# 3. Container/Kubernetes

- centos7에서 docker 설치
- container에서 the input device is not a TTY 로그 출력시 조치
- docker 와 cri-o 비교
- Docker컨테이너 저장경로 변경방법
- k8s 강제로 pod 종료시키는 방법
- kubernetes-Istio구성정보
- Runtime별 컨테이너 정보 설정(Docker/podman)
- Podman기반의 DNS연동
- podman사용하기
- podman에서 컨테이너가 실행되지 않을때 조치방법
- etcd의 member확인
- kubeadm 노드추가
- etcd member 제외 방법
- 컨테이너 구동시 unknown server host 메시지 뜨면서 구동실패
- pod 구동될때 Imageinspecterror or readlink invalid argument error 메시지 발생
- 워커노드 제외방법
- 특정 노드에만 배포할 수 있는 taint 설정하기
- etcd 데이터 백업 및 복원방법

# centos7에서 docker 설치

1. yum 리포지터리 추가

```
$> wget https://download.docker.com/linux/centos/docker-ce.repo -O /etc/yum.repos.d/docker.repo
$> yum install docker-ce -y
```

2. docker서비스 활성화 후 시작

```
$> systemctl enable docker --now
```

3. Docker 정보 확인

```
$> docker info
docker info
Client:
Context: default
Debug Mode: false
Plugins:
app: Docker App (Docker Inc., v0.9.1-beta3)
buildx: Build with BuildKit (Docker Inc., v0.5.0-docker)
Server:
Containers: 10
Running: 10
Paused: 0
Stopped: 0
Images: 17
Server Version: 20.10.1
Storage Driver: overlay2
Backing Filesystem: xfs
Supports d_type: true
Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Cgroup Version: 1
Plugins:
Volume: local
Network: bridge host ipvlan macvlan null overlay
Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
Swarm: inactive
Runtimes: io.containerd.runc.v2 io.containerd.runtime.v1.linux runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 269548fa27e0089a8b8278fc4fc781d7f65a939b
runc version: ff819c7e9184c13b7c2607fe6c30ae19403a7aff
init version: de40ad0
Security Options:
seccomp
Profile: default
Kernel Version: 3.10.0-1160.6.1.el7.x86_64
Operating System: CentOS Linux 7 (Core)
OSType: linux
Architecture: x86_64
CPUs: 8
Total Memory: 15.42GiB
Name: test.co.kr
ID: YDW3:HAGH:ZX2W:WBP4:US75:UK3H:A55N:M2CG:D7A7:5Q5Q:45JS:GUWV
Docker Root Dir: /home/docker
Debug Mode: false
Registry: https://index.docker.io/v1/
Labels:
Experimental: false
Insecure Registries:
127.0.0.0/8
Live Restore Enabled: false
```

# container에서 the input device is not a TTY 로그 출력시 조치

container에 저장된 데이터를 export할때 편리하게 하려고 crontab에 걸어놨는데,

다음날 보니 export된 데이터의 사이즈가 0k.

```
$> ls -l
total 212
-rw-r--r-- 1 root root      0 Jan 23 01:02 back-2022-01-23.sql
```

혹시나 싶어서 mail로그를 보니. 이런게 똭..

```
$> cat /var/spool/mail/root
...
X-Cron-Env: <SHELL=/bin/sh>
X-Cron-Env: <HOME=/root>
X-Cron-Env: <PATH=/usr/bin:/bin>
X-Cron-Env: <LOGNAME=root>
X-Cron-Env: <USER=root>
Message-Id: <20220123160541.test-machine>
Date: Mon, 24 Jan 2022 01:02:01 +0900 (KST)

the input device is not a TTY
...
```

생각해보니 docker exec에서 input + terminal 옵션을 넣었는데 terminal때문에 꼬인듯.

```
$> cat /root/db_dump.sh
#!/bin/bash

NEW_DATE=$(date +%Y-%m-%d)
OLD_DATE=$(date +%Y-%m-%d -d '-30 days')

if [[ ! -d /home/backup/$NEW_DATE ]]
then
 mkdir -p /home/backup/$NEW_DATE
fi
```

#예전코드

```
$> docker exec -it postgres pg_dump -U test -d testdb > back-$NEW_DATE.sql
```

#변경코드

```
$> docker exec -i postgres pg_dump -U test -d testdb > back-$NEW_DATE.sql
```

음.. 잘됨.

```
$> ls -l
total 212
```

```
-rw-r--r-- 1 root root   102488 Jan 23 09:54 back-2022-01-23.sql
```

# docker 와 cri-o 비교

## 소 개

1. Container 표준규격이 없을때 Container 포맷과 런타임의 사실상 표준은  Docker가 지배.
2. Google / Redhat / MS / IBM 등 Container 간의 이식성을 표준화 하기 위해 OCI(Open Container Initiative) 구성
3. k8s의 컨테이너 런타임 구성을 위해 CRI(Container Runtime Interface) 등장
4. RHEL 8버전이상, k8s 1.20 버전이상, Openstack 16버전이상, Openshift 4버전이상 , awx 18버전 이상에서 컨테이너 런타임을 podman으로 변경

## 컨테이너 런타임 레벨 종류

1. low-level runtime
    1. 각 컨테이너별 cpu나 메모리 같은 리소스 양 제한하는 역할 (ex. runC, Container, Docker)
2. high-level runtime
    1. 어플리케이션을 실행하고 모니터링을 위한 API제공
    2. low-level 위에 실행되는 runtime (ex. containerd, docker, cri-o)

## CRI-O 소개

1. cri-o는 컨테이너 실행만 가능하기 때문에 이미지 생성이나 관리를 하기 위해서는 추가 Component들이 필요함. (추가 Component정보는 아래 기술)
2. fork/exec 모델로 작동
3. 인식가능한 파일시스템은 Overlayfs, devicemapper, btrfs 기반에서 가능하고, nfs,glusterfs,cephfs는 아직 stable하지 않음)
4. docker schema v1/2 모두 지원

## CRI-O 컴포넌트 종류

1. podman(포드맨) : Container 실행을 위한 Tool
2. Buildah(빌다) : Container 이미지 생성
3. skopeo(스코피오) : 이미지 저장소 관리 Tool

## crictl과 podman 차이

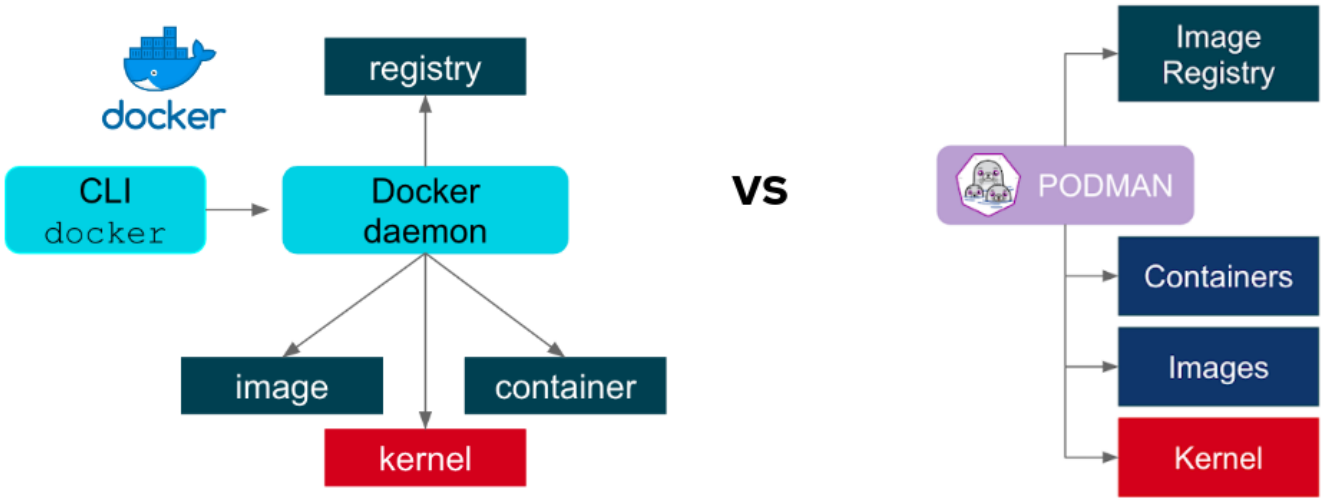
1. crictl은 cri프로토콜을 사용해서 cli  제공
2. podman은 daemon-less로 pod과 컨테이너를 관리하는 cli, docker와 동일한 cli기능 제공.
3. 다만 cri-o와 직접 통신하지는 않기 때문에 cri-o에서 생성된 컨테이너를 볼수 없음.(libpod라는 podman의 컨테이너 관리 라이브러리 개발을 통해 개선할 예정)
4. Docker에서 실행 혹은 빌드된 컨테이너들을 podman으로 실행 가능.

## 프로세스 구동모델 소개

1. fork/exec 모델과 서버/클라이언트 모델의 동작 방식 비교


[그림 6] fork·exec 모델 및 클라이언트-서버 모델에서의 UID, auid 설정 동작 방식

2. podman 과 Docker의 프로세스 실행 비교




## 3. 프로세스 구동방식 비교

| | Docker | Podman |
|---|---|---|
| **구동 방식** | Docker daemon 구동필요 | Daemon-less (systemd 에서 podman 실행) |
| **실행 권한** | 프로세스 구동시 root 권한 필요 | root-less (1024 이하 포트 실행시에만 root권한 필요) |
| **프로세스 실행 모델** | 서버/클라이언트 | fork/exec |

# *reference*


- *https://cri-o.io/*
- *https://www.openshift.com/blog/crictl-vs-podman*
- *https://www.s-core.co.kr/insight/view/oci%EC%99%80-cri-%EC%A4%91%EC%8B%AC%EC%9C%BC%EB%A1%9C-%EC%9E%AC%ED%8E%B8%EB%90%98%EB%8A%94-%EC%BB%A8%ED%85%8C%EC%9D%B4%EB%84%88-%EC%83%9D%ED%83%9C%EA%B3%84-%ED%9D%94%EB%93%A4%EB%A6%AC%EB%8A%94/*
- *https://www.redhat.com/ko/blog/introducing-cri-o-10*
- *https://docs.openshift.com/container-platform/3.11/crio/crio_runtime.html*

# Docker컨테이너 저장경로 변경방법

Docker를 rpm이나 deb같은 패키지로 설치하는 경우 기본 경로가 /var/lib/docker 인데, 대부분 /, /var가 같은 디스크를 사용하고 있기 때문에 컨테이너 용량이 커지면 OS영역에서 사용하는 디스크 사용량이 같이 증가하게 되는데...
바로 이렇게....

```
$ df -h
Filesystem Size Used Avail Use% Mounted on
devtmpfs   7.7G       0  7.7G      0% /dev
tmpfs         7.8G 0 7.8G 0% /dev/shm
tmpfs         7.8G 738M 7.0G 10% /run
tmpfs         7.8G 0 7.8G 0% /sys/fs/cgroup
/dev/sda2 438G 430G  8G    99% /
/dev/sdb1      2T  500G 1.5T     25% /data/
/dev/sda1 497M 275M 223M 56% /boot
overlay 438G 430G 8G 99% /var/lib/docker/overlay2/53de091a6179957607c19a836f65ce1f9f4a43308cf7b24911907958c9e9f2a2/merged
```

```
$ du -hs /var/lib/docker
420G /var/lib/docker
```

OS영역을 총 438G 할당했는데 Docker에서 420G를 사용하면서 전체 사용율 99%로...

/data파티션에 총 2T중 500G를 쓰고 있어서 Docker 데이터를 /data으로 이동하는것으로 조치 진행.

1. Docker stop
   ```
   $ systemctl stop docker
   ```
2. docker 컨테이너 데이터를 /data/docker로 구성
   ```
   $ cat /etc/docker/daemon.json
   {
   "data-root": "/data/docker"
   }
   ```
   - Docker Engine버전에 따라서 daemon.json 파일로 적용되는 버전이 있고 안되는 버전이 있는 것으로 확인
3. docker 데이터 이전
   ```
   $ mv -f /var/lib/docker /data/docker
   ```
4. docker 시작
   ```
   $ systemctl start docker
   ```
5. docker 설정정보 확인
   ```
   $ docker info
   ...
   Docker Root Dir: /data/docker
   ...
   ```

# 컨테이너 저장경로 변경

1. podman 기반에서 컨테이너 스토리지 경로 변경
   ```
   $> vi /etc/containers/storage.conf
   ...
   runroot = "/service/containers/storage"
   graphroot = "/service/containers/storage"
   ...
   ```

2.

# k8s 강제로 pod 종료시키는 방법

k8s기반에서 강제로 pod 종료

1. 죽지않는 pod 정보

```
[root@control1 ~]# kubectl get  pod test-68d6bf4d95-zhx55 -n test_user
NAME                    READY   STATUS        RESTARTS   AGE
test-68d6bf4d95-zhx55   1/4     Terminating   0          68d
```

2. 강제로 pod 삭제

```
[root@control1 ~]# kubectl delete  pod test-68d6bf4d95-zhx55 -n test_user -grace-period=0 -force
warning: Immediate deletion does not wait for confirmation that the running resource has been terminated. The resource may continue to run on the cluster indefinitely.
pod "test-68d6bf4d95-zhx55" force deleted
```

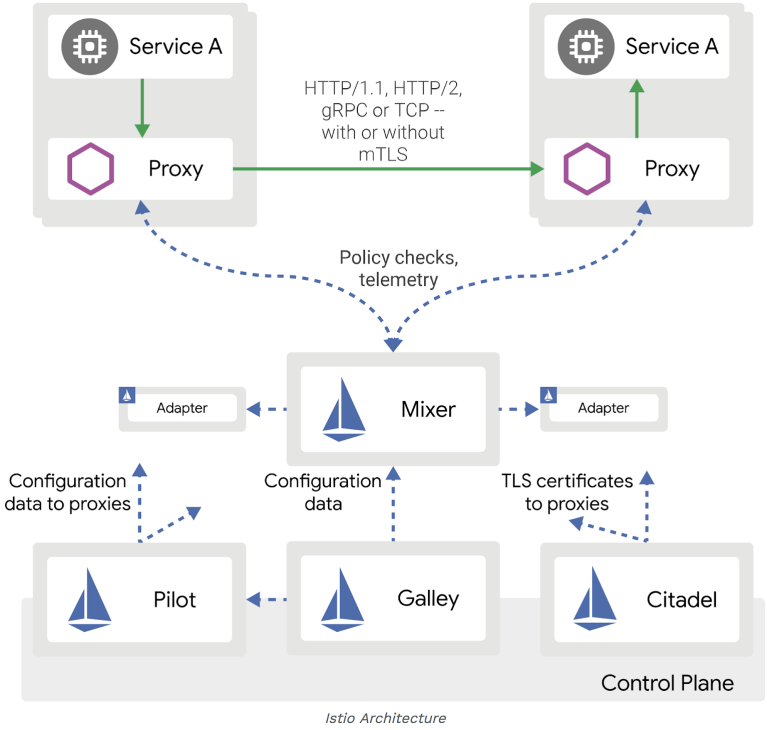
3. pod 정보 없는거 확인

```
[root@control1 ~]# kubectl get  pod test-68d6bf4d95-zhx55 -n test_user
Error from server (NotFound): pods "test-68d6bf4d95-zhx55" not found
```

# kubernetes-Istio구성정보

# Istio Concept & Data Flow

1. Istio의 전체 Concept & Data Flow




*Istio Architecture*

2. Concept
   1. ServiceMesh를 이용해 다양한 트래픽을 제어하는 역활
   2. http / websocket / http 등 트래픽 제어/관리에 대한 부하분산 수행
3. Component별 역활
   1. Data Plane
      - Service A / B에 구성된 Pod에는  Proxy용 Envoy Sidecar container가 배포
   2. Control Plane
      - Mixer - 정책 설정 / ACL / 인증 역활
      - Pilot : ingress routing, traffic mirroring, traffic shifting, canary deployments, circuit breaking, fault injection 역활 수행
      - Galley  : yaml을 istio용으로 변환 후 pilot으로 전송하고.
      - Citadel : 데이터 전송시 암호화 전송(TLS) 역활, 현재 내부 통신에는 암호화과정이 없어서 사용하지 않음
4. 구성시 유의사항
   - istio기반의 traffic shaping을 적용하려면, k8s에서 사용하는 ingres(or service nodeport)를 사용하면 적용이 안되고 service는 clusterip로 적용하고, istio ingressgateway에서 port를 정의해주어야 함.

*Servicemesh : MSA기반의 아키텍텨는 각각 개별기능을 수행하는데, 분산 서비스 배포의 크기와 복잡성이 증가하면서 시스템을 이해하고 관리하기 더 어려워지짐. 클러스터 내부와 외부의 통신 라우팅도 복잡해지기 때문에 이러한 복잡성을 줄기기 위해 프록시를 사용하여 모든 트래픽을 확인 후 사용자가 설정한 구성에 따라 어플리케이션 트래픽을 관리하는 역활.*

# Istio 설치 절차

1. 설치

```
$ curl -L https://istio.io/downloadIstio | sh -
$ cd istio-1.9.2
$ ./istioctl install --set profile=default
This will install the Istio 1.9.2  profile with ["Istio core"gateways"] components into the cluster. Proceed? (y/N) y
✔ Istio core installed
```

```
✔ Istiod installed
✔ Ingress gateways installed
✔ Installation complete
```

```
# Istio기반의 proxy를 설치하기 위해서 namespace에 istio용 envoy 설치
$ kubectl label namespace default istio-injection=enabled
namespace/default labeled
```

2. istio profile별 제공 기능

| | default | demo | minimal | remote | empty | preview |
|---|---|---|---|---|---|---|
| Core components | | | | | | |
| istio-egressgateway | | ✓ | | | | |
| istio-ingressgateway | ✓ | ✓ | | | | ✓ |
| istiod | ✓ | ✓ | ✓ | | | ✓ |

3. istio 서비스 상태 확인

```
$ kubectl get all -n istio-system
NAME READY STATUS RESTARTS AGE
pod/istio-ingressgateway-78d7b9b7db-zpxxf 1/1 Running 2 19d
pod/istiod-85c8645bbc-4jkbj 1/1 Running 1 19d

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
service/istio-
ingressgateway LoadBalancer 10.233.14.72 <pending> 15021:31094/TCP,80:32134/TCP,443:31338/TCP,15012:30093/TCP,15443:30233/TCP 20d
service/istiod ClusterIP 10.233.43.248 <none> 15010/TCP,15012/TCP,443/TCP,15014/TCP 20d
service/tracing NodePort 10.233.13.45 <none> 16686:30008/TCP 17d

NAME READY UP-TO-DATE AVAILABLE AGE
deployment.apps/istio-ingressgateway 1/1 1 1 20d
deployment.apps/istiod 1/1 1 1 20d

NAME DESIRED CURRENT READY AGE
replicaset.apps/istio-ingressgateway-78d7b9b7db 1 1 1 20d
replicaset.apps/istiod-85c8645bbc 1 1 1 20d

NAME REFERENCE TARGETS MINPODS MAXPODS REPLICAS AGE
horizontalpodautoscaler.autoscaling/istio-ingressgateway Deployment/istio-ingressgateway <unknown>/80% 1 5 1 20d
horizontalpodautoscaler.autoscaling/istiod Deployment/istiod <unknown>/80% 1 5 1 20d
```

4. Addon 설치
   1. Kiali : Istio 모니터링을 위한 대쉬보드
   2. Jager / zipkin : 분산 시스템 모니터링
      1. zipkin : Twitter에서 개발한 오픈소스
      2. jaeger: Uber에서 개발하고 CNCF 프로젝트로 진행중인 오픈소스. (k8s환경에서는 jaeger가 효율적이라는...)
5. addon 설치

```
$ wget http://172.21.115.91:28080/...
$ kubectl apply -f ./sample/
$ kubectl get all -n istio-system
NAME READY STATUS RESTARTS AGE
pod/istio-ingressgateway-78d7b9b7db-zpxxf 1/1 Running 2 19d
pod/istiod-85c8645bbc-4jkbj 1/1 Running 1 19d
pod/jaeger-7f78b6fb65-jcrgz 1/1 Running 1 17d
pod/kiali-dc84967d9-cqn8v 1/1 Running 1 19d
pod/prometheus-7bfddb8dbf-vsddf 2/2 Running 4 19d

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
service/istio-
ingressgateway LoadBalancer 10.233.14.72 <pending> 15021:31094/TCP,80:32134/TCP,443:31338/TCP,15012:30093/TCP,15443:30233/TCP 20d
```

```
service/istiod               ClusterIP      10.233.43.248   <none>        15010/TCP,15012/TCP,443/TCP,15014/TCP                                  20d
service/jaeger-collector     ClusterIP      10.233.35.73    <none>        14268/TCP,14250/TCP                                                    17d
service/kiali                NodePort       10.233.21.50    <none>        20001:30007/TCP,9090:31990/TCP                                         20d
service/prometheus           ClusterIP      10.233.24.217   <none>        9090/TCP                                                               20d
service/tracing              NodePort       10.233.13.45    <none>        16686:30008/TCP                                                        17d
service/zipkin               ClusterIP      10.233.34.17    <none>        9411/TCP                                                               17d

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/istio-ingressgateway   1/1     1            1           20d
deployment.apps/istiod                 1/1     1            1           20d
deployment.apps/jaeger                 1/1     1            1           17d
deployment.apps/kiali                  1/1     1            1           20d
deployment.apps/prometheus             1/1     1            1           20d

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/istio-ingressgateway-78d7b9b7db   1         1         1       20d
replicaset.apps/istiod-85c8645bbc                 1         1         1       20d
replicaset.apps/jaeger-7f78b6fb65                 1         1         1       17d
replicaset.apps/kiali-dc84967d9                   1         1         1       20d
replicaset.apps/prometheus-7bfddb8dbf             1         1         1       20d

NAME                                                                    REFERENCE                         TARGETS          MINPODS   MAXPODS   REPLICAS   AGE
horizontalpodautoscaler.autoscaling/istio-ingressgateway   Deployment/istio-ingressgateway   <unknown>/80%   1         5         1          20d
horizontalpodautoscaler.autoscaling/istiod                 Deployment/istiod                 <unknown>/80%   1         5         1          20d
```

6. 대쉬보드 외부 접속하기
   1. kiali : http://{{서버IP}}:30007
   2. jaeger: http://{{서버IP}}:30008

      1.삭제

```
$ istioctl x uninstall --purge
kubectl delete namespace istio-system
```

7. 시스템 설정을 위한 설정값 안내

   Component list

| 항목 | 용도 | 기타 |
|---|---|---|
| gateway | http / tcp를 연결하기 위해 구성하는 로드밸런서 | |
| virtualservice | service | 게이트웨이에 바인딩 후 트래픽을 구성된 여러 엔드포인트로 전달 (트래픽 비율설정) |
| virtualservice | version(a.k.a subset) | 특정 서비스에 대해 어플리케이션 바이너리의 버전 변경을 실행하는 집합 |
| virtualservice | source | 서비스를 호출하는 다운스트림용 클라이언트 |
| virtualservice | host | 클라이언트가 서비스 연결할때 사용하는 주소 |
| destinationRule | 라우팅 규칙을 처리한 후 연결할 네트워크 서비스를 설정 | |
| tcproute | tcp트래픽에 대해 라우팅 규칙을 위한 조건 설정 | |
| tcproute | match | 활성화할 규칙조건 |
| tcproute | route | 연결할 대상 |

8. Example yml
   1. gateway 설정 (tcp/30011에 대해 gateway 설정)

```
$vi gateway.yml
---
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: gateway
  namespace: test
spec:
  selector:
    app: test
  servers:
    - hosts:
        - '*'
      port:
        name: tcp
        number: 30011
        protocol: TCP
```

9. VirtualService 설정 (tcp/30011이 들어오면 service111-1, service111-2의 tcp/8080으로 연결하되  50%씩 트래픽 분할 처리)

```
$ vi vs.yaml
---
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: test-vs
  namespace: test
spec:
  gateways:
    - gateway
  hosts:
    - appid111
  tcp:
    - match:
        - port: 30011
      route:
        - destination:
            host: appid111-1
            port:
              number: 3390
            subset: v1
          weight: 50
        - destination:
            host: appid111-2
            port:
              number: 3390
            subset: v2
          weight: 50
```

10. destinationrule 설정 (service111에 대해 v1, v2 설정)

```
$ vi rule.yml
---
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: test-rule
  namespace: test
spec:
  host: appid111
  subsets:
    - labels:
        version: 'v1'
      name: v1
    - labels:
        version: 'v2'
      name: v2
  trafficPolicy:
    loadBalancer:
```

```
    simple: ROUND_ROBIN
  tls:
    mode: DISABLE
```

reference
https://istio.io/latest/docs/setup/getting-started/

# Runtime별 컨테이너 정보 설정 (Docker/podman)

## 시작하는말

안녕하세요, 고니입니다.
기존에 작성했던 컨텐츠들 업데이트를 하면서 문서의 리팩토링(Refactoring)을 진행해보려고 합니다.

## Insecure registry란?

레지스트리는 통한 컨테이너 pulling은 https가 기본통신이지만, 특수한 환경에 의해 http로 통신해야 하는 경우 insecure registry 설정을 수행해야 http로 작업이 가능합니다.

## 설정작업

### 1. Docker 기반에서 수행

- Docker기반의 경우 설치된 환경에 따라 참조하는 파일 위치가 분리되어 있기 때문에 현재 사용중인 환경에 맞추어 한군데의 파일만 설정하면 됩니다.
- systemd에 설정된 설정값 변경

```
$> cat /etc/systemd/system/docker.service.d/docker-options.conf
[Service]
Environment="DOCKER_OPTS=--bip 172.10.255.1/24 \
...
--insecure-registry={{ 등록할 호스트1 }} --insecure-registry={{ 등록할 호스트2 }} \
...
```

- daemon.json에 정의된 값 변경

```
$> cat /etc/docker/daemon.json
{
 "insecure-registries":["등록할 호스트1", "등록할 호스트2"]
}
```

- Docker 재기동

```
$> systemctl daemon-reload
$> systemctl restart docker
```

### 2. Podman 기반에서 수행

```
$> vi /etc/containers/registries.conf
...
unqualified-search-registries = ["registry.fedoraproject.org", "registry.access.redhat.com", "registry.centos.org", "docker.io", "20.20.20.20", "10.10.10.10"]
...
[[registry]]
location = "10.10.10.10:80"
insecure = true

[[registry]]
location = "20.20.20.20:80"
insecure = true
```

# 확인방법

1. Docker 기반에서 확인

```
$> docker info
...
Insecure Registries
{{ 등록할 호스트1 }}
{{ 등록할 호스트2 }}
```

2. Podman에서 확인

```
$> podman login 10.10.10.10:80
Username:
Password:
Login Successed
```

- daemonless이기 때문에 바로 적용
- 기존 Docker에서 컨테이너 내려받을때는 포트정보가 없어도 되었지만, podman에서 내려받을때는 :80을 꼭 명시해야 함(default가 443이기 때문에 포트정보가 없는 경우 443으로 내려받으려고 시도함

# Podman기반의 DNS연동

podman을 사용할 경우 네트워크는 cni를 사용하게 되는데, 기본값은 호스트의 DNS를 사용하기 때문에 컨테이너 내부의 DNS를 사용해야할 경우 호출이 불가능.

1. docker기반에서 컨테이너간 dns통신상태 확인

```
$> docker exec -it awx_task ping redis
PING redis (172.18.0.5) 56(84) bytes of data.
64 bytes from awx_redis.awxcompose_default (172.18.0.5): icmp_seq=1 ttl=64 time=0.240 ms
64 bytes from awx_redis.awxcompose_default (172.18.0.5): icmp_seq=2 ttl=64 time=0.090 ms
64 bytes from awx_redis.awxcompose_default (172.18.0.5): icmp_seq=3 ttl=64 time=0.074 ms
^C
--- redis ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 3ms
rtt min/avg/max/mdev = 0.074/0.134/0.240/0.075 ms
```

2. podman 기반에서 컨테이너간 dns통신상태 확인

```
$ podman exec  -it awx_task ping redis
ping: redis: Name or service not known
```

# 조치사항

1. cni에 dnsname 플러그인 설치 및 NetworkManager 연동

2. dns패키지 설치

```
$> yum install dnsmasq podman-plugins -y
```

3. NetworkManager dnsname 플러그인 연동 후 서비스 재시작

```
$> vi /etc/NetworkManager/NetworkManager.conf
...
[main]
dns=dnsmasq
...
```

4. NetworkManager에 dnsname 플러그인 설치 여부 확인

```
$> ls -l /usr/libexec/cni/
합계 73252
...
-rwxr-xr-x 1 root root 3980624  2월  2 09:00 dnsname
...
```

5. cni에 dnsname 플러그인 설치

```
$> vi /etc/cni/net.d/awxcompose_default.conflist
...
    {
      "type": "firewall",
      "backend": ""
    },
    {
      "type": "tuning"
    },
    {
       "type": "dnsname",
       "domainName": "awxcompose_default",
       "capabilities": {
           "aliases": true
       }
    }
```

```
  ]
}
```

6. **서버 재부팅**
7. **내부 통신 확인**

```
$>  podman exec  -it awx_task ping redis
PING redis (10.89.0.2) 56(84) bytes of data.
64 bytes from awx_redis (10.89.0.2): icmp_seq=1 ttl=64 time=0.119 ms
64 bytes from awx_redis (10.89.0.2): icmp_seq=2 ttl=64 time=0.047 ms
64 bytes from awx_redis (10.89.0.2): icmp_seq=3 ttl=64 time=0.070 ms
64 bytes from awx_redis (10.89.0.2): icmp_seq=4 ttl=64 time=0.072 ms
^C
--- redis ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 155ms
rtt min/avg/max/mdev = 0.047/0.069/0.119/0.021 ms
```

# podman사용하기

1. docker없이 컨테이너 실행하기
    1. podman-docker 패키지를 통해 docker가 설치되어 있지 않아도 docker 명령어를 실행할 수 있습니다.
    2. 사실 커맨드가 매핑도이 있는것이고 실제로는 podman을 실행하게 됩니다.

```
$> yum install podman-docker -y
마지막 메타자료 만료확인 1:40:24 이전인: 2022년 02월 07일 (월) 오후 12시 21분 09초.
종속성이 해결되었습니다.
================================================================================
 꾸러미    구조   버전                          레포지터리    크기
================================================================================
설치 중:
 podman-docker
          noarch 3.3.1-9.0.1.module+el8.5.0+20416+d687fed7 ol8_appstream  56 k

연결 요약
================================================================================
설치  1 꾸러미

총계 내려받기 크기: 56 k
설치된 크기 : 230
꾸러미 내려받기중:
podman-docker-3.3.1-9.0.1.module+el8.5.0+20416+d 288 kB/s |  56 kB     00:00
--------------------------------------------------------------------------------
합계                                             284 kB/s |  56 kB     00:00
연결 확인 실행 중
연결 확인에 성공했습니다.
연결 시험 실행 중
연결 시험에 성공했습니다.
연결 실행 중
  준비 중          :                                        1/1
  설치 중          : podman-docker-3.3.1-9.0.1.module+el8.5.0+20416+d687f   1/1
  스크립트릿 실행 중: podman-docker-3.3.1-9.0.1.module+el8.5.0+20416+d687f   1/1
[/usr/lib/tmpfiles.d/pesign.conf:1] Line references path below legacy directory /var/run/, updating /var/run/pesign → /run/pesign; please
update the tmpfiles.d/ drop-in file accordingly.

  확인 중          : podman-docker-3.3.1-9.0.1.module+el8.5.0+20416+d687f   1/1

설치되었습니다:
  podman-docker-3.3.1-9.0.1.module+el8.5.0+20416+d687fed7.noarch

완료되었습니다!
```

```
# yum module install container-tools -y
마지막 메타자료 만료확인 0:09:24 이전인: 2022년 02월 07일 (월) 오후 03시 57분 13초.
종속성이 해결되었습니다.
================================================================================
 꾸러미        구조   버전                          레포지터리    크기
================================================================================
그룹/모듈 꾸러미 설치:
 crun           x86_64 1.0-1.module+el8.5.0+20416+d687fed7   ol8_appstream 193 k
 python3-podman noarch 3.2.0-2.module+el8.5.0+20416+d687fed7 ol8_appstream 148 k
 udica          noarch 0.2.5-2.module+el8.5.0+20416+d687fed7 ol8_appstream  51 k
종속 꾸러미 설치 중:
 python3-pytoml noarch 0.1.14-5.git7dea353.el8               ol8_appstream  25 k
 python3-pyxdg  noarch 0.25-16.el8                           ol8_appstream  94 k
모듈 프로파일 설치:
 container-tools/common


연결 요약
================================================================================
설치  5 꾸러미
```

```
총계 내려받기 크기: 510 k
설치된 크기 : 1.6 M
꾸러미 내려받기중:
(1/5): crun-1.0-1.module+el8.5.0+20416+d687fed7. 1.6 MB/s | 193 kB     00:00
(2/5): python3-podman-3.2.0-2.module+el8.5.0+204  65 kB/s | 148 kB     00:02
(3/5): python3-pytoml-0.1.14-5.git7dea353.el8.no  10 kB/s |  25 kB     00:02
(4/5): python3-pyxdg-0.25-16.el8.noarch.rpm       39 kB/s |  94 kB     00:02
(5/5): udica-0.2.5-2.module+el8.5.0+20416+d687fe  33 kB/s |  51 kB     00:01
--------------------------------------------------------------------------------
합계                                              133 kB/s | 510 kB     00:03
연결 확인 실행 중
연결 확인에 성공했습니다.
연결 시험 실행 중
연결 시험에 성공했습니다.
연결 실행 중
  준비 중        :                                                 1/1
  설치 중        : python3-pyxdg-0.25-16.el8.noarch                1/5
  설치 중        : python3-pytoml-0.1.14-5.git7dea353.el8.noarch   2/5
  설치 중        : python3-podman-3.2.0-2.module+el8.5.0+20416+d687fed7   3/5
  설치 중        : udica-0.2.5-2.module+el8.5.0+20416+d687fed7.noarch     4/5
  설치 중        : crun-1.0-1.module+el8.5.0+20416+d687fed7.x86_64        5/5
  스크립트릿 실행 중: crun-1.0-1.module+el8.5.0+20416+d687fed7.x86_64        5/5
  확인 중        : crun-1.0-1.module+el8.5.0+20416+d687fed7.x86_64        1/5
  확인 중        : python3-podman-3.2.0-2.module+el8.5.0+20416+d687fed7   2/5
  확인 중        : python3-pytoml-0.1.14-5.git7dea353.el8.noarch   3/5
  확인 중        : python3-pyxdg-0.25-16.el8.noarch                4/5
  확인 중        : udica-0.2.5-2.module+el8.5.0+20416+d687fed7.noarch     5/5

설치되었습니다:
  crun-1.0-1.module+el8.5.0+20416+d687fed7.x86_64
  python3-podman-3.2.0-2.module+el8.5.0+20416+d687fed7.noarch
  python3-pytoml-0.1.14-5.git7dea353.el8.noarch
  python3-pyxdg-0.25-16.el8.noarch
  udica-0.2.5-2.module+el8.5.0+20416+d687fed7.noarch
```

완료되었습니다!

3. docker의 socket api가 지원되기 때문에 podman-docker 패키지를 설치하면 /var/run/docker.sock과 /var/run/podman/podman.sock의 링크를 설정하기 때문에 docker-py, docker-compose를 이용한 docker api도 그대로 사용할 수 있습니다.
4. podman에서 지원하지 않는 docker 옵션은 네트워크, 노드, 플러그인, 이름변경, 시크릿, 서비스, 스택, docker swarm이 대상입니다.

2. Rooless설정
   1. 호스트 컨테이너 스토리지 경로는 사용자 정보에 따라 달라집니다. (root - /var/lib/containers/storage, non root - $HOME/.local/share/containers/storage)
   2. rootless 컨테이너는 1024미만의 포트에 액세스 할 수 없습니다. (tcp/80을 사용하는 apache의 경우 서비스에서는 tcp/80을 표시하시만 서버 외부에서 실제 접속은 불가능)
   3. 1024 이하 포트를 사용하려면 커널값 변경을 통해 임시로 사용할 수는 있으나, 프로덕션 환경에서는 사용하는것은 권장되지 않습니다. (테스트 목적)

      ```
      $> echo 80 > /proc/sys/net/ipv4/ip_unprivileged_port_start
      ```

   4. 8.1부터 rootless Containers기능을 사용하면 일반사용자로 컨테이너 실행할 수 있습니다. (rootless는 기본기능)

      ```
      $> podman pull docker.io/library/httpd
      Trying to pull docker.io/library/httpd:latest...
      Getting image source signatures
      Copying blob 5eb5b503b376 done
      Copying blob 10c4d45228bf done
      Copying blob a43a76ccc967 done
      Copying blob 942bd346e7f7 done
      Copying blob cdb155854ae6 done
      Copying config a8ea074f45 done
      Writing manifest to image destination
      Storing signatures
      a8ea074f4566addcd01f9745397f32be471df4a4abf200f0f10c885ed14b1d28
      [user@test~]$ docker  image ls
      ```

```
Emulate Docker CLI using podman. Create /etc/containers/nodocker to quiet msg.
REPOSITORY              TAG         IMAGE ID      CREATED      SIZE
docker.io/library/httpd  latest      a8ea074f4566  11 days ago  148 MB
```

3. 레지스트리 구성
    1. registries.conf에 컨테이너를 내려받기 위한 레지스트리 목록이 저장되어 있습니다, 해당 파일에서 검색할 레지스트리를 설정 할 수 있습니다.

```
$> vi /etc/containers/registries.conf
...
unqualified-search-registries = ["container-registry.oracle.com", "docker.io", "container.test.com"]
...
```

    2. test.com은 TLS 없이 pull 설정

```
$> vi /etc/containers/registries.conf
...
[[registry]]
location="container.test.com"
insecure = true
...
```

    3. 특정 레지스트리 검색 차단

```
$> vi /etc/containers/registries.conf
...
[[registry]]
location="container.test.com"
blocked = true
...
```

4. 이미지 검사
    1. skopeo 명령어를 통해 이미지 정보를 확인할 수 있습니다.
    2. httpd container 정보 확인 방법

```
# skopeo inspect docker://docker.io/library/httpd:latest
{
  "Name": "docker.io/library/httpd",
  "Digest": "sha256:5cc947a200524a822883dc6ce6456d852d7c5629ab177dfbf7e38c1b4a647705",
  "RepoTags": [
    "2",
    "2-alpine",
    "2-alpine3.13",
    "2-alpine3.14",
    "2-alpine3.15",
    "2-bullseye",
    "2-buster",
    "2.2",
    "2.2-alpine",
    "2.2.29",
    "2.2.31",
    "2.2.31-alpine",
    "2.2.32",
    "2.2.32-alpine",
    "2.2.34",
    "2.2.34-alpine",
    "2.4",
    "2.4-alpine",
    "2.4-alpine3.13",
    "2.4-alpine3.14",
    "2.4-alpine3.15",
    "2.4-bullseye",
    "2.4-buster",
    "2.4.10",
    "2.4.12",
    "2.4.16",
    "2.4.17",
    "2.4.18",
    "2.4.20",
```

```
      "2.4.23",
      "2.4.23-alpine",
      "2.4.25",
      "2.4.25-alpine",
      "2.4.27",
      "2.4.27-alpine",
      "2.4.28",
      "2.4.28-alpine",
      "2.4.29",
      "2.4.29-alpine",
      "2.4.32",
      "2.4.32-alpine",
      "2.4.33",
      "2.4.33-alpine",
      "2.4.34",
      "2.4.34-alpine",
      "2.4.35",
      "2.4.35-alpine",
      "2.4.37",
      "2.4.37-alpine",
      "2.4.38",
      "2.4.38-alpine",
      "2.4.39",
      "2.4.39-alpine",
      "2.4.41",
      "2.4.41-alpine",
      "2.4.43",
      "2.4.43-alpine",
      "2.4.46",
      "2.4.46-alpine",
      "2.4.47",
      "2.4.47-alpine",
      "2.4.48",
      "2.4.48-alpine",
      "2.4.48-alpine3.13",
      "2.4.48-alpine3.14",
      "2.4.48-buster",
      "2.4.49",
      "2.4.49-alpine",
      "2.4.49-alpine3.14",
      "2.4.49-buster",
      "2.4.50",
      "2.4.50-alpine",
      "2.4.50-alpine3.14",
      "2.4.50-buster",
      "2.4.51",
      "2.4.51-alpine",
      "2.4.51-alpine3.14",
      "2.4.51-alpine3.15",
      "2.4.51-bullseye",
      "2.4.51-buster",
      "2.4.52",
      "2.4.52-alpine",
      "2.4.52-alpine3.15",
      "2.4.52-bullseye",
      "alpine",
      "alpine3.13",
      "alpine3.14",
      "alpine3.15",
      "bullseye",
      "buster",
      "latest"
    ],
    "Created": "2022-01-26T08:38:51.175633696Z",
    "DockerVersion": "20.10.7",
    "Labels": null,
    "Architecture": "amd64",
    "Os": "linux",
    "Layers": [
```

```
      "sha256:5eb5b503b37671af16371272f9c5313a3e82f1d0756e14506704489ad9900803",
      "sha256:a43a76ccc96739928c7e884f2210dde01ae5b1fd8822f8cdd56e6ba64ec3125a",
      "sha256:942bd346e7f719d26e022dfc42eea7e1fa5cf9ad60ec80ed0ef79ded05288be6",
      "sha256:cdb155854ae6a9e25834459a7c9dfc7be157a2ebfca5adbdb036aeea43ce3128",
      "sha256:10c4d45228bf56285a1e2c828d60e34d8413ee80e1abd738ef190be843d9dc1e"
    ],
    "Env": [
      "PATH=/usr/local/apache2/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
      "HTTPD_PREFIX=/usr/local/apache2",
      "HTTPD_VERSION=2.4.52",
      "HTTPD_SHA256=0127f7dc497e9983e9c51474bed75e45607f2f870a7675a86dc90af6d572f5c9",
      "HTTPD_PATCHES="
    ]
```

# podman에서 컨테이너가 실행되지 않을때 조치방법

## 시작하는 말

안녕하세요, 고니 입니다.
기존에 작성했던 컨텐츠들 업데이트를 하면서 문서의 리팩토링(Refactoring)을 진행해보려고 합니다.

podman에서 컨테이너 띄울때 아래 메시지가 나오면서 정상적으로 실행되지 않는 경우가 있어요.

## 실행절차

1. podman에서 컨테이너 띄울때 에러메시지 발생

```
$> podman run -d --name=test
...
Error: runc: container_linux_go:370: staring container process caised: error adding seccomp filter rule for syscall bdflush: permission denied
```

2. 컨테이너 상태를 보면 아래와 같이 state가 구동(Up)이 아니라 생성(created)만 되게 되어요

```
$> docker ps -a
Emulate Docker CLI using podman. Create /etc/containers/nodocker to quiet msg.
CONTAINER ID  IMAGE                     COMMAND               CREATED       STATUS                  PORTS                   NAMES
bd3896eb439d  localhost:5000                                  5 months ago  Created                 0.0.0.0:80->80/tcp  test
```

3. container runtime은 runc와 crun을 사용할 수 있는데요. 기본적으로 사용할 수 있는 runtime이 무엇인지는  docker(podman) info로 확인 할 수 있습니다.

### runtime 확인 방법

1. PODMAN 기반인 경우

```
$> podman info
...
  ociRuntime:
    name: runc
    package: runc-1.1.12-1.module+el8.10.0+1825+623b0c20.x86_64
    path: /usr/bin/runc
...
```

2. DOCKER기반인 경우

```
$> docker info
...
  ociRuntime:
    name: crun
    package: crun-1.8.7-1.el9.x86_64
    path: /usr/bin/crun
...
```

3. runtime의 차이점
    1. crun은 runc에 비해 최대 2배 빠르고, 메모리 사용량은 최대 50배 적습니다
    2. crun은 최소 프로세스 수 설정, rootless 컨테이너의 그룹별 파일 공유, OCI 후크의 stdout/stderr 제어 등 추가 기능을 제공합니다

## 조치방안

1. 한꺼번에 다 할 필요는 없고 1번해보고 해결이 안되면 2번, 2번도 안되면 3번순으로 진행)
2. podman 패키지 업데이트

```
$> yum update podman -y
...
```

3. podman 실행 옵션에서 security설정 변경

```
$> podman run -d --security-opt=seccomp=unconfined --name=test
...
```

4. runtime 종류 변경

```
$> podman run --runtime crun -d --name=test ...
```

# 확인방법

```
$> docker ps -a
Emulate Docker CLI using podman. Create /etc/containers/nodocker to quiet msg.
CONTAINER ID  IMAGE                COMMAND           CREATED       STATUS           PORTS                 NAMES
bd3896eb439d  localhost:5000                        5 months ago  Up 2 weeks ago   0.0.0.0:80->80/tcp  test
```

# *Reference*


1. *https://docs.redhat.com/ko/documentation/red_hat_enterprise_linux/9/html/building_running_and_managing_containers/sela-container-runtime_building-running-and-managing-containers#selecting-a-container-runtime_building-running-and-managing-containers*
2. *https://docs.redhat.com/ko/documentation/red_hat_enterprise_linux/8/html/building_running_and_managing_containers/prcontainers-with-runc-and-crun_selecting-a-container-runtime*

# etcd의 member확인

k8s의 DB성격인 etcd의 상태 점검시 확인할 수 있는 몇가지 포인트들

1. member list

```
$>  /usr/local/bin/etcdctl.sh -w table member list
+------+---------+---------+-----------+------------+
|  ID  | TATUS | PEER ADDRS | CLIENT ADDRESS | IS LEARNER |
+------+---------+---------+-----------+------------+
| abcd |  3.4.13 |   11 MB |     true  |      false |
| 1234 |  3.4.13 |   11 MB |     false |      false |
| efgh |  3.4.13 |   11 MB |     false |      false |
+------+---------+---------+-----------+------------+
```

2. endpoint (데이터 상태 크기 확인)

```
$>  /usr/local/bin/etcdctl.sh -w table endpoint status --cluster=true
+--------------------------+------+---------+---------+-----------+------------+
|       ENDPOINT           |  ID  | VERSION | DB SIZE | IS LEADER | IS LEARNER |
+--------------------------+------+---------+---------+-----------+------------+
| https://192.168.0.10:2379 | abcd |  3.4.13 |   11 MB |     true  |      false |
| https://192.168.0.11:2379 | 1234 |  3.4.13 |   11 MB |     false |      false |
| https://192.168.0.12:2379 | efgh |  3.4.13 |   11 MB |     false |      false |
+--------------------------+------+---------+---------+-----------+------------+
```

# kubeadm 노드추가

**클러스터 노드 추가시 진행되는 작업절차**

1. **클러스터 마스터 노드에서** token **발급진행**

```
$> kubeadm token create --print-join-command
123123123
```

2. join**할 노드에서 아래 명령어 수행 : 추가하 노드는** kubeadm / kubelet **등 기본적인** provisioning**이 되어 있어야 함.**

```
$> kubeadm join {{ masterip }}:6443 --node-name {{ 호스트네임 }} --token 123123 --discovery-token-ca-cert-hash abcd123
```

# etcd member 제외 방법

kuebernetes DB격인 etcd의 member리스트를 제외한느 절차를 기술한다.
다중 마스터(3대로 구현)되어 있을때 특정 노드를 제외하는 방법이다.

1. etcd memver list 확인

```
$> cd /usr/local/bin
$> ./etcdctl.sh -w table endpoint status --cluter=true
+----------------------+------+---------+----------+-----------+------------+------------+------------+--------------------+--------+
|      ENDPOINT        |  ID  | VERSION |  DB SIZE | IS LEADER | IS LEARNER | RAFT TERM  | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+----------------------+------+---------+----------+-----------+------------+------------+------------+--------------------+--------+
| https://1.2.3.4:2379 | abcd | 3.4.13  |    11 MB | true      | false      | 16         |    2607488 |            2607488 |        |
| https://5.6.7.8:2379 | efgh | 3.4.13  |    11 MB | false     | false      | 16         |    2607488 |            2607488 |        |
| https://9.8.7.6:2379 | ijkl | 3.4.13  |    11 MB | false     | false      | 16         |    2607488 |            2607488 |        |
+----------------------+------+---------+----------+-----------+------------+------------+------------+--------------------+--------+
```

2. 제외할 인스턴스 ID입력

```
$> ./etcdctl.sh  member remove efgh
Memeber efgh removed from cluster 1234
```

3. 제외상태 확인

```
$> cd /usr/local/bin
$> ./etcdctl.sh -w table endpoint status --cluter=true
+----------------------+------+---------+----------+-----------+------------+------------+------------+--------------------+--------+
|      ENDPOINT        |  ID  | VERSION |  DB SIZE | IS LEADER | IS LEARNER | RAFT TERM  | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+----------------------+------+---------+----------+-----------+------------+------------+------------+--------------------+--------+
| https://1.2.3.4:2379 | abcd | 3.4.13  |    11 MB | true      | false      | 16         |    2607488 |            2607488 |        |
| https://9.8.7.6:2379 | ijkl | 3.4.13  |    11 MB | false     | false      | 16         |    2607488 |            2607488 |        |
+----------------------+------+---------+----------+-----------+------------+------------+------------+--------------------+--------+
```

# 컨테이너 구동시 unknown server host 메시지 뜨면서 구동실패

## 발생현상

컨테이너 내부에서 통신시 호스트를 못찾는 문제가 있어서(WEB ->DB) 구동실패

## 원 인

Docker 내부에서 hosts를 찾지 않고 docker-dns.conf를 따라서 외부 DNS를 참조해서 리졸빙하는것으로 보임.

## 조치방법

docker-dns를 제외하고 구동.

```
$> mv /etc/systemd/system/docker.service.d/docker-dns.conf /etc/systemd/system/docker.service.d/docker-dns.conf_ori

$> systemctl daemon-reload
$> systemctl restart docker
```

# pod 구동될때 Imageinspecterror or readlink invalid argument error 메시지 발생

## 발생현상

1. pod 구동될때 Imageinspecterror 메시지를 출력하면서 구동되지 않는 문제

```
$>  kubectl describe pod {{ pod명 }}
...
Events
...
Imageinspecterror
```

2. docker data root가 변경된 경우 일부 pod가 캐싱된 경로를 가지고 있어서 발생하는 것으로 보임 - Docker컨테이너 저장경로 변경방법

## 조치방법

1. docker 캐쉬정보 삭제

```
$> docker system prune -af
$> docker volume prune -af
```

2. pod 삭제 (auto healing 기능에 따라 자동으로 다시 배포됨)

```
$> kubectl delete pod --grace-period=0 --force {{ pod명 }}
```

# 워커노드 제외방법

## 시작하는말

안녕하세요, 고니입니다.
기존에 작성했던 컨텐츠들 업데이트를 하면서 문서의 리팩토링(Refactoring)을 진행해보려고 합니다.

## 사전 설명

k8s환경에서 특정 노드 제외할때 그냥 삭제해도 kube scheduling에 의해 재배치가 있긴하지만 계획된 정지 작업같은 작업이 있는경우

보다 안정적으로 노드를 제외하려면 cordon / drain 절차를 거치면 좋습니다.

cordon / drain은 약간의 차이가 있는데, cordon의 경우 단순 스케쥴링을 제외하는 절차, drain의 경우 제외할 노드를 모두 제거하는 절차를 수행합니다. (drain은 cordon절차를 포함합니다.)

## 작업절차

### 1. cordon 설정

- 노드 정보 확인

```
$>  kubectl get no
NAME    STATUS   ROLES                  AGE   VERSION
masr1   Ready    control-plane,master   37d   v1.23.7
work1   Ready    <none>                 37d   v1.23.7
work2   Ready    <none>                 37d   v1.23.7
```

- cordon 수행 (work1노드를 제외할껍니다.)

```
$>  kubectl cordon work1
node/work1 cordoned

$>  kubectl get no
NAME    STATUS                     ROLES                  AGE   VERSION
masr1   Ready                      control-plane,master   37d   v1.23.7
work1   Ready                      <none>                 37d   v1.23.7
work2   Ready,SchedulingDisabled   <none>                 37d   v1.23.7
```

### 2. drain 수행

- 해당 노드에 daemonset으로 구동되는 pod가 있거나 local storage가 구동되고 있는 경우 drain 작업이 수행되지 않습니다.
- 때문에 중요한 데이터는 PV를 사용해서 운영하는걸로.....

```
kubectl drain work2
node/work2 already cordoned
error: unable to drain node "work2", aborting command...

There are pending nodes to be drained:
 k8sw2
cannot delete DaemonSet-managed Pods (use --ignore-daemonsets to ignore): kube-system/calico-node-fb862, kube-system/kube-proxy-4qhsx, kube-system/nodelocaldns-6cptm
```

```
cannot delete Pods with local storage (use --delete-emptydir-data to override): kubernetes-dashboard/dashboard-metrics-scraper-66dd8bdd86-6cnbv, kubernetes-dashboard/kubernetes-dashboard-844749bcff-mmpq2
```

- 강제로 drain 처리

```
kubectl drain work2  --ignore-daemonsets --delete-emptydir-data
node/work2 already cordoned
WARNING: ignoring DaemonSet-managed Pods: kube-system/calico-node-fb862, kube-system/kube-proxy-4qhsx, kube-system/nodelocaldns-6cptm
evicting pod kubernetes-dashboard/kubernetes-dashboard-844749bcff-mmpq2
evicting pod cert-manager/cert-manager-cainjector-75c94654d-r7trs
evicting pod cert-manager/cert-manager-webhook-d4fd4f479-cr6z2
evicting pod gitlab-agent/house-gitlab-agent-v1-586d6d6797-vm68t
evicting pod kube-system/coredns-657959df74-j2zbq
evicting pod kubernetes-dashboard/dashboard-metrics-scraper-66dd8bdd86-6cnbv
pod/cert-manager-cainjector-75c94654d-r7trs evicted
pod/cert-manager-webhook-d4fd4f479-cr6z2 evicted
pod/house-gitlab-agent-v1-586d6d6797-vm68t evicted
pod/kubernetes-dashboard-844749bcff-mmpq2 evicted
pod/dashboard-metrics-scraper-66dd8bdd86-6cnbv evicted
pod/coredns-657959df74-j2zbq evicted
node/work2 evicted
```

## 3. 작업완료

- 작업이 끝난노드를 다시 클러스터로 포함시키면 됩니다.

```
kubectl uncordon work2
node/work2 already uncordoned
```

- 스케줄링 활성화 확인

```
$>  kubectl get no
NAME    STATUS   ROLES                  AGE   VERSION
masr1   Ready    control-plane,master   37d   v1.23.7
work1   Ready    <none>                 37d   v1.23.7
work2   Ready    <none>                 37d   v1.23.7
```

# 특정 노드에만 배포할 수 있는 taint 설정하기

k8s를 사용하다보면 특정 서버또는 특정환경에 매칭되는 pod만 배포할 경우가 필요합니다.

예를들면.... 디스크 타입이 ssd 장착된 노드와 hdd가 장착된 노드로 분리되어 있는 경우 disk io의 성능을 끌어올리려면 hdd보다는 ssd가 낫겠죠.

그렇게 특정 노드에 속성을 걸고 pod에 배포하기 위해서는 taint  + Tolerations 이라는 기능을 사용하면 됩니다.

taint는 노드의 속성값으로 추가하는 것이고, toleration은 적용된 taint에 어떻게 반영할것인지를 정의하는 겁니다.

> taint = node2의 type은 ssd이야
>
> toleration = type 이 ssd로 정의된 노드에 배포해줘.

1. 스케쥴 정보

| 스케쥴 정보 | 용 도 |
|---|---|
| NoSchedule | 앞으로 배포될 리소스에 대해서 적용 (현재 배포된 POD는 현상유지) |
| PreferNoSchedule | 앞으로 배포될 리소스에 대해서 적용하지만 어쩔수 없을때는 무시하고 배포<br>(현재 배포된 POD는 현상유지) |
| NoExecute | 현재 배포된 리소스와 앞으로 배포될 리소스 모두 정책에 맞지 않으면 모두 제거 |

2. 노드에 taint 적용하기
   - taint 정보 확인

```
$> kubectl get nodes -o custom-columns=node:.metadata.name,taint:.spec.taints
node          taint
controlplane   [map[effect:NoSchedule key:node-role.kubernetes.io/control-plane]]
node01        <none>
```

   taint 설정 없는거 확인하였쥬?
   - taint 설정해봅시다.

```
$> kubectl taint node node01 disk=ssd:NoSchedule
node/node01 tainted
```

   - 이제  taint 설정 한거 확인해봅시다.

```
$> kubectl get nodes -o custom-columns=node:.metadata.name,taint:.spec.taints
node          taint
controlplane   [map[effect:NoSchedule key:node-role.kubernetes.io/control-plane]]
node01        [map[effect:NoSchedule key:disk value:ssd]]
```

3. deployment에 toleration 정의하기
   1. deployment에 toleration 정의하기

```
$> vi deploy.yaml
...
spec:
  containers:
  - image: nginx
  ...
  tolerations:
  - key: "disk"
    operator: "Equal"
```

```
    value: "ssd"
    effect: "NoSchedule"
...
```

2. 배포하기

```
$> kubectl apply -f deploy.yaml
```

4. 노드에 적용했던 taint 정보 삭제 (적용했던 정책에서 -를 추가하면 삭제된다)

```
$> kubectl taint node node01 disk=ssd:NoSchedule-
node/node01 untainted
```

5. 지연된 pod 제거 방법
   1. toleration에서 tolerationSeconds 값(초기준)을 추가하면 정의된 초 이후로 pod가 제거된다.

```
$> vi deploy.yaml
...
  tolerations:
  - key: "disk"
    operator: "Equal"
    value: "ssd"
    effect: "NoExecute"
    tolerationSeconds: 3600
...
```

# etcd 데이터 백업 및 복원방법

## 시작하는말

안녕하세요, 고니입니다.
이번에는 kubernetes에서 가장 중요한 역활을 하는 Component가 API서버와 kubernetes의 정보를 저장하고 있는 ETCD  백업/복구를 하기 위한 방법을 기술하려고 합니다.

> ⚠ etcd  3.6이후부터는 etcdctl 명령어는 deprecated 된다고 합니다. (etcdutil -> etcdutl)[1)]

## 소 개

etcd는 정족수에 따라 (n-1)/2의 노드수 만큼 일시적인 장애를 허용하지만, H/W의 장애나 네트워크의 영구적인 문제 등 단기간내에 복구가 이루어 지지 않으면 ETCD 자체에 문제가 발생할 수도 있습니다, 이럴때 주기적으로 백업(snapshot)을 받아놓았다면, 해당 백업(snapshot)을 가지고 복구를 수행 후 kubernetes를 운영할 수 있습니다.

etcd  백업(snapshot)은 다중 노드에서 수행해도 되나, 복구는 동일한 백업(snapshot)을 가지고 복구를 해야 합니다, 특히 다중 노드에서 개별 노드에 백업(snapshot)된 데이터를 복구할 경우 오히려 kubernetes의 데이터의 문제로 장애가 발생 할 수 있으니, 복구는 동일한 snaphot을 가지고 복구를 수행하셔야 합니다.

백업(snapshot)수행시 무결성 검증을 위한 해쉬값이 포함되는데, 백업(snapshot)된 데이터의 수정을 하는 경우 해쉬데이터가 깨질수 있습니다. 불가피한 경우로 해쉬데이터를 무시하고 복구를 수행이 필요한 경우 `--skip-hash-check` 옵션을 추가하여 복구를 수행하면 복구가 가능합니다.

## 작업절차

### 1. etcd스크립트 생성

```
$> cat /usr/local/bin/etcdctl.sh
#!/bin/bash
# Ansible managed
# example invocation: etcdctl.sh get --keys-only --from-key ""

etcdctl \
  --cacert /etc/ssl/etcd/ssl/ca.pem \
  --cert /etc/ssl/etcd/ssl/admin-master1.pem \
  --key /etc/ssl/etcd/ssl/admin-master1-key.pem "$@"
```

### 2. /tmp/backup 파일로 etcd snapshot 수행

```
$> ./etcdctl.sh snapshot save /tmp/backup
{"level":"info","ts":"2024-09-23T23:43:44.67136+0900","caller":"snapshot/v3_snapshot.go:65","msg":"created temporary db file","path":"/tmp/backup.part"}
{"level":"info","ts":"2024-09-23T23:43:44.672148+0900","logger":"client","caller":"v3@v3.5.10/maintenance.go:212","msg":"opened snapshot stream; downloading"}
{"level":"info","ts":"2024-09-23T23:43:44.672741+0900","caller":"snapshot/v3_snapshot.go:73","msg":"fetching snapshot","endpoint":"127.0.0.1:2379"}
{"level":"info","ts":"2024-09-23T23:43:44.71756+0900","logger":"client","caller":"v3@v3.5.10/maintenance.go:220","msg":"completed snapshot read; closing"}
{"level":"info","ts":"2024-09-23T23:43:44.730295+0900","caller":"snapshot/v3_snapshot.go:88","msg":"fetched snapshot","endpoint":"127.0.0.1:2379","size":"6.2 MB","took":"now"}
{"level":"info","ts":"2024-09-23T23:43:44.730359+0900","caller":"snapshot/v3_snapshot.go:97","msg":"saved","path":"/tmp/backup"}
```

## 3. 백업(snapshot)된 데이터 확인

```
$> ./etcdctl.sh snapshot status /tmp/backup  -w table
+----------+----------+------------+------------+
|   HASH   | REVISION | TOTAL KEYS | TOTAL SIZE |
+----------+----------+------------+------------+
| 844eca85 |  1914345 |       1100 |     6.2 MB |
+----------+----------+------------+------------+
```

## 4. 백업된 데이터 복구 방법

```
$> ./etcdctl.sh snapshot restore /tmp/backup
2024-09-23T23:48:49+09:00	info	snapshot/v3_snapshot.go:260	restoring snapshot	{"path": "/tmp/backup", "wal-dir": "default.etcd/member/wal", "data-dir": "default.etcd", "snap-dir": "default.etcd/member/snap"}
2024-09-23T23:48:49+09:00	info	membership/store.go:141 Trimming membership information from the backend...
2024-09-23T23:48:49+09:00	info	membership/cluster.go:421	added member	{"cluster-id": "cdf818194e3a8c32", "local-member-id": "0", "added-peer-id": "8e9e05c52164694d", "added-peer-peer-urls": ["http://localhost:2380"]}
2024-09-23T23:48:49+09:00	info	snapshot/v3_snapshot.go:287	restored snapshot	{"path": "/tmp/backup", "wal-dir": "default.etcd/member/wal", "data-dir": "default.etcd", "snap-dir": "default.etcd/member/snap"}
```

## 5. timer기반의 정기 백업 절차 구성

- **백업스크립트 구성**

```
$> vi /root/etcd_backup.sh

#!/bin/bash

PATH=/usr/local/bin:$PATH

BACK_DATE=$(date +%Y-%m-%d)
ORI_DATE=$(date +%Y-%m-%d -d '7days')
BACK_DIR=/tmp/backup

#백업 디렉토리 없으면 생성
if [[ ! -d $BACK_DIR ]]
then
 mkdir -p $BACK_DIR
fi

#백업 수행
/usr/local/bin/etcdctl.sh snapshot save $BACK_DIR/etcd-$BACK_DATE

#백업파일 확인 후 미생성시 에러
if [[ ! -f $BACK_DIR/etcd-$BACK_DATE ]]
  echo "Backup failed"
  exit 1
fi

#오래된 데이터 삭제
rm -f $BACK_DIR/etcd-$ORI_DATE
```

```
$> chmod +x /root/etcd_backup.sh
$> ls -l  /root/etcd_backup.sh
-rwxr-xr-x 1 root root 483  9월 23 23:55 /root/etcd_backup.sh
```

- **timer 구성 (매일 04시에 백업수행)**

```
$> cat /etc/systemd/system/etcd_backup.timer
[Unit]
```

```
Description=ETCD Backup

[Timer]
OnCalendar=*-*-* 04:00:00

[Install]
WantedBy=multi-user.target
```

- timer에 연결된 서비스 구성

```
$> cat /etc/systemd/system/etcd_backup.service
[Unit]
Description=ETCD Backup script

[Service]
Type=oneshot
ExecStart= /root/etcd_backup.sh
```

- backup timer 활성화

```
$>  systemctl daemon-reload
$> systemctl enable etcd_backup.timer --now
Created symlink /etc/systemd/system/multi-user.target.wants/etcd_backup.timer → /etc/systemd/system/etcd_backup.timer.
```

- timer 확인

```
$> systemctl status etcd_backup.timer
● etcd_backup.timer - ETCD Backup
     Loaded: loaded (/etc/systemd/system/etcd_backup.timer; enabled; vendor preset: disabled)
     Active: active (waiting) since Mon 2024-06-23 23:58:42 KST; 1s ago
      Until: Mon 2024-06-23 23:58:42 KST; 1s ago
    Trigger: Tue 2024-06-24 04:00:00 KST; 6h left
   Triggers: ● etcd_backup.service
```

# *Reference*


- *https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/#etcd-verify-snapshot-1*
- *https://etcd.io/docs/v3.5/op-guide/maintenance/*