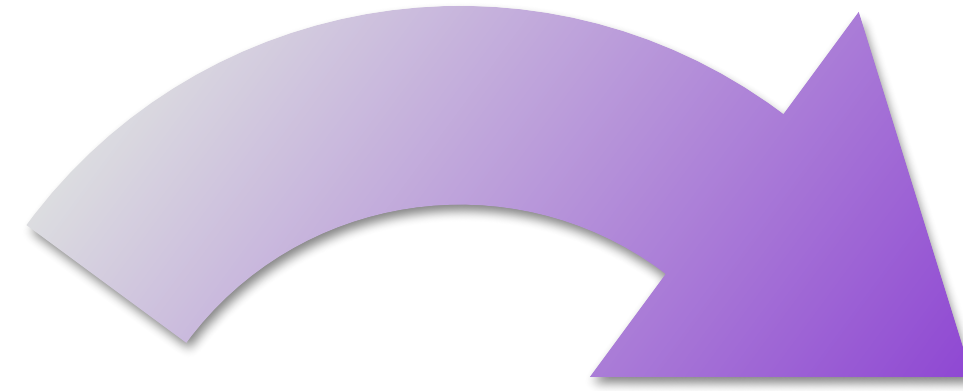


대용량 멀티테넌트 시큐어 하둡 클러스터 운영 경험기

이영곤 Cloud Solution

2019
클러스터
구성



~ 2020.10
운영중

[deview 2019]

대용량 멀티테넌트 시큐어 하둡
클러스터를 **시행착오 없이 만들기**

- 시큐어 하둡의 개요
- 시큐어 하둡의 구성방법
- 운영 중 팁

[deview 2020]

대용량 멀티테넌트 시큐어 하둡
클러스터 **운영 경험기**

- 운영 중 주요 이슈와 개선사항
- 운영 중 주요 트러블슈팅

CONTENTS

1. 데이터플랫폼 C3S

2. 주요이슈와 개선사항

2.1 Cross realm authentication을 이용한 HDFS 클러스터간 데이터 교환

2.2 KnoxSSO를 통해 하둡 웹서버들에 편리하게 접근하기

2.3 하둡 웹서버들의 Access Log 출력 방법

2.4 HDP-3.1 환경에서 Apache Spark를 사용하는 방법

3. 트러블슈팅

3.1 LDAP 커넥션 증가

3.2 SSL 적용 시 JDK버전의 이슈

3.3 Mapreduce & Oozie Server 이슈

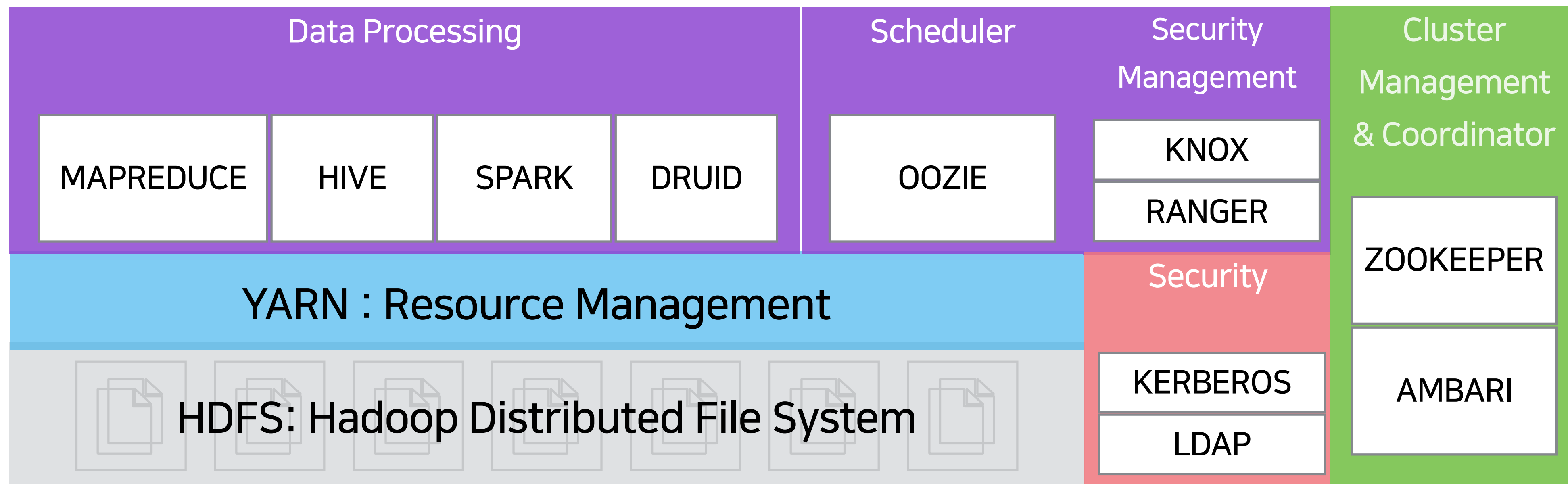
3.4 사내에서 DNS를 운영하고, 하둡관련 DNS 등록 시 유의할 사항

1. 데이터플랫폼 C3S

1.1 데이터플랫폼 C3S

네이버 공통 보안 하둡 클러스터
2000+ nodes
900+ users

Hadoop 3.1.2(kerberized)
HDP-3.1.1
Ambari 2.7.3

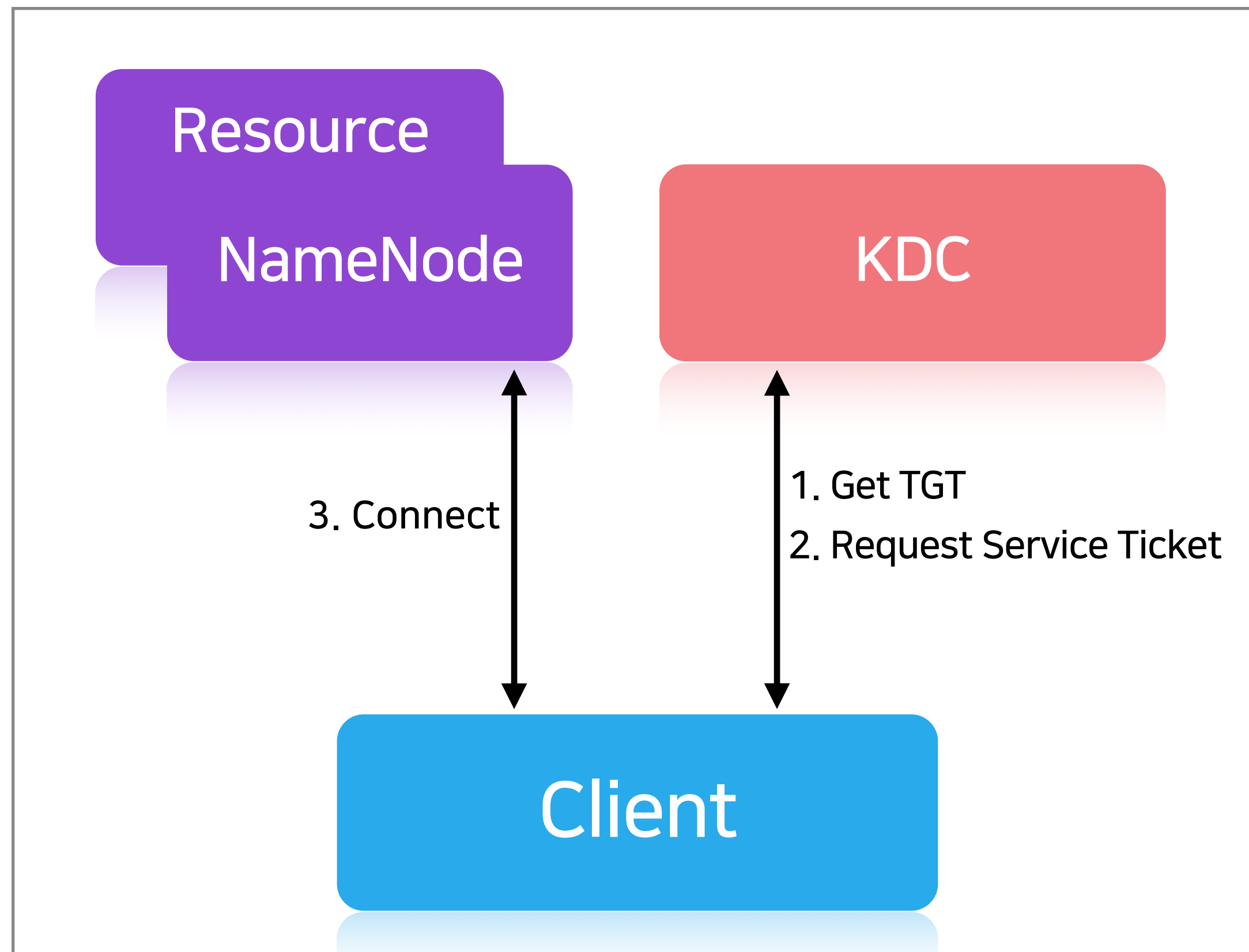


2. 주요이슈와 개선사항

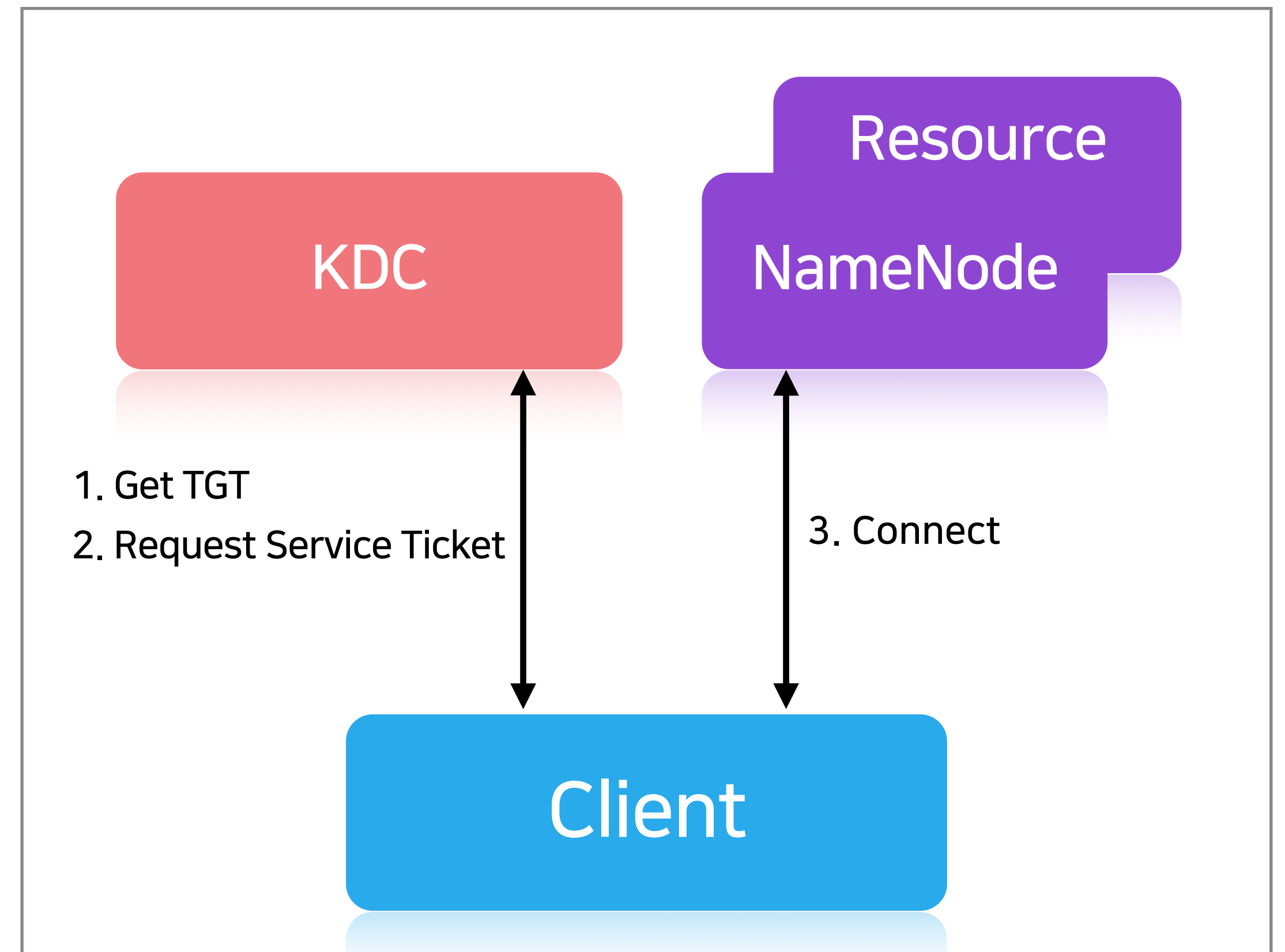
2.1 Cross realm authentication을 이용한 HDFS 클러스터간 데이터 교환

2.1.1 두개의 클러스터에서의 데이터 접근

같은 클러스터의 리소스 접근(o)



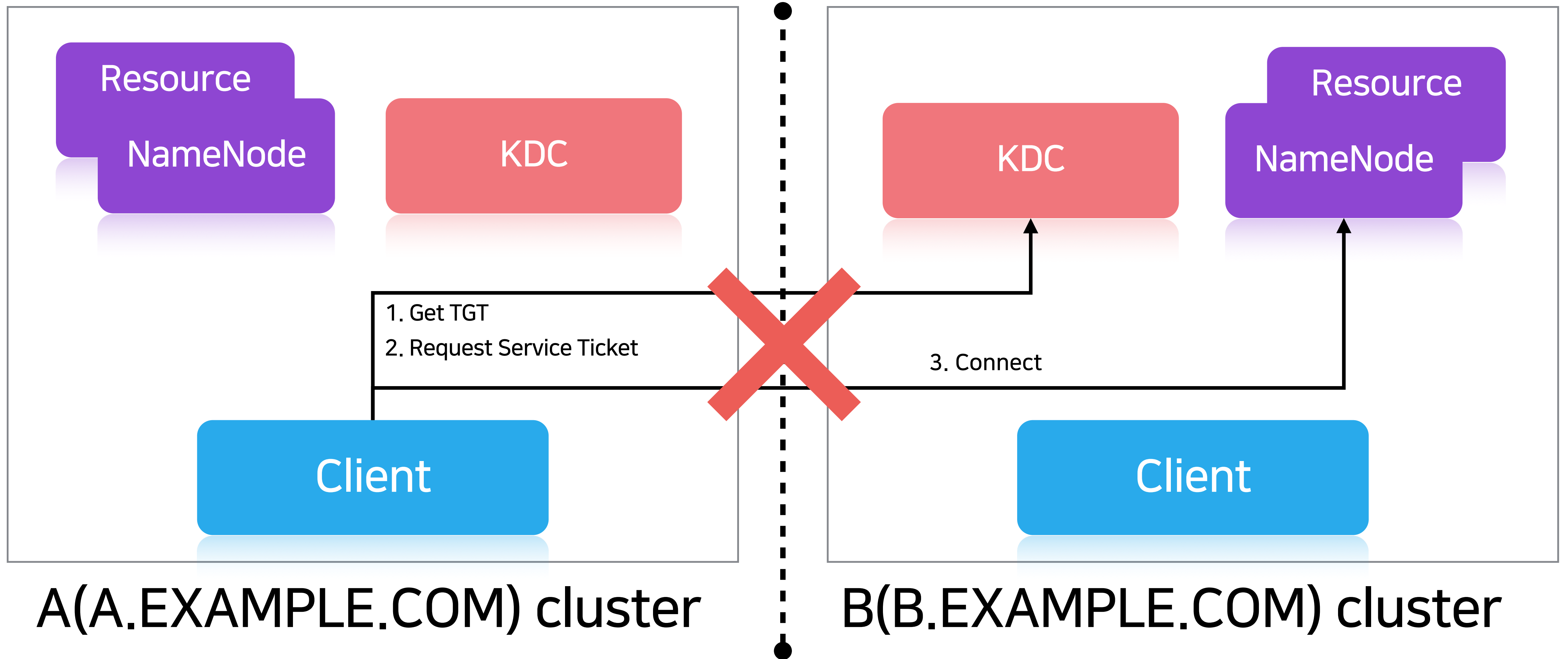
A(A.EXAMPLE.COM) cluster



B(B.EXAMPLE.COM) cluster

2.1.1 두개의 클러스터에서의 데이터 접근

다른 클러스터의 리소스 접근(x)



2.1.2 Cross realm authentication

Kerberos*에서 사용 되는 개념

두개의 클러스터에 각각의 KDC*와 Realm*이 있는 경우, 서로 다른 클러스터의 리소스에 접근을 위한 개념

예를 들면 **다른 클러스터의 HDFS에 접근을 하여** 데이터를 가져와 자신의 클러스터에 넣고자 할 때 사용

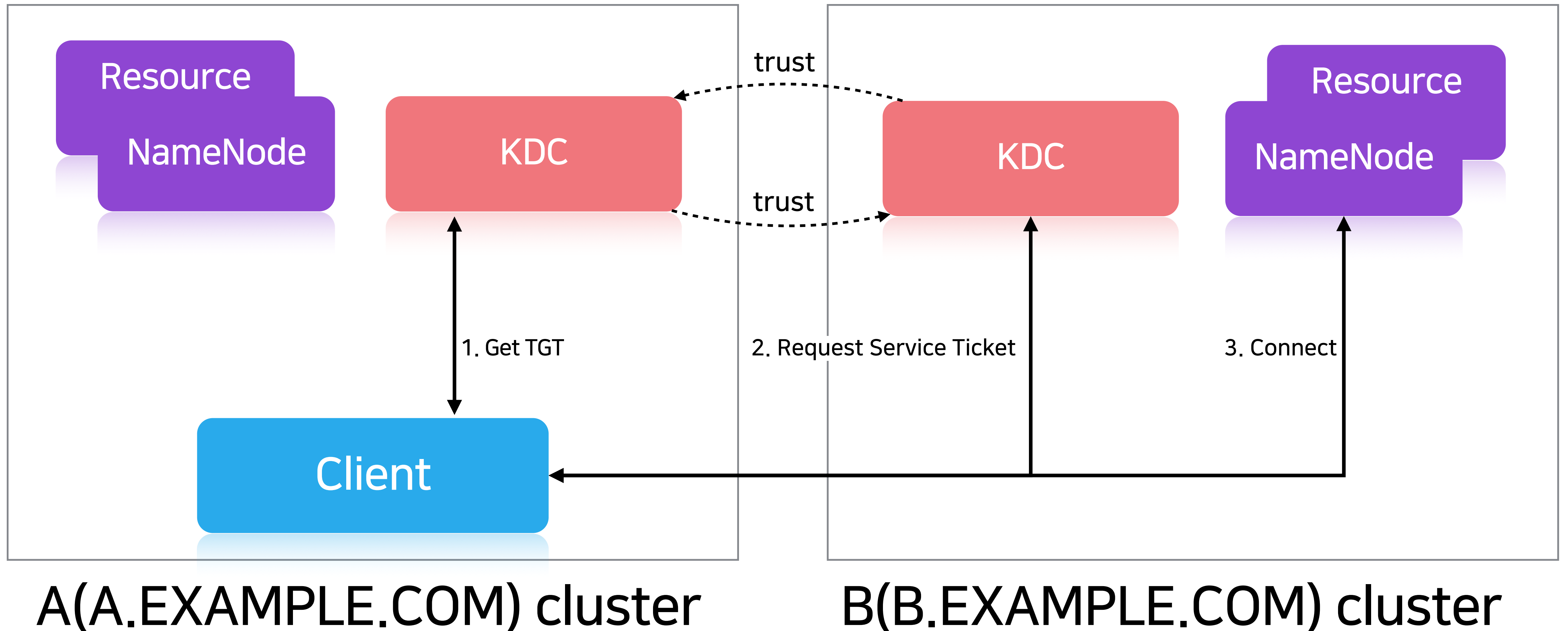
Kerberos: Secure Hadoop 클러스터에서 사용하는 인증수단

KDC: key distribution center의 약어로 kerberos의 구성요소

Realm: Kerberos에서 인증을 제공하는 범위. 대문자의 도메인을 사용(e.g. A.EXAMPLE.COM)

2.1.2 Cross realm authentication

다른 클러스터의 리소스 접근(o)



2.1.3 Trust 설정

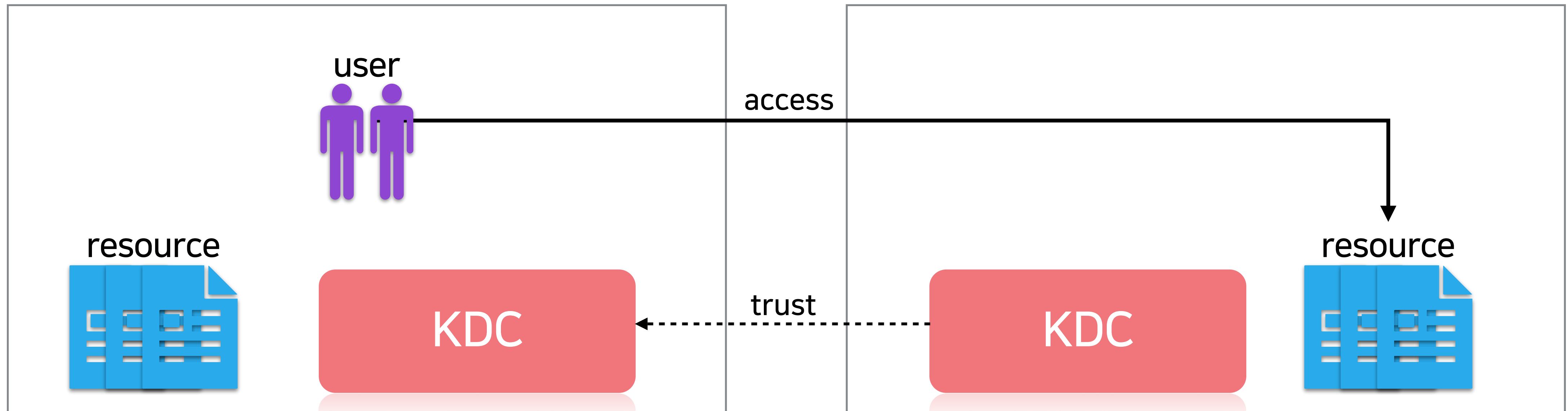
Trust: 한 realm의 사용자가 다른 realm에도 속하는 것처럼 다른 realm의 리소스에 접근할 수 있는 것

설정방법: 두 realm에서 믿을 수 있는 공유 principal를 생성하여 Trust를 설정

2.1.3 Trust 설정(A -> B)

A(A.EXAMPLE.COM) cluster

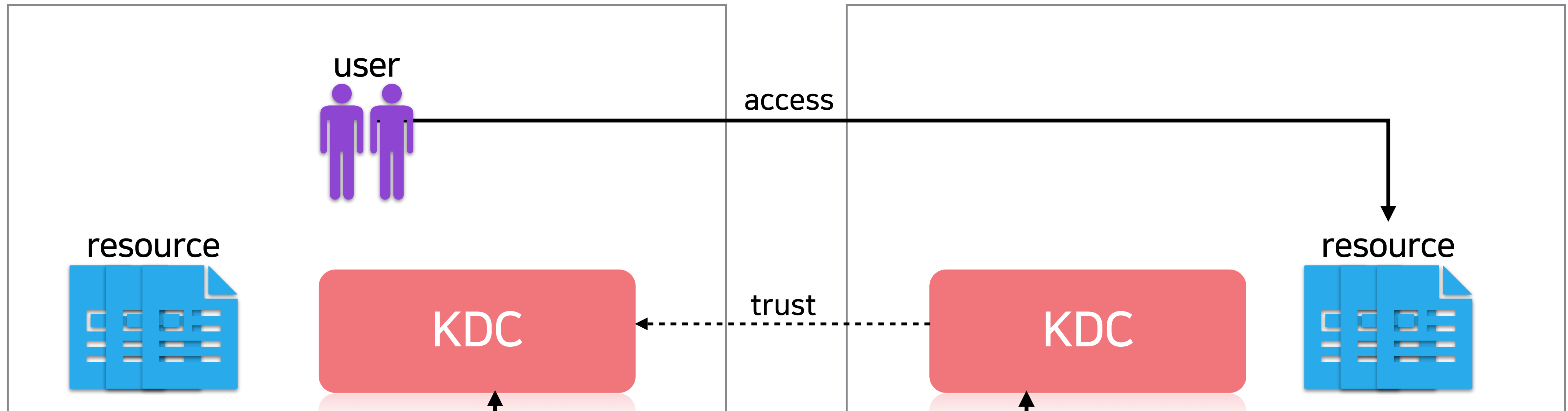
B(B.EXAMPLE.COM) cluster



2.1.3 Trust 설정(A -> B)

A(A.EXAMPLE.COM) cluster

B(B.EXAMPLE.COM) cluster

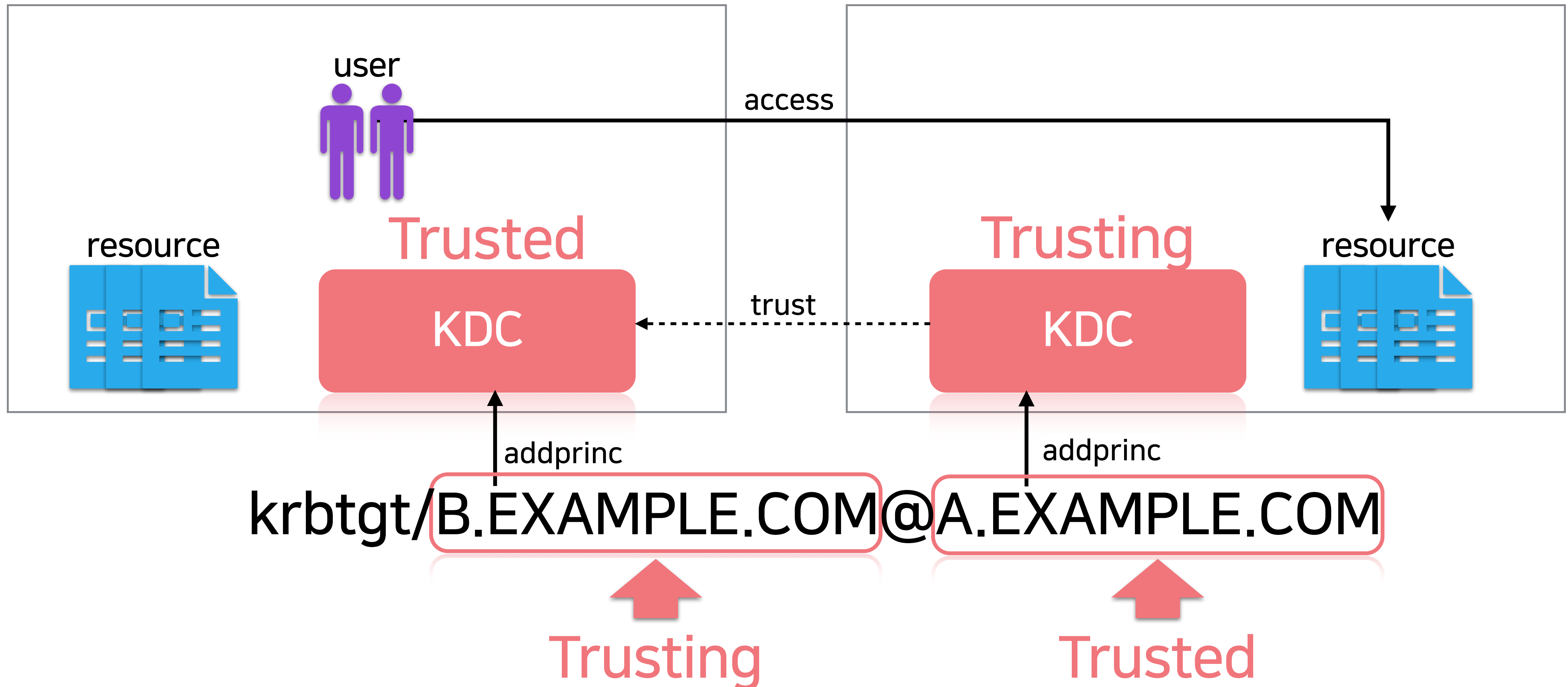


krbtgt/B.EXAMPLE.COM@A.EXAMPLE.COM

2.1.3 Trust 설정(A -> B)

A(A.EXAMPLE.COM) cluster

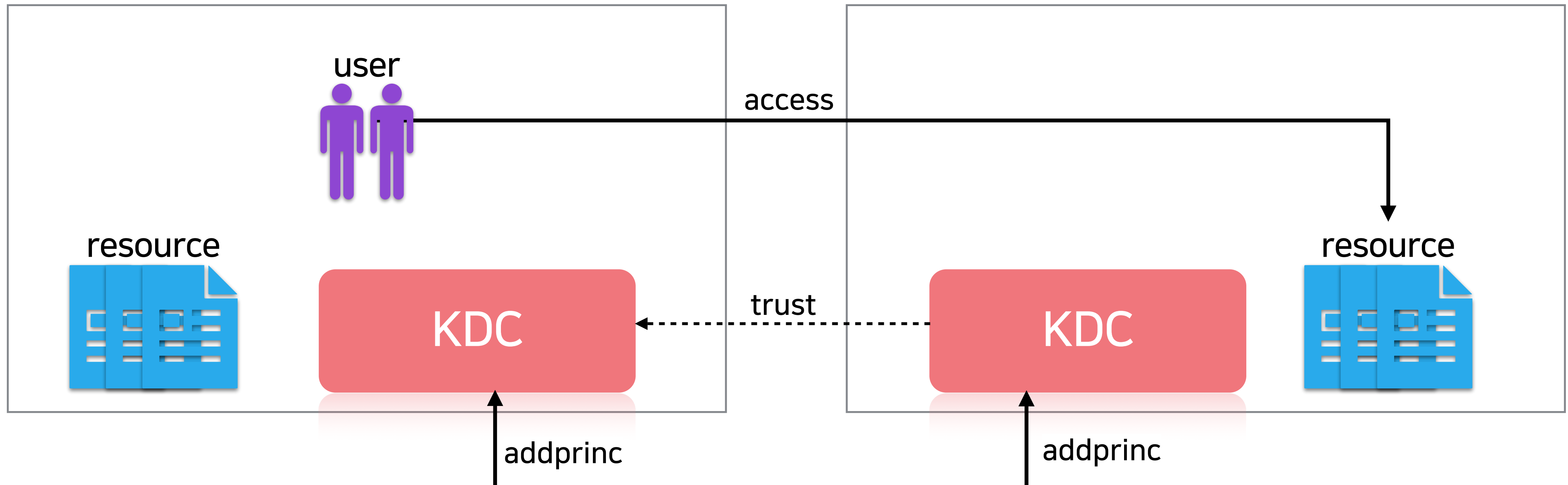
B(B.EXAMPLE.COM) cluster



2.1.3 Trust 설정(A -> B)

A(A.EXAMPLE.COM) cluster

B(B.EXAMPLE.COM) cluster

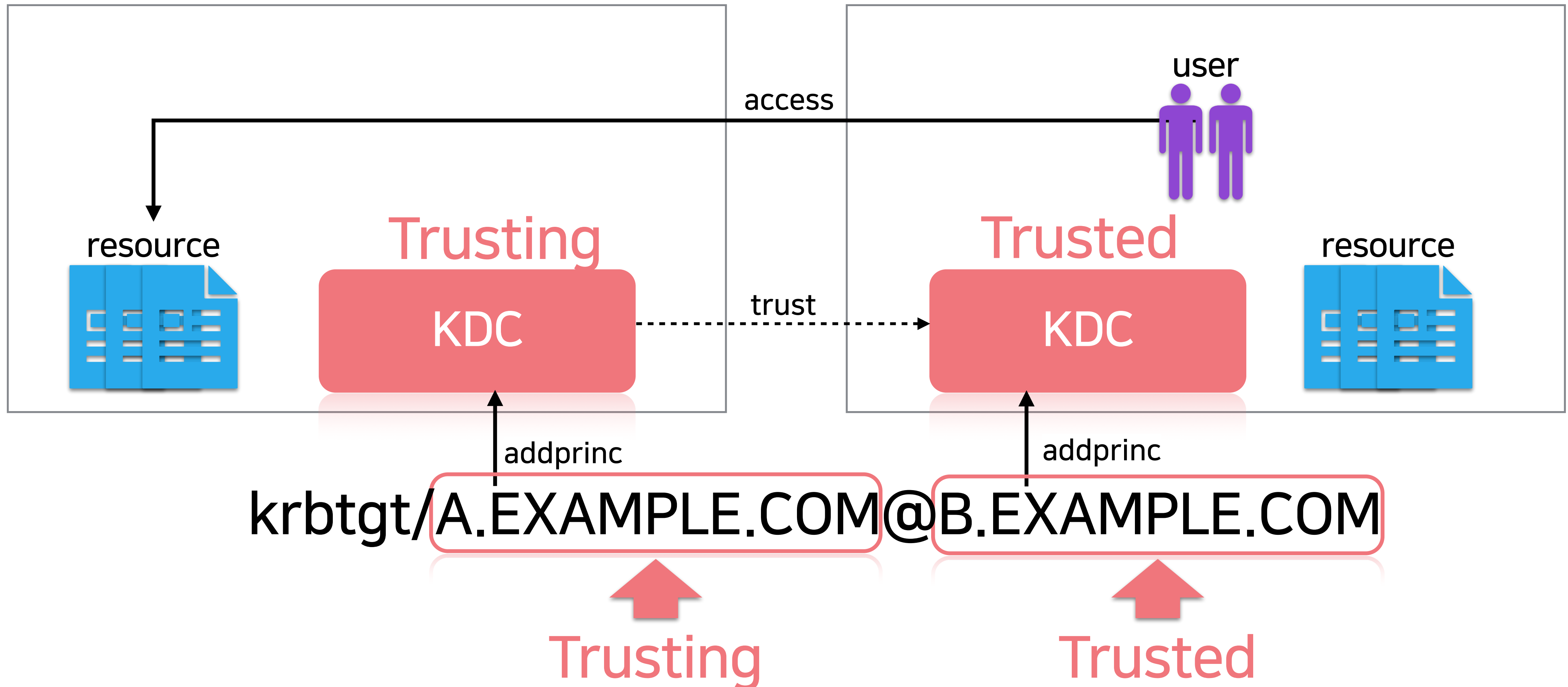


```
root$ kadmin.local
kadmin.local: addprinc krbtgt/B.EXAMPLE.COM@A.EXAMPLE.COM
Enter password for principal "krbtgt/B.EXAMPLE.COM@A.EXAMPLE.COM":
Re-enter password for principal "krbtgt/B.EXAMPLE.COM@A.EXAMPLE.COM":
Principal "krbtgt/B.EXAMPLE.COM@A.EXAMPLE.COM" created.
```


2.1.3 Trust 설정(B -> A)

A(A.EXAMPLE.COM) cluster

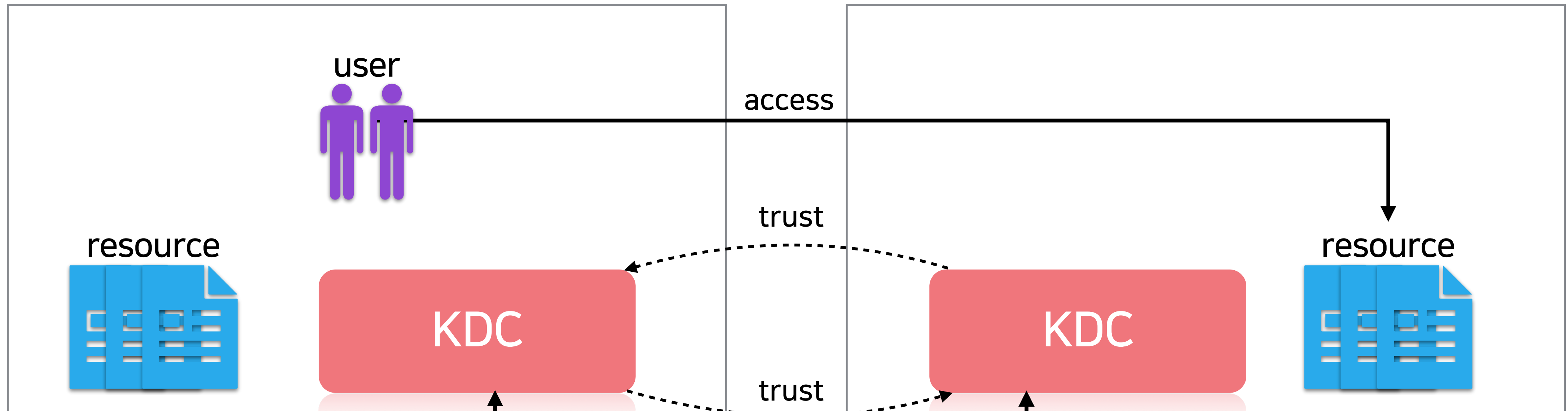
B(B.EXAMPLE.COM) cluster



2.1.3 Trust 설정(A <-> B)

A(A.EXAMPLE.COM) cluster

B(B.EXAMPLE.COM) cluster

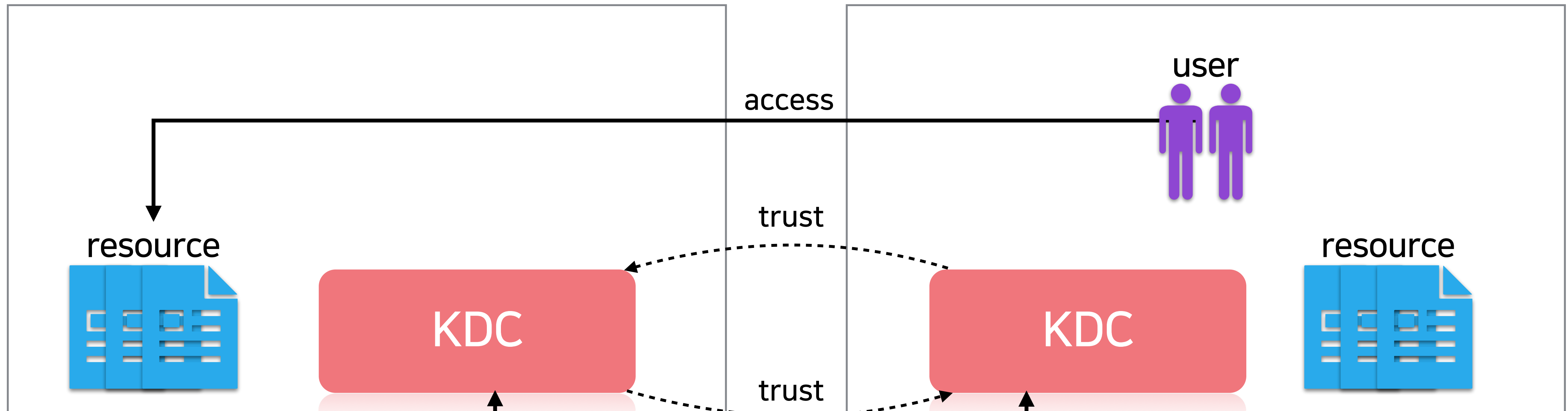


krbtgt/B.EXAMPLE.COM@A.EXAMPLE.COM
krbtgt/A.EXAMPLE.COM@B.EXAMPLE.COM

2.1.3 Trust 설정(A <-> B)

A(A.EXAMPLE.COM) cluster

B(B.EXAMPLE.COM) cluster



krbtgt/B.EXAMPLE.COM@A.EXAMPLE.COM

krbtgt/A.EXAMPLE.COM@B.EXAMPLE.COM

2.1.3 Trust 설정(addprinc시 주의점)

같은 암호화 타입

같은 비밀번호

같은 KVNO(Key Version Number)

- KVNO는 principal 생성 시 1로 설정되고, 비밀번호 변경 혹은 키탭 생성 시 1씩 증가
- 즉 각 KDC에 공유 principal(e.g. krbtgt/B.EXAMPLE.COM@A.EXAMPLE.COM)의 추가 시 비밀번호를 변경하거나 keytab을 생성은 지양

MIT의 가이드에 따라 최소 24자리 이상의 비밀번호를 사용하는것을 추천

2.1.4 Pre-Authentication(PREAUTH) 확인

Pre-Authentication: Brute-Force* 공격을 막기위해 도입된 표준 (RFC 6113)

즉 보안이 좀 더 강화된 설정
각각의 principal을 기준으로 부여가능
- addprinc 시 +requires_preauth 옵션

2.1.4 Pre-Authentication(PREAUTH) 확인

Principal Name별 PREAUTH 절차

- UPN*(사용자) + PREAUTH - 로그인 시점에 PREAUTH 절차를 진행
- SPN*(서비스) + PREAUTH - 해당 SPN의 티켓을 발급받으려면 로그인한 principal 이 PREAUTH로 로그인이 되어 있어야 함

즉 namenode의 service principal에 PREAUTH가 부여가 되어있다면 접속하려는 모든 사용자(principal)에도 PREAUTH가 부여되어 있어야 함

UPN : user principal name

SPN : service principal name

2.1.5 접근하려는(Trusted) 클러스터의 하둡설정

리소스매니저의 설정변경 필요

- 리소스매니저에서 상대방의 네임노드에 접속해서 Delegation token의 Renew를 하기 때문에 상대방의 네임노드, KDC, HA정보가 필요

리소스 매니저의 /etc/hosts 수정(appendix 참고)

- 접근해야 하는 네임노드 호스트정보 추가

리소스 매니저의 /etc/krb5.conf 수정(appendix 참고)

- 접근해야 하는 KDC정보 추가

리소스매니저의 hdfs-site.xml 수정(appendix 참고)

- 접근해야 하는 네임노드의 HA정보 추가

리소스매니저 재시작

2.1.5 접근을 허가한(Trusting) 클러스터의 하둡설정

네임노드의 core-site.xml 수정(appendix 참고)

- Realm별 사용자명 변환 룰 추가

네임노드 재시작

KMS의 kms-site.xml 수정(appendix 참고)

- Realm별 사용자명 변환 룰 추가

KMS 재시작

2.1.5 접근을 허가한(Trusting) 클러스터의 하둡설정

Realm별 사용자명 변환 룰 예시

```
RULE:[1:$1@$0](.*@A.EXAMPLE.COM)s/(.+)_/_cross_realm_cluster_a_$1/
```

- 위 설정 후 A.EXAMPLE.COM의 foo 사용자가 접속 시 _cross_realm_cluster_a_foo로 변환

Ranger를 통한 접근권한 부여(부여 시 위의 사용자명 변환 룰을 참고하여 부여)

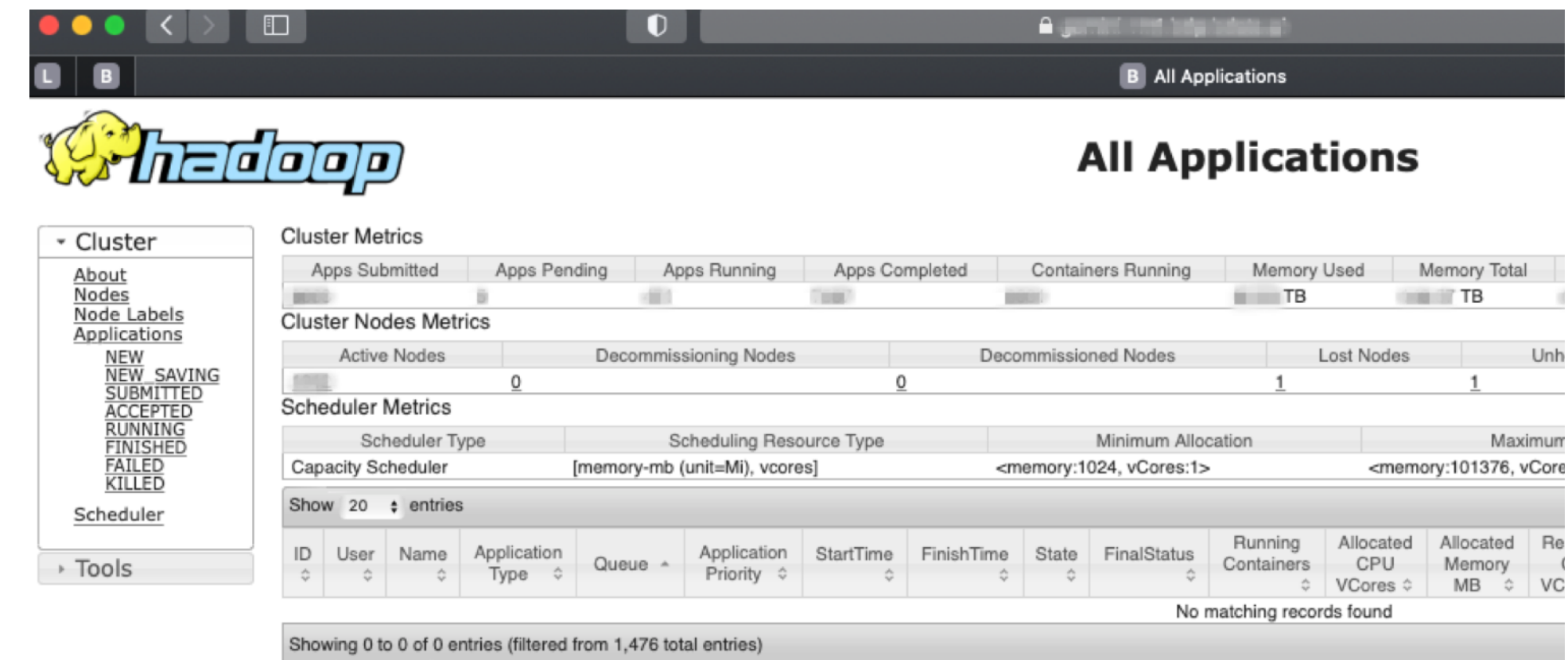
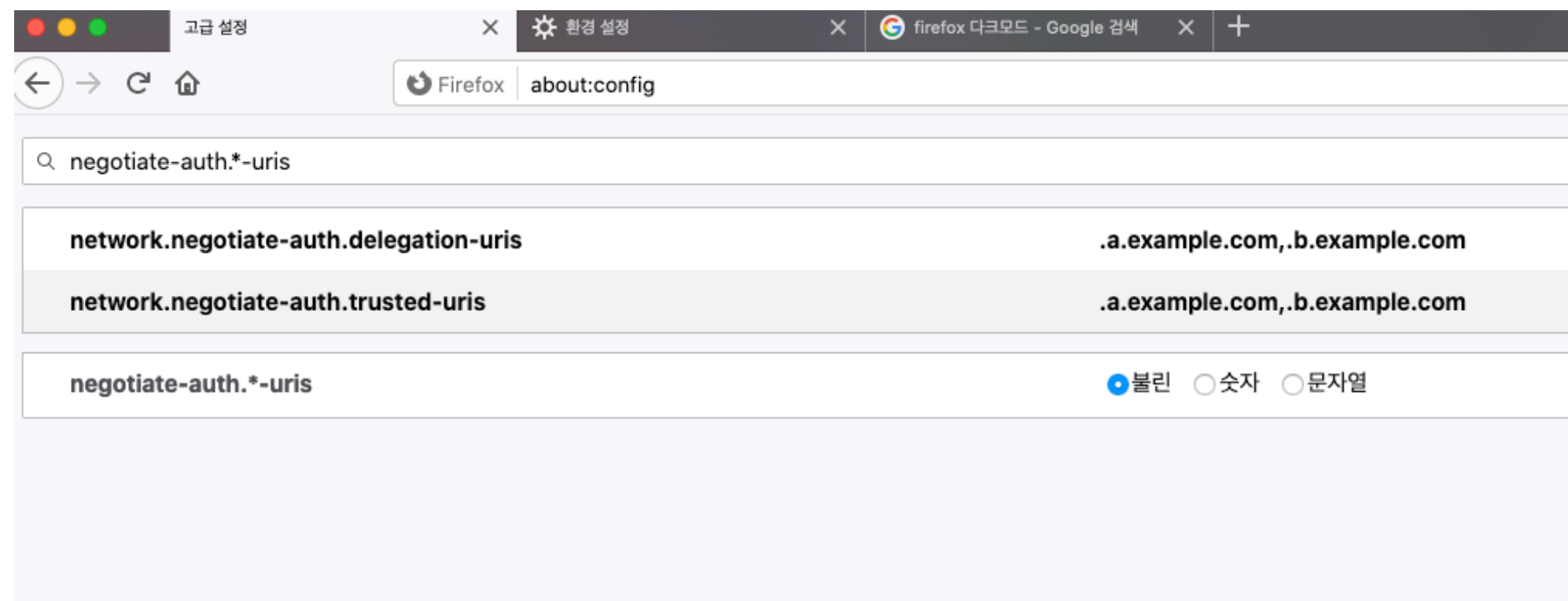
2.1.6 요약

1. Trust 설정
2. PREAUTH 확인
3. Trusted 클러스터의 하둡설정
4. Trusting 클러스터의 하둡설정
5. Cross realm authentication 완성

2.2 KnoxSSO를 통해 하둡 웹서버들에 편리하게 접근하기

2.2.1 KnoxSSO도입 전(SPNEGO) Web UI 접근 절차

1. 하둡 클러스터의 /etc/krb5.conf를 로컬에 복사
2. 브라우저에서 SPNEGO*가 되도록 설정(appendix 참고)
3. 셸에서 kinit 명령으로 kerberos 로그인
4. Web UI에 접속(e.g. 리소스매니저, 네임노드)



SPNEGO: HTTP 프로토콜에서 kerberos를 지원하는 프로토콜

2.2.2 KnoxSSO도입 전 이슈

환경에 따라 설정 방법이 제각각

- OS환경(맥/리눅스/윈도우 버전)
- 브라우저 환경(웨일, 크롬, 사파리, 파이어폭스 등)

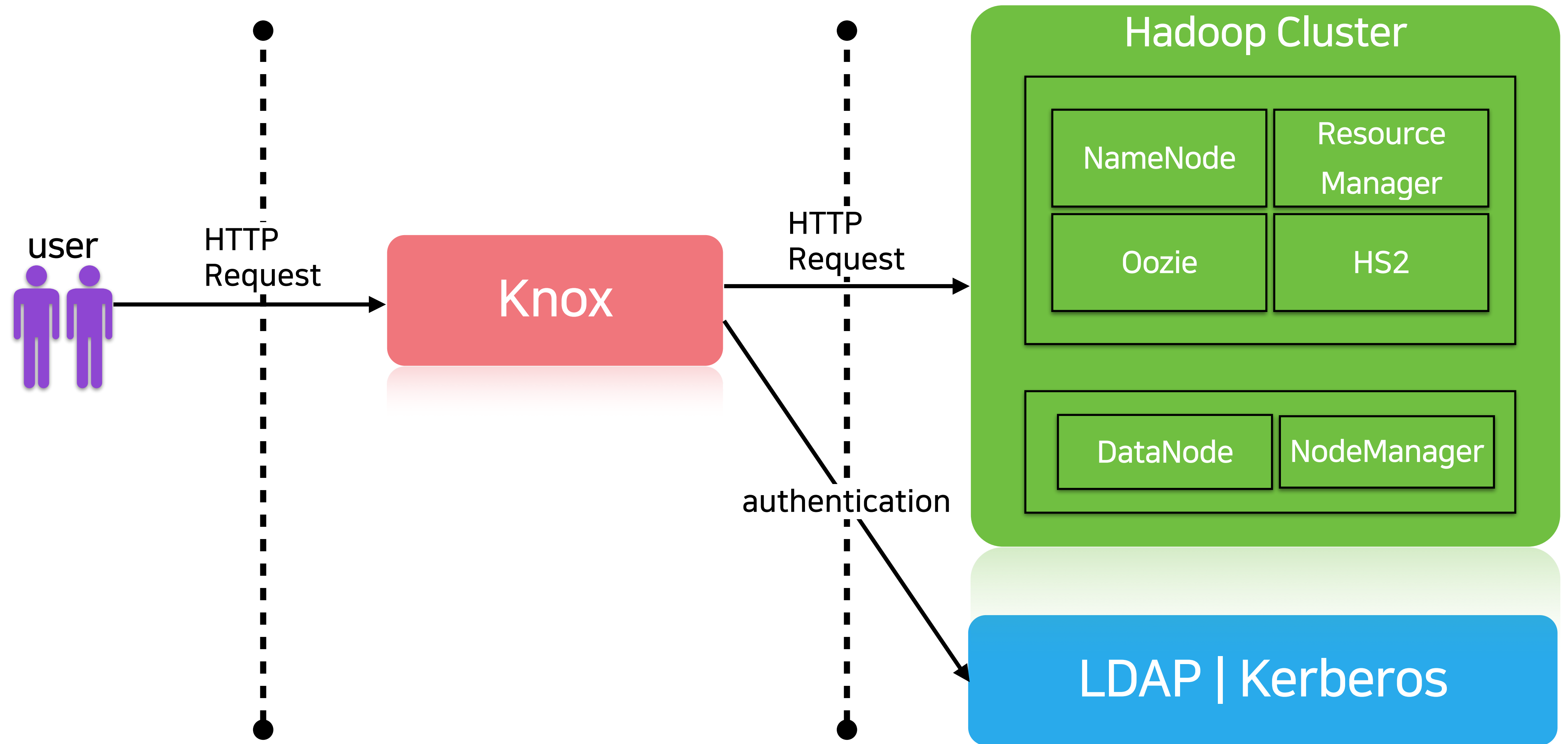
문제점을 파악하기가 어려움

모바일에서는 접근 불가

접근불가에 대한 다양한 문의가 발생했고, 환경과 문제점이 제각각으로 다양했습니다.

2.2.3 Knox

하둡 에코시스템의 Rest API와 Web UI를 위한 게이트웨이



2.2.4 KnoxSSO 원리

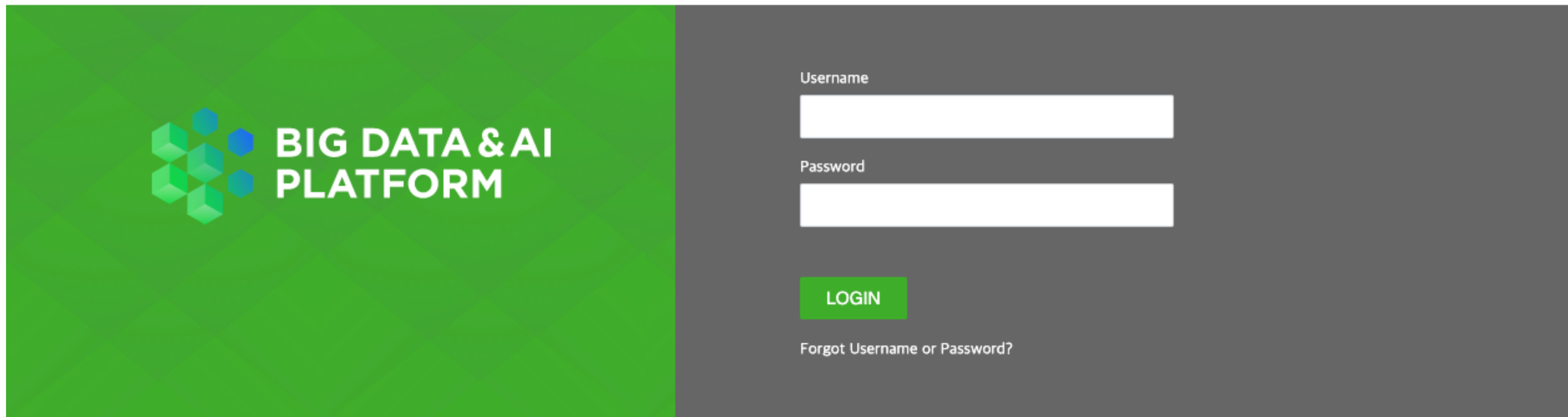
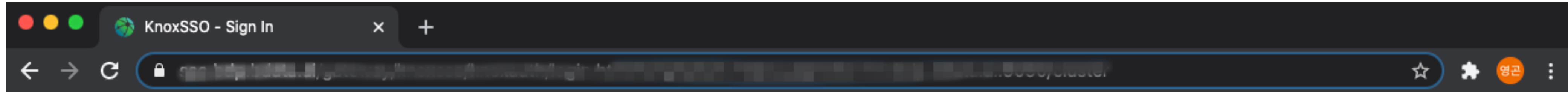
인증방식을 kerberos에서 JWT토큰방식으로 변경
웹서버 접근 시 쿠키에 hadoop-jwt 토큰이 있는지 체크해서 없을 경우
Knox로 redirect
Knox에서는 hadoop-jwt 토큰을 생성해서 웹서버로 다시 redirect
이후 hadoop-jwt 토큰을 이용해서 인증

Web UI에만 적용을 하였습니다.

Rest API는 그대로 SPNEGO를 사용 중 입니다.

2.2.5 KnoxSSO도입 후 Web UI 접근 절차

1. Web UI에 접속
2. LDAP ID/PW 입력 후 로그인



2.2.6 설치 및 설정 방법

1. Ambari를 통한 설치
2. public key export(appendix 참고)
3. 하둡 설정(appendix 참고)
4. Spark 설정(appendix 참고)
5. Oozie 설정(appendix 참고)

2.2.7 Knox의 SSL 인증서

Knox는 JWT 토큰을 만드는 개인키와 SSL에서 사용되는 개인키가 동일
JWT 공개키는 모든 서버 전파된 상황

인증서 교체(=JWT 토큰을 만드는 개인키 교체)

- 모든 서버의 JWT 공개키 변경 필요
- 즉 모든 노드의 공개키 변경 후 재시작
- 교체하는 동안 재시작 하지 않는 노드에는 SSO 접속이 불가
- 안정적으로 교체하기는 현실적으로 불가능

인증서는 self signed를 사용

2.2.8 미적용 컴포넌트 및 issue

ATS 1.5에는 미적용

- 적용할 경우 ATS에서 발급하는 Delegation token을 정상적으로 가져올 수 없기 때문에 문제가 발생
- 이로인해 tez UI 접근 시 kerberos 인증 필요

데이터노드 Web-UI 접속할때 SSO 를 위한 redirect 시 데이터노드주소가 http 로 지정되어서 로그인후에 데이터노드 접속이 되지 않음

- 보통은 네임노드를 통해서 로그인 후 데이터노드를 접근하기 때문에 별도의 조치를 취하지 않음

2.3 하둡 웹서버들의 Access Log 출력 방법



2.3.1 하둡 웹서버들의 Access Log

필요성

- 네임노드나 리소스매니저의 Rest API요청이 부하가 심할 경우 해당 데몬에 직접 영향
- 또한 네임노드나 리소스매니저의 Rest API가 응답을 하지 못하면 다른 연계 시스템에서 장애가 발생할 수 있음
- Rest API의 빈번한 요청, 이상한 패턴의 요청을 찾아내서 빠르게 대응하여 장애를 방지

출력 방법

- 하둡의 웹서버들은 모두 jetty기반으로 동작을 함
- 하둡쪽에 문서화가 잘 되어있지는 않지만 log4j의 설정을 통해서 모든 웹서버들의 Access Log를 출력 가능

2.3.2 네임노드 적용방법

Advanced hdfs-log4j수정(HDP 기준)

```
log4j.logger.http.requests.namenode=INFO,namenoderequestlog  
log4j.appender.namenoderequestlog=org.apache.hadoop.http.HttpRequestLogAppender  
log4j.appender.namenoderequestlog.FileName=${hadoop.log.dir}/access-namenode-yyyy_mm_dd.log  
log4j.appender.namenoderequestlog.RetainDays=3
```

네임노드 재시작

위와 같이 설정할경우 날짜 별 로테이트 + 3일간의 로그를 보관

2.3.2 리소스매니저 적용방법

Advanced yarn-log4j수정(HDP 기준)

```
log4j.logger.http.requests.resourcemanager=INFO,resourcemanagerrequestlog  
log4j.appender.resourcemanagerrequestlog=org.apache.hadoop.http.HttpRequestLogAppender  
log4j.appender.resourcemanagerrequestlog.FileName=${hadoop.log.dir}/access-resourcemanager-yyyy_mm_dd.log  
log4j.appender.resourcemanagerrequestlog.RetainDays=3
```

리소스매니저 재시작

위와 같이 설정할경우 날짜 별 로테이트 + 3일간의 로그를 보관

2.3.3 Access Log량이 많을 경우

날짜 별 로테이트 + 보관도 부담이 될 수 있음
즉 하루의 로그만으로도 디스크 full이 발생가능
그 경우 아래와 같이 설정

```
log4j.logger.http.requests.namenode=INFO,namenoderequestlog  
log4j.appender.namenoderequestlog=org.apache.hadoop.http.HttpRequestLogAppender  
log4j.appender.namenoderequestlog.FileName=${hadoop.log.dir}/access-namenode.log  
log4j.appender.namenoderequestlog.RetainDays=1
```

이후 cron등을 통해 주기적으로 수동 로테이트를 수행
현재 하둡에서는 날짜별 로테이트/보관만 지원(크기 제공 x)
나머지 컴포넌트들의 설정(appendix 참고)

2.4 HDP-3.1 환경에서 Apache Spark를 사용하는 방법

2.4.1 HDP의 SPARK

HDP Spark

- Apache Spark를 베이스로 하여 HDP에서 기능개선/버그패치등의 변경을 한 Spark
- HDP를 통해서 Spark를 설치할 경우 기본적으로 HDP Spark가 설치
- HDP Spark는 현재 cloudera에서 관리

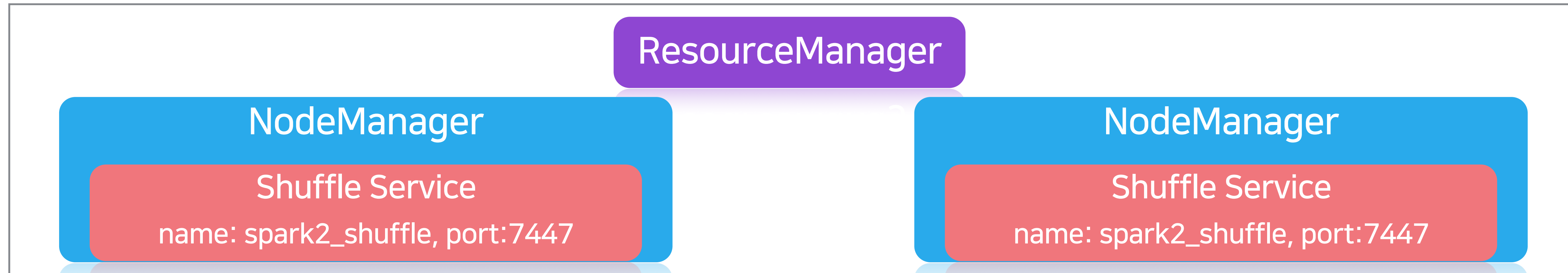
Apache Spark 호환

- Shuffle service를 제외한 대부분의 구조적인 기능은 거의 동일
- Dynamic Allocation*기능을 제외한 모든 기능은 설정을 통해 사용가능

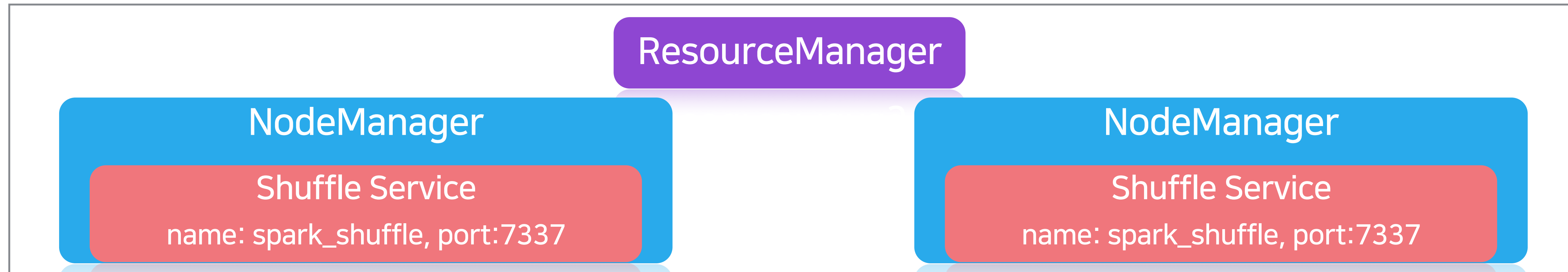
Dynamic Allocation: Spark 작업의 부하에 따라 동적으로 Executor를 할당/해제 하는 기능

2.4.2 HDP Vs. Apache Spark(Shuffle Service)

HDP Spark



Apache Spark



2.4.3 Apache Spark 작업의 제출방법

1. 임의의 스파크버전 다운로드

2. SPARK_SUBMIT_OPTS에 설정 추가

```
export SPARK_SUBMIT_OPTS="-Dhdp.version=x.x.x"
```

3. 작업 제출시 설정 추가

```
--conf spark.driver.extraJavaOptions=-Dhdp.version=x.x.x
```

```
--conf spark.yarn.am.extraJavaOptions=-Dhdp.version=x.x.x
```

2.4.4 Apache Spark의 Dynamic Allocation 지원

Dynamic Allocation은 Shuffle service를 통해서 지원이 가능

- Shuffle service를 사용해야 동적으로 Executor가 해제될 때 Shuffle data의 유실이 발생하지 않음

Apache Spark의 Dynamic Allocation 지원하는 방법

1. Apache Shuffle service 추가
2. HDP Shuffle service를 사용하도록 코드수정

2.4.4 Apache Spark의 Dynamic Allocation 지원

Apache Shuffle service의 추가를 통한 Dynamic Allocation 지원

1. 모든 노드매니저에 Apache Spark의 spark-2.x.x-yarn-shuffle.jar파일 배포, 혹은 패키징하여 설치
2. yarn-site.xml 수정(appendix 참고)
3. 모든 노드매니저 재시작

2.4.4 Apache Spark의 Dynamic Allocation 지원

코드 수정을 통한 Dynamic Allocation 지원

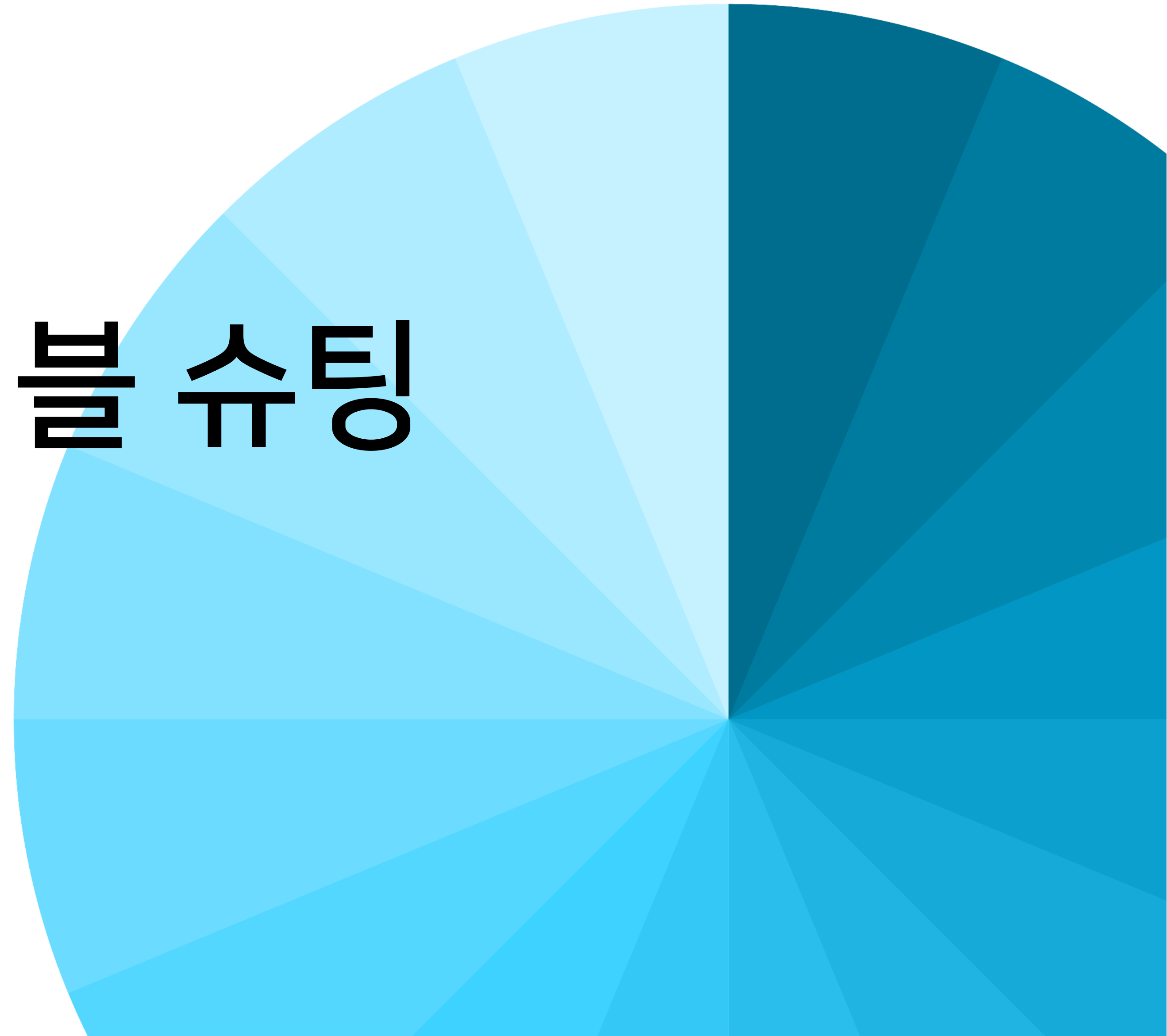
HDP Spark의 shuffle service를 사용하도록 아래의 코드를 수정
ExecutorRunnable.scala 파일 수정

```
-ctx.setServiceData(Collections.singletonMap("spark_shuffle", secretBytes))  
+ctx.setServiceData(Collections.singletonMap("spark2_shuffle", secretBytes))
```

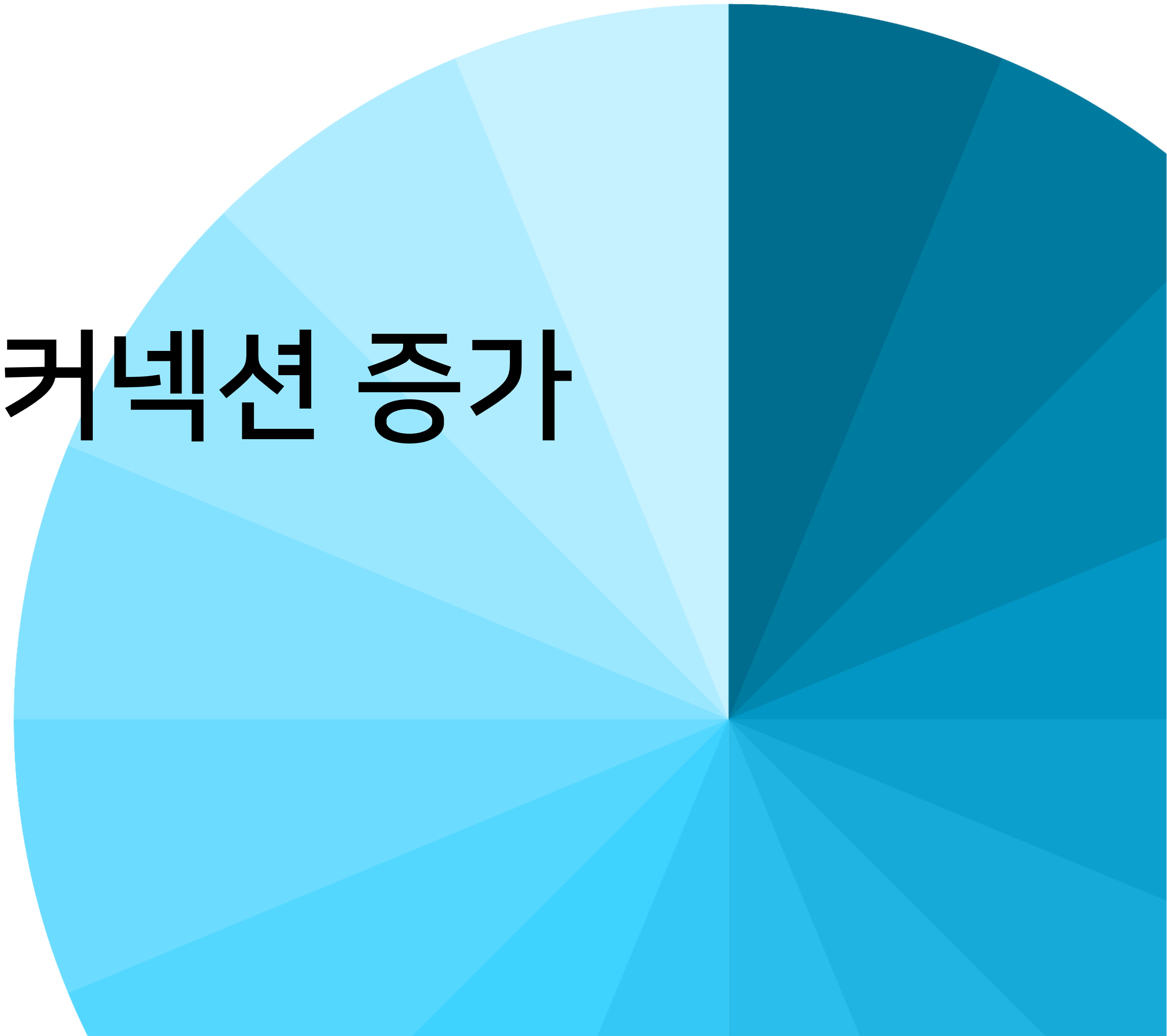
빌드 하여 사용

빌드를 해야하는 단점이 있지만 노드매니저의 재시작은 수행하지 않아도 됨

3. 트러블 슈팅



3.1 LDAP 커넥션 증가



3.1.1 LDAP(Lightweight Directory Access Protocol)

디렉터리 서비스

현실 세계의 주소록을 컴퓨터로 옮겨놓은 프로토콜

사용자 계정과 그룹관리를 위해 사용

시큐어 하둡에서 필수는 아니지만 클러스터 운영의 편리성을 위해 사용

3.1.2 LDAP 커넥션

LDAP 커넥션: 고정 커넥션 + 사용자 커넥션

고정 커넥션: 노드매니저의 수(sssd*의 커넥션)

사용자 커넥션: 사용자 요청에 따른 LDAP 커넥션

커넥션 이슈(Centos7.x 기준)

- OpenLDAP을 별도의 설정없이 설치할 경우 동시 커넥션은 1024로 제한
- 즉 2대로 운용을 하더라도 노드매니저를 2천대이상 증설이 불가

3.1.3 OpenLDAP 동시 커넥션 증가(Centos 7.x기준)

nofile(number of file)의 증가를 통해 동시 커넥션을 늘릴 수 있음
slpad(OpenLDAP)은 시스템 데몬이기 때문에 `/usr/lib/systemd/system/slapd.service` 파일에 아래와 같은 설정의 추가가 필요

```
[Service]
LimitNOFILE=8192
```

추가 후 데몬 재시작

```
[root ~]$ systemctl daemon-reload
[root ~]$ systemctl restart slapd
```

아래 명령을 통해 확인 가능

```
[root ~]$ ps -ef | grep slapd | grep -v grep
ldap      129994      1 11 Sep07 ?        3-10:31:55 /usr/sbin/slapd -u ldap -h ldapi:/// ldap:///
[root ~]$ cat /proc/129994/limits | grep "open files"
Max open files      8192              8192              files
```

3.1.4 OpenLDAP 동시 커넥션 증가 시 주의사항

/etc/security/limits.conf 파일의 수정을 통해서는 시스템 데몬의 **nofile** 변경불가

- 해당 파일은 PAM을 통해 로그인 한 사용자들의 리소스만 제한이 가능
- 즉 시스템 서비스에는 적용되지 않음

런타임 중 nofile 변경(prlimit)시 적용되지 않음

- OpenLDAP은 최초 기동 시 nofile을 참고하여 동시 커넥션을 설정하기 때문에 런타임 중 nofile의 변경을 해도 적용되지 않음

3.2 SSL 적용 시 JDK버전의 이슈



3.2.1 SSL 적용

보안을 위해 하둡의 모든 웹서버에서 https 적용
Https 적용 시 암호화를 위해 TLS(SSL) 프로토콜을 사용
JDK를 사용할 경우 sun.security.ssl.* 사용

3.2.2 JDK SSL session cache

관련 구현체

`sun.security.ssl.SSLSessionContextImpl.java`

`sun.security.ssl.SSLSessionImpl.java`

`sun.security.ssl.SSLSessionId.java`

`sun.security.util.Cache.java`

SSL session cache의 역할: session ID와 session 정보를 저장하고, 재사용을 통해 별도의 handshake 없이 빠르게 커넥션을 맺을 수 있도록 함

3.2.3 JDK SSL session cache issue

SSL session cache의 default 설정

SSL session cache size = 0(infinite)

SSL session cache timeout = 24hour

발생하는 문제

- 큰 힙 사이즈(e.g. NameNode)를 사용하고, 대량의 요청이 있을 경우에는 cache entry가 무한으로 쌓이고 이로 인해 get/put 요청 시 blocked가 발생하고, 네임노드 UI와 webhdfs의 응답불가
- 또한 full GC 발생 시 수십초의 STW를 야기하고, 이것은 서비스의 장애로 이어짐

JDK 8u261버전에서 패치(기본 cache size를 20480으로 변경)

이슈 - <https://bugs.openjdk.java.net/browse/JDK-8210985>

3.2.4 JDK SSL session cache issue 해결방안

JDK 8u261미만 버전에서 해결방안

1. 설정변경을 이용한 cache size 변경(timeout은 변경 불가)

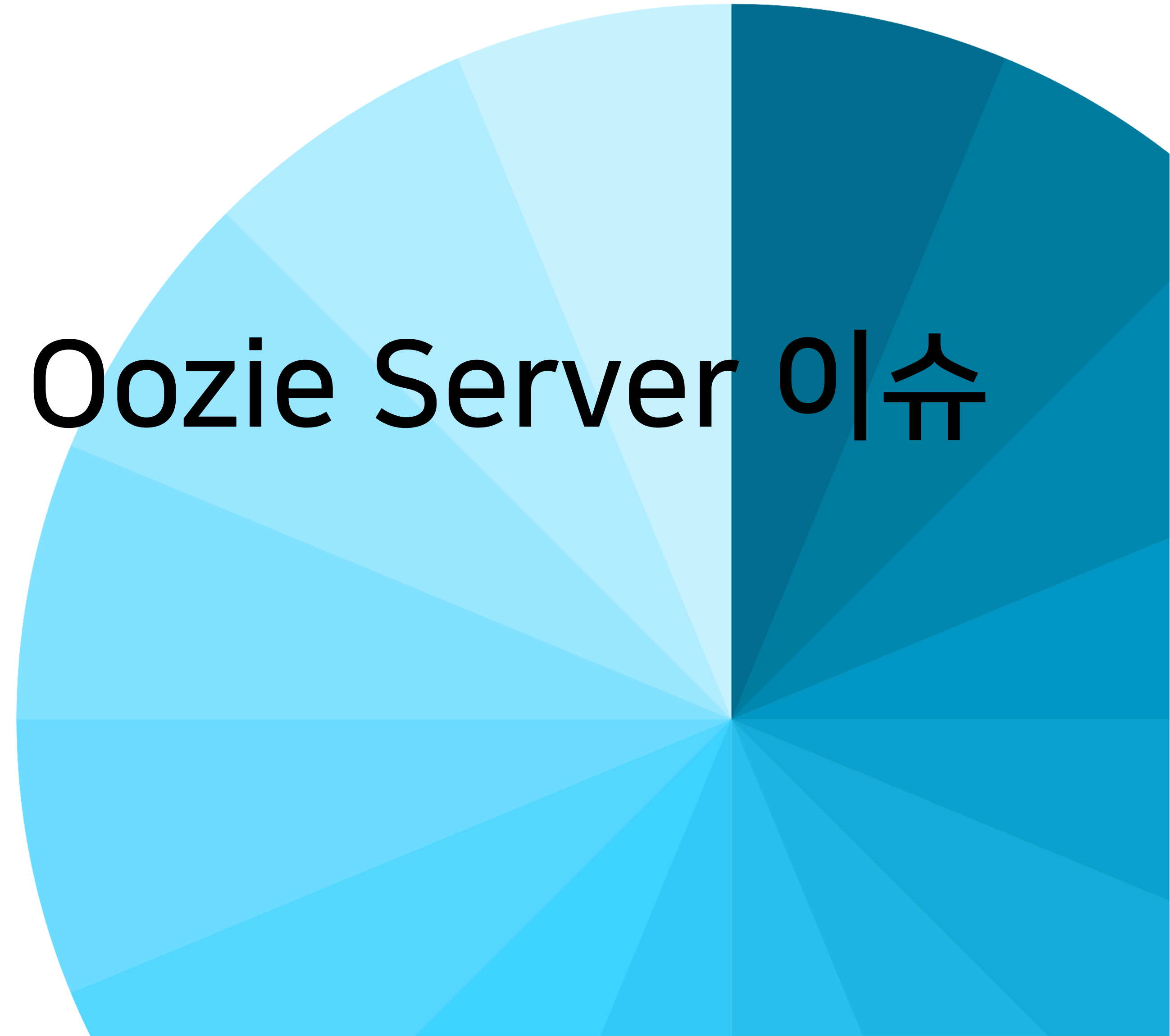
-Djavax.net.ssl.sessionCacheSize=20480

2. 코드 수정을 이용한 cache size와 timeout 변경

- hadoop-common의 org.apache.hadoop.http.HttpServer2.java 파일 수정
- SslContextFactory 생성 후 setSslSessionCacheSize()와 setSslSessionTimeout()의 호출을 통해 설정 가능

현재 저희는 설정변경을 통해 cache size를 변경하여 운영중입니다.
하둡 관련 모든 웹서버에 적용을 하였습니다.

3.3 Mapreduce & Oozie Server 이슈



3.3.1 Mapreduce 작업 중 hang되는 이슈

현상

- Mapreduce 작업 중 task의 attempt가 실패한 이후 새로운 attempt가 NEW상태에서 지속, 이로 인해서 Mapreduce 작업 자체가 무한정 대기하는 이슈

상황

- mapreduce.job.maps(또는 mapreduce.job.reduces)설정이 mapreduce.job.running.map.limit(또는 mapreduce.job.running.reduce.limit)과 동일한 상황에서 task 실패 후 빠르게 재 할당을 받을 경우 NEW상태에서 지속

3.3.2 Mapreduce 작업 중 hang되는 이슈

원인

- 동시 task를 제한하는 부분의 패치(MAPREDUCE-6697)로 인한 side-effect

해결

- 해당 패치(MAPREDUCE-6697)를 revert 함으로써 해결
- Apache에도 이슈(MAPREDUCE-7291)는 올려놓은 상태

3.3.3 Oozie - 작업 tracking이 지연되는 이슈

현상

- Oozie에서 작업량이 늘어날 경우 작업의 상태변화를 점점 늦게 감지

상황

- 수행중인 작업이 동시에 체크할 수 있는 스레드의 수를 초과할 경우 상태변화의 감지가 늦어지는 상황
- 상태변화의 감지가 늦어질 경우 작업의 종료를 인지하지 못하기 때문에 다음 작업들의 수행이 점점 늦어지고 이로 인한 서비스의 지연 및 장애가 발생

3.3.4 Oozie - 작업 tracking이 지연되는 이슈

원인

- 작업의 상태 변화 체크시 AM(Application Master)에 요청을 하게 되는데 Kerberized AM과 Oozie가 정상적으로 통신을 할 수 없었고, 체크하는 스레드는 작업이 종료 될 때까지 sleep 후 요청을 반복
- 즉 체크하는 스레드 수 만큼만 작업들이 상태체크가 가능(이후 작업은 체크불가)

해결

- Oozie server의 mapred-site의 mapreduce.job.am-access-disabled값을 true로 변경하여서 해결(리소스매니저를 통해 체크를 하도록 변경)

3.4 사내에서 DNS를 운영하고, 하둡관련 DNS 등록 시 유의사항

3.4.1 CNAME 레코드 사용시 유의사항

인프라에서 제공되는 장비정보

- IP: 10.100.99.88, domainName: machine001.infra.com

해당 장비에 부여해야 하는 도메인

- namenode01.apps.com

A 레코드와 CNAME 레코드의 차이점

- A 레코드 - namenode01.apps.com. A 10.100.99.88
- CNAME 레코드 - namenode01.apps.com. CNAME machine001.infra.com.

CNAME 레코드의 장점

- 장비의 IP가 변경되어도 신경쓸 필요가 없음(운영의 편리성 증가)

3.4.1 CNAME 레코드 사용시 유의사항

CNAME레코드가 아래와 같이 설정된 상황

- namenode01.apps.com. CNAME machine001.infra.com.

SPNEGO를 사용하는 호스트에 CNAME 레코드 사용시 문제점

- namenode01.apps.com에 접속을 할 경우 HTTP/namenode01.apps.com에 대한 서비스 티켓의 발급을 요청해야 하는데, HTTP/machine001.infra.com에 대한 서비스 티켓의 발급을 요청하고, 없으니 실패

브라우저 환경에 따라 문제가 발생(macOS 기준)

- 파이어폭스, 사파리에서는 정상동작
- 크롬에서 실패

SPNEGO를 사용하는 호스트에 CNAME 레코드 사용시 문제가 발생할 수 있습니다.

3.4.2 PTR 레코드의 활용

다음의 장비는 클라이언트의 /etc/hosts에 IP/FQDN 정보가 필요


- HiveServer2, Hive metastore, Zookeeper

IP/FQDN을 추가하지 않으면 문제점 발생

- Hive, Zookeeper의 코드상의 이슈로 인한 접속 불가

클라이언트에서 /etc/hosts를 수정하기가 힘든 경우(e.g. 컨테이너 환경) DNS에 PTR 레코드(Reverse Domain)를 추가

클라이언트에서 /etc/hosts 를 수정하기 힘든경우 PTR 레코드의 추가를 통해 해결할 수 있습니다.

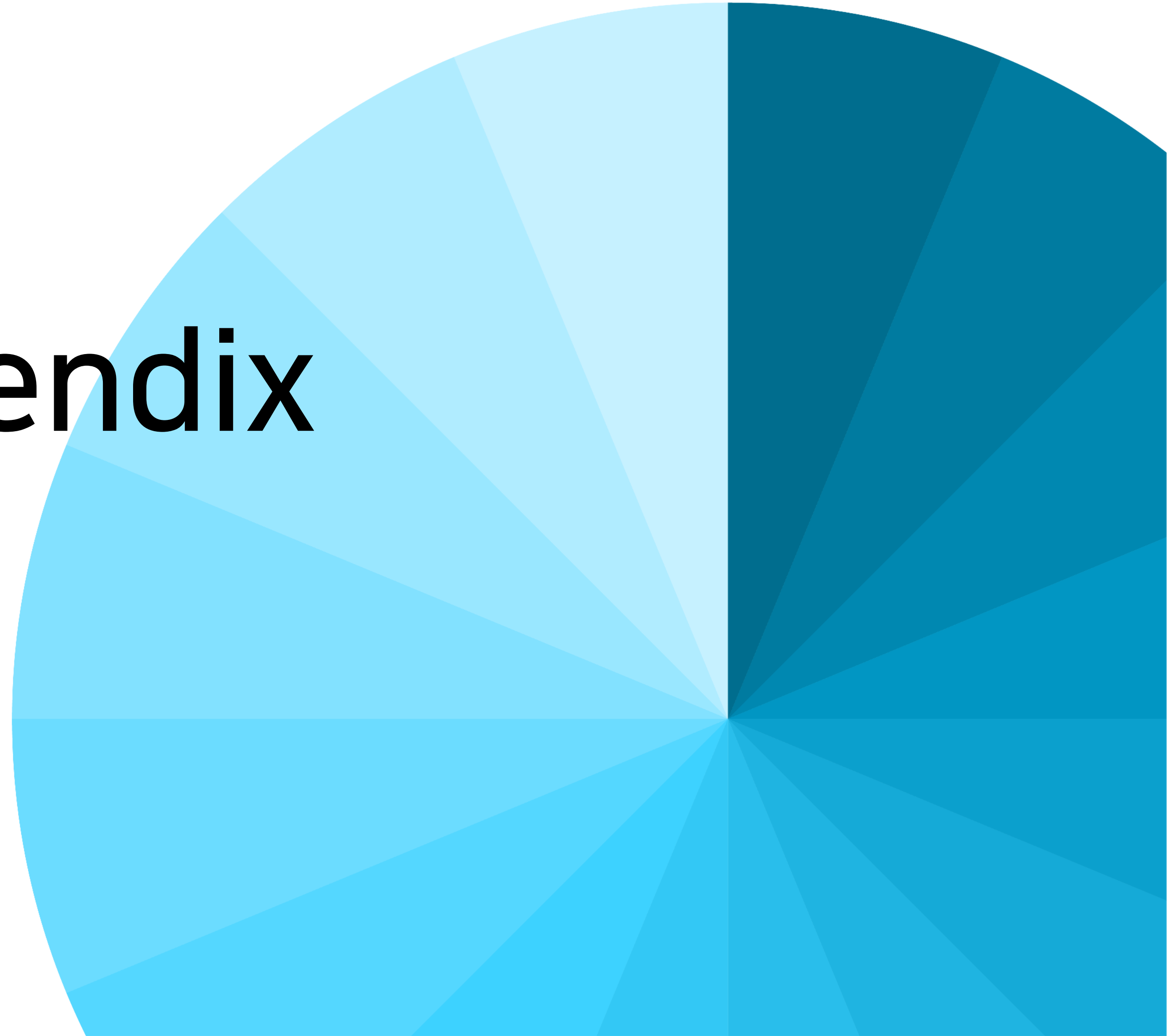


Q & A



Thank You

Appendix



2.1.5 접근하려는(Trusted) 클러스터의 하둡설정

/etc/hosts 수정(접근을 허락한 클러스터의 네임노드의 정보를 추가)

```
x.x.x.x test-nn001.domain.com  
x.x.x.x test-nn002.domain.com
```

/etc/krb5.conf 수정(접근을 허락한 클러스터의 KDC 정보를 추가)

```
[domain_realm]  
..  
  
.dest.com = B.EXAMPLE.COM  
dest.com = B.EXAMPLE.COM  
  
[realms]  
..  
  
B.EXAMPLE.COM = {  
  kdc = kerberos.dest.com  
}
```

2.1.5 접근하려는(Trusted) 클러스터의 하둡설정

hdfs-site.xml(접근을 허락한 클러스터의 HA 정보를 추가)
리소스매니저에만 적용한다.

```
dfs.nameservices=${기존 네임서비스},${접근할 네임서비스}
```

```
dfs.client.failover.proxy.provider.datalake=org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider
```

```
dfs.ha.namenodes.${접근할 네임서비스}=nn1,nn2
```

```
dfs.namenode.rpc-address.${접근할 네임서비스}.nn1=test-nn001.dest.com:8020
```

```
dfs.namenode.rpc-address.${접근할 네임서비스}.nn2=test-nn002.dest.com:8020
```

```
dfs.namenode.http-address.${접근할 네임서비스}.nn1=test-nn001.dest.com:50070
```

```
dfs.namenode.http-address.${접근할 네임서비스}.nn2=test-nn002.dest.com:50070
```

dfs.internal.nameservices는 수정하지 않는다.

2.1.5 접근을 허가한(Trusting) 클러스터의 하둡설정

core-site.xml 수정(realmb별 사용자명 변환 룰 추가)

```
<property>
  <name>hadoop.security.auth_to_local</name>
  <value>
...
    RULE:[1:$1@$0](.*@A.EXAMPLE.COM)s/(.+)_/_cross_realm_a_cluster_users_$1/
    RULE:[2:$1@$0](.*@A.EXAMPLE.COM)s/(.+)_/_cross_realm_a_cluster_users_$1/
...
    DEFAULT
  </value>
</property>
```

위와 같이 설정할 경우 A.EXAMPLE.COM에서 foo라는 사용자가 접근 시 사용자명은 `_cross_realm_a_cluster_users_foo`로 치환된다.

2.1.5 접근을 허가한(Trusting) 클러스터의 하둡설정

kms-site.xml 수정(realm별 사용자명 변환 룰 추가)

```
<property>
  <name>hadoop.kms.authentication.kerberos.name.rules</name>
  <value>
...
    RULE:[1:$1@$0](.*@A.EXAMPLE.COM)s/(.+)_/_cross_realm_a_cluster_users_$1/
    RULE:[2:$1@$0](.*@A.EXAMPLE.COM)s/(.+)_/_cross_realm_a_cluster_users_$1/
...
    DEFAULT
  </value>
</property>
```

위와 같이 설정할 경우 A.EXAMPLE.COM에서 foo라는 사용자가 접근 시 사용자명은 `_cross_realm_a_cluster_users_foo`로 치환된다.

2.2.1 KnoxSSO도입 전(SPNEGO) Web UI 접근 절차



브라우저에서 SPNEGO*가 되도록 설정

대용량 멀티테넌트 시큐어 하둡 클러스터를 시행착오 없이 만들기

발표자료 참고(p131~p135)

2.2.6 Knox 설치 및 설정 방법

2. public key export

Knox 호스트에 접속

아래 명령으로 cert 추출

```
sudo su - knox /usr/hdp/current/knox-server/bin/knoxcli.sh export-cert
```

아래 명령으로 Cert를 한줄로 추출

```
grep -v '^-----' /usr/hdp/3.1.0.0-78/knox/data/security/keystores/gateway-identity.pem | tr -d '\n'
```

2.2.6 Knox 설치 및 설정 방법

3. 하둡 설정

core-site.xml 수정

```
hadoop.http.authentication.authentication.provider.url=${KNOX_URL}/gateway/knoxssso/api/v1/websso
hadoop.http.authentication.type=org.apache.hadoop.security.authentication.server.JWTRedirectAuthenticationHandler
hadoop.http.authentication.alt-kerberos.non-browser.user-agents=java,curl,wget,perl,python,commons-httpclient
hadoop.http.authentication.signer.secret.provider=file
hadoop.http.authentication.signature.secret.file=${HTTP_SECRET_PATH}
hadoop.http.authentication.cookie.domain=${JWT_COOKIE_DOMAIN}
hadoop.http.authentication.public.key.pem=${KNOX_PUBLIC_KEY}
```

hdfs-site.xml 수정

```
dfs.web.authentication.authentication.provider.url=${KNOX_URL}/gateway/knoxssso/api/v1/websso
dfs.web.authentication.type=org.apache.hadoop.security.authentication.server.JWTRedirectAuthenticationHandler
dfs.web.authentication.alt-kerberos.non-browser.user-agents=java,curl,wget,perl,python,commons-httpclient
dfs.web.authentication.signer.secret.provider=file
dfs.web.authentication.signature.secret.file=${HTTP_SECRET_PATH}
dfs.web.authentication.cookie.domain=${JWT_COOKIE_DOMAIN}
dfs.web.authentication.public.key.pem=${KNOX_PUBLIC_KEY}
```

2.2.6 Knox 설치 및 설정 방법

4. Spark 설정

Advanced spark2-env 수정

```
{% if security_enabled %}
export SPARK_HISTORY_OPTS='-Dspark.ui.filters=org.apache.hadoop.security.authentication.server.AuthenticationFilter
-Dspark.org.apache.hadoop.security.authentication.server.AuthenticationFilter.param.alt-kerberos.non-browser.user-
agents=java,curl,wget,perl,python,commons-httpclient
-Dspark.org.apache.hadoop.security.authentication.server.AuthenticationFilter.params="type=org.apache.hadoop.security.a
uthentication.server.JWTRedirectAuthenticationHandler,kerberos.principal={{spnego_principal}},kerberos.keytab={{spnego_
keytab}},authentication.provider.url=${KNOX_URL}/gateway/knoxssso/api/v1/
websso,signer.secret.provider=file,signature.secret.file=${HTTP_SECRET_PATH},cookie.domain=${
JWT_COOKIE_DOMAIN},public.key.pem=${KNOX_PUBLIC_KEY}'"
{% endif %}
```

2.2.6 Knox 설치 및 설정 방법

5. Oozie 설정

oozie-site.xml 수정

```
oozie.server.authentication.type=kerberos
```

```
oozie.authentication.authentication.provider.url=${KNOX_URL}/gateway/knoxssso/api/v1/websso
```

```
oozie.authentication.type=org.apache.hadoop.security.authentication.server.JWTRedirectAuthenticationHandler
```

```
oozie.authentication.alt-kerberos.non-browser.user-agents=java,curl,wget,perl,python,commons-httpclient
```

```
oozie.authentication.signer.secret.provider=file
```

```
oozie.authentication.signature.secret.file=${HTTP_SECRET_PATH}
```

```
oozie.authentication.cookie.domain=${JWT_COOKIE_DOMAIN}
```

```
oozie.authentication.public.key.pem=${KNOX_PUBLIC_KEY}
```


2.3.3 나머지 컴포넌트들의 Access Log 설정

HttpFS

```
log4j.logger.http.requests.webhdfs=INFO,webhdfsrequestlog  
log4j.appender.webhdfsrequestlog=org.apache.hadoop.http.HttpRequestLogAppender  
log4j.appender.webhdfsrequestlog.FileName=${hadoop.log.dir}/access-webhdfs-yyyy_mm_dd.log  
log4j.appender.webhdfsrequestlog.RetainDays=3
```

SecondaryNameNode

```
log4j.logger.http.requests.secondary=INFO,secondaryrequestlog  
log4j.appender.secondaryrequestlog=org.apache.hadoop.http.HttpRequestLogAppender  
log4j.appender.secondaryrequestlog.FileName=${hadoop.log.dir}/access-secondary-yyyy_mm_dd.log  
log4j.appender.secondaryrequestlog.RetainDays=3
```

JournalNode

```
log4j.logger.http.requests.journal=INFO,journalrequestlog  
log4j.appender.journalrequestlog=org.apache.hadoop.http.HttpRequestLogAppender  
log4j.appender.journalrequestlog.FileName=${hadoop.log.dir}/access-journal-yyyy_mm_dd.log  
log4j.appender.journalrequestlog.RetainDays=3
```


2.3.3 나머지 컴포넌트들의 Access Log 설정

NFS3

```
log4j.logger.http.requests.nfs3=INFO,nfs3requestlog  
log4j.appender.nfs3requestlog=org.apache.hadoop.http.HttpRequestLogAppender  
log4j.appender.nfs3requestlog.FileName=${hadoop.log.dir}/access-nfs3-yyyy_mm_dd.log  
log4j.appender.nfs3requestlog.RetainDays=3
```

DataNode

```
log4j.logger.http.requests.datanode=INFO,datanoderequestlog  
log4j.appender.datanoderequestlog=org.apache.hadoop.http.HttpRequestLogAppender  
log4j.appender.datanoderequestlog.FileName=${hadoop.log.dir}/access-datanode-yyyy_mm_dd.log  
log4j.appender.datanoderequestlog.RetainDays=3
```

NodeManager

```
log4j.logger.http.requests.nodemanager=INFO,nodemanagerrequestlog  
log4j.appender.nodemanagerrequestlog=org.apache.hadoop.http.HttpRequestLogAppender  
log4j.appender.nodemanagerrequestlog.FileName=${hadoop.log.dir}/access-nodemanager-yyyy_mm_dd.log  
log4j.appender.nodemanagerrequestlog.RetainDays=3
```

2.3.3 나머지 컴포넌트들의 Access Log 설정

Timeline

```
log4j.logger.http.requests.timeline=INFO,timelinerequestlog  
log4j.appender.timelinerequestlog=org.apache.hadoop.http.HttpRequestLogAppender  
log4j.appender.timelinerequestlog.FileName=${hadoop.log.dir}/access-timeline-yyyy_mm_dd.log  
log4j.appender.timelinerequestlog.RetainDays=3
```

KMS

```
log4j.logger.http.requests.kms=INFO,kmsrequestlog  
log4j.appender.kmsrequestlog=org.apache.hadoop.http.HttpRequestLogAppender  
log4j.appender.kmsrequestlog.FileName=${hadoop.log.dir}/access-kms-yyyy_mm_dd.log  
log4j.appender.kmsrequestlog.RetainDays=3
```

2.4.4 Apache Spark의 Dynamic Allocation 지원

yarn-site.xml 수정

```
yarn.nodemanager.aux-services=mapreduce_shuffle,spark2_shuffle,spark_shuffle
```

```
yarn.nodemanager.aux-services.spark_shuffle.class=org.apache.spark.network.yarn.YarnShuffleService
```

```
yarn.nodemanager.aux-services.spark_shuffle.classpath=${JAR_DIRECTORY}/*
```