HANDS-ON LABS

실습 모음 — 20 시나리오

Hands-on Labs · 20 시나리오 · 통합 핸드아웃

IT 인프라 아키텍처 설계

GenAI Labs

강사 박수현 · CEO / CTO

실습 목차 (20 개)

각 실습은 표지 → 목적/구성 → 본문 → 부록(설정 파일·setup.sh) 순서로 구성됩니다.

HANDS-ON LAB

실습 #1

VirtualBox VM 만들고 SSH 접속 (스냅샷·네트워크·포트포워딩)

[필수]

IT 인프라 아키텍처 설계

Hands-on Labs · 20 시나리오

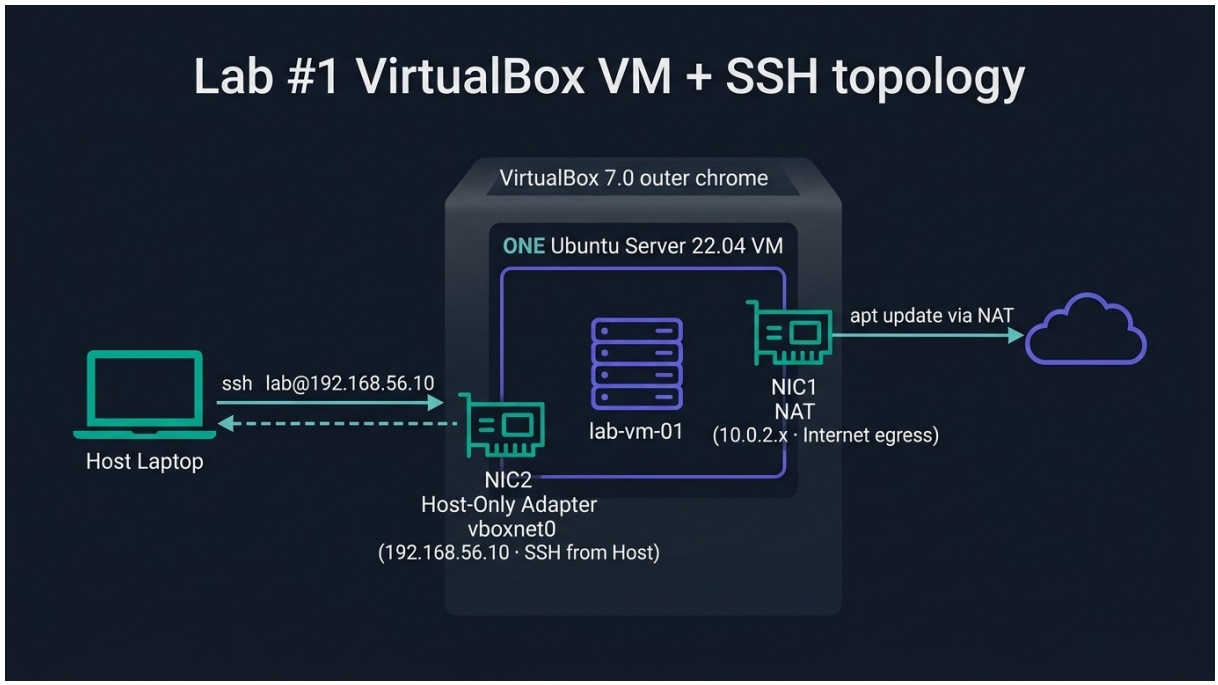
GenAI Labs

강사 박수현 · CEO / CTO

실습 목적

- **VirtualBox 7.0** 에 Ubuntu Server 22.04 VM 1 대 구축
- **NAT + Host-Only Adapter** 듀얼 NIC 구성 (인터넷 출구 + 호스트 SSH 입구)
- **스냅샷** 활용 (실패 시 5 초 만에 복구)
- **포트 포워딩** (NAT 모드에서 외부 → VM 서비스 노출)
- 호스트 ↔ VM SSH 접속 정착

아키텍처 구성 (Topology)



위 토폴로지를 머리에 그린 상태로 아래 실습을 진행한다.

도구

- VirtualBox 7.0 (virtualbox.org)
- Ubuntu Server 22.04 LTS ISO (releases.ubuntu.com/22.04/)
- 호스트 SSH 클라이언트 (OpenSSH for Windows · macOS Terminal · Linux ssh)

기본 예제 — VM 1 대 만들기

1. VM 생성

```
VirtualBox Manager → "New"
├── 이름: lab-vm-01
├── Type: Linux
├── Version: Ubuntu (64-bit)
├── 메모리: 4096 MB (호스트 16GB 기준)
└── 디스크: 동적 할당 40GB
```

2. 네트워크 설정 (듀얼 NIC)

```
설정 → Network
├── Adapter 1: NAT                    ← 인터넷 출구 (apt update 용)
└── Adapter 2: Host-Only Adapter
    └── Name: vboxnet0                ← 호스트 ↔ VM SSH 채널
```

호스트에서 vboxnet0 IP 확인:

```
# Linux/macOS
ifconfig vboxnet0       # 보통 192.168.56.1

# Windows
ipconfig /all | findstr /i "host-only"
```

3. Ubuntu 설치

- 사용자: `lab` / 비밀번호 강력하게
- OpenSSH server **체크** (Featured server snaps)
- Static IP 권장: `192.168.56.10/24` (Host-Only 인터페이스)

4. 첫 부팅 후 Host-Only IP 설정 (netplan)

```
# /etc/netplan/00-installer-config.yaml
network:
  version: 2
  ethernets:
    enp0s3:                 # NAT
      dhcp4: true
    enp0s8:                 # Host-Only
```

```
      dhcp4: false
      addresses: [192.168.56.10/24]

sudo netplan apply
ip -br addr                 # IP 확인
```

5. 호스트 → VM SSH 접속

```
ssh lab@192.168.56.10       # 비밀번호 입력 → 접속 성공
```

응용 예제 ① — 스냅샷 (운영 안전벨트)

GUI 에서:

```
VirtualBox Manager → VM 선택 → "Snapshots" 탭
└── "Take" → 이름: clean-install
```

CLI:

```
# 스냅샷 생성
VBoxManage snapshot lab-vm-01 take "clean-install" --description "OS 설치 직후"

# 목록
VBoxManage snapshot lab-vm-01 list

# 복원
VBoxManage controlvm lab-vm-01 poweroff
VBoxManage snapshot lab-vm-01 restore "clean-install"
VBoxManage startvm lab-vm-01 --type headless

# 삭제 (디스크 회수)
VBoxManage snapshot lab-vm-01 delete "old-snapshot"
```

운영 룰: 위험한 작업 (패키지 업그레이드·설정 변경) **전에 무조건 스냅샷**.

응용 예제 ② — 포트 포워딩 (NAT 모드)

NAT 는 외부에서 직접 접근 불가 → 포트 포워딩으로 호스트→NAT→VM 매핑.

```
# 호스트:2222 → VM:22 (SSH)
VBoxManage modifyvm lab-vm-01 --natpf1 "ssh,tcp,,2222,,22"

# 호스트:8080 → VM:80 (HTTP)
VBoxManage modifyvm lab-vm-01 --natpf1 "http,tcp,,8080,,80"

# 룰 목록
VBoxManage showvminfo lab-vm-01 | grep "NIC 1 Rule"

# 사용
```

```
ssh lab@127.0.0.1 -p 2222
curl http://127.0.0.1:8080
```

응용 예제 ③ — Headless 모드 (서버 운영)

```
# GUI 없이 시작
VBoxManage startvm lab-vm-01 --type headless

# 종료
VBoxManage controlvm lab-vm-01 acpipowerbutton    # graceful
VBoxManage controlvm lab-vm-01 poweroff           # force

# 상태
VBoxManage list runningvms
```

응용 예제 ④ — Clone (복제로 다중 VM)

```
# 빠른 복제 (linked clone — 디스크 절약)
VBoxManage clonevm lab-vm-01 --name lab-vm-02 --register --options=link

# 완전 복제
VBoxManage clonevm lab-vm-01 --name lab-vm-prod --register

# MAC 재생성 (네트워크 충돌 방지)
VBoxManage modifyvm lab-vm-02 --macaddress1 auto
```

응용 예제 ⑤ — VirtualBox 네트워크 5 종 비교

모드	VM → Internet	Host → VM	VM ↔ VM	격리
NAT		(포트포워딩 필요)		강함
NAT Network				중간
Host-Only				강함
Bridged	(호스트 NIC 공유)	(실제 LAN)		없음
Internal Network				최강

조합 표준:

- NAT + Host-Only → 인터넷 + SSH (가장 흔함, 본 실습)
- Internal + Host-Only → 클러스터 격리 (lab07·11·18 에서 사용)
- Bridged → VM 이 LAN 의 일원처럼 (운영 환경 mimicking)

관전 포인트

1. **NAT + Host-Only 듀얼** = 인터넷 출구 + 호스트 SSH 입구 = 운영 표준
2. **스냅샷**으로 실습 실패 시 5 초 복원
3. **포트 포워딩**은 NAT 에서 외부 접근의 유일한 방법
4. **Headless 모드**로 호스트 CPU·메모리 절약
5. **Linked clone** 으로 디스크 절약 다중 VM

자주 만나는 오류

- VT-x/AMD-V 비활성 → BIOS 에서 가상화 기능 켜기
- 게스트 추가 안 깔림 → `sudo apt install virtualbox-guest-utils`
- Host-Only Adapter 없음 → File → Host Network Manager → Create
- SSH 안 됨 → `sudo systemctl status ssh` + UFW 점검
- VBoxManage 못 찾음 → `C:\Program Files\Oracle\VirtualBox\` PATH 추가

운영 팁

- **스냅샷 명명 규칙**: `YYYYMMDD_상태` (예: `20260601_clean`)
- **Vagrant** 로 코드화 (`vagrant up`)
- **메모리 풀**: 호스트 RAM 의 50% 이하 합계로 다중 VM
- **CPU**: 호스트 코어 수 - 1 이하 합계
- **백업**: `VBoxManage export lab-vm-01 -o lab-vm-01.ova` (OVF 표준)

부록 — setup.sh

본 실습의 명령어 통합 — `chmod +x setup.sh && ./setup.sh` 로 실행 가능.

```
#!/bin/bash
# 실습 #1 — VirtualBox VM + SSH 접속   [필수]
# 실행 시 sudo 권한 필요
set -e

# 1) VirtualBox UI 에서 VM 생성
#    Name: lab-vm-01 / Type: Linux / Ubuntu 64-bit
#    Memory: 4096 MB / CPU: 2 / Disk: 40 GB VDI

# 2) Settings → Network
#    Adapter 1: NAT       (외부 인터넷)
#    Adapter 2: Host-Only (vboxnet0, 192.168.56.0/24)

# 3) Ubuntu 22.04 ISO 마운트 + 설치
#    User: lab / Hostname: lab-vm-01
#    "Install OpenSSH server" 체크

# 4) VM 안에서 (콘솔 접속)
sudo systemctl enable --now ssh
sudo ufw allow 22
ip addr show enp0s8     # 192.168.56.10 확인

# 5) 호스트에서 SSH 접속
ssh lab@192.168.56.10
uname -a
sudo systemctl status ssh
```

HANDS-ON LAB

실습 #2

Nginx Reverse Proxy + 라우팅 매칭 + LB 패턴

[필수]

IT 인프라 아키텍처 설계

Hands-on Labs · 20 시나리오

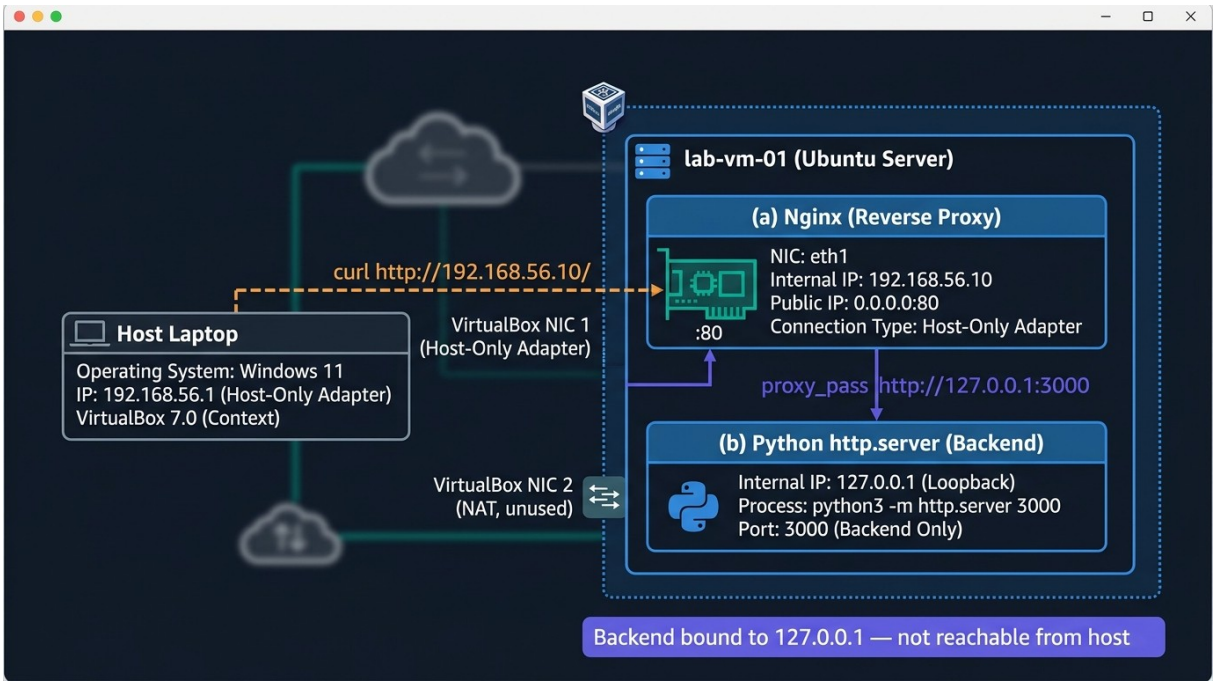
GenAI Labs

강사 박수현 · CEO / CTO

실습 목적

- **Reverse Proxy** 동작 원리 손에 잡기
- Nginx 설정 파일 (`proxy_pass`·헤더 전달) 작성
- **location 매칭 5 종** 패턴 (=, ~, ~*, ^~, prefix) 차이 이해
- **upstream 로드 밸런서** 한 줄 확장
- 백엔드 1 대 → 다중 → 헬스체크까지 자연스러운 확장

아키텍처 구성 (Topology)



위 토폴로지를 머리에 그린 상태로 아래 실습을 진행한다.

도구

- VirtualBox + Ubuntu Server 22.04 (실습 #1 에서 만든 VM)
- **Nginx** (`apt install nginx`)
- **Python http.server** (백엔드 mock)
- `curl` (검증)

시나리오 (약 30 분)

```
# 1) Nginx 설치 + 시작
sudo apt update && sudo apt install -y nginx
sudo systemctl status nginx     # active (running) 확인

# 2) 백엔드 띄우기 (다른 터미널)
mkdir -p /tmp/site && cd /tmp/site
echo '<h1>Backend OK</h1>' > index.html
python3 -m http.server 3000 --bind 127.0.0.1

# 3) 백엔드 두 번째 (응용 예제용)
mkdir -p /tmp/site2 && cd /tmp/site2
echo '<h1>Backend 2 — API</h1>' > index.html
python3 -m http.server 3001 --bind 127.0.0.1
```

기본 예제 — Reverse Proxy 1 대

가장 단순한 reverse proxy

```
# /etc/nginx/sites-available/lab-basic
server {
    listen 80;
    server_name _;
    location / {
        proxy_pass http://127.0.0.1:3000;
    }
    access_log /var/log/nginx/lab-basic.access.log;
}

# 활성화
sudo ln -s /etc/nginx/sites-available/lab-basic /etc/nginx/sites-enabled/
sudo rm -f /etc/nginx/sites-enabled/default
sudo nginx -t                       # syntax 검증 (운영 필수 습관)
sudo systemctl reload nginx
curl -v http://localhost/           # 200 OK + Backend OK
```

헤더 전달 — 백엔드가 실제 클라이언트를 알도록

```
location / {
    proxy_pass http://127.0.0.1:3000;
    proxy_set_header Host $host;
```

```
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
}
```

헤더	의미	백엔드 활용
`Host`	원본 Host 헤더	다중 도메인 라우팅 (vhost)
`X-Real-IP`	클라이언트 실제 IP	로그·통계·차단
`X-Forwarded-For`	프록시 체인 IP	LB 다단 추적
`X-Forwarded-Proto`	원본 스킴 (http/https)	백엔드 리다이렉트 결정

응용 예제 ① — location 매칭 5 종

Nginx 는 `location` 블록을 다음 5 가지 매칭 모드로 처리한다 — **순서가 중요**.

우선순위	모드	구문	의미
1	정확 일치	`location = /path`	URI 가 정확히 `/path` 일 때만
2	우선 접두사	`location ^~ /path`	접두사 일치 시 정규식 무시
3	정규식 (대소문자 구분)	`location ~ ^/api/`	regex 매칭
4	정규식 (대소문자 무시)	`location ~* \.(jpg\	png)$`
5	일반 접두사	`location /`	가장 긴 prefix 가 이김

종합 예제 — 라우팅 패턴 매핑

```
server {
    listen 80;
    server_name api.lab.local;

    # 1. 정확 일치 — 헬스체크 endpoint
    location = /healthz {
        access_log off;
        return 200 "OK\n";
    }

    # 2. 우선 접두사 — 정적 자원 (regex 평가 스킵 → 성능)
    location ^~ /static/ {
        alias /var/www/static/;
        expires 7d;
    }

    # 3. 정규식 매칭 — 이미지 캐시 (대소문자 무시)
    location ~* \.(jpg|jpeg|png|gif|webp|svg)$ {
        expires 30d;
        add_header Cache-Control "public, immutable";
```

```
    }

    # 4. 정규식 — API 버전 캡처
    location ~ ^/api/(v\d+)/(.*)$ {
        proxy_pass http://127.0.0.1:3000/$1/$2;
        proxy_set_header X-API-Version $1;
    }

    # 5. 일반 접두사 — 관리자 (가장 긴 prefix)
    location /admin/ {
        auth_basic "Restricted";
        auth_basic_user_file /etc/nginx/.htpasswd;
        proxy_pass http://127.0.0.1:3001/;
    }

    # 6. 기본 (catch-all)
    location / {
        proxy_pass http://127.0.0.1:3000;
    }
}
```

매칭 우선순위 검증

```
curl http://api.lab.local/healthz              # → 200 "OK"  (정확 일치)
curl http://api.lab.local/static/logo.png      # → 정적 파일 (^~ 우선)
curl http://api.lab.local/img/banner.JPG       # → 30 일 캐시 (~* 정규식)
curl http://api.lab.local/api/v1/users         # → 백엔드 :3000 (~ 정규식 + capture)
curl http://api.lab.local/admin/dashboard      # → 백엔드 :3001 + Basic Auth
curl http://api.lab.local/anything             # → 기본 백엔드 :3000
```

proxy_pass 슬래시의 함정

```
# (A) URI 그대로 전달
location /api/ {
    proxy_pass http://backend;       # 슬래시 없음 → /api/users → /api/users
}

# (B) URI rewrite (prefix 제거)
location /api/ {
    proxy_pass http://backend/;      # 슬래시 있음 → /api/users → /users
}
```

암기 룰: `proxy_pass` 끝에 슬래시 → prefix 제거, 슬래시 없음 → prefix 유지.

응용 예제 ② — upstream 로드 밸런서

`proxy_pass` 를 **upstream** 으로 바꾸면 즉시 LB 가 된다.

라운드로빈 (기본)

```
upstream backend_pool {
    server 127.0.0.1:3000;
    server 127.0.0.1:3001;
}

server {
    listen 80;
    location / {
        proxy_pass http://backend_pool;
    }
}

# 두 번째 백엔드 시작 (앞 시나리오에서 띄운 :3001 사용)
for i in {1..6}; do curl -s http://localhost/; done
# Backend OK / Backend 2 — API / Backend OK / Backend 2 — API ... 순환
```

가중치 + 헬스체크

```
upstream backend_pool {
    server 127.0.0.1:3000 weight=3 max_fails=3 fail_timeout=10s;   # 3배 트래픽
    server 127.0.0.1:3001 weight=1 max_fails=3 fail_timeout=10s;   # 1배
    server 127.0.0.1:3002 backup;                                   # 둘 다 죽으면만 사
용
}
```

IP 해시 (sticky session)

```
upstream backend_pool {
    ip_hash;
    server 127.0.0.1:3000;
    server 127.0.0.1:3001;
}
```

Least Connections (가장 부담 적은 서버)

```
upstream backend_pool {
    least_conn;
    server 127.0.0.1:3000;
    server 127.0.0.1:3001;
}
```

응용 예제 ③ — TLS termination + HTTP→HTTPS 리다이렉트

```
# 80번은 무조건 443으로 리다이렉트
server {
    listen 80;
    server_name api.lab.local;
    return 301 https://$host$request_uri;
}

# 443에서 TLS 종료 + 백엔드 평문
```

```
server {
    listen 443 ssl http2;
    server_name api.lab.local;

    ssl_certificate     /etc/nginx/certs/api.lab.local.crt;
    ssl_certificate_key /etc/nginx/certs/api.lab.local.key;
    ssl_protocols       TLSv1.2 TLSv1.3;
    ssl_ciphers         HIGH:!aNULL:!MD5;
    ssl_session_cache   shared:SSL:10m;

    location / {
        proxy_pass http://127.0.0.1:3000;
        proxy_set_header X-Forwarded-Proto https;
    }
}
```

자가 서명 인증서 빠른 생성:

```
sudo mkdir -p /etc/nginx/certs
sudo openssl req -x509 -newkey rsa:2048 -nodes -days 365 \
  -keyout /etc/nginx/certs/api.lab.local.key \
  -out /etc/nginx/certs/api.lab.local.crt \
  -subj "/CN=api.lab.local"
```

응용 예제 ④ — 캐싱 (proxy_cache)

```
proxy_cache_path /var/cache/nginx levels=1:2 keys_zone=api_cache:10m
                 max_size=1g inactive=60m use_temp_path=off;

server {
    listen 80;
    location /api/ {
        proxy_cache api_cache;
        proxy_cache_valid 200 5m;
        proxy_cache_valid 404 1m;
        proxy_cache_use_stale error timeout updating http_500 http_502;
        proxy_cache_lock on;
        add_header X-Cache-Status $upstream_cache_status;
        proxy_pass http://127.0.0.1:3000;
    }
}

curl -I http://localhost/api/users    # X-Cache-Status: MISS
curl -I http://localhost/api/users    # X-Cache-Status: HIT
```

관전 포인트

1. `proxy_pass` + 헤더 4 종 = 운영 Nginx 기본 패턴
2. `location` **우선순위** — 정확 일치(=) > 우선 접두사(^~) > 정규식(~ ~*) > 일반 접두사
3. `upstream` 한 줄로 즉시 LB — `weight`·`max_fails`·`backup` 은 무중단 운영의 핵심

4. `nginx -t` reload 전 항상 — 운영 사고의 70% 예방
5. `access_log` + `error_log` 두 로그 같이 보기 — 502 디버깅의 시작

자주 만나는 오류

- `nginx: [emerg] bind() to 0.0.0.0:80 failed` → `sudo ss -tlnp | grep :80` 다른 프로세스 종료
- `502 Bad Gateway` → 백엔드 다운 또는 타임아웃 (`curl http://127.0.0.1:3000` 확인)
- `403 Forbidden` (정적 파일) → 디렉터리 권한 또는 SELinux (`ls -lZ /var/www/`)
- `nginx: [emerg] location ... directive is not allowed here` → location 은 server 블록 안에만
- `proxy_pass` 슬래시 함정 → URI 더블 prefix 또는 prefix 누락
- `worker_connections are not enough` → `worker_connections 4096` 또는 `16384`

운영 팁

- **무중단 reload**: `sudo systemctl reload nginx` (`restart` 아님)
- **로그 회전**: `/etc/logrotate.d/nginx` 기본 14 일 → 운영 30~90 일
- **rate limit**: `limit_req_zone $binary_remote_addr zone=api:10m rate=10r/s;`
- **상태 모니터**: `--with-http_stub_status_module` + `location /nginx_status`
- **튜닝**: `worker_processes auto` + `worker_connections 16384` + `keepalive_timeout 65`

부록 — 설정 파일

01_lab-proxy

```
server {
    listen 80;
    server_name _;
    location / {
        proxy_pass http://127.0.0.1:3000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
    access_log /var/log/nginx/lab-proxy.access.log;
}
```

02_lab-location-match

```
## 응용 예제 ① — location 매칭 5 종 (=, ^~, ~, ~*, prefix)
## 매칭 우선순위: 정확 일치(=) > 우선 접두사(^~) > 정규식(~/~*) > 일반 접두사
server {
    listen 80;
    server_name api.lab.local;

    # 1. 정확 일치 — 헬스체크
    location = /healthz {
        access_log off;
        return 200 "OK\n";
    }

    # 2. 우선 접두사 — 정적 자원 (regex 평가 스킵 → 성능)
    location ^~ /static/ {
        alias /var/www/static/;
        expires 7d;
    }

    # 3. 정규식 (대소문자 무시) — 이미지 캐시
    location ~* \.(jpg|jpeg|png|gif|webp|svg)$ {
        expires 30d;
        add_header Cache-Control "public, immutable";
    }

    # 4. 정규식 — API 버전 캡처
    location ~ ^/api/(v\d+)/(.*)$ {
        proxy_pass http://127.0.0.1:3000/$1/$2;
        proxy_set_header X-API-Version $1;
    }

    # 5. 일반 접두사 — 관리자 (가장 긴 prefix 가 이김)
```

```
    location /admin/ {
        auth_basic "Restricted";
        auth_basic_user_file /etc/nginx/.htpasswd;
        proxy_pass http://127.0.0.1:3001/;
    }

    # 6. 기본 catch-all
    location / {
        proxy_pass http://127.0.0.1:3000;
    }
    access_log /var/log/nginx/lab-location.access.log;
}
```

03_lab-upstream-lb

```
## 응용 예제 ② — upstream 로드 밸런서 4 종 (RR / weighted / ip_hash / least_conn)
## 한 번에 한 upstream 블록만 활성화 (나머지는 주석)

## (A) 라운드로빈 (기본)
upstream backend_pool {
    server 127.0.0.1:3000;
    server 127.0.0.1:3001;
}

## (B) 가중치 + 헬스체크 + backup
# upstream backend_pool {
#     server 127.0.0.1:3000 weight=3 max_fails=3 fail_timeout=10s;
#     server 127.0.0.1:3001 weight=1 max_fails=3 fail_timeout=10s;
#     server 127.0.0.1:3002 backup;
# }

## (C) IP 해시 (sticky session)
# upstream backend_pool {
#     ip_hash;
#     server 127.0.0.1:3000;
#     server 127.0.0.1:3001;
# }

## (D) Least Connections
# upstream backend_pool {
#     least_conn;
#     server 127.0.0.1:3000;
#     server 127.0.0.1:3001;
# }

server {
    listen 80;
    server_name _;
    location / {
        proxy_pass http://backend_pool;
        proxy_set_header Host $host;
```

```
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
    access_log /var/log/nginx/lab-upstream.access.log;
}
```

04_lab-tls

```
## 응용 예제 ③ — TLS termination + HTTP→HTTPS 리다이렉트
## 자가서명 인증서를 /etc/nginx/certs/ 에 미리 생성 (setup_04_tls.sh 참조)

# 80 → 443 강제 리다이렉트
server {
    listen 80;
    server_name api.lab.local;
    return 301 https://$host$request_uri;
}

# 443 TLS termination → 백엔드 평문
server {
    listen 443 ssl http2;
    server_name api.lab.local;

    ssl_certificate     /etc/nginx/certs/api.lab.local.crt;
    ssl_certificate_key /etc/nginx/certs/api.lab.local.key;
    ssl_protocols       TLSv1.2 TLSv1.3;
    ssl_ciphers         HIGH:!aNULL:!MD5;
    ssl_session_cache   shared:SSL:10m;

    location / {
        proxy_pass http://127.0.0.1:3000;
        proxy_set_header X-Forwarded-Proto https;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
    access_log /var/log/nginx/lab-tls.access.log;
}
```

05_lab-cache

```
## 응용 예제 ④ — proxy_cache (응답 캐싱)
## /etc/nginx/nginx.conf 의 http {} 블록 또는 conf.d 에 proxy_cache_path 필요

proxy_cache_path /var/cache/nginx levels=1:2 keys_zone=api_cache:10m
                 max_size=1g inactive=60m use_temp_path=off;

server {
    listen 80;
    server_name _;
```

```
    location /api/ {
        proxy_cache api_cache;
        proxy_cache_valid 200 5m;
        proxy_cache_valid 404 1m;
        proxy_cache_use_stale error timeout updating http_500 http_502;
        proxy_cache_lock on;
        add_header X-Cache-Status $upstream_cache_status;
        proxy_pass http://127.0.0.1:3000;
    }
    access_log /var/log/nginx/lab-cache.access.log;
}
```

HANDS-ON LAB

실습 #3

Docker 실전 (단일 → compose → 네트워크·볼륨)

[필수]

IT 인프라 아키텍처 설계

Hands-on Labs · 20 시나리오

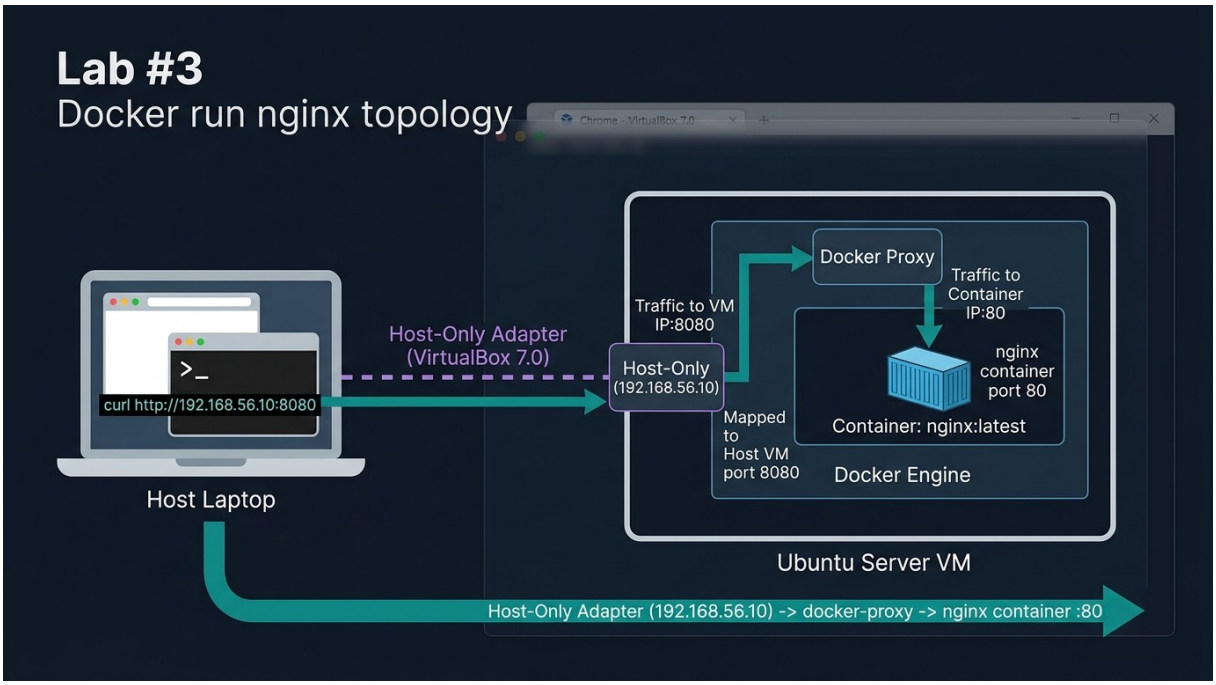
GenAI Labs

강사 박수현 · CEO / CTO

실습 목적

- `**docker run**` 한 줄로 컨테이너 띄우기 → 점진적으로 운영 패턴까지
- **이미지 / 컨테이너 / 볼륨 / 네트워크** 4 가지 개념 손에 잡기
- **docker compose** 로 다중 컨테이너 관리
- **볼륨**으로 데이터 영속화 + **네트워크 3 종** (bridge / host / none)
- 실전 운영 패턴: 로그 회전·헬스체크·재시작 정책

아키텍처 구성 (Topology)



위 토폴로지를 머리에 그린 상태로 아래 실습을 진행한다.

도구

- Docker Engine 24+ (`docker-ce` 또는 `docker.io`)
- Docker Compose v2 (`docker compose` 명령 — 하이픈 없음)

기본 예제 — 단일 컨테이너

```
# 1) Docker 설치 (Ubuntu)
curl -fsSL https://get.docker.com | sudo sh
sudo usermod -aG docker $USER     # sudo 없이 사용
newgrp docker                      # 그룹 적용

# 2) 가장 단순한 컨테이너
docker run -d --name web -p 8080:80 nginx:1.27
curl http://localhost:8080        # nginx 페이지 뜸

# 3) 상태 / 로그 / 재시작
docker ps                          # 실행 중
docker ps -a                       # 종료된 것 포함
docker logs -f web                 # 실시간 로그
docker restart web
docker stop web && docker rm web   # 종료 + 삭제

# 4) 이미지 / 컨테이너 정보
docker images                      # 이미지 목록
docker inspect web                 # 메타데이터 JSON
docker stats                       # 실시간 자원 (top 과 유사)
docker exec -it web bash           # 안으로 들어가기
```

응용 예제 ① — 볼륨 (데이터 영속화)

```
# 1) 명명된 볼륨 (Docker 관리)
docker volume create web-data
docker run -d --name web -v web-data:/usr/share/nginx/html nginx:1.27

# 2) 바인드 마운트 (호스트 경로)
mkdir -p /tmp/html && echo '<h1>From Host</h1>' > /tmp/html/index.html
docker run -d --name web -v /tmp/html:/usr/share/nginx/html:ro nginx:1.27
# :ro = read-only 마운트

# 3) tmpfs (메모리 마운트, 빠르지만 비영속)
docker run -d --name app --tmpfs /run --tmpfs /tmp myapp

# 4) 볼륨 관리
docker volume ls
```

```
docker volume inspect web-data
docker volume rm web-data
docker volume prune                 # 미사용 볼륨 일괄 삭제
```

응용 예제 ② — 네트워크 3 종

```
# 1) bridge (기본) - 컨테이너 격리 + 내부 통신
docker network create app-net
docker run -d --name db --network app-net postgres:16
docker run -d --name web --network app-net -p 8080:80 nginx:1.27
# web 에서 db 를 호스트명으로 접근: `db:5432`

# 2) host - 호스트 네트워크 그대로 (성능 최강, 격리 X)
docker run -d --network host nginx:1.27   # 호스트 80 번 그대로

# 3) none - 네트워크 없음 (격리 최강)
docker run -it --network none alpine ip a   # lo 만

# 네트워크 검사
docker network ls
docker network inspect app-net
```

응용 예제 ③ — docker compose

```
# docker-compose.yml
version: '3.9'
services:
  db:
    image: postgres:16
    environment:
      POSTGRES_PASSWORD: secret
      POSTGRES_DB: appdb
    volumes:
      - pgdata:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 5s
      timeout: 3s
      retries: 5

  web:
    image: nginx:1.27
    ports:
      - "8080:80"
    depends_on:
      db:
        condition: service_healthy
    restart: unless-stopped
    logging:
      driver: json-file
```

```
      options:
        max-size: "10m"
        max-file: "5"

volumes:
  pgdata:
```

```
docker compose up -d                # 백그라운드 시작
docker compose ps                   # 상태
docker compose logs -f web          # 로그
docker compose exec web bash        # exec
docker compose restart db
docker compose down                 # 정지 + 컨테이너 삭제 (볼륨은 남음)
docker compose down -v              # 볼륨까지 삭제
```

응용 예제 ④ — 운영 패턴

```
# 헬스체크
docker run -d --name web     --health-cmd='curl -f http://localhost/ || exit 1'
--health-interval=10s     --health-timeout=3s     --health-retries=3
nginx:1.27
docker inspect --format='{{.State.Health.Status}}' web

# 재시작 정책
docker run -d --restart=always nginx:1.27        # 무조건 재시작
docker run -d --restart=unless-stopped nginx     # 수동 중지 제외
docker run -d --restart=on-failure:5 nginx       # 실패 시 5 회

# 자원 제한
docker run -d --cpus=2 --memory=512m     --pids-limit=100 nginx:1.27

# 로그 회전
docker run -d     --log-driver=json-file     --log-opt max-size=10m     --log-opt
max-file=5     nginx:1.27
```

관전 포인트

1. **이미지 vs 컨테이너 vs 볼륨** — 명확히 구분
2. `-p 8080:80` = 호스트 8080 → 컨테이너 80
3. **bridge 네트워크**에서 컨테이너 이름이 곧 DNS
4. `docker compose`로 운영 배포 코드화
5. **헬스체크 + 재시작 정책**으로 자가 치유

자주 만나는 오류

- `Cannot connect to the Docker daemon` → `sudo systemctl start docker`
- `permission denied /var/run/docker.sock` → `usermod -aG docker $USER` 후 재로그인
- 포트 충돌 → `ss -tlnp | grep :8080` 다른 프로세스 종료
- 디스크 풀 → `docker system df` + `docker system prune -af`
- 컨테이너에 진입 안 됨 → `docker exec` 대신 `docker run -it ... bash`

운영 팁

- **이미지 태그 고정** — `nginx:1.27` (절대 `:latest` 운영 사용 X)
- **multi-stage build** 로 이미지 크기 축소
- **`.dockerignore`** 로 불필요한 파일 제외
- **레지스트리**: Harbor·ECR·GHCR (사내 레지스트리 권장)
- **K8s 전 단계로 compose** — 운영 입문에 충분

부록 — 설정 파일

04_docker-compose.yml

```
# 실습 #3 ④ — db(postgres) + web(nginx) 최소 운영 패턴
# - db 가 healthy 가 되기 전에는 web 이 뜨지 않음 (depends_on.condition)
# - 로그 회전으로 디스크 폭주 방지 (json-file 드라이버 + max-size · max-file)
# - 재시작 정책 unless-stopped (운영 표준; 수동 stop 외엔 무조건 재기동)

services:
  db:
    image: postgres:16-alpine
    environment:
      POSTGRES_PASSWORD: secret
      POSTGRES_DB: appdb
    volumes:
      - pgdata:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 5s
      timeout: 3s
      retries: 5
    restart: unless-stopped
    logging:
      driver: json-file
      options:
        max-size: "10m"
        max-file: "5"

  web:
    image: nginx:1.27
    ports:
      - "8085:80"
    depends_on:
      db:
        condition: service_healthy
    restart: unless-stopped
    logging:
      driver: json-file
      options:
        max-size: "10m"
        max-file: "5"

volumes:
  pgdata:
```

HANDS-ON LAB

실습 #4

Linux 시스템 모니터링 종합 (CPU·메모리·디스크·네트워크·프로세스)

[필수]

IT 인프라 아키텍처 설계

Hands-on Labs · 20 시나리오

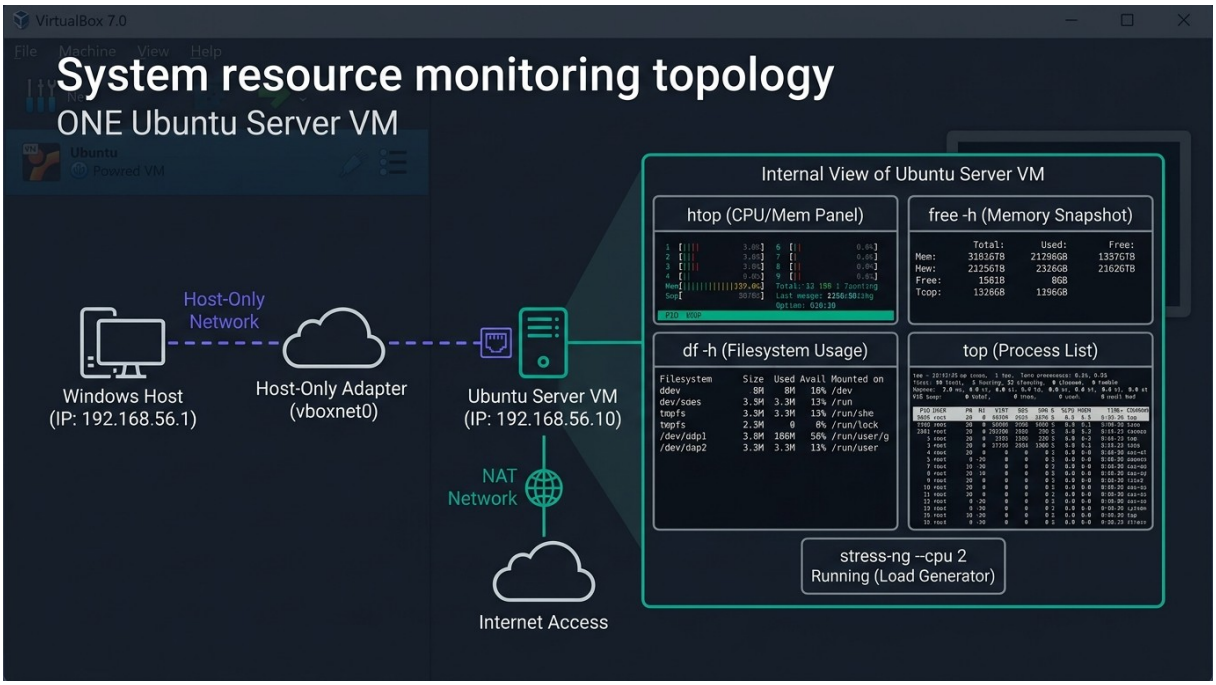
GenAI Labs

강사 박수현 · CEO / CTO

실습 목적

- **1 분 안에 「이 서버 지금 바쁜가?」** 진단 능력 체득
- **CPU / 메모리 / 디스크 / 네트워크 / 프로세스 / 파일** 5 축 모니터링 도구 한 번에
- `htop` `free` `df` 외 실무 도구 — `ps` `lsof` `ss`/`netstat` `iostat` `vmstat` `pidstat`
- **`stress-ng` 부하 스크립트**로 부하 발생·해소 패턴 보면서 도구 출력 변화 관찰
- 운영 1 차 디버깅 도구 정착

아키텍처 구성 (Topology)



위 토폴로지를 머리에 그린 상태로 아래 실습을 진행한다.

도구

영역	기본 (내장)	강화
CPU/Mem 인터랙티브	`top`	`htop` (컬러·트리·필터)
CPU 시계열	`vmstat 1`	`pidstat -u 1` (PID 별)
메모리	`free -h`	`smem slabtop`
디스크 용량	`df -h`	`du -sh *` `ncdu`
디스크 I/O	`iostat -xz 1`	`iotop` `pidstat -d 1`
네트워크 연결	`ss -tnp`	`netstat -tnp` (구버전)
네트워크 대역	`iftop`	`nethogs` (PID 별)
프로세스	`ps aux`	`pgrep` `pstree`
열린 파일	`lsof`	`lsof -i :PORT` `lsof -p PID`

시나리오 (약 30 분)

```
# 1) 도구 설치 (기본 외에 강화 도구들)
sudo apt update
sudo apt install -y htop sysstat iotop iftop nethogs ncdu lsof \
                    stress-ng strace tcpdump

# 2) tmux 4-pane 으로 동시 관찰 권장
tmux new -s monitor
# Ctrl-b % (가로 분할), Ctrl-b " (세로 분할)
# 각 pane: htop / vmstat 1 / iostat -xz 1 / iftop
```

기본 예제 — 5 축 모니터링 한 번에

1. CPU — `htop` (가장 자주 쓰는 도구)

```
htop
# F2  — 설정 (PerCPU bars·트리 모드)
# F3  — 프로세스 검색 (이름)
# F4  — 필터
# F5  — 트리 (부모-자식 관계)
# F6  — 정렬 키 변경 (CPU%·MEM%·TIME+ 등)
# F9  — 시그널 전송 (KILL·TERM 등)
# t   — 트리 토글
# H   — 사용자 스레드 숨김
# K   — 커널 스레드 숨김
# /   — 검색
# u   — 사용자 필터
```

핵심 컬럼:

- PR/NI — 우선순위/Nice (높을수록 양보)
- S — 상태 (R 실행·S 슬립·D 디스크 대기·Z 좀비)
- VIRT/RES/SHR — 가상/실/공유 메모리 (실제 메모리 = RES)
- CPU%/MEM% — 누적 vs 순간

2. 메모리 — `free`

```
free -h
#               total        used        free      shared  buff/cache   available
# Mem:           7.8Gi       1.2Gi       4.5Gi        12Mi       2.1Gi       6.3Gi
# Swap:          2.0Gi          0B       2.0Gi
```

암기 룰: available 만 보면 된다 (page cache 빼고 진짜 가용량).

3. 디스크 용량 — `df` + `du`

```
df -h                          # 마운트 별 사용량
df -i                          # inode 사용량 (file 갯수 폭증 시)

du -sh /var/log                # 디렉터리 합계
du -sh /var/* 2>/dev/null | sort -h    # 큰 순
ncdu /var                      # 인터랙티브 디스크 분석
```

4. 디스크 I/O — `iostat` + `iotop`

```
iostat -xz 1                   # 디바이스별 r/s w/s await %util
# %util 100% 가까우면 I/O 병목
# await > 50ms 면 응답 느림

sudo iotop                     # PID 별 IO 사용량 (실시간)
```

5. 네트워크 연결 — `ss` (`netstat` 후속)

```
ss -tnp                        # TCP 연결 + 프로세스
ss -tunlp                      # TCP+UDP 리스닝 (-l)
ss -s                          # 요약
ss -t state established        # 상태별 필터
ss -t state time-wait '( dport = :80 or sport = :80 )'

# 구식 동등
netstat -tnp                   # 동일
netstat -tunlp                 # 동일
```

6. 네트워크 대역폭 — `iftop` / `nethogs`

```
sudo iftop -i eth0             # 인터페이스 트래픽
sudo nethogs                   # PID 별 트래픽 (어떤 프로세스가 통신?)
```

7. 프로세스 — `ps` 패턴 모음

```
ps aux                          # 전체 (BSD 스타일)
ps -ef                          # 전체 (Unix 스타일)
ps -p PID -o pid,ppid,user,%cpu,%mem,rss,vsz,stat,start,time,command
ps -eo pid,user,%mem,rss,command --sort=-%mem | head    # 메모리 Top
ps -eo pid,user,%cpu,command --sort=-%cpu | head        # CPU Top

# 프로세스 트리
pstree -p
pstree -p $(pgrep -d, nginx)

# 검색
pgrep -a nginx                  # 이름으로 PID + cmd
pgrep -U lab                    # 사용자별
pidof nginx
```

8. 열린 파일 — `lsof` (운영 디버깅의 끝판)

```
sudo lsof -i :80                # 80 번 포트 누가 쓰나
sudo lsof -i TCP:443            # TCP 443 모든 연결
sudo lsof -i -P -n              # 모든 NW 연결 (포트 숫자·DNS 안풀이)
sudo lsof -p $(pgrep nginx | head -1)   # PID 가 연 파일 전부
sudo lsof | grep deleted        # 삭제됐는데 프로세스가 잡고 있는 파일 (디스크 안 비는 이유)
sudo lsof -u lab                # 사용자가 연 파일
sudo lsof /var/log/syslog       # 이 파일 누가 쓰나
```

응용 예제 ① — `stress-ng` 부하 발생 + 동시 관찰

시나리오: CPU 부하 5 초 → 휴식 5 초 → 10 초 → 휴식 10 초 반복

이걸 보면서 `htop` 이 어떻게 반응하는지 관찰.

```
# stress-ng 부하 생성 스크립트
cat > /tmp/cpu_wave.sh <<'EOF'
#!/bin/bash
# CPU 파동 부하 — 5/5, 10/10, 20/20 패턴 반복
set -e

declare -a PATTERN=(5 5 10 10 20 20 30 30)

while true; do
    for dur in "${PATTERN[@]}"; do
        # 짝수 index (0, 2, 4, 6) = 부하 시간 / 홀수 = 휴식
        if (( $(($(date +%s) % 2)) == 0 )); then
            true   # 단순 짝/홀 판단
        fi
    done
```

```
    # 명시적 패턴: 부하-휴식 쌍
    for i in 0 2 4 6; do
        load_dur=${PATTERN[i]}
        rest_dur=${PATTERN[i+1]}
        echo "   부하 ${load_dur}초"
        stress-ng --cpu $(nproc) --timeout ${load_dur}s --metrics-brief
        echo "   휴식 ${rest_dur}초"
        sleep ${rest_dur}
    done
done
EOF
chmod +x /tmp/cpu_wave.sh

# 실행
/tmp/cpu_wave.sh
```

옆 터미널에서 동시 관찰

```
# Terminal 1
/tmp/cpu_wave.sh

# Terminal 2 (또는 tmux pane)
htop                            # CPU 그래프가 파도처럼 위/아래
# CPU% 컬럼이 100→0→100 반복하는 것 확인

# Terminal 3
vmstat 1                        # us 컬럼 (사용자 CPU) 시계열로 보임

# Terminal 4
pidstat -u 1                    # PID 별 CPU 누가 먹나 (stress-ng 보임)
```

메모리 부하 — 같은 패턴

```
# 1GB 메모리를 5 초 점유 후 해제, 반복
while true; do
    echo "   1GB 메모리 5 초"
    stress-ng --vm 1 --vm-bytes 1G --vm-keep --timeout 5s
    echo "   휴식 5 초"
    sleep 5
done

# 관찰
watch -n 0.5 'free -h'          # 0.5 초 새로고침
```

I/O 부하

```
# 디스크 쓰기 부하
stress-ng --hdd 2 --hdd-bytes 500M --timeout 10s &
```

```
# 관찰
iostat -xz 1                    # %util 100% 보임
iotop                           # stress-ng-hdd 프로세스 위
```

종합 부하 + 종합 관찰 (1 줄)

```
# 4 축 동시 부하 (CPU 4 + memory 1GB + IO 2 + network 1) 30 초
stress-ng --cpu 4 --vm 1 --vm-bytes 1G --hdd 2 --sock 1 --timeout 30s --metrics-brief
```

응용 예제 ② — 「이 서버 왜 느려?」 5 분 진단 루틴

```
# Step 1: 로드 평균 (1·5·15 분)
uptime
# load average: 4.20, 2.10, 0.80   ← 5 분 전부터 가속

# Step 2: CPU 누가 먹나
top -bn1 | head -20
# 또는
ps -eo pid,user,%cpu,command --sort=-%cpu | head

# Step 3: 메모리 상태
free -h
ps -eo pid,user,%mem,rss,command --sort=-%mem | head

# Step 4: 디스크 차서?
df -h
df -i                           # inode 도

# Step 5: I/O wait 높은가
vmstat 1 5
# wa 컬럼 > 20 이면 디스크 I/O 병목

# Step 6: 누가 디스크 먹나
sudo iotop -ao -d 5             # 누적 (10 초 관찰)

# Step 7: 네트워크 연결 폭증?
ss -s
ss -tn state established | wc -l
ss -tn state time-wait | wc -l   # TIME_WAIT 폭증은 클라이언트 폭증

# Step 8: 어떤 프로세스가 통신?
sudo nethogs -t -d 5

# Step 9: 어떤 파일을 잡고 있나
sudo lsof -p $(pgrep -d, -f your-app) | head -30
```

응용 예제 ③ — 자주 쓰는 1-liner 패턴

```
# 메모리 상위 5
ps aux --sort=-%mem | awk 'NR<=6{print $2, $4, $11}'

# CPU 상위 5
ps -eo pid,%cpu,command --sort=-%cpu | head -6

# 좀비 프로세스 찾기
ps aux | awk '$8 ~ /Z/ {print}'

# D 상태 (uninterruptible — I/O 대기) 프로세스
ps aux | awk '$8 ~ /D/ {print}'

# 80 번 포트 누가 듣나
sudo ss -tlnp '( sport = :80 )'
sudo lsof -i TCP:80 -sTCP:LISTEN

# 한 디렉터리에서 가장 큰 10 개 파일
du -ah /var/log | sort -rh | head -10

# 시스템 부팅 후 누적 CPU 시간
ps -eo pid,user,time,command --sort=-time | head -10

# 1 분 동안 가장 트래픽 많이 쓴 IP (nginx access log)
sudo awk '{print $1}' /var/log/nginx/access.log | \
  sort | uniq -c | sort -rn | head -10

# 시간대별 트래픽 (시간 단위)
sudo awk '{print $4}' /var/log/nginx/access.log | \
  cut -c2-15 | sort | uniq -c | tail
```

관전 포인트

1. **`htop`의 CPU 그래프**가 `stress-ng` 부하 패턴과 정확히 일치
2. **`vmstat`의 `wa` 컬럼**이 I/O 부하 시 치솟음 (CPU 는 노는데 디스크 대기)
3. **`free`의 `available`** vs `free` 차이 — buff/cache 는 즉시 회수 가능
4. **`ss`가 `netstat`보다 10 배 빠름** — netstat 는 deprecated
5. **`lsof -p PID`**가 운영 디버깅의 끝판 — 파일·소켓·라이브러리 모두 보임

자주 만나는 오류

- `htop: command not found` → `apt install htop`
- `iotop: requires sudo` → `sudo iotop`

- `iostat: command not found` → `apt install sysstat`
- `vmstat` 의 `wa` 항상 0 → 디스크 부하 없음 (정상)
- `free` 의 used 가 높아 보임 → buff/cache 포함, `available` 보는 게 정답
- `ps aux | grep nginx | grep -v grep` → `pgrep -a nginx` 한 줄로
- TIME_WAIT 누적으로 포트 부족 → `net.ipv4.tcp_tw_reuse = 1` (sysctl)

운영 팁

- **모니터링 4 종 동시 화면**: tmux 4-pane + (htop / vmstat / iostat / iftop)
- **터미널 색**: `htop` 안 t 키 트리 + F6 정렬
- **로그 분석 1 줄**: `tail -f /var/log/syslog | grep --line-buffered ERROR`
- **자원 5 초 평균**: `sar -u 5 12` (5 초 12 번 = 1 분)
- **Cron 모니터**: `pidstat -u -p ALL 60 1` → 1 분 평균 한 번 → 로그 적재
- **`stress-ng` 더 다양**: `--matrix 4` (행렬 연산) `--cache 2` (캐시 부하) `--io 4` `--switch 2` (컨텍스트 스위치)

부록 — 설정 파일

04_oneliners.txt

```
# lab04 자주 쓰는 1-liner 모음
# 각 줄을 그대로 복사해서 실행해 보세요.

# ── 프로세스 정렬 ──────────────────────────────────────────────────────────
ps aux --sort=-%mem | awk 'NR<=6{print $2, $4, $11}'        # 메모리 Top 5
ps -eo pid,%cpu,command --sort=-%cpu | head -6              # CPU Top 5
ps -eo pid,user,time,command --sort=-time | head -10        # 누적 CPU Top 10
ps -eo pid,user,%mem,rss,command --sort=-%mem | head        # 메모리 (RSS)
ps aux | awk '$8 ~ /Z/ {print}'                             # 좀비
ps aux | awk '$8 ~ /D/ {print}'                             # I/O 대기
(uninterruptible)

# ── 포트·소켓 ──────────────────────────────────────────────────────────────
sudo ss -tlnp                                               # LISTEN 전부
sudo ss -tlnp '( sport = :80 )'                             # :80 누가 듣나
sudo ss -tn state established | wc -l                       # ESTABLISHED 갯수
sudo ss -tn state time-wait | wc -l                         # TIME_WAIT 갯수
sudo ss -t state established '( dport = :443 or sport = :443 )'
sudo lsof -i TCP:80 -sTCP:LISTEN                            # 80 LISTEN
sudo lsof -i -P -n                                          # 모든 NW 연결 (DNS 안 풀
이)

# ── 파일·디스크 ────────────────────────────────────────────────────────────
sudo du -ah /var/log | sort -rh | head -10                  # 큰 파일 Top 10
sudo du -shx /* 2>/dev/null | sort -hr | head -10           # root 하위 큰 디렉터리
df -h                                                       # 마운트 별 사용량
df -i                                                       # inode 사용량
sudo lsof | grep deleted                                    # 삭제됐는데 잡힌 파일
sudo lsof -p $(pgrep nginx | head -1)                       # PID 가 연 파일 전부
sudo lsof /var/log/syslog                                   # 이 파일 누가 쓰나

# ── 메모리 ─────────────────────────────────────────────────────────────────
free -h
watch -d -n 1 'free -h'                                     # 갱신 + 변경 강조
```

```
cat /proc/meminfo | head -10
sudo dmidecode --type 17 | grep -E "Size|Speed"             # DIMM 슬롯

# ─── CPU·load ──────────────────────────────────────────────
uptime
top -bn1 -c | head -15                                      # 배치 1회
vmstat 1 5                                                  # 1초 × 5회
sar -u 5 12                                                 # 5초 12회 = 1분
pidstat -u 1                                                # PID 별 CPU 시계열
pidstat -d 1                                                # PID 별 디스크 I/O 시계열

# ─── 트리·검색 ──────────────────────────────────────────────
pstree -p
pstree -p $(pgrep -d, nginx)
pgrep -a nginx                                              # 이름 → PID + cmd
pgrep -U lab                                                # 사용자 별
pidof nginx

# ─── 디스크 I/O ─────────────────────────────────────────────
iostat -xz 1                                                # 디바이스 별
sudo iotop -ao -d 5                                         # 누적 (10초 관찰)

# ─── 네트워크 트래픽 ────────────────────────────────────────
sudo iftop -i eth0
sudo nethogs

# ─── 로그 분석 ──────────────────────────────────────────────
tail -f /var/log/syslog | grep --line-buffered ERROR
sudo awk '{print $1}' /var/log/nginx/access.log | sort | uniq -c | sort -rn | head
-10
sudo awk '{print $4}' /var/log/nginx/access.log | cut -c2-15 | sort | uniq -c |
tail
```

HANDS-ON LAB

실습 #5

SSH 공개키 인증 (RSA vs ed25519, Windows·Linux)

[필수]

IT 인프라 아키텍처 설계

Hands-on Labs · 20 시나리오

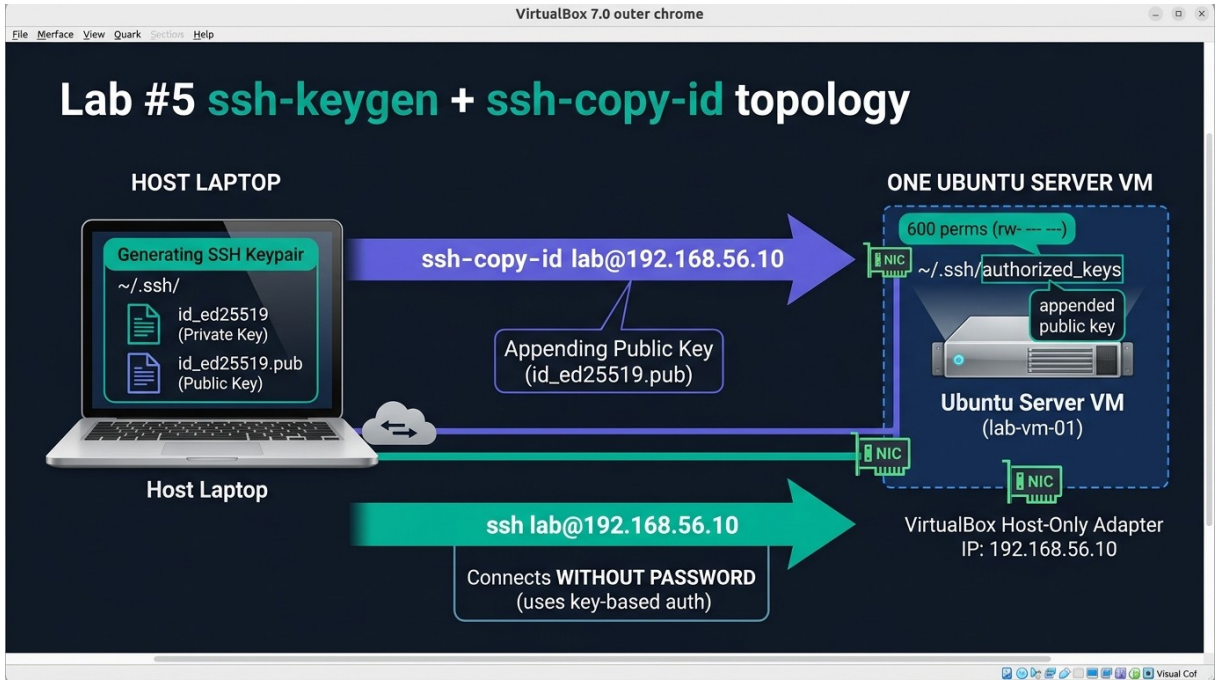
GenAI Labs

강사 박수현 · CEO / CTO

실습 목적

- **공개키 인증**으로 비밀번호 없는 SSH 접속 구축
- **RSA vs ed25519** 키 타입 비교 (길이·속도·보안·호환성)
- **Windows vs Linux** 키 저장 경로 차이 (OpenSSH for Windows · WSL · Git Bash · PuTTY)
- `ssh-agent` 로 키 패스프레이즈 자동 관리
- `~/.ssh/config` 로 호스트 별칭 + Jump Host 패턴
- SSH 운영 보안 핵심 (`PermitRootLogin`·`PasswordAuthentication` 끄기)

아키텍처 구성 (Topology)



위 토폴로지를 머리에 그린 상태로 아래 실습을 진행한다.

도구

- VirtualBox + Ubuntu Server 22.04 (target)
- Host 측 SSH 클라이언트 (OpenSSH · PuTTY · Windows Terminal)
- `ssh-keygen` `ssh-copy-id` `ssh-agent` `ssh-add`

기본 예제 — RSA vs ed25519 키 생성

키 타입 비교표

항목	RSA 2048	RSA 4096	**ed25519** (권장)
알고리즘	RSA (정수 인수분해)	RSA	**EdDSA (타원곡선)**
키 길이	2048 bit	4096 bit	**256 bit**
보안 강도	~112 bit	~140 bit	**~128 bit** (RSA 3072 동등)
생성 속도	빠름	느림 (1~3s)	**매우 빠름**
서명 속도	느림	매우 느림	**빠름**
키 파일 크기	~1.7 KB	~3.4 KB	**~400 B**
호환성	모든 SSH	모든 SSH	OpenSSH 6.5+ (2014)
권장 사용처	레거시 호환성	최고 보안 (느림 감수)	**신규 모든 환경**

RSA 4096 키 생성

```
ssh-keygen -t rsa -b 4096 -C "lab@example.com"

# 프롬프트:
# Enter file in which to save the key (~/.ssh/id_rsa): [Enter]
# Enter passphrase: [강력한 비밀번호 권장]
# Enter same passphrase again:

# 출력 파일:
# ~/.ssh/id_rsa      ← 비공개 키 (모드 600 필수)
# ~/.ssh/id_rsa.pub  ← 공개 키 (모드 644)
```

ed25519 키 생성 (권장)

```
ssh-keygen -t ed25519 -C "lab@example.com"

# 출력 파일:
# ~/.ssh/id_ed25519     ← 비공개
# ~/.ssh/id_ed25519.pub ← 공개

# 더 강력한 KDF iterations (선택)
```

```
ssh-keygen -t ed25519 -a 100 -C "lab@example.com"
# -a 100 → bcrypt KDF 100 라운드 (브루트포스 저항 ↑)
```

생성된 키 확인

```
# 비공개 키 (절대 공유 X)
cat ~/.ssh/id_ed25519
# -----BEGIN OPENSSH PRIVATE KEY-----
# b3BlbnNzaC1rZXktdjEAAAAACmFlczI1Ni1jdHIAAAAGYmNyeXB0...
# -----END OPENSSH PRIVATE KEY-----

# 공개 키 (서버에 등록할 것)
cat ~/.ssh/id_ed25519.pub
# ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIM... lab@example.com

# 키 fingerprint
ssh-keygen -lf ~/.ssh/id_ed25519.pub
# 256 SHA256:xxx... lab@example.com (ED25519)
```

Windows vs Linux 키 저장 경로

Linux / macOS

항목	경로
키 디렉터리	`~/.ssh/` (예: `/home/lab/.ssh/`)
비공개 키	`~/.ssh/id_ed25519` (모드 600)
공개 키	`~/.ssh/id_ed25519.pub` (모드 644)
`authorized_keys`	`~/.ssh/authorized_keys` (서버 측, 600)
`known_hosts`	`~/.ssh/known_hosts` (자동 생성)
`config`	`~/.ssh/config` (호스트 별칭, 600)

```
# 권한 (대부분의 SSH 실패 원인)
chmod 700 ~/.ssh
chmod 600 ~/.ssh/id_ed25519
chmod 644 ~/.ssh/id_ed25519.pub
chmod 600 ~/.ssh/authorized_keys
chmod 600 ~/.ssh/config
```

Windows — OpenSSH for Windows (Windows 10+)

항목	경로
키 디렉터리	`C:\Users\<사용자>\.ssh\`
비공개 키	`C:\Users\<사용자>\.ssh\id_ed25519`
공개 키	`C:\Users\<사용자>\.ssh\id_ed25519.pub`
`known_hosts`	`C:\Users\<사용자>\.ssh\known_hosts`
`config`	`C:\Users\<사용자>\.ssh\config`

```
# PowerShell — Windows 기본 OpenSSH 클라이언트
ssh-keygen -t ed25519 -C "lab@example.com"
# 키는 %USERPROFILE%\.ssh\ 에 저장됨
```

Windows — Git Bash

Git Bash 의 ssh-keygen 은 Linux 와 동일하게 동작하지만 키를:

- **MINGW 경로**: `C:\Users\<사용자>\.ssh\` (Windows OpenSSH 와 같은 위치)
- 단, `C:\Program Files\Git\usr\bin\ssh.exe` 가 OpenSSH for Windows 와 다른 바이너리

Windows — WSL (Ubuntu)

```
# WSL 안에서 — 진짜 Linux 환경
ls ~/.ssh/
# WSL 내부 경로: /home/<사용자>/.ssh/
# Windows 에서 보면: \\wsl$\Ubuntu\home\<사용자>\.ssh\

# Windows OpenSSH 키 공유하려면 심볼릭 링크
rm -rf ~/.ssh
ln -s /mnt/c/Users/$WIN_USER/.ssh ~/.ssh
```

Windows — PuTTY (별도 포맷 .ppk)

항목	경로
PuTTY 키 (.ppk)	사용자 지정 (보통 `C:\Users\<사용자>\Documents\keys\`)
변환 도구	**PuTTYgen** (.pub ↔ .ppk)

```
# OpenSSH → PuTTY
PuTTYgen → "Conversions" 메뉴 → "Import key" → OpenSSH 비공개 키 선택 → "Save private key" → .ppk 저장

# PuTTY → OpenSSH
PuTTYgen → "Conversions" → "Export OpenSSH key"
```

기본 예제 — 공개 키를 서버에 등록

방법 1: `ssh-copy-id` (가장 간단)

```
# Host 에서 — 비밀번호 입력 1 회 후 키 등록
ssh-copy-id -i ~/.ssh/id_ed25519.pub lab@192.168.56.10

# 이후로는 비밀번호 없이 SSH 가능
ssh lab@192.168.56.10
```

방법 2: 수동 (Windows OpenSSH 는 ssh-copy-id 없음)

```
# PowerShell 에서
type $env:USERPROFILE\.ssh\id_ed25519.pub | ssh lab@192.168.56.10 "mkdir -p ~/.ssh
&& cat >> ~/.ssh/authorized_keys && chmod 700 ~/.ssh && chmod 600
~/.ssh/authorized_keys"
```

방법 3: 서버에 직접 붙여넣기 (BMC 콘솔만 가능한 경우)

```
# 서버에서
mkdir -p ~/.ssh && chmod 700 ~/.ssh
nano ~/.ssh/authorized_keys
# 공개 키 한 줄 통째로 붙여넣기 → 저장
chmod 600 ~/.ssh/authorized_keys
```

응용 예제 ① — ssh-agent 로 패스프레이즈 자동 관리

```
# Linux/macOS
eval "$(ssh-agent -s)"            # agent 시작 (PID 출력)
ssh-add ~/.ssh/id_ed25519          # 패스프레이즈 1 회 입력 → 메모리 등록
ssh-add -l                         # 등록된 키 목록
ssh-add -L                         # 공개 키 출력
ssh-add -D                         # 모든 키 제거

# .bashrc / .zshrc 에 추가 (자동 시작)
if [ -z "$SSH_AUTH_SOCK" ]; then
    eval "$(ssh-agent -s)" >/dev/null
    ssh-add ~/.ssh/id_ed25519 2>/dev/null
fi

# Windows OpenSSH
Get-Service ssh-agent
Set-Service ssh-agent -StartupType Automatic
Start-Service ssh-agent
ssh-add C:\Users\$env:USERNAME\.ssh\id_ed25519
```

응용 예제 ② — `~/.ssh/config` 호스트 별칭

```ssh-config
# ~/.ssh/config (Linux) 또는 C:\Users\<사용자>\.ssh\config (Windows)

Host lab
  HostName 192.168.56.10
```

```
    User lab
    IdentityFile ~/.ssh/id_ed25519
    Port 22
Host prod
    HostName api.prod.example.com
    User deploy
    IdentityFile ~/.ssh/id_prod_ed25519
    Port 2222
    ServerAliveInterval 60
    ServerAliveCountMax 3
```

```
ssh lab             # = ssh -i ~/.ssh/id_ed25519 -p 22 lab@192.168.56.10
ssh prod            # 별칭 사용
```

```
##    응용 예제 ③ — Jump Host (Bastion)

```ssh-config
# Bastion 호스트
Host bastion
    HostName bastion.example.com
    User jump
    IdentityFile ~/.ssh/id_ed25519

# 내부망 서버 — bastion 경유
Host inner-db
    HostName 10.0.10.5
    User db-admin
    IdentityFile ~/.ssh/id_ed25519
    ProxyJump bastion          # ← 핵심

ssh inner-db                 # 한 줄로 bastion → inner-db
```

응용 예제 ④ — SSH 운영 보안 (서버 측 sshd_config)

```
sudo nano /etc/ssh/sshd_config

# 비밀번호 인증 끄기 (키 인증만 허용)
PasswordAuthentication no
ChallengeResponseAuthentication no

# root 로그인 금지
```
```

```
PermitRootLogin no

# SSH 키 인증만 허용
PubkeyAuthentication yes

# 빈 비밀번호 금지
PermitEmptyPasswords no

# 포트 변경 (보안 by obscurity, 무차별 봇 차단에는 효과)
Port 22                      # 운영은 보통 2222 또는 22XX

# 최대 동시 인증 시도
MaxAuthTries 3
MaxSessions 10

# 유휴 타임아웃
ClientAliveInterval 300
ClientAliveCountMax 2

# 사용자 화이트리스트
AllowUsers lab deploy
# 또는 그룹
AllowGroups sshusers

sudo sshd -t                   # 설정 검증 (운영 필수)
sudo systemctl restart sshd
```

응용 예제 ⑤ — 키 종류 다중 관리

```
# 키 분리: 사이트별 / 환경별
~/.ssh/id_lab_ed25519          # 실습용
~/.ssh/id_github_ed25519       # GitHub
~/.ssh/id_prod_ed25519         # 운영
~/.ssh/id_personal_rsa         # 개인용 (RSA, 레거시)

# config 에서 호스트별 매핑
Host github.com
    IdentityFile ~/.ssh/id_github_ed25519

Host *.prod.example.com
    IdentityFile ~/.ssh/id_prod_ed25519
```

관전 포인트

1. **ed25519 이 RSA 보다 모든 면에서 우월** (속도·키 크기·보안) — 신규는 ed25519 만
2. **`~/.ssh/` 권한 700 / 키 600** 이 SSH 실패의 90% 원인
3. **공개 키만 서버로** — 비공개 키는 절대 호스트 밖 X

4. `ssh-agent` + `~/.ssh/config` 로 매번 옵션 안 적어도 됨
5. `PasswordAuthentication no` + 키 인증만 = 봇 무차별 대입 99% 차단

자주 만나는 오류

- `Permissions are too open` → `chmod 600 ~/.ssh/id_ed25519`
- `Agent admitted failure to sign using the key` → `ssh-add ~/.ssh/id_ed25519`
- `ssh: connect to host ... port 22: Connection refused` → 서버 sshd 다운 또는 방화벽
- `Permission denied (publickey)` → `~/.ssh/authorized_keys` 또는 그 권한 점검
- `Host key verification failed` → `ssh-keygen -R 호스트` (known_hosts 갱신)
- Windows 에서 `id_ed25519: Permissions are too open` → 우클릭 → 속성 → 보안 → 현재 사용자만 권한
- WSL ↔ Windows 키 공유 시 권한 불일치 → WSL 안에서 `chmod 600` 다시

운영 팁

- **신규 키는 ed25519** — RSA 는 레거시 호환성만
- **키 백업**: `~/.ssh/` 통째 암호화 백업 (1Password·Bitwarden·하드웨어 키)
- **하드웨어 키**: YubiKey 5 → FIDO2 + ed25519-sk (`-t ed25519-sk`)
- **CA 인증서**: 대규모는 SSH CA 로 키 한 번 서명, 만료/취소 관리
- **감사**: `/var/log/auth.log` 또는 `journalctl -u sshd` 로그인 기록
- **WAF·Bastion** 앞에 두고 직접 SSH 노출 금지

부록 — 설정 파일

02_ssh_config

```
# === lab block ===
# 실습 #5 ② — 별칭 + ProxyJump 패턴
# 사용: ssh lab / ssh prod / ssh inner-db

Host lab
    HostName 192.168.56.10
    User lab
    IdentityFile ~/.ssh/id_ed25519
    Port 22

Host prod
    HostName api.prod.example.com
    User deploy
    IdentityFile ~/.ssh/id_prod_ed25519
    Port 2222
    ServerAliveInterval 60
    ServerAliveCountMax 3

# Bastion (DMZ)
Host bastion
    HostName bastion.example.com
    User jump
    IdentityFile ~/.ssh/id_ed25519

# 내부망 — bastion 경유 한 줄 진입
Host inner-db
    HostName 10.0.10.5
    User db-admin
    IdentityFile ~/.ssh/id_ed25519
    ProxyJump bastion
```

03_sshd_hardening.conf

```
# 실습 #5 ③ — sshd drop-in 하드닝
# 위치: /etc/ssh/sshd_config.d/99-lab-hardening.conf
# 우선순위: drop-in 이 본체보다 늦게 로드되어 덮어쓰기 적용

# 키 인증만 허용 — 비밀번호 로그인 차단 (봇 무차별 대입 99% 차단)
PasswordAuthentication no
ChallengeResponseAuthentication no
PubkeyAuthentication yes
PermitEmptyPasswords no
```

```
# root 직접 로그인 금지 (운영 표준)
PermitRootLogin no

# 인증 시도 / 세션 제한
MaxAuthTries 3
MaxSessions 10

# 유휴 세션 타임아웃 — 5 분 × 2 회 = 10 분 후 종료
ClientAliveInterval 300
ClientAliveCountMax 2

# 사용자 화이트리스트 (필요 시 주석 해제 후 사용)
# AllowUsers lab deploy
# AllowGroups sshusers
```

HANDS-ON LAB

실습 #6

HTTP·DNS·네트워크 1 차 디버깅 (curl·dig·ping·traceroute)

[필수]

IT 인프라 아키텍처 설계

Hands-on Labs · 20 시나리오

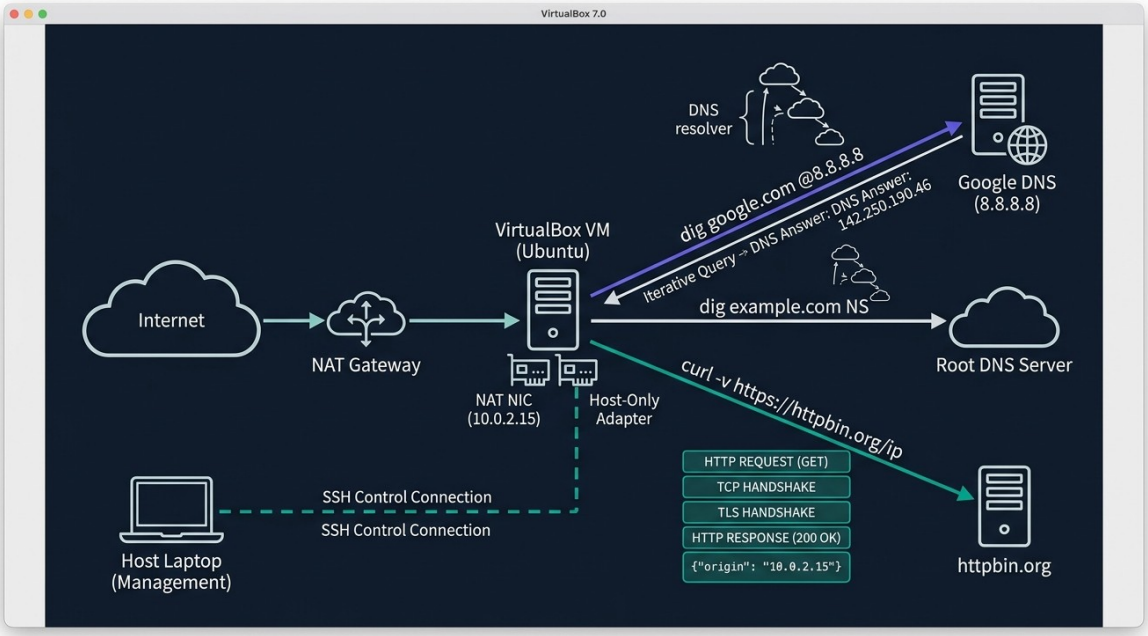
GenAI Labs

강사 박수현 · CEO / CTO

실습 목적

- **운영의 9 할은 1 차 디버깅 도구로 잡힘** — 4 종 도구 한 번에 정착
- `curl` 다양한 옵션 — `-I` 헤더만, `-v` 상세, `-L` 리다이렉트 추적, `-X` 메서드
- `dig` 다양한 record type (A·AAAA·MX·NS·TXT·CNAME·SOA)
- `ping` 의 한계 (ICMP 만 / 일부 사이트 ICMP 차단)
- **`traceroute` 가 요즘 잘 안 쓰이는 이유** (중간 라우터·방화벽이 ICMP/UDP 차단)
- 대안: `mtr`·`tcptraceroute`·`paris-traceroute`
- 실전: `curl www.naver.com` → 302 → HTTPS 리다이렉트의 의미

아키텍처 구성 (Topology)



위 토폴로지를 머리에 그린 상태로 아래 실습을 진행한다.

도구

- `curl` (HTTP 클라이언트, 기본 내장)
- `dig` (DNS 조회, `dnsutils` 패키지)
- `ping` (ICMP, 기본 내장)
- `traceroute` (UDP/ICMP 경로, `traceroute` 패키지)
- `mtr` (My Traceroute, 트래픽 + 손실율)
- `nslookup` (구식 DNS, 호환성용)

환경 준비

```
sudo apt update
sudo apt install -y curl dnsutils iputils-ping traceroute mtr-tiny tcpdump
```

기본 예제 — `curl` 옵션 8 종 모음

1. 헤더만 보기 — `-I` (가장 자주 씀)

```
curl -I https://www.google.com/
# HTTP/2 200
# date: Sat, 31 May 2026 14:22:08 GMT
# content-type: text/html; charset=ISO-8859-1
# ...
```

언제 쓰나: 상태 코드·캐시 헤더·서버 종류 확인.

2. 바디만 보기 — 기본 동작

```
curl https://api.github.com/
# {"current_user_url":"https://api.github.com/user","..."}
```

3. 헤더 + 바디 함께 — `-i`

```
curl -i https://www.google.com/
# HTTP/2 200       ← 헤더
# date: ...
# (빈 줄)
# <!DOCTYPE html>  ← 바디
# ...
```

4. 상세 추적 — `-v` (디버깅 1 순위)

```
curl -v https://www.google.com/
# *   Trying 142.250.196.36:443...
# * Connected to www.google.com (142.250.196.36) port 443
# * ALPN: server accepted h2
# * SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384
# * Server certificate:
```

```
# *  subject: CN=www.google.com
# *  start date: May  6 11:11:39 2026 GMT
# *  expire date: Jul 29 11:11:38 2026 GMT
# > GET / HTTP/2
# > Host: www.google.com
# > User-Agent: curl/7.81.0
# >
# < HTTP/2 200
# < date: ...
```

각 prefix 의 의미:

- `*` — curl 메타정보 (DNS·TCP·TLS·인증서)
- `>` — 송신 헤더 (request)
- `<` — 수신 헤더 (response)

5. 리다이렉트 추적 — `-L`

```
# 리다이렉트 무시 (기본)
curl -I http://www.naver.com
# HTTP/1.1 301 Moved Permanently
# Location: https://www.naver.com/
# (바디 따라가지 않음)

# 리다이렉트 추적
curl -IL http://www.naver.com
# HTTP/1.1 301 Moved Permanently        ← 1 번째
# Location: https://www.naver.com/
#
# HTTP/2 200                            ← 2 번째 (HTTPS 로 이동 후)
# server: NWS
# ...
```

`-L` 의 의미: Location 헤더 따라 자동 GET. 운영에서는 거의 항상 `-L` 사용 권장.

6. 메서드 지정 — `-X`

```
curl -X POST https://httpbin.org/post -d 'name=lab&id=1'
curl -X PUT  https://httpbin.org/put  -d 'value=10'
curl -X DELETE https://httpbin.org/delete
```

7. 헤더 추가 — `-H`

```
# JSON API
curl -X POST https://httpbin.org/post \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_TOKEN" \
  -d '{"name":"lab","id":1}'

# User-Agent 위장
curl -A "Mozilla/5.0" https://example.com
```

```
# 쿠키
curl -b "session=abc123" https://example.com
```

8. 파일 저장 — `-o` / `-O`

```
curl -o page.html https://example.com/
curl -O https://example.com/file.tar.gz   # 원본 파일명 그대로
curl -OJ https://example.com/download.cgi  # Content-Disposition 사용
```

9. 시간 측정 — `-w`

```
curl -o /dev/null -s -w "DNS: %{time_namelookup}\nConnect: %{time_connect}\nTTFB:
%{time_starttransfer}\nTotal: %{time_total}\n" https://www.google.com/
# DNS: 0.025
# Connect: 0.045
# TTFB: 0.180
# Total: 0.200
```

10. 인증서·SSL 점검 — `--cert-status` `-k`

```
# 자가서명 인증서 무시
curl -k https://192.168.56.10/

# 인증서 만료일 한 줄로
curl -vI https://www.google.com/ 2>&1 | grep -E "expire|subject:"

# 인증서 체인 보기
openssl s_client -connect www.google.com:443 -showcerts < /dev/null 2>&1 | grep -E
"(subject=|issuer=|Verify)"
```

실전 — `curl www.naver.com` 302 의미

```
curl -v www.naver.com
# 1) DNS 조회: www.naver.com → 223.130.200.219
# 2) TCP 80 (HTTP) 연결
# 3) GET / HTTP/1.1
# 4) 응답: HTTP/1.1 302 Found ← 또는 301
#    Location: https://www.naver.com/
```

왜 HTTPS 로 이동시키나?

- 평문 HTTP 는 중간자(MITM) 가능 — ISP·공용 와이파이 도청
- HTTPS 는 TLS 로 암호화 + 서버 인증
- 보안 표준 (HSTS) 으로 점차 HTTP 는 사라짐
- 검색 엔진(Google) 가산점

리다이렉트 코드 종류:

코드	의미	메서드 보존

301	Moved Permanently	GET 으로 변환 가능
302	Found (Moved Temporarily)	GET 으로 변환 가능
303	See Other	항상 GET 으로
307	Temporary Redirect	**메서드 보존** (POST→POST)
308	Permanent Redirect	**메서드 보존**

기본 예제 — `dig` DNS 조회

A 레코드 (IPv4) — 기본

```
dig www.google.com
# ;; ANSWER SECTION:
# www.google.com.    300    IN    A    142.250.196.36

# 짧게
dig +short www.google.com
# 142.250.196.36
```

다양한 record type

```
dig www.google.com AAAA       # IPv6
dig google.com MX             # 메일 서버
dig google.com NS             # 네임 서버
dig google.com TXT            # 텍스트 (SPF·DKIM)
dig google.com SOA            # Start of Authority
dig _dmarc.google.com TXT     # DMARC 정책
dig mail.google.com CNAME     # 별칭
dig google.com ANY            # 모두 (요즘은 막힘 — RFC 8482)
```

특정 DNS 서버 지정 — `@server`

```
dig @8.8.8.8 google.com           # Google DNS
dig @1.1.1.1 google.com           # Cloudflare
dig @168.126.63.1 google.com       # KT DNS (한국)
dig @192.168.1.1 mycompany.local   # 사내 DNS
```

역방향 (IP → 호스트명)

```
dig -x 142.250.196.36
# PTR    142.196.250.142.in-addr.arpa. → www.google.com
```

Trace — 루트부터 차근차근

```
dig +trace www.google.com
# .       (root) → com NS → google.com NS → www.google.com A
# 어디서 끊겨 있는지 진단에 유용
```

기본 예제 — `ping`

```
ping -c 4 google.com            # 4 번만
ping -i 0.5 google.com          # 0.5 초 간격 (빠르게)
ping -s 1400 google.com         # 1400 byte 패킷 (MTU 테스트)
ping -W 1 google.com            # 1 초 타임아웃
ping -f google.com              # flood (root 만)

# IPv6
ping6 google.com
```

ping 이 안 되는 이유 4 가지:

1. **호스트 다운** — 진짜로 꺼짐
2. **방화벽이 ICMP 차단** — AWS·일부 사이트는 ICMP echo reply 끔
3. **네트워크 경로 차단** — 중간 라우터 문제
4. **DNS 실패** — 호스트명 풀이가 안 됨 (`dig` 로 확인)

```
# 단계별 진단
dig google.com                  # DNS 풀이 OK?
ping -c 2 8.8.8.8               # IP 로 직접 (DNS 무관)
ping -c 2 google.com            # 호스트명으로
```

기본 예제 — `traceroute` & 한계

기본 사용

```
traceroute google.com
# 1  _gateway          1.2 ms
# 2  10.20.30.1        15.4 ms
# 3  * * *             ← 중간이 ICMP/UDP 차단
# 4  some-router       42.1 ms
# 5  * * *
# ...
# 12 142.250.196.36    105 ms
```

⚠ traceroute 가 요즘 잘 안 쓰이는 이유

1. **UDP 기반** (Linux 기본)
   - 중간 라우터의 방화벽이 UDP 다 차단
   - `* * *` 행이 너무 많아 진단 불가
2. **ICMP TTL Exceeded 응답에 의존**
   - 라우터가 ICMP 응답 자체를 끄거나 rate-limit
3. **로드밸런스·ECMP**

- 같은 목적지인데 패킷마다 다른 경로 → 일관성 없음

대안 ① — `mtr` (My Traceroute, 운영 1 순위)

```
mtr google.com                    # 인터랙티브 (실시간 loss/avg/best/worst)
mtr -r -c 100 google.com          # 100 번 평균 보고서 (운영 dump)
mtr -T google.com                 # TCP 모드 (방화벽 회피 ↑)
mtr -P 443 -T google.com          # TCP 443 으로 (HTTPS 경로 추적)
```

대안 ② — `tcptraceroute` (TCP 모드)

```
sudo apt install -y tcptraceroute
sudo tcptraceroute google.com 443
# 80, 443, 22 같은 일반 포트로 추적 — 방화벽 통과율 ↑
```

대안 ③ — `paris-traceroute` (ECMP 일관성)

```
sudo apt install -y paris-traceroute
paris-traceroute google.com
# ECMP 에서도 같은 경로 추적 (flow ID 고정)
```

응용 예제 — 운영 디버깅 시나리오

시나리오 1: 「사이트가 안 떠요」

```
# Step 1: DNS 문제인가
dig api.example.com
# 응답 없으면 → DNS 서버 확인 (/etc/resolv.conf)

# Step 2: 호스트 도달 가능?
ping -c 2 api.example.com
# 실패면 → 방화벽 또는 네트워크

# Step 3: TCP 포트 도달 가능?
nc -zv api.example.com 443
# 또는
curl -v --connect-timeout 5 https://api.example.com/
# 실패면 → 서버 측 sshd/nginx 다운, 또는 방화벽

# Step 4: HTTP 응답 받아짐?
curl -IL https://api.example.com/healthz
# 200 → OK, 5xx → 서버 측 문제, 4xx → 인증·URL 문제

# Step 5: 인증서 만료?
curl -vI https://api.example.com/ 2>&1 | grep "expire date"

# Step 6: 경로 어디서 문제?
mtr -r -c 50 api.example.com
```

시나리오 2: 「DNS 는 되는데 사이트가 안 떠요」

```
dig +short api.example.com       # 10.20.30.40
ping -c 2 10.20.30.40            # IP 로 직접 — 막힘
mtr -r -c 30 10.20.30.40         # 어느 라우터까지 가는지

# 다른 DNS 서버에서 어떻게 응답하나
dig @8.8.8.8 api.example.com
dig @1.1.1.1 api.example.com
# 응답 다르면 → DNS 캐시 문제 또는 GeoDNS
```

시나리오 3: 「리다이렉트 루프」

```
curl -ILv https://example.com/ 2>&1 | grep -E "(HTTP|Location)"
# HTTP/1.1 301 → Location: https://example.com/login
# HTTP/2 302  → Location: https://example.com/
# HTTP/2 302  → Location: https://example.com/login
# ... 반복 = 무한 루프
```

관전 포인트

1. **`curl -IL`** 한 줄로 리다이렉트 체인 + 응답 코드 다 보임
2. **`-v` 출력의 `*`, `>`, `<`** prefix 가 디버깅의 시각화
3. **`dig +short`** + 특정 DNS 서버 (`@8.8.8.8`)로 DNS 캐시 격리
4. **ping 실패 ≠ 사이트 다운** — ICMP 차단 가능성 항상 의심
5. **traceroute 대신 `mtr -T`** — 운영에선 더 신뢰

자주 만나는 오류

- `curl: (6) Could not resolve host` → DNS 문제 (`dig` 로 확인)
- `curl: (7) Failed to connect` → TCP 도달 불가 (방화벽·다운)
- `curl: (28) Operation timed out` → 응답 지연 (`--connect-timeout` `--max-time`)
- `curl: (35) SSL connect error` → TLS 인증서 / 시간 동기화 (NTP)
- `dig: connection refused` → DNS 서버 다운 또는 차단
- `ping: socket: Operation not permitted` → root 권한 또는 setuid (`sudo` / `setcap`)
- `traceroute: Cannot allocate memory` → root 권한
- TIME_WAIT 폭증 → 클라이언트 폭증 또는 keep-alive 미사용

운영 팁

- **`curl -w`** 시간 분해 (DNS·Connect·TTFB·Total) → 어느 단계 느린지

- **DNS 캐시 비우기**: `sudo systemd-resolve --flush-caches` (Linux)
- **MTR 운영 표준**: `mtr -r -c 100 -T -P 443 target` → 보고서용
- **HAR 캡처**: 브라우저 DevTools → Network → Save HAR — UI 디버깅
- **curl + jq**: `curl -s api.url | jq '.data[]'` JSON 파싱
- **HTTPie 권장**: `http https://api.url` — curl 대안, 출력 컬러
- **`-k` 운영 금지** — 자가서명만 임시 우회용

부록 — 설정 파일

01_curl_options.txt

```
# lab06 curl 옵션 패턴 모음

# ─── 기본 ──────────────────────────────────────────────────────────────────
curl -I  URL                                  # 헤더만 (HEAD)
curl -i  URL                                  # 헤더 + 바디
curl -v  URL                                  # 상세 (DNS/TCP/TLS/인증서)
curl -L  URL                                  # 리다이렉트 추적
curl -IL URL                                  # 위 두 조합 (가장 자주)
curl -s  URL                                  # silent (진행률 숨김)
curl -k  URL                                  # 자가서명 인증서 무시 (운영 금지)
curl --connect-timeout 5 --max-time 10 URL  # 타임아웃

# ─── 메서드 + 데이터 ──────────────────────────────────────────────────────
curl -X POST URL -d 'a=1&b=2'                                    # form
curl -X POST URL -H "Content-Type: application/json" -d '{...}' # JSON
curl -X PUT  URL -d 'value=10'
curl -X DELETE URL
curl --data-urlencode "q=hello world" URL                        # URL encode

# ─── 헤더 ──────────────────────────────────────────────────────────────────
curl -H "Authorization: Bearer TOKEN" URL
curl -H "X-Forwarded-For: 1.2.3.4" URL
curl -A "Mozilla/5.0" URL                    # User-Agent
curl -e "https://referer.com" URL            # Referer
curl -b "session=abc123" URL                 # Cookie 보내기
curl -c cookies.txt URL                      # 응답 쿠키 저장
curl -b cookies.txt URL                      # 저장된 쿠키 사용

# ─── 저장 ──────────────────────────────────────────────────────────────────
curl -o page.html URL                        # 지정 파일명
curl -O https://x.com/file.tar.gz            # 원본 파일명
curl -OJ https://x.com/download.cgi          # Content-Disposition
```

```
# ── 시간 측정 (-w) ──────────────────────────────────────
curl -o /dev/null -s -w \
"DNS=%{time_namelookup}s Connect=%{time_connect}s TLS=%{time_appconnect}s TTFB=%
{time_starttransfer}s Total=%{time_total}s\n" URL

# ── 인증서 ──────────────────────────────────────────────
curl -vI URL 2>&1 | grep -E "expire|subject:"
openssl s_client -connect HOST:443 -showcerts < /dev/null 2>&1 | grep -E
"(subject=|issuer=|Verify)"
curl --cacert /path/ca.pem URL              # 내부 CA
curl --cert client.pem --key client.key URL # 클라이언트 인증서 (mTLS)

# ── HTTP/2 / HTTP/3 ─────────────────────────────────────
curl --http2 URL
curl --http3 URL                            # curl 7.66+ + nghttp3

# ── 운영 자주 ──────────────────────────────────────────
curl -ILs URL | grep -E "^(HTTP|Location)"  # 리다이렉트 체인 한 줄씩
curl -s URL | jq '.data[]'                  # JSON 파싱
while sleep 5; do curl -s -o /dev/null -w "%{http_code} %{time_total}\n" URL; done
# 헬스체크 루프
```

03_dig_examples.txt

```
# lab06 dig 패턴 모음

# ── 레코드 타입별 ──────────────────────────────────────
dig DOMAIN A                # IPv4
dig DOMAIN AAAA             # IPv6
dig DOMAIN MX               # 메일 서버
dig DOMAIN NS               # 네임 서버
dig DOMAIN TXT              # SPF / DKIM / 검증 토큰
dig DOMAIN SOA              # Start of Authority
dig DOMAIN CNAME            # 별칭
dig DOMAIN CAA              # 인증서 발급 권한
dig DOMAIN SRV              # 서비스 위치 (SIP/XMPP)
dig _dmarc.DOMAIN TXT       # DMARC 정책
dig _spf.DOMAIN TXT         # SPF 자세히

# ── 짧게 / 자세히 ──────────────────────────────────────
dig +short DOMAIN           # 응답만
```

```
dig +noall +answer DOMAIN   # ANSWER 섹션만
dig +trace DOMAIN           # 루트→TLD→권한 서버
dig +tcp DOMAIN             # TCP 강제 (DNS over TCP)
dig +dnssec DOMAIN          # DNSSEC 서명 포함

# ─── 특정 DNS 서버 (@) ──────────────────────────────────────
dig @8.8.8.8  DOMAIN        # Google Public DNS
dig @1.1.1.1  DOMAIN        # Cloudflare
dig @9.9.9.9  DOMAIN        # Quad9 (보안)
dig @168.126.63.1 DOMAIN    # KT
dig @203.248.252.2 DOMAIN   # LG U+
dig @164.124.101.2 DOMAIN   # SKB
dig @192.168.1.1  DOMAIN    # 사내 DNS

# ─── 역방향 / 일괄 ──────────────────────────────────────────────────────
dig -x 8.8.8.8              # IP → 호스트명 (PTR)
dig -f hosts.txt            # 파일에서 도메인 목록 일괄

# ─── 응답 시간 / 시리얼 ─────────────────────────────────────────────────
dig DOMAIN | grep "Query time"
dig DOMAIN SOA | awk '/SOA/ {print $7}'   # 시리얼 (변경 추적)

# ─── 운영 패턴 ──────────────────────────────────────────────────────────
# 캐시 비우기 (Ubuntu systemd-resolved)
sudo resolvectl flush-caches
sudo systemd-resolve --flush-caches

# nslookup (구식, 호환성용)
nslookup DOMAIN
nslookup DOMAIN 8.8.8.8

# host (간단 한줄)
host DOMAIN
host -t MX DOMAIN
```

HANDS-ON LAB

실습 #7

백업·전송 종합 (tar·gzip·xz·zstd·scp·rsync)

[필수]

IT 인프라 아키텍처 설계

Hands-on Labs · 20 시나리오

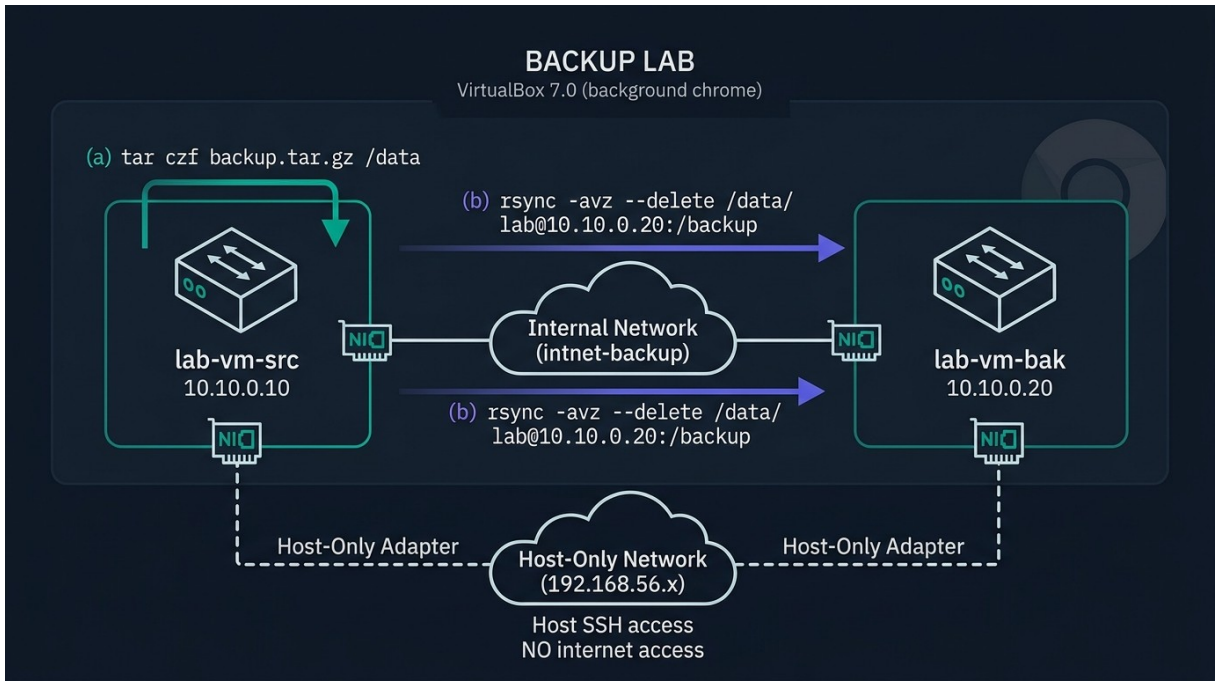
GenAI Labs

강사 박수현 · CEO / CTO

실습 목적

- **압축·아카이브**: tar / gzip / bzip2 / xz / zstd — 각 압축률·속도·CPU 비교
- **로컬 ↔ 원격 파일 전송**: scp / sftp / rsync — 언제 무엇을 쓰나
- **rsync 핵심 패턴**: 증분 전송·미러링·삭제 동기화·SSH·dry-run·exclude
- **운영 백업 패턴**: 풀백업·증분·차등·로테이션·검증
- 실전: 디렉터리 → tar.gz → 원격 전송 → 압축 해제 → 검증 전 과정

아키텍처 구성 (Topology)



위 토폴로지를 머리에 그린 상태로 아래 실습을 진행한다.

도구

- `tar` — 아카이브 (압축 X, 묶기만)
- `gzip gunzip zcat` — 표준 압축
- `bzip2 xz zstd` — 고압축 (압축률 ↑, 속도 ↓)
- `scp` — Secure Copy (SSH 위 단순 복사)
- `sftp` — Secure FTP (SSH 위 양방향)
- `rsync` — 증분 + 압축 + 동기화 (최강)

기본 예제 — tar 6 패턴

패턴 1: 묶기 (압축 없이)

```
tar cf data.tar /var/log
# c = create, f = file (대상 파일 지정)
```

패턴 2: gzip 압축

```
tar czf data.tar.gz /var/log
# z = gzip
```

패턴 3: bzip2 압축 (더 작지만 더 느림)

```
tar cjf data.tar.bz2 /var/log
# j = bzip2
```

패턴 4: xz 압축 (가장 작지만 가장 느림)

```
tar cJf data.tar.xz /var/log
# J = xz
```

패턴 5: zstd 압축 (최신 — 빠르면서도 작음)

```
tar --use-compress-program=zstd -cf data.tar.zst /var/log
# 또는
zstd -T0 < data.tar > data.tar.zst    # 멀티스레드
```

패턴 6: 해제

```
tar xf data.tar                # 자동 압축 감지 (gzip/bz2/xz)
tar xzf data.tar.gz            # 명시
tar xjf data.tar.bz2
tar xJf data.tar.xz
tar xf data.tar -C /target/    # 특정 디렉터리에 풀기
tar tf data.tar.gz             # 목록만 보기 (t = list)
tar tvf data.tar.gz            # 상세 목록 (-v)
```

압축률·속도 비교 (1GB log 기준)

도구	압축률	압축 시간	해제 시간	CPU 코어
없음 (tar)	1.00x	0.5s	0.5s	1
`gzip` (-6 기본)	0.35x	12s	4s	1
`bzip2`	0.28x	60s	20s	1
`xz` (-6 기본)	0.22x	90s	8s	1
`zstd` (-3 기본)	0.32x	6s	2s	**다중**
`zstd -19 --long`	0.24x	60s	2s	다중

권장:

- **빠른 백업** → `zstd` 또는 `gzip`
- **공간 절약** → `xz` 또는 `zstd -19`
- **호환성** → `gzip` (어디서나 풀림)

기본 예제 — scp 단순 복사

로컬 → 원격

```
scp file.txt lab@192.168.56.10:/tmp/
scp -r mydir/ lab@192.168.56.10:/tmp/          # 디렉터리 (-r)
scp -P 2222 file.txt lab@host:/tmp/            # 포트 변경
scp -i ~/.ssh/id_ed25519 file.txt lab@host:/tmp/   # 키 지정
```

원격 → 로컬

```
scp lab@192.168.56.10:/tmp/file.txt ./
scp lab@host:/var/log/syslog ~/Downloads/
```

원격 → 원격 (host 경유)

```
scp lab@host1:/tmp/file.txt lab@host2:/tmp/
scp -3 lab@host1:/tmp/file.txt lab@host2:/tmp/   # -3: 로컬 경유
```

`scp` 한계 (운영에선 거의 안 씀)

- 증분 X (매번 전체 전송)
- 진행률 표시 미흡
- 대용량 디렉터리 시 느림
- 끊기면 처음부터 다시
- `--exclude` 같은 필터 없음

대안: `rsync` (사실상 표준)

기본 예제 — `rsync` 7 가지 패턴

패턴 1: 로컬 → 로컬

```
rsync -av /src/ /backup/
# -a = archive (-rlptgoD: 재귀+심볼릭링크+권한+시간+그룹+소유자+장치파일)
# -v = verbose
```

중요: `/src/` (슬래시 있음) vs `/src` (없음) 차이

- `/src/` → 안의 내용물만 복사 → `/backup/file1 /backup/file2`
- `/src` → src 자체를 복사 → `/backup/src/file1 /backup/src/file2`

패턴 2: 로컬 → 원격 (SSH)

```
rsync -avz /src/ lab@192.168.56.10:/backup/
# -z = SSH 위 압축 전송 (대역폭 절약)

# 포트 / 키 지정
rsync -avz -e "ssh -p 2222 -i ~/.ssh/id_ed25519" /src/ lab@host:/backup/
```

패턴 3: 원격 → 로컬

```
rsync -avz lab@host:/var/log/ ./logs/
```

패턴 4: 증분만 전송 (rsync 의 핵심)

```
# 첫 실행: 모두 전송
rsync -av /src/ /backup/

# 두 번째: 변경된 파일만 (자동 — rsync 기본 동작)
rsync -av /src/ /backup/
```

패턴 5: 미러링 (--delete: 원본에 없는 파일 백업에서도 삭제)

```
rsync -av --delete /src/ /backup/

# 안전 운영: --delete-after (전부 복사 후에 삭제)
rsync -av --delete-after /src/ /backup/
```

패턴 6: dry-run (--dry-run / -n: 무엇이 전송될지 미리 확인)

```
rsync -avn /src/ /backup/
# 실제 전송 X — 변경 예정 파일 목록만 출력
```

패턴 7: 제외 (`--exclude` / `--exclude-from`)

```
rsync -av --exclude='*.log' --exclude='cache/' /src/ /backup/

# 여러 규칙 파일로
cat > /tmp/exclude.list <<EOF
```

```
*.log
*.tmp
cache/
node_modules/
.git/
EOF
rsync -av --exclude-from=/tmp/exclude.list /src/ /backup/
```

응용 예제 ① — 백업 운영 패턴

일일 풀 백업 (간단)

```
#!/bin/bash
DATE=$(date +%Y%m%d)
SRC=/var/lib/myapp
DST=/backup
TARGET=lab@backup-host:/backup

# 1) 로컬 압축
tar czf /tmp/myapp_${DATE}.tar.gz -C ${SRC} .

# 2) 무결성 체크섬
sha256sum /tmp/myapp_${DATE}.tar.gz > /tmp/myapp_${DATE}.tar.gz.sha256

# 3) 원격 전송
rsync -avz /tmp/myapp_${DATE}.tar.gz* ${TARGET}/

# 4) 로컬 임시 파일 정리
rm -f /tmp/myapp_${DATE}.tar.gz*

# 5) 7일 이전 백업 삭제 (원격)
ssh lab@backup-host "find /backup -name 'myapp_*.tar.gz' -mtime +7 -delete"
```

증분 백업 (rsync + hardlink)

```
#!/bin/bash
DATE=$(date +%Y%m%d_%H%M)
PREV=$(ls -1dt /backup/snapshot.* 2>/dev/null | head -1)

# 새 스냅샷 — 이전과 변경되지 않은 파일은 hardlink (디스크 절약)
rsync -av --delete \
    ${PREV:+--link-dest=${PREV}} \
    /src/ /backup/snapshot.${DATE}/

# 30일 이전 스냅샷 삭제
find /backup -maxdepth 1 -name 'snapshot.*' -mtime +30 -exec rm -rf {} +
```

핵심: `--link-dest` 로 이전 스냅샷과 같은 파일은 하드링크 → 100GB 백업 30 일치가 105GB 로 끝남.

차등 백업 (가장 최근 풀 백업과의 차이만)

```
# 매주 일요일 풀 백업
DATE=$(date +%Y%m%d)
[ "$(date +%u)" -eq 7 ] && tar czf /backup/full_${DATE}.tar.gz /src/

# 매일 차등 (지난 풀백업 이후 변경)
LAST_FULL=$(ls -1t /backup/full_*.tar.gz | head -1)
LAST_FULL_DATE=$(stat -c %y "$LAST_FULL" | cut -d' ' -f1)
tar czf /backup/diff_${DATE}.tar.gz \
    --newer="$LAST_FULL_DATE" /src/
```

응용 예제 ② — 실전 시나리오: 디렉터리 → 원격 → 검증

```
SRC=/var/www
TMP_TGZ=/tmp/www_$(date +%s).tar.gz
REMOTE=lab@192.168.56.20:/backup/

# 1) 로컬 압축 + 체크섬
tar czf $TMP_TGZ -C $SRC . && \
    sha256sum $TMP_TGZ > $TMP_TGZ.sha256

# 2) 원격 전송 (dry-run 으로 미리 확인)
rsync -avzn $TMP_TGZ* $REMOTE   # 미리 보기
rsync -avz  $TMP_TGZ* $REMOTE   # 실제 전송

# 3) 원격에서 무결성 검증
ssh ${REMOTE%:*} "cd ${REMOTE#*:} && sha256sum -c $(basename $TMP_TGZ).sha256"
# OK 또는 FAILED 출력

# 4) 원격에서 풀기
ssh ${REMOTE%:*} "mkdir -p /restore && tar xzf ${REMOTE#*:}$(basename $TMP_TGZ) -C
/restore/"

# 5) 로컬 임시 파일 정리
rm -f $TMP_TGZ*
```

응용 예제 ③ — rsync 진행률·속도 제어

```
# 진행률 표시
rsync -avz --progress /src/ /backup/

# 대역폭 제한 (KB/s) — 운영 환경에서 다른 트래픽 보호
rsync -avz --bwlimit=10000 /src/ remote:/backup/   # 10MB/s 제한

# 부분 전송 재시도 (끊겨도 이어받기)
rsync -avz --partial --partial-dir=.rsync-partial /src/ remote:/backup/

# 대용량 전송: 압축 X (이미 압축된 파일은 -z 빼는 게 빠름)
rsync -av --progress /backup/big.tar.gz remote:/dst/
```

응용 예제 ④ — rsync 데몬 모드 (SSH 없이)

```
# 서버 측 /etc/rsyncd.conf
[backup]
    path = /backup
    read only = no
    auth users = lab
    secrets file = /etc/rsyncd.secrets
    hosts allow = 192.168.56.0/24

# 시작
sudo rsync --daemon

# 클라이언트
rsync -avz /src/ rsync://lab@server/backup/
# 또는 (덜 자주 씀)
rsync -avz /src/ server::backup/
```

언제 데몬 모드? — SSH 없는 옛 환경, NAS, 컨테이너 백업. 기본은 SSH 모드 권장.

응용 예제 ⑤ — tar + ssh 한 줄 (임시 백업)

```
# 로컬 → 원격으로 압축하면서 바로 전송 (디스크 임시 파일 없이)
tar czf - /var/www | ssh lab@host "cat > /backup/www_$(date +%s).tar.gz"

# 원격 → 로컬로 받기
ssh lab@host "tar czf - /var/www" > www.tar.gz

# 압축 + 암호화 + 전송 (gpg)
tar czf - /var/www | gpg --symmetric --cipher-algo AES256 | \
    ssh lab@host "cat > /backup/www.tar.gz.gpg"
```

응용 예제 ⑥ — 비교표: scp vs sftp vs rsync

항목	scp	sftp	rsync
증분 전송			
진행률	기본	일부	`--progress`
끊김 재시도		수동	`--partial`
양방향		(인터랙티브)	
압축	SSH-level	SSH-level	`-z` 또는 SSH-level
권한·시간	기본 보존	옵션	`-a` 자동
`--exclude`			
`--delete`			
적용처	단발 복사	인터랙티브 탐색	**운영 동기화**

일반 규칙: 한 파일 단발 = `scp`, 디렉터리·반복 = `rsync`.

관전 포인트

1. **`tar czf` / `xf`** — 압축 자동 감지 (운영 1 순위)
2. **rsync 의 슬래시** — `/src/` (안의 내용) vs `/src` (자체)
3. **`-n` (dry-run)** — 실제 전송 전 반드시 확인
4. **`--link-dest`** — 디스크 절약 증분 백업
5. **체크섬 (sha256sum)** — 전송 후 무결성 검증 운영 표준

자주 만나는 오류

- `tar: Removing leading /` → 안전 동작 (절대경로 → 상대경로). 복원 시 위치 확인.
- `rsync: command not found on remote` → 원격에 rsync 설치 (`apt install rsync`)
- `Permission denied` → 키 인증·sudo 권한·SELinux
- 디스크 풀 → `df -h` 확인. tar 작업 중 풀리면 .tar.gz 손상
- 압축률 안 줄어듦 → 이미 압축된 파일 (.jpg·.mp4) — gzip 무의미
- rsync 매번 전체 전송 → 시간이 변경되어 그럴 수 있음 (NTP 동기화 확인)

운영 팁

- **자동 백업 스크립트**는 `flock` 으로 중복 실행 방지
- **백업 검증**은 무조건 매일 — 백업이 안 되는 거 모르고 사고 나면 끝
- **3-2-1 룰**: 3 복사본, 2 다른 매체, 1 외부
- **GFS 로테이션** (Grandfather-Father-Son): 일/주/월 단위
- **rsync + cron**: `0 2 * * * rsync -avz --delete /src/ remote:/backup/`
- **Restic / Borg**: 운영용 백업 솔루션 (암호화·중복제거 자동)
- **압축 알고리즘**: zstd 가 가장 균형 좋음 (gzip 대체 진행 중)

부록 — 설정 파일

05_rsync_exclude.list

```
# rsync 제외 규칙 — rsync --exclude-from=configs/05_rsync_exclude.list
# 운영 백업에서 흔히 제외하는 패턴
*.log
*.tmp
*.swp
*.cache
cache/
node_modules/
.git/
__pycache__/
*.pyc
.DS_Store
Thumbs.db
```

HANDS-ON LAB

실습 #8

네트워크 경로 진단 (ping·traceroute·mtr·tracepath)

[필수]

IT 인프라 아키텍처 설계

Hands-on Labs · 20 시나리오

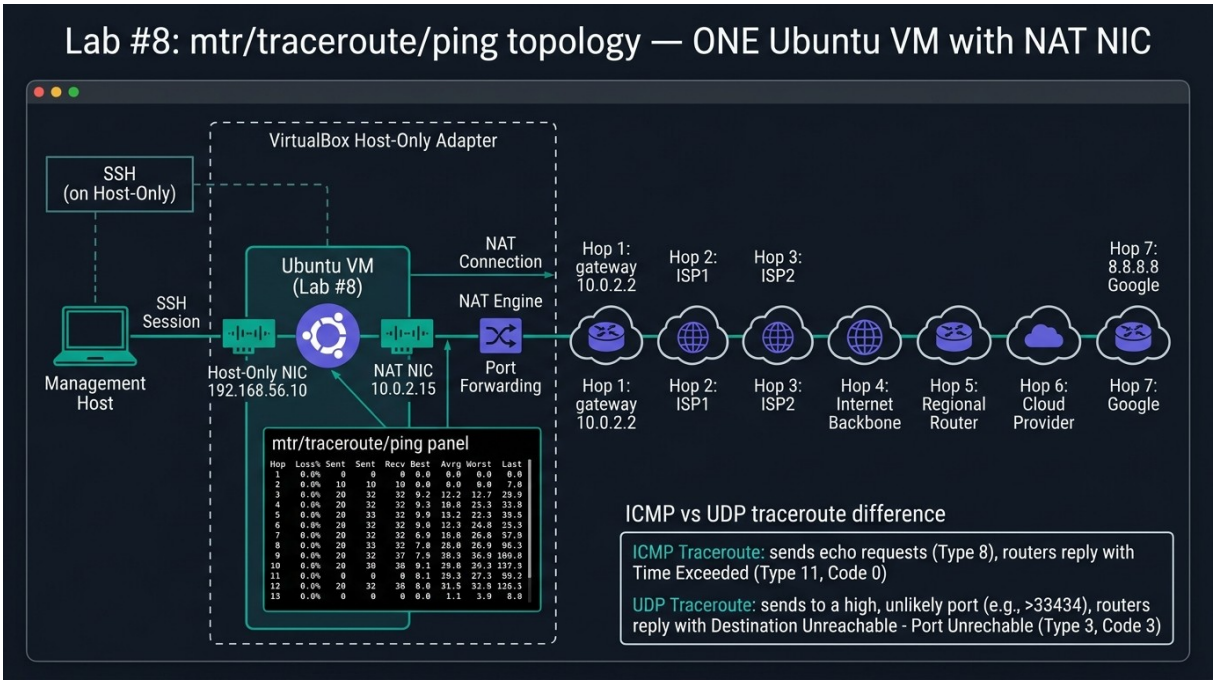
GenAI Labs

강사 박수현 · CEO / CTO

실습 목적

- ICMP / UDP / TCP 기반 경로 진단 도구 비교
- `mtr` 한 줄로 운영 보고서
- **TCP 모드 추적** (방화벽 우회)
- **MTU 탐색**, **MPLS 라벨**, **AS 번호** 추적

아키텍처 구성 (Topology)



위 토폴로지를 머리에 그린 상태로 아래 실습을 진행한다.

도구

- `ping` `ping6` (ICMP echo)
- `traceroute` (UDP 기본, ICMP/TCP 옵션)
- `mtr` (My Traceroute, 실시간 통계)
- `tracepath` (MTU 탐색)
- `tcptraceroute` (TCP 모드)

기본 예제

```
sudo apt install -y mtr-tiny tcptraceroute traceroute

# ping
ping -c 4 google.com              # 4번
ping -i 0.2 -c 20 google.com      # 0.2초 간격, 20번
ping -s 1472 -M do google.com     # MTU 1500 테스트 (1472+28=1500)
ping6 google.com                  # IPv6

# traceroute (UDP 기본)
traceroute google.com
traceroute -I google.com          # ICMP 모드
traceroute -T -p 443 google.com   # TCP 443

# mtr 인터랙티브
mtr google.com
# d: display mode 토글, n: DNS 토글, c: cycle count

# mtr 보고서
mtr -r -c 100 google.com          # 100번 통계 보고서
mtr -r -c 50 -T -P 443 google.com  # TCP 443, 50번
mtr -r -c 50 -b google.com        # IP+호스트명 같이
```

응용 예제 ① — MTU 탐색

```
# 패킷이 큰 사이즈에서 끊기면 PMTU 문제 의심
ping -s 1472 -M do -c 3 google.com    # 1500 (성공이면 OK)
ping -s 1500 -M do -c 3 google.com    # 1528 (DF bit, MTU 1500 초과면 실패)
tracepath google.com                  # 자동 PMTU 탐색
```

응용 예제 ② — AS 번호·BGP 경로

```
# AS 번호 표시
mtr --aslookup google.com
# AS15169 (Google), AS3356 (Lumen), ...
```

```
# whois 로 IP 소속 확인
whois 142.250.196.36 | grep -E "^(OrgName|country)"
```

응용 예제 ③ — 운영 보고서 자동 생성

```
# /etc/cron.hourly/network-mtr.sh
#!/bin/bash
DATE=$(date +%Y%m%d_%H%M)
mtr -r -c 50 -T -P 443 api.example.com > /var/log/mtr_${DATE}.txt
mtr -r -c 50 -T -P 443 db.example.com >> /var/log/mtr_${DATE}.txt
# Slack 알림 (loss > 5%)
LOSS=$(mtr -r -c 30 api.example.com | awk 'END{print $3}' | tr -d '%')
[ "${LOSS%.*}" -gt 5 ] && curl -X POST -d "text=Network loss: ${LOSS}%"
$SLACK_HOOK
```

관전 포인트

1. **`mtr -r -c 100 -T -P 443`** = 운영 표준 (TCP·100 회·보고서)
2. ****`* * *`**** 라인은 보통 방화벽 — 진짜 끊김 아닐 수 있음
3. **Loss % vs Drop %** — mtr 출력 해석
4. PMTU 문제는 ping 이 작은 패킷은 OK, 큰 건 실패
5. AS 번호로 어느 회선·통신사인지 추적

자주 만나는 오류

- `Operation not permitted` → `setcap cap_net_raw+p $(which mtr)`
- 모든 hop 이 `* * *` → 방화벽 또는 TCP 모드(`-T`) 시도
- 시간 측정이 부정확 → NTP 동기화 (`timedatectl status`)

운영 팁

- **운영 진단은 `mtr -T`** — UDP/ICMP 보다 방화벽 통과율 ↑
- **장기 모니터링**: SmokePing 또는 SNMP RTT
- **클라우드 환경**: AWS VPC Flow Logs, GCP VPC logs

부록 — 설정 파일

01_mtr_patterns.txt

```
# lab08 mtr 패턴 모음

# ── 인터랙티브 ──────────────────────────────────────────────────────────
sudo mtr -n TARGET                          # IP 만 표시 (DNS off)
sudo mtr TARGET                             # DNS on
sudo mtr -b TARGET                          # IP + 호스트명 양쪽
sudo mtr --aslookup TARGET                  # AS 번호 표시

# ── 보고서 모드 (-r) ──────────────────────────────────────────────────
sudo mtr -r -c 100 -n TARGET                # 100 회 평균 보고서
sudo mtr -r -c 50 -T -P 443 -n TARGET       # 운영 표준 (TCP 443)
sudo mtr -r -c 30 -I -n TARGET              # ICMP 모드 보고서
sudo mtr -r -c 30 -u TARGET                 # UDP 모드 (기본)
sudo mtr -j -c 30 TARGET                    # JSON 출력 (후처리용)

# ── 패킷 옵션 ──────────────────────────────────────────────────────────
sudo mtr --psize 1400 TARGET                # 패킷 사이즈 1400
sudo mtr --interval 0.5 TARGET              # 0.5 초 간격 (빠르게)
sudo mtr --first-ttl 5 TARGET               # 5 번째 hop 부터
sudo mtr --max-ttl 20 TARGET                # 최대 20 hop

# ── 운영 1-liner ──────────────────────────────────────────────────────
# 손실율만 추출 (마지막 hop)
sudo mtr -r -c 30 -T -P 443 TARGET | awk 'END{print $3}'

# RTT 평균만 (마지막 hop)
sudo mtr -r -c 30 -T -P 443 TARGET | awk 'END{print $6}'

# JSON 으로 받아 jq 처리
sudo mtr -j -c 10 TARGET | jq '.report.hubs[-1] | {Host, Loss, Avg}'

# ── 비교: traceroute ──────────────────────────────────────────────────
traceroute -n TARGET                        # UDP 기본
sudo traceroute -I TARGET                   # ICMP
sudo traceroute -T -p 443 TARGET            # TCP SYN
sudo tcptraceroute TARGET 443               # TCP, 방화벽 회피
tracepath TARGET                            # 자동 PMTU 탐색
```

```
# ─── 관련 도구 ──────────────────────────────────────────────────────────
ping -c 4 TARGET
ping -M do -s 1472 TARGET                # PMTU 1500 확인
ping6 -c 4 TARGET                        # IPv6
ip route                                 # 라우팅 테이블
ip route get TARGET                      # 어느 인터페이스로 나가나
ip neigh                                 # ARP / NDP 테이블

# ─── 인증·SUID ──────────────────────────────────────────────────────────
# mtr 가 setuid 없이 실행되도록 cap 부여
sudo setcap cap_net_raw+p $(which mtr)
```

HANDS-ON LAB

실습 #9

시스템 자원 4 축 동시 관찰 (htop·vmstat·iostat·sar)

[권장]

IT 인프라 아키텍처 설계

Hands-on Labs · 20 시나리오

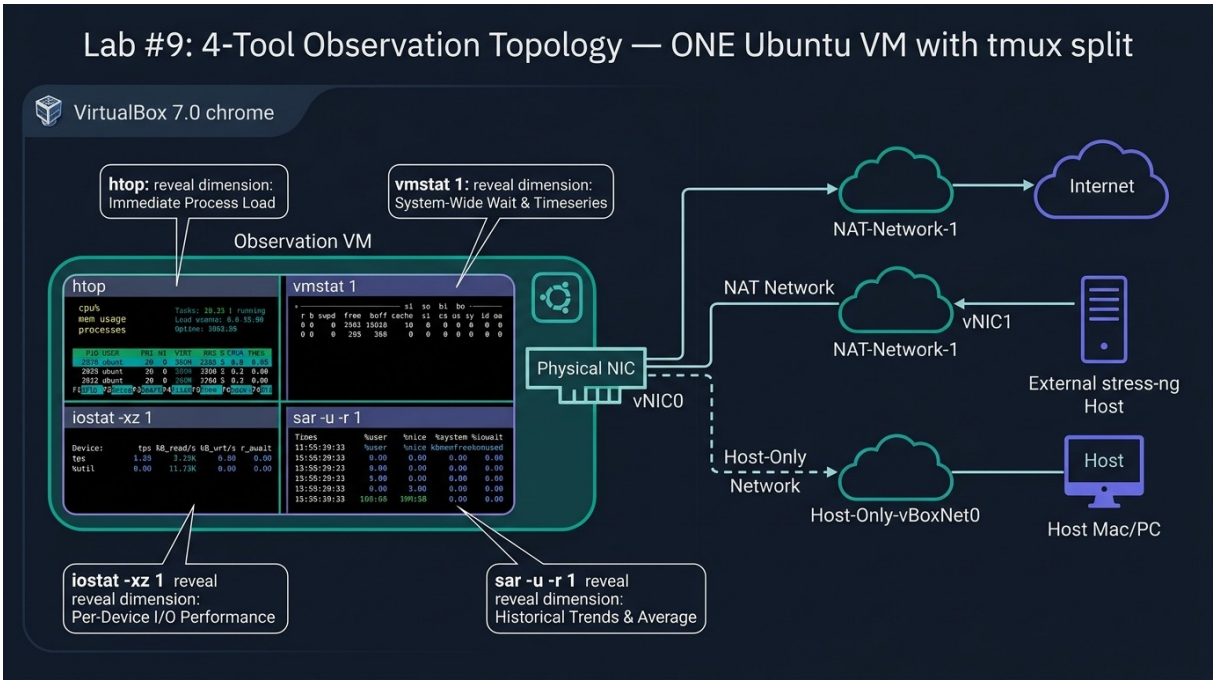
GenAI Labs

강사 박수현 · CEO / CTO

실습 목적

- **htop** (인터랙티브) + **vmstat** (CPU/MEM 시계열) + **iostat** (디스크) + **sar** (시계열 누적)
- tmux 4-pane 으로 동시 관찰
- 부하 발생 + 4 도구가 어떻게 반응하는지 시간순 관찰
- 운영 1 차 분석 도구 정착

아키텍처 구성 (Topology)



위 토폴로지를 머리에 그린 상태로 아래 실습을 진행한다.

도구

- `htop vmstat iostat sar pidstat dstat`
- `stress-ng` (부하 발생)
- `tmux` (4-pane)

기본 예제

```
sudo apt install -y htop sysstat stress-ng dstat tmux

# tmux 4-pane
tmux new -s mon
# Ctrl-b %  (가로 분할)
# Ctrl-b "  (세로 분할)
# Pane 1: htop
# Pane 2: vmstat 1
# Pane 3: iostat -xz 1
# Pane 4: free -h && sleep 1 (또는 watch -n 1 free)
```

부하 + 관찰

```
# 30 초 CPU 100% 4 코어 + 메모리 512MB
stress-ng --cpu 4 --vm 1 --vm-bytes 512M --timeout 30s --metrics-brief
```

응용 예제 ① — vmstat 컬럼 해독

```
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 4194304 102400 1048576    0    0     0     0  120  200  5  3 90  2  0
```

- `r` = 실행 대기 (CPU 부족 신호 > 코어 수)
- `b` = 비차단 대기 (D 상태, 디스크 I/O 대기)
- `si/so` = 스왑 in/out (메모리 부족 신호, 0 이어야)
- `bi/bo` = 블록 in/out (디스크 I/O)
- `cs` = 컨텍스트 스위치 (수만이면 정상, 수십만이면 의심)
- `us/sy/id/wa/st` = user/system/idle/io-wait/steal

응용 예제 ② — iostat 컬럼 해독

```
Device  r/s     w/s     rkB/s     wkB/s   rrqm/s wrqm/s %rrqm %wrqm     r_await
w_await  aqu-sz rareq-sz wareq-sz  svctm  %util
sda     100   250    1200      3000        2      5    2.0    2.0       12.0     8.5
3.5     12.0    12.0  3.000   95.0
```

- `r/s w/s` = 초당 IOPS

- `await` = 평균 응답시간 ms (HDD 10ms / SSD 1ms 기준)
- **`%util` 100%** = 디스크 포화 (I/O 병목)

응용 예제 ③ — sar 시계열

```
# 백그라운드 누적 (sysstat 패키지가 자동)
sudo systemctl enable --now sysstat

# 5 분 평균 24 개 (=2 시간)
sar -u 300 24      # CPU
sar -r 300 24      # 메모리
sar -b 300 24      # 디스크
sar -n DEV 300 24  # 네트워크

# 어제 기록
sar -f /var/log/sysstat/sa$(date -d yesterday +%d)

# 1 주일치 CPU 일평균
for i in {1..7}; do sar -u -f /var/log/sysstat/sa$(date -d "$i days ago" +%d);
done
```

응용 예제 ④ — dstat (모든 걸 한 화면)

```
sudo apt install -y dstat
dstat -tcmdn          # 시간·CPU·메모리·디스크·NW
dstat --top-cpu --top-mem --top-io   # Top 프로세스 같이
```

관전 포인트

1. 부하 발생 시 **`vmstat`의 `r`** 증가 ↔ **`htop`의 CPU%** 증가
2. **`iostat`의 `%util`** 100% 이면 무엇 해도 느림
3. **`sar`** 백그라운드 누적이 사후 분석의 핵심
4. **`swap si/so`** 가 0 아니면 메모리 부족 (스왑인/아웃 발생)
5. `dstat` 은 한 화면에 모두 — 발표·데모에 최고

자주 만나는 오류

- `iostat: command not found` → `apt install sysstat`
- `sar` 이 빈 출력 → `sudo systemctl enable --now sysstat` (수집기 활성화)
- htop 의 색 안 보임 → `TERM=xterm-256color`

운영 팁

- 4 축 동시 관찰을 **녹화 (asciinema)** 해 사고 사후 검증
- `pidstat -d` 로 어느 PID 가 디스크 먹는지
- Prometheus + node_exporter 로 장기 운영 (lab19)

HANDS-ON LAB

실습 #10

Wireshark + tcpdump 패킷 분석

[권장]

IT 인프라 아키텍처 설계

Hands-on Labs · 20 시나리오

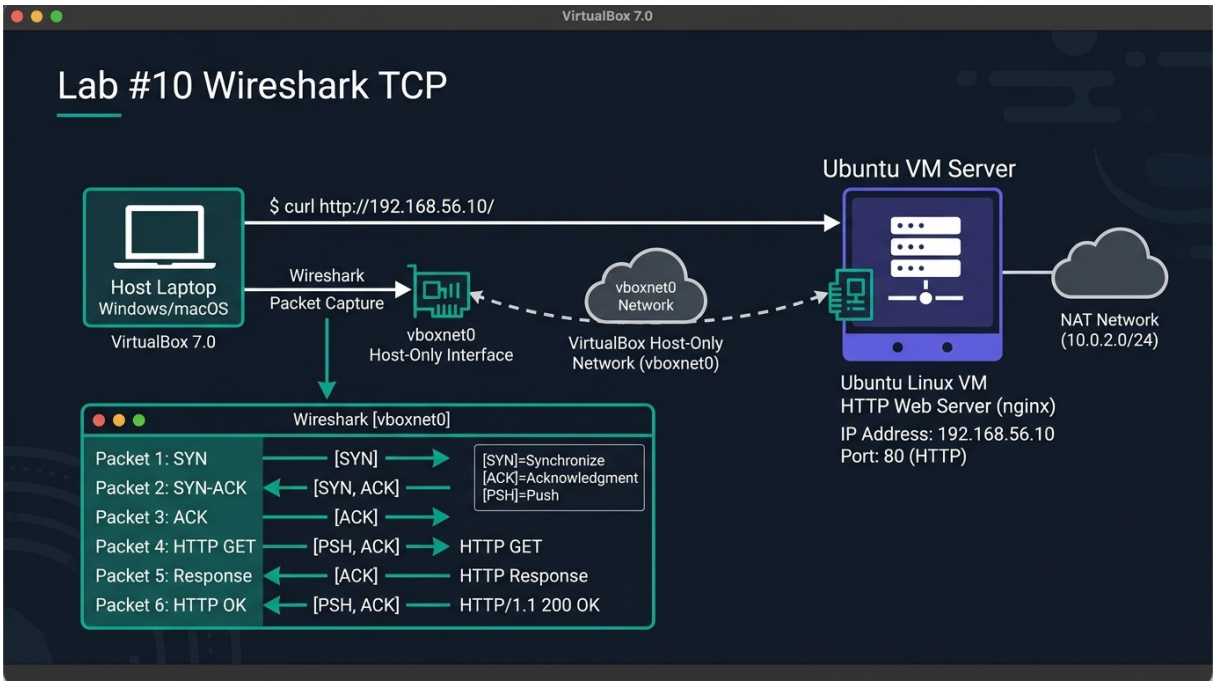
GenAI Labs

강사 박수현 · CEO / CTO

실습 목적

- **TCP 3-way handshake** 패킷 단위 관찰
- **tcpdump (CLI)** + **Wireshark (GUI)** 병행 사용
- **Capture filter (BPF)** vs **Display filter (Wireshark)** 구문 차이 이해
- 자주 쓰는 필터 패턴 — 호스트 · 출발지 · 목적지 · 포트 · 프로토콜 · 다중 조건
- 운영 네트워크 분석 1 차 도구 정착

아키텍처 구성 (Topology)



위 토폴로지를 머리에 그린 상태로 아래 실습을 진행한다.

도구

- **tcpdump** (CLI · 모든 Unix 기본 내장 · BPF 구문)
- **Wireshark** (GUI · wireshark.org/download.html · BPF + Display filter)
- **tshark** (Wireshark CLI 버전 · SSH 환경에서 유용)
- `curl` / `ping` (트래픽 발생)

시나리오 (약 25 분)

```
# 1) 도구 설치
sudo apt install -y wireshark tshark tcpdump
sudo usermod -aG wireshark $USER     # 비특권 캡처
# 재로그인 필요

# 2) 인터페이스 확인
ip -br link                          # 활성 NIC 목록
tcpdump -D                           # tcpdump 인터페이스 번호
tshark -D                            # tshark 도 동일

# 3) 가장 기본 캡처 — 모든 인터페이스, 100 패킷
sudo tcpdump -i any -c 100

# 4) 파일로 저장 (Wireshark UI 에서 열기 위해)
sudo tcpdump -i any -w /tmp/cap.pcap "tcp port 80" -c 200
# (다른 창에서 트래픽 발생)
curl -v http://example.com/
ping -c 4 google.com

# 5) Wireshark GUI 로 열기
wireshark /tmp/cap.pcap &

# 6) tshark CLI 로 분석 (SSH 환경)
tshark -r /tmp/cap.pcap -Y "tcp.flags.syn==1 && tcp.flags.ack==0"
tshark -r /tmp/cap.pcap -q -z conv,tcp      # TCP 대화 통계
tshark -r /tmp/cap.pcap -q -z io,stat,1     # 초당 패킷 수
```

핵심 — Capture filter (BPF) vs Display filter (Wireshark)

항목	Capture filter (BPF)	Display filter (Wireshark)
사용처	tcpdump · Wireshark "Capture options"	Wireshark 상단 필터 바 · tshark `-Y`
적용 시점	캡처 중 (커널 BPF)	캡처 후 표시
구문	`host`, `src`, `dst`, `and`, `or`, `not`	`ip.addr`, `&&`, `
속도	빠름 (커널 필터)	느림 (디스플레이 필터)

장단점	작성 단순·검색 어려움	표현력 풍부·후처리 가능
언제 쓰나	대용량 캡처 시 미리 거름	캡처본 분석 시

자주 쓰는 필터 — 양쪽 구문 비교

1. 특정 호스트만

```
# Capture filter (tcpdump)
sudo tcpdump -i any 'host 192.168.1.10'

# Display filter (Wireshark)
ip.addr == 192.168.1.10
```

2. 출발지만 / 목적지만

```
# Capture filter
sudo tcpdump -i any 'src host 192.168.1.10'    # 출발지만
sudo tcpdump -i any 'dst host 192.168.1.10'    # 목적지만

# Display filter
ip.src == 192.168.1.10        # 출발지만
ip.dst == 192.168.1.10        # 목적지만
```

3. 포트별

```
# Capture filter
sudo tcpdump -i any 'port 80'           # 양방향 80
sudo tcpdump -i any 'src port 80'       # 출발지 포트만
sudo tcpdump -i any 'dst port 443'      # 목적지 포트만
sudo tcpdump -i any 'portrange 1000-2000'  # 범위

# Display filter
tcp.port == 80                # TCP 80 (양방향)
tcp.srcport == 80             # TCP 출발 포트만
tcp.dstport == 443            # TCP 목적 포트만
udp.port == 53                # UDP 53 (DNS)
```

4. 프로토콜별

```
# Capture filter
sudo tcpdump -i any 'tcp'               # TCP 만
sudo tcpdump -i any 'udp'               # UDP 만
sudo tcpdump -i any 'icmp'              # ICMP (ping)
sudo tcpdump -i any 'arp'               # ARP
sudo tcpdump -i any 'tcp port 80'       # TCP + 포트 결합

# Display filter
```

```
tcp                             # TCP 만
udp                             # UDP 만
icmp                            # ICMP
arp                             # ARP
http                            # HTTP (디코딩됨 — 매우 유용)
dns                             # DNS (디코딩됨)
tls                             # TLS
```

5. 다중 조건 (and/or/not)

```
# Capture filter
sudo tcpdump -i any 'host 10.0.0.1 and port 22'
sudo tcpdump -i any 'host 10.0.0.1 or host 10.0.0.2'
sudo tcpdump -i any 'tcp and not port 22'

# Display filter
ip.addr == 10.0.0.1 && tcp.port == 22
ip.addr == 10.0.0.1 || ip.addr == 10.0.0.2
tcp && !(tcp.port == 22)
```

6. TCP 플래그 — Wireshark 가 압도적으로 편함

```
# Capture filter (어려움 — 비트마스크)
sudo tcpdump -i any 'tcp[tcpflags] & tcp-syn != 0'     # SYN 포함
sudo tcpdump -i any 'tcp[tcpflags] & (tcp-rst|tcp-fin) != 0'  # RST or FIN

# Display filter (쉬움)
tcp.flags.syn == 1                      # SYN
tcp.flags.syn == 1 && tcp.flags.ack == 0 # 첫 SYN (3-way 시작)
tcp.flags.ack == 1                      # ACK
tcp.flags.reset == 1                    # RST (비정상 종료)
tcp.flags.fin == 1                      # FIN (정상 종료)
```

7. 응용 — HTTP / DNS / TLS

```
# Capture filter (포트 기준만)
sudo tcpdump -i any 'tcp port 80 or tcp port 443'
sudo tcpdump -i any 'udp port 53'

# Display filter (페이로드 디코딩)
http.request.method == "POST"           # POST 요청
http.request.uri contains "/login"       # URL 필터
http.host == "example.com"
dns.qry.name == "google.com"
dns.flags.response == 1                 # DNS 응답만
tls.handshake.type == 1                 # TLS Client Hello
```

8. 비정상 진단 — 운영 1 순위

```
# Display filter (Wireshark 전용)
tcp.analysis.retransmission              # 재전송
tcp.analysis.duplicate_ack               # 중복 ACK (혼잡 신호)
tcp.analysis.zero_window                 # Zero Window (수신 버퍼 풀)
tcp.analysis.out_of_order                # 순서 어긋남
tcp.window_size < 1000                   # 윈도우 작아짐
icmp.type == 3                           # ICMP Destination Unreachable
```

관전 포인트 (5 종)

1. **3-way: SYN → SYN-ACK → ACK** 패킷 #1·#2·#3 순서 확인
2. **Seq·Ack 번호** — Seq 0 → Ack 1 (ISN, Initial Sequence Number)
3. **Window Size** — 받을 수 있는 버퍼 (혼잡 제어 시작점)
4. **Flow Graph** — `Statistics → Flow Graph` 시간순 시각화
5. **재전송 패턴** — `tcp.analysis.retransmission` 한 줄로 진단

자주 만나는 오류

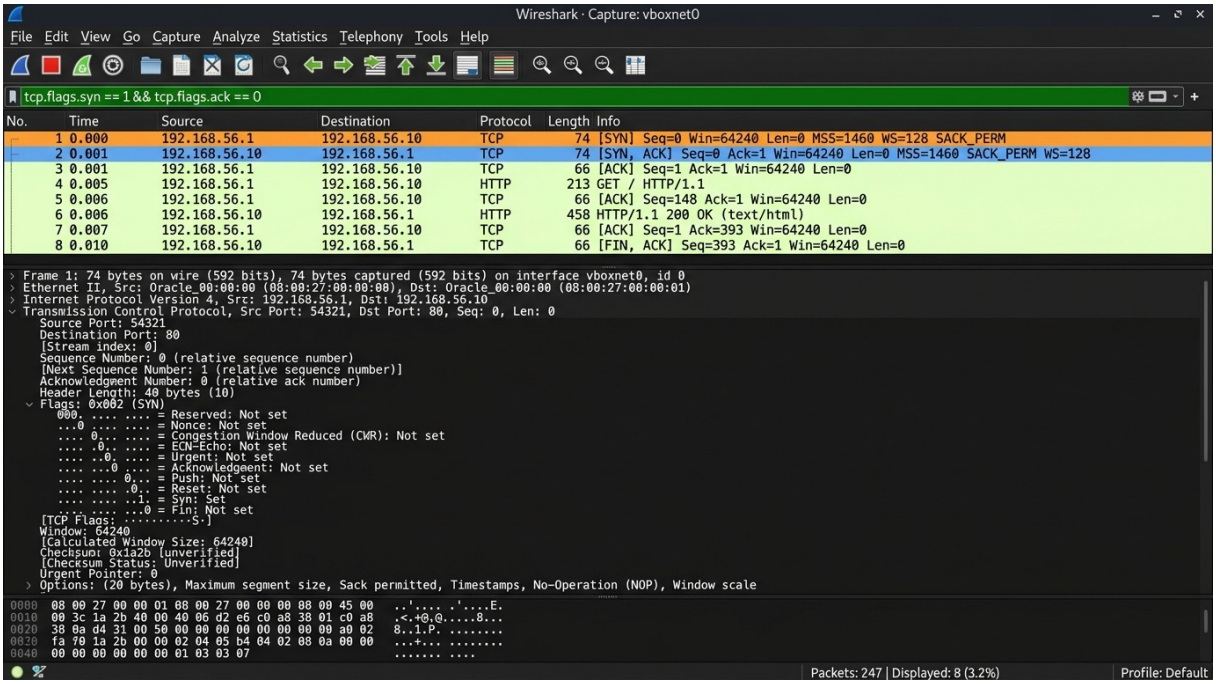
- 캡처 권한 → `sudo usermod -aG wireshark $USER` + 재로그인
- 패킷 0 개 → 인터페이스 확인 (`tcpdump -D`, `-i any` 사용)
- HTTPS 페이로드 안 보임 → TLS 키 필요 (`SSLKEYLOGFILE=/tmp/sslkeys.log curl ...` 후 Wireshark 에서 Preferences → Protocols → TLS → (Pre)-Master-Secret log filename 지정)
- **Display filter 와 Capture filter 구문 다름** — `host 1.2.3.4` (BPF) vs `ip.addr == 1.2.3.4` (display)
- tcpdump 대용량 캡처 시 디스크 풀 → `-W 5 -G 60` (60 초·5 개 회전)
- tshark CLI 에서 `-Y` (display filter) vs `-f` (capture filter) 옵션 헷갈림

운영 팁

- **대용량 캡처 회전**: `sudo tcpdump -i any -w cap_%Y%m%d_%H%M.pcap -G 300 -W 12 'port 443'` (5 분 회전, 1 시간 보관)
- **링 버퍼**: `-C 100 -W 10` (100MB × 10 파일 = 1GB)
- **시간 스탬프 정밀도**: `-tttt` (절대시각 마이크로초)
- **헤더만**: `-s 96` (snaplen 96B — 전체 페이로드 안 받음, 디스크 절약)
- **promiscuous 끄기**: `-p` (자기 트래픽만)

최종 결과 화면 (Reference)

본 실습 완료 시 보일 것으로 예상되는 대시보드/UI 예시.



부록 — 설정 파일

01_bpf_examples.txt

```
# lab10 Capture Filter (BPF) 예시 모음
# 사용처: tcpdump · Wireshark "Capture Options" 화면 · tshark -f
# 적용 시점: 캡처 중 (커널에서 필터링 → 빠름, 페이로드 디코딩 불가)

# ─── 호스트 / 출발지·목적지 ──────────────────────────────────────────────
host 192.168.1.10
src host 192.168.1.10
dst host 192.168.1.10
host 192.168.1.10 and host 10.0.0.5
not host 192.168.1.10

# ─── 네트워크 ────────────────────────────────────────────────────────────
net 192.168.1.0/24
src net 10.0.0.0/8
dst net 172.16.0.0/12

# ─── 포트 ────────────────────────────────────────────────────────────────
port 80
src port 80
dst port 443
portrange 1000-2000
port 80 or port 443

# ─── 프로토콜 ────────────────────────────────────────────────────────────
tcp
udp
icmp
arp
ip6                                 # IPv6
ether proto 0x88cc                  # LLDP 등 raw EtherType

# ─── 조합 ────────────────────────────────────────────────────────────────
tcp port 80
udp port 53
host 10.0.0.1 and port 22
host 10.0.0.1 or host 10.0.0.2
tcp and not port 22
```

```
'(tcp port 80 or tcp port 443) and host www.example.com'

# ── 플래그 (비트마스크 — 어려움) ──────────────────────────
'tcp[tcpflags] & tcp-syn != 0'                      # SYN 포함
'tcp[tcpflags] & (tcp-syn|tcp-ack) == tcp-syn'      # 첫 SYN (ACK 없음)
'tcp[tcpflags] & (tcp-rst|tcp-fin) != 0'            # RST or FIN

# ── ICMP 타입 ──────────────────────────────────────────────
'icmp[icmptype] == icmp-echo'                        # ping 요청
'icmp[icmptype] == icmp-echoreply'                   # ping 응답
'icmp[icmptype] == icmp-unreach'                     # destination unreachable

# ── 패킷 사이즈 ────────────────────────────────────────────
greater 1500                        # 1500 B 초과
less 100                            # 100 B 미만

# ── VLAN / MPLS ────────────────────────────────────────────
vlan
vlan 100
mpls

# ── 자주 쓰는 운영 패턴 ────────────────────────────────────
'tcp port 80 or tcp port 443'                        # 웹 트래픽
'udp port 53'                                        # DNS
'tcp port 22 and host 10.0.0.5'                      # 특정 서버 SSH
'host www.example.com and port 443'                  # HTTPS 특정 사이트
'not (host 10.0.0.1 and port 22)'                    # 내 SSH 빼고
```

02_display_filter_examples.txt

```
# lab10 Display Filter (Wireshark) 예시 모음
# 사용처: Wireshark 상단 필터 바 · tshark -Y
# 적용 시점: 캡처 후 표시단 (페이로드 디코딩 가능 — http / dns / tls / kerberos)

# ── 호스트 / IP ────────────────────────────────────────────
ip.addr == 192.168.1.10
ip.src == 192.168.1.10
ip.dst == 192.168.1.10
ip.addr == 192.168.1.10 || ip.addr == 10.0.0.5
!(ip.addr == 192.168.1.10)
ip.src in {192.168.1.10, 192.168.1.20, 192.168.1.30}

# ── 네트워크 / 서브넷 ──────────────────────────────────────
ip.addr == 192.168.1.0/24
```

```
ip.src == 10.0.0.0/8

# ─── 포트 ──────────────────────────────────────────────────────────────────────
tcp.port == 80
tcp.srcport == 80
tcp.dstport == 443
udp.port == 53
tcp.port in {80, 443, 8080}

# ─── 프로토콜 (페이로드 디코딩) ──────────────────────────────────
tcp
udp
icmp
arp
http
http2
dns
tls
ssh
smb / smb2
kerberos
quic
ntp
mqtt

# ─── TCP 플래그 ──────────────────────────────────────────────────────────────
tcp.flags.syn == 1                              # SYN
tcp.flags.syn == 1 && tcp.flags.ack == 0        # 첫 SYN (3-way 시작)
tcp.flags.syn == 1 && tcp.flags.ack == 1        # SYN-ACK
tcp.flags.ack == 1
tcp.flags.fin == 1                              # FIN (정상 종료)
tcp.flags.reset == 1                            # RST (비정상)
tcp.flags.push == 1

# ─── TCP 비정상 / 분석 ──────────────────────────────────────────────────────
tcp.analysis.retransmission                     # 재전송
tcp.analysis.fast_retransmission                # 빠른 재전송
tcp.analysis.duplicate_ack                      # 중복 ACK
tcp.analysis.zero_window                        # Zero Window
tcp.analysis.window_full                        # 윈도우 가득
tcp.analysis.out_of_order                       # 순서 어긋남
tcp.analysis.lost_segment                       # 세그먼트 손실
tcp.analysis.keep_alive
tcp.window_size < 1000
tcp.time_delta > 1                              # 1 초 이상 패킷 간격
```

```
# ─── HTTP ──────────────────────────────────────────────────────
http.request                                    # 요청만
http.response                                   # 응답만
http.request.method == "POST"
http.request.method == "GET"
http.request.uri contains "/login"
http.request.uri matches "^/api/v[12]/"
http.host == "example.com"
http.user_agent contains "curl"
http.response.code == 500
http.response.code >= 400
http.referer contains "google"
http.content_type contains "json"

# ─── DNS ───────────────────────────────────────────────────────
dns.qry.name == "google.com"
dns.qry.name contains "naver"
dns.qry.type == 1                               # A   (1=A, 28=AAAA, 15=MX, 16=TXT,
5=CNAME)
dns.qry.type == 28                              # AAAA
dns.flags.response == 0                         # 질의만
dns.flags.response == 1                         # 응답만
dns.flags.rcode != 0                            # 응답 에러 (3=NXDOMAIN)

# ─── TLS ───────────────────────────────────────────────────────
tls.handshake.type == 1                         # Client Hello
tls.handshake.type == 2                         # Server Hello
tls.handshake.type == 11                        # Certificate
tls.record.version == 0x0303                    # TLS 1.2
tls.handshake.extensions_server_name == "example.com"

# ─── ICMP ──────────────────────────────────────────────────────
icmp                                            # 전체
icmp.type == 0                                  # echo reply
icmp.type == 3                                  # destination unreachable
icmp.type == 8                                  # echo request (ping)
icmp.type == 11                                 # TTL exceeded (traceroute)

# ─── ARP ───────────────────────────────────────────────────────
arp
arp.opcode == 1                                 # request
arp.opcode == 2                                 # reply

# ─── 비교 연산자
──────────────────────────────────────────────────────────────────────────

frame.len > 1000                                # 크기
frame.time_delta > 1
ip.ttl < 10                                     # TTL 작음
tcp.window_size < 1000
```

```
# ─── 다중 조건 ──────────────────────────────────────────────────────────────
ip.addr == 10.0.0.1 && tcp.port == 22
ip.addr == 10.0.0.1 || ip.addr == 10.0.0.2
tcp && !(tcp.port == 22)
http.request.method == "POST" && http.host == "example.com"

# ─── 운영 1-liner ──────────────────────────────────────────────────────────
# IP 별 SYN 갯수 (포트 스캔 / SYN flood 탐지)
# tshark -r FILE -Y 'tcp.flags.syn==1 && tcp.flags.ack==0' -T fields -e ip.src |
sort | uniq -c | sort -rn

# HTTP 응답 코드별 분포
# tshark -r FILE -q -z http,tree

# URI 별 호출 수
# tshark -r FILE -q -z http_req,tree

# TCP 대화 별 바이트
# tshark -r FILE -q -z conv,tcp
```

HANDS-ON LAB

실습 #11

네트워크 성능 측정 + 지연·손실 시뮬레이션 (iperf3 + tc)

[참고]

IT 인프라 아키텍처 설계

Hands-on Labs · 20 시나리오

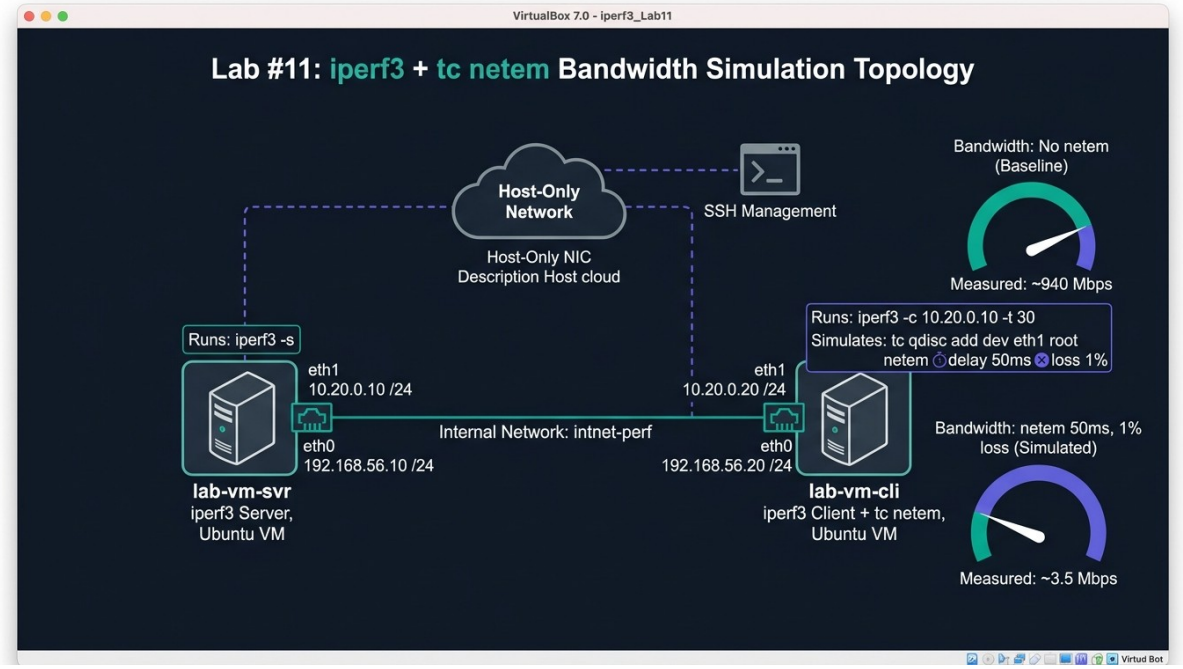
GenAI Labs

강사 박수현 · CEO / CTO

실습 목적

- **iperf3** 로 두 호스트간 대역폭·지연·지터 측정
- **`tc qdisc netem`** 으로 지연·손실·재정렬 시뮬레이션
- TCP/UDP 모드, 병렬 스트림, JSON 출력
- 운영 환경 회선 점검 + 응용 SLA 검증 시나리오

아키텍처 구성 (Topology)



위 토폴로지를 머리에 그린 상태로 아래 실습을 진행한다.

도구

- `iperf3` (서버 + 클라이언트 한 바이너리)
- `tc` (Traffic Control, iproute2 기본 내장)
- `netem` (Network Emulator, kernel module)

기본 예제 — VM 2 대 환경

VM 둘을 Internal Network 로 연결 (10.20.0.10 = 서버, 10.20.0.20 = 클라이언트).

```
sudo apt install -y iperf3

# 서버 측 (10.20.0.10)
iperf3 -s              # 5201 포트 리스닝

# 클라이언트 측 (10.20.0.20)
iperf3 -c 10.20.0.10                  # 10 초 TCP 측정 (기본)
iperf3 -c 10.20.0.10 -t 30            # 30 초
iperf3 -c 10.20.0.10 -u -b 100M       # UDP 100Mbps
iperf3 -c 10.20.0.10 -P 4             # 4 병렬 스트림
iperf3 -c 10.20.0.10 -R               # 역방향 (다운로드 측정)
iperf3 -c 10.20.0.10 --json > out.json  # JSON 출력
```

응용 예제 ① — tc netem 지연 시뮬

```
# 클라이언트 eth1 에 50ms 지연 부가
sudo tc qdisc add dev eth1 root netem delay 50ms

# 측정
iperf3 -c 10.20.0.10 -t 10
# 50ms 지연 → TCP 대역폭 자동 감소 (Bandwidth-Delay Product)

# 지터 추가 (±10ms)
sudo tc qdisc change dev eth1 root netem delay 50ms 10ms distribution normal

# 손실 1% 추가
sudo tc qdisc change dev eth1 root netem delay 50ms loss 1%

# 재정렬 25% 추가
sudo tc qdisc change dev eth1 root netem delay 50ms reorder 25% 50%

# 모두 해제
sudo tc qdisc del dev eth1 root
```

응용 예제 ② — 대역폭 제한 (tbf)

```
# 100Mbit 제한 (Token Bucket Filter)
sudo tc qdisc add dev eth1 root tbf rate 100mbit burst 32kbit latency 400ms

# 1Gbit 로 변경
sudo tc qdisc change dev eth1 root tbf rate 1gbit burst 256kbit latency 200ms

# 해제
sudo tc qdisc del dev eth1 root
```

응용 예제 ③ — 운영 SLA 검증 시나리오

```
# 시나리오: 서울 ↔ 부산 (RTT 25ms, 패킷 손실 0.1%) 환경에서
# 응용이 견디는지 검증

sudo tc qdisc add dev eth1 root netem delay 25ms 5ms distribution normal loss 0.1%
iperf3 -c 10.20.0.10 -t 60 -i 5

# 결과로 응용 SLA (예: 동영상 1080p = 5Mbps 필요) 충족 여부 판단
# 실제 운영 회선 (RTT·loss) 적용해 SLA Risk 평가
```

관전 포인트

1. **TCP 대역폭은 RTT × loss 제곱근에 반비례** (Mathis 공식)
2. **iperf3 UDP** 가 진정한 회선 능력 (TCP 는 알고리즘에 영향)
3. **`-P 4`** 병렬은 단일 흐름 제한 우회
4. `tc netem` 은 가상 회선을 거의 실제와 같이 재현

자주 만나는 오류

- `iperf3: error - unable to connect` → 서버 측 `-s` 실행·방화벽 (`ufw allow 5201`)
- `tc: RTNETLINK answers: Operation not supported` → `modprobe sch_netem`
- 측정값 매번 다름 → ECMP 멀티 경로 또는 무선 (WiFi 1st hop 지연)

운영 팁

- **연속 모니터**: cron + iperf3 -t 5 매시간 + Grafana
- **TCP·UDP 둘 다** 측정 (UDP 는 회선 한계, TCP 는 응용 한계)
- 운영 환경에선 짧게 (`-t 5`) — 트래픽 영향 최소화

부록 — 설정 파일

01_iperf3_options.txt

```
# lab11 iperf3 옵션 모음

# ── 서버 ──────────────────────────────────────────────────────────────────────
iperf3 -s                               # 5201 리스닝
iperf3 -s -p 5202                       # 포트 변경
iperf3 -s -D                            # 데몬 모드
iperf3 -s -B 10.20.0.10                 # 특정 IP 만 바인드
iperf3 -s -J                            # JSON 출력

# ── 클라이언트 기본 ──────────────────────────────────────────────────────────
iperf3 -c SERVER                        # 10 초 TCP 기본
iperf3 -c SERVER -t 30                  # 30 초
iperf3 -c SERVER -t 60 -i 5             # 60 초, 5 초마다 리포트
iperf3 -c SERVER -p 5202                # 포트 지정

# ── 병렬 / 양방향 / 역방향 ──────────────────────────────────────────────────
iperf3 -c SERVER -P 4                   # 4 병렬 스트림
iperf3 -c SERVER -R                     # Reverse (다운로드 측정)
iperf3 -c SERVER --bidir                # 동시 양방향

# ── UDP ──────────────────────────────────────────────────────────────────────
iperf3 -c SERVER -u -b 100M             # UDP 100 Mbps
iperf3 -c SERVER -u -b 1G               # UDP 1 Gbps (회선 한계 측정)
iperf3 -c SERVER -u -b 0                # UDP 무제한 (보낼 수 있는 만큼)

# ── 윈도우 / MSS / DSCP / Cubic ──────────────────────────────────────────────
iperf3 -c SERVER -w 1M                  # TCP 윈도우 사이즈
iperf3 -c SERVER -M 1400                # MSS
iperf3 -c SERVER -S 0x10                # DSCP (low delay)
iperf3 -c SERVER --congestion bbr       # BBR 사용
iperf3 -c SERVER --congestion cubic     # Cubic (기본)

# ── 출력 / 후처리 ────────────────────────────────────────────────────────────
iperf3 -c SERVER --json > out.json
iperf3 -c SERVER --logfile /var/log/iperf.log
```

```
# ─── jq 후처리 1-liner ──────────────────────────────────────
# TCP 처리량 (Mbps)
jq '.end.sum_received.bits_per_second / 1e6' out.json

# 재전송 횟수
jq '.end.sum_received.retransmits' out.json

# UDP 손실율
jq '.end.sum.lost_percent' out.json

# UDP 지터 (ms)
jq '.end.sum.jitter_ms' out.json

# ─── 운영 모니터 (cron) ─────────────────────────────────────
# 매시간 5 초 측정 + 처리량 로깅
# 0 * * * *  iperf3 -c SERVER -t 5 --json | jq '.end.sum_received.bits_per_second'
>> /var/log/iperf_mbps.log
```

02_tc_netem_tbf.txt

```
# lab11 tc qdisc 패턴 모음 (netem · tbf · htb)

# ─── 현재 상태
──────────────────────────────────────────────────────────────────────────────

tc qdisc show                           # 전체
tc qdisc show dev eth0                  # 인터페이스 별
tc -s qdisc show dev eth0               # 통계 포함 (sent / drop / overlimits)
tc -d qdisc show dev eth0               # 상세 (파라미터)

# ─── netem (Network Emulator) ──────────────────────────────
# 지연
sudo tc qdisc add dev eth0 root netem delay 50ms
sudo tc qdisc add dev eth0 root netem delay 50ms 10ms                       # ±지터
sudo tc qdisc add dev eth0 root netem delay 50ms 10ms distribution normal   # 정규분
포
sudo tc qdisc add dev eth0 root netem delay 50ms 10ms 25%                   # 상관

# 손실
sudo tc qdisc change dev eth0 root netem loss 1%
sudo tc qdisc change dev eth0 root netem loss random 1% 25%                 # 25%
는 상관
sudo tc qdisc change dev eth0 root netem loss state 1% 99% 100% 0%          #
state model
sudo tc qdisc change dev eth0 root netem loss gemodel 1% 10% 70% 0.1%       #
Gilbert-Elliott

# 중복 / 변조 / 재정렬
```

```
sudo tc qdisc change dev eth0 root netem duplicate 1%
sudo tc qdisc change dev eth0 root netem corrupt 0.1%
sudo tc qdisc change dev eth0 root netem reorder 25% 50%

# 조합
sudo tc qdisc add dev eth0 root netem delay 25ms 5ms distribution normal loss 0.1%

# ── tbf (Token Bucket Filter) — 대역폭 제한 ──────────────────────────
sudo tc qdisc add dev eth0 root tbf rate 100mbit burst 32kbit latency 400ms
sudo tc qdisc change dev eth0 root tbf rate 1gbit burst 256kbit latency 200ms
sudo tc qdisc change dev eth0 root tbf rate 10mbit burst 16kbit latency 200ms

# ── HTB (Hierarchical Token Bucket) — 클래스별 분배 ──────────────────
sudo tc qdisc add dev eth0 root handle 1: htb default 12
sudo tc class add dev eth0 parent 1: classid 1:1 htb rate 100mbit
sudo tc class add dev eth0 parent 1:1 classid 1:11 htb rate 30mbit ceil 100mbit
sudo tc class add dev eth0 parent 1:1 classid 1:12 htb rate 70mbit ceil 100mbit
# 필터: 192.168.1.10 의 트래픽은 1:11 로
sudo tc filter add dev eth0 protocol ip parent 1: prio 1 u32 match ip src
192.168.1.10 flowid 1:11

# ── tbf + netem 조합 (대역폭 + 지연) ─────────────────────────────────
sudo tc qdisc add dev eth0 root handle 1: tbf rate 100mbit burst 32kbit latency
400ms
sudo tc qdisc add dev eth0 parent 1:1 handle 10: netem delay 25ms 5ms

# ── 원복
──────────────────────────────────────────────────────────────────────────
──────
sudo tc qdisc del dev eth0 root
sudo tc qdisc del dev eth0 ingress

# ── ingress (들어오는 트래픽) ────────────────────────────────────────
sudo tc qdisc add dev eth0 handle ffff: ingress
sudo tc filter add dev eth0 parent ffff: u32 match ip protocol 1 0xff police rate
1mbit burst 10k drop

# ── kernel module ────────────────────────────────────────────────────
sudo modprobe sch_netem
sudo modprobe sch_tbf
sudo modprobe sch_htb
lsmod | grep sch_

# ── 현실 회선 프로파일 (delay JITTER loss) ──────────────────────────
# 서울↔부산    25ms  5ms  0.1%
# 서울↔도쿄    35ms 10ms  0.05%
# 서울↔미서부  130ms 20ms  0.2%
# 서울↔미동부  200ms 30ms  0.3%
# LTE 모바일   50ms 30ms  0.5%
# 저궤도 위성  30ms 20ms  1%
```

```
# 정지궤도 위성  550ms 50ms  0.5%
# 사내 LAN          1ms  0.5ms 0.001%
# WiFi (혼잡)      20ms 15ms  1%
```

HANDS-ON LAB

실습 #12

HAProxy L7 LB (ACL·sticky·SSL termination·stats)

[권장]

IT 인프라 아키텍처 설계

Hands-on Labs · 20 시나리오

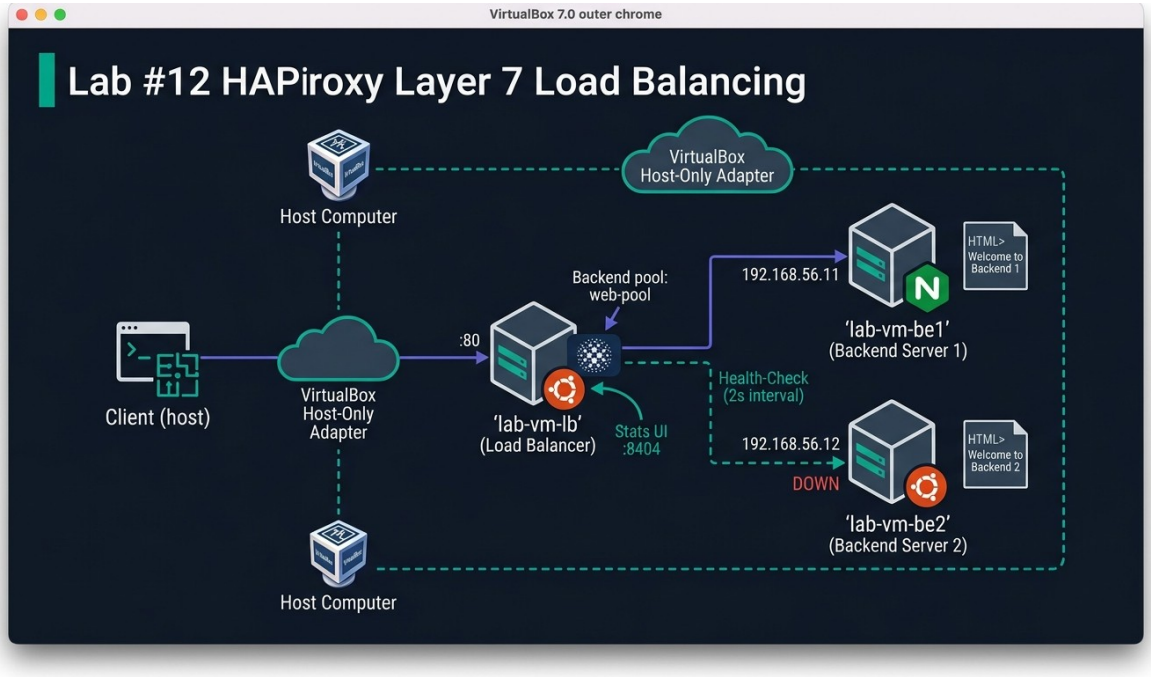
GenAI Labs

강사 박수현 · CEO / CTO

실습 목적

- **HAProxy 2.6+** L4/L7 로드밸런서 설정
- **헬스체크** + **failover** 시연
- **ACL** 라우팅 (URL·헤더·쿠키)
- **sticky session** (cookie/IP)
- **SSL termination** + 백엔드 평문
- **stats UI** 운영 표준 활용

아키텍처 구성 (Topology)



위 토폴로지를 머리에 그린 상태로 아래 실습을 진행한다.

도구

- VM 3 대: lb (192.168.56.10) + be1 (.11) + be2 (.12) — Host-Only Network
- HAProxy 2.6+
- 백엔드 mock: Python http.server

기본 예제 — 가장 단순한 LB

```
# /etc/haproxy/haproxy.cfg
global
    log /dev/log local0
    maxconn 4096

defaults
    log global
    mode http
    option httplog
    option dontlognull
    timeout connect 5s
    timeout client 30s
    timeout server 30s

frontend web_in
    bind *:80
    default_backend web_pool

backend web_pool
    balance roundrobin
    option httpchk GET /healthz
    server be1 192.168.56.11:80 check inter 2s fall 3 rise 2
    server be2 192.168.56.12:80 check inter 2s fall 3 rise 2
```

```
sudo systemctl restart haproxy
for i in {1..10}; do curl -s http://192.168.56.10/; done
# be1 / be2 / be1 / be2 ... 순환
```

응용 예제 ① — ACL 라우팅

```
frontend web_in
    bind *:80
    # ACL 정의
    acl is_api   path_beg /api/
    acl is_admin path_beg /admin/
    acl is_static path_end .css .js .png .jpg
    acl from_office src 10.10.0.0/16

    # 백엔드 라우팅
    use_backend api_pool   if is_api
    use_backend admin_pool if is_admin from_office
```

```
    use_backend static_pool if is_static
    default_backend web_pool

backend api_pool
    server api1 10.20.0.30:8080 check
    server api2 10.20.0.31:8080 check

backend admin_pool
    server admin 10.20.0.40:8080 check

backend static_pool
    server cdn 10.30.0.50:80 check
```

응용 예제 ② — Sticky Session

```
# 쿠키 기반 sticky
backend web_pool
    balance roundrobin
    cookie SERVERID insert indirect nocache
    server be1 192.168.56.11:80 check cookie be1
    server be2 192.168.56.12:80 check cookie be2

# IP 해시 (쿠키 못 쓰는 환경)
backend web_pool
    balance source
    hash-type consistent
    server be1 192.168.56.11:80 check
    server be2 192.168.56.12:80 check
```

응용 예제 ③ — SSL termination

```
frontend web_https
    bind *:443 ssl crt /etc/haproxy/certs/api.pem alpn h2,http/1.1
    redirect scheme https if !{ ssl_fc }
    http-request set-header X-Forwarded-Proto https

    default_backend web_pool

backend web_pool
    server be1 192.168.56.11:80 check
    server be2 192.168.56.12:80 check

# 인증서 결합 (cert + key 한 파일로)
sudo cat api.lab.crt api.lab.key > /etc/haproxy/certs/api.pem
sudo chmod 600 /etc/haproxy/certs/api.pem
```

응용 예제 ④ — Stats UI

```
listen stats
    bind *:8404
    mode http
```

```
    stats enable
    stats uri /
    stats refresh 5s
    stats auth admin:secret
    stats admin if TRUE       # 백엔드 enable/disable 가능
```

브라우저: `http://192.168.56.10:8404/` (admin/secret)

응용 예제 ⑤ — 헬스체크 + 가중치

```
backend web_pool
    option httpchk GET /healthz HTTP/1.1\r\nHost:\ api.lab
    http-check expect status 200
    server be1 192.168.56.11:80 weight 3 check inter 2s fall 3 rise 2
    server be2 192.168.56.12:80 weight 1 check inter 2s fall 3 rise 2
    server be3 192.168.56.13:80 backup check     # 둘 다 죽으면만
```

관전 포인트

1. **healthcheck** + `fall 3 rise 2` → 운영 표준 (1 번 죽음 ≠ 죽음 판정)
2. **ACL** 은 path / header / src / method / 등 모든 것 기반 가능
3. **stats UI** 가 운영 시각화의 1 순위
4. **`balance` 종류**: roundrobin·leastconn·source·uri·random
5. **SSL termination** 후 백엔드는 평문 — 내부망에서 만

자주 만나는 오류

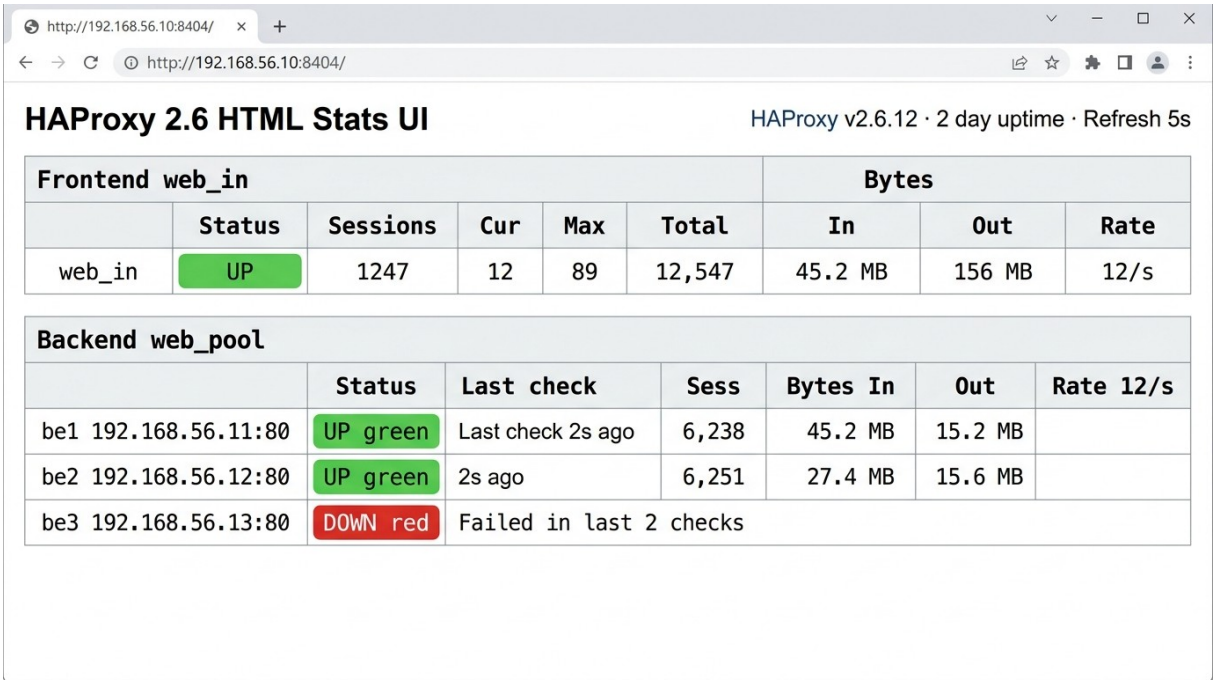
- `haproxy[xxx]: ALERT: starting proxy ...: cannot bind socket` → 포트 충돌
- `502 Bad Gateway` → 백엔드 모두 죽음 (`http-check` 응답 확인)
- `503 Service Unavailable` → 응답 너무 느림 (`timeout server` 늘리기)
- `crt /path/api.pem cannot be opened` → cert+key 한 파일로 결합

운영 팁

- **무중단 reload**: `systemctl reload haproxy` (`restart` 아님)
- **로그**: `local0` → `/var/log/haproxy.log` (`rsyslog` 설정 필요)
- **모니터링**: Prometheus haproxy_exporter
- **Keepalived 와 페어** (lab13) → HAProxy 자체 SPOF 제거

최종 결과 화면 (Reference)

본 실습 완료 시 보일 것으로 예상되는 대시보드/UI 예시.



부록 — 설정 파일

01_haproxy_basic.cfg

```
## 기본 LB — roundrobin + healthcheck + stats UI
global
    daemon
    maxconn 4096
    log /dev/log local0

defaults
    mode http
    timeout connect 5s
    timeout client 30s
    timeout server 30s
    option httplog

frontend lab_fe
    bind *:80
    default_backend lab_be

backend lab_be
    balance roundrobin
    option httpchk GET /health
    http-check expect status 200
    server be1 127.0.0.1:8001 check fall 3 rise 2
    server be2 127.0.0.1:8002 check fall 3 rise 2

listen stats
    bind *:8404
    stats enable
    stats uri /
    stats refresh 5s
```

02_haproxy_acl_routing.cfg

```
## 응용① — ACL 라우팅 (path / src / extension 기반)
global
    daemon
    maxconn 4096
defaults
    mode http
    timeout connect 5s
    timeout client 30s
    timeout server 30s
    option httplog

frontend web_in
    bind *:80
```

```
    # ACL 정의
    acl is_api    path_beg /api/
    acl is_admin  path_beg /admin/
    acl is_static path_end .css .js .png .jpg
    acl from_office src 127.0.0.0/8 192.168.0.0/16

    # 백엔드 라우팅
    use_backend api_pool    if is_api
    use_backend admin_pool  if is_admin from_office
    use_backend static_pool if is_static
    default_backend web_pool

backend api_pool
    server api1 127.0.0.1:8001 check
backend admin_pool
    server admin 127.0.0.1:8002 check
backend static_pool
    server cdn 127.0.0.1:8001 check
backend web_pool
    balance roundrobin
    server be1 127.0.0.1:8001 check
    server be2 127.0.0.1:8002 check

listen stats
    bind *:8404
    stats enable
    stats uri /
```

03_haproxy_sticky.cfg

```
## 응용 ② — Sticky Session (cookie 또는 source IP 해시)
global
    daemon
defaults
    mode http
    timeout connect 5s
    timeout client 30s
    timeout server 30s
    option httplog

frontend lab_fe
    bind *:80
    default_backend lab_be

# (A) 쿠키 기반 sticky — HAProxy 가 SERVERID 쿠키 삽입
backend lab_be
    balance roundrobin
    cookie SERVERID insert indirect nocache
    server be1 127.0.0.1:8001 check cookie be1
    server be2 127.0.0.1:8002 check cookie be2
```

```
# (B) source IP 해시 (대체) — 쿠키 못 쓰는 환경
#backend lab_be
#    balance source
#    hash-type consistent
#    server be1 127.0.0.1:8001 check
#    server be2 127.0.0.1:8002 check

listen stats
    bind *:8404
    stats enable
    stats uri /
```

04_haproxy_ssl.cfg

```
## 응용 ③ — SSL Termination + HTTP→HTTPS 리다이렉트
## 사전: /etc/haproxy/certs/api.pem 에 cert + key 결합 필요 (setup_04_ssl.sh 가 생성)
global
    daemon
defaults
    mode http
    timeout connect 5s
    timeout client 30s
    timeout server 30s
    option httplog

frontend web_http
    bind *:80
    # HTTP 는 모두 HTTPS 로 강제 redirect
    http-request redirect scheme https code 301 unless { ssl_fc }

frontend web_https
    bind *:443 ssl crt /etc/haproxy/certs/api.pem alpn h2,http/1.1
    http-request set-header X-Forwarded-Proto https
    default_backend web_pool

backend web_pool
    balance roundrobin
    # SSL termination 후 백엔드는 평문 (내부망 전제)
    server be1 127.0.0.1:8001 check
    server be2 127.0.0.1:8002 check

listen stats
    bind *:8404
    stats enable
    stats uri /
```

05_haproxy_weight_backup.cfg

```
## 응용 ⑤ — 가중치 + backup (헬스체크 강화)
global
```

```
    daemon
defaults
    mode http
    timeout connect 5s
    timeout client 30s
    timeout server 30s
    option httplog

frontend lab_fe
    bind *:80
    default_backend lab_be

backend lab_be
    # 강화된 헬스체크 — Host 헤더 명시
    option httpchk GET /health HTTP/1.1\r\nHost:\ api.lab
    http-check expect status 200
    server be1 127.0.0.1:8001 weight 3 check inter 2s fall 3 rise 2   # 3배
    server be2 127.0.0.1:8002 weight 1 check inter 2s fall 3 rise 2   # 1배
    server be3 127.0.0.1:8003 backup  check inter 2s fall 3 rise 2    # 둘 다 죽으면만

listen stats
    bind *:8404
    stats enable
    stats uri /
```

HANDS-ON LAB

실습 #13

Keepalived VRRPv3 Active-Standby + VIP 페일오버

[권장]

IT 인프라 아키텍처 설계

Hands-on Labs · 20 시나리오

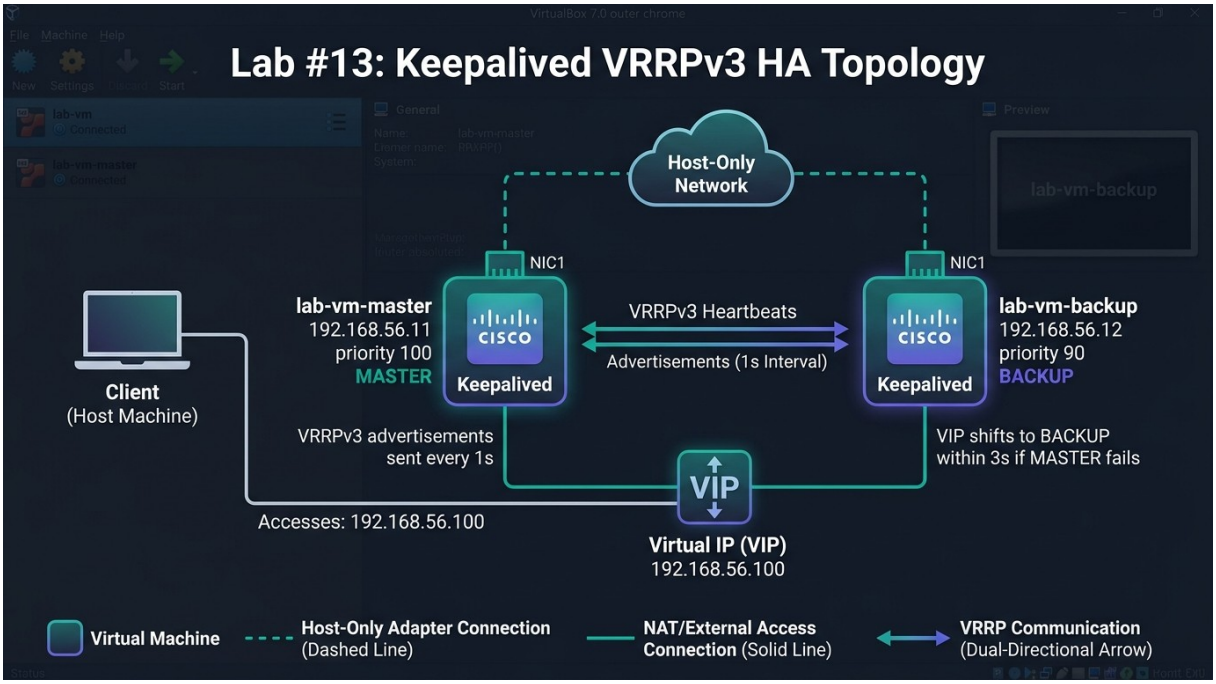
GenAI Labs

강사 박수현 · CEO / CTO

실습 목적

- **VRRPv3** (Virtual Router Redundancy Protocol)로 Floating IP (VIP) 관리
- **2-node Active-Standby** — Master 죽으면 3 초 내 VIP 이동
- HAProxy + Keepalived 페어로 LB SPOF 제거
- 헬스체크 + notify 스크립트로 페일오버 자동화

아키텍처 구성 (Topology)



위 토폴로지를 머리에 그린 상태로 아래 실습을 진행한다.

도구

- VM 2 대: master (192.168.56.11) + backup (.12) — Host-Only
- VIP: 192.168.56.100 (둘 사이 떠다님)
- Keepalived 2.2+

기본 예제

```
sudo apt install -y keepalived

# master 측 /etc/keepalived/keepalived.conf
vrrp_instance VI_1 {
    state MASTER
    interface enp0s8
    virtual_router_id 51
    priority 100              # ← Master 는 높게
    advert_int 1              # 1 초 간격
    authentication {
        auth_type PASS
        auth_pass labpass
    }
    virtual_ipaddress {
        192.168.56.100/24
    }
}

# backup 측 — priority 만 다르게
vrrp_instance VI_1 {
    state BACKUP
    interface enp0s8
    virtual_router_id 51
    priority 90               # ← Backup 은 낮게
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass labpass
    }
    virtual_ipaddress {
        192.168.56.100/24
    }
}

# 양쪽 다 시작
sudo systemctl enable --now keepalived

# master 에서 VIP 확인
ip addr show enp0s8 | grep 192.168.56.100   # 보임

# backup 에서
```

```
ip addr show enp0s8 | grep 192.168.56.100   # 안 보임

# 호스트에서 VIP 로 ping
ping 192.168.56.100      # OK

# master kill → 3 초 안에 backup 이 VIP 가져감
sudo systemctl stop keepalived       # master
# backup 에서 ip addr 확인 — VIP 보임
```

응용 예제 ① — track_script (서비스 단위 페일오버)

```
vrrp_script chk_nginx {
    script "/usr/bin/pgrep nginx"
    interval 2
    fall 3
    rise 2
    weight -20             # nginx 죽으면 priority -20
}

vrrp_instance VI_1 {
    state MASTER
    ...
    track_script {
        chk_nginx
    }
}
```

효과: master VM 은 살아 있어도 nginx 가 죽으면 backup 으로 VIP 이전.

응용 예제 ② — notify 스크립트 (페일오버 알림)

```
vrrp_instance VI_1 {
    ...
    notify_master /etc/keepalived/notify_master.sh
    notify_backup /etc/keepalived/notify_backup.sh
    notify_fault  /etc/keepalived/notify_fault.sh
}

# /etc/keepalived/notify_master.sh
#!/bin/bash
logger -t keepalived "→ MASTER 전환"
curl -X POST -d "Master: $(hostname)" https://hooks.slack.com/...
systemctl start haproxy

# notify_backup.sh
#!/bin/bash
logger -t keepalived "→ BACKUP 전환"
systemctl stop haproxy
```

응용 예제 ③ — Unicast (멀티캐스트 차단 환경)

```
vrrp_instance VI_1 {
    state MASTER
    interface enp0s8
    virtual_router_id 51
    priority 100
    advert_int 1

    unicast_src_ip 192.168.56.11
    unicast_peer {
        192.168.56.12
    }

    virtual_ipaddress {
        192.168.56.100/24
    }
}
```

클라우드·일부 스위치는 멀티캐스트 차단 → unicast 모드 권장.

응용 예제 ④ — HAProxy + Keepalived 페어

```
   client → 192.168.56.100 (VIP, 떠다님)
                ↓
   ┌──────────────────────────┐
   │ Active LB (master)       │ keepalived + haproxy
   │ 192.168.56.11            │
   └──────────────────────────┘
                ↕  VRRP advert (1s)
   ┌──────────────────────────┐
   │ Standby LB (backup)      │ keepalived + haproxy
   │ 192.168.56.12            │
   └──────────────────────────┘
```

- 양쪽 LB 모두 같은 HAProxy 설정
- Keepalived 가 VIP 관리, HAProxy 가 백엔드 분산
- LB 자체 SPOF 제거

관전 포인트

1. `priority` 차이가 → master/backup 결정 (높은 게 이김)
2. `advert_int 1` + `fall 3` → 3 초 안에 전환
3. **VRRPv3 멀티캐스트** = 224.0.0.18 (방화벽 차단 시 unicast 사용)
4. **track_script + notify** 로 서비스 단위 페일오버 + 알림 자동화
5. **MAC 주소도 같이 이동** (VRRP virtual MAC)

자주 만나는 오류

- VIP 양쪽 다 보임 → **Split-brain**, 네트워크 분리 점검 (advert 안 닿음)
- VIP 안 보임 → `auth_pass` 불일치 또는 `virtual_router_id` 다름
- 페일오버 안 됨 → `chk_*` 스크립트 종료 코드 확인 (0 = 정상)
- 클라우드에서 VRRP 안 됨 → 멀티캐스트 차단, unicast 모드 사용

운영 팁

- **3-node**: VRRP 는 2-node 가 일반적, 더 필요하면 Pacemaker (lab18)
- **로깅**: `journalctl -u keepalived -f`
- **STONITH** 부재 → 양쪽 살아 있을 때 Split-brain 위험 (notify 로 fence)
- **AWS·GCP**: ENI/Floating IP API 로 Keepalived 대체

부록 — 설정 파일

01_keepalived_basic_backup.conf

```
## 기본 — BACKUP 노드 (priority 100). MASTER 는 01_basic_master.
vrrp_instance VI_1 {
    state BACKUP
    interface eth0
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass lab123
    }
    virtual_ipaddress {
        192.168.56.100/24
    }
}
```

01_keepalived_basic_master.conf

```
## 기본 — MASTER 노드 (priority 150). BACKUP 은 02 파일 참조.
vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 150
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass lab123
    }
    virtual_ipaddress {
        192.168.56.100/24
    }
}
```

02_keepalived_track_script.conf

```
## 응용 ① — track_script: nginx 죽으면 priority -20 → BACKUP 으로 VIP 자동 이전
vrrp_script chk_nginx {
    script "/usr/bin/pgrep nginx"
    interval 2                  # 2 초마다 검사
    fall 3                       # 3 번 실패 = 다운 판정
    rise 2                       # 2 번 성공 = 복구 판정
```

```
    weight -20                  # 다운 시 priority -20
}

vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 150
    advert_int 1
    authentication { auth_type PASS; auth_pass lab123 }
    virtual_ipaddress { 192.168.56.100/24 }
    track_script {
        chk_nginx
    }
}
```

03_keepalived_notify.conf

```
## 응용 ② — notify 스크립트: state 변경 시 외부 알림 + 서비스 제어
## notify_master/backup/fault 스크립트는 setup_03_notify.sh 가 생성

vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 150
    advert_int 1
    authentication { auth_type PASS; auth_pass lab123 }
    virtual_ipaddress { 192.168.56.100/24 }

    notify_master /etc/keepalived/notify_master.sh
    notify_backup /etc/keepalived/notify_backup.sh
    notify_fault  /etc/keepalived/notify_fault.sh
}
```

04_keepalived_unicast.conf

```
## 응용 ③ — Unicast 모드 (클라우드·일부 스위치는 multicast 차단)
## MASTER 측 (BACKUP 은 src/peer 만 교환)
vrrp_instance VI_1 {
    state MASTER
    interface enp0s8
    virtual_router_id 51
    priority 100
    advert_int 1

    # multicast 대신 unicast — peer IP 명시
    unicast_src_ip 192.168.56.11
    unicast_peer {
        192.168.56.12
```

```
    }

    virtual_ipaddress {
        192.168.56.100/24
    }
}
```

05_keepalived_haproxy_pair.conf

```
## 응용 ④ — HAProxy + Keepalived 페어 (LB SPOF 제거)
## HAProxy 죽으면 priority -30 → BACKUP 노드로 VIP 이전
vrrp_script chk_haproxy {
    script "/usr/bin/pgrep haproxy"
    interval 2
    fall 3
    rise 2
    weight -30
}

vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 150
    advert_int 1
    authentication { auth_type PASS; auth_pass lab123 }
    virtual_ipaddress { 192.168.56.100/24 }
    track_script {
        chk_haproxy
    }
}
```

HANDS-ON LAB

실습 #14

PostgreSQL pgbench (TPC-B 부하 + 튜닝)

[참고]

IT 인프라 아키텍처 설계

Hands-on Labs · 20 시나리오

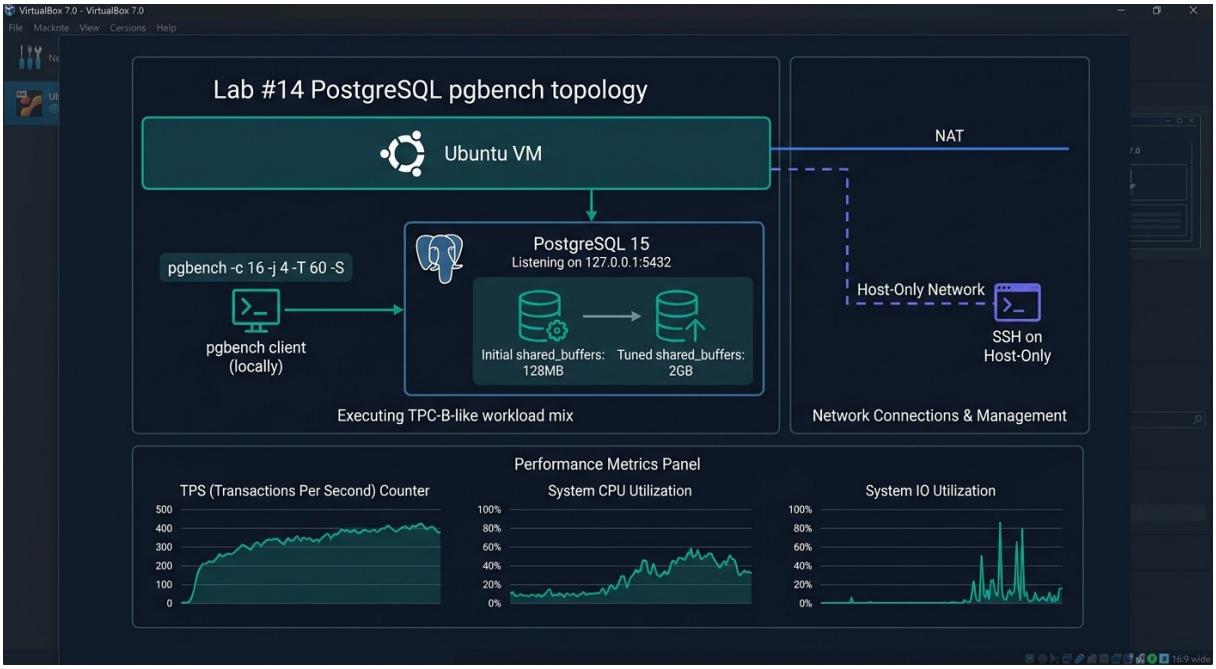
GenAI Labs

강사 박수현 · CEO / CTO

실습 목적

- **pgbench** 로 TPS 측정 → 튜닝 후 재측정 (Before/After)
- `shared_buffers` `work_mem` `effective_cache_size` 핵심 3 종
- TPC-B 표준 워크로드 + custom SQL
- Lock/Slow Query 분석

아키텍처 구성 (Topology)



위 토폴로지를 머리에 그린 상태로 아래 실습을 진행한다.

도구

- PostgreSQL 16
- pgbench (postgresql-contrib 에 포함)
- pg_stat_statements (slow query)

기본 예제

```
sudo apt install -y postgresql-16 postgresql-contrib-16

sudo -u postgres psql
CREATE DATABASE bench;
CREATE USER lab WITH PASSWORD 'lab';
GRANT ALL ON DATABASE bench TO lab;
\q

# pgbench 초기화 (scale=50 → 약 750MB)
pgbench -i -s 50 -U lab bench

# TPS 측정 (16 client, 4 thread, 60 초)
pgbench -c 16 -j 4 -T 60 -U lab bench
# tps = 1234.5 (excluding connections establishing)
```

응용 예제 ① — 튜닝 Before/After

```
# Before
pgbench -c 16 -j 4 -T 60 bench
# 예: tps = 800

# /etc/postgresql/16/main/postgresql.conf 수정
shared_buffers = 2GB              # 메모리 25% (기본 128MB)
effective_cache_size = 6GB        # OS 캐시 추정치
work_mem = 16MB                   # join/sort 용
maintenance_work_mem = 512MB
checkpoint_completion_target = 0.9
wal_buffers = 16MB
random_page_cost = 1.1            # SSD 라면 (HDD 4.0)
max_worker_processes = 8
max_parallel_workers_per_gather = 4

sudo systemctl restart postgresql

# After
pgbench -c 16 -j 4 -T 60 bench
# 예: tps = 1800 (2.2 배 향상)
```

응용 예제 ② — pg_stat_statements (slow query)

```
ALTER SYSTEM SET shared_preload_libraries = 'pg_stat_statements';
-- 재시작 필요

CREATE EXTENSION pg_stat_statements;

-- 가장 느린 10 개 쿼리
SELECT query, calls, mean_exec_time, max_exec_time
FROM pg_stat_statements
ORDER BY mean_exec_time DESC LIMIT 10;
```

응용 예제 ③ — Custom workload

```
cat > /tmp/custom.sql <<EOF
\set aid random(1, 100000 * :scale)
BEGIN;
UPDATE pgbench_accounts SET abalance = abalance + 100 WHERE aid = :aid;
COMMIT;
EOF

pgbench -c 8 -j 2 -T 60 -f /tmp/custom.sql bench
```

관전 포인트

1. **`shared_buffers` 25%** → 가장 큰 효과
2. **vmstat 동시 관찰** — wa (I/O 대기) 줄어드는 게 보임
3. **CHECKPOINT 주기**에 TPS 출렁 — `checkpoint_completion_target = 0.9`
4. **autovacuum** 동작 시 잠시 느려짐
5. `pg_stat_statements` 로 어느 쿼리가 느린지 한 줄

자주 만나는 오류

- `peer authentication failed` → `/etc/postgresql/16/main/pg_hba.conf` `peer` → `md5`
- `out of memory` → `work_mem` 너무 큰 값 (사용자×쿼리수만큼 곱)
- `too many connections` → `max_connections` 또는 PgBouncer (커넥션 풀)

운영 팁

- **PgBouncer** 로 커넥션 폭증 차단
- 운영 튜닝 시작점: `pgtune.leopard.in.ua` 자동 생성
- 정기 `VACUUM ANALYZE`

- 백업: `pg_basebackup` + WAL archive

부록 — 설정 파일

02_postgresql_tuned.conf

```
# 실습 #14 ② — postgresql.conf 튜닝 drop-in
# 위치: /etc/postgresql/16/main/conf.d/99-lab-tuning.conf
# 기준: 4 GB RAM VM 가정. 실제 환경은 pgtune.leopard.in.ua 로 계산 권장.

# === 핵심 3 종 ===
# 1) shared_buffers — 전체 RAM 의 약 25%
shared_buffers = 1GB

# 2) effective_cache_size — OS 페이지 캐시 + shared_buffers 추정치 (RAM 의 50~75%)
effective_cache_size = 2GB

# 3) work_mem — 정렬·해시 조인 메모리. 사용자×쿼리수만큼 곱해지므로 보수적으로.
work_mem = 16MB

# === 유지보수·체크포인트 ===
maintenance_work_mem = 256MB
checkpoint_completion_target = 0.9
wal_buffers = 16MB

# === SSD 환경 ===
random_page_cost = 1.1
effective_io_concurrency = 200

# === 병렬 쿼리 ===
max_worker_processes = 4
max_parallel_workers_per_gather = 2
max_parallel_workers = 4
```

03_pg_stat_statements.conf

```
# 실습 #14 ③ — pg_stat_statements 활성화 drop-in
# 위치: /etc/postgresql/16/main/conf.d/95-pg_stat_statements.conf

shared_preload_libraries = 'pg_stat_statements'

# 추적 SQL 종류
pg_stat_statements.track = all      # all | top | none
pg_stat_statements.max  = 10000     # 보관할 distinct query 수
track_io_timing = on                # I/O 시간도 함께 (운영 표준)
```

HANDS-ON LAB

실습 #15

Docker + cAdvisor + cgroup 자원 격리

[참고]

IT 인프라 아키텍처 설계

Hands-on Labs · 20 시나리오

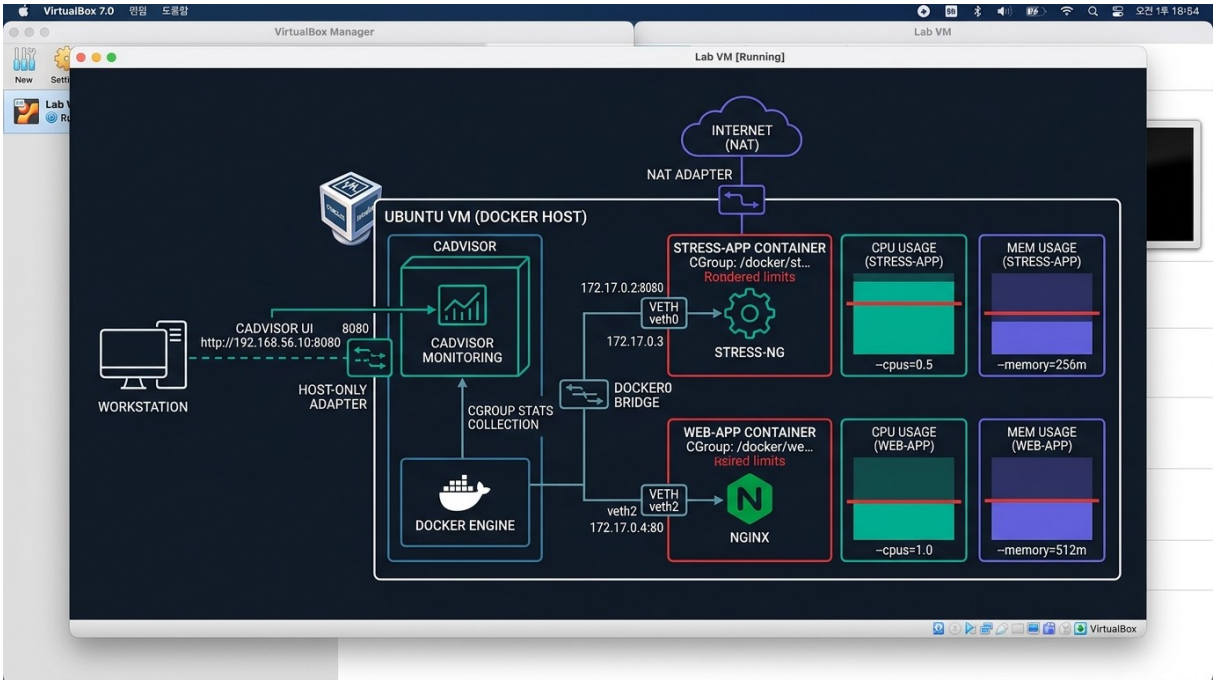
GenAI Labs

강사 박수현 · CEO / CTO

실습 목적

- **cAdvisor** 로 컨테이너 자원 모니터링
- `--cpus` `--memory` `--pids-limit` cgroup 제한
- 컨테이너 OOM 시뮬레이션
- Prometheus 연동 데이터 소스

아키텍처 구성 (Topology)



위 토폴로지를 머리에 그린 상태로 아래 실습을 진행한다.

도구

- Docker (lab03 에서 설치)
- cAdvisor `gcr.io/cadvisor/cadvisor`

기본 예제

```
# cAdvisor 띄우기
docker run -d --name=cadvisor \
    -p 8080:8080 \
    -v /:/rootfs:ro \
    -v /var/run:/var/run:ro \
    -v /sys:/sys:ro \
    -v /var/lib/docker/:/var/lib/docker:ro \
    --privileged \
    --device=/dev/kmsg \
    gcr.io/cadvisor/cadvisor:v0.49.1

# 브라우저 http://localhost:8080
```

응용 예제 ① — cgroup 자원 제한

```
# CPU 0.5 코어 + Mem 256MB 제한
docker run -d --name stressed \
    --cpus=0.5 \
    --memory=256m \
    --memory-swap=256m \
    --pids-limit=100 \
    polinux/stress \
    stress --cpu 4 --vm 1 --vm-bytes 200M

# cgroup 직접 확인
cat /sys/fs/cgroup/system.slice/docker-*.scope/cpu.max
# 50000 100000  ← 0.5 CPU (50000µs / 100000µs)

cat /sys/fs/cgroup/system.slice/docker-*.scope/memory.max
# 268435456  ← 256MB
```

응용 예제 ② — OOM 시뮬

```
# 1GB 메모리 요청, 제한은 256MB → OOM Killed
docker run -d --name oom-test \
    --memory=256m \
    polinux/stress \
    stress --vm 1 --vm-bytes 1G

docker logs oom-test       # 종료 메시지
```

```
docker inspect oom-test --format='{{.State.OOMKilled}}'   # true
dmesg | tail -20           # 커널 OOM killer 로그
```

응용 예제 ③ — Prometheus 연동

```
# Prometheus 띄워 cAdvisor 메트릭 수집
cat > /tmp/prometheus.yml <<EOF
global:
  scrape_interval: 15s
scrape_configs:
  - job_name: 'cadvisor'
    static_configs:
      - targets: ['cadvisor:8080']
EOF

docker run -d --name prom -p 9090:9090 \
    -v /tmp/prometheus.yml:/etc/prometheus/prometheus.yml \
    --network bridge \
    prom/prometheus

# http://localhost:9090 → Status → Targets → cAdvisor UP
```

관전 포인트

1. `--cpus 0.5` = CPU 50% 할당 (CFS 쿼터)
2. `--memory 256m` 초과 시 OOM Killed (자동)
3. **cgroup v2** 가 기본 (Ubuntu 22.04+)
4. cAdvisor UI 에 컨테이너별 CPU/Mem 그래프
5. Prometheus + Grafana 로 장기 시계열

자주 만나는 오류

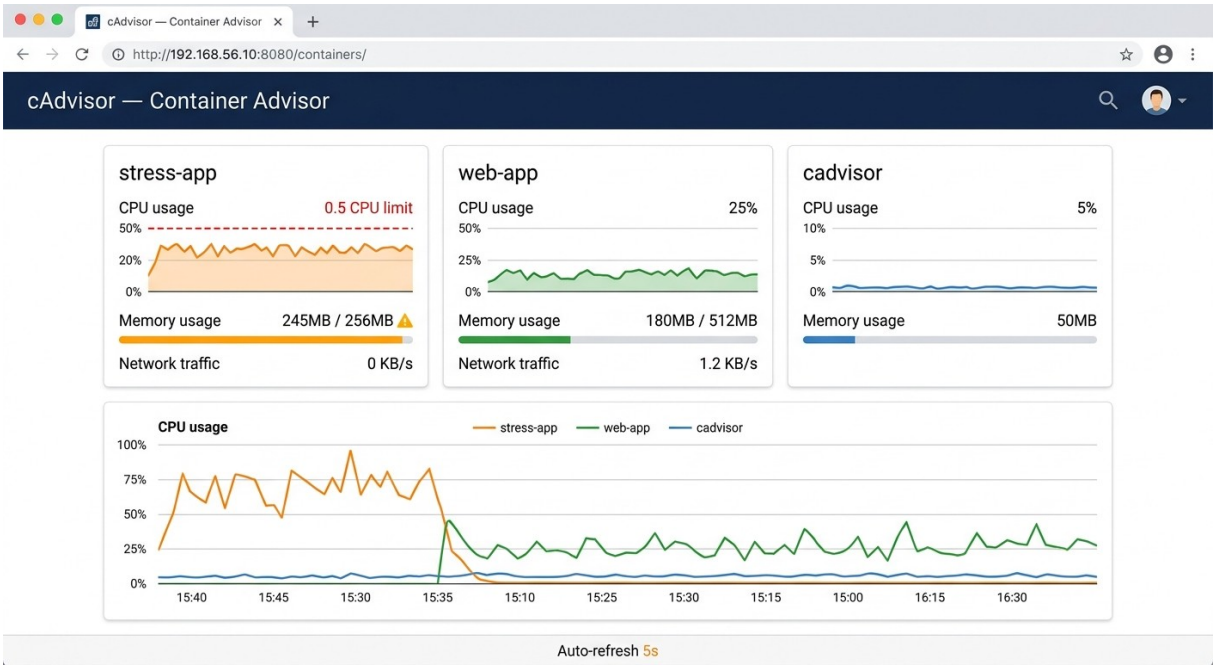
- cAdvisor 컨테이너 OOM → `--privileged` 누락
- cgroup v1 vs v2 혼동 → `stat -fc %T /sys/fs/cgroup/` 로 확인
- Mac 에서 동작 안 함 → Linux VM 필요

운영 팁

- 운영은 Prometheus + node_exporter + cAdvisor 조합
- K8s 환경은 cAdvisor 기본 내장 (kubelet)
- 메모리 swappy 제한: `--memory-swappiness=0`

최종 결과 화면 (Reference)

본 실습 완료 시 보일 것으로 예상되는 대시보드/UI 예시.



HANDS-ON LAB

실습 #16

호스트 방화벽 (iptables + nftables + fail2ban + nmap)

[권장]

IT 인프라 아키텍처 설계

Hands-on Labs · 20 시나리오

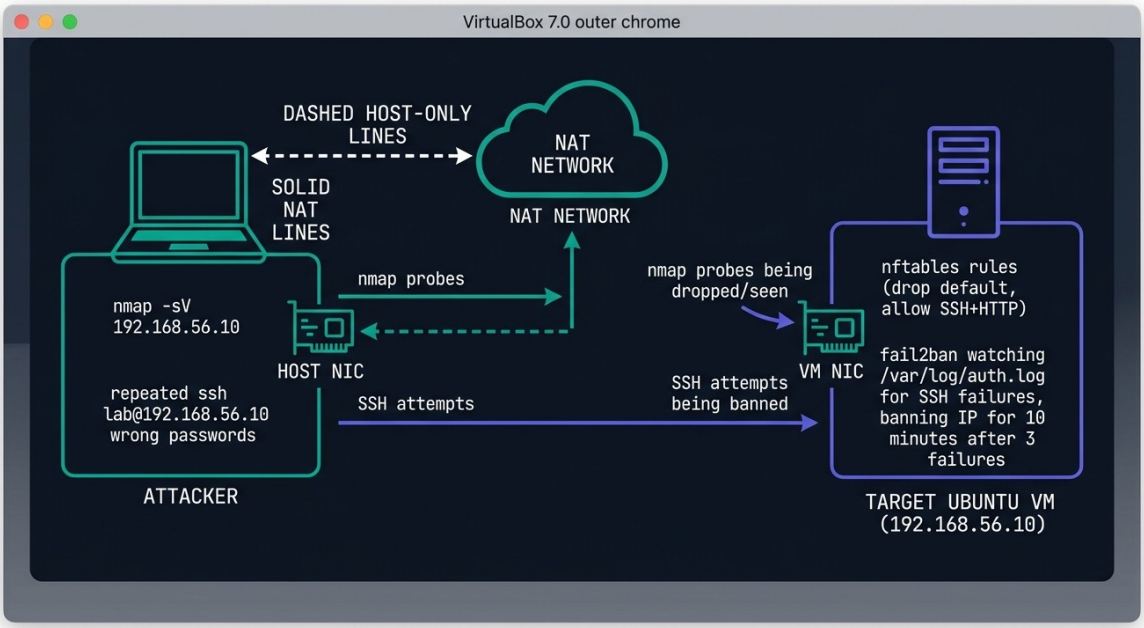
GenAI Labs

강사 박수현 · CEO / CTO

실습 목적

- **iptables** (전통적, 교육 표준) — SI/공공/금융 운영에서 여전히 압도적
- **nftables** (iptables 후속) — Ubuntu 22.04 기본 backend
- **fail2ban** SSH 무차별 대입 자동 차단
- **nmap** 으로 방화벽 효과 검증
- 운영 표준 inbound default-drop + 화이트리스트 패턴

아키텍처 구성 (Topology)



위 토폴로지를 머리에 그린 상태로 아래 실습을 진행한다.

도구

- `iptables` (xtables) + `iptables-save`/`iptables-restore`
- `nft` (nftables CLI) + `nftables.service`
- `fail2ban` + `fail2ban-client`
- `nmap` (공격자 측)
- `conntrack`·`ipset` (보조)

기본 예제 ① — iptables 구조 + 명령

iptables 는 **5 테이블 × 5 체인** 구조:

테이블	용도	주요 체인
`filter`	허용/거부 (기본)	INPUT · FORWARD · OUTPUT
`nat`	주소 변환	PREROUTING · POSTROUTING
`mangle`	헤더 변조	PREROUTING·INPUT·FORWARD·OUTPUT·POSTROUTING
`raw`	conntrack 우회	PREROUTING · OUTPUT
`security`	SELinux 표시	INPUT · FORWARD · OUTPUT

핵심 명령

```
sudo apt install -y iptables iptables-persistent

# 현재 규칙
sudo iptables -L -v -n --line-numbers
sudo iptables -t nat -L -v -n

# 기본 정책 (운영 표준)
sudo iptables -P INPUT DROP
sudo iptables -P FORWARD DROP
sudo iptables -P OUTPUT ACCEPT

# 핵심 룰 (반드시 -P DROP 전에)
sudo iptables -A INPUT -i lo -j ACCEPT
sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 22 -s 192.168.56.0/24 -j ACCEPT
sudo iptables -A INPUT -p tcp -m multiport --dports 80,443 -j ACCEPT
sudo iptables -A INPUT -p icmp -j ACCEPT
sudo iptables -A INPUT -j LOG --log-prefix "DROP: " --log-level 4

# 저장 + 재부팅 후 복원
sudo iptables-save | sudo tee /etc/iptables/rules.v4
sudo ip6tables-save | sudo tee /etc/iptables/rules.v6
```

iptables 자주 쓰는 매처

```
# 출발지·목적지
sudo iptables -A INPUT -s 1.2.3.4 -j DROP
sudo iptables -A INPUT -s 192.168.0.0/16 -j ACCEPT
sudo iptables -A INPUT -d 10.0.0.1 -j ACCEPT

# 포트 패턴
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT                    # 단일
sudo iptables -A INPUT -p tcp --dport 8000:9000 -j ACCEPT             # 범위
sudo iptables -A INPUT -p tcp -m multiport --dports 80,443,8080 -j ACCEPT  # 복수

# 연결 상태
sudo iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
sudo iptables -A INPUT -m conntrack --ctstate NEW -p tcp --dport 22 -j ACCEPT

# Rate limit (SSH 분당 4 회)
sudo iptables -A INPUT -p tcp --dport 22 -m hashlimit \\
    --hashlimit-mode srcip --hashlimit-name ssh \\
    --hashlimit 4/min --hashlimit-burst 4 -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 22 -j DROP
```

iptables NAT (포트 포워딩)

```
# 호스트 8080 → 내부 10.20.0.30:80
sudo iptables -t nat -A PREROUTING -p tcp --dport 8080 -j DNAT --to 10.20.0.30:80
sudo iptables -t nat -A POSTROUTING -j MASQUERADE
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
```

기본 예제 ② — nftables (iptables 후속)

```
sudo apt install -y nftables

# /etc/nftables.conf
table inet filter {
    chain input {
        type filter hook input priority filter; policy drop;
        iif lo accept
        ct state established,related accept
        ip protocol icmp accept
        ip saddr 192.168.56.0/24 tcp dport 22 accept
        tcp dport { 80, 443 } accept
    }
    chain forward {
        type filter hook forward priority filter; policy drop;
    }
    chain output {
        type filter hook output priority filter; policy accept;
    }
}
```

```
# 적용
sudo systemctl enable --now nftables
sudo nft -f /etc/nftables.conf
sudo nft list ruleset
```

iptables ↔ nftables 명령 매핑

iptables	nftables
`iptables -L -v -n`	`nft list ruleset`
`iptables -P INPUT DROP`	`chain input { policy drop; }`
`iptables -A INPUT -p tcp --dport 22 -j ACCEPT`	`add rule inet filter input tcp dport 22 accept`
`iptables -t nat -A PREROUTING -j DNAT`	`add rule ip nat prerouting dnat to ...`
`iptables -F INPUT`	`flush chain inet filter input`
`iptables-save > rules.v4`	`nft list ruleset > nft.rules`
`iptables-restore < rules.v4`	`nft -f nft.rules`

변환 도구

```
sudo apt install -y iptables-nftables-compat

# 단일 룰 변환
iptables-translate -A INPUT -p tcp --dport 22 -j ACCEPT
# → nft add rule ip filter INPUT tcp dport 22 counter accept

# 전체 변환
sudo iptables-save > /tmp/iptables.bak
sudo iptables-restore-translate -f /tmp/iptables.bak > /etc/nftables/migrated.nft
```

응용 예제 ① — fail2ban (자동 차단)

```
sudo apt install -y fail2ban

# /etc/fail2ban/jail.local
[DEFAULT]
bantime  = 10m
findtime = 10m
maxretry = 3
banaction = iptables-multiport
# 또는 nftables-multiport (nftables 사용 시)

[sshd]
enabled = true
port = 22
logpath = %(sshd_log)s
backend = systemd

sudo systemctl restart fail2ban
sudo fail2ban-client status
sudo fail2ban-client status sshd
```

```
# 차단 IP 조회·관리
sudo fail2ban-client get sshd banip
sudo fail2ban-client set sshd unbanip 1.2.3.4
sudo fail2ban-client set sshd banip   1.2.3.4
```

차단 시뮬 (공격자 측)

```
for i in 1 2 3 4; do
    sshpass -p WRONG ssh -o StrictHostKeyChecking=no lab@192.168.56.10
done
# 4 번째에서 연결 안 됨 (fail2ban 차단)

# 서버 측
sudo fail2ban-client status sshd     # Currently banned: 1
sudo iptables -L f2b-sshd -n         # 룰 보기
```

응용 예제 ② — nmap 검증

```
# 공격자 측 (호스트 OS 에서)
nmap -sV 192.168.56.10              # 서비스 + 버전
nmap -p 22,80,443 192.168.56.10     # 특정 포트만
nmap -p- 192.168.56.10              # 전 65535 포트 (느림)
nmap -A 192.168.56.10               # OS + 스크립트
nmap -sS 192.168.56.10              # SYN scan (sudo 필요)
nmap -sU -p 53,123 192.168.56.10    # UDP

# 방화벽 효과 검증
nmap -sV --reason 192.168.56.10
# 열려야 할 포트만 "open" · 나머지 "filtered"
```

응용 예제 ③ — ipset (대량 IP)

```
sudo apt install -y ipset

# 차단 IP 집합
sudo ipset create blacklist hash:ip
sudo ipset add blacklist 1.2.3.4
sudo ipset add blacklist 5.6.7.8

# iptables 에서 한 줄
sudo iptables -A INPUT -m set --match-set blacklist src -j DROP

# 영구화
sudo ipset save > /etc/ipset.conf
```

응용 예제 ④ — conntrack 디버깅

```
sudo apt install -y conntrack

sudo conntrack -L                           # 전체
sudo conntrack -L -p tcp --dport 22         # SSH 만
sudo conntrack -E                           # 이벤트 스트림
sudo conntrack -S                           # 통계

# 테이블 풀이면
sudo sysctl -w net.netfilter.nf_conntrack_max=2097152
```

관전 포인트

1. **iptables 는 여전히 운영의 90%** — 교육 표준
2. **nftables 통합 구문** — `inet` (v4+v6) 등 신구문 강력
3. **`-P INPUT DROP` 전에** SSH 룰 먼저 (자기 차단 방지)
4. **fail2ban + iptables/nftables 페어** = SSH 보안 표준
5. **nmap 의 `open` vs `filtered`** 로 룰 효과 직관 검증

자주 만나는 오류

- 룰 적용 후 SSH 끊김 → BMC 콘솔로 `iptables -F` / `nft flush ruleset`
- iptables ↔ nftables 충돌 → `update-alternatives --config iptables`
- fail2ban 차단 안 됨 → `banaction` backend 와 실제 방화벽 일치 확인
- conntrack table full → `nf_conntrack_max` 증가
- `iptables-legacy` vs `iptables-nft` 모드 차이 → `iptables --version` 확인

운영 팁

- 운영 룰은 **git + Ansible** 로 관리
- 클라우드 SG/NACL 이 호스트 방화벽 위 — 동시 점검
- `nft -j list ruleset` JSON 출력 → 모니터링 자동화
- **iptables-legacy 모드** 유지 vs **iptables-nft 모드** 전환 결정
- 운영에선 fail2ban 외 **CrowdSec** (분산 IP reputation) 검토
- 룰 변경은 `iptables -I` (insert) 보다 명시적 번호로

부록 — 설정 파일

01_nftables.conf

```
#!/usr/sbin/nft -f
# 실습 #16 ① — 기본 호스트 방화벽 (nftables)
flush ruleset

table inet filter {
    chain input {
        type filter hook input priority filter; policy drop;

        # established/related 허용 (이미 맺어진 세션은 통과)
        ct state established,related accept

        # loopback 무조건 허용
        iifname "lo" accept

        # ICMP — rate limit (Ping flood 방어)
        icmp type echo-request limit rate 5/second accept

        # SSH·HTTP·HTTPS
        tcp dport { 22, 80, 443 } accept

        # drop 되는 패킷 로그 (운영 디버깅용)
        log prefix "[nft drop] " level info
    }
    chain forward {
        type filter hook forward priority filter; policy drop;
    }
    chain output {
        type filter hook output priority filter; policy accept;
    }
}
```

02_fail2ban_jail.local

```
## 응용 ① — fail2ban: SSH 무차별 시도 자동 차단
[DEFAULT]
bantime  = 10m              # 차단 유지 시간
findtime = 10m              # 이 시간 안에
maxretry = 3                # 3 번 실패 → 차단
banaction = nftables-multiport
backend = systemd

[sshd]
enabled  = true
```

```
port     = 22
filter   = sshd
logpath  = %(sshd_log)s
```

HANDS-ON LAB

실습 #17

mdadm RAID-5 + 디스크 장애 시뮬

[선택]

IT 인프라 아키텍처 설계

Hands-on Labs · 20 시나리오

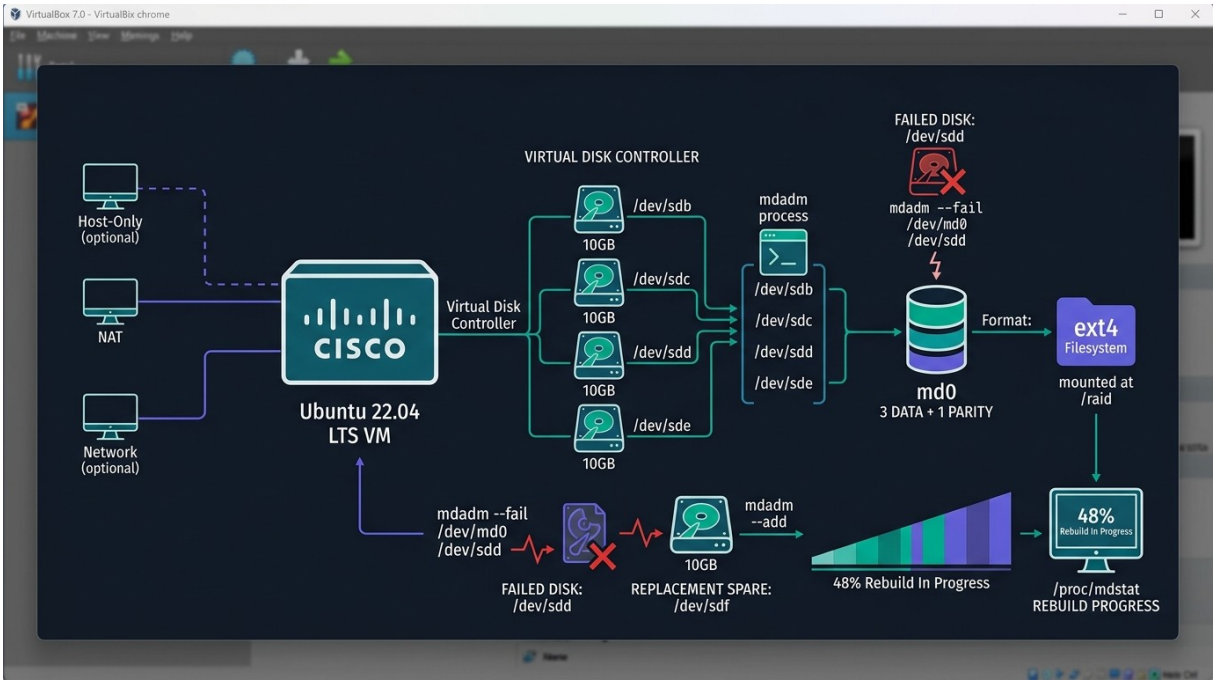
GenAI Labs

강사 박수현 · CEO / CTO

실습 목적

- **mdadm** 으로 소프트웨어 RAID 구성
- RAID-5 (3 data + 1 parity) → 1 대 죽어도 무중단
- 디스크 장애 시뮬 + 핫스페어 교체 + rebuild
- RAID 5/6/10 차이 이해

아키텍처 구성 (Topology)



위 토폴로지를 머리에 그린 상태로 아래 실습을 진행한다.

도구

- VM 1 대에 가상 디스크 5 개 (lab-vm-01 에 sdb~sdf 추가)
- mdadm (Ubuntu 기본)

기본 예제 — RAID-5 4-disk

```
# 디스크 확인
lsblk
# sdb sdc sdd sde sdf (각 10GB)

# RAID-5 생성 (3 data + 1 parity)
sudo mdadm --create /dev/md0 \
    --level=5 \
    --raid-devices=4 \
    /dev/sd[bcde]

# 상태 확인
cat /proc/mdstat
sudo mdadm --detail /dev/md0

# 포맷 + 마운트
sudo mkfs.ext4 /dev/md0
sudo mkdir -p /raid
sudo mount /dev/md0 /raid
df -h /raid    # ~30GB (4 - 1 parity = 3 × 10GB)

# 영구 마운트
sudo mdadm --detail --scan | sudo tee -a /etc/mdadm/mdadm.conf
sudo update-initramfs -u
echo "/dev/md0 /raid ext4 defaults 0 2" | sudo tee -a /etc/fstab
```

응용 예제 ① — 디스크 장애 시뮬

```
# 실제 데이터 쓰기
sudo dd if=/dev/urandom of=/raid/file1.bin bs=1M count=500

# 디스크 1 대 죽었다고 가정
sudo mdadm --fail /dev/md0 /dev/sdd

cat /proc/mdstat
# [_UUU]   ← 1 번 디스크 빠짐, 다른 3 개로 동작 중

# 데이터 접근 계속 됨 (RAID-5 성공)
ls -lh /raid/file1.bin       # 정상
```

```
# 죽은 디스크 제거
sudo mdadm --remove /dev/md0 /dev/sdd
```

응용 예제 ② — 핫스페어 추가 + rebuild

```
# 새 디스크 추가 (또는 핫스페어로 미리)
sudo mdadm --add /dev/md0 /dev/sdf

# rebuild 진행 상황
watch cat /proc/mdstat
# md0 : active raid5 sdf[4] sdb[0] sdc[1] sde[3]
#       30464000 blocks ...
#       [=======>.............]  recovery = 35.2% (3580000/10240000)

# 완료 후
sudo mdadm --detail /dev/md0
# State: clean
```

응용 예제 ③ — RAID 종류 비교

RAID	디스크 사용량	내결함성	읽기	쓰기	권장 사용처
0	100%	0 (없음)			캐시·스크래치 (위험)
1	50%	1 대			OS 부팅 디스크
5	(N-1)/N	1 대			일반 스토리지 (요즘 6 권장)
6	(N-2)/N	2 대			대용량 NAS
10	50%	1~N/2 대			DB·고성능

```
# RAID-10 (4 disk)
sudo mdadm --create /dev/md1 --level=10 --raid-devices=4 /dev/sd[bcde]
# RAID-6 (4 disk)
sudo mdadm --create /dev/md2 --level=6 --raid-devices=4 /dev/sd[bcde]
```

응용 예제 ④ — 모니터링

```
# mdadm monitor (메일 알림)
sudo nano /etc/mdadm/mdadm.conf
# MAILADDR ops@example.com

sudo systemctl enable --now mdmonitor.service

# 수동 점검 (체크섬 일치 확인)
echo check | sudo tee /sys/block/md0/md/sync_action
cat /proc/mdstat       # 진행 상황
```

관전 포인트

1. RAID-5 → 1 대 죽어도 정상 동작 (degraded mode)
2. `--add` → 자동 rebuild 시작
3. **rebuild 중에 또 1 대 죽으면 데이터 손실** (RAID-6 사용 이유)
4. 디스크 크기 다르면 가장 작은 거 기준
5. mdadm 은 소프트웨어 RAID — H/W RAID 카드 vs S/W 비교 필요

자주 만나는 오류

- `mdadm: cannot open /dev/sdb` → 이미 마운트 또는 다른 RAID
- rebuild 매우 느림 → `echo 200000 > /proc/sys/dev/raid/speed_limit_min`
- 부팅 후 RAID 안 뜸 → `/etc/mdadm/mdadm.conf` + `update-initramfs`

운영 팁

- **운영은 RAID-6 또는 RAID-10** (RAID-5 는 대용량 디스크에 위험)
- 매월 sync check 자동화 (cron)
- 디스크 교체는 **같은 모델·같은 크기** 권장
- SMART 모니터링 (`smartctl -a /dev/sdb`)

부록 — 설정 파일

04_mdadm.conf

```
# 실습 #17 ④ — mdadm 모니터링 설정 (append 용)
# 위치: /etc/mdadm/mdadm.conf 끝에 append

# 디스크 자동 검색 범위 (전체 partitions 스캔)
DEVICE partitions

# 알림 수신 주소 — 운영 환경 맞게 수정
MAILADDR ops@example.com
MAILFROM raid-monitor@$(hostname)

# 프로그램 알림 (선택) — 이메일 외에 사용자 스크립트 호출
# PROGRAM /usr/local/sbin/mdadm-event.sh
```

HANDS-ON LAB

실습 #18

Pacemaker + Corosync 2-node 클러스터 HA

[선택]

IT 인프라 아키텍처 설계

Hands-on Labs · 20 시나리오

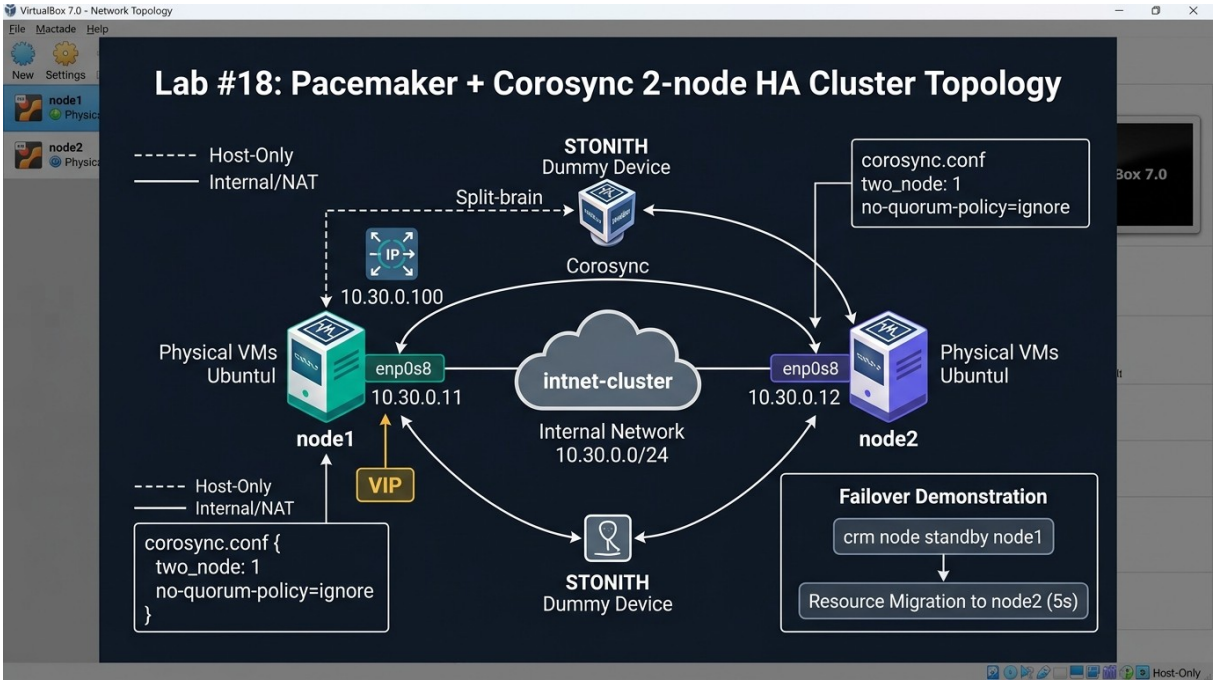
GenAI Labs

강사 박수현 · CEO / CTO

실습 목적

- **Corosync**: 클러스터 멤버십 + 메시징
- **Pacemaker**: 자원 관리 (시작·중지·페일오버)
- 2-node 클러스터에서 Floating IP + 서비스 페일오버
- **Split-brain 방지** (quorum / STONITH)

아키텍처 구성 (Topology)



위 토폴로지를 머리에 그린 상태로 아래 실습을 진행한다.

도구

- VM 2 대 (10.30.0.11, 10.30.0.12) Internal Network
- Pacemaker 2.1+ + Corosync 3+
- Resource Agent: ocf:heartbeat:IPaddr2, systemd

기본 예제 — 클러스터 구축

```
# 양쪽 VM 에서
sudo apt install -y pacemaker pcs corosync

# hacluster 사용자 비밀번호
sudo passwd hacluster

# 양쪽에서 pcsd 시작
sudo systemctl enable --now pcsd

# 한쪽 (node1) 에서 인증 + 클러스터 생성
sudo pcs host auth node1 node2 -u hacluster -p PASS
sudo pcs cluster setup labcluster node1 node2 --start --enable

# 클러스터 상태
sudo pcs status
# Online: [ node1 node2 ]
```

응용 예제 ① — Floating IP 자원

```
sudo pcs resource create vip ocf:heartbeat:IPaddr2 \
    ip=10.30.0.100 \
    cidr_netmask=24 \
    op monitor interval=10s

sudo pcs status      # vip 가 node1 에서 실행 중

# node1 standby → vip 가 node2 로 이동
sudo pcs node standby node1
sudo pcs status      # vip 가 node2

# 다시 정상
sudo pcs node unstandby node1
```

응용 예제 ② — systemd 서비스 페일오버

```
# nginx 를 클러스터 자원으로
sudo pcs resource create webserver systemd:nginx \
    op monitor interval=10s timeout=30s
```

```
# vip + webserver 묶기 (같은 노드에서 동작)
sudo pcs constraint colocation add webserver with vip INFINITY

# 시작 순서 (vip 먼저 → webserver)
sudo pcs constraint order vip then webserver
```

응용 예제 ③ — 2-node quorum 설정

2-node 클러스터는 quorum 이 어려움 (1/2 = no quorum). 설정 필요:

```
# /etc/corosync/corosync.conf
quorum {
    provider: corosync_votequorum
    two_node: 1                    # 2-node 모드 (살아남은 한쪽이 자원 가져감)
    wait_for_all: 1                # 초기 시작 시 양쪽 다 살아야 시작
}
```

응용 예제 ④ — STONITH (Split-brain 방지)

```
# 더미 STONITH (실습용, 운영은 IPMI/iLO 등 진짜 펜싱)
sudo pcs property set stonith-enabled=false   # 실습은 비활성

# 운영은 진짜 펜싱 (예: IPMI)
sudo pcs stonith create fence_node1 fence_ipmilan \
    ipaddr=10.30.0.111 login=admin passwd=admin \
    pcmk_host_list=node1 pcmk_host_check=static-list
```

응용 예제 ⑤ — 자원 의존성·제약

```
# 자원 그룹 (vip + nginx 함께)
sudo pcs resource group add webgroup vip webserver

# 위치 선호 (가능하면 node1)
sudo pcs constraint location vip prefers node1=50

# 자원 모니터 빈도
sudo pcs resource update webserver op monitor interval=5s
```

관전 포인트

1. `pcs status` → 노드/자원 한눈에
2. **Split-brain** = 양쪽 다 master → 두 백엔드 모두 활성 → 데이터 손상
3. **STONITH** = 양쪽 살았다고 판단되면 한쪽 강제 정지
4. `two_node: 1` 없으면 quorum 부족으로 둘 다 정지

5. Keepalived (lab13)보다 복잡하지만 **다양한 자원 타입** 지원

자주 만나는 오류

- 노드 인증 실패 → `pcs host auth` 다시
- 자원 시작 안 됨 → `pcs status` failed actions + `journalctl -u pacemaker`
- Split-brain → 네트워크 분리 점검 (corosync 멀티캐스트)
- STONITH 비활성 경고 → 운영은 무조건 STONITH

운영 팁

- **3-node 권장** (정상 quorum)
- **분리된 corosync ring** 2 개 (이중화)
- 자원 변경은 `crm_resource --cleanup` 후
- DRBD + Pacemaker = 공유 스토리지 없는 HA

최종 결과 화면 (Reference)

본 실습 완료 시 보일 것으로 예상되는 대시보드/UI 예시.

```
root@node1:~# pcs status
Cluster Summary:
  Last updated: Sat Jun  1 14:20:00
  Current DC: node1 (version pacemaker-2.1.4) -- partition with quorum
  Nodes:      2 (configured)
  Resources:  2 instances

Node List:
  Online: [ node1 node2 ]
Active Resources:
  vip (ocf:heartbeat:IPaddr2): Started node1
  webserver (systemd:nginx):   Started node1
PCSD Status:
  node1: Online
  node2: Online
Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: active/enabled
```

부록 — 설정 파일

04_corosync_two_node.conf

```
# 실습 #18 ④ — corosync.conf 의 quorum 블록 패치
# 위치: /etc/corosync/corosync.conf 의 기존 quorum { ... } 블록을 아래로 교체

quorum {
    provider: corosync_votequorum

    # 2-node 모드 — 살아남은 한쪽이 quorum 1 로 자원 가져감
    two_node: 1

    # 초기 부팅 시 양쪽 다 살아야 시작 (split-start 방지)
    wait_for_all: 1

    # last_man_standing: 노드가 하나둘 떨어져도 마지막까지 살아남은 쪽이 quorum 유지
    # last_man_standing: 1
    # last_man_standing_window: 10000
}
```

HANDS-ON LAB

실습 #19

Prometheus + Grafana + Node Exporter 관측성

[선택]

IT 인프라 아키텍처 설계

Hands-on Labs · 20 시나리오

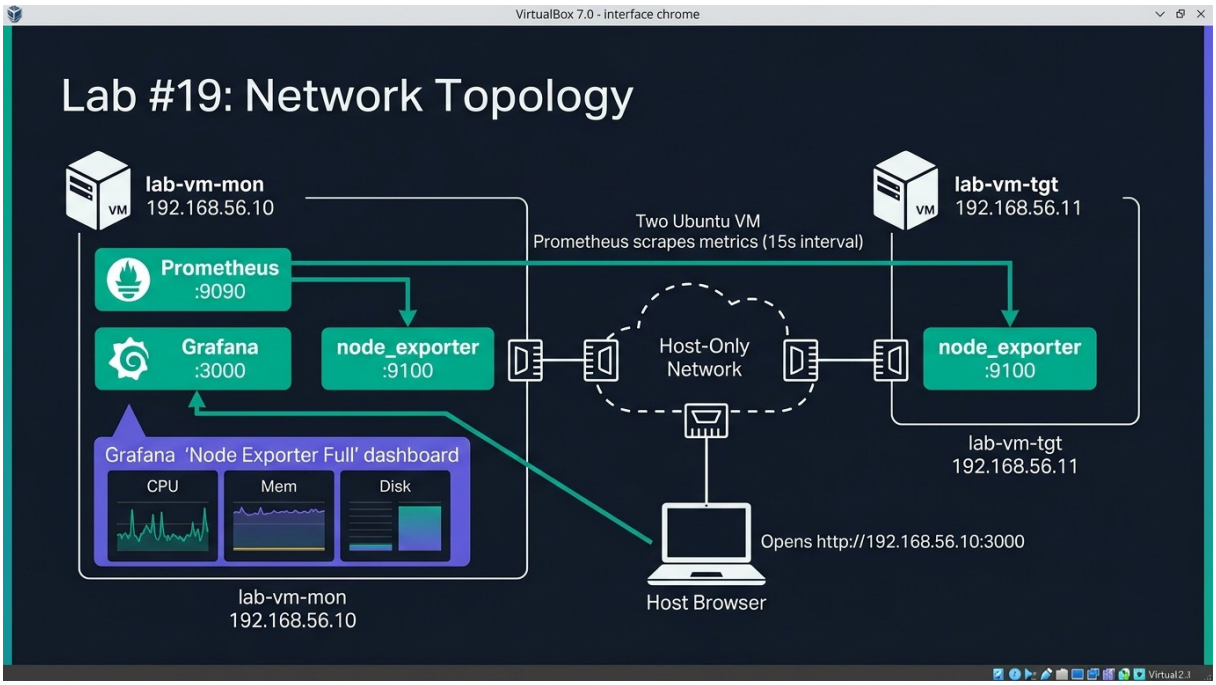
GenAI Labs

강사 박수현 · CEO / CTO

실습 목적

- **Pull 방식 메트릭 수집** (Prometheus → exporter)
- **Node Exporter**: OS 메트릭 (CPU · Mem · Disk · NW)
- **Grafana**: 대시보드 시각화
- **AlertManager**: 임계 초과 알림
- **PromQL** 기본

아키텍처 구성 (Topology)



위 토폴로지를 머리에 그린 상태로 아래 실습을 진행한다.

도구

- VM 2 대: mon (192.168.56.10) + tgt (192.168.56.11)
- Prometheus 2.50+, Grafana 10+, node_exporter 1.7+

기본 예제

```
# Target 측 (192.168.56.11) — node_exporter
wget https://github.com/prometheus/node_exporter/releases/download/v1.7.0/
node_exporter-1.7.0.linux-amd64.tar.gz
tar xzf node_exporter-*.tar.gz
sudo cp node_exporter-*/node_exporter /usr/local/bin/

sudo useradd --no-create-home --shell /bin/false prometheus
sudo tee /etc/systemd/system/node_exporter.service <<EOF
[Unit]
Description=Node Exporter
After=network.target
[Service]
User=prometheus
ExecStart=/usr/local/bin/node_exporter
[Install]
WantedBy=multi-user.target
EOF
sudo systemctl enable --now node_exporter

# 검증
curl http://192.168.56.11:9100/metrics | head

# Monitor 측 (192.168.56.10) — Prometheus
wget https://github.com/prometheus/prometheus/releases/download/v2.50.0/
prometheus-2.50.0.linux-amd64.tar.gz
tar xzf prometheus-*.tar.gz

# /etc/prometheus/prometheus.yml
global:
  scrape_interval: 15s
scrape_configs:
  - job_name: 'node'
    static_configs:
      - targets:
          - '192.168.56.10:9100'
          - '192.168.56.11:9100'

# 시작
./prometheus --config.file=prometheus.yml
# http://192.168.56.10:9090 → Status → Targets → UP

# Grafana
curl -s https://packages.grafana.com/gpg.key | sudo apt-key add -
```

```
sudo apt install -y grafana
sudo systemctl enable --now grafana-server
# http://192.168.56.10:3000 (admin/admin)
```

응용 예제 ① — PromQL

```
# CPU 사용률 (5분 평균)
100 - (avg by (instance) (rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)

# 메모리 사용률
(node_memory_MemTotal_bytes - node_memory_MemAvailable_bytes) /
node_memory_MemTotal_bytes * 100

# 디스크 사용률
(node_filesystem_size_bytes - node_filesystem_avail_bytes) /
node_filesystem_size_bytes * 100

# 네트워크 RX (5분 평균 Mbps)
rate(node_network_receive_bytes_total[5m]) * 8 / 1024 / 1024

# 머신 다운 (5분간 메트릭 없음)
up == 0
```

응용 예제 ② — Grafana 대시보드

```
Grafana → Dashboards → Import → 1860 (Node Exporter Full)
→ Prometheus data source 선택 → Import
```

CPU/Mem/Disk/NW 70+ 패널 한 번에.

응용 예제 ③ — AlertManager

```
# prometheus.yml
rule_files:
  - /etc/prometheus/alerts.yml

# alerts.yml
groups:
  - name: node
    rules:
      - alert: HighCPU
        expr: 100 - (avg by (instance) (rate(node_cpu_seconds_total{mode="idle"}
[5m])) * 100) > 85
        for: 5m
        annotations:
          summary: "{{ $labels.instance }} CPU 85% 초과 5분"

      - alert: InstanceDown
        expr: up == 0
```

```
        for: 2m
        annotations:
          summary: "{{ $labels.instance }} 다운 2 분"
```

응용 예제 ④ — service_discovery (정적 → 동적)

```
# 파일 기반 SD
scrape_configs:
  - job_name: 'node'
    file_sd_configs:
      - files: ['/etc/prometheus/targets/*.json']
        refresh_interval: 30s

// /etc/prometheus/targets/node.json
[
  {"targets": ["192.168.56.10:9100", "192.168.56.11:9100"],
   "labels": {"env": "lab"}}
]
```

관전 포인트

1. **Pull** vs Push (Push 는 statsd) — Prometheus 는 pull
2. **scrape_interval 15s** = 운영 표준
3. PromQL `**rate()**` = 카운터를 초당 비율로
4. **labels** 가 운영 정렬·필터의 핵심 (env, region, role)
5. Grafana 1860 대시보드 = 95% 케이스 커버

자주 만나는 오류

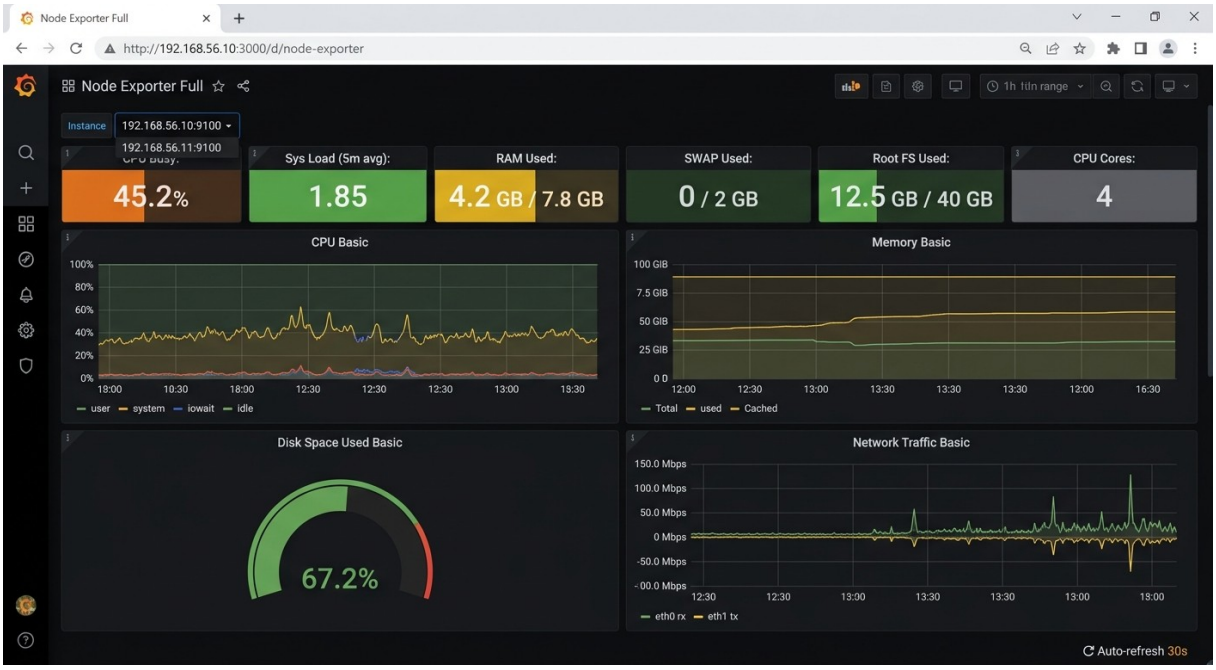
- Target DOWN → 방화벽 (9100, 9090) 또는 node_exporter 죽음
- PromQL 출력 비어있음 → label 매칭 (`{instance="X"}`)
- Grafana 데이터 없음 → Data source URL `http://prometheus:9090` (Docker 일 때)

운영 팁

- 운영은 PVC + 장기 보관 (Cortex·Thanos)
- 알람 Slack/PagerDuty 연동
- Service Discovery: K8s·Consul·AWS EC2 자동
- 이력 분석: 최소 30 일 (90 일 권장)

최종 결과 화면 (Reference)

본 실습 완료 시 보일 것으로 예상되는 대시보드/UI 예시.



부록 — 설정 파일

01_node_exporter.service

```
[Unit]
Description=Node Exporter
After=network.target

[Service]
User=prometheus
Group=prometheus
ExecStart=/usr/local/bin/node_exporter
Restart=always
RestartSec=3

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

01_prometheus.service

```
[Unit]
Description=Prometheus
After=network.target

[Service]
User=prometheus
Group=prometheus
ExecStart=/usr/local/bin/prometheus \
    --config.file=/etc/prometheus/prometheus.yml \
    --storage.tsdb.path=/var/lib/prometheus \
    --web.console.templates=/etc/prometheus/consoles \
    --web.console.libraries=/etc/prometheus/console_libraries \
    --web.enable-lifecycle
ExecReload=/bin/kill -HUP $MAINPID
Restart=always
RestartSec=3

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

01_prometheus.yml

```
# 실습 #19 ① — Prometheus 기본 설정 (단일 머신)
global:
  scrape_interval: 15s
  evaluation_interval: 15s

scrape_configs:
```

```
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'node'
    static_configs:
      - targets: ['localhost:9100']
```

02_grafana_dashboard_provider.yml

```
# 실습 #19 ② — Grafana dashboard provisioning
# /var/lib/grafana/dashboards/ 아래 JSON 을 자동 import
apiVersion: 1

providers:
  - name: lab
    orgId: 1
    folder: Lab
    type: file
    disableDeletion: false
    updateIntervalSeconds: 30
    allowUiUpdates: true
    options:
      path: /var/lib/grafana/dashboards
```

02_grafana_datasource.yml

```
# 실습 #19 ② — Grafana data source provisioning
apiVersion: 1

datasources:
  - name: Prometheus
    type: prometheus
    access: proxy
    url: http://localhost:9090
    isDefault: true
    editable: true
```

03_alertmanager.service

```
[Unit]
Description=Alertmanager
After=network.target

[Service]
User=prometheus
Group=prometheus
ExecStart=/usr/local/bin/alertmanager \
    --config.file=/etc/alertmanager/alertmanager.yml \
```

```
    --storage.path=/var/lib/alertmanager
Restart=always
RestartSec=3

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

03_alertmanager.yml

```
# 실습 #19 ③ — Alertmanager 라우팅 + 수신처
# 본 실습은 webhook 1개만. 운영은 receivers 에 slack/email/pagerduty 추가.

global:
  resolve_timeout: 5m

route:
  receiver: 'default'
  group_by: ['alertname', 'instance']
  group_wait: 10s
  group_interval: 1m
  repeat_interval: 1h

receivers:
  - name: 'default'
    webhook_configs:
      - url: 'http://127.0.0.1:5001/'
        send_resolved: true

# 운영 예시 (slack 추가)
# - name: 'slack-ops'
#   slack_configs:
#     - api_url: 'https://hooks.slack.com/services/T000/B000/XXXX'
#       channel: '#ops-alerts'
#       send_resolved: true

inhibit_rules:
  - source_match:
      severity: 'critical'
    target_match:
      severity: 'warning'
    equal: ['alertname', 'instance']
```

03_alerts.yml

```
# 실습 #19 ③ — 알림 룰
groups:
  - name: node
    rules:
      - alert: HighCPU
        expr: 100 - (avg by (instance) (rate(node_cpu_seconds_total{mode="idle"}
```

```
[2m])) * 100) > 80
        for: 1m
        labels:
          severity: warning
        annotations:
          summary: "{{ $labels.instance }} CPU 80% 1분 초과"
          description: "현재 사용률: {{ $value | printf \"%.1f\" }}%"

      - alert: InstanceDown
        expr: up == 0
        for: 2m
        labels:
          severity: critical
        annotations:
          summary: "{{ $labels.instance }} 다운 2분"

      - alert: HighMemory
        expr: (node_memory_MemTotal_bytes - node_memory_MemAvailable_bytes) /
node_memory_MemTotal_bytes * 100 > 85
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "{{ $labels.instance }} 메모리 85% 초과 5분"

      - alert: DiskFullSoon
        expr: predict_linear(node_filesystem_avail_bytes{mountpoint="/"}[1h],
4*3600) < 0
        for: 30m
        labels:
          severity: warning
        annotations:
          summary: "{{ $labels.instance }} / 디스크 4시간 안에 가득 찰 예상"
```

03_prometheus_with_alerts.yml

```
# 실습 #19 ③ — Prometheus + Alertmanager 연결
global:
  scrape_interval: 15s
  evaluation_interval: 15s

alerting:
  alertmanagers:
    - static_configs:
        - targets: ['localhost:9093']

rule_files:
  - /etc/prometheus/alerts.yml

scrape_configs:
  - job_name: 'prometheus'
```

```
    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'node'
    static_configs:
      - targets: ['localhost:9100']

  - job_name: 'alertmanager'
    static_configs:
      - targets: ['localhost:9093']
```

04_prometheus_multitarget.yml

```
# 실습 #19 ④ — file_sd + relabel 다중 타겟
global:
  scrape_interval: 15s
  evaluation_interval: 15s

alerting:
  alertmanagers:
    - static_configs:
        - targets: ['localhost:9093']

rule_files:
  - /etc/prometheus/alerts.yml

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'node'
    file_sd_configs:
      - files:
          - '/etc/prometheus/targets/*.json'
        refresh_interval: 30s
    relabel_configs:
      # instance 라벨을 "host:port" → "host" 만 남김
      - source_labels: [__address__]
        regex: '([^:]+):.*'
        target_label: host
        replacement: '${1}'
      # env 라벨이 비어 있으면 unknown 으로
      - source_labels: [env]
        regex: ''
        target_label: env
        replacement: unknown

  - job_name: 'alertmanager'
    static_configs:
```

```
      - targets: ['localhost:9093']
```

04_targets_node.json

```
[
  {
    "targets": ["127.0.0.1:9100"],
    "labels": {
      "env": "lab",
      "role": "monitor",
      "region": "kr-cen-1"
    }
  }
]
```

HANDS-ON LAB

실습 #20

Wazuh SIEM·HIDS·EDR 통합 관제

[선택]

IT 인프라 아키텍처 설계

Hands-on Labs · 20 시나리오

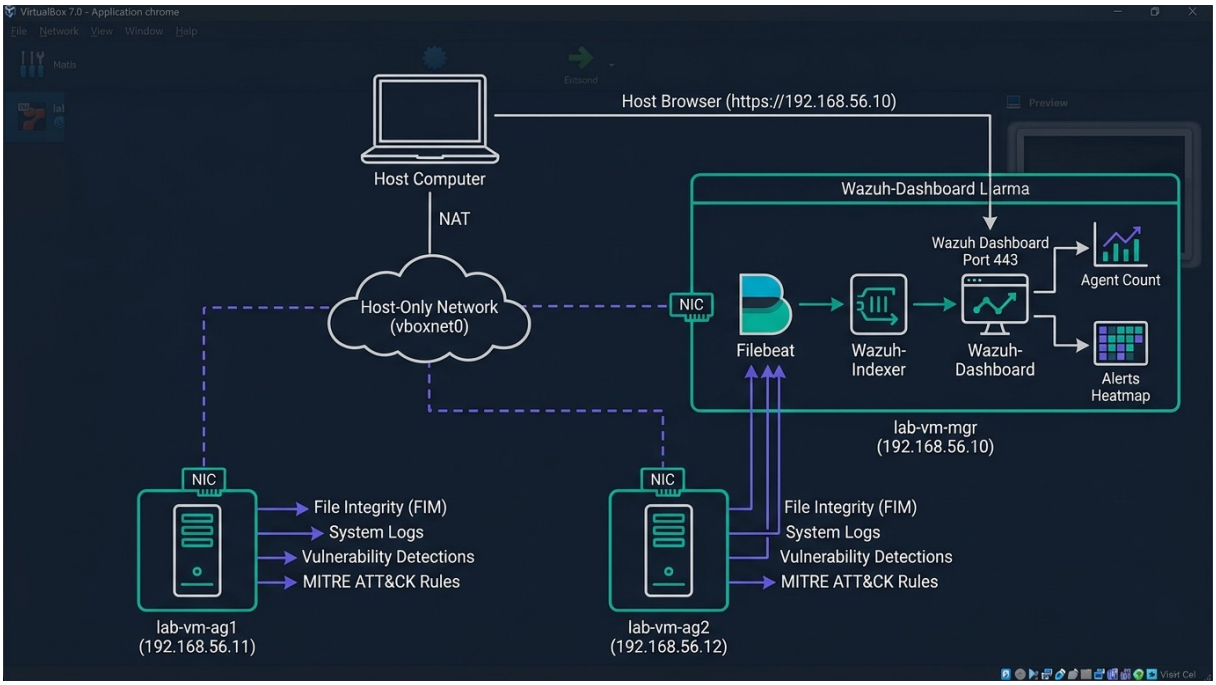
GenAI Labs

강사 박수현 · CEO / CTO

실습 목적

- **Wazuh manager + agents** 구축
- **FIM (File Integrity Monitoring)** 파일 변조 감지
- **SCA (Security Configuration Assessment)** CIS 벤치마크
- **MITRE ATT&CK** 매핑 + 알림
- **취약점 감지** (CVE 매핑)

아키텍처 구성 (Topology)



위 토폴로지를 머리에 그린 상태로 아래 실습을 진행한다.

도구

- VM 3 대: mgr (.10) + ag1 (.11) + ag2 (.12)
- Wazuh 4.7 (manager + indexer + dashboard)

기본 예제 — manager 설치

```
# manager (192.168.56.10)
curl -sO https://packages.wazuh.com/4.7/wazuh-install.sh
sudo bash wazuh-install.sh --all-in-one

# 약 15 분 후 — 비밀번호 출력
# https://192.168.56.10/ (admin/생성된 비밀번호)

# agent (192.168.56.11, .12)
curl -sO https://packages.wazuh.com/4.7/wazuh-install.sh
sudo WAZUH_MANAGER='192.168.56.10' bash wazuh-install.sh --wazuh-agent

sudo systemctl enable --now wazuh-agent

# manager 에서 확인
sudo /var/ossec/bin/agent_control -l
# ID: 001, Name: ag1, IP: 192.168.56.11, Status: Active
```

응용 예제 ① — FIM (파일 무결성)

```
<!-- /var/ossec/etc/ossec.conf (agent 측) -->
<syscheck>
    <directories check_all="yes" realtime="yes">/etc</directories>
    <directories check_all="yes" realtime="yes">/usr/bin</directories>
    <directories check_all="yes">/var/www/html</directories>
</syscheck>

sudo systemctl restart wazuh-agent

# 변조 테스트
sudo echo "BAD" >> /etc/passwd
# Wazuh 대시보드 → Security events → File integrity changes
# 5 초 안에 알람 발생
```

응용 예제 ② — SCA (CIS 벤치마크)

```
대시보드 → Modules → Security configuration assessment
→ Ubuntu 22.04 LTS Benchmark 자동 실행
→ Pass/Fail 항목별 점수 + 권고 조치
```

응용 예제 ③ — Active Response (자동 차단)

```
<!-- manager 측 ossec.conf -->
<active-response>
    <command>firewall-drop</command>
    <location>local</location>
    <rules_id>5712</rules_id>     <!-- SSH brute force -->
    <timeout>600</timeout>
</active-response>
```

SSH 무차별 대입 감지 → 자동 nftables drop 10 분.

응용 예제 ④ — Custom rule

```
<!-- /var/ossec/etc/rules/local_rules.xml -->
<group name="local,custom,">
    <rule id="100001" level="10">
        <if_sid>5712</if_sid>
        <description>SSH brute force from $(srcip)</description>
        <mitre>
            <id>T1110</id>     <!-- Brute Force -->
        </mitre>
    </rule>
</group>
```

응용 예제 ⑤ — Vulnerability detection

```
대시보드 → Modules → Vulnerabilities
→ NVD 데이터베이스 동기화
→ 설치된 패키지 vs CVE 매핑
→ Critical / High / Medium 분류
```

관전 포인트

1. **FIM realtime** = inotify 활용 즉시 감지
2. **MITRE ATT&CK** 매핑이 운영 SOC 의 표준
3. **Active Response** 로 부분 자동 대응 (인간 검토 필수)
4. **취약점 자동 매핑** = 수동 패치 관리의 종말
5. **Wazuh = OSSEC fork** + 현대화 (대시보드·인덱서)

자주 만나는 오류

- agent Disconnected → 1514·1515 포트 차단
- 인덱서 디스크 풀 → 로그 회전 또는 디스크 확장

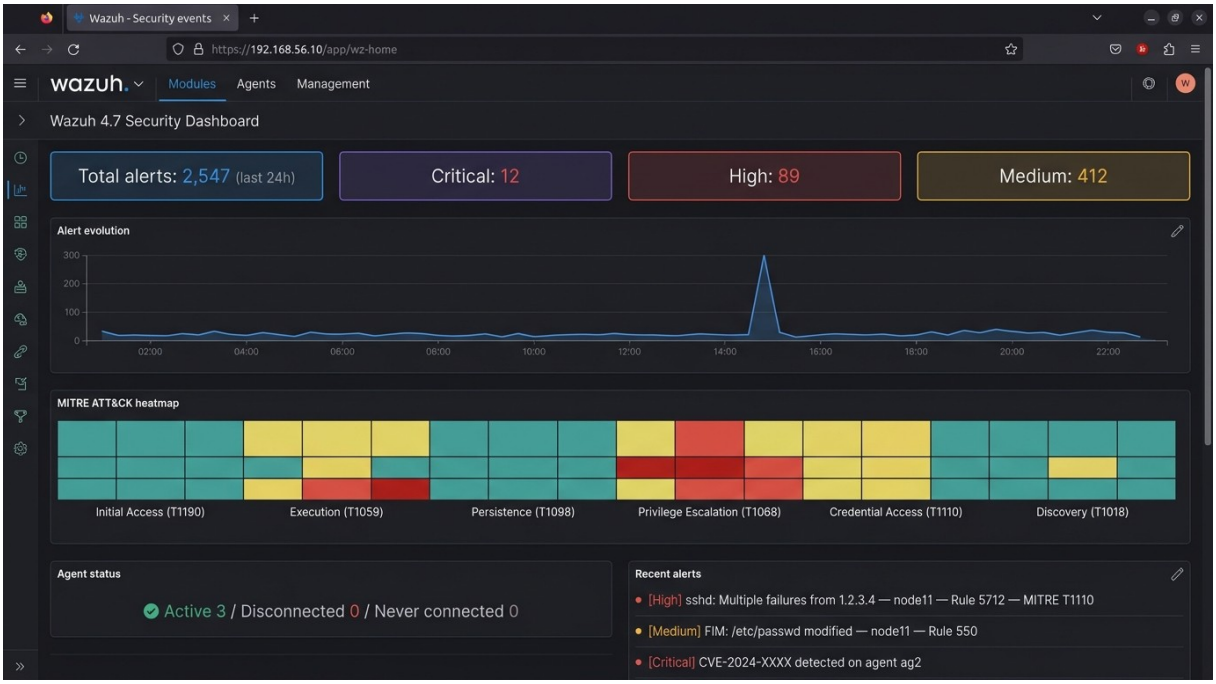
- 대시보드 느림 → JVM 힙 부족 (`/etc/wazuh-indexer/jvm.options`)

운영 팁

- agent 그룹별 정책 분리 (`/var/ossec/etc/shared/`)
- 로그 분석은 인덱서 보관 30~90 일
- 외부 위협 정보 (Threat Intel) 통합
- 사고 대응 SOP 와 연계 (Runbook 자동화)

최종 결과 화면 (Reference)

본 실습 완료 시 보일 것으로 예상되는 대시보드/UI 예시.



부록 — 설정 파일

03_ossec_syscheck.xml

```
  <!-- 실습 #20 ③ — agent 측 syscheck 추가 디렉터리 -->
  <!-- 실시간(inotify) 감시: 변경되면 5 초 이내 alert -->
  <directories check_all="yes" realtime="yes">/var/www/html-lab</directories>
  <directories check_all="yes" realtime="yes">/tmp/lab-suspicious</directories>
  <directories check_all="yes">/etc/lab-fim-canary</directories>

  <!-- 무시할 임시 파일 -->
  <ignore>/etc/mtab</ignore>
  <ignore>/etc/hosts.deny</ignore>
  <ignore type="sregex">.log$|.swp$</ignore>
```

04_local_rules.xml

```
<!-- 실습 #20 ④ — Custom rules + MITRE ATT&CK 매핑 -->
<group name="local,custom,">

  <!-- SSH brute force 에 MITRE T1110 매핑 -->
  <rule id="100001" level="10">
    <if_sid>5712</if_sid>
    <description>SSH brute force from $(srcip)</description>
    <mitre>
      <id>T1110</id>
    </mitre>
  </rule>

  <!-- /etc/passwd 변조 -->
  <rule id="100002" level="12">
    <if_sid>550</if_sid>
    <match>/etc/passwd</match>
    <description>Critical: /etc/passwd modified</description>
    <mitre>
      <id>T1098</id>
    </mitre>
  </rule>

  <!-- web root 변조 (defacement 의심) -->
  <rule id="100003" level="10">
    <if_sid>550,554</if_sid>
    <match>/var/www/html</match>
    <description>Web defacement candidate: $(file) changed</description>
    <mitre>
      <id>T1491.001</id>
    </mitre>
```

```
  </rule>

</group>
```

04_vulnerability_detector.xml

```
  <!-- 실습 #20 ④ — Vulnerability Detector (Wazuh 4.7) -->
  <vulnerability-detector>
    <enabled>yes</enabled>
    <interval>5m</interval>
    <min_full_scan_interval>6h</min_full_scan_interval>
    <run_on_start>yes</run_on_start>

    <!-- Ubuntu CVE feed -->
    <provider name="canonical">
      <enabled>yes</enabled>
      <os>jammy</os>           <!-- 22.04 LTS -->
      <os>focal</os>           <!-- 20.04 LTS -->
      <update_interval>1h</update_interval>
    </provider>

    <!-- NVD (모든 패키지) -->
    <provider name="nvd">
      <enabled>yes</enabled>
      <update_from_year>2020</update_from_year>
      <update_interval>1h</update_interval>
    </provider>
  </vulnerability-detector>
```