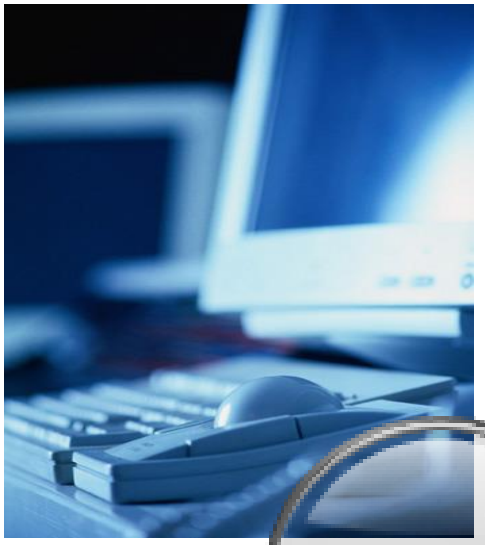
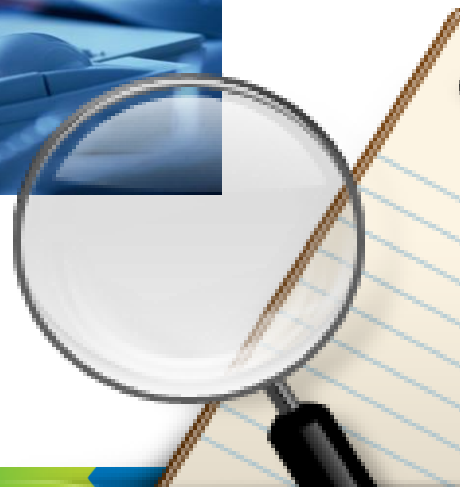
클러스터 유지와 보안, 트러블 슈팅

- 노드 OS 업그레이드
- 쿠버네티스 버전 업데이트
- 백업과 복원 방법
- 보안을 위한 다양한 리소스
- TLS 인증서를 활용한 통신 이해
- RBAC를 활용한 롤 기반 엑세스 컨트롤시큐리티 콘텍스트
- 네트워크 정책 적용
- 클러스터, 애플리케이션, 네트워크 트러블 슈팅



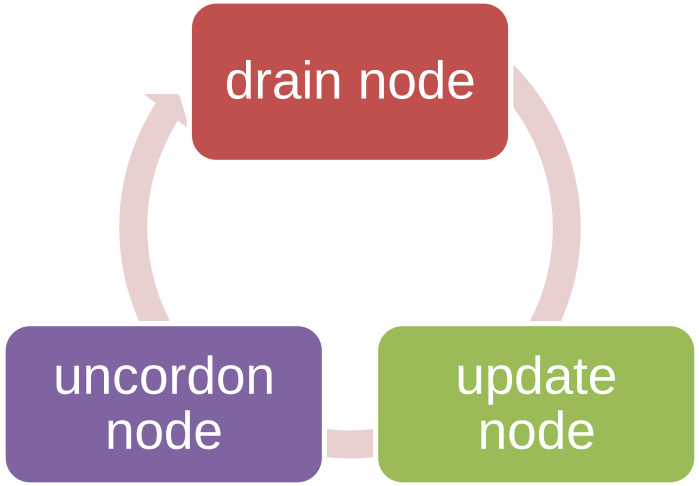


노드 OS 업그레이드

노드 OS 업그레이드

노드의 유지 보수

- 커널 업그레이드, libc 업그레이드, 하드웨어 복구 등과 같이 노드를 재부팅해야 하는 경우
- 노드를 갑자기 내리는 경우 다양한 문제가 발생할 수 있음
- 포드는 노드가 내려가서 5분 이상 돌아오지 않으면 그때 다른 노드에 포드를 복제
- 업그레이드 프로세스를 보다 효율적으로 제어하려면 하나의 노드 씩 다음 절차를 진행



노드 OS 업그레이드

현재 노드 확인

```
$ kubectl get node
NAME                                                STATUS   ROLES    AGE   VERSION
gke-standard-cluster-1-default-pool-bdf36b75-dlzj   Ready    <none>   30h   v1.13.7-gke.8
gke-standard-cluster-1-default-pool-bdf36b75-rct0   Ready    <none>   30h   v1.13.7-gke.8
gke-standard-cluster-1-default-pool-bdf36b75-x5j5   Ready    <none>   30h   v1.13.7-gke.8
```

노드 drain(배수)

```
$ kubectl drain gke-standard-cluster-1-default-pool-bdf36b75-dlzj
node/gke-standard-cluster-1-default-pool-bdf36b75-dlzj cordoned
error: unable to drain node "gke-standard-cluster-1-default-pool-bdf36b75-dlzj", aborting command...

There are pending nodes to be drained:
 gke-standard-cluster-1-default-pool-bdf36b75-dlzj
error: pods with local storage (use --delete-local-data to override): two-containers; DaemonSet-
managed pods (use --ignore-daemonsets to ignore): fluentd-gcp-v3.2.0-vwfzv, prometheus-to-sd-r7jcw
```

노드 OS 업그레이드

현재 노드 확인

```
$ kubectl get node
NAME                                                STATUS   ROLES    AGE   VERSION
gke-standard-cluster-1-default-pool-bdf36b75-dlzj   Ready    <none>   30h   v1.13.7-gke.8
gke-standard-cluster-1-default-pool-bdf36b75-rct0   Ready    <none>   30h   v1.13.7-gke.8
gke-standard-cluster-1-default-pool-bdf36b75-x5j5   Ready    <none>   30h   v1.13.7-gke.8
```

노드 drain(배수)

- 노드에 스케줄링된 포드들을 모두 내리는 작업
- 노드를 내리는 작업에서 문제가 발생하는 경우 권장하는 옵션을 사용

```
$ kubectl drain gke-standard-cluster-1-default-pool-bdf36b75-dlzj
node/gke-standard-cluster-1-default-pool-bdf36b75-dlzj cordoned
error: unable to drain node "gke-standard-cluster-1-default-pool-bdf36b75-dlzj", aborting command...

There are pending nodes to be drained:
 gke-standard-cluster-1-default-pool-bdf36b75-dlzj
error: pods with local storage (use --delete-local-data to override): two-containers; DaemonSet-
managed pods (use --ignore-daemonsets to ignore): fluentd-gcp-v3.2.0-vwfzv, prometheus-to-sd-r7jcw
```

노드 OS 업그레이드

옵션을 모두 사용하여 다시 시도

```
$ kubectl drain gke-standard-cluster-1-default-pool-bdf36b75-dlzj --delete-local-data --ignore-
daemonsets --force
node/gke-standard-cluster-1-default-pool-bdf36b75-dlzj already cordoned
WARNING: Deleting pods not managed by ReplicationController, ReplicaSet, Job, DaemonSet or
StatefulSet: two-containers; Ignoring DaemonSet-managed pods: fluentd-gcp-v3.2.0-vwfzv, prometheus-
to-sd-r7jcw; Deleting pods with local storage: two-containers
pod/command-demo evicted
pod/my-scheduler-79ccf6b89-gtbjd evicted
pod/heapster-v1.6.0-beta.1-6cdb9cd95b-vdk24 evicted
pod/metrics-server-v0.3.1-57c75779f-6w2nr evicted
pod/two-containers evicted
pod/php-apache-84cc7f889b-6wzbx evicted
pod/kube-dns-autoscaler-bb58c6784-mw7cs evicted
pod/fluentd-gcp-scaler-59b7b75cd7-wq6b6 evicted
node/gke-standard-cluster-1-default-pool-bdf36b75-dlzj evicted
```

```
$ kubectl get node
NAME                                          STATUS                     ROLES    AGE   VERSION
gke-standard-cluster-1-default-pool-bdf36b75-dlzj   Ready,SchedulingDisabled   <none>   30h   v1.13.7-gke.8
gke-standard-cluster-1-default-pool-bdf36b75-rct0   Ready                      <none>   30h   v1.13.7-gke.8
gke-standard-cluster-1-default-pool-bdf36b75-x5j5   Ready                      <none>   30h   v1.13.7-gke.8
```

노드 OS 업그레이드

옵션을 모두 사용하여 다시 시도

```
$ kubectl drain gke-standard-cluster-1-default-pool-bdf36b75-dlzj --delete-local-data --ignore-
daemonsets --force
node/gke-standard-cluster-1-default-pool-bdf36b75-dlzj already cordoned
WARNING: Deleting pods not managed by ReplicationController, ReplicaSet, Job, DaemonSet or
StatefulSet: two-containers; Ignoring DaemonSet-managed pods: fluentd-gcp-v3.2.0-vwfzv, prometheus-
to-sd-r7jcw; Deleting pods with local storage: two-containers
pod/command-demo evicted
pod/my-scheduler-79ccf6b89-gtbjd evicted
pod/heapster-v1.6.0-beta.1-6cdb9cd95b-vdk24 evicted
pod/metrics-server-v0.3.1-57c75779f-6w2nr evicted
pod/two-containers evicted
pod/php-apache-84cc7f889b-6wzbx evicted
pod/kube-dns-autoscaler-bb58c6784-mw7cs evicted
pod/fluentd-gcp-scaler-59b7b75cd7-wq6b6 evicted
node/gke-standard-cluster-1-default-pool-bdf36b75-dlzj evicted
```

결과 확인

```
$ kubectl get node
NAME                                          STATUS                     ROLES    AGE   VERSION
gke-standard-cluster-1-default-pool-bdf36b75-dlzj   Ready,SchedulingDisabled   <none>   30h   v1.13.7-gke.8
gke-standard-cluster-1-default-pool-bdf36b75-rct0   Ready                      <none>   30h   v1.13.7-gke.8
gke-standard-cluster-1-default-pool-bdf36b75-x5j5   Ready                      <none>   30h   v1.13.7-gke.8

$ kubectl get pod -o wide | grep dlzj # dlzj에서 활동중인 포드 확인
출력 결과 없음
```

노드 OS 업그레이드

OS 업그레이드 후 노드를 복구 (uncordon; 저지선을 철거하다)

```
$ kubectl uncordon gke-standard-cluster-1-default-pool-bdf36b75-dlzj
node/gke-standard-cluster-1-default-pool-bdf36b75-dlzj uncordoned

$ kubectl get node
NAME                                                  STATUS   ROLES    AGE   VERSION
gke-standard-cluster-1-default-pool-bdf36b75-dlzj   Ready    <none>   30h   v1.13.7-gke.8
gke-standard-cluster-1-default-pool-bdf36b75-rct0   Ready    <none>   30h   v1.13.7-gke.8
gke-standard-cluster-1-default-pool-bdf36b75-x5j5   Ready    <none>   30h   v1.13.7-gke.8
```

cordon과 drain의 차이

- drain은 스케줄링된 모든 포드를 다른 노드로 리스케줄링을 시도하고 저지선을 만들어 더 이상 노드에 포드가 만들어지지 않도록 한다.
- 반면 cordon은 저지선만 설치하는 것으로 현재 갖고 있는 포드를 그대로 유지하면서 새로운 포드에 대해서만 스케줄링을 거부한다.

생각해보기

- uncordon을 사용하면 노드의 저지선을 풀리면서 포드들이 원래의 스케줄 상태로 되돌아올까?

쿠버네티스 버전 업데이트

쿠버네티스 버전 업데이트

쿠버네티스 버전

- 2014년 처음 v0.4을 시작하여 2015년에 v1.0으로 메이저 버전 업그레이드

- 버전이 나타내는 내용
  - MAJOR
  - MINOR: 특징, 기능
  - PATCH: 버그 픽스

v1.14.1

MAJOR MINOR PATCH

- 모든 버그를 고치기 위해 alpha와 beta로 나누어 출시 후 정식 버전 진행
  - alpha: 추가되는 기능들을 disable한 형태로 배포
  - beta: 추가 기능들을 enable한 형태로 배포
  - release: 안정화된 버전을 배포

- 패키지마다 모든 컴포넌트는 동일한 버전으로 배포
  - API 서버, 컨트롤러 매니저, 스케줄러, 큐블렛, 큐브프록시 등
  - ETCD, 네트워크, CoreDNS는 제외

쿠버네티스 버전 업데이트

쿠버네티스에서 버전 호환

- 쿠버네티스는 apiserver를 기준으로 호환성을 제공
- apiserver의 minor 버전이 x인 경우 다음과 같음
  - kubelet: x+1 ~ x-1
  - controller-manager: x 또는 x-1
  - kube-scheduler: x 또는 x-1
  - kubelet: x ~ x-2
  - kubeproxy: x ~ x-2

쿠버네티스 버전 업데이트

쿠버네티스의 버전 업그레이드가 필요한 경우

- 노드의 OS 업데이트와 하나씩 drain 하며 버전을 업데이트 하는 방법을 사용(롤링 업데이트)
- 업데이트를 적용할 때는 마이너 버전이 하나씩 업데이트 되도록 설정하는 것이 좋음

```
master $ kubectl drain node-1
master $ ssh node-1

node-1 $ apt-get upgrade -y kubeadm=1.12.0-00
node-1 $ apt-get upgrade -y kubelet=1.12.0-00
node-1 $ kubeadm upgrade node config --kubelet-version v1.12.0
node-1 $ systemctl restart kubelet
node-1 $ exit

master $ kubectl uncordon node-1
```

백업과 복원 방법

백업과 복원 방법

백업 리소스의 종류

- 포드의 정보 파일 YAML
  - $ kubectl get all --all-namspaces -o yaml > all-deploy-services.yaml
  - $ kubectl create -f all-deploy-services.yaml

- Etcd 데이터베이스
  - https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/#backing-up-an-etcd-

```
ETCDCTL_API=3 etcdctl --endpoints $ENDPOINT snapshot save snapshotdb
# exit 0

# verify the snapshot
ETCDCTL_API=3 etcdctl --write-out=table snapshot status snapshotdb
+----------+----------+------------+------------+
|   HASH   | REVISION | TOTAL KEYS | TOTAL SIZE |
+----------+----------+------------+------------+
| fe01cf57 |       10 |          7 | 2.1 MB     |
+----------+----------+------------+------------+
```

- Persistent Volume: 일반적인 방법으로 백업

백업과 복원 방법

연습문제

- 현재 활동 중인 모든 리소스의 내역을 yaml 파일로 저장하라.
- 현재 활동 중인 etcd의 데이터베이스를 백업하라.

보안을 위한 다양한 리소스

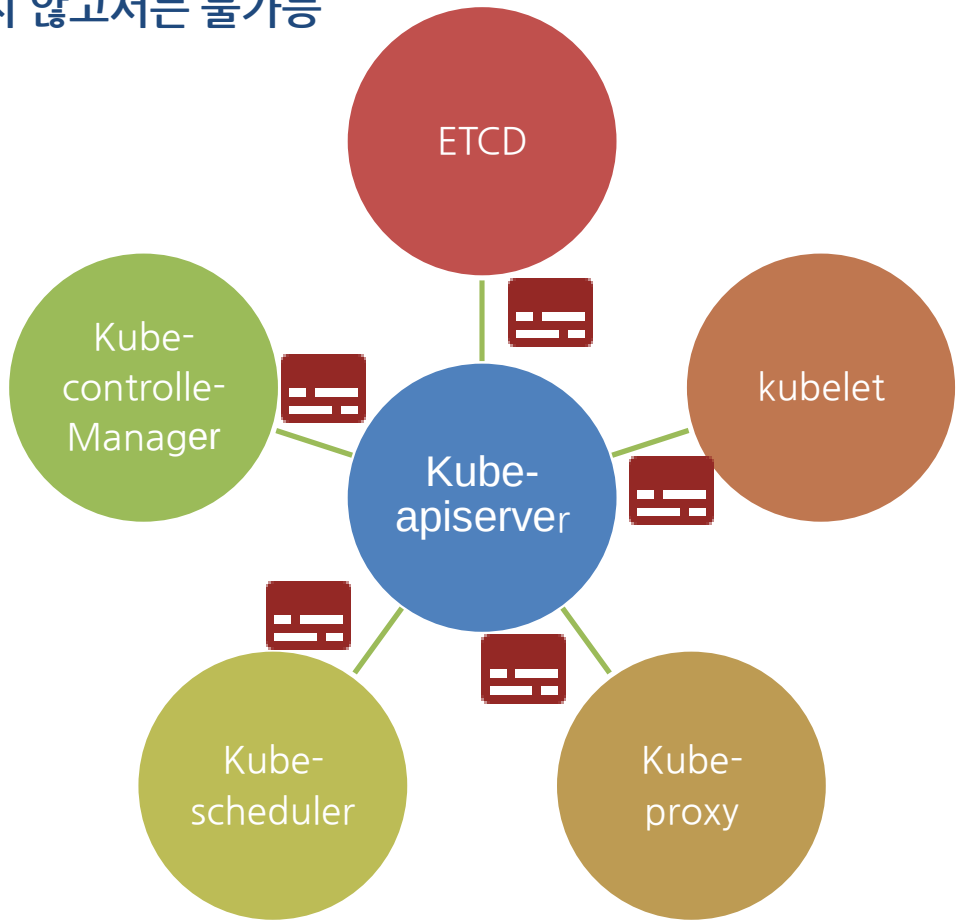
보안을 위한 다양한 리소스

모든 통신은 TLS로

- 대부분의 엑세스는 Kube-apiserver를 통하지 않고서는 불가능

- 엑세스 가능한 유저
  - 파일 - 유저 이름과 토큰
  - Service Accounts
  - 인증서(Certificates)
  - External Authentication Providers - LDAP

- 무엇을 할 수 있는가?
  - RBAC Authorization
  - ABAC Authorization
  - Node Authorization
  - WebHook Mode



Accounts

Accounts

- 어카운츠에는 두 가지의 타입이 존재
  - 사용자를 위한 user
  - 애플리케이션(포드 외)를 위한 service account

Accounts

Static Token File

- Apiserver 서비스를 실행할 때 --token-auth-file=SOMEFILE.csv 전달 (kube-apiserver 수정 필요)
- API 서버를 다시 시작해야 적용됨
- 토큰, 사용자 이름, 사용자 uid, 선택적dmfh 그룹 이름 다음에 최소 3 열의 csv 파일

SOMEFILE.csv 파일 내용

```
# 사용 형식: token,user,uid,"group1,group2,group3“
password1, user1, uid001, group1
password2, user2, uid002
password3, user3, uid003
password4, user4, uid004
```

Accounts

Static Token File 을 적용했을 때 사용 방법

- HTTP 요청을 진행할 때 다음과 같은 내용을 헤더에 포함해야함
- Authorization: Bearer 31ada4fd-adec-460c-809a-9e56ceb75269

- curl -v -k https://master-node-ip:6443/api/v1/pods --header “Authorization: Bearer 31ada4fd-adec-460c-809a-9e56ceb75269”

Accounts

서비스 어카운트 만들기

- 포드에 별도의 설정을 주지 않으면 기본적은 service account가 생성

```
$ kubectl get sa default -o yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  creationTimestamp: "2019-07-06T06:12:26Z"
  name: default
  namespace: default
  resourceVersion: "279"
  selfLink:
/api/v1/namespaces/default/serviceaccounts/default
  uid: 06b46c38-9fb5-11e9-a520-42010a920176
secrets:
- name: default-token-ppzqk
```

- 명령어를 사용하여 serviceaccount sa1를 생성
  - Kubectl create serviceaccount sa1

TLS 인증서를 활용한 통신 이해

TLS 인증서를 활용한 통신 이해

SSL 통신 과정 이해

- 응용계층인 HTTP와 TCP계층사이에서 작동,
- 어플리케이션에 독립적 -> HTTP제어를 통한 유연성
- 데이터의 암호화 (기밀성), 데이터 무결성, 서버인증기능, 클라이언트 인증기능

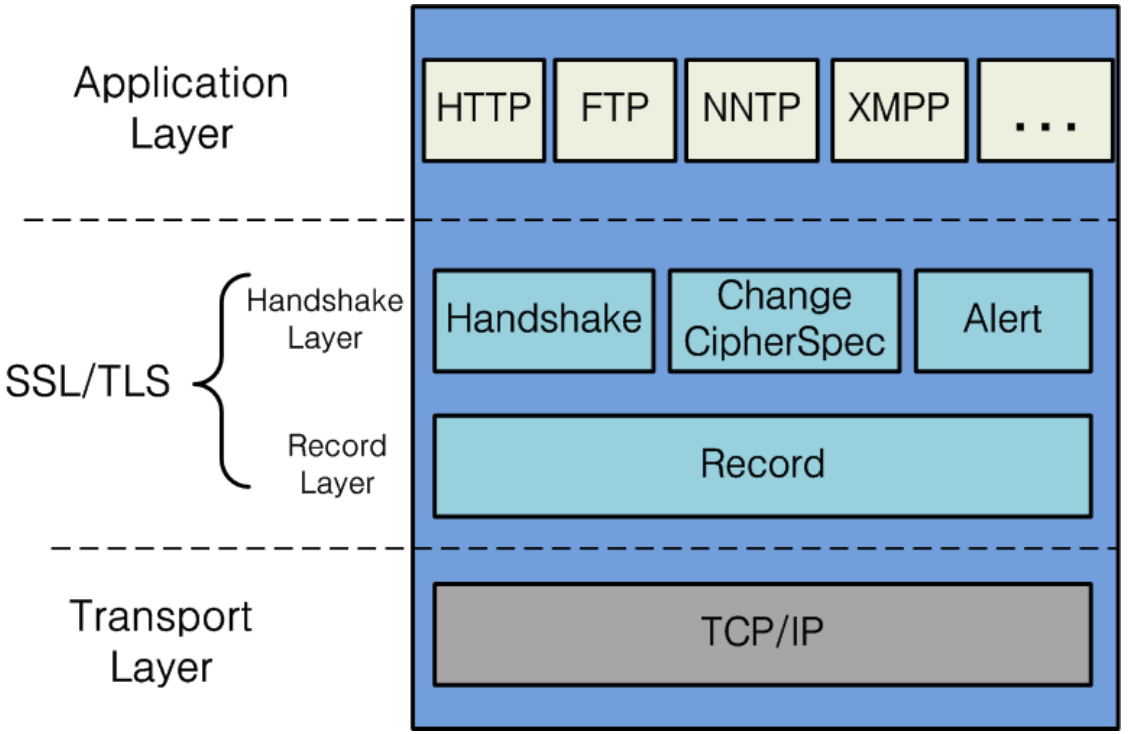


그림 출처: https://soul0.tistory.com/510

TLS 인증서를 활용한 통신 이해

- Certificate를 보장하는 방법! 인증기관(CA)

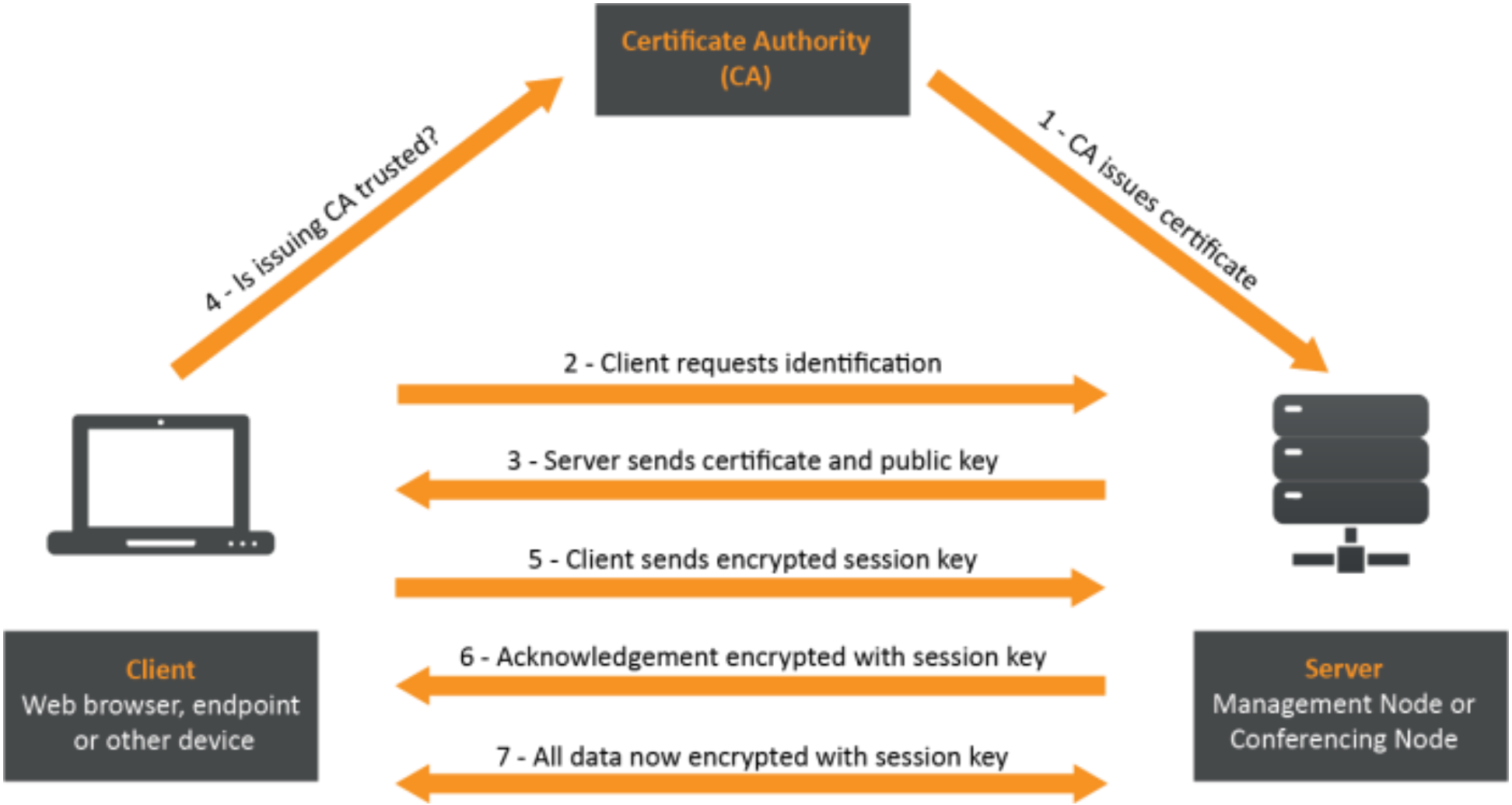


그림 출처: https://docs.pexip.com/admin/certificate_management.htm

TLS 인증서를 활용한 통신 이해

kubernete의 인증서 위치

```
$ sudo ls /etc/kubernetes/pki
apiserver-etcd-client.crt     apiserver.key         front-proxy-ca.key
apiserver-etcd-client.key     ca.crt                front-proxy-client.crt
apiserver-kubelet-client.crt  ca.key                front-proxy-client.key
apiserver-kubelet-client.key  etcd                  sa.key
apiserver.crt                 front-proxy-ca.crt    sa.pub

$ sudo ls /etc/kubernetes/pki/etcd
ca.crt  healthcheck-client.crt  peer.crt  server.crt
ca.key  healthcheck-client.key  peer.key  server.key
```

TLS 인증서를 활용한 통신 이해

정확한 TLS 인증서 사용 점검

- 적절한 키를 사용하는지 확인하려면 manifests 파일에서 실행하는 certificate 확인 필요
- 적절한 키를 사용하지 않으면 에러가 발생

```
$ sudo ls /etc/kubernetes/manifests/
etcd.yaml              kube-controller-manager.yaml
kube-apiserver.yaml    kube-scheduler.yaml
```

TLS 인증서를 활용한 통신 이해

정확한 TLS 인증서 사용 점검

- Kubelet의 위치는 조금 다름

```
server1@MASTER:~$ sudo ls /var/lib/kubelet/pki
kubelet-client-2019-07-09-12-44-22.pem   kubelet.crt
kubelet-client-2019-07-09-12-44-54.pem   kubelet.key
kubelet-client-current.pem
server1@MASTER:~$ sudo ls /var/lib/kubelet/
config.yaml         kubeadm-flags.env  plugins            pods
cpu_manager_state      pki                    plugins_registry
device-plugins      plugin-containers  pod-resources
server1@MASTER:~$ sudo cat /var/lib/kubelet/config.yaml
address: 0.0.0.0
apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
  anonymous:
    enabled: false
  webhook:
    cacheTTL: 2m0s
    enabled: true
  x509:
    clientCAFile: /etc/kubernetes/pki/ca.crt
```

TLS 인증서를 활용한 통신 이해

TLS 인증서를 올바르지 않을 때 발생하는 현상 확인하기

- 먼저 현재 MASTER 노드의 상태를 스냅샷으로 저장하라.
- Kube-apiserver.yaml을 찾아 특정 경로를 수정하는 방식으로 certificate를 찾을 수 없도록 만들어라.

TLS 인증서를 활용한 통신 이해

인증서 정보 확인하기

```
$ sudo openssl x509 -in <certificate> -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            c4:70:38:32:90:74:e1:37:8a:77:36:5b:f4:39:06:a0
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=8bc6e46e-289a-4e7a-a182-a9cd4cfa5ab8
        Validity
            Not Before: Jul  6 05:09:38 2019 GMT
            Not After : Jul  4 06:09:38 2024 GMT
        Subject: CN=8bc6e46e-289a-4e7a-a182-a9cd4cfa5ab8
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:b4:da:76:9f:0d:ef:08:60:32:d5:39:a6:99:51:
                    8a:97:5b:7f:11:98:20:d9:b6:55:af:95:9d:aa:a7:
                    fc:e1:68:a0:0d:31:64:69:d2:7f:cb:20:eb:8b:38:
                    22:92:c7:e3:92:5d:b9:67:60:16:fe:2a:35:02:4d:
                    ...
```

Component	Type	Certificate Path	CN Name	ALT Names	Organ	Issuer
Certificate Authority	Server	/etc/kubernetes/pki/ca.crt	kubernetes			kubernetes
Certificate Authority	Server	/etc/kubernetes/pki/ca.key				
kube-apiserver	Server	/etc/kubernetes/pki/apiserver.crt	kube-apiserver	DNS:master DNS:kubernetes DNS:kubernetes.default DNS:kubernetes.default.svc DNS:kubernetes.default.svc.cluster.local IP Address:10.96.0.1 IP Address:172.17.0.27		kubernetes
kube-apiserver	Server	/etc/kubernetes/pki/apiserver.key				
kube-apiserver	Server	/etc/kubernetes/pki/ca.crt	kubernetes			kubernetes
kube-apiserver	Client (Kubelet)	/etc/kubernetes/pki/apiserver-kubelet-client.crt	kube-apiserver-kubelet-client		system:masters	kubernetes
kube-apiserver	Client (Kubelet)	/etc/kubernetes/pki/apiserver-kubelet-client.key				
kube-apiserver	Client (Etcd)	/etc/kubernetes/pki/apiserver-etcd-client.crt	kube-apiserver-etcd-client		system:masters	kubernetes
kube-apiserver	Client (Etcd)	/etc/kubernetes/pki/apiserver-etcd-client.key				
kube-apiserver	Client (Etcd)	/etc/kubernetes/pki/etcd/ca.crt	kubernetes			kubernetes
kubelet	Server	/var/lib/kubelet/pki/kubelet.crt	nodes			
kubelet	Server	/var/lib/kubelet/pki/kubelet.key				
kubelet	Client	/var/lib/kubelet/pki/kubelet-client-<date>.pem	system:node:node01		system:nodes	kubernetes
kubelet	Client					
Certificate Authority (ETCD)	Server	/etc/kubernetes/pki/etcd/ca.crt	kubernetes			kubernetes
Certificate Authority (ETCD)	Server	/etc/kubernetes/pki/etcd/ca.key				

TLS 인증서를 활용한 통신 이해

연습문제

- Apiserver가 사용하는 ca.key 정보를 텍스트 형태로 출력하라.
- ca.key의 CN은 무엇인가?
- ca.key를 이슈화한 CN은 무엇인가?

TLS 인증서를 활용한 유저 생성하기

TLS 인증서를 활용한 유저 생성하기

개인 키 생성하기

- 길이 2048 만큼의 개인 키 생성

```
$ openssl genrsa -out gasbugs.key 2048
```

private 키를 기반으로 인증서 서명 요청하기

- CN: 사용자 이름
- O: 그룹 이름
- CA에게 csr 파일로 인증을 요청할 수 있음!

```
$ openssl req -new -key gasbugs.key -out gasbugs.csr -subj
"/CN=gasbugs/O=boanproject"
```

TLS 인증서를 활용한 유저 생성하기

ca를 사용하여 직접 csr 승인하기

- Kubernetes 클러스터 인증 기관(CA) 사용이 요청을 승인해야 함
- 내부에서 직접 승인하는 경우 pki 디렉토리에 있는 ca.key와 ca.crt를 사용하여 승인 가능
- gasbugs.csr을 승인하여 최종 인증서인 gasbugs.crt를 생성
- -days 옵션을 사용해 며칠간 인증서가 유효할 수 있는지 설정(예제에서는 500일)

```
$ openssl x509 -req -in gasbugs.csr -CA /etc/kubernetes/pki/ca.crt
-CAkey /etc/kubernetes/pki/ca.key -CAcreateserial -out gasbugs.crt
-days 500

Signature ok
subject=CN = gasbugs, O = boanproject
Getting CA Private Key
```

TLS 인증서를 활용한 유저 생성하기

ca를 사용하여 직접 csr 승인하기

- 인증 사용을 위해 쿠버네티스에 crt를 등록
  - crt를 사용할 수 있도록 kubectl을 사용하여 등록

```
$ kubectl config set-credentials gasbugs --client-
certificate=.certs/gasbugs.crt  --client-key=.certs/gasbugs.key
$ kubectl config set-context gasbugs-context --cluster=kubernetes --
namespace=office --user=gasbugs
```

  - 다음 명령을 사용하여 사용자 권한으로 실행 가능(지금은 사용자에게 권한을 할당하지 않아 실행되지는 않는다)

```
$ kubectl --context=gasbugs-context get pods
Error from server (Forbidden): pods is forbidden: User "gasbugs" cannot
list resource "pods" in API group "" in the namespace "office"
```

TLS 인증서를 활용한 유저 생성하기

ca를 사용하여 직접 csr 승인하기

- **사용자 권한 할당**
  - 롤바인딩을 사용하여 office 네임스페이스 권한 할당
  - kubectl create -f 를 사용하여 생성
  - 해당 네임 스페이스에 모든 권한을 생성

```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: deployment-manager-binding
  namespace: office
subjects:
- kind: User
  name: gasbugs
  apiGroup: ""
roleRef:
  kind: Role
  name: deployment-manager
  apiGroup: ""
```

```yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  namespace: office
  name: deployment-manager
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["deployments", "replicasets", "pods"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

TLS 인증서를 활용한 유저 생성하기

ca를 사용하여 직접 csr 승인하기

- **사용자 권한 테스트**
  - 실행돼야 하는 명령어

```
$ kubectl --context=gasbugs-context get pods -n office
$ kubectl --context=gasbugs-context run --generator=pod-run/v1 nginx --
image=nginx -n office
```

  - 실행되면 안되는 명령어

```
$ kubectl --context=gasbugs-context get pods
$ kubectl --context=gasbugs-context run --generator=pod-run/v1 nginx --
image=nginx
```

TLS 인증서를 활용한 유저 생성하기

kubectl의 ceritiface 기능을 사용하여 유저 승인하기

- 먼저 앞에서 사용한 동일한 방식으로 유저가 사용할 csr과 key를 생성

```
$ openssl genrsa -out isc0304.key 2048
$ openssl req -new -key isc0304.key -out isc0304.csr -subj
"/CN=isc0304/O=boanproject"
```

- isc0304.csr 파일의 내용과 함께 isc0304라는 이름으로 CertificateSigningRequest 개체 생성

```
cat <<EOF | kubectl apply -f -
apiVersion: certificates.k8s.io/v1beta1
kind: CertificateSigningRequest
metadata:
  name: isc0304
spec:
  request: $(cat isc0304.csr | base64 | tr -d '\n')
  usages:
  - digital signature
  - key encipherment
  - server auth
EOF
```

TLS 인증서를 활용한 유저 생성하기

kubectl의 ceritiface 기능을 사용하여 유저 승인하기

- csr 생성 확인

```
$ kubectl get csr
NAME    AGE REQUESTOR          CONDITION
isc0304 16s kubernetes-admin Pending
```

- 유저 csr 요청 승인하기

```
$ kubectl certificate approve isc0304
certificatesigningrequest.certificates.k8s.io/isc0304 approved
```

- 유저 csr 요청 거절하기

```
$ kubectl certificate deny isc0304
certificatesigningrequest.certificates.k8s.io/isc0304 denied
$ kubectl delete csr isc0304
```

TLS 인증서를 활용한 유저 생성하기

연습문제

- Dev1팀에 john이 참여했다. John을 위한 인증서를 만들고 승인한 뒤 dev1 네임스페이스에 대한 권한을 할당하라.

kube config 파일을 사용한 인증

kube config 파일을 사용한 인증

직접 curl을 사용하여 요청

- key와 cert, cacert 키를 가지고 직접 요청 가능
- 그러나 매번 이 요청을 사용하기에는 무리가 있음

```
$ curl https://kube-api-server:6443/api/v1/pods\
--key user.key \
--cert user.crt \
--cacert ca.key \
```

kube config 파일을 사용한 인증

kube config 파일 확인하기

- $ kubectl config view
- $ kubectl config view --kubeconfig=<config file>

```
$ kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://10.0.2.10:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes # 현재 kubectl이 사용중인 쿠버네티스 계정
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
```

kube config 파일을 사용한 인증

kubeconfig의 구성

- ~/.kube/config 파일을 확인하면 세 가지 부분으로 작성됨
- clusters: 연결할 쿠버네티스 클러스터의 정보 입력
- users: 사용할 권한을 가진 사용자 입력
- contexts: cluster와 user를 함께 입력하여 권한 할당

```
$ curl https://kube-api-server:6443/api/v1/pods\
--key user.key \
--cert user.crt \
--cacert ca.key \
```

Clusters

```
name: kube-cluster
cluster:
  certificate-authority: ca.crt
  server: https://cluster-ip:6443
```

Contexts

```
name: user-name@kube-cluster
context:
  cluster: kube-cluster
  user: user-name
```

Users

```
name: user-name
user:
  client-certificate: user.crt
  client-key: user.key
```

kube config 파일을 사용한 인증

kubeconfig의 구성

- 각각의 user와 cluster가 잘 맞는지 확인

Clusters

```
name: kube-cluster
cluster:
  certificate-authority: ca.crt
  server: https://cluster-ip:6443
```

Contexts

```
name: user-name@kube-cluster
context:
  cluster: kube-cluster
  user: user-name
```

Users

```
name: user-name
user:
  client-certificate: user.crt
  client-key: user.key
```

kube config 파일을 사용한 인증

- 인증 사용자 바꾸기
  - kubectl config use-context user@kube-cluster

- 인증 테스트

RBAC를 활용한 롤 기반 액세스 컨트롤

롤, 클러스터 롤

RBAC를 활용한 롤 기반 액세스 컨트롤

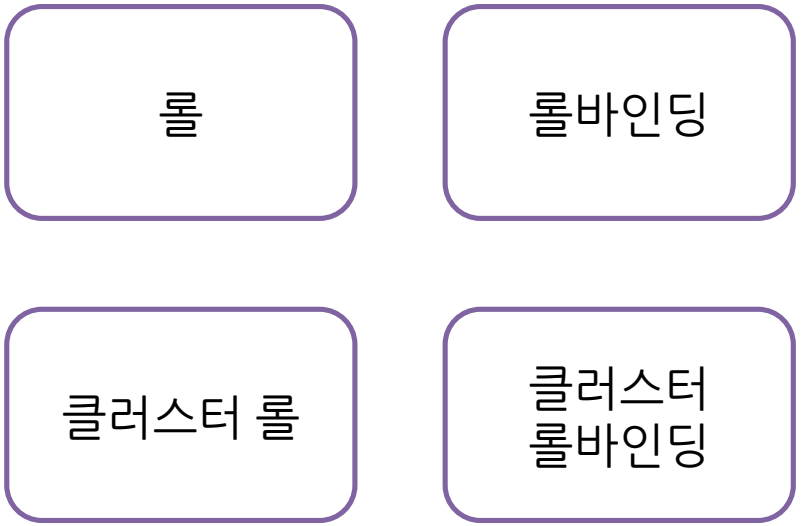
역할 기반 액세스 제어 (RBAC)

- 기업 내에서 개별 사용자의 역할을 기반으로 컴퓨터, 네트워크 리소스에 대한 액세스를 제어
- rbac.authorization.k8s.ioAPI를 사용하여 정의
- 권한 결정을 내리고 관리자가 Kubernetes API를 통해 정책을 동적으로 구성
- RBAC를 사용하여 룰을 정의하려면 apiserver에 --authorization-mode=RBAC 옵션이 필요

RBAC를 활용한 롤 기반 액세스 컨트롤

rbac.authorization.k8s.ioAPI

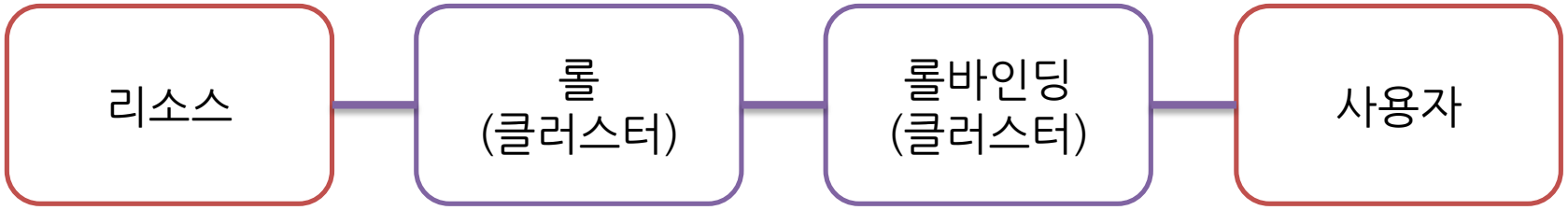
- RBAC를 다루는 이 API는 총 4가지의 리소스를 컨트롤
  - Role
  - RoleBinding
  - ClusterRole
  - ClusterRoleBinding



RBAC를 활용한 롤 기반 액세스 컨트롤

rbac.authorization.k8s.ioAPI

- **롤과 롤바인딩의 차이**
  - 롤
    - ✓ “누가”하는 것인지는 정하지 않고 롤만을 정의
    - ✓ 일반 롤의 경우에는 네임스페이스 단위로 역할을 관리
    - ✓ 클러스터롤은 네임스페이스 구애 받지 않고 특정 자원을 전체 클러스터에서 자원을 관리할 롤을 정의
  - 롤 바인딩
    - ✓ “누가”하는 것인지만 정하고 롤은 정의하지 않음
    - ✓ 롤을 정의하는 대신에 참조할 롤을 정의 (roleRef)
    - ✓ 어떤 사용자에게 어떤 권한을 부여할지 정하는(바인딩) 리소스
    - ✓ 클러스터롤에는 클러스터롤바인딩, 일반 롤에는 롤바인딩을 필요



RBAC를 활용한 롤 기반 액세스 컨트롤

- rbac.authorization.k8s.ioAPI
  - 롤과 롤바인딩 예제

```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: deployment-manager-binding
  namespace: office
subjects:
- kind: User
  name: gasbugs
  apiGroup: ""
roleRef:
  kind: Role
  name: deployment-manager
  apiGroup: ""
```

```yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  namespace: office
  name: deployment-manager
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["deployments", "replicasets", "pods"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

RBAC를 활용한 롤 기반 액세스 컨트롤

rbac.authorization.k8s.ioAPI

- 룰(Rule) 작성 요령

```yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  namespace: office
  name: deployment-manager
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["deployments", "replicasets", "pods"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

- 룰에는 api 그룹, 리소스, 작업 가능한 동작을 작성
  - API를 사용하게 할 그룹 정의

시큐리티 콘텍스트

시큐리티 콘텍스트

보안 컨텍스트

- 서버 침해사고 발생 시 침해사고를 당한 권한을 최대한 축소하여 그 사고에 대한 확대를 방지
- 침해사고를 당한 서버가 root 권한으로 돌아가는 경우 서버를 공격한 공격자는 그대로 서버의 권한을 사용할 수 있음
- 포드 또는 컨테이너에 대한 권한 및 액세스 제어 설정 정의
- 보안 컨텍스트 설정에는 다음 사항을 포함
  - 임의 액세스 제어 : 파일과 같은 오브젝트에 대한 액세스 권한은 사용자 ID(UID) 및 그룹 ID(GID)를 기반
  - SELinux (Security Enhanced Linux) : 오브젝트에 보안 레이블이 지정
  - 권한이 있거나 권한이없는 사용자로 실행 중
  - Linux 기능 : 프로세스에 일부 권한을 부여하지만 루트 사용자의 모든 권한은 부여하지 않음
  - AppArmor : 프로그램 프로필을 사용하여 개별 프로그램의 기능을 제한하십시오.
  - Seccomp : 프로세스의 시스템 호출을 필터링합니다.
  - AllowPrivilegeEscalation : 프로세스가 상위 프로세스보다 많은 권한을 얻을 수 있는지 여부를 제어합니다. 이 bool은 no_new_privs플래그가 컨테이너 프로세스에서 설정 되는지 여부를 직접 제어합니다 . AllowPrivilegeEscalation은 컨테이너가 다음과 같은 경우 항상 true입니다. 1) Privileged OR 2로 실행됩니다 CAP_SYS_ADMIN.

시큐리티 콘텍스트

security-context 예제

```
$ kubectl exec -it security-context-demo /bin/bash
$ id
uid=1000 gid=0(root) groups=0(root),2000
```

security-context.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: sec-ctx-vol
    emptyDir: {}
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: sec-ctx-vol
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: false
```

시큐리티 콘텍스트

security-context 예제2

- 다수의 컨테이너가 실행할 때는 포드 전체와 컨테이너에 모두 설정 가능
- 포드 전체에 대한 설정이 기본적으로 설정
- 컨테이너에 내부에 새로운 유저를 사용하면 그 설정이 우선됨

security-context-2.yaml

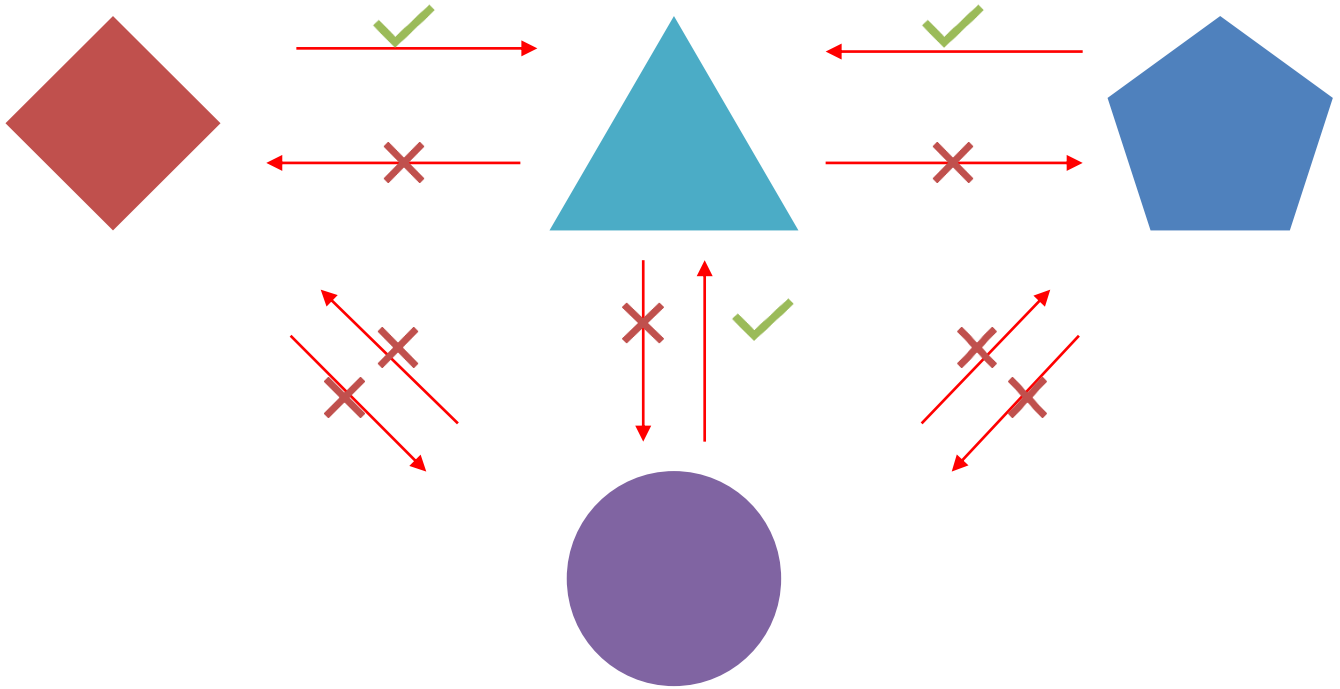
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo-2
spec:
  securityContext:
    runAsUser: 1000
  containers:
  - name: sec-ctx-demo-2
    image: gcr.io/google-samples/node-hello:1.0
    securityContext:
      runAsUser: 2000
      allowPrivilegeEscalation: false
```

네트워크 정책 적용

네트워크 정책 적용

네트워크 정책(Network Policy)

- 포드 그룹이 서로 및 다른 네트워크 끝점과 통신하는 방법을 지정
- NetworkPolicy 리소스는 라벨을 사용하여 포드를 선택
- 선택한 포드에 허용되는 트래픽을 지정하는 규칙을 정의
- 특정 포드를 선택하는 네임 스페이스에 NetworkPolicy가 있으면 해당 포드는 NetworkPolicy에서 허용하지 않는 연결을 거부



네트워크 정책 적용

이그레스

- 선택된 포드에서 나가는 트래픽에 대한 정책 설정
- ipBlock을 통해 Block 하고자 하는 ip 대역 설정 가능
- Ports에는 어떤 포드를 막고자 하는지 명시

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Egress
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 5978
```

네트워크 정책 적용

인그레스

- 선택된 포드로 들어오는 트래픽에 대한 정책 설정
- IpBlock은 기본적으로 막고자 하는 IP 대역을 설정하나
- Exept를 사용하여 예외 항목 설정 가능
- NamespaceSelector와 podSelector를 사용
  - 그룹별 더욱 상세한 정책 설정 가능

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
```

클러스터, 애플리케이션 트러블 슈팅 가이드

클러스터, 애플리케이션 트러블 슈팅 가이드

트러블은 언제나 발생

- 트러블은 매번 이슈가 됨
- 쿠버네티스도 예외는 아니며 시스템이 단순하지 않기 때문에 더 철저한 점검 방법이 필요
- 트러블 슈팅의 기본은 최소단위로 테스트하기!
- 크게 두 가지로 클러스터, 애플리케이션 트러블이 발생 가능

클러스터, 애플리케이션 트러블 슈팅 가이드

애플리케이션 트러블 슈팅

- 애플리케이션에서 트러블이 발생하는 경우에는 해결 방법이 비교적 단순
- 각각의 포드, 컨피그맵 등의 로직에서 무엇이 잘못됐는지 확인
- 일반적으로 이름이나 라벨링이 다른 데서 발생

클러스터, 애플리케이션 트러블 슈팅 가이드

클러스터 트러블 슈팅

- 클러스터에서 발생하는 트러블 슈팅의 경우는 쿠버네티스 시스템에 대한 깊은 이해를 요구
- 일반적으로 다수에서 발생하기 보다는 한 두 군데의 장애가 존재
- Kubectl이 접속되지 않을 때는 serverapi 자체의 문제일 가능성이 큼
- 컨테이너가 장시간 배치되지 않는 현상은 스케줄러의 문제일 가능성이 존재
- Kubectl에 접속이 됐으나 다른 시스템에 사용되는 포드들이 올라오지 않을 경우에는 개별 yaml과 conf를 각각 확인 필요
- 특정 노드나 포드가 통신되지 않을 때는 kubelet이 제대로 동작하는지 확인, 특별한 경우 Certifiace의 이슈가 있을 수 있음
- 참고하면 좋은 문서 : https://kubernetes.io/docs/tasks/debug-application-cluster/debug-cluster/

- 마스터 노드의 로그 저장 위치
  - /var/log/kube-apiserver.log
  - /var/log/kube-scheduler.log
  - /var/log/kube-controller-manager.log

- 워커 노드의 로그 저장 위치
  - /var/log/kubelet.log
  - /var/log/kube-proxy.log

클러스터, 애플리케이션 트러블 슈팅 가이드

클러스터 트러블 슈팅

- Journalctl을 사용하여 동작하지 않는 노드의 데이터 확인
  - journalctl -u kubelet -f

WordPress Mysql 구축과 YAML 분석

시스템 구축 방법을 정리하고 네트워크를 그리시오.