

3. Container/Kubernetes

- [centos7 docker 安装](#)
- [container the input device is not a TTY 问题排查](#)
- [docker 与 cri-o](#)
- [Docker 容器网络配置](#)
- [k8s pod 网络配置](#)
- [kubernetes-Istio](#)
- [podman 容器管理](#)
- [Podman DNS 配置](#)
- [podman 容器](#)
- [podman 容器网络配置](#)
- [etcd 成员](#)
- [kubeadm 安装](#)
- [etcd 成员](#)
- [容器网络 unknown server host 问题排查](#)
- [pod 容器 Imageinspecterror or readlink invalid argument error 问题排查](#)
- [容器网络](#)
- [容器网络 taint 配置](#)

centos7에 docker 설치

1. yum 패키지 설치

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

2. docker 서비스 시작 및 설정

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

3. Docker 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
```

container의 input device is not a TTY

container의 환경 변수 export의 환경 변수 crontab의 환경 변수,

환경 변수 export의 환경 변수 0k.

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

환경 변수 mail의 환경 변수. 환경 변수..

```
$> 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
...
```

환경 변수 docker exec의 input + terminal 환경 변수 terminal의 환경 변수.

```
$> 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
```

환경 변수

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

환경 변수

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

환경 변수..

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

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

docker & cri-o

- 1. Container >>>> Container >>> >>>> Docker
- 2. Google / Redhat / MS / IBM >> Container >> >>>> OCI(Open Container Initiative)
- 3. k8s >> >>>> CRI(Container Runtime Interface)
- 4. RHEL 8, k8s 1.20, Openstack 16, Openshift 4, aws 18 >>>> podman

- 1. low-level runtime
 - 1. >>>> cpu >>>> >>>> (ex. runC, Container, Docker)
- 2. high-level runtime
 - 1. >>>> API
 - 2. low-level >>>> runtime (ex. containerd, docker, cri-o)

CRI-O

- 1. cri-o >>>> Component
- 2. fork/exec
- 3. Overlayfs, devicemapper, btrfs, nfs, glusterfs, cephfs
- 4. docker schema v1/2

CRI-O

- 1. podman : Container Tool
- 2. Buildah : Container
- 3. skopeo : Container Tool

crictl & podman

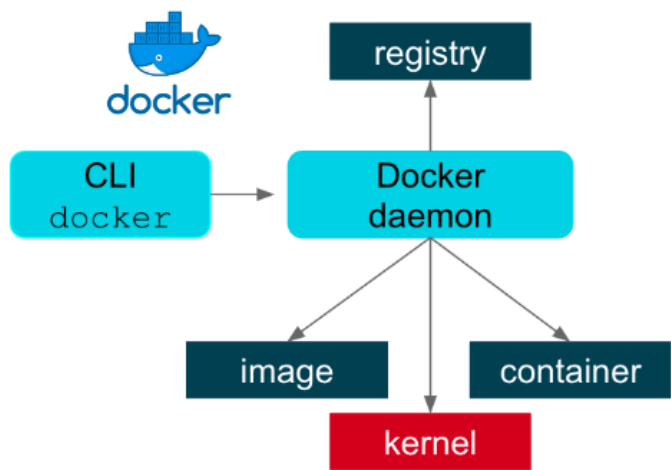
- 1. crio >>>> cli
- 2. podman >>>> cli, docker >>>> cli
- 3. cri-o >>>> podman
- 4. Docker >>>> podman

- 1. fork/exec

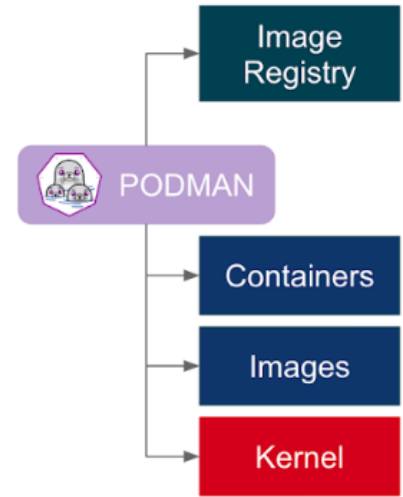


[그림 6] fork·exec 모델 및 클라이언트-서버 모델에서의 UID, auid 설정 동작 방식

- 2. podman >>>> Docker



VS



3. 对比 Docker 和 Podman

	Docker	Podman
运行模式	Docker daemon 守护进程	Daemon-less (systemd 管理 podman 进程)
用户权限	需要 root 权限运行	root-less (1024 个普通用户可运行 root 权限)
操作方式	通过 Docker API	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

Docker安装 遇到的问题 解决方法

Docker 的 rpm 和 deb 包默认安装到 /var/lib/docker 目录，但是 /var 目录默认挂载在 OS 的根目录 / 上，如果 /var 目录满了，就会导致 Docker 无法安装。解决方法如下：

```
$ 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
```

OS 根目录 / 438G 快满了，Docker 安装 420G 的空间，占用 99%...

/data 目录有 2T 500G 的空间，把 Docker 安装到 /data 目录，问题解决。

1. Docker stop

```
$ systemctl stop docker
```

2. docker 配置文件 /data/docker 目录

```
$ cat /etc/docker/daemon.json
{
  "data-root": "/data/docker"
}
```

- Docker Engine 的 daemon.json 配置文件默认在 /etc/docker 目录，需要移动到 /data/docker 目录。

3. docker 移动文件

```
$ mv -f /var/lib/docker /data/docker
```

4. docker 启动

```
$ systemctl start docker
```

5. docker 检查

```
$ docker info
...
Docker Root Dir: /data/docker
...
```

k8s pod 删除

k8s pod 删除

1. 查看 pod

```
[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. 删除 pod

```
[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. 验证 pod 删除

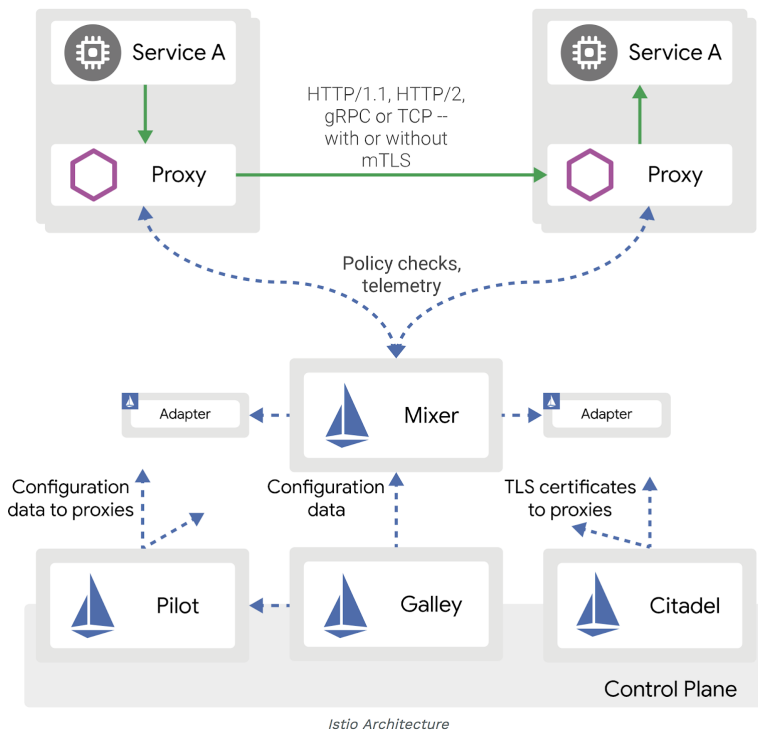
```
[root@control1 ~]# kubectl get pod test-68d6bf4d95-zhx55 -n test_user
```

Error from server (NotFound): pods "test-68d6bf4d95-zhx55" not found

kubernetes-Istio

Istio Concept & Data Flow

1. Istio Concept & Data Flow



2. Concept

1. ServiceMesh는 서비스들 간의 통신을 관리하는 것
2. http / websocket / http2 / grpc / tcp / udp / mTLS / ...

3. Component

1. Data Plane
 - Service A / B의 Pod에 Proxy(Envoy Sidecar container)를 추가
2. Control Plane
 - Mixer - 인증 / ACL / ...
 - Pilot : ingress routing, traffic mirroring, traffic shifting, canary deployments, circuit breaking, fault injection
 - Galley : yaml로 istio를 구성하고 pilot에게 전달.
 - Citadel : 인증서 관리 (TLS)를 담당, 인증서 발급을 관리하는 역할.

4. Istio 설치

- istio를 traffic shaping, k8s ingress(or service nodeport)를 통해 서비스 클러스터에 설치, istio ingressgateway port를 설정.

ServiceMesh : MSA(서비스 메쉬 아키텍처)는 서비스들 간의 통신을 관리하는 것, http / websocket / http2 / grpc / tcp / udp / mTLS / ...

Istio 설치

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

```
# Istio proxy is installed in namespace istio envoy
$ kubectl label namespace default istio-injection=enabled
namespace/default labeled
```

2. istio profile

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

3. istio

```
$ 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

1. Kiali : Istio dashboard
2. Jager / zipkin : distributed tracing
 1. zipkin : Twitter distributed tracing
 2. jaeger: Uber CNCF distributed tracing. (k8s jaeger)

5. addon

```
$ 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. 删除 istio 系统组件
1. kiali : `http://{IP}:30007`
 2. jaeger: `http://{IP}:30008`
 1. 删除

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

7. 查看 istio 组件列表

Component list

名称	类型	描述
gateway	http / tcp	网关组件，用于路由流量
virtualservice	service	虚拟服务，用于定义流量路由规则（包括权重、路由等）
virtualservice	version(a.k.a subset)	虚拟服务的版本（子集）
virtualservice	source	虚拟服务的源地址
virtualservice	host	虚拟服务的主机名
destinationRule	destinationRule	目标规则，用于定义流量路由规则（包括权重、路由等）
tcproute	tcp	tcp 路由规则
tcproute	match	tcp 路由规则的匹配条件
tcproute	route	tcp 路由规则的路由目标

8. Example yml

1. gateway 配置 (tcp/30011 端口 gateway 配置)

```
$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 (tcp/30011 (service111-1, service111-2) tcp/8080 (50% (v1, v2)))

```
$ 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 (service111 (v1, v2))

```
$ 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/>

podman 容器 管理 工具

insecure registry 配置

1. podman 配置 insecure registry 配置

```
$> 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
```

2. daemonless 配置 容器 管理 工具

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

- Docker 容器 管理 工具 默认 配置 为 80 端口，podman 容器 管理 工具 默认 配置 为 443 端口，默认 配置 为 443 端口，容器 管理 工具 默认 配置 为 443 端口

容器 管理 工具

1. podman 配置 容器 管理 工具 配置

```
$> vi /etc/containers/storage.conf
...
runroot = "/service/containers/storage"
graphroot = "/service/containers/storage"
...
```

Podman 安装 DNS

podman 安装 使用 cni 网络模式, 安装 使用 DNS 网络模式 使用 DNS 网络模式 使用 DNS 网络模式.

1. docker 安装 使用 dns 网络模式

```
$> 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 安装 使用 dns 网络模式

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

安装

1. cni 安装 使用 NetworkManager 网络

2. dns 安装

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

3. NetworkManager 安装 使用 网络 网络

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

4. NetworkManager 安装 使用 网络 网络

```
$> ls -l /usr/libexec/cni/
lrwxr-xr-x 1 root root 3980624 2020 09:00 dnsmasq
...
```

5. cni 安装 使用

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

```
}
```

6. 检查 Redis 是否可用
7. 检查 Redis 是否可用

```
$> 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
```

podman

1. docker 安装与配置

1. podman-docker ㄱㄱㄱㄱ docker ㄱㄱㄱㄱ ㄱㄱ docker ㄱㄱㄱㄱ ㄱㄱ ㄱ ㄱㄱㄱㄱ.
2. ㄱㄱ ㄱㄱㄱㄱ ㄱㄱㄱㄱ ㄱㄱㄱㄱㄱㄱ podman ㄱㄱㄱㄱ ㄱㄱㄱ.

```
$> yum install podman-docker -y
#####
   000  0000  0000  1:40:24  000:  20220  020  070  (0)  00  120  210  090.
   0000  00000000.
=====
   000  00  00                000000  00
=====
00 0:
podman-docker
    noarch 3.3.1-9.0.1.module+el8.5.0+20416+d687fed7 ol8_appstream 56 k

00 00
=====
00 1 000

00 0000 00: 56 k
000 00 : 230
000 00000:
podman-docker-3.3.1-9.0.1.module+el8.5.0+20416+d 288 kB/s | 56 kB    00:00
-----
00                                284 kB/s | 56 kB    00:00
00 00 00 0
00 000 0000000.
00 00 00 0
00 000 0000000.
00 00 0
00 0      :                      1/1
00 0      : podman-docker-3.3.1-9.0.1.module+el8.5.0+20416+d687f 1/1
000000 00 0: 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.

00 0      : podman-docker-3.3.1-9.0.1.module+el8.5.0+20416+d687f 1/1

00000000:
podman-docker-3.3.1-9.0.1.module+el8.5.0+20416+d687fed7.noarch

00000000!
```

```
# yum module install container-tools -y
000 0000 0000 0:09:24 000: 2022 02 07 ( ) 00 03 57 13.
0000 00000000.

=====
000      00 00                00000  00
=====

00/00 000 00:

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

00 000 00 00:

python3-pytml noarch 0.1.14-5.git7dea353.el8                ol8_appstream 25 k
python3-pyxdg noarch 0.25-16.el8                        ol8_appstream 94 k

00 0000 00:

container-tools/common
```

□ □ □ □

□□ 5 □□□


```

□□ □□□□ □□: 510 k
□□□ □□ : 1.6 M
□□□ □□□□□:
(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-pytml-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
-----
□□ 133 kB/s | 510 kB   00:03
□□ □□ □□ □
□□ □□□ □□□□□□.
□□ □□ □□ □
□□ □□□ □□□□□□.
□□ □□ □
□□ □ : 1/1
□□ □ : python3-pyxdg-0.25-16.el8.noarch 1/5
□□ □ : python3-pytml-0.1.14-5.git7dea353.el8.noarch 2/5
□□ □ : python3-podman-3.2.0-2.module+el8.5.0+20416+d687fed7 3/5
□□ □ : udica-0.2.5-2.module+el8.5.0+20416+d687fed7.noarch 4/5
□□ □ : crun-1.0-1.module+el8.5.0+20416+d687fed7.x86_64 5/5
□□□□□ □□ □: crun-1.0-1.module+el8.5.0+20416+d687fed7.x86_64 5/5
□□ □ : crun-1.0-1.module+el8.5.0+20416+d687fed7.x86_64 1/5
□□ □ : python3-podman-3.2.0-2.module+el8.5.0+20416+d687fed7 2/5
□□ □ : python3-pytml-0.1.14-5.git7dea353.el8.noarch 3/5
□□ □ : python3-pyxdg-0.25-16.el8.noarch 4/5
□□ □ : udica-0.2.5-2.module+el8.5.0+20416+d687fed7.noarch 5/5

□□□□□□□:
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-pytml-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

```

□□□□□□□!

3. docker□ socket api□ □□□□ □□□ podman-docker □□□□ □□□□ /var/run/docker.sock□
/var/run/podman/podman.sock□ □□□ □□□□ □□□ docker-py, docker-compose□ □□□ docker api□ □□□ □□□ □ □□□□.
4. podman□□□ □□□□ □□ docker □□□ □□□□, □□, □□□□, □□□□, □□□, □□□, □□, docker swarm□ □□□□□.

2. Rootless□□

1. □□□ □□□□ □□□□ □□□ □□□ □□ □□□□□. (root - /var/lib/containers/storage, non root - \$HOME/.local/share/containers/storage)
2. rootless □□□□□ 1024□□□□ □□□ □□□ □ □□□□. (tcp/80□ □□□□□ apache□ □□ □□□□□□□□ tcp/80□ □□□□□□ □□ □□□□ □□ □□□ □□)
3. 1024 □□ □□□ □□□□□ □□□ □□□ □□ □□□ □□ □□□, □□□□ □□□□□ □□□□□□□ □□□□ □□□□. (□□□ □□)

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

4. 8.1□□ rootless Containers□□□□ □□□□□□□ □□□□ □□□ □ □□□□. (rootless□ □□□□)

```

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

1. registries.conf 파일을 편집하여 container-registry.oracle.com, docker.io, container.test.com을 추가합니다.

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

2. test.com에 TLS 없이 pull합니다.

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

3. test.com에 접근을 차단합니다.

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

4. skopeo

1. skopeo를 사용하여 docker.io/library/httpd:latest의 Digest를 확인합니다.
2. httpd container의 Digest를 확인합니다.

```
# 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:cdb155854ae6a9e25834459a7c9dfc7be157a2ebfca5adbbdb036aeea43ce3128",
  "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="
```

```
]
```

podman 容器 管理 工具 介紹

podman 容器 管理 工具 介紹 文章 內容 如下。

安裝

```
$> 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
```

容器 管理 工具 介紹 文章 內容 如下 state 容器 (Up) 容器 管理 (created) 容器 管理

```
$> 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
```

容器

1. podman/runc 容器 管理

容器 (容器 管理 工具 介紹 文章 內容 1 容器 管理 2, 2 容器 管理 3 容器 管理)

1. podman 容器 管理

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

2. podman 容器 管理 security 容器 管理

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

3. runtime 容器 管理

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

容器

```
$> 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
```

etcd member

k8s DB etcd 成员列表

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 (节点 状态 列表)

```
$> /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 실행

이제 이 단계를 실행합니다

1. 토큰 생성 토큰 생성 token 생성

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

2. join 명령어 실행 : 토큰 생성 kubeadm / kubelet 실행 provisioning 실행

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

etcd member 增加 删除

kubernetes DB 的 etcd 的 member 增加 删除 操作。

增加 删除(3个 节点)的 操作 的 命令 如下。

1. etcd memver list 查看

```
$> 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. 删除 节点 ID

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

3. 增加 节点

```
$> 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	

unknown server host 問題の解決方法

問題の概要
アプリケーションがデータベース（WEB ->DB）に接続できない

原因
Docker コンテナの hosts ファイルが docker-dns.conf に正しく設定されていない。

解決方法
docker-dns ファイルを修正する。

```
$> 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 出现 Imageinspecterror or readlink invalid argument error 怎么办

怎么办

1. pod 出现 Imageinspecterror 怎么办 可能是镜像 没拉取

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

2. docker data root 目录 是否 在 pod 目录 下面 如果 不在 请 [参考 Docker 安装 教程](#)

怎么办

1. docker 清理 缓存

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

2. pod 重启 (auto healing 功能 是否 开启 是否 成功)

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

0000 0000

00 00

k8s0000 00 00 0000 00 0000 kube scheduling0 00 0000 000000 0000 00 0000 0000 0000

00 00000 000 000000 cordon / drain 0000 000 00000.

cordon / drain0 0000 0000 0000, cordon0 00 00 000000 00000 00, drain0 00 0000 0000 00 00000 0000 000000. (drain0 cordon0000 000000.)

00000

1. cordon 00

1. 00 00 00

```
$> 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
```

2. cordon 00 (work10000 00000000.)

```
$> 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 00

1. 00 0000 daemonset00 00000 pod0 0000 local storage0 00000 00 00 drain 0000 00000 00000.

```
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
```

2. 0000 drain 00

```
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. 节点

1. 节点 节点 节点 节点 节点.

```
kubectll uncordon work2
node/work2 already uncordoned
```

2. 节点 节点 节点

```
$> kubectll 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
```

如何 使用 节点 的 taint 功能

k8s 的 节点 的 功能 是 通过 pod 的 节点 选择 器。

节点.... 节点 的 类型 是 hdd 节点 的 类型 是 disk io 节点 的 类型 是 hdd 节点 的 类型 是 ssd 节点。

节点 的 类型 是 pod 的 类型 是 taint + Tolerations 节点 的 类型 是 pod。

taint 节点 的 类型 是 pod, toleration 节点 的 类型 是 taint 节点 的 类型 是 pod。

```
// taint = node2 type ssd
toleration = type ssd 节点 的 类型 是 pod。
```

1. 节点 的 类型

节点 的 类型	节点 的 类型
NoSchedule	节点 的 类型 是 pod (节点 的 类型 是 POD 节点)
PreferNoSchedule	节点 的 类型 是 pod 节点 的 类型 是 pod (节点 的 类型 是 POD 节点)
NoExecute	节点 的 类型 是 pod 节点 的 类型 是 pod 节点 的 类型 是 pod 节点 的 类型 是 pod

2. 节点 的 taint 节点

- 节点 的 类型 是 pod

```
$> 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 节点 的 类型 是 pod?

- 节点 的 类型 是 pod。

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

- 节点 的 类型 是 pod 节点 的 类型 是 pod。

```
$> 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. deployment 的 toleration 节点

1. deployment 的 toleration 节点

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

2. 실행

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

4. 노드 라벨 taint 추가 (노드 라벨 -에 라벨을 추가)

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

5. pod에 라벨 추가

1. toleration을 tolerationSeconds (초)로 설정하여 pod가 실행되도록.

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