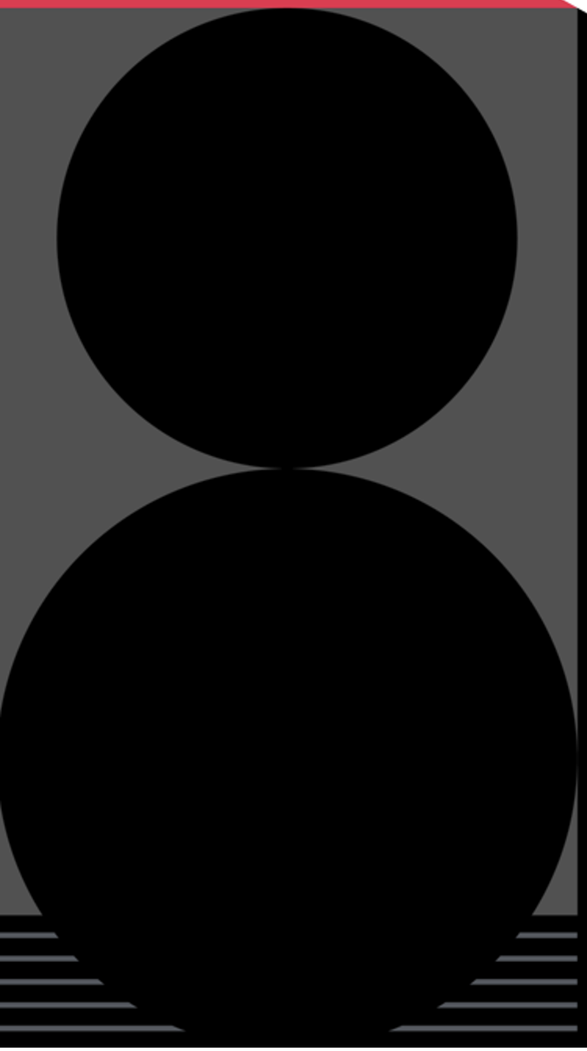


8 Docker 트러블 슈팅 방법

01 Docker 트러블 슈팅 방법 소개

Chapter 소개

Docker 트러블 슈팅 방법

1. Docker 트러블 슈팅 방법 소개
2. Docker, CRI-O Runtime 데몬 상태 확인 및 이슈 로깅 방법
3. 네트워크 정보 확인 및 트래픽 Dump/디버깅 방법
4. Nexus를 활용한 Private 환경 Docker 빌드
5. Squid를 활용한 Private 환경 Docker 빌드

Docker 트러블 슈팅 방법 소개

Docker 리소스 상태 및 이슈 확인

- Docker, CRI-O Runtime 데몬 상태 확인 및 이슈 로깅 방법

Docker 관련 네트워크 상태 및 이슈 확인

- 네트워크 정보 확인 및 트래픽 Dump/디버깅 방법

Private 환경에서 Nexus 활용

- Nexus를 활용한 Private 환경 Docker 빌드

Private 환경에서 Proxy 활용

- Squid를 활용한 Private 환경 Docker 빌드

CRI-O 소개

- 레드햇에서 주도적으로 개발한 오픈소스 런타임
- Docker보다 가볍고 최적화 된 Runtime (특히 Kubernetes)
- Kubernetes 1.20 이상 버전 부터는 CRI를 지원하는 런타임으로 CRI-O 사용
- CRI-O는 기존 도커 이미지와 호환성이 뛰어나고 사용 경험도 그대로 적용가능
- CRI-O는 Kubernetes 환경에서만 사용이 가능함 (없는 환경에선 사용 불가)

```
ubuntu@ip-172-31-7-91:~$ crictl images
IMAGE                     TAG                 IMAGE ID            SIZE
localhost:5000/test       spring-docker       06935c1c2ad7d       233MB
```

Docker Network 트래픽 Dump 방법

- tcpdump 명령어를 사용
- Docker 컨테이너가 사용하고 있는 특정 네트워크 인터페이스를 확인(리눅스)
- 해당 인터페이스에서 발생하는 트래픽 정보(패킷)등을 tcpdump로 수집
- 수집한 트래픽은 파일로 dump

```
ubuntu@ip-172-31-7-91:~$ sudo tcpdump -i docker0 -w tcpdump.pcap
tcpdump: listening on docker0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C27 packets captured
27 packets received by filter
0 packets dropped by kernel
ubuntu@ip-172-31-7-91:~$ ls
tcpdump.pcap
```

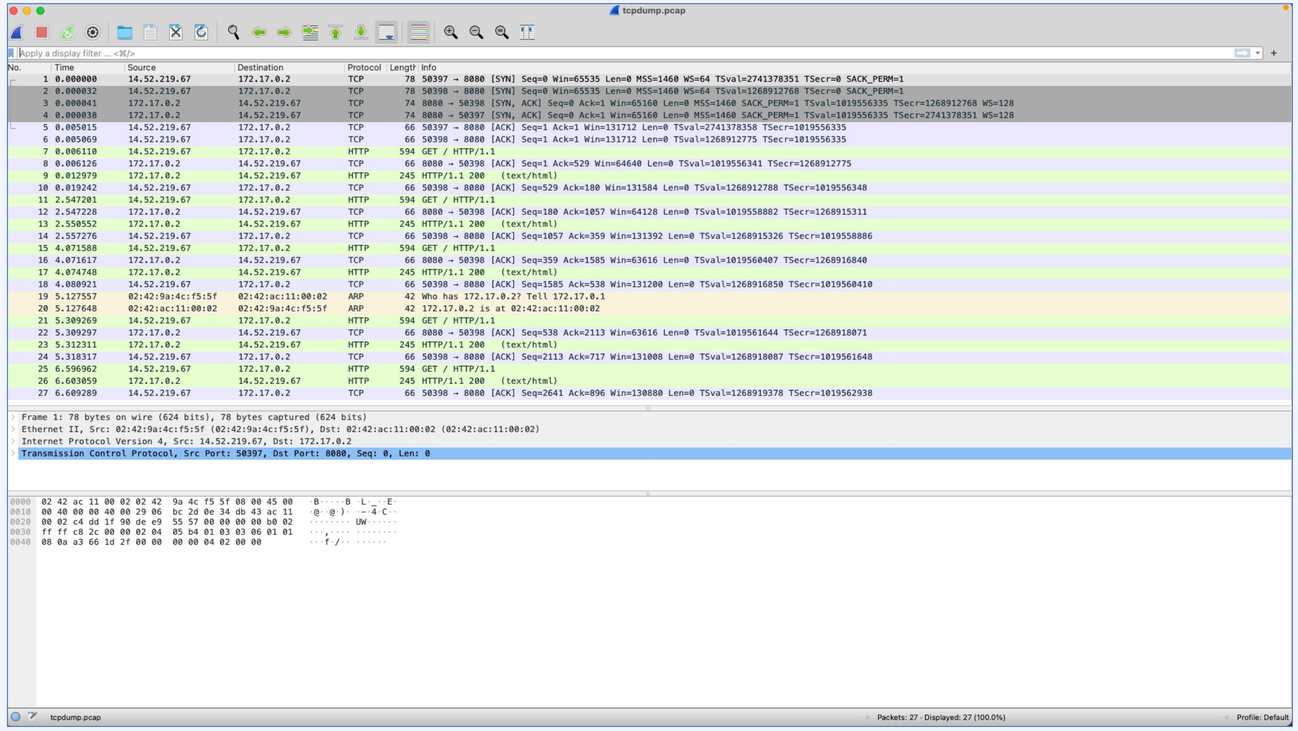
Docker 컨테이너 네트워크 인터페이스 확인 예

```
ubuntu@ip-172-31-7-91:~$ docker inspect 0f966cb7c997
[
    {
        "Id": "0f966cb7c99703e3a8f346f1d291e628fe072711e2475
        "Created": "2022-02-04T13:23:44.14833391Z",
        "Path": "/cnb/process/web",
        "Args": [],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 12508,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2022-02-04T14:29:00.280881746Z",
            "FinishedAt": "2022-02-04T14:28:20.131957433Z"
        },
        "Image": "sha256:06935c1c2ad7dfefa2d3d1b63d49ab530b8
        "ResolvConfPath": "/var/lib/docker/containers/0f966c
        "HostnamePath": "/var/lib/docker/containers/0f966cb7
        "HostsPath": "/var/lib/docker/containers/0f966cb7c99
        "LogPath": "/var/lib/docker/containers/0f966cb7c9970
e072711e247583746d14b18a7cbb42c-json.log",
        "Name": "/elated_hofstadter",
        "RestartCount": 0,
        "Driver": "overlay2",
        "Platform": "linux",
        "MountLabel": "",
        "ProcessLabel": "",
        "AppArmorProfile": "docker-default",
        "ExecIDs": null,
        "HostConfig": {
            "Binds": null,
```

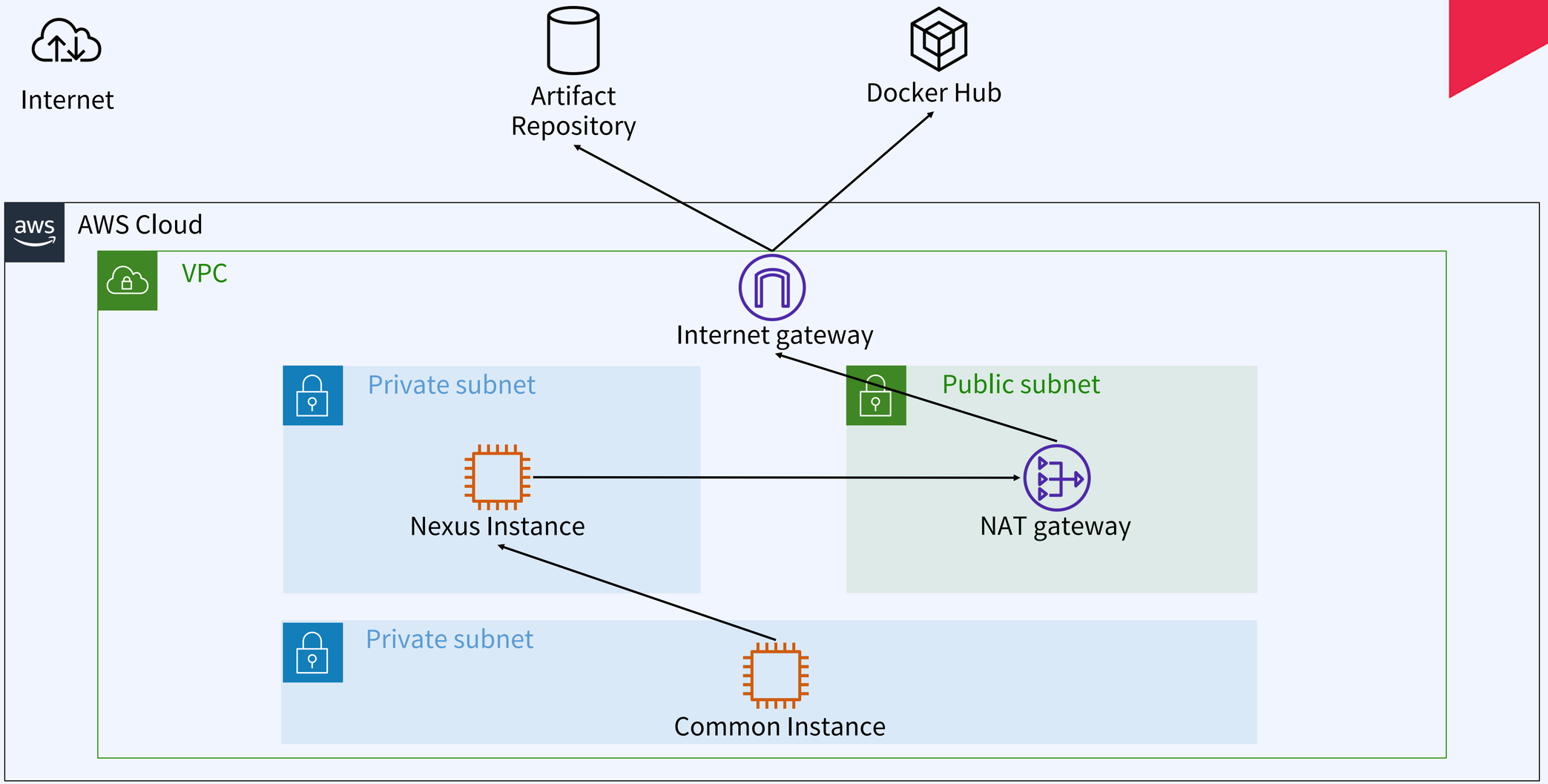
```
ubuntu@ip-172-31-7-91:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc mq state UP group default qlen 1000
    link/ether 02:8d:9a:10:22:7a brd ff:ff:ff:ff:ff:ff
    inet 172.31.7.91/20 brd 172.31.15.255 scope global dynamic ens5
       valid_lft 1894sec preferred_lft 1894sec
    inet6 fe80::8d:9aff:fe10:227a/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:9a:4c:f5:5f brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:9aff:fe4c:f55f/64 scope link
       valid_lft forever preferred_lft forever
9: veth798feee@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether 16:f8:e2:82:f5:2f brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::14f8:e2ff:fe82:f52f/64 scope link
       valid_lft forever preferred_lft forever
11: veth4d67935@if10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether 36:6d:10:10:7b:04 brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet6 fe80::346d:10ff:fe10:7b04/64 scope link
       valid_lft forever preferred_lft forever
```

Docker Network 트래픽 Debugging 방법

- tcpdump로 수집된 트래픽의 dump파일을 로컬로 가져옴
- Wireshark에 dump 파일 import
- Wireshark를 통한 Debugging 수행



Private 환경에서 Nexus 활용



Private 환경에서 Proxy 활용