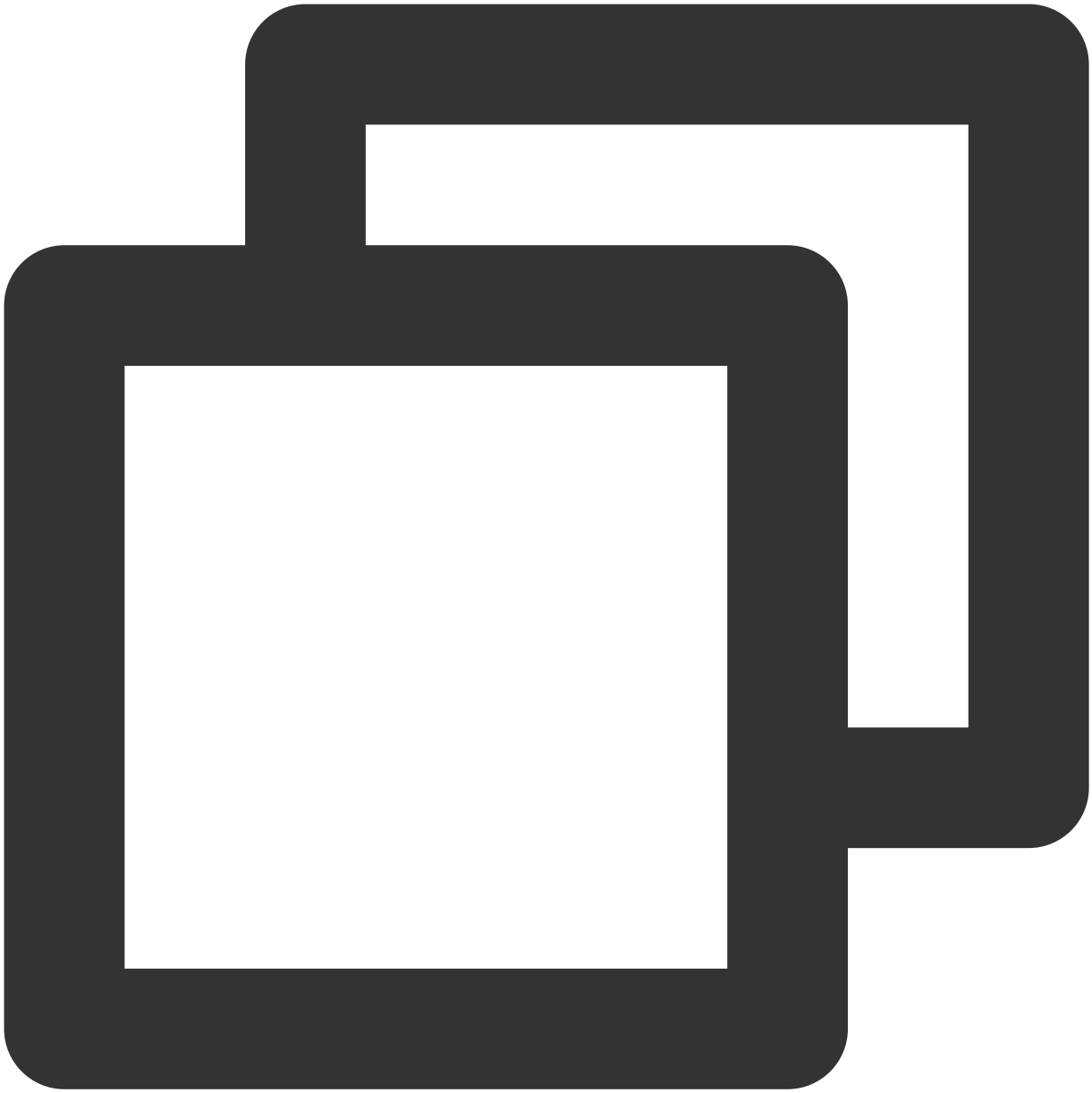
설명：
INTELLIGENT TIERING 또는 DEEP ARCHIVE 스토리지 클래스를 사용하려면 **구성 파일 수정** 방법을 사용한 다음 storage_class 값을 INTELLIGENT_TIERING 또는 DEEP_ARCHIVE로 설정합니다. 스토리지 클래스에 대한 자세한 내용은 스토리지 유형 개요를 참고하십시오.
15. `Edit advanced config? (y/n)` 이 표시되면 **Enter**를 누릅니다.
16. 정보에 오류가 없는지 확인한 후 **Enter**를 누릅니다.
17. **q**를 입력하여 구성을 완료합니다.

구성 파일 수정

이전 구성이 완료되면 `C:\\Users\\Username\\AppData\\Roaming\\rclone` 디렉터리에 rclone.conf라는 구성 파일이 생성됩니다. 파일을 직접 수정하여 rclone 구성을 업데이트할 수 있습니다. 파일을 찾을 수 없는 경우 명령줄 창에서 `rclone config file` 명령을 실행하여 rclone 구성 파일을 볼 수 있습니다.

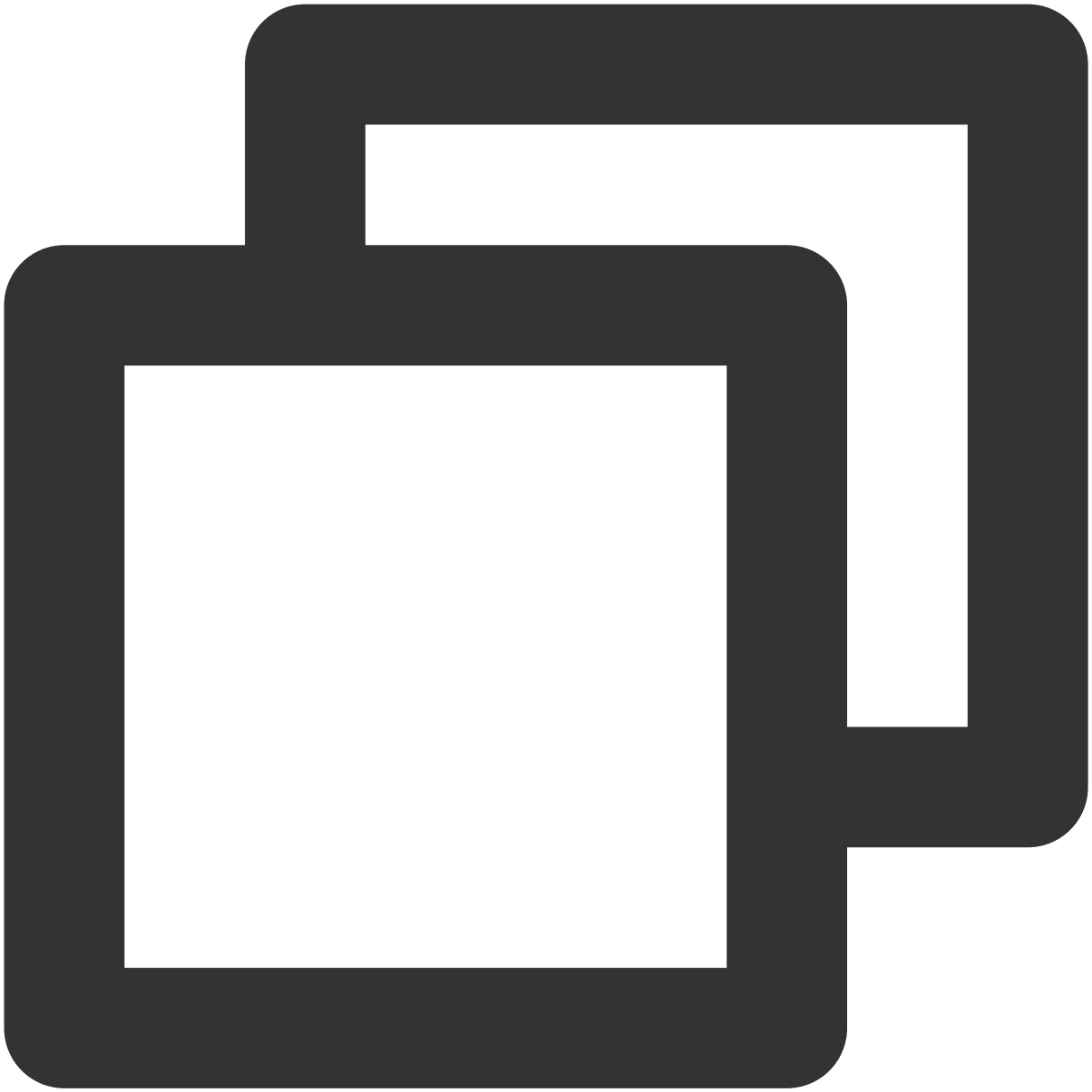
COS를 로컬 드라이브로 마운트

1. 설치된 Git Bash를 열고 명령 라인 툴에 실행 명령을 입력합니다. 여기에는 두 가지 사용 시나리오가 제공되며(둘 중 하나 선택), 실제 필요에 따라 둘 중 하나를 선택할 수 있습니다.
LAN에서 COS를 공유 드라이브로 마운트하려면(권장) 다음 명령을 실행하십시오.



```
rclone mount myCOS:/ Y: --fuse-flag --VolumePrefix=\\server\\share --cache-dir E:\\
```

로컬 디스크로 매핑된 경우 다음과 같이 명령을 실행합니다:



```
rclone mount myCOS:/ Y: --cache-dir E:\\temp --vfs-cache-mode writes &
```

myCOS: 사용자 정의 디스크 이름으로 변경합니다.

Y: 마운트할 디스크로 변경한 후 디스크의 드라이브 문자를 변경하면 됩니다. 로컬의 C, D, E 드라이브 등과 중복되지 않아야 합니다.

E:\\temp: 필요에 따라 설정할 수 있는 로컬 캐시 디렉터리입니다. 디렉터리에 대한 권한이 있습니다.

"The service rclone has been started"가 안내되면 마운트가 성공적으로 완료되었음을 의미합니다.

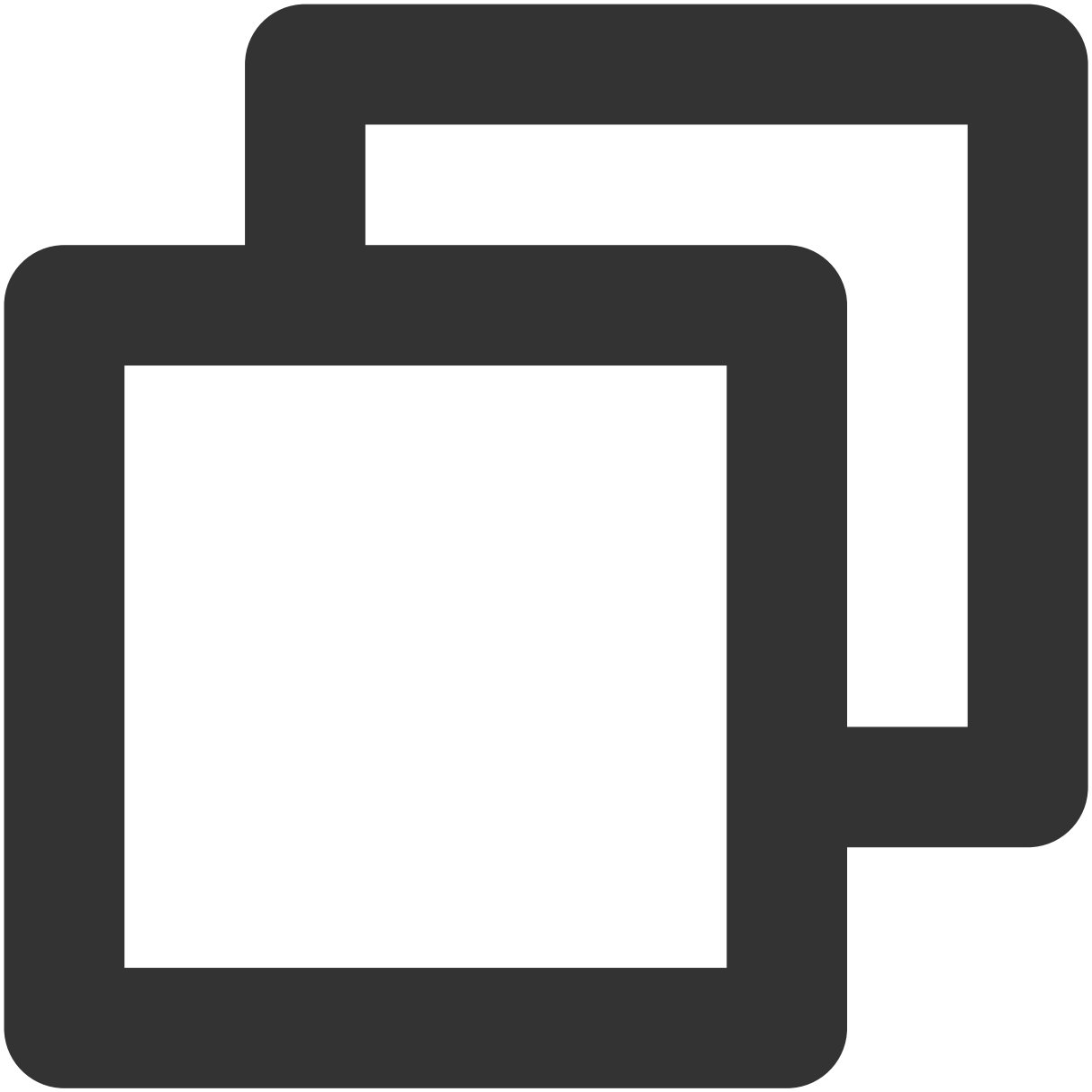
2. **exit**를 입력하고 터미널을 종료합니다.

3. 로컬 컴퓨터의 **내 PC**에서 myCOS(Y:)란 이름의 디스크를 찾을 수 있습니다.
해당 디스크를 열면 광저우 리전 전체에 속한 모든 버킷 이름을 조회할 수 있습니다. 이때 업로드, 다운로드, 생성, 삭제 등 로컬 디스크에서 자주 쓰는 작업을 진행할 수 있습니다.

주의：
작업 도중 오류가 발생하면 git bash 소프트웨어에서 상세 오류 정보를 확인하십시오.
디스크를 마운트하는 도중 버킷에서 삭제 작업을 하면 버킷 내 파일 존재 여부와 상관없이 모두 삭제될 수 있으니 신중히 작업하시기 바랍니다.
디스크를 마운트하는 도중 버킷 이름을 변경하는 경우 COS 버킷 이름이 변경될 수 있으니 신중히 작업하시기 바랍니다.

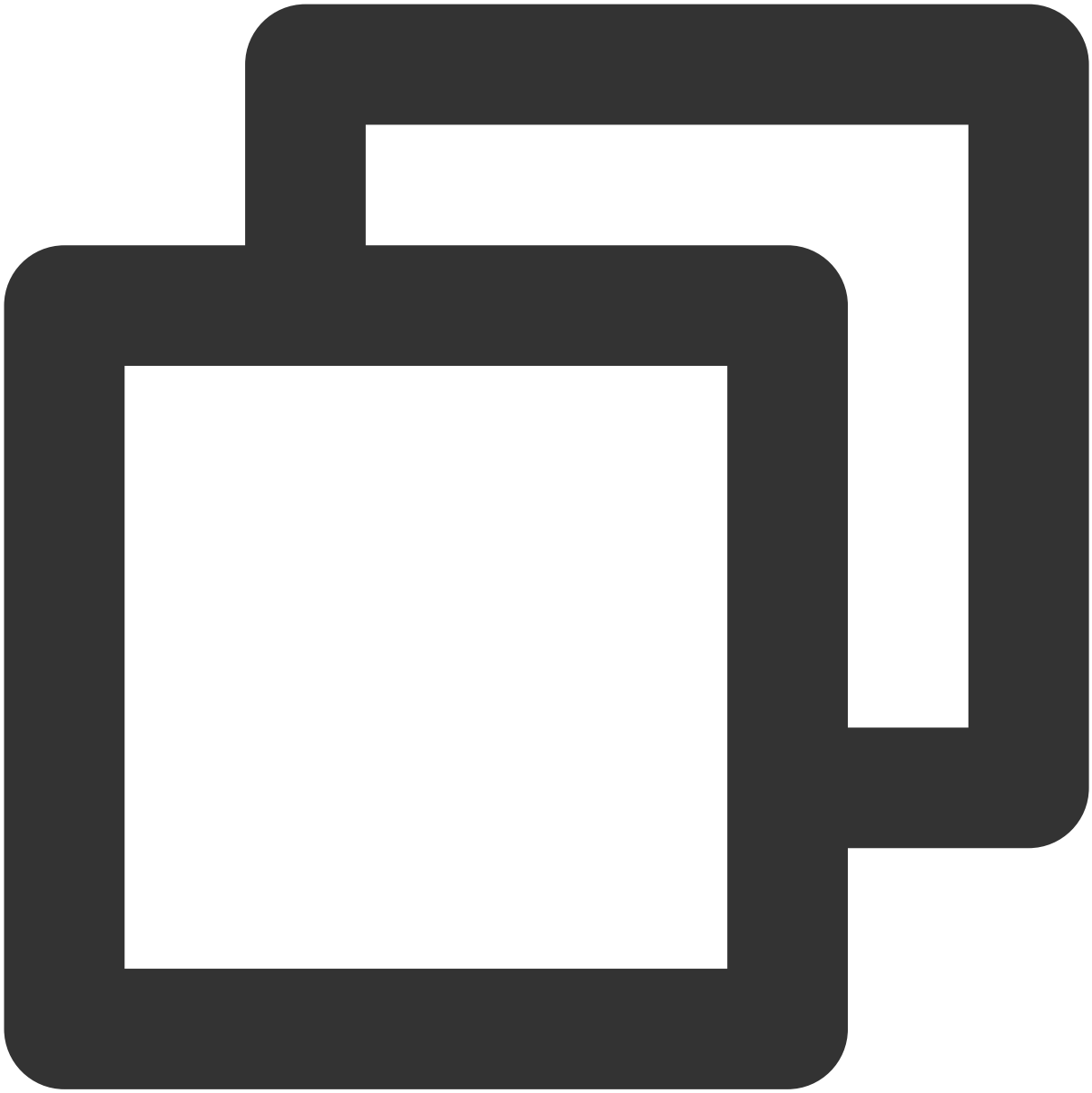
시작 시 자동으로 마운트된 드라이브 실행

마운트된 드라이브는 서버가 재시작되면 사라집니다. 따라서 다음 작업을 수행하여 시작 시 드라이브가 자동 실행되도록 설정할 수 있습니다.
1. E:\\AutoRclone 디렉터리에 startup_rclone.vbs 및 startup_rclone.bat 파일을 생성합니다.
설명：
Powershell을 통해 텍스트 파일을 생성할 때 올바른 인코딩을 사용하십시오. 그렇지 않으면 생성된 .bat 및 .vbs 파일을 실행할 수 없습니다.
2. startup_rclone.bat에 다음 마운트 명령어를 입력합니다.
COS가 LAN에서 공유 드라이브로 마운트된 경우 다음 명령을 실행합니다.



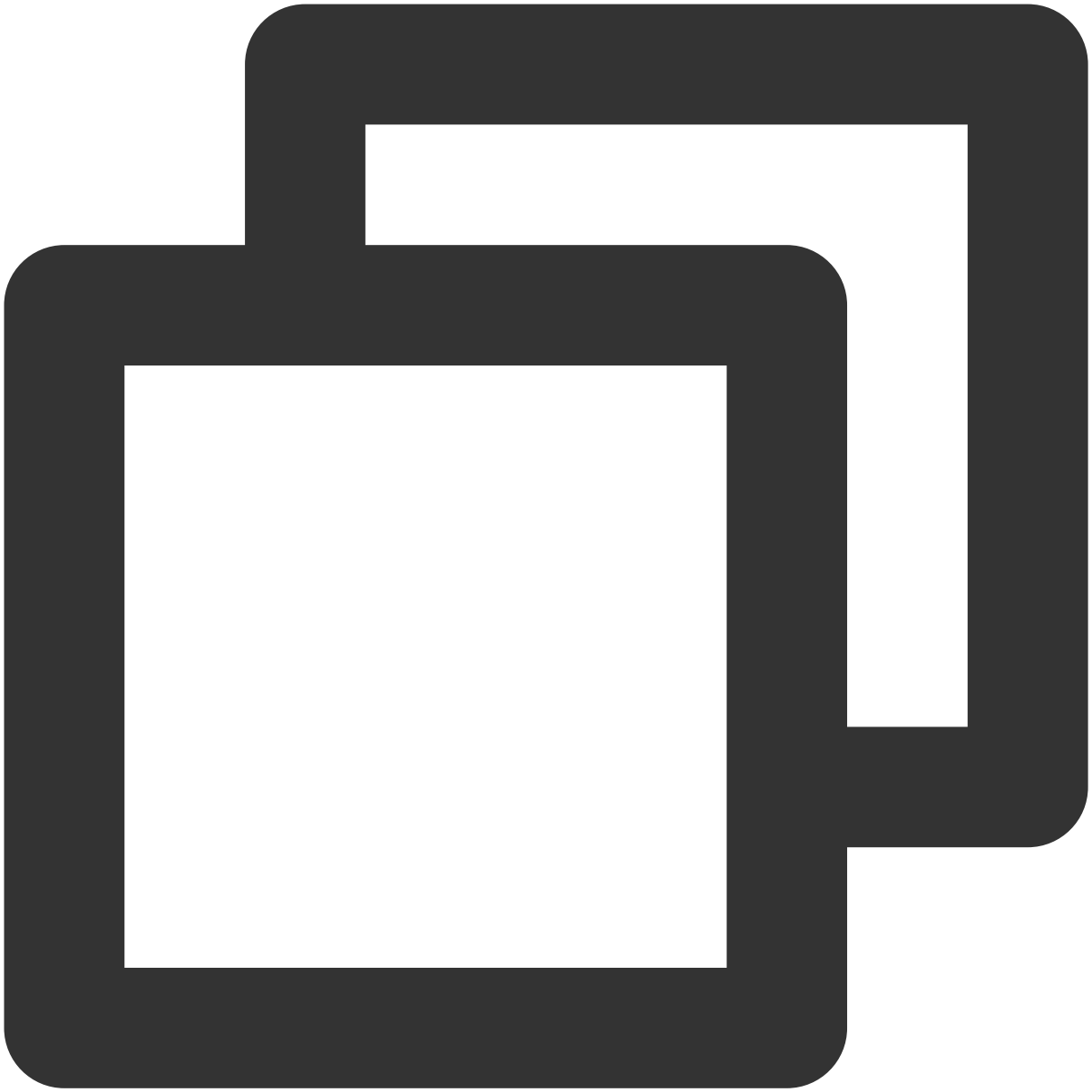
```
rclone mount myCOS:/ Y: --fuse-flag --VolumePrefix=\\server\\share --cache-dir E:\\
```

로컬 디스크로 매핑한 경우 다음 명령어를 입력합니다.



```
rclone mount myCOS:/ Y: --cache-dir E:\\temp --vfs-cache-mode writes &
```

3. startup_rclone.vbs에 다음 코드를 입력합니다.



```
CreateObject("WScript.Shell").Run "cmd /c E:\\AutoRclone\\startup_rclone.bat",0
```

주의 :

코드 경로를 실행 경로로 수정하십시오.

4. startup_rclone.vbs 파일을 잘라내 %USERPROFILE%\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Startup 폴더에 붙여 넣습니다.

5. 서버를 재시작합니다.

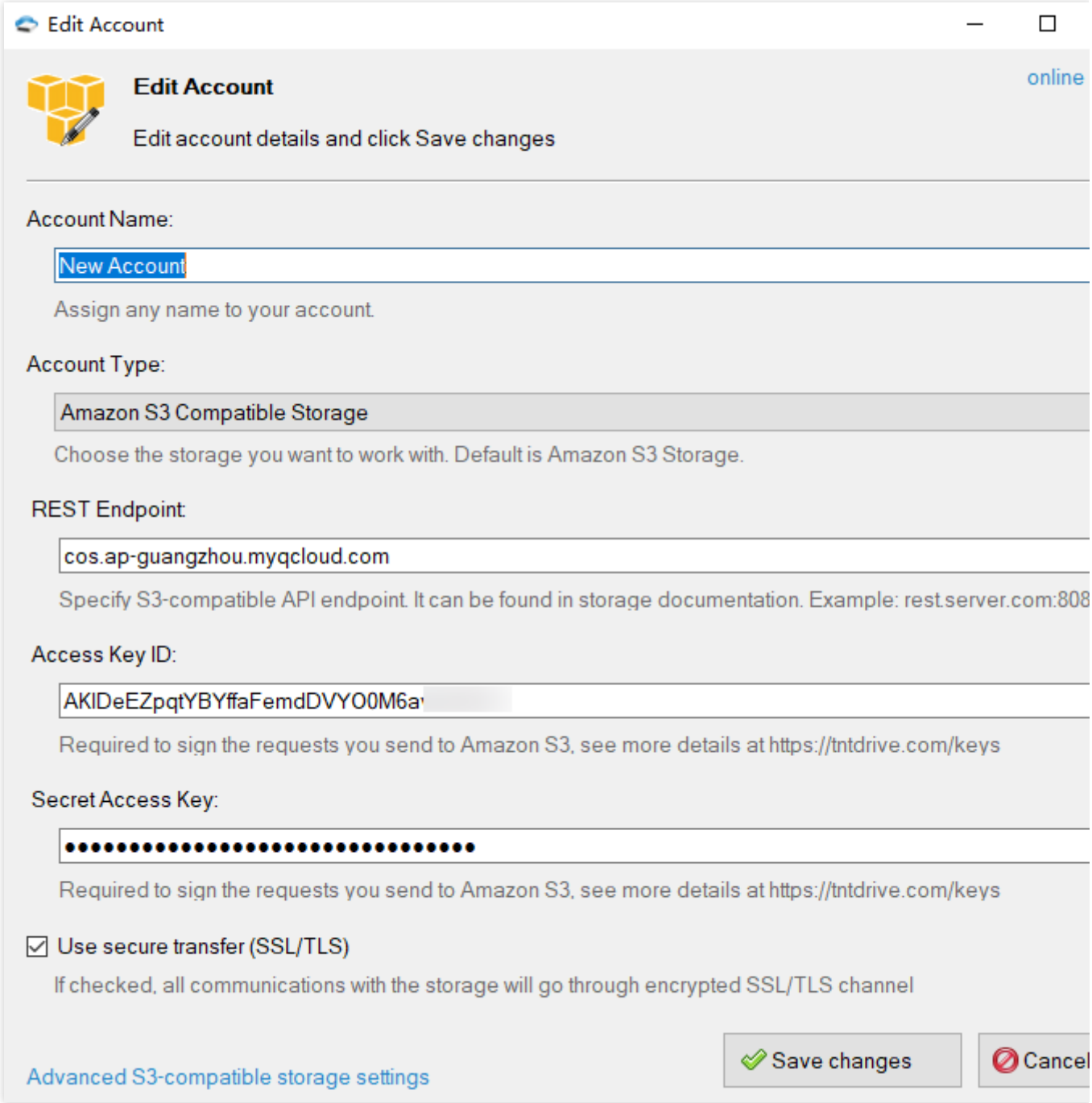
설명 :

일반적으로 자동 마운트 구성이 완료되고 서버가 다시 시작되면 마운트 성공 메시지가 수십 초 후에 표시됩니다.

관련 작업

3rd party 상용화 유료 툴을 사용해 COS를 Windows 서버에 마운트한 후 로컬 디스크로 매핑할 수도 있습니다. 다음 작업은 TntDrive 툴 예시입니다.

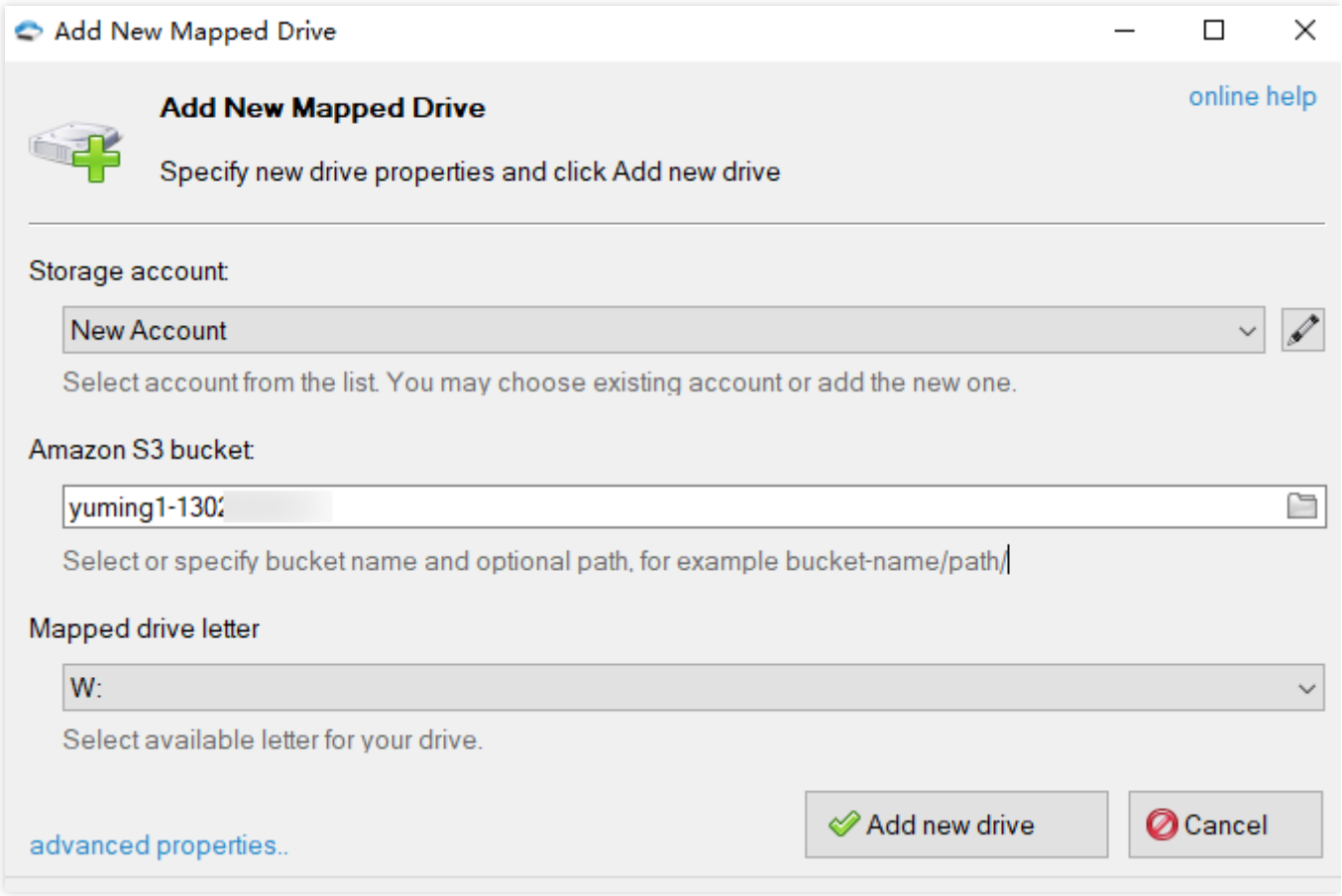
1. TntDrive를 다운로드 및 설치합니다.
2. TntDrive를 열고 **Account > Add New Account**를 클릭해 계정을 생성합니다.



주요 매개변수 정보는 다음과 같습니다.

Account Name: 사용자 정의 계정 이름

Account Type: COS는 S3와 호환되므로, 여기서 **Amazon S3 Compatible Storage**를 선택할 수 있습니다.
REST Endpoint: 버킷이 위치한 리전을 입력합니다. 예를 들어, 버킷이 광저우 리전에 있다면 cos.ap-guangzhou.myqcloud.com을 입력합니다.
Access Key ID: SecretId를 입력합니다. API 키 관리 페이지에서 생성 및 획득할 수 있습니다.
Secret Access Key: SecretKey를 입력합니다.
3. **Add new account**를 클릭합니다.
4. TntDrive 인터페이스에서 **Add New Mapped Drives**를 클릭해 Mapped Drives를 생성합니다.



주요 매개변수 정보는 다음과 같습니다.
Amazon S3 Bucket: 버킷 경로를 입력하거나 버킷 이름을 선택합니다. 우측 버튼을 클릭해 버킷을 선택할 수 있습니다. 2단계에서 설정한 광저우 리전의 버킷이 표시되며, 버킷 1개당 1개의 디스크가 독립적으로 매핑됩니다.
Mapped drives letter: 디스크의 드라이브 문자를 설정합니다. 로컬의 C, D, E 드라이브 등과 중복되지 않아야 합니다.
5. 위의 정보를 확인하고 **Add new drive**를 클릭합니다.
6. 로컬 컴퓨터의 **내 PC**에서 드라이브를 찾습니다. 모든 버킷을 Windows 서버에 매핑하려면 위의 단계를 반복합니다.

PicGo+Typora+COS를 사용하여 이미지 호스팅 서비스 구축

최종 업데이트 날짜: : 2024-06-24 16:53:18

소개

이미지 호스팅 서비스는 이미지 저장, 처리, 배포 등 다양한 기능을 제공하여 전 세계의 수많은 블로그 사이트와 커뮤니티 포럼에 백엔드 이미지 서비스를 제공합니다. 개발자들은 **Cloud Object Storage(COS)**를 사용하여 이미지 호스팅 서비스를 설정할 수 있습니다. COS는 Tencent Cloud가 출시한 분산 스토리지 서비스로 대용량 파일을 저장하고 더 높은 성능과 안정성을 보장합니다.
이미지 호스팅 시나리오에서 COS의 장점은 다음과 같습니다.
저렴한 비용: 스토리지의 단가가 저렴하고 사용한 만큼만 비용을 지불하면 됩니다.
무제한 속도: 업로드 및 다운로드 속도가 제한되지 않으므로 사용자는 더 이상 느린 loading을 기다릴 필요가 없이 더 나은 액세스 경험을 즐길 수 있습니다.
고가용성: COS는 저장된 데이터가 최대 99.9999999999%의 내구성을 보장하는 고가용성을 위한 SLA를 제공합니다.
무제한 용량: COS는 주문형 용량 사용을 위해 많은 수의 파일을 분산 방식으로 저장합니다.

실행 시나리오

시나리오1: COS로 이미지 호스팅 서비스를 설정하기 위해 이미지 추가

이 시나리오에서는 다음 도구가 사용됩니다.
PicGo: 여러 클라우드 스토리지 구성을 지원하고 이미지 URL을 빠르게 생성하는 도구입니다.
Typora: 여러 출력 형식을 지원하고 로컬 이미지를 이미지 호스팅 서비스에 빠르게 업로드할 수 있는 경량 Markdown 파일 편집기입니다.

작업 단계

1. PicGo를 설치하고 관련 COS 매개변수를 설정합니다.
설명：
이 시나리오에서는 PicGo 2.3.1이 사용됩니다. 구성 프로세스는 버전에 따라 다를 수 있습니다.
PicGo 웹사이트에서 PicGo를 다운로드하여 설치한 후 이미지 호스팅 서비스 설정에서 **Tencent Cloud COS**를 찾아 다음 매개변수를 구성합니다.
COS 버전: COS v5를 선택합니다.
SecretId 설정: 프로젝트에 사용되는 개발자 소유의 비밀 ID로, API 키 관리 페이지에서 생성 및 획득할 수 있습니다.

SecretKey 설정: 프로젝트에 사용되는 개발자 소유의 비밀 키이며 API 키 관리 페이지에서 얻을 수 있습니다.
Bucket 설정: 버킷. COS에서 데이터 저장에 사용되는 컨테이너이며, 버킷과 관련된 자세한 내용은 버킷 개요 문서를 참고하십시오.
AppId 설정: COS 액세스에 대한 고유한 사용자 레벨 리소스 식별자이며, API 키 관리 페이지에서 얻을 수 있습니다.
스토리지 리전 설정: 버킷 소속 리전 정보, ap-beijing, ap-hongkong 및 eu-frankfurt와 같은 열거 값은 리전 및 액세스 도메인을 참고하십시오.
스토리지 경로 설정: COS 버킷에서 이미지가 저장되는 경로입니다.
사용자 정의 도메인 이름 설정: 이 매개변수는 선택사항입니다. 위에서 지정한 스토리지 공간에 대해 사용자 지정 원본 도메인 이름을 구성한 경우 여기에 입력할 수 있습니다. 자세한 내용은 사용자 정의 원본 서버 도메인 활성화하기를 참고하십시오.
URL 접미사 설정: URL 접미사에 COS 데이터 처리 매개변수를 추가하여 이미지 압축, 크롭핑, 형식 변환 및 기타 작업을 구현합니다. 자세한 내용은 Image Processing을 참고하십시오.
2. typora를 구성합니다(선택 사항).
설명：
편집 요구 사항에 Markdown이 포함되지 않은 경우 이 단계를 건너뛰고 이전 단계에서 설치된 PicGo 도구를 이미지 호스팅 도구로 사용할 수 있습니다.
다음과 같이 구성합니다.
1. typora의 기본 설정 **이미지**에서 다음을 구성합니다.
이미지 삽입 시이미지 업로드를 선택합니다
이미지 업로드 설정에서 **PicGo(app)**를 선택하고 방금 설치한 PicGo.exe의 위치를 설정합니다.
2. 설정을 적용하려면 typora를 다시 시작하십시오.
3. typora 편집기 영역에 들어가 이미지를 직접 드래그 앤 드롭하거나 붙여넣어 업로드하면 자동으로 COS 파일 URL로 대체됩니다(붙여넣기 후 COS URL로 자동 교체되지 않는다면 PicGo에서 server가 활성화되어 있는지 확인하십시오).

시나리오2: 이미지 호스팅 리포지토리의 이미지를 COS로 빠르게 마이그레이션

이미지 호스팅 서비스를 예로 들면, 로컬 이미지 호스팅 폴더를 찾거나 인터넷에서 전체 폴더를 다운로드한 후 폴더의 모든 이미지를 COS 버킷으로 전송할 수 있습니다. 그런 다음 URL 도메인 이름을 균일하게 교체하여 웹 사이트를 복원합니다.

작업 단계

1단계: 원본 이미지 호스팅 서비스에서 이미지 다운로드

원본 이미지 호스팅 사이트에 로그인하여 이전에 업로드한 이미지 폴더를 다운로드합니다.

2단계: COS 버킷 생성 및 링크 도용 방지 설정

1. Tencent Cloud 계정에 가입하고 버킷 생성의 지침에 따라 **공개 읽기/비공개 쓰기** 액세스 권한이 있는 버킷을 생성합니다. 버킷 생성을 참고하십시오.

2. 버킷이 생성된 후 이미지가 핫링크되지 않도록 링크 도용 방지 설정의 지침에 따라 버킷에서 링크 도용 방지를 활성화합니다.

3단계: 버킷에 폴더 업로드

방금 생성한 COS 버킷에서 폴더 업로드를 클릭하여 준비된 이미지 폴더를 버킷에 업로드합니다.

설명：

이미지 수가 많으면 COSBrowser를 사용하여 이미지를 빠르게 업로드할 수도 있습니다.

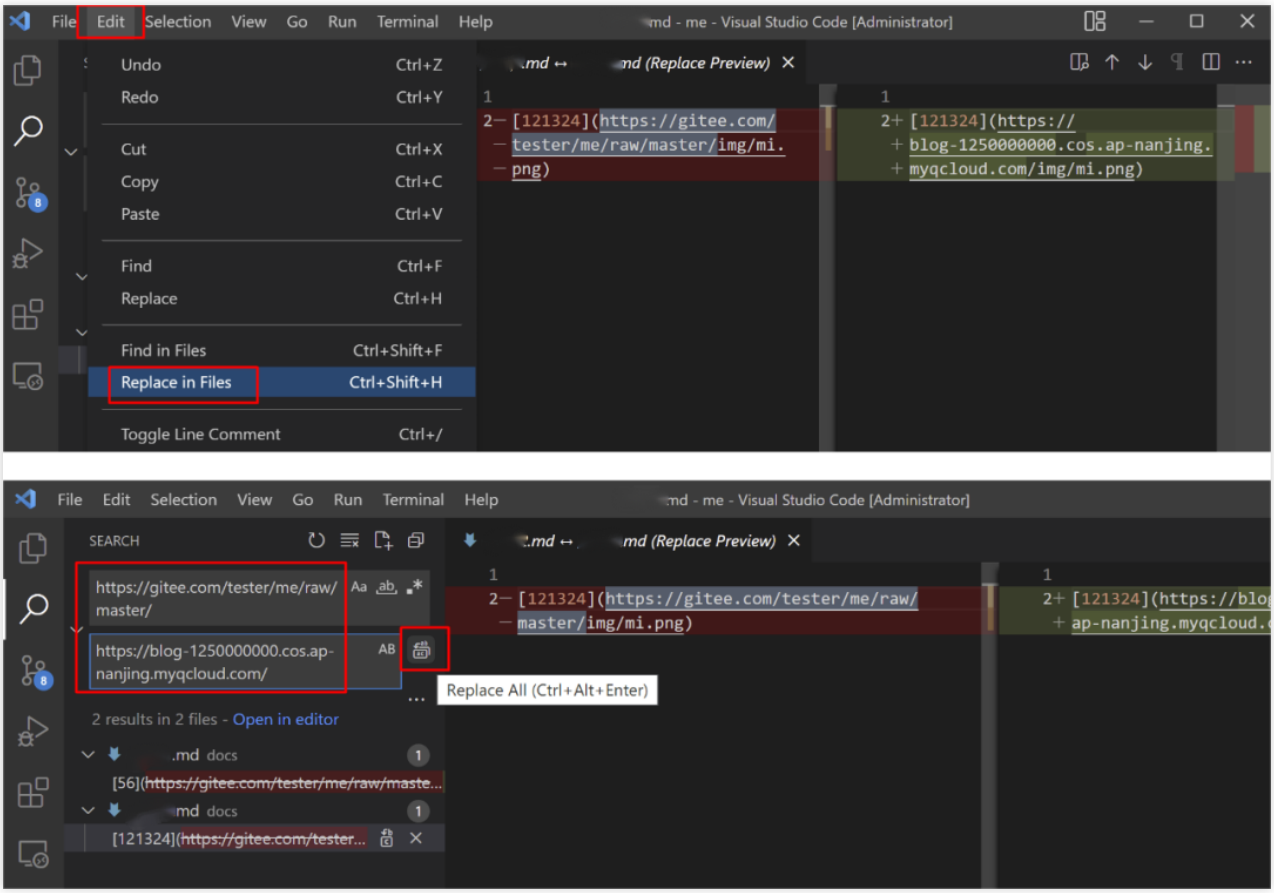
4단계: 도메인 이름을 전역적으로 교체

COS 콘솔의 버킷 개요 페이지에서 버킷의 기본 도메인 이름을 복사합니다(사용자 지정 CDN 가속 도메인 이름을 연결할 수도 있음). 그런 다음 공통 코드 편집기를 사용하여 유효하지 않은 URL 접두사를 검색하고 COS 버킷의 기본 도메인 이름으로 전역적으로 바꿉니다.

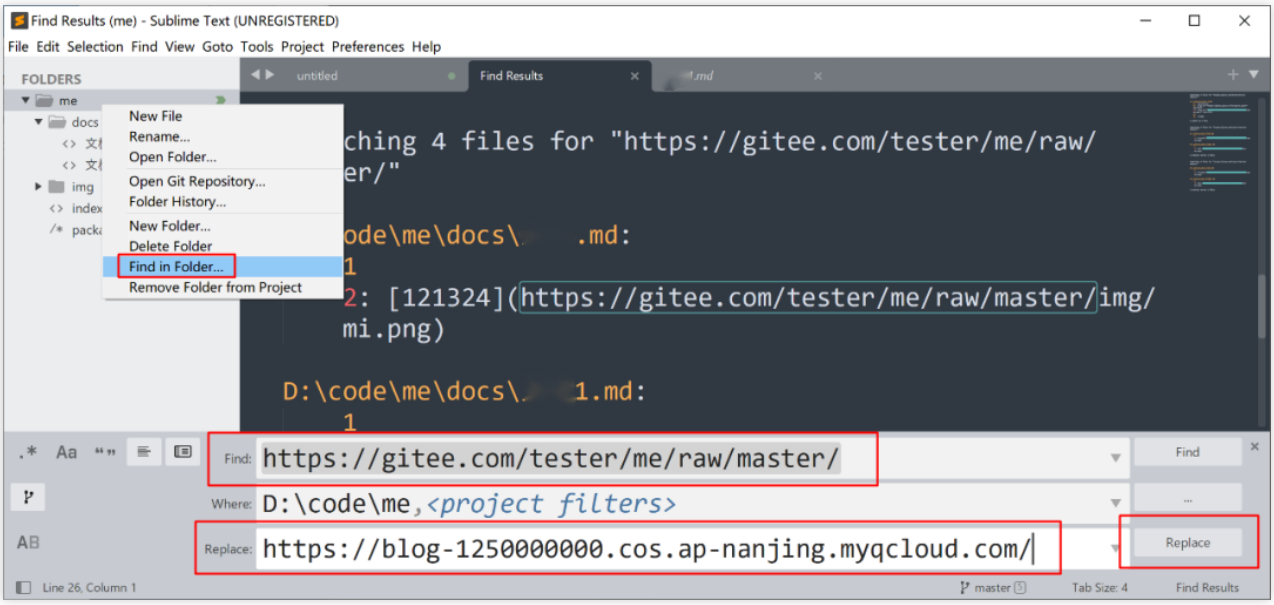
설명：

기본 도메인 이름에 대한 자세한 내용은 리전 및 액세스 도메인을 참고하십시오.

vscode를 사용한 검색 및 교체 예시:



sublime text를 사용한 검색 및 교체 예시:



CloudBerry Explorer를 통한 COS 리소스 관리

최종 업데이트 날짜: : 2024-06-24 16:53:18

소개

CloudBerry Explorer는 COS(Cloud Object Storage) 관리에 사용할 수 있는 클라이언트 툴입니다. COS는 CloudBerry Explorer를 통해 Windows 등 운영 체제에 마운트할 수 있어 사용자의 COS 파일 액세스, 이동 및 관리가 편리합니다.

지원 시스템

Windows, macOS 시스템을 지원합니다.

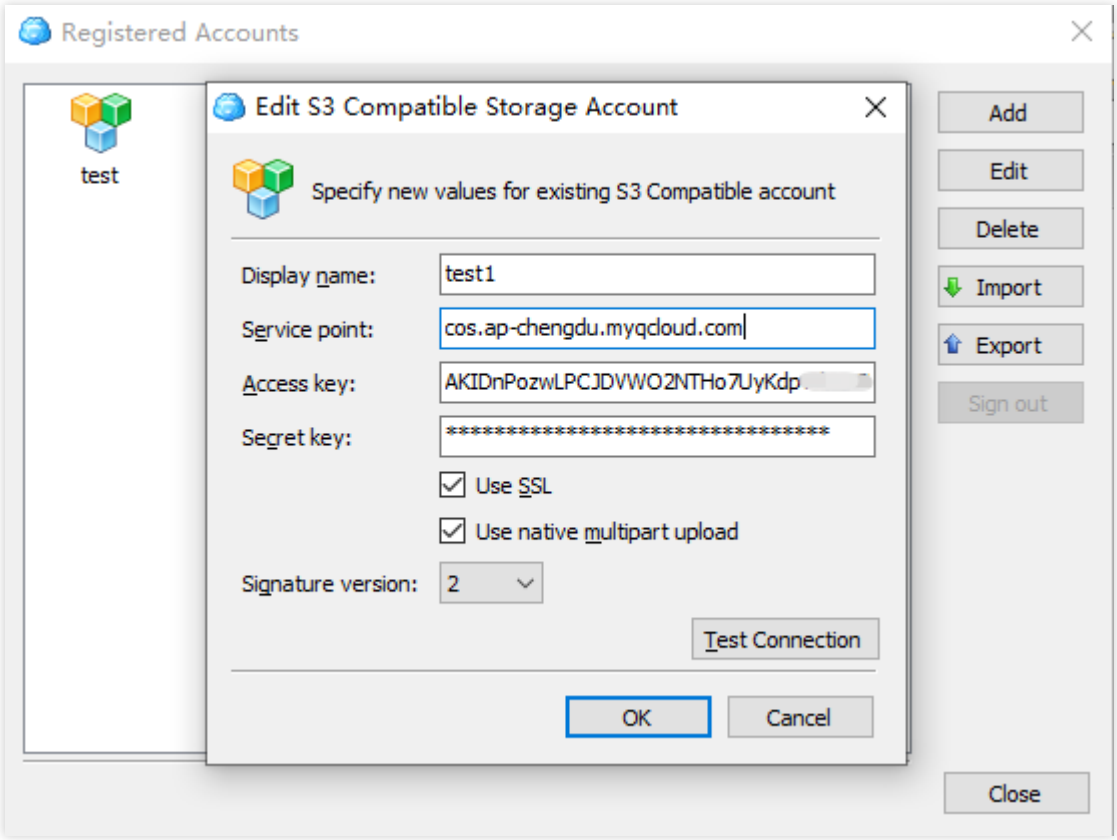
다운로드 주소

CloudBerry 공식 다운로드로 이동하십시오.

설치 및 구성

주의 :
하기 구성 단계는 CloudBerry Explorer Windows v6.3을 예로 들어 설명합니다. 다른 버전의 구성 과정과 약간의 차이가 있을 수 있습니다.

1. 설치 패키지를 더블클릭하고 프롬프트에 따라 설치를 완료합니다.
2. 툴을 열고 S3 Compatible을 선택하고 더블클릭합니다.
3. 팝업 창에서 다음 정보를 설정하고 Test Connection을 클릭하면 연결이 성공적으로 완료됩니다.



다음은 설정 항목에 관한 설명입니다.

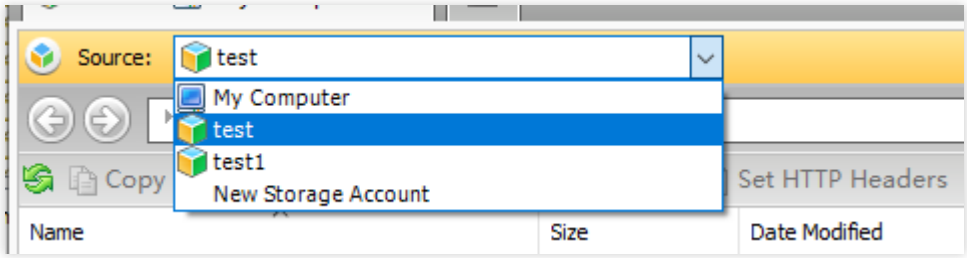
Display name: 사용자 정의 사용자 이름을 입력합니다.

Service point: 형식은 `cos.<Region>.myqcloud.com` 이며, 예를 들어 청두 리전의 버킷에 액세스하려면 `cos.ap-chengdu.myqcloud.com` 을 입력하십시오. 리전 약칭(region)은 리전 및 액세스 도메인을 참고하십시오.

Access key: 액세스 키 정보 SecretId를 입력합니다. Tencent Cloud API 키로 이동하여 생성 및 획득할 수 있습니다.

Secret key: 액세스 키 정보 Secretkey를 입력합니다. Tencent Cloud API 키로 이동하여 생성 및 획득할 수 있습니다.

4. 계정 정보 추가 완료 후 Source에서 이전에 설정한 사용자 이름을 선택하면 해당 사용자 이름 아래에 있는 버킷 리스트를 볼 수 있습니다. 이는 구성이 완료되었음을 의미합니다.



COS 파일 관리

버킷 리스트 조회

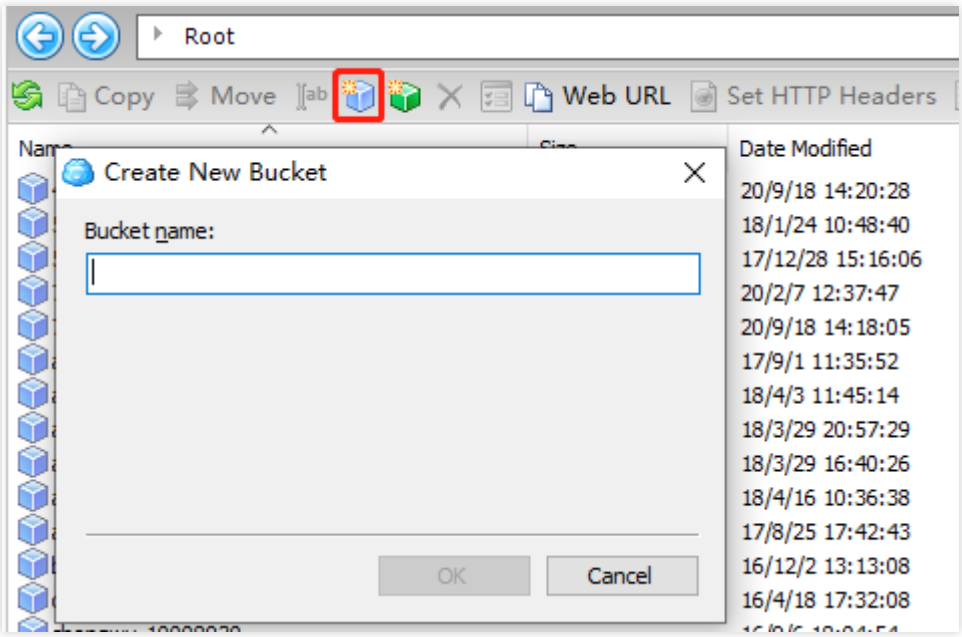
Source에서 이전에 설정한 사용자 이름을 선택하여 사용자 이름 아래의 버킷 리스트를 볼 수 있습니다.

주의 :

Service point에서 구성한 리전에 해당하는 버킷만 볼 수 있습니다. 다른 리전의 버킷을 보려면 **File > Edit Accounts**를 클릭하고 사용자 이름을 선택한 다음 Service point 매개변수를 다른 리전으로 수정합니다.

버킷 생성

이미지의 아이콘을 클릭하고 팝업 창에 전체 버킷 이름을 입력합니다(예: examplebucket-1250000000). 입력이 정확하면 **OK**를 클릭하여 생성을 완료합니다.
버킷의 이름 생성 규칙은 버킷 이름 생성 규칙을 참고하십시오.

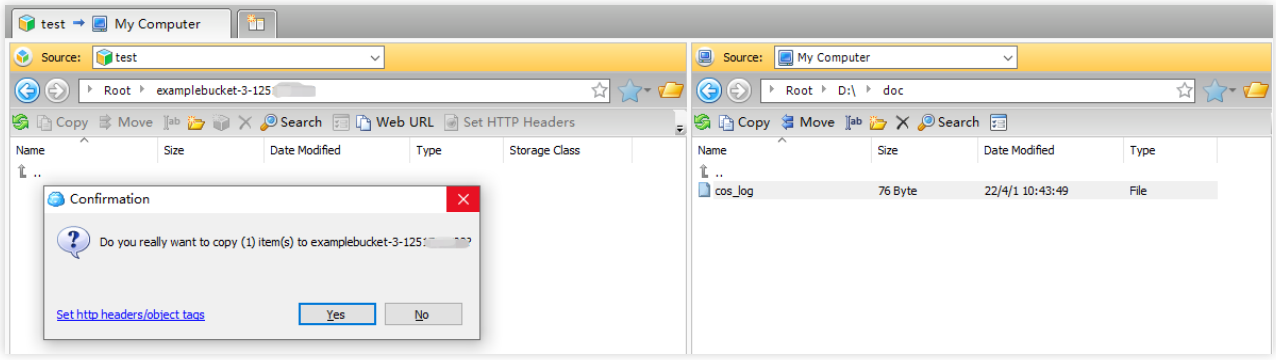


버킷 삭제

버킷 리스트에서 삭제할 버킷을 선택하고 **Delete**를 우클릭하여 삭제합니다.

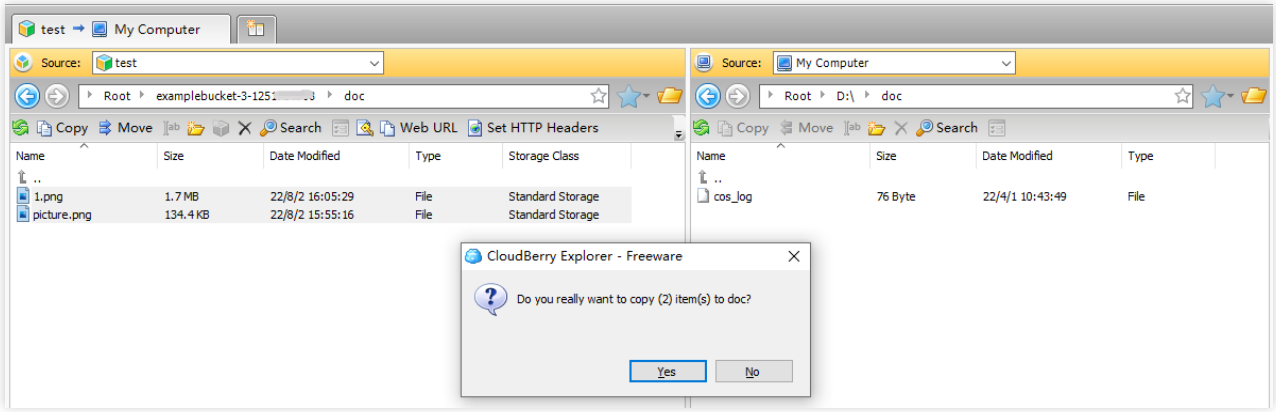
객체 업로드

버킷 리스트에서 업로드할 객체의 버킷 또는 경로를 선택한 후 로컬 컴퓨터에 업로드할 객체를 선택하고 왼쪽 창으로 드래그하면 업로드 작업이 완료됩니다.



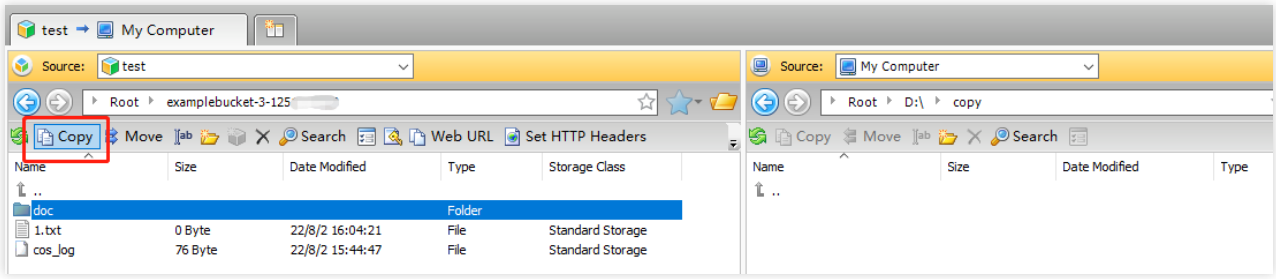
객체 다운로드

왼쪽 창에서 다운로드할 객체를 선택한 후 오른쪽에 있는 로컬 컴퓨터의 폴더로 객체를 드래그하여 다운로드 작업을 완료합니다.



객체 복사

툴의 오른쪽 창에서 복사된 객체 타깃 경로를 선택한 후 왼쪽 창에서 복사할 객체를 선택하고 **Copy**를 우클릭하여 팝업 정보를 확인하면 객체 복사 작업이 완료됩니다.



객체 이름 변경

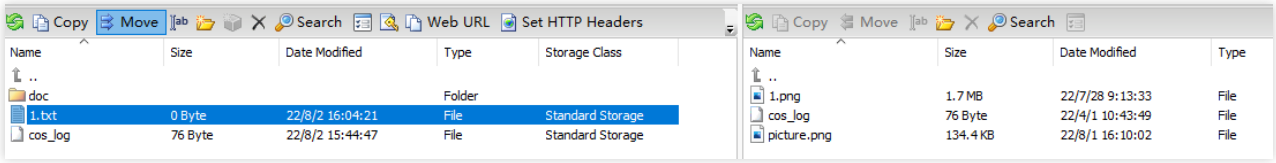
버킷에서 이름을 변경할 객체를 찾아 **Rename**을 우클릭하고 새 이름을 입력합니다.

객체 삭제

버킷에서 삭제하려는 객체를 찾아 **Delete**를 우클릭하여 객체를 삭제합니다.

객체 이동

툴 오른쪽 창에서 이동된 객체 타깃 경로를 선택한 후 왼쪽 창에서 이동할 객체를 선택하고 **Move**를 우클릭하여 팝업 창 정보를 확인하면 객체 이동 작업이 완료됩니다.



기타 기능

상기 기능 외에도 CloudBerry Explorer는 객체 ACL 설정, 객체 메타데이터 조회, 사용자 정의 Headers, 객체 URL 가져오기 등과 같은 다른 기능도 지원합니다. 사용자는 실제 필요에 따라 작업할 수 있습니다.